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

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.

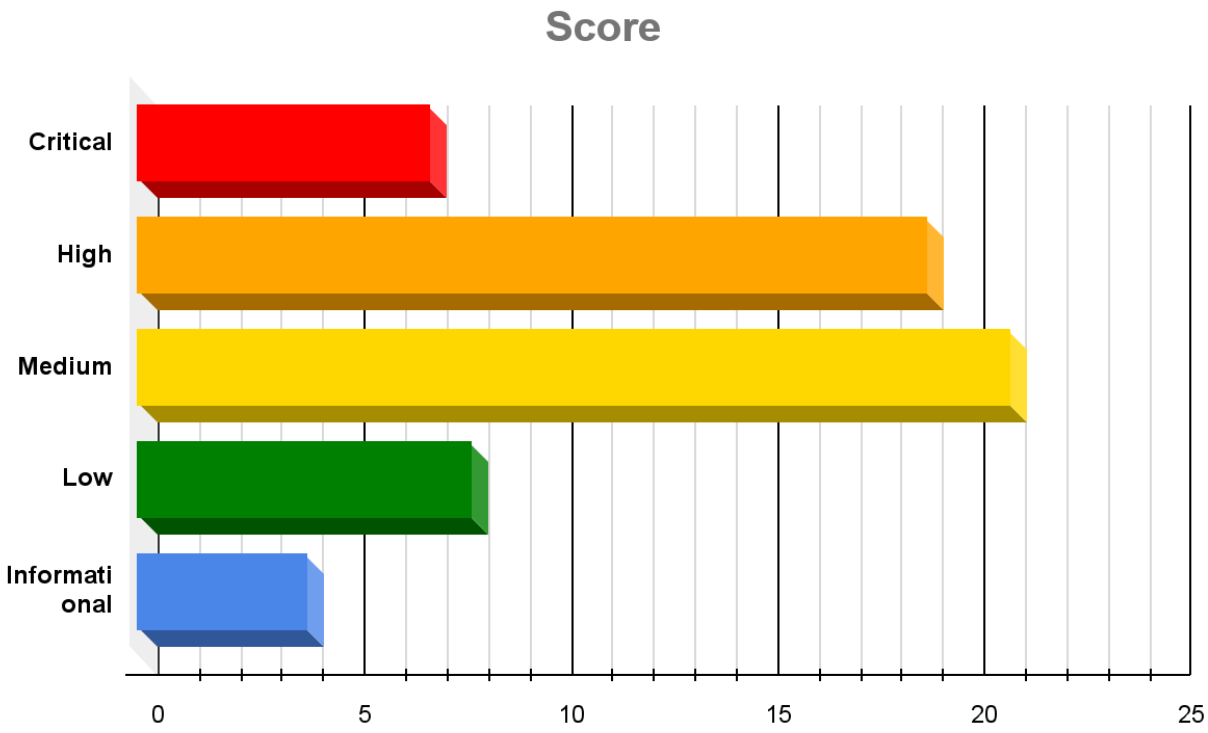
1.5 Testing Summary

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

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

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

8.9	Register with empty mail and password	**INFO**	passed
15.2	Unvalidated Redirect via Exposed Legacy URLs	**INFO**	passed

3.Technical 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

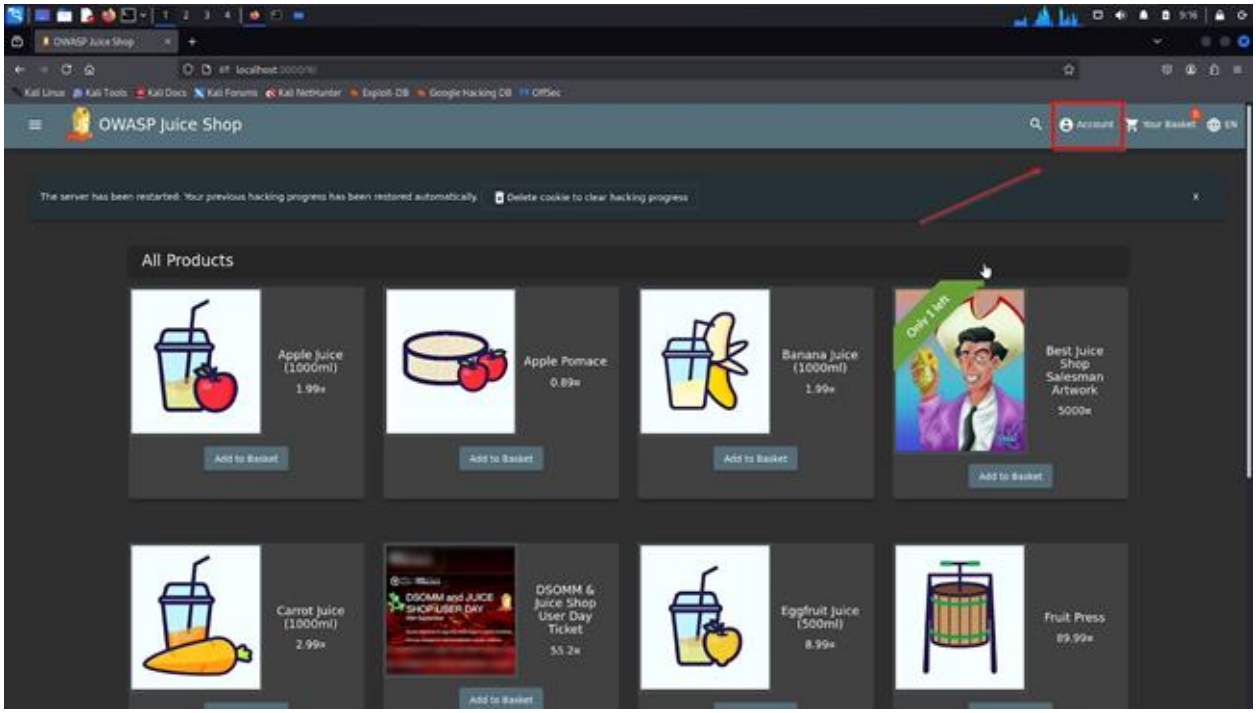
Resources:https://owasp.org/www-community/attacks/SQL_Injection

Recommendations:

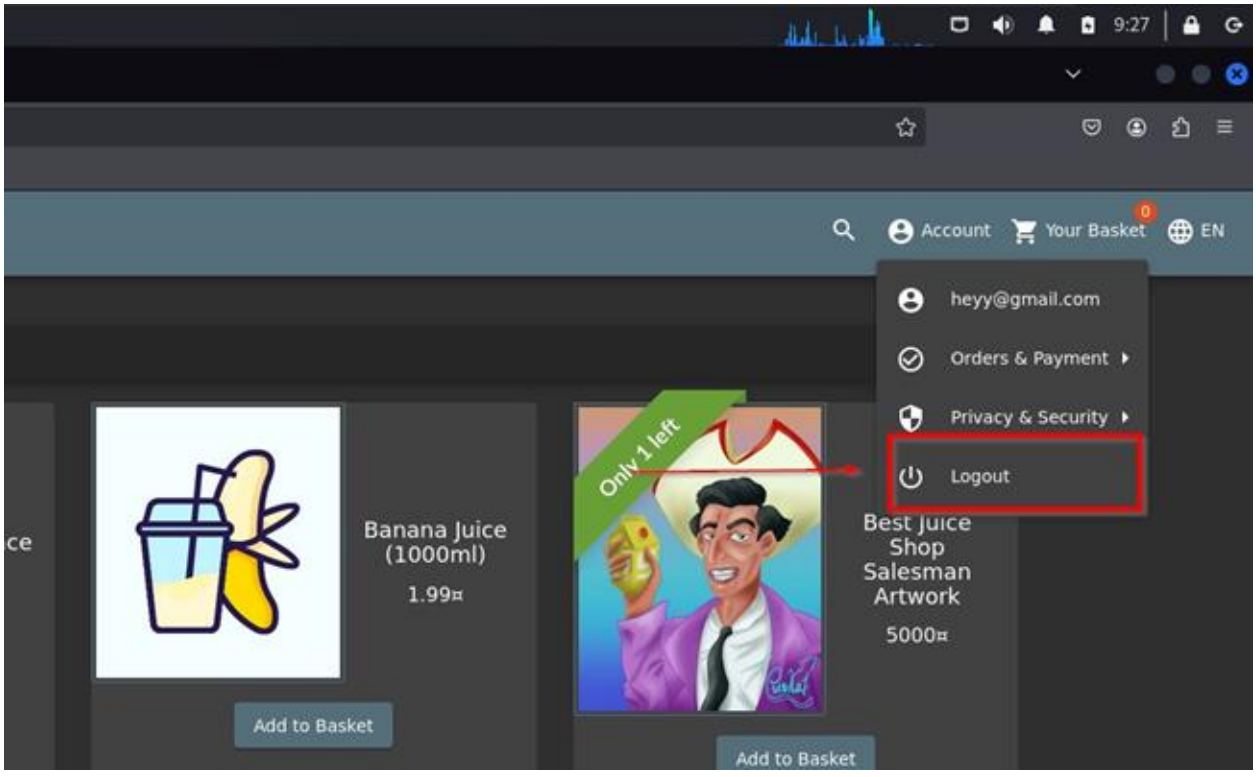
1. Use **prepared statements (parameterized queries)** to prevent injection.
2. Sanitize and validate all user inputs.
3. Implement proper **error handling** to avoid exposing SQL errors.
4. Apply **least privilege** principles on database accounts.

Proof Of Concept (P.O.C)

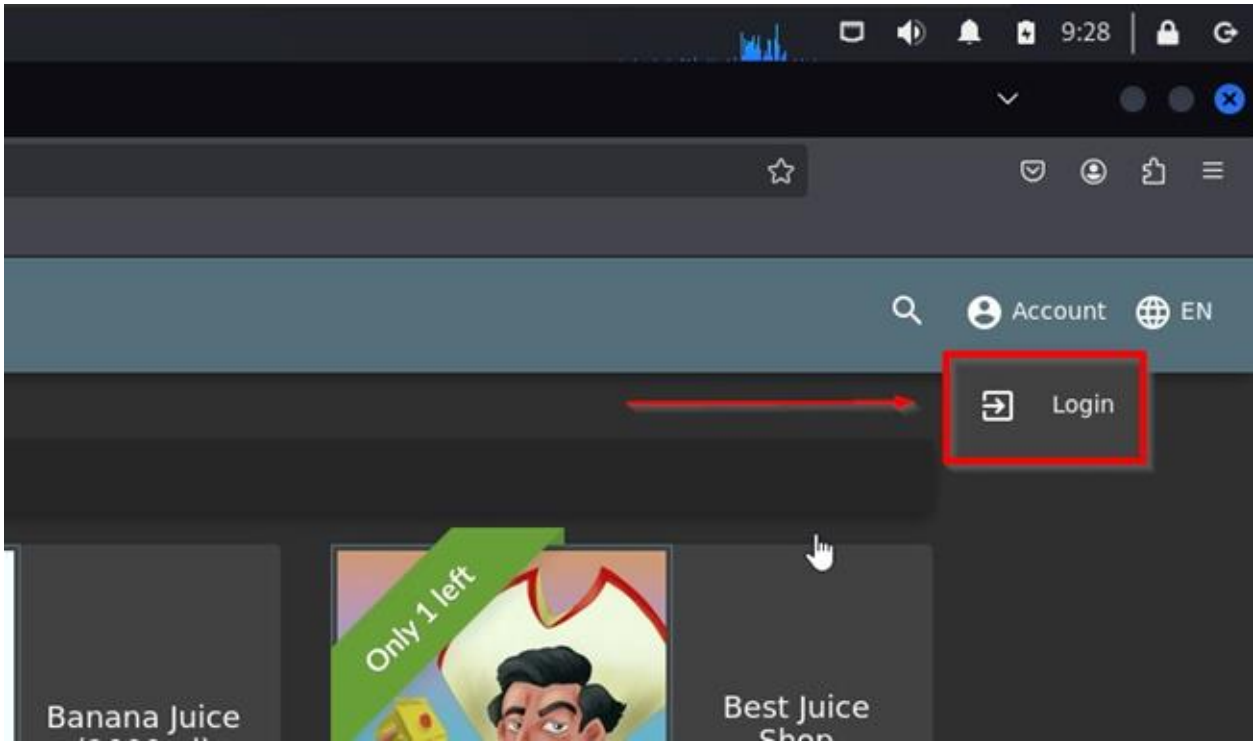
1. **Go to the account as in picture**



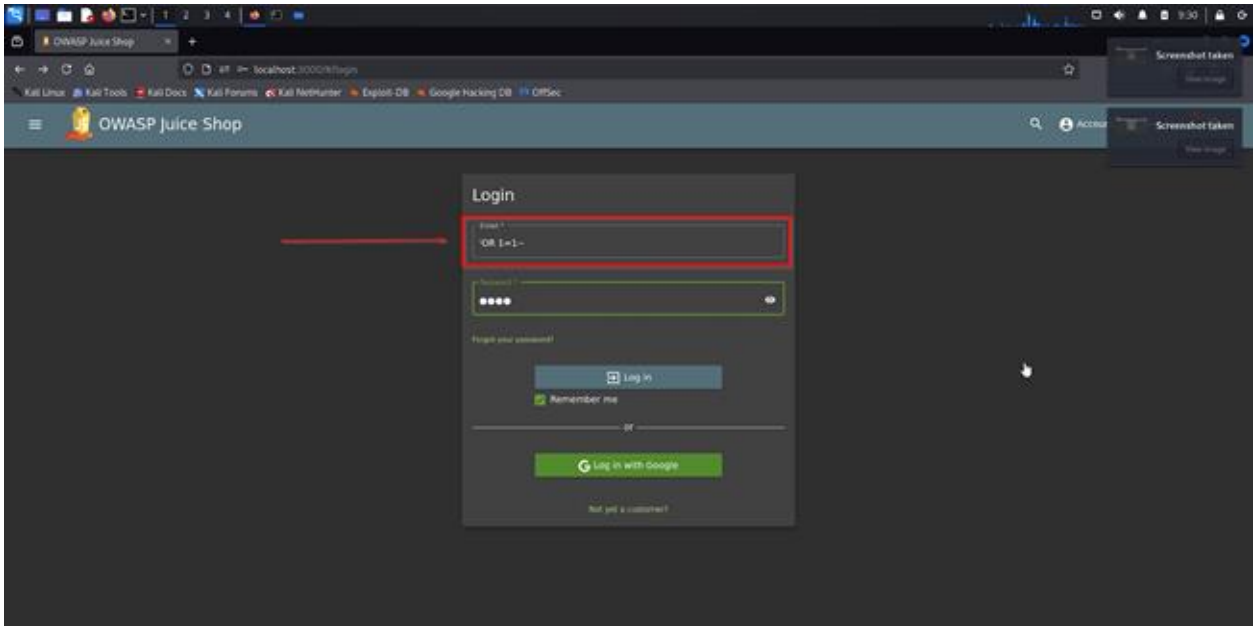
2. **If you are logged in then log out**



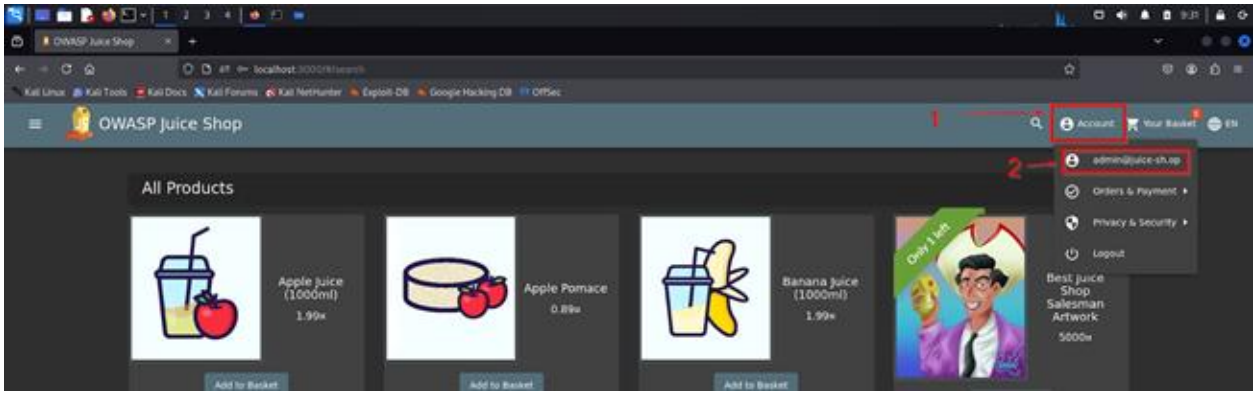
3. **Click on login**



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:

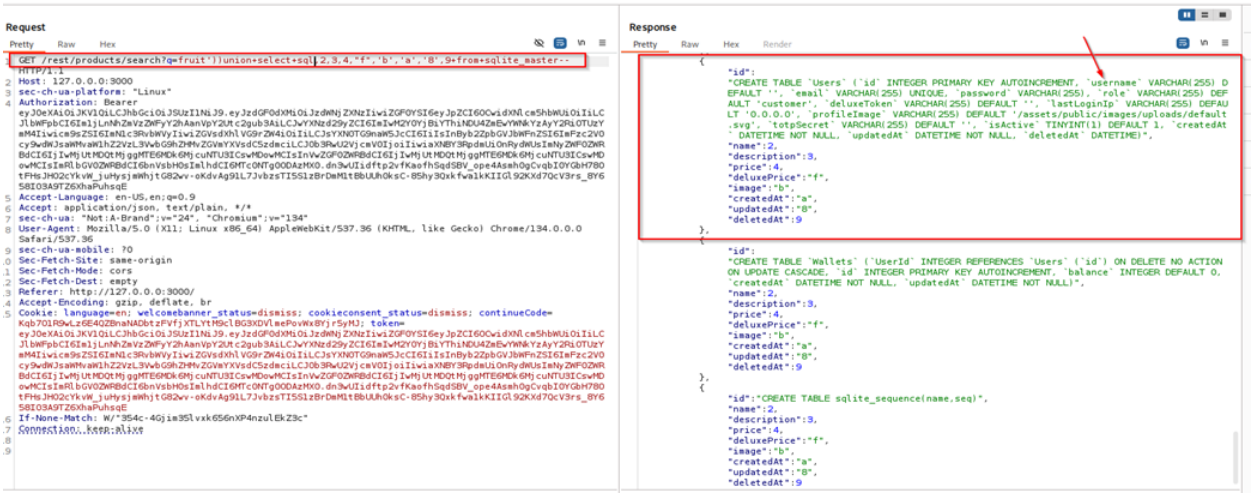
1. Use Prepared Statements

2. Strict Input Validation

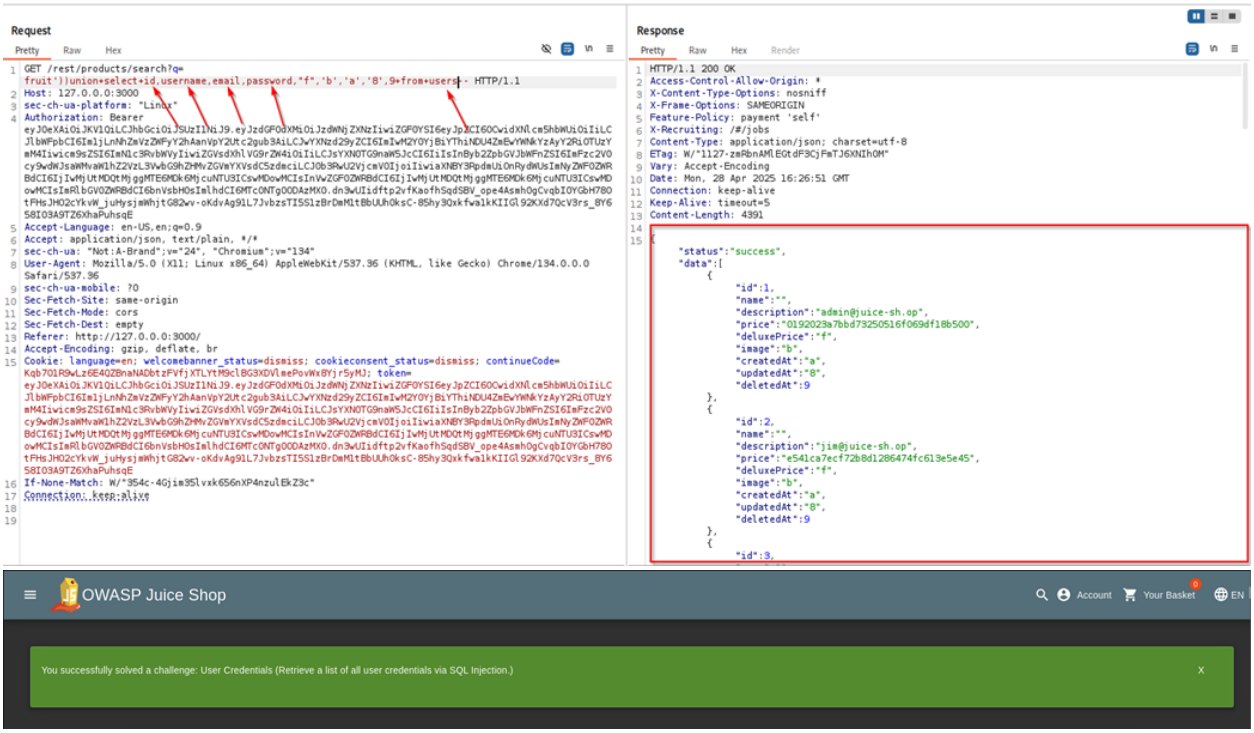
3. Error Handling**:** Ensure no detailed database errors are leaked to the client during failed queries.

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

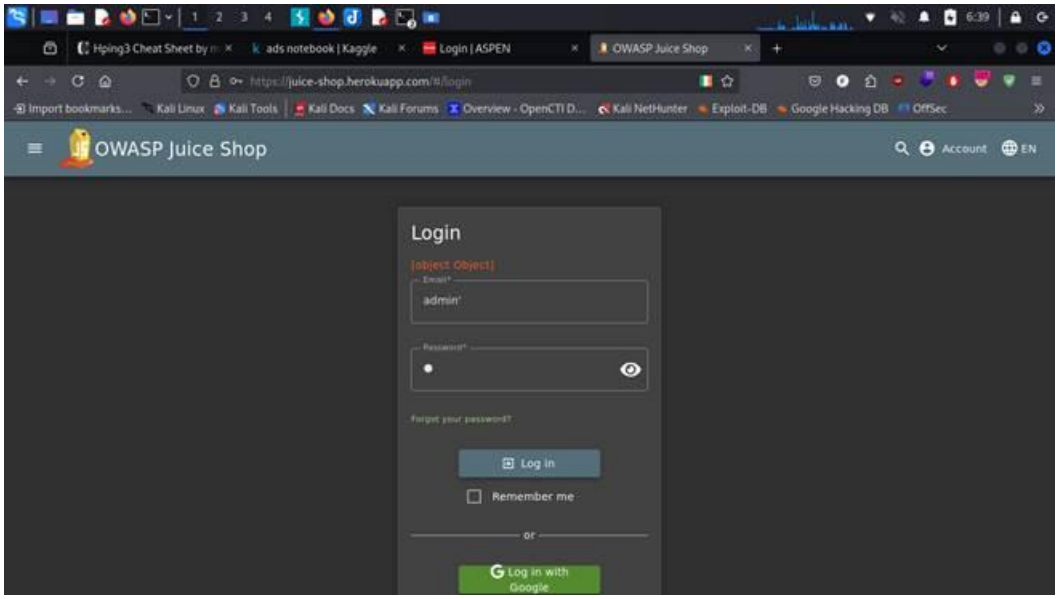
- Validate input

- Proper error handling

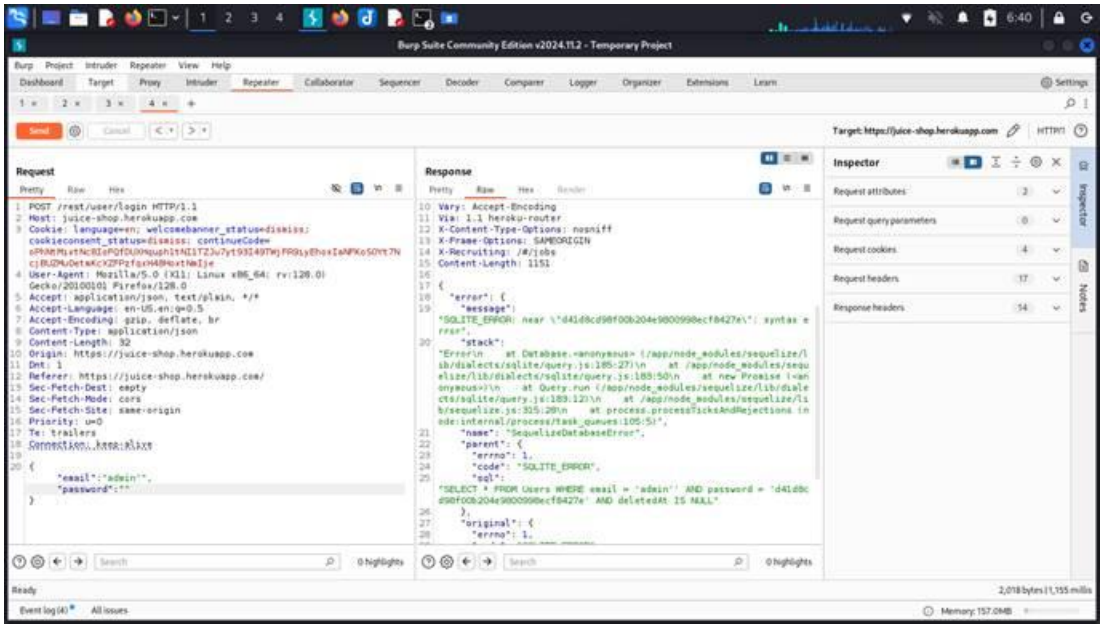
Proof of Concept (P.O.C)

(1) Move to login page

(2) Send ‘ to the system

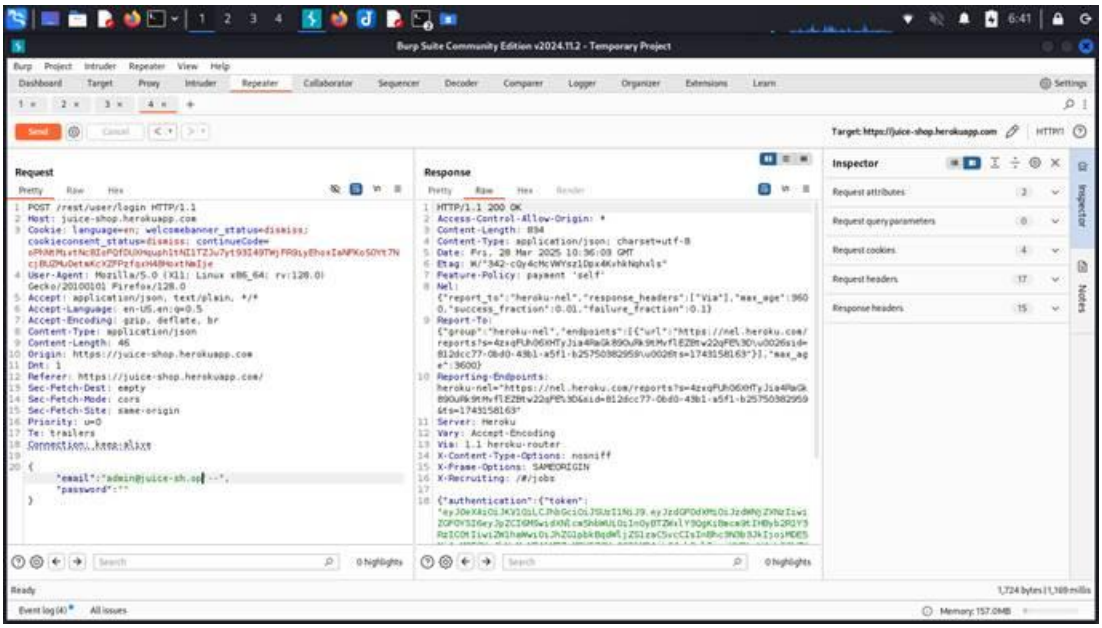


(3)now you see that there is an error happen then open burp and repeat the request

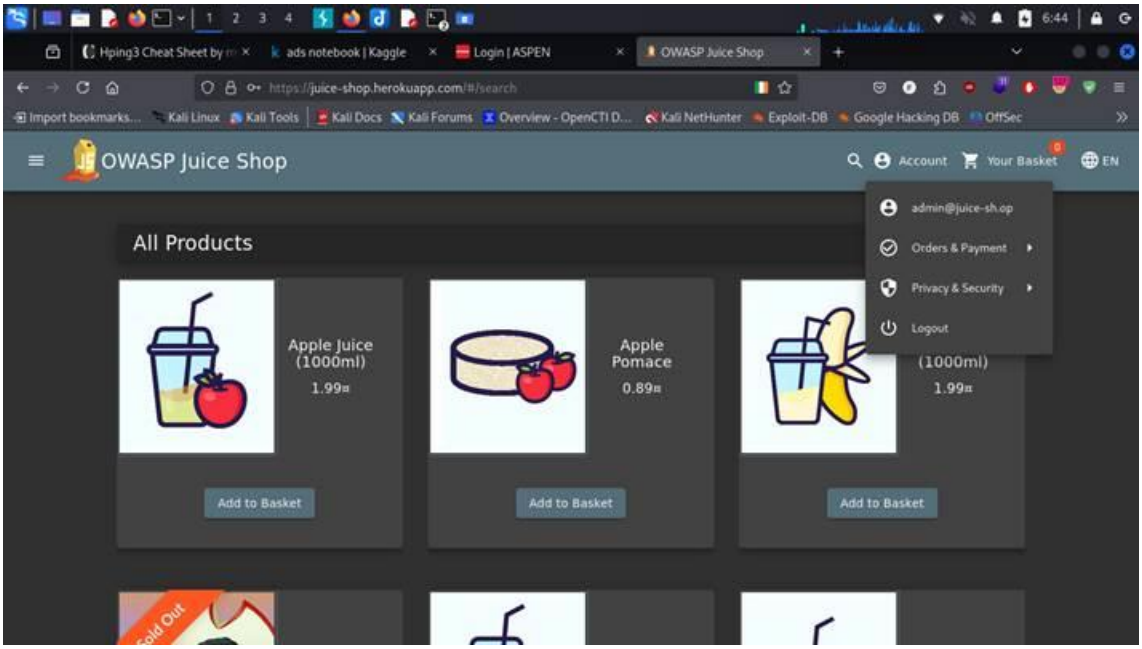


Then you see the details of the error and now you know that the system use sqlite

(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.



3.4 SQL Injection (Authentication bypass)

High

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

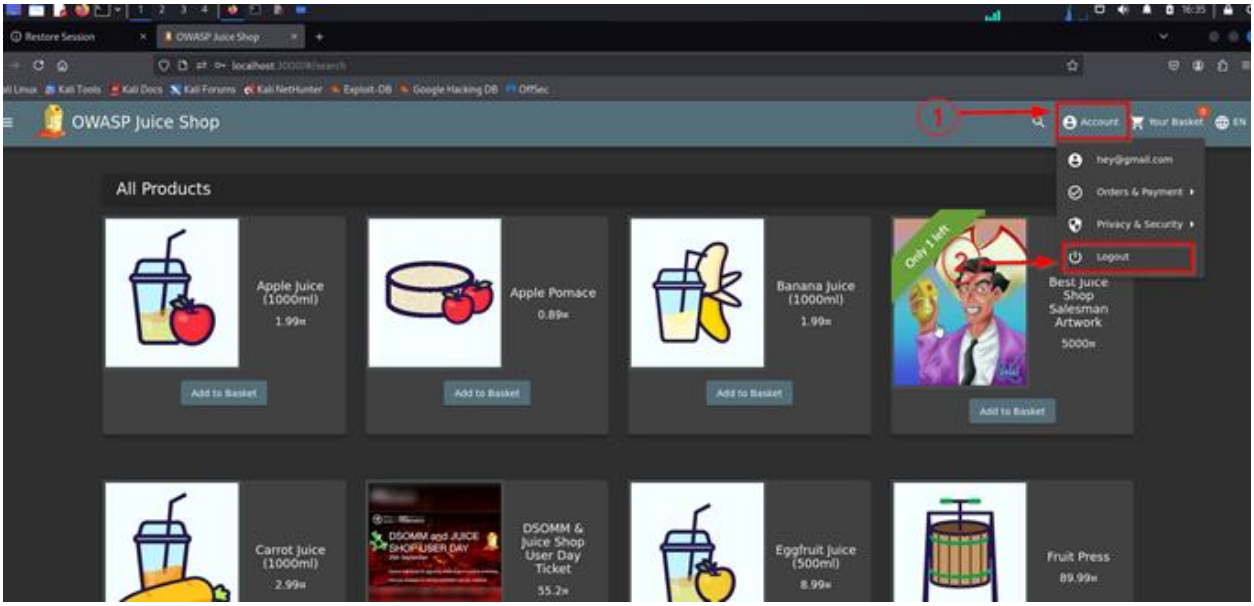
resources: https://portswigger.net/web-security/sql-injection

Recommendation

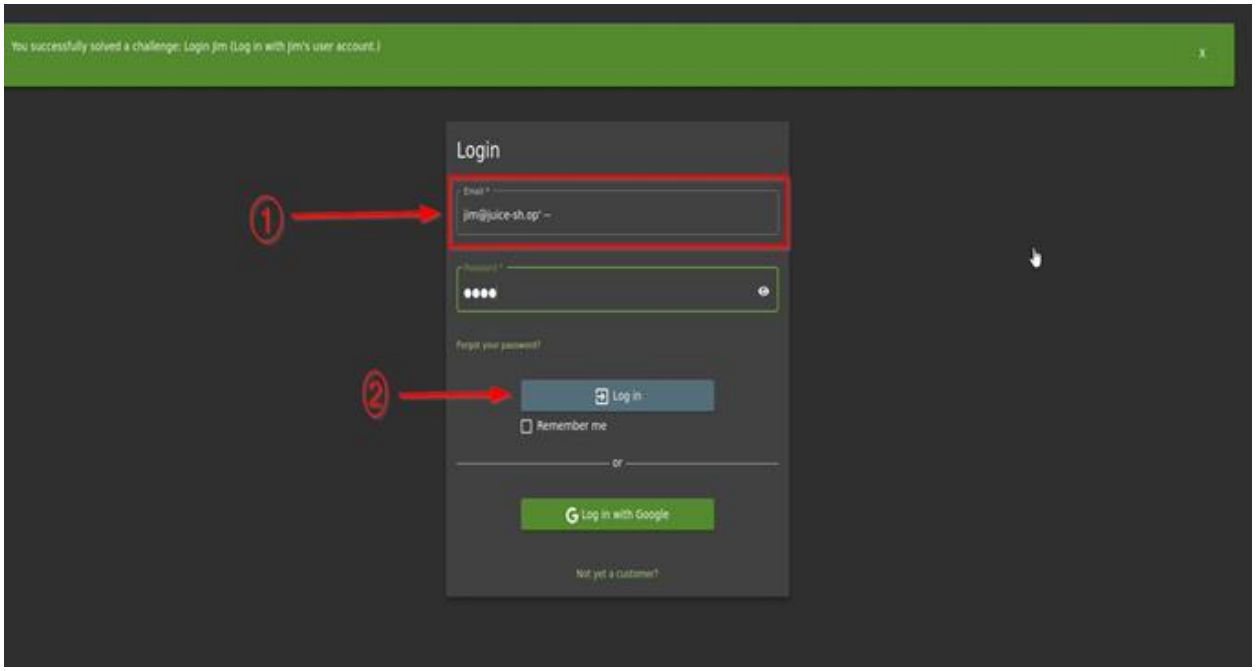
1. Input Validation
2. Input sanitizing
3. use least-privilege DB account

proof of concept

1- go to the Login page



2- Add the following credentials as shown below



3.5 SQL Injection (Authentication bypass)

High

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

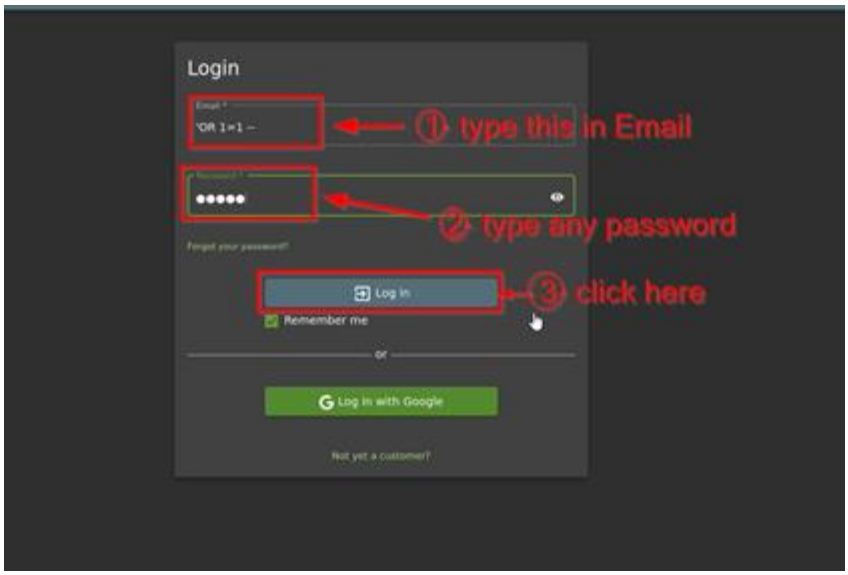
resources: https://portswigger.net/web-security/sql-injection

Recommendation

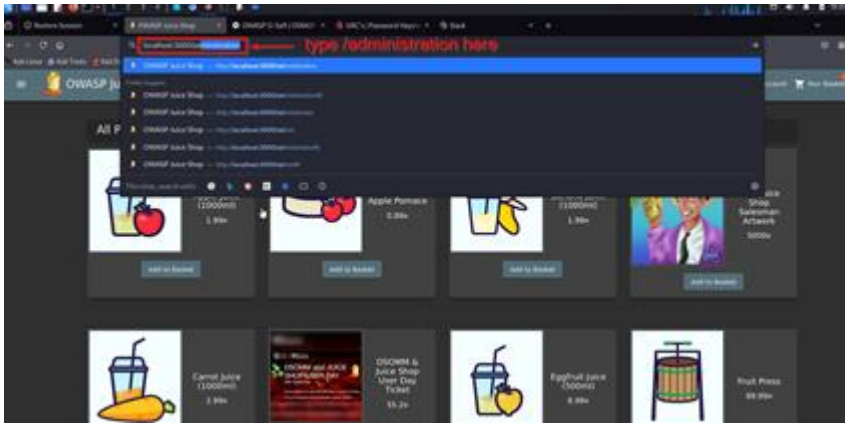
1. Input Validation
2. Input sanitizing
3. use least-privilege DB account

proof of concept

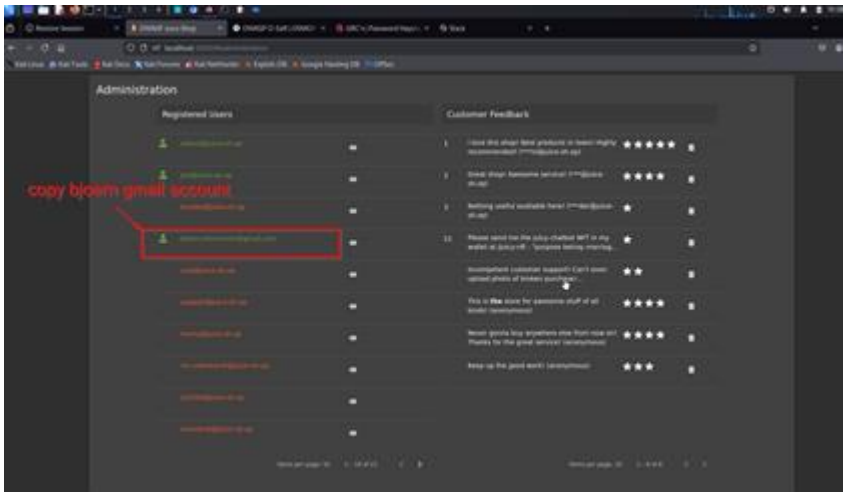
1. log out of your account and login with admin entering the following credentials 'OR 1=1--



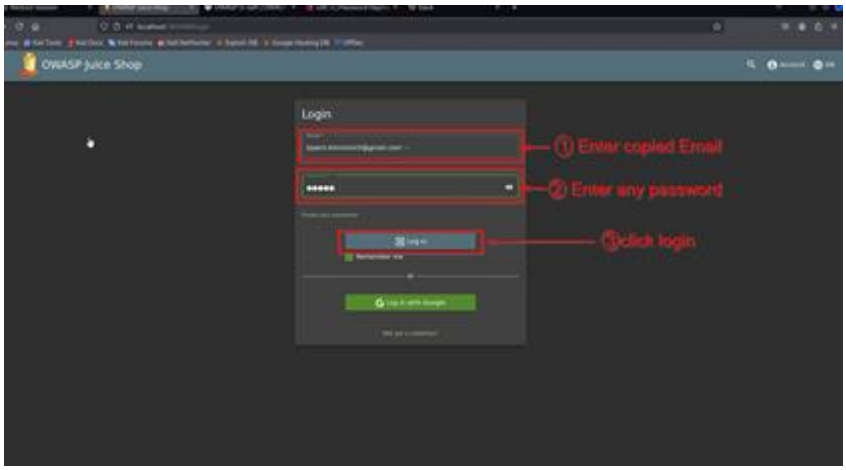
2.Go to http://localhost:3000/administration as shown below



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

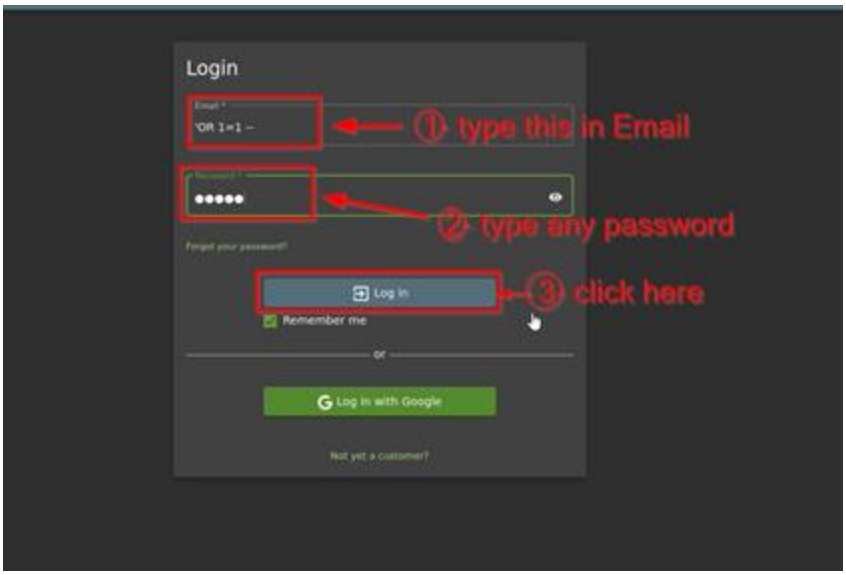
resources: https://owasp.org/www-community/attacks/SQL_Injection

Recommendation

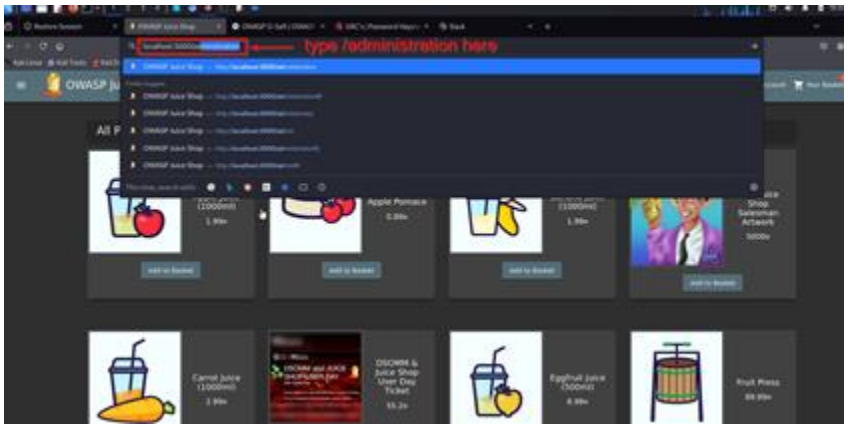
1. Input Validation
2. Input sanitizing
3. use least-privilege DB account
4. Log and alert on unexpected changes to key account fields.
5. server side validation

proof of concept:

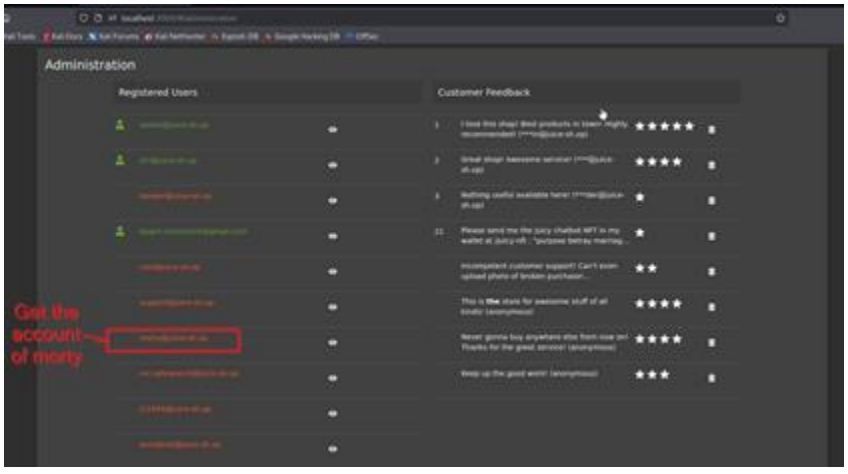
1. log out of your account and login with admin entering the following credentials 'OR 1=1 -- and add any password.



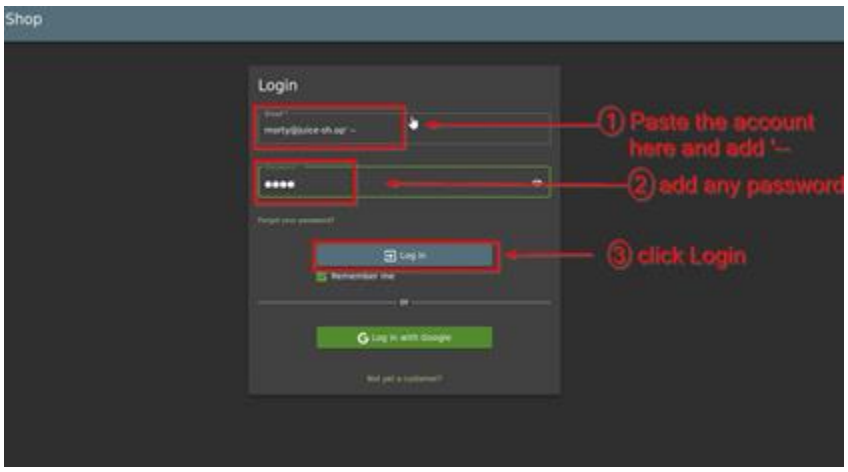
2.Visit http://localhost:3000/administration as shown below



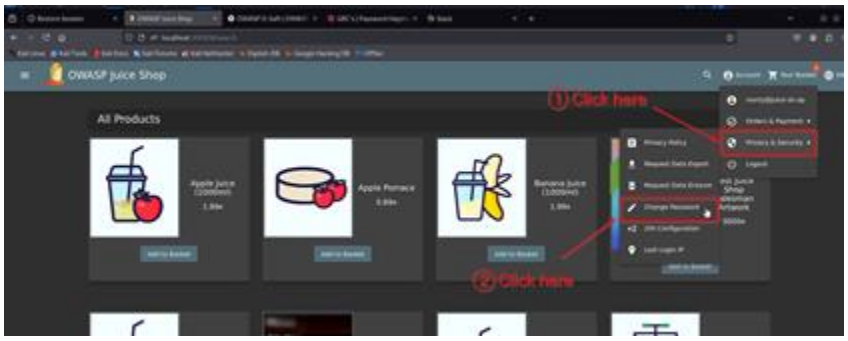
3. Go copy the Account of morty from the registered users as shown below



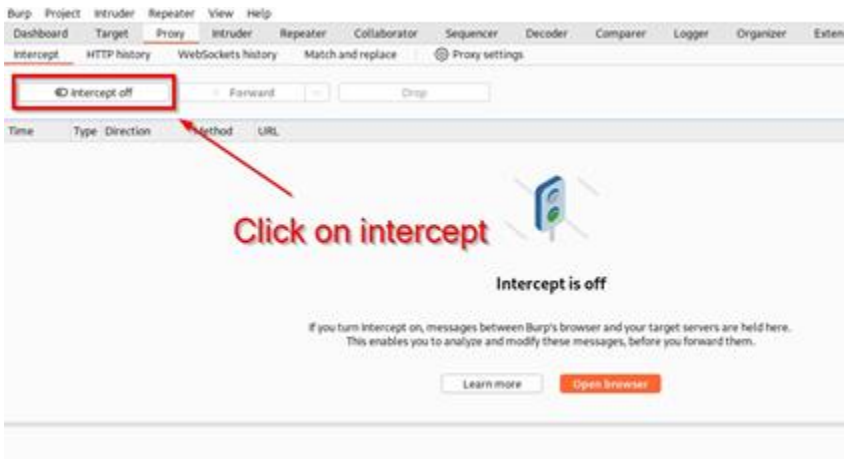
4. Enter the credentials as shown below and add ‘-- after mort account



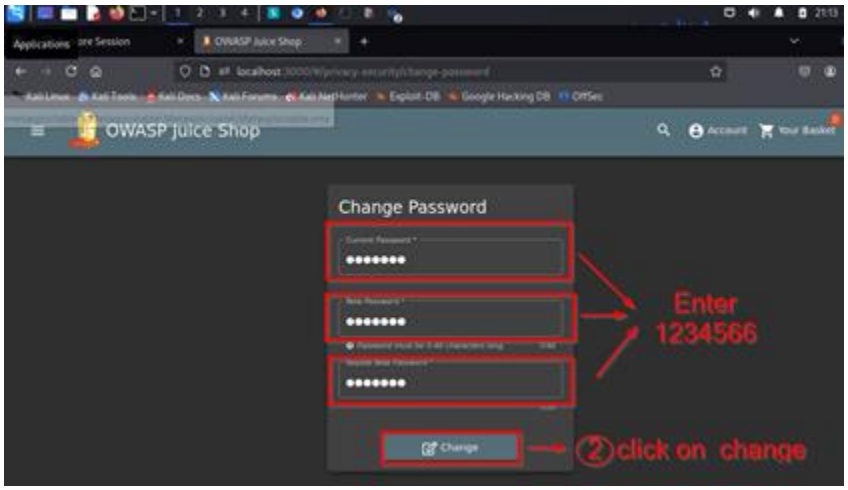
5. Now do as shown Below to go to change password



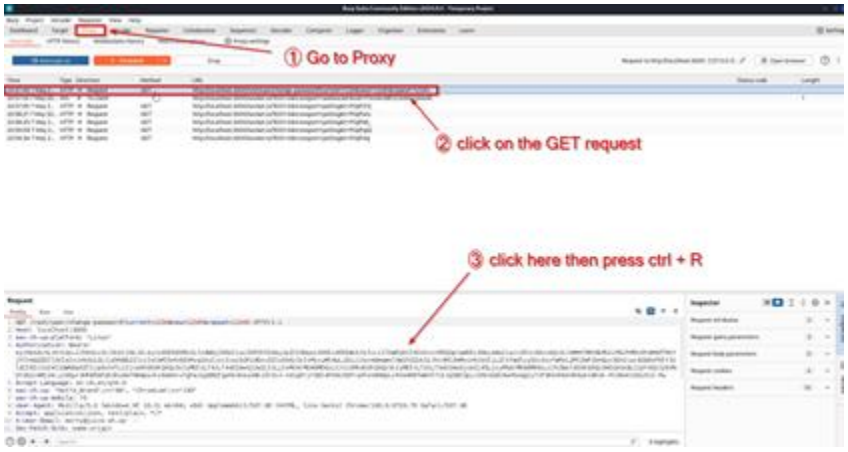
6. Open burb suite browser and copy this link http://localhost:3000/#/privacy-security/change-password to chromium after signing in with the same credentials in the last picture then apply following steps shown below



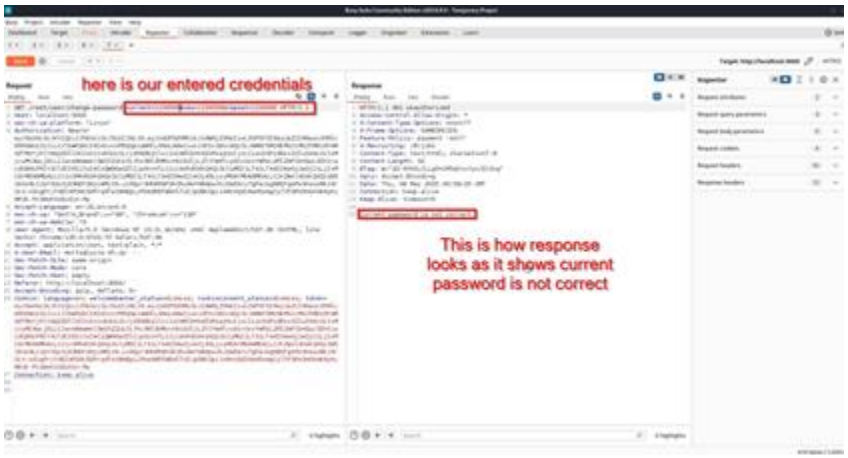
.7. Go to chromium and Enter 1234566 in every place as shown below and click change



8. Then go to burb and do as shown below

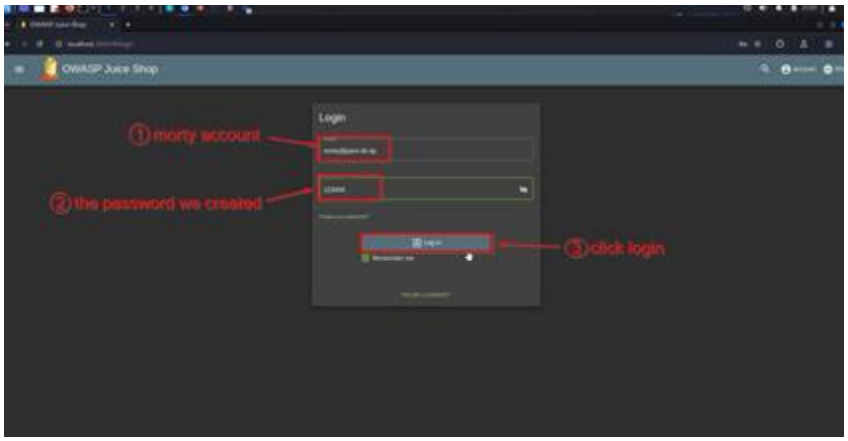


9. A preview of request and response before changing credentials

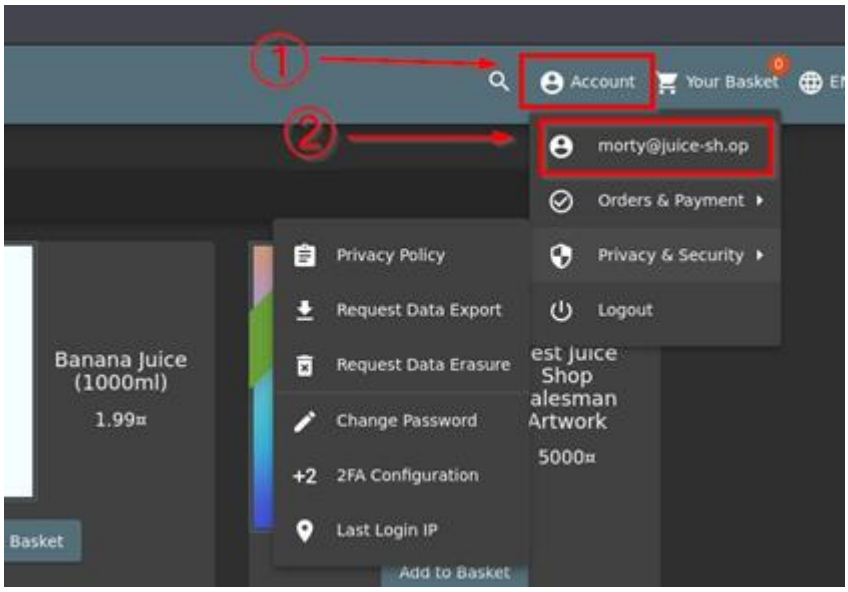


10. Now delete the current=1234556& from last picture and make new and repeat equal and then click on send

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



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

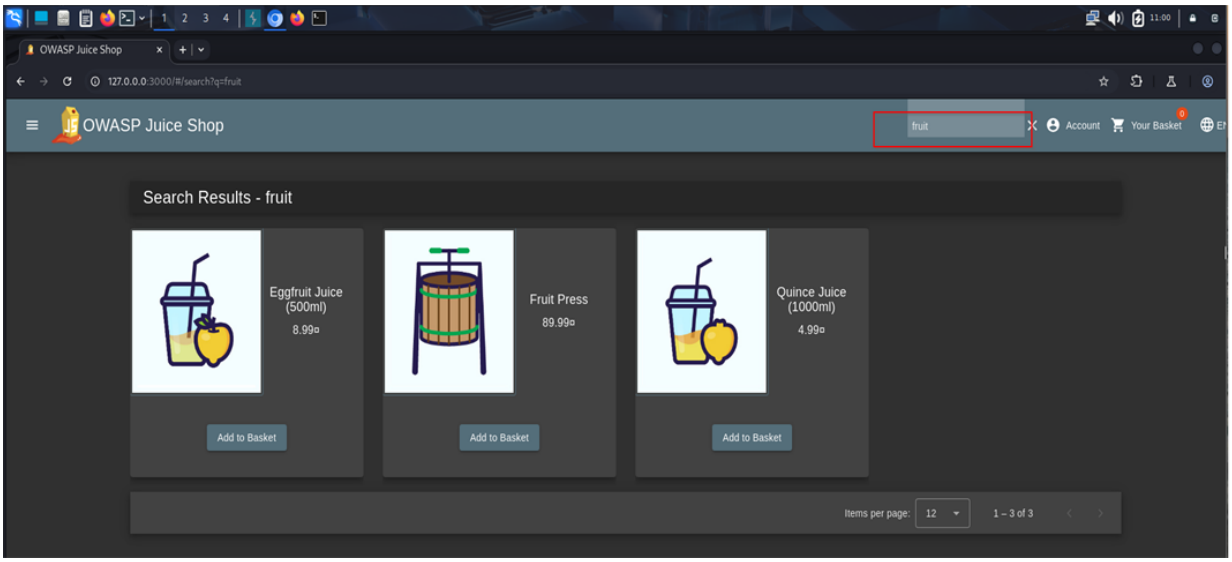
Recommendations:

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

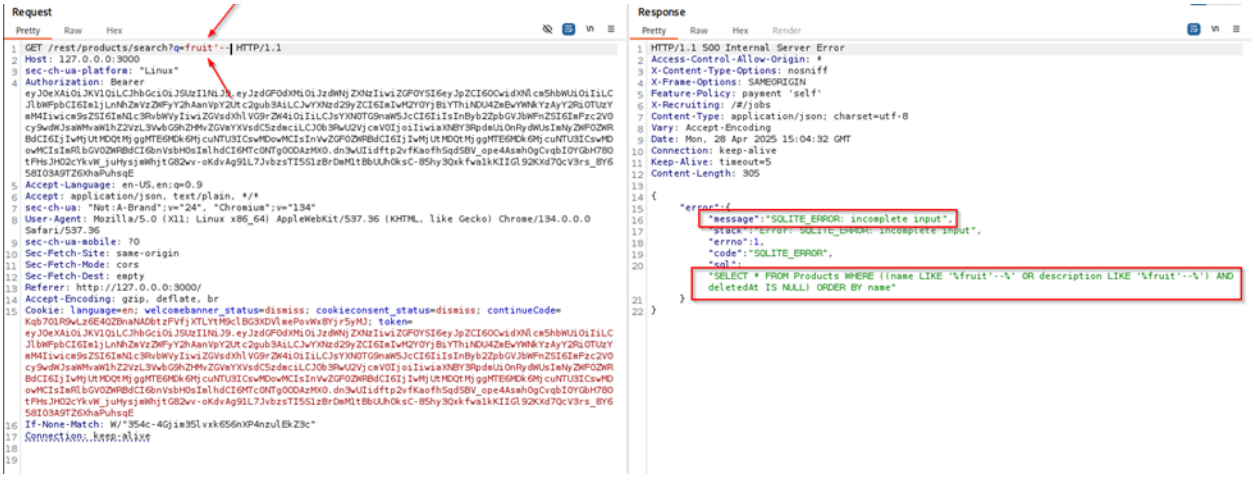
3. Hide Database Errors

Proof Of Concept(POC)

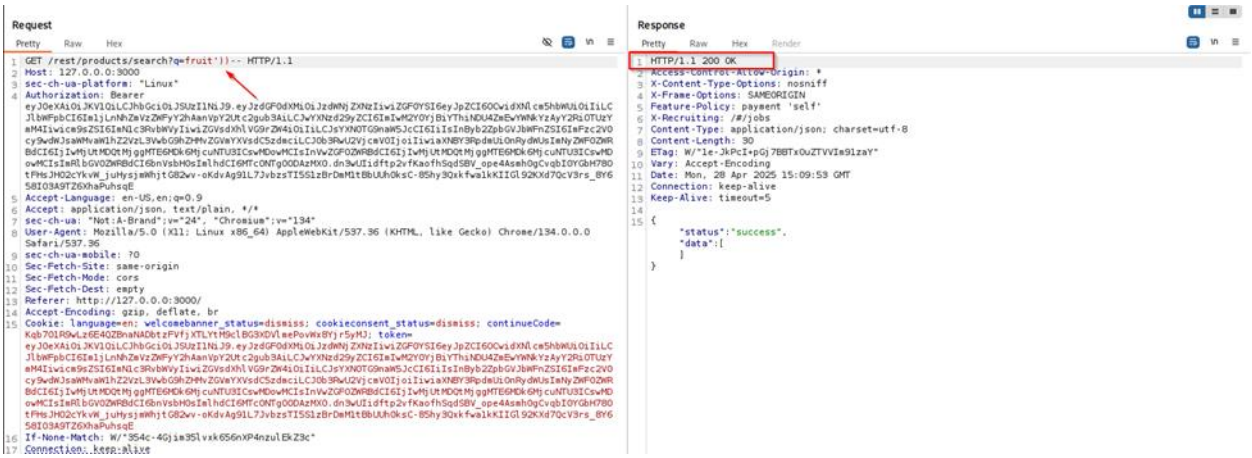
1. Perform any search in the search bar



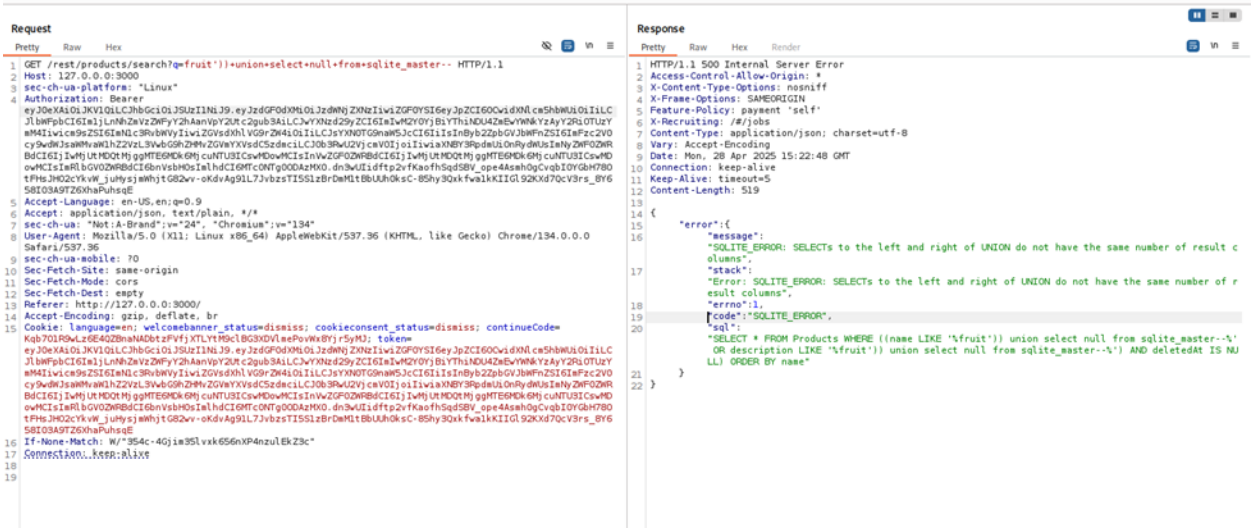
2. Get an error to know the database type and query structure that allows performing SQL injection



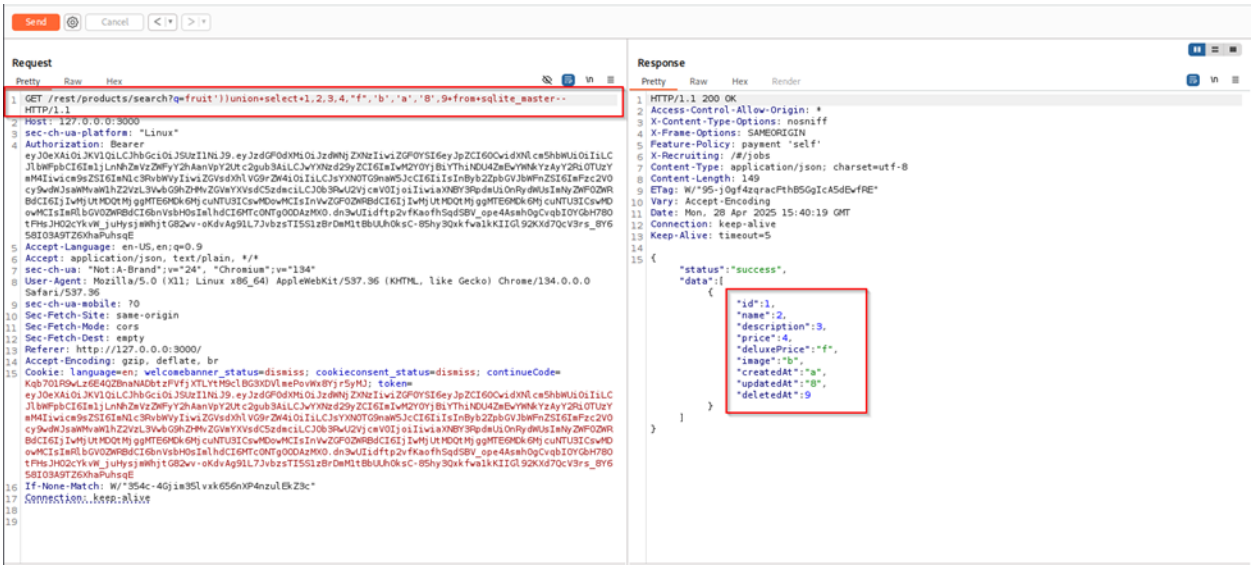
3. Breaks query



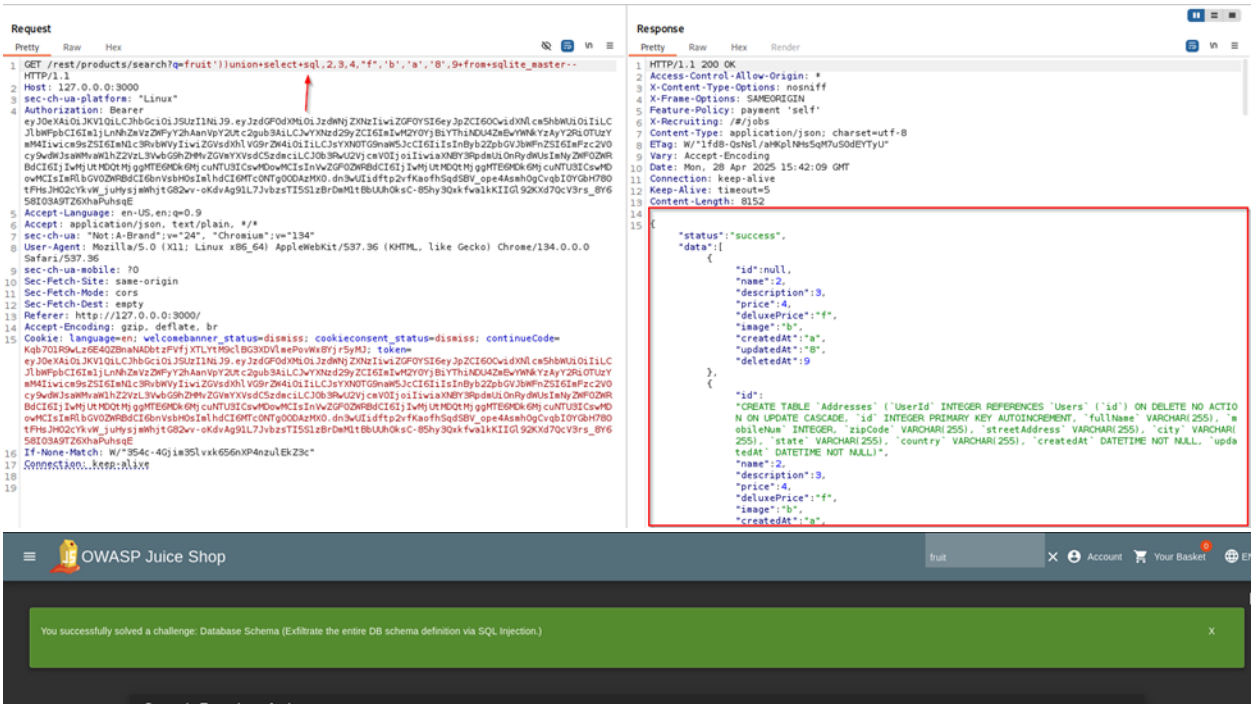
4. Try and error to get columns number



5. Specify number of columns which was 9



6. Display sqlite database schema



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 :

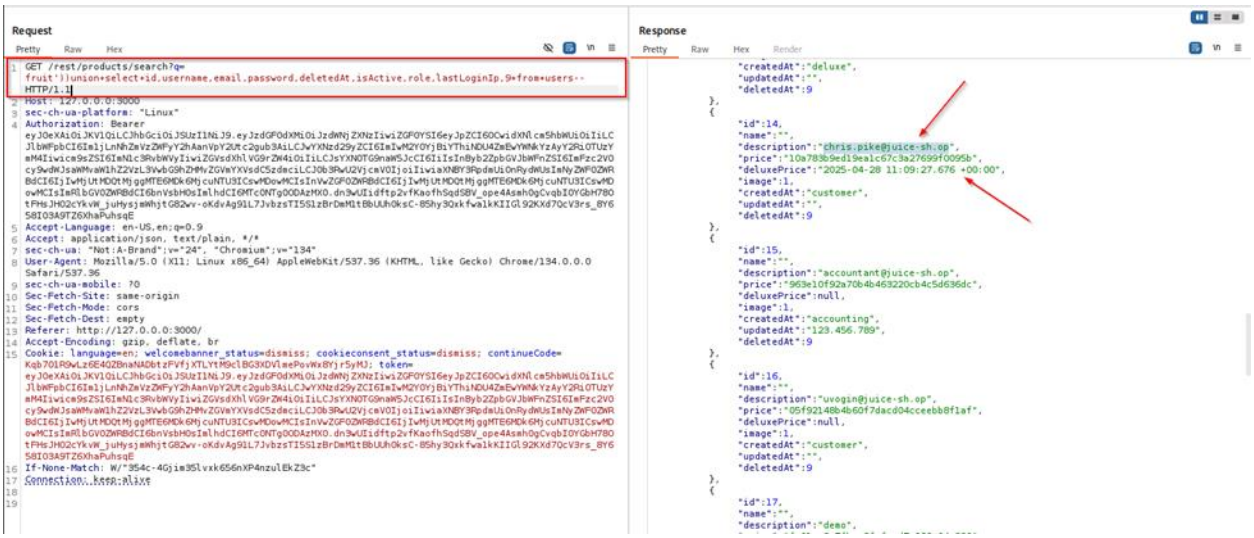
· Access Control for Deleted Users: Ensure that deleted or deactivated accounts are completely removed or blocked from login.

· Use Parameterized Queries

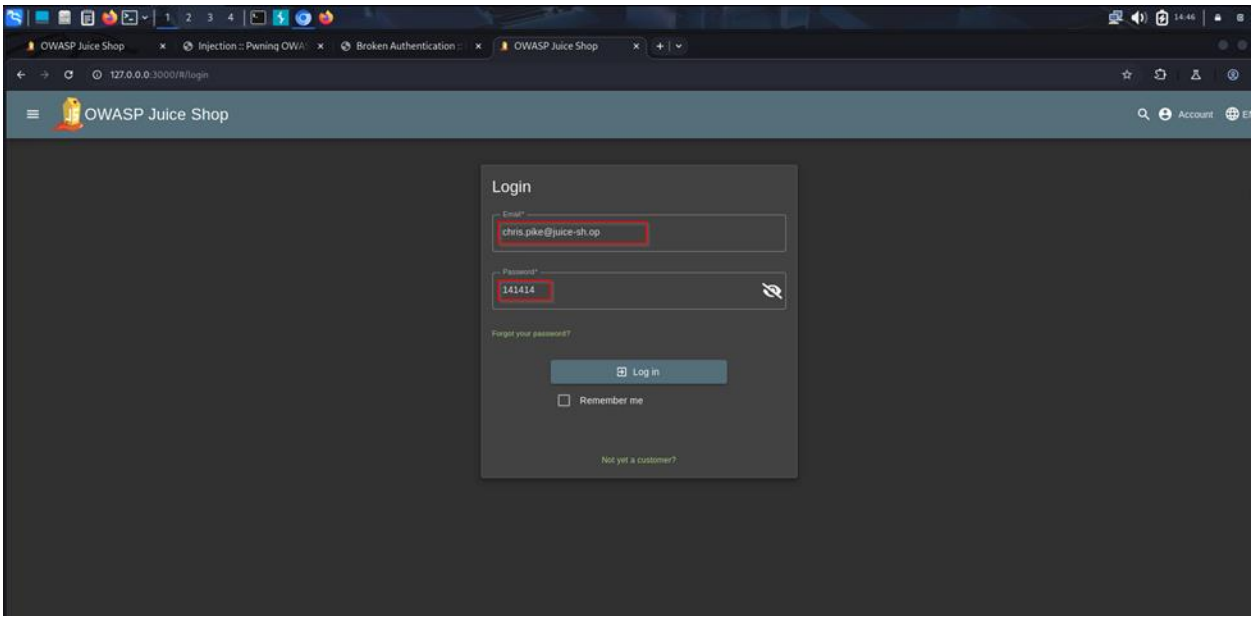
·**Sanitize Input**: Implement strict validation and allow only whitelisted search terms.

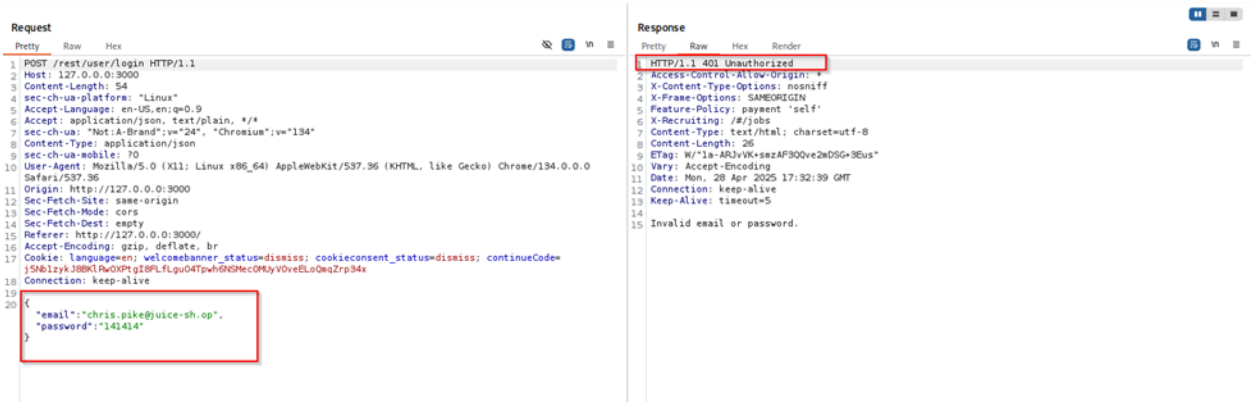
Proof Of Concept (POC):

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

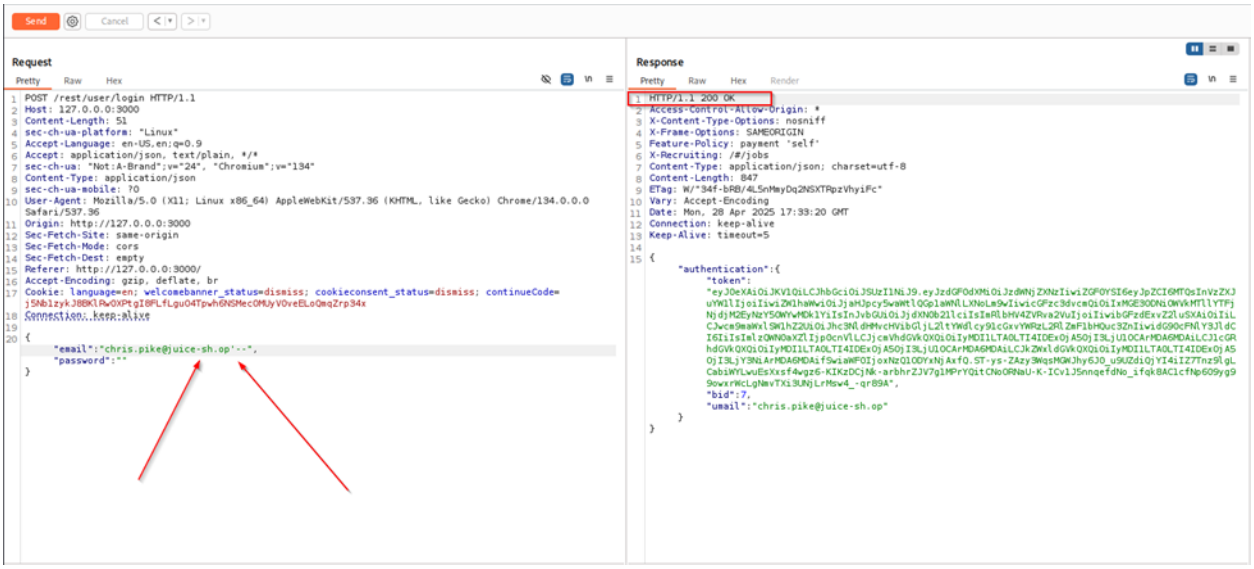


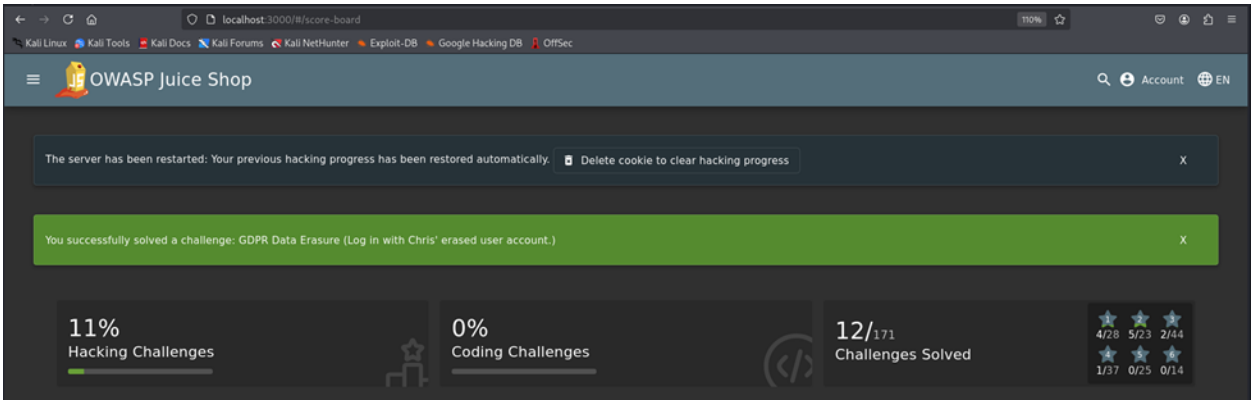
2. Write Chris' email and intercept using Burp Suite





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:

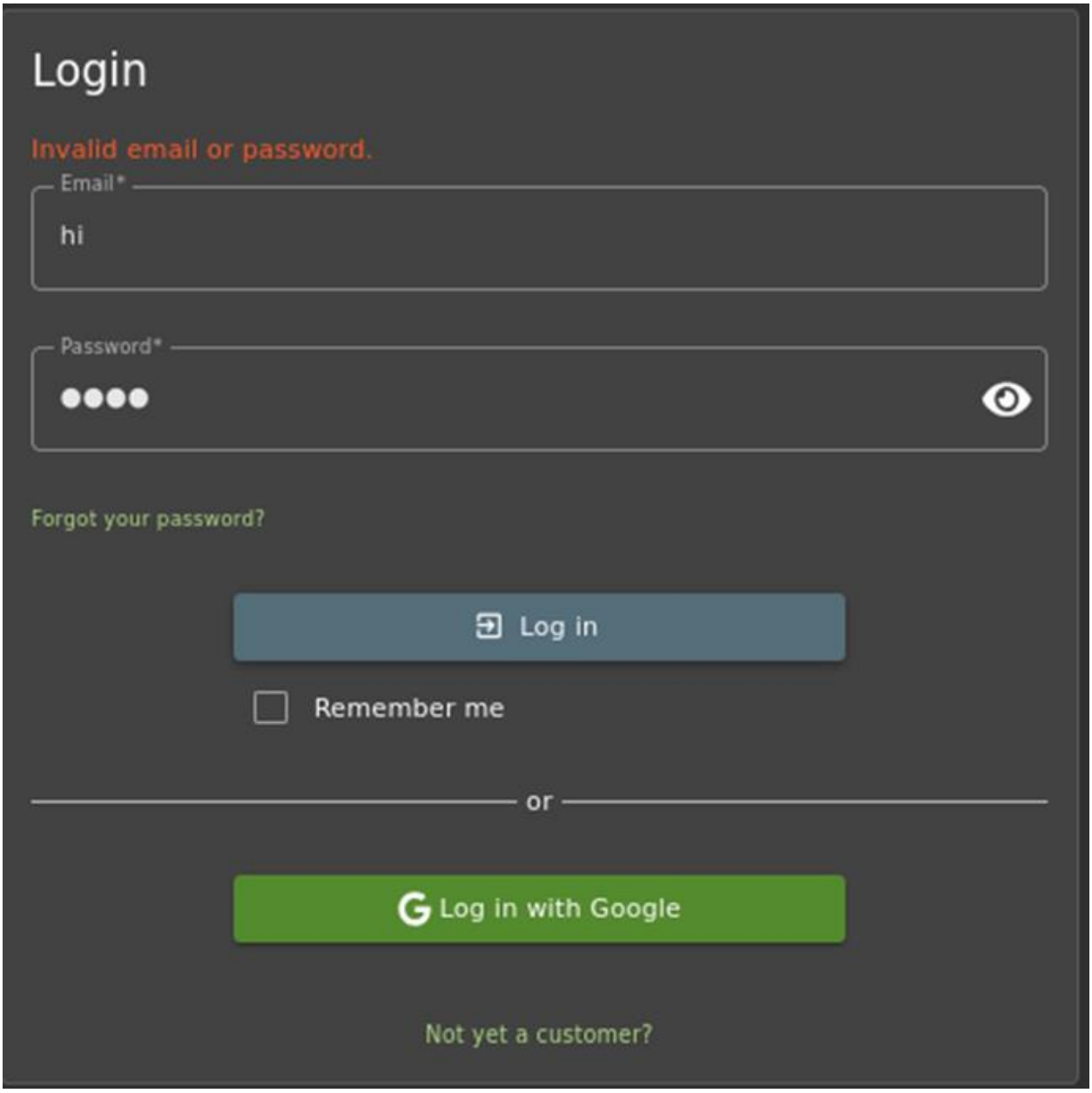
· OWASP A01:2021 – Broken Access Control
· OWASP A03:2021 – Injection
· OWASP Authentication Cheat Sheet

Recommendations:

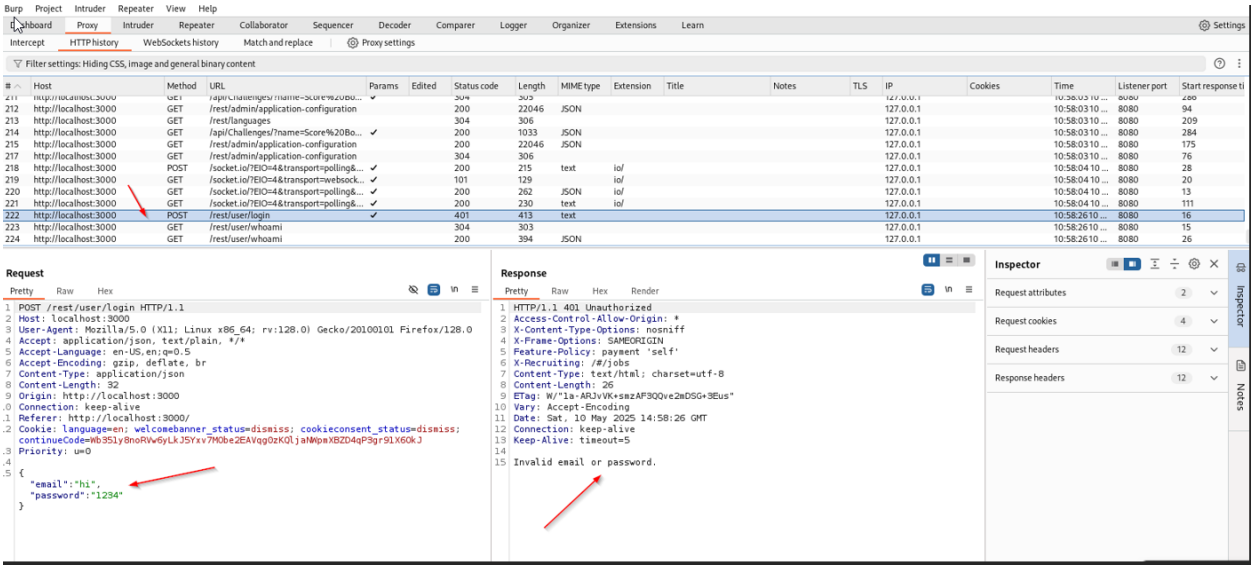
1. Enforce strict use of parameterized queries (prepared statements) to eliminate SQL Injection risk.
2. Apply server-side input validation and disallow control characters and SQL meta-syntax in user input.
3. Log and alert on suspicious authentication attempts or unusual session creation.
4. Conduct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

Proof of Concept (PoC):

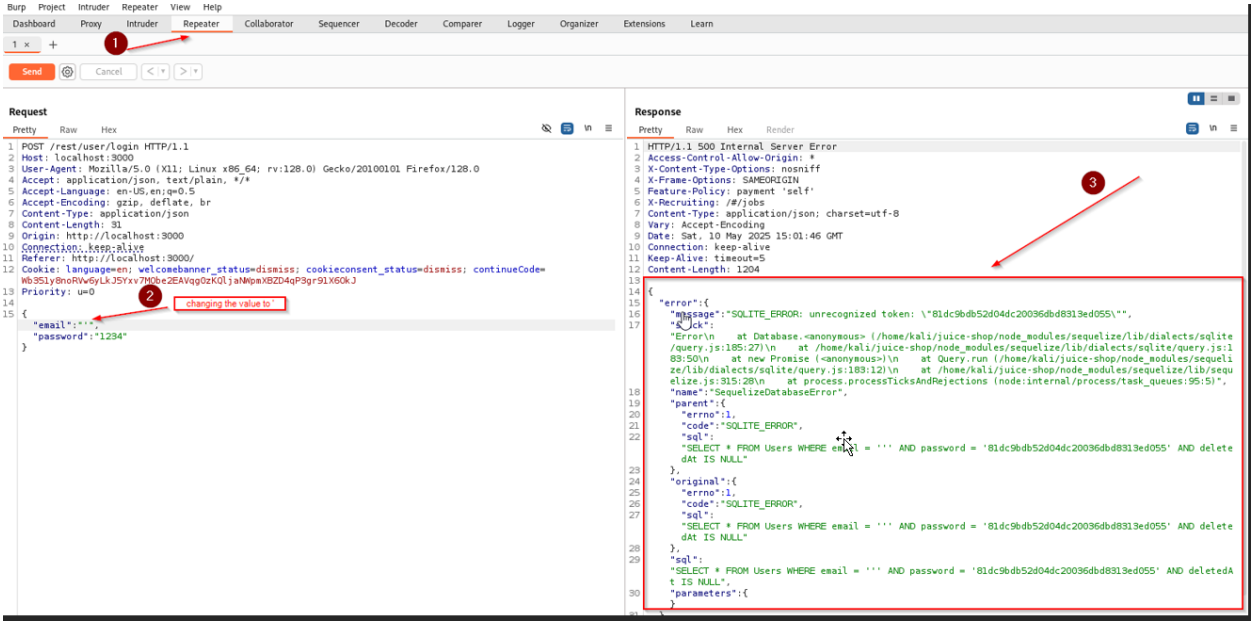
1. Navigate to the login page and enter arbitrary values in the form (e.g., email: hi, password: 123) and submit.



2. Intercept the request using Burp Suite.
   Capture the POST /rest/user/login request and send it to the Repeater tab.

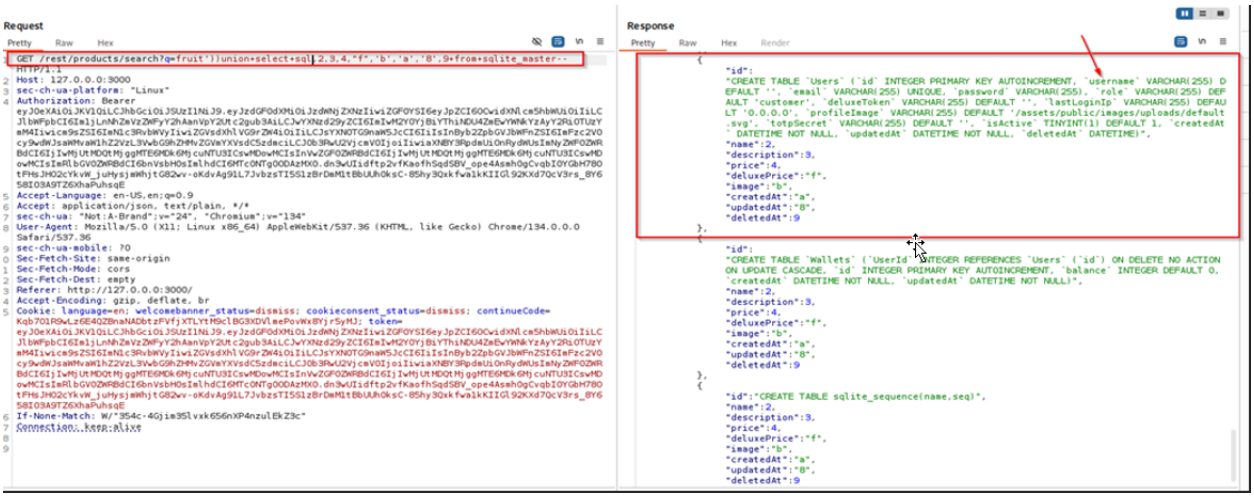


3. Modify the JSON body of the request by setting the email parameter to: "email": ""

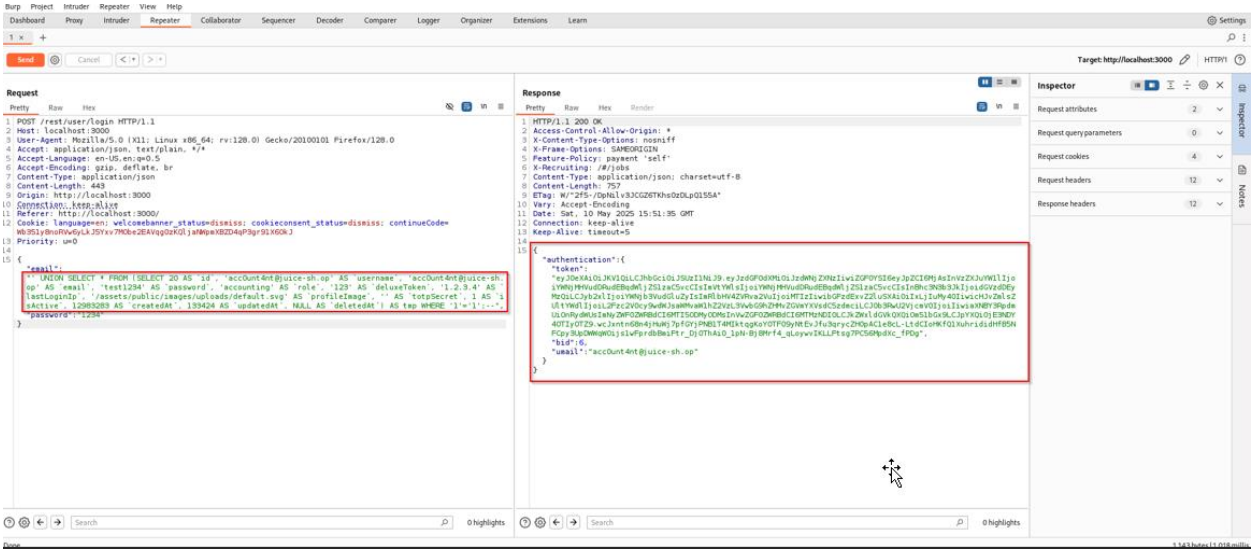


A verbose SQL error is returned in the response, indicating an injection point.

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

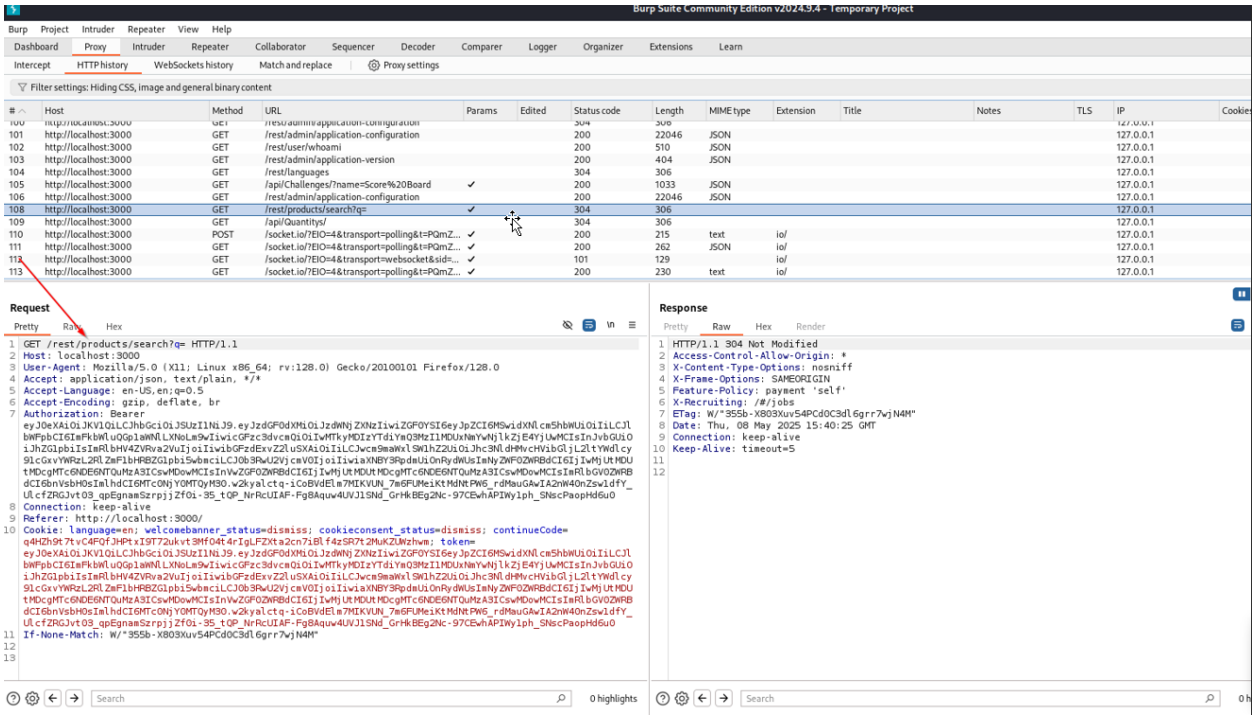
Recommendations:

- 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):

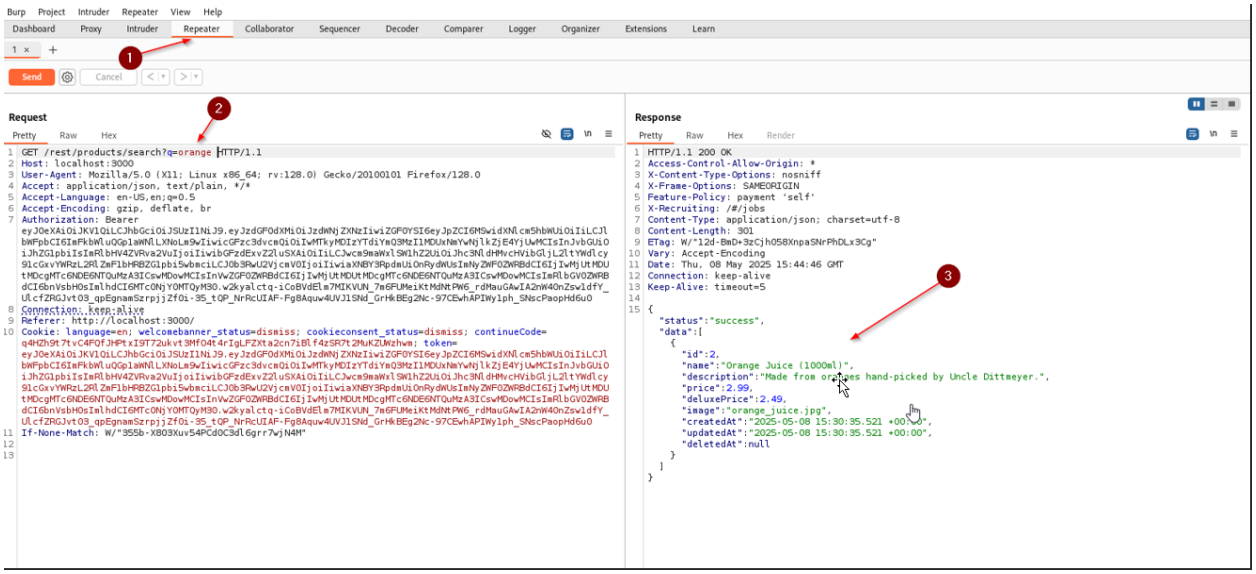
1. Log in to Juice Shop as a regular user.

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

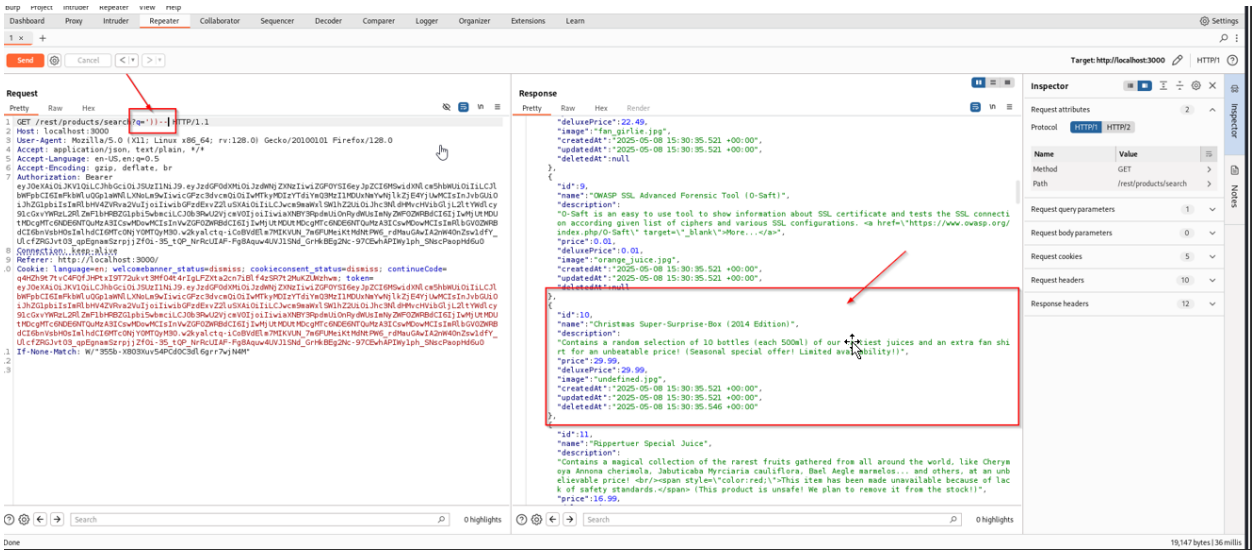


.

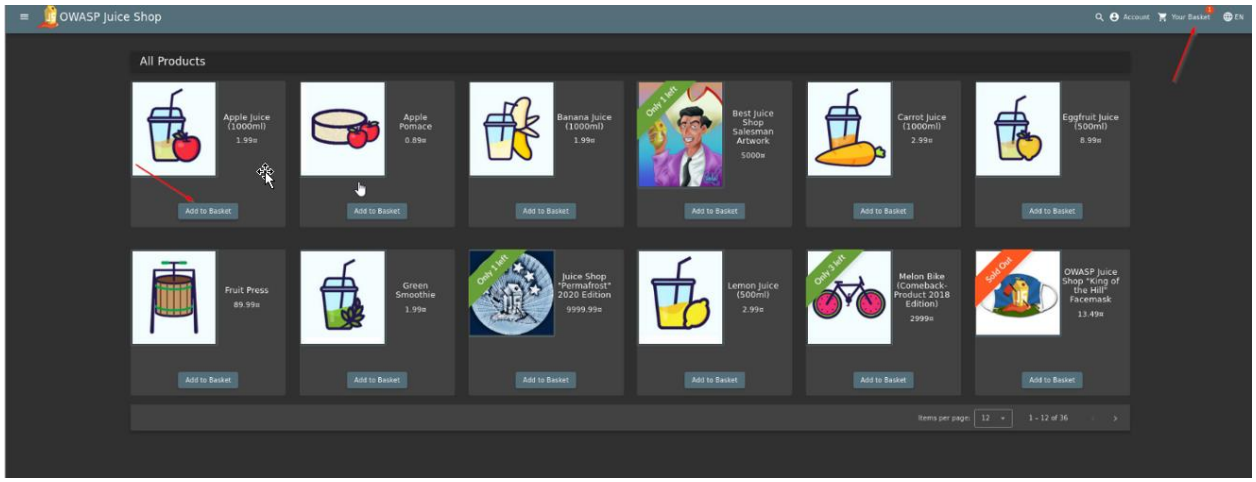
3. Changing the ‘q’ parameter with any item to get its information in the response.

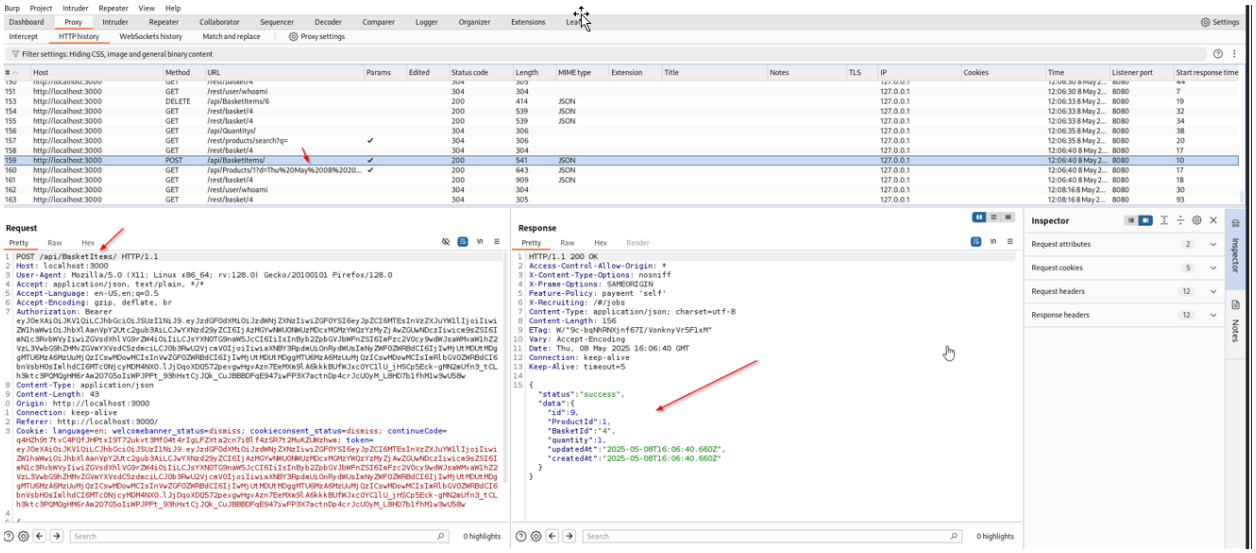


4. By applying the SQL query ‘))-- in the q parameter the response is all the products included hidden ones including Christmas special offer of 2014 with productId=10.

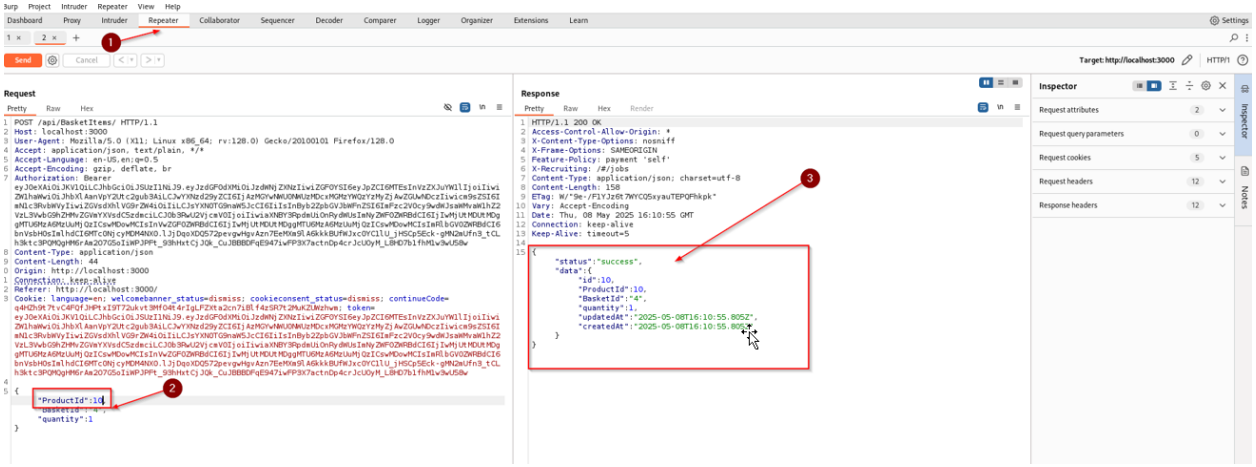


5.The penetration tester added any item to the basket and intercepted the POST request .



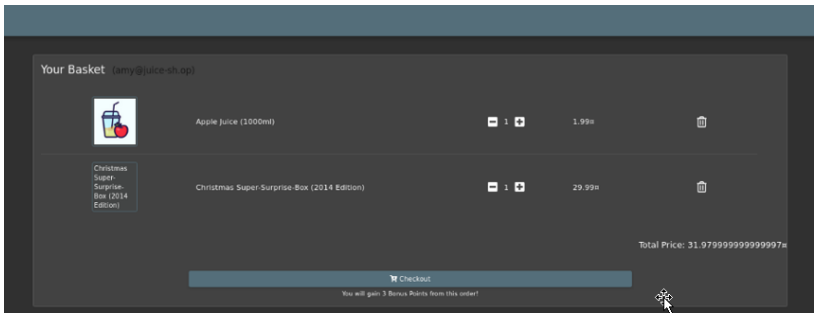


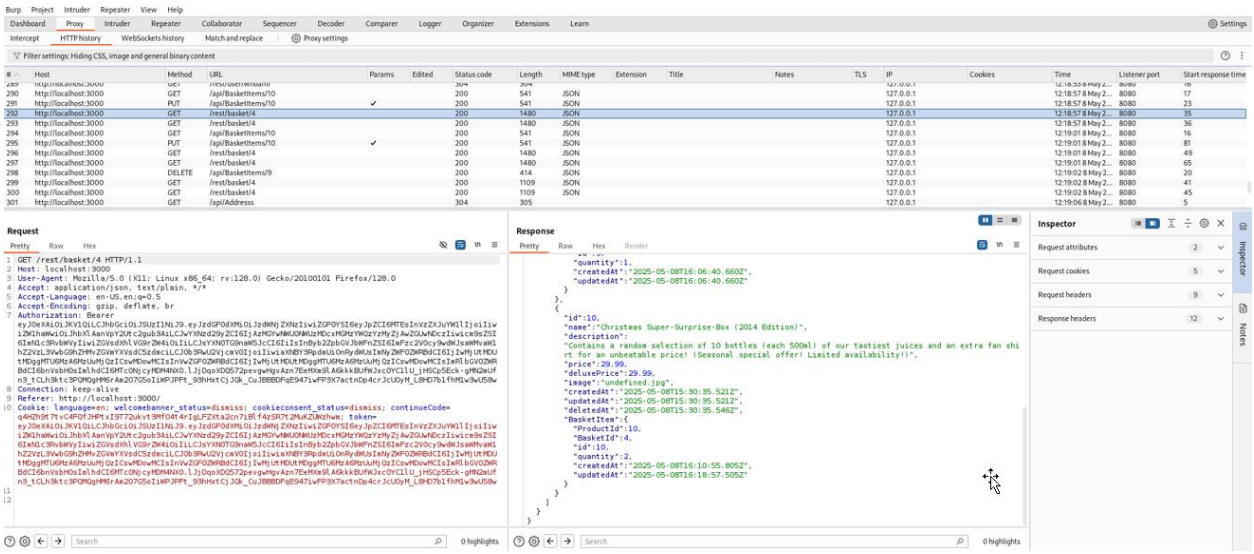
6.Sending the POST /api/BasketItems request to the repeater , then changing the productId parameter in the request body with ‘Order the Christmas special offer of 2014’ productid found in 4 .



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 .


Your Basket (amy@juice-sh.op)
Apple Juice (1000ml)
1.99¤
Christmas Super-Surprise-Box (2014 Edition)
29.99¤
Total Price: 31.979999999999997¤
Checkout
You will gain 3 Bonus Points from this order!


Request
Response
Inspector

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:

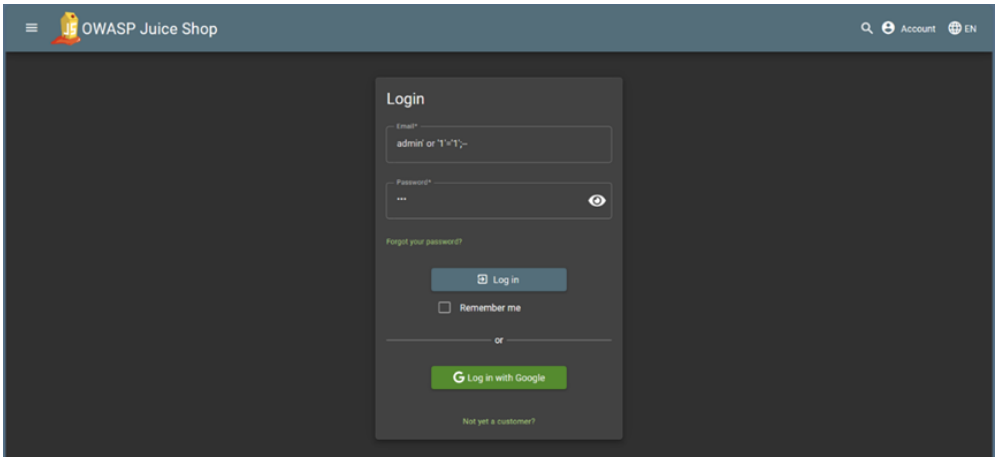
- CWE-284: Improper Access Control

Recommendations:

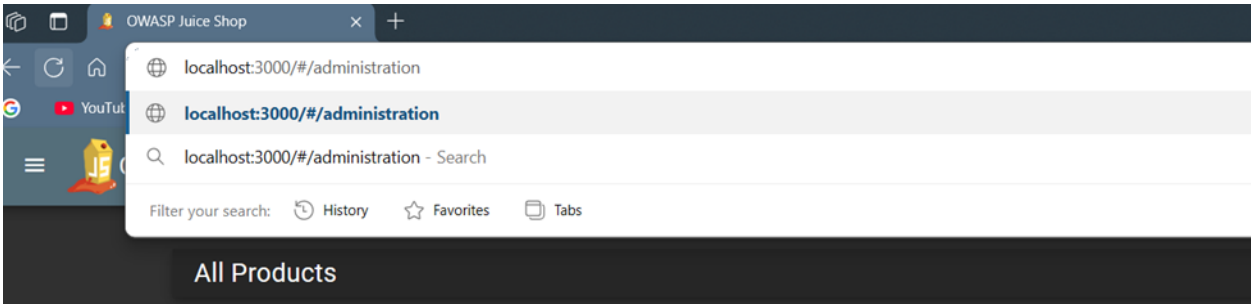
1. **Enforce Authorization Checks:**
    - Ensure users can only modify their own feedback.
2. **Reject Unauthorized Modification Attempts:**
    - Return a 403 Forbidden status for unauthorized edits.

Proof of Concept (PoC):

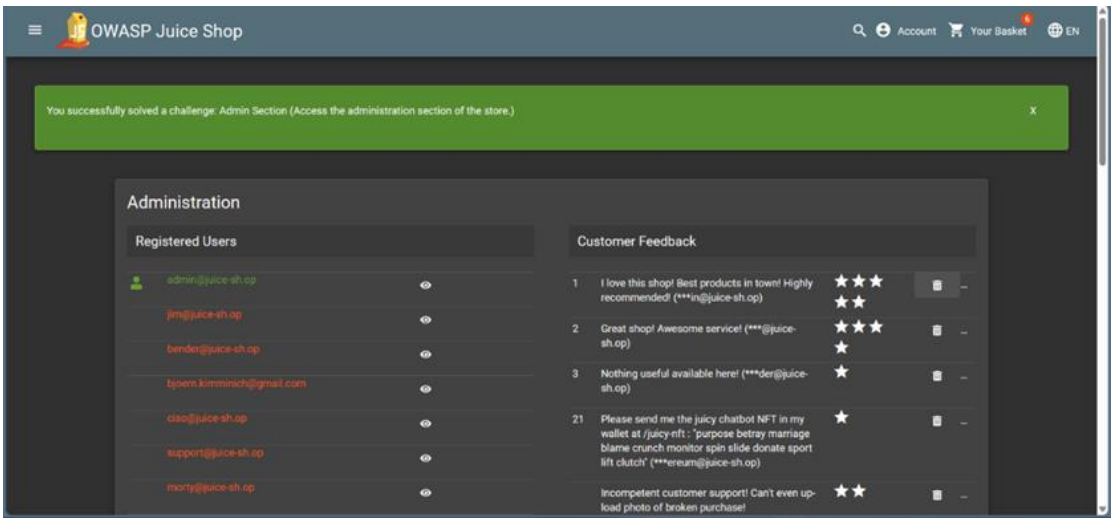
1. Log in to the Juice Shop as an admin using SQL injection.



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:

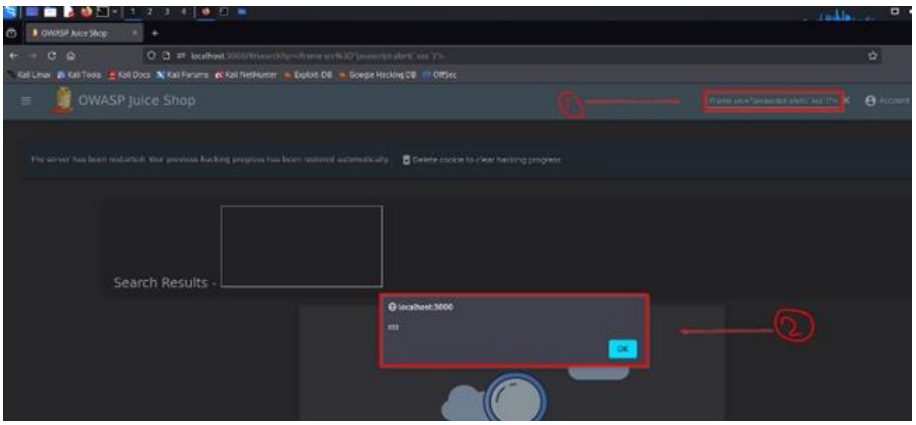
1. Sanitize and encode all user input before inserting it into the DOM
2. Avoid using inner HTML or other unsafe DOM methods with user data.
3. Use a library such as DOM Purify to clean dynamic content.
4. Implement a strict Content Security Policy CSP to reduce XSS risks.

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.

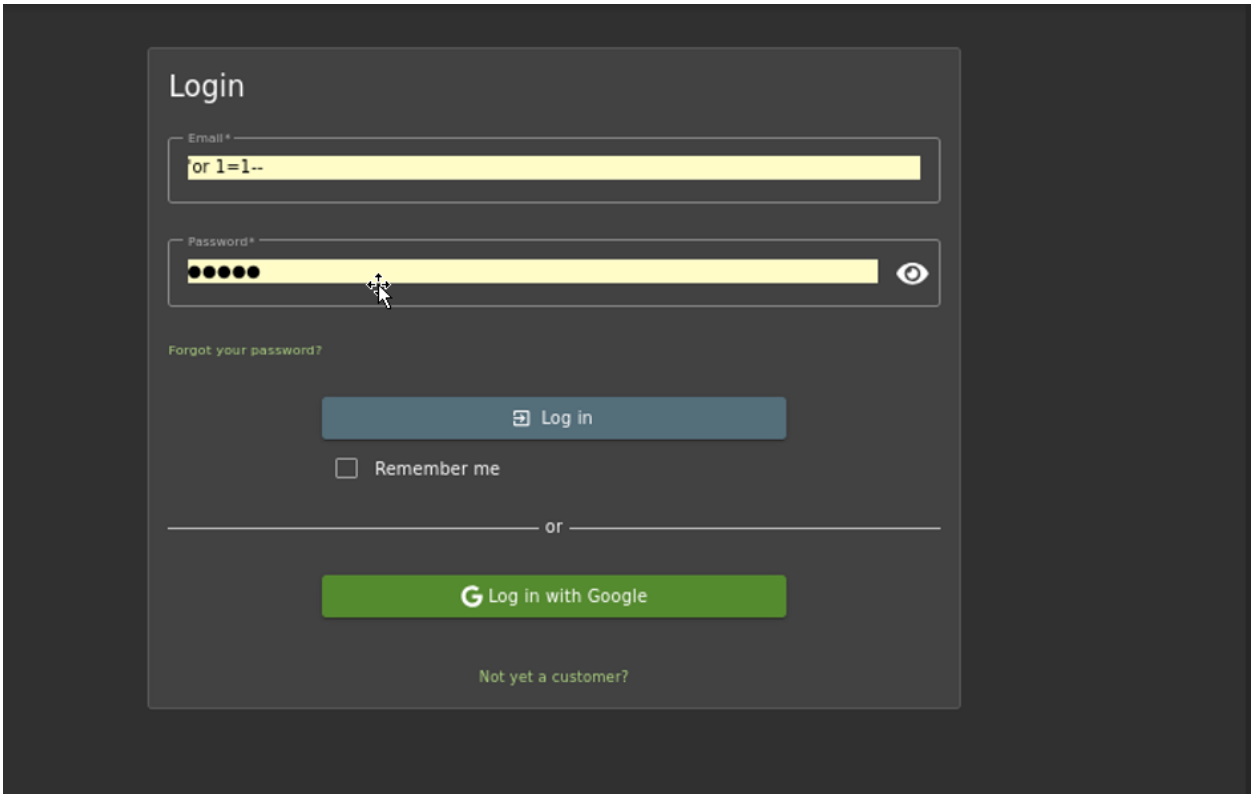
Apply Content Security Policy (CSP) headers to restrict script execution sources.

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

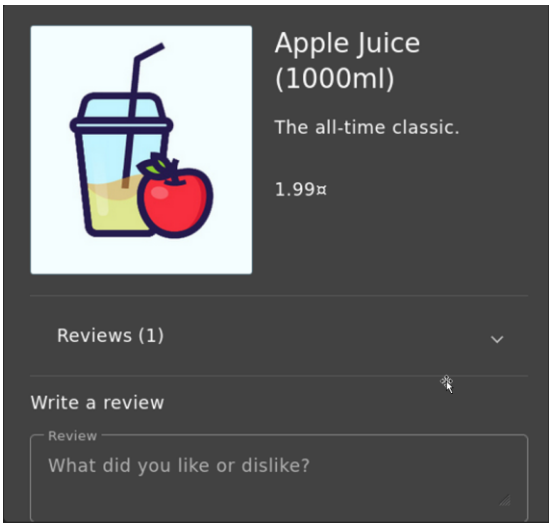
Implement proper access controls to prevent direct API manipulation.

Proof of Concept (POC):

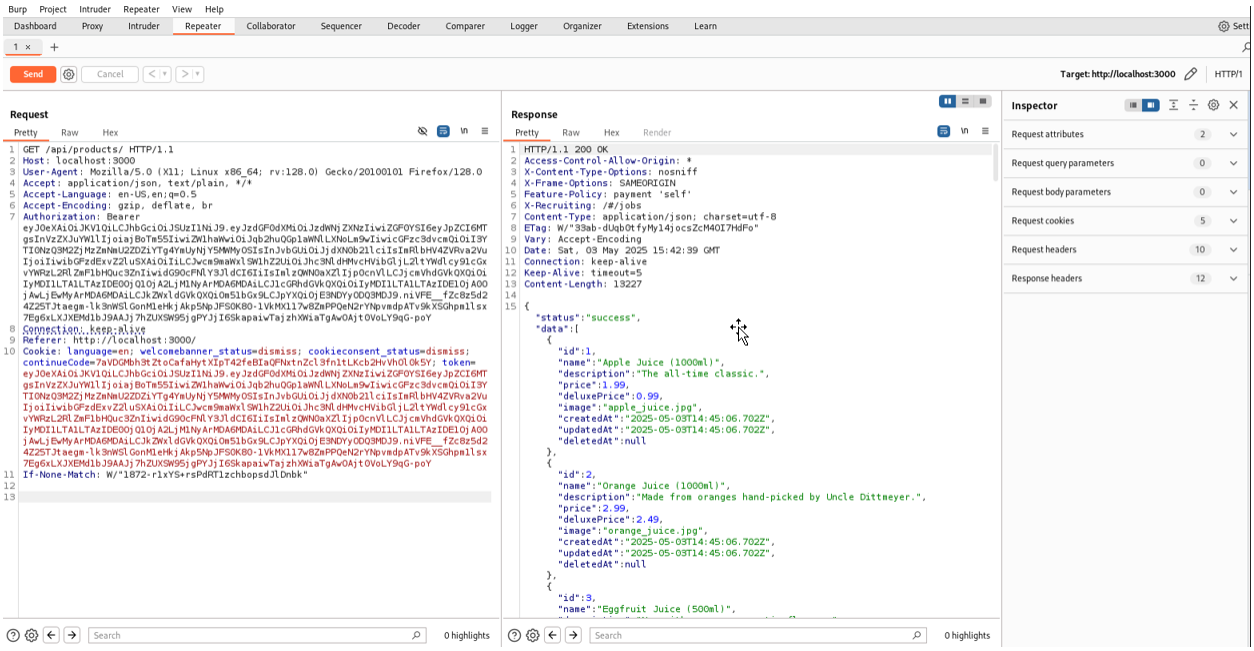
1. Log in to Juice Shop as a user with admin privileges using previous vulnerability (Ref. ID:3.1).



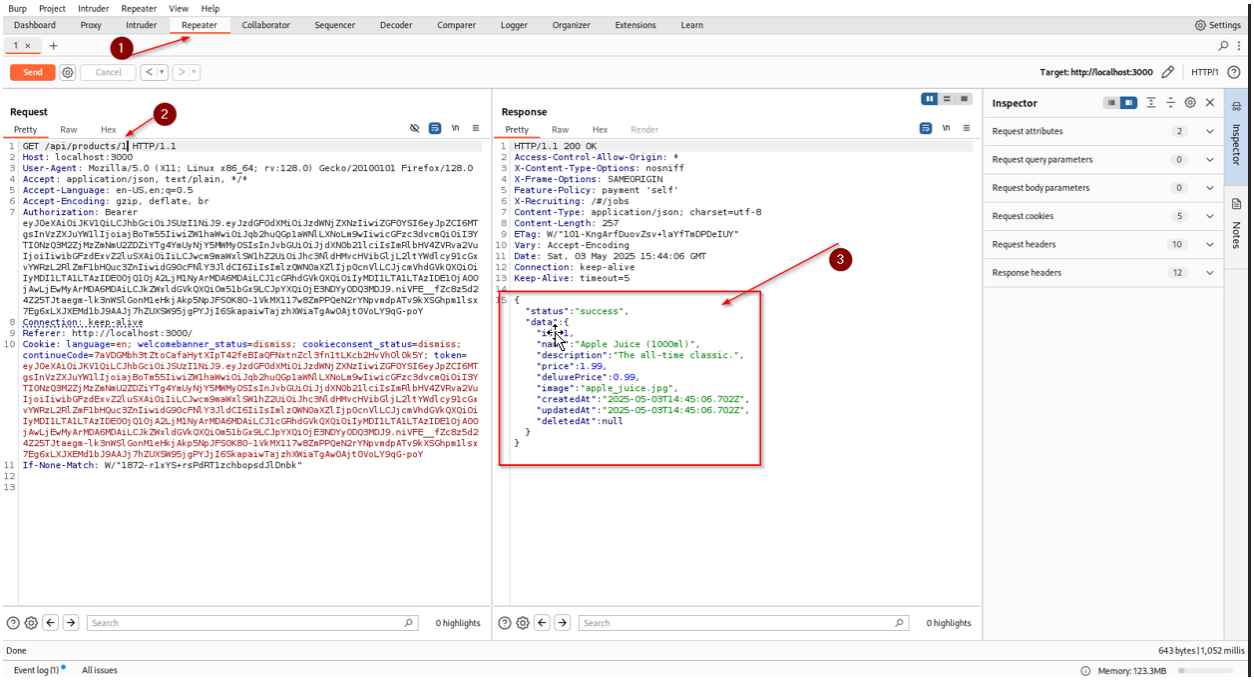
2. Click on any product to find its details .



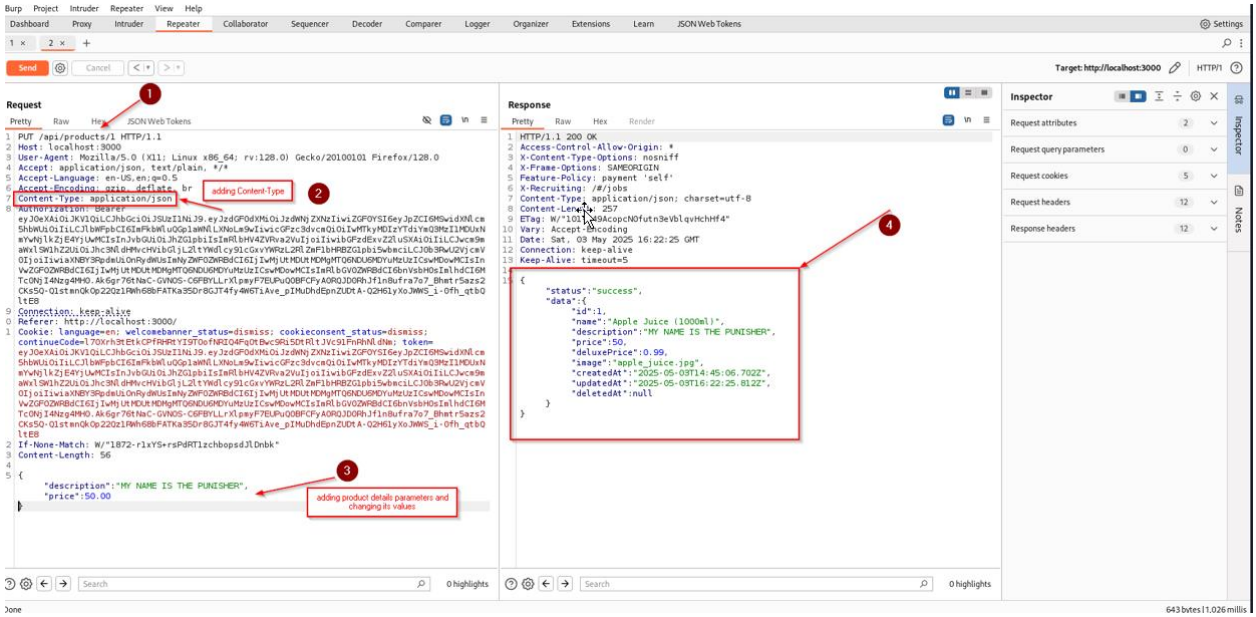
3. Use Burp Suite to intercept a GET request /api/Products/{id} and send it to the repeater.



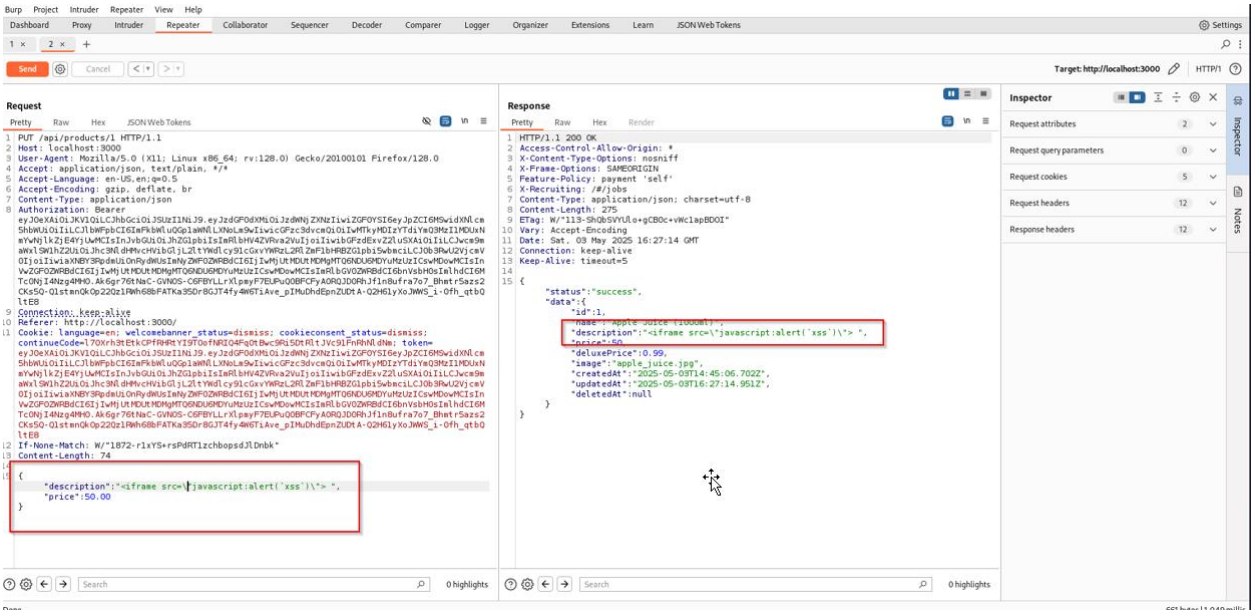
4. By adding any id number in /api/Products/{id} (e.g.1),the response is the details of the product .



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 .



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 .



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/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_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/

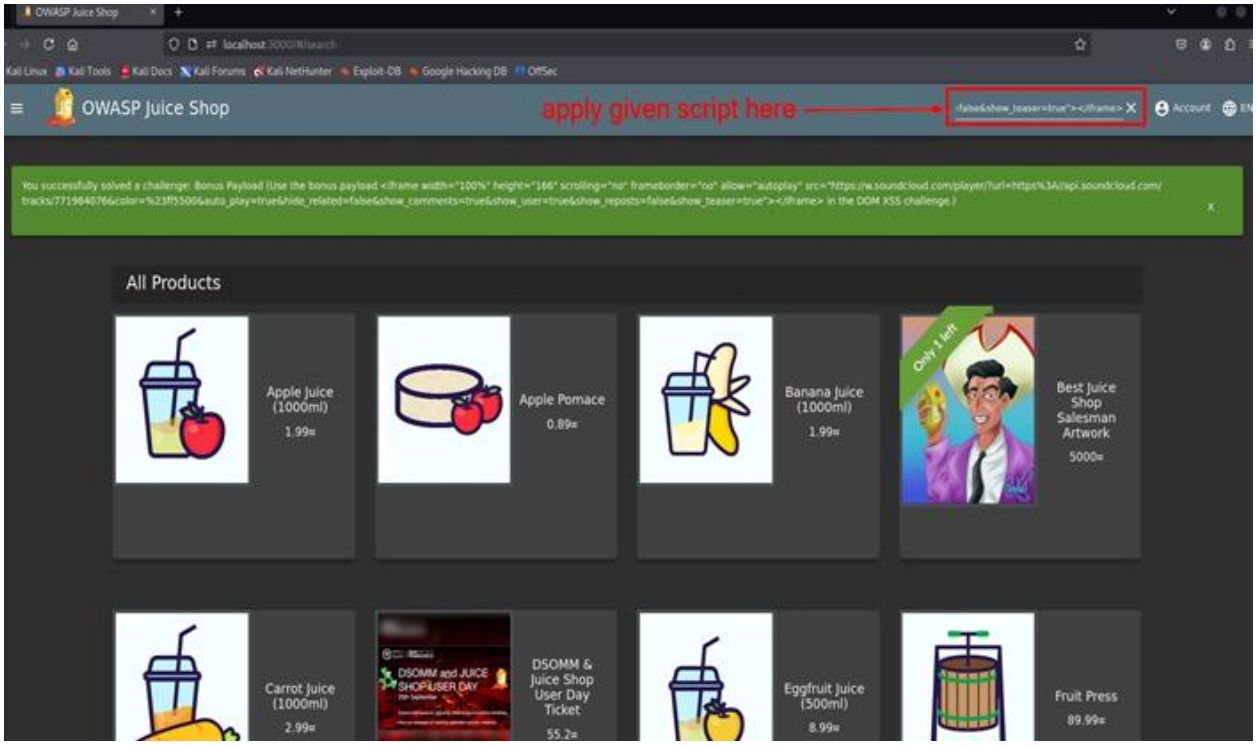
Recommendation

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.

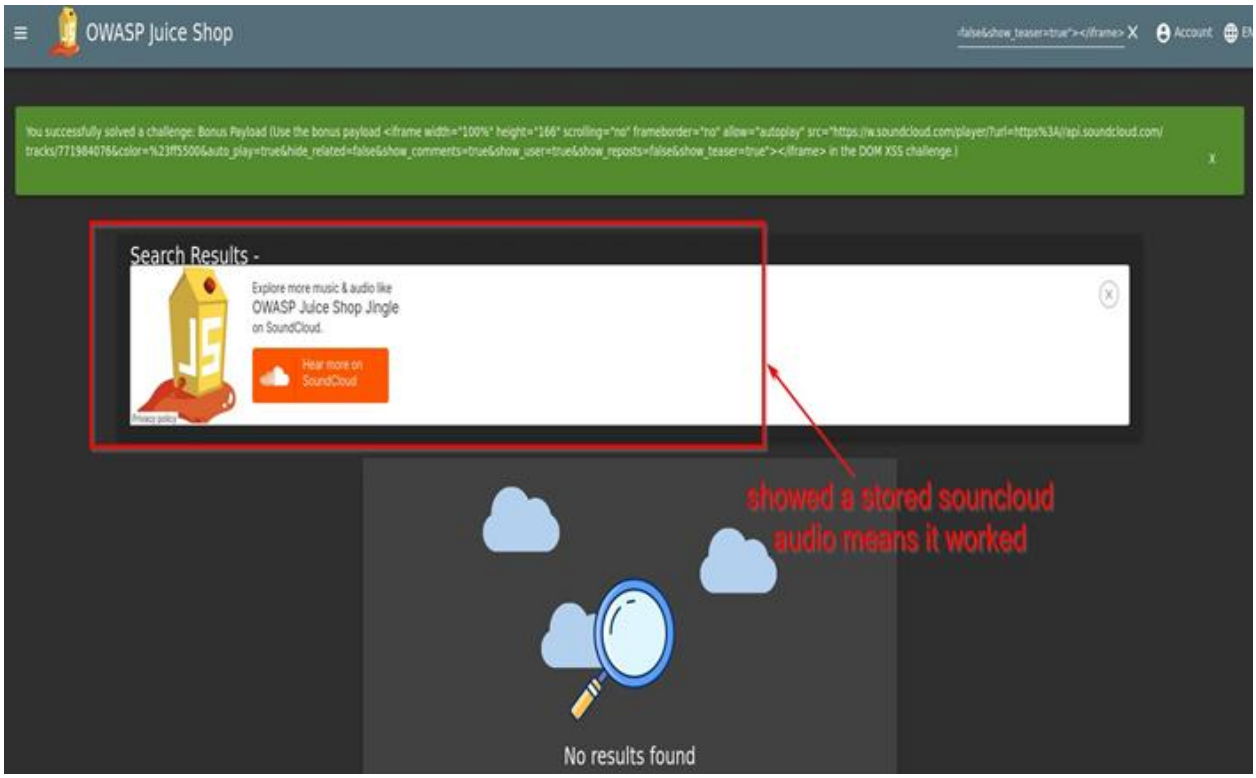
proof of concept

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=f
</iframe>



2. if it shows as below means it worked



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.

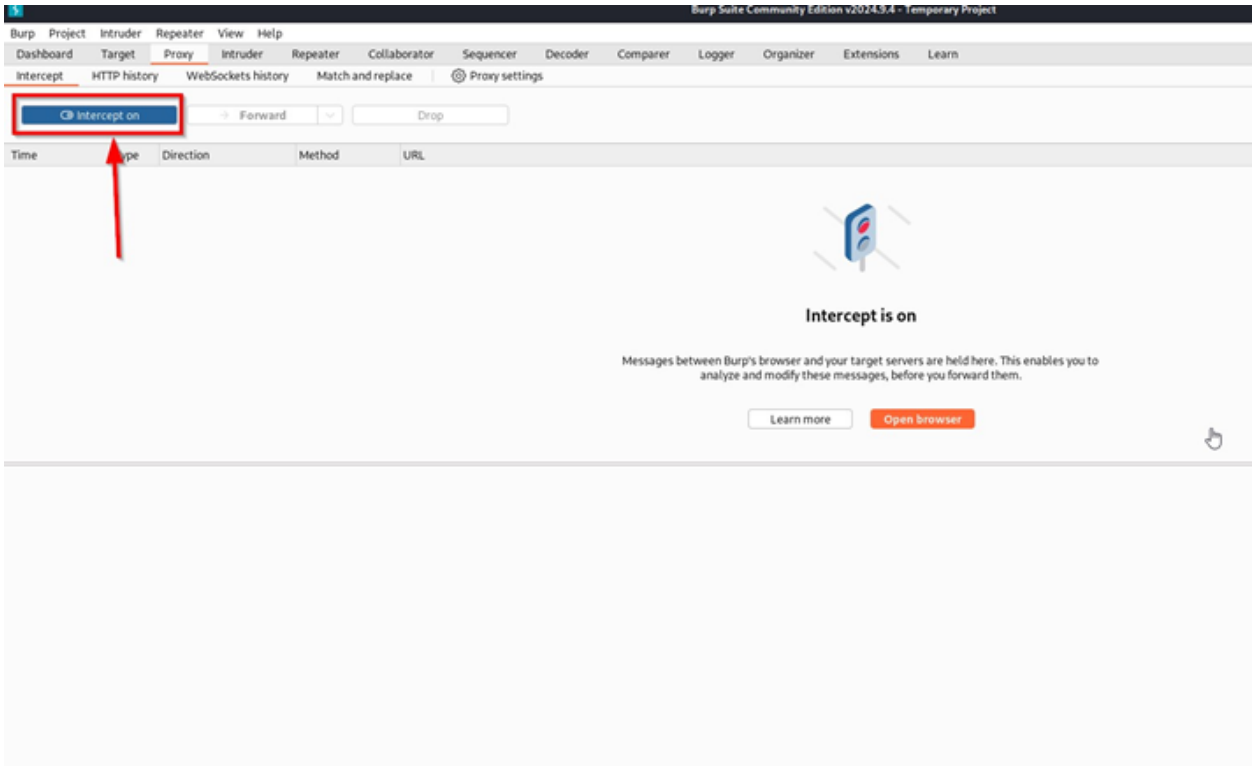
Avoid exposing internal identifiers (like user IDs or basket IDs) in URLs or client-side code whenever possible.

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

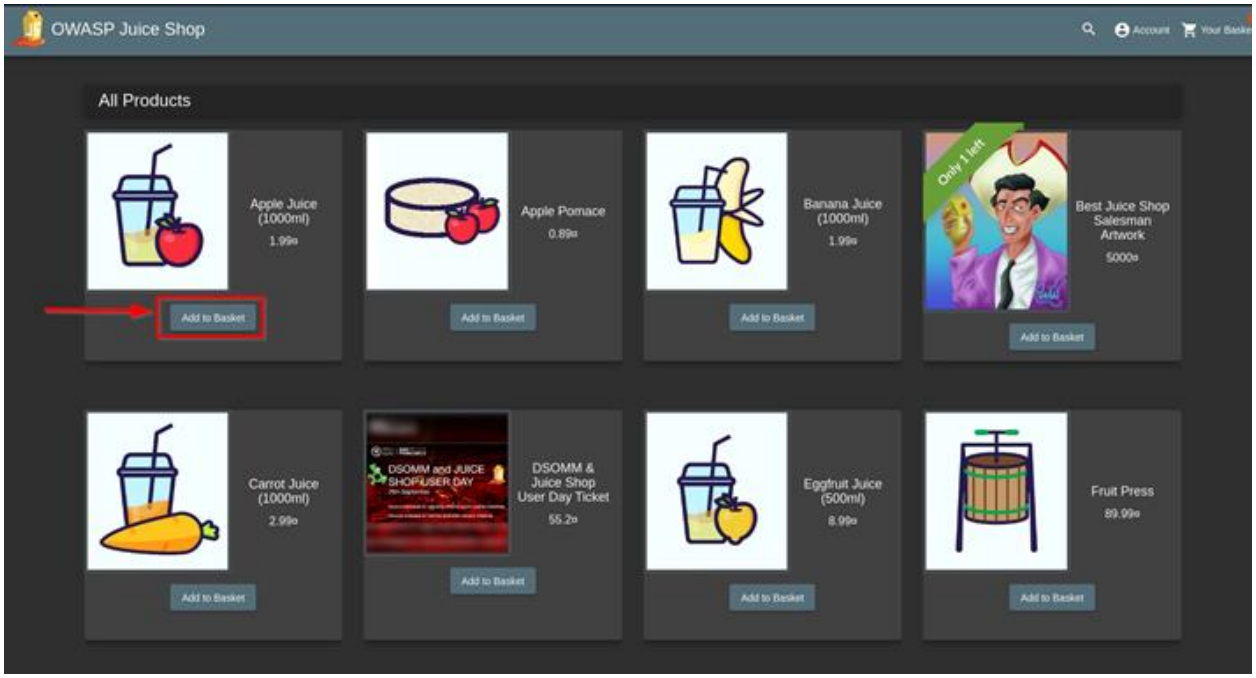
Perform regular security testing (e.g., access control reviews and pentests) to identify and fix IDOR issues.

proof of concept

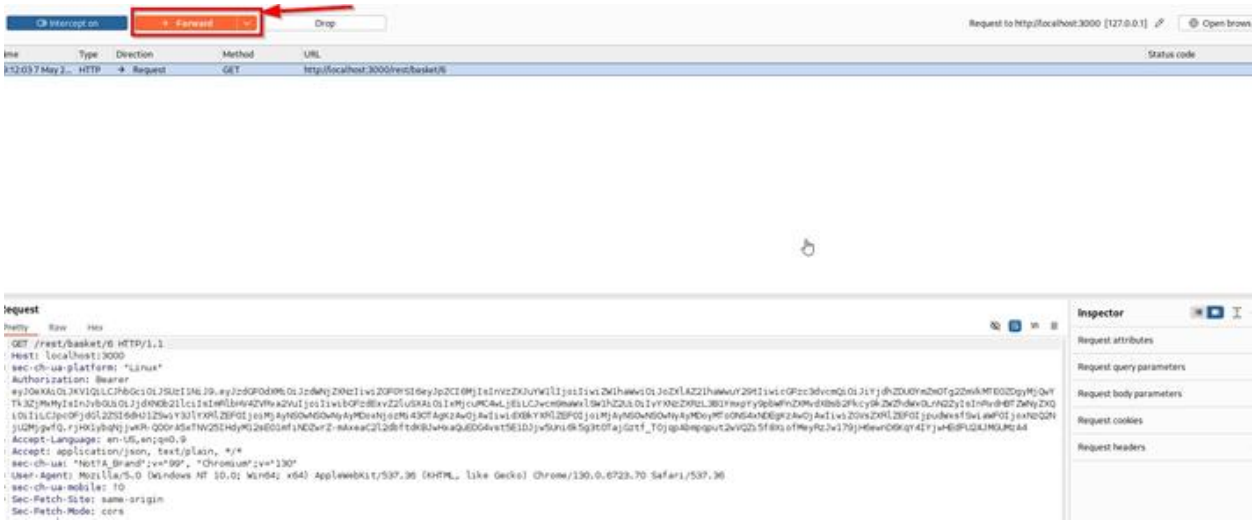
1.Open burb suite browser and open given location in it and click on intercept



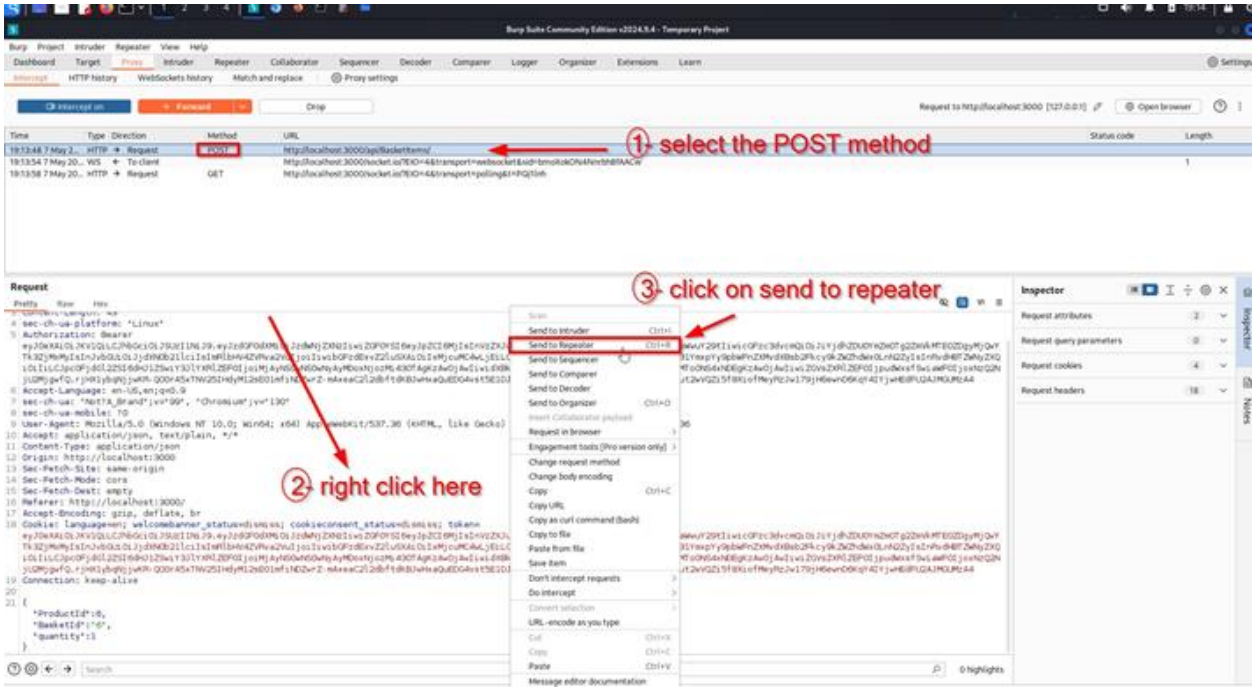
2.Click on add to basket as shown



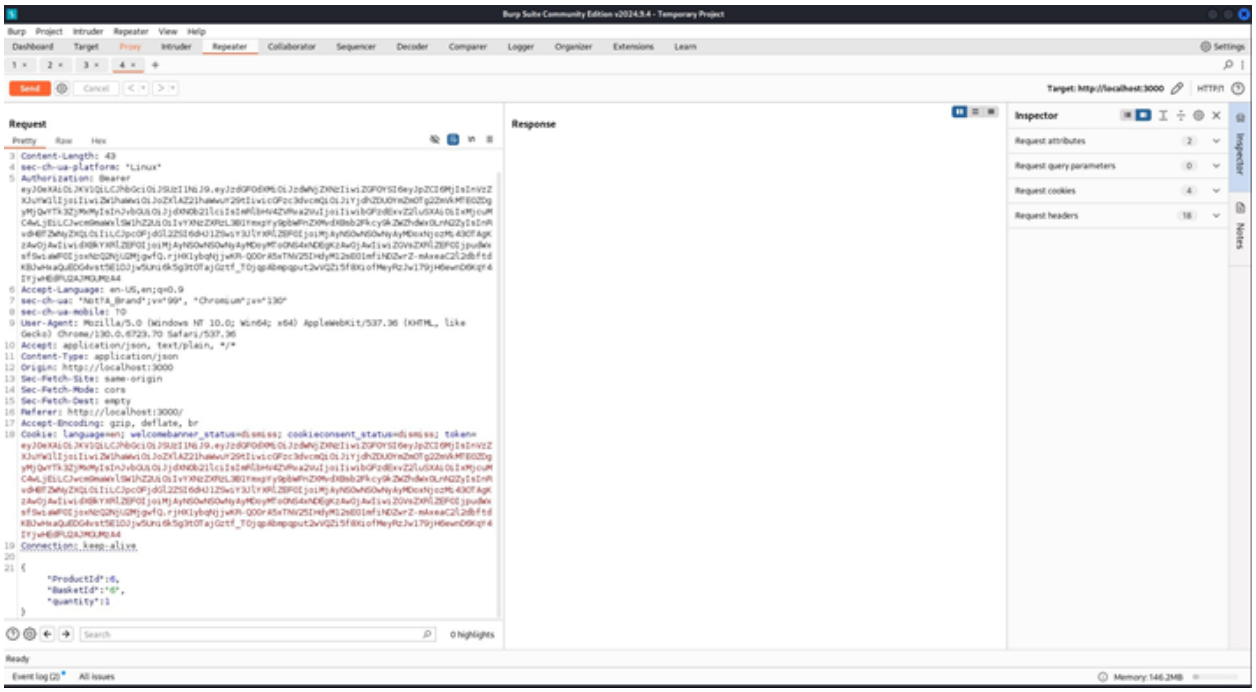
3.Open burb again and watch GET request then forward it



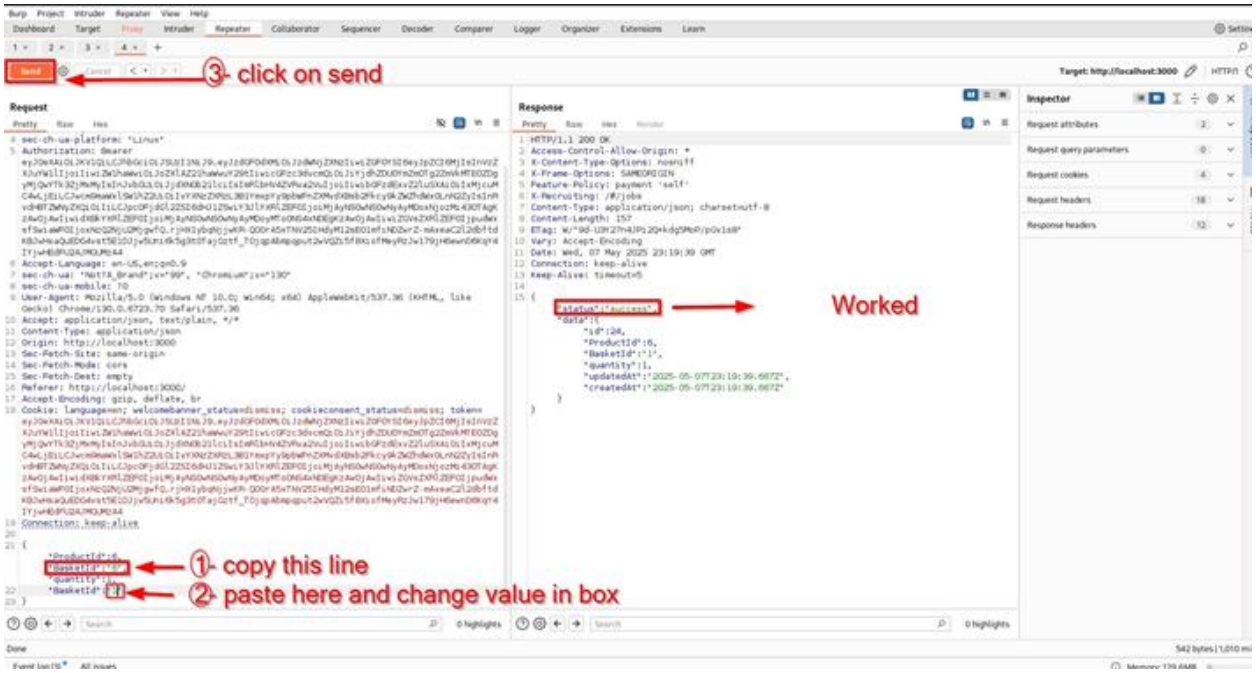
4.Do as shown below



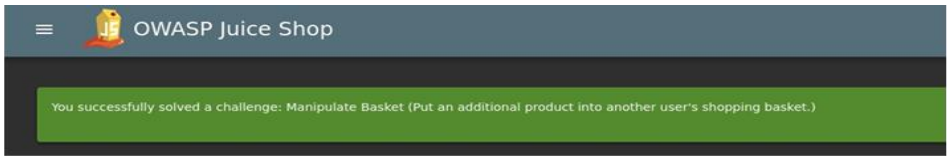
5.This is how it looked before editing



6.Copy BasketId and change it as shown below



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 Broken access control,should not be able to.

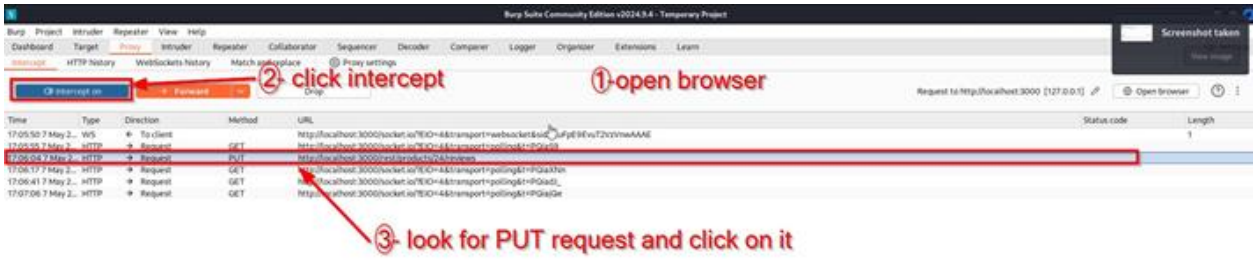
Recommendation

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

proof of concept

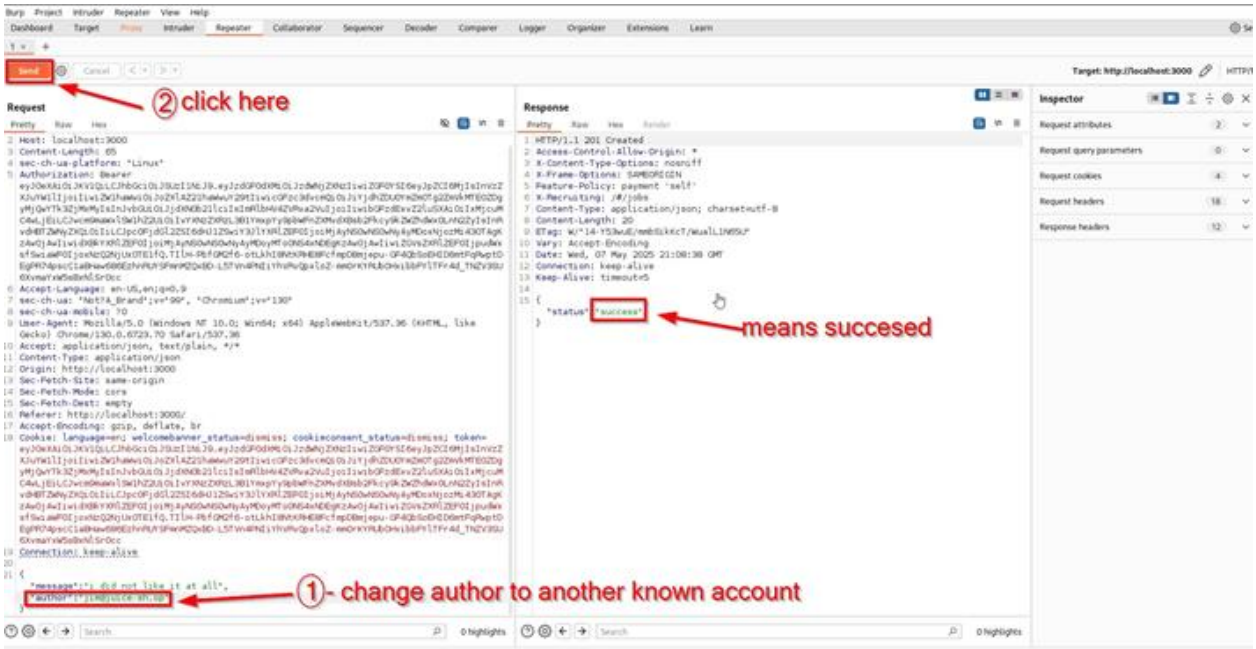
1. open burp suite
2. open browser and enter the link of owasp
3. open intercept then add a comment on any picture

4. Intercept to view what happened and click as shown

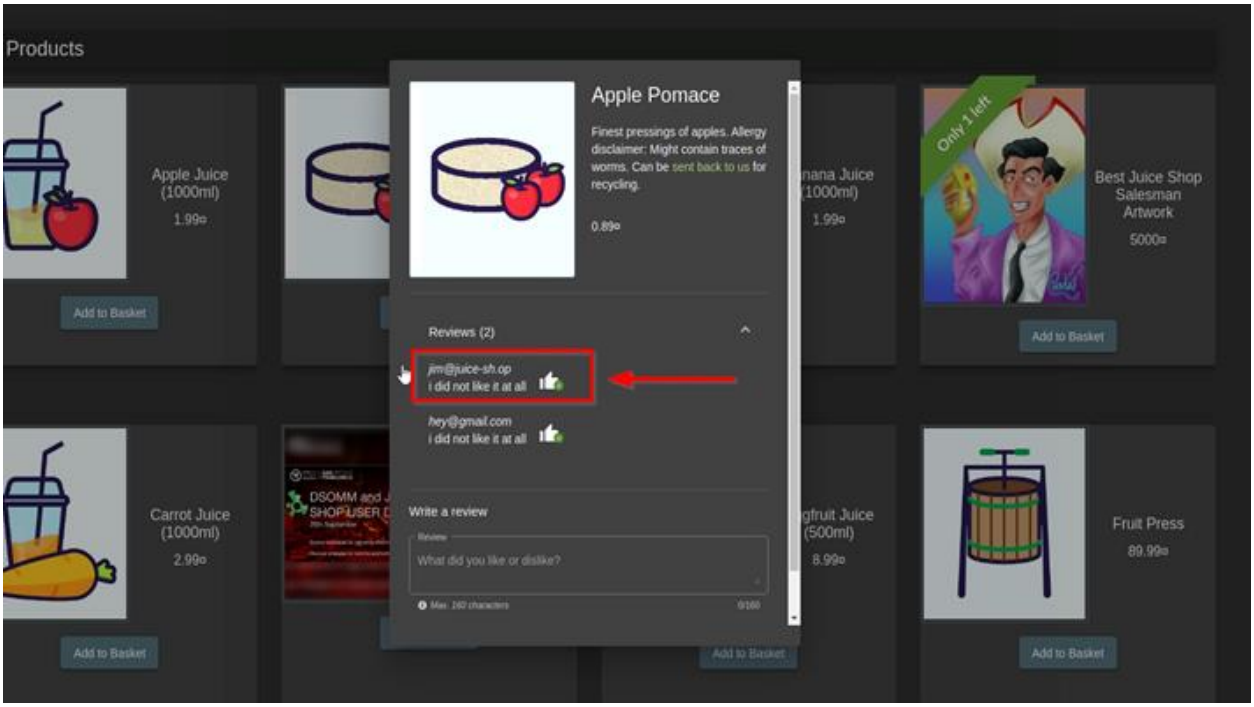


5.Right click in request place and send to repeater or click on ir and press (ctrl + R)

6.At repeater do as shown below



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



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)

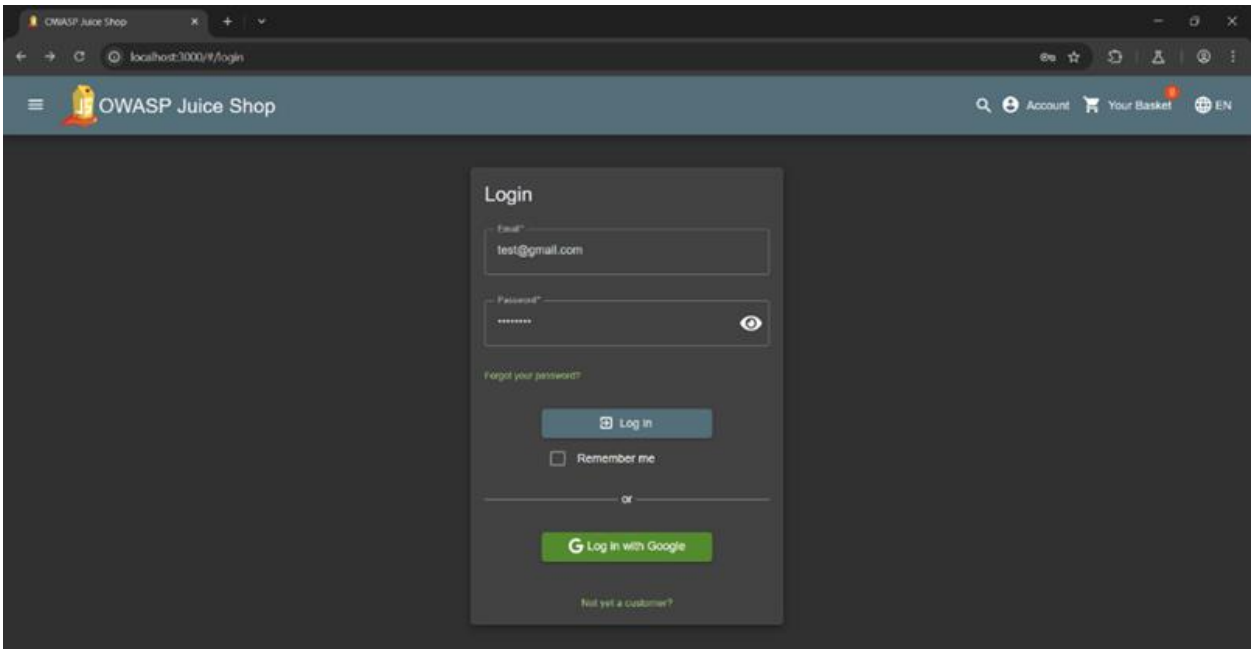
Recommendations:

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.

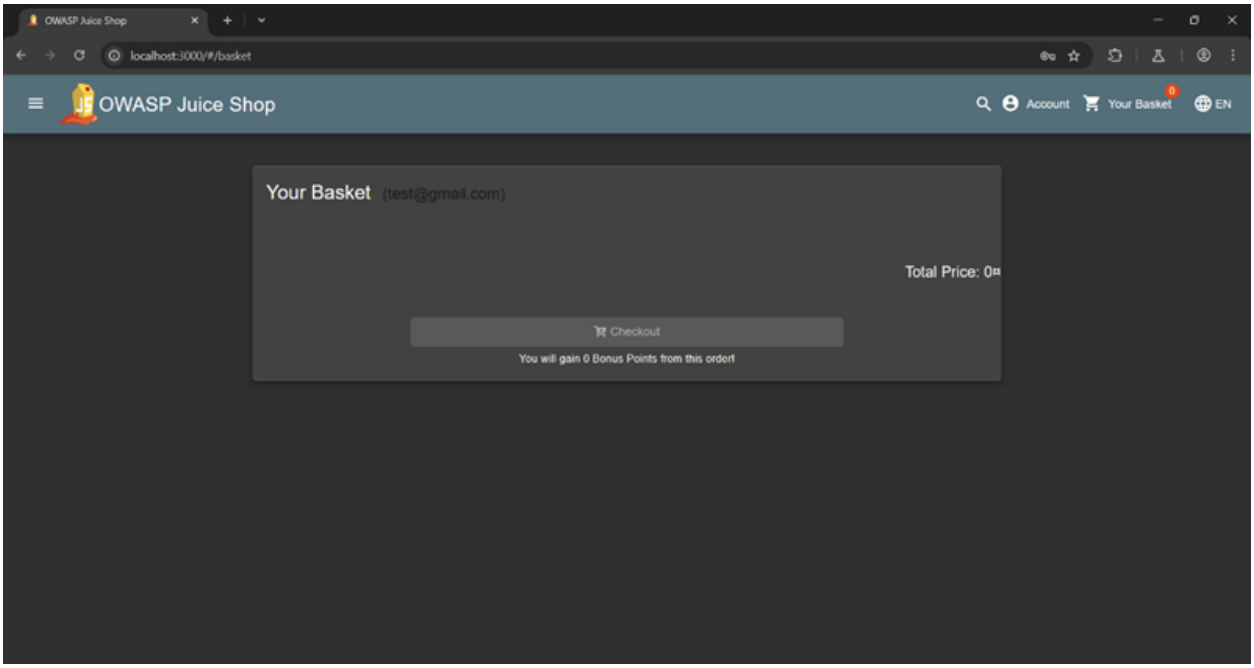
Proof of Concept (PoC):

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



2. Open the “**your basket**”

    2.1 The basket is empty

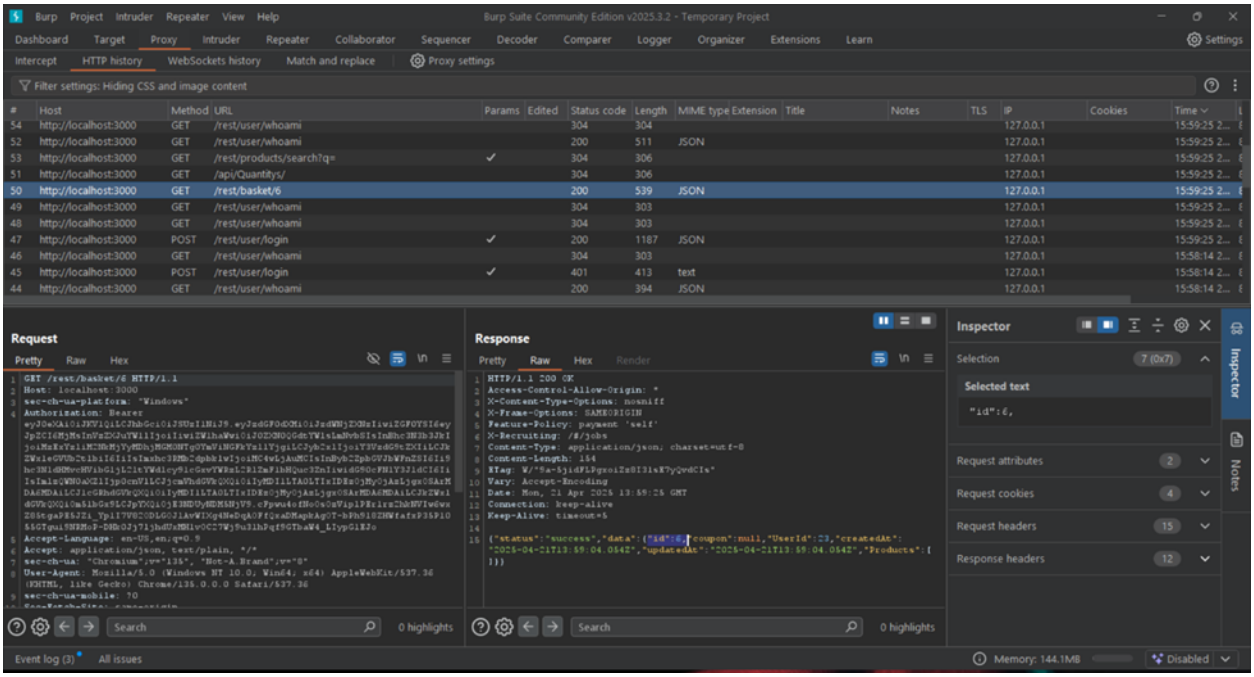


3. Open **Burp Suite** to intercept requests.

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

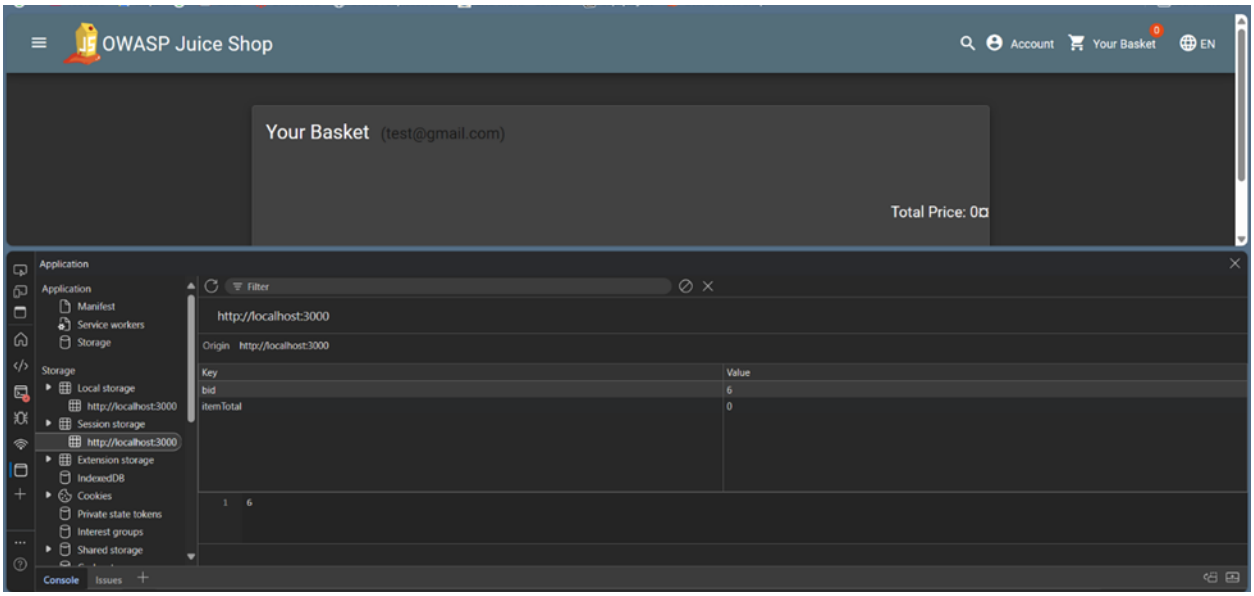
GET /rest/basket/1

where 1 is your current basket ID = “6”.

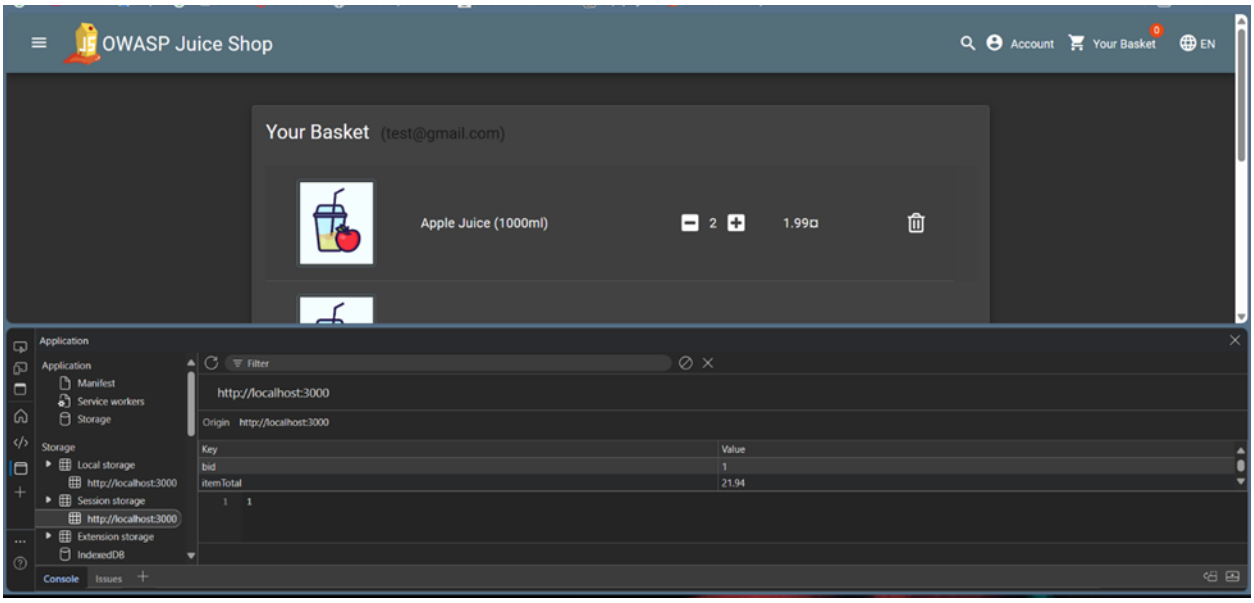


5. Open the **Developer Tools** in the browser and check session storage

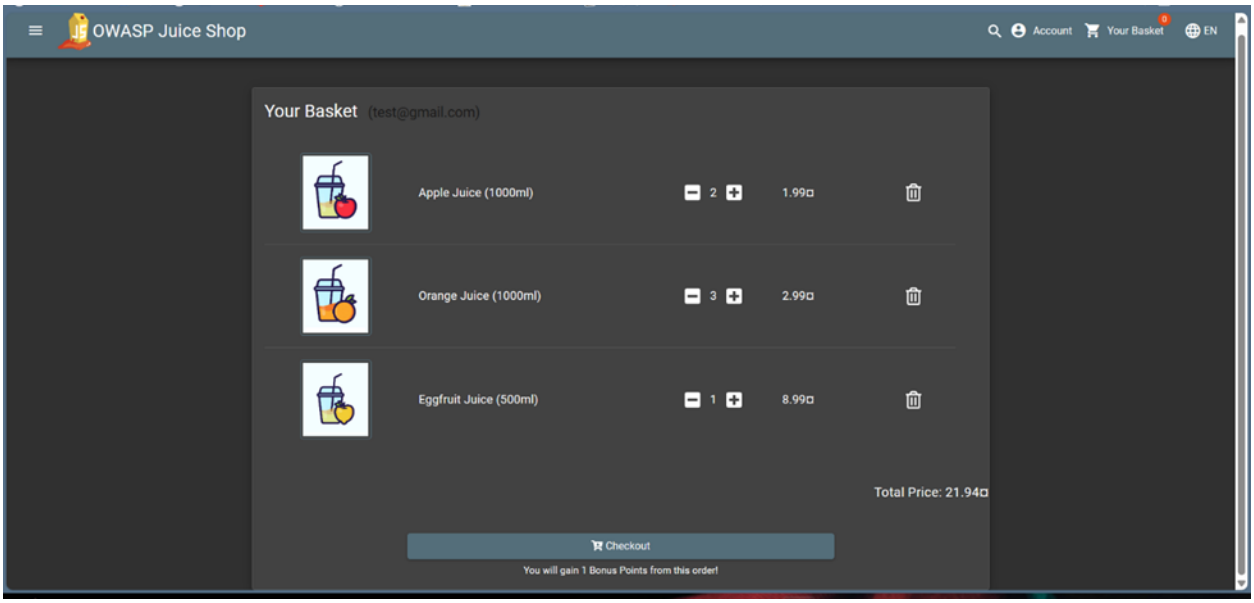
6. There [bid = “6”] that can change value to any number



6.1 Change the value to “1”



7. After change value the basket items changes



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

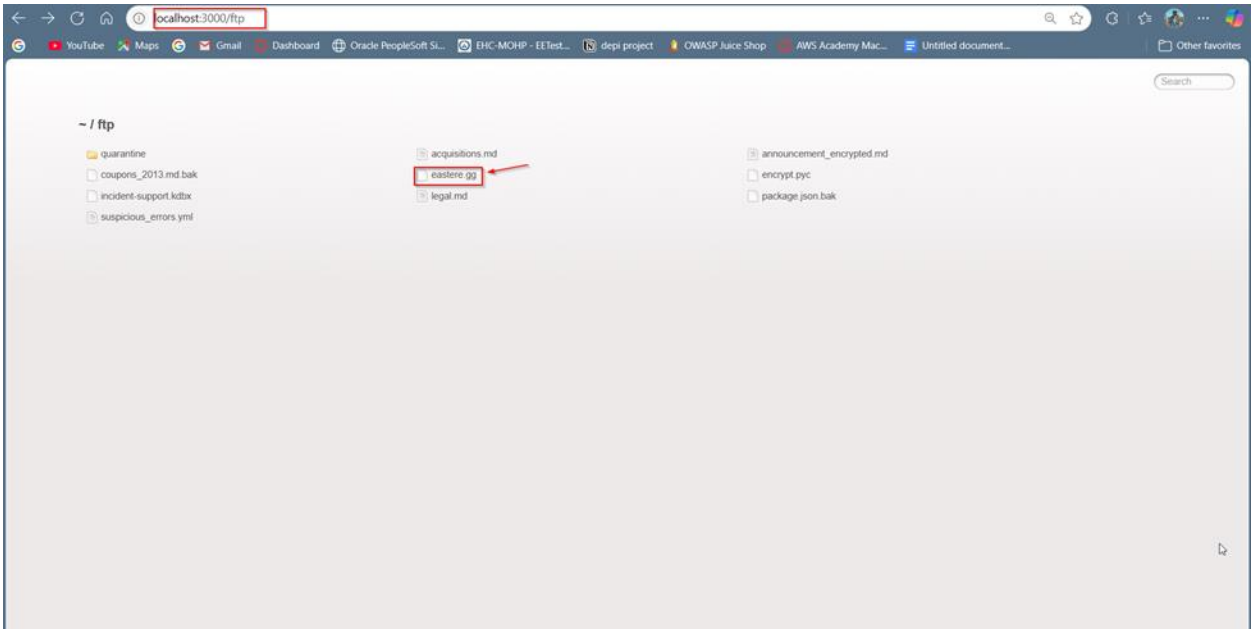
Recommendations:

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.

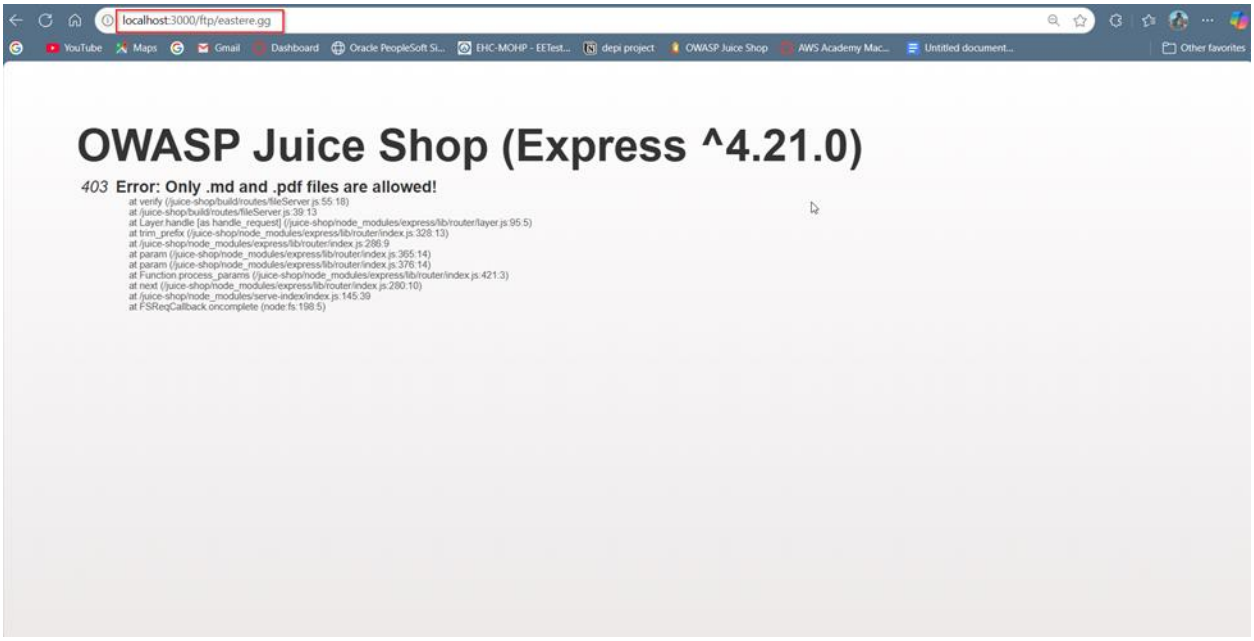
2. **Sanitize User Inputs:**
    - Properly sanitize and validate all user-supplied inputs to prevent injection attacks and unauthorized access.

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.

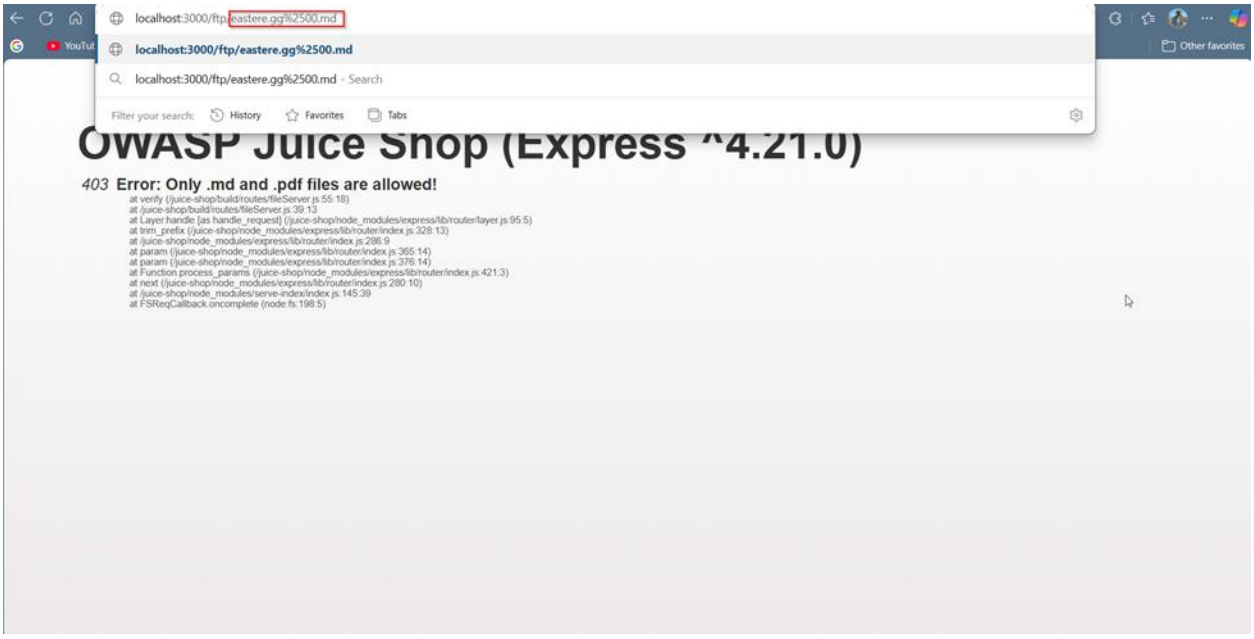


3. Attempt to access the file directly; the server should restrict this due to the .gg extension.

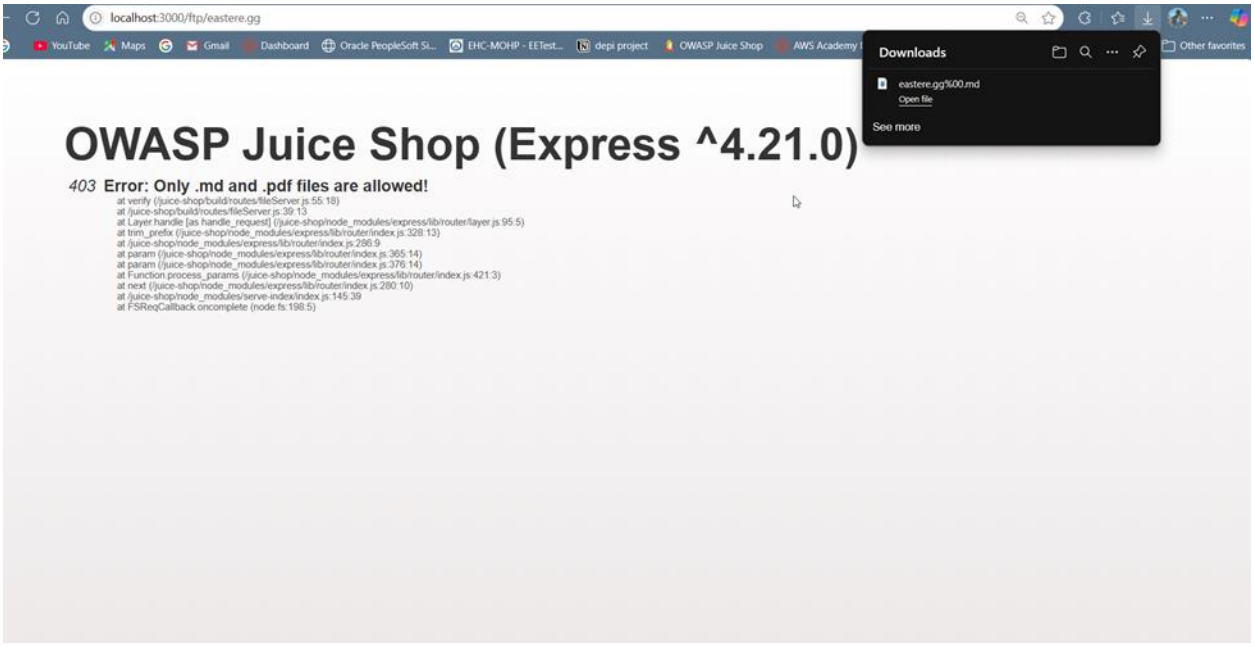


4. Modify the URL to bypass the restriction using null byte poisoning:

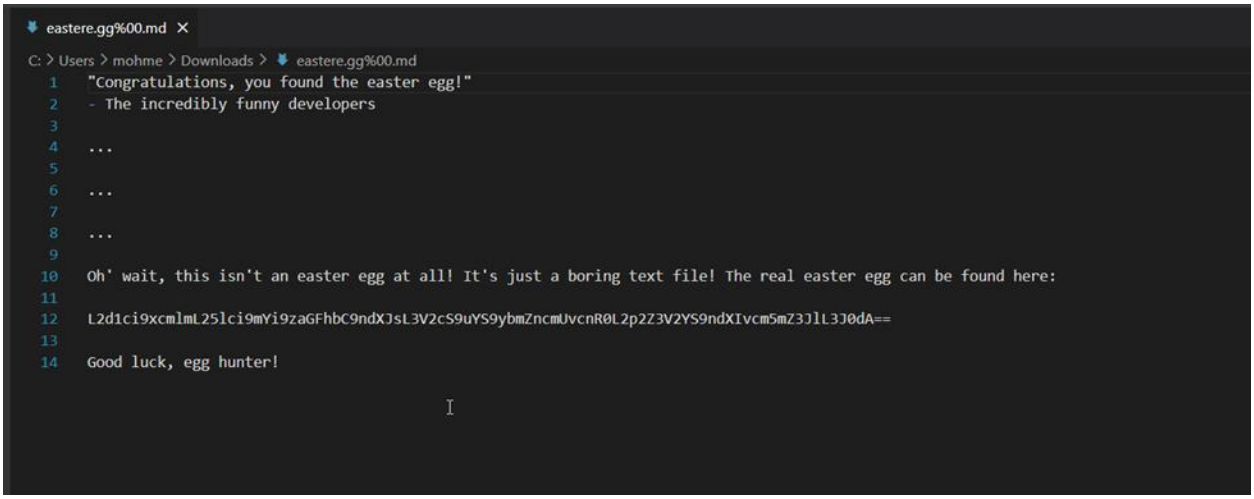
(http://127.0.0.1:3000/ftp/eastere.gg%2500.md)



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.



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

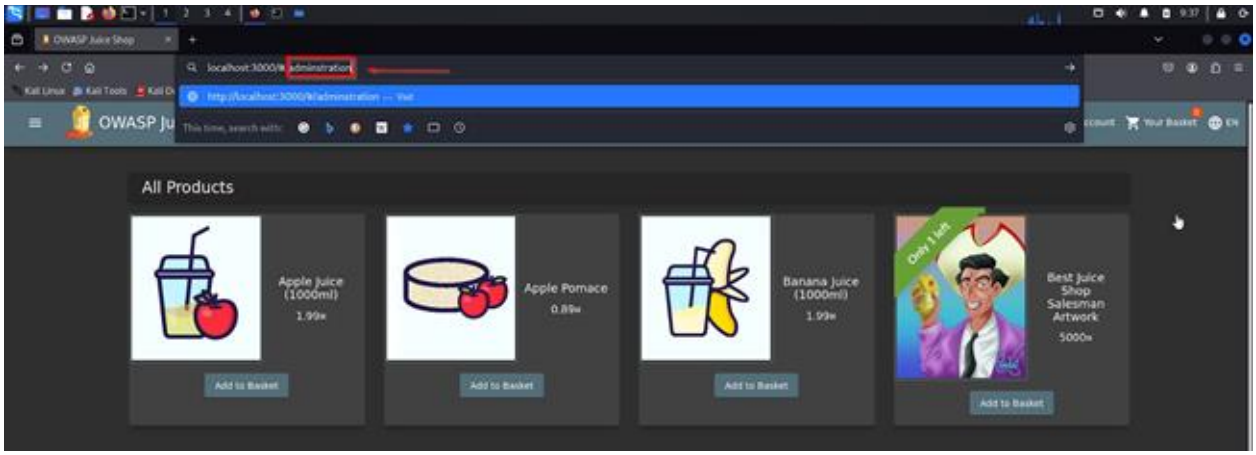
Recommendations:

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

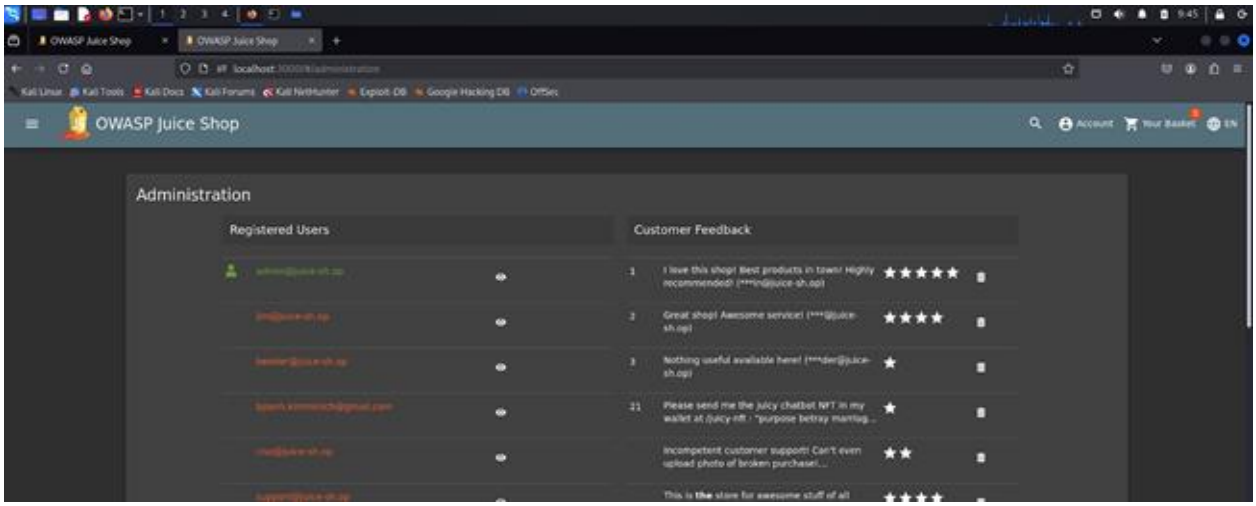
Resources : **OWASP IDOR Prevention Cheat Sheet**

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

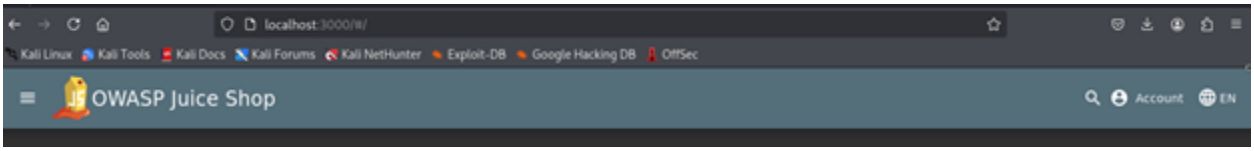
Resources: https://owasp.org/www-project-top-10-infrastructure-security-risks/docs/2023/INT08_2023-Information_Leakage

Recommendations: restricting access to important directories or files by

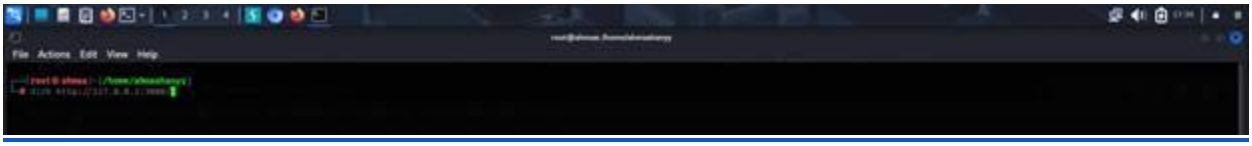
- Adopting a need-to-know requirement for both the document and server root
- Turning off features such as Automatic Directory Listings that could expose private files and provide information that could be utilized by an attacker when formulating or conducting an attack.

Proof of concept (POC):

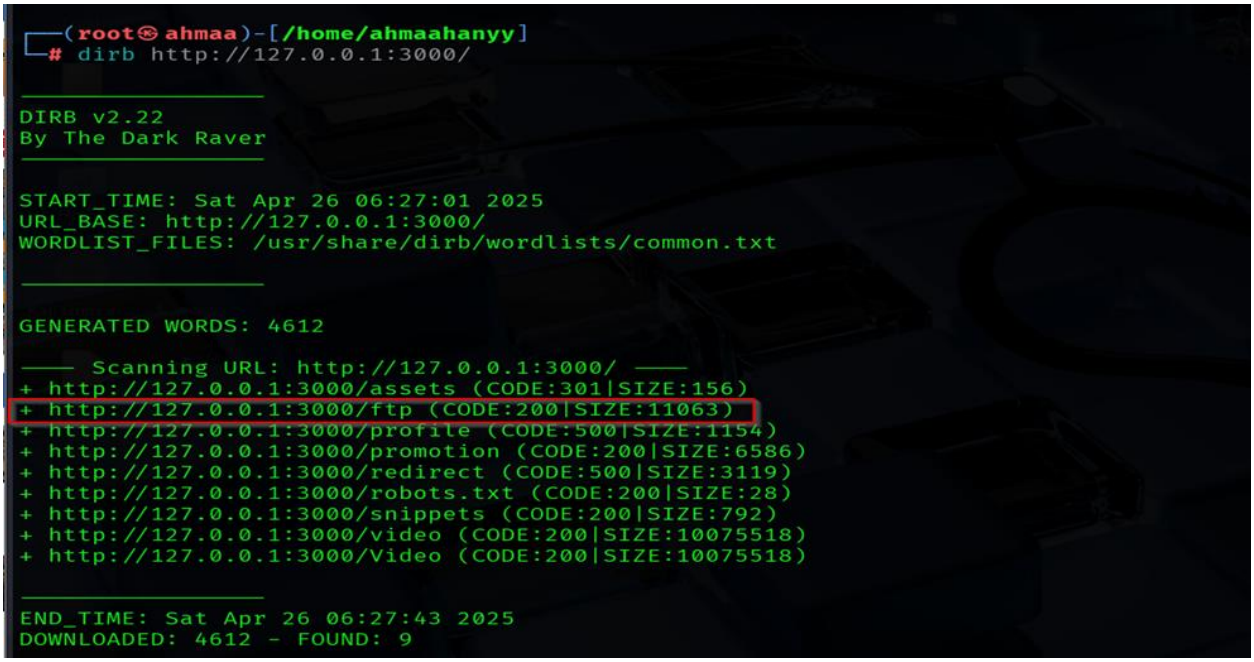
1. **website running on localhost (127.0.0.1) port (:3000)**



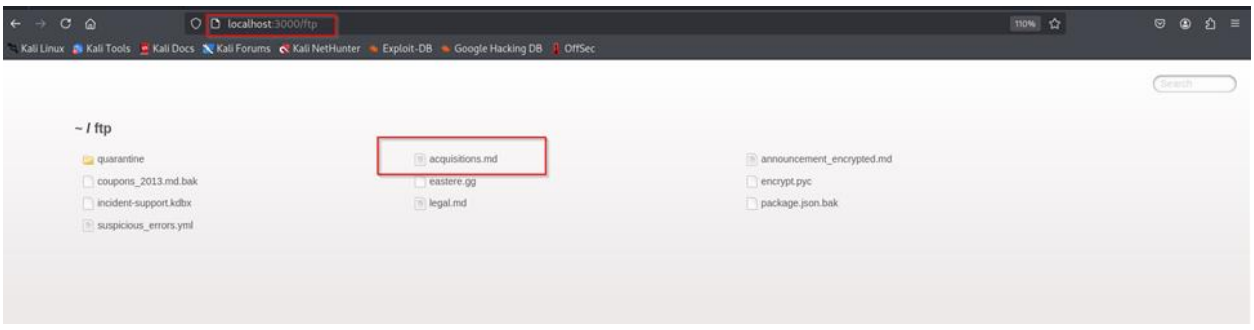
2. **Applying dirb command => dirb http://127.0.0.1:3000/**



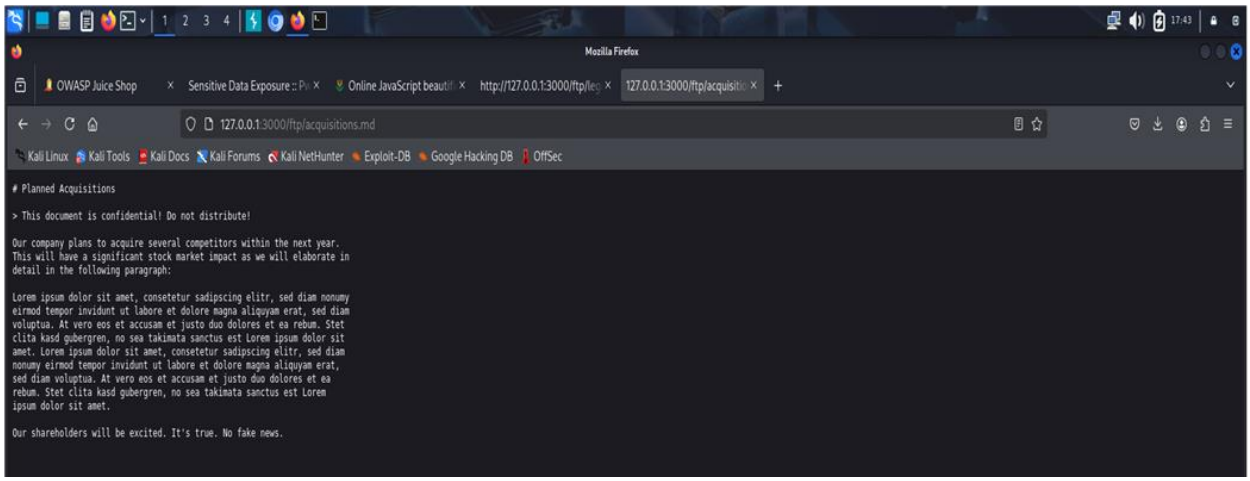
3. **Result contains a directory with \ftp.**

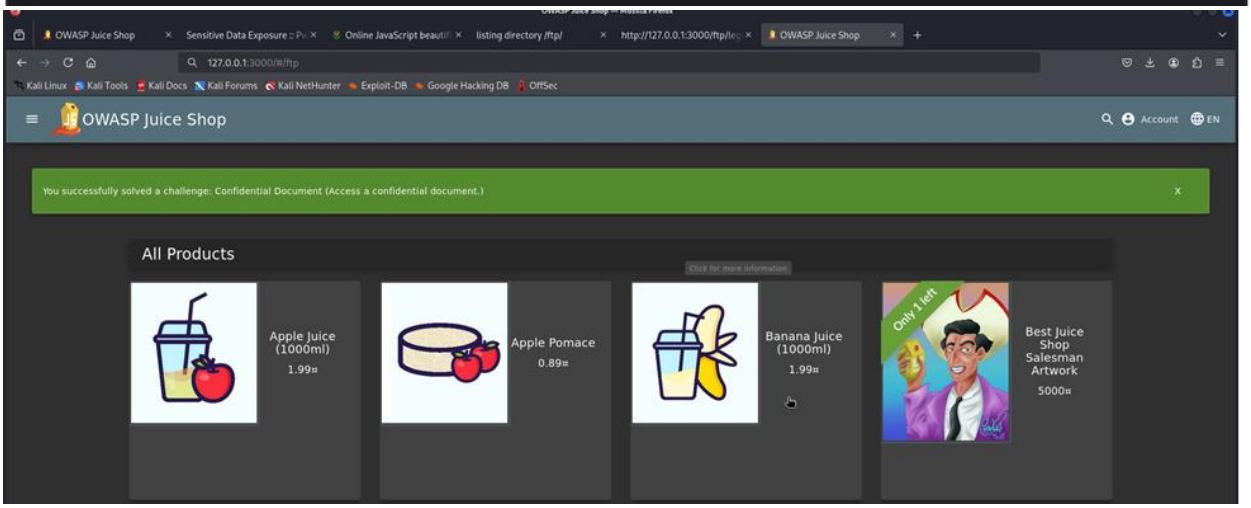


4. **Read (acquisitions.md).**



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





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

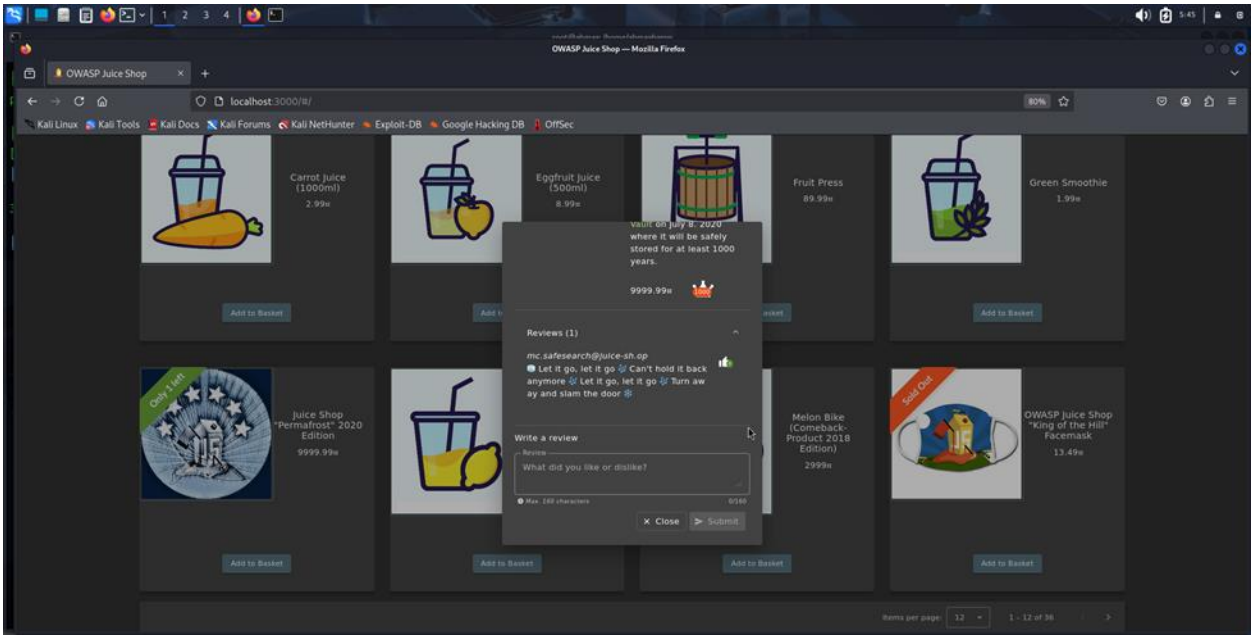
Resources: OWASP Top Ten 2017 | A3:2017-Sensitive Data Exposure | OWASP Foundation

Recommendations:

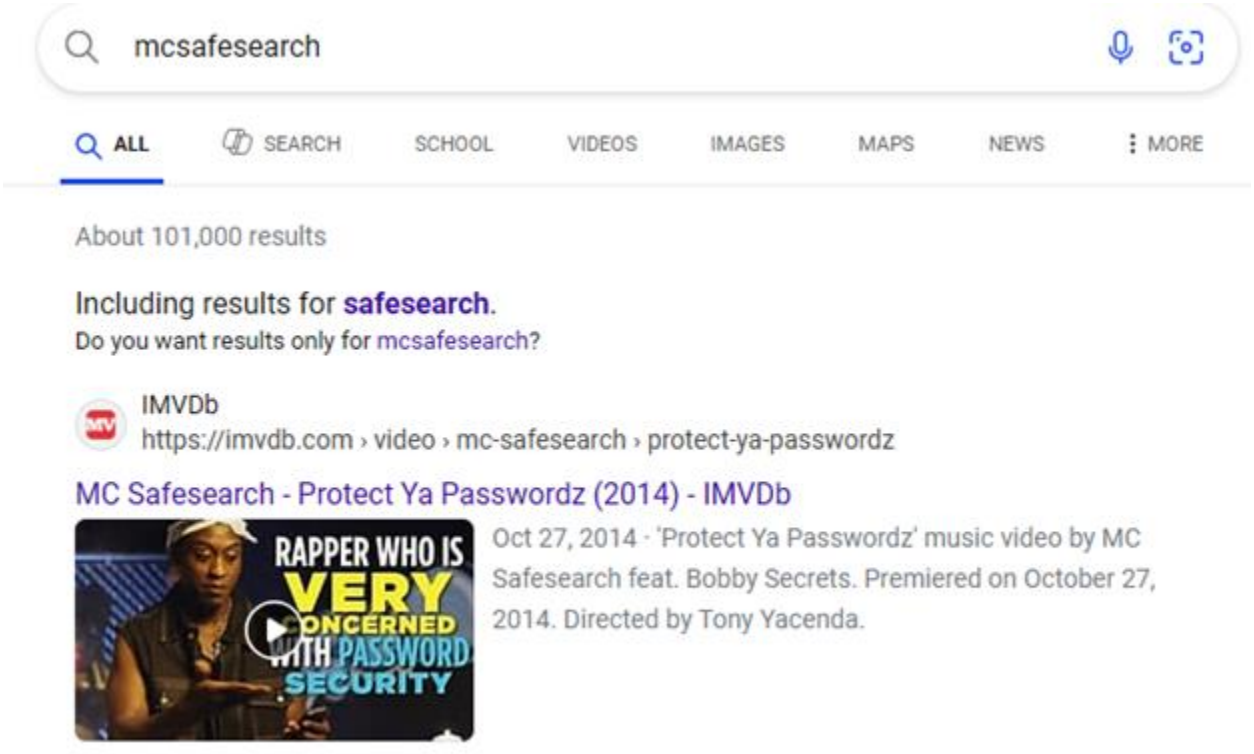
Educate users to **avoid using personal information as passwords**, and enforce strong password policies by applying restrictions at the registration level to make passwords harder to guess.

Proof of Concept (POC):

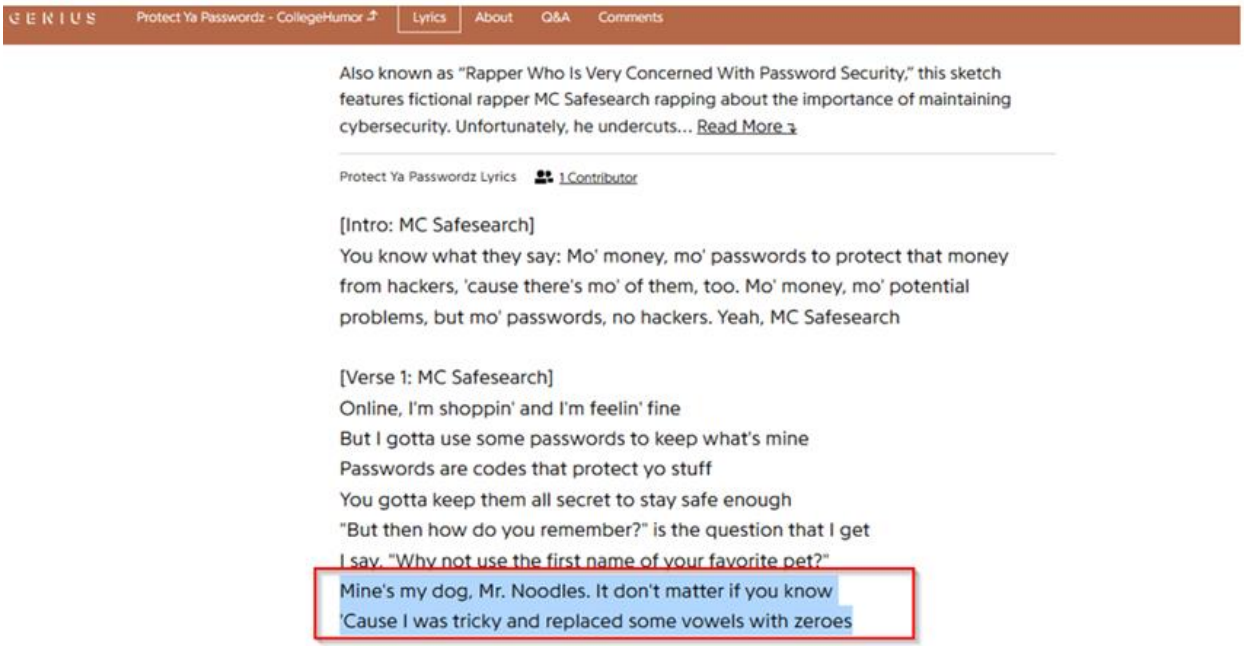
1. **Found a username in a review posted on a product**



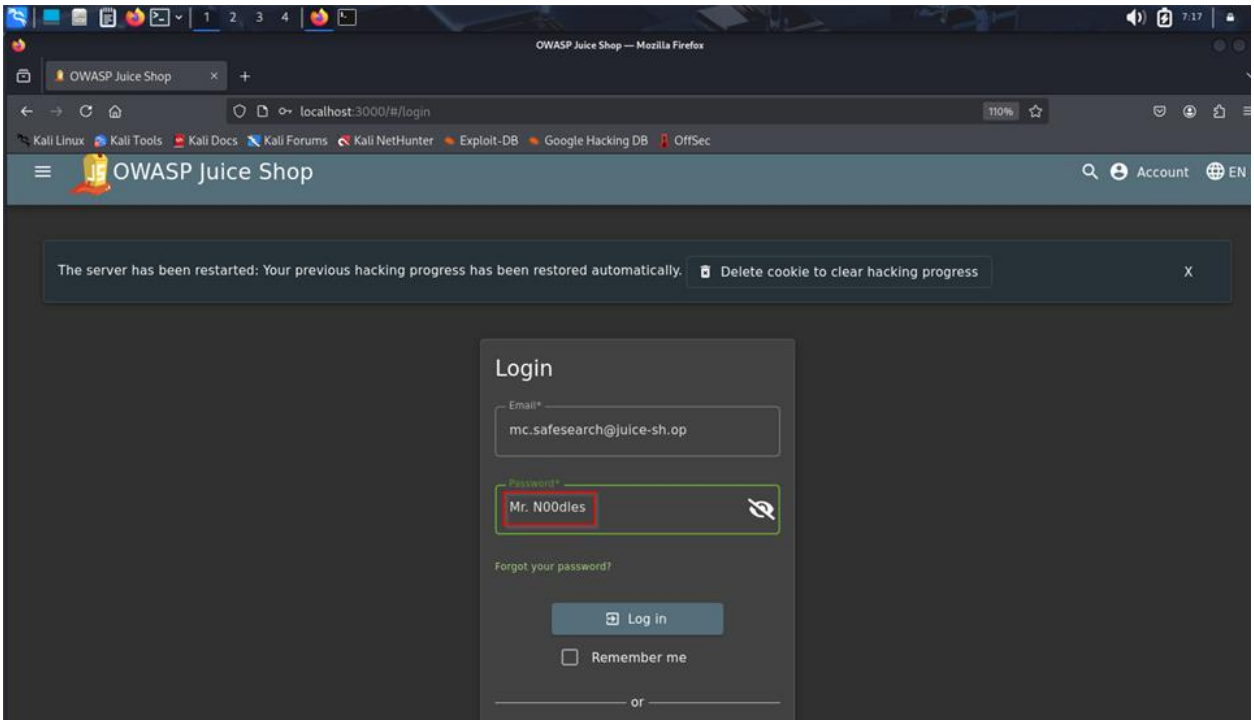
2. Search for this person's song

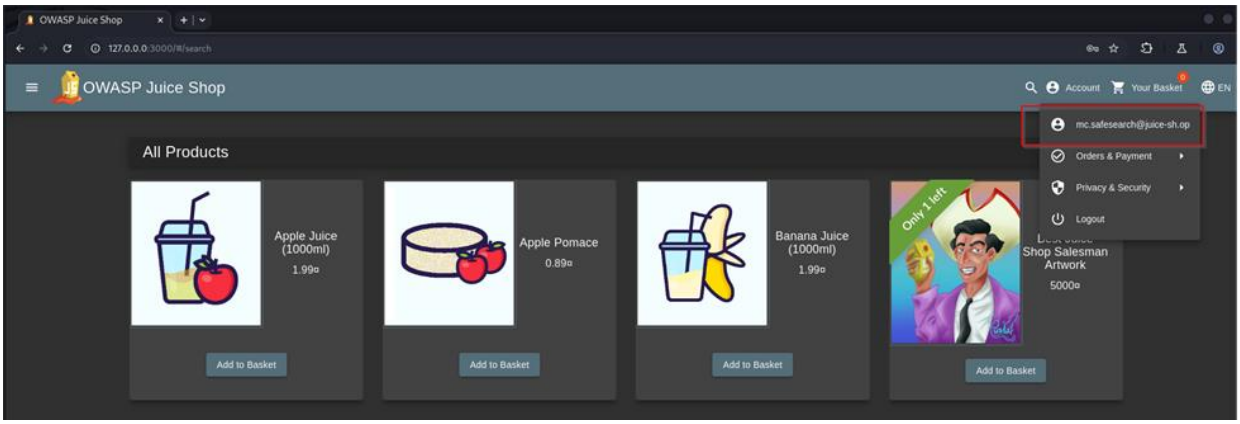


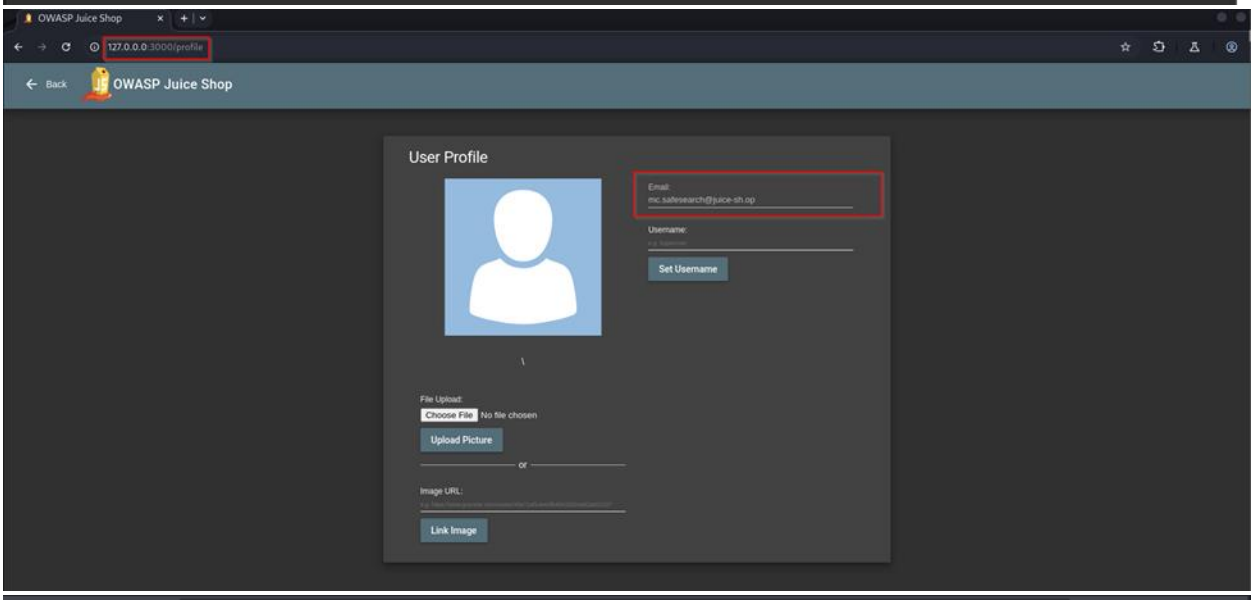
3. Analyzing the song's lyrics and found Password hints

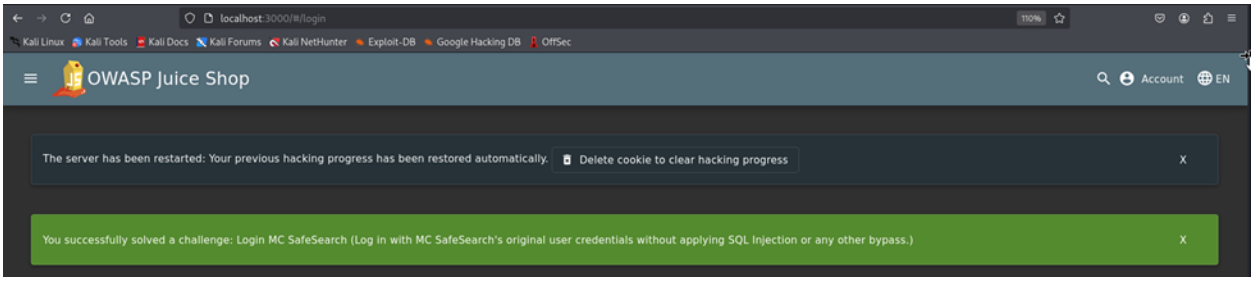


4. Apply these hints and successfully log in as (mc.safesearch@juice-sh.op)




OWASP Juice Shop
127.0.0.0:3000/#/search
OWASP Juice Shop
Account
Your Basket
EN
mc.safesearch@juice-sh.op
Orders & Payment
Privacy & Security
Logout
All Products
Apple Juice (1000ml)
1.99¤
Add to Basket
Apple Pomace
0.89¤
Add to Basket
Banana Juice (1000ml)
1.99¤
Add to Basket
Only 1 left
Shop Salesman Artwork
5000¤
Add to Basket


OWASP Juice Shop
127.0.0.0:3000/profile
Back
OWASP Juice Shop
User Profile
Email:
mc.safesearch@juice-sh.op
Username:
Set Username
File Upload:
Choose File
No file chosen
Upload Picture
or
Image URL:
Link Image


localhost:3000/#/login
110%
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
OWASP Juice Shop
Account
EN
The server has been restarted: Your previous hacking progress has been restored automatically.
Delete cookie to clear hacking progress
X
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.)
X

6.3 Disclosure of Node.js Files

Medium

Description
Application reveals sensitive info backend files.

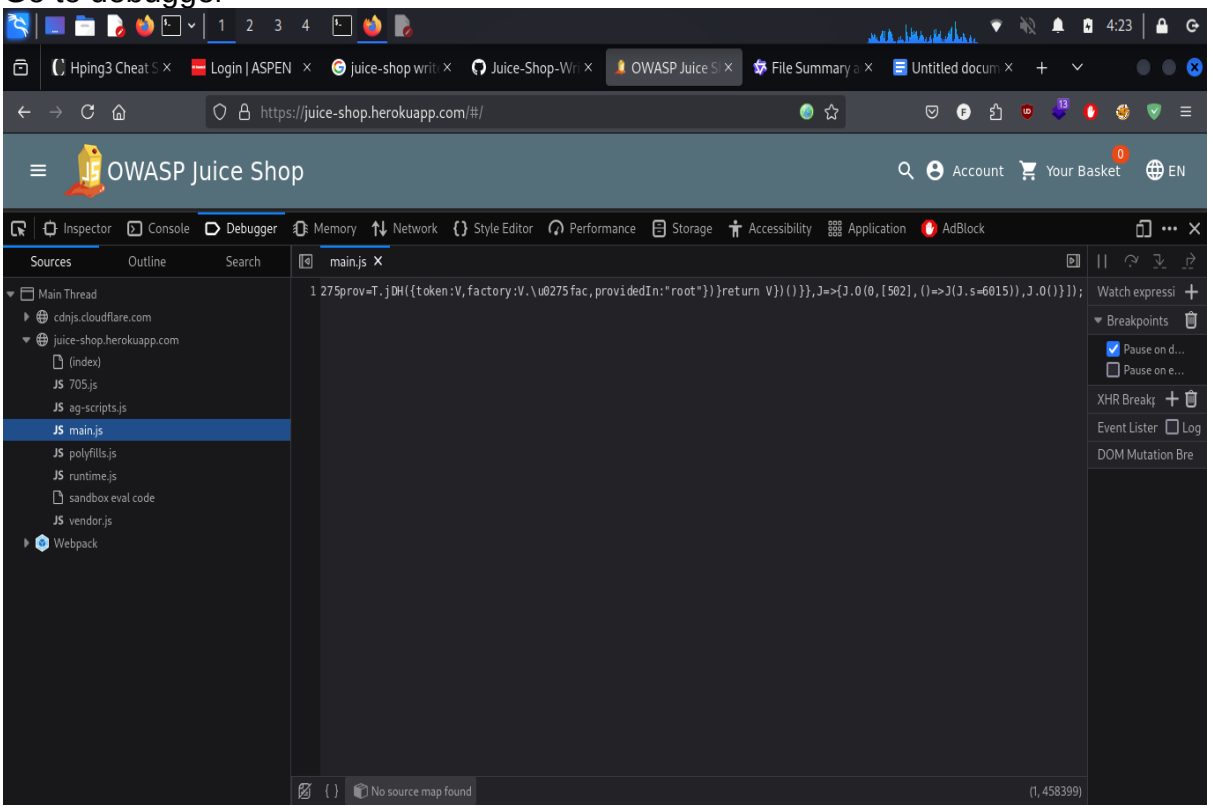
Impact
Could help attackers analysis the code and get critical information for routes and function used

Recommendation

1. Remove sensitive files to be viewed

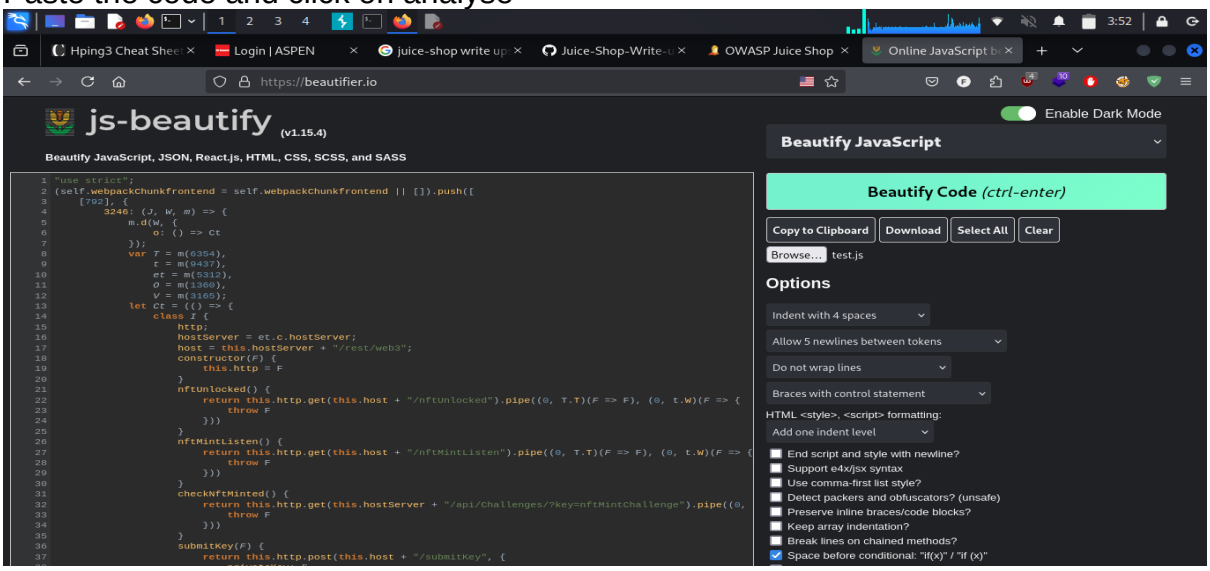
Proof of Concept

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

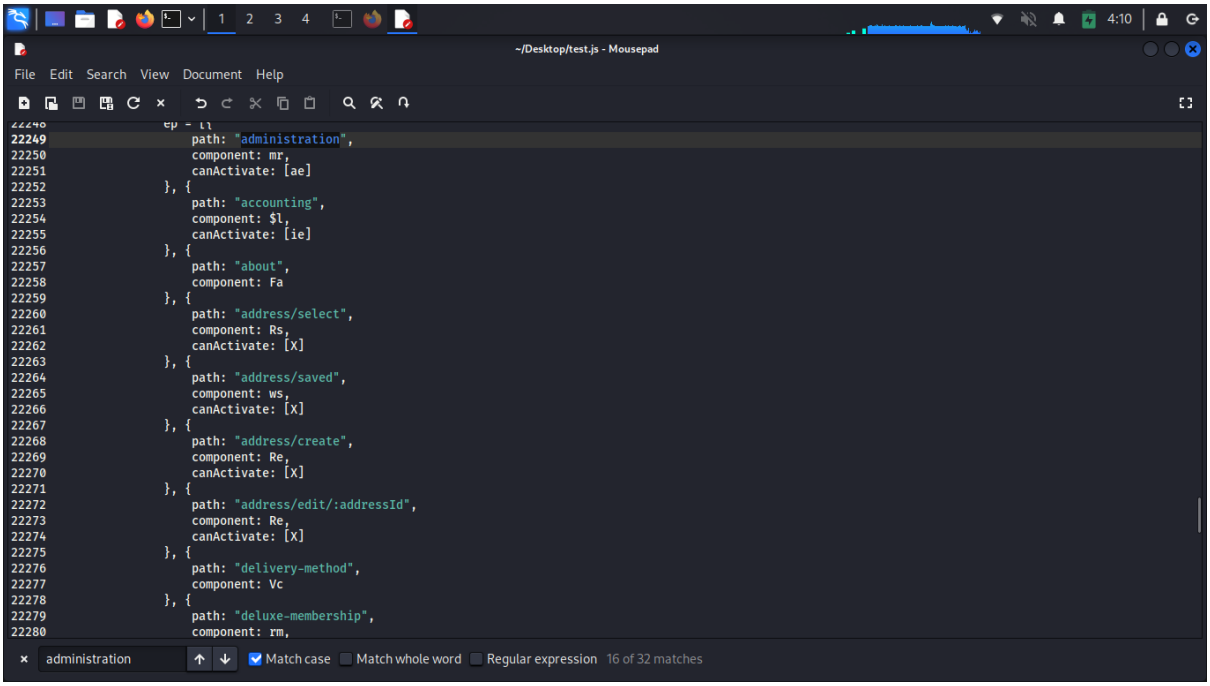


4. Take the code from main.js
5. Go to https://beautifier.io/

6. Paste the code and click on analyse



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



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

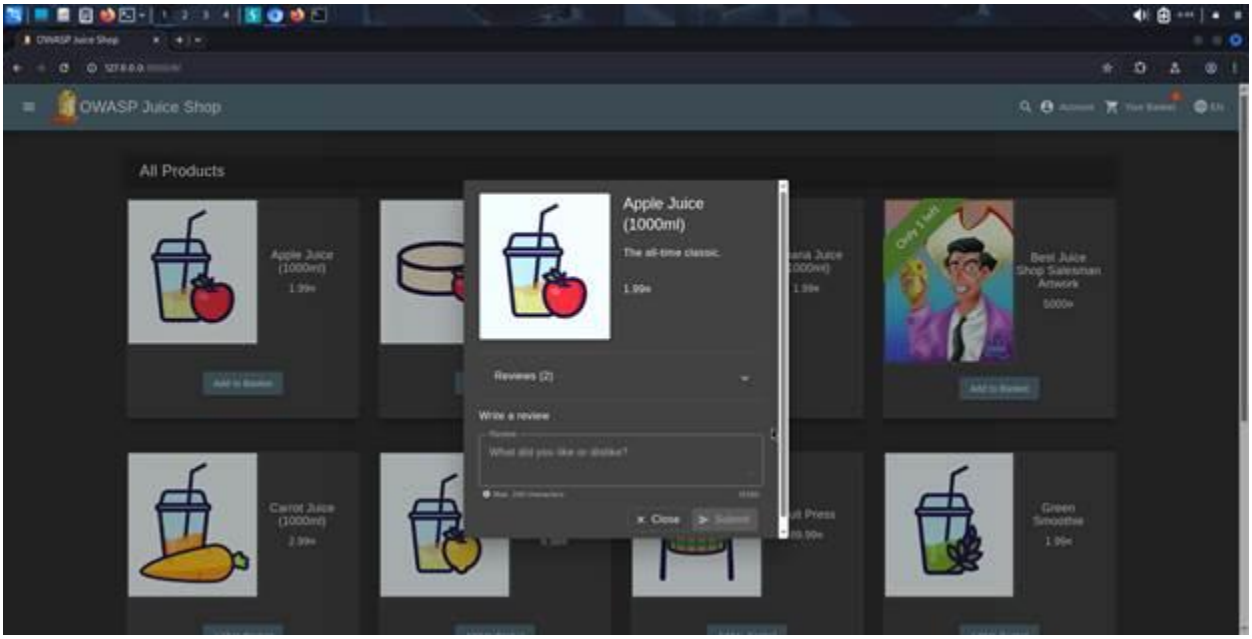
Recommendations:

· Customize error handling to return generic error messages without exposing stack traces or file paths.

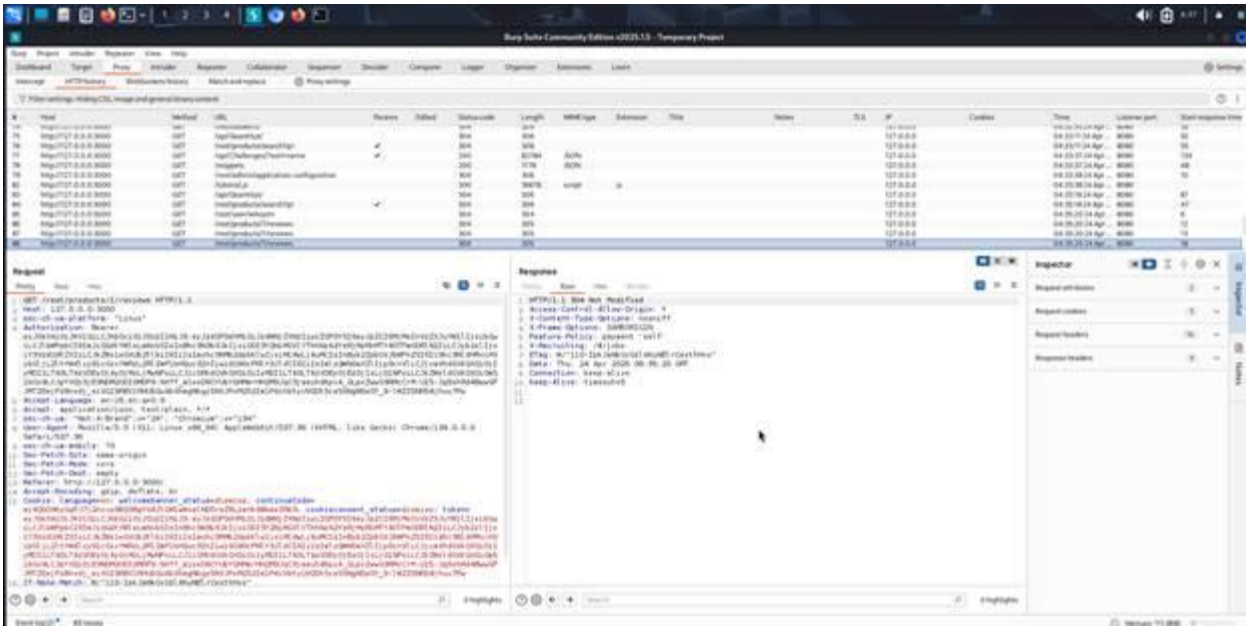
· Review logs internally without exposing details to users.

Proof of concept(POC):

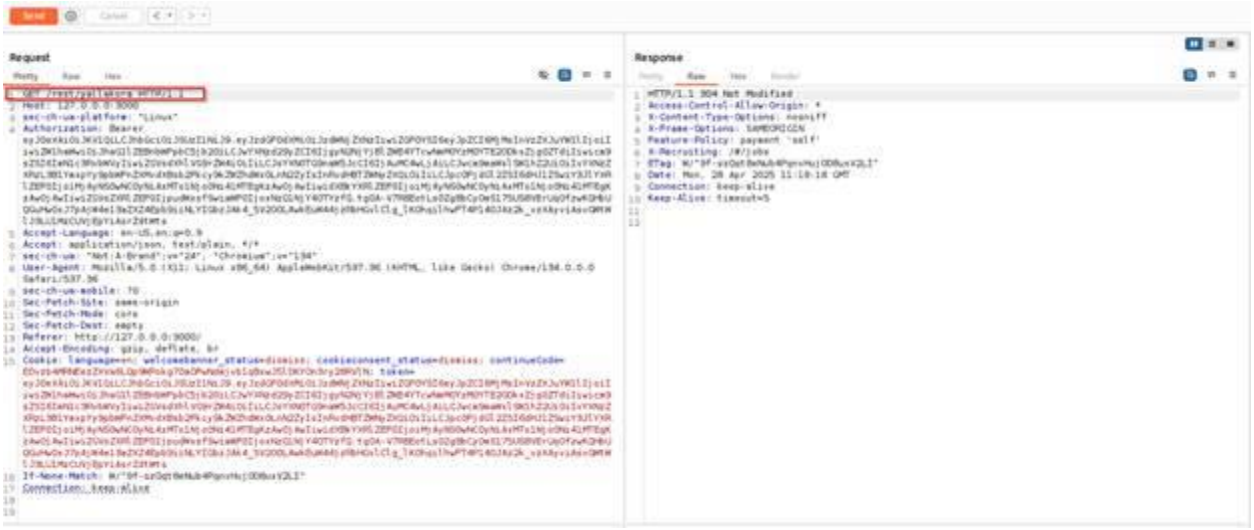
1. **Open any Product to see Reviews**



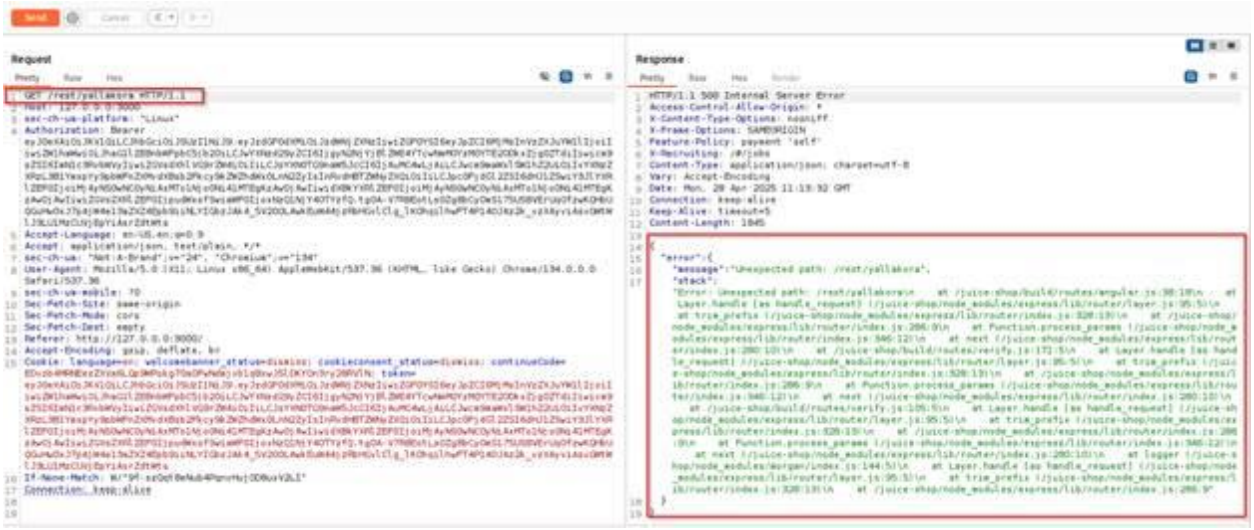
2. Intercept the packet and send it to the repeater

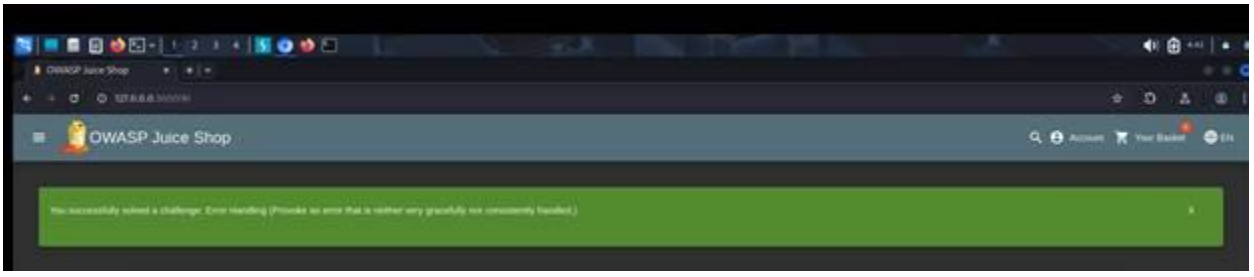


3. **Modify the request line to any external website that can cause an error and try to see the response**.



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





6.5 Admin Email

Informational

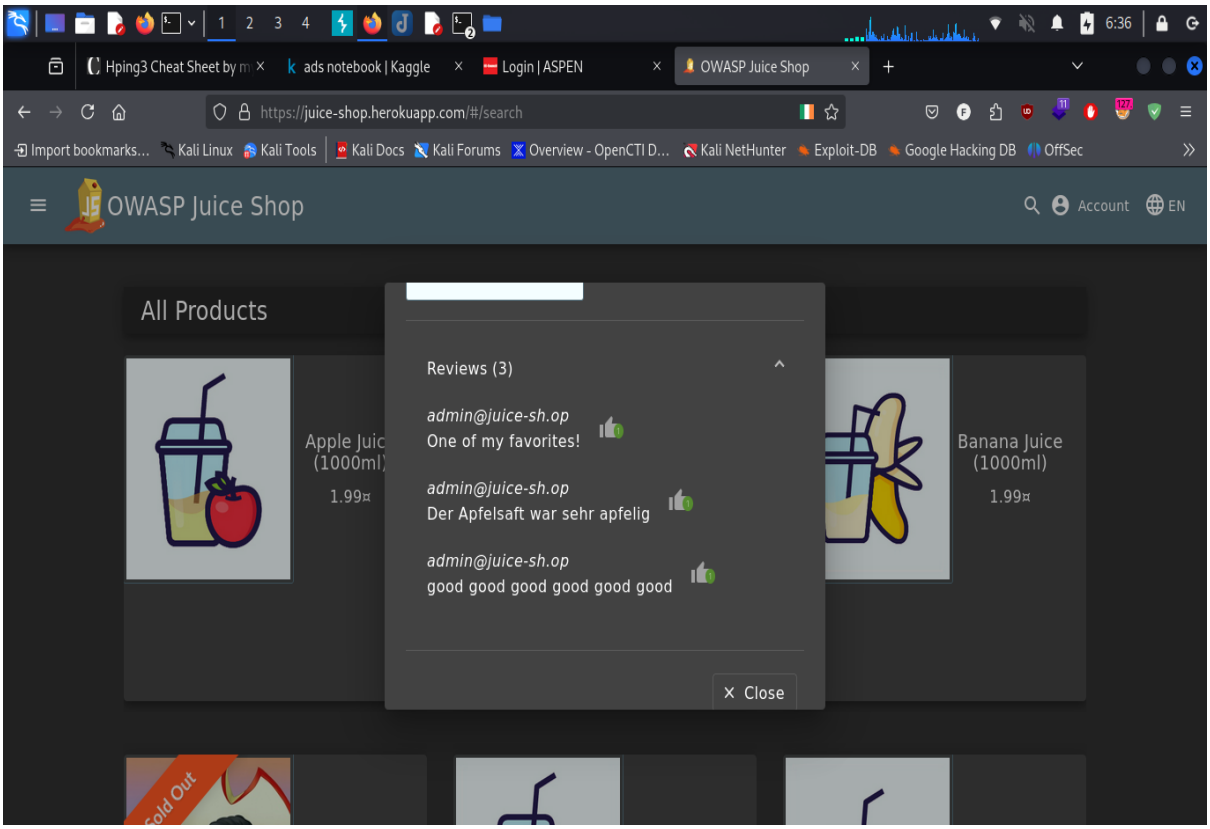
Location: Product info

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.

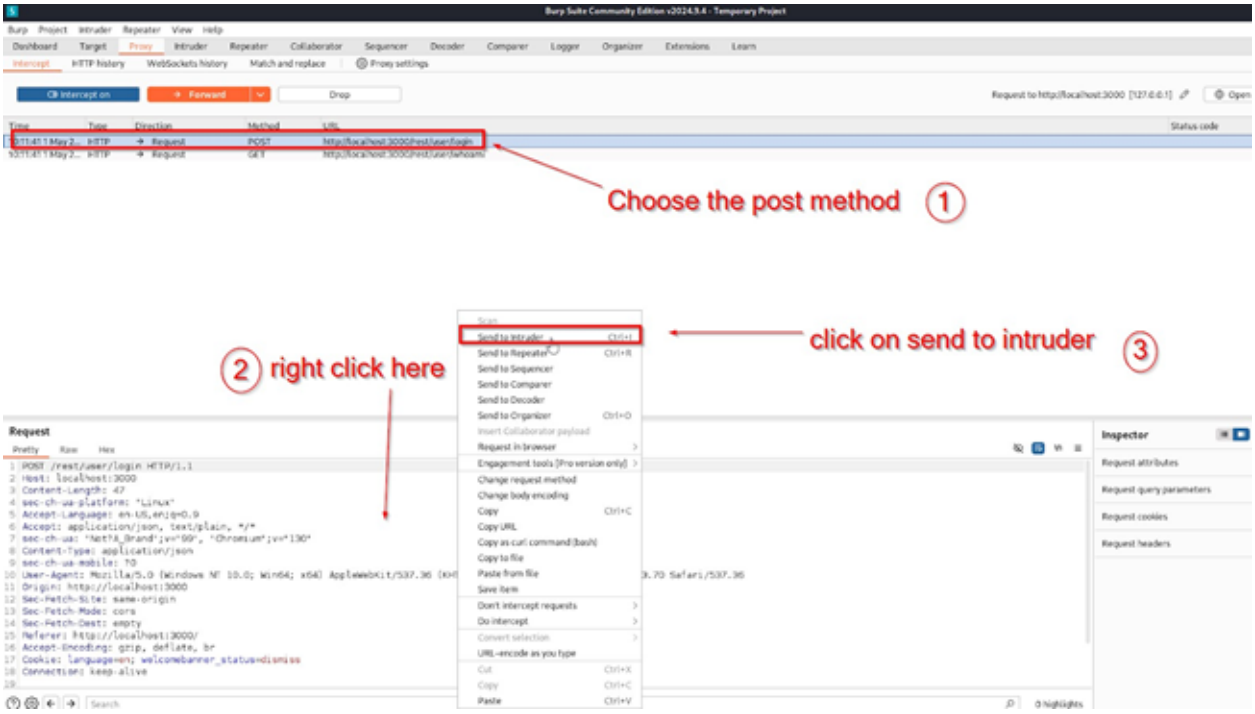
Proof Of Concept (P.O.C)

1. Open burp suite and open browser you will enter credentials in later

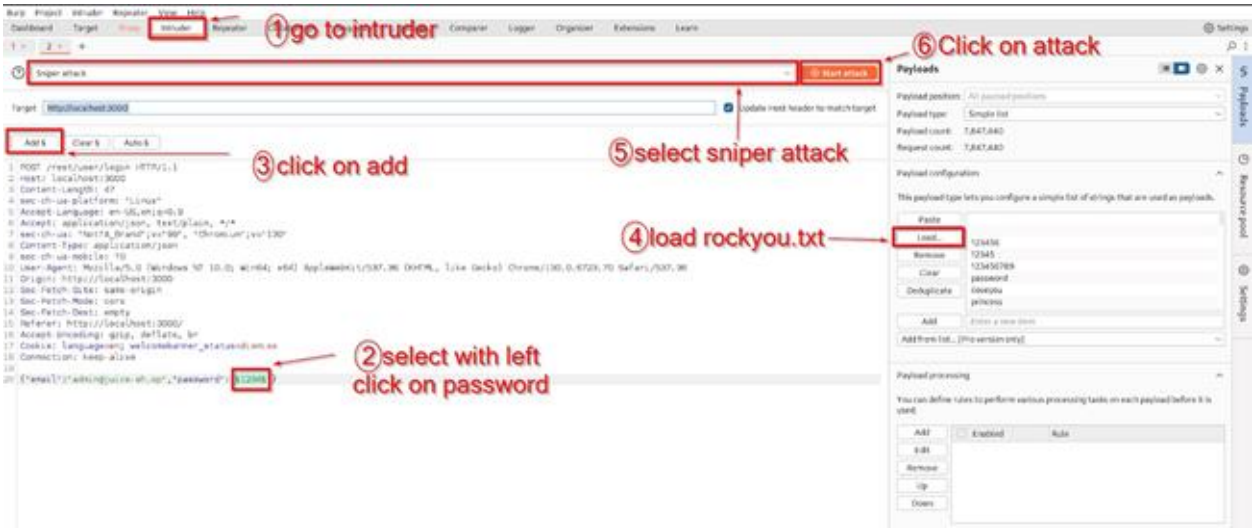
2. Go to Burpsuite proxy then turn intercept on

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

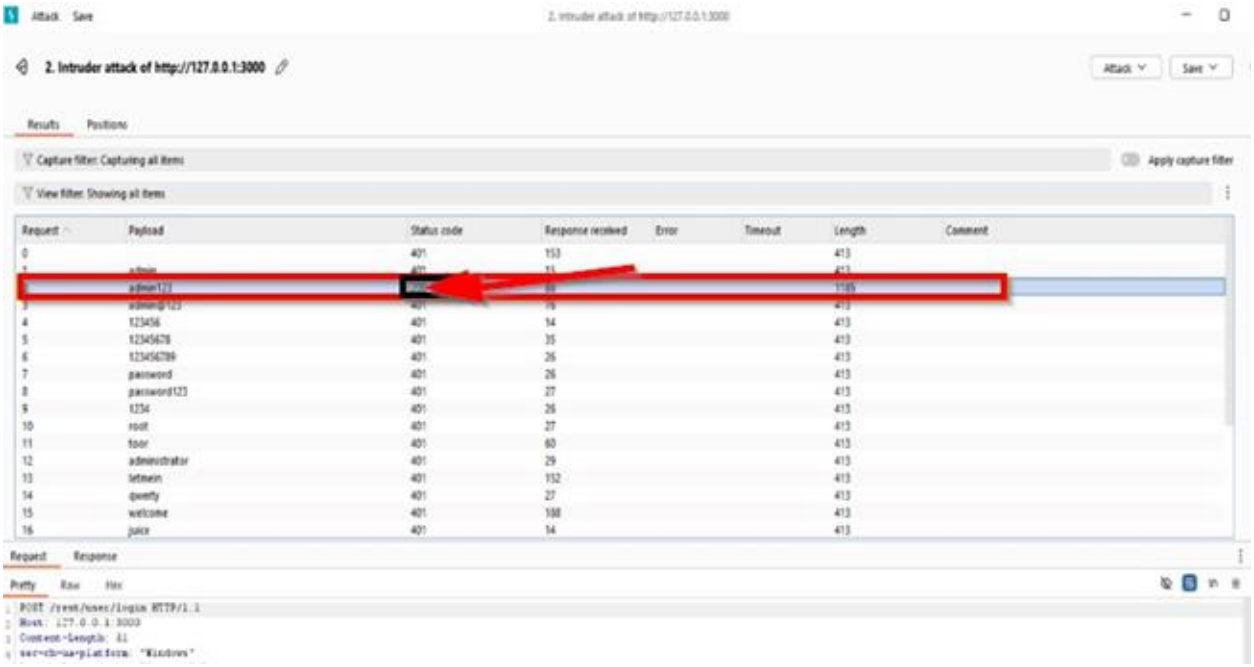
4. Do as shown below



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



6. Now look for status code 200 then take this 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—without 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)

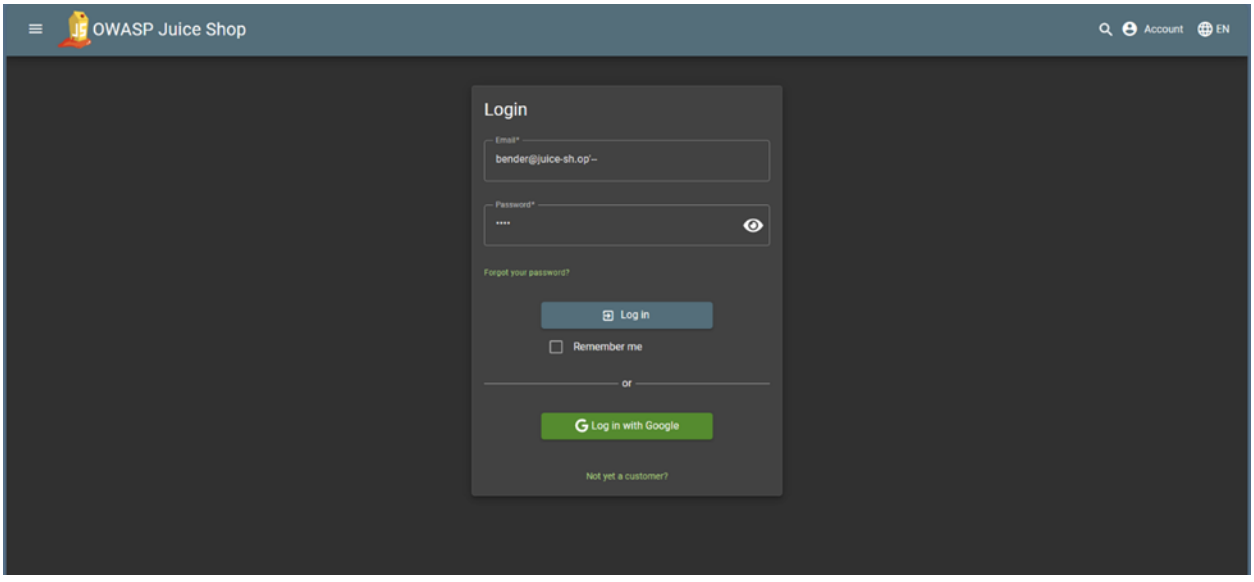
Recommendations:

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):

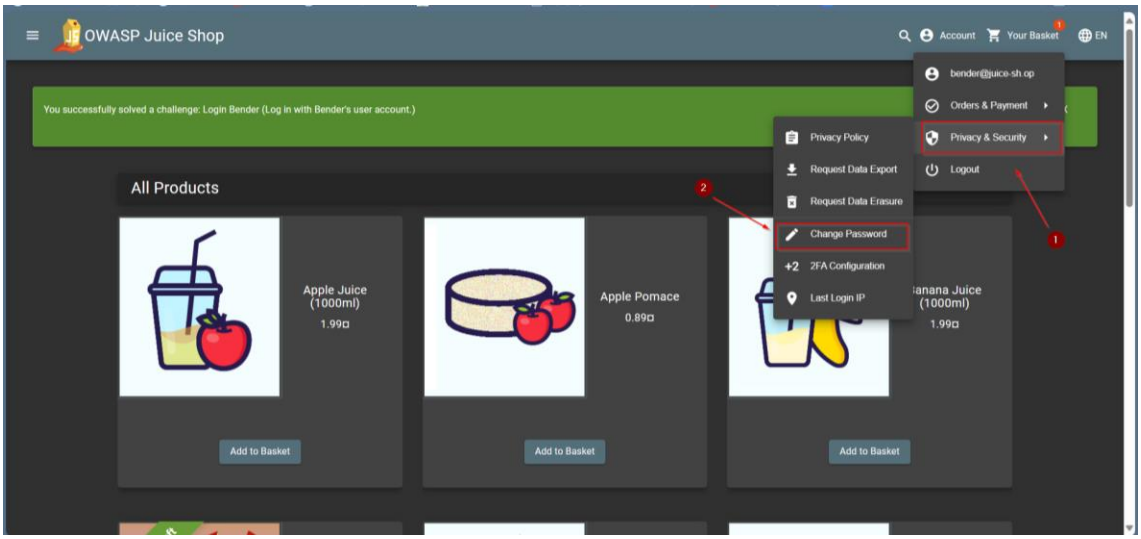
1. **Login using Bender's email (bender@juice-sh.op).**

    1.1 We broke the login form to bypass (**bender@juice-sh.op'--**).



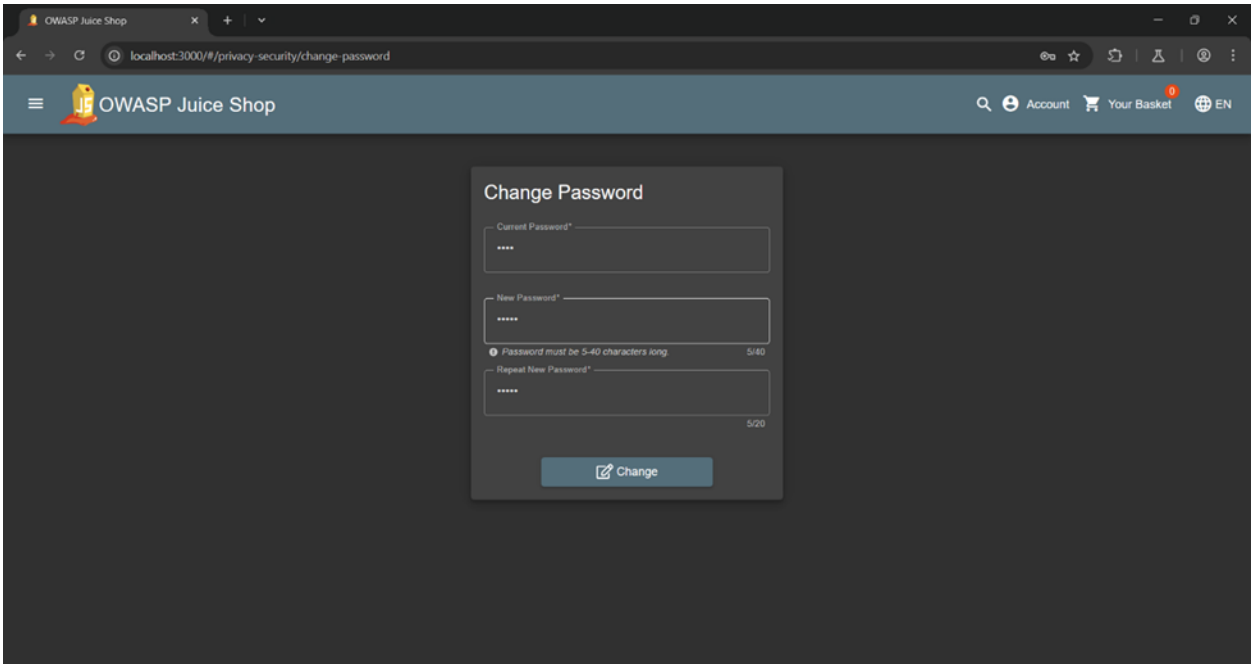
2. **Access the password change functionality:**

2.1 Navigate to the password change form within the application.



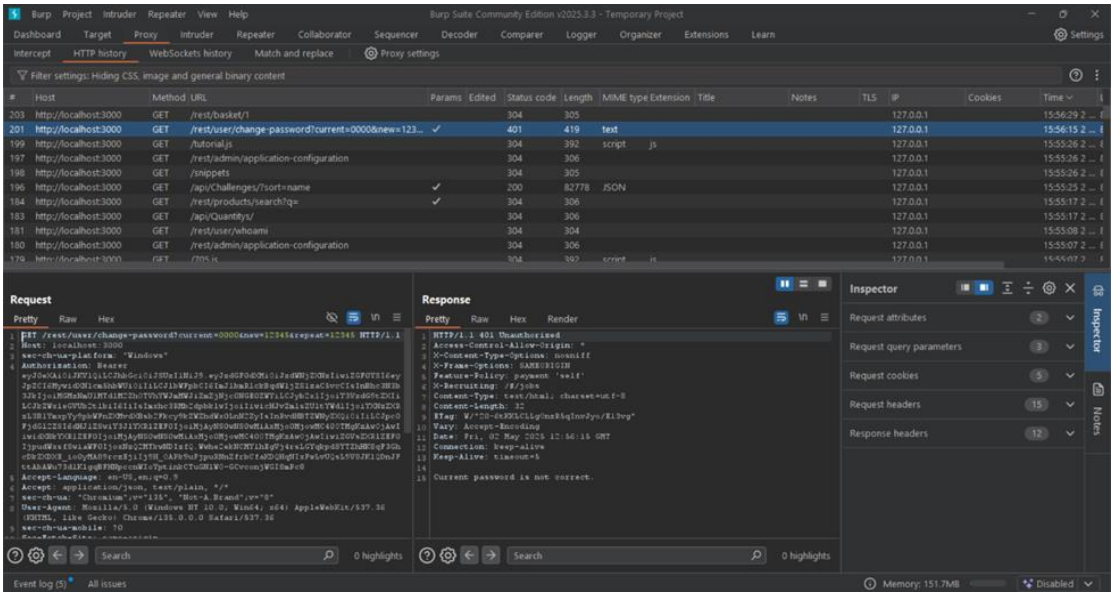
3. **Fill the form:**

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



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

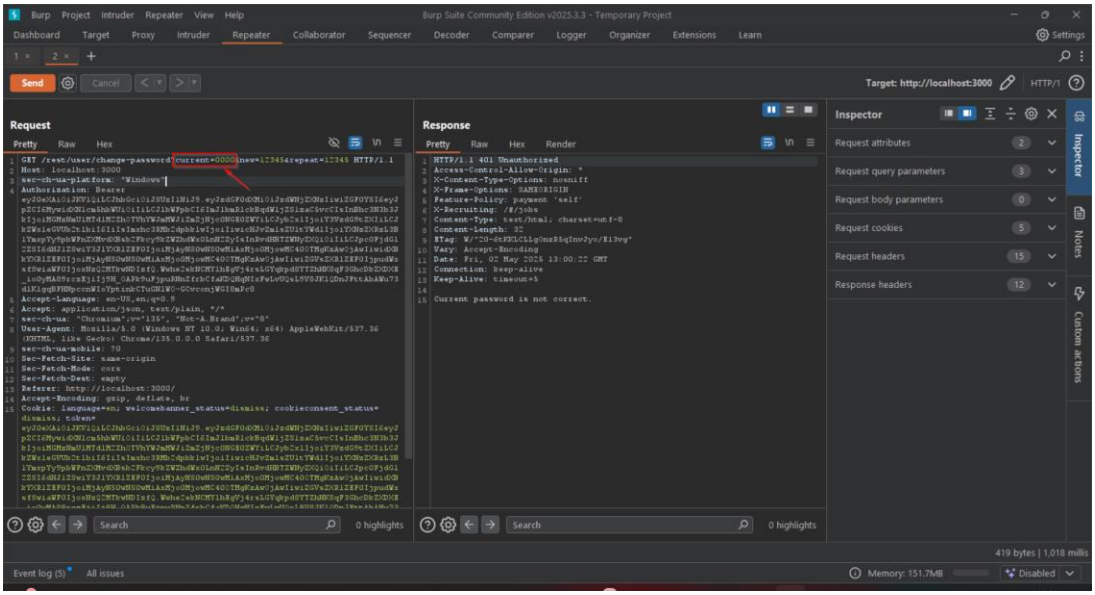
4.1 Capture the HTTP request sent when attempting to change the password and send to repeter.



5. **Modify the request:**

5.1 Remove the current password field.

5.2 Forward the altered request to the server.

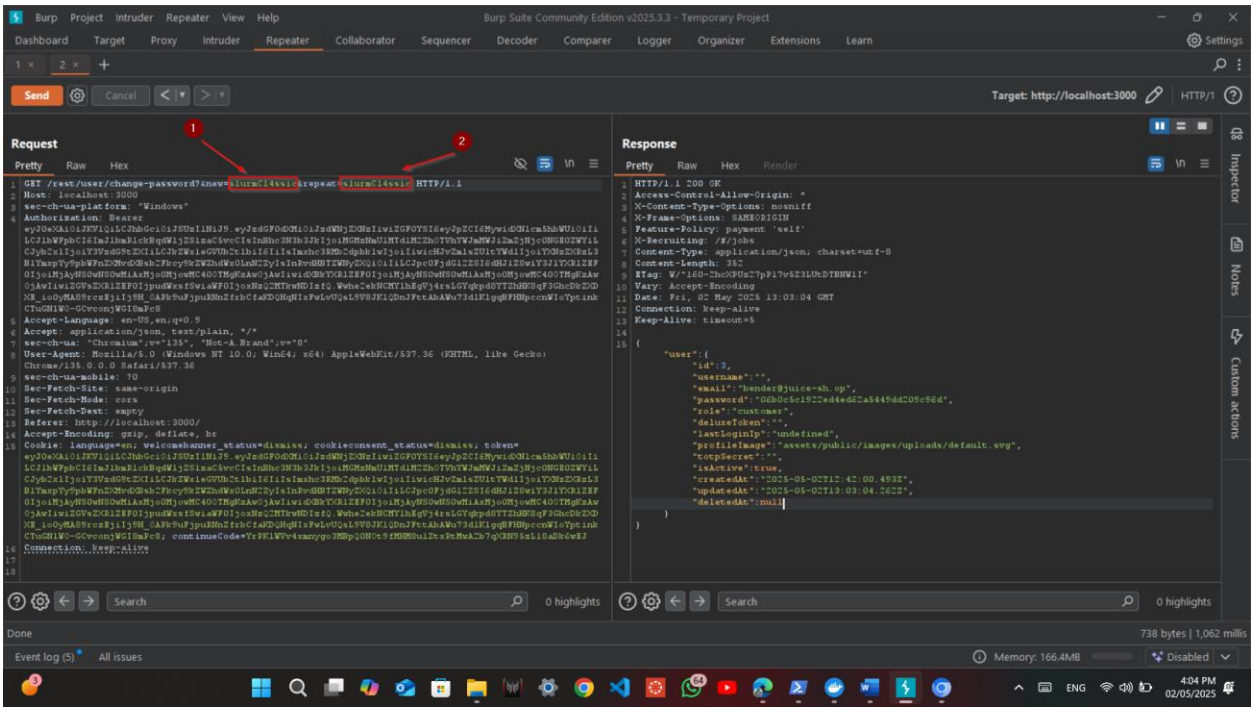


6 **Verify the password change:**

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

- 🔗 OWASP JWT Cheat Sheet
- 🔗 Auth0: Critical Vulnerabilities in JSON Web Token Libraries

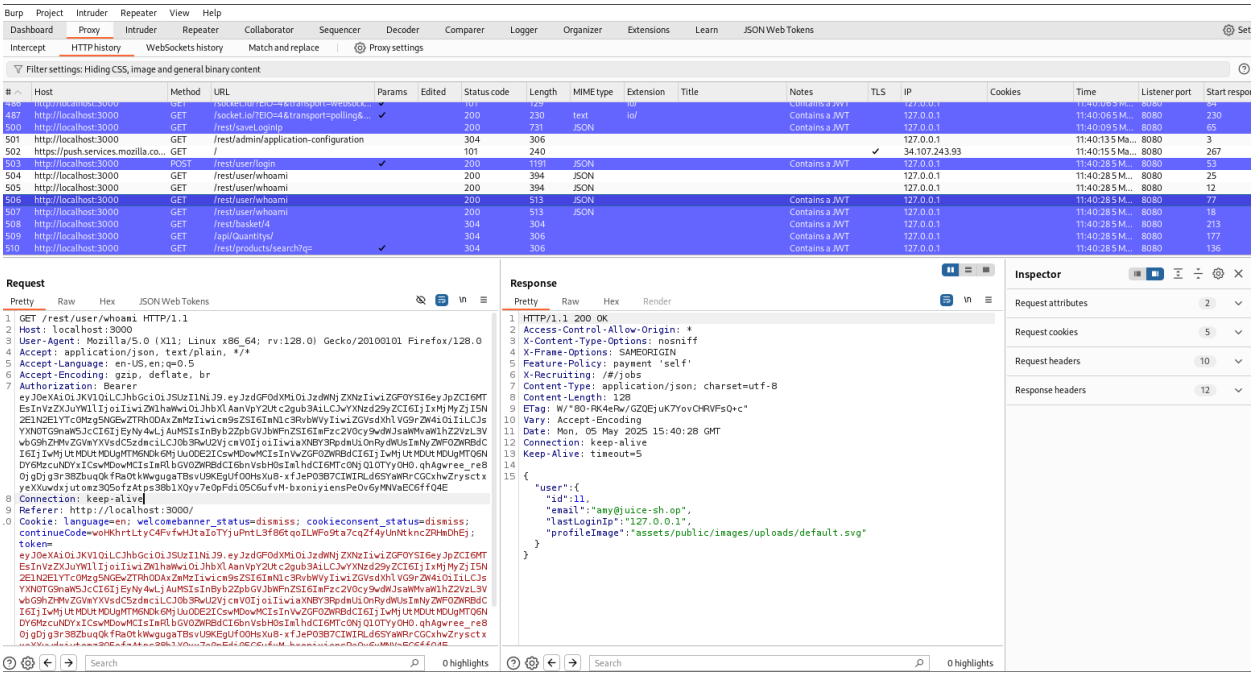
---

Recommendations:

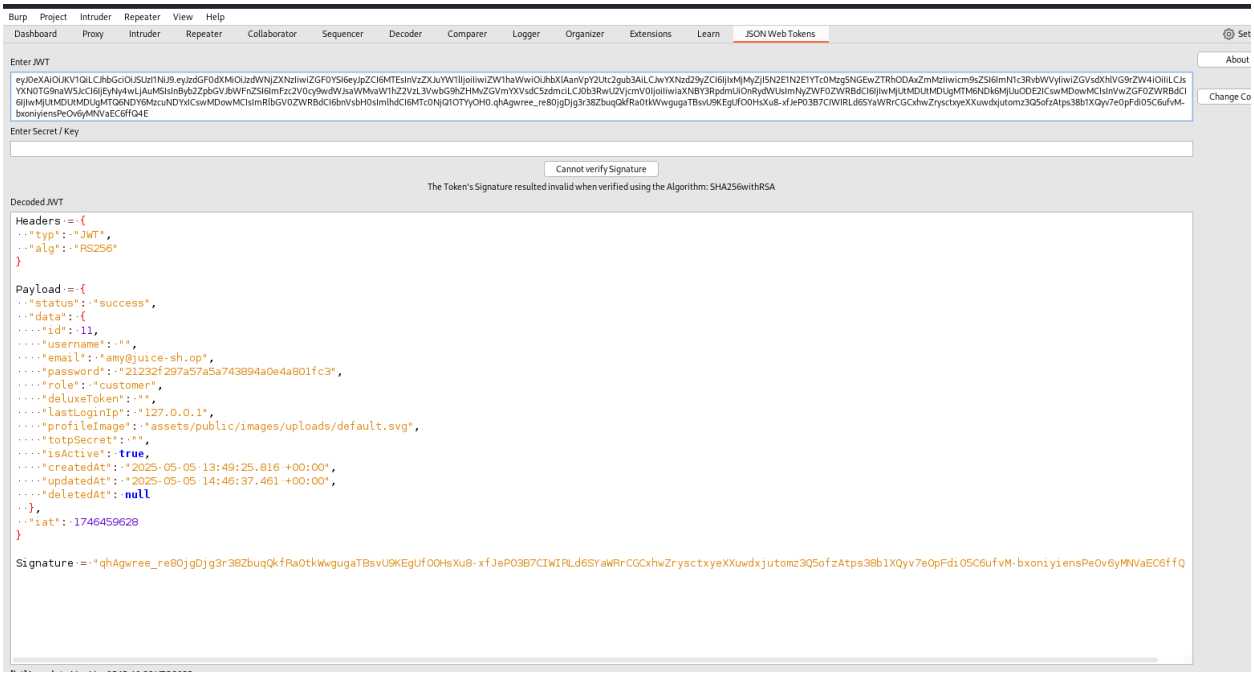
1. Explicitly Disallow `alg: none`: Ensure that your JWT parsing library and backend logic reject tokens with `alg: none`.
2. Enforce Signature Validation: All tokens must be signed using secure algorithms like `HS256`, `RS256`, or `ES256`, and verified on every request.
3. Use Trusted JWT Libraries: 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.

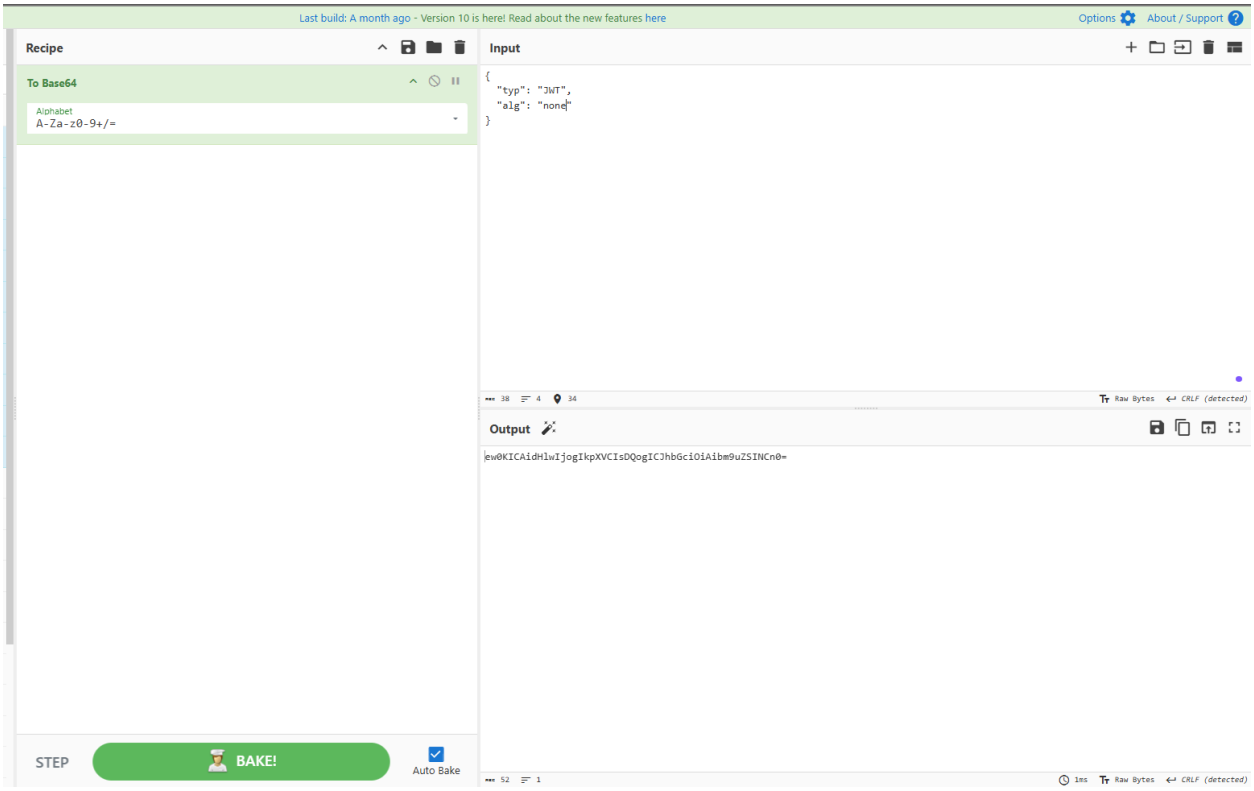


3. Using JSON Web Tokens extension in Burpsuite to decode the token and reveal its informations :

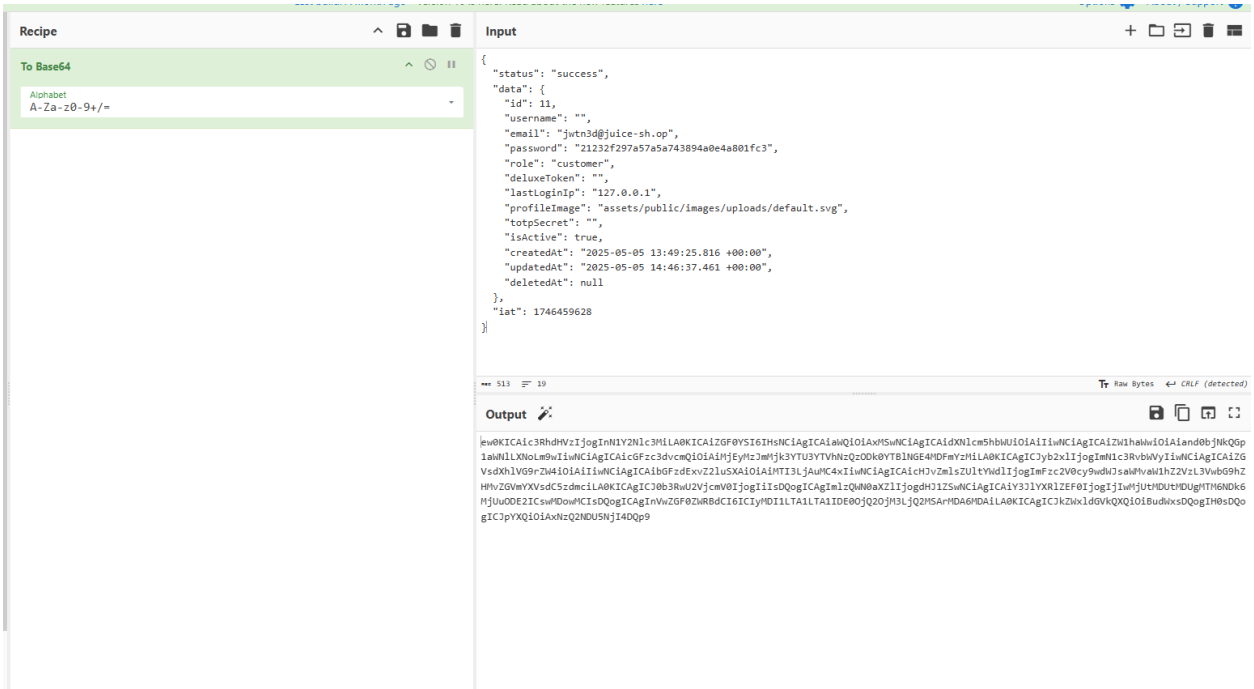


4. Craft a JWT with no signature :

   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)



4.2 In the payload portion:Change the email parameter with any existing users email (e.g jwtn3d@juice-sh.op) then encode it to Base64.



Header:

{ "alg": "none" }

Payload:

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

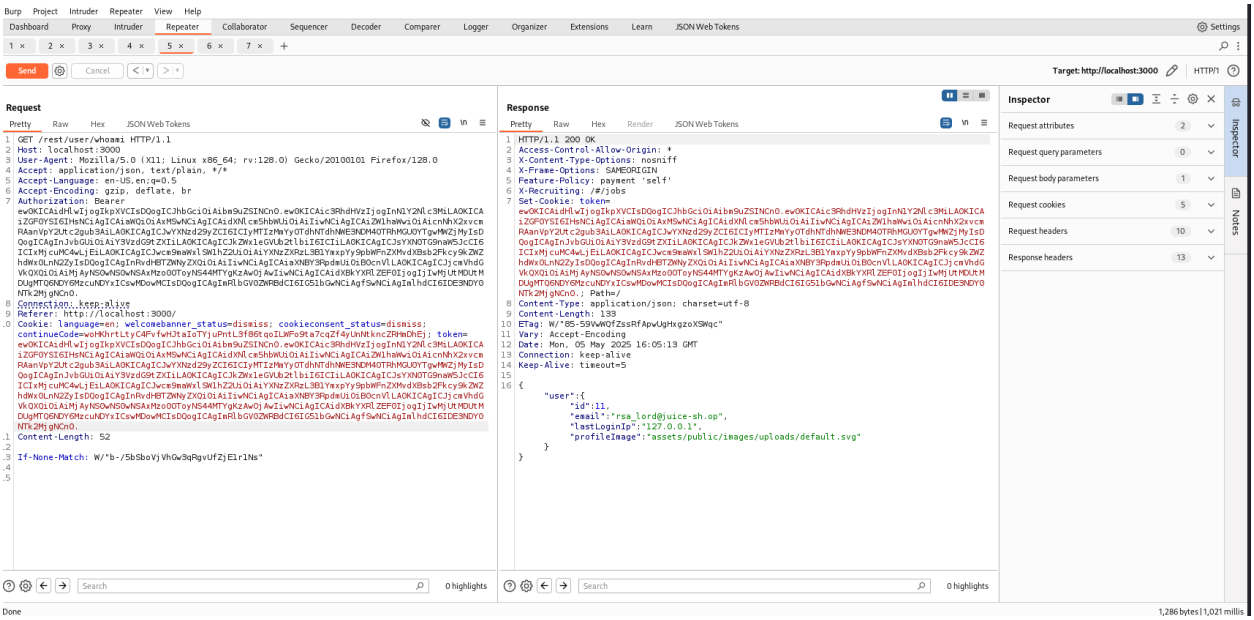
5. In the request change both Tokens values with the forged one:

ew0KICAidHlwIjogIkpXVCIsDQogICJhbGciOiAibm9uZSINCn0.ew0KICAic3RhdHVzIjogInN1Y2Nlc3MiLA0KICAiZGF0YSI6IHsNCiAgICAiaWQiOiAxMSwNCiAgICAidXNlcm5hbWUiOiAiIiwNCiAgICAiZW1haWwiOiAiand0bjNkQGp1aWNlLXNoLm9wIiwNCiAgICAicGFzc3dvcmQiOiAiMjEyMzJmMjk3YTU3YTVhNzQzODk0YTBINGE4MDFmYzMiLA0KICAgICJyb2xlIjogImN1c3RvbWVyIiwNCiAgICAiZGVsdXhlVG9rZW4iOiAiIiwNCiAgICAibGFzdExvZ2luSXAiOiAiMTI1LjAuMC4xIiwNCiAgICAicHJvZmlsZUltYWdlIjogImFzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdC5zdmciLA0KICAgICJ0b3RwU2VjcmV0IjogIiIsDQogICAgImlzQWN0aXZlIjogdHJ1ZSwNCiAgICAiY3JlYXRlZEF0IjogIjIwMjUtMDUtMDUgMTM6NDk6MjUuODE2ICswMDowMCIsDQogICAgInVwZGF0ZWRBdCI6ICIyMDI1LTA1LTA1IDE0OjQ2OjM3LjQ2MSArMDA6MDAiLA0KICAgICJkZWxldGVkQXQiOiBudWxsDQogIH0sDQogICJpYXQiOiAxNzQ2NDU5NjI4DQp9.

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 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:

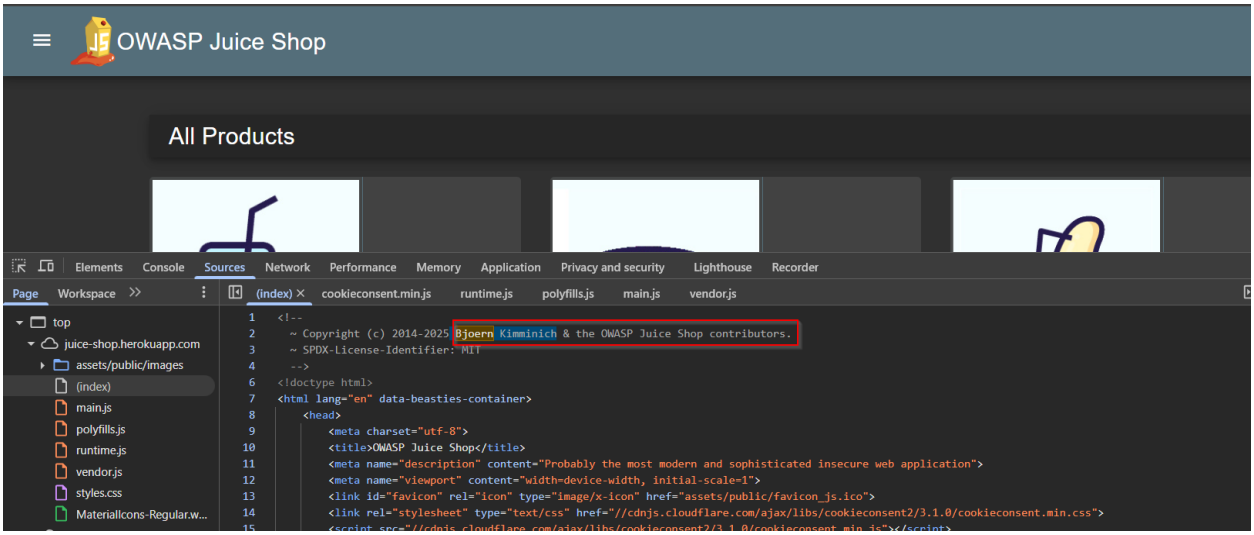
- Password reset (security question) functionality

Recommendations:

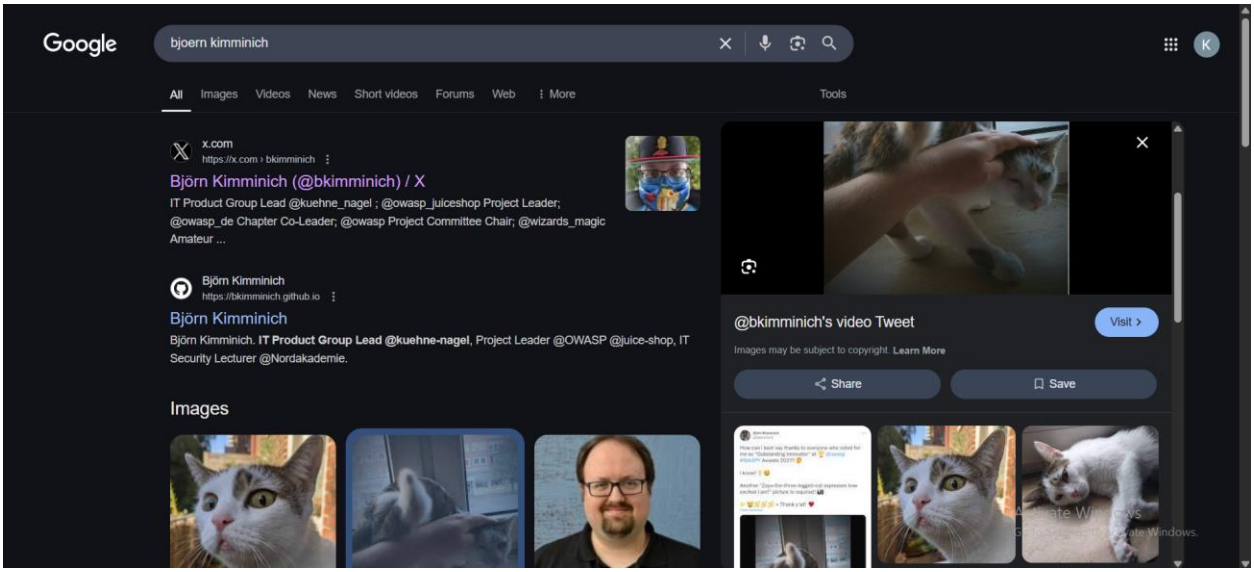
- Replace security questions with secure password recovery via time-limited, token-based email reset links.
- Avoid using static, guessable information as authentication factors.
- Educate privileged users about limiting personal information exposure that could aid in targeted attacks.
- Implement logging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

Proof of Concept:

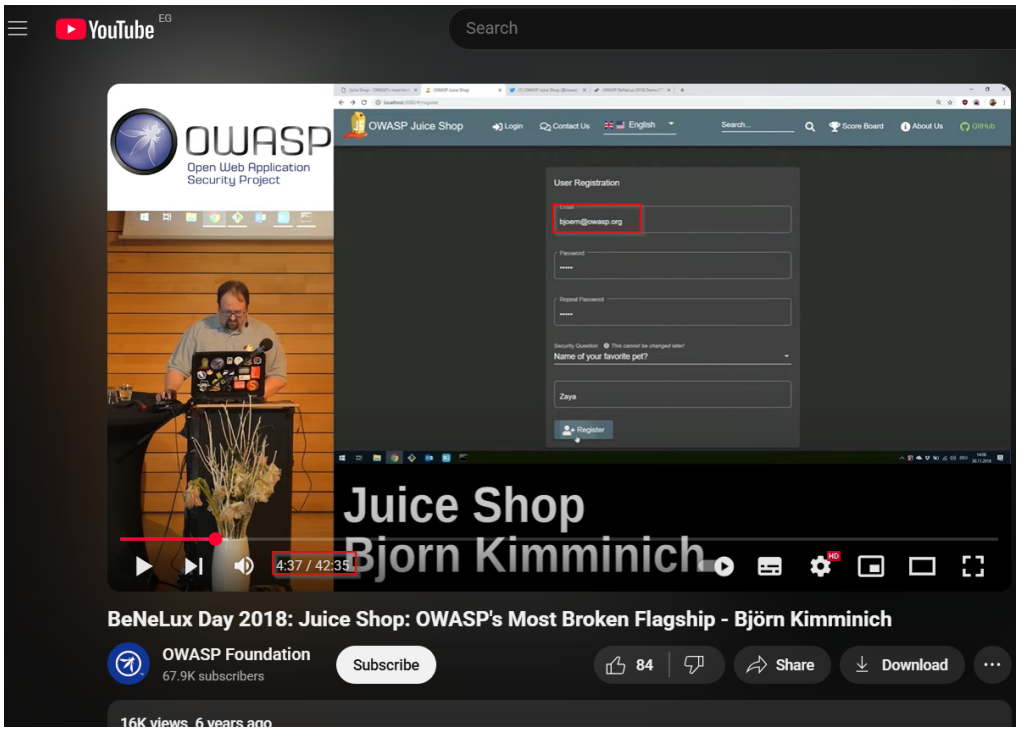
1. First I tried to Identify Bjoern so I searched in the source code for the name.



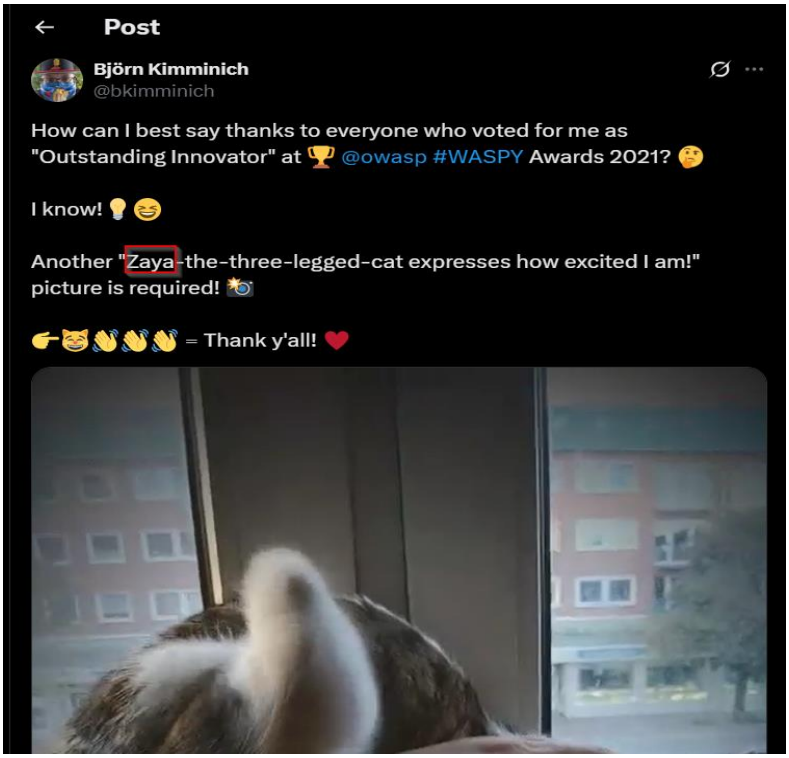
2. Turned out he is one of the contributors of the site so I searched on the internet to gain any info about him.



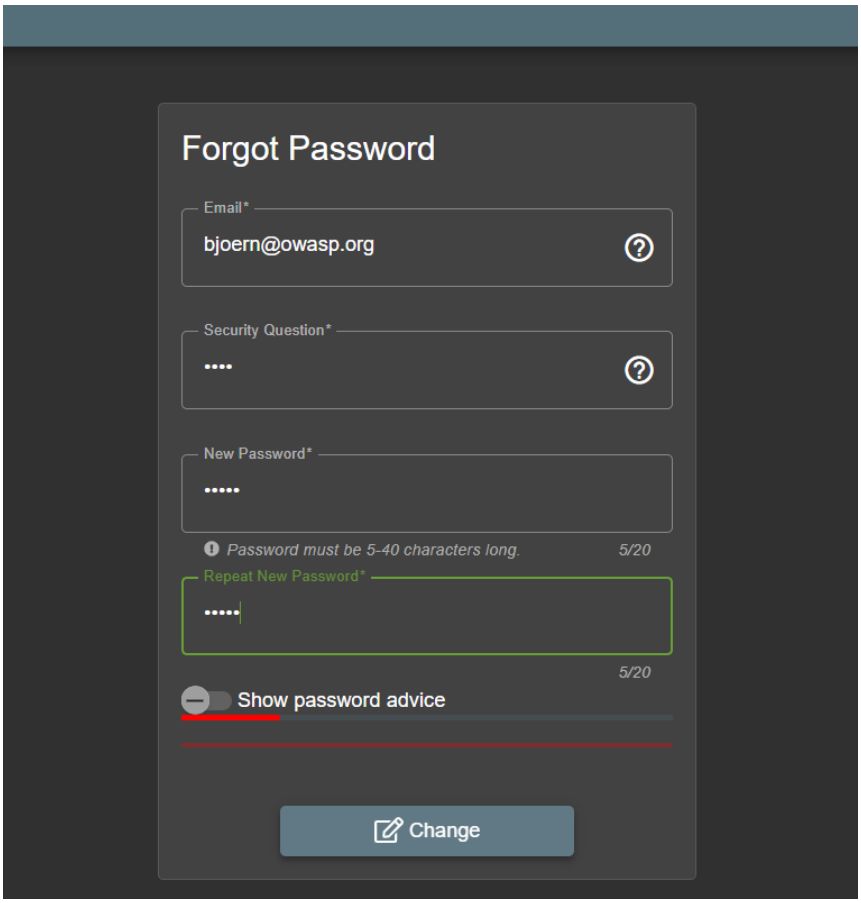
3. Then I found a video of him while he was signing up on the site.



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



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



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:

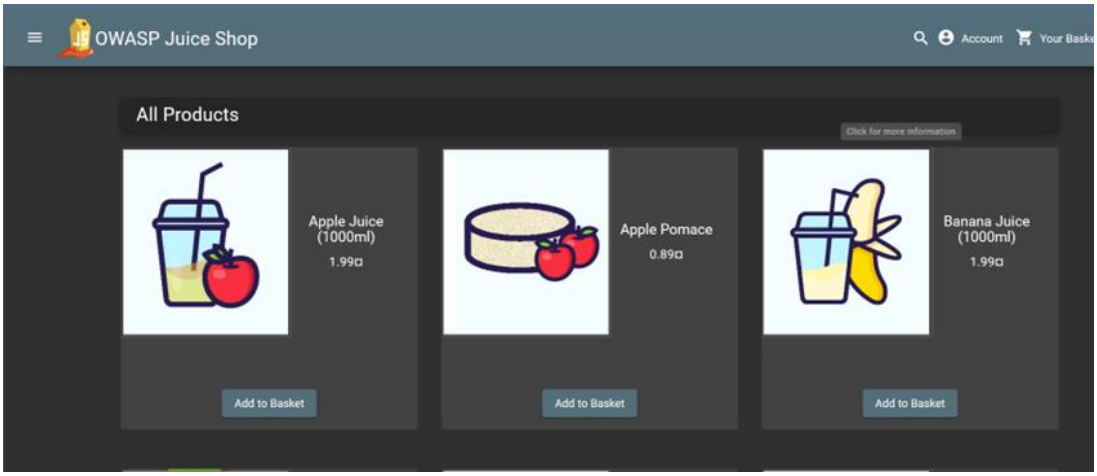
1. **Apply Server-Side Validation:**
    - Ensure the amount falls within a realistic and expected range.
2. **Enforce Numeric Value Checks:**

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

3. **Monitor Wallet Activity:**
   - Log and analyze unusual wallet activities.

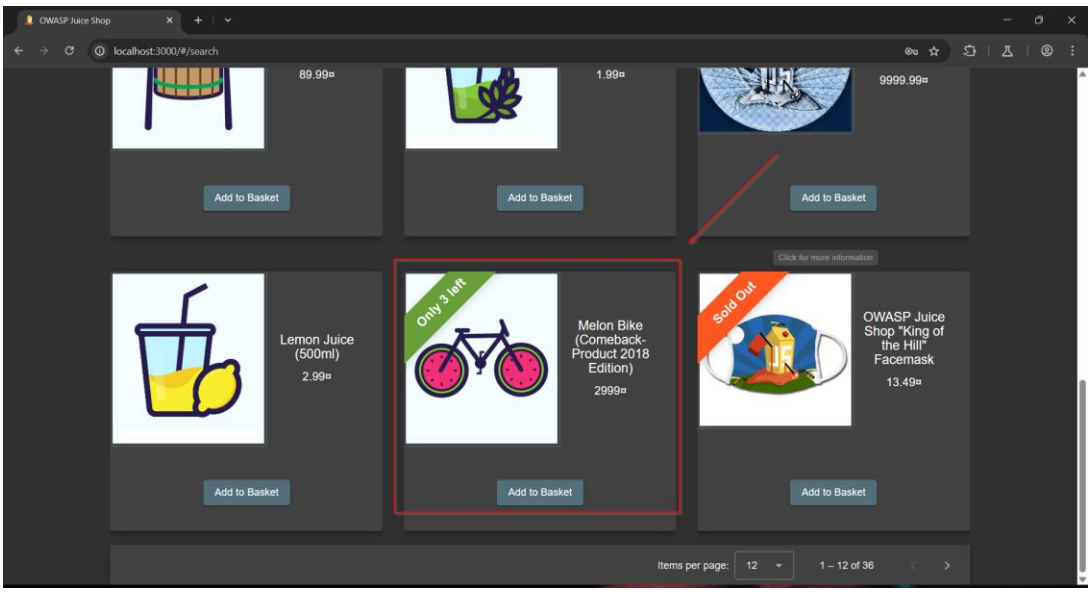
Proof of Concept (PoC):

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

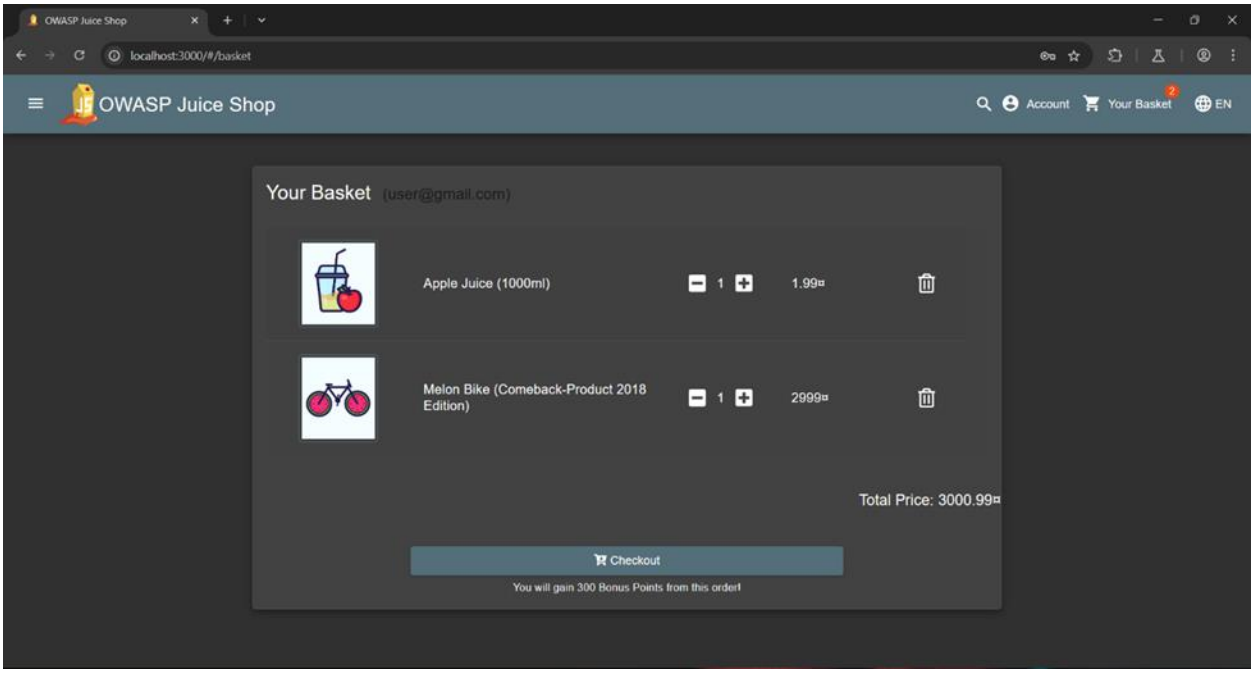


2. Open Burp suit and intercept this page

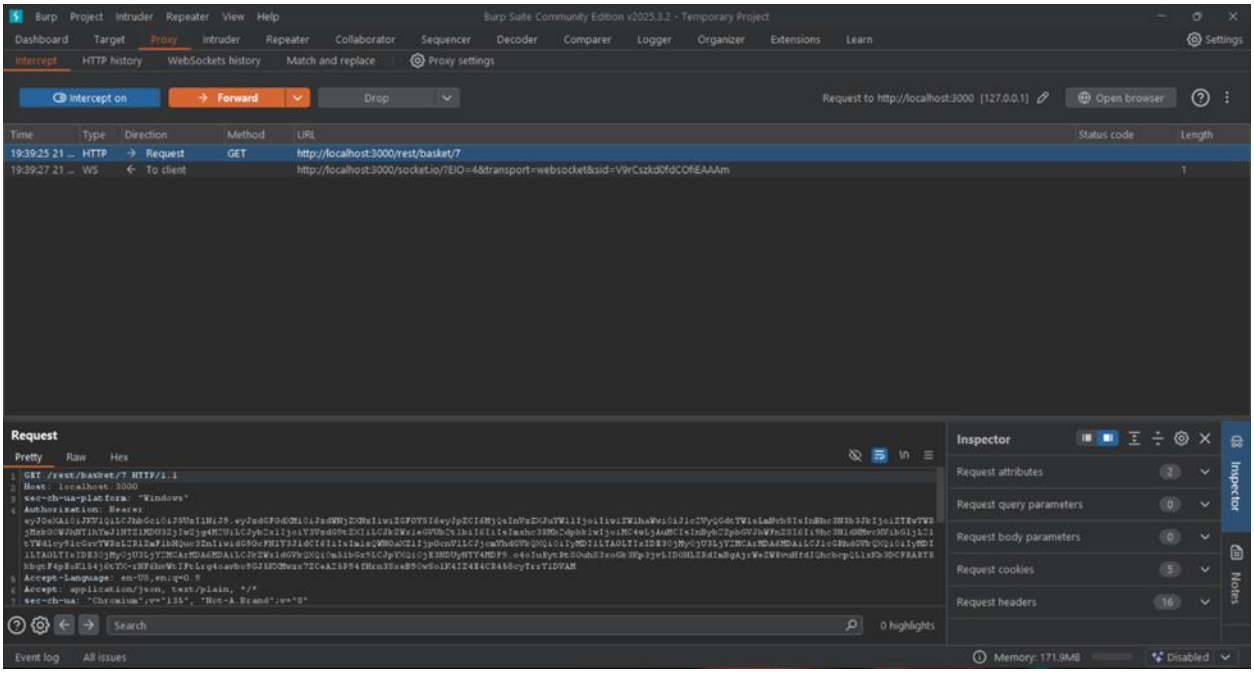
3. Search for big item and add to basket



4. Navigate to the **Basket** section

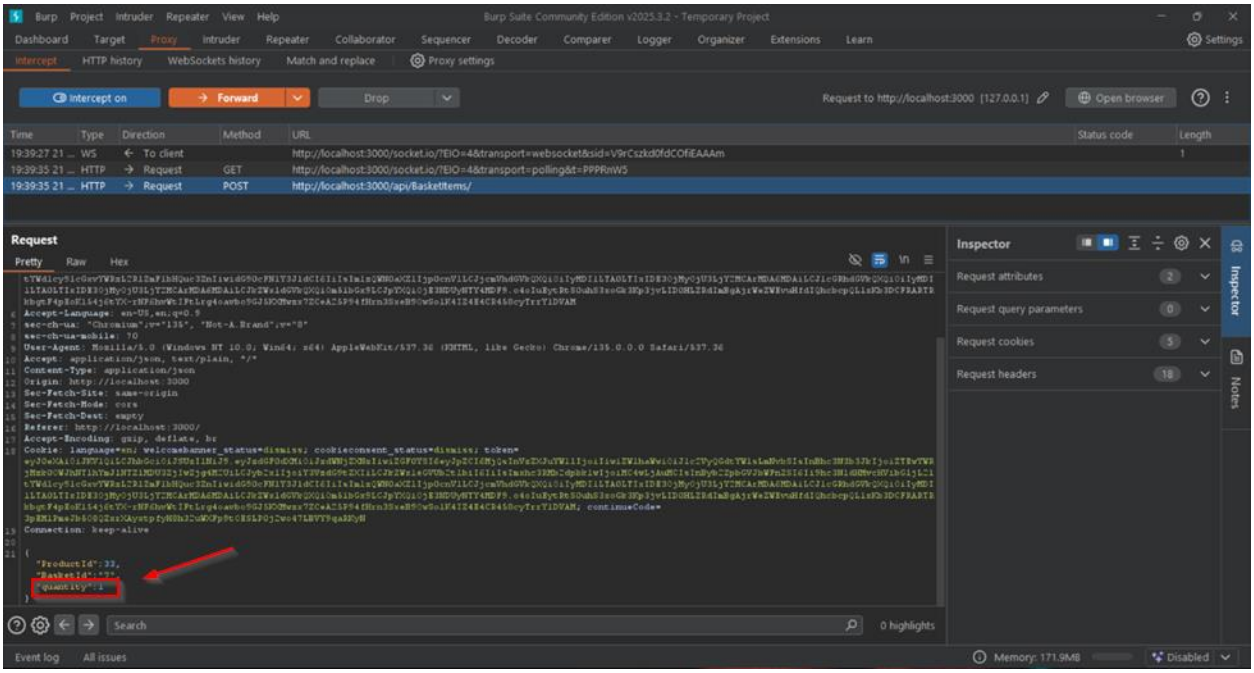


5. Forward **GET** request



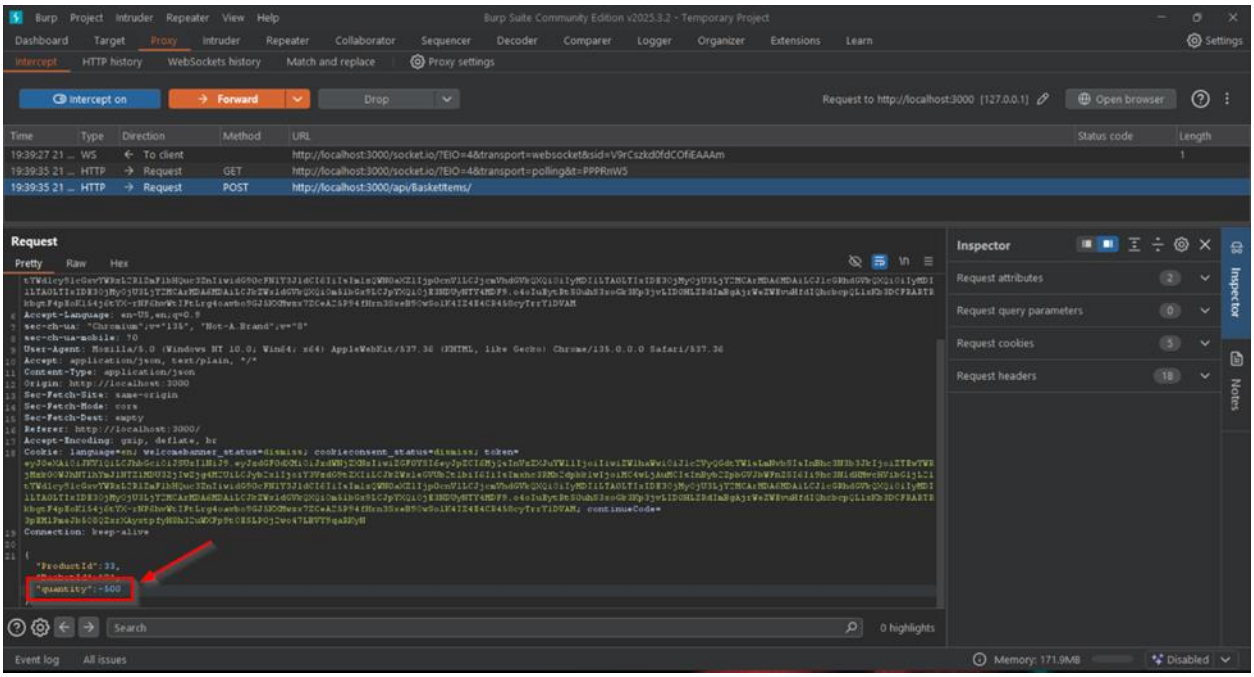
6. Modify “quantity” : 1 in request

POST /api/basketitem

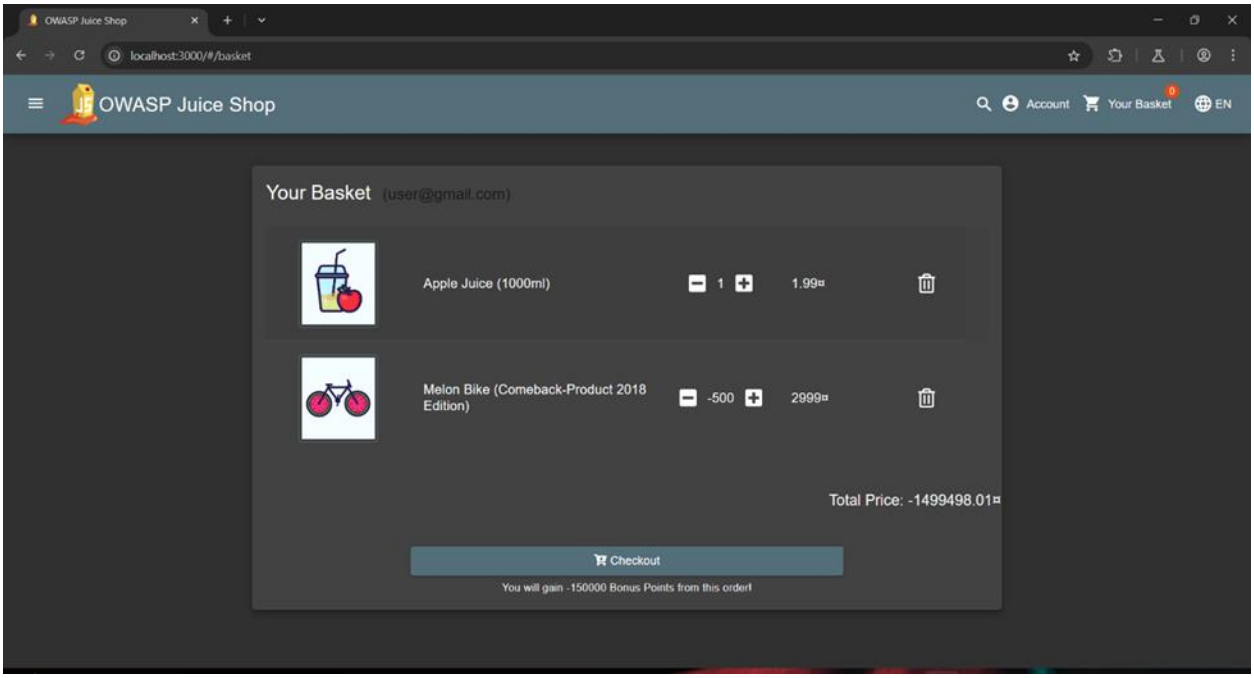


7. Modified top-up request showing an exaggerated amount (e.g., -500)

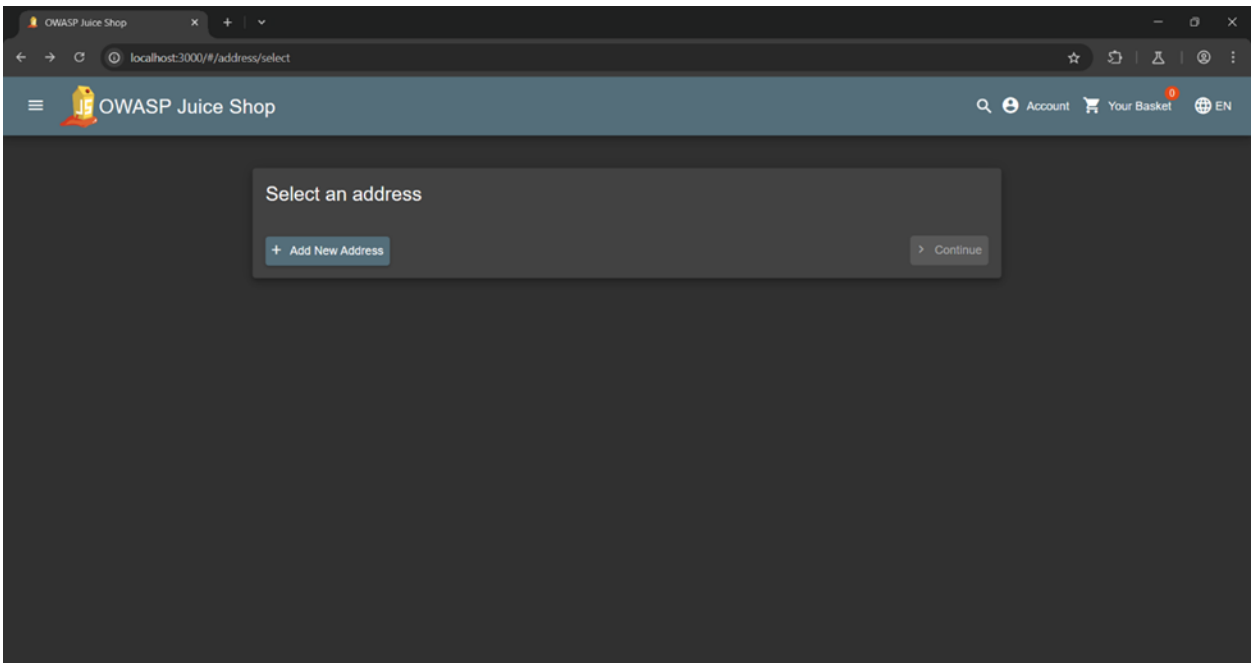
8. Forward request and stop interception

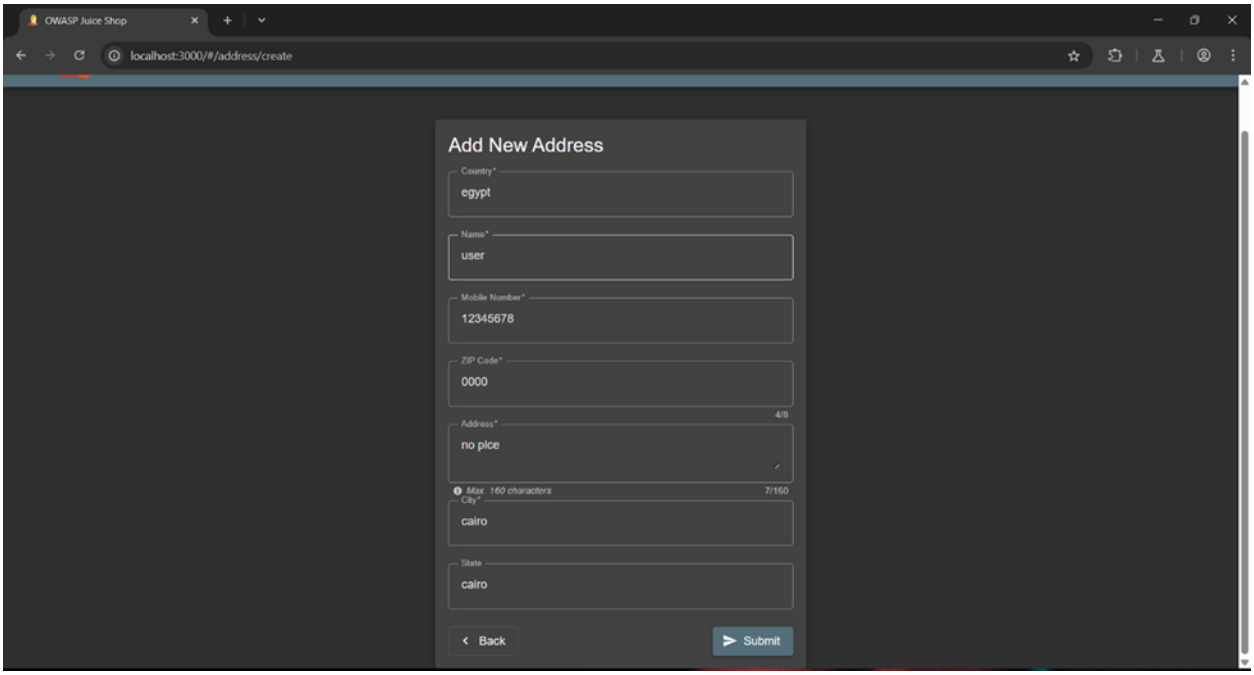


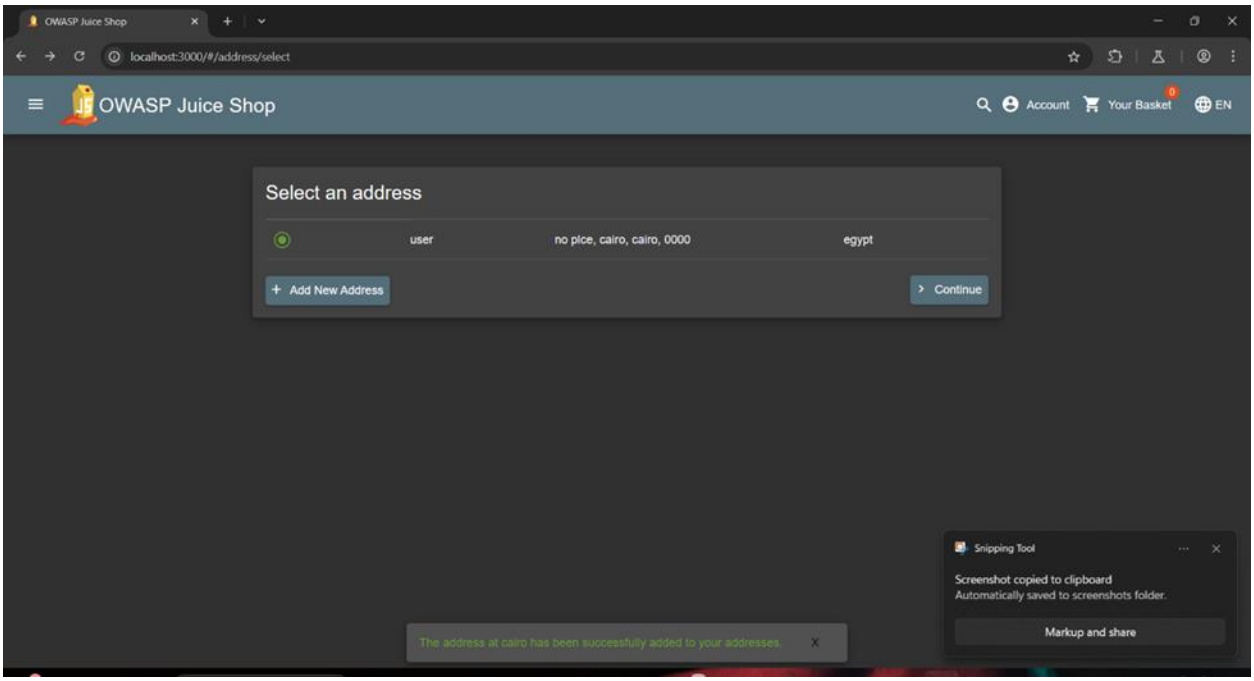
9. Open basket, the amount change

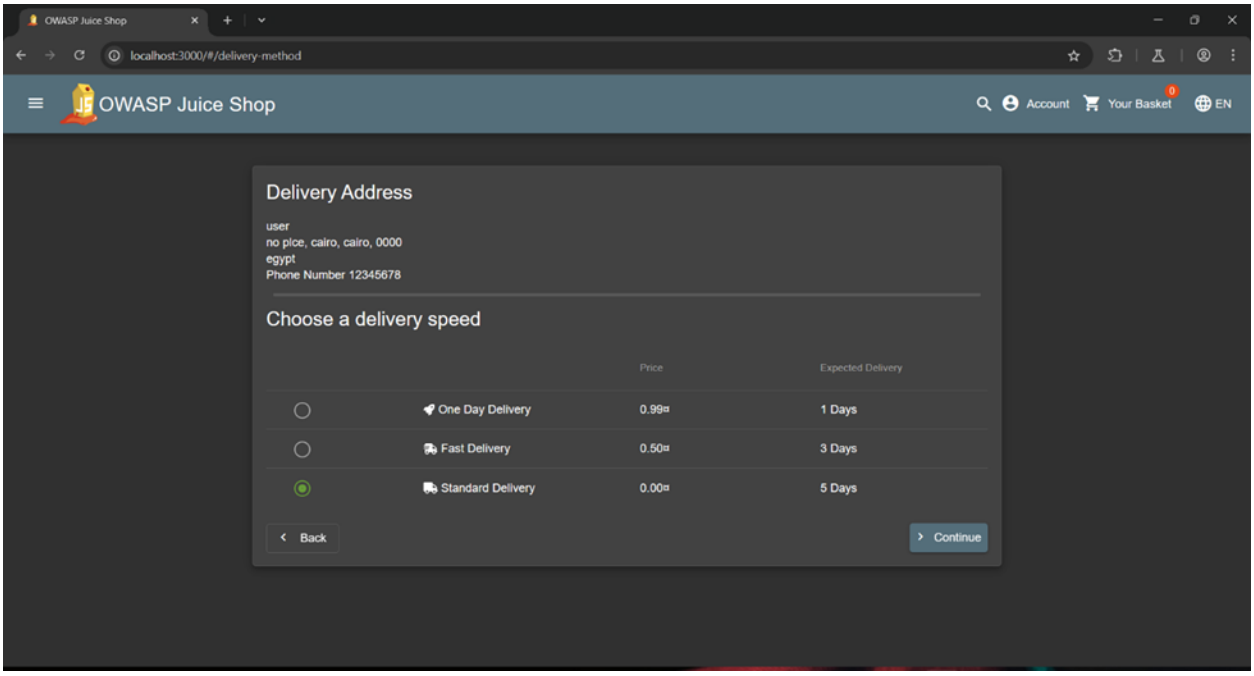


10. Select **checkout** and continue the operation

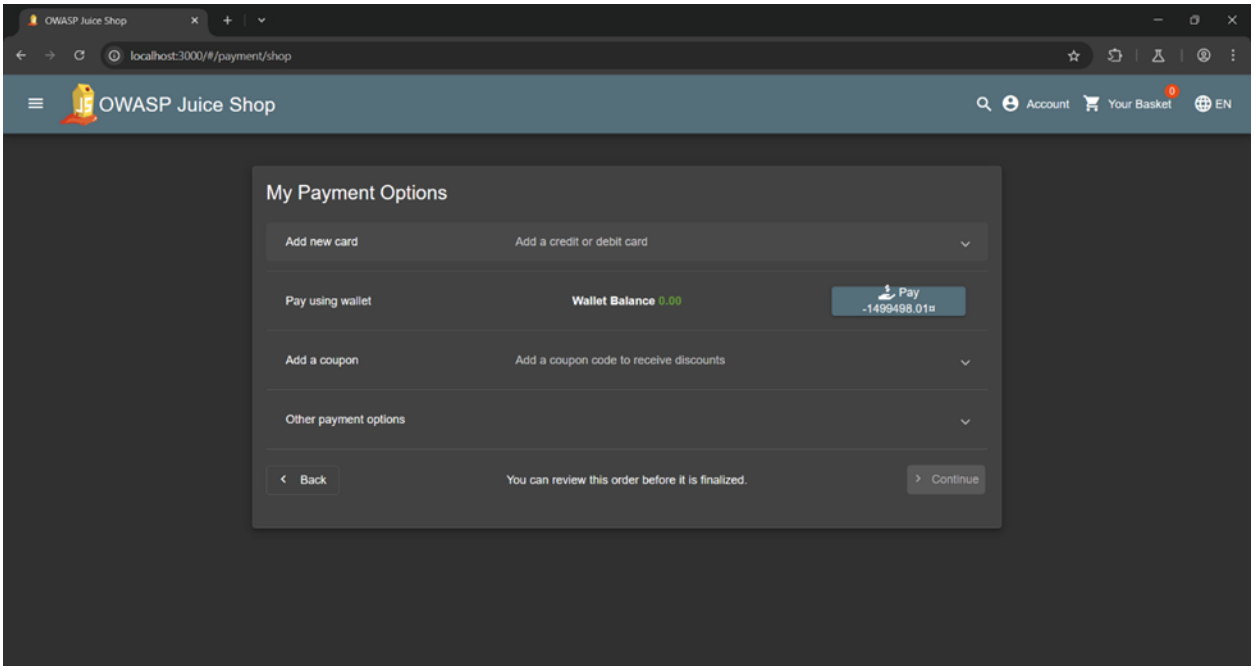


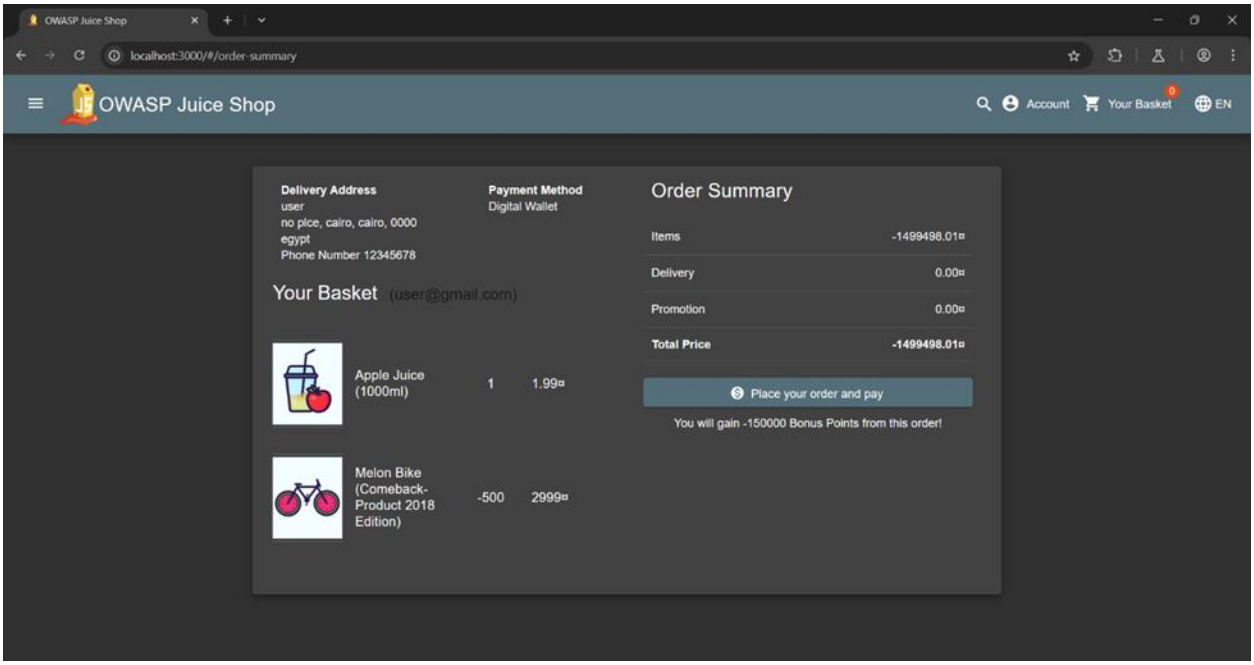


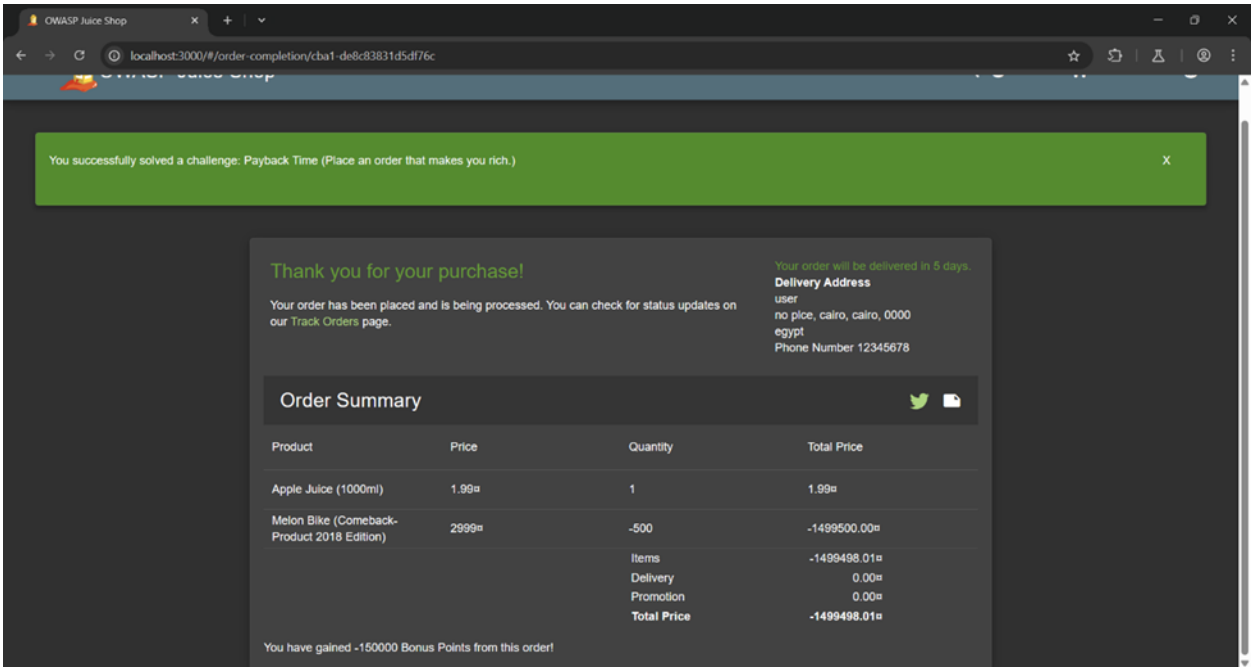

OWASP Juice Shop
localhost:3000/#/address/select
OWASP Juice Shop
Account
Your Basket
EN
Select an address
user
no plce, cairo, cairo, 0000
egypt
Add New Address
Continue
Snipping Tool
Screenshot copied to clipboard
Automatically saved to screenshots folder.
Markup and share
The address at cairo has been successfully added to your addresses.


OWASP Juice Shop
localhost:3000/#/delivery-method
OWASP Juice Shop
Account
Your Basket
EN
Delivery Address
user
no plce, cairo, cairo, 0000
egypt
Phone Number 12345678
Choose a delivery speed
Price
Expected Delivery
One Day Delivery
0.99¤
1 Days
Fast Delivery
0.50¤
3 Days
Standard Delivery
0.00¤
5 Days
Back
Continue

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.)
X
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 pice, 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¤
Total Price
-1499498.01¤
You have 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:

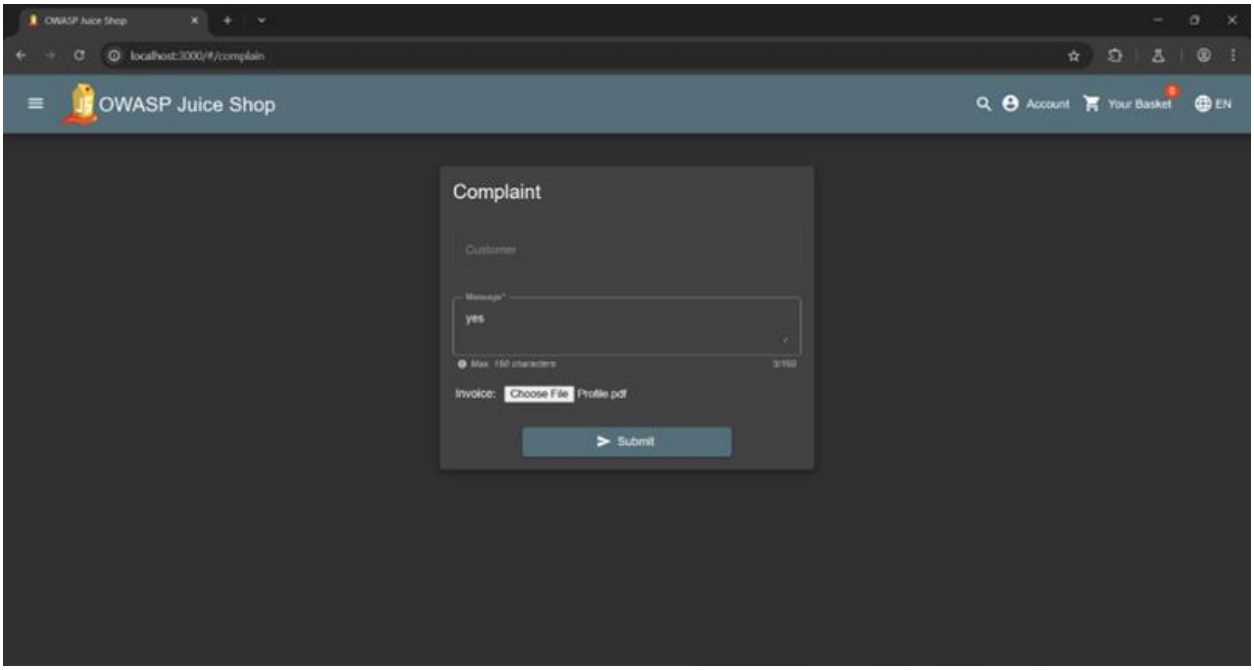
1. **Restrict File Extensions:**
    - Allow only known safe file extensions (e.g., .png, .jpg).
2. **Store Files Outside Web Root:**
    - Avoid executing or interpreting uploaded content.

Proof of Concept (PoC):

1.1 Log in to the Juice Shop as a user.

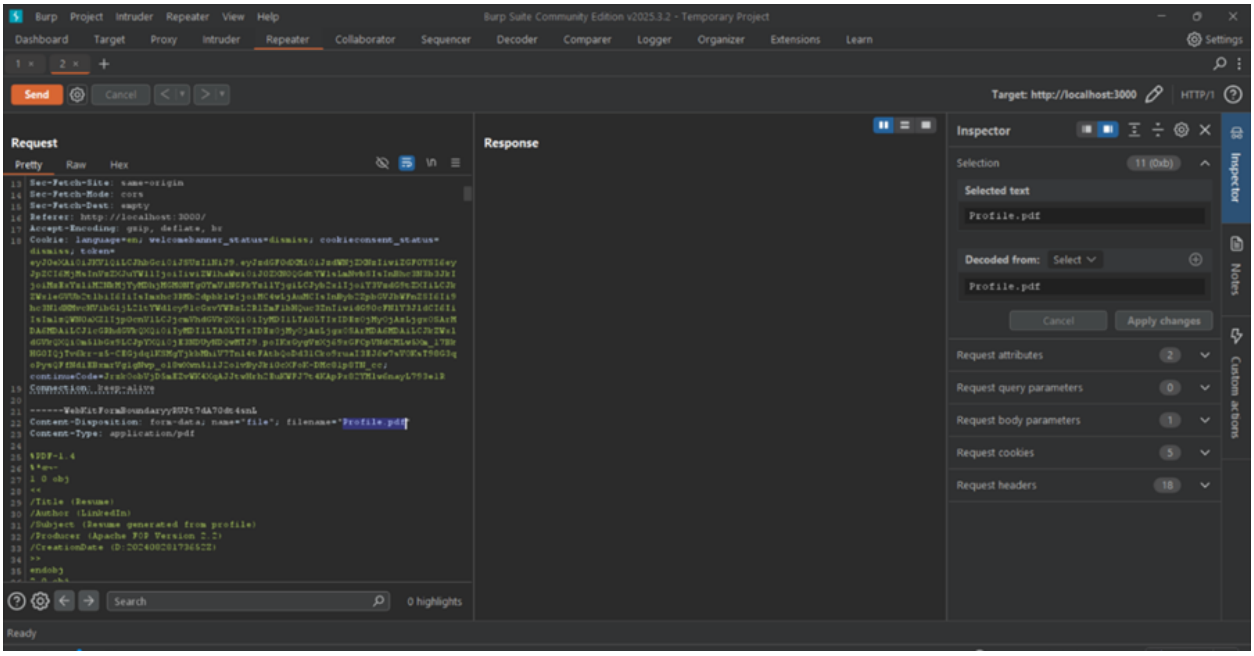
1.2 Navigate to the **Complaint** section.

1.3 Fill the form and upload .pdf or .zip only

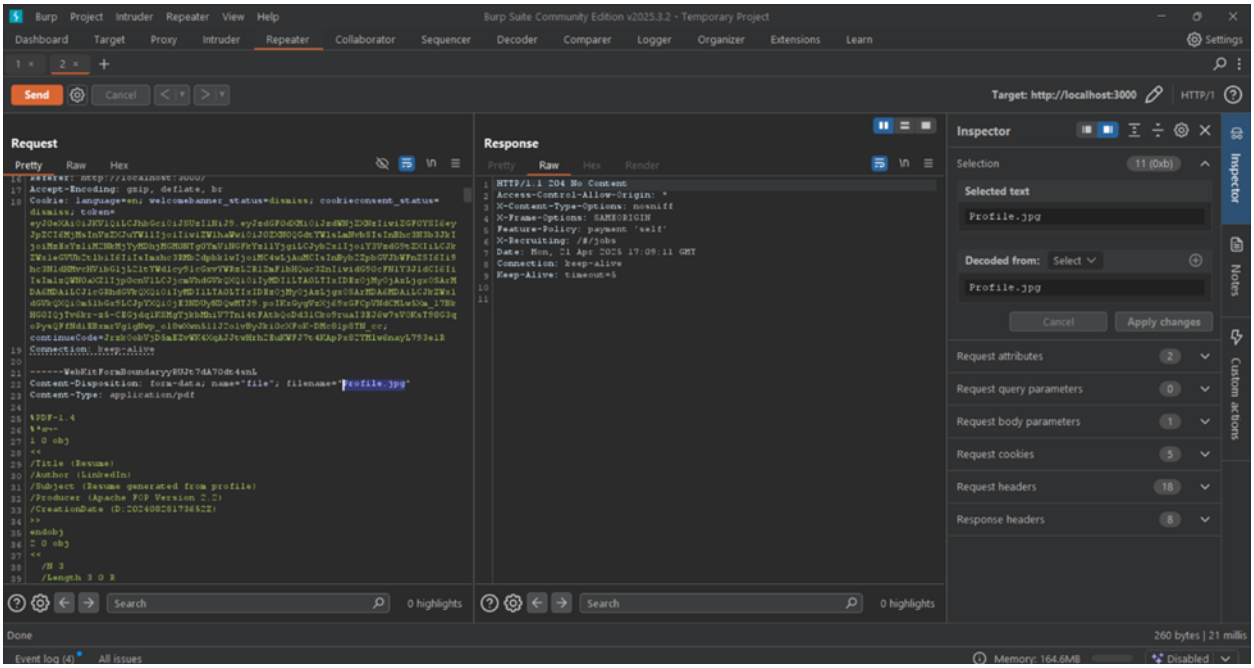


2. Intercept upload request with Burp Suite

3. Modify file extension to (ex: .jpg)



4. The Server response request is accepted



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:

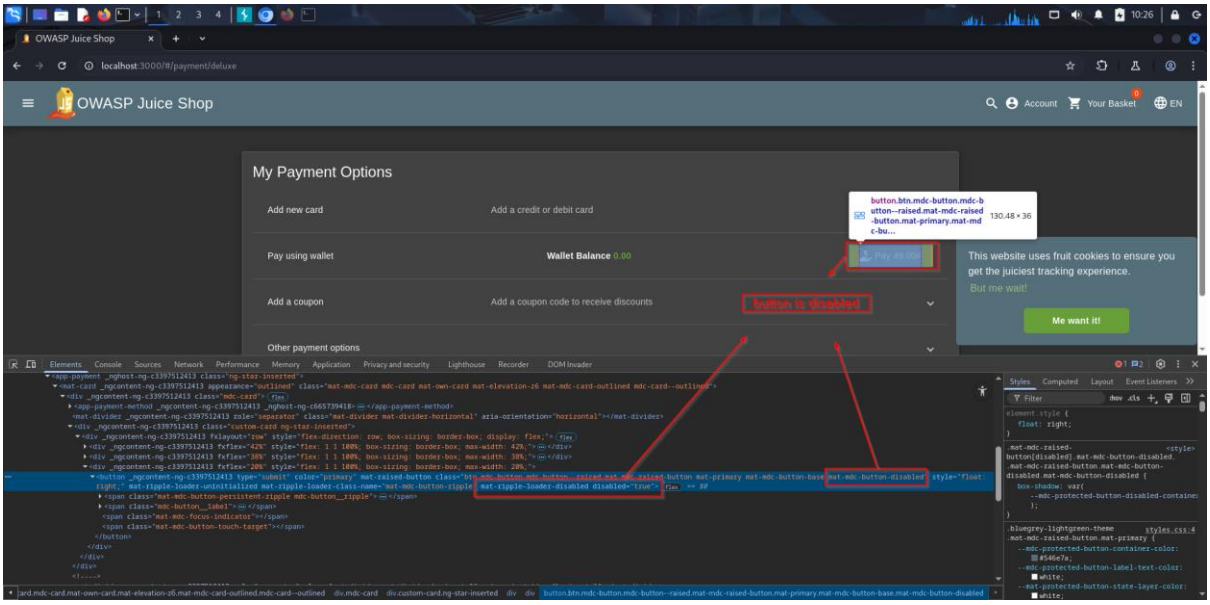
- http://localhost:3000/#/payment/deluxe

Recommendations:

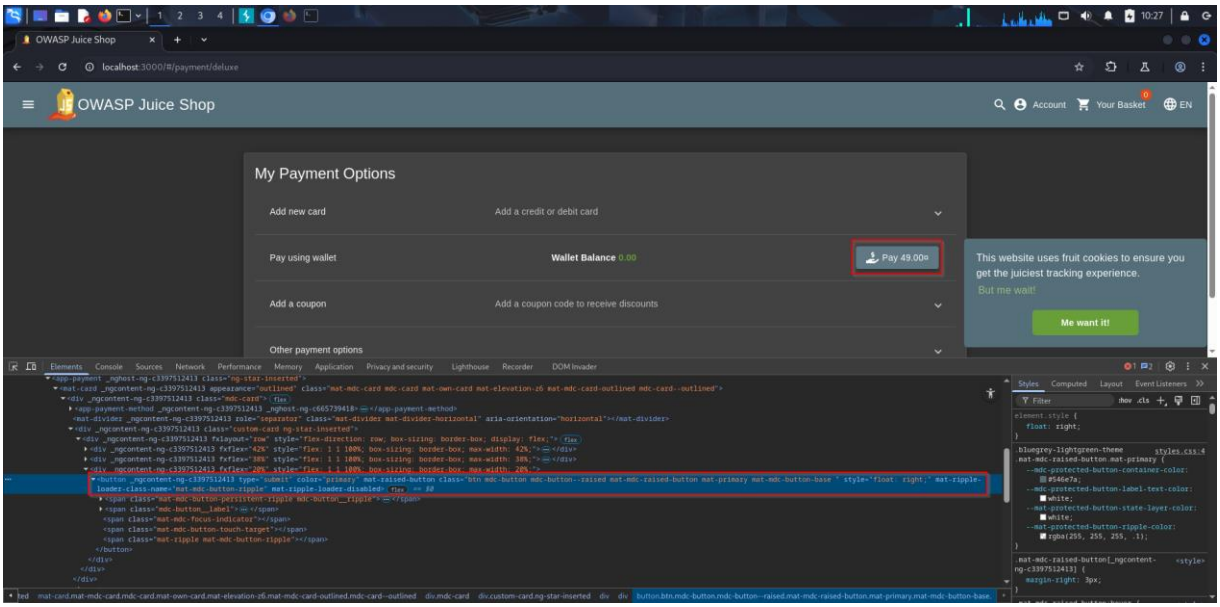
- Enforce all membership validation and payment logic **on the server side**.
- Never rely on client-side attributes (like `disabled`) for security decisions.
- Ensure backend verifies wallet/funds existence and successful payment before granting access.

Proof of Concept:

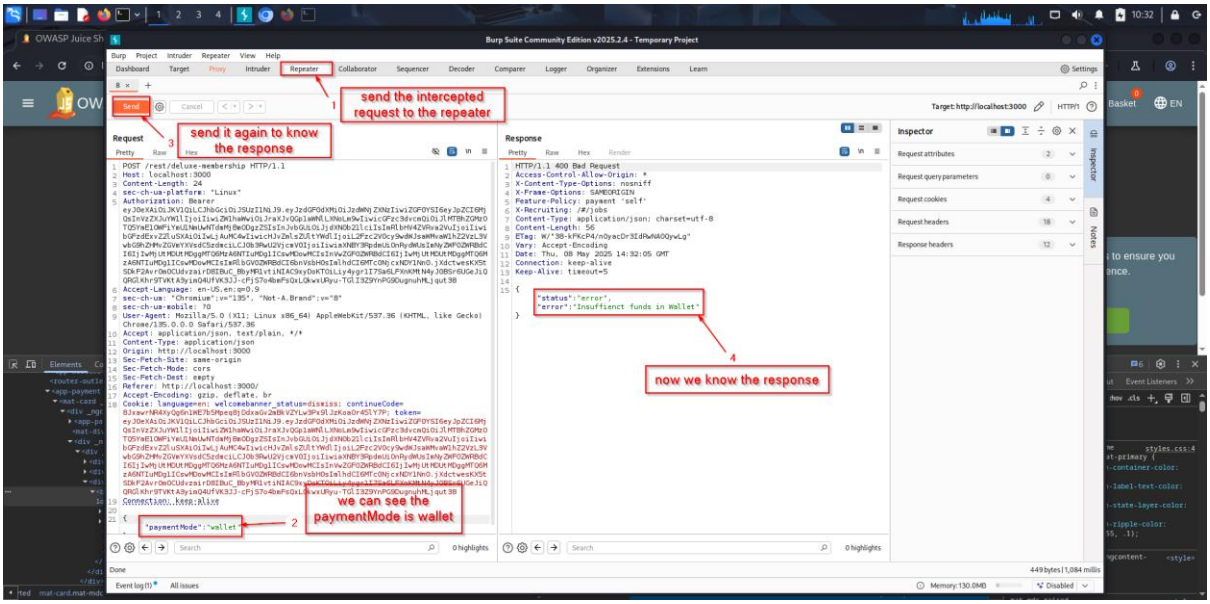
1. In the payment page we can see that pay button is disabled so let's inspect the source code



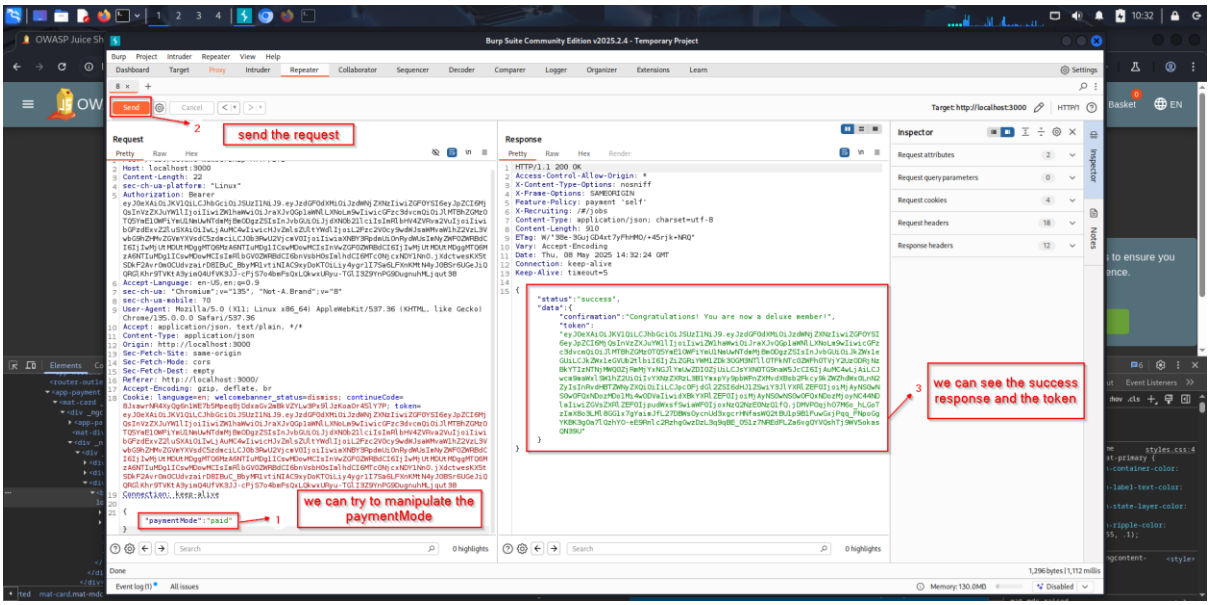
2. delete disability attribute



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



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



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:

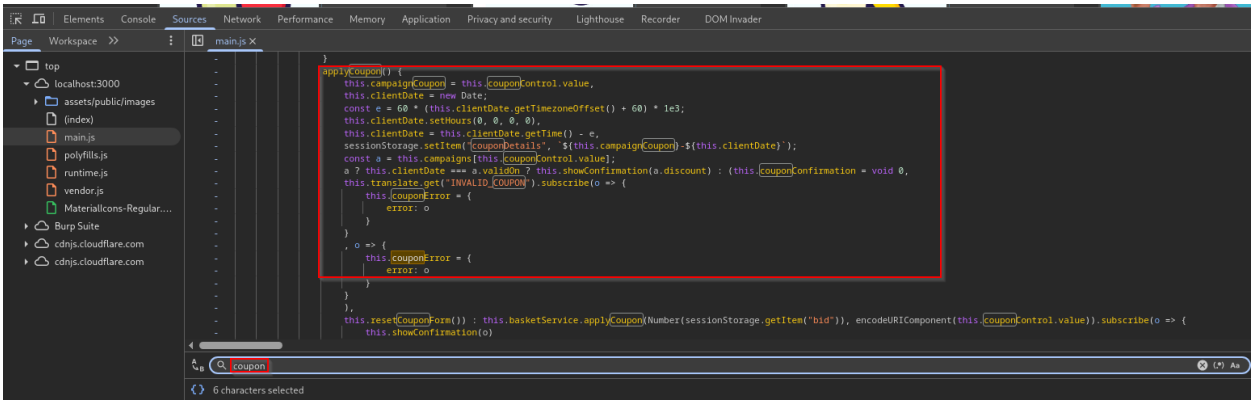
- coupon validation logic (main.js)

Recommendations:

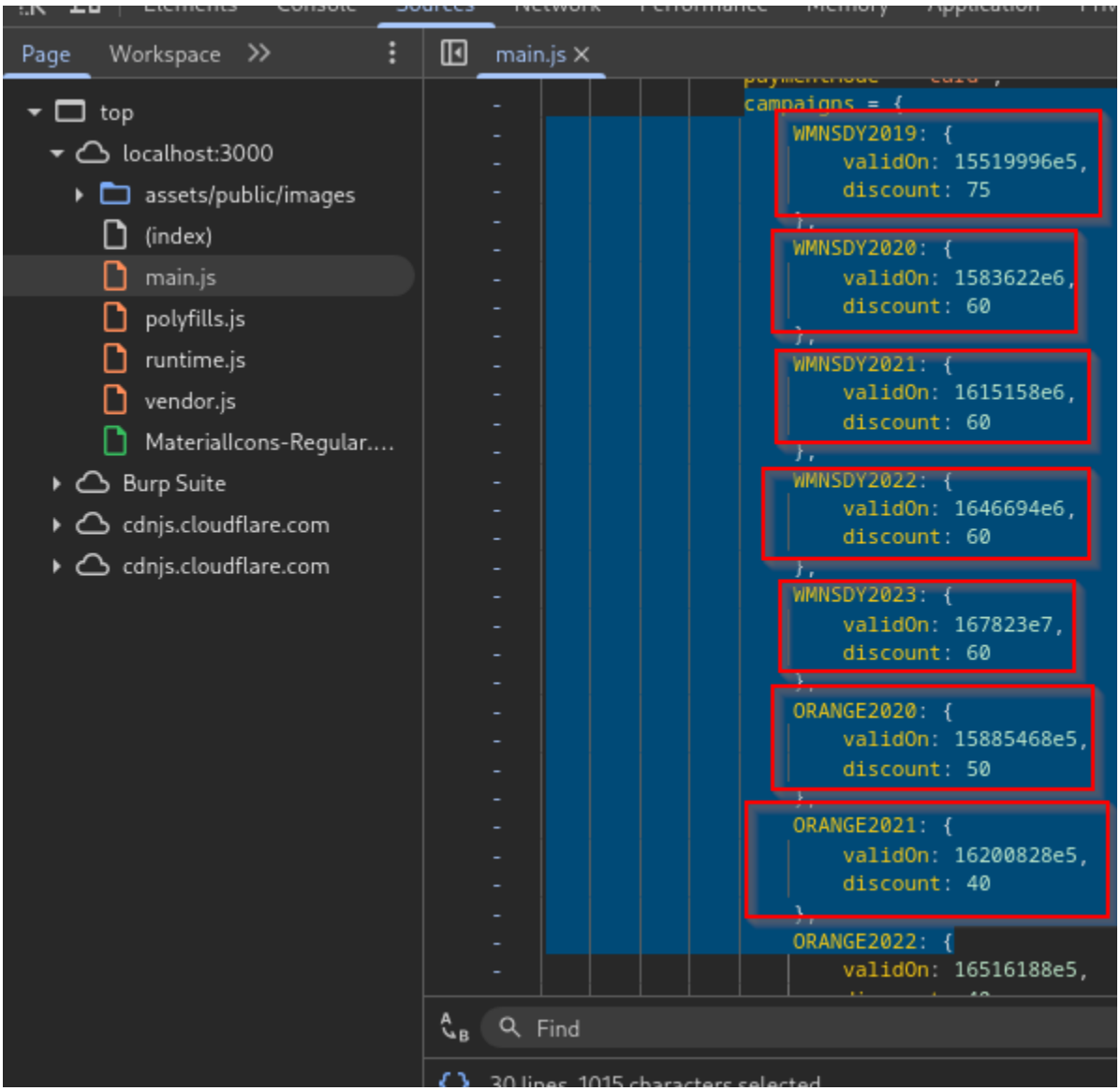
- Move coupon validation and expiration checks to the server.
- Never trust client-side data or time values for enforcing business logic.

Proof of Concept:

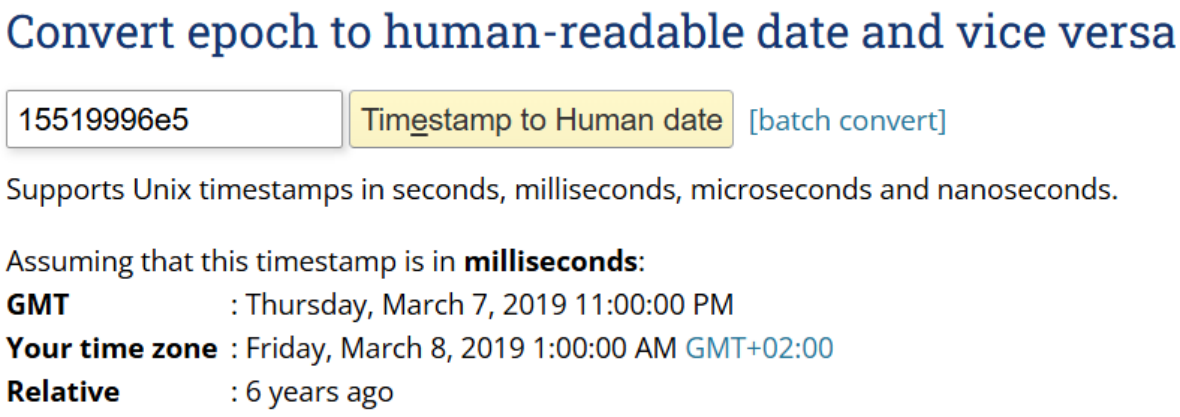
1. searching "coupon" into the JavaScript code of main.js revealed the function responsible for applying coupons. This function checks the coupon's validity against a client-side timestamp and a hardcoded list of valid campaigns.



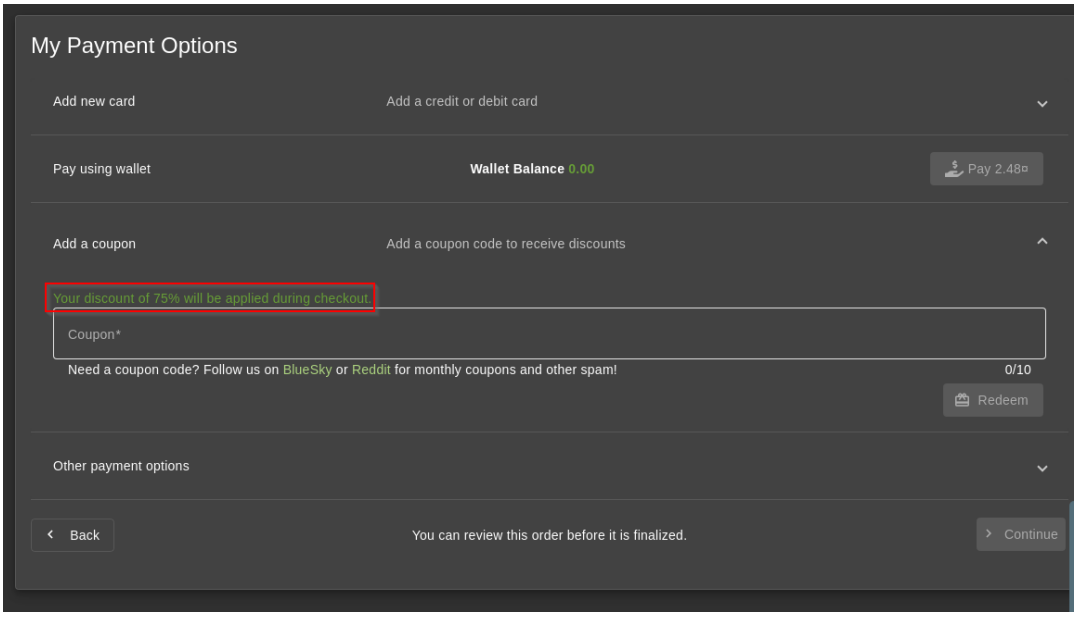
2. Found the list of campaigns by using CTRL+F in the JavaScript code, with their respective expiration timestamps and discounts.

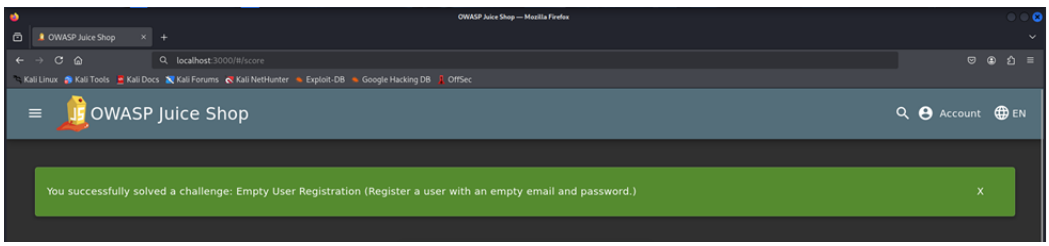


3. select a coupon randomly from the list that we get, and then, we use an online converter to convert the value of the timestamp, which is in milliseconds, in a valid date



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





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:

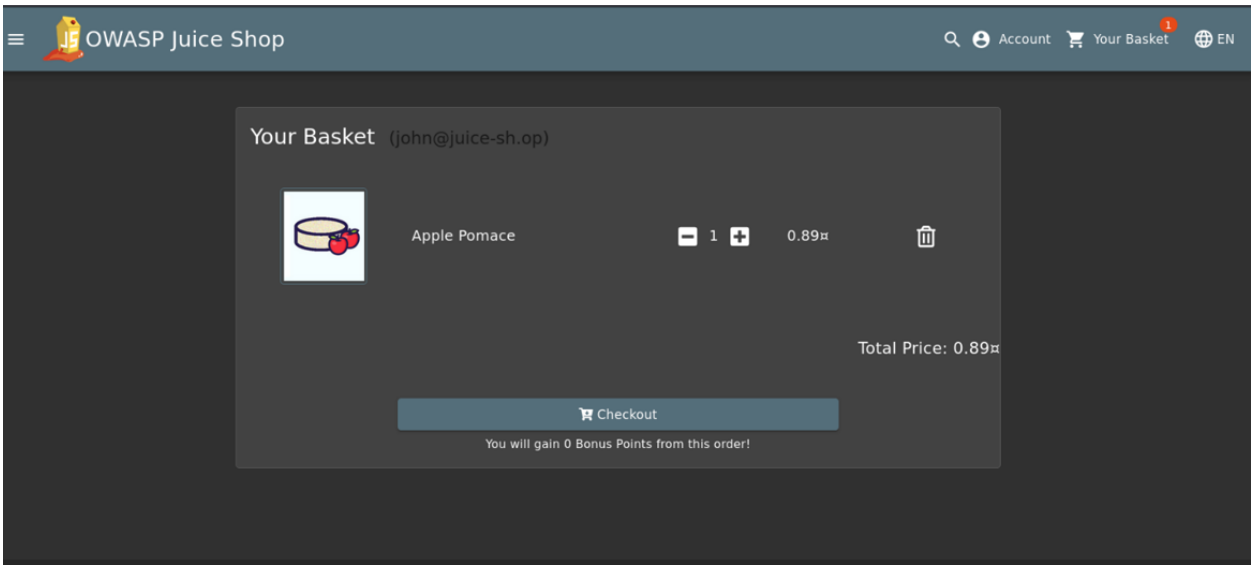
- OWASP Input Validation Cheat Sheet
- OWASP Business Logic Security Cheat Sheet

Recommendations:

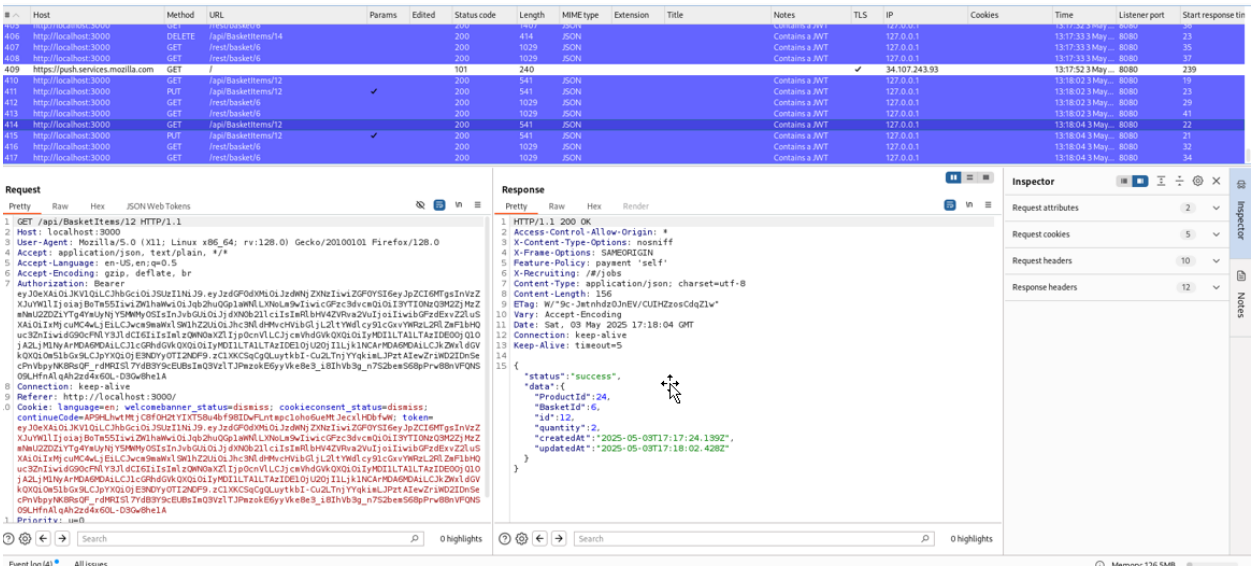
- Enforce **rule validations** to reject negative quantity values.
- Implement **business logic checks** that validate total order values before processing.

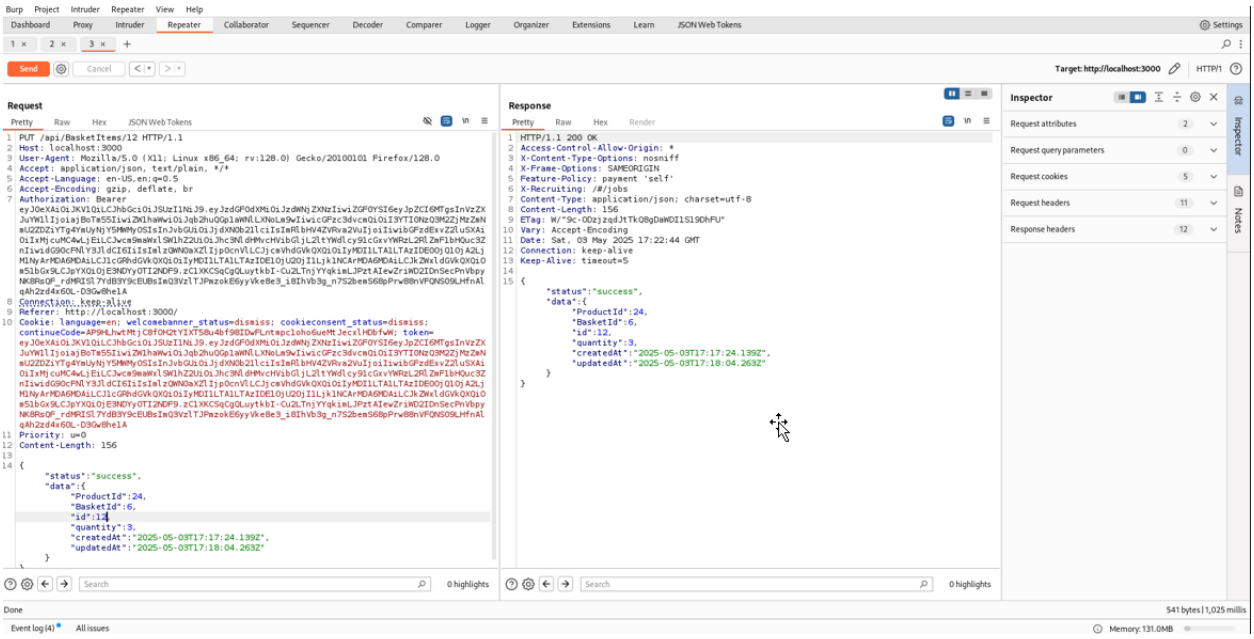
Proof of Concept (POC):

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

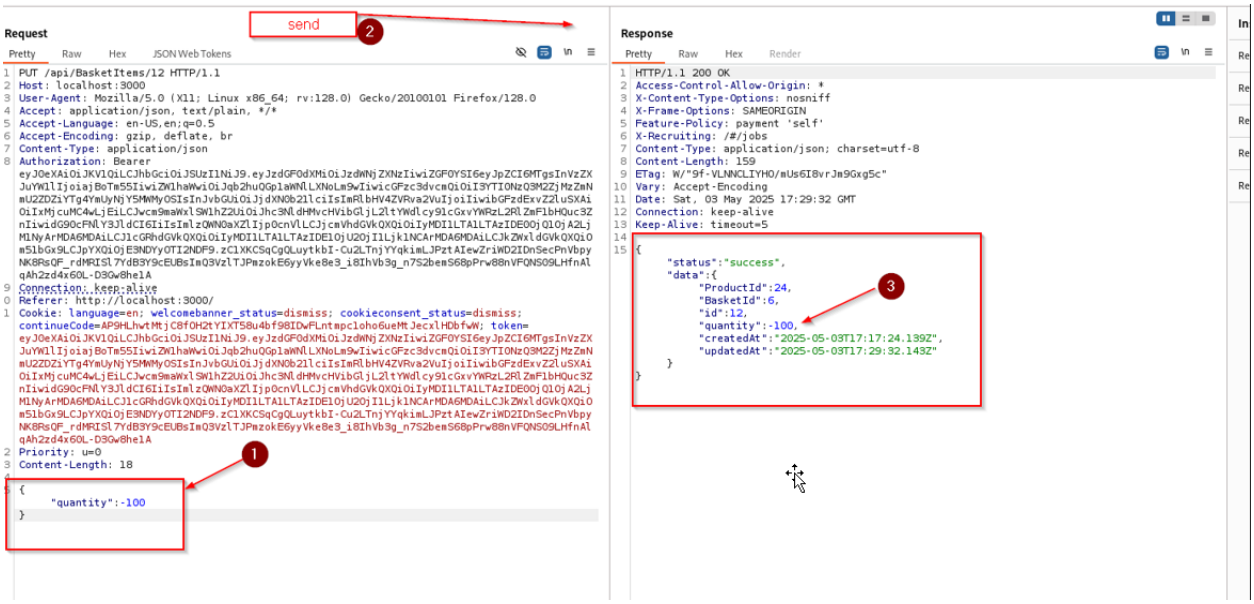


3. Intercept the PUT request /api/BasketItems/{id} using Burpsuite and send it to the repeater.



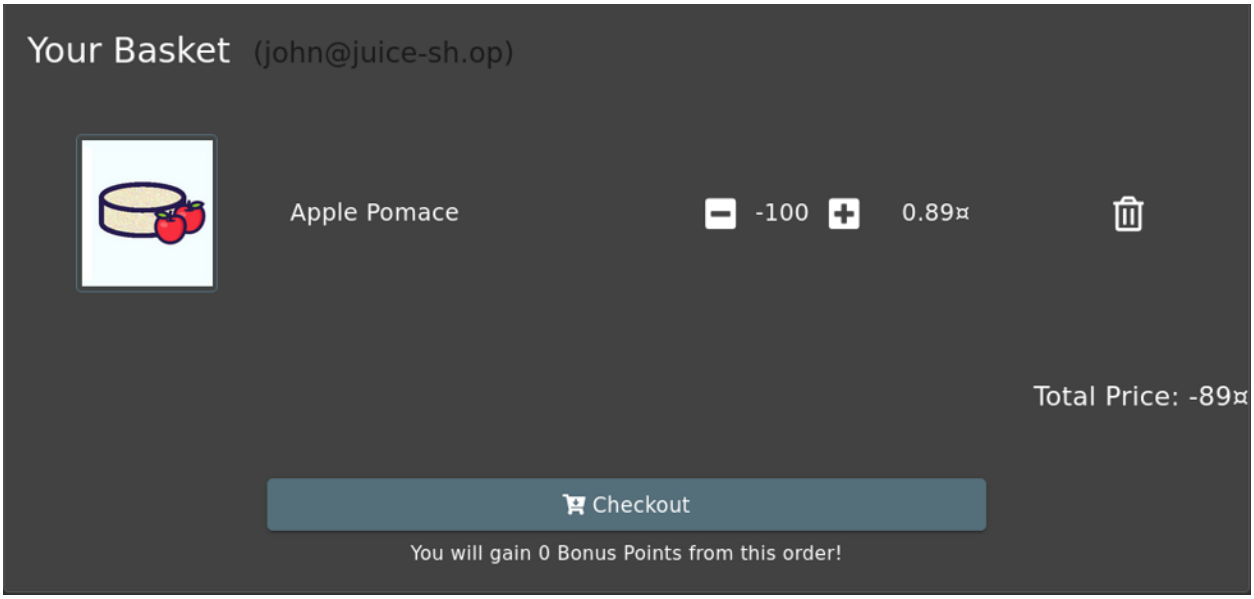


4. Modify the quantity parameter to a negative value (e.g., -100).

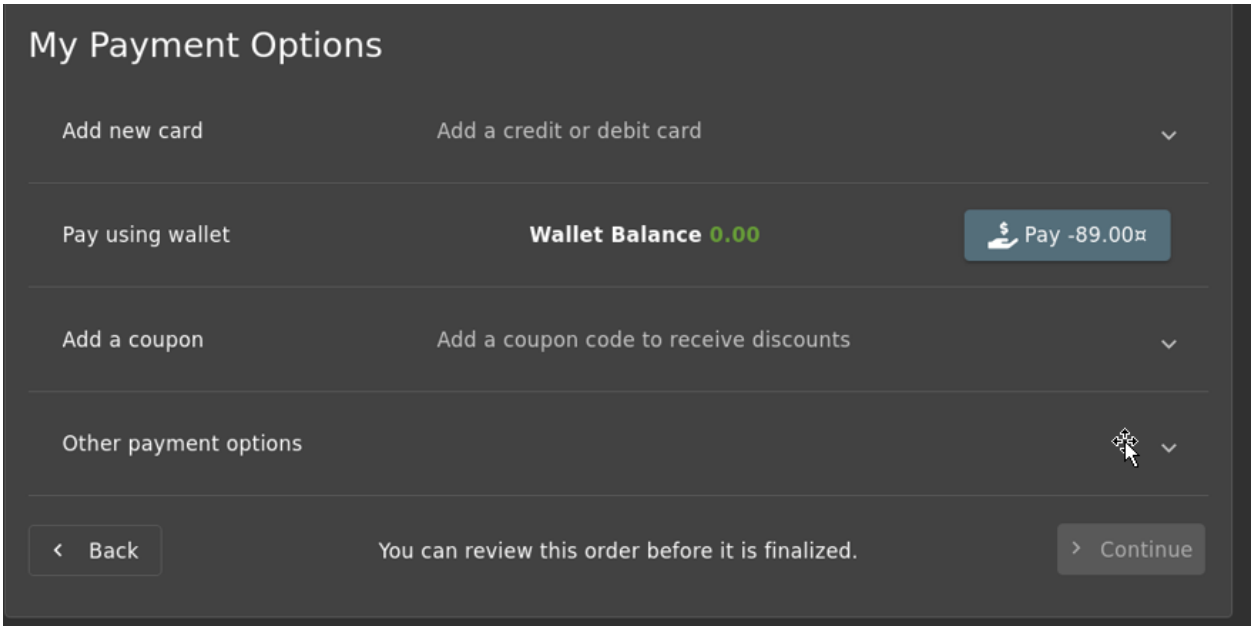


Forward the modified request, which updates the basket with the negative quantity.

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



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



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:

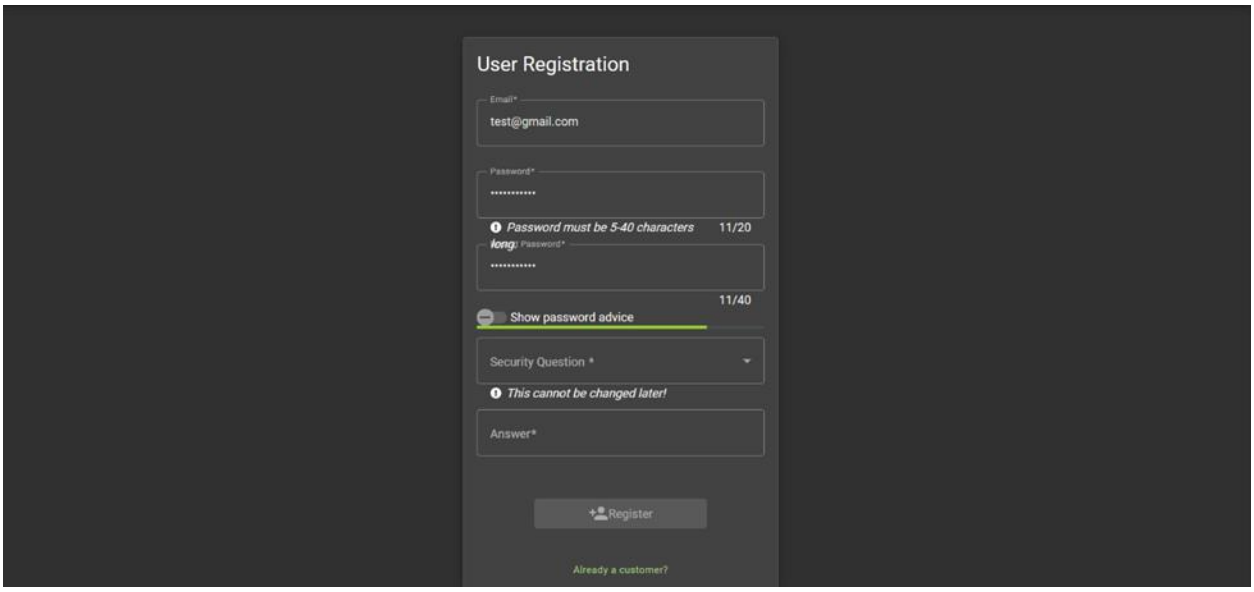
- CWE-640: Weak Password Recovery Mechanism for Forgotten Password

Recommendations:

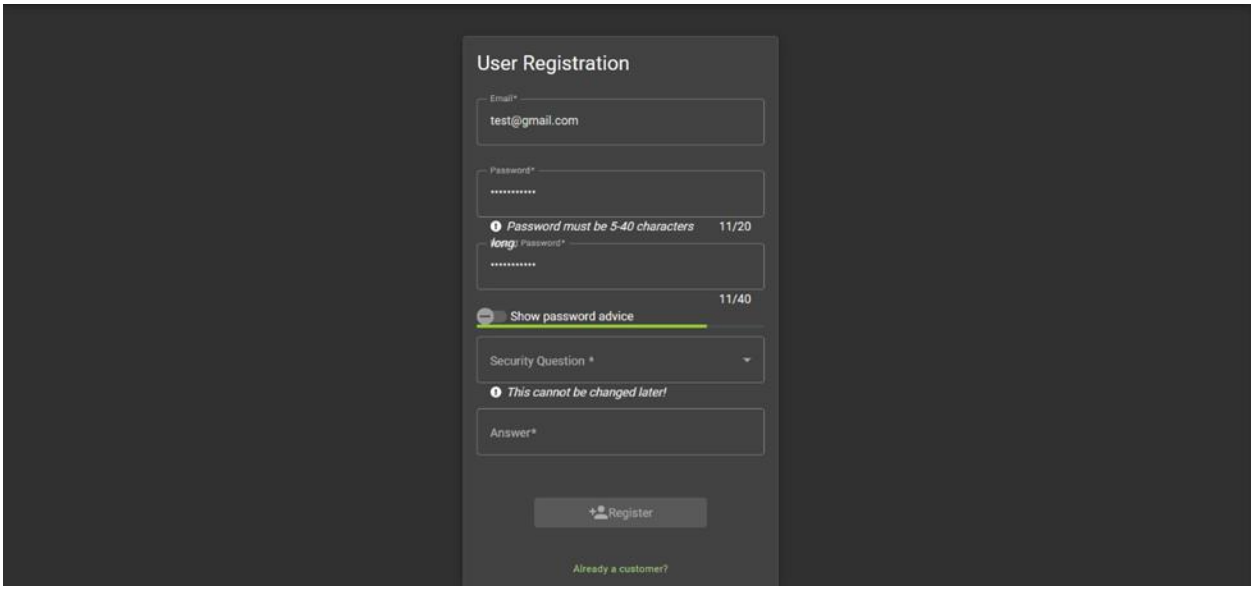
- **Implement Server-Side Validation:**
  Ensure that the server validates that the "Password" and "Repeat Password" fields match before account creation.

Proof of Concept (PoC):

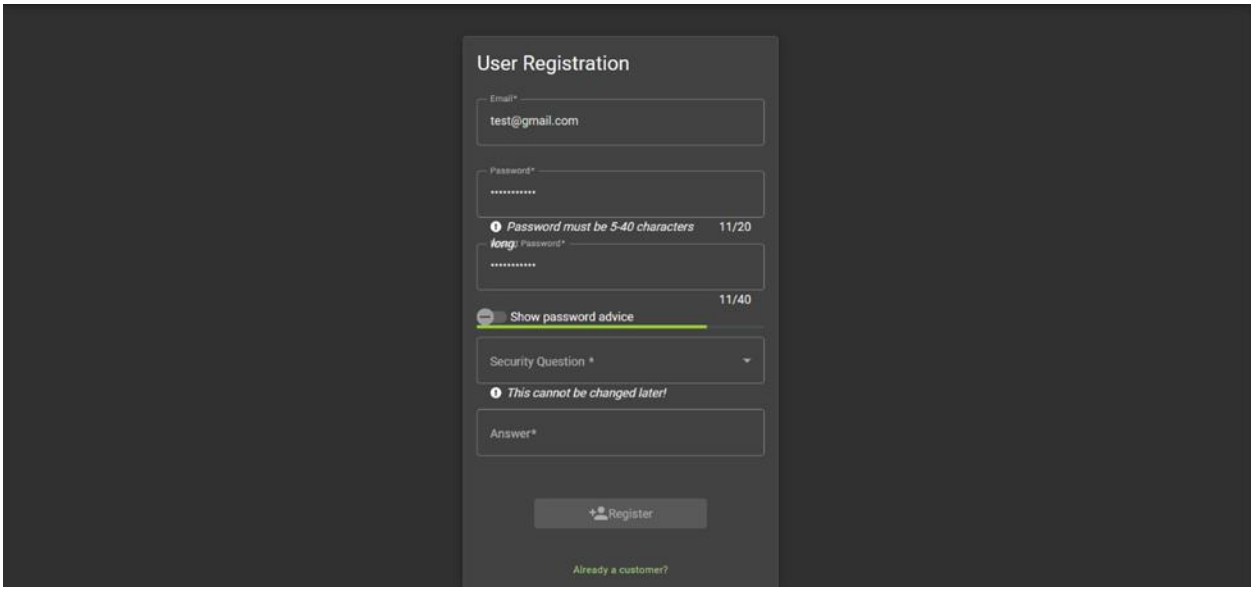
1. Navigate to the Juice Shop website and go to the Register page.
2. Fill out the registration form with any password and repeat password.



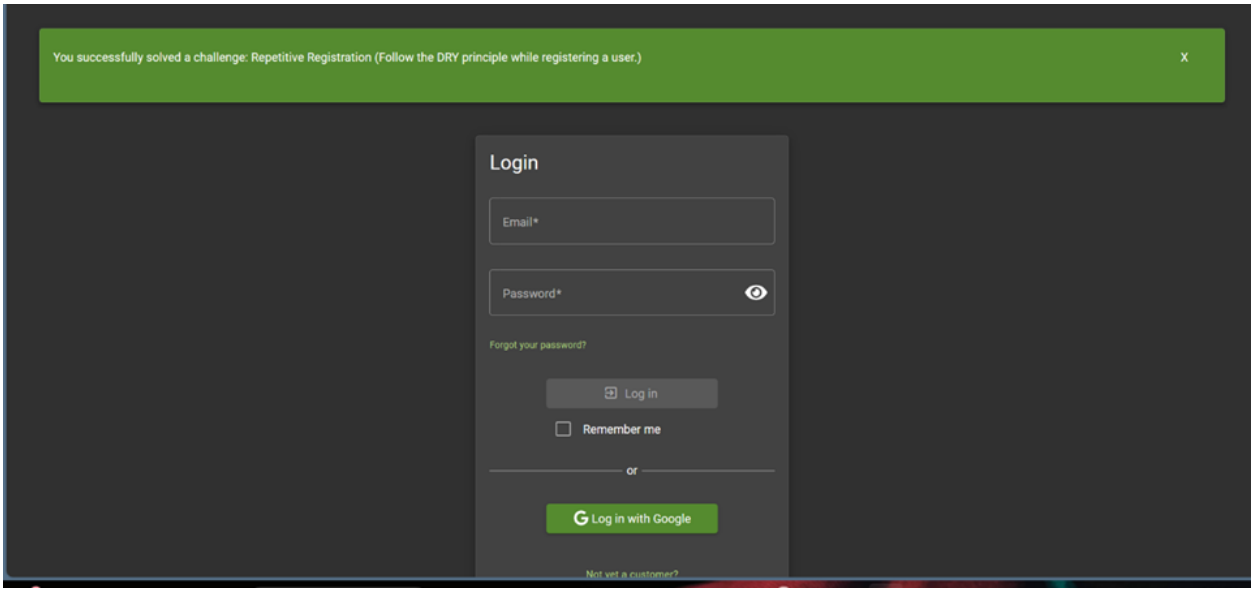
3. In password bar write any password and in repeat password bar write wrong password, it refused



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



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



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:

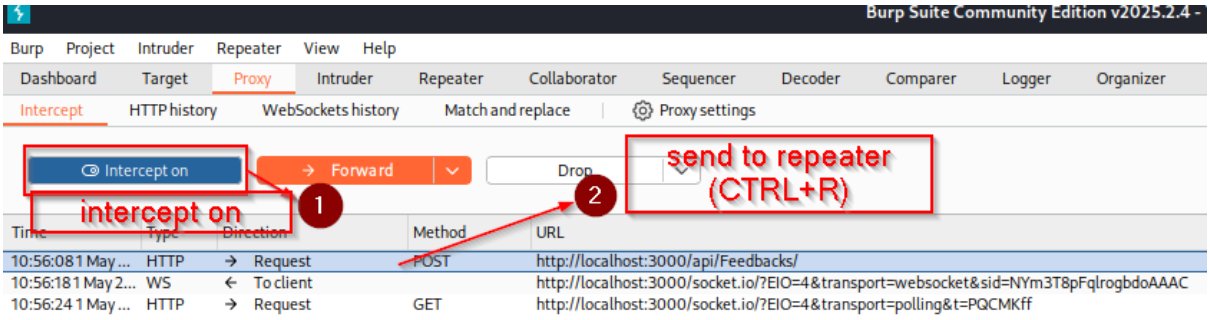
- http://localhost:3000/#/contact

Recommendations:

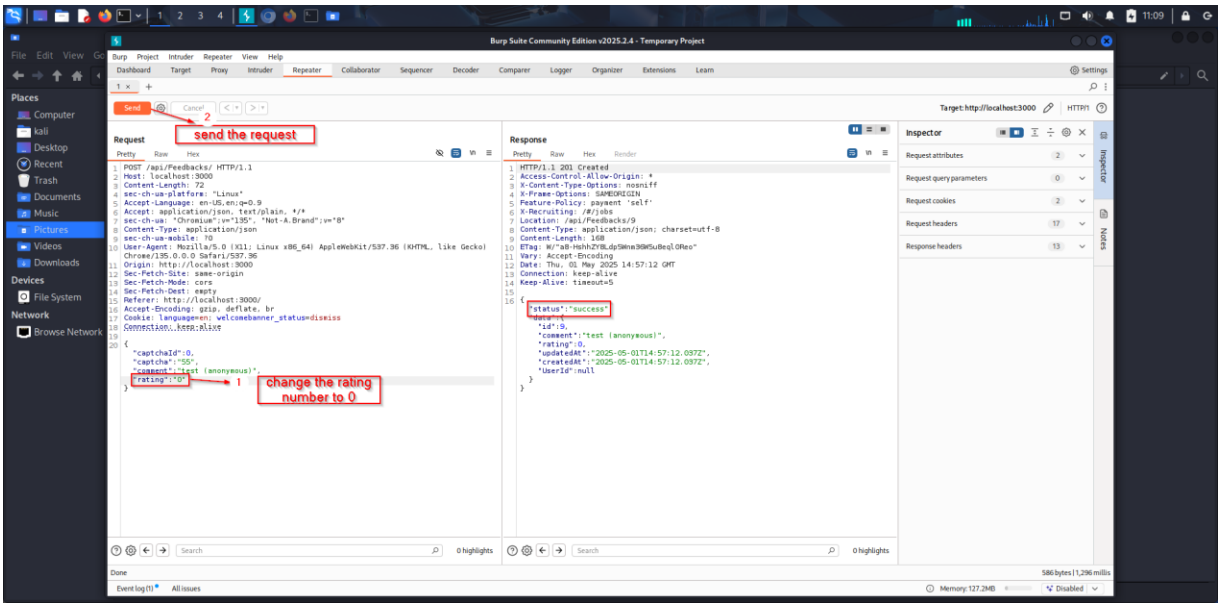
- Enforce rating boundaries (1–5) on the **server side**.
- Validate input against business logic rules before processing feedback.

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



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


send the request
change the rating number to 0

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

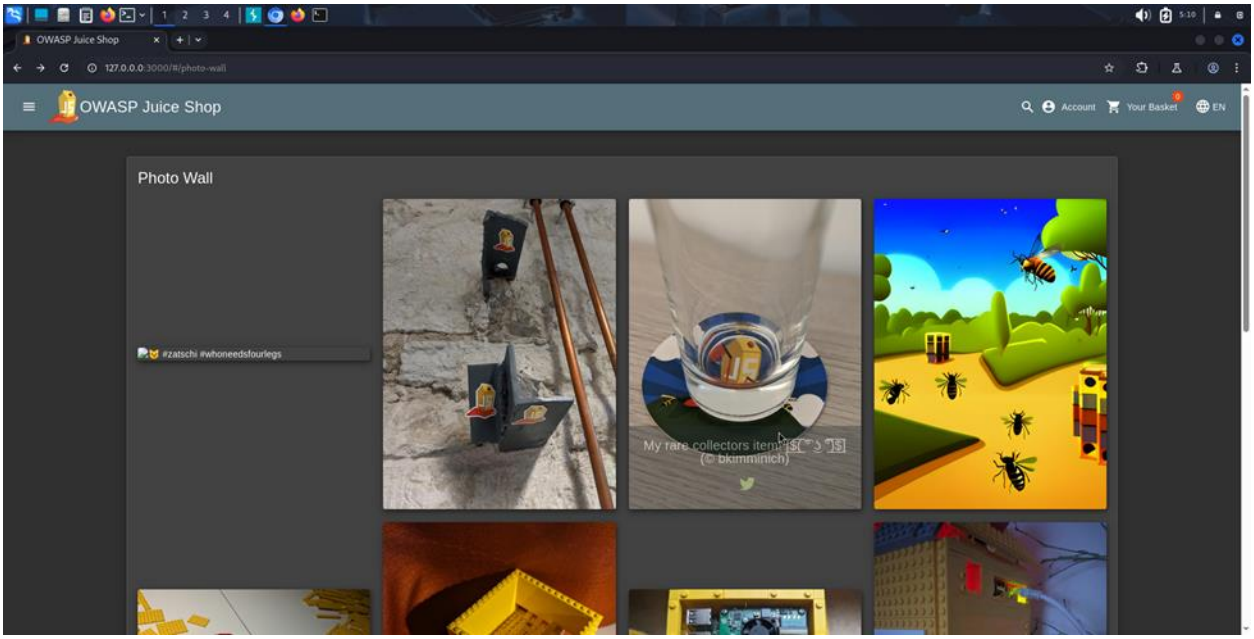
Resources:
https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

Recommendations:

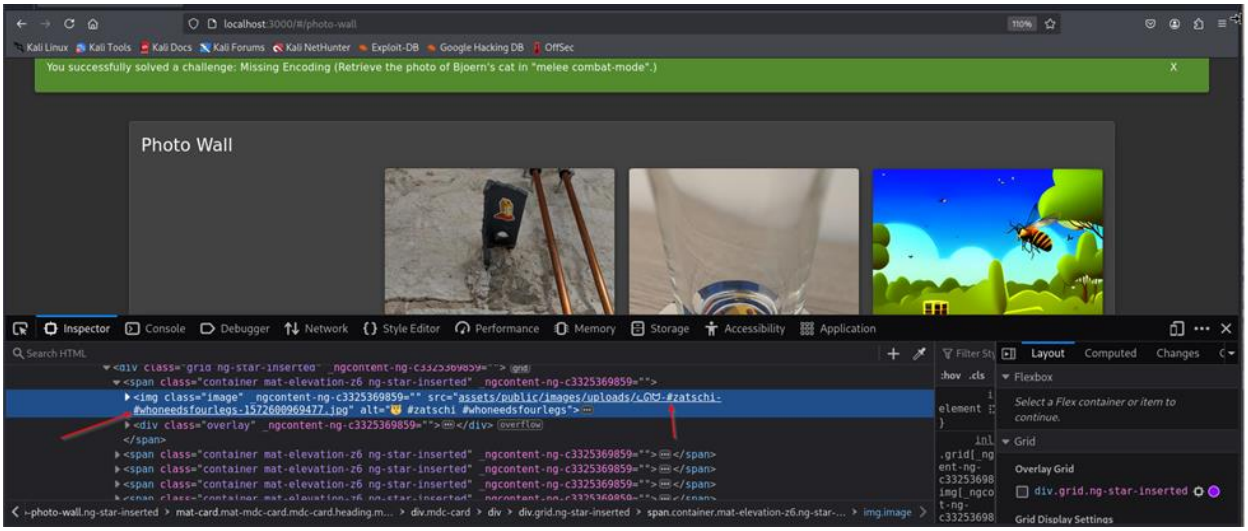
- Always URL-encode file names when generating links.
- Validate and sanitize file names during upload and link generation.
- Avoid using special characters like #, ?, &, and others in file names where possible

Proof of concept(POC):

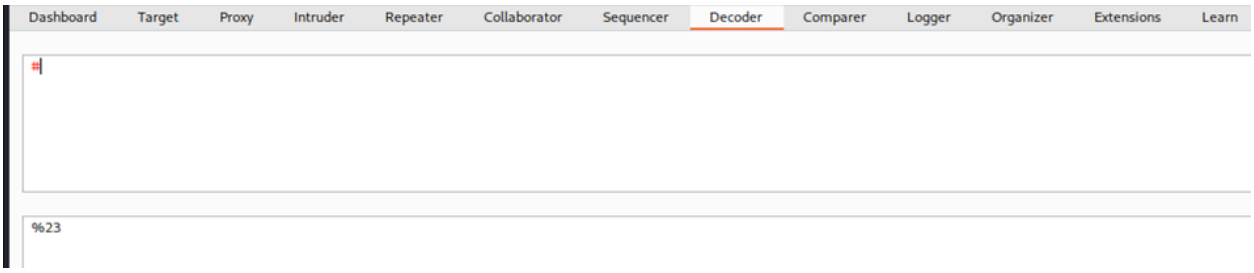
1. **First, the image cannot appear**



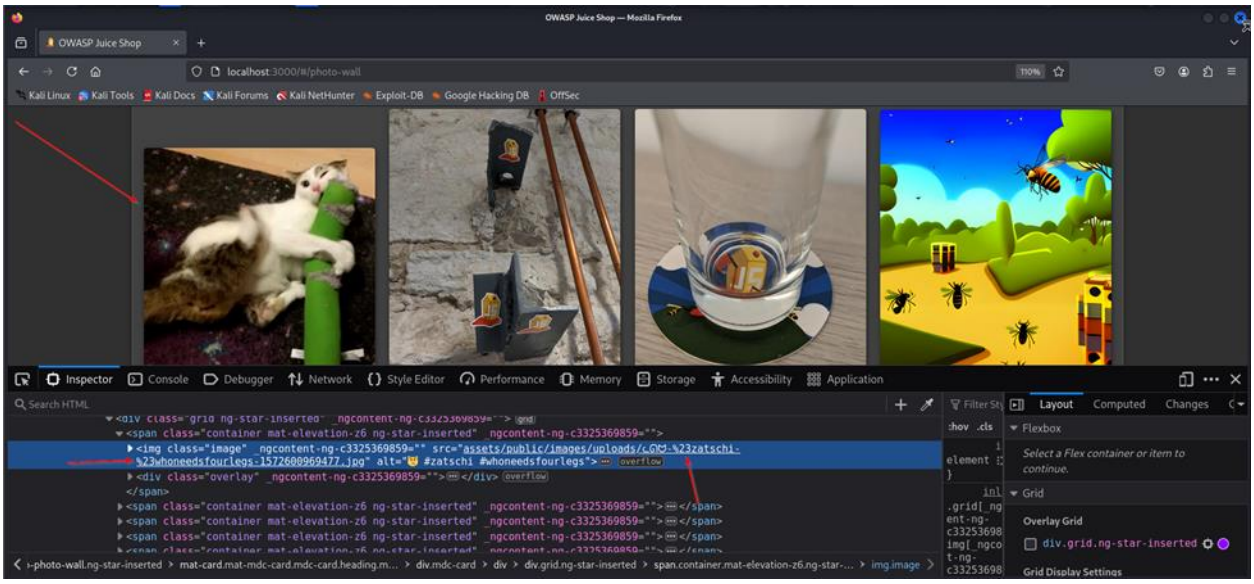
2. Inspect source Code

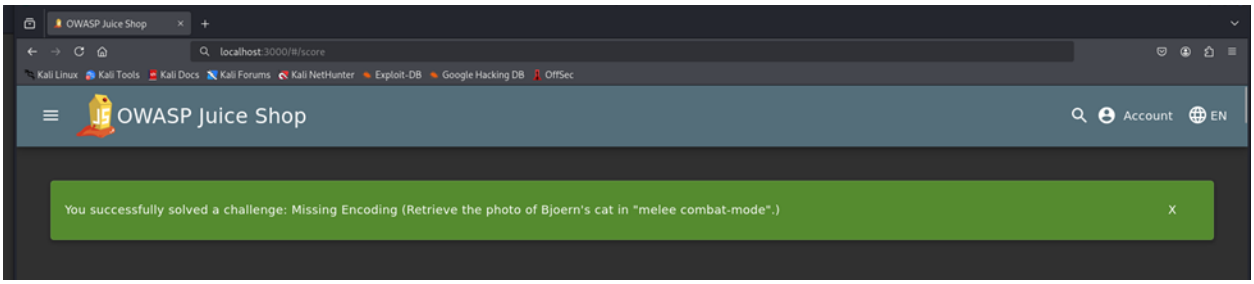


3. change all(#) with its url encoded which is %23



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





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**

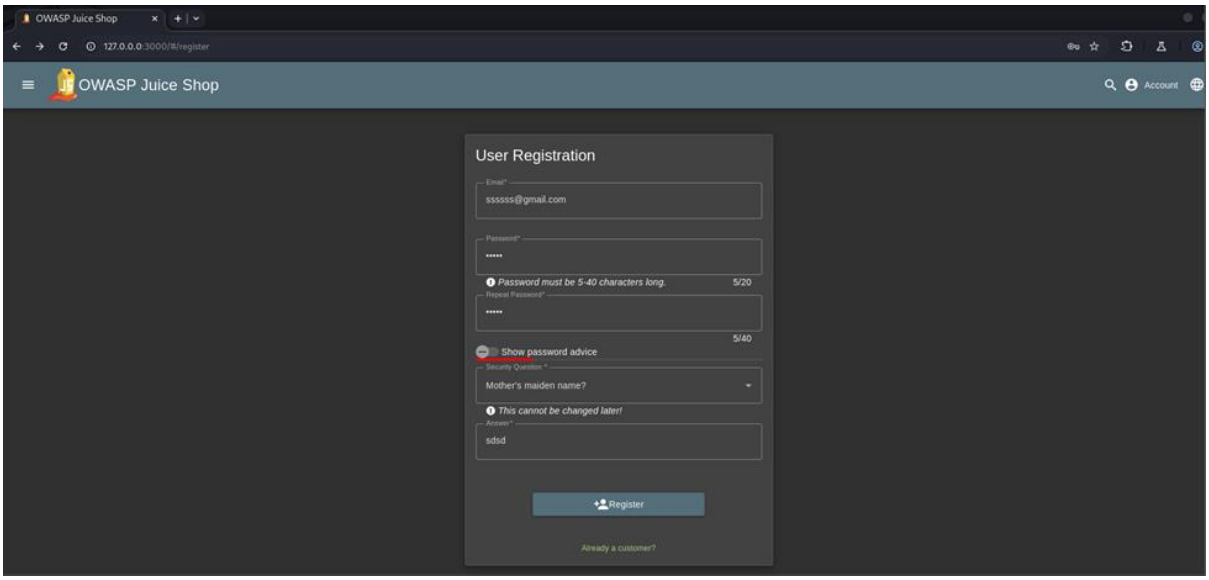
Resources:
https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

Recommendations:

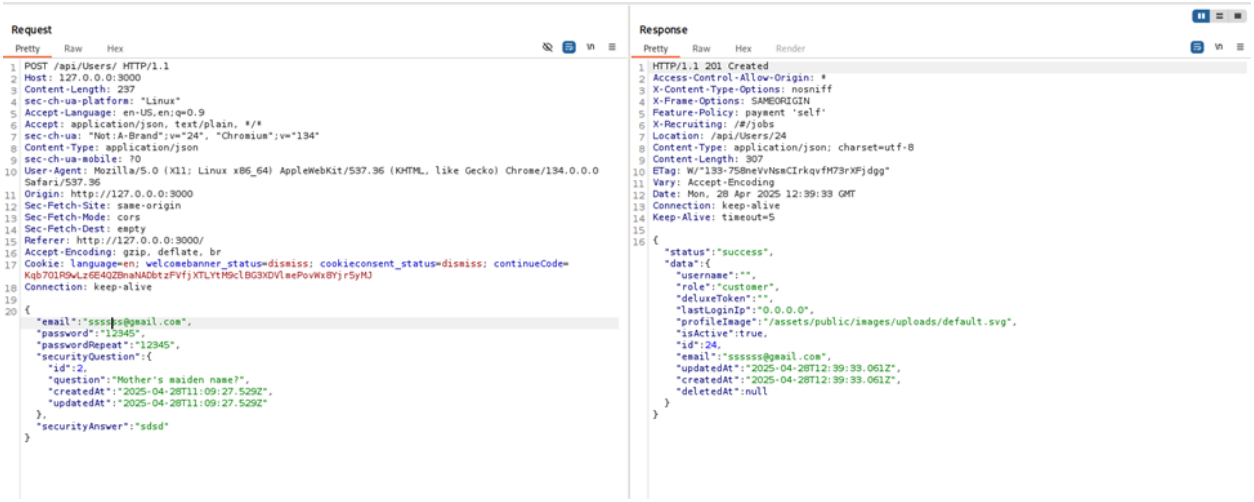
Implement strict server-side validation to ensure fields (password and email) are never null, empty, or invalid

Proof Of Concept (POC)

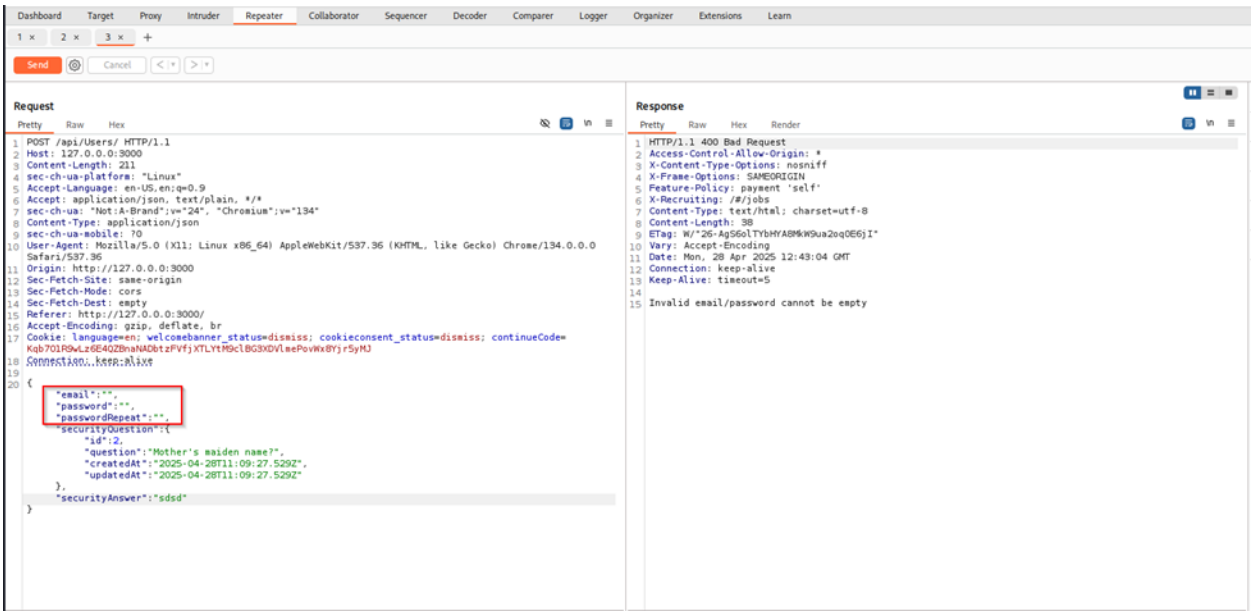
1. Register with a valid account and inspect the packet

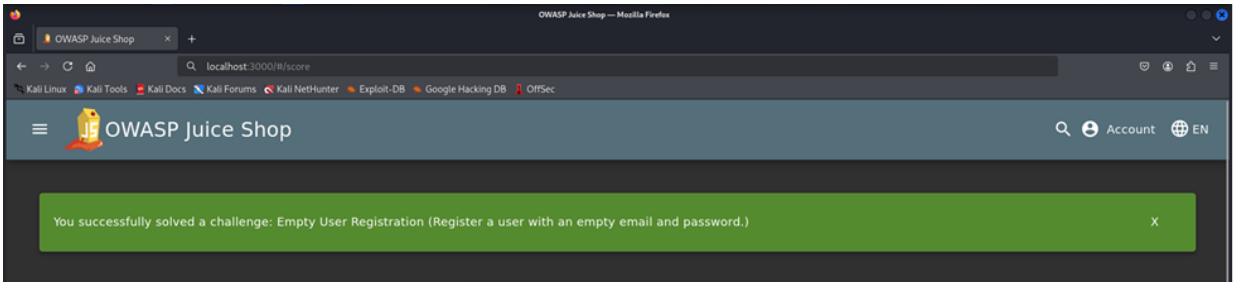


2. Open the request in the repeater



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





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

Resources: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

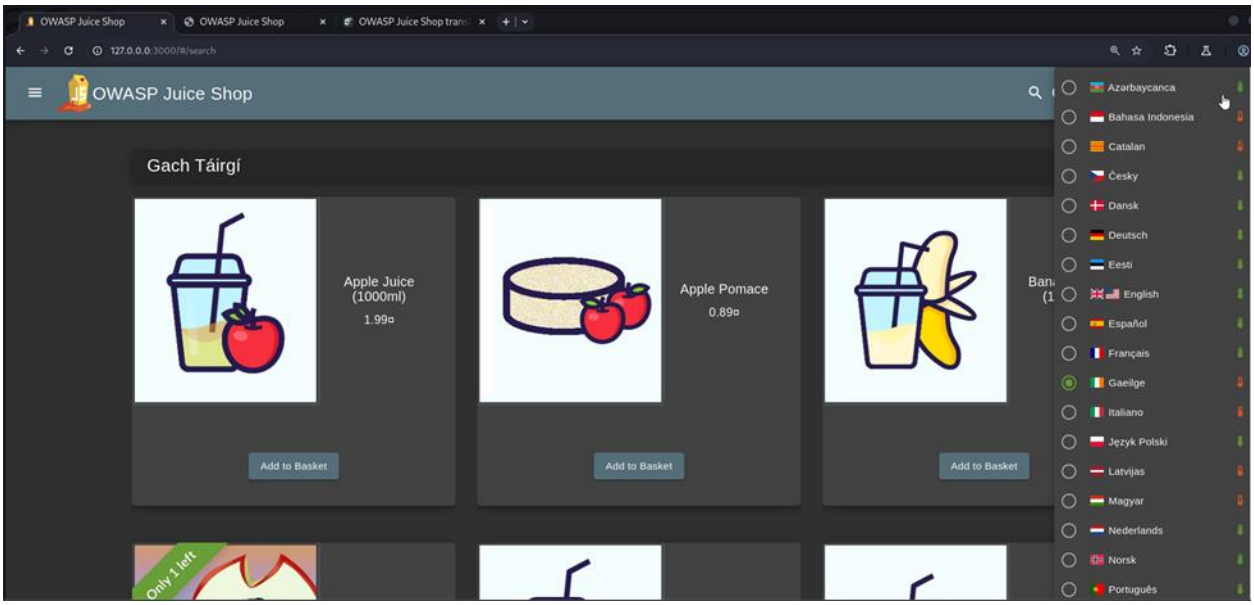
Recommendations:

· Restrict access to unused or hidden translation files by validating language codes against an allowed list.

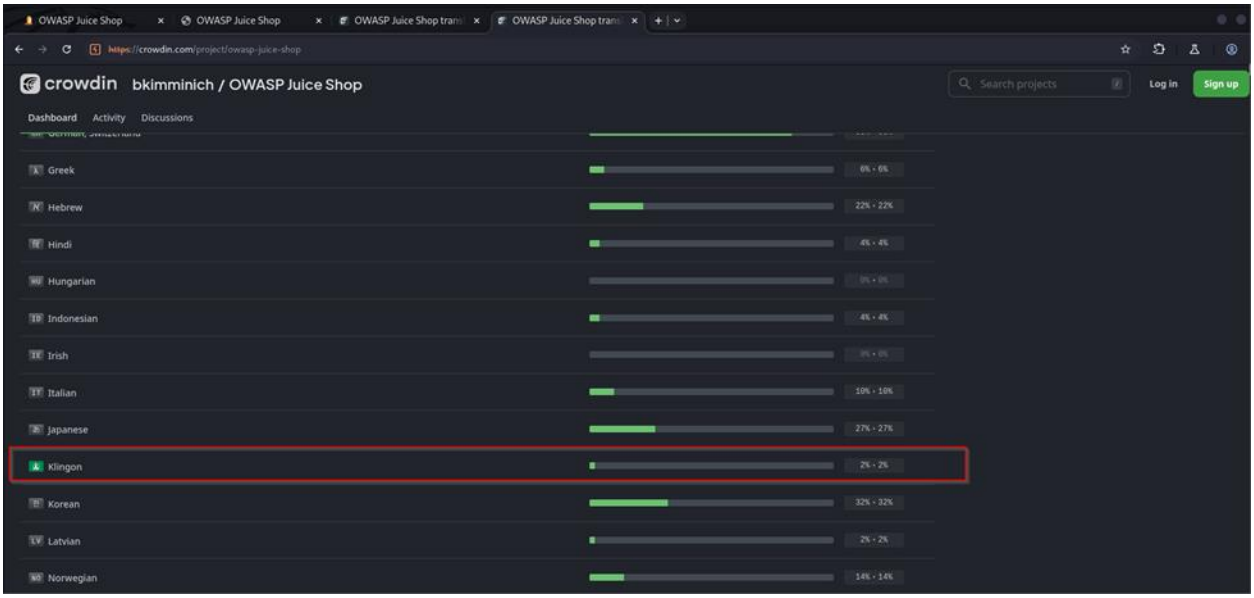
· Synchronize UI and backend languages to avoid confusion or potential abuse.

Proof Of Concept (POC)

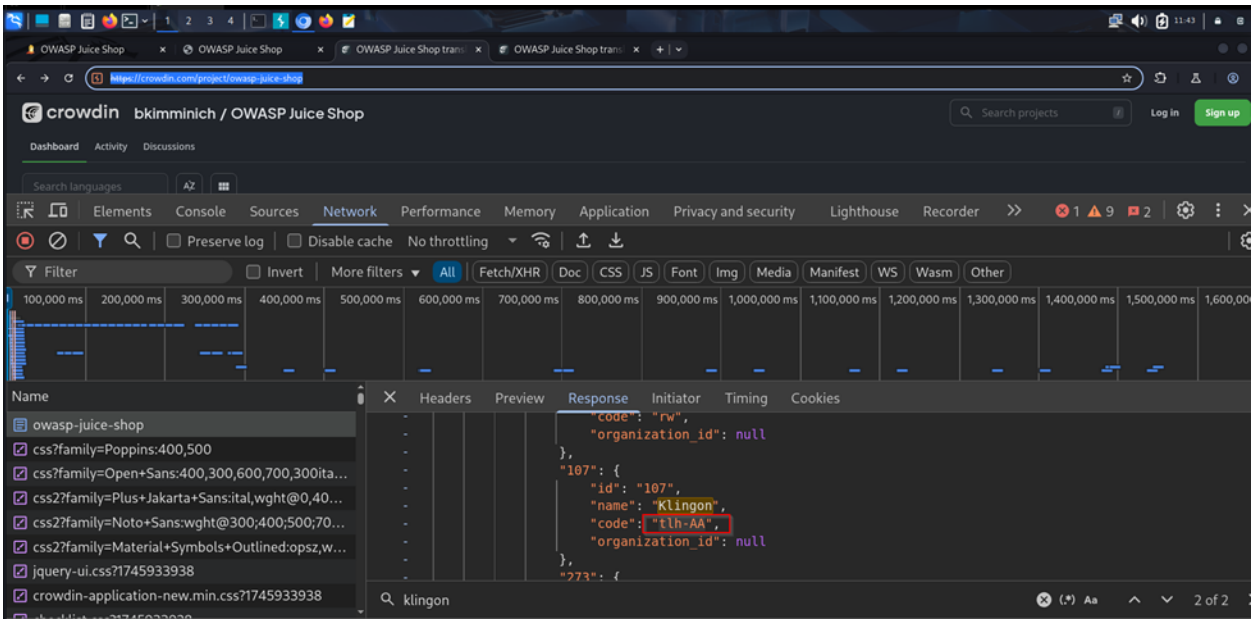
1. Show all languages that appear on the website



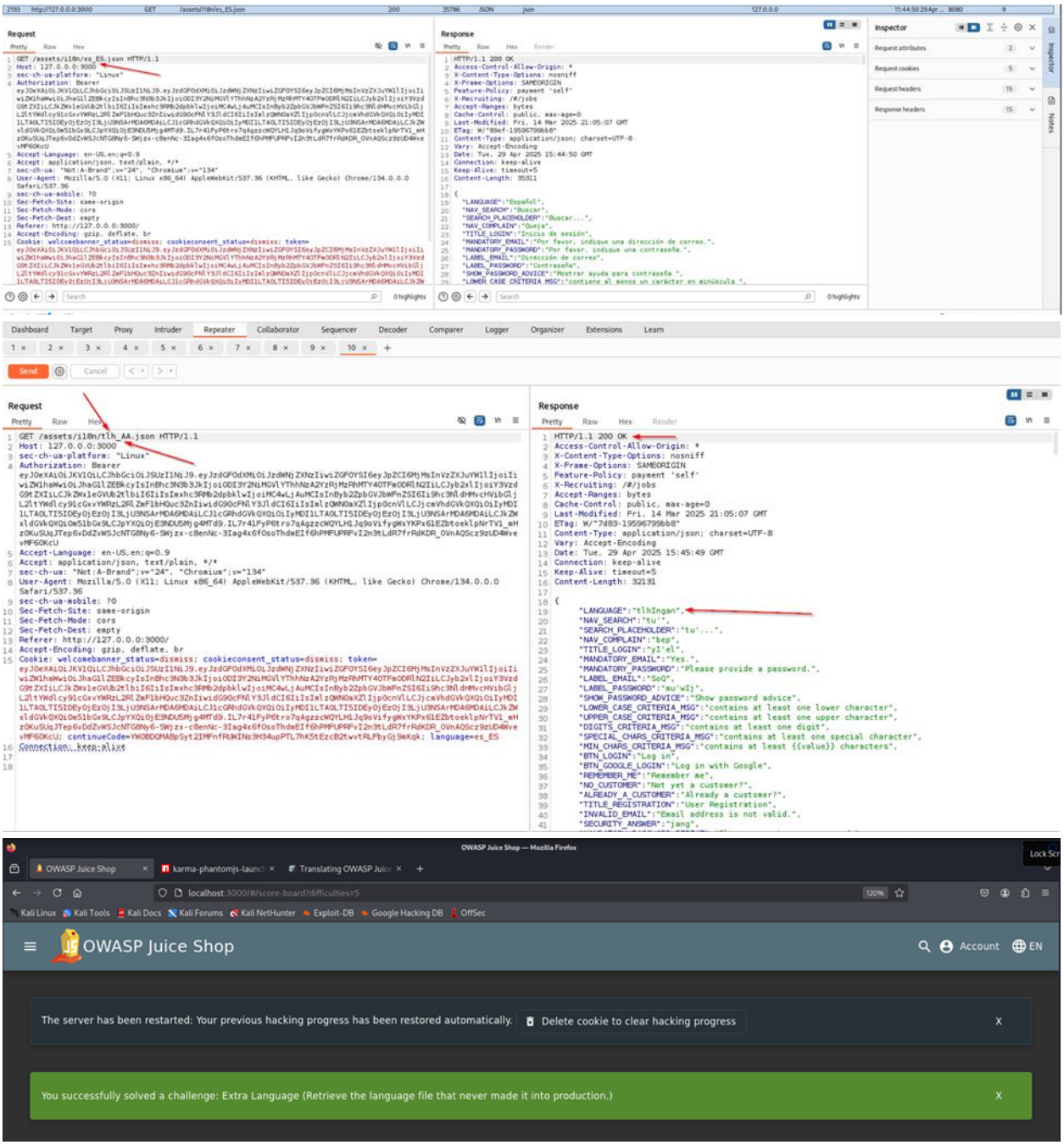
2. Search for languages juice shop support and find one that does not appear in the list on the website.



3. Get the code for this language.



4. Put it in the request line.



9.2 Email leak

Medium

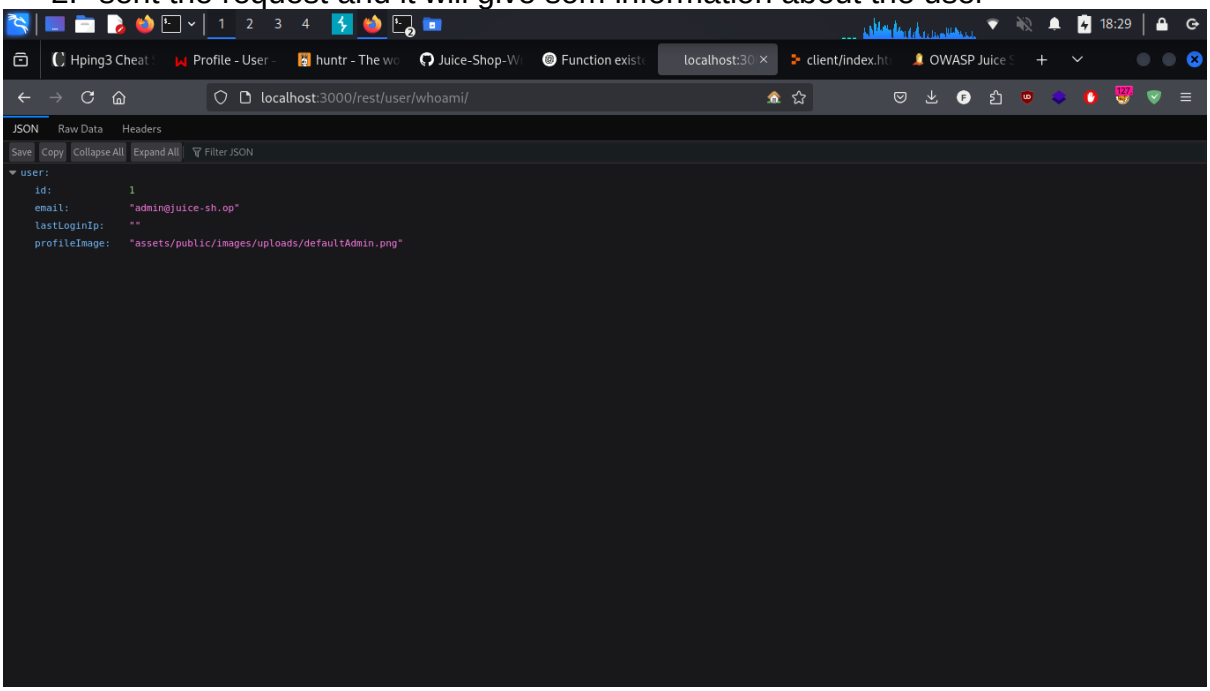
Description:
The OWASP Juice Shop application exposes a sensitive API endpoint /rest/user/whoami that leaks user data through an improperly secured JSONP implementation. The endpoint responds with user-related information without enforcing authentication, and supports JSONP-style callbacks, allowing 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



9.3 File Upload Size Restriction

Low

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.

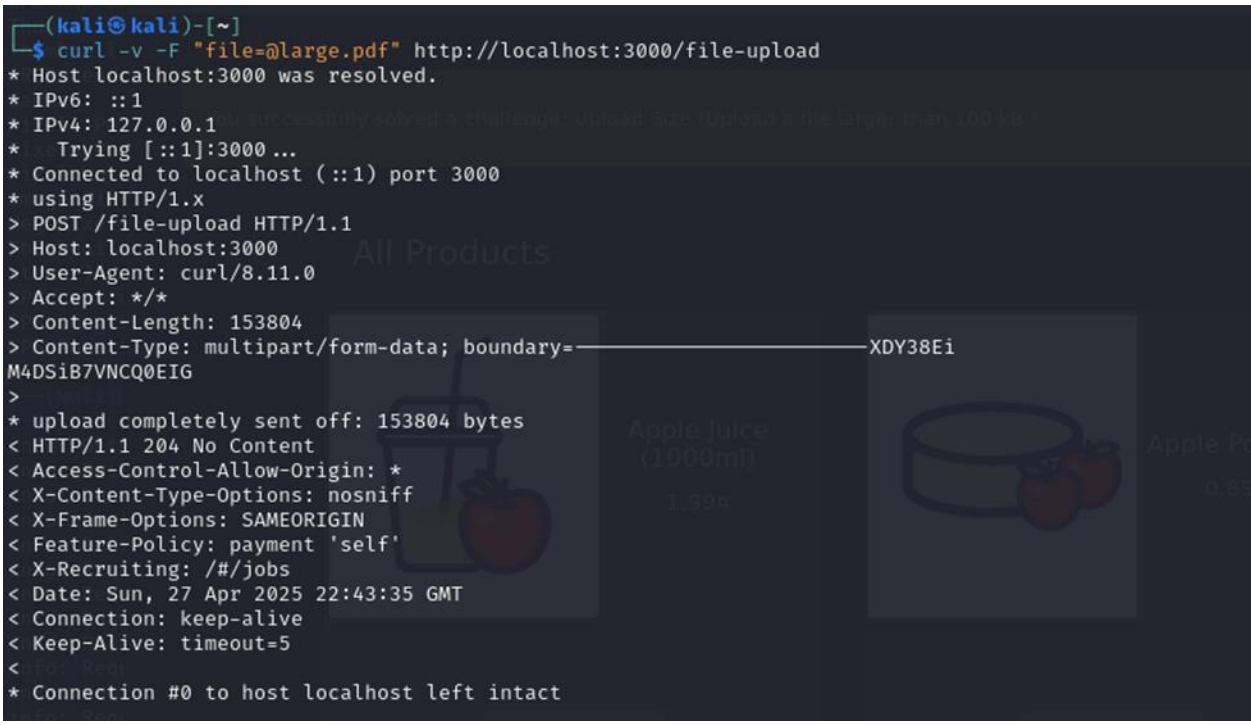
Proof of Concept (PoC):

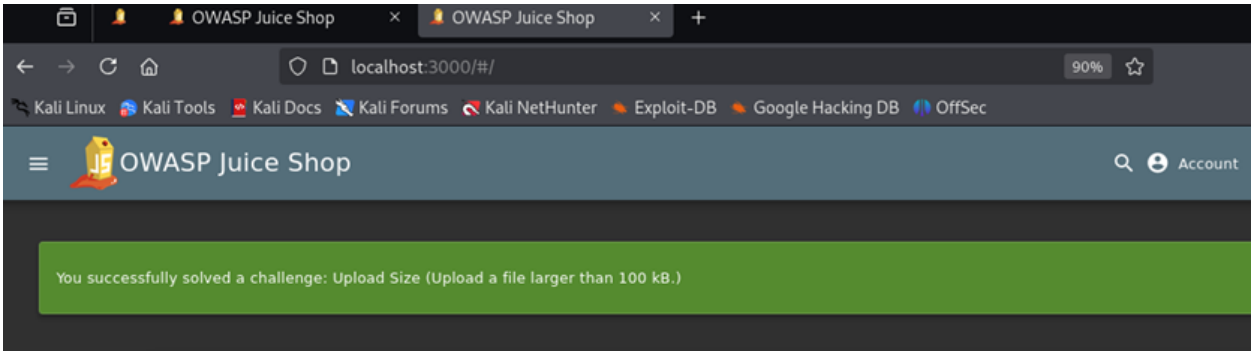
1. Create a .pdf file with a size exceeding 100 KB (e.g., large.pdf ~157 KB).

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.





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

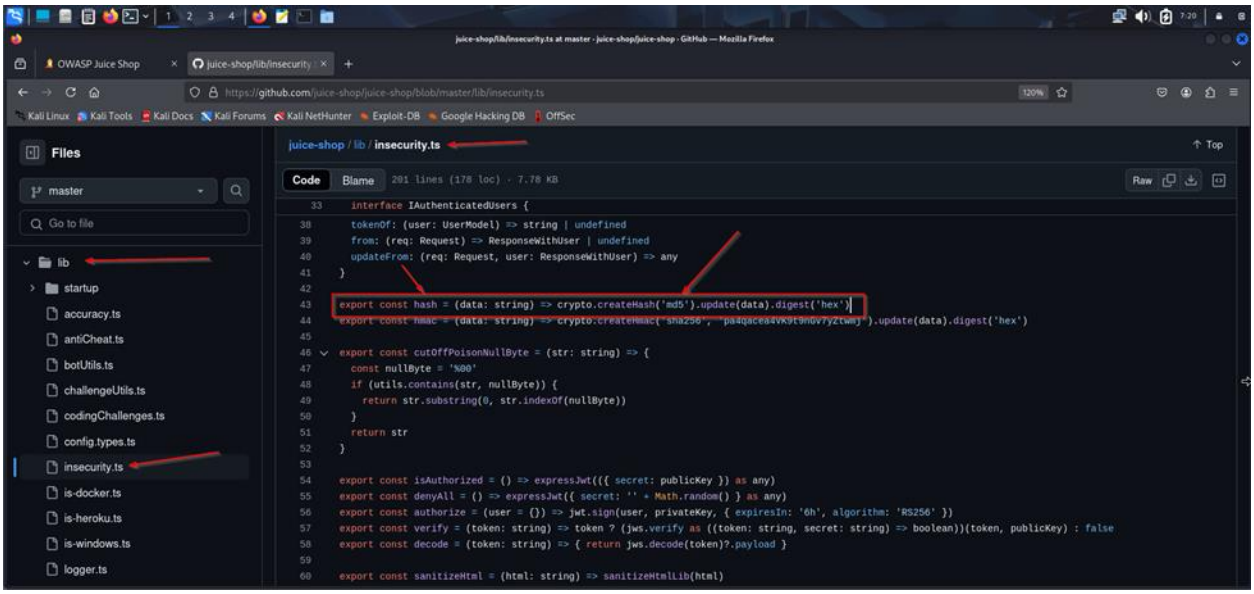
Resources: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/04-Testing_for_Weak_Encryption

Recommendations :

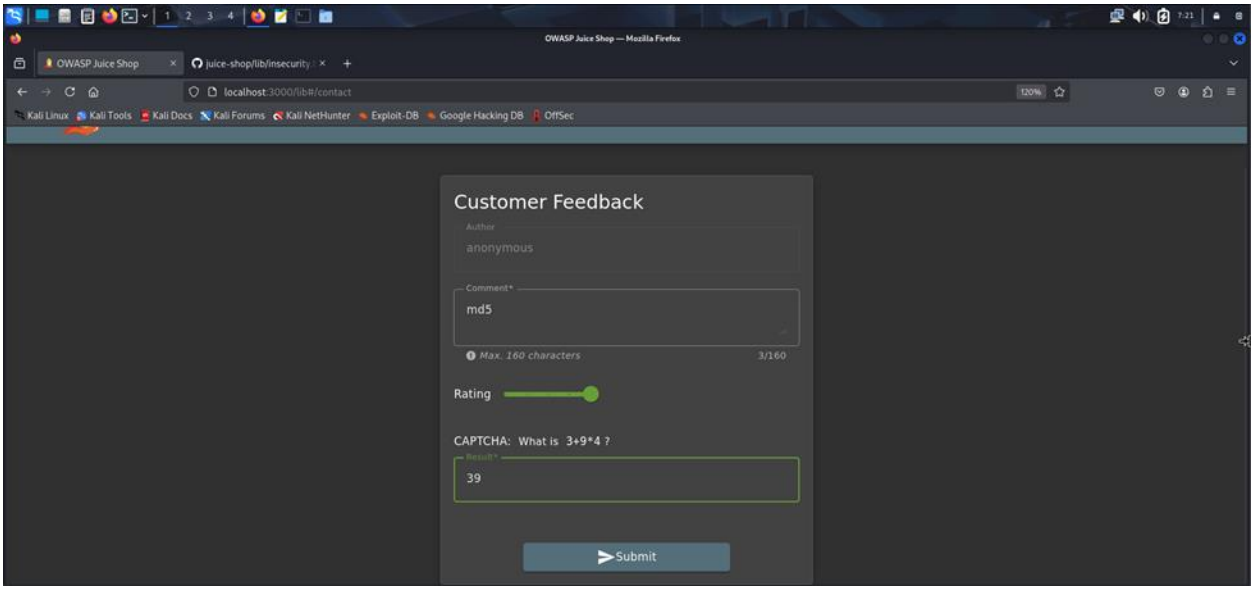
1. **Use Strong Hashing Algorithms: Instead of MD5, use modern and secure hashing algorithms, such as SHA-256 or bcrypt, for password hashing. These algorithms offer stronger cryptographic properties and resistance to brute force and collision attacks.**

Proof Of Concept (POC)

1. Go to the targeted files



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



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 :

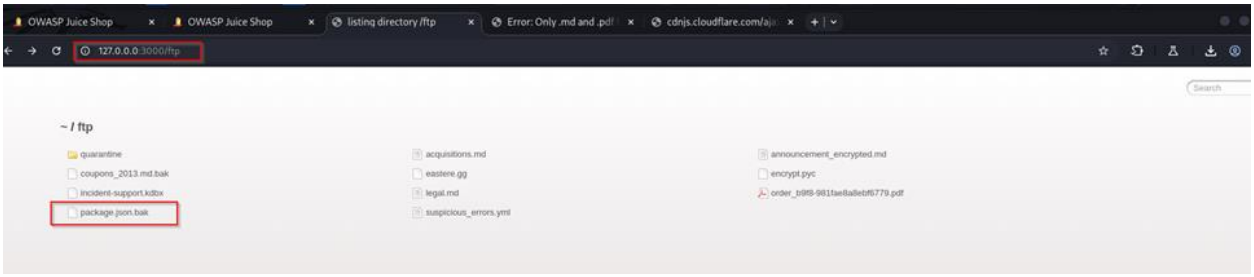
1. Remove the malicious epilogue-js package and replace it with the correct epilogue.
2. Validate all file downloads/extensions securely — avoid null byte parsing issues

3. Regularly review and pin dependency versions to avoid silent updates.

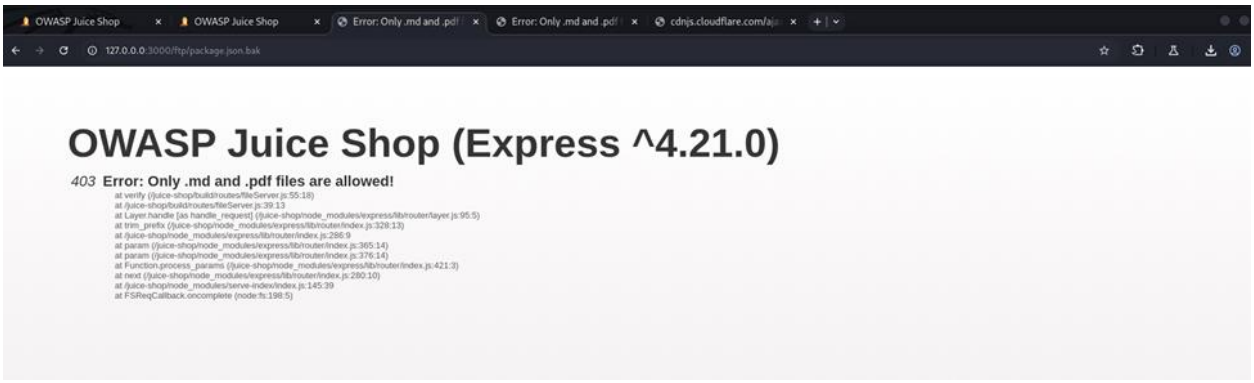
4. Educate developers about supply chain attacks and secure package selection.

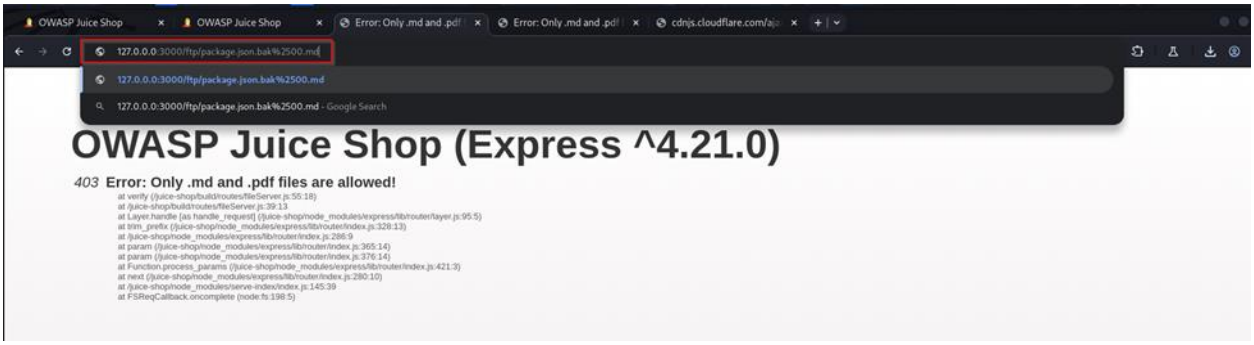
Proof Of Concept (POC):

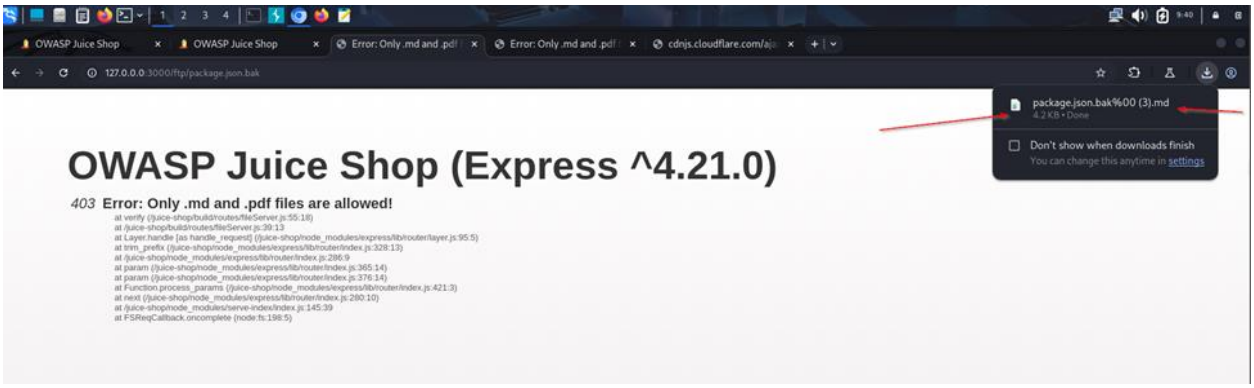
1. Go to: http://127.0.0.1:3000/ftp



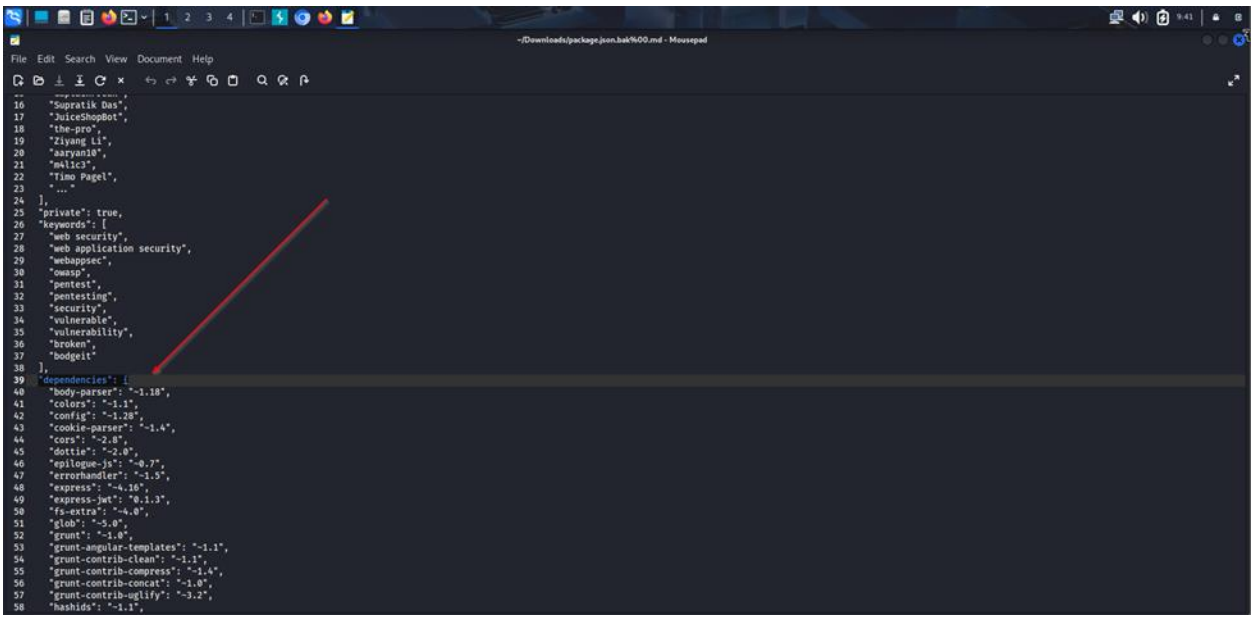
2. Bypass null byte to read the file

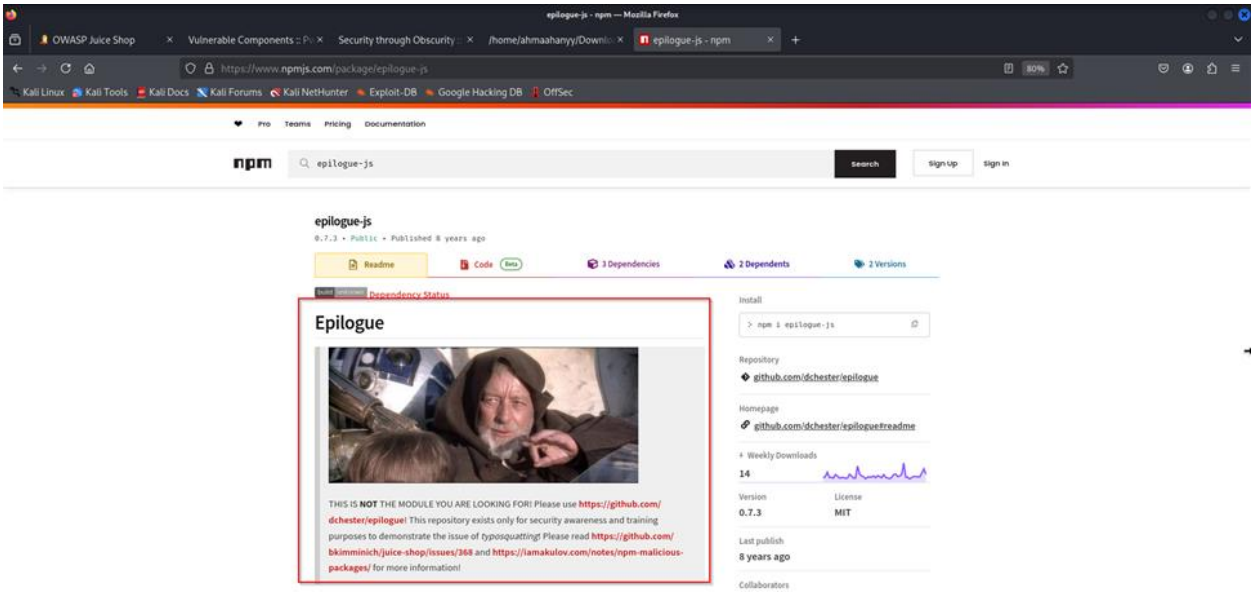




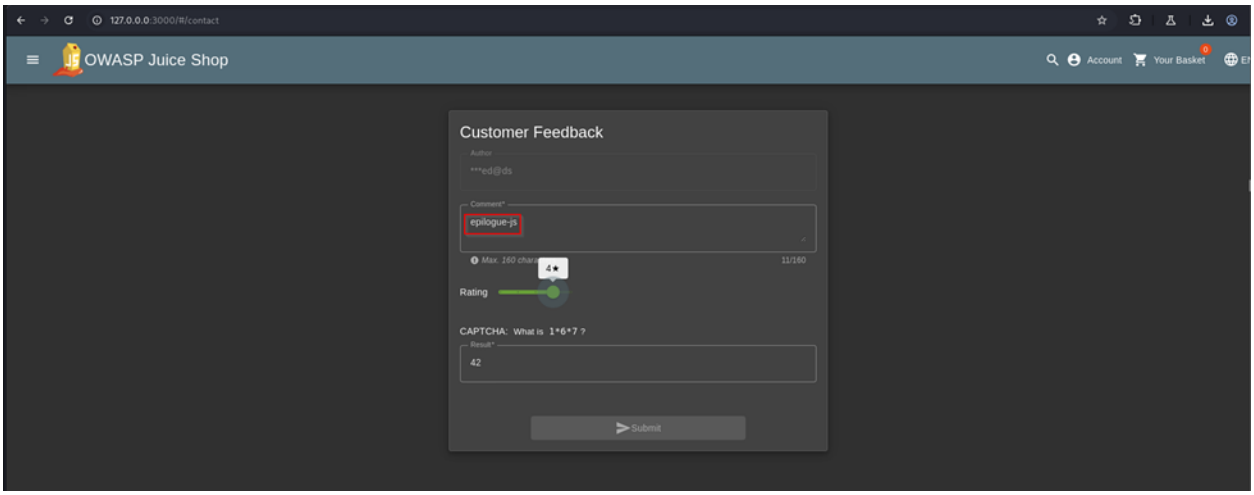


3. Search in dependencies for the wrong library





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**:

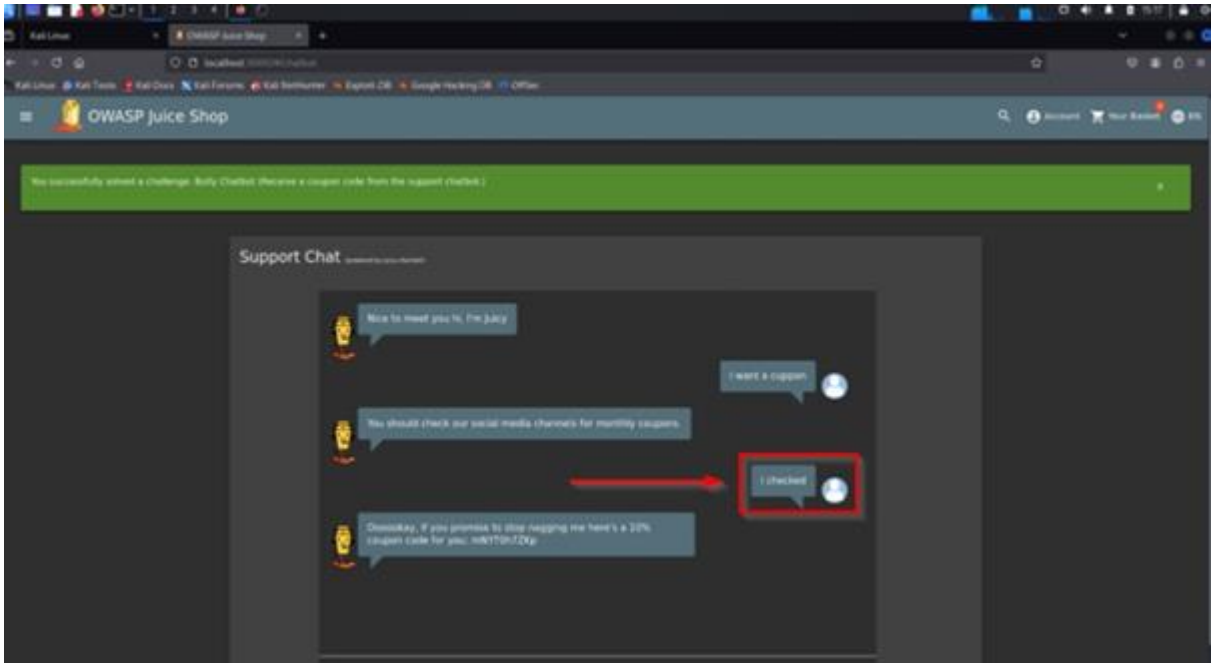
   Implement rate-limiting for coupon requests to prevent abuse (e.g., by restricting how often a user can request a coupon in a certain timeframe).

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:

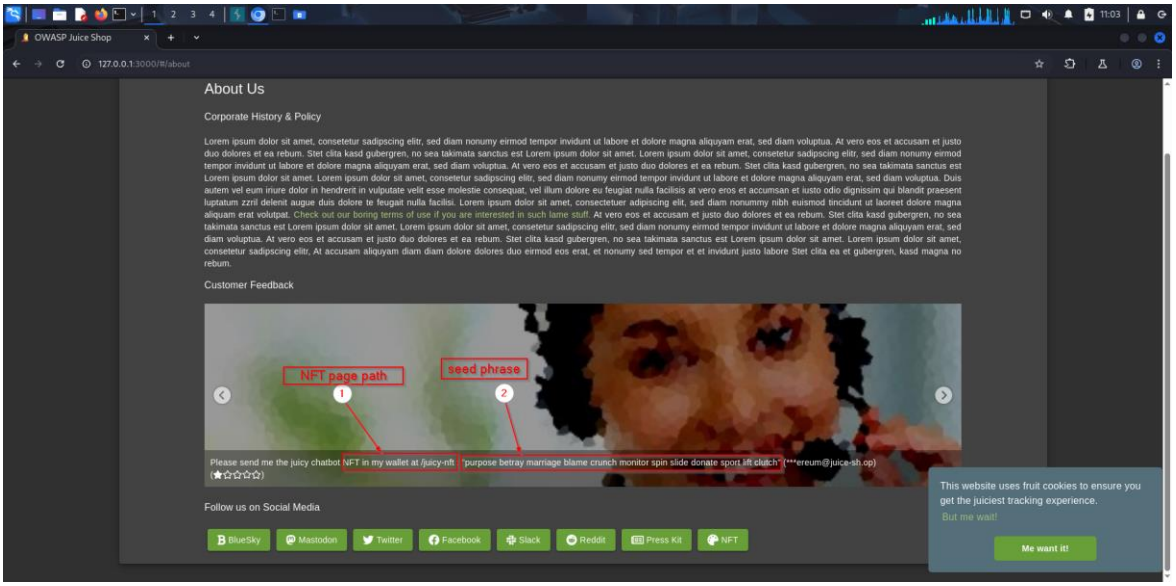
- http://127.0.0.1:3000/#/juicy-nft

Recommendations:

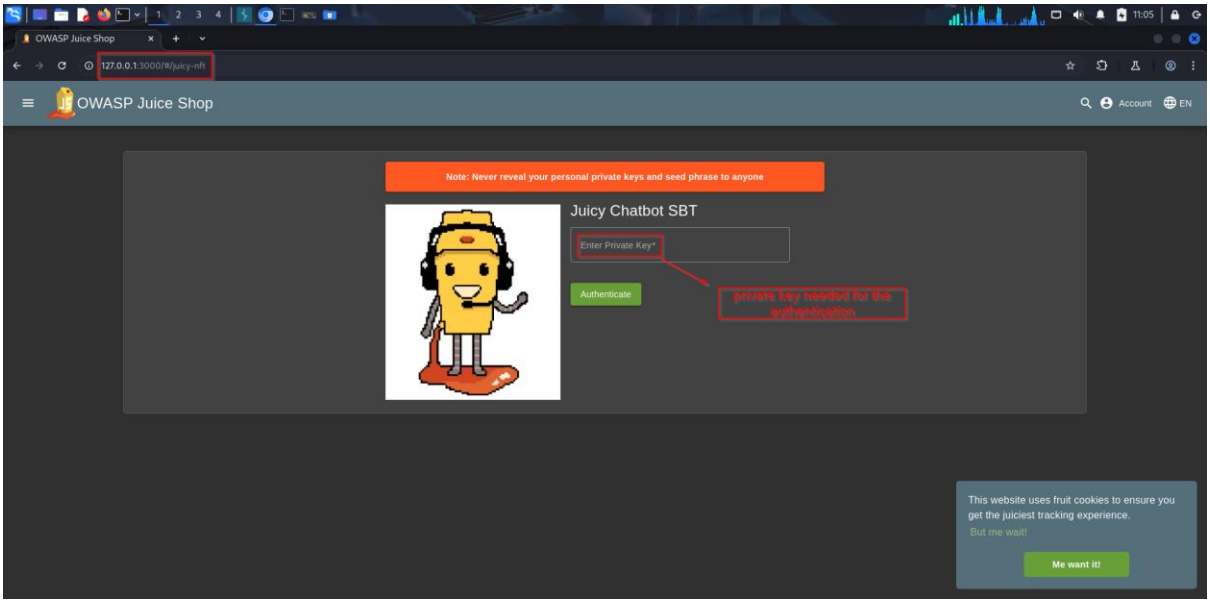
- Secure Information Handling — Ensure that any form of sensitive information, such as cryptographic keys or seed phrases, is neither stored or transmitted in clear text within user-accessible areas.
- Input Validation and Sanitization — Implement stringent input validation and sanitization on all user-submitted 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.



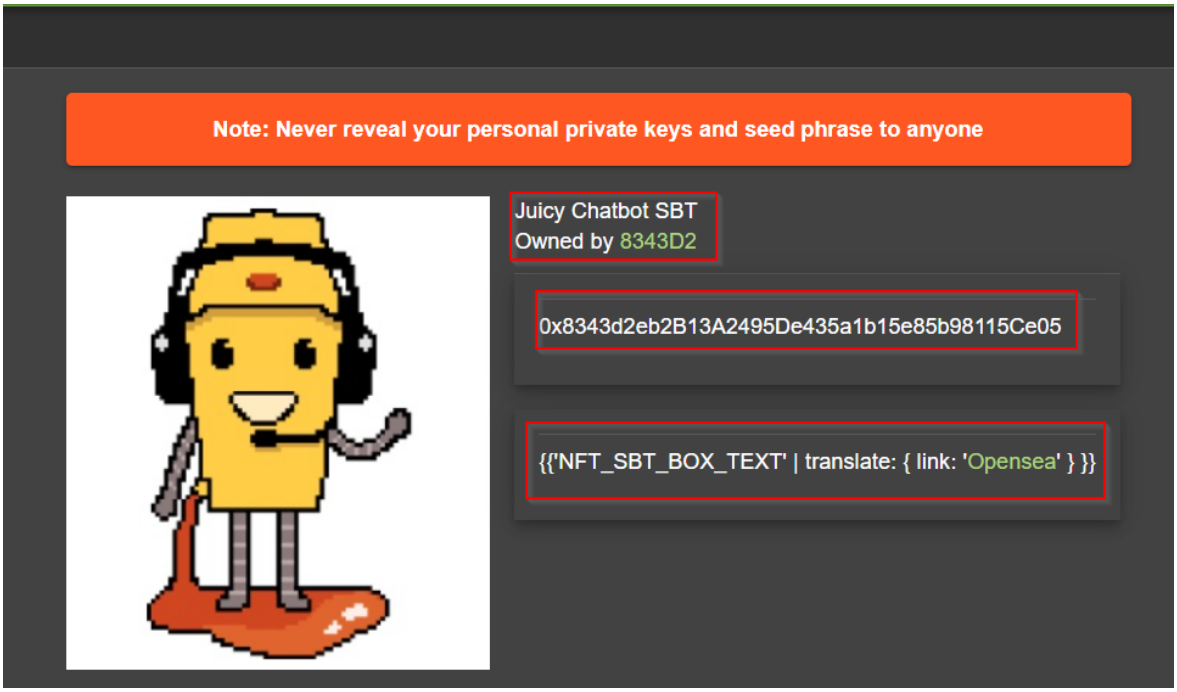
2. Now we can go to the nft page.



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

BIP39 Mnemonic	purpose betray marriage blame crunch monitor spin slide donate sport lift clutch
	☐ Show split mnemonic cards
BIP39 Passphrase (optional)	
BIP39 Seed	552b89904540a9d8751f1c7e31f71feb584bb62af857fbfb65bcb8e48c80dcb8654614379a2a1e294f759134c0008beeee778fb353f98e15edf3adad2a728e17
Coin	ETH - Ethereum
BIP32 Root Key	xprv9s21ZrQH143K4DfTxz9Ygc6kvSBEV8LgZPk7BcXzJzT49gj6VoY5xqD21Q9jnyZQXaeWqp7wRs44vbeWU1FwRzbXFazix1hc7qFhSoyD6ub

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.

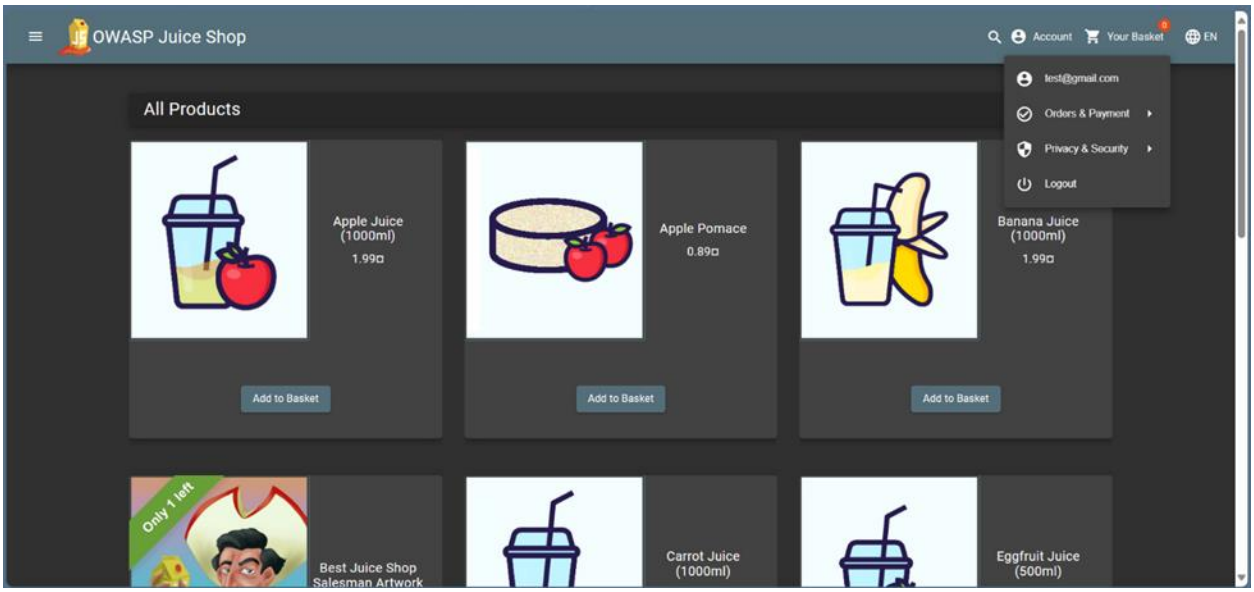
- **Perform Regular Access Control Audits:**

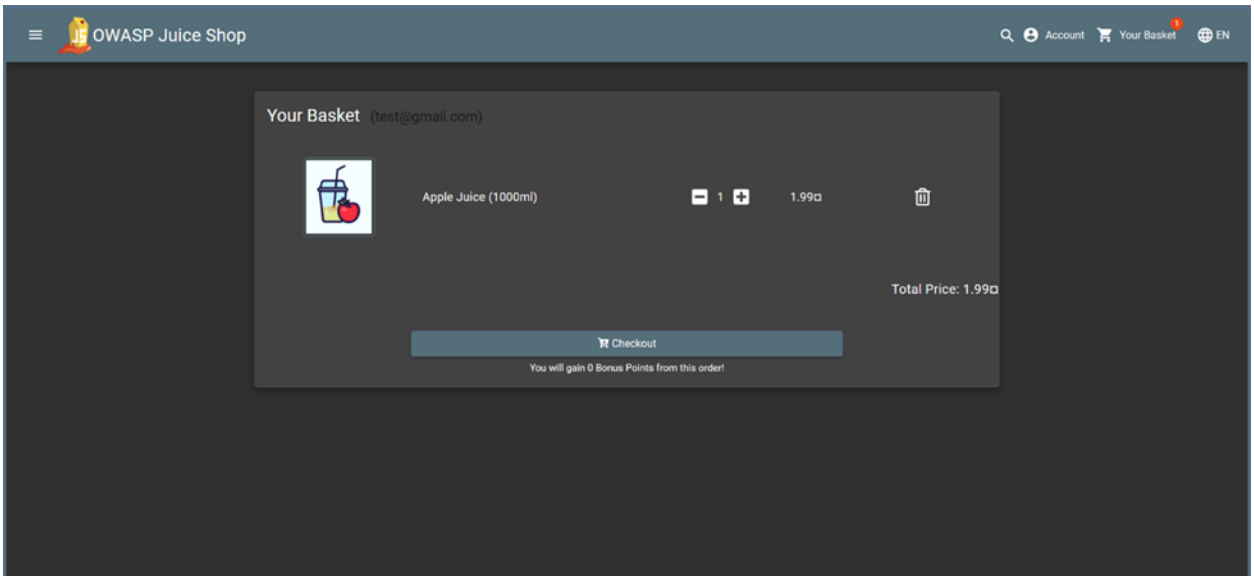
Ensure all user-level resources are properly restricted through role and identity checks.

Proof of Concept (PoC):

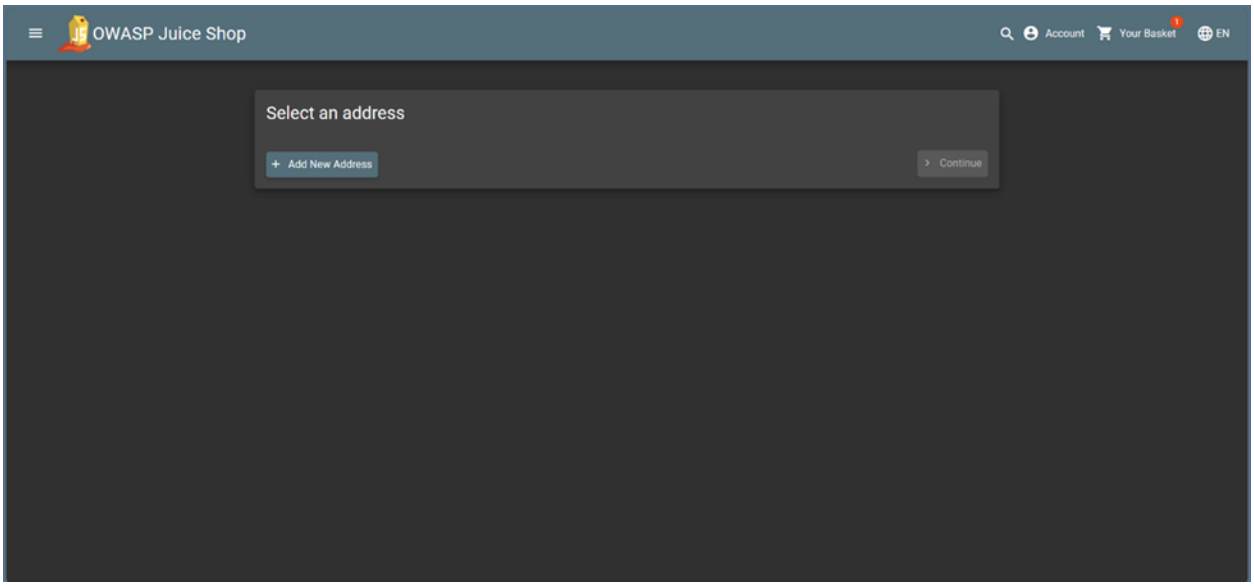
1. **Log in as an existing user:**

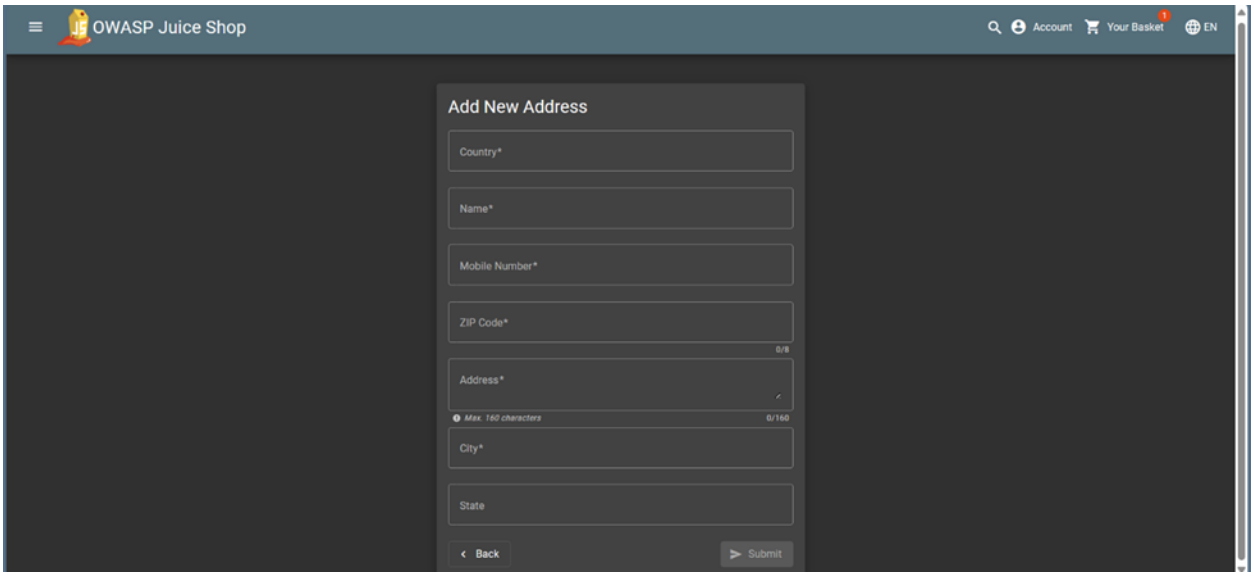
1.1 Add items to the basket and complete an order.

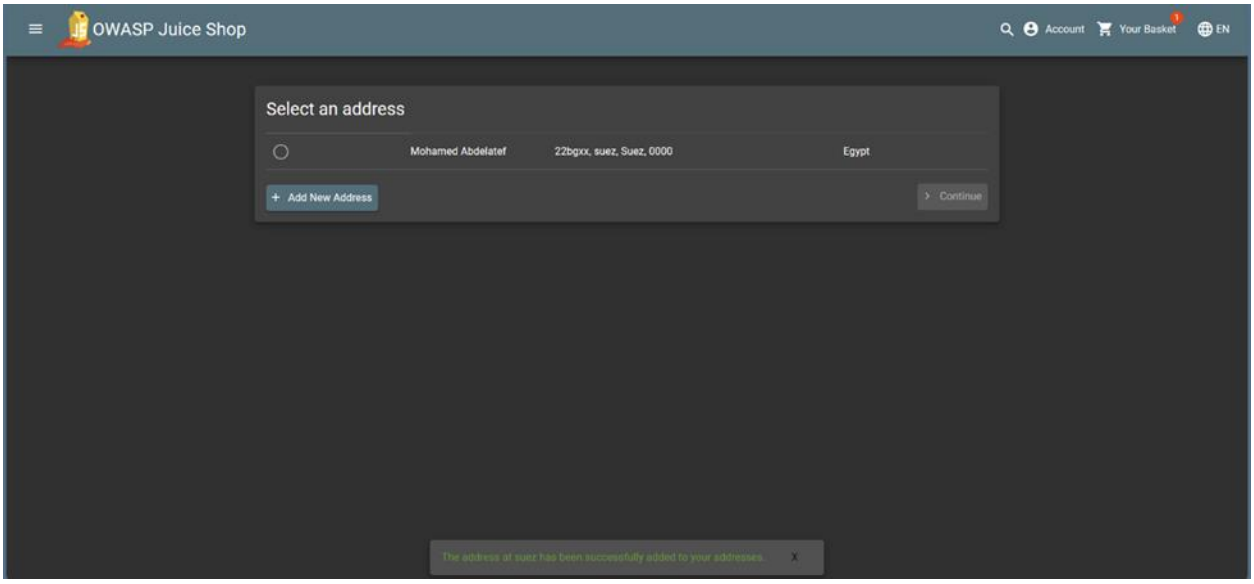


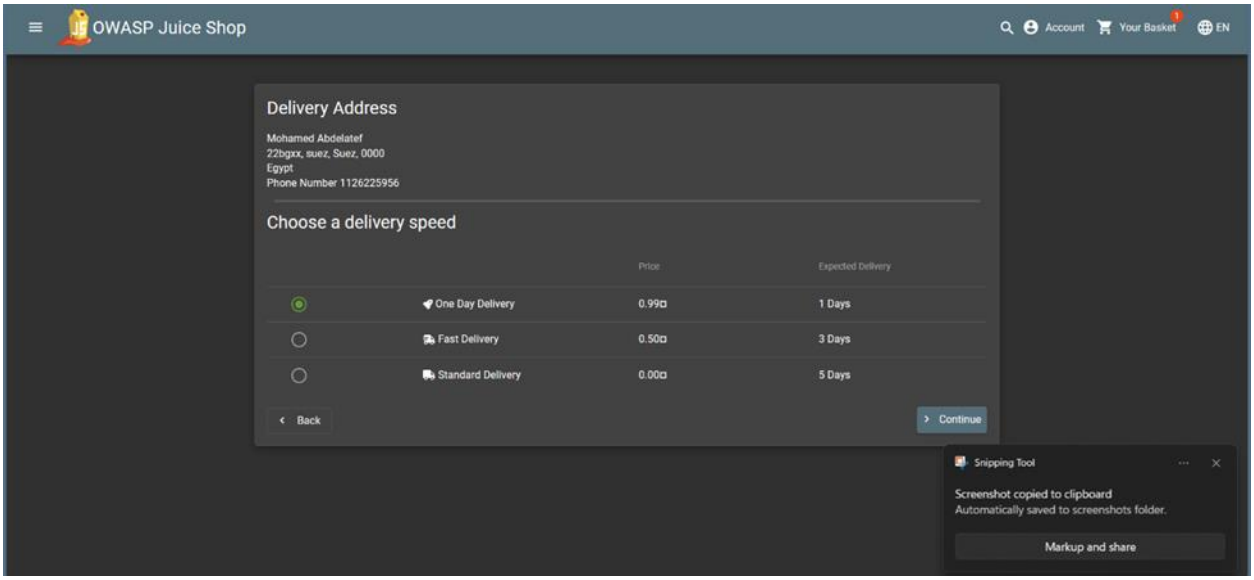


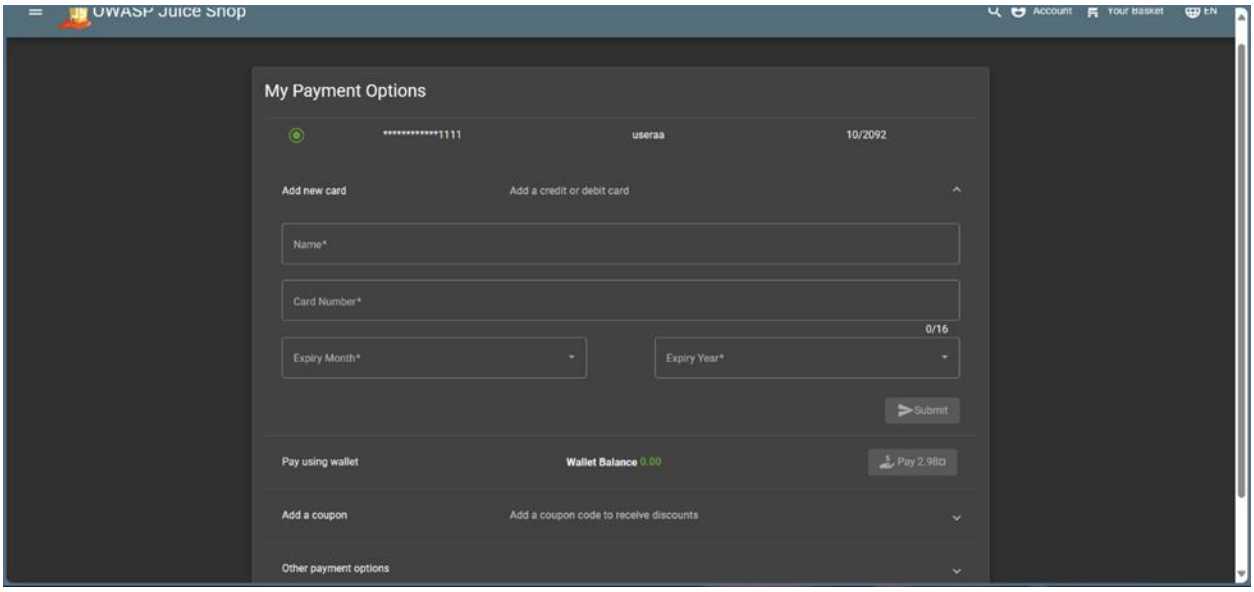
2. **Observe the Order Completion URL:**

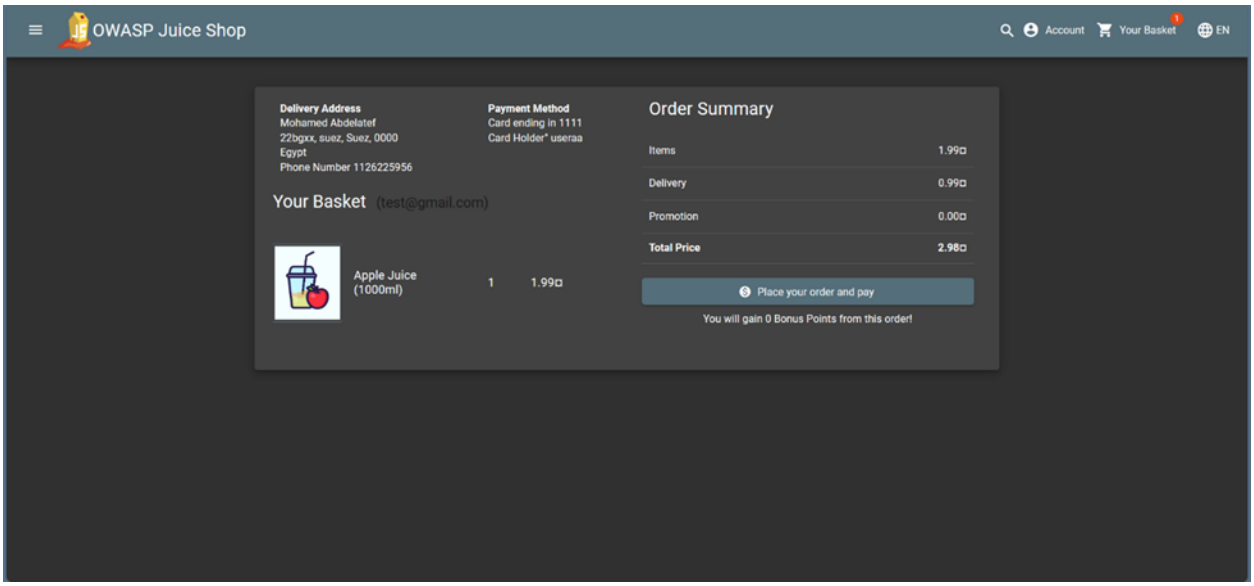

OWASP Juice Shop
Account
Your Basket
EN
Select an address
Add New Address
Continue

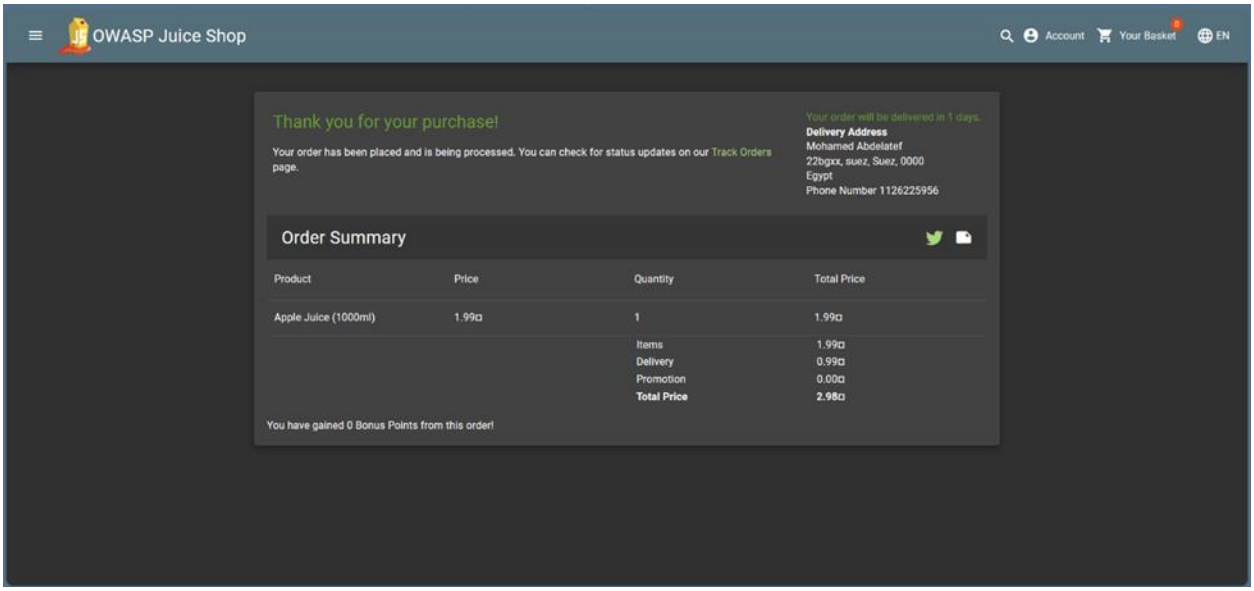

OWASP Juice Shop
Account
Your Basket
EN
Add New Address
Country*
Name*
Mobile Number*
ZIP Code*
0/8
Address*
Max. 160 characters
0/160
City*
State
Back
Submit


OWASP Juice Shop
Account
Your Basket
EN
Select an address
Mohamed Abdelatef
22bgxx, suez, Suez, 0000
Egypt
Add New Address
Continue
The address at suez has been successfully added to your addresses

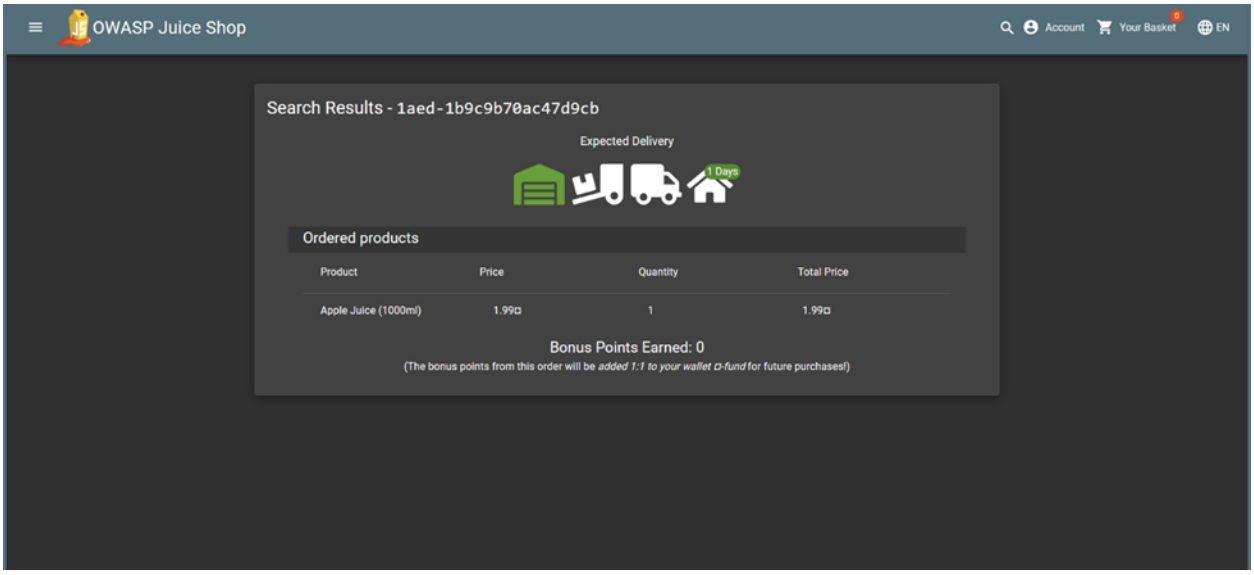

OWASP Juice Shop
Account
Your Basket
EN
Delivery Address
Mohamed Abdelatef
22bgxx, suez, Suez, 0000
Egypt
Phone Number 1126225956
Choose a delivery speed
Price
Expected Delivery
One Day Delivery
0.99¤
1 Days
Fast Delivery
0.50¤
3 Days
Standard Delivery
0.00¤
5 Days
Back
Continue
Snipping Tool
Screenshot copied to clipboard
Automatically saved to screenshots folder.
Markup and share


OWASP Juice Shop
Account
Your Basket
EN
My Payment Options
**********1111
useraa
10/2092
Add new card
Add a credit or debit card
Name*
Card Number*
0/16
Expiry Month*
Expiry Year*
Submit
Pay using wallet
Wallet Balance 0.00
Pay 2.98¤
Add a coupon
Add a coupon code to receive discounts
Other payment options

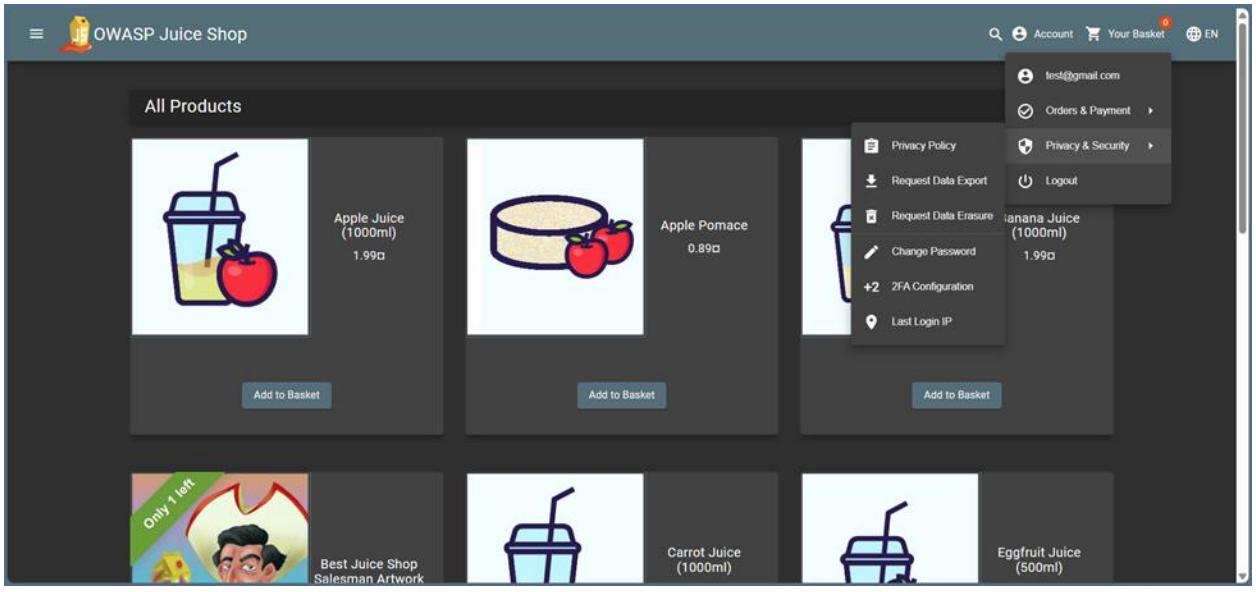

OWASP Juice Shop
Account
Your Basket
EN
Delivery Address
Mohamed Abdelatef
22bgxx, suez, Suez, 0000
Egypt
Phone Number 1126225956
Payment Method
Card ending in 1111
Card Holder* useraa
Order Summary
Items 1.99¤
Delivery 0.99¤
Promotion 0.00¤
Total Price 2.98¤
Your Basket (test@gmail.com)
Apple Juice (1000ml)
1
1.99¤
Place your order and pay
You will gain 0 Bonus Points from this order!

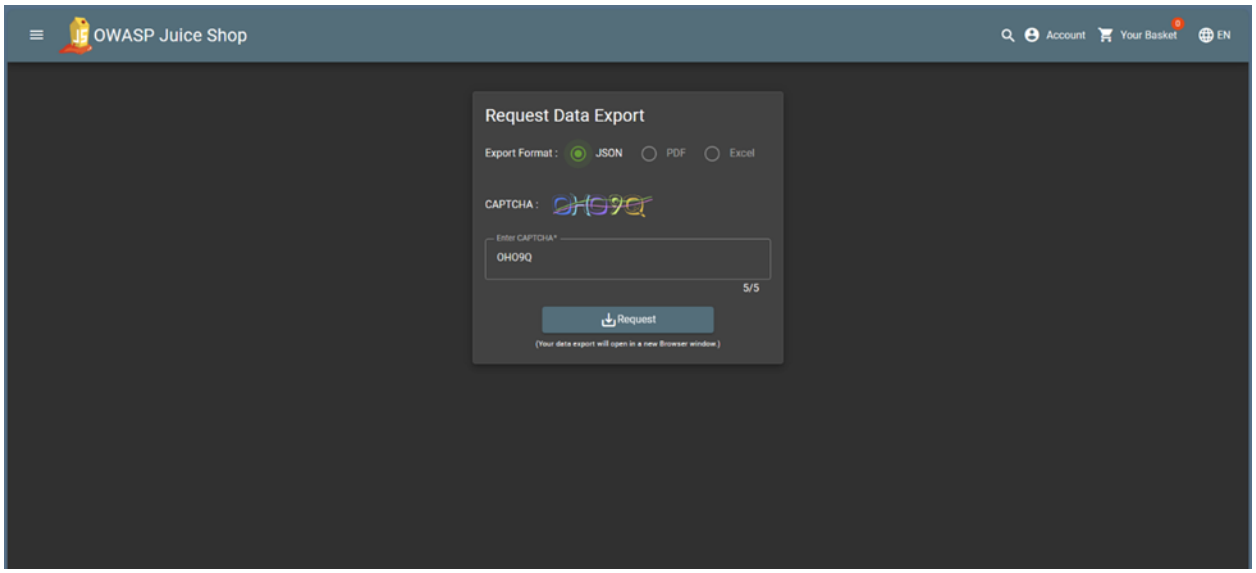


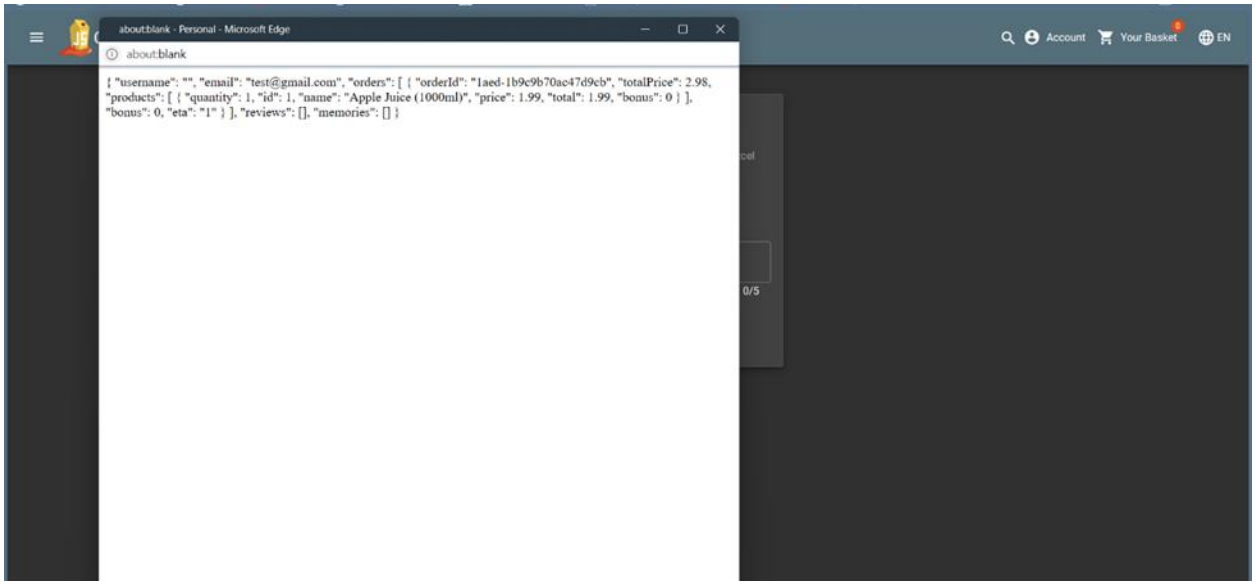
3. **Access the Order Tracking Page:**



4. **Request Data Export:**

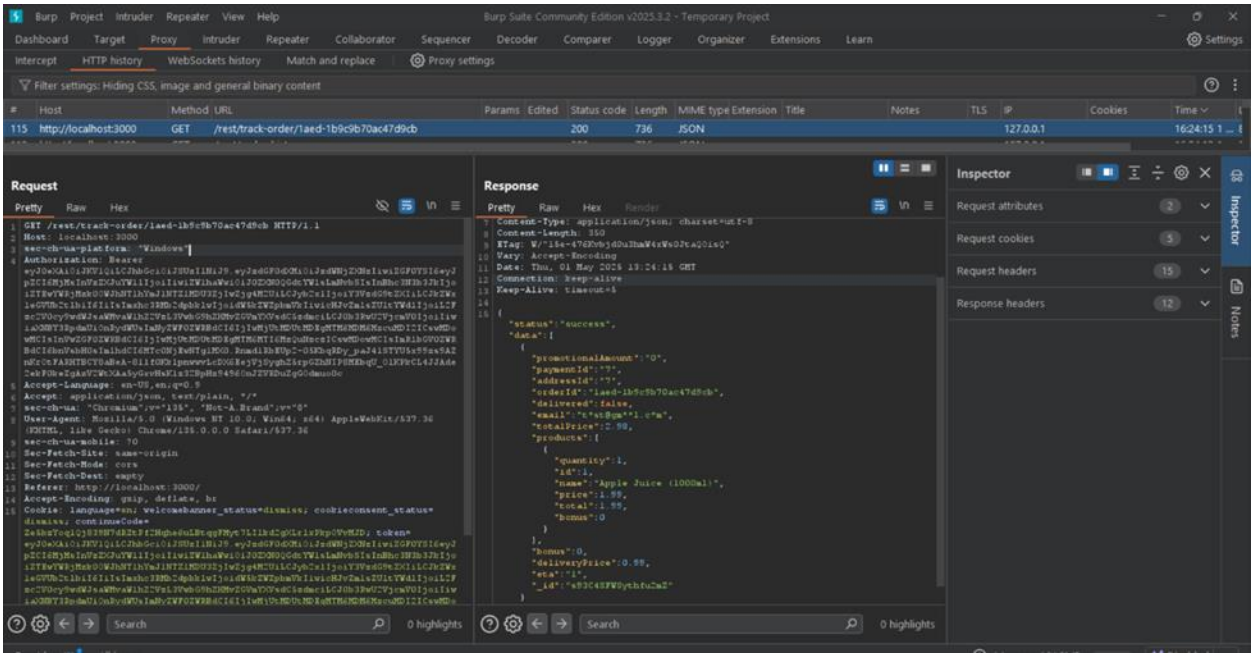



OWASP Juice Shop
Account
Your Basket
EN
Request Data Export
Export Format :
JSON
PDF
Excel
CAPTCHA :
Enter CAPTCHA*
OHO9Q
5/5
Request
(Your data export will open in a new Browser window.)


about:blank - Personal - Microsoft Edge
about:blank
{ "username": "", "email": "test@gmail.com", "orders": [{ "orderId": "1aed-1b9c9b70ac47d9cb", "totalPrice": 2.98, "products": [{ "quantity": 1, "id": 1, "name": "Apple Juice (1000ml)", "price": 1.99, "total": 1.99, "bonus": 0 }], "bonus": 0, "eta": "1" }], "reviews": [], "memories": [] }
Account
Your Basket
EN
0/5

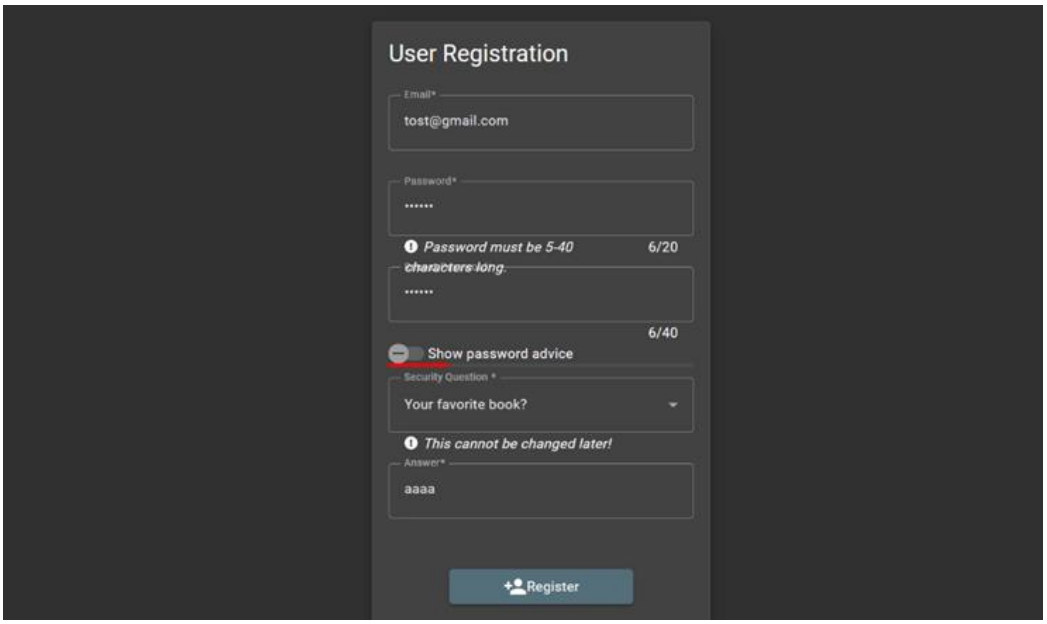
5. Open **Burp Suite** and intercept the HTTP request

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

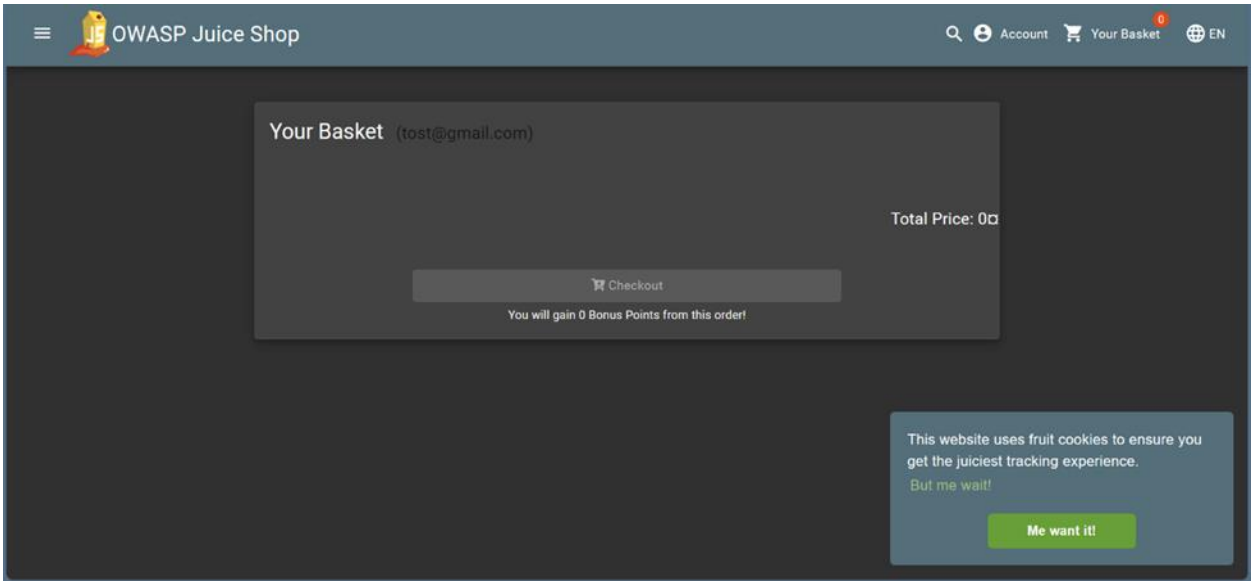


6. **Register a New User:**

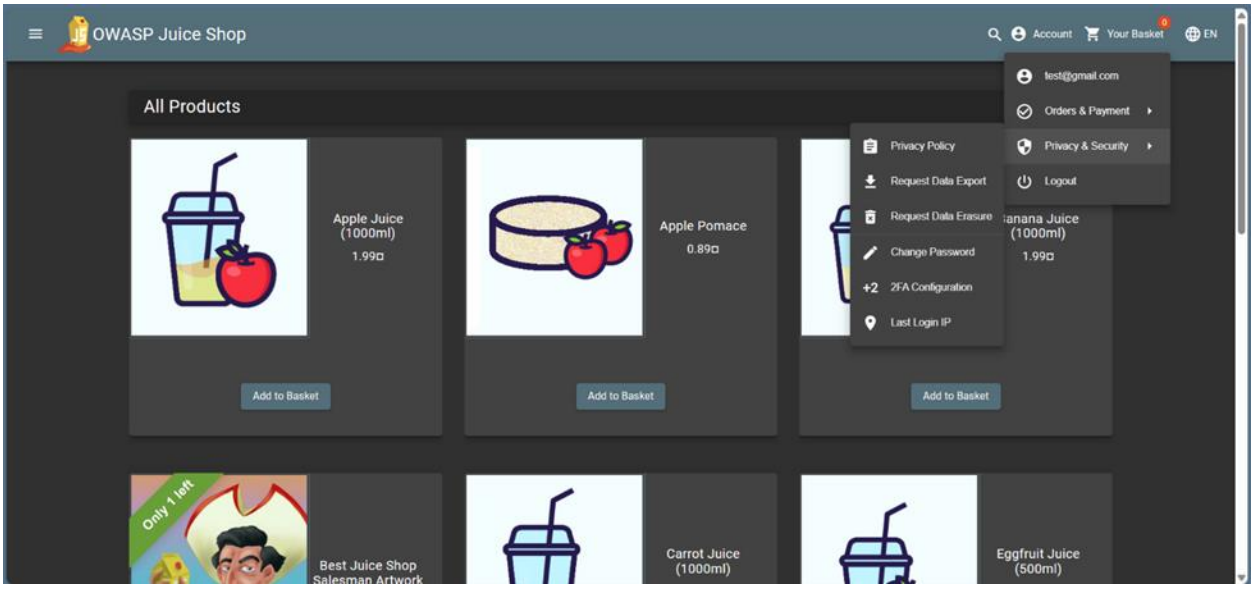
- I Create an account with an email that, once obfuscated, matches the existing user's obfuscated email, **tost@gmail.com**.

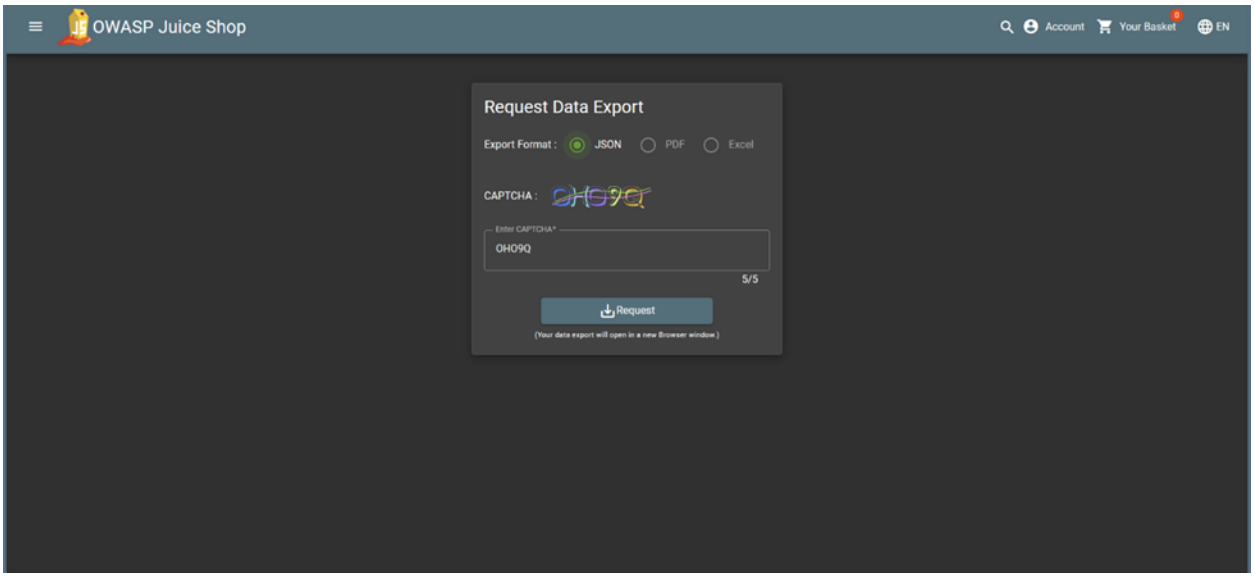


7. **Open View Basket:**
    - The basket is empty.

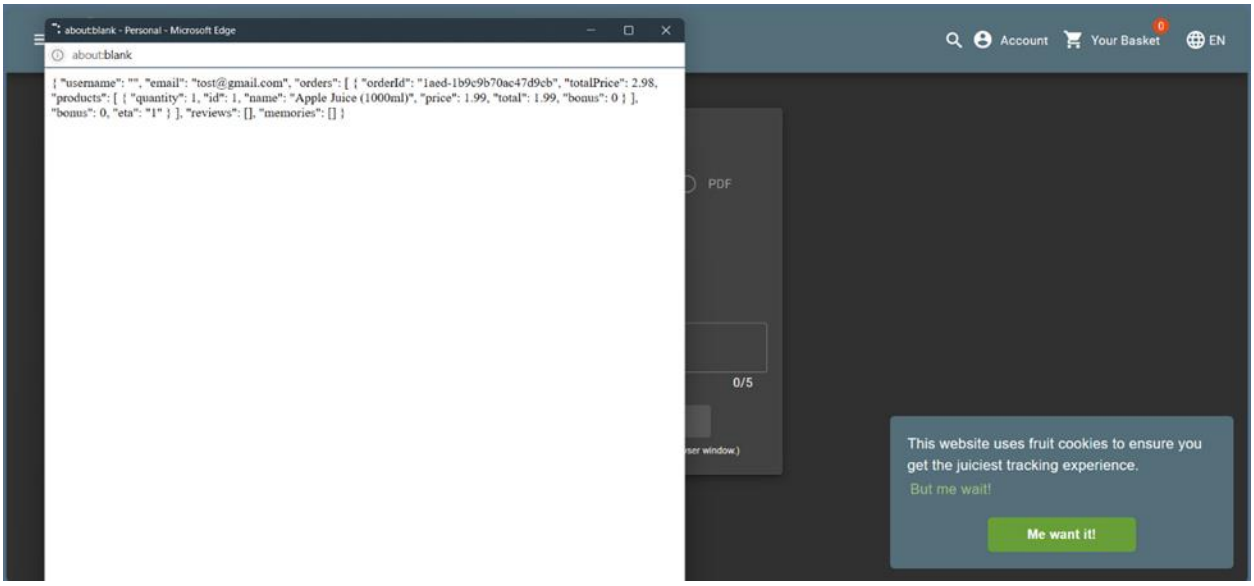


8. **Request Data Export:**
    - Log in as the new user and navigate to the Data Export page and export:





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:

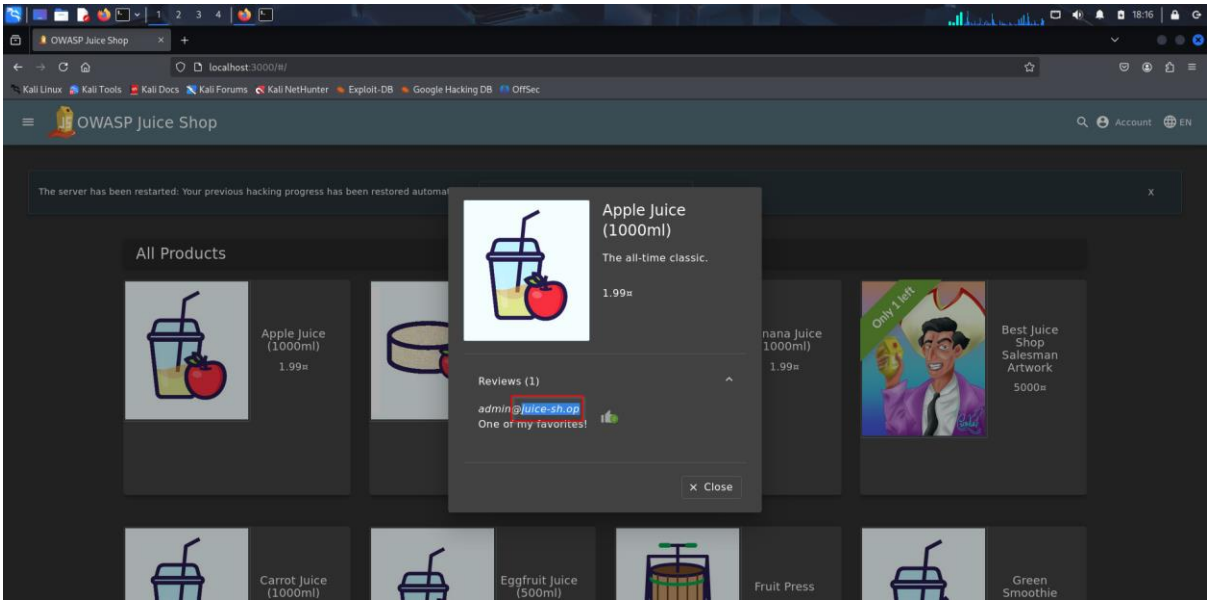
- Forgot Password recovery flow
- http://127.0.0.1:3000/#/photo-wall

Recommendations:

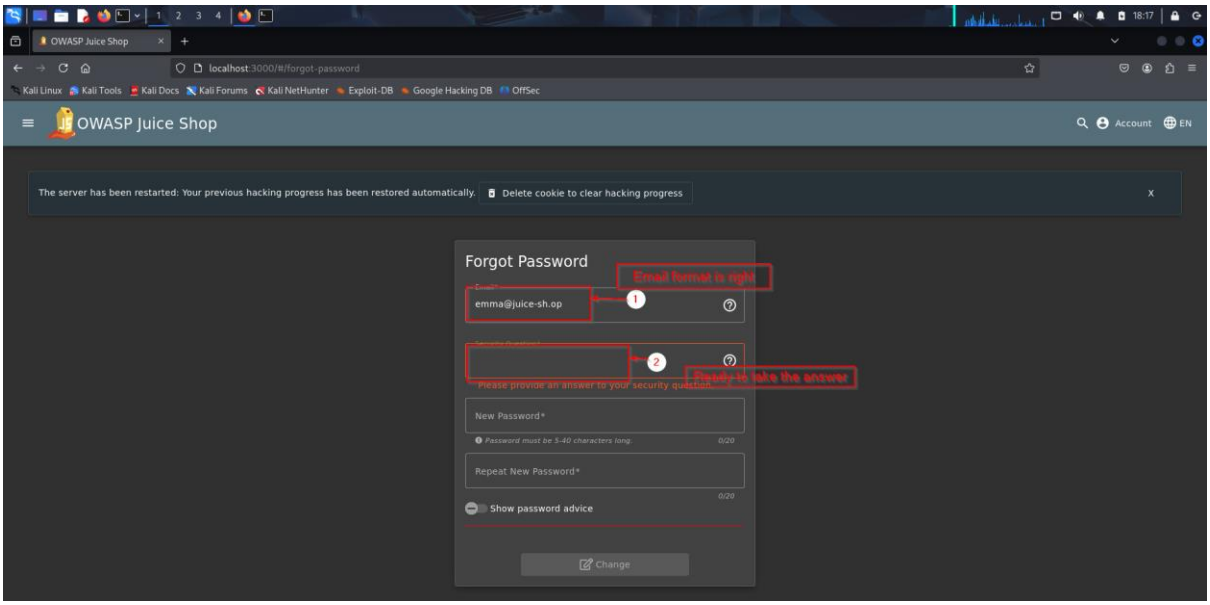
- Educate Users.
- Encourage the use of strict privacy settings on social media and other platforms to control who can view personal information and posts.

Proof of Concept:

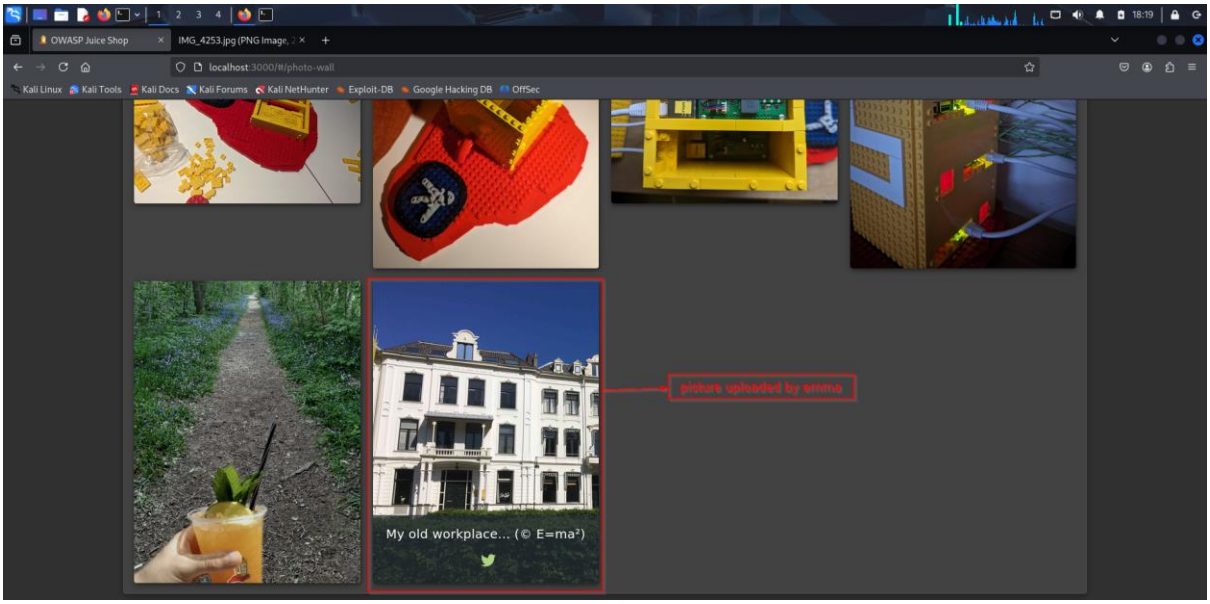
1. First we need to know the email format on the site we can see it in the review section in any product.



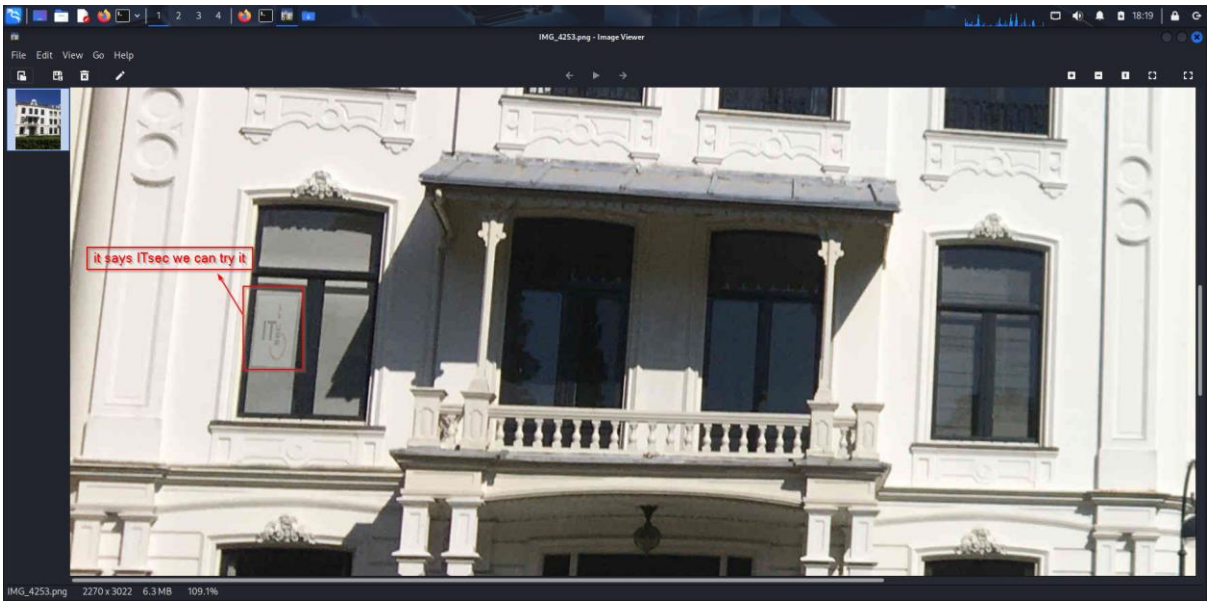
2. Then we use the victim name (emma) with the right email format and it waits for the security question answer.



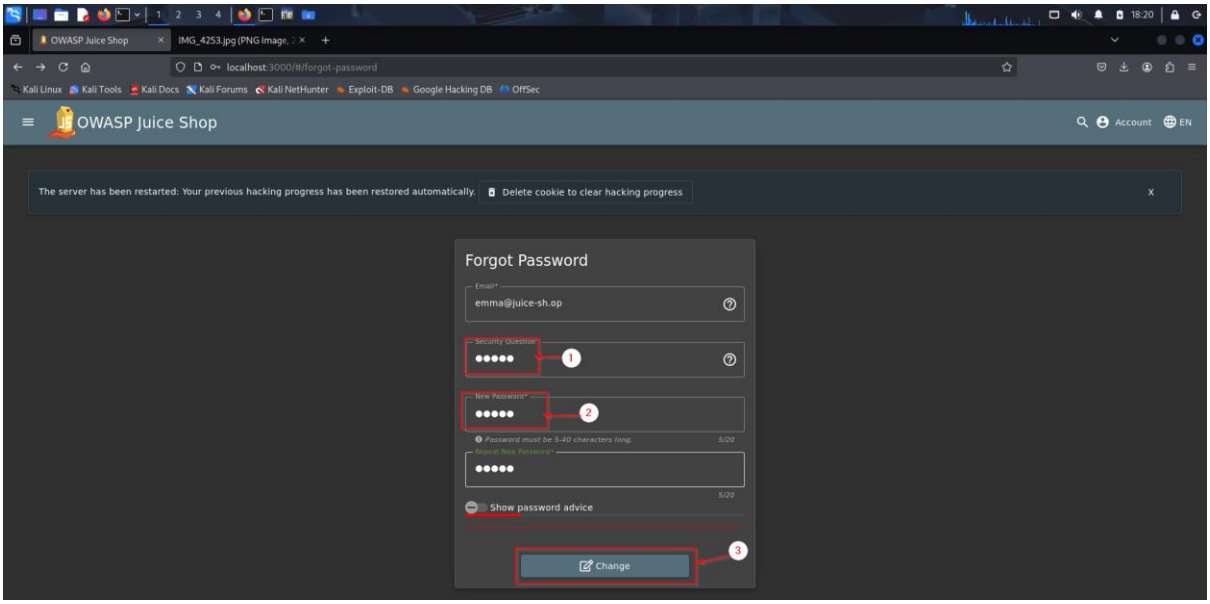
3. The victim uploaded a picture we can get it (photo wall section) and see what we can extract from it.



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



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


OWASP Juice Shop
IMG_4253.jpg (PNG Image, 2
localhost:3000/#/forgot-password
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
OWASP Juice Shop
Account
EN
The server has been restarted: Your previous hacking progress has been restored automatically.
Delete cookie to clear hacking progress
Forgot Password
Email*
emma@juice-sh.op
Security Question*
Password must be 5-40 characters long.
5/20
Repeat New Password*
5/20
Show password advice
Change
1
2
3

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:

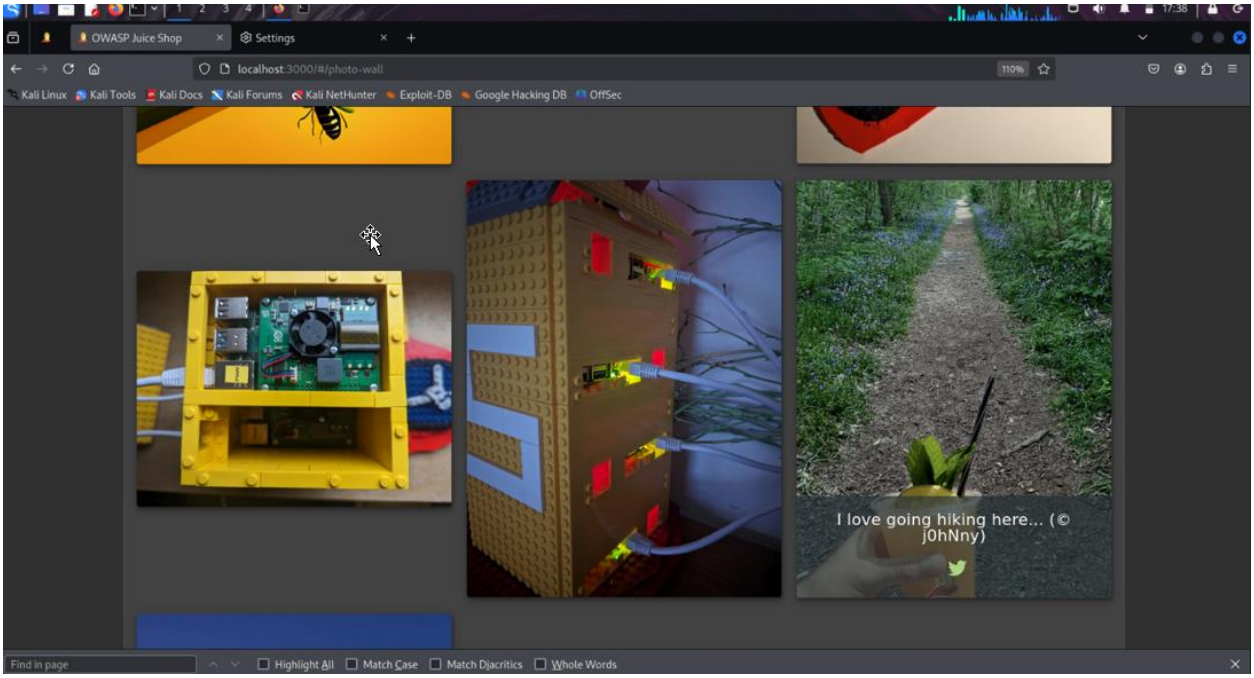
- **Endpoint:** http://127.0.0.1:3000/#/forgot-password
- **User Targeted:** John Doe (john@juice-sh.op)

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.

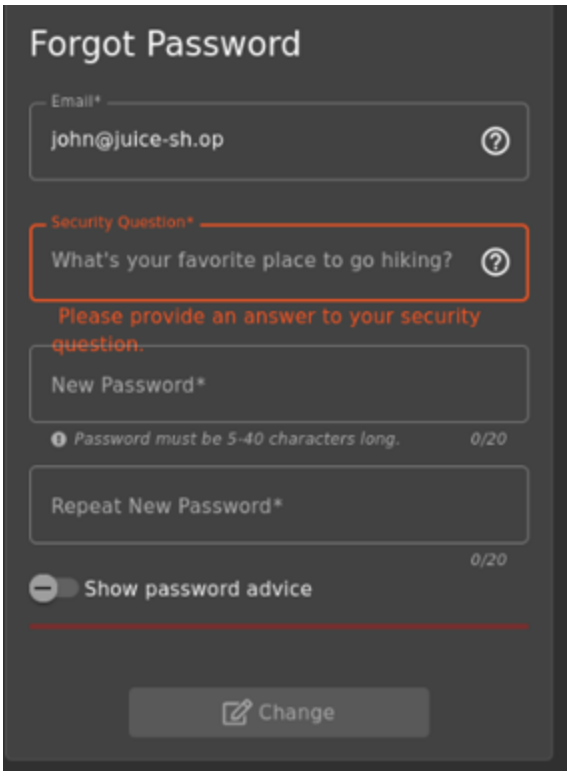
Proof of Concept (PoC):

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

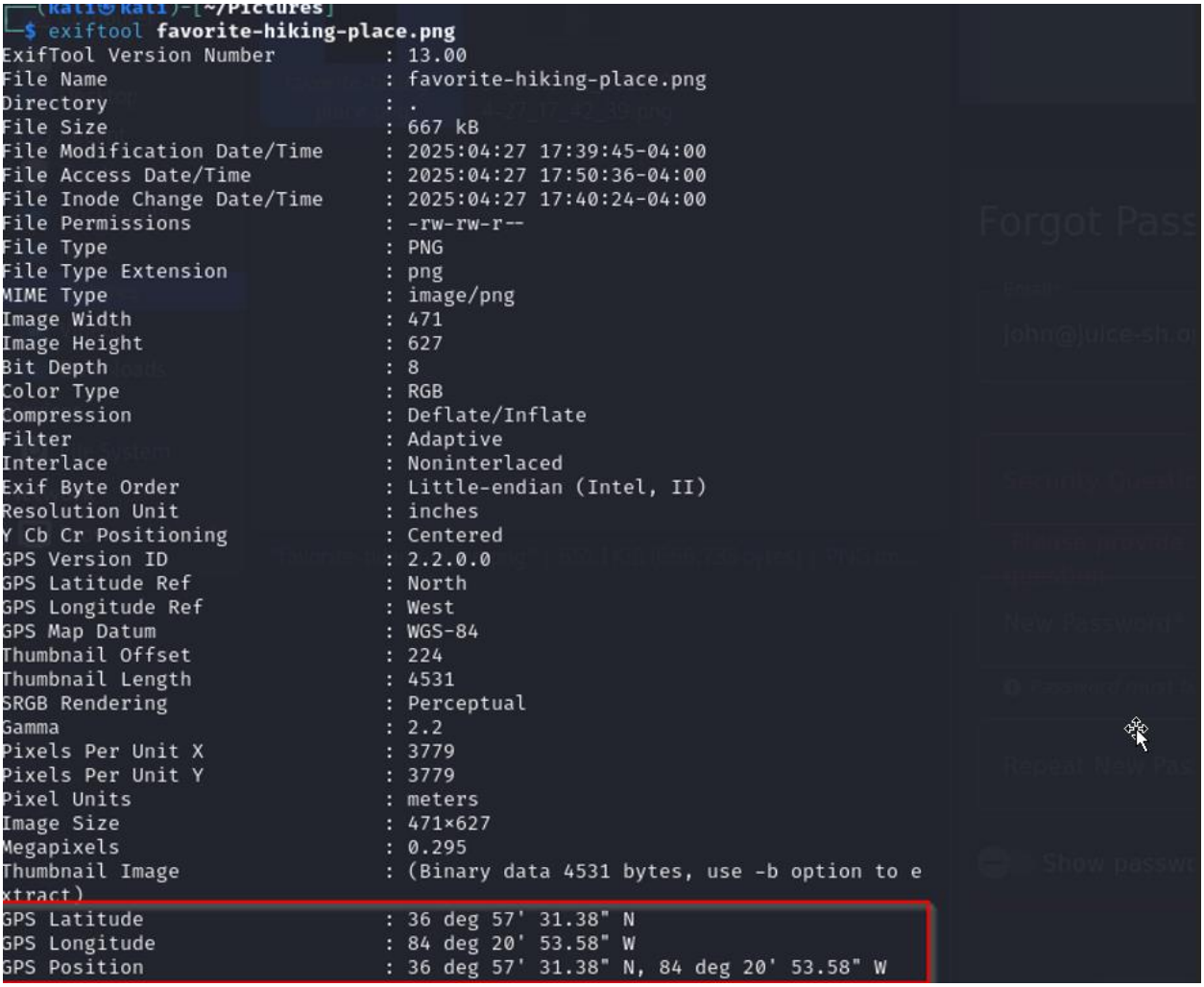


.

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 .



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.



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

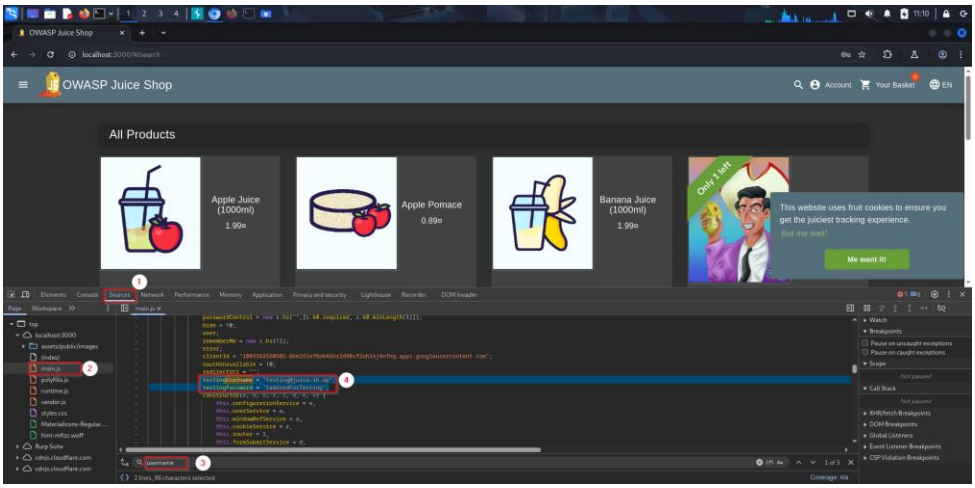
Vulnerability Location:
`(main.js) file`

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).



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

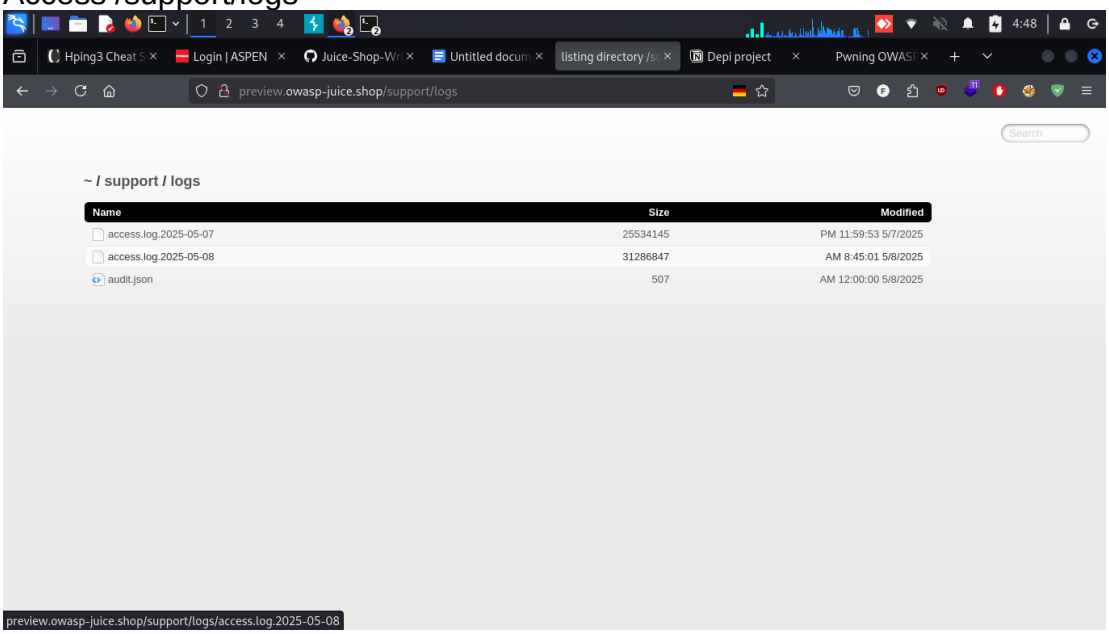
1. Highly valuable source of leaked sensitive data
2. Can expose server structure
3. May lead to discovery of hidden endpoints, debug URLs
4. Helps map application behavior for targeted attacks

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:

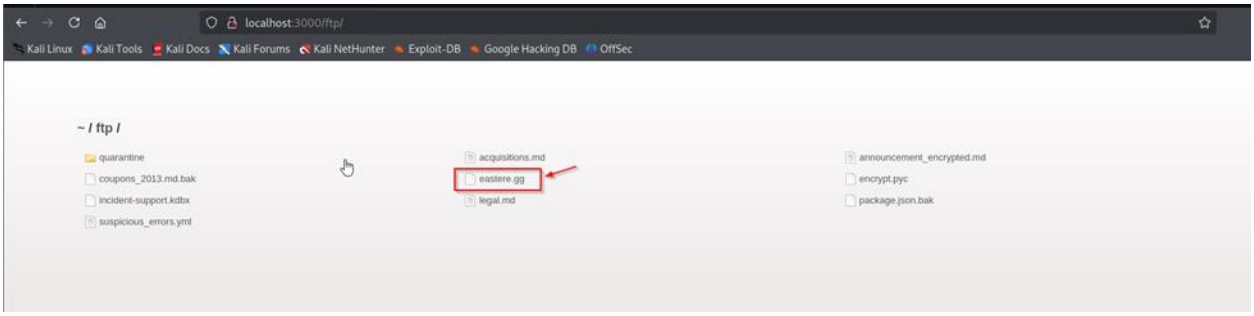
- OWASP A01:2021 – Broken Access Control
- OWASP A05:2021 – Security Misconfiguration
- OWASP Path Traversal Cheat Sheet
- OWASP Obfuscation and Encoding

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):

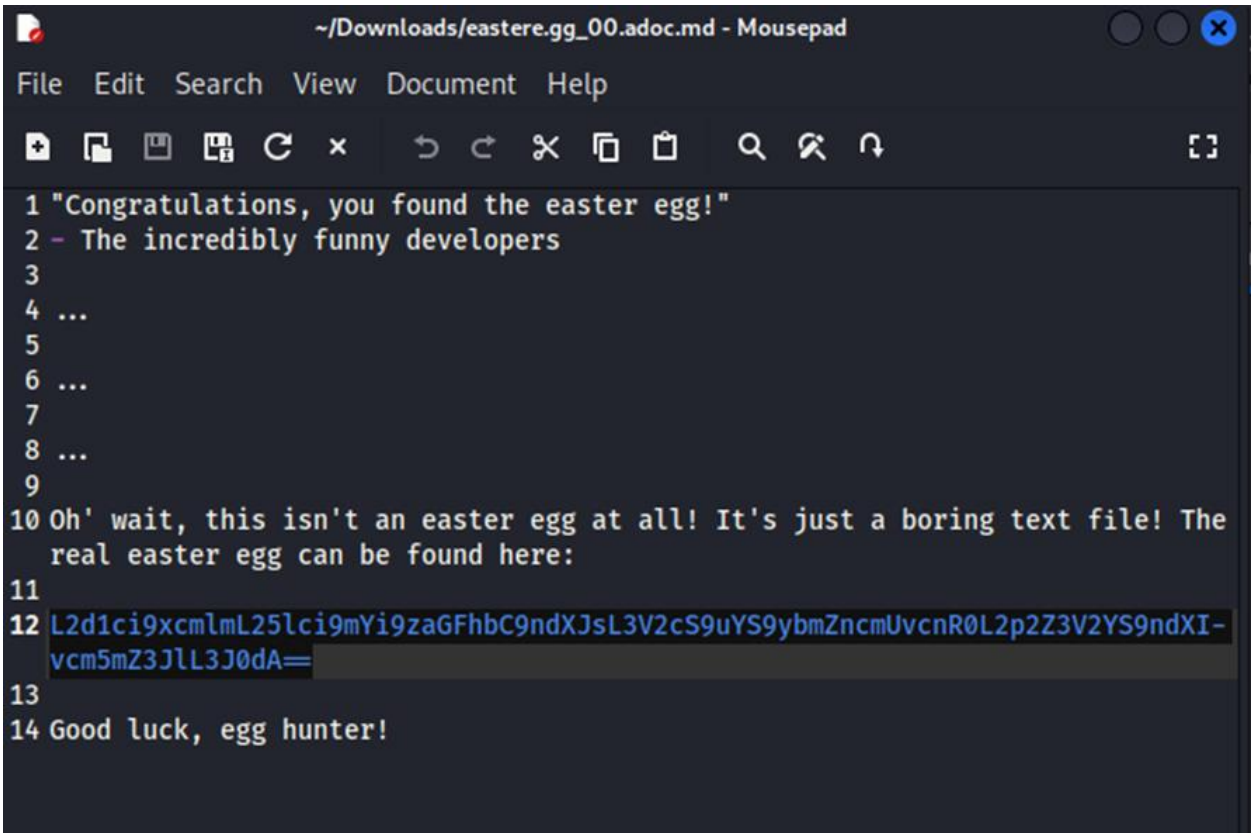
1. Initial Discovery:
   Based on Ref. ID: 5.4, the tester had access to hidden developer content.



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.



Extracting the Encoded Payload:
L2d1ci9xcmlmL25lci9mYi9zaGFhbC9ndXJsL3V2cS9uYS9ybmZncmUvcnR0L2p2Z3V2YS9ndXIvcm5mZ3JlL3J0dA==

3. Decoding the Payload:

Base64 decode result:

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

-

ROT13 decode result:

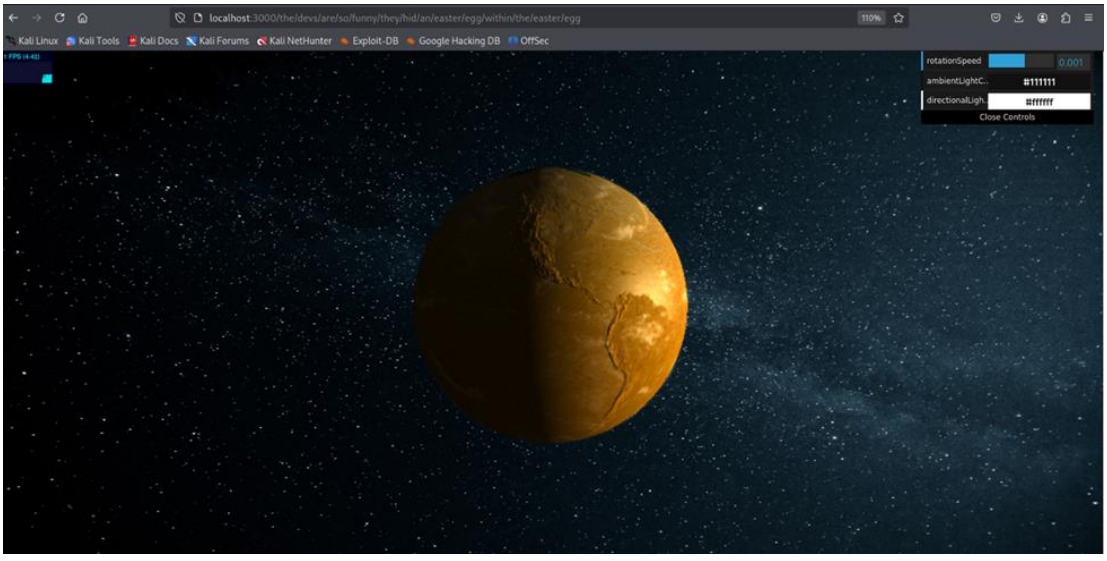
/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg

-

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.

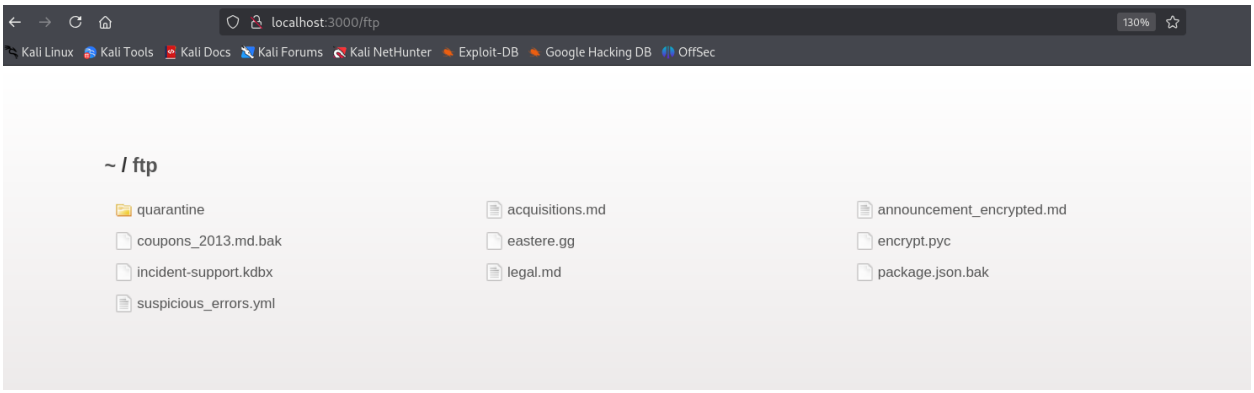
3. **Implement Whitelisting:** Validate file requests using a strict allowlist of expected and safe file extensions.

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:

    - `http://localhost:3000/ftp/`

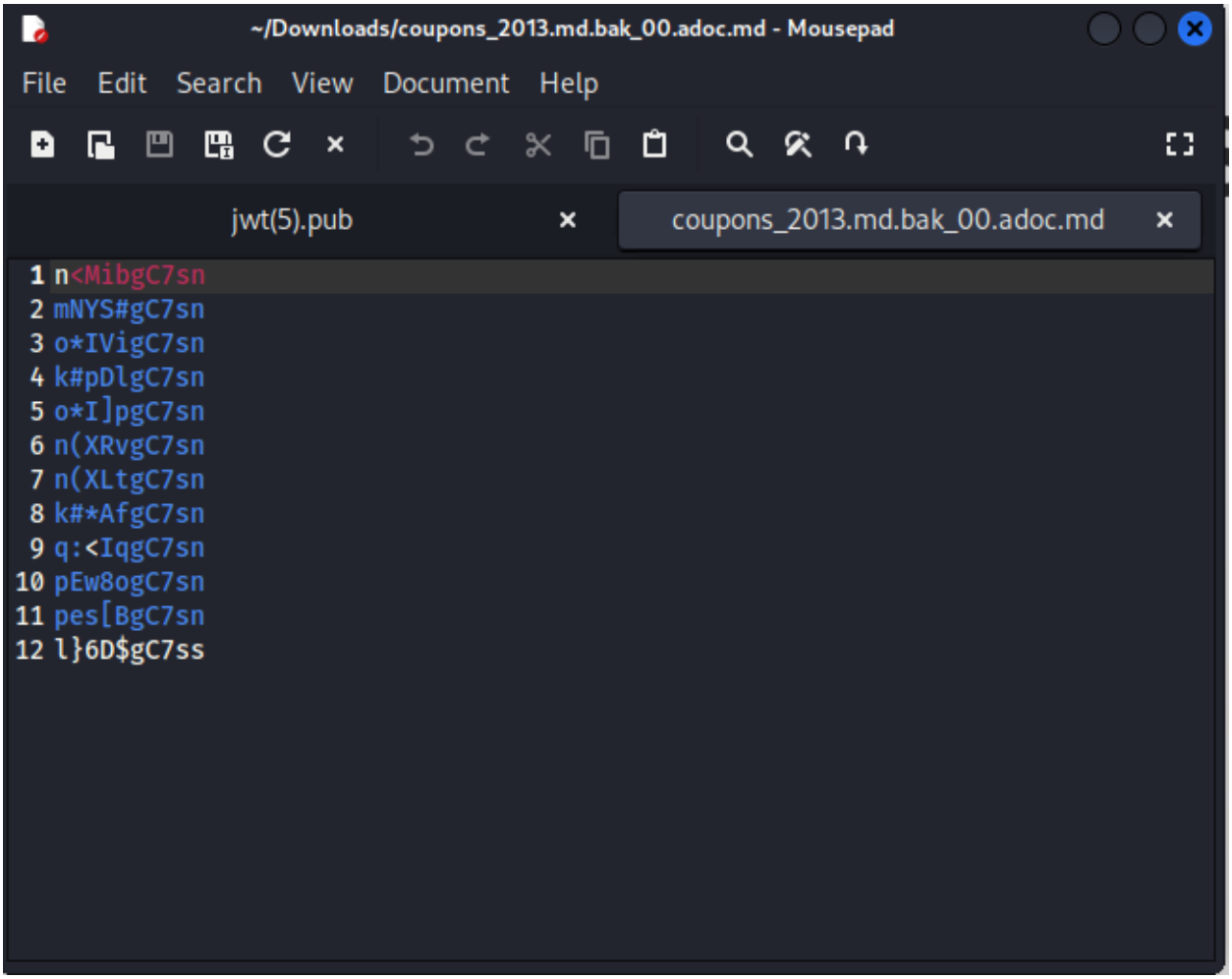


2. Identify the backup file:

    - `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:

    - `http://localhost:3000/ftp/coupons_2013.md.bak%2500.adoc.md`



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:

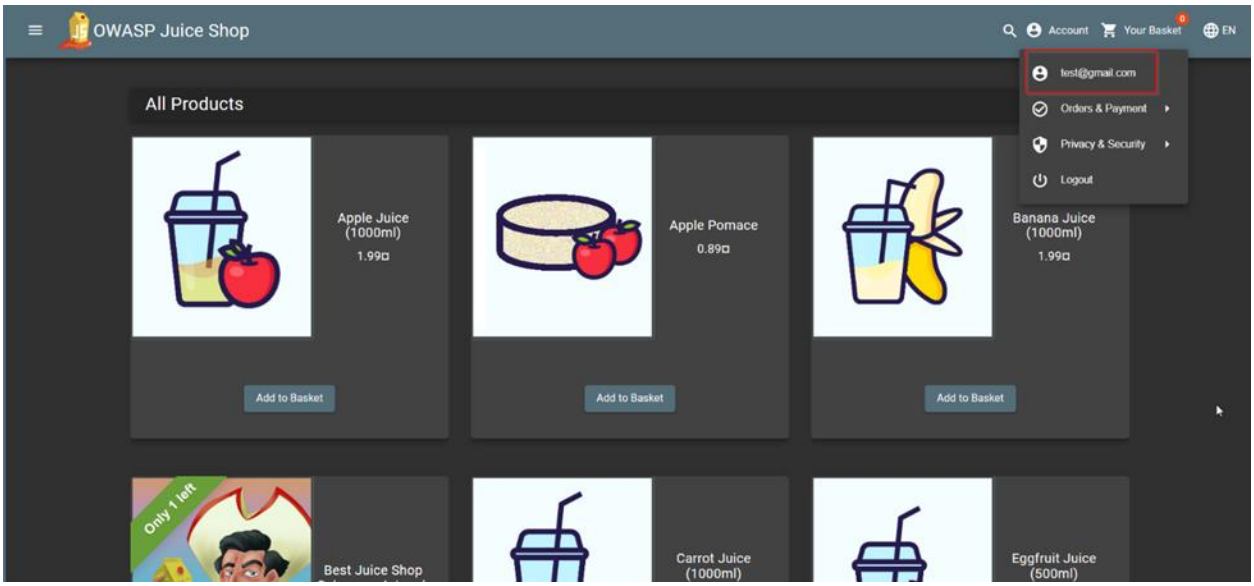
- CWE - CWE-352: Cross-Site Request Forgery (CSRF) (4.17)
- A01 Broken Access Control - OWASP Top 10:2021

Recommendations:

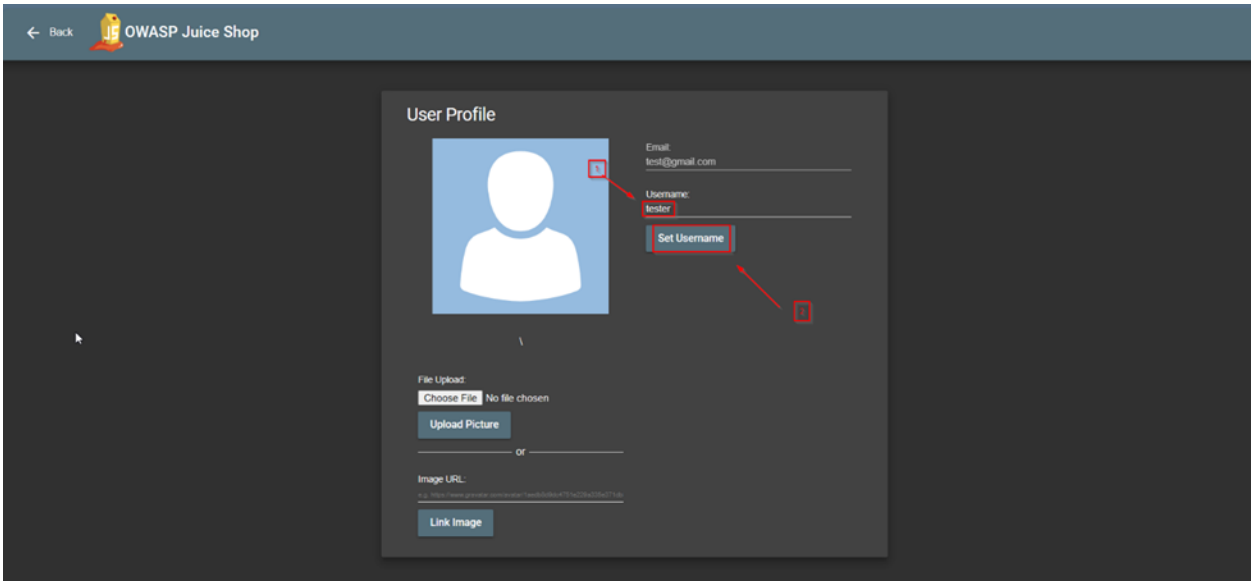
1. **Implement CSRF Tokens:**
    - Use anti-CSRF tokens in every state-changing request to ensure authenticity.
2. **Verify Origin and Referer Headers:**
    - Reject requests originating from unknown or unauthorized domains.

Proof of Concept (PoC):

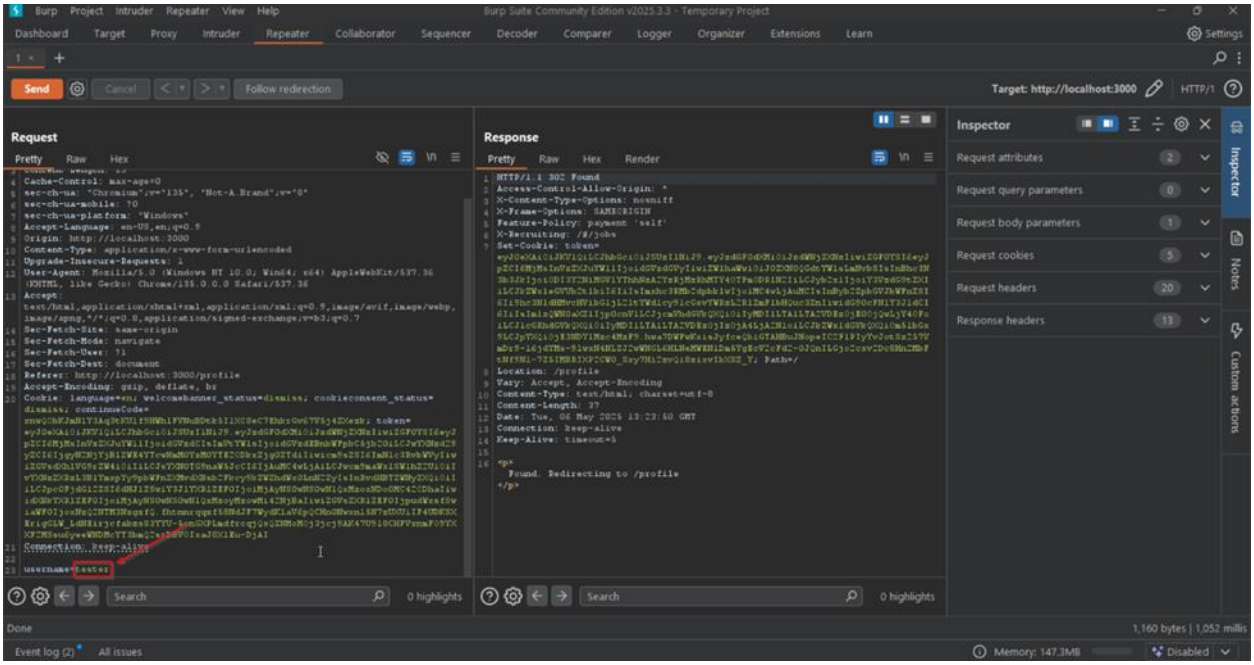
1. **Login as a regular user** on the Juice Shop in one browser tab and navigate to profile:



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



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



Malicious code:

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

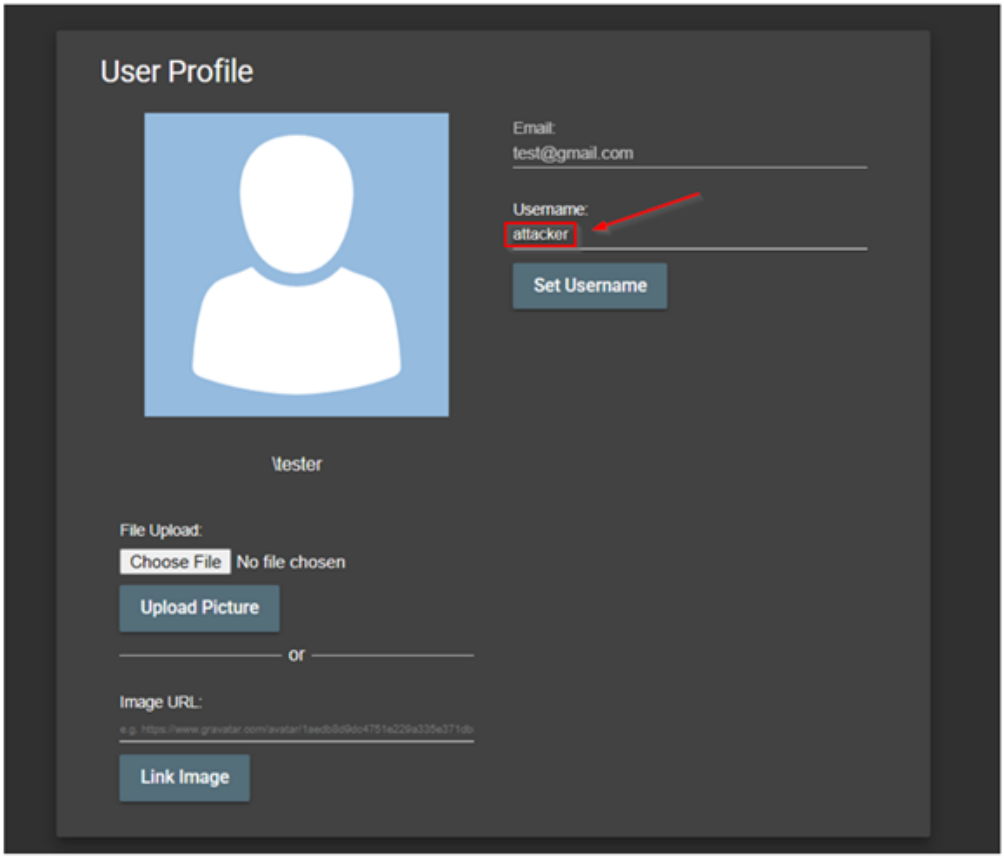
```
<script>history.pushState( ' ' , ' ' , '/' )</script>

<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:

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



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:

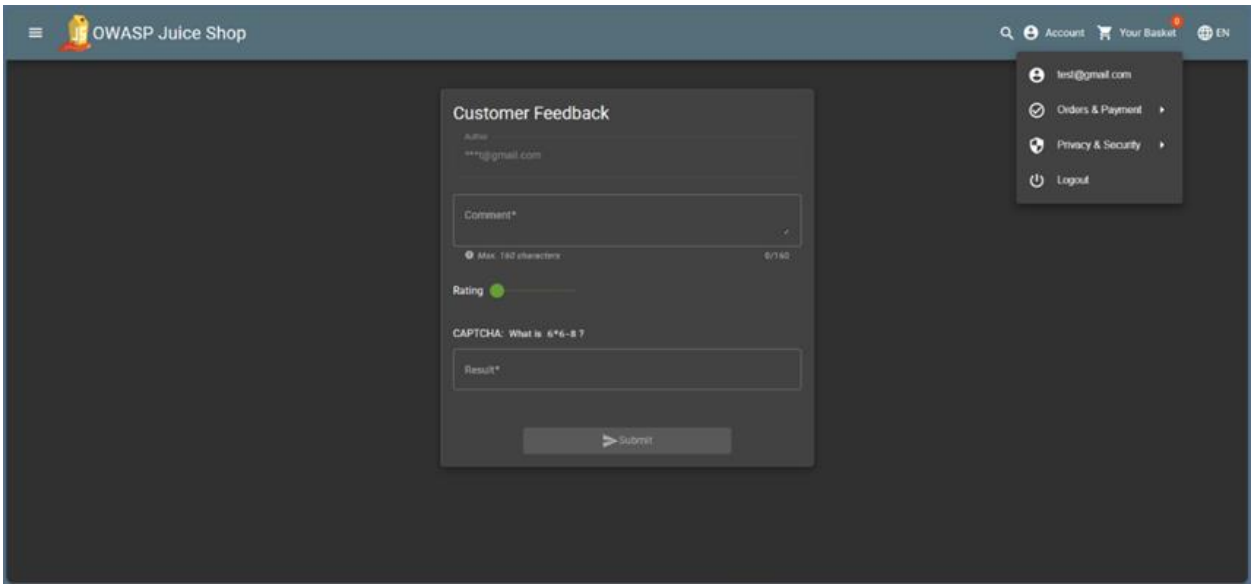
1. **Enforce Ownership Validation:**
    - Ensure only the original author of the feedback can edit or delete it.

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.

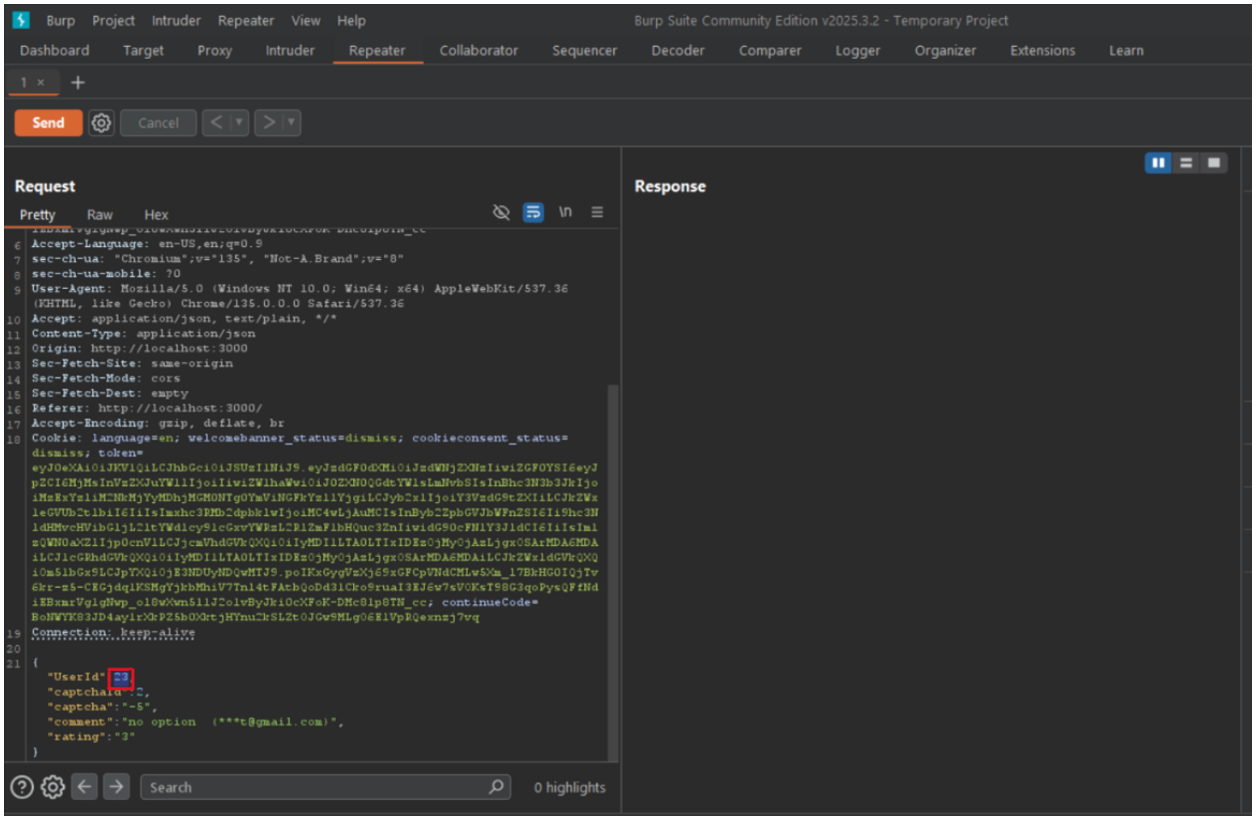
2. Submit a feedback message from the UI with a **star rating** (e.g., 3 star).



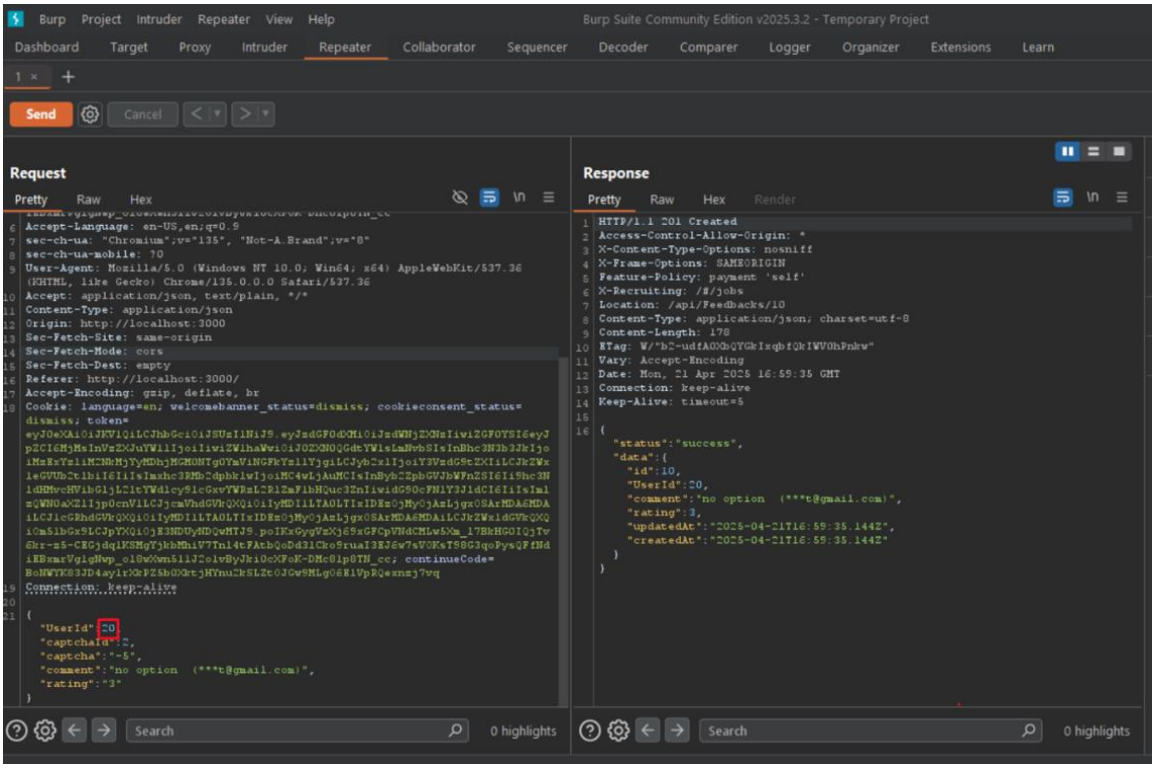
3. Open **Burp Suite** and intercept the HTTP request

4. Capture the PUT /api/Feedback/ request that updates the feedback.

5. Modify ""Userid"": 23 in the request to any id num.



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



15. Unvalidated Redirects

15.1 Unvalidated Redirect via Allowlist bypass

Medium

Description:
The application restricts redirects to a specific set of allowlisted 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:

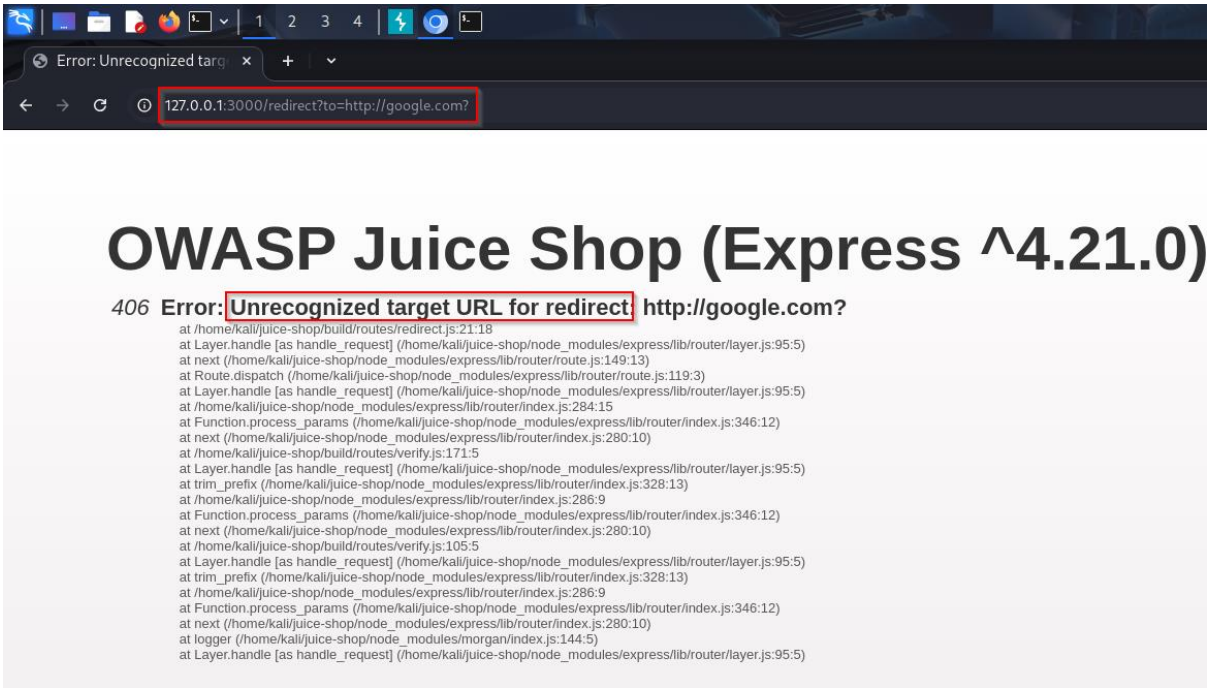
/juice-shop/build/routes/redirect.js (Redirection mechanism)

Recommendations:

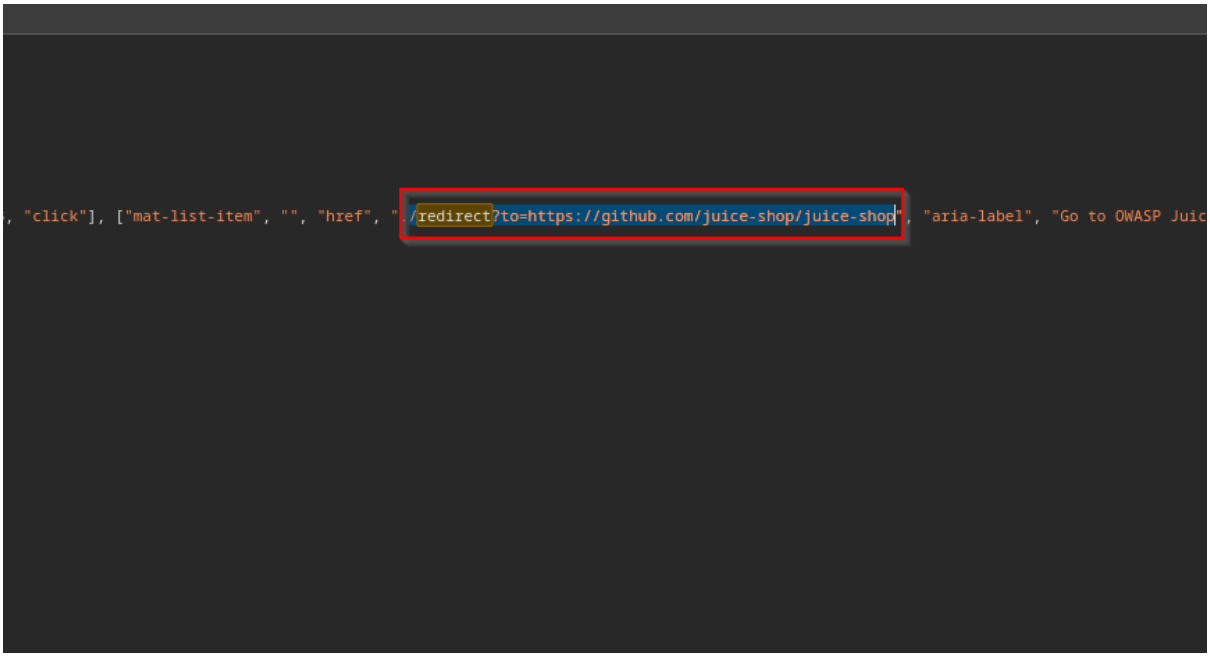
- Validate Full URL
- Use More Robust Redirect Mechanisms

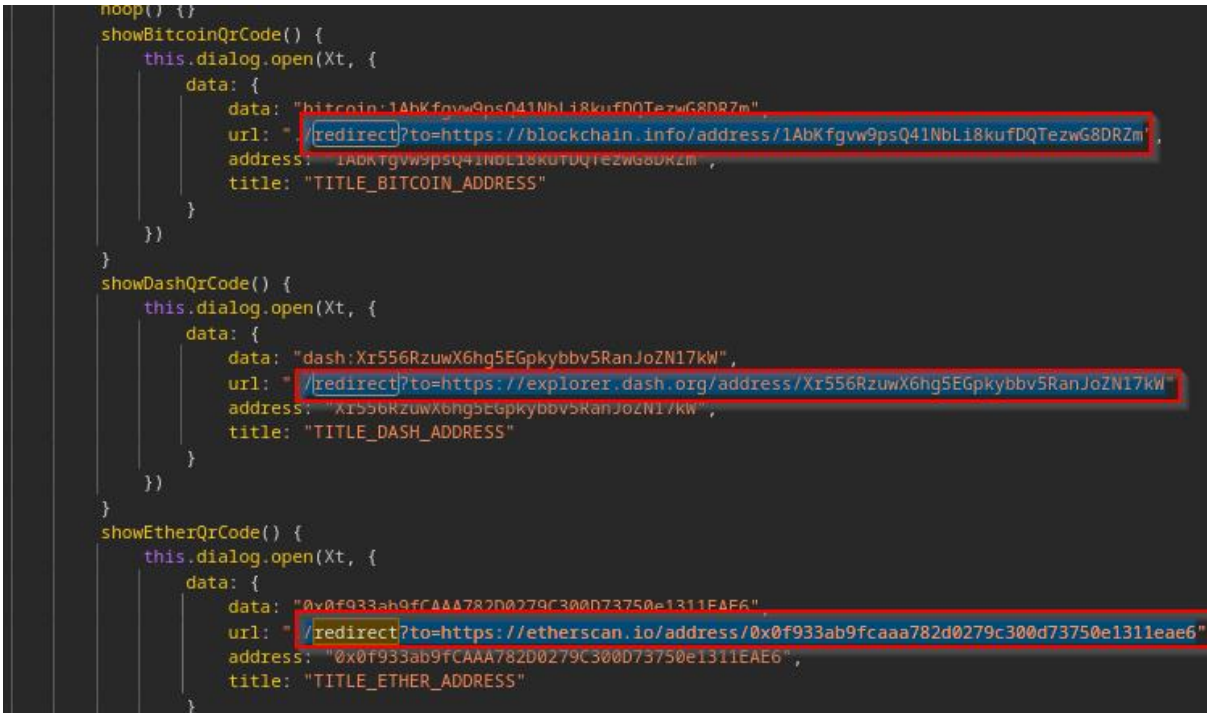
Proof of Concept:

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

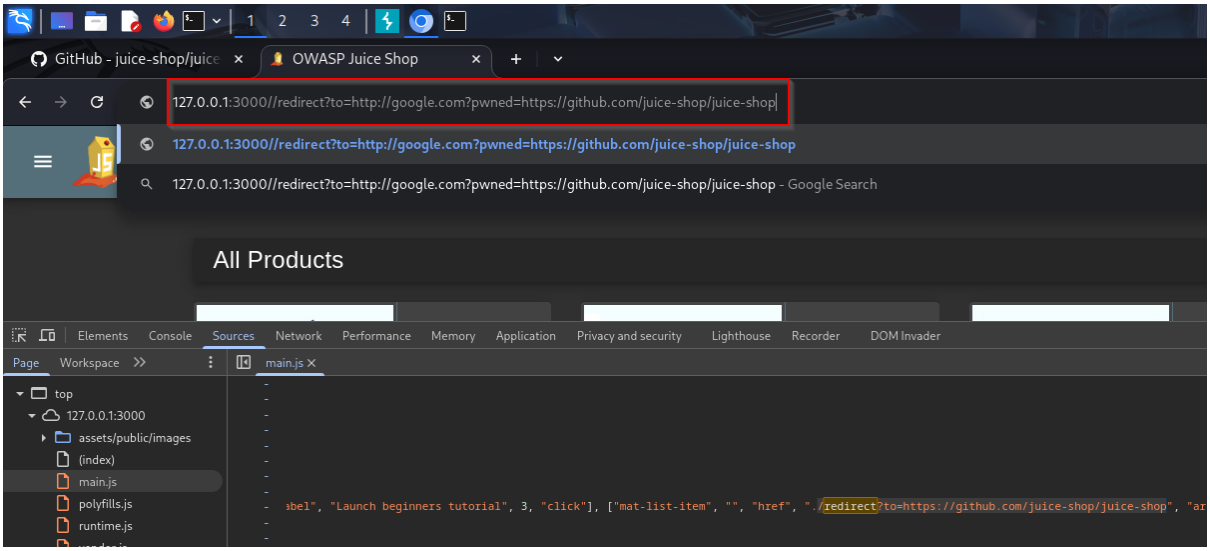


2. The previous attempt was blocked, confirming the presence of an allowlist mechanism so we can try one of the allowed redirections links.

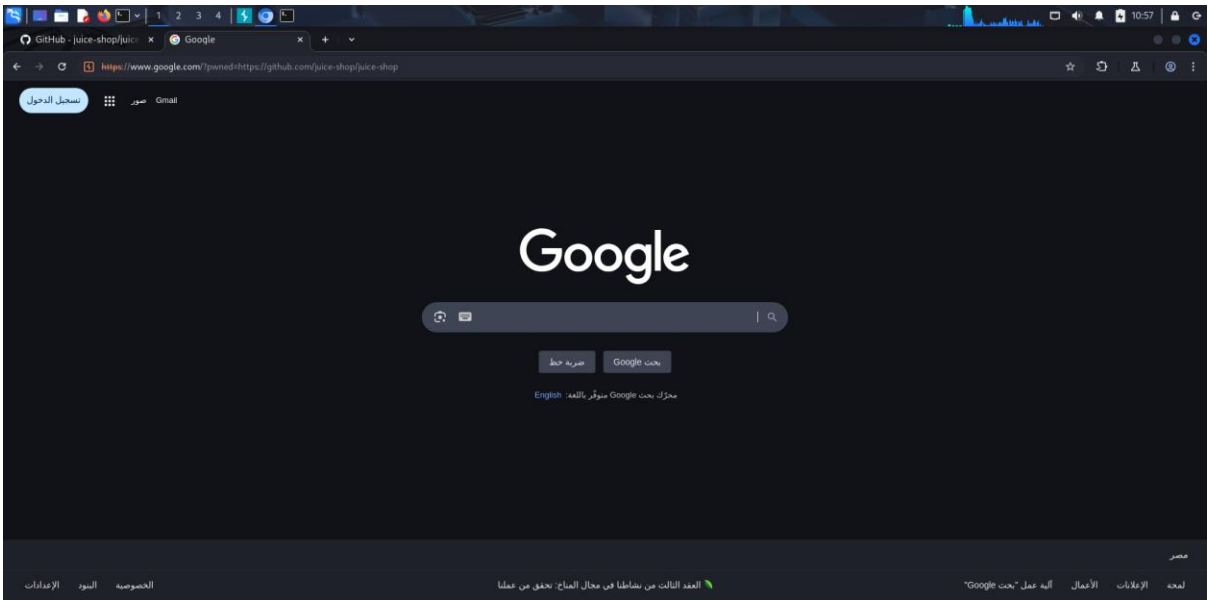




3. try to add one of it's allowed urls for redirection (https://github.com/juice-shop/juice-shop)



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.

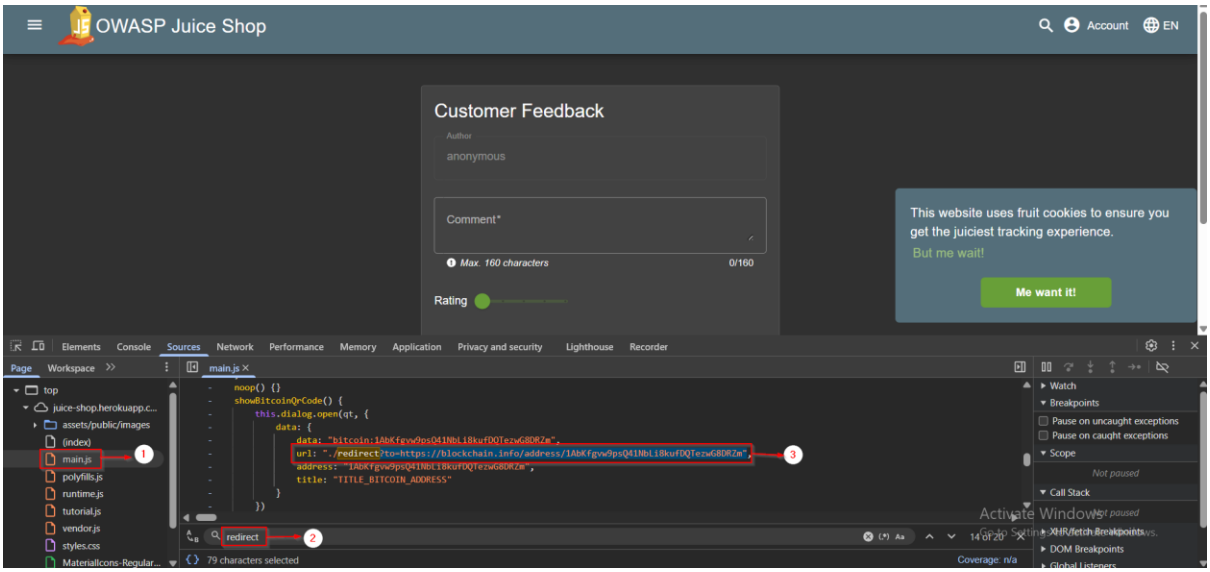
Vulnerability Location:
(main.js) file

Recommendations:

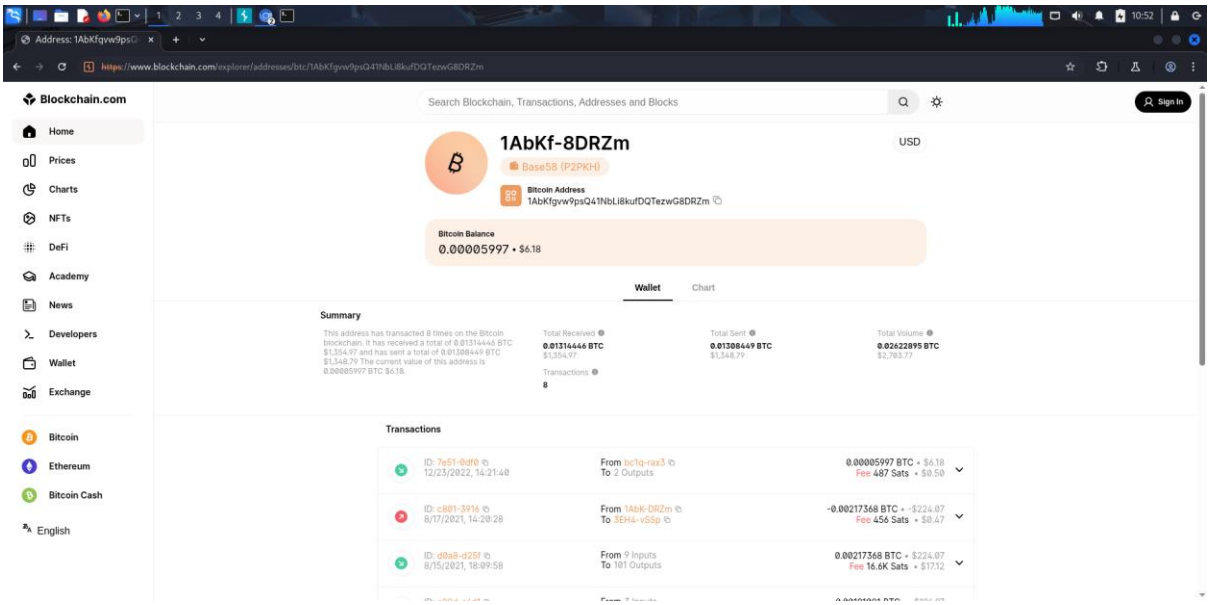
- Regularly Update and Validate Allowlists

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/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm



16.Privilege escalation

16.1 Privilege escalation through registration

HIGH

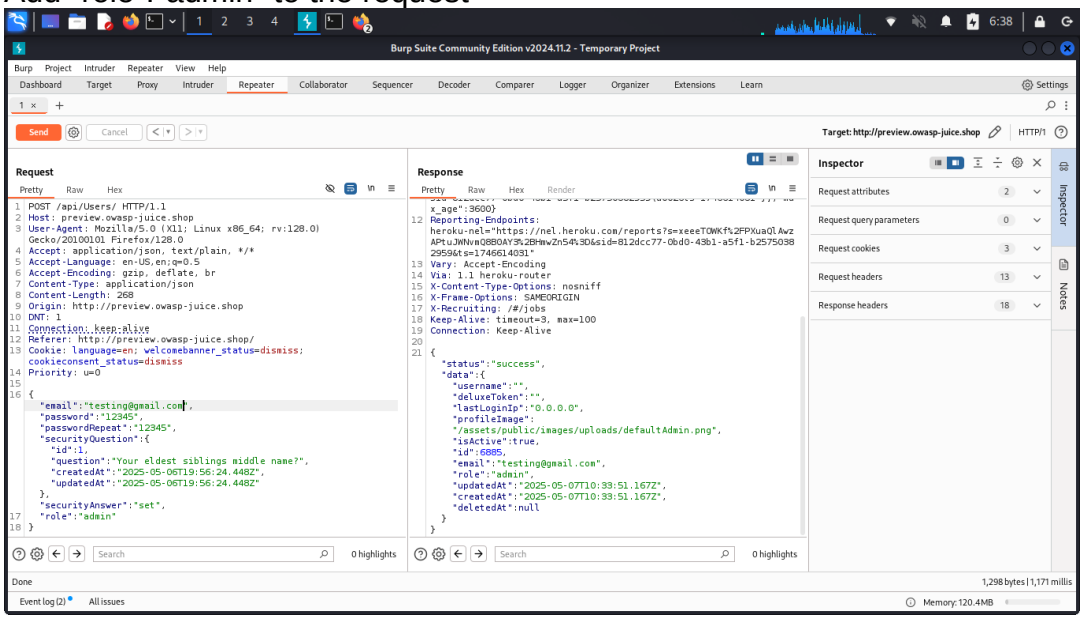
Location: registration page
Description:
Privilege Escalation is a type of attack where an adversary gains higher access rights or permissions than originally intended. It allows attackers to move from a limited user account to one with administrative or root privileges, enabling broader control over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



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.

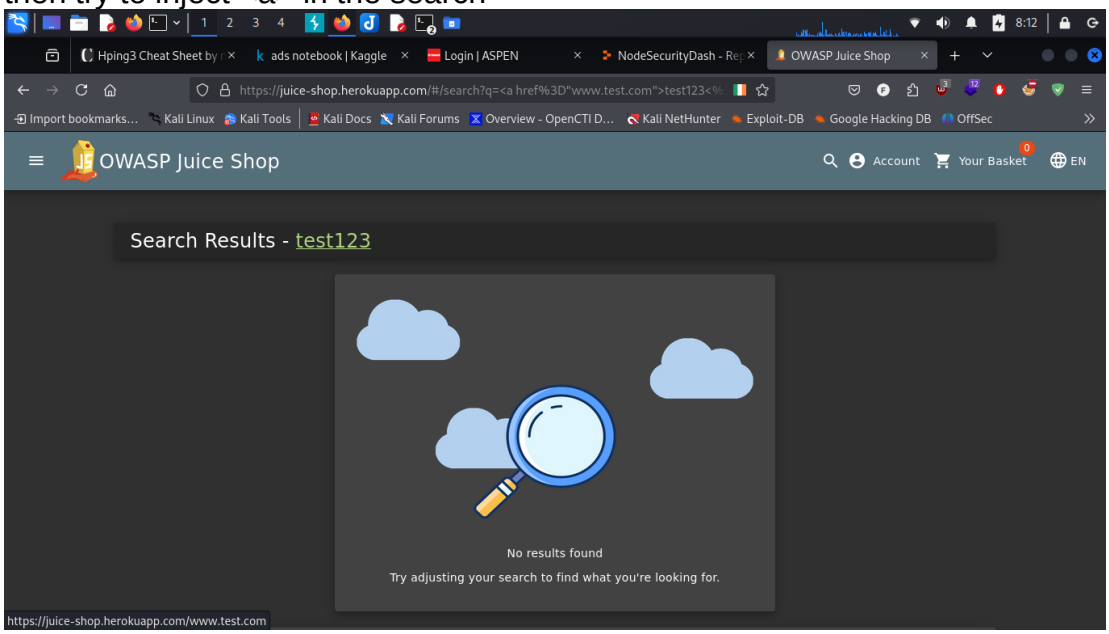
vulnerability was discovered in the application's search functionality. The application reflects user input without proper sanitization, allowing attackers to inject malicious scripts.
Impact
This could allow attackers to redirect users to malicious websites via <a>
Recommendations
- Implement output encoding
- Validate and sanitize all input

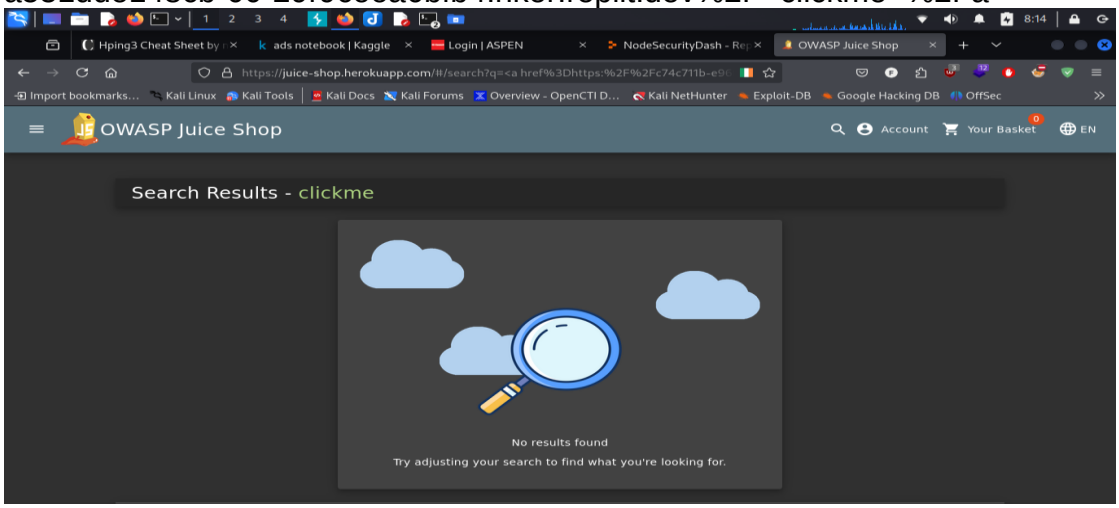
Resources:

1. **HTML Injection**
2. **WSTG - Latest | OWASP Foundation**

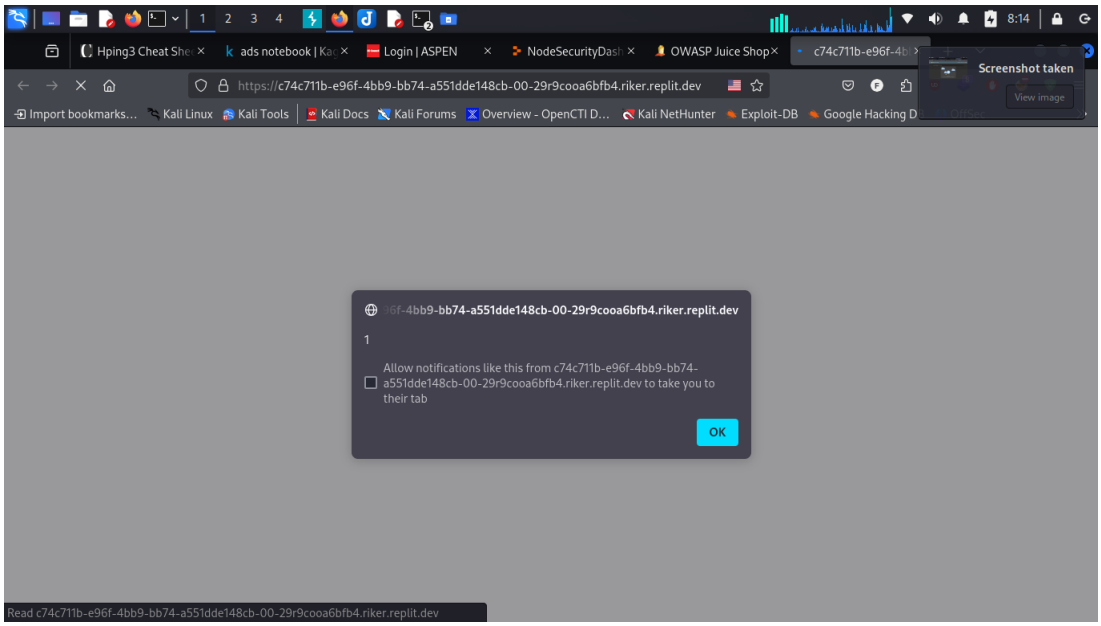
Proof Of Concept (POC)
1. Going to search bar
2. Type anything then you will see that the text reflected in span tag

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



4. **T**he 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-00-29r9cooa6bfb4.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:

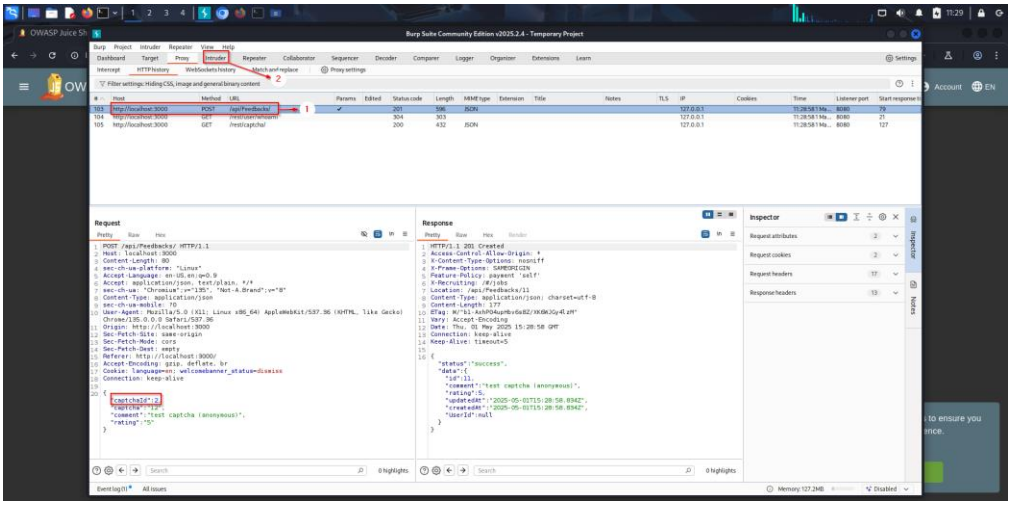
- http://localhost:3000/#/contact

Recommendations:

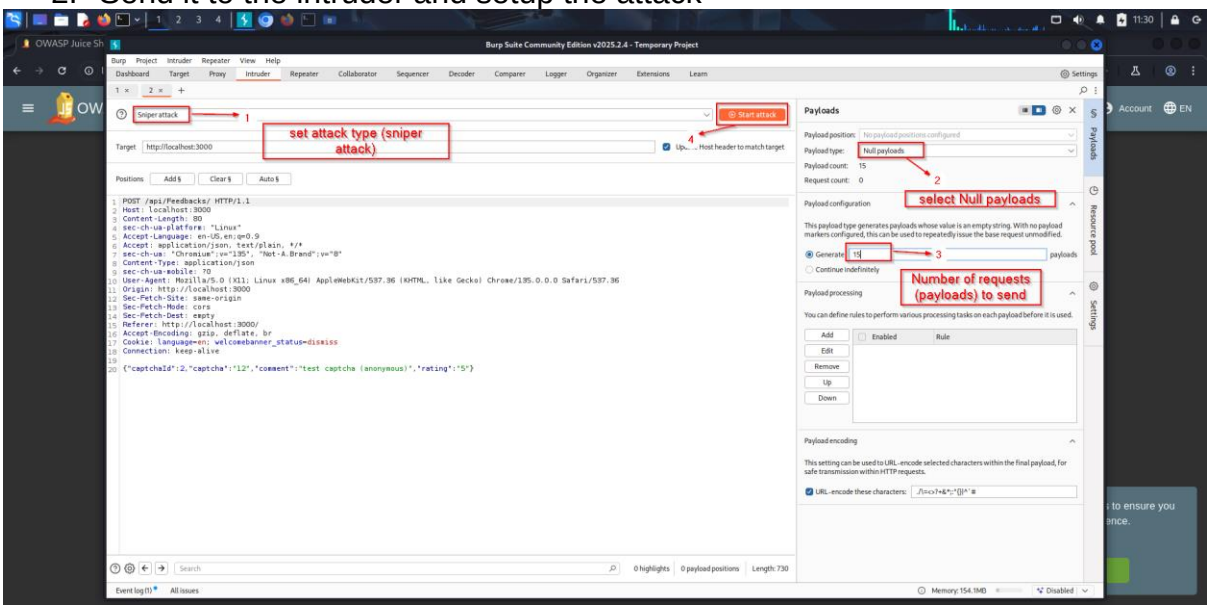
- Implement server-side CAPTCHA token validation.
- Invalidate tokens after one-time use to prevent replay.

Proof of Concept:

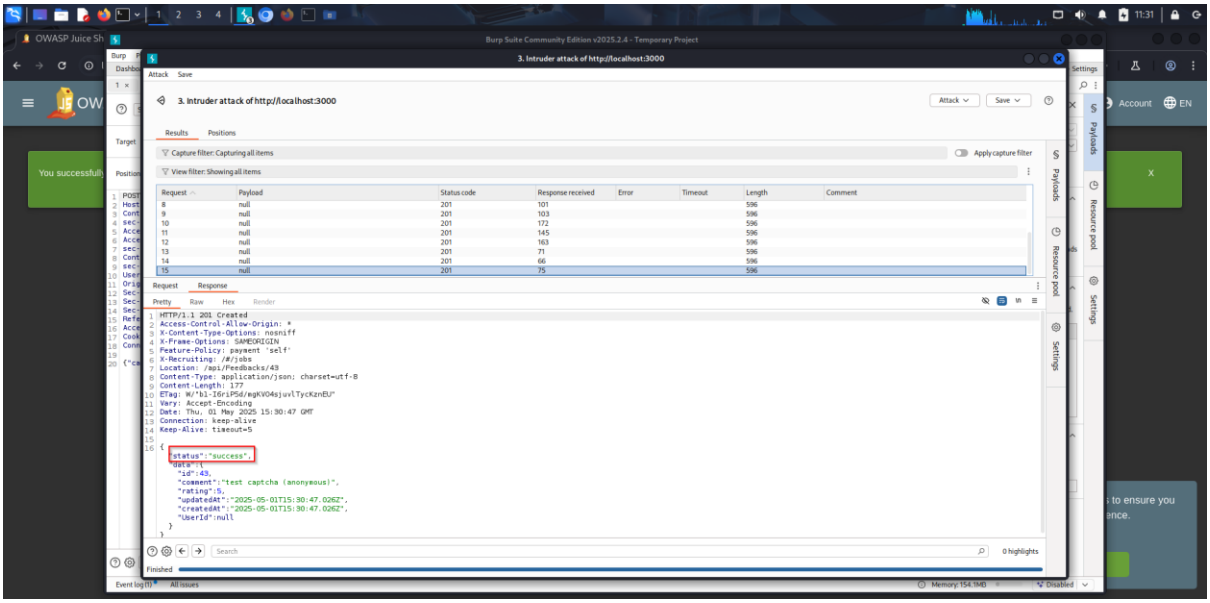
1. Submit a feedback then intercept it with burp



2. Send it to the intruder and setup the attack



3. Lunch it and see the result



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:

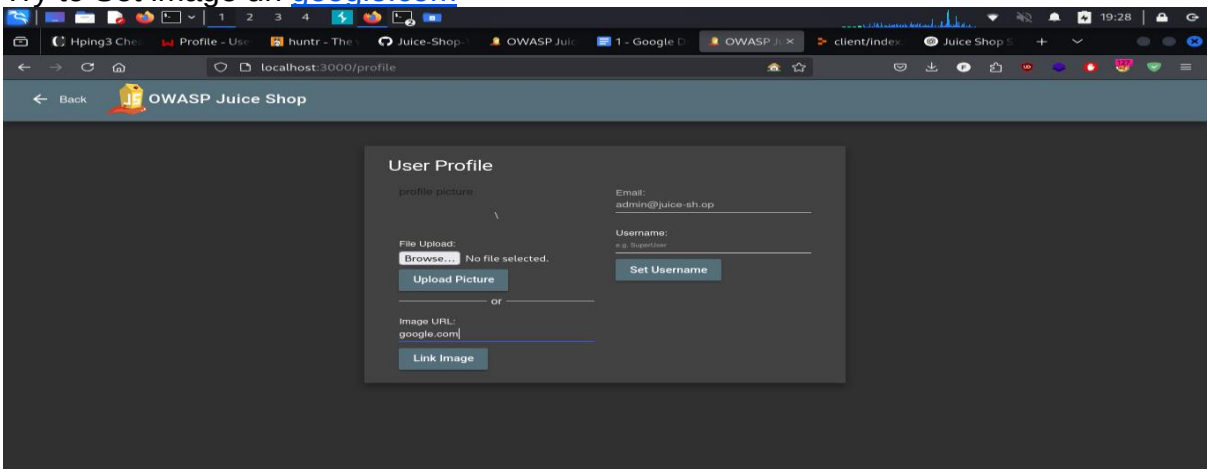
1. Whitelist URLs/Domains:
    a. Only allow requests to trusted, predefined external domains.
2. Block Local/Internal Requests:

Resources:

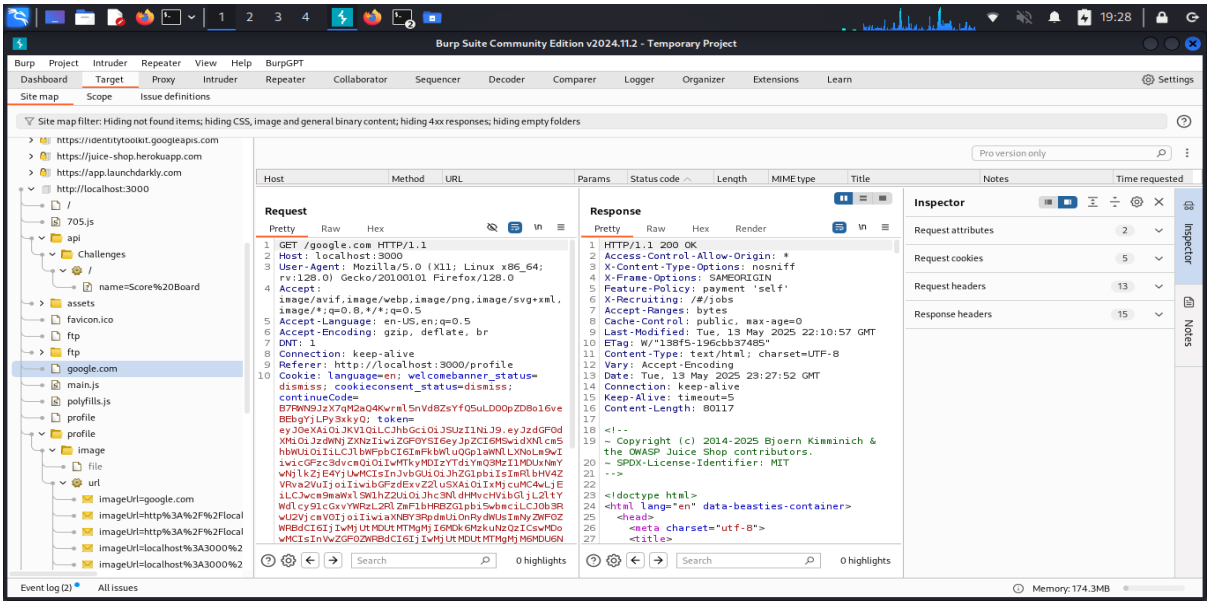
[What is SSRF (Server-side request forgery)? Tutorial & Examples | Web Security Academy]

Proof of concept (POC):

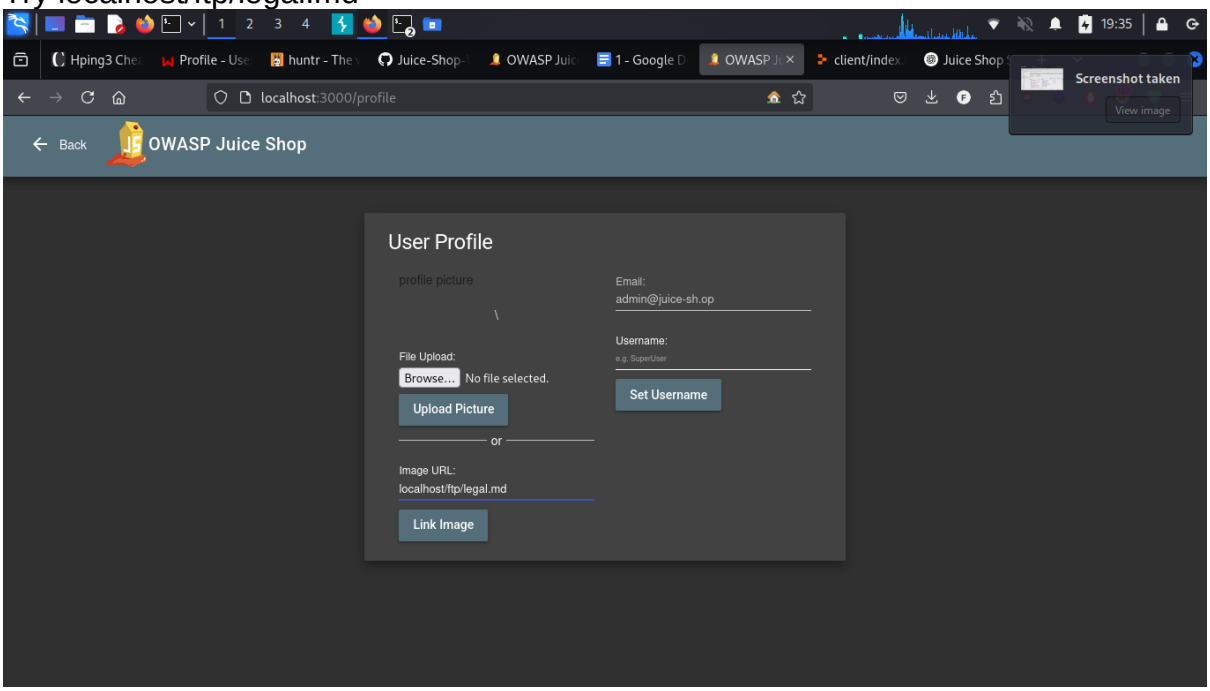
1. Go to /profile
2. Try to Set image url google.com



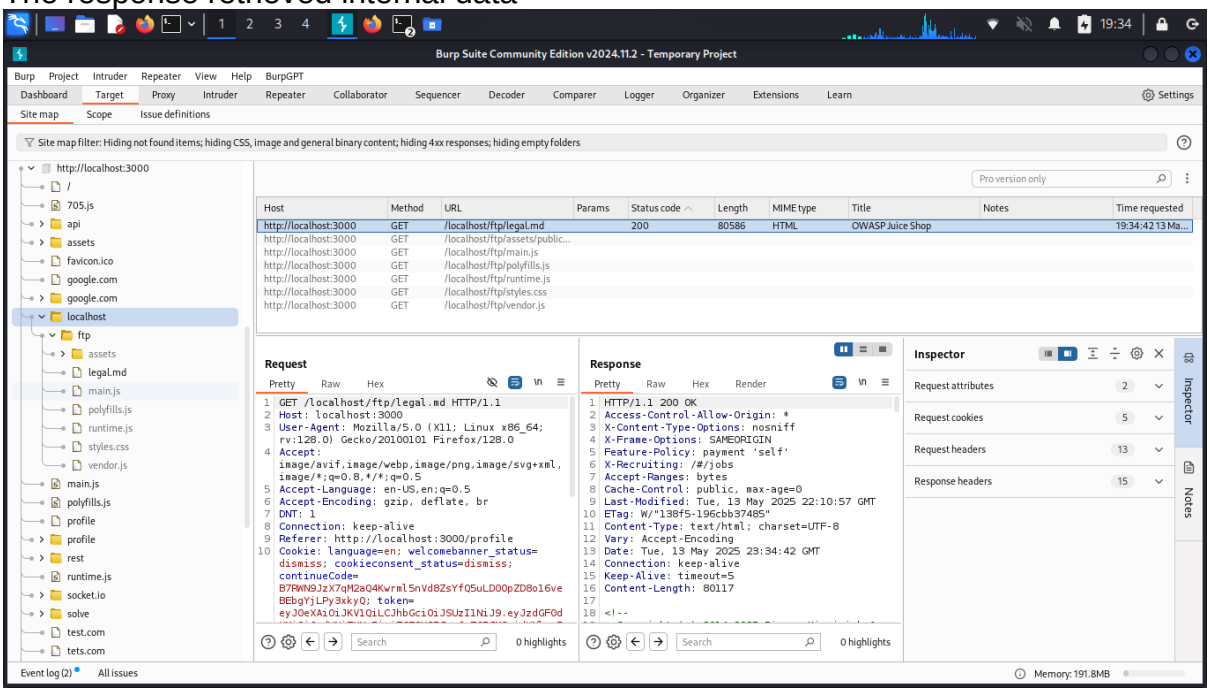
3. In burp it request google.com



4. Try localhost/ftp/legal.md



5. The response retrieved internal data



20.NOSQL

20.1 NOSQL Infiltration

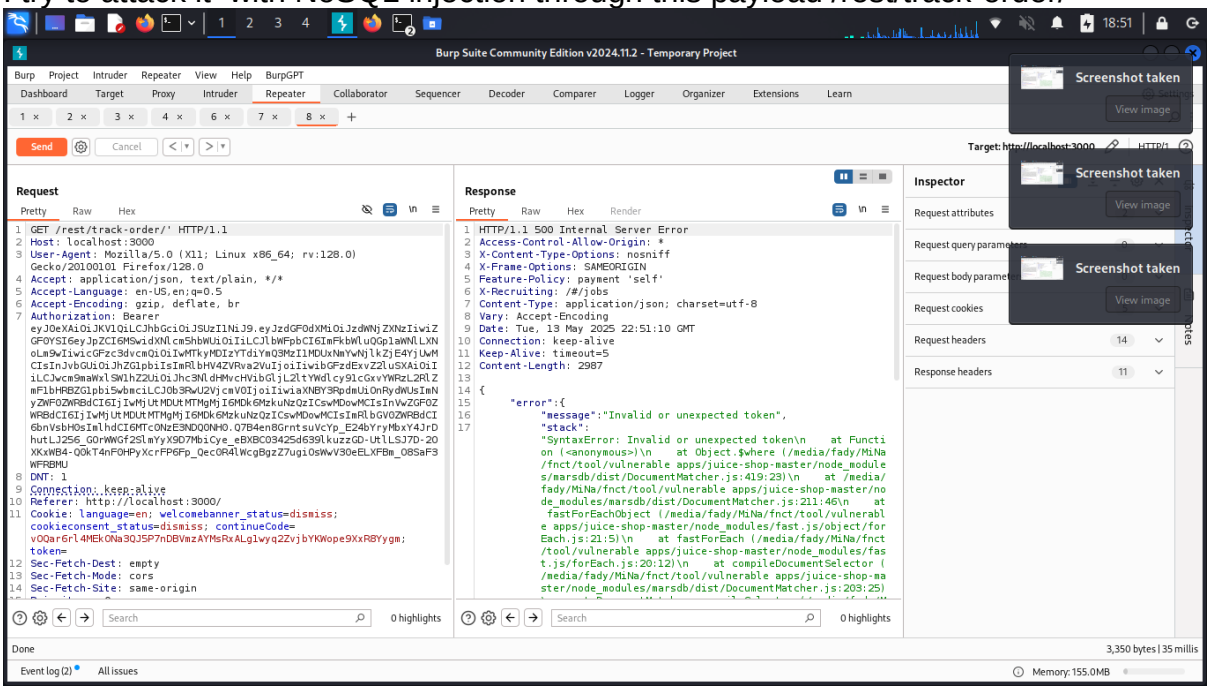
High

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

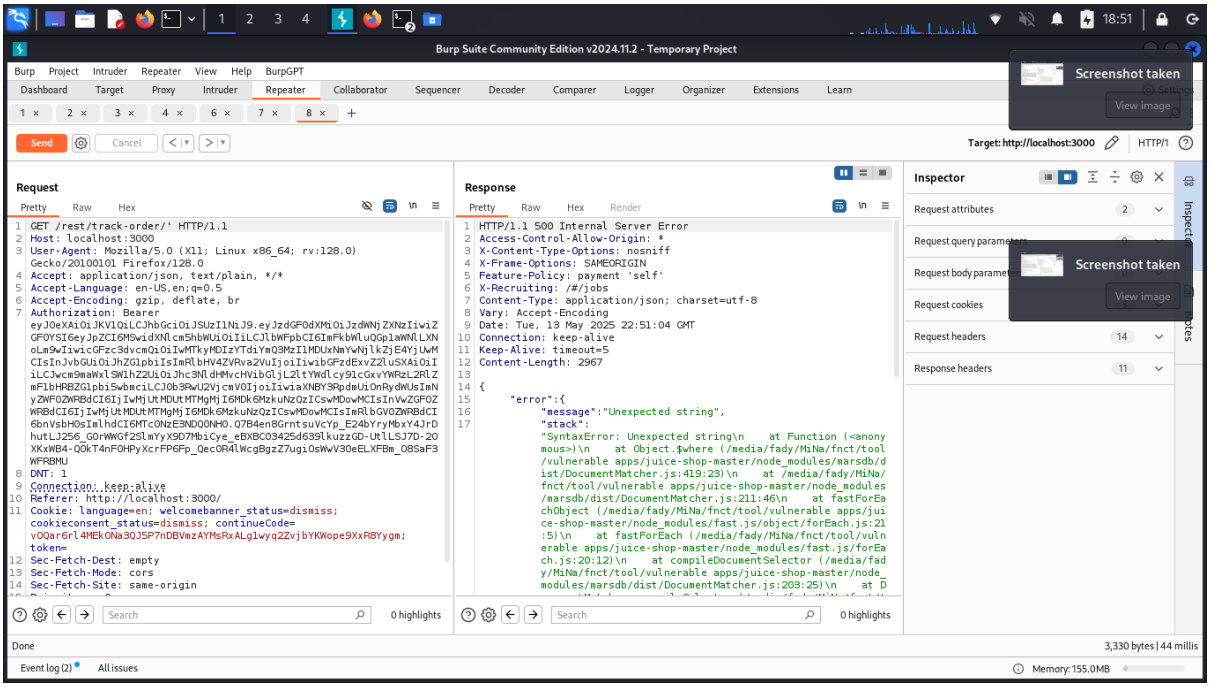
Resources:[NoSQL injection | Web Security Academy]

Proof of concept:

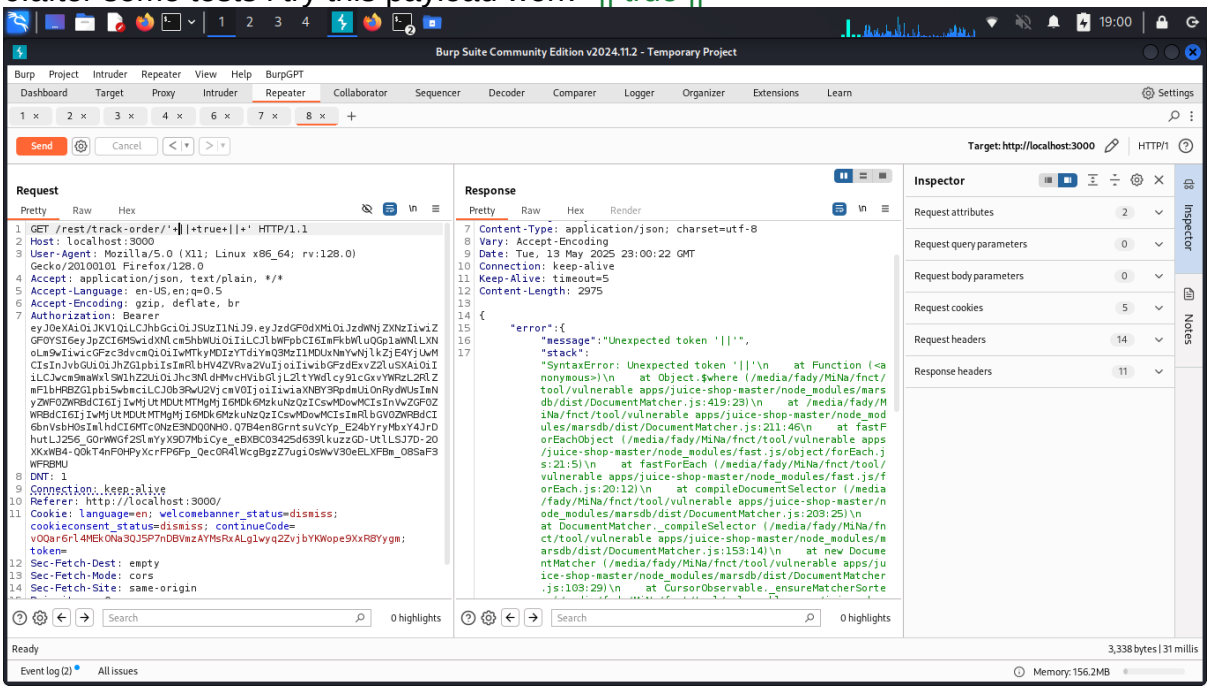
1. Go to http://localhost:3000/#/order-history
2. Analyse the the path how it work i notice that it fetch data through Endpoint: /rest/track-order/[id]
3. I try to attack it with NoSQL injection through this payload /rest/track-order/'



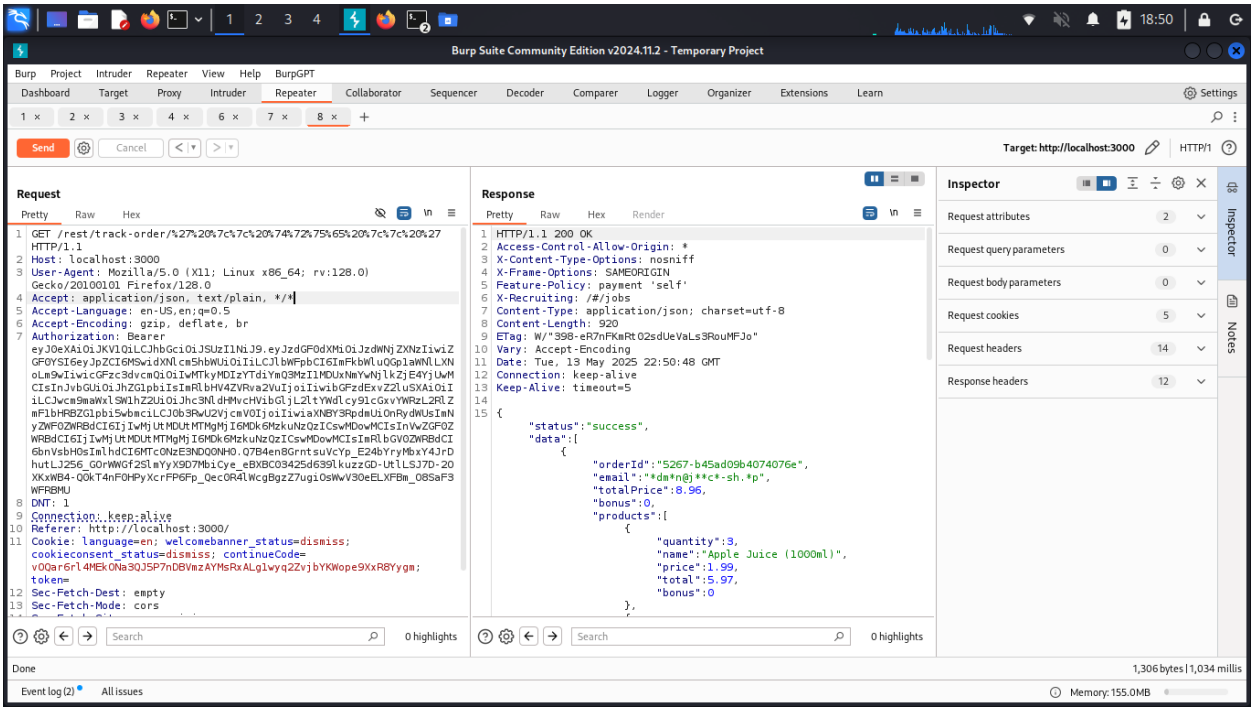
4.i try this payload /rest/track-order/’’



5.the error changed but stell 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
Request
GET /rest/track-order/%27%20%7c%7c%20%74%72%75%65%20%7c%7c%20%27 HTTP/1.1
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, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
Response
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Content-Length: 920
ETag: W/"398-eR7nFKmRt02sdUeVaLs3RouMFJo"
Vary: Accept-Encoding
Date: Tue, 13 May 2025 22:50:48 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{
"status":"success",
"data":[
{
"orderId":"5267-b45ad09b4074076e",
"email":"*dm*n@j**c*-sh.*p",
"totalPrice":8.96,
"bonus":0,
"products":[
{
"quantity":3,
"name":"Apple Juice (1000ml)",
"price":1.99,
"total":5.97,
"bonus":0
},
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers
Response headers
Done
1,306 bytes | 1,034 millis
Memory: 155.0MB