WEB APPLICATION PENETRATION TESTING REPORT

[OWASP Juice Shop]

Digital Egypt Pioneers

وزارة الاتصالات
وتكنولوجيا المعلومات

Table of Contents

1. Date of Issue 4
2. Document Control 4
3. Executive Summary 4
3.1 Summary of results: 4
3.2 Activity Checklist 5
3.3 Graphical Summary 7
4. Scope of Engagement 8
5. Methodology (OWASP-Based) 9
5.1 Information Gathering (Reconnaissance) 9
5.2 Threat Modeling 9
5.3 Vulnerability Analysis 10
5.4 Exploitation 10
5.5 Post-Exploitation 10
5.6 Reporting 10
6. Technical Findings (POC Section) 11
6.1 Multiple SQL Injection 11
6.1.1 11
6.1.2 13
6.1.3 14
6.1.4 16
6.1.5 17
6.1.6 22
6.1.7 24
6.1.8 25
6.2 Broken Authentication 26
6.2.1 26
6.2.2 30
6.2.3 33
6.2.4 35
6.2.5 37
6.2.6 39
6.2.7 39
6.2.8 40
6.3 Broken Access Control / Authorization 41
6.3.1 41

6.3.2 42
6.3.3 43
6.3.4 45
6.3.5 46
6.3.6 46
6.3.7 47
6.3.8 49
6.3.9 51
6.3.10 52
6.4 Improper Input Validation 54
6.4.1 54
6.4.2 58
6.4.3 61
6.4.4 63
6.4.5 63
6.5 Cryptography 65
6.5.1 65
6.6 Cross-Site-Scripting (XSS) 68
6.6.1 68
6.6.2 68
6.6.3 68
6.6.4 68
6.6.5 68
6.6.6 68
6.6.7 68
6.7 Business Logic 69
6.7.1 69
6.7.2 69
6.7.3 72
6.7.4 75
6.7.5 75
6.7.6 77
6.8 Information Disclosure 79
6.8.1 79
6.8.2 82
6.8.3 88
6.8.4 88

6.8.5 90
6.8.6 91
6.8.7 92
6.8.8 93
6.8.9 93
6.8.10 95
6.9 Security Misconfiguration 96
6.9.1 96
6.9.2 98
6.9.3 98
6.9.4 100
6.9.5 101
6.9.6 102
6.9.7 104
6.9.8 106
6.10 Error Handeling 108
6.10.1 108
8. Conclusion 110

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

3.1 Summary of results:

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.

3.2 Activity Checklist

Category	Name (Challenges Under This Category)	Status
Injection	Database Schema	Failed
	Login Admin	Failed
	Login Jim	Failed
	Login bender	Failed
	NoSQL DOS	Failed
	NoSQL Manipulation	Failed
	Christmas Special	Failed
	Ephemeral Accountant	Failed
Broken Authentication	Reset Bender's Password	Failed
	Reset Jim's Password	Failed
	Login Bjoern	Failed
	Password Strength	Failed
	GDPR Data Erasure	Failed
	Empty User Registration	Failed
	Bjoern's Favorite Pet	Failed
	View Basket	Failed
Broken Access Control / Authorization	Admin Section	Failed
	GDPR Data Theft	Failed
	Manipulate Basket	Failed
	Admin Registration	Failed
	CSRF	Failed
	Leaked Unsafe Product	Failed
	Five-Star Feedback	Failed
	Forged Review	Failed
	Web3 Sandbox	Failed
	Easter Egg	Failed
Improper Input Validation	Payback Time	Failed
	Deluxe Fraud	Failed
	Zero Stars	Failed
	Mint the Honey Pot	Failed
	Missing Encoding	Failed
Cryptographic Issues	Weird Crypto	Failed
Cross-Site Scripting (XSS)	API-only XSS	Not Applicable
	Reflected XSS	Failed
	Client-side XSS Protection	Not Applicable
	DOM XSS	Failed

	CSP Bypass (XSS)	Not Applicable
	HTTP Header XSS	Not Applicable
	Bonus Payload	Failed
Business Logic Vulnerabilities	Forged Feedback	Failed
	Login MC SafeSearch	Failed
	Expired Coupon	Failed
	Repetitive Registration	Failed
	Mass Dispel	Failed
	Visual Geo Stalking	Failed
Information Disclosure	Exposed Credentials	Failed
	Login Amy	Failed
	NFT Takeover	Failed
	Forgotten Developer Backup	Failed
	Forgotten Sales Backup	Failed
	Meta Geo Stalking	Failed
	Misplaced Signature File	Failed
	Privacy Policy (public info)	Failed
	Confidential Document	Failed
	Exposed Metrics	Failed
Security Misconfiguration	Deprecated Interface	Failed
	Access Log	Failed
	Legacy Typosquatting	Failed
	Security Policy	Failed
	Outdated Allowlist	Failed
	Allowlist Bypass	Failed
	CAPTCHA Bypass	Failed
	Bully Chatbot	Failed
Error Handling	Error Handling	Failed

3.3 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:

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. Technical Findings (POC Section)

6.1 Multiple SQL Injection

6.1.1

Critical

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

Impact:

- Useful for SQL injection attacks.
- Assist attackers in understanding database layout.

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

Proof of Concept (PoC):

Instpect the sourcecode about the search and found this

Then tried this url and used the sql injection ‘

Then tried this command to know the version of the database “SELECT * FROM v$version;”

then used the union command t know the no of columns and tried from no 1-9 using this command “'))UNION SELECT '1','2','3','4','5','6','7','8','9' FROM sqlite_master—”

then removed the no of first column and wrote sql

Mitigation:

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

6.1.2

Critical

Description:

Weak authentication controls allow attackers to brute-force or guess the admin password.

Impact:

- Full admin account takeover.
- Complete compromise of the system.

Mitigation:

- Enforce strong passwords, rate limiting, CAPTCHA, and MFA.
- Lock accounts on repeated failed login attempts.

Reference: https://owasp.org/www-community/controls/Authentication_Cheat_Sheet

Affected Endpoint/Parameter: https://juice-shop.herokuapp.com/#/login

Proof of Concept (PoC):

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

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

We are in as Administrator

Mitigation:

Using Parameterized Query

6.1.3

Description:

Critical

Weak authentication controls allow attackers to brute-force or guess the admin password.

Impact:

- Full admin account takeover.
- Complete compromise of the system.

Reference: https://owasp.org/www-community/controls/Authentication_Cheat_Sheet

Affected Endpoint/Parameter: https://juice-shop.herokuapp.com/#/login

Proof of Concept (PoC):

Since the web application is already vulnerable to SQLi, and we only need an Email and Password to login…

Let's see what happens if we tried to login using another email…

And, We are in using a user : **Jim**

Mitigation:

- Enforce strong passwords, rate limiting, CAPTCHA, and MFA.
- Lock accounts on repeated failed login attempts.

6.1.4

Critical

Description: Similar to Login Jim; weak credentials can be brute-forced.

Impact:

- Unauthorized login.
- Leakage of personal or internal data.

Mitigation:

- Use strong passwords and MFA.
- Implement rate limiting.

Reference: https://owasp.org/www-community/controls/Authentication_Cheat_Sheet

Affected Endpoint/Parameter: https://juice-shop.herokuapp.com/#/login

Proof of Concept (PoC):

Once the pentester logged with a user creds, he tries to login with a different user

Now the pentester make sure that he can login with multiple users since the web app is vulnerable to SQLi

And we are in again using different user this time: **bender**

6.1.5

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.

3.Analyzing the Request

The intercepted request for modifying a comment looks like this:

```
PUT /rest/products/30/reviews HTTP/1.1
Host: 127.0.0.1:3000
Content-Length: 55
sec-ch-ua-platform: "Linux"
...
{"message":"shsherdkjkkm","author":"admin@juice-sh.op"}
```

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:

```
{"message":"shsherdkjkkm","author":"sleep(30000)"}
```

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

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

```
PUT /rest/products/sleep(20000)/reviews HTTP/1.1
Host: localhost:3000
```

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
```

The server become unresponsive

Mitigation:

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

6.1.6

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 `

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.

2. 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:

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).

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.

6.1.7

High

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

Impact:

- Getting an old offer and reuse it

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

Proof of Concept (PoC):
tried this url to get all the products

Then inspected the source code a searched for “Christmas special”

Then added a product to the basket and changed it’s id in the request

Request

Pretty Raw Hex

```
1  POST /api/BasketItems/ HTTP/1.1
2  Host: localhost:3000
3  Content-Length: 44
4  sec-ch-ua-platform: "Linux"
5  Authorization: Bearer
   eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW
   1haWwiOiJ0ZXN0aW5nQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiJiNjE2YTY0NjA1YTA3OTQxZmJkMzE4NjhhZWEzYjU0YiIsInJvbGUiO
   iJhZGlpbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGlj
   L2ltYWdlcy9lcGxvYWRzL2RlZmFlbHRBZGlpbi5wbmciLCJ0b3RwU2VjcmVOIjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6Ij I
   wMjUtMTEtMjUgMjA6MDE6MzkuODI4ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMTEtMzAgMTE6NTQ6NDcuNTgzICswMDowMCIsImRlbG
   V0ZWRBdCI6bnVsbHOsImlhdCI6MTc2NDUyNTg5MX0.EyNooRxfBSzRSvQGq0zlL9JTiqBlLcqyc6_VcqkwtjKH6KHoXY5k9-GV2Z1-L5Bcx
   PXhyC-mxSIUw113sKljMLAsyx1fGG_Wb2dQc3Otx5uuHDjYPHWKIlciq0C-D8RF1zUg7Q1RSp6ohjWcCcOfPOZyCxbYS8xDChhMa51Ord8
6  Accept-Language: en-GB,en;q=0.9
7  sec-ch-ua: "Not_A Brand";v="99", "Chromium";v="142"
8  sec-ch-ua-mobile: ?0
9  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0
   Safari/537.36
10 Accept: application/json, text/plain, */*
11 Content-Type: application/json
12 Origin: http://localhost:3000
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer: http://localhost:3000/
17 Accept-Encoding: gzip, deflate, br
18 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss;
   code-fixes-component-format=LineByLine; continueCodeFindIt=
   l7PODGv6dpoWXMgm2qRb9P7zJZ0yjtr7uyM4j3nyQKk5YElNBaVLOw1erxb2; continueCodeFixIt=
   lND8dDB6PeO3q2yLjMWE7orvZxbKkhBRTq7Abkpm9zSQVn0KGwY4XRgNJlMB; continueCode=
   mgZd5OtJSDtmcoIETZf4hXZuNnukWhJYIyDsDMtvzukgcmKfgbHOKFm8OyL6; token=
   eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW
   1haWwiOiJ0ZXN0aW5nQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiJiNjE2YTY0NjA1YTA3OTQxZmJkMzE4NjhhZWEzYjU0YiIsInJvbGUiO
   iJhZGlpbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGlj
   L2ltYWdlcy9lcGxvYWRzL2RlZmFlbHRBZGlpbi5wbmciLCJ0b3RwU2VjcmVOIjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6Ij I
   wMjUtMTEtMjUgMjA6MDE6MzkuODI4ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMTEtMzAgMTE6NTQ6NDcuNTgzICswMDowMCIsImRlbG
   V0ZWRBdCI6bnVsbHOsImlhdCI6MTc2NDUyNTg5MX0.EyNooRxfBSzRSvQGq0zlL9JTiqBlLcqyc6_VcqkwtjKH6KHoXY5k9-GV2Z1-L5Bcx
   PXhyC-mxSIUw113sKljMLAsyx1fGG_Wb2dQc3Otx5uuHDjYPHWKIlciq0C-D8RF1zUg7Q1RSp6ohjWcCcOfPOZyCxbYS8xDChhMa51Ord8
19 Connection: keep-alive
20
21 {
     "ProductId":10,
     "BasketId":"6",
     "quantity":1
   }
```

then completed the checkout process and got the offer

6.1.8

Medium

Description

The login endpoint is vulnerable to SQL Injection via UNION SELECT. Attackers can craft a payload that results in issuing a valid JWT for a non-existent, temporary user.

Impact

- Authentication bypass
- Forged or temporary user identities

- Unauthorized system access

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

Affected Endpoint/Parameter:

Proof of Concept (PoC):

Mitigation

- Use parameterized queries or prepared statements
- Sanitize all user input
- Verify actual user records exist before issuing tokens

6.2 Broken Authentication

6.2.1

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.**

2. **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"}

3. **Send the Request to Intruder**

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

4. Load the Wordlist

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

```
BendingUnit
BenderBendingUnit
SuicideBooth
Stop'n'Drop
StopAndDrop
RobotArms
BendersWork
BendersJob
BenderWorkplace
BenderEmployer
BenderCompany
BenderFactory
BenderPlant
BenderIndustry
BenderCorporation
BenderInc
BenderLtd
BenderCo
```

5. 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`

6. **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.

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.

6.2.2

Critical

Description:

Faulty password reset logic allows attackers to reset Jim's password without identity verification.

Impact:

- Full account takeover.
- Privacy breach and potential abuse.

Mitigation:

- Use secure, random, time-limited reset tokens.
- Validate identity through email or MFA.

- Log suspicious resets.

Reference: https://owasp.org/www-community/vulnerabilities/Weak_password_reset

Affected Endpoint/Parameter: https://juice-shop.herokuapp.com/#/forget-password

Proof of Concept (PoC):

Once the pentester login to an account, he tries to change its credentials

We can see here that there is a security question should be answered to **Authorize** user

Let's send the request to the intruder, the try to brute-force this security question

As we see here we have a successful try under the name **Samuel**

Now let’s try to login with new password we created…

And we are in!!

6.2.3

Critical

Description:

Log in with Bjoern's Gmail account without previously changing his password, applying SQL Injection, or hacking his Google account.

Impact:

The Bjoern Login vulnerability allows an attacker to bypass authentication mechanisms by exploiting weak or improperly implemented login logic. This can lead to:

- Unauthorized access to user accounts, including administrative accounts.

- Exposure of sensitive personal data.
- Potential manipulation of user-specific actions, such as orders or settings.
- Compromise of application integrity and trustworthiness.

Affected Endpoint/Parameter:

Affected Endpoint: /#/login

Proof of Concept (PoC):

1. Go to login page https://juice-shop.herokuapp.com/#/login
2. Right-click then choose Inspect
3. Go to Console and right the code in the screenshot
4. You will get the password
5. You can login with bjoern email and password

Mitigation:

Secure Password Generation: Avoid using predictable algorithms for password generation, and most importantly, don't use them client side. Use strong, random, and non-reversible methods.

6.2.4

Critical

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: http://localhost:3000/#/about

Proof of Concept (PoC):

After inspecting the customers reviews on the products I found the admins mail

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

Then I got the password of the admin

Mitigation:

- Enforce password rules on the server.

6.2.5

High

Description:

Log in with Chris' erased user account

Impact:

The GDPR Data Erasure vulnerability allows attackers or users to exploit weaknesses in the account deletion process. Improper handling can lead to:

- Residual personal data remaining accessible even after deletion requests.
- Unauthorized access to supposedly deleted accounts.

Affected Endpoint/Parameter:

POST /rest/user/login

Proof of Concept (PoC):

1. Go to https://juice-shop.herokuapp.com/#/login

2. Inject ' in Email and write any password

3. Notice the error appeared in the response of Burp Suite meaning that there is a SQL injection

4. Focus on the response and you will see **DeletedAt IS NOT NULL --**

5. Back to login page, inject in the Email **' OR DeletedAt IS NOT NULL --** and any password you want in the password

6. You will see in the response in Burp Suite that you take over Chris's account

Mitigation:

- Using prepared statements with bound parameters effectively prevents SQL injection attacks.
- Implementing multi-factor authentication strengthens login security and reduces unauthorized access risks.
- Proper handling of account deletion or deactivation ensures that removed accounts cannot be exploited.

6.2.6

High

6.2.7

Medium

Description

An admin's email is visible, and public social media reveals the answer to his security question. Attackers use this OSINT to reset his account password.

Impact

- Account takeover using publicly available information
- Leakage of sensitive administrative information

Reference: https://owasp.org/www-community/vulnerabilities/Information_exposure_through_other_means

Affected Endpoint/Parameter:

Proof of Concept (PoC):

Mitigation

- Remove security questions entirely
- Use MFA or email verification for password resets
- Hide sensitive administrative details from regular users

6.2.8

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):

I went to the basket page and inspected the request that was sent to the server

Request

Pretty Raw Hex

```
1  GET /rest/basket/6 HTTP/1.1
2  Host: localhost:3000
3  sec-ch-ua-platform: "Linux"
4  Authorization: Bearer
   eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYWllIjoiIiwiZW
   1haWwiOiJ0ZXN0aW5nQGplaWNlLXNoLm9wIiwicGFzc3dvcmQiOiJiNjE2YTY0NjA1YTA3OTQxZmJkMzE4NjhhZWEzYjU0YiIsInJvbGUiO
   iJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGlj
   L2ltYWdlcy91cGxvYWRzL2RlZmFlbHRBZG1pbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjI
   wMjUtMTEtMjUgMjA6MDE6MzkuODI4ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMTEtMzAgMTE6NTQ6NDcuNTgzICswMDowMCIsImRlbG
   V0ZWRBdCI6bnVsbH0sImlhdCI6MTc2NDUwMzcwN30.MJ8yu3l5fqwczfIrtlC2TnfcgNNIwjdntCdG58YwGl_34xsrZOm6OAhHuWnWHgLpX
   cuUCAwUJwPD-M3BCAfpNSgGdstv4c2CRFg6IfKBjWmbDD5NEionC4-RqwyU_qjNCe6vAArtzsem5v5CvRlf2RZrmm2OPVY4w0DmfqNSZlo
5  Accept-Language: en-GB,en;q=0.9
6  Accept: application/json, text/plain, */*
7  sec-ch-ua: "Not_A Brand";v="99", "Chromium";v="142"
8  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0
   Safari/537.36
9  sec-ch-ua-mobile: ?0
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: http://localhost:3000/
```

then I changed the id of the basket and received the response

Mitigation:

- Enforce authorization checks on object IDs.

6.3 Broken Access Control / Authorization

6.3.1

Critical

Description:

Admin dashboard is accessible without proper authorization.

Impact:

- Attackers can access admin functionalities.
- Unauthorized data modification or deletion.
- Protect all admin endpoints with session validation.

Reference: https://owasp.org/www-community/attacks/Access_Control_Vulnerabilities

Affected Endpoint/Parameter: https://juice-shop.herokuapp.com/#/

Proof of Concept (PoC):

After logging in as administrator, pentester see the main files into this website looking for any important indicators

Here we see there is a path called **administration,** can hold something…

After trying navigating to this path we found this path contains all registered users…

Mitigation:

- Implement RBAC and strict server-side authorization checks.

6.3.2

Critical

Description

The GDPR export feature matches users using obfuscated email strings. Attackers can craft an email that fits the same obfuscated pattern, resulting in unauthorized data exports.

Impact

- Theft of GDPR export data
- Severe privacy and compliance violations

OWASP Reference

https://owasp.org/www-community/vulnerabilities/Information_exposure

Affected Endpoint/Parameter:

Proof of Concept (PoC):

Mitigation

- Use real, unique identifiers instead of obfuscated strings
- Apply strict authentication and authorization checks
- Validate export requests against logged-in user identity

6.3.3

High

Description:

Put an additional product into another user's shopping basket

Impact:

Attackers can modify basket contents, quantities, or prices through client-side tampering.

Without server-side validation, items can be purchased for free or at reduced prices.

Unauthorized access to restricted products or features is possible.

Financial loss and disruption of normal business operations may occur.

Affected Endpoint/Parameter:

POST /rest/deluxe-membership

Proof of Concept (PoC):

1. Begin by navigating to the product search page at **(https://juice-shop.herokuapp.com/#/search)** and add any product to the basket.
2. Proceed through the purchasing process to observe the basket and checkout requests being generated.

3. By reviewing these requests, it becomes evident that item prices, quantities, and identifiers are controlled client-side without proper validation.

4. Modifying these values shows that the server accepts manipulated data, demonstrating how the basket can be altered improperly as shown in the burp screen.

5. You can add any product to other users' accounts.

Mitigation:

- Ensure Proper Parameter Handling: Servers should be designed to handle unexpected, duplicated, or out-of-order parameters securely.
- Implement Robust Access Controls: Ensure that all sensitive operations verify the user's permission to perform the action on the specified resource.

6.3.4

High

Description:

The registration request contains a "role" field. Attackers can modify this value to "admin" and create privileged accounts due to missing server-side role enforcement.

Impact:

- Unauthorized creation of admin accounts
- Full system compromise

Reference: https://owasp.org/www-community/vulnerabilities/Privilege_Escalation

Affected Endpoint/Parameter:

Proof of Concept (PoC):

Mitigation:

- Do not accept role assignments from clients
- Enforce RBAC strictly on the server side
- Whitelist valid roles and apply them internally

6.3.5

High

6.3.6

High

Description

The product search functionality is vulnerable to SQL Injection using UNION SELECT. Attackers can retrieve deleted or hidden products, including unsafe items.

Impact

- Exposure of deleted or sensitive product entries
- Leakage of hazardous product details
- Bypassing intended business logic restrictions

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

Affected Endpoint/Parameter:

Proof of Concept (PoC):

Mitigation

- Sanitize and parameterize SQL queries
- Restrict access to deleted or sensitive records
- Implement strong backend validation checks

6.3.7

Low

Description:

Client-side tampering allows an attacker to force a 5-star review regardless of the input.

Impact:

- Fake reviews.
- Manipulation of business reputation.

Reference: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/11-Testing_for_Business_Logic

Affected Endpoint/Parameter:

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

Proof of Concept (PoC):

After pentester found the administration path he can tries to modify or edit on something like we see here:

He can see some registered users data and he also can delete their rates!!

See here the difference between the two images that there is a deleted **Five star feedback**

Mitigation:

- Validate rating values on the server side.
- Block tampered parameters.
- Log suspicious activity.

6.3.8

Low

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:

http://localhost:3000/rest/captcha

Proof of Concept (PoC):

I inspected the source code and found this path

```
    )()
      , le = ( () => {
        class n {
            http = (0,
            i.WQX)(w.Qq);
            hostServer = x.c.hostSer...,
            host = this.hostServer + "/rest/captcha";
            getCaptcha() {
                return this.http.get(this.host + "/").pipe((0,
                g.W)(e => {
                    throw e
                }
                ))
            }
```

when I tried using this path it shows me a captcha id and its answers every time it is loaded

http://localhost:3000/rest/captcha

Pretty print

{"captchaId":12,"captcha":"3*9-4","answer":"23"}

Than used this id in a request

Request (Pretty / Raw / Hex)

```
2  Host: localhost:3000
3  Content-Length: 94
4  sec-ch-ua-platform: "Linux"
5  Authorization: Bearer
   eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYWllIjoiIiwiZW
   lhaWwiOiJ0ZXN0aW5nQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiJ1NjE2YTY0NjA1YTA3OTQxZmJkMzE4NjhhZWEzYjU0YiIsInJvbGUiO
   iJhZGlpbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGlj
   L2ltYWdlcy91cGxvYWRzL2RlZmFlbHRBZGlpbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjI
   wMjUtMTEtMjUgMjA6MDE6MzkuODI4ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMTEtMzAgMTE6NTQ6NDcuNTgzICswMDowMCIsImRlbG
   V0ZWRBdCI6bnVsbH0sImlhdCI6MTc2NDUwMzcwN30.MJ8yu3l5fqwczfIrtlC2TnfcgNNIwjdntCdG58YwGl_34xsrZOm60AhHuWnWHgLpX
   cuUCAwUJwPD-M3BCAfpNSgGdstv4c2CRFg6IfKBjWmbDD5NEionC4-RqwyU_qjNCe6vAArtzsem5v5CvRlf2RZrmm2OPVY4w0DmfqNSZlo
6  Accept-Language: en-GB,en;q=0.9
7  sec-ch-ua: "Not_A Brand";v="99", "Chromium";v="142"
8  sec-ch-ua-mobile: ?0
9  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/142.0.0.0
   Safari/537.36
10 Accept: application/json, text/plain, */*
11 Content-Type: application/json
12 Origin: http://localhost:3000
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer: http://localhost:3000/
17 Accept-Encoding: gzip, deflate, br
18 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss;
   code-fixes-component-format=LineByLine; continueCodeFindIt=
   l7PODGv6dpoWXMgm2qRb9P7zJZOyjtr7uyM4j3nyQKk5YElNBaVLOwlerxb2; continueCodeFixIt=
   lND8dDB6PeO3q2yLjMWE7orvZxbKkhBRTq7Abkpm9z5QVnOKGwY4XRgNJlMB; token=
   eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYWllIjoiIiwiZW
   lhaWwiOiJ0ZXN0aW5nQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiJ1NjE2YTY0NjA1YTA3OTQxZmJkMzE4NjhhZWEzYjU0YiIsInJvbGUiO
   iJhZGlpbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIxMjcuMC4wLjEiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGlj
   L2ltYWdlcy91cGxvYWRzL2RlZmFlbHRBZGlpbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjI
   wMjUtMTEtMjUgMjA6MDE6MzkuODI4ICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMTEtMzAgMTE6NTQ6NDcuNTgzICswMDowMCIsImRlbG
   V0ZWRBdCI6bnVsbH0sImlhdCI6MTc2NDUwMzcwN30.MJ8yu3l5fqwczfIrtlC2TnfcgNNIwjdntCdG58YwGl_34xsrZOm60AhHuWnWHgLpX
   cuUCAwUJwPD-M3BCAfpNSgGdstv4c2CRFg6IfKBjWmbDD5NEionC4-RqwyU_qjNCe6vAArtzsem5v5CvRlf2RZrmm2OPVY4w0DmfqNSZlo;
    continueCode=v6qQyD0b9toSRtLcQIDTvfxhb9uJ7ho5Ir2sOLtXQuj5fNjIP3ilPGVlJ4n3
19 Connection: keep-alive
20
21 {
       "UserId":22,
       "captchaId":12,
       "captcha":"23",
       "comment":"smsms (***ting@juice-sh.op)",
       "rating":5
   }
```

Response (Pretty / Raw / Hex / Render)

```
1  HTTP/1.1 201 Created
2  Access-Control-Allow-Origin: *
3  X-Content-Type-Options: nosniff
4  X-Frame-Options: SAMEORIGIN
5  Feature-Policy: payment 'self'
6  X-Recruiting: /#/jobs
7  Location: /api/Feedbacks/43
8  Content-Type: application/json; charset=utf-8
9  Content-Length: 178
10 ETag: W/"b2-SR5WFH2Pzy99iGa6ieZnF7NQdXk"
11 Vary: Accept-Encoding
12 Date: Sun, 30 Nov 2025 17:04:06 GMT
13 Connection: keep-alive
14 Keep-Alive: timeout=5
15
16 {
       "status":"success",
       "data":{
           "id":43,
           "UserId":22,
           "comment":"smsms (***ting@juice-sh.op)",
           "rating":5,
           "updatedAt":"2025-11-30T17:04:06.818Z",
           "createdAt":"2025-11-30T17:04:06.818Z"
       }
   }
```

Mitigation:

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

6.3.9

Low

Description:

The application exposes the Web3 Sandbox feature publicly. This interface allows users to perform Web3 RPC calls (e.g., eth_blockNumber, eth_call) without authentication. This functionality is intended for development and should not be accessible in a production environment.

Impact:

- Expands the attack surface by exposing internal testing tools.
- Allows attackers to interact with blockchain-related functions that should not be publicly available.
- Enables reconnaissance that may reveal internal logic or misconfigurations.

Affected Endpoint/Parameter:

/#/web3-sandbox

Proof of Concept (PoC):

Navigate to:

https://juice-shop.herokuapp.com/#/web3-sandbox

Execute a Web3 RPC request, such as:

The application returns Web3 responses without requiring authentication.

Mitigation:

- Disable the Web3 Sandbox in production.
- Restrict access to this feature to developers only.
- Add authentication or role-based access control around debugging interfaces.
- Remove or hide unnecessary RPC functionality.

6.3.10

Info

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):
tried the /robots.txt to get all the files

Then I tried this URL http://localhost: 3000/ftp

Then to download the file I used the null byte %00.md and it got downloaded

Mitigation:

- No fix needed.

6.4 Improper Input Validation

6.4.1

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:

` http://127.0.0.1:3000/# /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.

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

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

Payload used : ` accountant@juice-sh.op' -- - `

Now I logged in as `accountant@juice-sh.op'

2. **Exploiting the Business Logic**

1.1 Access Accounting Page as the Accountant:

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

1.2 Attempt to Modify Product Prices:

Intercepted the price update request using Burp Proxy.

1.3 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

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.

6.4.2

High

Description:

Obtain a Deluxe Membership without paying for it.

Impact:

The Deluxe Fraud vulnerability allows an attacker to bypass the payment process and gain unauthorized access to the Deluxe Membership without actually completing a payment. This can lead to:

Unauthorized access to premium features or content.

Loss of revenue for the application.

Exposure of weaknesses in server-side validation and payment verification logic.

Affected Endpoint/Parameter:

POST /rest/deluxe-membership

Proof of Concept (PoC):

1. Go to https://juice-shop.herokuapp.com/#/deluxe-membership
2. Choose Become a member
3. Complete payment options process
4. Open Burp Suite, then open Intercept
5. Click confirm, go to the burp to see the request intercepted
6. Remove card and forward the request
7. You became a member without paying

Improper Input Validation
Deluxe Fraud
Obtain a Deluxe Membership without paying for it.
0/4

Mitigation:

- Rigorous input validation ensures only legitimate and correct data is processed.
- Securing payment logic prevents unauthorized or incorrect transactions from being executed.
- Integrating trusted payment gateways adds extra verification and protection.

6.4.3

Low

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 `

Reference

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

Proof of Concept (PoC):

1. **Accessing Customer Feedback**

After logging into the application, I navigated directly to the **Customer Feedback**

2. **Submitting a Normal Review**

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

3. **Intercepting the Request**

4. Using Burp Suite's Proxy feature, I captured the outgoing feedback submission request.

5. **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**.

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.

6.4.4

Low

6.4.5

Description:

Retrieve the photo of Bjoern's cat in "melee combat-mode".

Impact:

- May allow reflected or stored XSS in specific UI components.
- Attackers can inject scripts that steal session data or manipulate the interface.

Affected Endpoint/Parameter:

/#/photo-wall

Proof of Concept (PoC):

- Navigate to https://juice-shop.herokuapp.com/#/photo-wall
- Right-click on the " 😸 " icon and inspect it.
- Replace # in the url with %23 (url encoding of #)
- Refresh the page

Mitigation:

The primary remediation step here is to ensure that all URLs, especially those that might contain non-standard characters, are properly URL encoded before being embedded in web pages. This can generally be handled in two ways:

- Backend Solution: Ensure that the server-side code that generates URLs encodes them before sending them to the client.
- Client-Side Checks: Implement client-side checks or functions that encode URLs as needed before using them in an HTML document.

6.5 Cryptography

6.5.1

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:

` http://127.0.0.1:3000/#/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**.

The payload included sensitive user data, including the admin's password hash.

2. Decode JWT

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'

4. Login as Admin

Using the credentials:

Email: admin@juice-sh.op
Password: admin123

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.

6.6 Cross-Site-Scripting (XSS)

6.6.1

6.6.2

6.6.3

6.6.4

6.6.5

6.6.6

6.6.7

6.7 Business Logic

6.7.1

Critical

Description

The application trusts client-specified user IDs in feedback submissions. Attackers can modify this value to impersonate any user, including administrators.

Impact

- Full user impersonation
- Ability to submit messages as admin
- Breakdown of authentication and authorization boundaries

Reference: https://owasp.org/www-community/vulnerabilities/Parameter_Tampering

Mitigation

- Validate identity server-side
- Do not trust user-provided IDs
- Bind feedback to authenticated session data only

6.7.2

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:

` http://127.0.0.1:3000/#/login `

Reference https://owasp.org/Top10/A07_2021-Broken_Authentication/

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.

Proof of Concept (PoC):

1. OSINT Search

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

"MC SafeSearch" "password"

2. 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. Building the Password

Using the extracted information:

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

Capitalization preserved

Final Password : Mr. N00dles

2. Login Attempt

Using the discovered credentials:

- **Username:** mc.safesearch@juice-sh.op
- **Password:** Mr. N00dles

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

6.7.3

Medium

Description:

Attackers can redeem expired or invalid coupons by manipulating requests.

Impact:

- Direct financial loss.
- Abuse of pricing logic.

Reference: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/11-Testing_for_Business_Logic

Affected Endpoint/Parameter: https://juice-shop.herokuapp.com/ftp

Mitigation:

- Validate coupons strictly on the server.
- Enforce expiration date and usage limits.
- Log misuse attempts.

Proof of Concept (PoC):

In the same file the pentester found in the path **/robots.txt** that one navigates to **/ftp**

We can see there is a file called **Coupons_2013.md.bak**

When opening this file we find there are many coupons that is expired…

After searching for those coupons, we found the coupons are (z85 Encoded)

When decoding it we found it representing a (month + year + ’-‘ + coupon percentage)

Once we know how, we can generate a new coupon using a new date:

Now let's try to use it…

It actually worked!!

6.7.4

Medium

Description

The registration form validates password confirmation only on the client side. By modifying the HTML, attackers can submit mismatched passwords because the backend does not verify the confirmation field.

Impact

- Users can register with inconsistent passwords
- Backend allows tampered requests
- Demonstrates lack of server-side validation

OWASP Reference

https://owasp.org/www-community/vulnerabilities/Improper_Input_Validation

Affected Endpoint/Parameter:

Mitigation

- Enforce server-side validation for password confirmation
- Reject registration requests with mismatched passwords
- Avoid relying on client-side validations for authentication logic

Proof of Concept (PoC):

6.7.5

Medium

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.

This was the home page before :

and this after pressing the “Shift” button

Mitigation:

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

6.7.6

Medium

Description:

Determine the answer to Emma’s security question by looking at an upload of her to the Photo Wall and use it to reset her password via the Forgot Password mechanism.

Impact:

The Visual Geo Stalking vulnerability allows an attacker to gather sensitive personal information from publicly shared images (e.g., workplace details or location clues) to answer security questions. This can lead to unauthorized account access, password resets, and potential full account takeover

Affected Endpoint/Parameter:

/#/forgot-password

Mitigation:

By increasing user awareness, strengthening privacy controls, and promoting the use of secure and non-obvious security question answers, individuals can greatly reduce the risk of sensitive data exposure through publicly accessible information. Together, these practices create a stronger layer of protection against social engineering and unauthorized access.

Proof of Concept (PoC):

1. Go to https://juice-shop.herokuapp.com/#/photo-wall, the last photo of Emma
2. The clues in Emma's photo, including the building and caption, guided the initial search for her former workplace.
3. A closer look at the image revealed the crucial "ITsec" sign, which ultimately provided the key detail needed.
4. Using "ITsec" as the security question answer provided the solution required to proceed.
5. Go to Forgot your password at: https://juice-shop.herokuapp.com/#/forgot-password
6. Enter the email emma@juice-sh.op, then the answer of security question and account takeover is achieved

6.8 Information Disclosure

6.8.1

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:

` http://127.0.0.1:3000/main.js `

Reference

https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password

[Use of hard-coded password]

Mitigation

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

Proof of Concept (PoC):

1. **Navigate to the Application**

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

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

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

3. **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";

4. **Attempt Login Using Exposed Credentials**

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

6.8.2

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:

` http://127.0.0.1:3000/#/forgot-password `

Reference

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

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.

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

2. **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

While exploring the site, I read the **“One Important Final Note”**, which advised:

One Important Final Note

The example with “D0g.....” should not be taken literally because if everyone began padding their passwords with simple dots, attackers would soon start adding dots to their guesses to bypass the need for full searching through **unknown** padding. Instead, **YOU should invent** your own **personal padding policy**. You could put some padding in front, and/or interspersed through the phrase, and/or add some more to the end. You could put some characters at the beginning, padding in the middle, and more characters at the end. And also mix-up the padding characters by using simple memorable character pictures like “**<->**” or “**[*]**” or “**^~^**” **. . .** but do invent your own!

If you make the result long **and** memorable, you'll have super-strong passwords that are also easy to use!

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

3. **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.

GRC's Interactive Brute Force Password "Search Space" Calculator

(NOTHING you do here ever leaves your browser. What happens here, stays here.)

1 Uppercase | 1 Lowercase | 1 Digit | 21 Symbols | 24 Characters

A8y . . . . . .

Enter and edit your test passwords in the field above while viewing the analysis below.

Brute Force Search Space Analysis:

Search Space Depth (Alphabet):	26+26+10+33 = **95**
Search Space Length (Characters):	24 characters
Exact Search Space Size (Count): (count of all possible passwords with this alphabet size and up to this password's length)	295,095,290,555, 142,625,648,321,021,999, 764,315,625,454,517,120
Search Space Size (as a power of 10):	2.95×10^{47}

Time Required to Exhaustively Search this Password's Space:

Online Attack Scenario: (Assuming one thousand guesses per second)	93.83 billion trillion trillion centuries
Offline Fast Attack Scenario: (Assuming one hundred billion guesses per second)	9.38 hundred trillion trillion centuries
Massive Cracking Array Scenario: (Assuming one hundred trillion guesses per second)	9.38 hundred billion trillion centuries

Note that typical attacks will be online password guessing limited to, at most, a few hundred guesses per second.

-

4. Researching the Character "Amy"

To generate candidate passwords, I researched **Amy Wong** from *Futurama*:

Personality

Amy is friendly and easily gets along with others but she is also a bit of a klutz. She has attracted several men in her time, most notably Kif Kroker, her fonfon ru. Amy also appears to have some level of vanity and is concerned about her appearance. She is engaged in a constant low-intensity feud with Leela, the only other female at Planet Express, whom she often accuses of not being ladylike enough to be a true woman. Even so, they remain friends.

Despite her usual intelligence, Amy can be absent-minded. A notable example being removing Dr. Zoidberg's restraints no less than eight times, despite injuries she endured each time. Amy also has little concept of money, having offered to pay more for a hovercar than its sticker price and needing to be reminded by Leela that $10,000 is more than $5,000.

Relatives See family section
First appearance "The Series Has Landed" (1ACV02)
Voiced by Lauren Tom

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

5. Building a Wordlist

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

```
K1f.......................
Kif.......................
KifKroker................
AmyKif...................
AmyKif2025...............
Fonfonru.................
F0nf0nru.................
AmyFonfonru..............
Leela....................
AmyLeela.................
Zoidberg.................
AmyZoidberg..............
KlutzAmy.................
AmyVain..................
AbsentAmy................
Hovercar.................
Amy10000.................
Amy5000..................
Amy1.....................
AMY123...................
A.my.....................
Amy......................
AmyLovesKif..............
AmyLovesKif..............
KifKroker2025............
```

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

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

4. **Brute-Forcing Amy's Password**

Go to login page to log with amy@juice-sh.op account

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

I monitored the responses to identify a successful login.

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
1	K1f.....	200	634			1179	Contains a JWT
0		401	73			413	
2	Kif.....	401	617			413	
3	KifKroker.....	401	640			413	
4	AmyKif.....	401	701			413	
5	AmyKif2025.....	401	781			413	
6	Fonfonru.....	401	773			413	
7	F0nf0nru.....	401	837			413	
8	AmyFonfonru.....	401	870			413	
9	Leela.....	401	927			413	
10	AmyLeela.....	401	885			413	
11	Zoidberg.....	401	790			413	
12	AmyZoidberg.....	401	743			413	
13	KlutzAmy.....	401	656			413	

5.Identifying the Valid Credential

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

`K1f.....`

6.8.3

Critical

Description:

A user review publicly contains a full 12-word seed phrase. This allows anyone to derive the private key and access the NFT wallet, resulting in full compromise.

Impact:

- Complete NFT wallet takeover
- Unauthorized access to SBTs
- Exposure of cryptographic secrets

Reference: https://owasp.org/www-community/vulnerabilities/Sensitive_Data_Exposure

Affected Endpoint/Parameter:

Mitigation:

- Never store or display seed phrases in clear text
- Sanitize user-generated content
- Block submissions matching cryptographic phrase patterns

Proof of Concept (PoC):

6.8.4

High

Description:

A development backup file is publicly exposed, revealing internal application code or secrets.

Impact:

- Source code disclosure.
- Leakage of credentials, API keys, or internal configs.

Mitigation:

- Remove all backup and development files from production.
- Use automated scans to detect exposed files.

Reference: https://owasp.org/www-community/vulnerabilities/Information_Leakage

Affected Endpoint/Parameter:

https://juice-shop.herokuapp.com/ftp

Proof of Concept (PoC):

In the same exposed file **/ftp,** We can see a lot of files we can access

We can see this file which contains very important information about the application including dependencies which contains also the encoding method (z85):

6.8.5

High

Description:

A publicly exposed sales backup provides attackers with commercially sensitive internal data.

Impact:

- Exposure of private business metrics.
- Competitive and financial risk.
- Violates data protection policies.

Mitigation:

- Remove unnecessary backups from server directories.
- Apply proper access control to all stored files.
- Encrypt sensitive backups.

Reference: https://owasp.org/www-community/vulnerabilities/Sensitive_Data_Exposure

Affected Endpoint/Parameter:

https://juice-shop.herokuapp.com/ftp

Proof of Concept (PoC):

After reviewing the same content in the path **/ftp,** We can notice there are many files including a lot of valuable information that can be used to compromise this application

This file **Coupons_2013** Contains an expired coupons that can be used again if we changed a little something in it…

```
n<MibgC7sn
mNYS#gC7sn
oIVigC7sn
k#pDlgC7sn
oI]pgC7sn
n(XRvgC7sn
n(XLtgC7sn
k#*AfgC7sn
q:<IqgC7sn
pEw8ogC7sn
pes[BgC7sn
l}6D$gC7ss
```

Coupons

6.8.6

Medium

Description:

Uploaded images retain EXIF metadata including GPS coordinates. Attackers can extract GPS data to identify physical locations and exploit this information to reset passwords.

Impact:

- Unauthorized password resets
- Exposure of real-world user locations
- Enables targeted attacks through metadata

Reference:

https://owasp.org/www-community/vulnerabilities/Exposure_of_Private_Personal_Information

Affected Endpoint/Parameter:

Mitigation

- Strip EXIF metadata server-side
- Avoid security questions based on personal details
- Use email-based verification or MFA for password resets

Proof of Concept (PoC):

6.8.7

Low

Description:

Access a misplaced SIEM signature file.

Impact:

- Exposes signature files used for validating file integrity.
- Allows attackers to understand or bypass integrity verification mechanisms.
- Enables potential tampering or substitution of application files with malicious content.
- Reveals internal security mechanisms, reducing trust in the application.
- Weakens overall reliability of security controls.

Affected Endpoint/Parameter: /ftp

Mitigation:

- Store signature or hash files outside publicly accessible directories.
- Implement strict server-side access controls for verification resources.
- Perform integrity checks server-side using strong cryptographic hashing (e.g., SHA-256) with private keys.
- Conduct regular security scans to detect publicly exposed sensitive files.
- Apply least-privilege principles to restrict file access to only necessary backend processes.

Proof of Concept (PoC):

1. Go to https://juice-shop.herokuapp.com/ftp
2. You will notice some files
3. Go to https://juice-shop.herokuapp.com/ftp/suspicious_errors.yml
4. You will notice 403 forbidden saying Only .md and .pdf files are allowed!
5. Go to https://juice-shop.herokuapp.com/ftp/suspicious_errors.yml%00.md
6. You bypass 403 forbidden and it opened the suspicious_errors.yml file

---

6.8.8

Low

6.8.9

Low

Description:

A confidential file is accessible without proper authorization controls.

Impact:

- Exposure of internal information.
- Sensitive data leakage.
- Attackers may use exposed knowledge for further exploitation.

Mitigation:

- Enforce RBAC on all sensitive endpoints.
- Move internal documents outside public directories.
- Implement access tokens.

Reference: https://owasp.org/www-community/Broken_Access_Control

Affected Endpoint/Parameter: https://juice-shop.herokuapp.com/ftp

Proof of Concept (PoC):

First, the pentester check if there are a path to robots.txt file or not

Since we know that **robots.txt** file can hold important instructions about which parts of the site are allowed and or disallowed to access, we can try to see what's in there….

Here we can see there are many files which is an indicator for a potential retrieval or something important…

And we already have access to all those files..

But this file contained confidential data!!

6.8.10

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:
http://localhost:3000/metrics

Mitigation:

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

Proof of Concept (PoC):

- Open browser and navigate to /metrics.
- The server returns sensitive Prometheus-style metrics.

6.9 Security Misconfiguration

6.9.1

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: http://localhost:3000/#/complain

Mitigation:

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

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

- 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

6.9.2

6.9.3

Description:

Inform the shop about a typosquatting trick it has been a victim of at least in v6.2.0-SNAPSHOT. (Mention the exact name of the culprit)

Impact:

The Legacy Typosquatting vulnerability occurs when attackers register domains that closely resemble legitimate ones (e.g., with typos or slight variations). Users may inadvertently visit these malicious sites, which can lead to:

- Phishing attacks to steal credentials or personal information.
- Delivery of malware or malicious scripts.
- Exploitation of user trust in the legitimate application's branding.
- Potential account compromise or unauthorized access to sensitive data.

Affected Endpoint/Parameter:

Affected Endpoint: /#/contact

Mitigation:

- Regular Audits: Perform regular audits of all third-party packages used in projects to ensure they are from trusted sources.
- Use of Verified Sources: Only use packages from verified sources and maintain a list of approved packages for development use.

Proof of Concept (PoC):

1. Go to https://juice-shop.herokuapp.com/#/contact
2. See the screenshot and after filling all data required click submit

6.9.4

Description:

Behave like any “white-hat” should before getting into the action.

Impact:

A strong security policy reduces security risks, prevents unauthorized access, improves data protection, and ensures that systems and users follow safe and consistent practices. It strengthens the organization’s overall security posture and lowers the likelihood and impact of cyber incidents.

Affected Endpoint/Parameter:

/.well-known/security.txt or /security.txt

Mitigation:

By increasing user awareness, strengthening privacy controls, and promoting the use of secure and non-obvious security question answers, individuals can greatly reduce the risk of sensitive data exposure through publicly accessible information. Together, these practices create a stronger layer of protection against social engineering and unauthorized access.

Proof of Concept (PoC):

Navigate to
https://juice-shop.herokuapp.com/.well-known/security.txt
or
https://juice-shop.herokuapp.com/security.txt

6.9.5

Description

Redirection logic relies on a hard-coded allowlist inside main.js. Attackers can bypass these checks by manipulating the to parameter, allowing redirects to arbitrary blockchain URLs.

Impact

- Redirection to unintended or malicious destinations
- Potential spoofing of cryptocurrency payment pages
- Phishing-like attacks enabled

OWASP Reference

https://owasp.org/www-community/attacks/Unvalidated_Redirects_and_Forwards

Mitigation

- Implement strict URL validation
- Avoid outdated allowlists
- Allow redirects only to trusted internal URLs

6.9.6

Description:

Improper URL validation allows redirecting users to external websites.

Impact:

- Phishing attacks.
- User redirection to malicious domains.

Affected Endpoint/Parameter:

http://localhost:3000/redirect?to=http://google.com/?test=http://shop.spreadshirt.de/juiceshop%22

Mitigation:

Validate redirect targets against strict allowlist

Proof of Concept (PoC):

Inspected the main.js for the word redirect

then tried the url

Then tried to direct it to google using this url http://localhost:3000/redirect?to=http://google.com/?test=http://shop.spreadshirt.de/juiceshop%22

6.9.7

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

Impact:

Affected Endpoint/Parameter:
POST /api/Feedbacks/

Mitigation:

Validate CAPTCHA server-side.

Proof of Concept (PoC):

I sent a feedback with the correct

Then sent the request to the repeater and made group of the same request and sent them in parallel

6.9.8

Description:

The chatbot feature does not enforce proper input filtering or content restrictions, allowing an attacker to manipulate the bot into producing harmful, inappropriate, or abusive responses. This demonstrates a lack of Business Logic and insufficient content moderation.

Impact:

- The application may display offensive or harmful output to users.
- Reputation damage and legal issues if automated messages contain abusive content.
- Can be chained with social engineering attacks.

Mitigation:

- Implement strict input validation and content sanitization.
- Add server-side content filtering and toxicity detection.
- Restrict chatbot actions and outputs through allowlist-based logic.

Reference: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/11-Testing_for_Business_Logic

Affected Endpoint/Parameter:

Proof of Concept (PoC):

First, the pentester wasn't able to see the support chat

So, let's register first and see what changed…

After registration, the pentester can see the support chat now

After multiple tries the chatbot successfully sent the coupon: **"pes[Ch7ZKp"**

6.10 Error Handeling

6.10.1

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:

` http://127.0.0.1:3000/#/coupons_2013.md.bak `

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

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

2.Access the /ftp path: ` http://localhost:3000/ftp/ `

3.Attempt to open any file inside the directory.

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.

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.