

**Web Application Penetration Testing**

**[OWASP Juice Shop]**



A yellow carton with a red cap on it

AI-generated content may be incorrect.

**Table of Contents**

[**1. Date of Issue** 3](#_Toc215394639)

[**2. Document Control** 3](#_Toc215394640)

[**3. Executive Summary** 3](#_Toc215394641)

[**3.1 Graphical Summary** 4](#_Toc215394642)

[**4. Scope of Engagement** 5](#_Toc215394643)

[**5. Methodology (OWASP-Based)** 6](#_Toc215394644)

[**5.1 Information Gathering (Reconnaissance)** 6](#_Toc215394645)

[**5.2 Threat Modeling** 7](#_Toc215394646)

[**5.3 Vulnerability Analysis ()** 7](#_Toc215394647)

[**5.4 Exploitation** 7](#_Toc215394648)

[**5.5 Post-Exploitation** 7](#_Toc215394649)

[**5.6 Reporting** 7](#_Toc215394650)

[**6. Checklist (OWASP Based)** 7](#_Toc215394651)

[**7. Technical Findings (POC Section)** 9](#_Toc215394652)

[**7.1 SQL Injection** 9](#_Toc215394653)

[**7.2 Broken Authentication** 10](#_Toc215394654)

[**7.3 Sensitive Data Exposure / Information Disclosure** 14](#_Toc215394655)

[**7.4 Broken Authentication** 15](#_Toc215394656)

[**7.5 NoSQL Injection** 20](#_Toc215394657)

[**7.6 NoSQL Injection** 24](#_Toc215394658)

[**7.7 Business Logic** 26](#_Toc215394659)

[**7.8 Improper Input Validation** 30](#_Toc215394660)

[**7.9 Weak Cryptography** 32](#_Toc215394661)

[**7.10 Broken Authentication via OSINT** 35](#_Toc215394662)

[**7.11 Error Handling** 1](#_Toc215394663)

[**7.12 Improper Input Validation** 2](#_Toc215394664)

[**8. Conclusion** 3](#_Toc215394665)

# **1. Date of Issue**

**JUN 2025 – DEC 2025**

# **2. Document Control**

|  |  |
| --- | --- |
| **Authors** | **Reviewer** |
| Mohamed Ziad Mohamed Elrahwan | **Hesham saleh** |
| Haneen Sherif |
| Shorouk Gamal |
| Eyad Hesham |
| Mariam Mohamed |
| Beshoy Safwat |

# **3. Executive Summary**

This penetration test assessed the overall security posture of the OWASP Juice Shop web application. The objective was to identify vulnerabilities, evaluate their impact, and provide actionable remediation recommendations.

The assessment followed OWASP-based testing methodology and focused on identifying weaknesses related to authentication, access control, business logic, injection flaws, insecure data handling, and client-side vulnerabilities.

**Overall Risk Posture:**

A number of vulnerabilities were identified across the application, ranging from critical-impact flaws to low-risk issues. The most severe findings include authentication bypasses, business logic abuse, and injection vulnerabilities that could lead to account takeover, unauthorized administrative access, or data leakage.

## **3.1 Graphical Summary**



Executive-Level Summary per Vulnerability Category:

* Authentication Weaknesses: Several features allowed bypassing login controls and resetting user credentials without proper verification.
* Access Control Issues: Unauthorized access to restricted sections such as administrative interfaces and internal resources was possible.
* Cross-Site Scripting (XSS): The application contained both reflected and DOM-based XSS vulnerabilities, allowing malicious JavaScript execution in user browsers.
* Business Logic Manipulation: Improper validation enabled altering basket totals, exploiting feedback systems, and tampering with application workflows.
* Exposure of Sensitive Data: Logs, metrics, and backup files revealed internal system information that could assist an attacker.
* Input Handling Flaws: Missing or insufficient validation allowed crafted payloads to bypass filters or exploit internal mechanisms.

These issues collectively demonstrate that the application is intentionally insecure (as designed for educational purposes) and would require extensive remediation efforts to achieve production-grade security.

# **4. Scope of Engagement**

|  |  |
| --- | --- |
| Category | Details |
| Target | OWASP Juice Shop |
| Pentest Type | Black Box |
| In-Scope URL | https://juice-shop.herokuapp.com/\* |
| In-Scope Functions | Web Application (all functionalities) |
| Authentication and authorization flows |
| API endpoints and backend responses |
| User interface interactions |
| Client-side logic |
| Business logic processes |
| Out-Of-Scope Functions | Social engineering |
| Physical security |
| 3rd party integrations |
| Network infrastructure |

# **5. Methodology (OWASP-Based)**

The testing methodology followed OWASP standards and industry best practices. The major phases included:

A diagram of a web application

AI-generated content may be incorrect.

## **5.1 Information Gathering (Reconnaissance)**

* Collecting publicly available and internal application information
* Identifying domains, subdomains, technologies, and exposed services
* Enumerating directories, parameters, and API endpoints
* Fingerprinting frameworks, versions, and server configurations
* Performing passive and active reconnaissance to map the attack surface

## **5.2 Threat Modeling**

* Mapping application components
* Identifying high-risk functionalities
* Recognizing sensitive data flows
* Understanding trust boundaries and user roles

## **5.3 Vulnerability Analysis ()**

* Manual and automated vulnerability discovery
* OWASP Top 10 and WSTG-aligned test cases
* Parameter tampering, injection attempts, logic testing
* Client-side and API testing

## **5.4 Exploitation**

* Validating discovered weaknesses
* Demonstrating exploit feasibility
* Crafting payloads
* Testing privilege escalation routes

## **5.5 Post-Exploitation**

* Determining data exposure possibilities
* Accessing internal resources
* Evaluating persistence or privilege retention

## **5.6 Reporting**

* Documenting findings with severity ratings
* Mapping to OWASP references
* Providing mitigations and business impact

# **6. Checklist (OWASP Based)**

|  |  |  |
| --- | --- | --- |
| **Category** | **Name (Challenges Under This Category)** | **Status** |
| **Authentication** | Reset Bender’s Password | Passed |
| Login Jim | Passed |
| Login Bender | Passed |
| Login Amy | Passed |
| Empty\_User\_Registration | Passed |
| Reset Jim’s Password | Passed |
| **Access Control / Authorization** | Login Admin | Passed |
| Admin Section | Passed |
| Admin Registration | Passed |
| GDPR Data Theft | Passed |
| Leaked Unsafe Product | Passed |
| Login MC SafeSearch | Passed |
| **Injection (NoSQL, Input, Validation)** | NoSQL DoS | Passed |
| NoSQL Manipulation | Passed |
| Allowlist Bypass | Passed |
| Outdated Allowlist | Passed |
| Missing Encoding | Passed |
| Web3 Sandbox | Passed |
| **Cryptography Issues** | Weird Crypto |  |
| Outdated Allowlist (crypto-related) |  |
| NFT Takeover (crypto/web3 logic) |  |
| **Cross-Site Scripting (XSS)** | API-only XSS |  |
| Reflected XSS |  |
| DOM XSS |  |
| Client-side XSS Protection |  |
| CSP Bypass (XSS) |  |
| HTTP Header XSS |  |
| **Cross-Site Request Forgery (CSRF)** | CSRF |  |
| **Business Logic Vulnerabilities** | Forged Feedback |  |
| Legacy Typosquatting |  |
| Manipulate Basket |  |
| Deluxe Fraud |  |
| Payback Time |  |
| Zero Stars |  |
| View Basket |  |
| Five-Star Feedback |  |
| Forged Review |  |
| Repetitive Registration |  |
| Mint the Honey Pot |  |
| Mass Dispel |  |
| Easter Egg |  |
| Christmas Special |  |
| Bjoern’s Favorite Pet |  |
| **Information Disclosure** | Exposed Credentials |  |
| Database Schema |  |
| Exposed Metrics |  |
| Access Log |  |
| Forgotten Developer Backup |  |
| Forgotten Sales Backup |  |
| Confidential Document |  |
| Password Strength |  |
| Exposed Metrics (non-sensitive) |  |
| Access Log (non-sensitive) |  |
| Privacy Policy (public info) |  |
| Security Policy (public info) |  |
| **Client-Side / Frontend Logic** | Bonus Payload |  |
| Visual Geo Stalking |  |
| Meta Geo Stalking |  |
| **Configuration Issues** | Deprecated Interface |  |
| Misplaced Signature File |  |
| Security Policy (config-based) |  |
| **Error Handling** | Error Handling |  |

# **7. Technical Findings (POC Section)**

## **7.1 SQL Injection**

**Critical**

**Description:**

The login endpoint fails to properly validate and sanitize user-supplied input.  
By manipulating the request parameters, an attacker can alter the backend query logic, causing the system to authenticate them as the admin user without valid credentials.  
This confirms a critical flaw in the authentication flow, caused by improper input handling combined with weak server-side verification.

**Impact:**

Injection in the login functionality allowed bypassing authentication controls and logging in directly as the administrator, this results in fulladministrativecompromise, giving an attacker unrestricted access to all backend functions, sensitive user data, and system-level actions. This effectively places the attacker in full control of the application.

**Affected Endpoint/Parameter:**

https://juice-shop.herokuapp.com/#/login

**Reference:** <https://owasp.org/www-community/attacks/SQL_Injection>

**Proof of Concept (PoC):**

First, when the pentester sees a login form he can think about SQL injection

A screenshot of a computer login

AI-generated content may be incorrect.

He can start trying to brute-force this form using a simple query: ' OR 1=1 -- -

**A screenshot of a computer

AI-generated content may be incorrect.**

We are in as Administrator

****

**Mitigation:**

Using Parameterized Query

## **7.2 Broken Authentication**

**Critical**

**Description:**

Reset Bender’s password via the Forgot Password mechanism with the original answer to his security question.

**Impact:**

* **Account Takeover** – The attacker can fully compromise the user’s account.
* **Unauthorized Access** – Sensitive data and actions become accessible without permission.
* **Privacy Breach** – Personal information is exposed due to weak password-recovery controls

**Affected Endpoint/Parameter:**

https://juice-shop.herokuapp.com /#/forgot-password

**Reference**

https://cheatsheetseries.owasp.org/cheatsheets/Authentication\_Cheat\_Sheet.html

[OWASP Authentication Cheat Sheet]

**Proof of Concept (PoC):**

* 1. **figure out what the security question is.**

**A screenshot of a computer

AI-generated content may be incorrect.**

* 1. **Capture the Reset Request in Burp Suite**

I submitted any random answer to trigger the request, then intercepted it using **Burp Proxy**.  
The captured request was:

{"email":"bender@juice-sh.op","answer":"test","new":"123456","repeat":"123456"}

* 1. **Send the Request to Intruder**

Inside the JSON body, I replaced the answer with a payload marker:

**A screenshot of a computer

AI-generated content may be incorrect.**

* 1. **Load the Wordlist**

I used a custom wordlist generated based on Bender’s company names from Futurama .

A screenshot of a computer

AI-generated content may be incorrect.

* 1. **Launch the Attack**

After starting the Intruder attack, most responses returned:

**401 Unauthorized – Wrong answer to security question**

But one payload returned a **200 OK**, indicating the correct answer:

**Correct Answer: Stop'n'Drop**

**A screenshot of a computer

AI-generated content may be incorrect.**

* 1. **Successful Password Reset & Login**

After running the Intruder attack with the prepared wordlist, the correct security answer was found.  
The server accepted the request and successfully reset Bender’s password to:

**New Password: 123456**

I then used this new password to log into Bender’s account successfully.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Mitigation:**

* Strong Password Policies – Require complex, non-predictable passwords.
* Rate-Limiting / Lockout – Limit failed attempts to prevent brute-force attacks.
* MFA – Add an extra verification layer.

## **7.3 Sensitive Data Exposure / Information Disclosure**

**Critical**

**Description:**

Test login credentials were left visible in the website’s public code, allowing anyone to access the account.

**Impact:**

* This allowed unauthorized access to the user account, exposing sensitive user data.

**Affected Endpoint/Parameter:**

`  [https://juice-shop.herokuapp.com /main.js](http://127.0.0.1:3000/main.js) `

**Reference**

[**https://owasp.org/www-community/vulnerabilities/Use\_of\_hard-coded\_password**](https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password)

[Use of hard-coded password]

**Proof of Concept (PoC):**

1. **Navigate to the Application**

I accessed the Juice Shop application normally to analyze the client-side behavior.

* 1. **Inspect Client-Side Source Code**

I opened the browser DevTools → Sources tab and reviewed the frontend files.

A screenshot of a computer

AI-generated content may be incorrect.

* 1. **Locate Hardcoded Credentials**

Inside main.js, I found hard-coded test credentials directly embedded in the client-side code:

testingUsername = "testing@juice-sh.op";

testingPassword = "IamUsedForTesting";

* 1. **Attempt Login Using Exposed Credentials**

I navigated to the login page and entered the discovered username and password.

A screenshot of a computer

AI-generated content may be incorrect.

**Mitigation**

* Remove hardcoded credentials from client-side code and enforce secure authentication practices.

## **7.4 Broken Authentication**

**Critical**

**Description:**

The "Login Amy" challenge involves using clues related to a fictional character to determine the password and gain access to the account.

**Impact:**

Weak and predictable passwords allow attackers to brute‑force Amy’s account and gain unauthorized access, exposing her personal data and increasing the risk of further attacks.

* **Account Takeover** – The attacker can fully compromise the user’s account.
* **Unauthorized Access** – Sensitive data and actions become accessible without permission.
* **Privacy Breach** – Personal information is exposed due to weak password-recovery controls

**Affected Endpoint/Parameter:**

` https://juice-shop.herokuapp.com /#/forgot-password `

**Reference**

<https://owasp.org/Top10/A07_2021-Broken_Authentication/>

**Proof of Concept (PoC):**

1. **Reading the Challenge**

The challenge description stated:

*“Log in with Amy's original user credentials. (This could take 93.83 billion trillion trillion centuries to brute force, but luckily she did not read the ‘One Important Final Note’).”*

From this, I noted two key points:

 The brute-force time refers to password strength/entropy.

 Amy ignored the “One Important Final Note”, suggesting predictable padding

1. **Investigating the Brute-Force Hint**

Since the challenge didn’t provide any tool, I **searched online** and found the **GRC Password Haystack calculator**:

🔗 [*https://www.grc.com/haystack.htm*](https://www.grc.com/haystack.htm)

While exploring the site, I read the **“One Important Final Note”**, which advised:A close up of a paper

AI-generated content may be incorrect.

Because Amy **ignored this note**, I inferred she likely used **simple dots** as padding.

1. **Determining Password Length**

I tested multiple lengths on the calculator to match the brute-force estimate (**93.83 billion trillion trillion centuries**) given in the challenge.

 The calculator indicated that **a 24-character password** would produce the correct brute- force duration.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Researching the Character “Amy”**

A black background with white text

AI-generated content may be incorrect.To generate candidate passwords, I researched **Amy Wong** from *Futurama*:

* She is romantically involved with **Kif Kroker**.
* Kif is the closest personal name associated with her

1. **Building a Wordlist**

Using the hints from Amy’s personality and predictable padding behavior, I created a **custom wordlist** with 24-character passwords:

A screenshot of a computer

AI-generated content may be incorrect.

All passwords are **24 characters** to match the brute-force estimate.

 Padding uses **dots** since Amy ignored the recommendation.

1. **Brute-Forcing Amy’s Password**

Go to login page to log with [amy@juice-sh.op](mailto:amy@juice-sh.op) account

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.I used **Burp Suite** to intercept the request:

Send the request to intruder to do a bruteforce with the custom wordlist:

* Attack Type : Sniper
* Parameter : password
* Payload : Custom wordlist

A screenshot of a computer

AI-generated content may be incorrect.

I monitored the responses to identify a successful login.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Identifying the Valid Credential**

One payload returned a **different response**, confirming the correct password:

**`K1f.....................`**

A screenshot of a user profile

AI-generated content may be incorrect.

**Mitigation**

* Strong Password Policies – Require complex, non-predictable passwords.
* Rate-Limiting / Lockout – Limit failed attempts to prevent brute-force attacks.
* MFA – Add an extra verification layer.

## **7.5 NoSQL Injection**

**Critical**

**Description:** this challenge exploits a NoSQL injection vulnerability in the Juice Shop application to perform a Denial of Service (DoS) attack by making the server sleep.

**Impact:**

* **Server Downtime** – The application becomes unresponsive during the injected sleep period.
* **Resource Exhaustion** – Malicious requests can consume server resources and degrade performance.
* **Service Disruption** – Legitimate users cannot access the app, leading to a full or partial Denial of Service.

**Affected Endpoint/Parameter:**

` <http://127.0.0.1:3000/# /rest/products/:id/reviews> `

**Reference**

<https://owasp.org/www-community/attacks/NoSQL_Injection>

**Proof of Concept (PoC):**

**1. Identifying the Vulnerable Endpoint**

I attempted to inject NoSQL payloads into the `id` parameter to check how the backend handles unexpected expressions.

**Payload Attempt :** {"id": "sleep(10000)", "message": "test"}

**The server accepted the request and returned:** HTTP/1.1 201 Created {"status":"success"}

This indicated the server is processing unvalidated input, so I proceeded with a more impactful payload.

**2. Intercepting the Request**

* **Navigate to the Review Section:** Go to a product page on Juice Shop and submit a review.
* **Intercept the Request:** Use Burp Suite to intercept the request when submitting or modifying a review.

A screenshot of a computer

AI-generated content may be incorrect.

**3.Analyzing the Request**

The intercepted request for modifying a comment looks like this:

A computer screen shot of a computer code

AI-generated content may be incorrect.

**4.Crafting the DoS Payload**

To exploit the NoSQL injection vulnerability, we need to craft a payload that will make the server sleep for a specified duration:

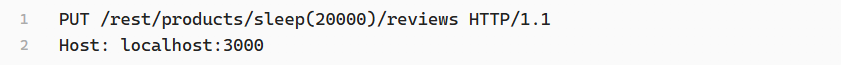
* **Payload Attempt 1:** Injecting `sleep` unction in the `author` parameter:

A screenshot of a computer

AI-generated content may be incorrect.

**Result:** This payload did not work as expected.

* **Payload Attempt 2:** Injecting the payload directly into the URL:



**5.Executing the Payload**

**Access URL Directly:** The direct injection via Burp Suite did not yield results, so we accessed the URL directly:



<http://localhost:3000/rest/products/sleep(20000)/reviews>

1

A screenshot of a computer

AI-generated content may be incorrect.

**The server become unresponsive**

**A screenshot of a phone

AI-generated content may be incorrect.**

**Mitigation**:

* **Validate & sanitize inputs** to block injections.
* **Use parameterized queries** to separate data from logic.
* **Apply rate-limiting & timeouts** to reduce DoS impact

## **7.6 NoSQL Injection**

**Critical**

**Description:**

This challenge involves exploiting a NoSQL injection vulnerability to update multiple product reviews simultaneously in a NoSQL database.

**Impact:**

* Mass Data Manipulation – Attackers can modify all user reviews at once, leading to loss of data integrity.
* Privilege Abuse – Unauthorized users can alter data they shouldn’t have access to.
* Trust Damage – Manipulated or corrupted reviews reduce the reliability of the entire system.

**Affected Endpoint/Parameter:**

`  [https://juice-shop.herokuapp.com/# /rest/products/reviews](http://127.0.0.1:3000/# /rest/products/reviews) `

**Reference**

<https://owasp.org/www-community/attacks/NoSQL_Injection>

**Proof of Concept (PoC):**

1. **Navigating to the Website and Accessing the Comment**

I first visited the website normally and navigated to an existing comment that had already been posted. Before attempting any exploitation, I reviewed the comment structure and previously edited it to understand how the update process worked.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Analyzing the Normal Update Request**

Initially, I captured a normal HTTP PATCH request by using Burp Suite. This request meant for updating a single product review. The request looked something like this:

A screenshot of a computer

AI-generated content may be incorrect.

**2.Crafting the NoSQL Injection Payload**

To exploit the NoSQL injection, the challenge was to update multiple reviews at once. This required manipulating the id parameter in the JSON payload. MongoDB, the underlying NoSQL database, interprets certain patterns in a specific way. For this task, I used the $ne (not equal) operator, which is a MongoDB operator to select the documents where the value of the field is not equal to the specified value. Using  {"$ne": null} ensures that the condition is true for all entries (as none of the ids are null).

A screenshot of a computer

AI-generated content may be incorrect.

Upon sending this request, the server processed it as a valid query, updating all reviews to have the same message due to the condition applied through NoSQL injection.

**Mitigation**

* Validate & sanitize inputs before processing.
* Use parameterized queries to prevent NoSQL injection.

## **7.7 Business Logic**

**High**

**Description:**

This challenge involves exploiting a business logic flaw to achieve unauthorized financial gain through manipulating product pricing in an e-commerce application.

**Impact:**

* **Financial Loss –** Attackers can exploit negative pricing to gain money or credits from the system.
* **Business Logic Abuse –** Manipulating core pricing logic leads to unauthorized discounts or payouts.
* **Revenue Manipulation –** Product prices can be altered, causing incorrect transactions and harming the business.

**Affected Endpoint/Parameter:**

`  [https://juice-shop.herokuapp.com/# /accounting](%20https://juice-shop.herokuapp.com/# /accounting) `

**Reference**

<https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/10-Business_Logic_Testing/>

**Proof of Concept (PoC):**

1. **Initial Steps and Discovery**

**1.1 Inspect Application Code**: Started by examining the main.js file, which often contains paths and client-side logic. Discovered a path /accounting which seemed promising for administrative actions .

**1.2** **Attempt to Access Accounting Page:** Accessed `http://127.0.0.1/#/accounting ` directly in the browser, leading to an authorization error page.

A screenshot of a computer

AI-generated content may be incorrect.

* 1. **Identify Account with Necessary Permissions:** Found the email of a user likely to have necessary permissions `accountant@juice-sh.op` from products comments

A screenshot of a computer

AI-generated content may be incorrect.

Used an SQL injection vulnerability previously identified to gain access to this account.

**Payload used :** ` [accountant@juice-sh.op'](mailto:accountant@juice-sh.op') -- - `

Now I logged in as `[accountant@juice-sh.op](mailto:accountant@juice-sh.op)'

A screenshot of a user profile

AI-generated content may be incorrect.

1. **Exploiting the Business Logic**
   1. Access Accounting Page as the Accountant:

Logged in as the accountant, re-accessed the accounting page which now displayed product pricing and allowed modifications

A screenshot of a computer

AI-generated content may be incorrect.

* 1. Attempt to Modify Product Prices:

Intercepted the price update request using Burp Proxy.

A screenshot of a computer

AI-generated content may be incorrect.

* 1. Exploit Negative Pricing :
* Purchased the product with the negative price.
* As the system processed the negative amount, it resulted in a credit to the account rather than a charge, effectively leading to a payout

A screenshot of a computer

AI-generated content may be incorrect.

**Mitigation**

* **Enforce Server-Side Validation** – Block negative or invalid pricing values before processing.
* **Implement Role-Based Access Control –** Ensure only authorized users can modify product prices, with strict permission checks.
* **Add Business Logic Safeguards –** Prevent transactions that result in negative totals or unintended credits.

## **7.8 Improper Input Validation**

**Medium**

**Description:**

This challenge involves giving a zero-star feedback rating to a store, exploiting improper input validation mechanisms. The challenge tests abilities to manipulate web forms and intercept network requests to alter data before submission.

**Impact:**

* Fake ratings can be submitted, harming the store’s reputation.
* Users may lose trust in the platform.
* Shows weak input validation that attackers can easily bypass.

**Affected Endpoint/Parameter:**

`  [https://juice-shop.herokuapp.com/#/contact](%20https://juice-shop.herokuapp.com/#/contact) `

**Reference**

<https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html?utm_source=chatgpt.com>

**Proof of Concept (PoC):**

1. **Accessing Customer Feedback**

A screenshot of a computer

AI-generated content may be incorrect.After logging into the application, I navigated directly to the **Customer Feedback**

1. **Submitting a Normal Review**

I attempted to submit a standard review to generate a valid HTTP request that could later be intercepted and analyzed.

A screenshot of a customer feedback

AI-generated content may be incorrect.

1. **Intercepting the Request**

A screenshot of a computer

AI-generated content may be incorrect.Using Burp Suite’s Proxy feature, I captured the outgoing feedback submission request.

1. **Modifying the Rating Parameter**

Despite the UI preventing zero‑star ratings, the server performed no validation on this field.  
I manually modified the rating parameter within the intercepted request, changing it **from 3 to 0**.

A screenshot of a computer

AI-generated content may be incorrect.

**Mitigation**

* + Validate all inputs on the server side.
  + Strengthen client-side checks, but don’t rely on them alone.
  + Enforce strict data types for ratings to prevent invalid values.

## **7.9 Weak Cryptography**

**Medium**

**Description:**

This challenge highlights the use of **weak and outdated password hashing**, making stored passwords easy to crack and putting user accounts at risk.

**Impact:**

* Weak password protection: MD5 can be cracked quickly, exposing user passwords.
* Unauthorized access: Attackers can log into user accounts after cracking the hash.

**Affected Endpoint/Parameter:**

`  [https://juice-shop.herokuapp.com/#/login](%20https://juice-shop.herokuapp.com/#/login) `

**Reference**

<https://owasp.org/Top10/A02_2021-Cryptographic_Failures/>

**Proof of Concept (PoC):**

1.Capture the JWT via Burp Suite

While browsing the application, a JWT was intercepted from **Burp Suite → Proxy → HTTP History**.

A screenshot of a computer

AI-generated content may be incorrect.The payload included sensitive user data, including the admin’s password hash.

2. Decode JWT

**A screenshot of a computer

AI-generated content may be incorrect.**The JWT was Base64-decoded to extract the payload:

**3. Crack the MD5 Hash**

The password hash **0192023a7bbd73250516f069df18b500** was identified as **MD5**

Cracked using **CrackStation**. ` admin123 `**A screenshot of a computer

AI-generated content may be incorrect.**

**4.Login as Admin**

Using the credentials:

**Email:** [admin@juice-sh.op](mailto:admin@juice-sh.op)

**Password:** admin123

A screenshot of a computer

AI-generated content may be incorrect.Login was successful, granting **full administrator privileges**

**Mitigation**

* Use stronger hashing algorithms (e.g., bcrypt or SHA-256) instead of MD5.
* Add unique salts to each password to prevent rainbow table attacks.

## **7.10 Broken Authentication via OSINT**

**Medium**

**Description:**

This challenge focuses on triggering an error to reveal backend system details caused by poor error handling and server misconfigurations.

**Impact:**

* **Information Disclosure** – Reveals sensitive backend details.
* **Facilitates Further Attacks** – Helps attackers craft targeted exploits.
* **Increased Attack Surface** – Exposes internal system weaknesses

**Affected Endpoint/Parameter:**

`  [https://juice-shop.herokuapp.com/#/login](%20https://juice-shop.herokuapp.com/#/login) `

**Reference**

<https://owasp.org/Top10/A07_2021-Broken_Authentication/>

**Proof of Concept (PoC):**

1. **OSINT Search**

Google Dorking was used to identify any online presence for the username *“MC SafeSearch”*. Queries such as:

A screenshot of a social media post

AI-generated content may be incorrect.

1. **Extracting Sensitive Information**

Reviewing the video lyrics revealed critical personal information:

*Pet name : “Mine's my dog, Mr. Noodles.”*

*Password PatternL : “I replaced some vowels with zeroes”*

1. **A screenshot of a computer

   AI-generated content may be incorrect.Building the Password**

Using the extracted information:

* Base word: **Mr. Noodles**
* Transformation: **o → 0**
* Capitalization preserved

**Final Password : Mr. N00dles**

1. **Login Attempt**

Using the discovered credentials:

**Username:** [mc.safesearch@juice-sh.op](mailto:mc.safesearch@juice-sh.op)

**Password:** Mr. N00dles

The login was successful, confirming the OSINT-based password deduction.

A screenshot of a computer

AI-generated content may be incorrect.

**Mitigation**

* Limit Public Exposure: Avoid posting sensitive account-related information publicly.
* Strong, Unique Passwords: Encourage users to use strong and unique passwords for each account.
* Monitor and Educate: Regularly monitor for exposed accounts and educate users on privacy and OSINT risks.

## **7.11 Error Handling**

**Low**

**Description:**

This challenge highlights how publicly accessible personal information can be leveraged to gain insights and assist in accessing user accounts, emphasizing the risks of oversharing online**.**

**Impact:**

* Account Compromise Risk: Publicly available information can make it easier to access user accounts.
* Privacy Exposure: Personal data becomes exposed due to excessive online sharing.
* Social Engineering Opportunities: Attackers can leverage publicly found information to craft convincing attacks.

**Affected Endpoint/Parameter:**

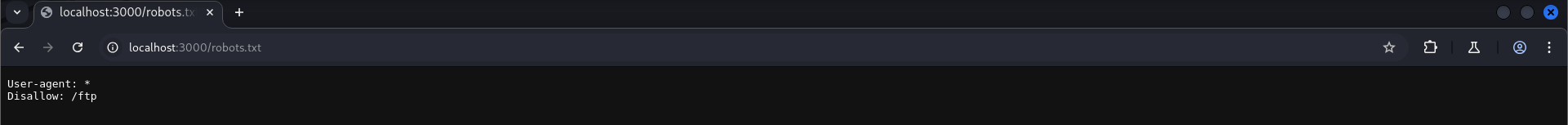
` <https://juice-shop.herokuapp.com/#/coupons_2013.md.bak> `

**Reference**

<https://owasp.org/www-community/Improper_Error_Handling>

**Proof of Concept (PoC):**

1. Navigate to the **/robots.txt** file to identify disallowed directories within the application.



1. Access the /ftp path: ` <https://juice-shop.herokuapp.com/ftp/> `

A screen with a white screen

AI-generated content may be incorrect.

3. Attempt to open any file inside the directory.

A screenshot of a computer

AI-generated content may be incorrect.

Observed Behavior

* 403 Forbidden: The server blocks files with non-permitted extensions, showing strict file-type restrictions.
* Verbose Error Message: The backend reveals Express JS v4.21.0, exposing sensitive framework details useful for attackers.
* Reference: The disclosed version has a known reported issue *(expressjs/express#6222)*, indicating potential security risks.

**Mitigation**

* Use generic error messages to avoid leaking backend details.
* Apply proper error-handling practices that hide system architecture info.
* Keep server software updated to reduce risks from disclosed versions

## **7.12 Improper Input Validation**

**Info**

“in progress”

## **7.13 Information Disclosure**

Severity: Low

**Description:**  
The ‘/‘ metrics endpoint exposes internal server statistics (memory, runtime, request counters) without authentication.

**Impact:**

* Attackers gain insight into server performance.
* Assists reconnaissance and attack planning.
* May expose debugging information.

**Affected Endpoint/Parameter:**  
GET /metrics

**Proof of Concept (PoC):**

* Open browser and navigate to /metrics.
* The server returns sensitive Prometheus-style metA screenshot of a computer

  AI-generated content may be incorrect.rics.

**Mitigation:**

* Restrict access to monitoring endpoints.
* Apply authentication or IP allowlist.
* Disable metrics in production environments.

## **7.14 Business Logic**

Severity: Low

**Description:**  
A hidden debug endpoint allows deletion of user notifications without proper authorization.

**Impact:**

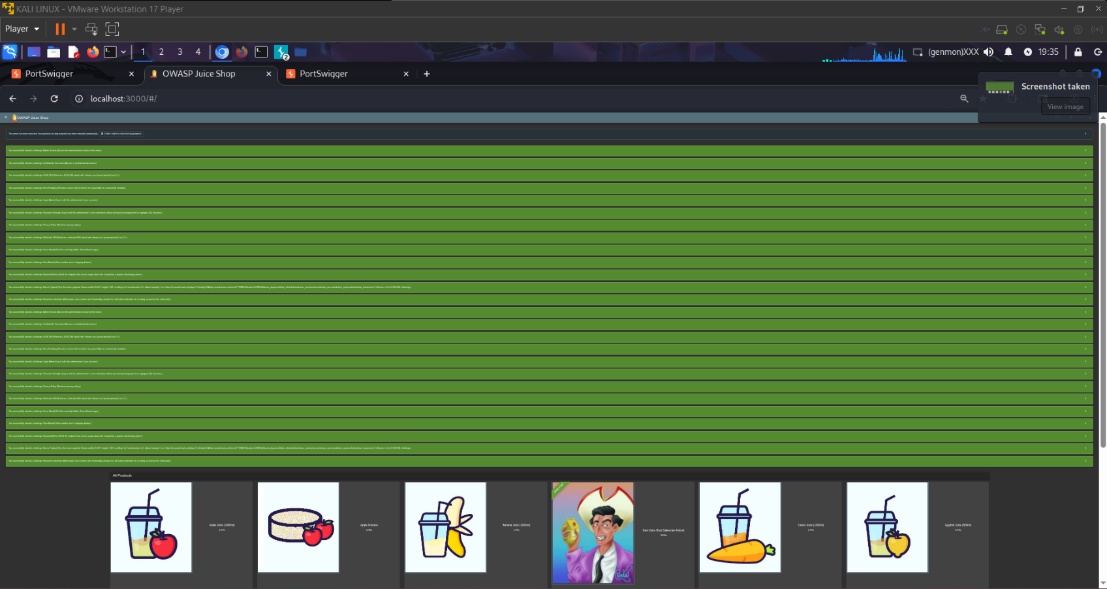
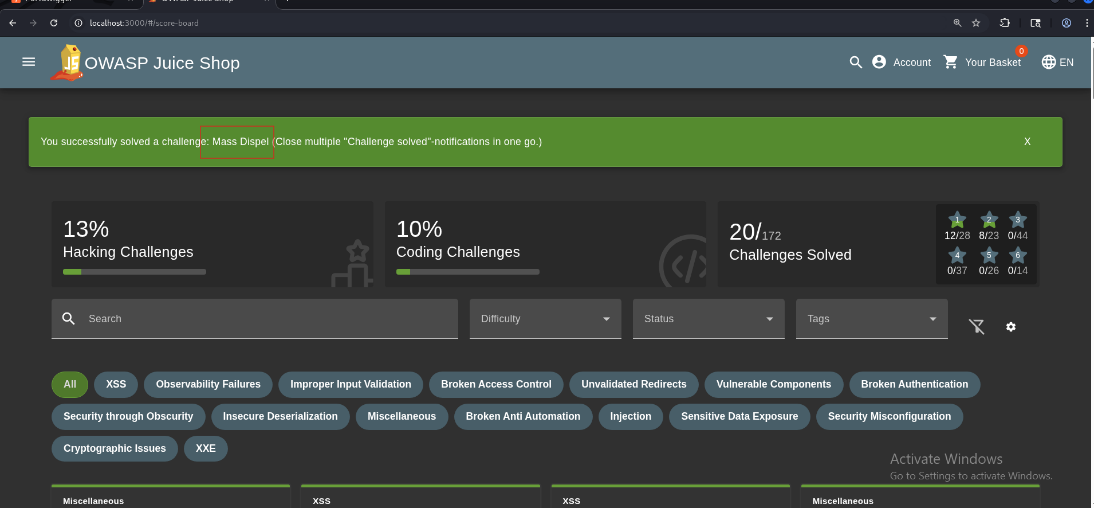
* Users can trigger internal debug functionality.
* May lead to unintended system behavior.

**Affected Endpoint/Parameter:**  
*(Hidden JS-triggered function; no direct API endpoint)*

**Proof of Concept (PoC):**

* Inspect frontend source.
* Trigger hidden notification-clearing function using browser console.  
  A screenshot of a computer program

  AI-generated content may be incorrect.

This was the home page before :  
  
and this after pressing the “Shift” button   


Mitigation:

* Remove unused debug features.
* Restrict internal functionality behind authentication.

## **7.15 Information Disclosure**

3. Password Strength

Severity: Low

**Description:**  
Password validation is performed only on the client side, allowing weak passwords to be accepted.

**Impact:**

* Users may create weak accounts.
* Increase risk of brute‑force attacks.

**Affected Endpoint/Parameter:**  
POST /api/Users/

**Proof of Concept (PoC):**

After inspecting the customers reviews on the products I found the admins mail   
A screenshot of a computer

AI-generated content may be incorrect.

then intercepted the login request and changed the mail to the admins main and brute forced the password and loaded the common wordlist to the payload configuration   
A screenshot of a computer

AI-generated content may be incorrect.

Then I got the password of the admin **A screenshot of a computer

AI-generated content may be incorrect.  
Mitigation:**

* Enforce password rules on the server.

## **7.16 Security Misconfiguration**

4. Deprecated Interface

Severity: Low

**Description:**  
Legacy API endpoints remain publicly accessible even though they should be disabled.

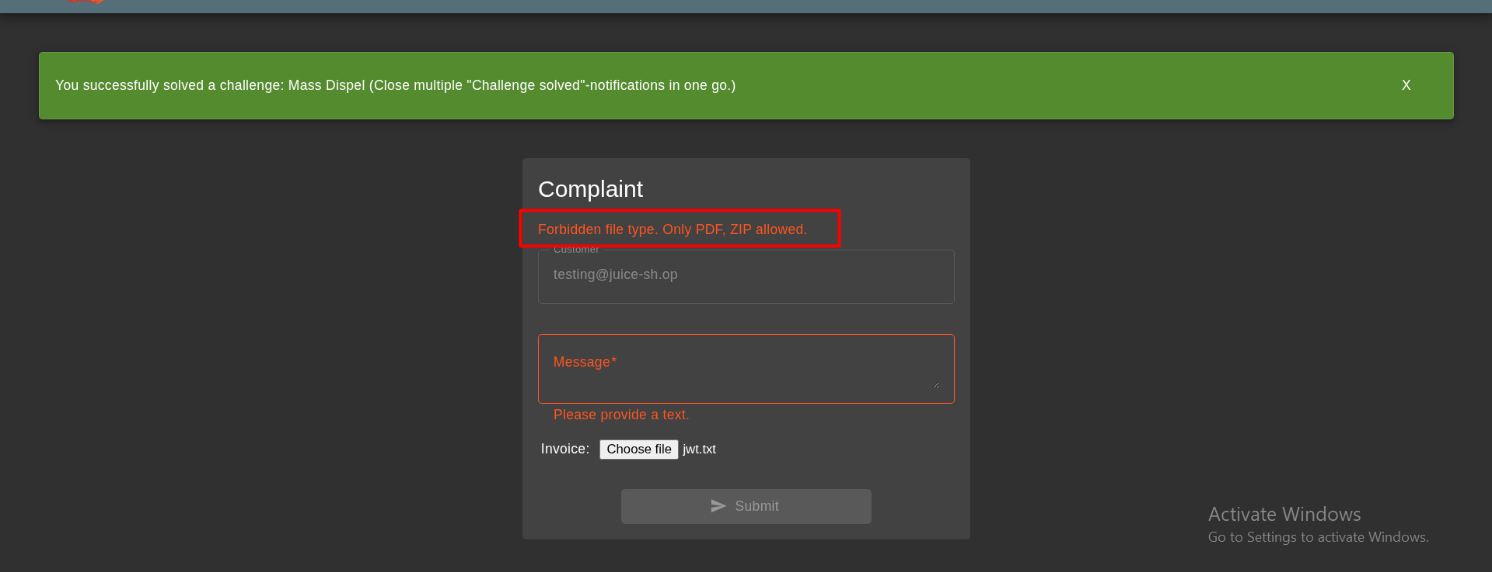
**Impact:**

* Attackers gain insights into old functionality.
* Potential entry point for exploitation if legacy code is vulnerable.

**Affected Endpoint/Parameter:**  
GET /rest/\*\*deprecated\*\*

**Proof of Concept (PoC):**

Searched the main.js about B2B and found a line that uploads pdf and xml and zip files in the complaint message   
A computer screen shot

AI-generated content may be incorrect.  
then when I inspected the complaint page it didn’t show that I can upload xml  
  
then I uploaded the xml file and it got uploaded A screenshot of a computer

AI-generated content may be incorrect.

Mitigation:

* Remove all deprecated endpoints.
* Implement strict API versioning.

## **7.17 Business Logic**

5. View Basket (IDOR)

Severity: Medium

**Description:**  
The application allows direct object reference manipulation, enabling viewing of other users’ baskets.

**Impact:**

* Unauthorized access to user data.
* Possible privacy violations.

**Affected Endpoint/Parameter:**  
GET /rest/basket/<basketId>

**Proof of Concept (PoC):**

* Modify basketId in request.
* Server returns another user's basket.

📌 [Insert Screenshot Here]

Mitigation:

* Enforce authorization checks on object IDs.

## **7.18 Security Misconfiguration**

6. CAPTCHA Bypass

Severity: Medium

**Description:**  
CAPTCHA validation occurs entirely on the client side and can be bypassed via request tampering.

**Impact:**

* Allows automated form submission.
* Increases spam and bot attacks.

**Affected Endpoint/Parameter:**  
POST /rest/user/login

**Proof of Concept (PoC):**

* Intercept login request.
* Change CAPTCHA value to any number or remove field.
* Server accepts it.

📌 [Insert Screenshot Here]

**Mitigation:**

* Validate CAPTCHA server-side.

## **7.19 Information Disclosure**

7. Database Schema Disclosure

Severity: Low

**Description:**  
Unhandled errors expose internal database structure (table names, fields).

**Impact:**

* Useful for SQL injection attacks.
* Assists attackers in understanding database layout.

**Affected Endpoint/Parameter:**  
GET /rest/products/search?q=' *(example)*

**Proof of Concept (PoC):**

* Trigger invalid SQL query.
* Server returns error containing schema information.

📌 [Insert Screenshot Here]

**Mitigation:**

* Return generic error messages.
* Avoid exposing internal database errors.

## **7.20 Business Logic**

8. Forged Review

Severity: High

**Description:**  
Attackers can submit product reviews as other users by manipulating the userId in the review request.

**Impact:**

* Impersonation of any user.
* Admin identity spoofing.
* Trust integrity compromised.

**Affected Endpoint/Parameter:**  
POST /rest/products/<id>/reviews

**Proof of Concept (PoC):**

{ "message": "Great product!",

"author": "2" // Modified ID = impersonation}

📌 [Insert Screenshot Here]

**Mitigation:**

* Bind review submissions to authenticated session user only.
* Ignore user-supplied IDs.

## **7.20 Business Logic**

9. Allowlist Bypass (Open Redirect)

Severity: Low

**Description:**  
Improper URL validation allows redirecting users to external websites.

**Impact:**

* Phishing attacks.
* User redirection to malicious domains.

**Affected Endpoint/Parameter:**  
GET /redirect?to=https://malicious.com

**Proof of Concept (PoC):**

* Replace to parameter with external domain.
* Browser redirects to attacker site.

📌 [Insert Screenshot Here]

Mitigation:

* Validate redirect targets against strict allowlist.

## **7.21 Business Logic**

10. Christmas Special (Hidden Feature)

Severity: Informational

**Description:**  
A hidden seasonal feature is triggered by accessing a secret client-side route.

**Impact:**

* No security impact.

**Affected Endpoint/Parameter:**  
GET /#/christmas-special

**Proof of Concept (PoC):**  
📌 [Insert Screenshot Here]

Mitigation:

* No fix required.

## **7.22 Business Logic**

11. Easter Egg (Hidden Route)

Severity: Informational

**Description:**  
An undocumented route reveals a hidden UI element or message.

**Impact:**

* No security risk.

**Affected Endpoint/Parameter:**  
GET /#/easter-egg

**Proof of Concept (PoC):**  
📌 [Insert Screenshot Here]

**Mitigation:**

* No fix needed.

# **8. Conclusion**

This assessment demonstrated that the OWASP Juice Shop application contains a wide range of vulnerabilities across multiple security categories, consistent with its design as a deliberately insecure training platform. The presence of critical authentication, authorization, business logic, and input validation weaknesses highlights the importance of secure development practices, input sanitization, and proper access controls.

Addressing these vulnerabilities would significantly improve the application's security posture and reduce the risk of compromise in a real-world deployment scenario.