



## **WEB APPLICATION SECURITY TESTING**

**NAME: SMRIDHI GERA**

**TASK 1: WEB APPLICATION SECURITY TESTING**

**PROGRAM: FUTURE INTERNS- CYBERSECURITY  
INTERNSHIP**

**DATE: OCTOBER 2025**

**TARGET:APPLICATION :OWASP JUICE SHOP**

## **Task Summary**

This assessment focused on identifying various loopholes and vulnerabilities in the **OWASP Juice Shop** application through **Burp Suite** and manual evaluation techniques. Several vulnerabilities were successfully detected, including **SQL Injection** and **User Information Leakage**, which could lead to potential attacks and account compromises. The findings were mapped to the **OWASP Top 10 (2021)** categories based on associated risk levels.

## **Tools Used**

- Docker
- OWASP Juice Shop
- Burp Suite

## **Steps & Procedure**

1. The **OWASP Juice Shop** application was deployed locally using **Docker**, ensuring a lightweight and isolated testing environment with all dependencies pre-configured.
2. **Burp Suite** was utilized for intercepting, scanning, and analyzing web requests and responses to identify vulnerabilities within the application.
3. All discovered issues were manually validated and mapped to the **OWASP Top 10 (2021)** categories based on their severity and potential impact.
4. Corresponding **mitigation strategies** were proposed for each identified vulnerability to enhance overall application security.

# Identified Vulnerabilities

## 1. SQL Injection

### Description:

The login form was vulnerable to SQL Injection, as it accepted unsanitized input that was directly concatenated into SQL queries. Attackers could inject payloads like ' OR '1'='1 to bypass authentication and access restricted areas.

### Security Impact:

- Complete bypass of authentication.
- Unauthorized access to database records.
- Potential full compromise of database contents.

**Risk Level:** High

### Evidence:

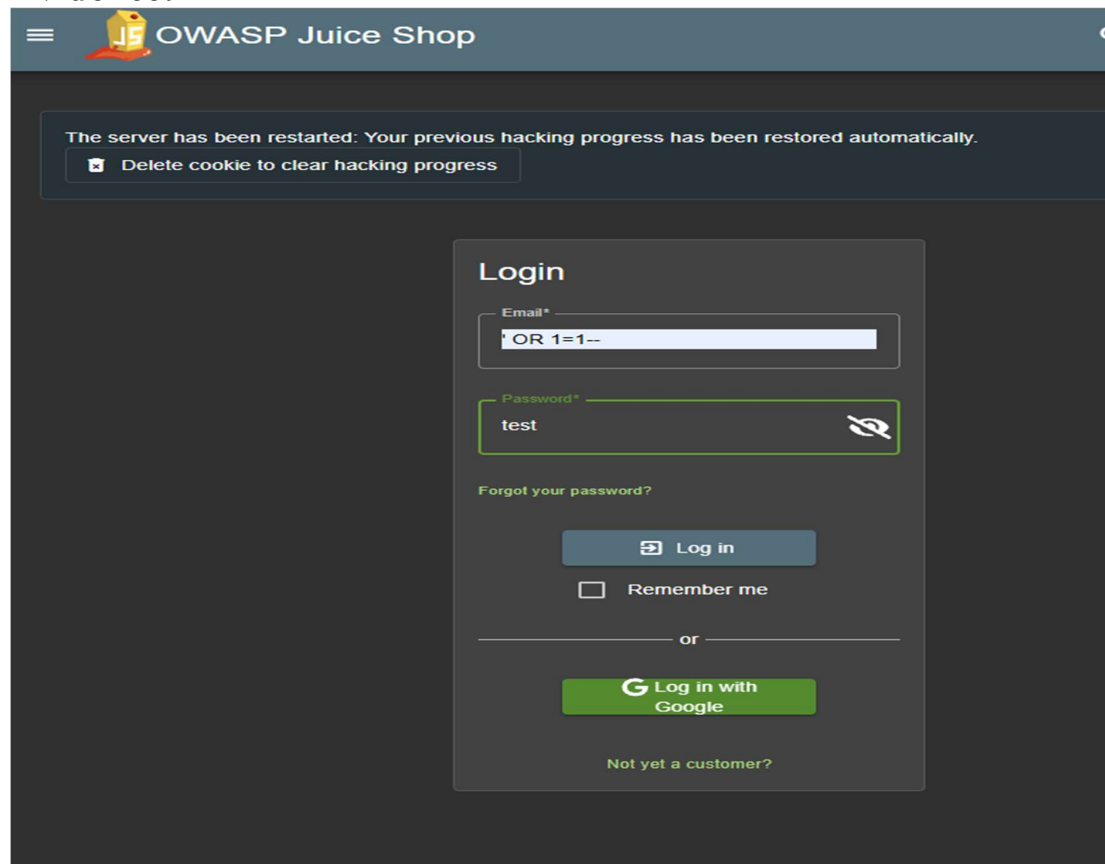


Figure 1: SQLi Payload in Login Form

89	http://localhost:3000	POST	/rest/user/login	✓			127.0.0.1
21	http://localhost:3000	PUT	/rest/continue-code/apply/kvvQWD...		434	200	JSON 127.0.0.1

**Request**

Pretty Raw Hex

1 POST /rest/user/login HTTP/1.1

2 Host: localhost:3000

3 Content-Length: 40

4 sec-ch-ua-platform: "Windows"

5 Accept-Language: en-US,en;q=0.9

6 Accept: application/json, text/plain, \*/\*

7 sec-ch-ua: "Chromium";v="141", "Not?A\_Brand";v="8"

8 Content-Type: application/json

9 sec-ch-ua-mobile: ?0

10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36

11 Origin: http://localhost:3000

12 Sec-Fetch-Site: same-origin

13 Sec-Fetch-Mode: cors

14 Sec-Fetch-Dest: empty

15 Referer: http://localhost:3000/

16 Accept-Encoding: gzip, deflate, br

17 Cookie: language=en; welcomebanner\_status=dismiss; cookieconsent\_status=dismiss; continueCode=kvvQWDpj5xnrN9V1bRgZeyZLOeetKfg6uXK060EkzMPYo8amJX8Kw4173q28

18 Connection: keep-alive

19

20 {

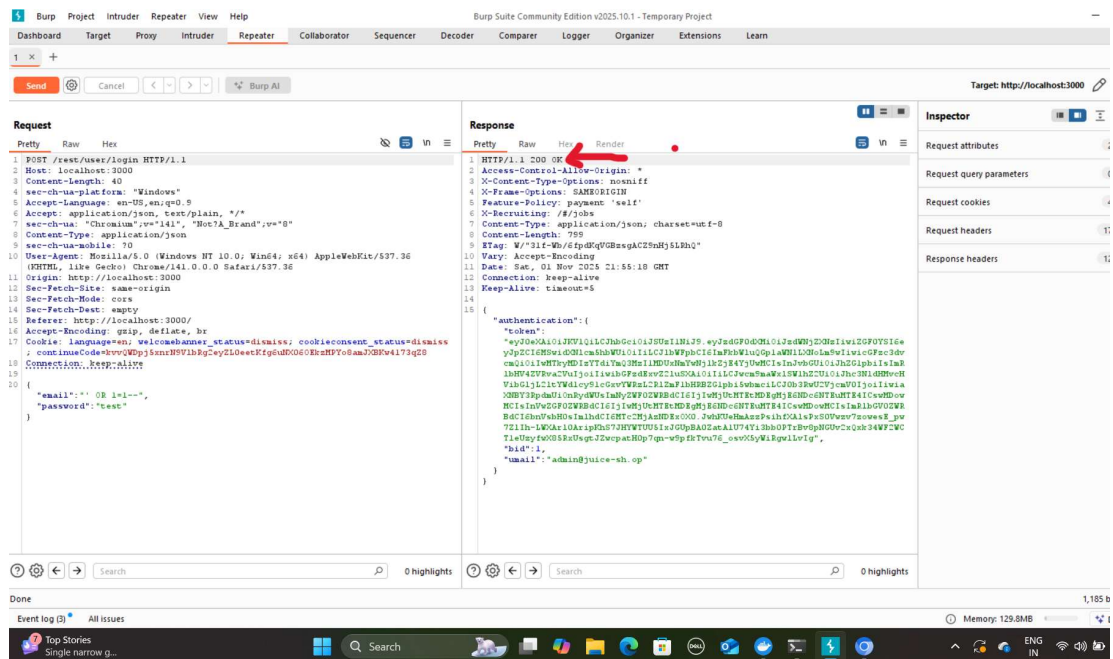
    "email":"' OR 1=1--",

    "password":"test"

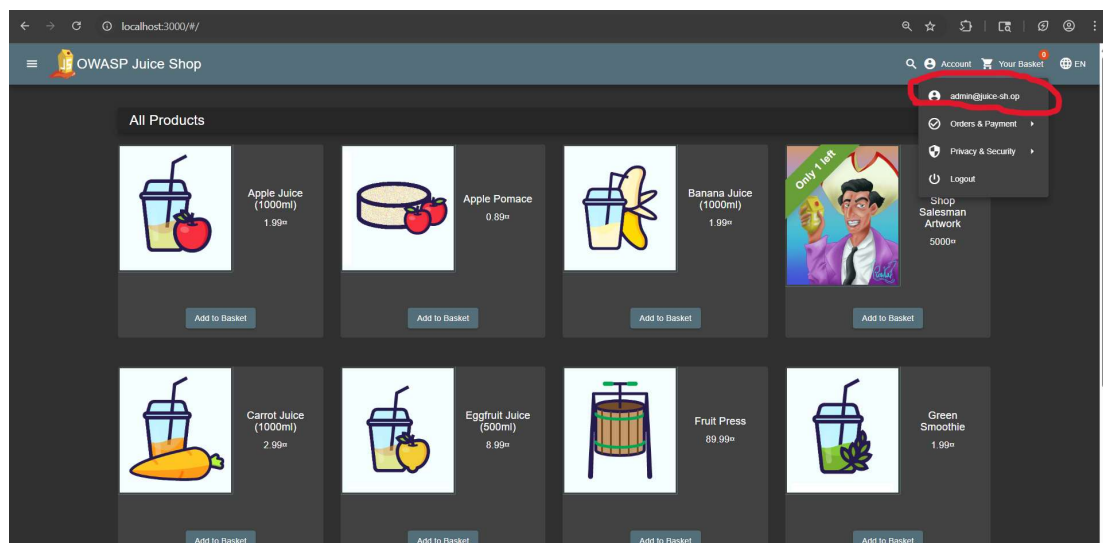
}

0 highlights

Figure 2: burp suite intercepted request containing malicious payload



**Figure 3: burp suite response showing http status 200 ok ,comforming successful login.**



**Figure 4: post login view showing access to admin account.**

## Mitigation Strategies:

- Use parameterized queries or prepared statements.
- Sanitize user inputs with strict whitelisting.
- Disable detailed SQL error messages in production.
- Implement least privilege access for the database user.

## 2. Cross-Site Scripting (XSS)

### Description:

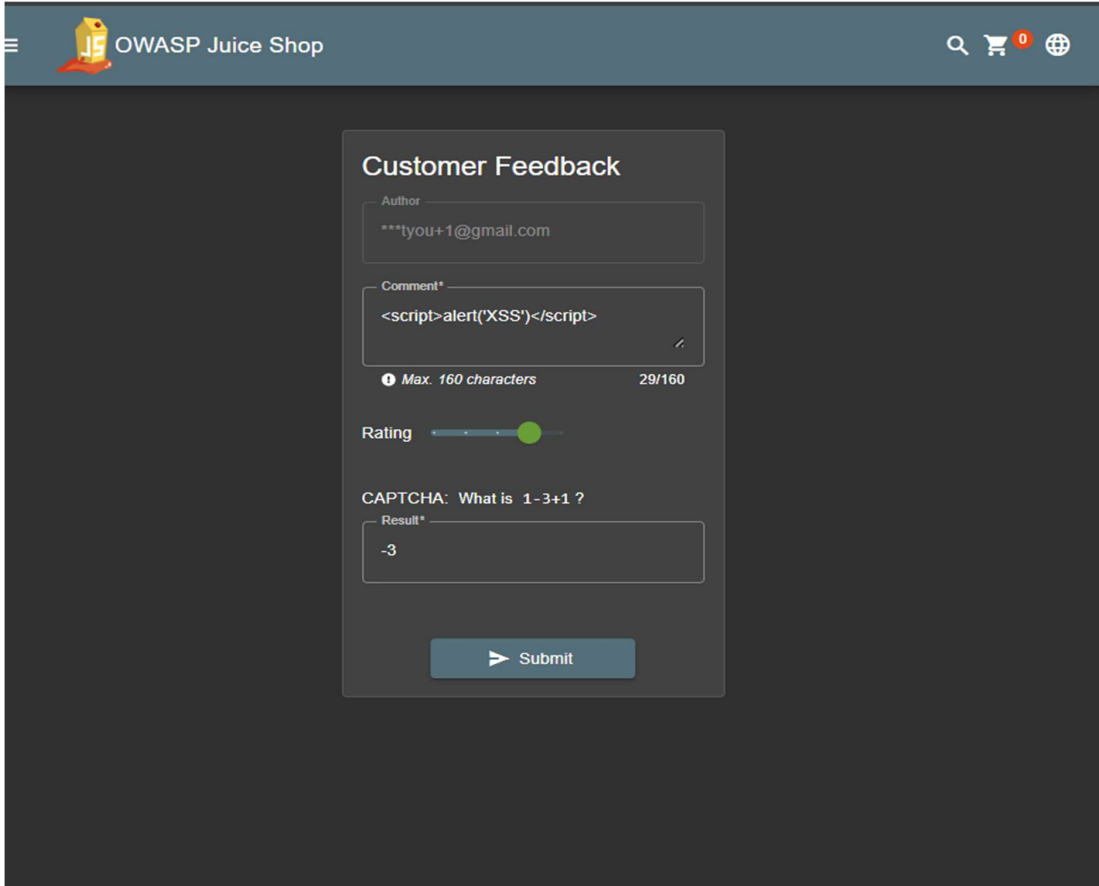
Certain input fields (such as feedback or product search) failed to properly sanitize user input, allowing JavaScript injection that executes in the victim's browser.

### Security Impact:

- Theft of session cookies or tokens.
- Execution of unauthorized actions in user accounts.
- Phishing or defacement attacks on legitimate users.

**Risk Level:** High

### EVIDENCE:

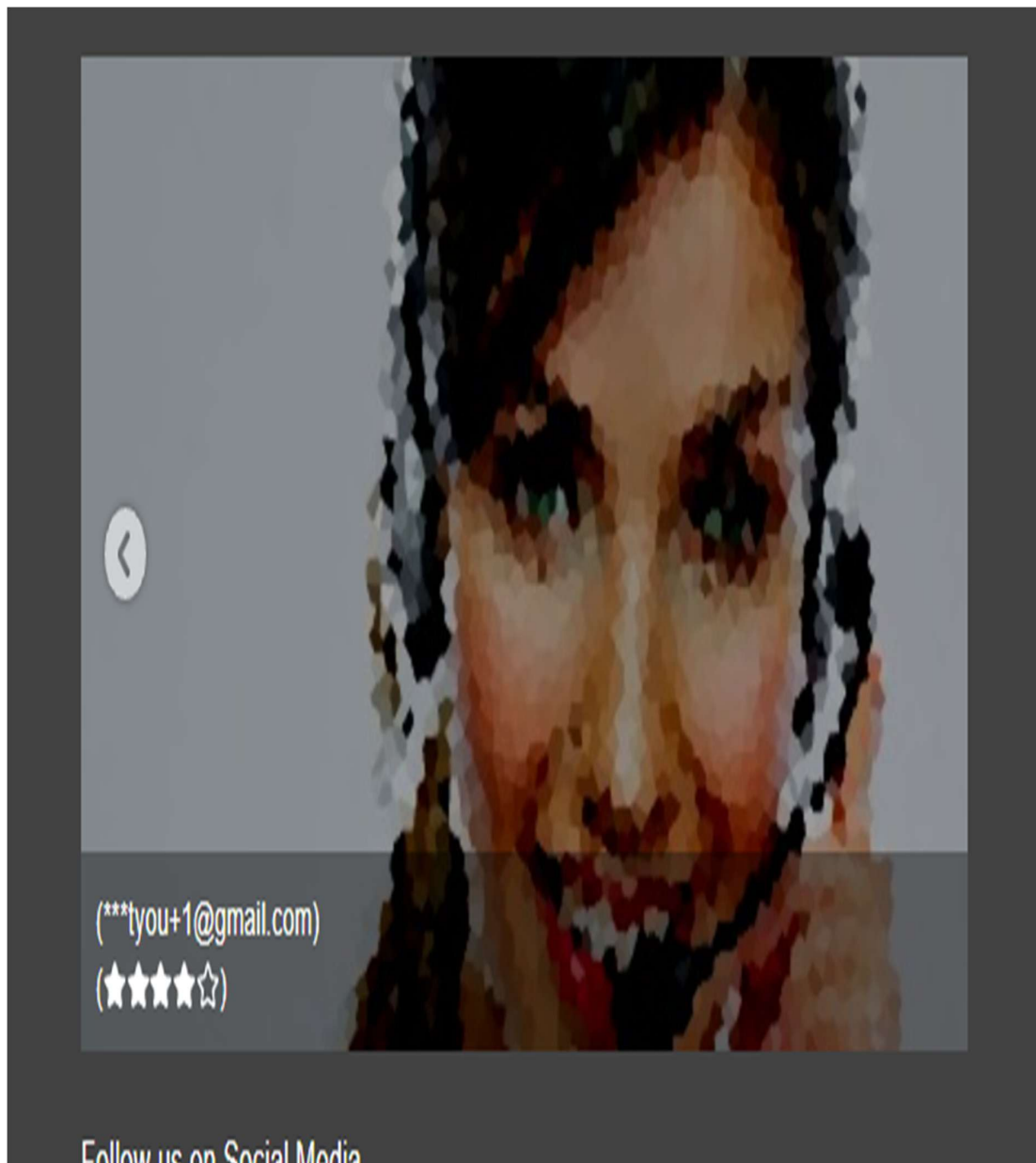


The screenshot shows the OWASP Juice Shop interface. At the top, the header includes the OWASP Juice Shop logo and navigation icons. The main content area displays a 'Customer Feedback' form. The form has the following fields and values:

- Author:** \*\*\*tyou+1@gmail.com
- Comment\*:** <script>alert('XSS')</script>
- Character Count:** Max. 160 characters, 29/160
- Rating:** A slider set to 3 stars.
- CAPTCHA:** What is 1-3+1 ?
- Result\*:** -3
- Submit Button:** A button with a right arrow and the text 'Submit'.

The form is set against a dark background, and the input fields are light gray with rounded corners.

**Figure1: feedback form with payload**



**Figure2: alert box triggered after visiting abou**

### **Mitigation Strategies:**

- Implement input validation and output encoding.
- Use frameworks or libraries with built-in XSS protection.
- Apply a strong Content Security Policy (CSP).

### 3. Replay and Impersonation

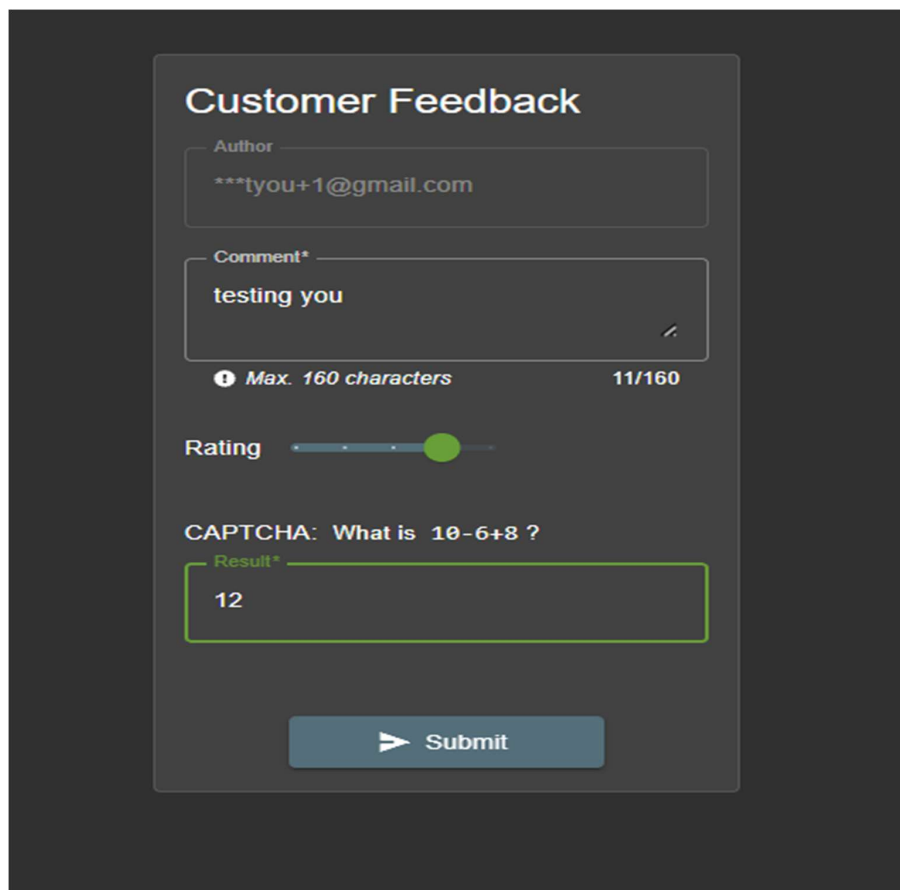
#### Description:

Session tokens and authenticated requests could be replayed without verification, meaning previously captured requests (like feedback submission) could be resent to perform repeated actions.

#### Security Impact:

- Impersonation of legitimate users.
- Replay of actions such as duplicate submissions or transactions.
- Potential exploitation of long-lived tokens.

**Risk Level:** Medium



The image shows a 'Customer Feedback' form on a dark background. The form contains the following elements:

- Author:** A text input field containing the email address '\*\*\*tyou+1@gmail.com'.
- Comment\*:** A text input field containing the text 'testing you'. Below the field is a character count: 'Max. 160 characters' and '11/160'.
- Rating:** A horizontal slider control with a green circle indicating a rating level.
- CAPTCHA:** A section titled 'CAPTCHA: What is 10-6+8 ?' with a 'Result\*' input field containing the number '12'.
- Submit:** A button with a right-pointing arrow and the text 'Submit'.

Figure 1: original feedback request submitted through the customer feedback form.





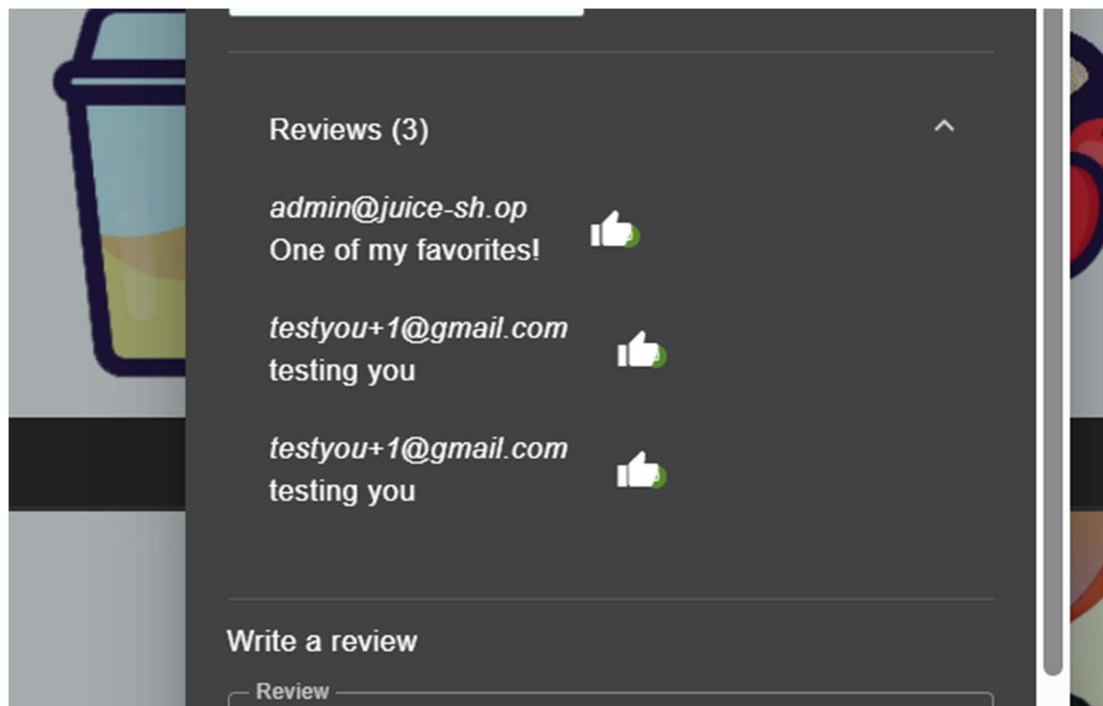


Figure 3: execution of replayed request multiple times

### Mitigation Strategies:

- Implement nonce or timestamp validation to prevent replay.
- Use short-lived session tokens and refresh them periodically.
- Invalidate tokens immediately after logout or session timeout.

## 4. Insecure Token Design

**Description:**

The application's JSON Web Tokens (JWTs) used weak algorithms such as MD5 for signing, which are susceptible to collision attacks. Tokens also contained hashed user credentials that can be cracked.

### Security Impact:

- Forged or modified tokens may be accepted by the server.
- Attackers can gain unauthorized access or persist sessions indefinitely.
- High risk of account takeover and privilege escalation.

**Risk Level:** High

**Evidence:**

[illegible]

### Mitigation Strategies:

- Use strong signing algorithms (e.g., HS256 or RS256).
- Include token expiration and implement key rotation.
- Avoid storing sensitive information inside JWTs.

## 5. Insecure Direct Object Reference (IDOR)

**Description:**

Internal references like user IDs were exposed in URLs and API endpoints without verifying ownership. Modifying these IDs (e.g., `userId=101`  $\rightarrow$  `userId=102`) allowed access to other users’ data.

### Security Impact:

- Unauthorized access to other users' personal data.
- Data manipulation and privacy violations.
- Breach of confidentiality and data integrity.

**Risk Level:** High

[illegible]

### Mitigation Strategies:

- Validate resource ownership server-side before processing requests.
- Apply “deny by default” access policies.
- Enforce authorization checks on every endpoint.

## OWASP Top 10 (2021) Mapping

Detected Vulnerability	OWASP Category	Risk Rating
SQL Injection	A03 - Injection	High
Cross-Site Scripting (XSS)	A03 - Injection	High
Replay & Impersonation	A07 - Identification & Authentication Failures	Medium
Insecure Token Design	A02 - Cryptographic Failures	High
Insecure Direct Object Reference (IDOR)	A01 - Broken Access Control	High

---

### Conclusion

The vulnerability assessment of OWASP Juice Shop using Docker and Burp Suite revealed multiple security weaknesses such as **SQL Injection, Cross-Site Scripting, Insecure Token Design, Replay & Impersonation, and Broken Access Control (IDOR)**.

These vulnerabilities can lead to **data breaches, account hijacking, and unauthorized access** if left unresolved.