Digital Egypt Pioneers

وزارة الاتصالات
وتكنولوجيا المعلومات

Vulnerability Assessment and Penetration Test Report for OWASP Juice Shop

Group Code: CAI2_ISS3_S1

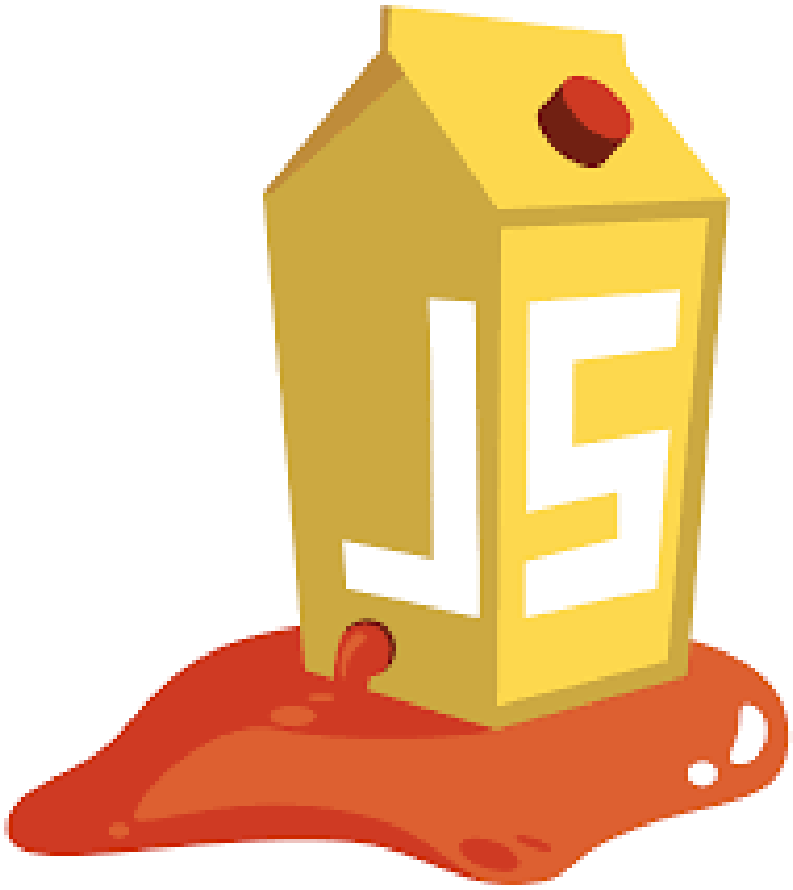


Table of Contents

1. Document Control

1.1 Description

The owners of Juice Shop commissioned DEPI to assess the security posture of their infrastructure by conducting a web application penetration test aligned with current industry best practices. The engagement followed a structured methodology consisting of the following phases:

- Planning – Define the scope and objectives in collaboration with the client and establish the rules of engagement.
- Discovery – Conduct comprehensive scanning and enumeration to uncover potential vulnerabilities, misconfigurations, and attack vectors.
- Exploitation – Validate identified vulnerabilities by safely exploiting them and exploring any resulting access for further weaknesses.
- Reporting – Compile detailed documentation of all findings, including confirmed vulnerabilities, attempted exploits, and an overall assessment of strengths and weaknesses in the security posture.

1.2 Document History

Date	Authors	Reviewer	Approver
14 MAY,2025	Ayat Bahii	Hesham Saleh	
	Mariam Ahmed		
	Mariam Fathi		
	Mai Hany		
	Haidy Nazmy		
	Helana		

2. Executive Summary

This report provides a detailed security assessment of the OWASP Juice Shop web application — a deliberately insecure platform maintained by OWASP for educational and penetration testing purposes. The assessment was conducted to simulate real-world attacks, identify vulnerabilities, and evaluate the application's resilience against modern threats, in alignment with the OWASP Top 10 risk categories.

Assessment Overview

The evaluation uncovered a broad range of vulnerabilities across multiple categories, including:

- Broken Authentication & Access Control
- SQL & NoSQL Injection
- Information Disclosure
- Cross-Site Scripting (XSS)
- Business Logic Flaws
- Security Misconfigurations
- Client-side Manipulations
- Deprecated and Insecure Interfaces
- CAPTCHA & Anti-Automation Weaknesses

Examples of Key Vulnerabilities

- Broken Anti-Automation (CAPTCHA Bypass)
  The feedback form CAPTCHA was validated only on the client-side, allowing penetration tester to bypass it and flood the system.
- SQL Injection (Full DB Schema Extraction)
  Improper sanitization in the search feature enabled full schema disclosure using UNION-based payloads.
- Access Control Bypass (Add Hidden Products)
  penetration testers could modify product IDs in intercepted requests to add unavailable or premium products to the cart.

- Cross-Site Request Forgery (CSRF)
  Profile updates could be triggered via external malicious sites due to missing CSRF protection mechanisms.
- File Upload Misconfiguration
  The complaint submission form allowed .xml files, exposing deprecated interfaces and raising risk of XXE attacks.

Severity Breakdown of Identified Vulnerabilities

Vulnerability Name	Severity
Login Admin	CRITICAL
DOM Cross-Site Scripting	CRITICAL
API-only XSS	CRITICAL
Reflected XSS	CRITICAL
Admin Registration	CRITICAL
CSRF	HIGH
View Another User's Basket	HIGH
Forged Review	HIGH
Reset Jim's Password	HIGH
Reset Bender's Password	HIGH
Login Amy	HIGH
Login MC SafeSearch	HIGH
Login Jim	HIGH
Database schema	HIGH
Ephemeral Accountant	HIGH
Login Bender	HIGH
NoSQL Manipulation	HIGH
Client side XSS Protection	HIGH
Weird Crypto	HIGH
Allow List Bypass	HIGH
Easter egg	HIGH
Christmas Special	HIGH
Zero Stars	HIGH
Payback Time	HIGH
Deprecated Interface	HIGH
CAPTCHA Bypass	HIGH
Bjoern's Favourite Pet	HIGH
NFT Takeover	HIGH

Forged Feedback	MEDIUM
Security.txt	MEDIUM
Exposed Metrics	MEDIUM
Forgotten Developer Backup	MEDIUM
Confidential Document	MEDIUM
Leaked Unsafe Product	MEDIUM
Bully Chatbot	MEDIUM
Repetitive Registration	MEDIUM
Expired Coupon	MEDIUM
Outdated Allowlist	MEDIUM
Meta Geo Stalking	MEDIUM
Empty User Registration	LOW
Mass Dispel	LOW
Privacy Policy	INFORMATIONAL

Implications of These Vulnerabilities

If exploited, these vulnerabilities could allow penetration testers to:

- Gain unauthorized access to user accounts or privileged functions.
- Extract sensitive information like internal schemas or user data.
- Manipulate business logic (e.g., add restricted items to basket).
- Launch phishing or malware delivery via redirect endpoints.
- Disrupt user trust and damage system integrity.

Conclusion and Recommendations

The assessment confirms the presence of multiple real-world security flaws in the application. Although this system is intentionally vulnerable for training, the findings mirror actual attack vectors that modern web applications face.

To address the discovered issues, we recommend:

- Enforcing robust server-side validation and authorization.
- Eliminating deprecated features and legacy file handlers.
- Strengthening CAPTCHA systems with session-bound tokens.
- Sanitizing all user inputs and using prepared statements.
- Applying CSRF protection, secure headers, and role-based access control.

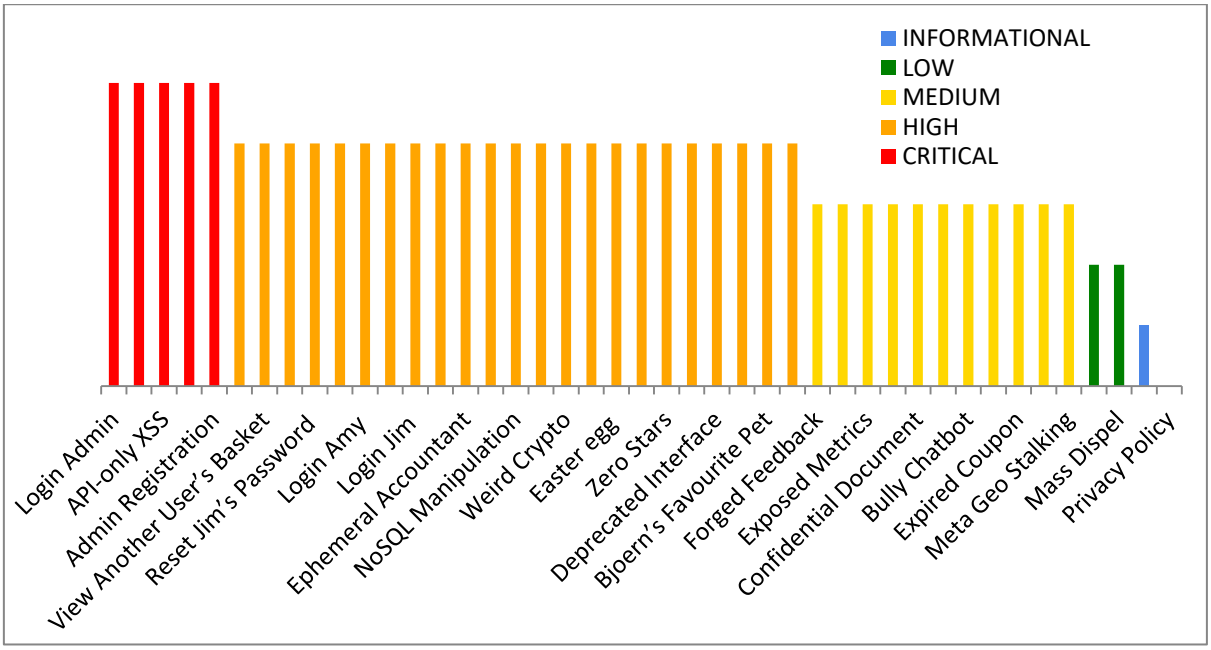
Check list

Category	Name	State
AppDOS	Application Flooding	Passed
	Application Lockout	Passed
AccessControl	Parameter Analysis	Failed
	Authorization	Failed
	Authorization Parameter Manipulation	Failed
	Authorized pages/functions	Failed
	Application Workflow	Failed
Authentication	Authentication endpoint should be HTTPS	Failed
	Authentication Bypass	Failed
Authentication.User	Credentials transport over an encrypted channel	Passed
	Default Accounts	Passed
	Username	Failed
	Password Quality	Failed
	Password Reset	Failed
	Password Lockout	Failed
	Password Structure	Passed
	Blank Passwords	Not Approved
Authentication.SessionManagement	Session Token Length	Not Approved
	Session Timeout	Not Approved
	Session Token Format	Passed
Configuration.Management	HTTP Methods	Failed
	Back-up Files	Failed
	Common Paths	Failed
	Language/Application defaults	Not Approved
Configuration.Management.Infrastructure	Infrastructure Admin Interfaces	Failed
Configuration.Management.Application	Application Admin Interfa	Failed
Error Handling	Application Error Messages	Failed
	User Error Messages	Failed
DataProtection	Sensitive Data in HTML	Failed
InputValidation	Script Injection	Failed

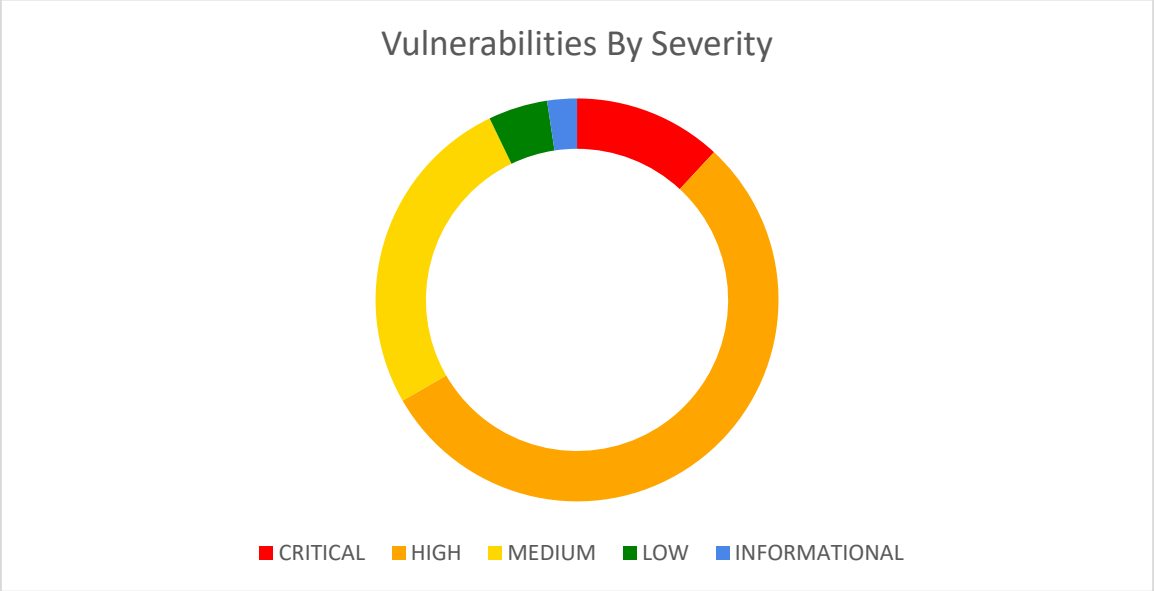
InputValidation.SQL	SQL Injection	Failed
InputValidation.OS	OS Command Injection	Passed
InputValidation.XSS	Cross Site Scripting	Failed
BufferOverflow	Overflows	Not Approved
	Stack Overflows	Not Approved

3.Graphical Summary

3.1 Finding Risk Rating



3.2 Finding Distribution



Web Site	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
OWASP Juice Shop	5	23	11	2	1

4.Scope

- In-Scope Components

Component	Description
Web Application	http://localhost:3000 (or designated hosted IP/domain)
Application Functionality	All features within the Juice Shop application, including login, registration, shopping cart, feedback system, administration panel, and challenge scenarios
API Endpoints	Any accessible RESTful APIs exposed by the application

- IP Addresses in Scope

IP Address / Host	Description
127.0.0.1 (or custom IP/host if deployed)	Local instance of OWASP Juice Shop

---

- Type of Testing
    - Type: Web Application Penetration Test
    - Methodology: Based on OWASP Testing Guide and SANS Penetration Testing Framework
    - Approach: Black-box and authenticated testing

- Out of Scope
    - Host operating system exploitation
    - Network-level attacks
    - Denial-of-Service (DoS/DDoS) simulations

5.Methodology

This penetration test was conducted in alignment with the OWASP Testing Guide v4 and other industry standards such as PTES and NIST SP 800-115. The purpose was to simulate real-world attack scenarios on the OWASP Juice Shop application and identify security weaknesses across the application stack.

The methodology followed a structured approach covering the following phases:

1. Planning and Scoping

- Defined the rules of engagement and obtained authorization for testing.
- Identified the scope of testing including the Juice Shop application functionalities, roles, and endpoints.
- Established testing boundaries, allowed attack types, and communication procedures.

2. Information Gathering (Reconnaissance)

- Collected publicly available data such as accessible directories (/robots.txt, /security.txt).
- Identified technologies and server configurations.
- Enumerated endpoints, hidden paths, and parameters using automated and manual tools.

3. Configuration and Deployment Management Testing

- Identified exposed development files, default configurations, and misconfigured endpoints.
- Tested for accessible backups, outdated components, and unnecessary services.

4. Authentication Testing

- Evaluated the login mechanism for brute force attacks, credential stuffing, and email enumeration.
- Tested password reset functionality for weak security questions and bypasses.
- Identified missing multi-factor authentication and CAPTCHA protections.

5. Authorization Testing

- Tested for Insecure Direct Object References (IDOR) to access other users' resources.
- Validated server-side authorization controls for admin sections and restricted functions.
- Attempted privilege escalation by modifying user roles during registration or session manipulation.

6 Session Management Testing

- Verified the security attributes of session cookies (HttpOnly, Secure, SameSite).
- Tested logout functionality, session expiration, and fixation risks.

7. Input Validation and Injection Testing

- Identified and exploited input validation issues including:
  - SQL Injection
  - NoSQL Injection
  - Cross-Site Scripting (XSS)
  - Local File Inclusion and XXE
- Used manual testing and tools like Burp Suite, SQLMap, and custom payloads.

8. Business Logic Testing

- Analyzed critical workflows for logic flaws, such as:
  - Abuse of expired coupons
  - Adding hidden products to basket
  - Bypassing rating limits (Zero Stars)
  - Gaining wallet credit using negative values

9. Client-Side and API Testing

- Discovered DOM-based XSS, Reflected XSS, and client-side CSRF vulnerabilities.
- Evaluated browser storage usage (sessionStorage/localStorage).
- Tested APIs for weak input validation and lack of access control.

10. Exploitation and Proof-of-Concept

- Safely exploited confirmed vulnerabilities to demonstrate business impact.
- Captured evidence such as unauthorized access, data leakage, and full account takeover.
- Maintained logs and screenshots for inclusion in the technical findings.

11. Reporting

- Documented each finding with the following structure:
  - Title and Severity
  - Description and Impact
  - Affected Components and Parameters
  - OWASP and CWE Mapping
  - Reproduction Steps and PoC
  - Recommended Remediation Actions

6. Technical Findings

6.1 Broken Access Control

6.1.1 Login admin

CRITICAL

Description:

A vulnerability was discovered in the authentication system that allows an atta penetration tester cker to log in to the administrator's account through a combination of email enumeration and brute force attack. The penetration tester was able to discover the admin's email address through a public product review and then performed a brute-force attack using a list of common admin passwords to gain unauthorized access.

When logged in as an administrator exposes a hidden administrative section that is not linked from the user interface.

Impact:

The penetration tester was able to:

- Identify the administrator's email address from publicly visible data.
- Successfully brute-force the admin password using a list of passwords.
- Log in as the administrator and gain full access to the backend functionality, which could lead to:
    - Full data access or manipulation.
    - Account takeover and privilege escalation.
- Discover the existence of the admin section by inspecting the frontend JavaScript (main.js).

Resource / References:

OWASP references:

- Authentication Cheat Sheet – OWASP
- Brute Force Attack – OWASP
- Wordlist used: password-list.txt-github

Vulnerability Location:

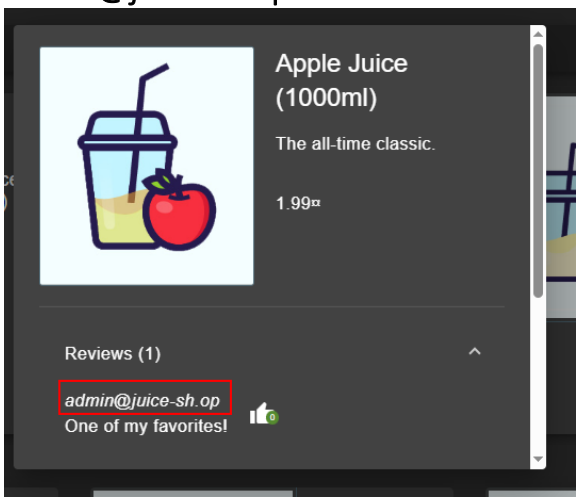
- **Login URL:** https://127.0.0.1:3000/#/login
- **Component:** Admin Dashboard / Panel
- **Accessed via URL:** https://127.0.0.1:3000 /#/administration
- **IP Address Used During Testing:** 127.0.0.1:3000/#/

Recommendations:

- Remove or mask email addresses in public content (e.g., reviews).
- Implement account lockout or rate-limiting after a number of failed login attempts.
- Enforce strong password policies for admin accounts.
- Add CAPTCHA or other bot prevention mechanisms on login forms.
- Use 2FA (Two-Factor Authentication) for all admin-level accounts.
- Avoid embedding sensitive information, such as hardcoded routes or credentials, in client-side code (e.g., main.js).

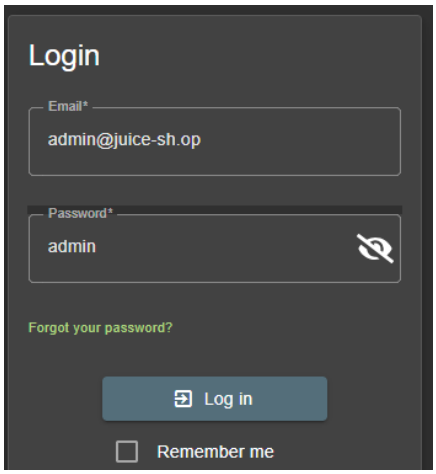
Proof of Concept (PoC):

1. Browse the product review section and noticed that the admin user had posted a review. The review showed the admin's email address, like: admin@juice-sh.op

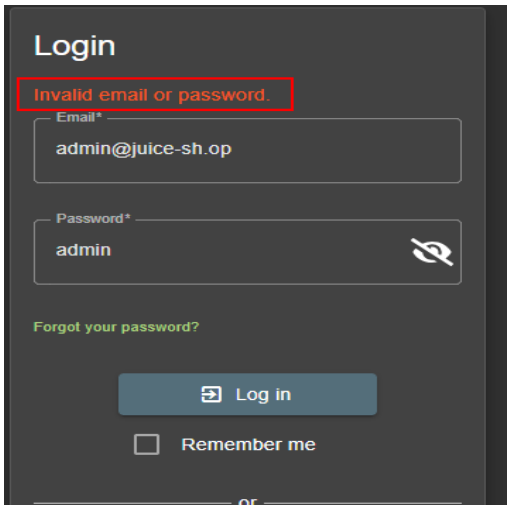


Now You had the admin's email, but you didn't know the password.

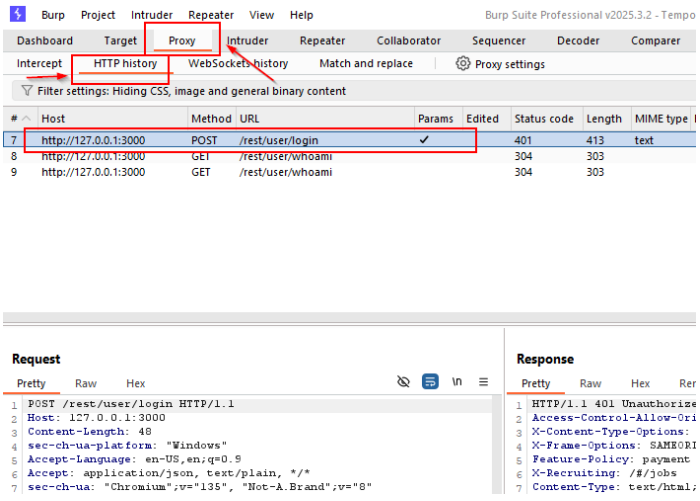
2. Attempted to log in with the administrator email and a random password.



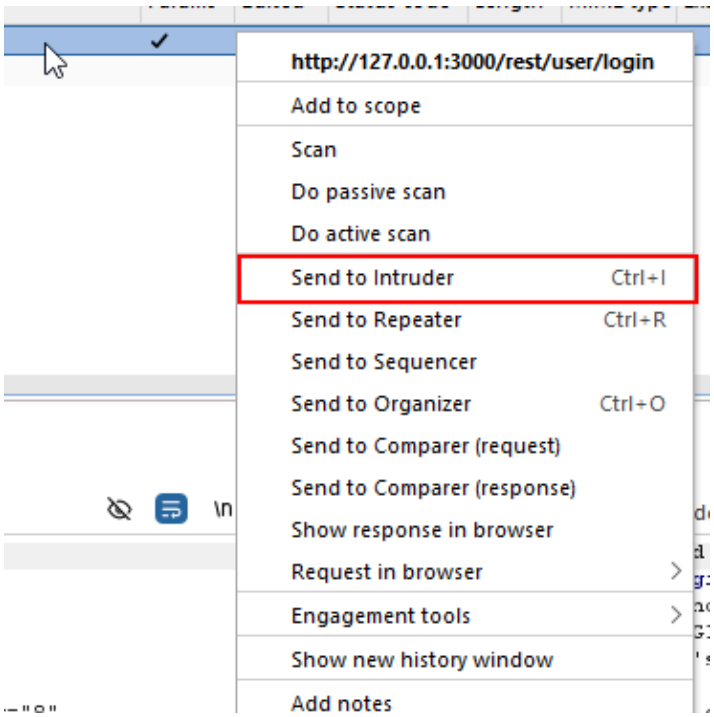
3. The password incorrect



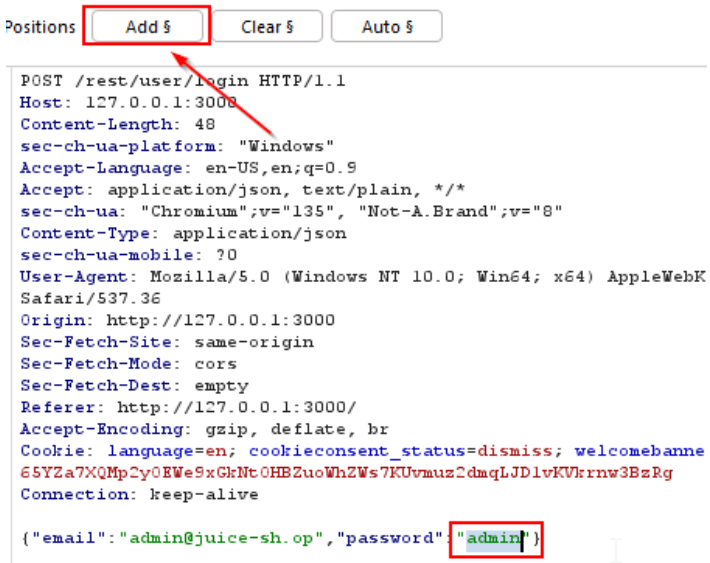
4. Intercepted the login request using Burp Suite to capture the request details necessary for the brute-force attack.



5. Then used Burp Intruder (within BurpSuite)

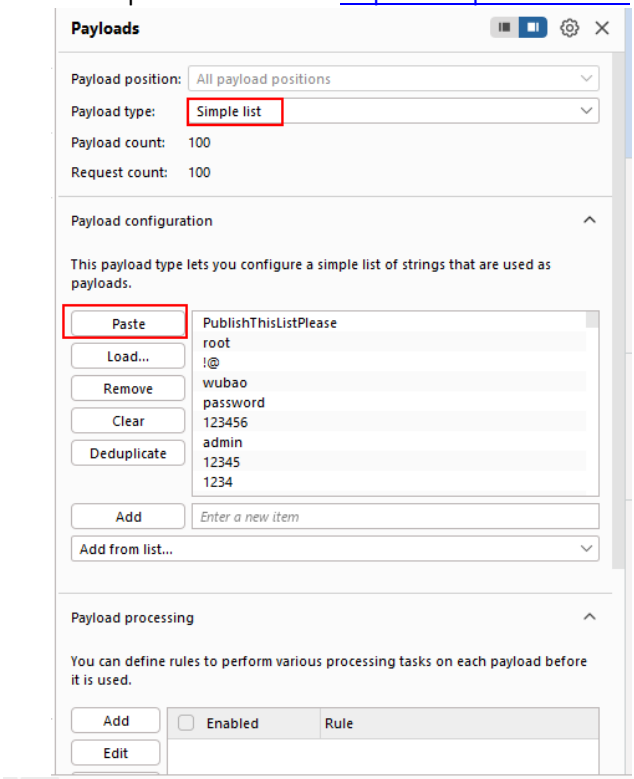


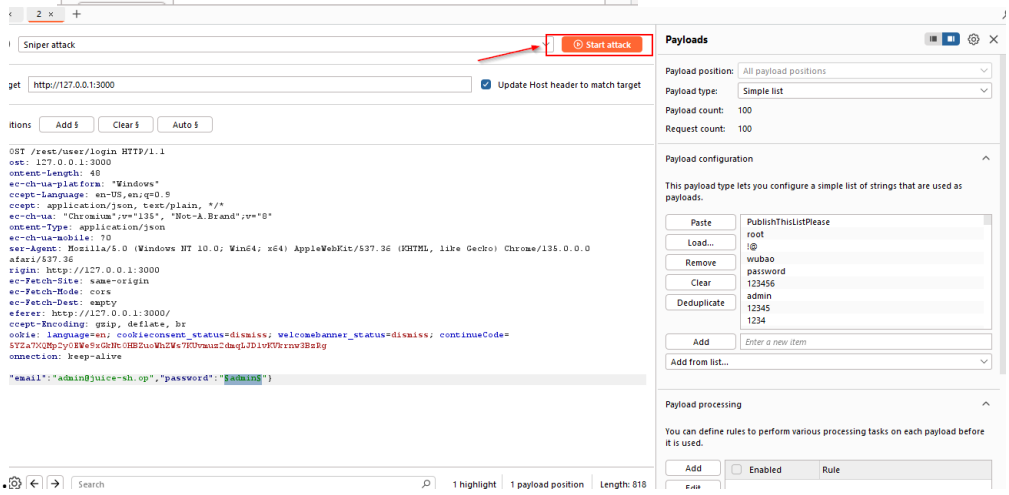
6. Configured Burp Suite's Intruder module to use a list of common passwords, replacing the placeholder password with each entry from the list during the attack.



I.

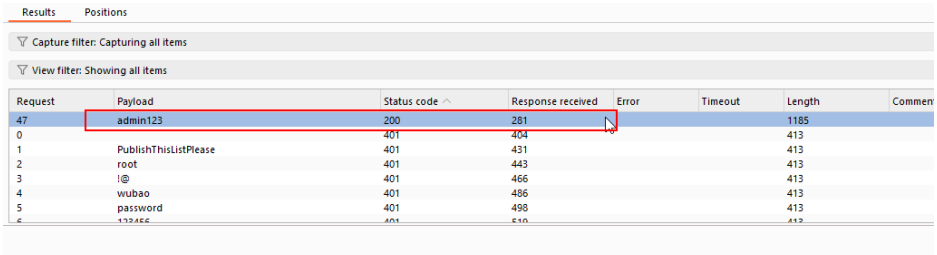
II.Use the password list: [top-100-passwords]



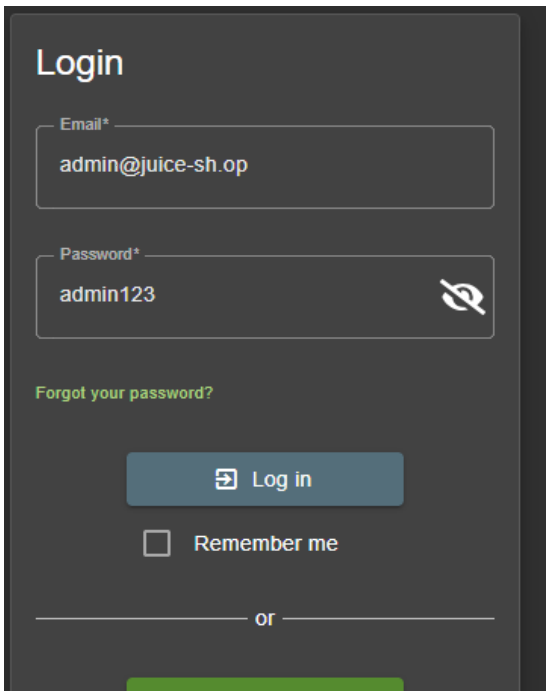


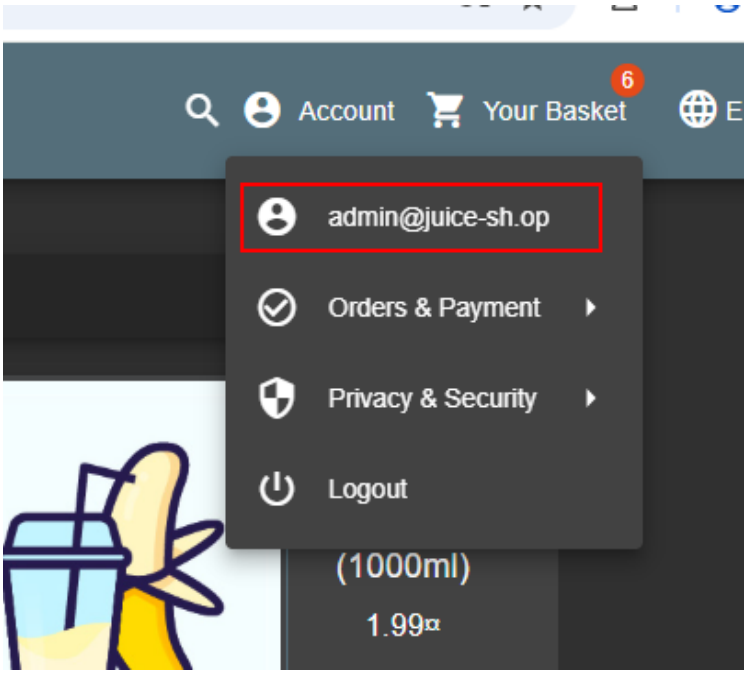
III.

Ran the brute force attack and monitored the responses for successful authentication indicators.



7. Successfully authenticated using the password: **admin123**



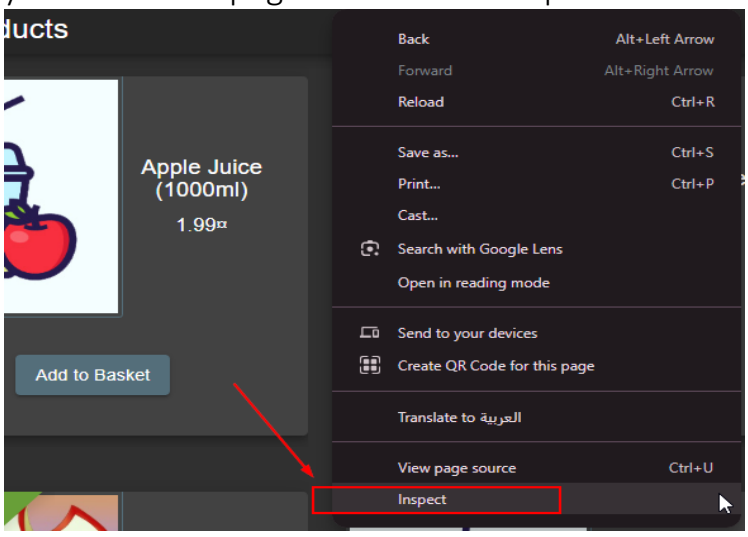


Scenario 2:

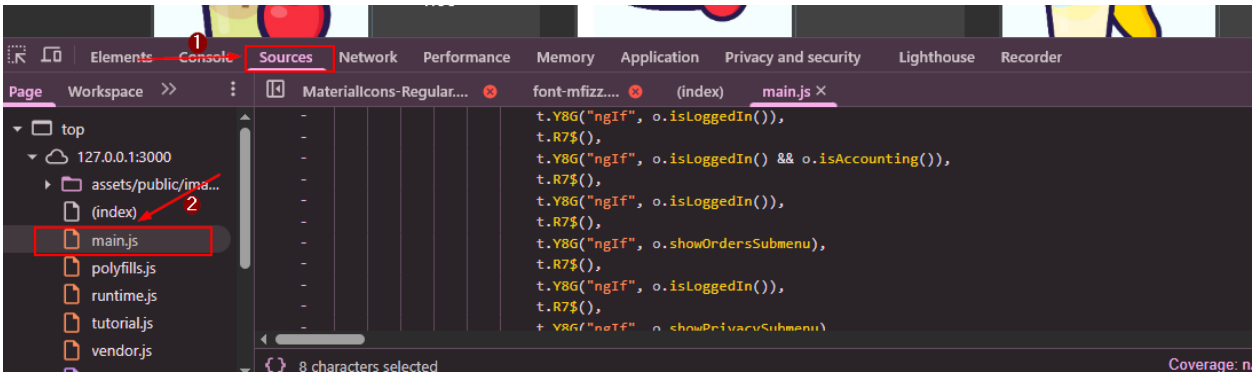
5.1.2Admin Section

Proof of Concept (PoC):

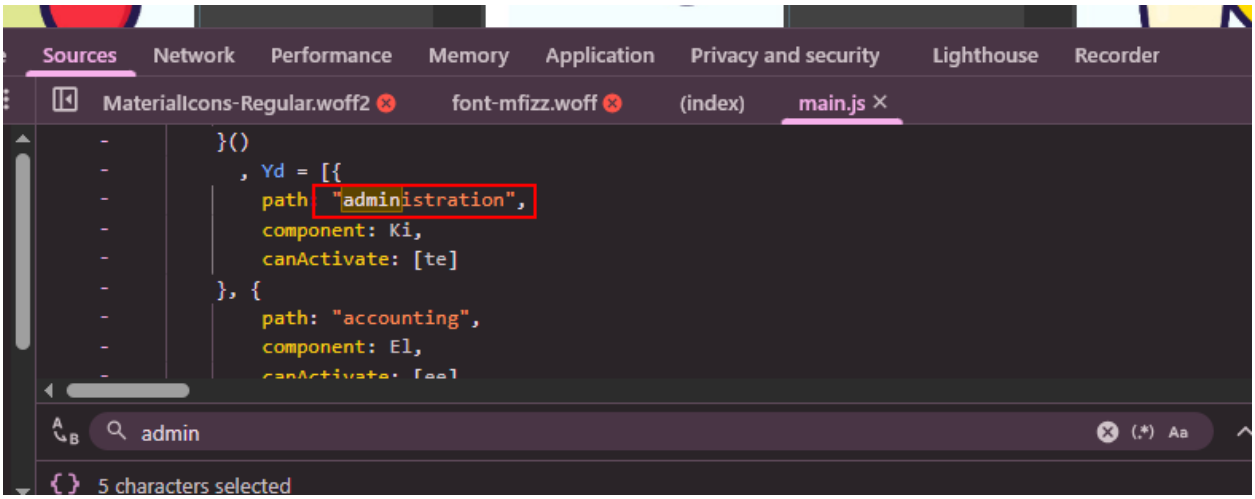
1. Wanted to check if there were any hidden parts of the app, so right-clicked anywhere on the page and clicked "Inspect".



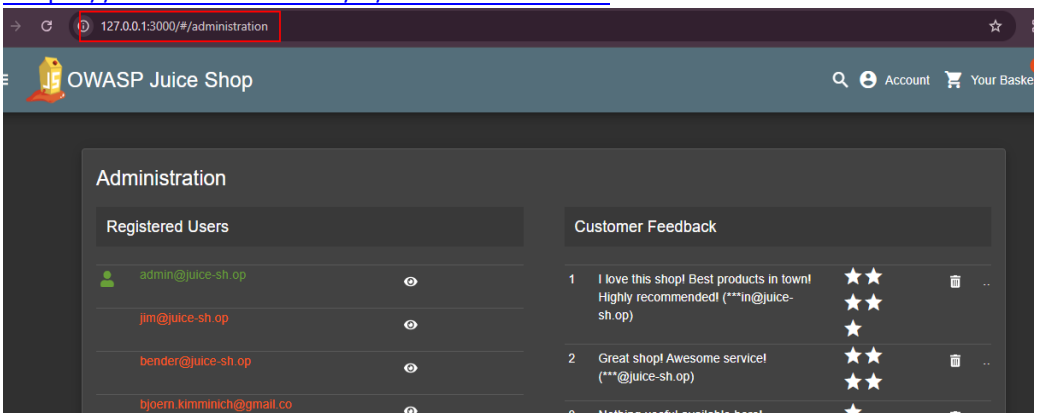
2. In the Developer Tools, Navigate to the Sources tab and opened the file called main.js, which contains the application logic.



3. Use the search function (Ctrl+F) and searched for the keyword: "admin"



4. Add it manually to the URL in the browser like this: https://127.0.0.1:3000/#/administration



This opened the Admin Section, even though there was no link for it anywhere in the interface.

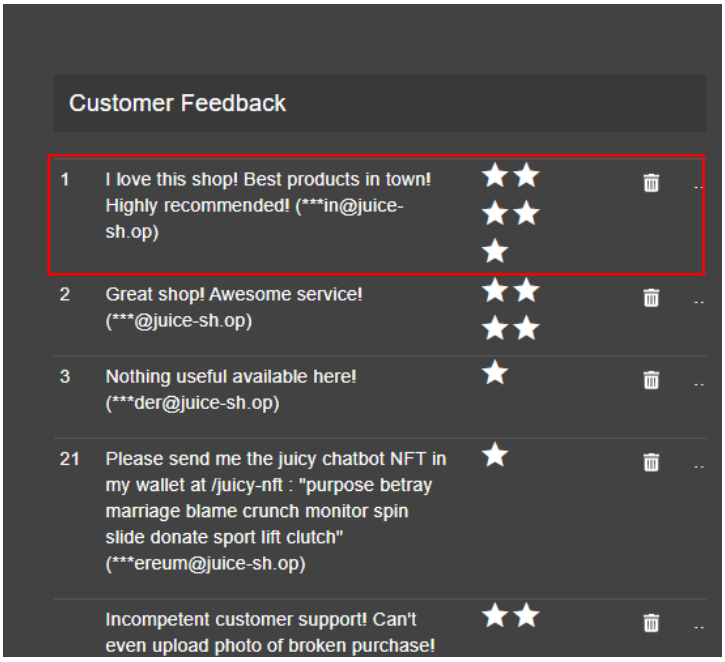
Scenario 3:

5.1.3 Five-Star Feedback

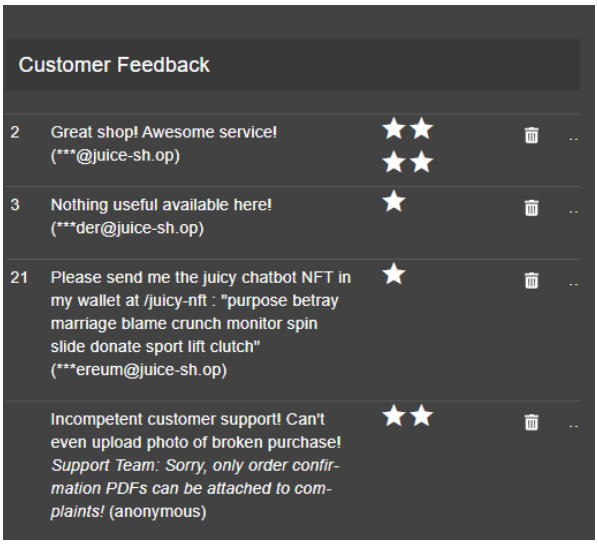
Proof of Concept (PoC):

After I Successfully accessed the **Admin Section displaying:**

- Registered users list.
- Customer feedback, including star ratings and email masks.



Remove Five-Star Feedback



6.1.4 Cross-Site Request Forgery (CSRF)

HIGH

Description:

The pentester found a **Cross-Site Request Forgery (CSRF)** vulnerability was discovered on the **profile update page** of the application in the section of , specifically in the functionality responsible for changing the user's **username**.

The application allows users to update their username by sending a POST request to a profile endpoint. As a result, the pentester craft a **CSRF payload** on an external site which, when visited by an authenticated user, causes their username to be changed **without their knowledge or consent**.

This vulnerability can be exploited silently by tricking a user into visiting a malicious site, allowing the penetration tester to manipulate user profile data.

Impact:

- Tricking a user into visiting a malicious site, allowing the penetration tester to manipulate user profile data.
- Unauthorized update of the victim's username.

Vulnerability Location:

- **Page:** 127.0.0.1/profile
- **Affected Parameter:** username

CVE / OWASP Reference:

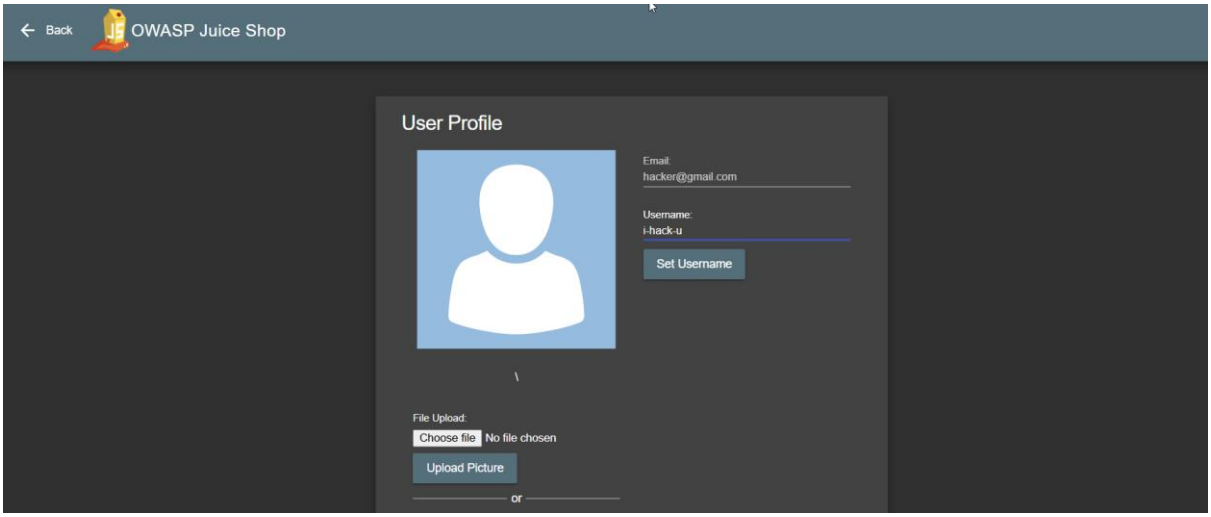
- OWASP A01:2021 – Broken Access Control (CSRF is related here)
- OWASP CSRF Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

Recommendations:

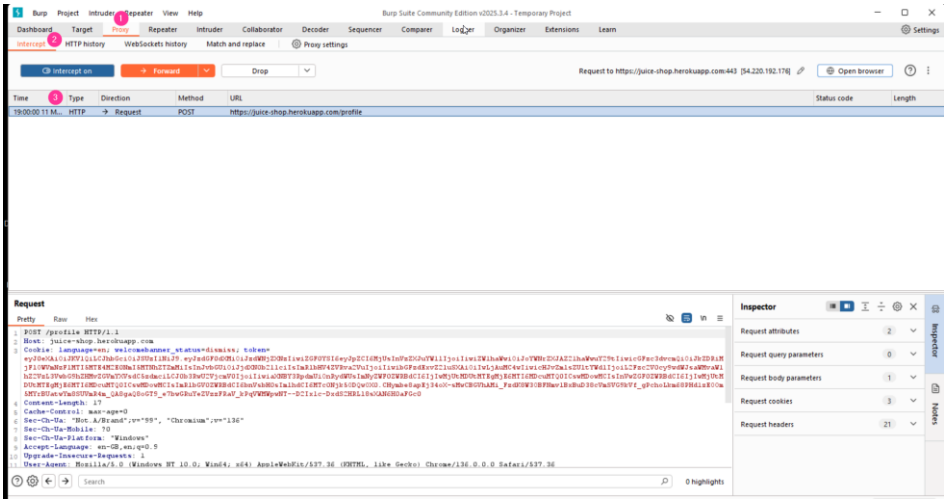
1. **Implement CSRF Tokens:**
   Use anti-CSRF tokens in all forms that change user data. These tokens must be unique per session and verified on the server.
2. **Use SameSite Cookies:**
   Set cookies with SameSite=Strict or SameSite=Lax to block cross-origin POST requests.

Proof of Concept:

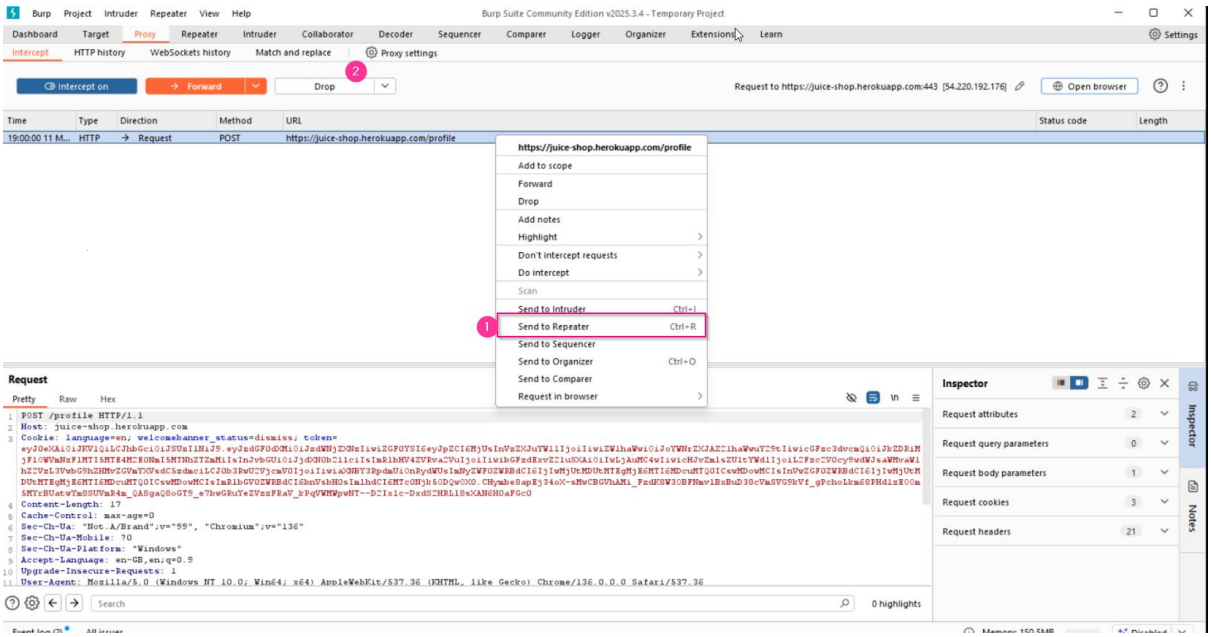
1. Open profile page of the user and write a name in the username field:



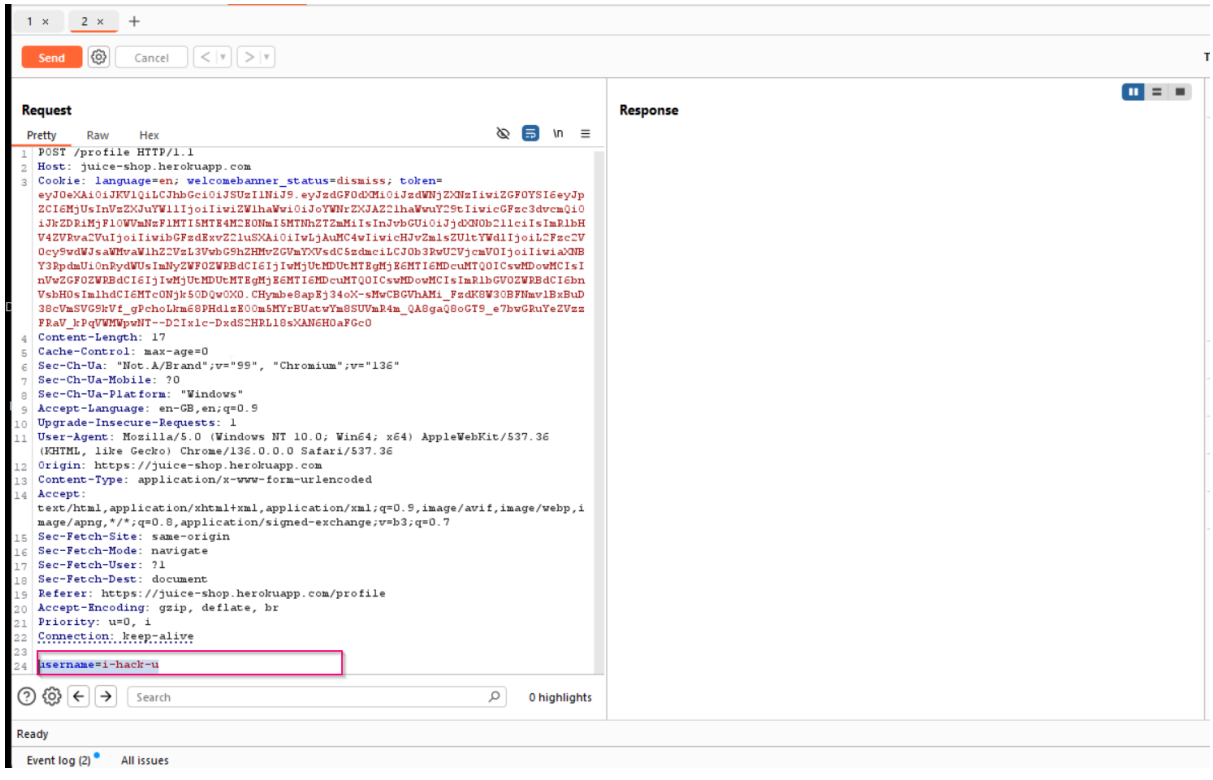
2. Open Burp suite, go to Proxy tap and make intercept on



3. Send the request to Repeater and then drop the request



4. Open the Repeater and note here is no csrf-token send with the request

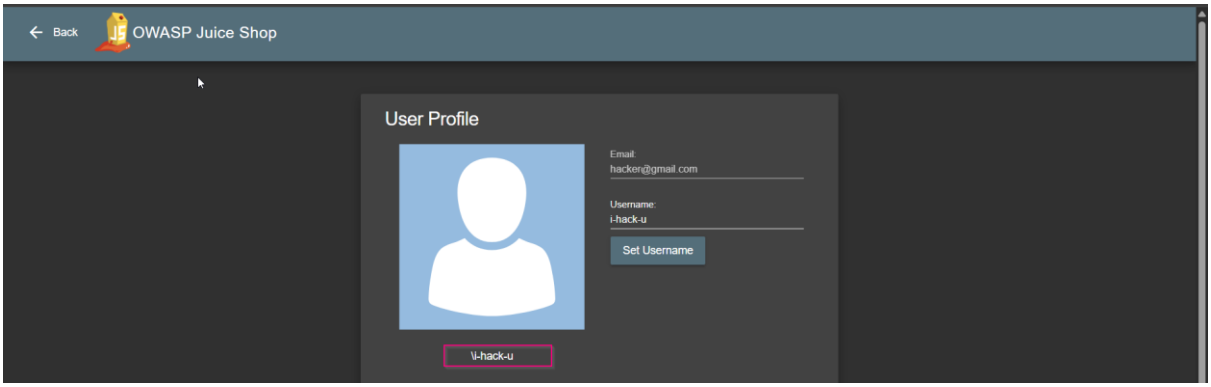


5. Use this code to make fake page

```
<html>
 <body>
  <form action="https://juice-shop.herokuapp.com/profile"
method="POST">
    <input type="hidden" name="username" value="i-hack-u" />
    <input type="submit" value="Click me!" />
  </form>

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

6. When the victim open the fake page his username will change directly



6.1.5 View Another User's Basket

HIGH

Description:

The application fails to properly enforce access controls on user-specific resources, particularly the shopping basket. The basket ID (bid) is stored in the browser's **sessionStorage**, and changing this value directly allows an penetration tester to view **other users' shopping baskets**. This indicates the backend is relying solely on the bid value for access control without validating the user's session or ownership of the basket.

Impact:

- **Privacy Violation:** An penetration tester can access other users' private shopping data.
- **Unauthorized Access:** This allows for horizontal privilege escalation.
- **Potential Manipulation:** In some cases, the penetration testers may be able to modify another user's cart, leading to fraud or data integrity issues.

Vulnerability Location:

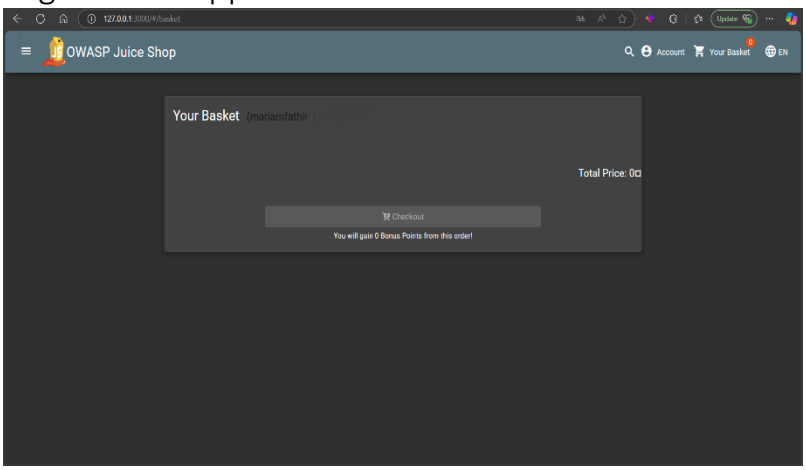
- **Component:** Shopping Basket (/basket)
- **Parameter Affected:** bid stored in sessionStorage

Recommendations:

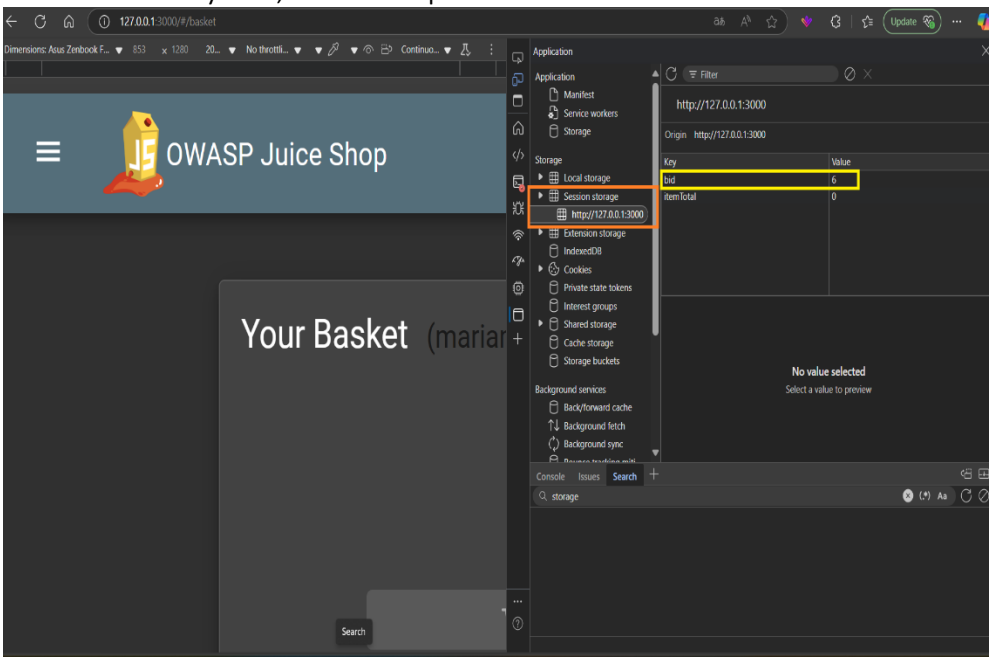
1. **Implement Proper Access Control:**
    - On the server-side, verify that the requesting user is the actual owner of the basket ID before displaying any data.
2. **Avoid Client-Controlled Identifiers:**
    - Never rely solely on client-side identifiers like bid stored in sessionStorage or localStorage.

Proof of Concept (PoC):

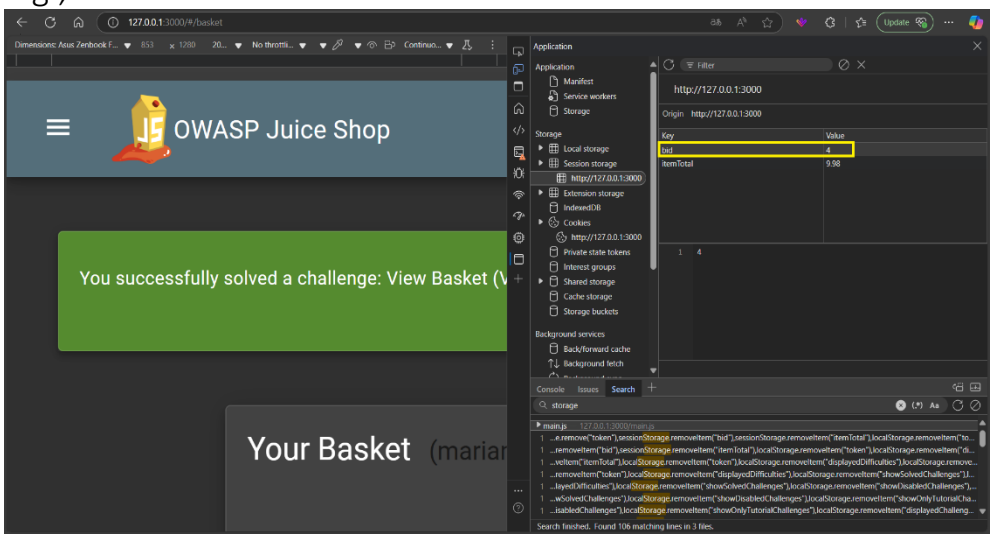
1. Log into the application with a valid user account and view his basket.



2. Navigate to the **Basket** page and open **Developer Tools > Application > sessionStorage**
   Locate the key bid, for example:bid = 6



3. Change the value of bid manually to another known or guessed value, e.g., bid =



4.Refresh the basket page. The application displays the contents of a different basket — confirming that access control checks are missing or improperly enforced.

6.1.6 Forged Review

HIGH

Description:

The application fails to properly enforce access control during feedback submission.
A Pentester can intercept a feedback submission request using a proxy (e.g., Burp Suite) and modify the **email** field or user identifier to impersonate another user.
The system incorrectly trusts the client-side data, allowing the feedback to be posted using the forged email address without authentication or verification.

Impact:

- **Identity Impersonation:** penetration testers can post feedback under another user's identity.
- **Loss of Trust:** Application users may lose trust in the platform due to possible manipulations.

Vulnerability Location:

- **Endpoint:** http://127.0.0.1:3000/#/contact (or the feedback page)
- **Component:** review Submission Form

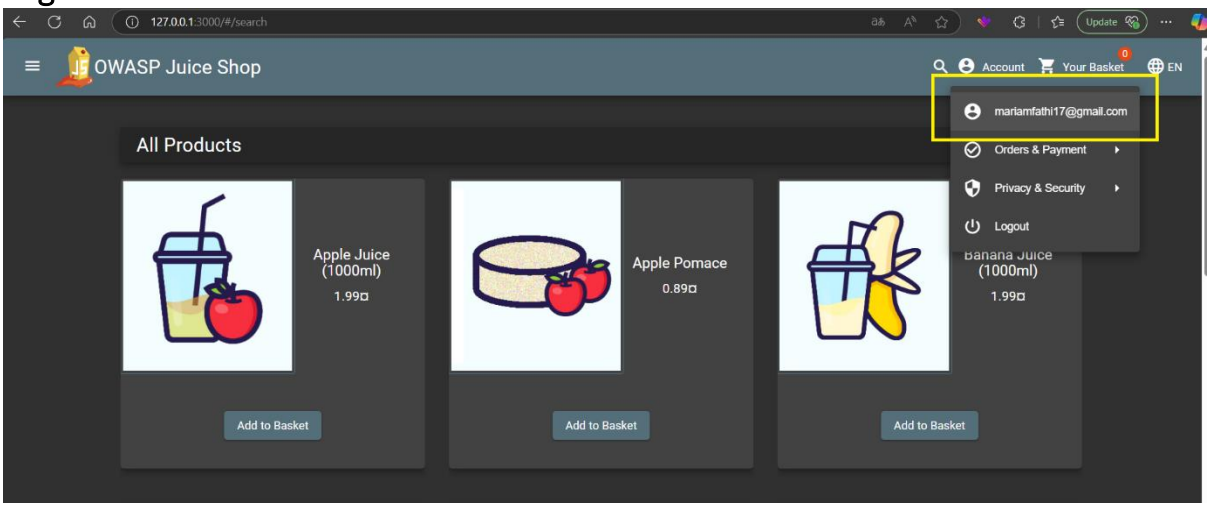
Recommendations:

1. **Enforce Server-Side User Validation:**
    - The server should always bind feedback to the authenticated session user, not based on input fields from the client side.
2. **Ignore User Identity Sent from Client:**
    - Remove any client-side control over user identifiers like email during sensitive actions.
3. **Implement Authorization Middleware:**

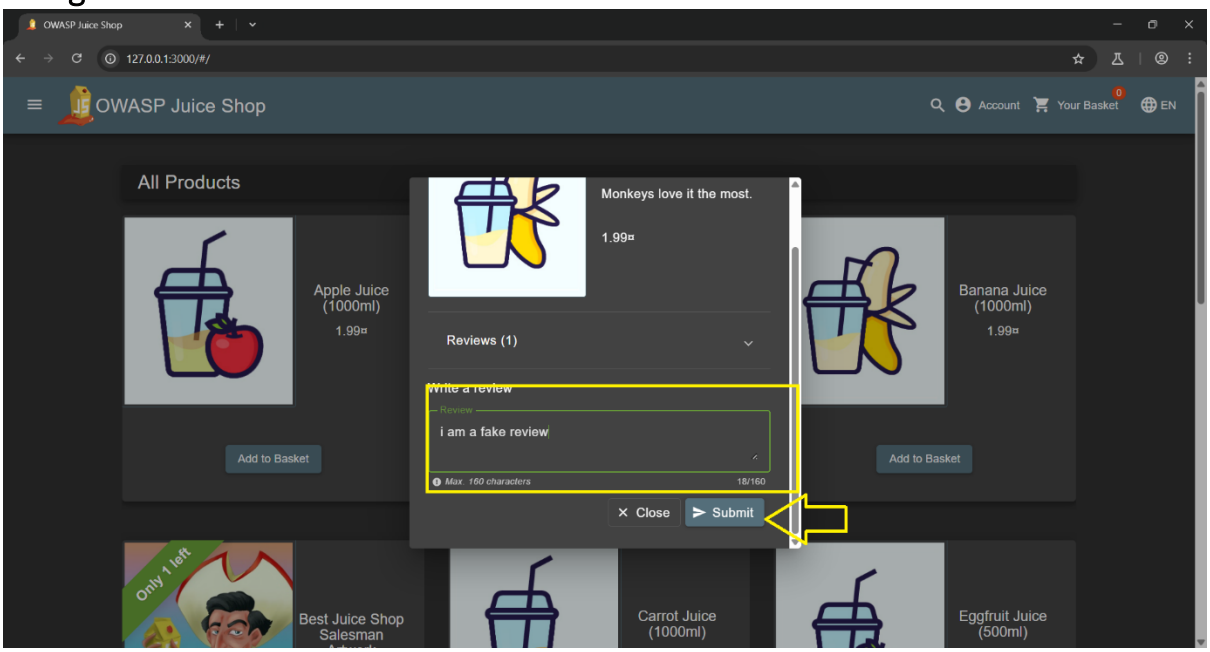
- Ensure that only the authenticated user's identity is used for feedback posting.

Proof of Concept (PoC):

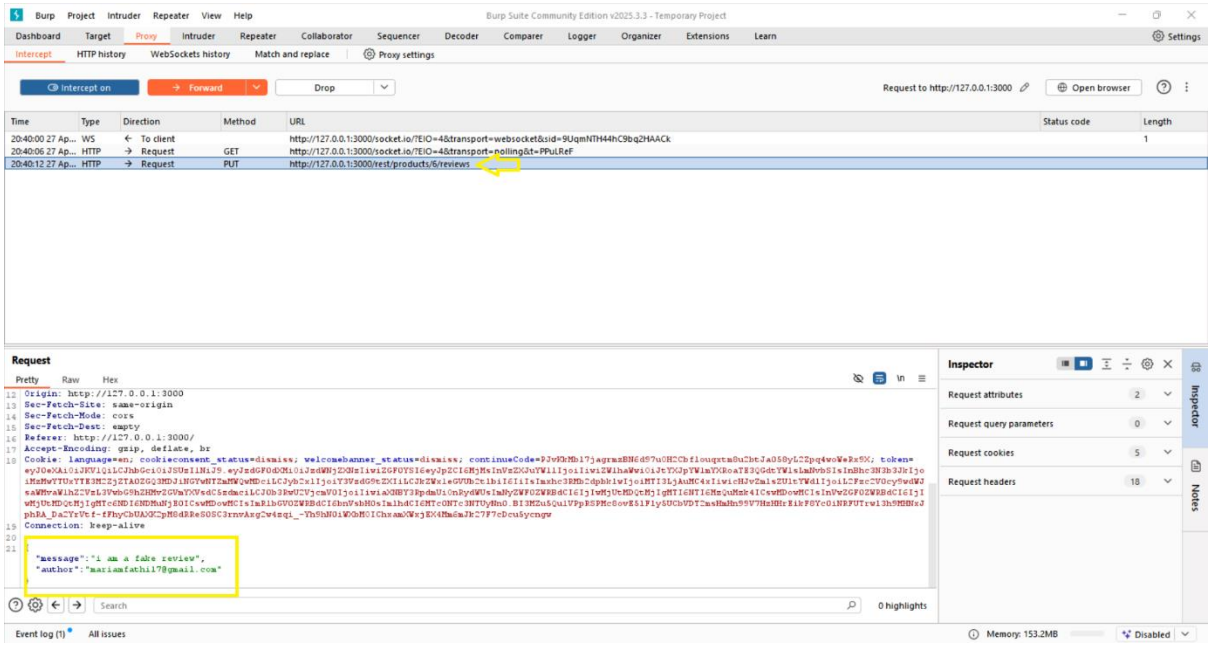
1. **Login** with a valid user account .



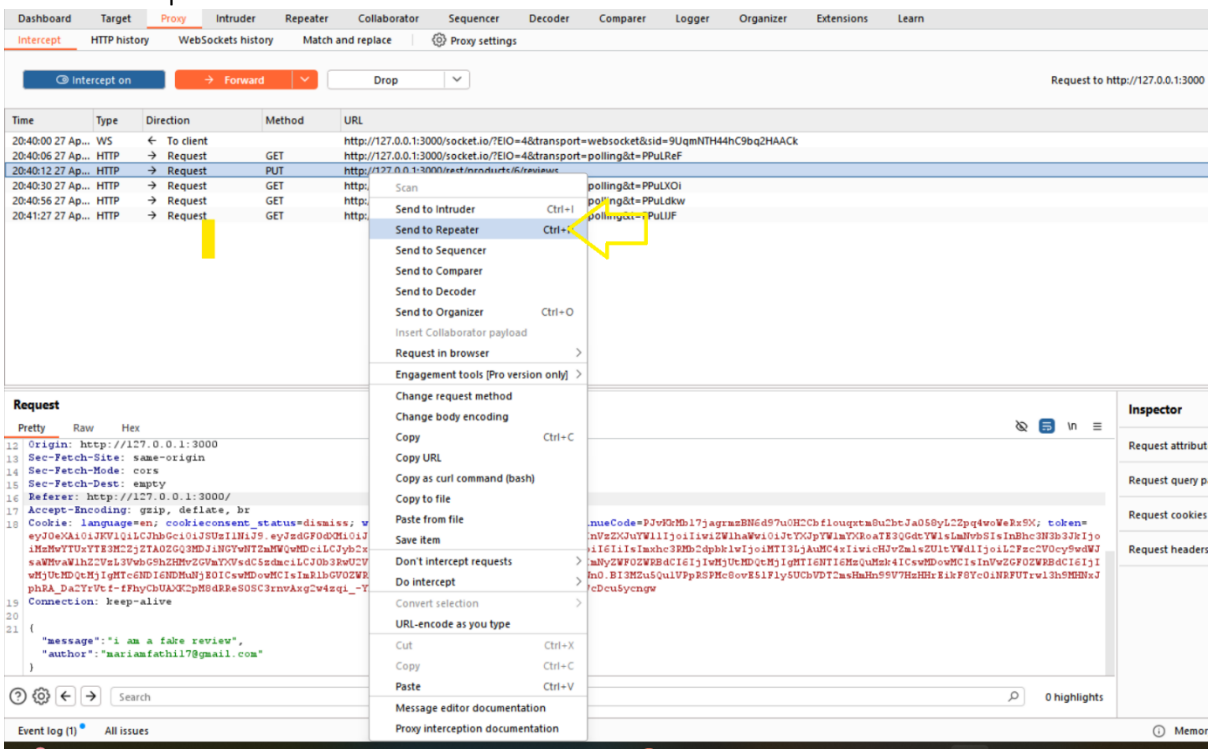
2. **Navigate** to the review submission form.



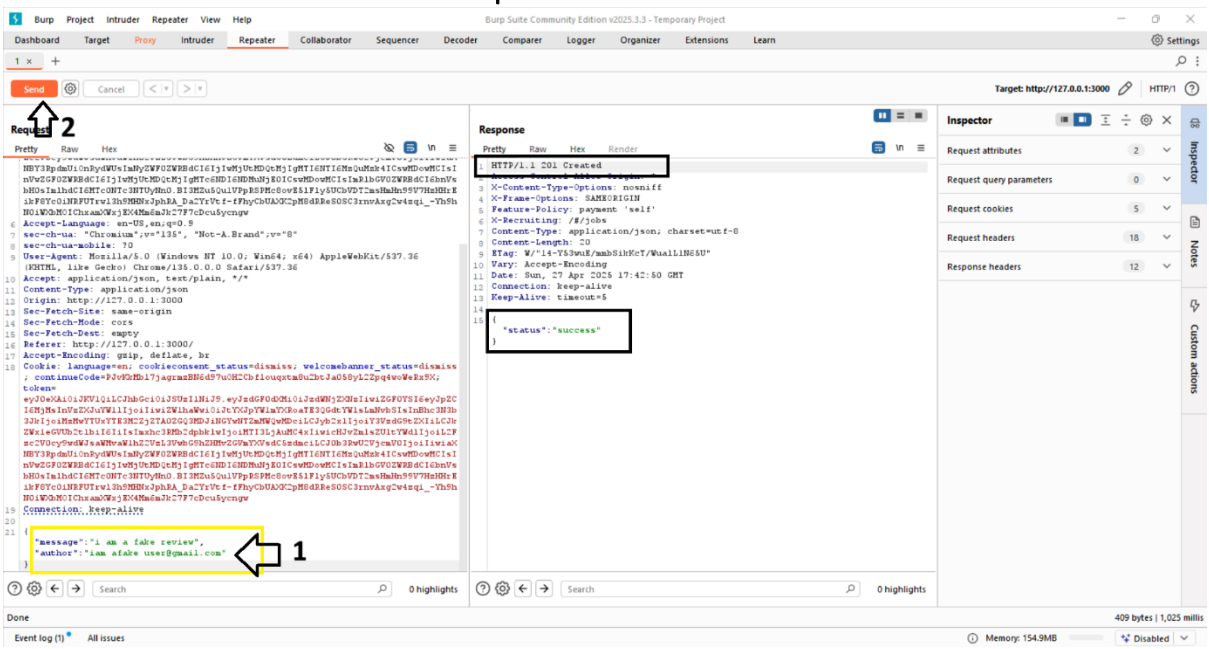
3. **Enable Intercept** in Burp Suite and **Submit Feedback** and capture the request before it reaches the server.



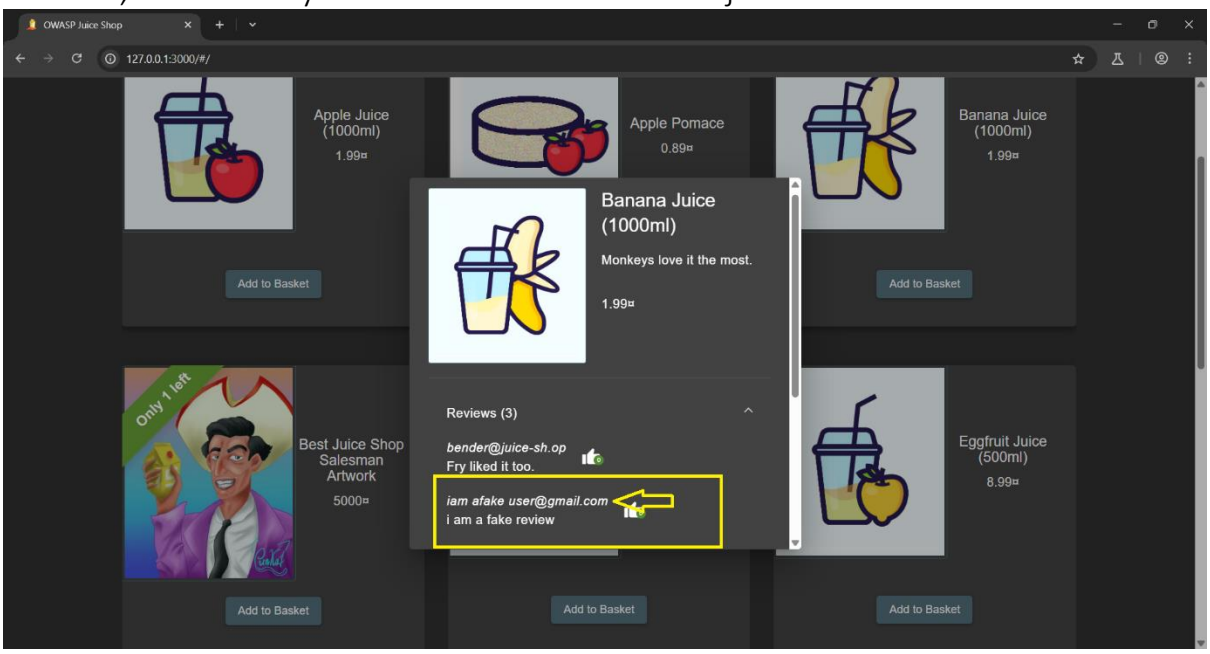
-send to repeater



4. **Modify the Request**: Change the email field (or any identifier) to another user's email and **Forward the Request** to the server.



5. The feedback is successfully posted and displayed with the forged email address, without any server-side validation or rejection.



6.1.7 Reset Jim's Password

HIGH

Description:

The "Forgot Password" mechanism for user accounts relies on a weak security question that can be easily brute-forced. In this case, user Jim had a security question:"**Your eldest sibling's middle name?**"

This type of personal question, especially for a public figure, is inherently insecure and vulnerable to brute force or OSINT techniques.

By brute-forcing the possible answers using a common middle-names.txt wordlist, a penetration tester was able to **successfully reset Jim's password** without needing access to his current password.

Impact:

The penetration tester was able to:

- Fully reset a target user's password.
- Gain unauthorized access to their account.
- Perform privilege escalation or access internal user data depending on the user role (Jim is a known public figure in the app context).
- Use the account to launch further attacks such as internal browsing, coupon abuse, or privilege escalation (if applicable).
- This undermines authentication integrity and shows the application relies on insecure password reset mechanisms.

Resource / References:

- OWASP: Authentication Cheat Sheet
- Wordlist used: middle-names.txt – GitHub

Vulnerability Location:

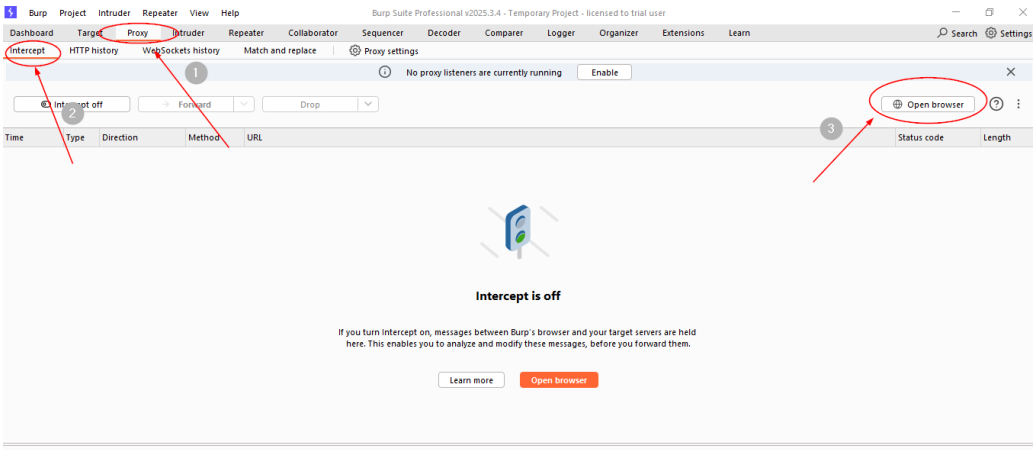
- **Page:** 127.0.0.1:3000/#/forgot-password
- **Test IP:** 127.0.0.1:3000/#/

Recommendations:

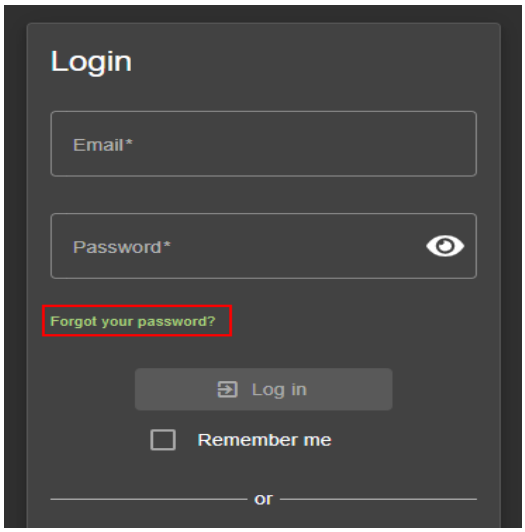
- Remove security questions entirely or replace them with **stronger MFA (Multi-Factor Authentication).**
- Do not use publicly available information as password reset mechanisms.
- Introduce rate-limiting or lockouts after a number of failed security question attempts

Proof of Concept (PoC):

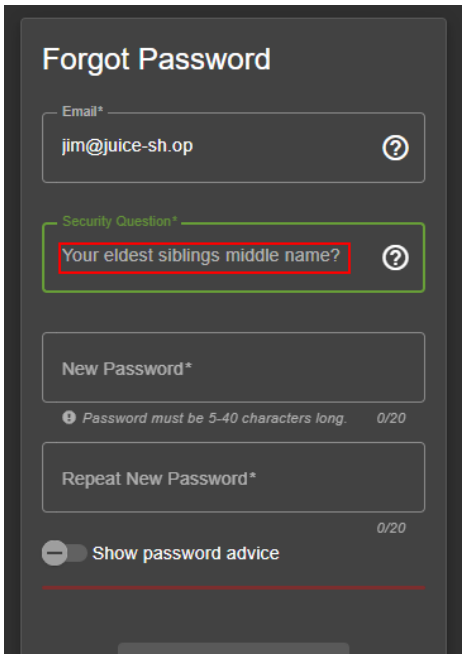
1.Open Burpsuite and open browser



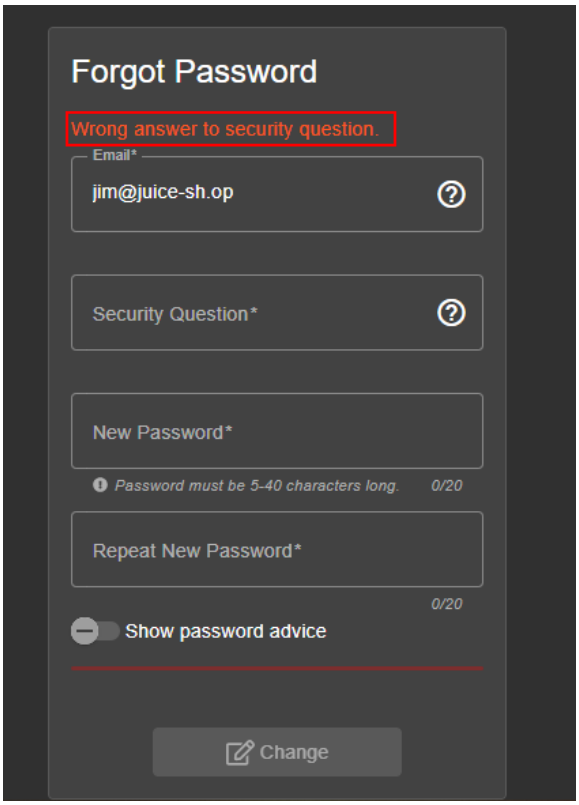
2.Went to the Login Page, clicked on **"Forgot Your Password?".**



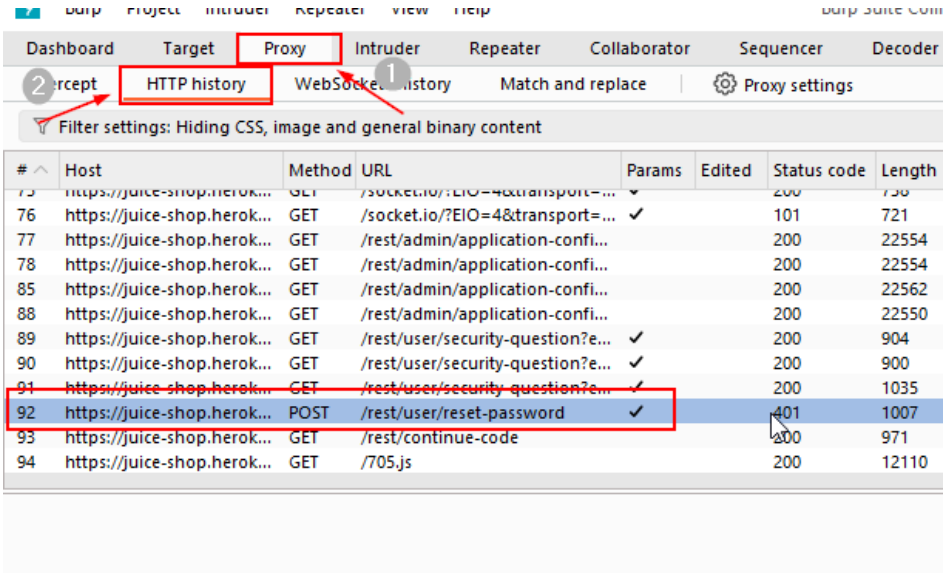
3.In the form, enter Jim's email address: **jim@juice-sh.op**
There is a security question: "**Your eldest siblings middle name?**"



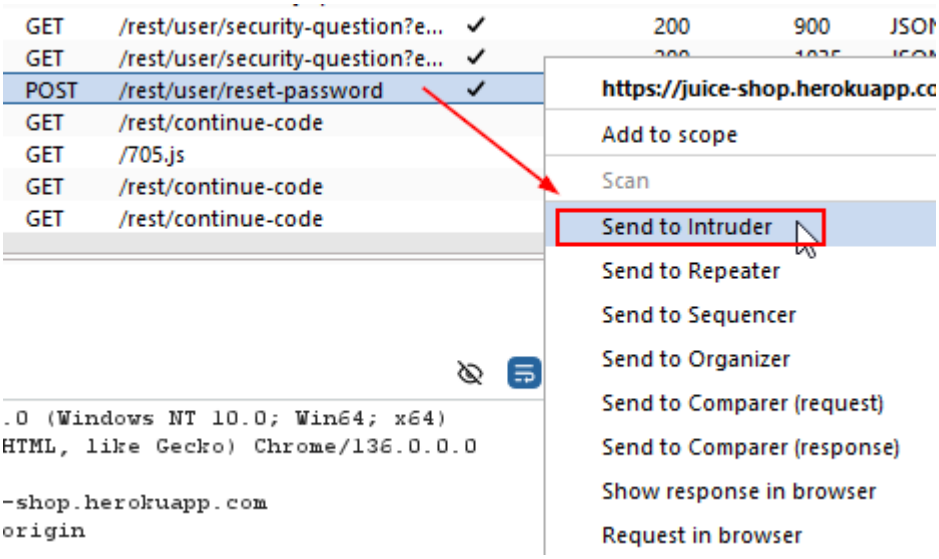
4.Enter any answer and new password



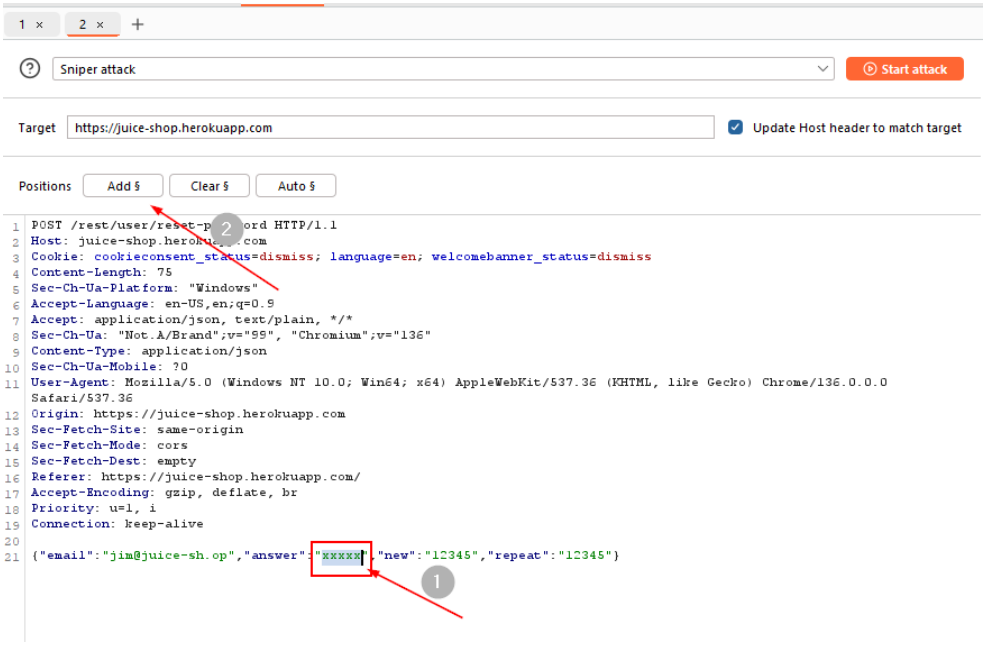
5.Open Burpsuite to catch the request



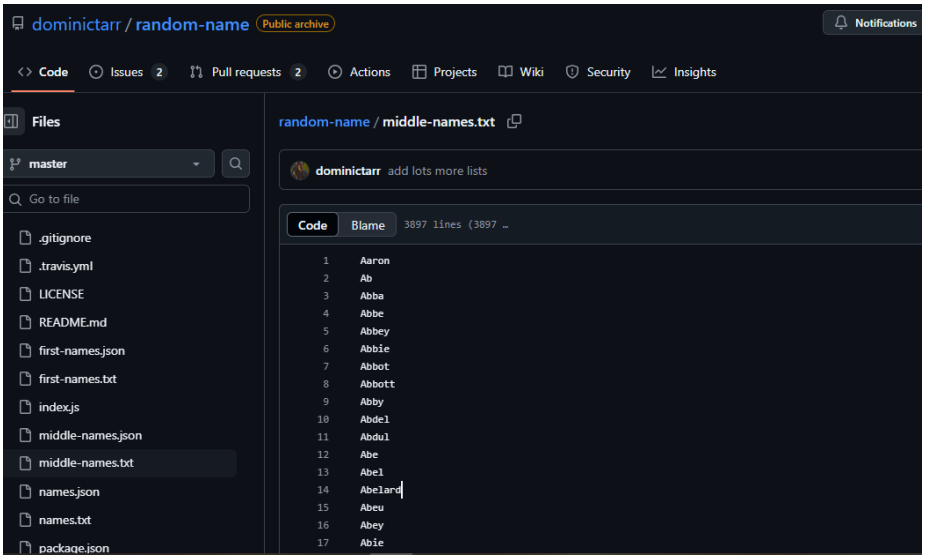
6.Send to intruder



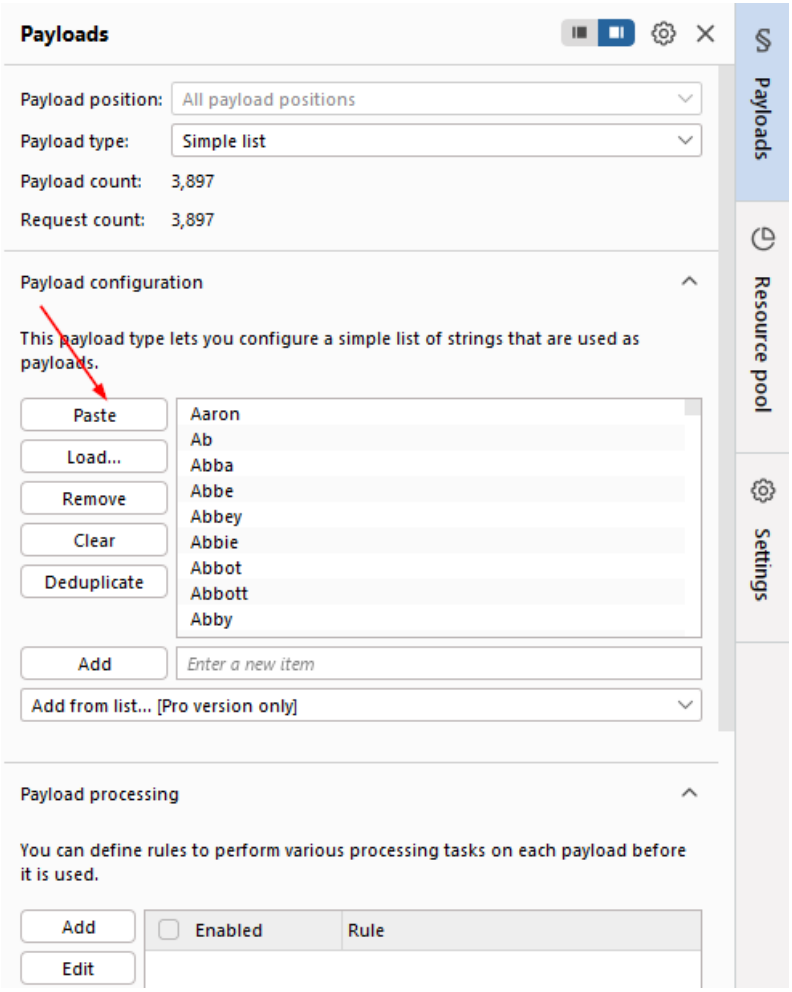
7.Select the answer and add



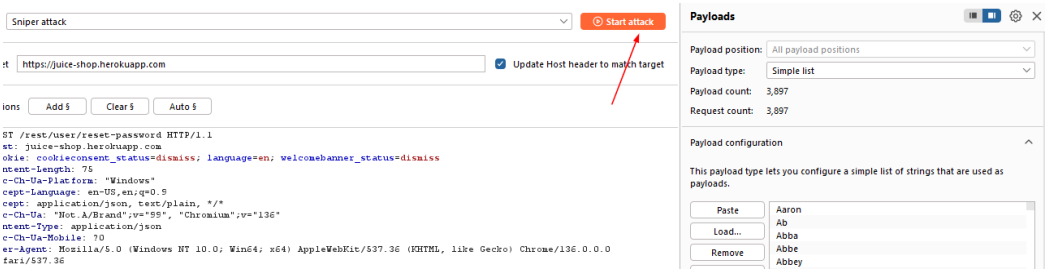
8.Use the middle-names.txt – GitHub



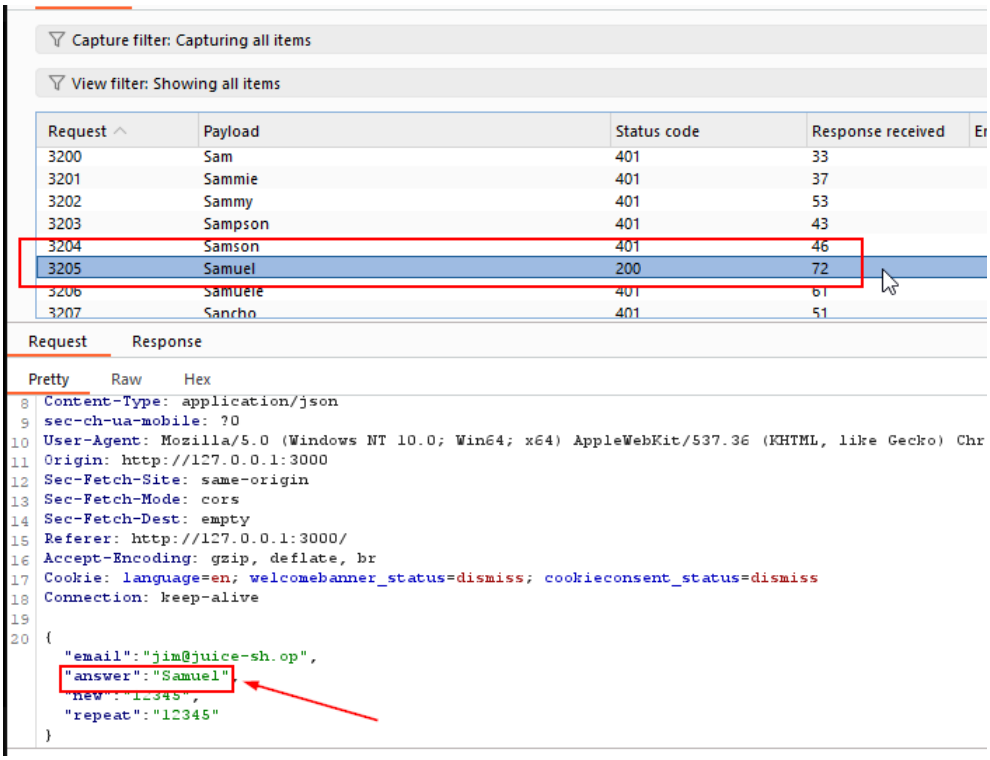
9.Copy the list and paste



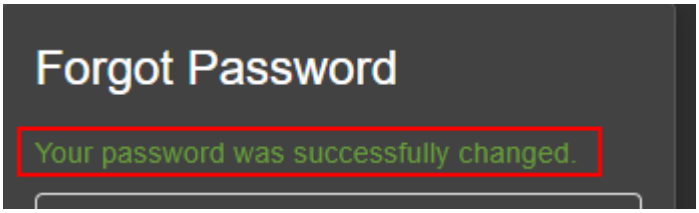
10. Start bruteforce attack

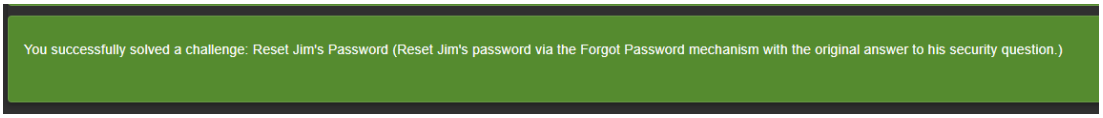


11. Monitor responses,Upon receiving a response with status 200 OK, identified the correct answer.



12. Went back to the forgot password form and entered: Samuel





6.1.8 Reset Bender's Password

HIGH

Description:

The application allows a **Pentester** to change the password of an authenticated user account without providing the correct current password. After bypassing login protection via SQL Injection ('--), the Pentester accessed **Bender's** account.
While attempting to change the password via the change password form, the application requested the current password.

However, by intercepting the request using **Burp Suite**, and **removing** the current parameter from the request altogether, the server skipped validation and accepted the new password directly.

This leads to **complete account control** and denies the original user access.

Impact:

- Full Account Takeover (ATO)
- Password Changed Without Owner Consent
- High Risk in Real-World Authentication Flows.

Resource / References:

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

Vulnerability Location:

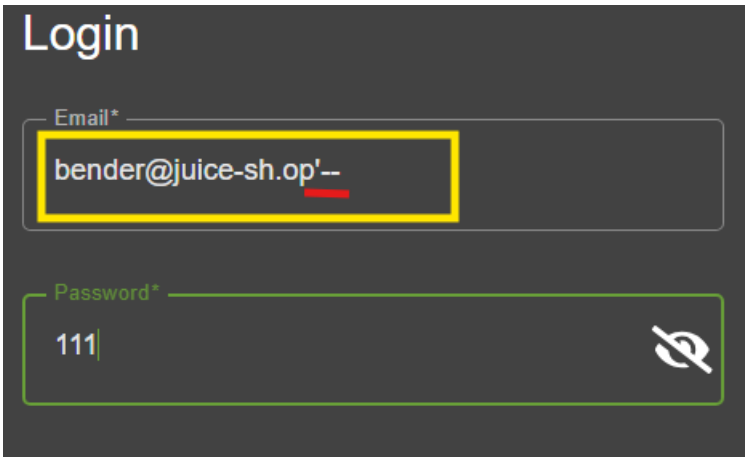
- **Component:** Change Password Functionality
- **Input Field:** current
- **Affected Route:** http://localhost:3000/login#/privacy-security/change-password

Recommendations:

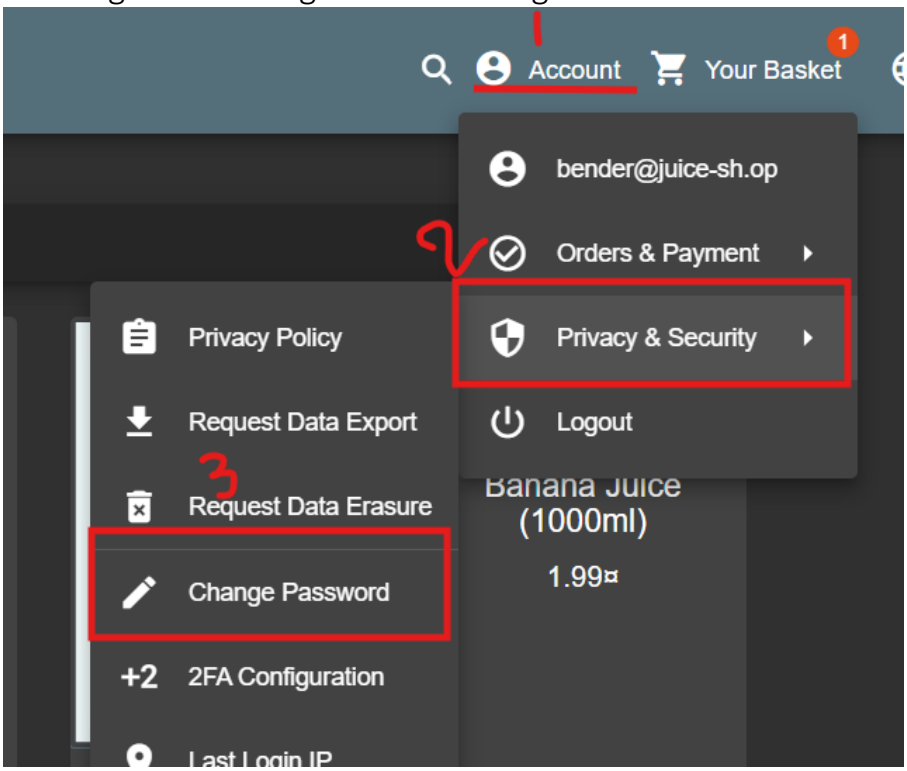
- Enforce current password validation on server-side regardless of request content.
- Reject requests missing required parameters like current.

Proof of Concept (PoC):

1- Login via SQL Injection



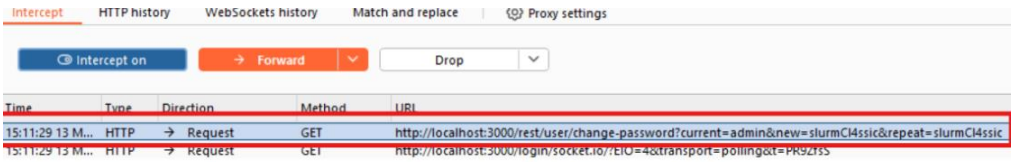
2- Navigate to Change Password Page



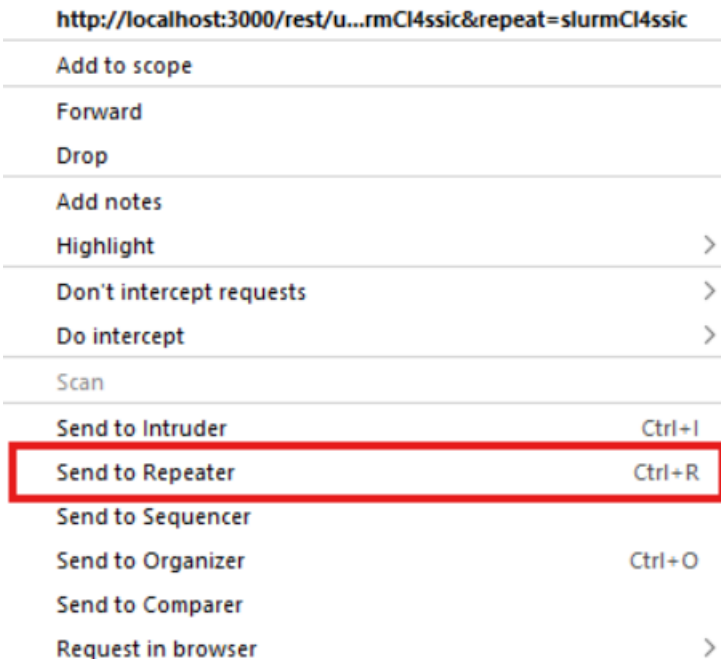
3- Submit

- **Current password:** admin (or any placeholder)
- **New password:** slurmCl4ssic

4- Intercept and Modify Request in Burp Suite

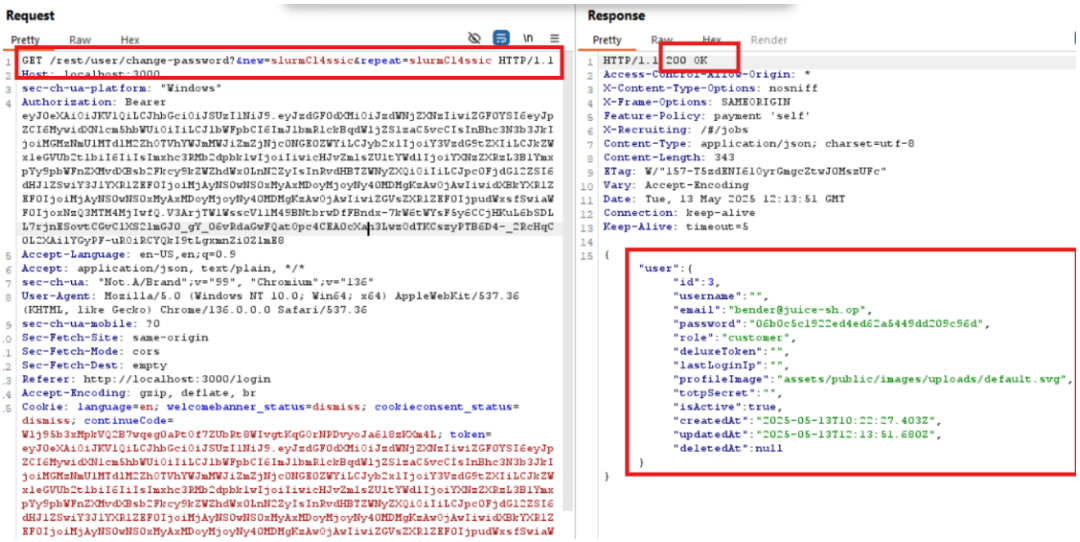


5- Right-click and send the request to the repeater



6- Open repeater tap and delete this “current=admin” from request

7- Send the modified request and Observe Server Response


Request
Pretty Raw Hex
GET /rest/user/change-password?&new=slurmCl4ssic&repeat=slurmCl4ssic HTTP/1.1
Host: localhost:3000
sec-ch-ua-platform: "Windows"
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJp
ZCI6MywidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImJlbmRlckBqdWljZSIsaC5vcCIsInBhc3N3b3JkI
joiMGMzNmU1MTdlM2Zh0TVhYWJmMWJiZmZjNjc0NGE0ZWYiLCJyb2xlIjoiY3VzdG9tZXIiLCJkZW
xleGVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoiIiwicHJvZmlsZUltYWdlIjoiYXNzZXRzL3B1Ymx
pYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6
dHJ1ZSwiY3J1YXRlZEF0IjoiMjAyNS0wNS0xMyAxMDoyMjoyNy40MDMgKzAw0jAwIiwidXBkYXRlZ
EF0IjoiMjAyNS0wNS0xMyAxMDoyMjoyNy40MDMgKzAw0jAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaW
F0IjoxNzQ3MTM4MjIwfQ.V3ArjTWlWsscVllM4SBNtbrwDfFBndx-7kW6tWYsF5y6CCjHKuL6bSDL
L7rjnESovtCGvC1XS2lmGJ0_gY_O6vRdaGwFQat0pc4CEA0cXah3Lwz0dTKCszyPTB6D4-_2RcHqC
OLZXAilYGyPF-uR0iRCYQkI9tLgxmnZi0ZlmE8
Accept-Language: en-US,en;q=0.9
Accept: application/json, text/plain, */*
sec-ch-ua: "Not.A/Brand";v="99", "Chromium";v="136"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
sec-ch-ua-mobile: ?0
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/login
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=
dismiss; continueCode=
Wlj95b3xMpkVQ2B7wqegOaPt0f7ZUbRt8WIvgtKqG0rNPDvyoJa6l8zKXm4L; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJp
ZCI6MywidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImJlbmRlckBqdWljZSIsaC5vcCIsInBhc3N3b3JkI
joiMGMzNmU1MTdlM2Zh0TVhYWJmMWJiZmZjNjc0NGE0ZWYiLCJyb2xlIjoiY3VzdG9tZXIiLCJkZW
xleGVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoiIiwicHJvZmlsZUltYWdlIjoiYXNzZXRzL3B1Ymx
pYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6
dHJ1ZSwiY3J1YXRlZEF0IjoiMjAyNS0wNS0xMyAxMDoyMjoyNy40MDMgKzAw0jAwIiwidXBkYXRlZ
EF0IjoiMjAyNS0wNS0xMyAxMDoyMjoyNy40MDMgKzAw0jAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaW
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
Content-Length: 343
ETag: W/"157-T5zdENI6l0yrGmgcZtwJ0MsnUFc"
Vary: Accept-Encoding
Date: Tue, 13 May 2025 12:13:51 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{
"user":{
"id":3,
"username":"",
"email":"bender@juice-sh.op",
"password":"06b0c5c1922ed4ed62a5449dd209c96d",
"role":"customer",
"deluxeToken":"",
"lastLoginIp":"",
"profileImage":"assets/public/images/uploads/default.svg",
"totpSecret":"",
"isActive":true,
"createdAt":"2025-05-13T10:22:27.403Z",
"updatedAt":"2025-05-13T12:13:51.680Z",
"deletedAt":null
}
}

6.1.9 Login Amy

HIGH

Description:

The application exposes the login credentials of the user amy@juice-sh.op (guessing the email pattern and brute-forcing a predictable password)through an insecure password policy based on a **predictable padding pattern**.
By analyzing the challenge hint and external documentation, the **Pentester** was able to reduce the password brute-force space dramatically by focusing on structured patterns instead of random guesses.

Impact:

- **Sensitive Data Exposure:** Unauthorized access to a registered user's private account.
- **Privacy Violation:** Compromise of user profile, orders, or personal data.
- **Authentication Weakness:** Demonstrates poor credential entropy and lack of layered defense mechanisms.

Resource / References:

- CWE-521: Weak Password Requirements
- Juice Shop Challenge Hint

Vulnerability Location:

- **Component:** Login Page
- **Input Field:** Password
- **Affected Account:** amy@juice-sh.op

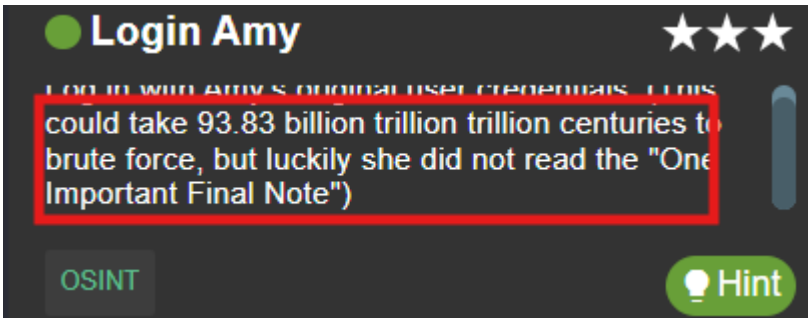
Recommendations:

- **Avoid Predictable Padding:** Encourage secure password policies that avoid simple repetition or character patterns.

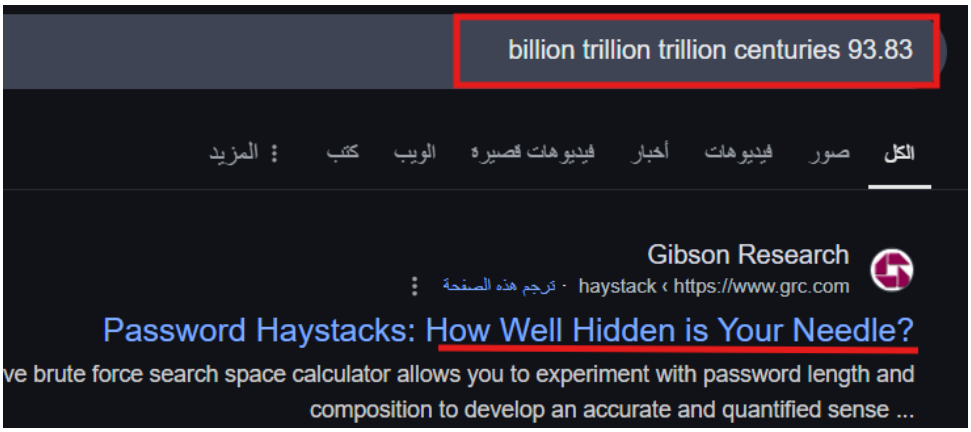
- **Implement Rate Limiting:** Block or delay brute-force attempts through IP-based thresholds.

Proof of Concept (PoC):

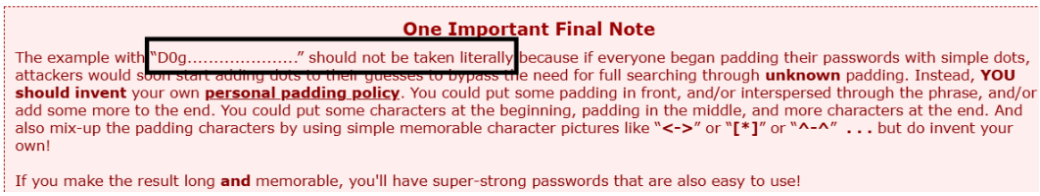
1 - Analyze Challenge Hint



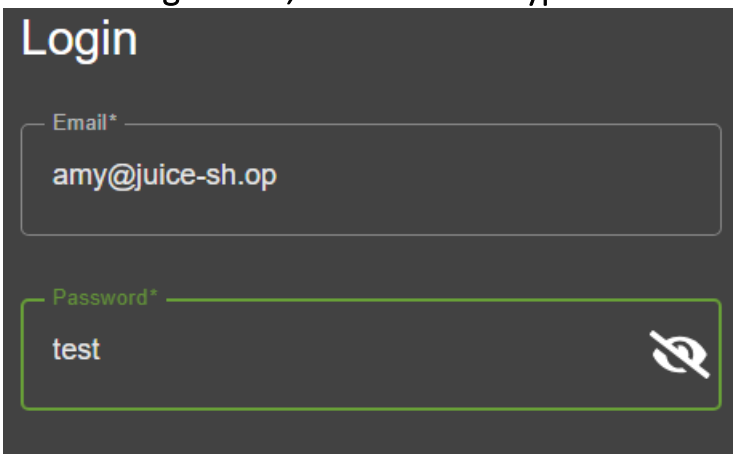
2- The Pentester searched this phrase on Google



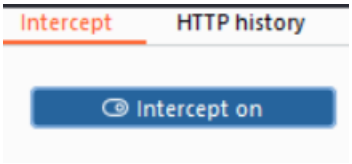
3- Found reference to “D0g.....”



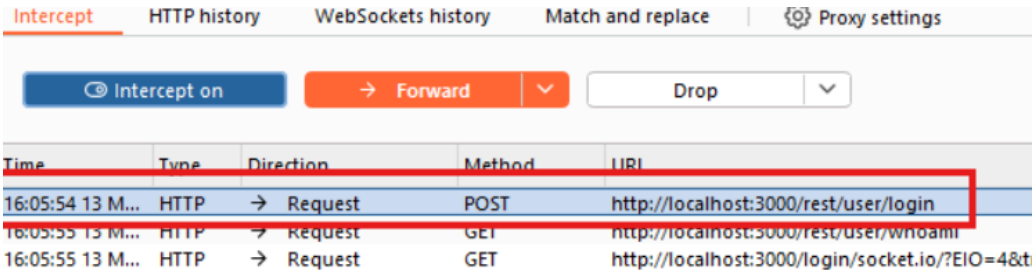
4- In the login form, the Pentester typed:



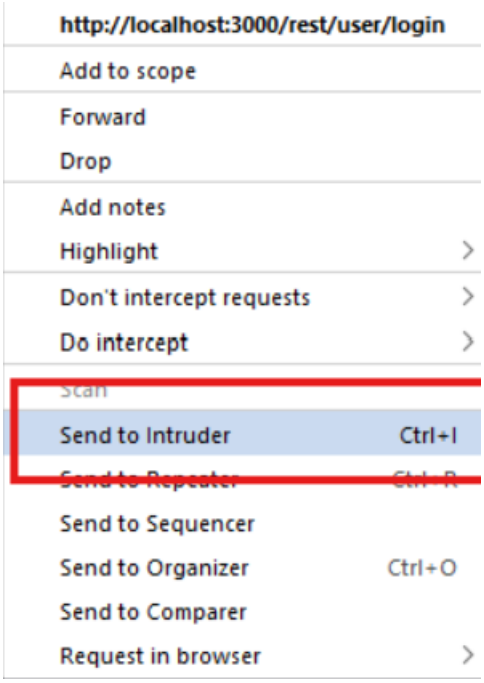
5- Before clicking Login, the Pentester enabled Intercept in Burp Suite under



6-Captured request looked like this

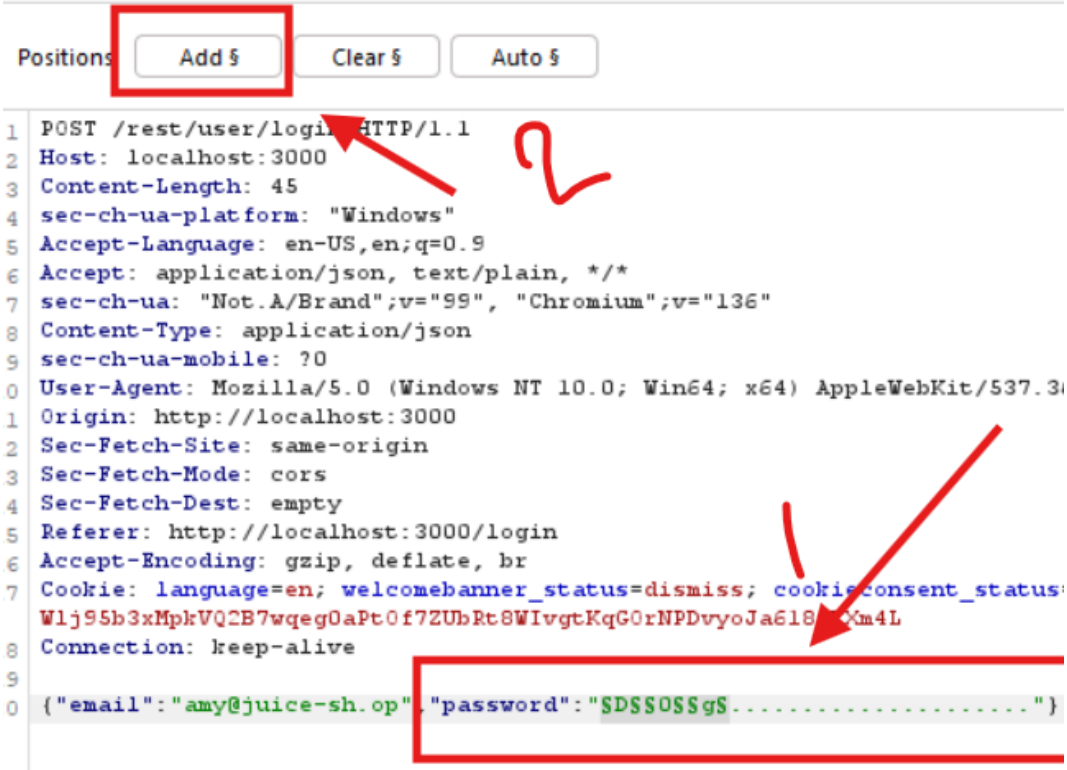


7-Send the Request to Intruder(Right-click → Send to Intruder

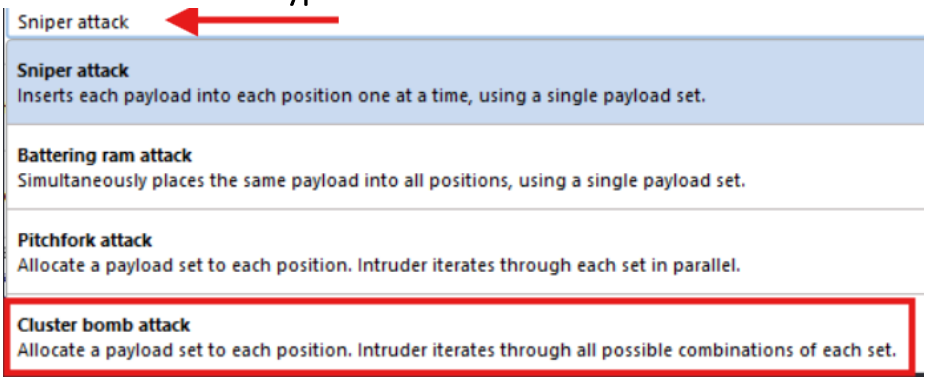


8- Go to intruder tap and modify request body

9 - The Pentester selected each char (D0g) then clicked add:



10- Selecte attack type



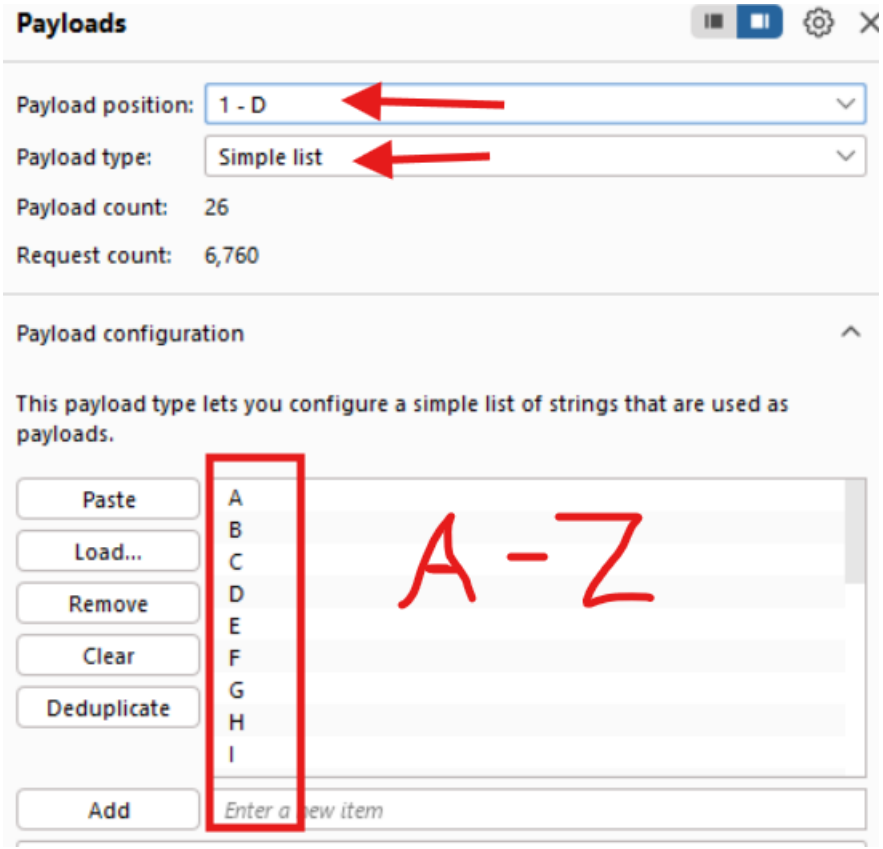
10- Set Up Payload Positions

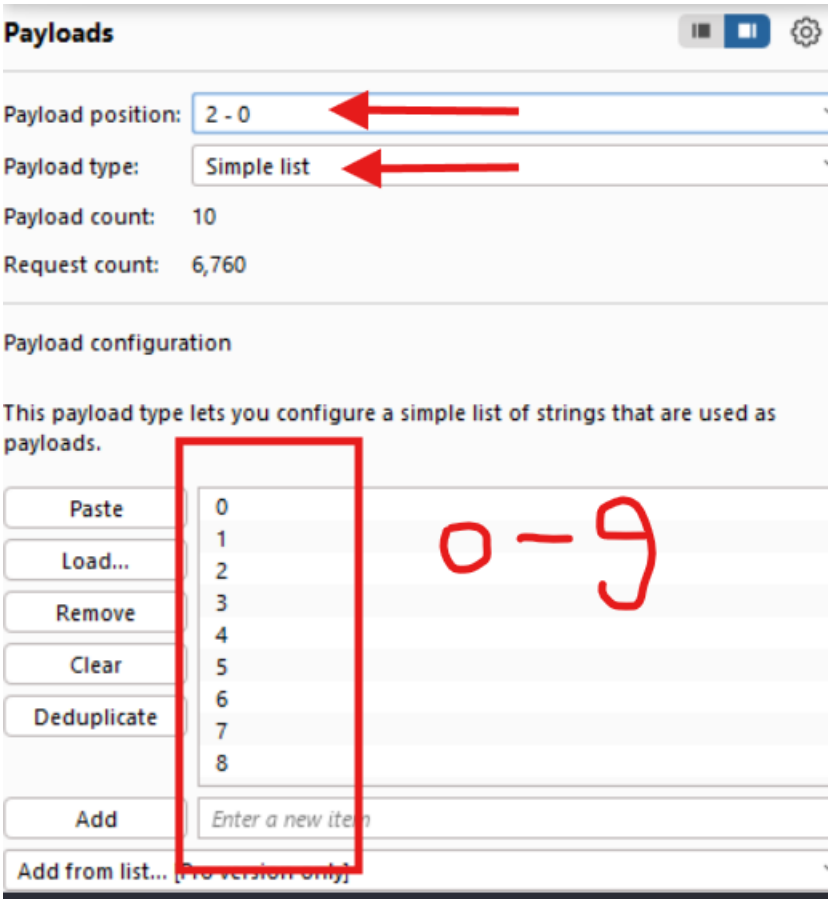
The Pentester selected:

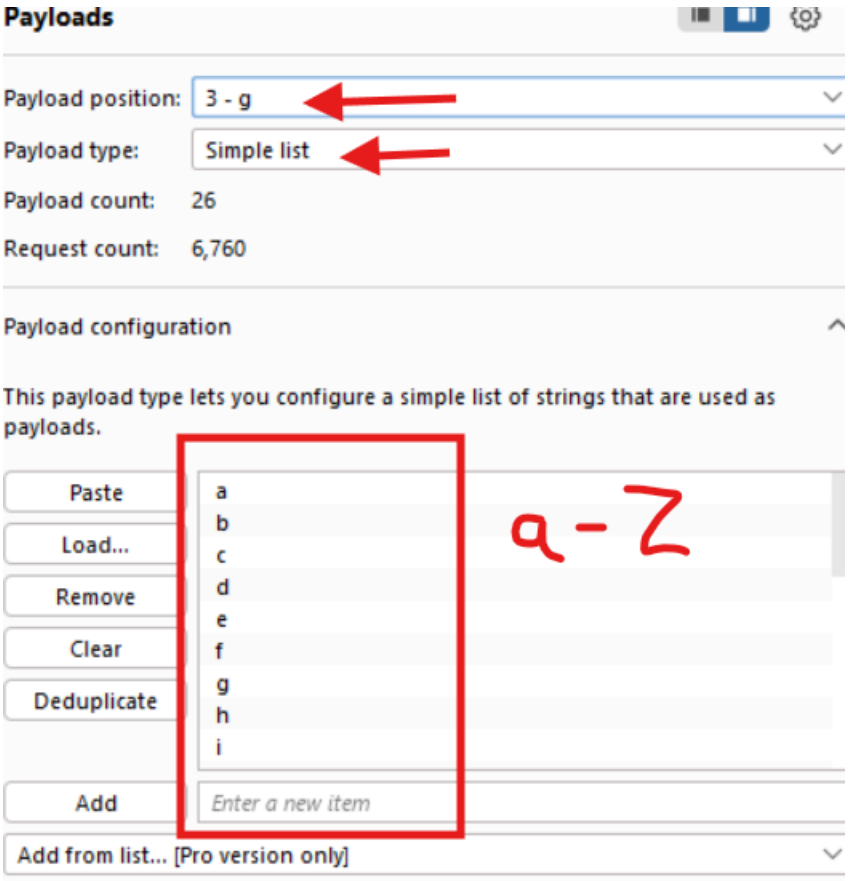
-The uppercase A-Z → Set as Payload Position 1

-The digits 0-9 → Set as Payload Position 2

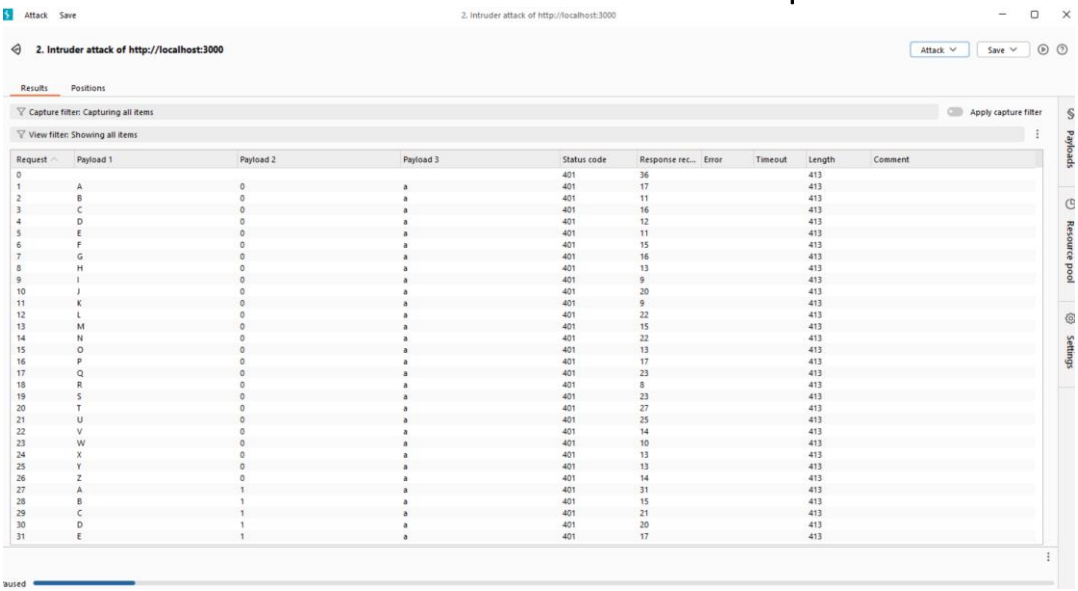
-The lowercase a -z → Set as Payload Position 3



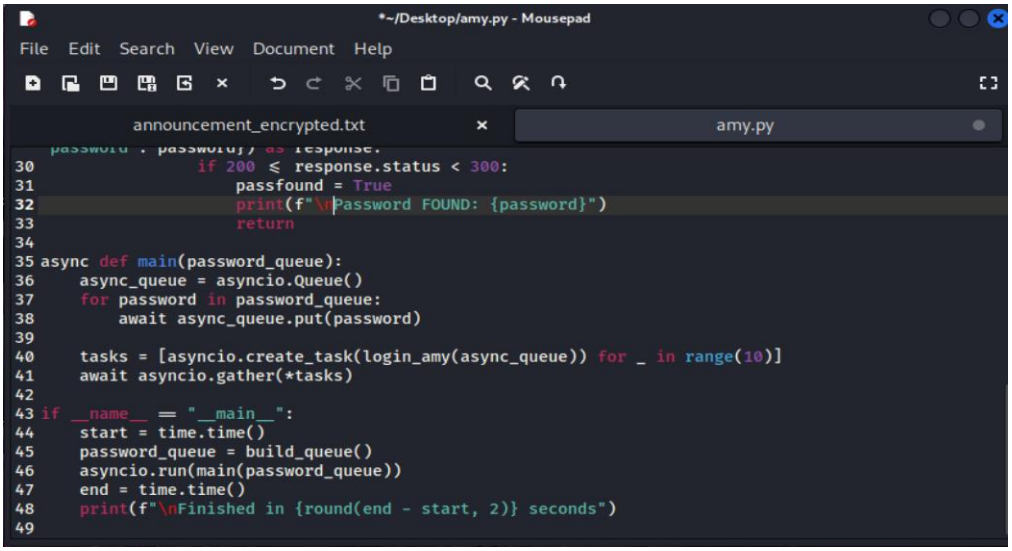

Payloads
Payload position: 2 - 0
Payload type: Simple list
Payload count: 10
Request count: 6,760
Payload configuration
This payload type lets you configure a simple list of strings that are used as payloads.
Paste
Load...
Remove
Clear
Deduplicate
0
1
2
3
4
5
6
7
8
Add
Enter a new item
Add from list... [Pro version only]
0-9


Payloads
Payload position: 3 - g
Payload type: Simple list
Payload count: 26
Request count: 6,760
Payload configuration
This payload type lets you configure a simple list of strings that are used as payloads.
Paste
Load...
Remove
Clear
Deduplicate
a
b
c
d
e
f
g
h
i
Add
Enter a new item
Add from list... [Pro version only]
a-z

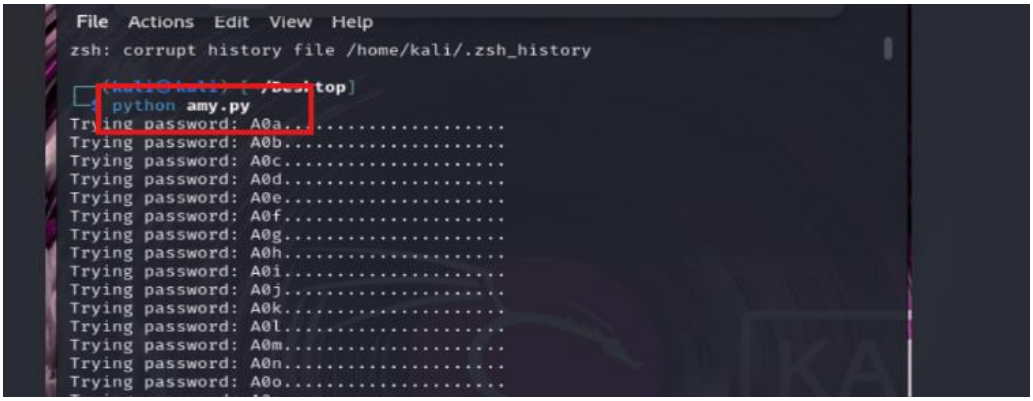
11- Click Start Attack then Burp began sending hundreds of combinations and wait for 200 status code this is the password



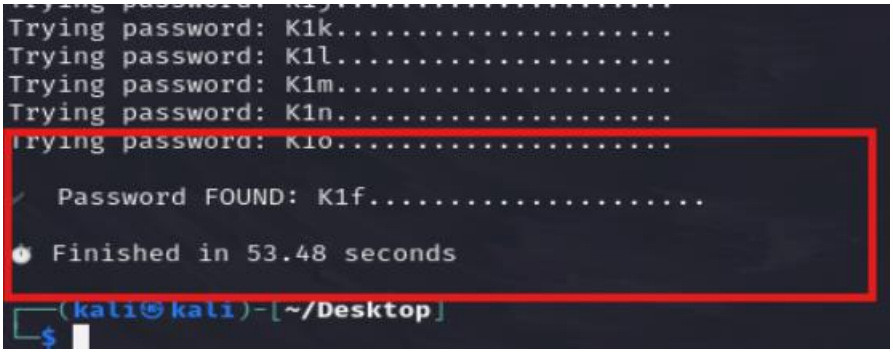
-- to automate the brute-force process faster Saved the following script as “amy.py” on your kali machine

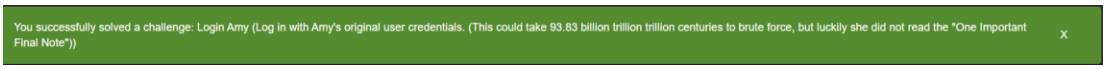


-then open terminal and write :



- After a short time, it printed





6.1.10 Login MC SafeSearch

HIGH

Description:

This challenge was about logging into the user account of **MC SafeSearch**, a fictional rapper mentioned within the Juice Shop application. The vulnerability lies in how publicly available information, combined with subtle hints from the application, can lead to full account takeover.

Impact:

- **User Account Compromise:** penetration testers could impersonate the user, access sensitive data, or perform malicious actions.
- **Reputation Damage:** Revealing that users (especially public figures like "MC SafeSearch") reuse weak or publicly known passwords reflects poorly on platform security culture.

Vulnerability Location:

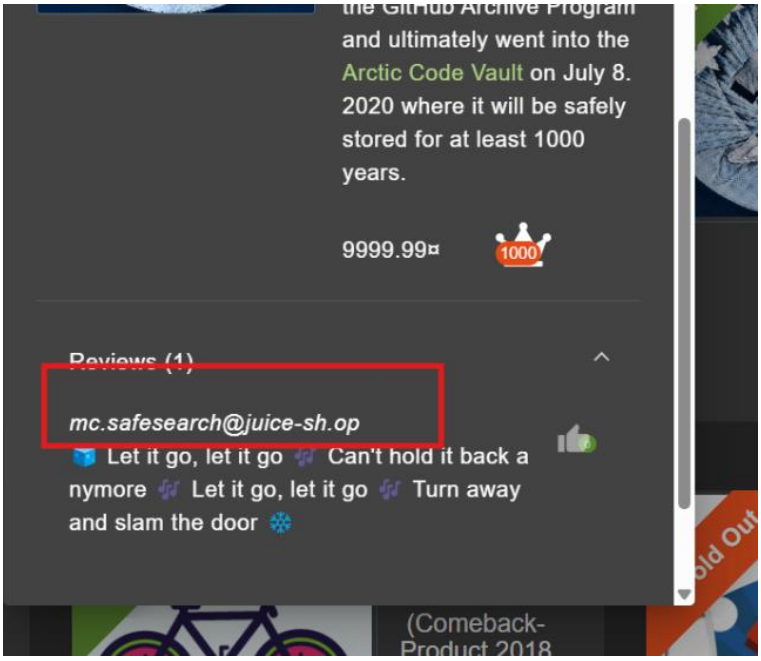
- **Component:** Login Page (/#/login)
- **Affected Route:** http://localhost:3000/login#/login

Recommendations:

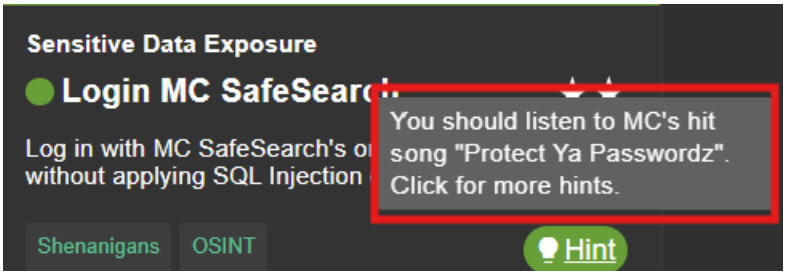
- Educate users on **secure password hygiene**.
- Do **not reference real passwords** in public-facing content.
- Use **2FA** (two-factor authentication) to prevent account takeover even if the password is leaked

Proof of Concept (PoC):

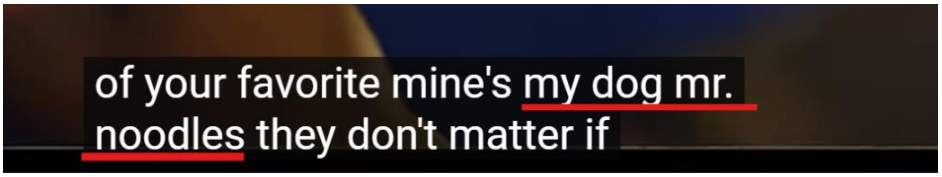
1- Identify the Username from products page => Juice Shop "Permafrost" 2020 Edition



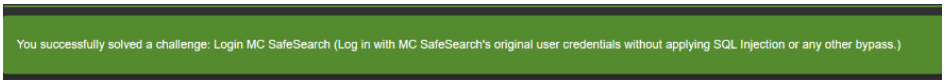
2- A hidden hint encouraged the Pentester to perform **Open-Source Intelligence (OSINT)**.



3- listening to the song, the Pentester identified a line where **MC SafeSearch says his password**



4- Successful Login



6.1.11 Forged Feedback

MEDIUM

Description:

This vulnerability allows an penetration tester to submit feedback using another user's identity without authorization. By tampering with the request sent to the feedback API, the penetration tester can inject a different userId and post feedback on behalf of another user, bypassing access control checks.

Impact:

A penetration tester was able to submit feedback using another user's account by modifying the userId parameter in the request body. This proves that there is no proper validation of user identity on the server side. The vulnerability can be abused to impersonate users, spam feedback under multiple user identities, and manipulate trust in user-generated content.

Vulnerability Location:

Feedback section: /#/contact

POST /api/Feedback/

Recommendations:

Enforce strict access control checks on the server side to ensure the userId in requests matches the currently authenticated user.

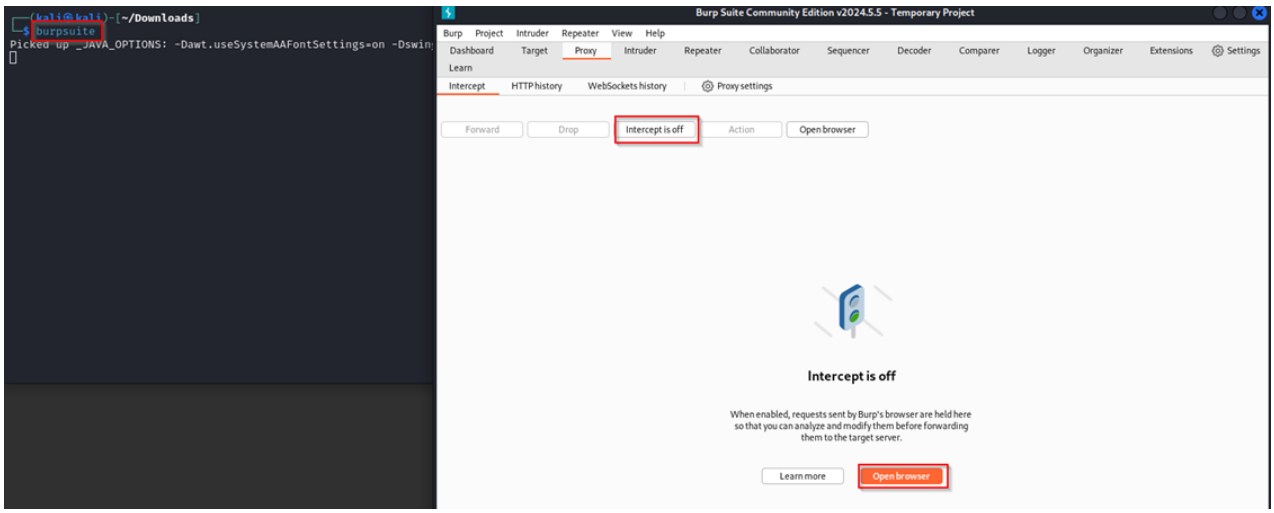
Remove the ability for users to define or modify sensitive parameters such as userId in client-side requests.

Implement authentication tokens to securely link feedback submissions to the authenticated user session.

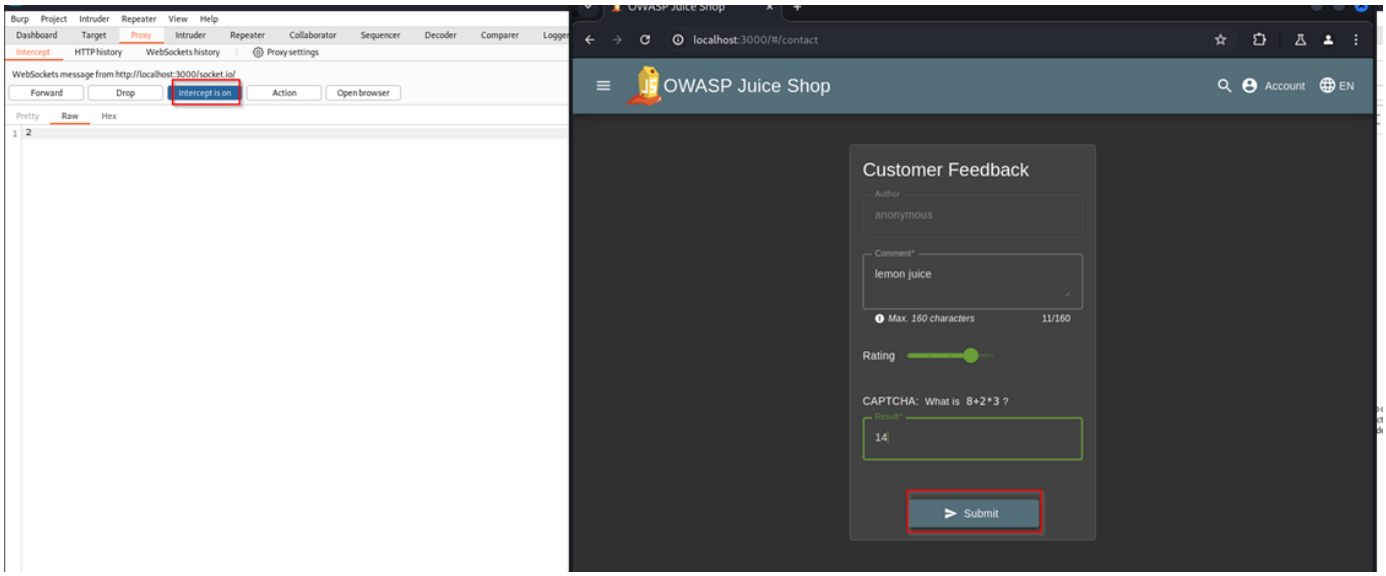
Log and monitor unusual activity such as high-frequency feedback submissions or identity mismatches

Proof Of Concept:

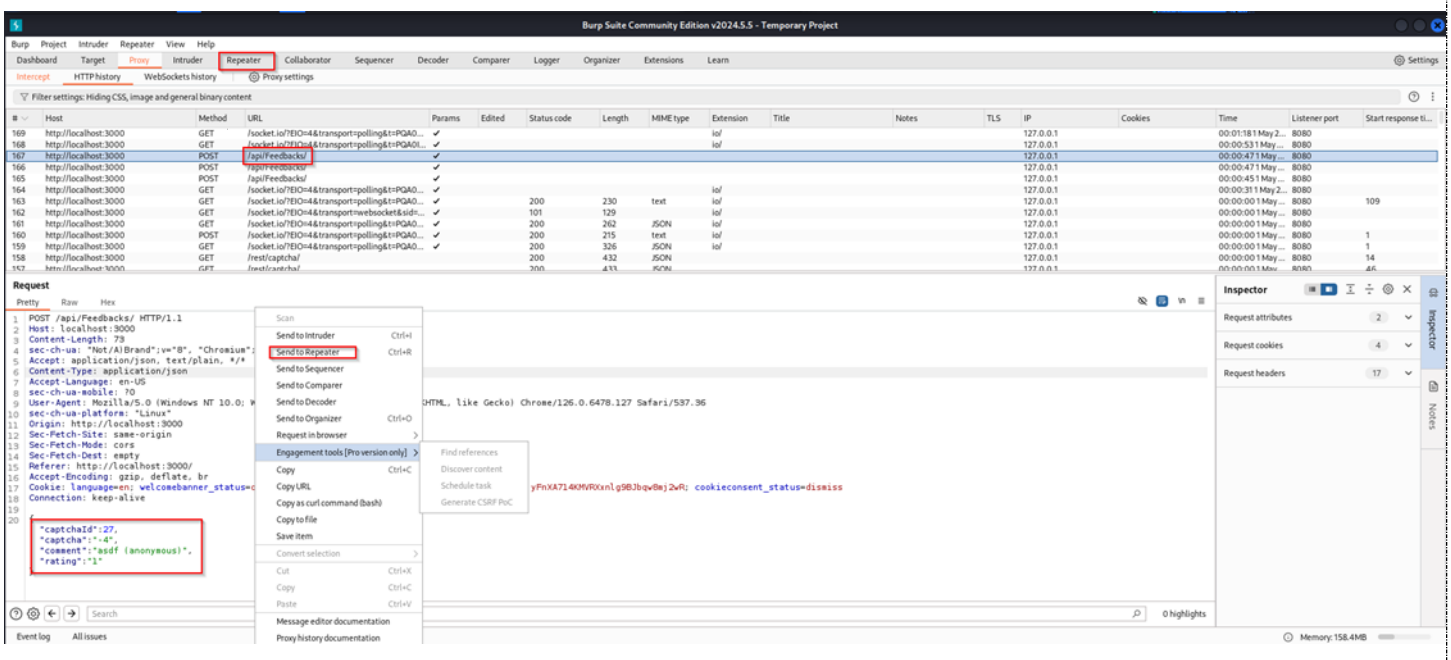
Open Burp Suite and turn on Intercept to capture the request.



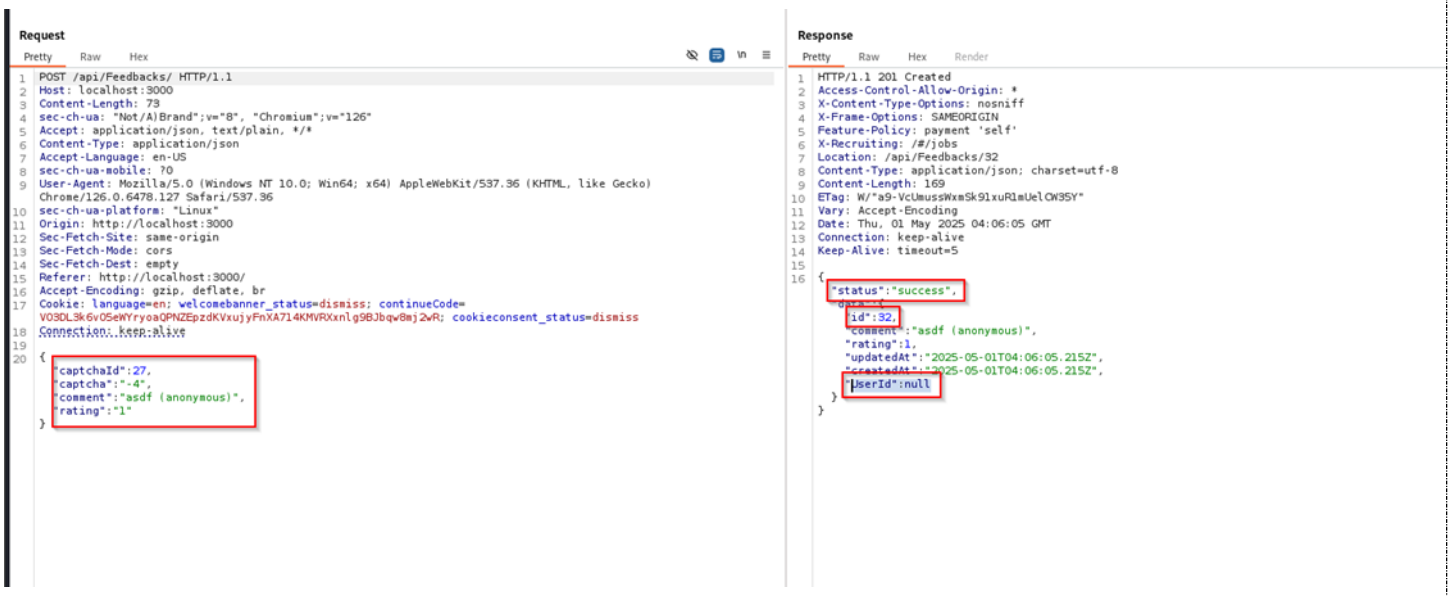
2. Go to Feedback page while logged out the site allows posting anonymously.



3. Submit a feedback, Capture the POST request to: “/api/Feedback/” and send it to the Repeater for analysis.

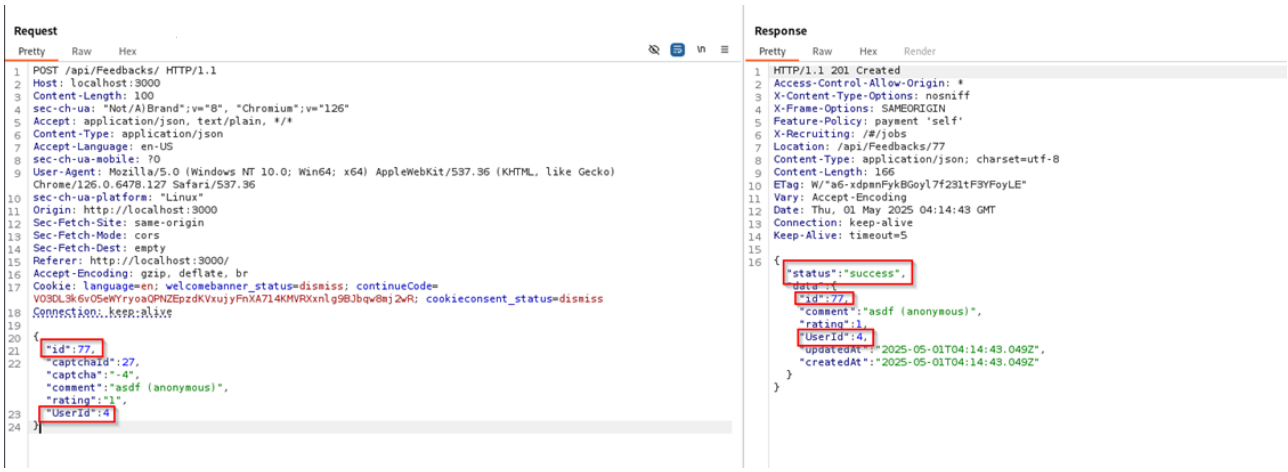


4. Observe the JSON body of the request and response. Identify useful parameters found in the response:"userId": null - "id": 32

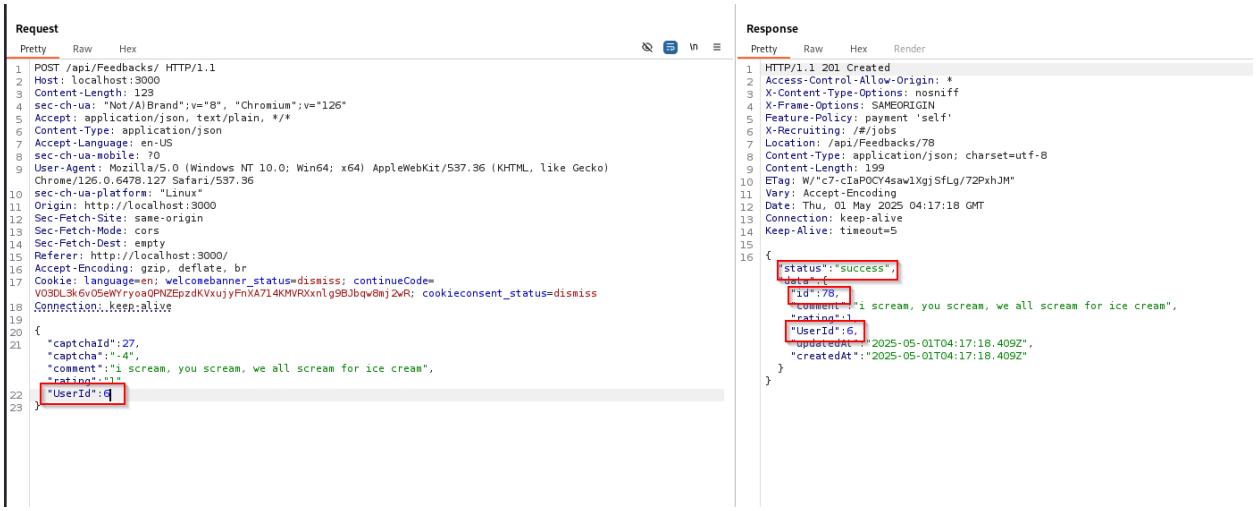


5. Modify the request JSON body by adding "userId": 4 and "id": 77 to impersonate another user and click Send.

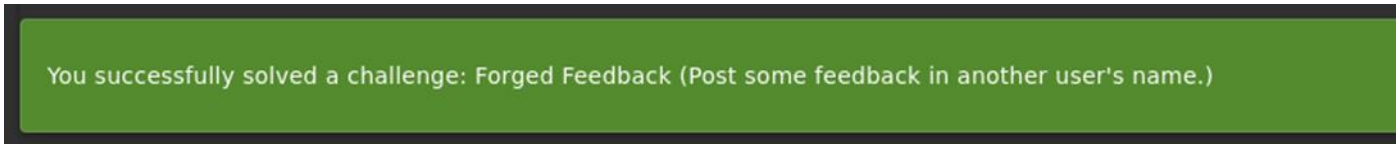
(Note: "id" is optional and can be removed.)



6. The response confirmed that the feedback was submitted successfully under user ID 4.



Also observed that the id field could be removed without affecting the outcome



6.2 SQL Injection

6.2.1 login Jim

HIGH

Description:

The login form of the Juice Shop application is vulnerable to **SQL Injection**, which allows penetration testers to bypass authentication mechanisms. By manipulating the input fields, an penetration tester can trick the SQL query into logging in without knowing the user's password.

In this case, the penetration tester did not initially know the user's email but identified it through the **Customer Feedback** section, where **Jim's email (or any user email)** appeared next to one of his reviews.

Once the email was obtained, the pentester crafted a simple SQL injection payload in the password field to bypass authentication and successfully log in as Jim.

Impact:

The penetration tester was able to:

- Discover Jim's email from publicly visible content.
- Use SQL Injection in the login form to log in as Jim without needing his password.

This vulnerability could lead to:

- Unauthorized access to customer accounts.
- Exposure of personal user data, order history, or saved payment methods.
- Full account takeover of any user whose email can be found or guessed.

Resource / References:

- OWASP Top 10: A03:2021 – Injection
- OWASP SQL Injection Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Vulnerability Location:

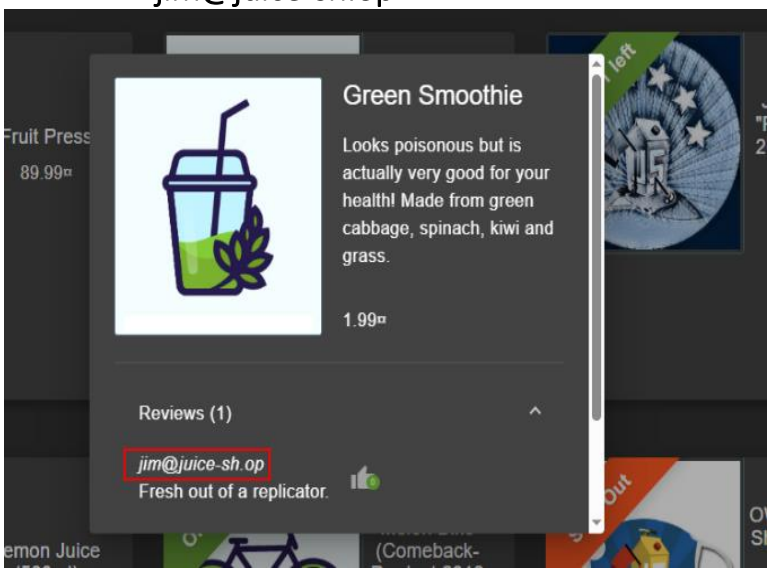
- **Component:** Login Page
- **Input Field:** Email and Password
- **Tested IP:** 127.0.0.1:3000/#/

Recommendations:

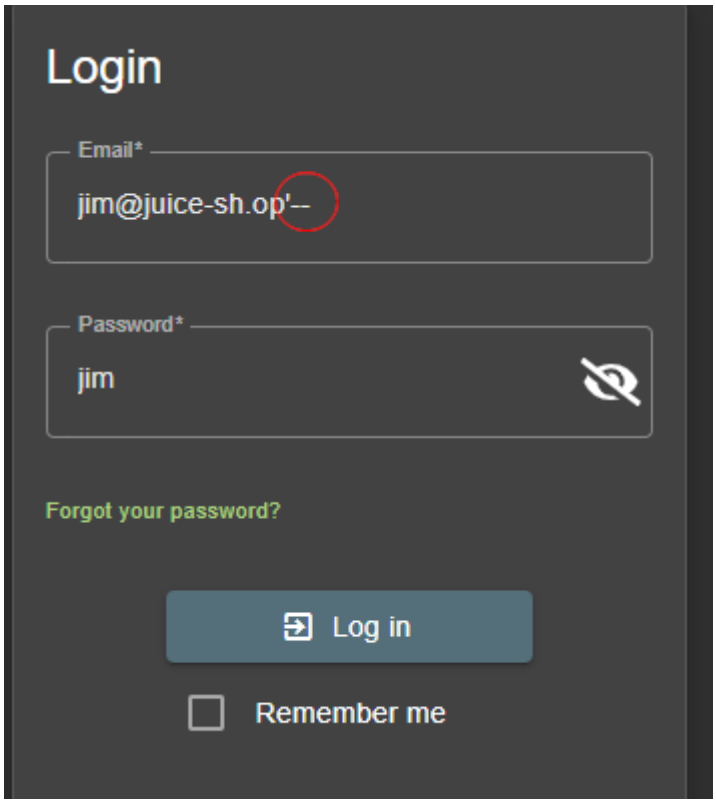
- Use **parameterized queries** (prepared statements) for all database interactions.
- Sanitize and validate all user inputs on both client and server sides.
- Implement security measures like **rate limiting** and **account lockout** on login attempts.
- Avoid exposing user identifiers (like email addresses) in public areas unless necessary.

Proof of Concept (PoC):

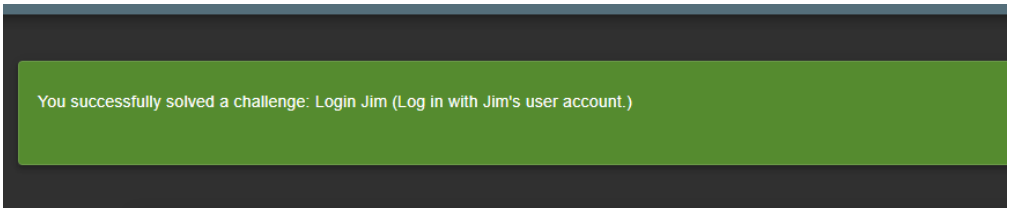
1. Browse the product reviews and found one made by Jim, which exposed his email: **jim@juice-sh.op**



2. In login page. In the email field, Enter: jim@juice-sh.op'--
   And added any random value in password field, because the SQL injection comments out the rest of the query.



3. When submit the form, successfully log in as Jim without knowing his password.



Payload used:

- **Email:** jim@juice-sh.op'--
- **Password:** anything (ignored)

6.2.2 Database Schema

HIGH

Decription:
The application is vulnerable to SQL Injection in the search functionality. By manipulating the search input parameter, the pentester can inject arbitrary SQL queries into the backend database. In this case, it was possible to extract the entire database schema by exploiting the vulnerability through the built-in **sqlite_master** table in SQLite.

The server improperly handles input validation, allowing the penetration tester to craft a malicious query to leak sensitive information about the database structure.

Impact:

- **Data Disclosure:** Full exposure of the database structure, including table names and column definitions.
- **Enumeration:** penetration tester can enumerate the internal database schema.
- **Future Attacks:** Enables penetration testers to plan further targeted attacks such as extracting sensitive user data.

Vulnerability Location:

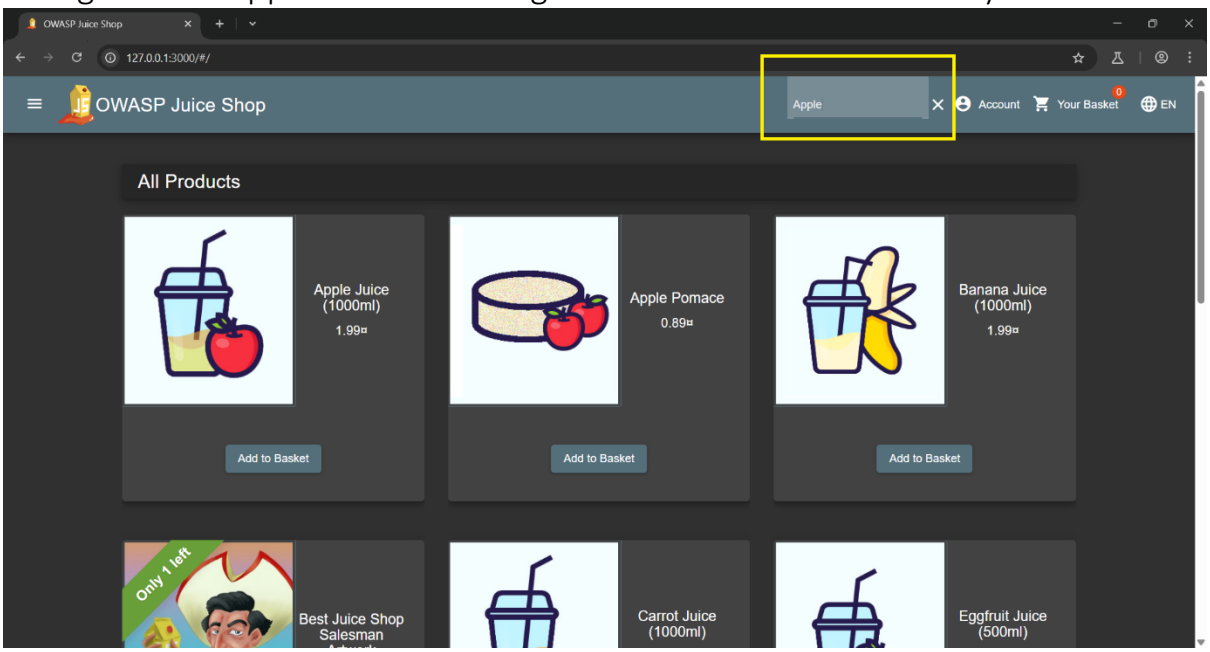
- **Component:** Product Search / Search Functionality
- **Parameter Affected:** Search Query Input Field

Recommendations:

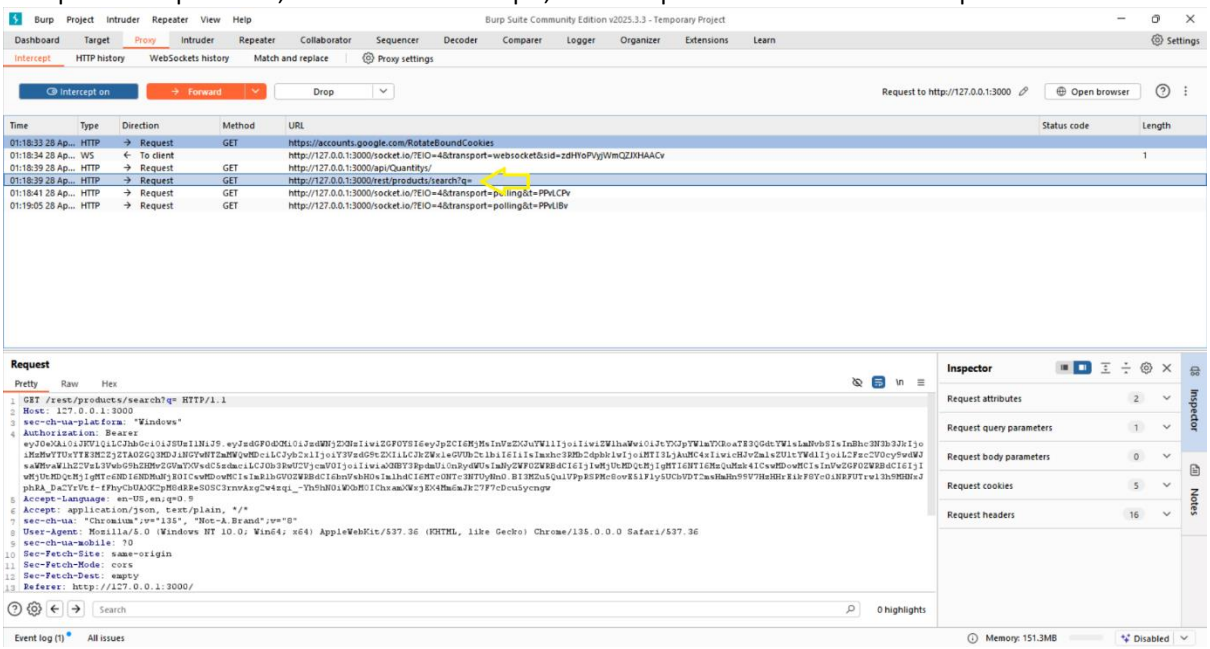
1. **Input Validation and Sanitization:**
   Strictly validate and sanitize all user inputs to prevent malicious characters and patterns.
2. **Error Handling:**
   Do not display detailed database errors to users; use generic error messages instead.

Proof of Concept (PoC):

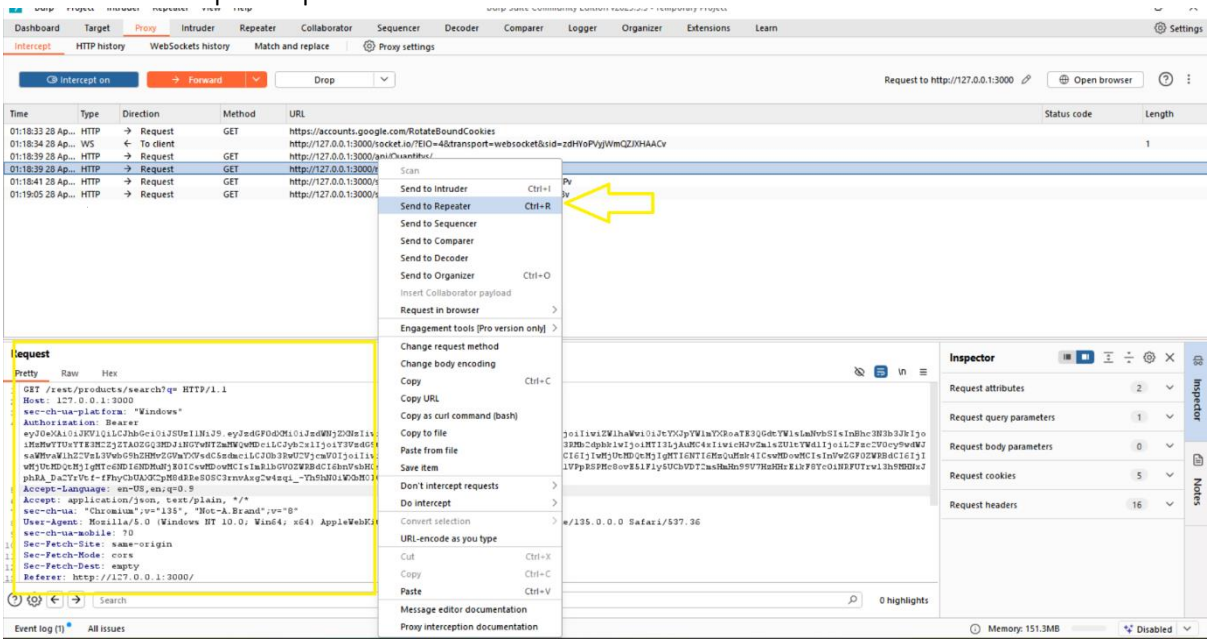
1-Login to the application and navigate to the search functionality.



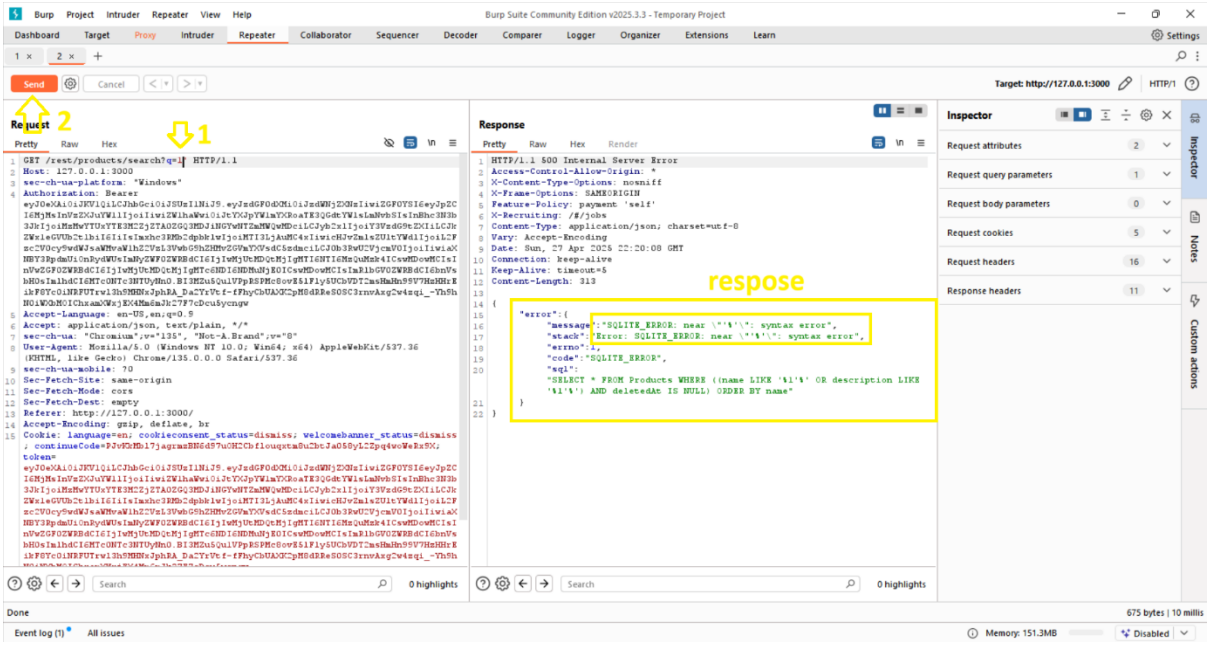
2:Open Burp Suite, enable Intercept, and capture the search request.



3:send the req to repeater



4:Modify the search input to inject a simple SQL payload:1'

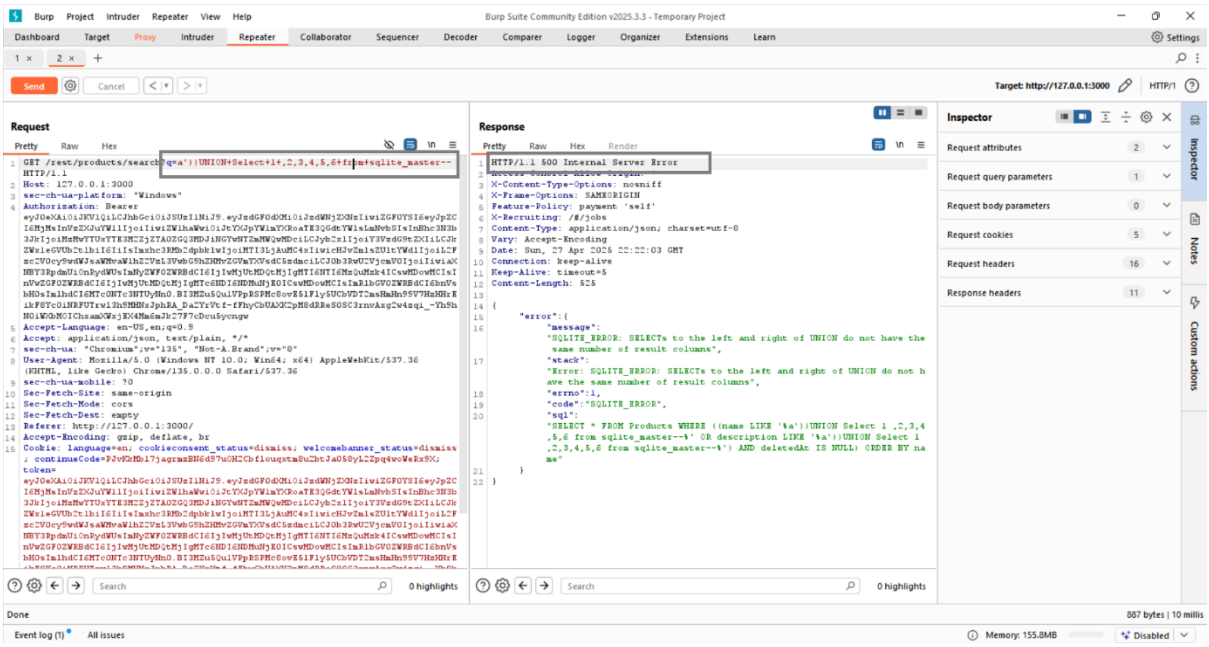


Observation:

An SQL error was triggered, revealing that the backend is using **SQLite**.

5:Based on the error information, proceed to find the number of columns by trial and error using ORDER BY technique:ORDER BY 1-- ORDER BY 2--ORDER

BY 3--...ORDER BY 9--

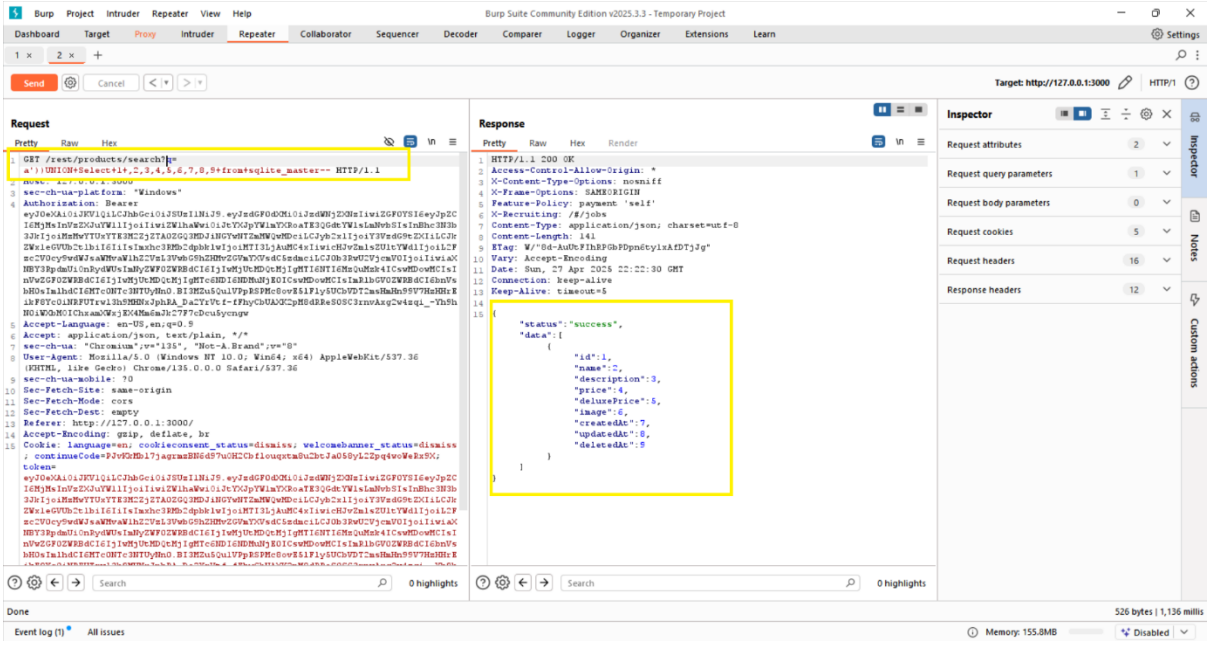


Result:

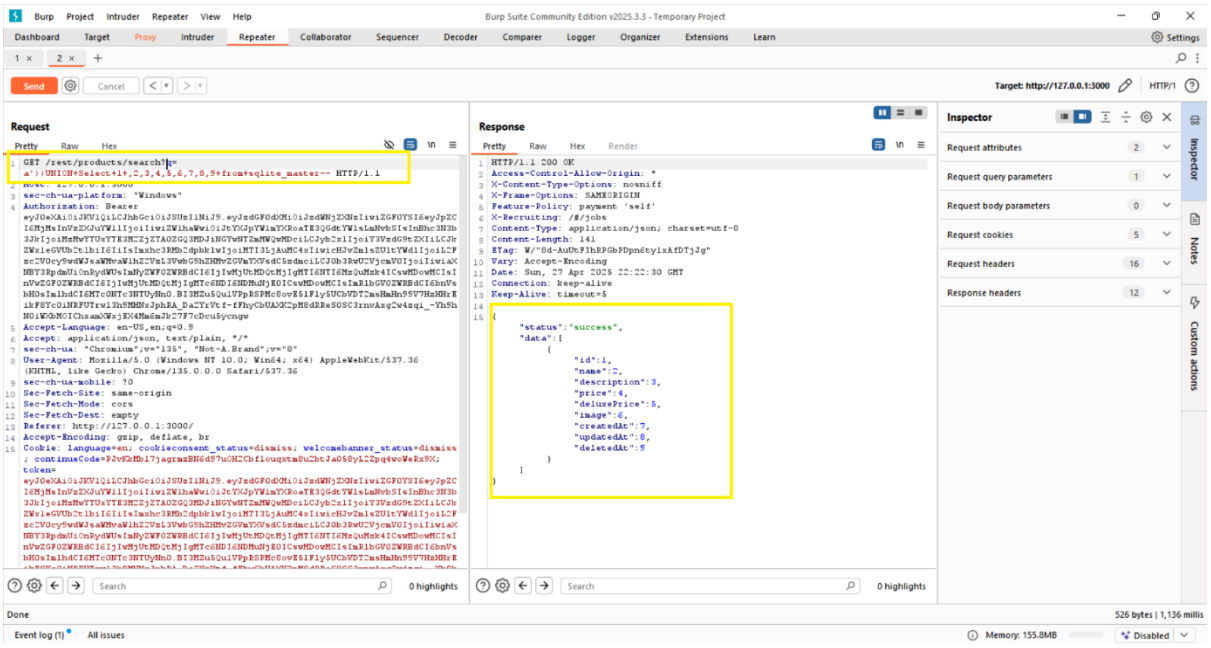
Discovered that the correct number of columns is **9**, because after ORDER BY 9-- the page loaded normally without an error.

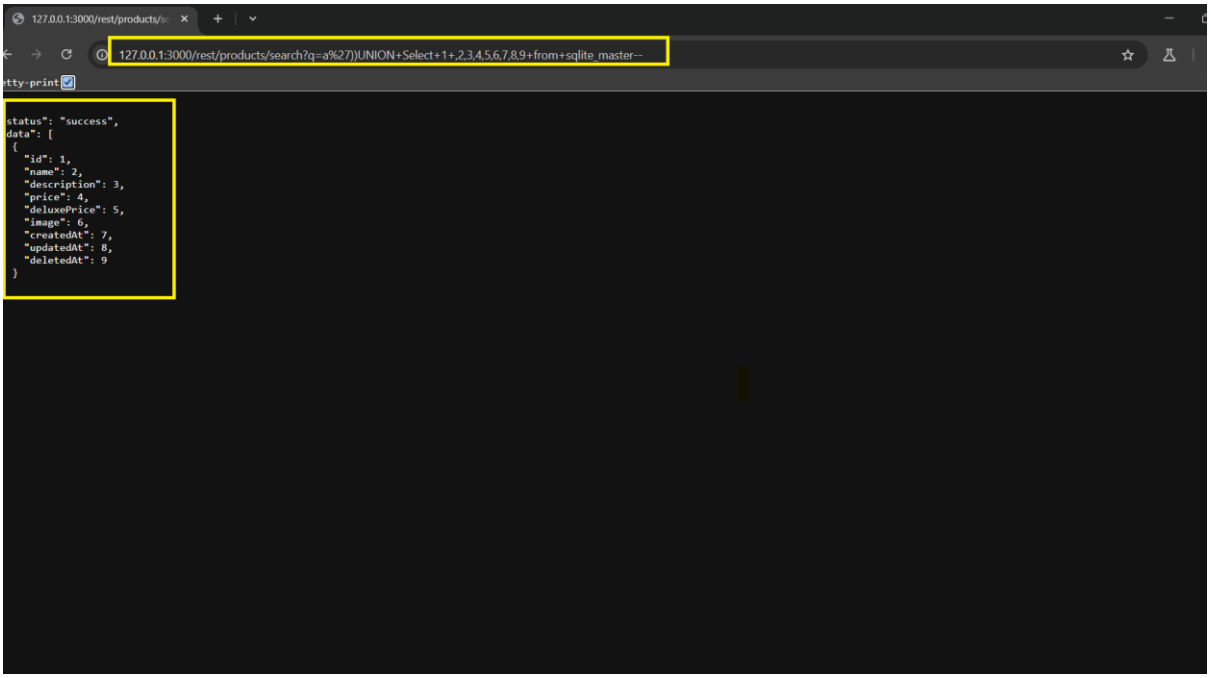
6:Craft a final UNION-based SQL Injection payload to extract the database schema:

a')) UNION SELECT 1,2,3,4,5,6,7,8,9 FROM sqlite_master--



7:Send the modified request and observe that the application returns information from sqlite_master.





6.2.3 Ephemeral Accountant

HIGH

Description:

This vulnerability allows login as a user that does not exist in the application's database. By injecting a custom SQL payload into the login endpoint, a temporary user record is forged and returned by the query. The application accepts this fake user as authenticated, granting access without the user ever being registered. This breaks the integrity of the authentication process.

Impact:

The penetration tester was able to successfully log in using a fabricated user that does not exist in the database. This is a critical weakness in input handling and authentication logic, where forged records can bypass registration and login validation checks entirely.

Vulnerability Location:

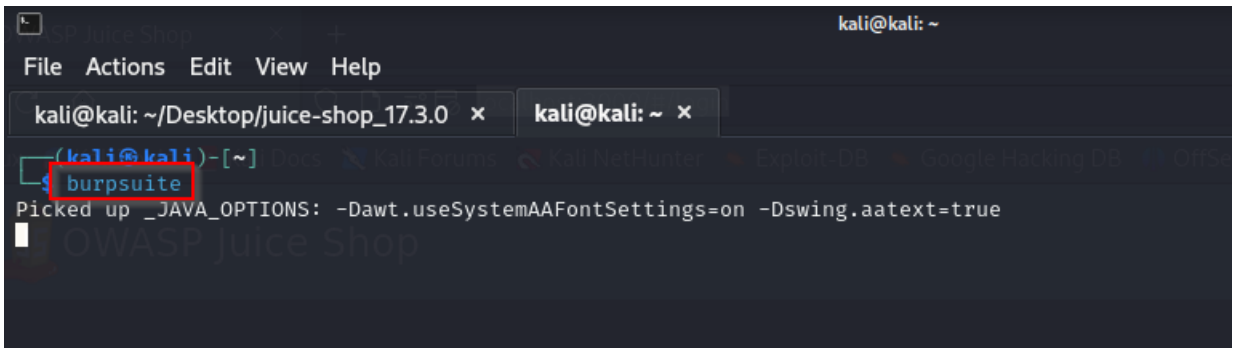
- POST /rest/user/login

Recommendations:

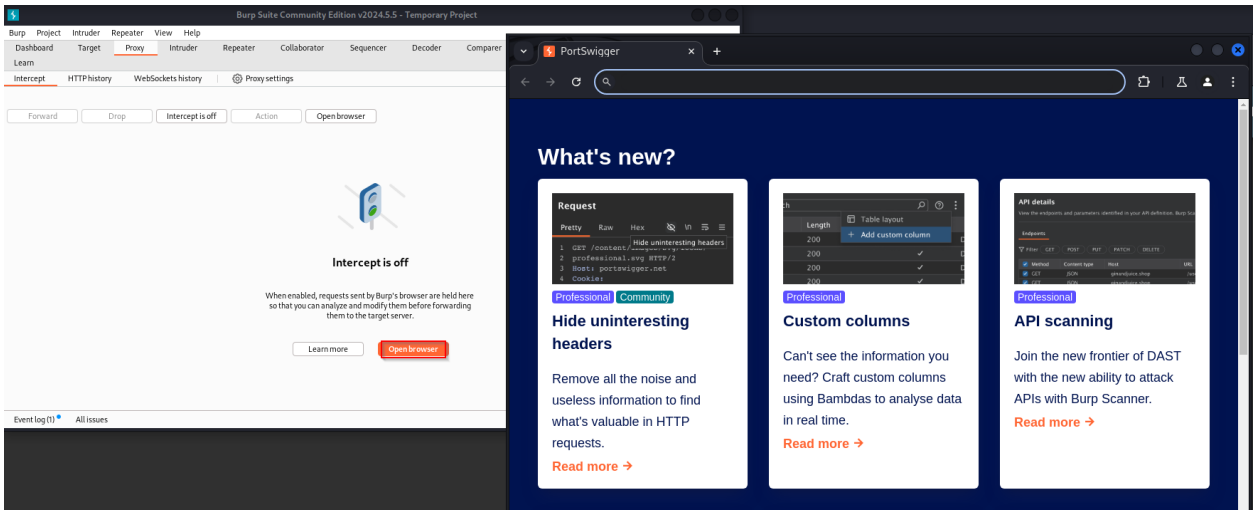
- Use prepared statements (parameterized queries) to prevent injection attacks This approach ensures that the SQL execution engine treats the inputs as data rather than executable code. (https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html)
- Use SQL Server-side verification to ensure that the account where user will log really exist in database by double checking it server-side.
- Input Validation of all user inputs can reduce the risk of injection attacks. Inputs should be checked against expected patterns and sanitized.

Proof Of Concept:

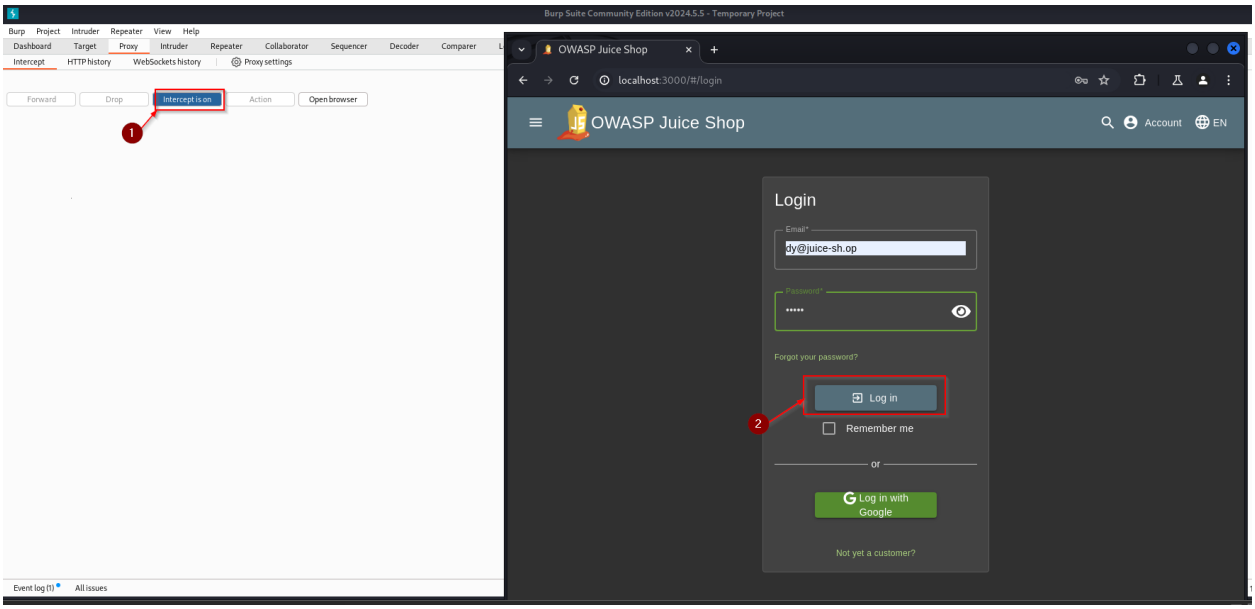
1. Open Burp Suite



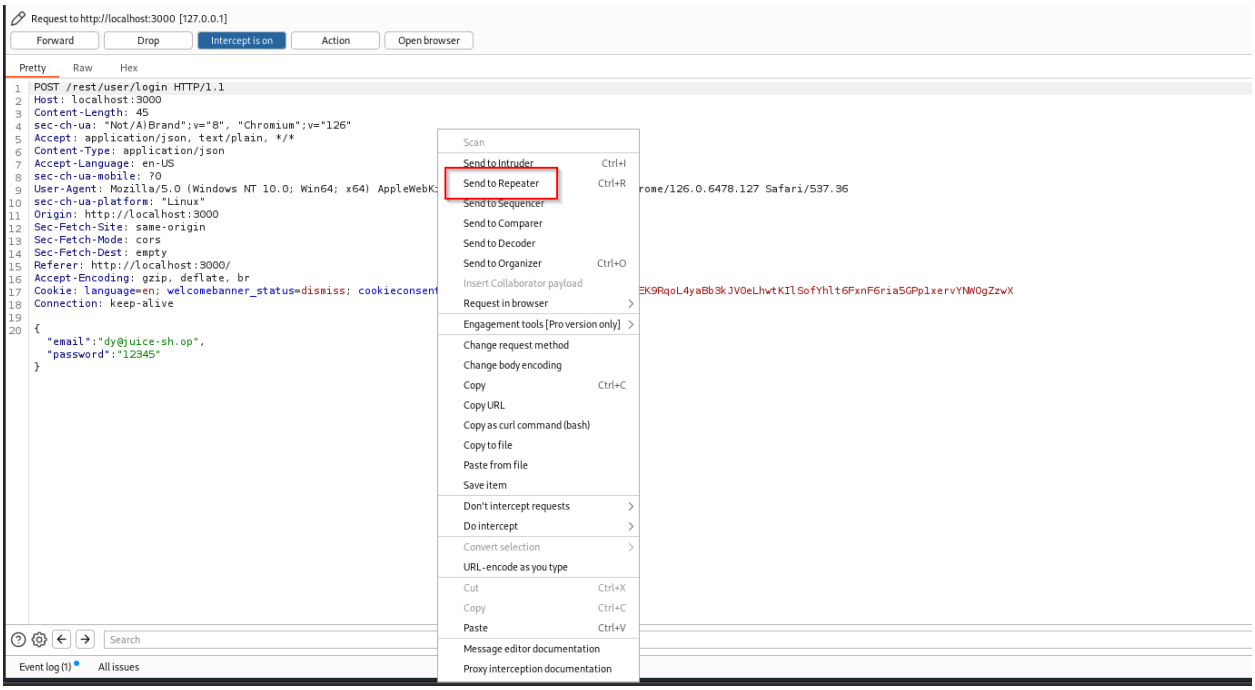
2. Open Browser



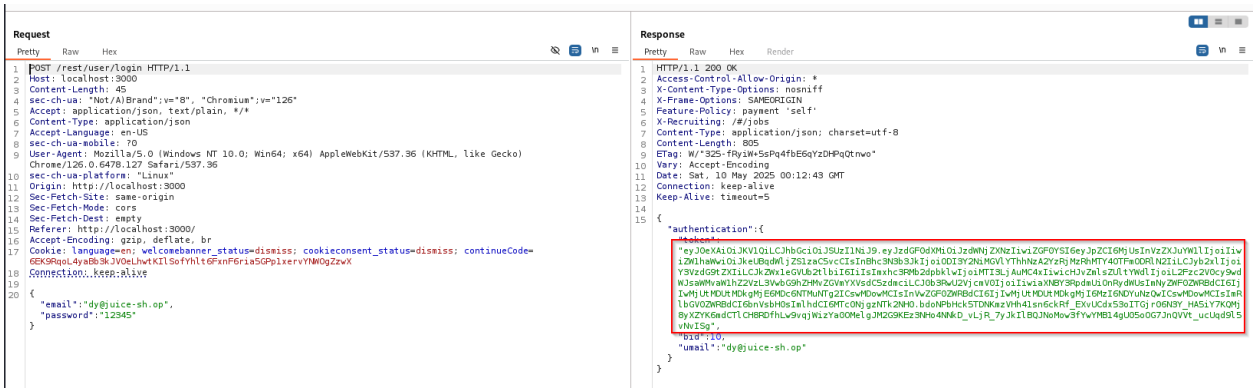
3. turn on Intercept to capture the request and go to Login page and login normally.



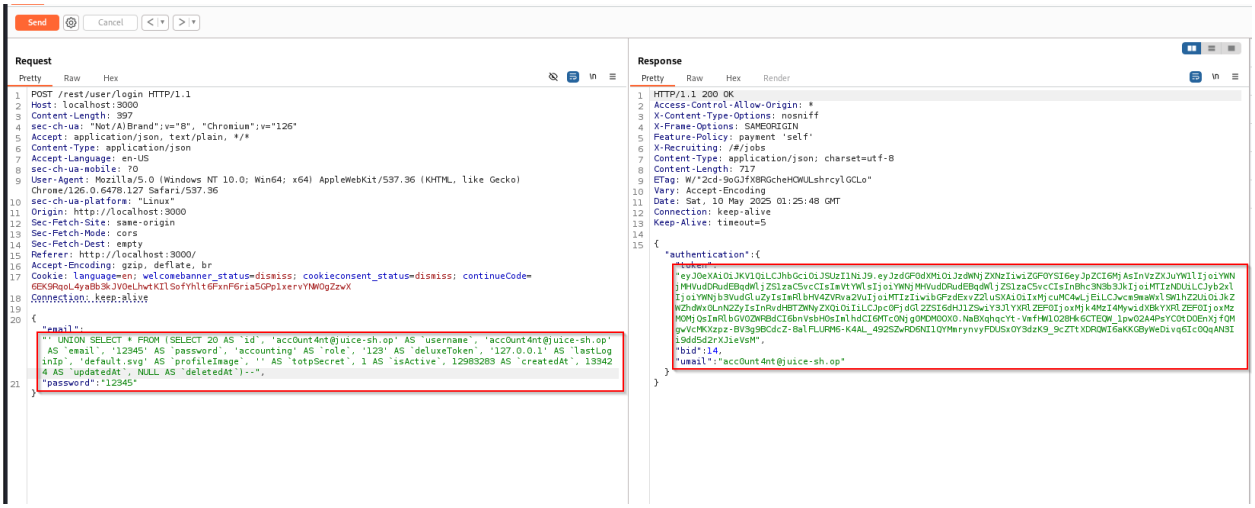
4. Capture the login request and send it to Repeater.



5. Observe the response and request the token parameter means that the login was successful



6. Modify JSON body to inject a SQL payload for the email field:
{“email": "' UNION SELECT * FROM (SELECT 20 AS id, 'acc0unt4nt@juice-sh.op' AS username, 'acc0unt4nt@juice-sh.op' AS email, '12345' AS password, 'accounting' AS role, '123' AS deluxeToken, '127.0.0.1' AS lastLoginIp, 'default.svg' AS profileImage, '' AS totpSecret, 1 AS isActive, 12983283 AS createdAt, 133424 AS updatedAt, NULL AS deletedAt)--"}
Send the request.



The server responds with a valid token means was a successful login with non-existing account

6.2.4 Login Bender

HIGH

Description:

The login functionality is vulnerable to SQL Injection, which allows unauthorized access to user accounts by bypassing authentication mechanisms.
During testing, the tester observed the email bender@juice-sh.op inside the public feedback section of the application. Using this information, a targeted login attempt was made.

By injecting a malicious SQL in the Email field, the application skipped password verification and granted access to Bender's account.

The vulnerability exists because the backend does not properly sanitize or parameterize user input during login processing.

Impact:

- Authentication Bypass: Full unauthorized access to a registered user account.
- Privilege Escalation (if reused): If exploited against admin accounts, this could lead to full compromise.
- Future Attacks: Enables penetration testers to plan further targeted attacks such as extracting sensitive user data.

Vulnerability Location:

- Component: Login Functionality
- Parameter Affected: Email Input Field

Recommendations:

3. Use Prepared Statements: Ensure SQL queries use parameterized inputs to eliminate injection possibilities.

4. Input Sanitization: Reject or escape special characters like ' or -- from inputs where not expected.

References:

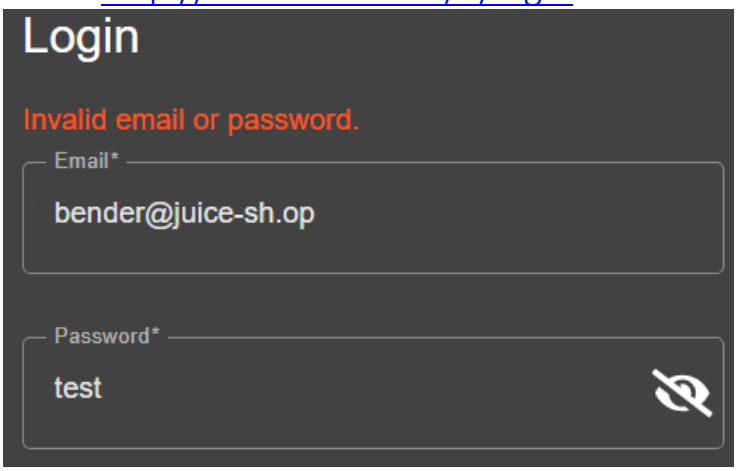
-CWE Reference: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

-CVSS v3.1 Base Score: 8.8 (High)

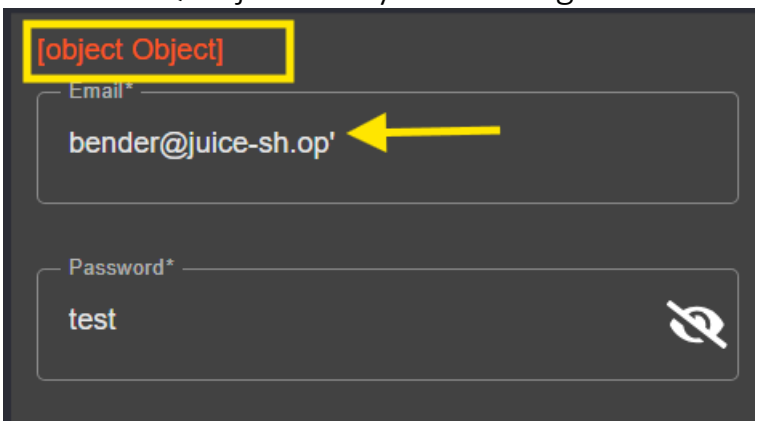
Proof of Concept (PoC):

1- Navigate to the **All Products** section
2- Click on **Banana Juice (1000ml)** to view details and reviews

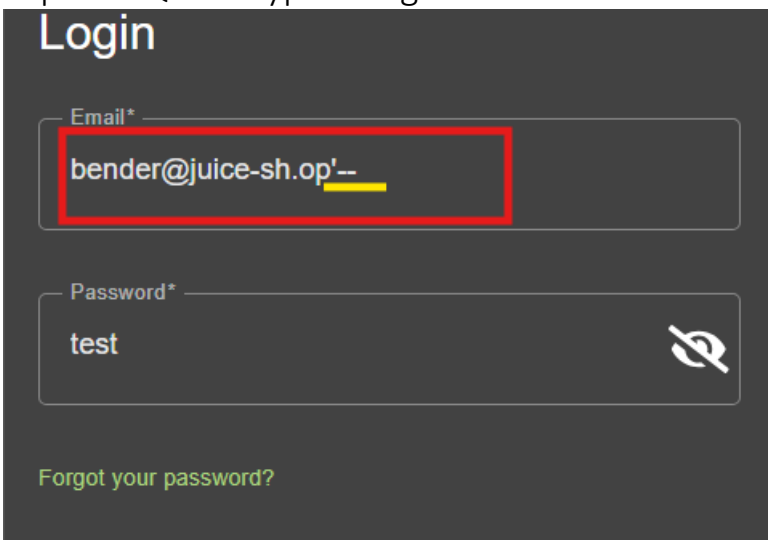
3- Go to http://localhost:3000/#/login

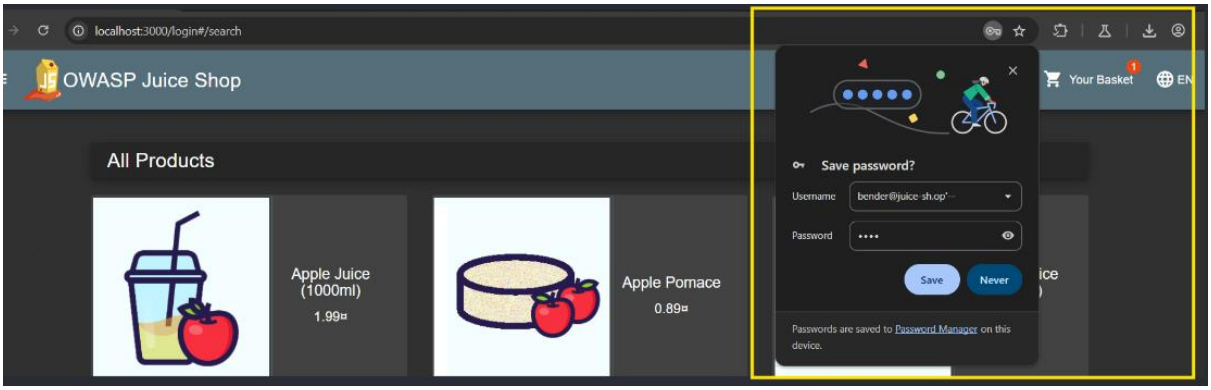


4- Test for SQL injection by submitting:



5- Exploit SQLi to Bypass Login




localhost:3000/login#/search
OWASP Juice Shop
Your Basket
All Products
Apple Juice (1000ml)
1.99¤
Apple Pomace
0.89¤
Save password?
Username
bender@juice-sh.op'
Password
Save
Never
Passwords are saved to Password Manager on this device.

6.3.1 NoSQL Manipulation

HIGH

Decription:
The product review functionality in Juice Shop was found to be vulnerable to a **NoSQL injection**, allowing a **pentester** to unintentionally modify **multiple product reviews** instead of just one.

During testing, the pentester submitted a product review while intercepting the request, the pentester inserted a NoSQL query. This caused the server to match all existing reviews where the ID was not equal to -1, which resulted in **bulk modification** of all reviews.

This indicates a lack of input validation and insufficient protection against NoSQL-specific query manipulation.

Impact:

- **Modify the content and author fields of all existing product reviews.**
- **Demonstrate how NoSQL operators can lead to unexpected data changes.**

Vulnerability Location:

- **Component:** Product Review
- **Endpoint:** /rest/products/reviews

Resource / References:

- OWASP Top 10: A03:2021 – Injection
- MongoDB $ne Operator: https://www.mongodb.com/docs/manual/reference/operator/query/ne/

Recommendations:

1- Implement strict server-side validation for all request payloads.

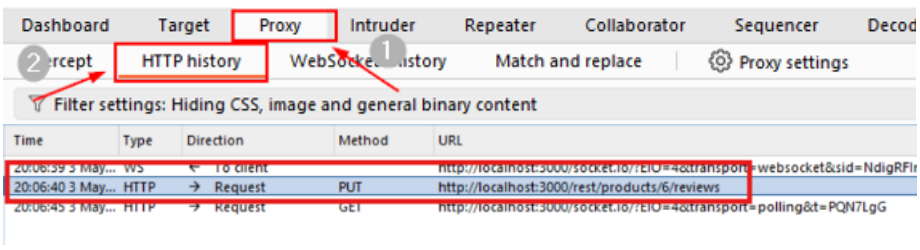
2-Disallow the use of MongoDB operators like $ne in user-supplied fields unless explicitly needed.

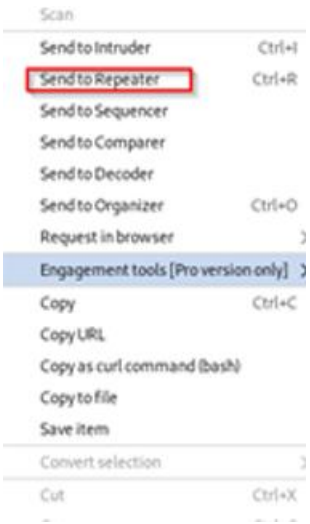
1

Proof of Concept (PoC):

1. Login to the site as a regular user.
2. Go to any product and submit a new review.

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

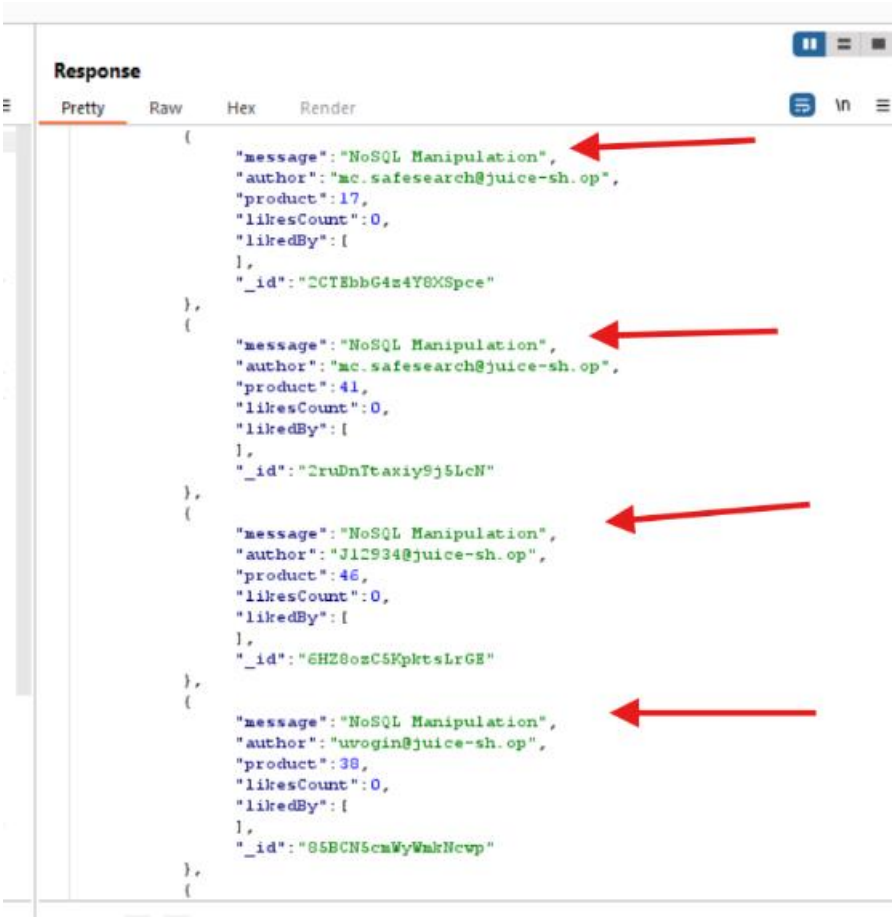




4. Open repeater tap and Change the request method from PUT to PATCH like this:

5. In the JSON body, change the "id" part to this and send the request

6. **Result:** All product reviews were updated to show the new message.



6.4 Information Disclosure

6.4.1 security.txt

MEDIUM

Description:
During the reconnaissance phase, the file /security.txt was accessed and analyzed. This file is designed to provide security researchers with contact details and other information for responsible disclosure. While the file was present and included several fields, two issues were identified:

1. The Contact field uses the email donotreply@owasp-juice.shop, which implies that submitted reports may not be monitored or responded to.
2. The CSAF (Common Security Advisory Framework) field points to a localhost URL: http://localhost:3000/.well-known/csaf/provider-metadata.json, which is inaccessible from external users and likely a placeholder.
3. \score-board Endpoint

These issues may obstruct proper vulnerability disclosure and reduce the file's effectiveness.

Impact:
While this is not a direct exploit, the misconfigurations present can lead to the following risks:

- **Missed Vulnerability Reports**: If security researchers cannot communicate with the organization due to the unmonitored email, critical vulnerabilities may go unreported.
- **False Sense of Security**: Including a broken or local CSAF link can mislead researchers into thinking the organization supports structured disclosure processes when it does not.
- **Found the page of score**-board

Vulnerability Location:

- File path: https://owasp-juice.shop /security.txt

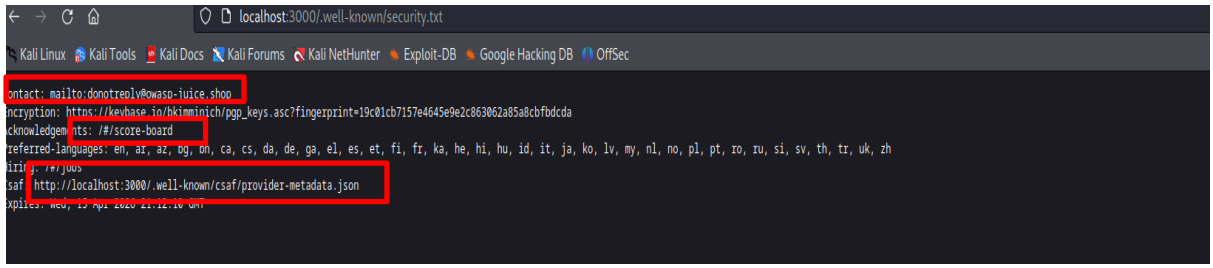
CVE Reference: https://www.rfc-editor.org/rfc/rfc9116.html#name-location-of-the-securitytxt

Recommendations:

1. Replace the Contact Email
   Use a monitored address (e.g., security@owasp-juice.shop) to encourage responsible disclosure.
2. Fix the CSAF Field
   Point the CSAF field to a valid, publicly accessible CSAF provider metadata URL or remove the field if unused.
3. Validate with Tools
   Use a compliance checker such as securitytxt.org to verify the accuracy and structure of the file.

Proof of Concept:

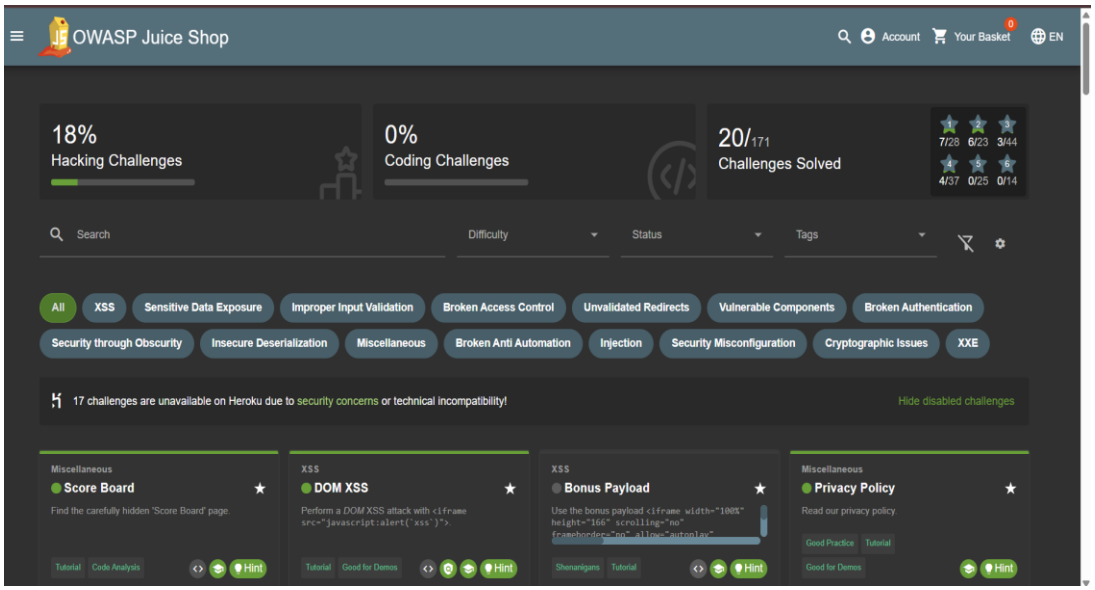
1. Open a browser and navigate to:
   https://127.0.0.1 /security.txt



2. Review the file content:
   - Note that the contact email is donotreply@owasp-juice.shop
   - Observe the CSAF field:
     http://127.0.0.1/.well-known/csaf/provider-metadata.json

```
{
  "canonical_url": "http://localhost:3000/.well-known/csaf/provider-metadata.json",
  "distributions": [
    {
      "directory_url": "http://localhost:3000/.well-known/csaf/"
    }
  ],
  "last_updated": "2024-03-05T20:20:56.169Z",
  "list_on_CSAF_aggregators": false,
  "metadata_version": "2.0",
  "mirror_on_CSAF_aggregators": false,
  "public_openpgp_keys": [
    {
      "fingerprint": "19c01cb7157e4645e9e2c863062a85a8cbfbdcda",
      "url": "https://keybase.io/bkimminich/pgp_keys.asc?fingerprint=19c01cb7157e4645e9e2c863062a85a8cbfbdcda"
    },
    {
      "fingerprint": "2372B2B12AEA7AE3001BB3FBD08FB16E2029D870",
      "url": "https://keybase.io/wurstbrot/pgp_keys.asc"
    },
    {
      "fingerprint": "91b7a09d34db0a5e662ea7546f4a7656807d4ff9",
      "url": "https://github.com/J12934.gpg"
    }
  ],
  "publisher": {
    "category": "vendor",
    "name": "OWASP Juice Shop",
    "namespace": "/juice-shop/juice-shop",
    "contact_details": "timo.pagel@owasp.org"
  },
  "role": "csaf_trusted_provider"
}
```


3. Enter to score-board path



6.4.2 Exposed Metrics

MEDIUM

Description:

The application exposes internal metrics data through an unauthenticated and accessible endpoint **(/metrics).** These metrics are typically scraped by monitoring systems like **Prometheus**, but if not properly protected, they can leak sensitive operational information such as memory usage, request paths, user-agent headers, error rates, and other internal statistics.

During the test, the tester recognized that Prometheus commonly uses the **/metrics** path on port 9090. The tester then manually added **/metrics** to the application's base URL and was able to access the full set of internal metrics, which should not be publicly available.

Impact:
Unauthorized access to this metrics endpoint can lead to:

- **Information Disclosure:** penetration testers can obtain insights into application behavior, errors, and system load.
- **Reconnaissance:** Helps penetration testers map out potential entry points or performance weaknesses.
- **Privacy Risks:** If any user-specific data or request patterns are exposed.

Vulnerability Location:

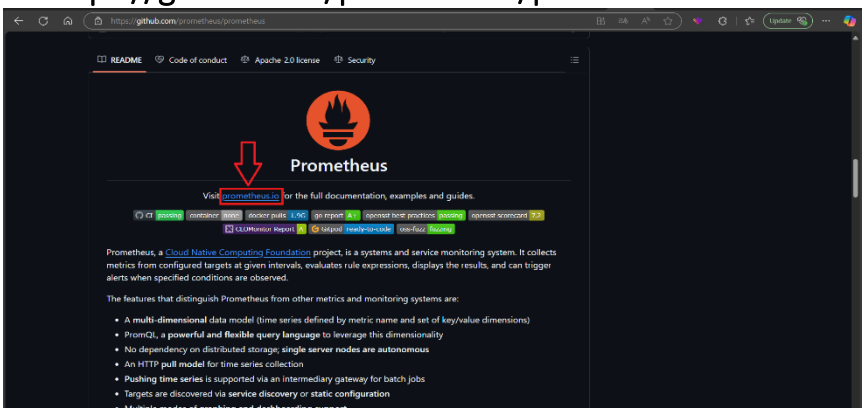
- Endpoint: /metrics
- URL: http://127.0.0.1:3000/metrics

Recommendations:

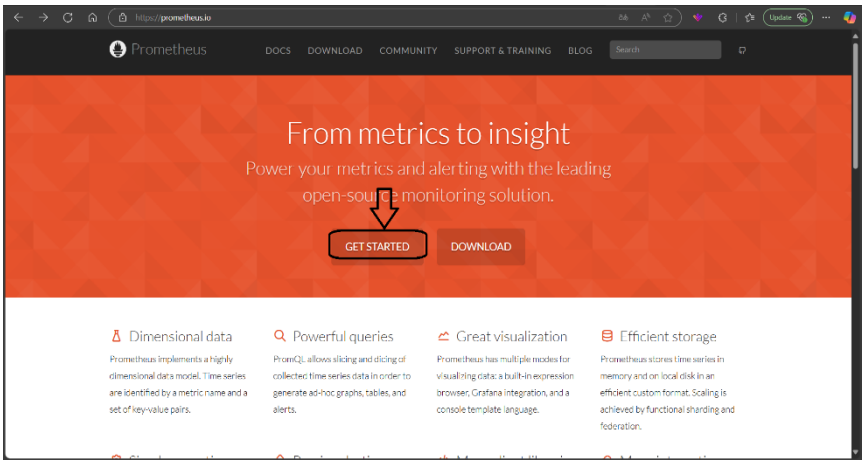
1. Restrict Access to Metrics Endpoint:
    - Apply authentication (**Basic Auth / Token**) or allowlisting (e.g., only Prometheus server can access).
2. Environment-Based Exposure:
    - Expose metrics only in **internal/test environments**. Avoid enabling metrics in production unless strictly needed.

Proof of Concept:

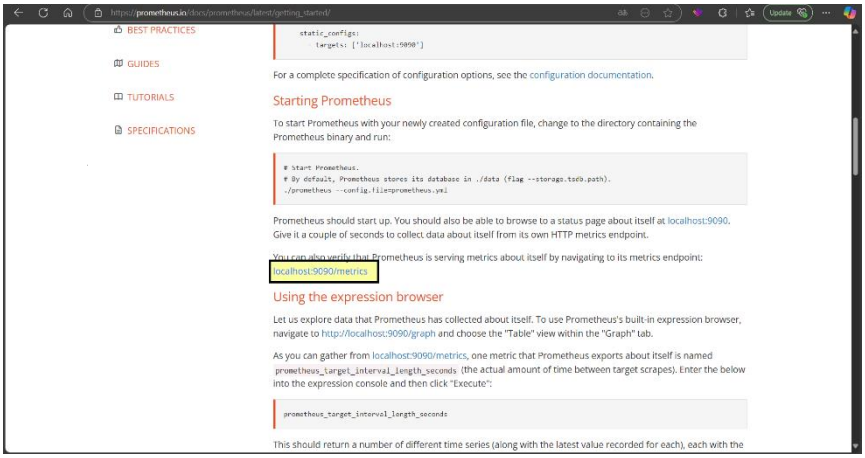
1- Open the official [Prometheus website] or its GitHub repository:
https://github.com/prometheus/prometheus



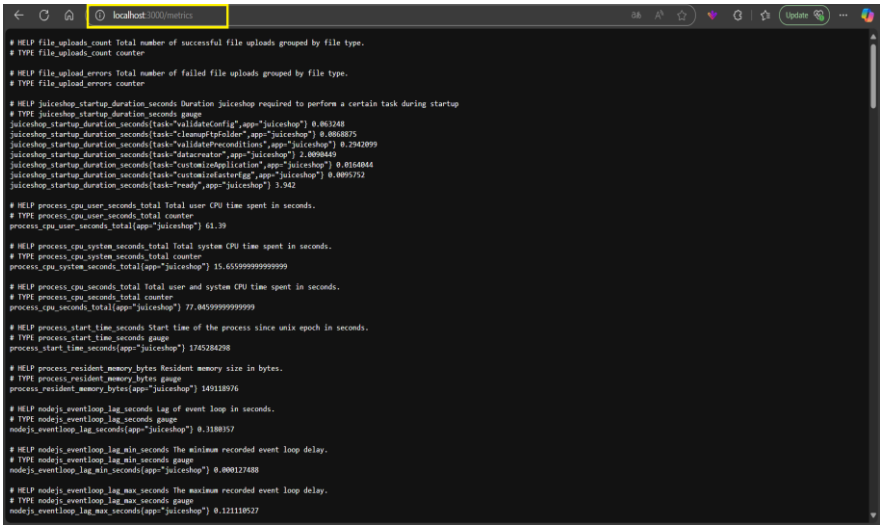
2- click on Get Start



3-find default url use end point (metrics)



4- returned to the OWASP Juice Shop application and manually appended /metrics to the base URL



6.4.3 Forgotten Developer Backup

MEDIUM

Description:

A sensitive backup files (package.json.bak, coupons_2013.md.bak) were accidentally left accessible in the server's /ftp directory. Although the application attempts to restrict access to .md and .pdf files only, the restriction can be bypassed by appending a fake file extension, effectively tricking the file validation mechanism.

Impact:

A penetration tester was able to:

- Bypass file type validation mechanisms using a known null-byte or extension spoofing technique.
- Download a developer's sensitive backup file.

Resource / Reference

- OWASP Top 10: A06:2021 – Vulnerable and Outdated Components
- Null Byte Injection: OWASP – Null Byte Injection

Vulnerability Location:

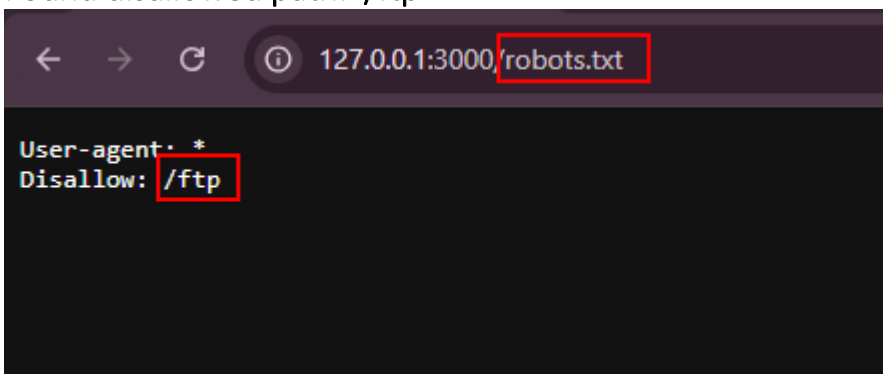
- **Path:**
  - /ftp/package.json.bak
  - /ftp/coupons_2013.md.bak
- **Accessible URL (after bypass):**
  - http://127.0.0.1:3000/ftp/package.json.bak%25%30%30.md
  - http://127.0.0.1:3000/ftp/coupons_2013.md.bak%25%30%30.md
- **Tested From IP:** 127.0.0.1:3000/#/

Recommendations:

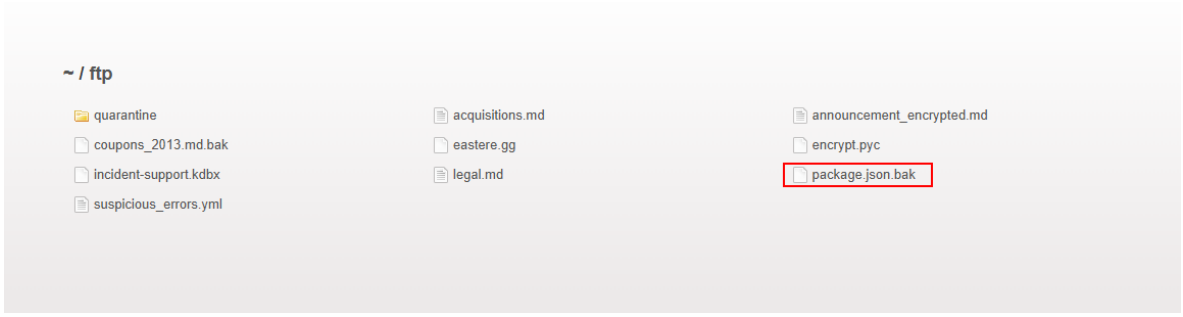
- Strictly validate file extensions on the server-side without relying on filename-based checks.
- Implement MIME-type checking on the server response.
- Remove any sensitive backups from publicly accessible directories.

Proof of Concept (PoC):

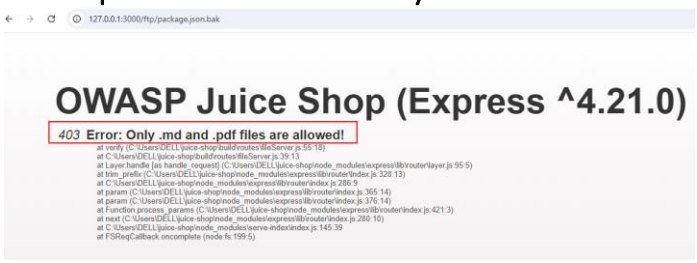
1. Navigated to: http://127.0.0.1:3000/robots.txt
   Found disallowed path: /ftp



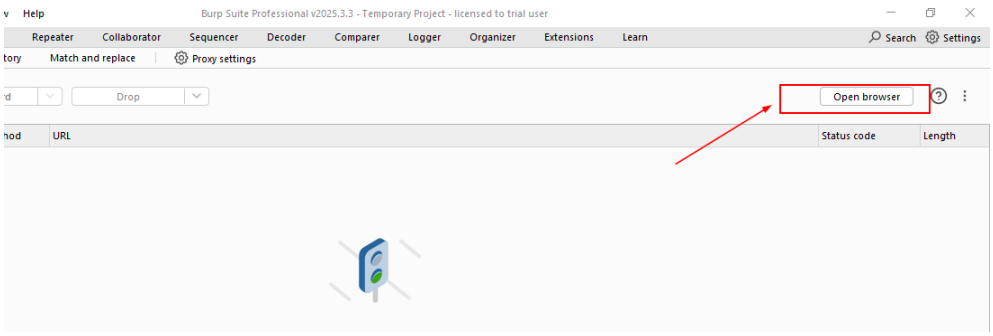
2. Accessed the directory manually: http://172.0.0.1:3000/ftp
   Discovered: package.json.bak



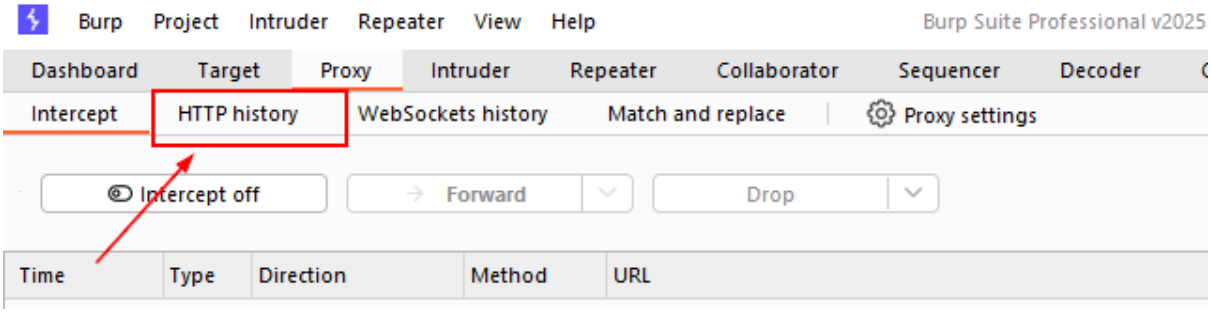
3. Attempted to access directly:



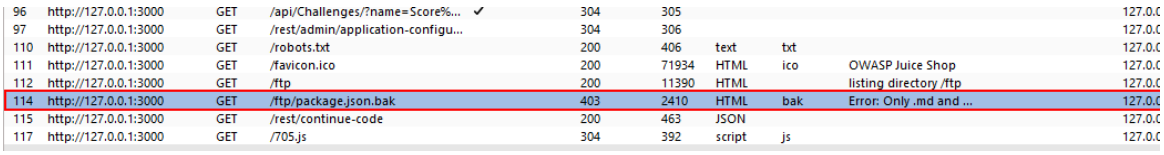
4. Open BurpSuite and repeat the previous steps exactly on the BurpSuite browser

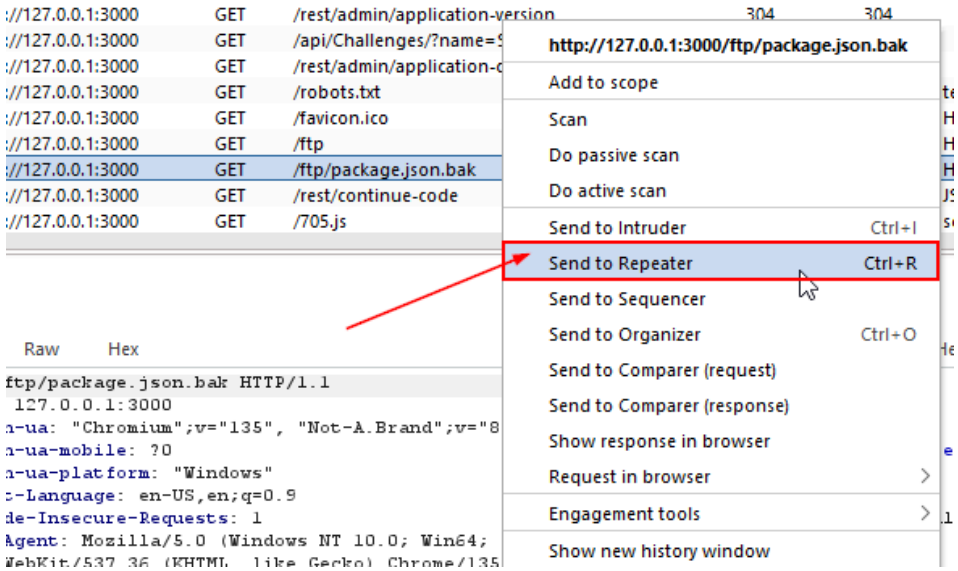


5. Open HTTP History



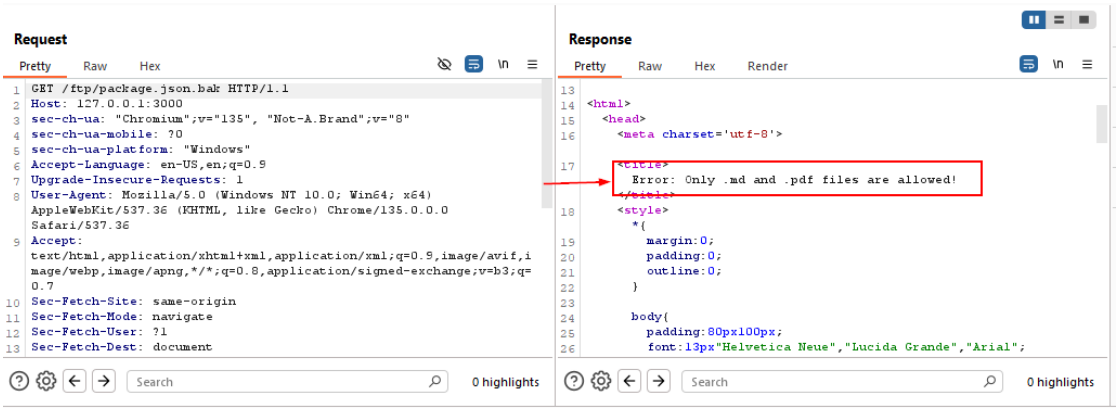
6. Check the HTTP requests in Burp's History tab and send to Repeater



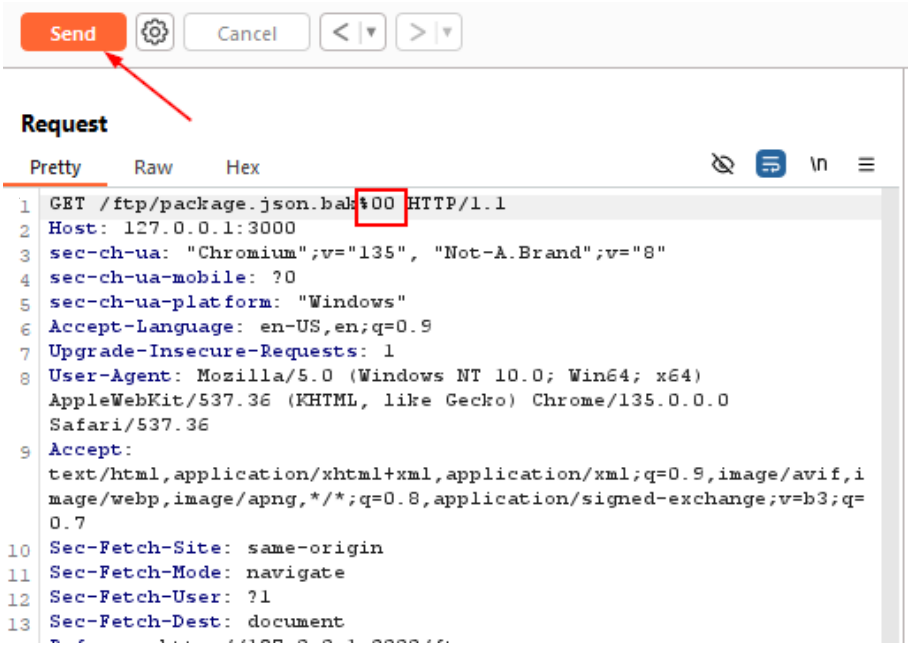


7. GET /ftp/package.json.bak HTTP/1.1

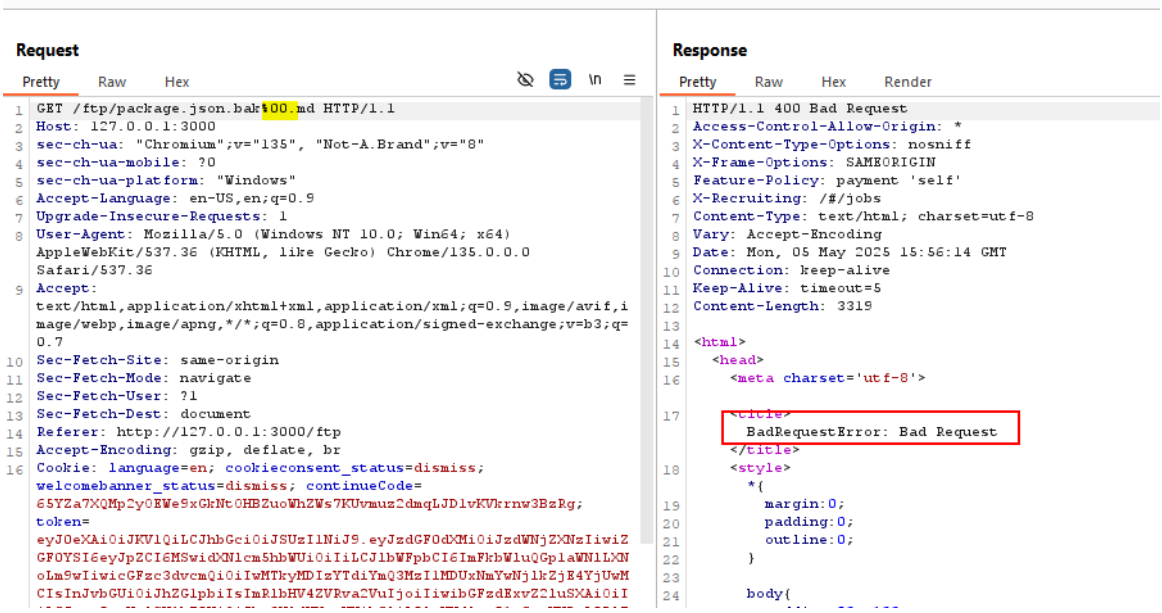
→ Response: 403 – Only .md and .pdf files are allowed



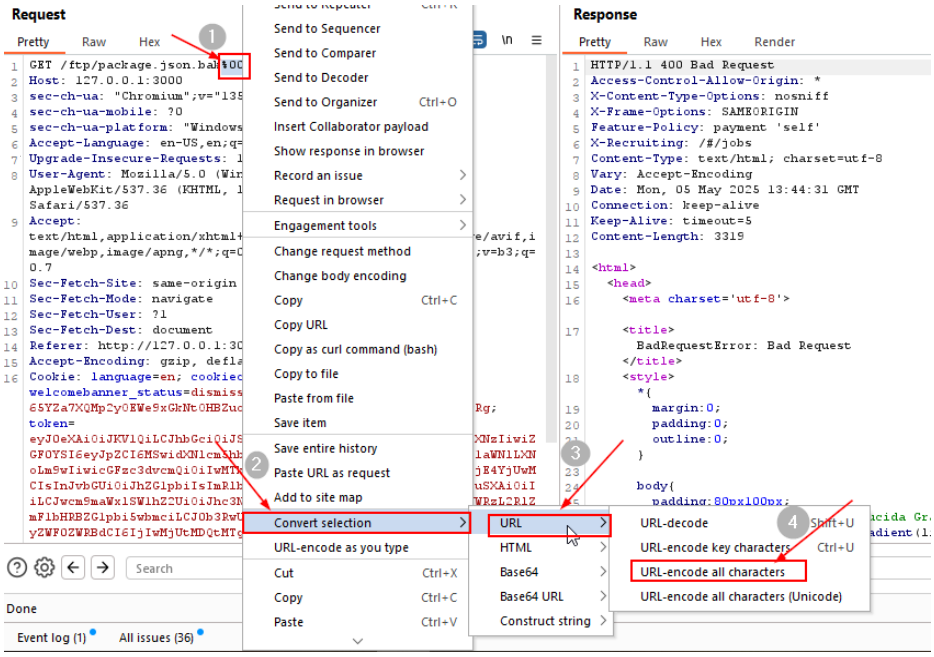
8. Tried appending a null byte:



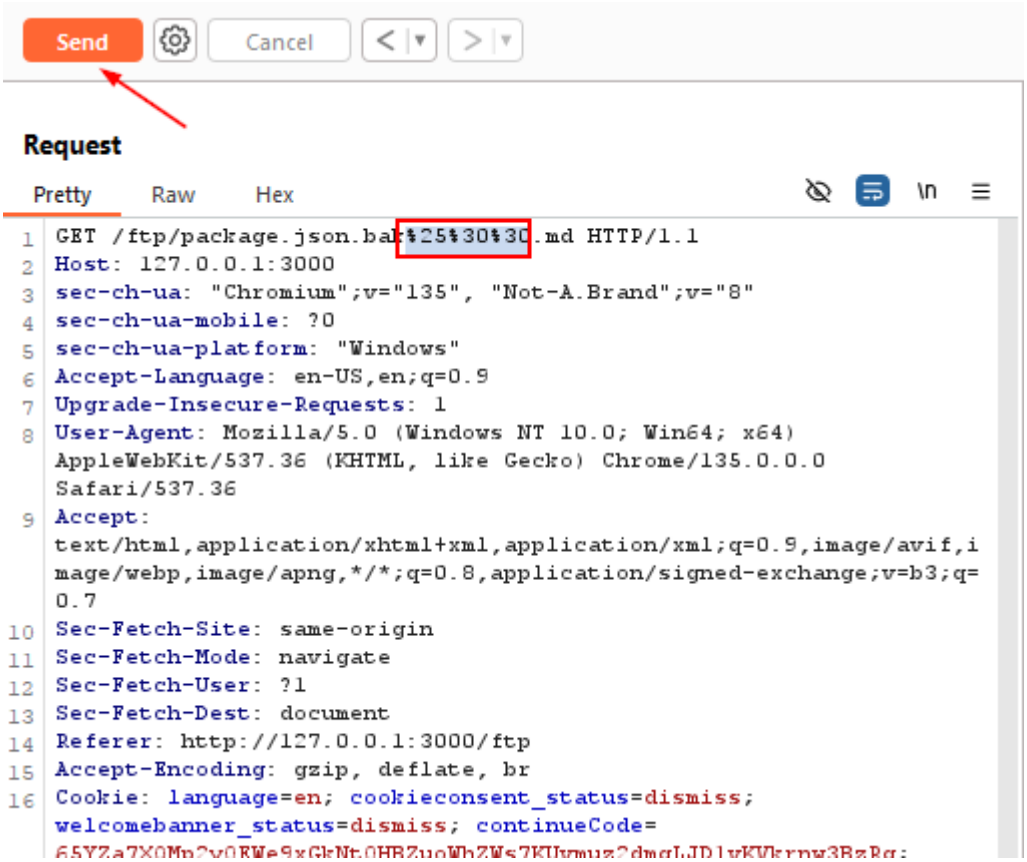
9. Tried %00.md, received "Bad Request":



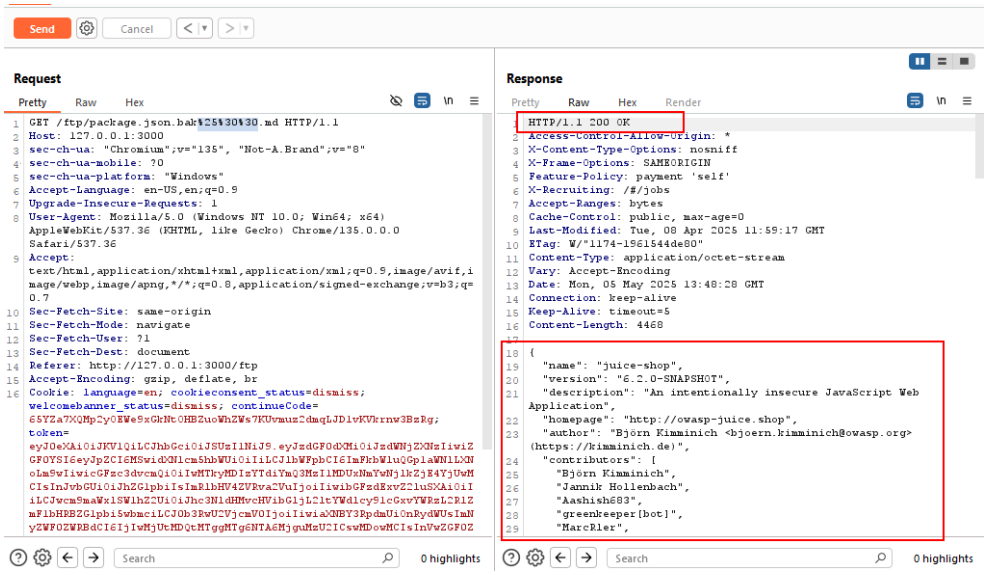
10. Then tried to encode %00:



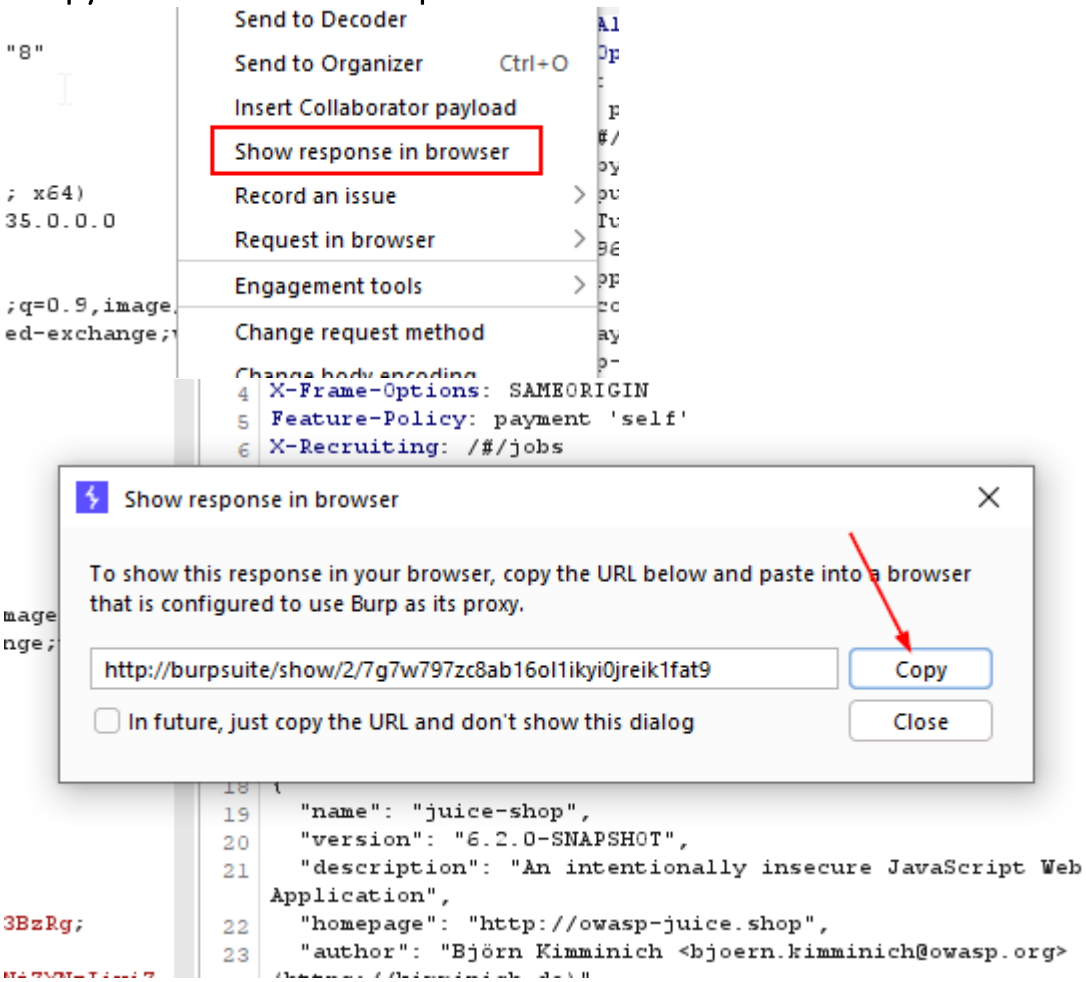
11. And send:



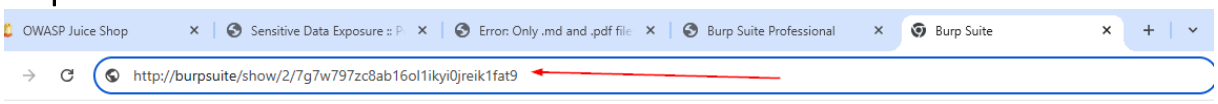
12. Success – File was downloaded:



13. Copy the full URL from BurpSuite



14. Open it in browser:

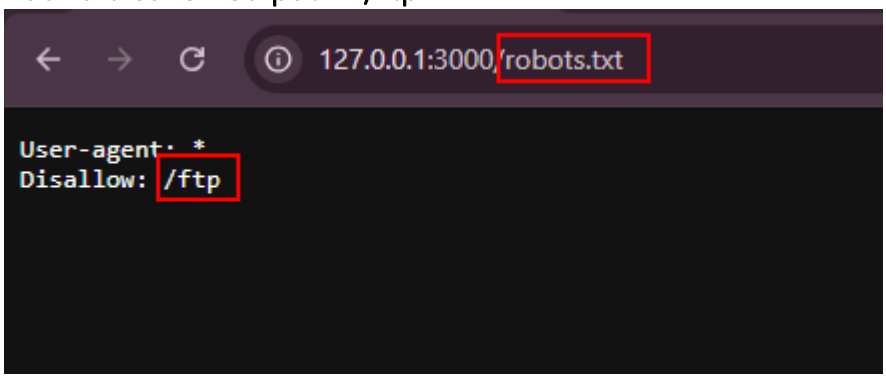


15. Backup file was successfully downloaded and saved locally:

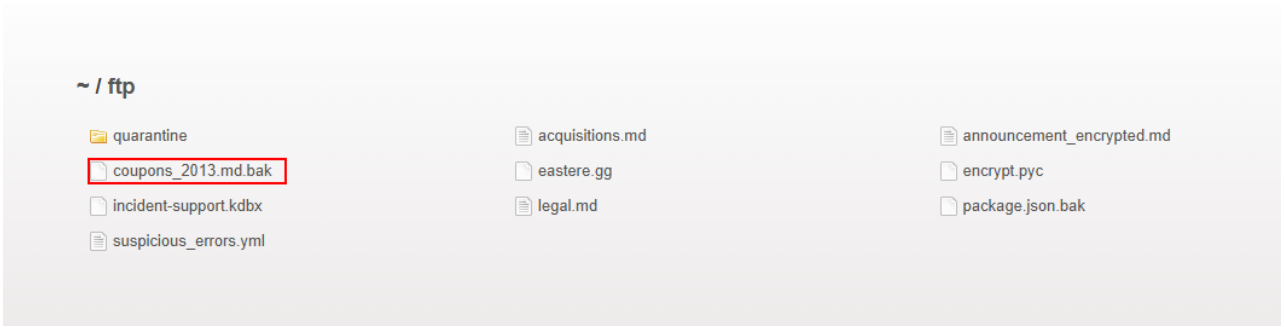
Scenario 2 → **"5.4.4 Forgotten Sales Backup"**:

Proof of Concept (PoC):

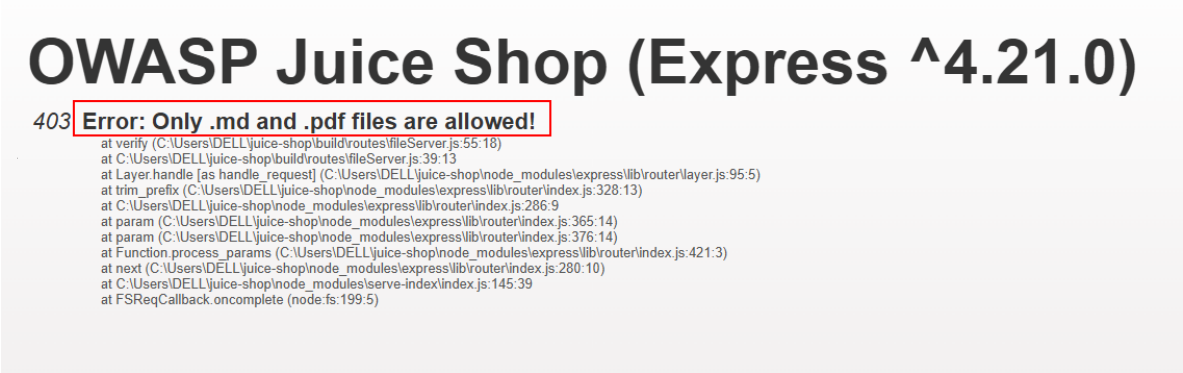
1. Navigated to: http://127.0.0.1:3000/robots.txt
   Found disallowed path: /ftp



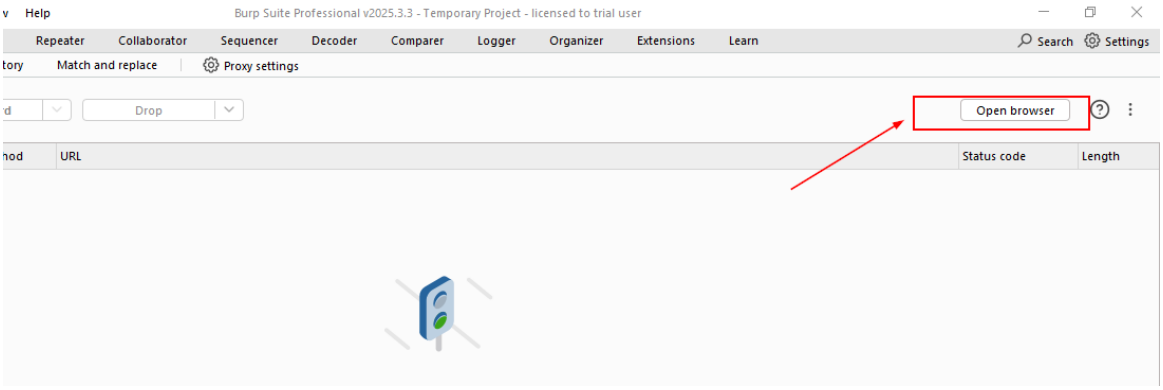
2. Accessed the directory manually: http://172.0.0.1:3000/ftp
   Discovered: coupons_2013.md.bak



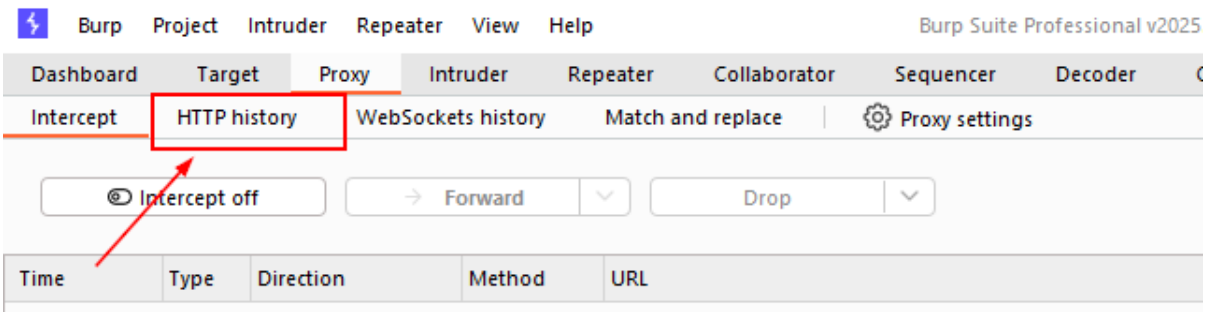
3. Attempted to access directly:



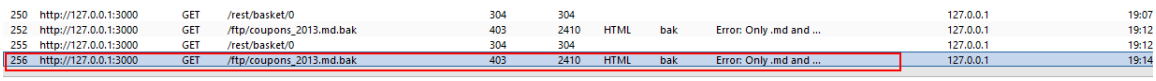
4. Open BurpSuite and repeat the previous steps exactly on the BurpSuite browser

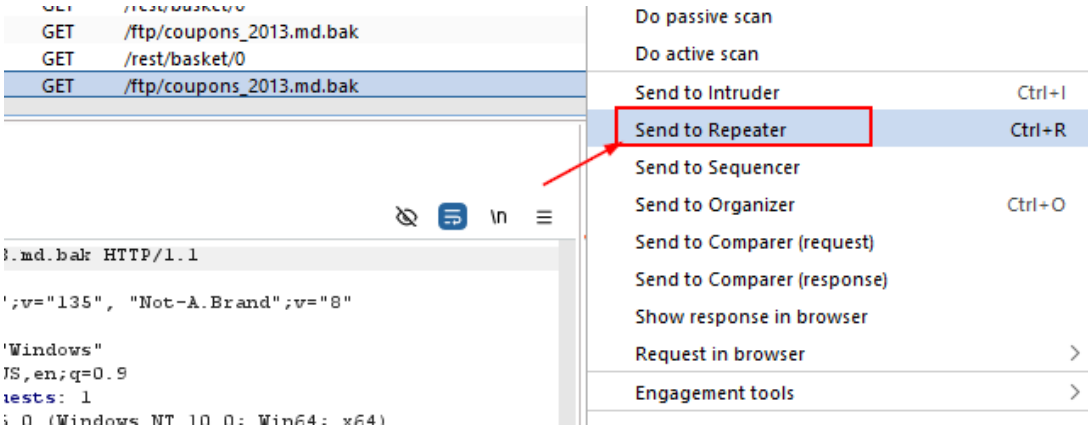


5. Open HTTP History



6. Check the HTTP requests in Burp's History tab and send to Repeater

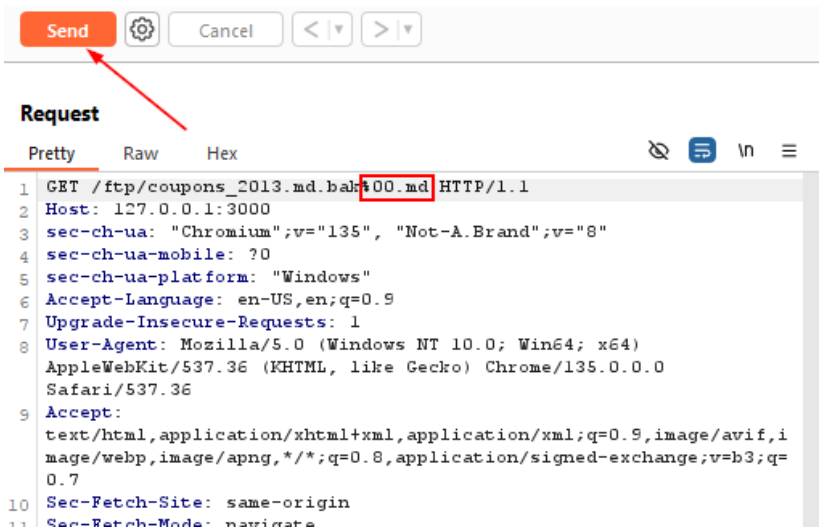




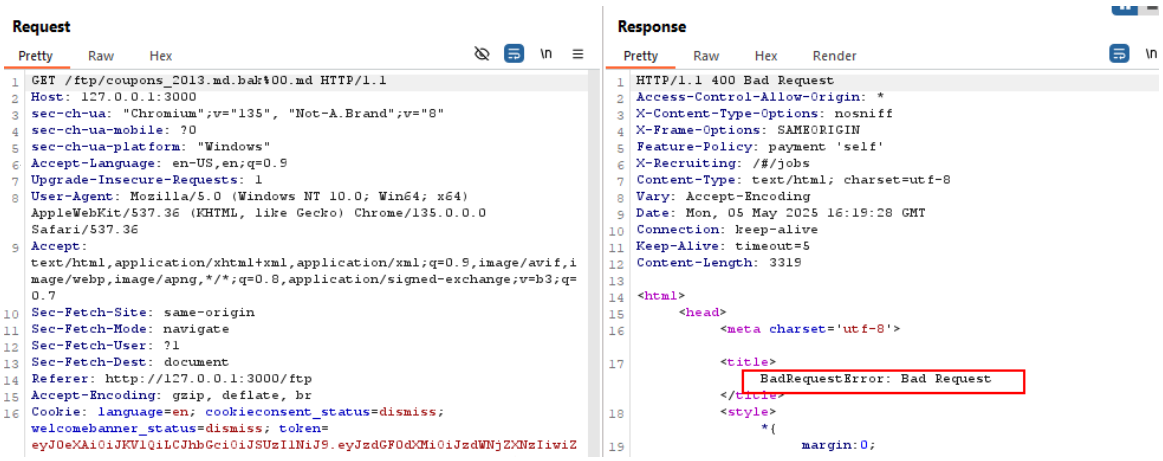
7. GET /ftp/coupons_2013.md.bak HTTP/1.1

   → Error – Only .md and .pdf files are allowed!

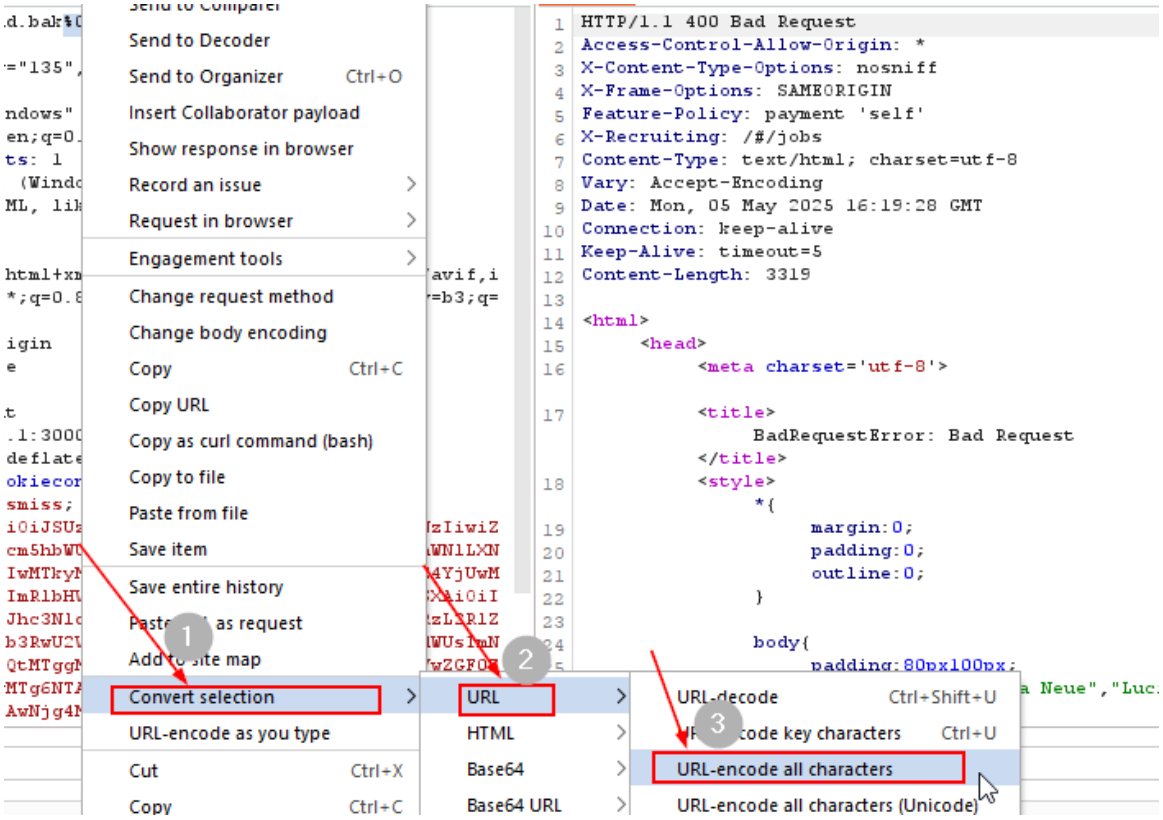
8. Tried appending a null byte:



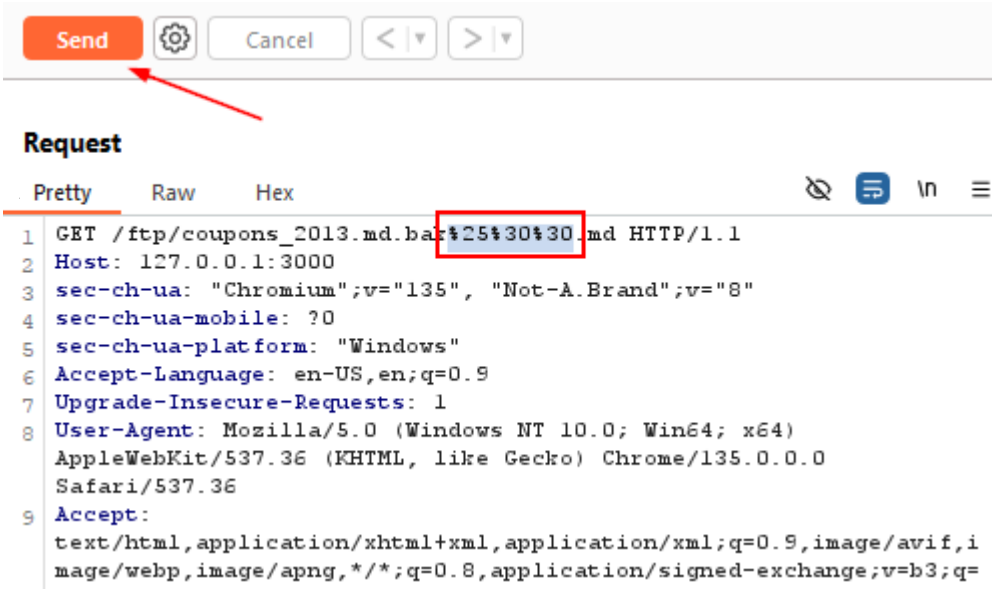
9. Tried %00.md, received "Bad Request":



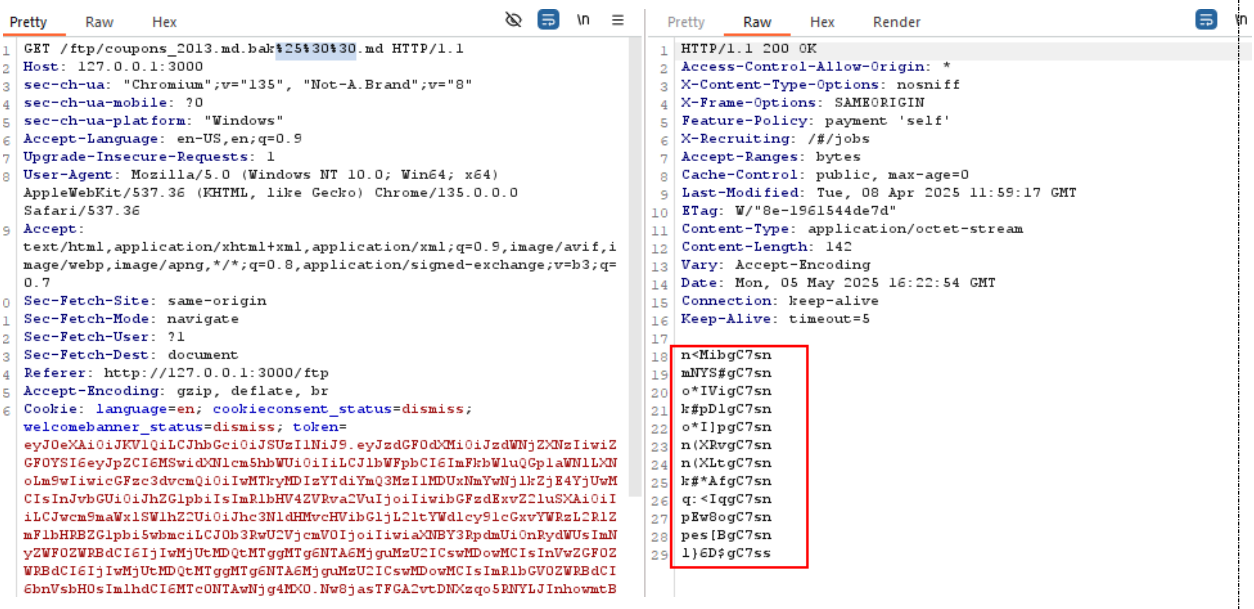
10. Then tried to encode %00:



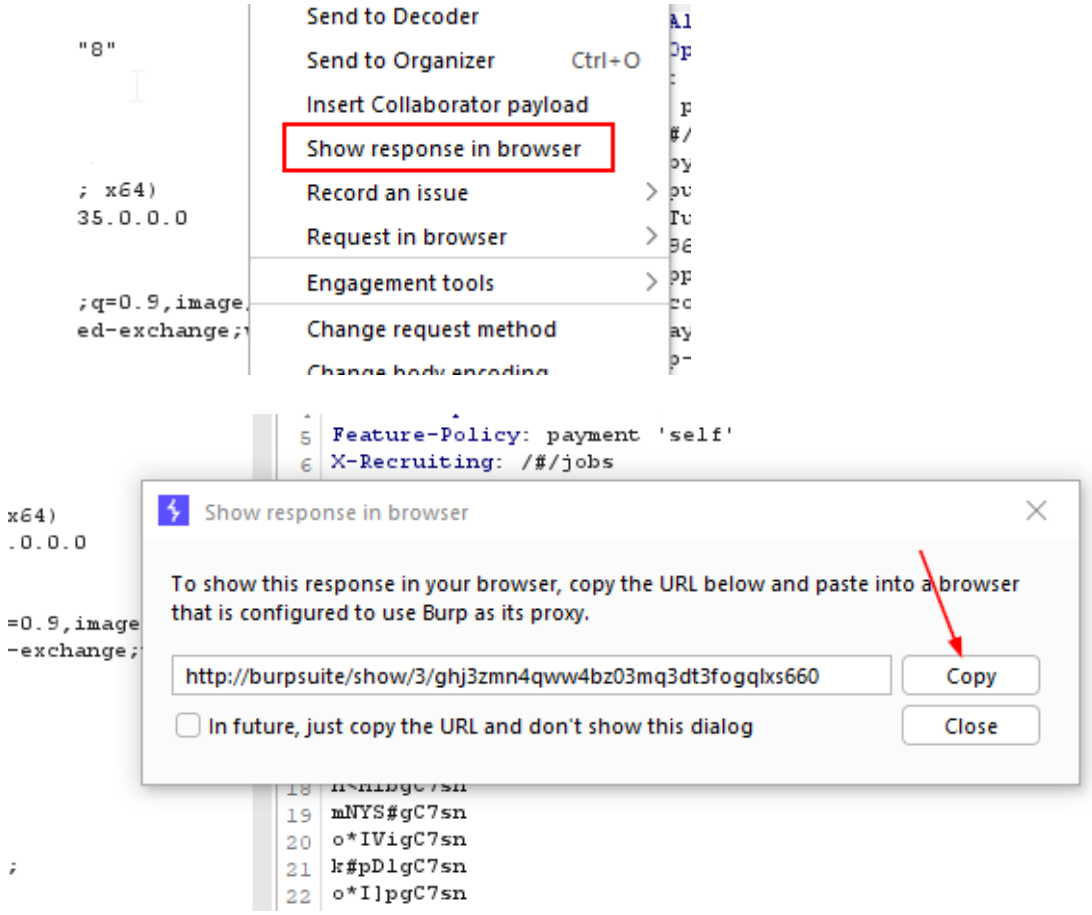
11. And send:



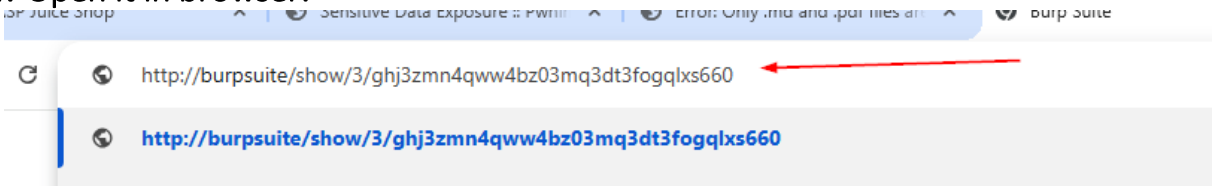
12. Success – File was downloaded:



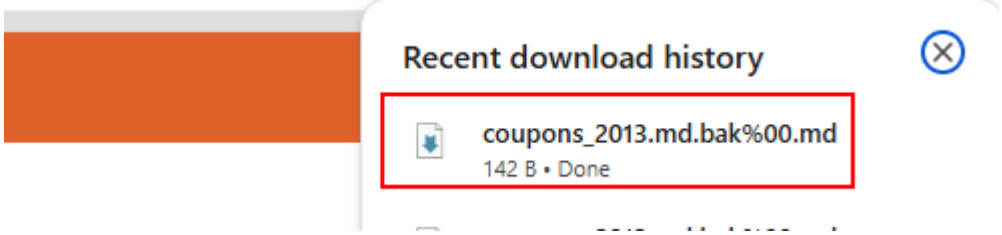
13. Copy the full URL from BurpSuite:



14. Open it in browser:



15. Backup file was successfully downloaded and saved locally:



6.4.6 Confidential Document

MEDIUM

Description:

A serious vulnerability was discovered in the application where a sensitive document containing confidential business acquisition plans can be accessed directly via URL tampering. No authentication or access controls are in place to prevent unauthorized access to this file. The file is exposed publicly and can be found by manually inspecting accessible directories like **/robots.txt** and then navigating to hidden folders listed there.

Impact:

The penetration tester was able to access a highly sensitive internal document revealing private business acquisition strategies. This could lead to:

- Exposure of confidential corporate plans, potentially damaging to the organization.
- Legal and compliance violations (e.g., GDPR, CCPA, etc.).
- Severe reputational harm and potential financial impact.
- Use of the information by malicious competitors or threat actors.

Vulnerability Location:

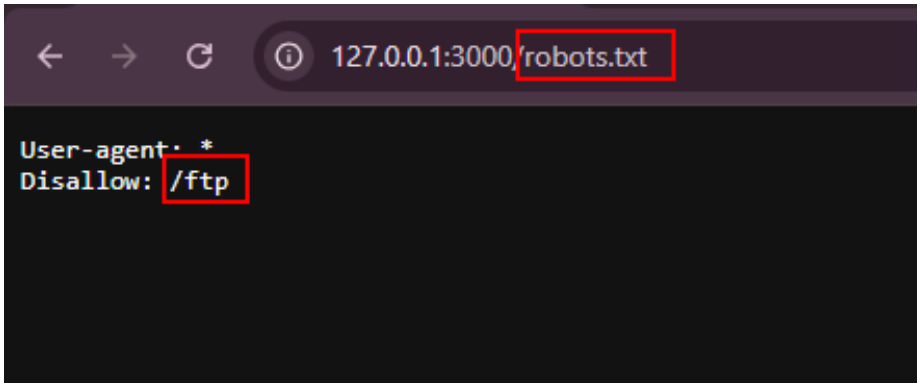
- **Component:** File Access via Direct URL
- **Vulnerable URL:** https://127.0.0.1:3000/ftp/acquisitions.md
- **Initial Clue Found At:** https://127.0.0.1:3000/robots.txt
- **IP Address Used During Testing:** 127.0.0.1:3000/#/

Recommendations:

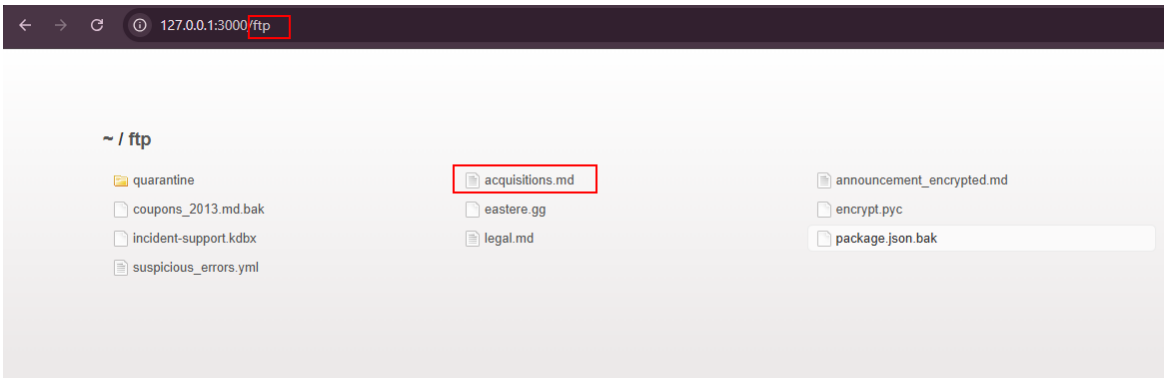
- Restrict access to sensitive files by implementing proper authentication and authorization mechanisms.
- Remove or properly secure any internal folders referenced in robots.txt.

Proof of Concept (PoC):

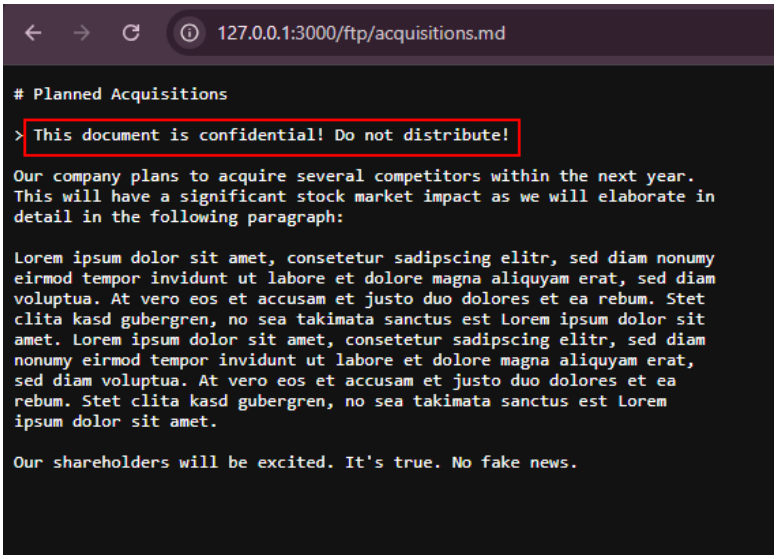
1. Type this into the address bar: 127.0.0.1:3000/robots.txt



2. This file usually tells web crawlers which parts of the site not to index. Inside it, I saw a line that said: Disallow: /ftp
3. That was a hint that there's a hidden directory called /ftp. I went directly to it:



4. There are multiple files listed. click on one of them called: acquisitions.md
5. This file opened immediately without any login or protection, and it contained confidential document.



6.4.6 Leaked Unsafe Product

MEDIUM

- **Description:**
  A previously deleted unsafe product was still accessible via a third-party source due to inadequate data leak prevention. The product and its dangerous ingredients should have been securely deleted from all systems, including backups and online references, but were still available, exposing sensitive product details.

- **Impact:**
  The penetration tester was able to retrieve and view the sensitive composition of a discontinued and unsafe product. This poses a health and safety risk to consumers and demonstrates the failure to fully remove or sanitize sensitive data related to decommissioned items.

- **Vulnerability Location:**
  - Location: Product database

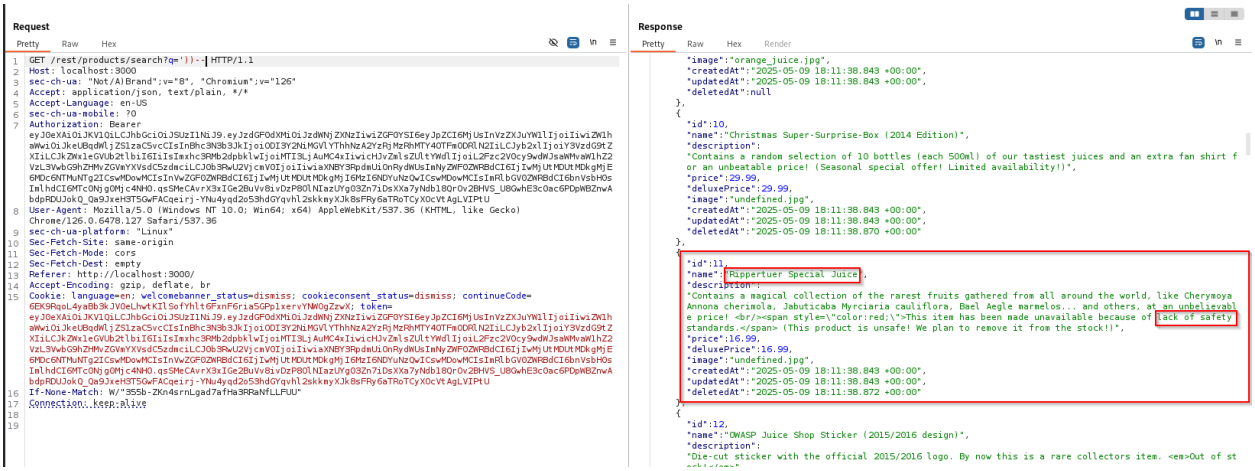
- **CVE Reference:**

- **Recommendations:**
  - Implement data leak prevention (DLP) mechanisms to detect and stop sensitive data from being shared or stored insecurely. (https://owasp.org/wwwcommunity/controls/Data_Leakage_Prevention)
  - Remove deprecated product data from all storage and third-party systems.
  - Monitor external platforms for leaked content and request takedowns when necessary.
  - Apply strong access controls and encryption to internal product databases.
  - Regularly audit database content for outdated or sensitive information.

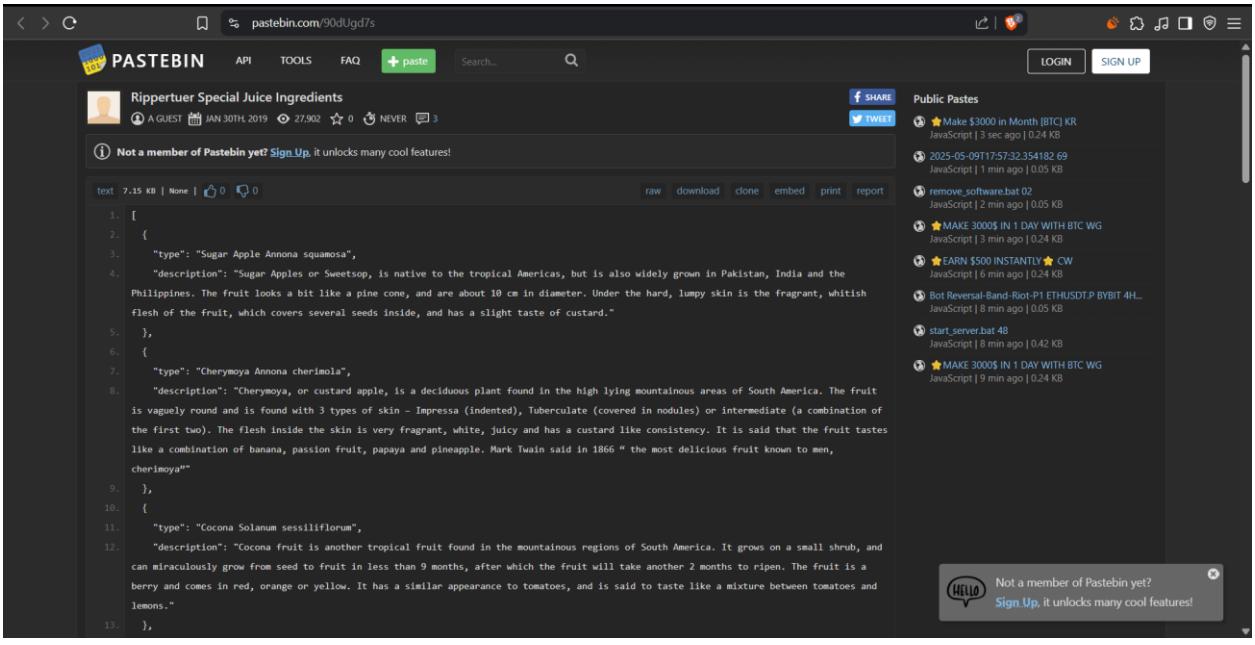
- **Proof Of Concept:**

  1. Used a previous SQL Injection vulnerability to access the database containing product records, including removed ones.

  2. Review the database and identify a removed product due to safety concerns.

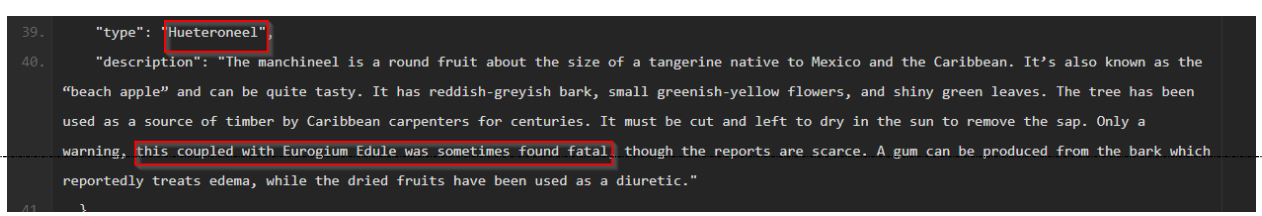


  3. searched for the product name online and “Rippertuer Special Juice".

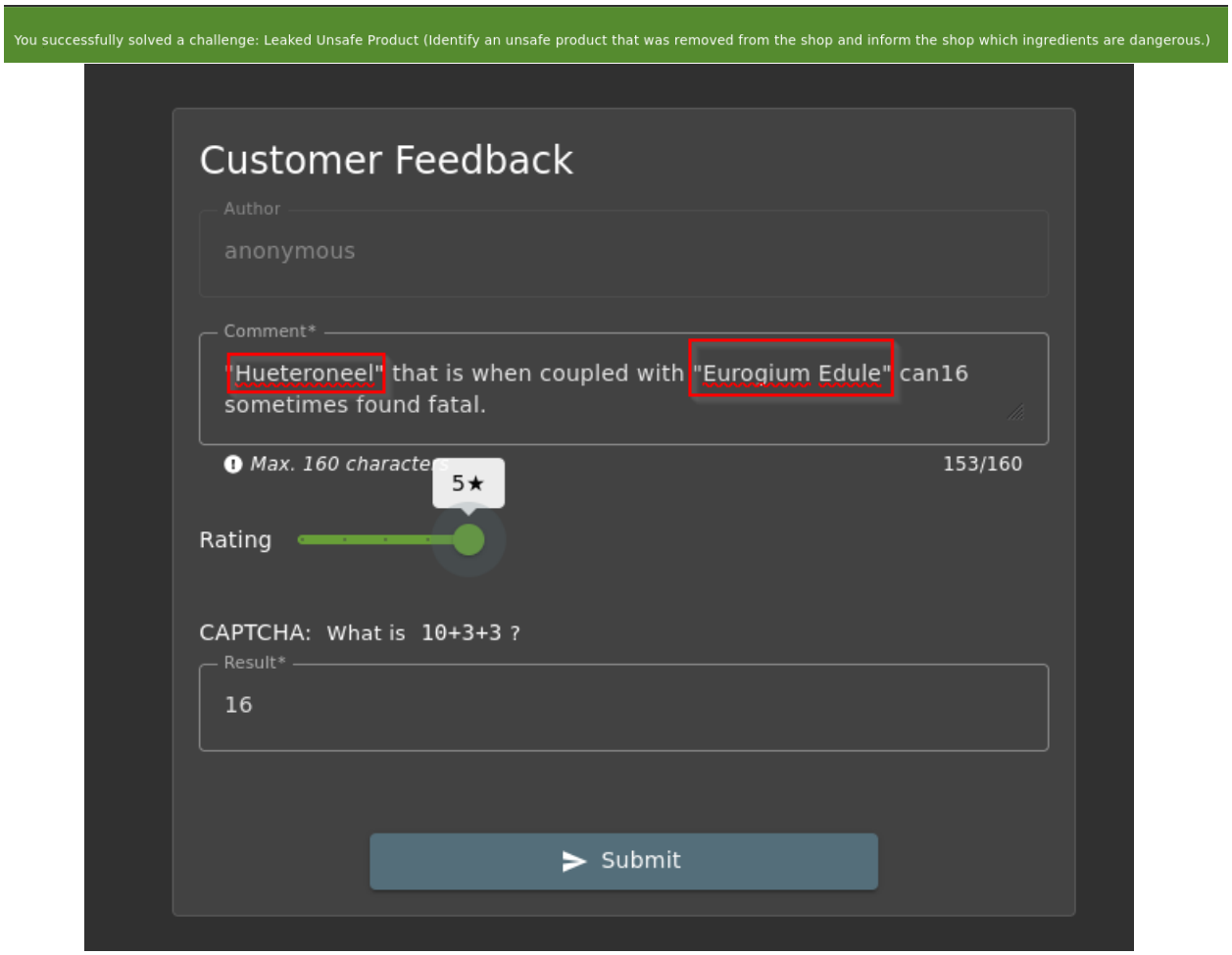


https://pastebin.com/90dUgd7s

  4. Review the exposed content and identify the dangerous components.



5. Report the unsafe product and leaked data to the website.



This kind of exposure requires proper data leakage prevention, backend security measures to protect against unauthorized data retrieval and complete removal from public access.

5.4.7 Privacy Policy

INFORATIONAL

Description:
The application's **Privacy Policy page is only accessible after user authentication**. This restricts unauthenticated users from reviewing the platform's data handling practices before account creation.

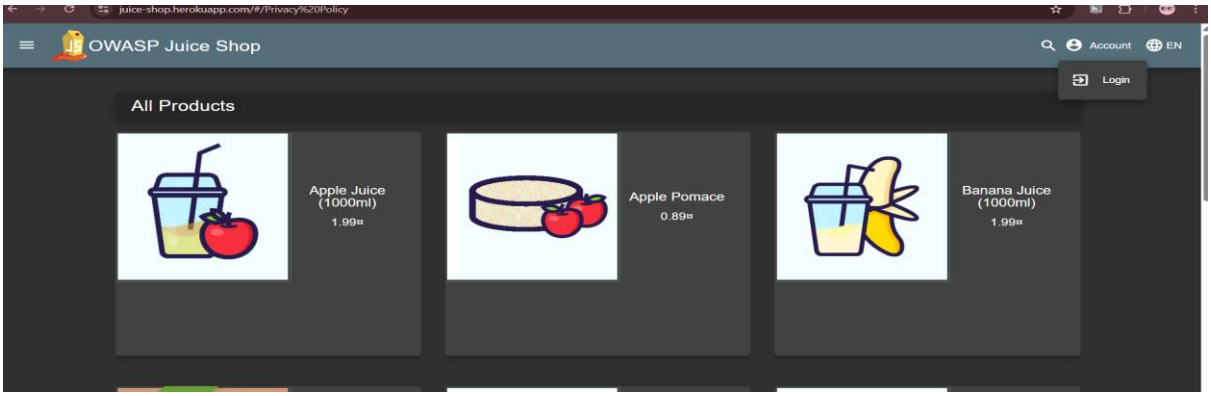
Impact:
While not a technical vulnerability, this behavior may **violate transparency and compliance standards** (e.g., GDPR Article 12), which require privacy policies to be publicly accessible.

Vulnerability Location: 127.0.0.1**/#/Privacy%20Policy**

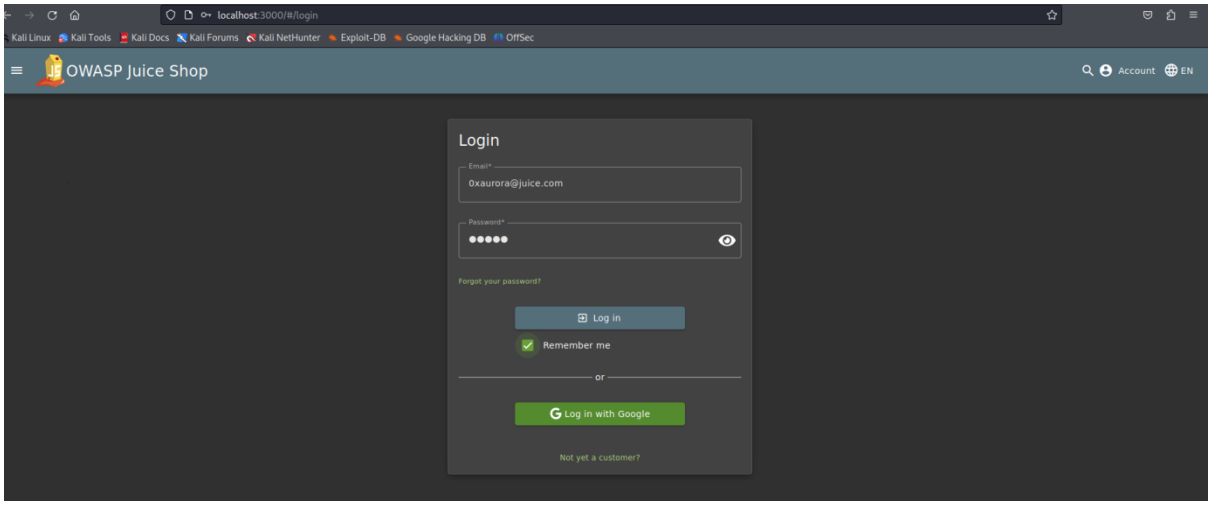
Recommendation:
Allow unauthenticated access to the Privacy Policy page to align with privacy regulations and improve transparency.

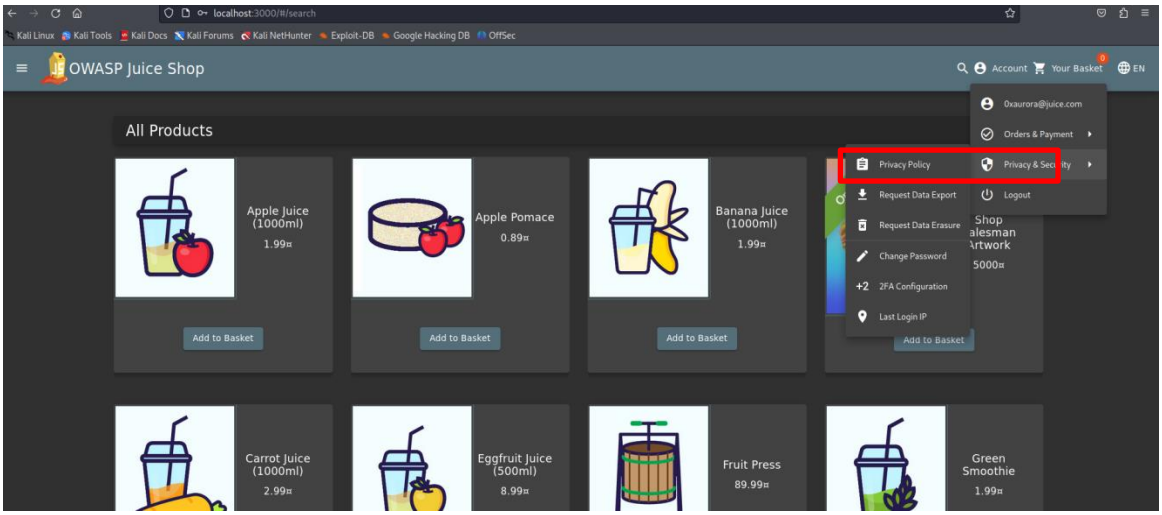
Proof of Concept:

1. Try to enter it in url but cant be reachable without login

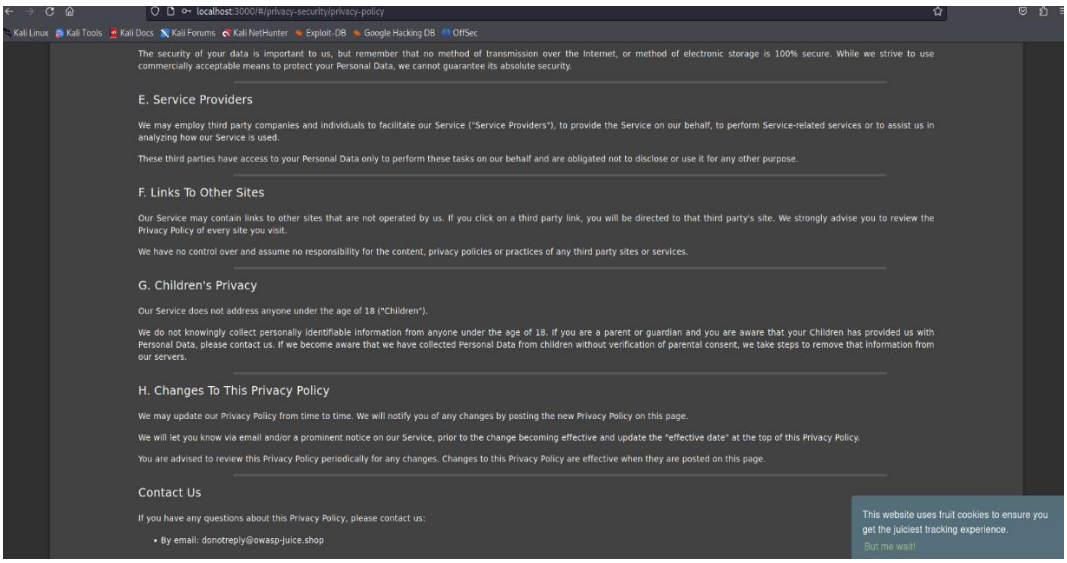


2. The pentester login as he had account on the website oredey





3. Final we enter to privacy policy



6.6 Cross-Site Scripting (XSS)

6.6.1 DOM-based XSS :

CRITICAL

Description:

The pentester found In the search field of products in the site, certain special characters like <, >, and ; are not filtered, which don't prevents to injection XSS payloads but JS tages such as <script> are filtered. However, despite these filters, the pentester was able to bypass the restriction by injecting an <iframe> tag. This results in the execution of malicious content via the src attribute of the iframe, which could allow an penetration tester to load a malicious page or steal sensitive information.

Impact:

The penetration tester was able to execute a XSS payload that uses an <iframe>. This could have the following impacts:

- **Loading Malicious Content**: The penetration tester can load external malicious websites or scripts inside the iframe.
- **Session Hijacking**: If the victim is logged into the website, an penetration tester could potentially steal session cookies or perform actions on behalf of the user using the iframe.
- **The pentester got special file (audio file) and run it**

Vulnerability Location:

The vulnerability is located in the search field

- **IP Address**: 127.0.0.1
- **Path** : l27.0.0.1/#/

CVE / OWASP Reference:

- OWASP - DOM-based Cross-Site Scripting (DOM XSS): https://owasp.org/www-community/attacks/DOM_Based_XSS

Recommendations :

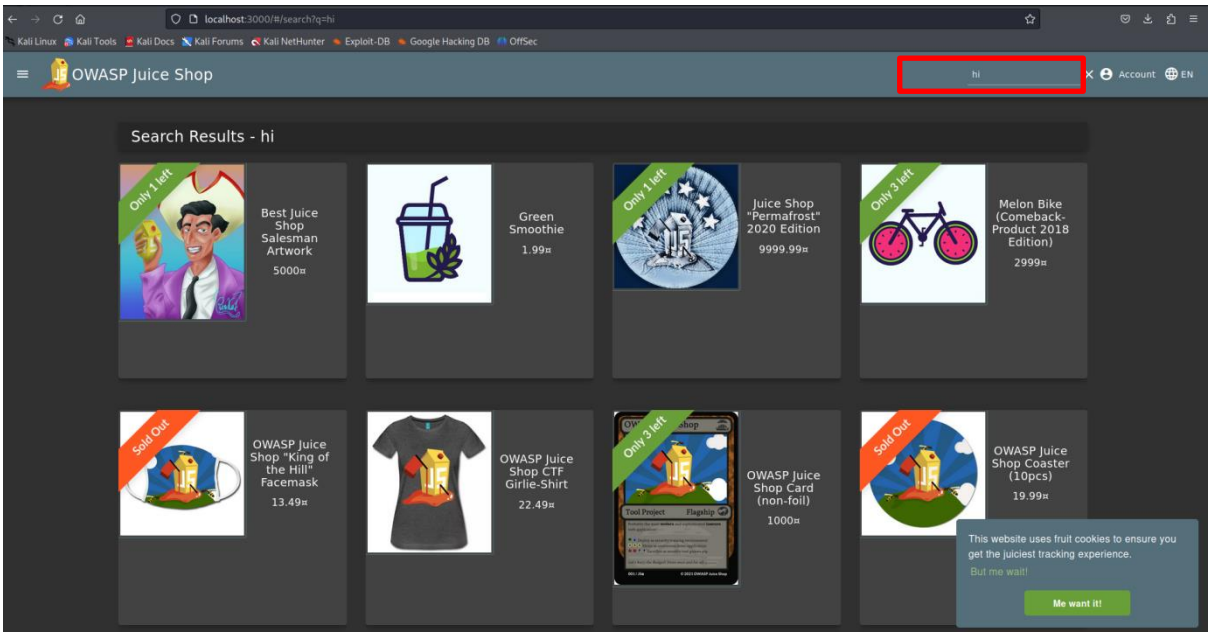
To mitigate this vulnerability, I recommend the following actions:

1. **Filter and Sanitize Input Thoroughly**:
   - Ensure that all special characters, including <iframe>, src, onerror, onload, and others, are properly sanitized.
2. **Restrict iframe Embedding**:
   - Use a **Content Security Policy (CSP)** that restricts the domains that can be loaded in an iframe. This will prevent penetration testers from loading malicious content from untrusted sources.
3. **Use JavaScript Event Handlers Safely**: Avoid injecting data directly into the HTML or DOM without proper escaping. Using methods like textContent or innerText (instead of innerHTML) to update content will prevent execution of scripts.

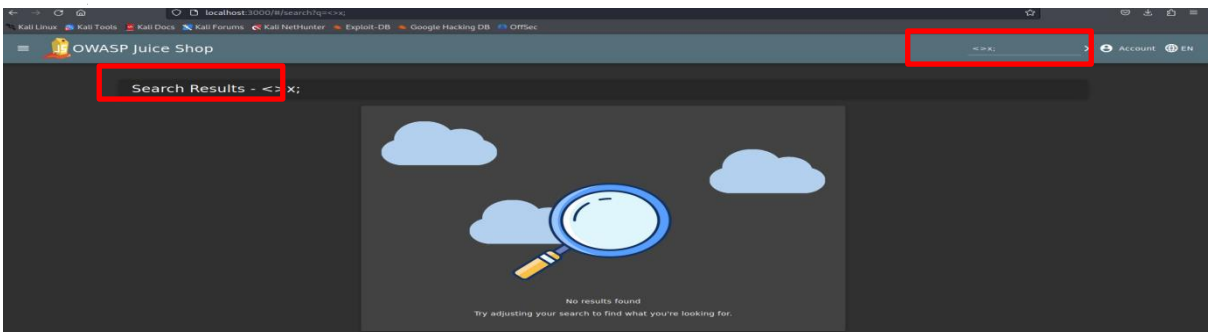
Proof of Concept:

Here's the detailed process to reproduce the issue:

1. Go to the site and locate the search field .

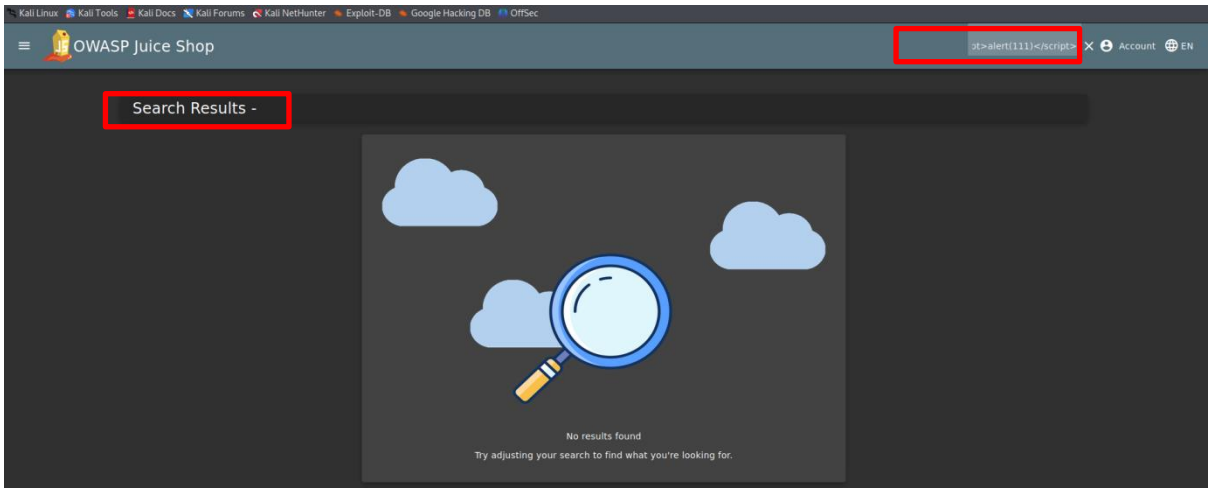


2. Try if the search can allow tages:

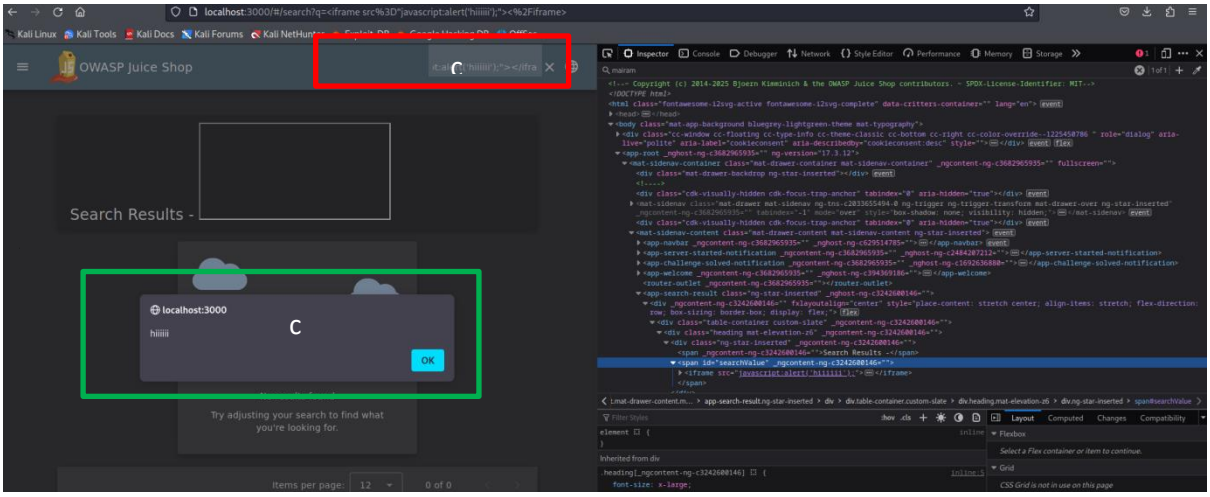


3. it allows the pentester to write tages

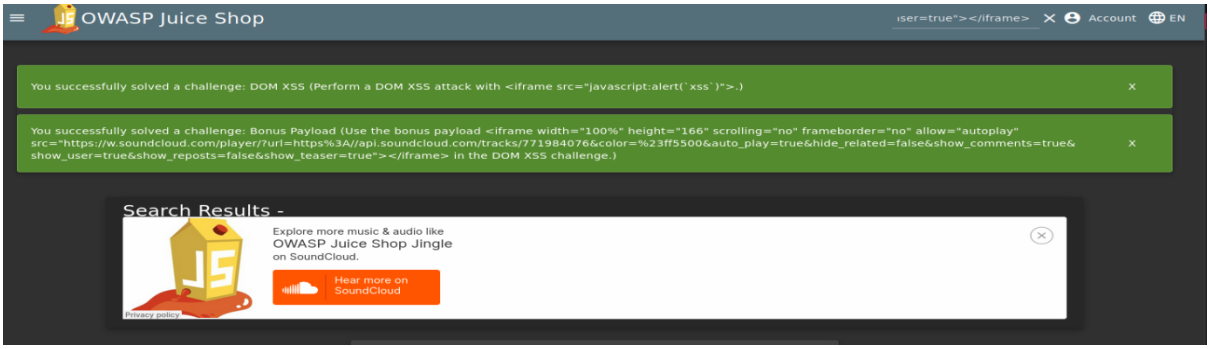
4. So he tried to inject xss payloads but there was a filter on JS codes



5. He tried to use payloads like `<iframe src="javascript:alert(`hiiii`)">` and guss what it's work! And bypass the js filter



6. Use the vuln to get and play private music file by special payload:
<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe>



6.6.2 Reflected XSS vulnerability:

CRITICAL

Description:

The pentester found a reflected XSS vulnerability which was identified in the order history section of the application in the track-result. The vulnerability exists in the id parameter of the order details page. The application fails to properly sanitize or validate user input in this parameter. As a result, the pentester can inject xss payload into the id parameter, which gets reflected back and executed in the target's browser.

To reproduce the vulnerability, the pentester first created an order as usual and then injected an XSS payload via the id parameter. The payload was reflected back in the response and executed within the browser, leading to the possibility of executing arbitrary JavaScript code on the victim's device.

Impact:

The impact of this vulnerability could be severe, as it allows an penetration tester to execute arbitrary JavaScript code on a victim's browser. This could lead to:

- **Session Hijacking**: An penetration tester can steal session cookies or perform actions on behalf of the victim.
- **Phishing Attacks**: The penetration tester could inject a fake login form to steal user credentials.
- **Malicious Redirects**: The penetration tester could redirect users to phishing sites or malicious URLs.

Vulnerability Location:

The vulnerability resides in the order history section at trace-resulte, specifically in the id parameter of the order details page

- **Path to Vulnerability**: 127.0.0.1/#/track-result?id=

CVE / OWASP Reference:

- OWASP - Reflected Cross-Site Scripting :https://owasp.org/www-community/attacks/xss/#reflected-xss-attacks

Recommendations:

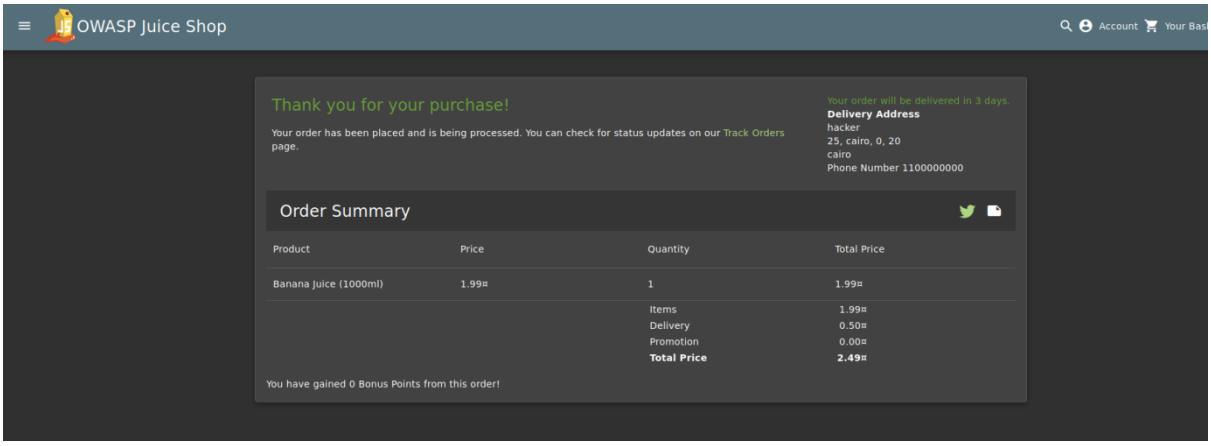
To mitigate this vulnerability, the following steps should be implemented:

1. **Sanitize and Escape Input**: Ensure that all user inputs, including URL parameters, are sanitized and validated to remove any potentially dangerous characters (e.g., <, >, ;, etc.).
2. **Output Encoding**: Apply proper output encoding when displaying user input on the page to ensure that it is treated as data, not executable code.
3. **Implement a Content Security Policy (CSP)**: Deploy a strict CSP to limit the sources from which JavaScript can be executed, reducing the impact of any potential XSS vulnerability.

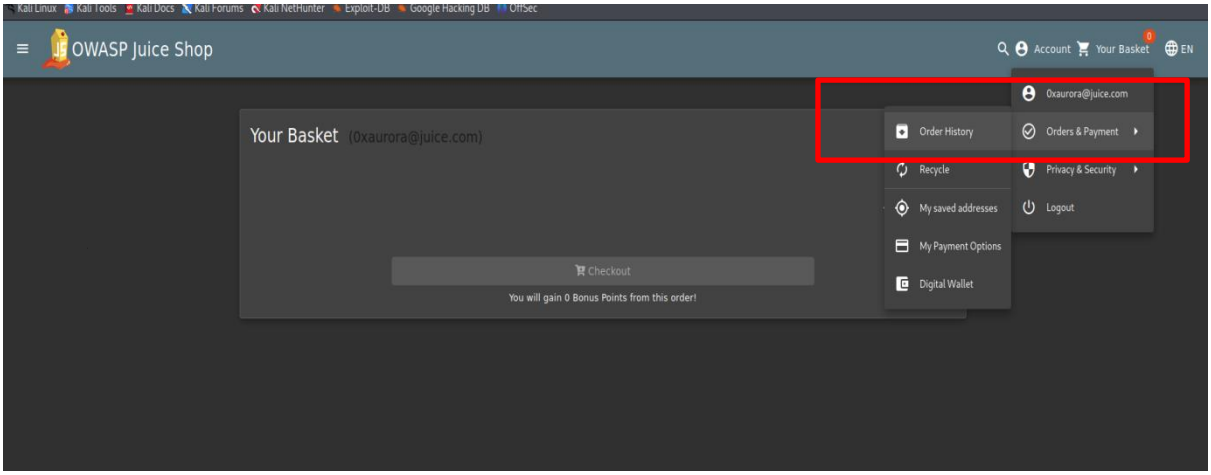
Proof of Concept:

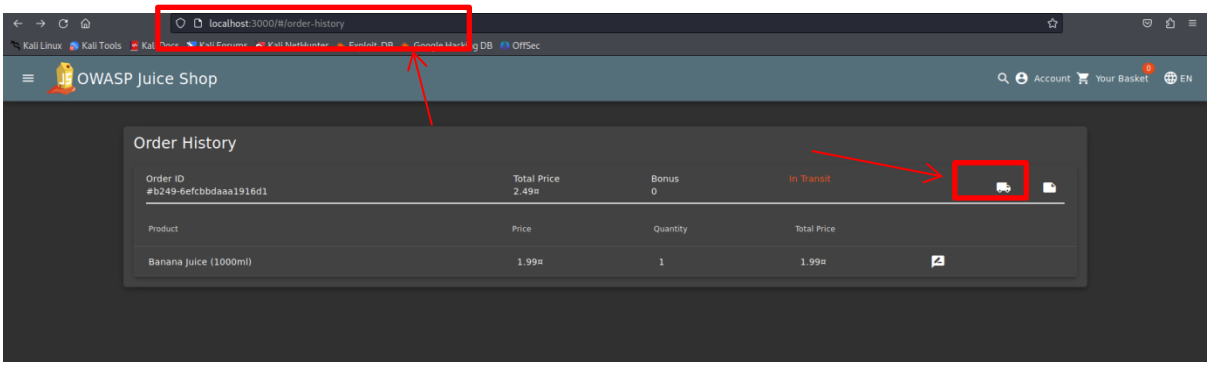
Here is the detailed process to reproduce the issue:

1. First, create an order within the application as usual.

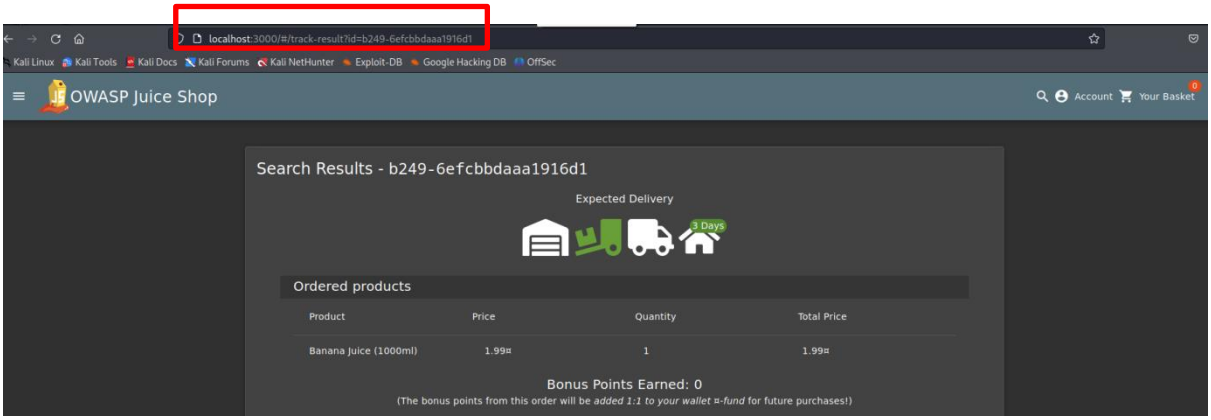


2. Then went to order history

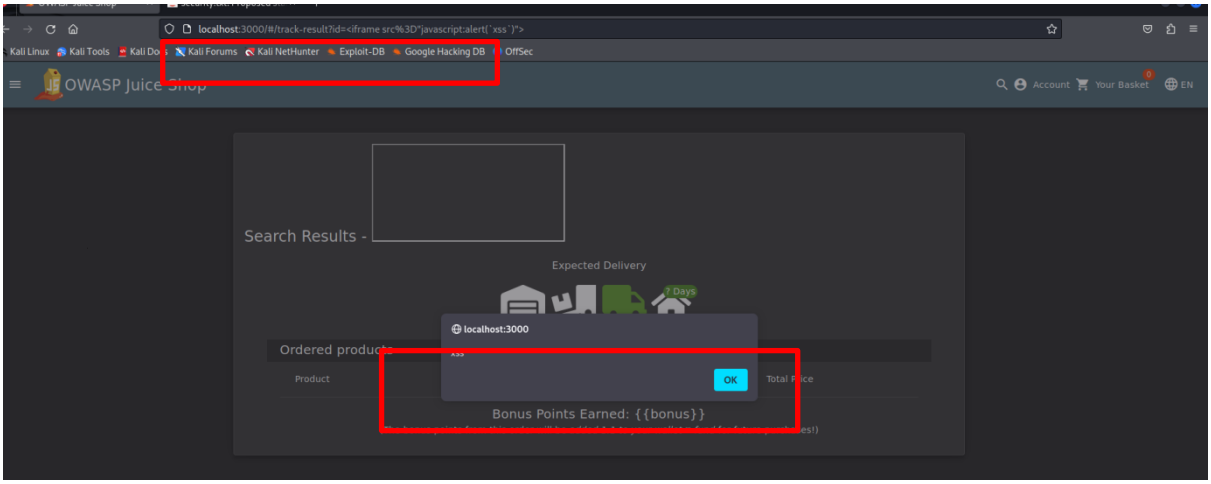




3. After that he want to trace-result



4. Notice that is an id parameter so he try to xss inject: <iframe src="javascript:alert(`xss`) "></iframe>



5. The malicious iframe has been reflected in the page and executed, demonstrating the vulnerability.

6.6.3 API-Reflectd-XSS

CRITICAL

Description:
The pentester found During API testing, a **Reflected Cross-Site Scripting (XSS)** vulnerability was discovered in the **Products API**. By intercepting and modifying the API request using **Burp Suite**, it was possible to inject malicious JavaScript code into the **description** field of a product's JSON data. When the manipulated product was later viewed on the website, the XSS payload was rendered and executed in the browser context — confirming the vulnerability.

This indicates that the application fails to properly sanitize or encode user-supplied input from API sources before rendering it in the UI.

Impact:
A successful reflected XSS attack can:

- Execute arbitrary JavaScript in the victim's browser.
- Steal session cookies or local storage tokens.
- Perform actions on behalf of the user (e.g., CSRF, phishing).
- Redirect users to malicious sites.

If exploited by an penetration tester with access to the product API, this could lead to **compromise of user accounts or data theft** when users view the infected product.

Vulnerability Location:

- **Endpoint:** /api/Products
- **Affected Parameter:** in the product JSON body

CVE Reference:
While there's no CVE for this specific case, this type of vulnerability is covered under:

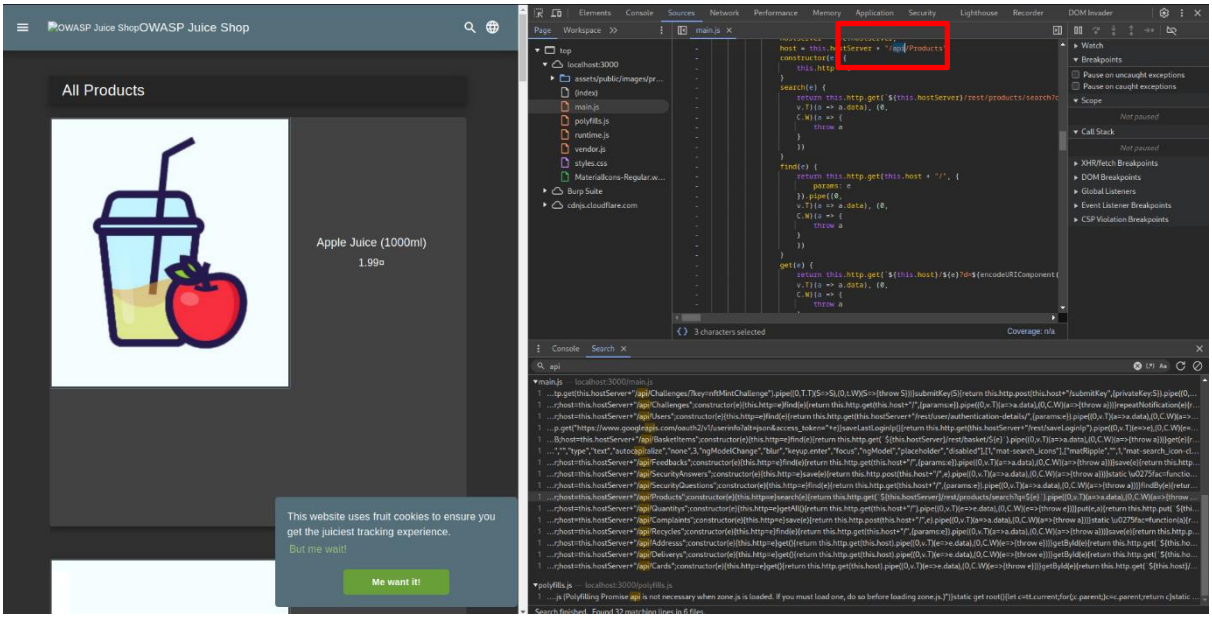
- **OWASP A07:2021 – Cross-Site Scripting (XSS)**
- Related OWASP info: https://owasp.org/www-community/attacks/xss/

Recommendations:

1. **Sanitize Input on Server-Side:**
   Apply proper input validation and output encoding before storing or displaying user-generated content, especially HTML-sensitive characters (<, >, ", ', etc.).
2. **Use Context-Aware Output Encoding:**
   Ensure that any content rendered in HTML, JS, or CSS contexts is appropriately encoded for its context (e.g., use HTML entity encoding in descriptions).
3. **Implement Content Security Policy (CSP):**
   Use a strong CSP header to reduce the impact of potential XSS attacks.

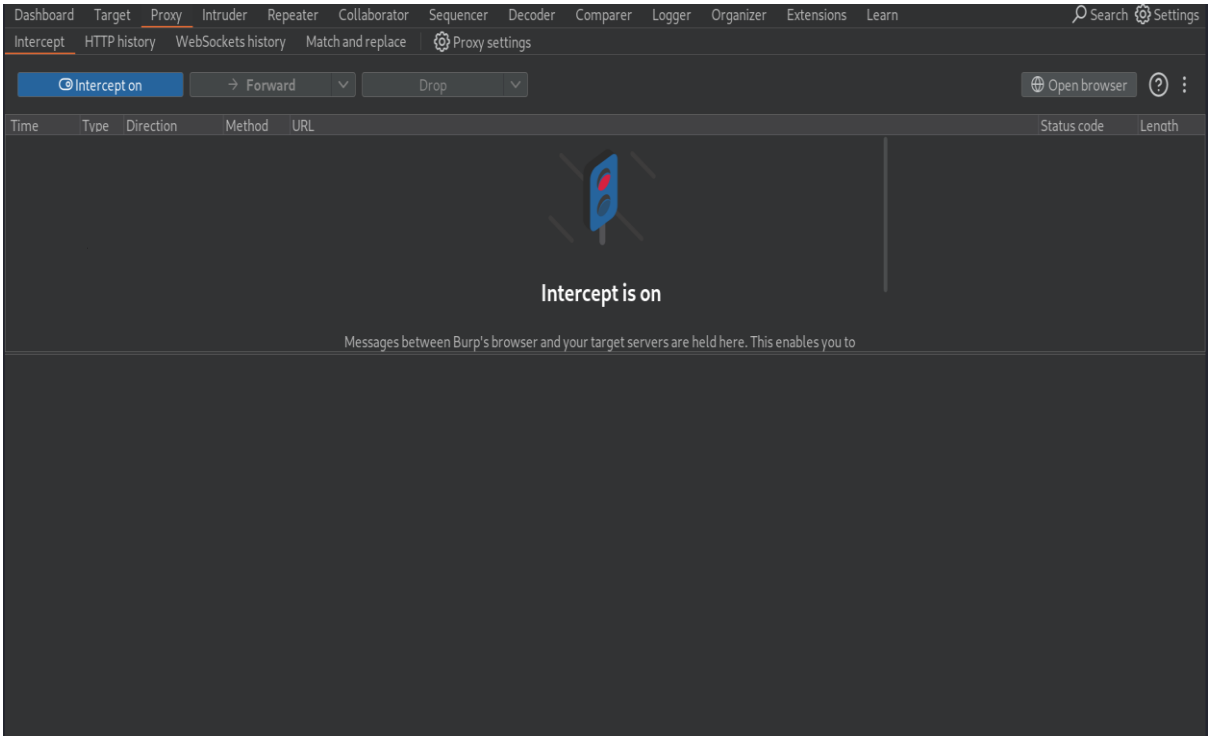
Proof of Concept:

1. Open inspect to search for APIs

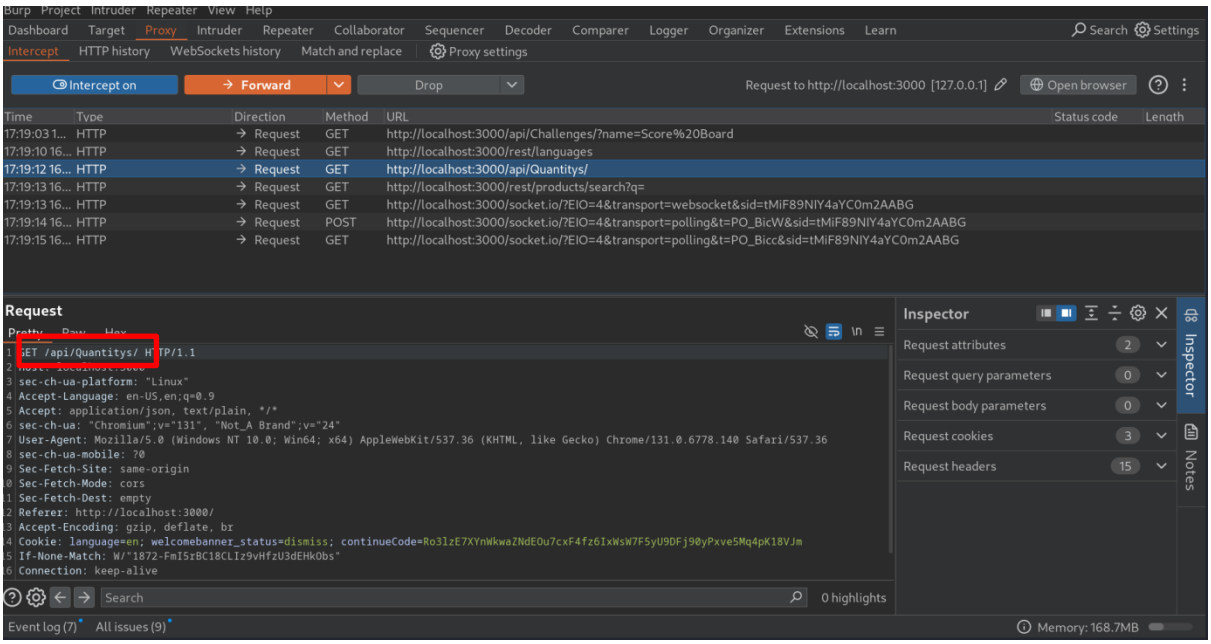


2. Found api/Products and the pentester thought that is reflected in products page so it can be injected by xss

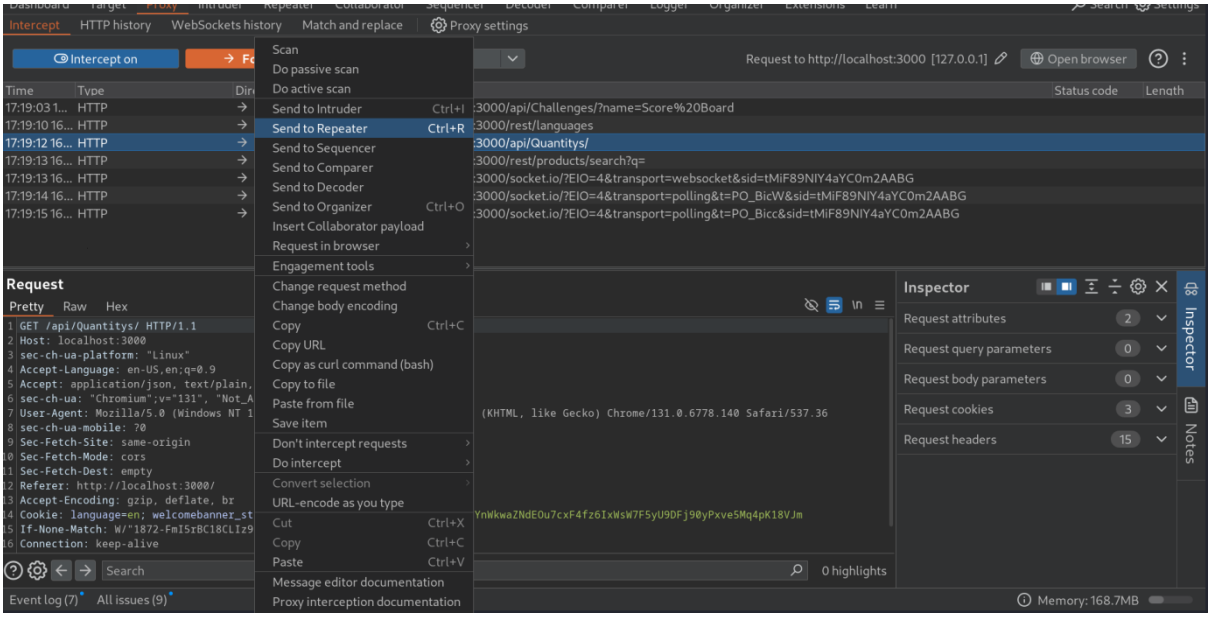
3. Start to refresh the products page and intercept the request by burp suit



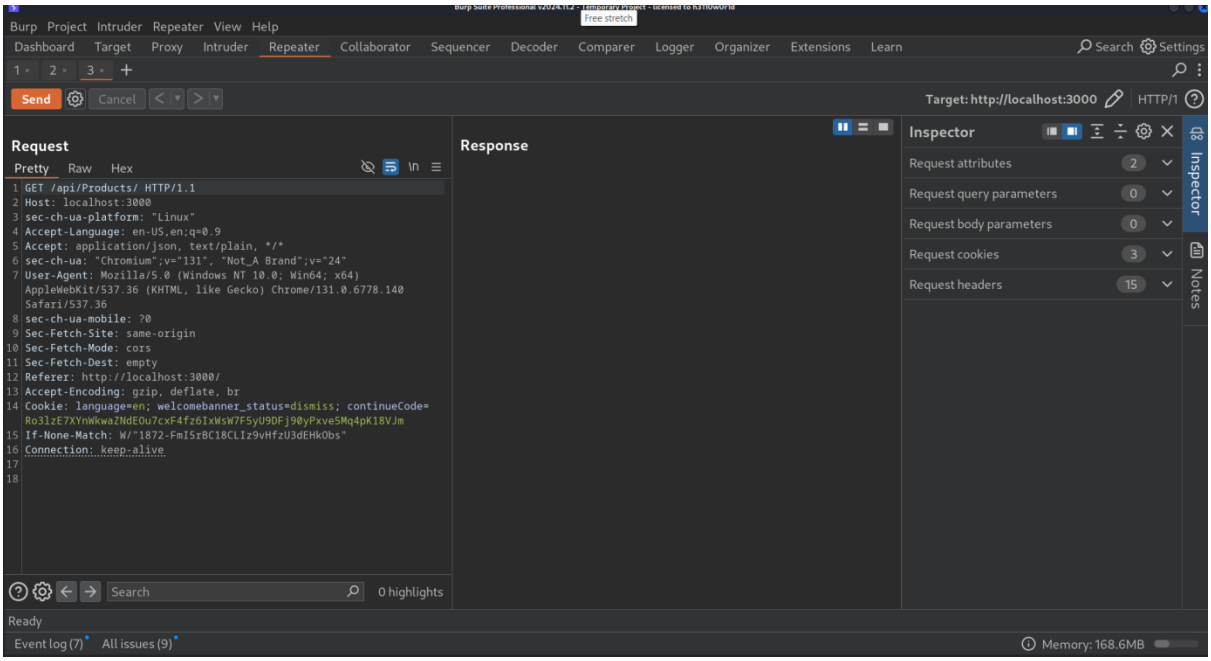
4. Start to check all requests to find api path



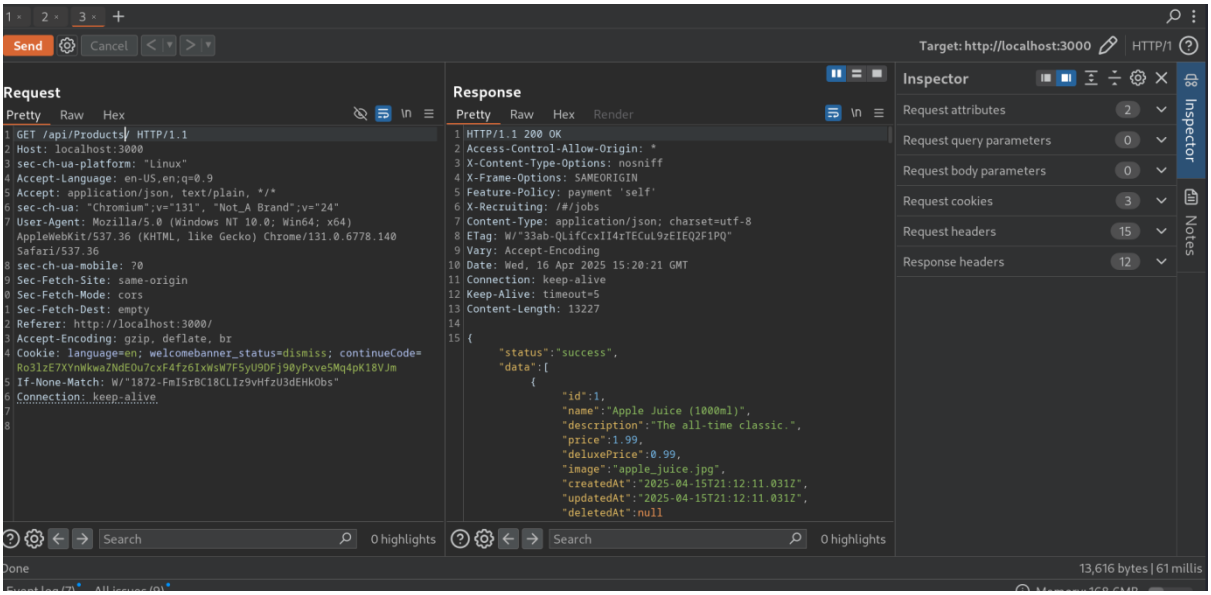
5. Send the request to repeater tool in burp suit



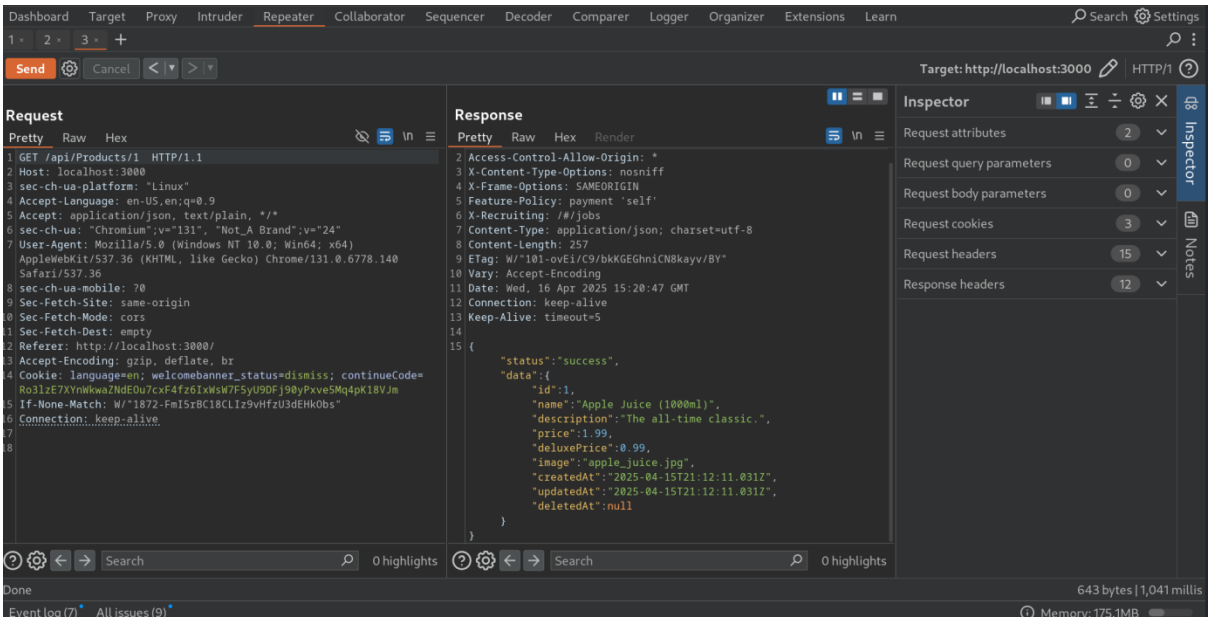
6. Open **Repeater and start to edit in the request and change the path from api/Quentitys to api/Products**



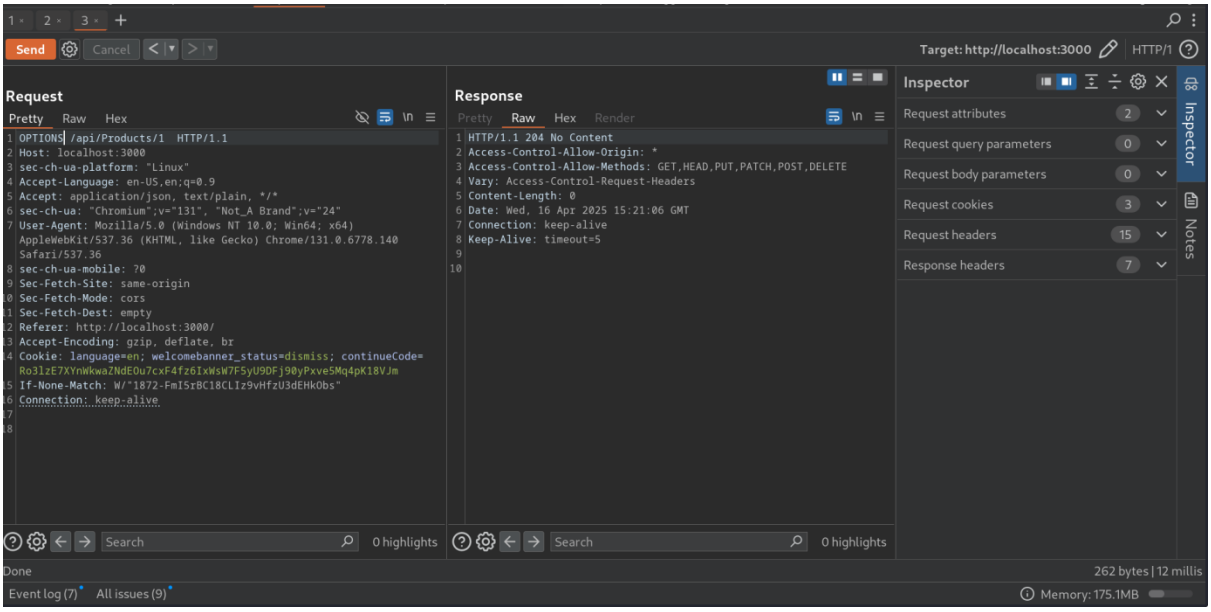
7. By click on send button the result was json file contain all products data which appear in login page



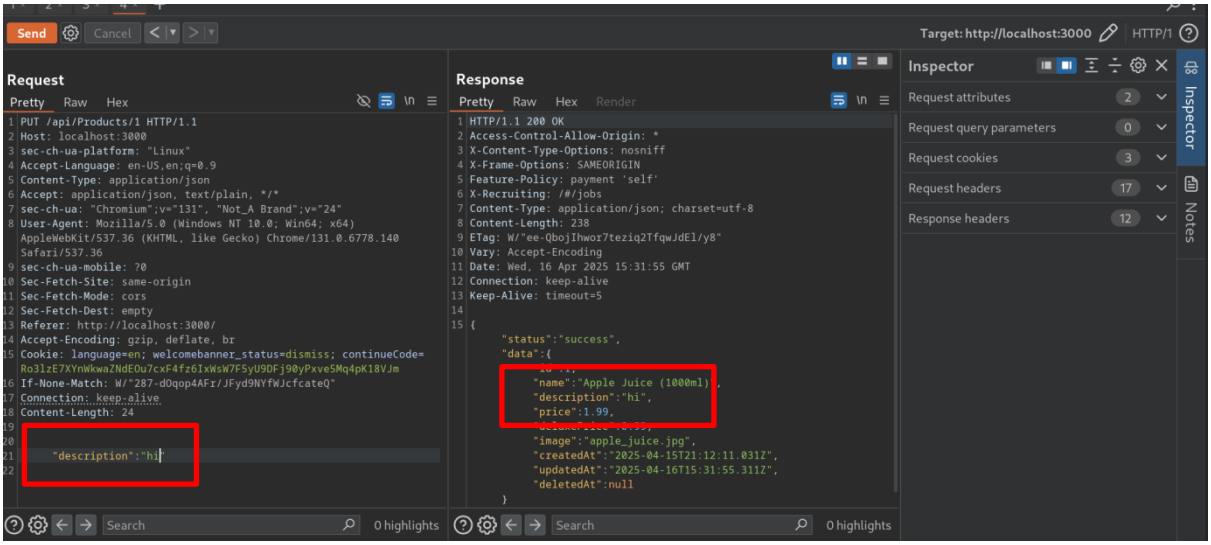
8. Now we get only one product by add id number after the Products/ and the result was apple juice product



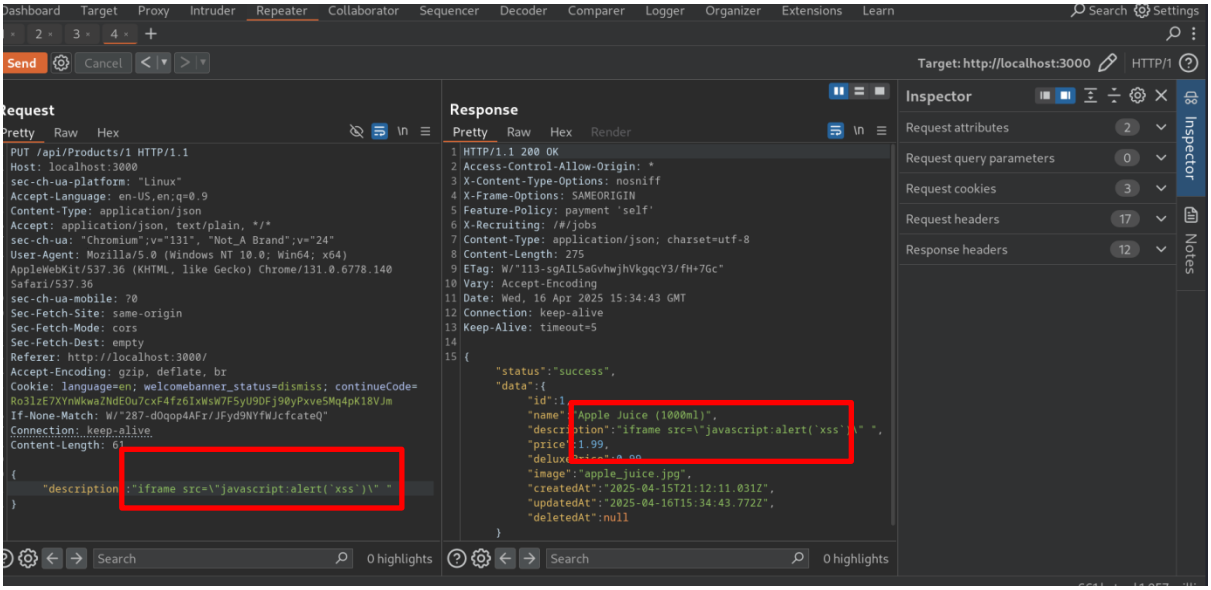
9. Now check if the requsit allow to use mehtods like put and post to edit in the data of the products by using OPTIONS mehtod



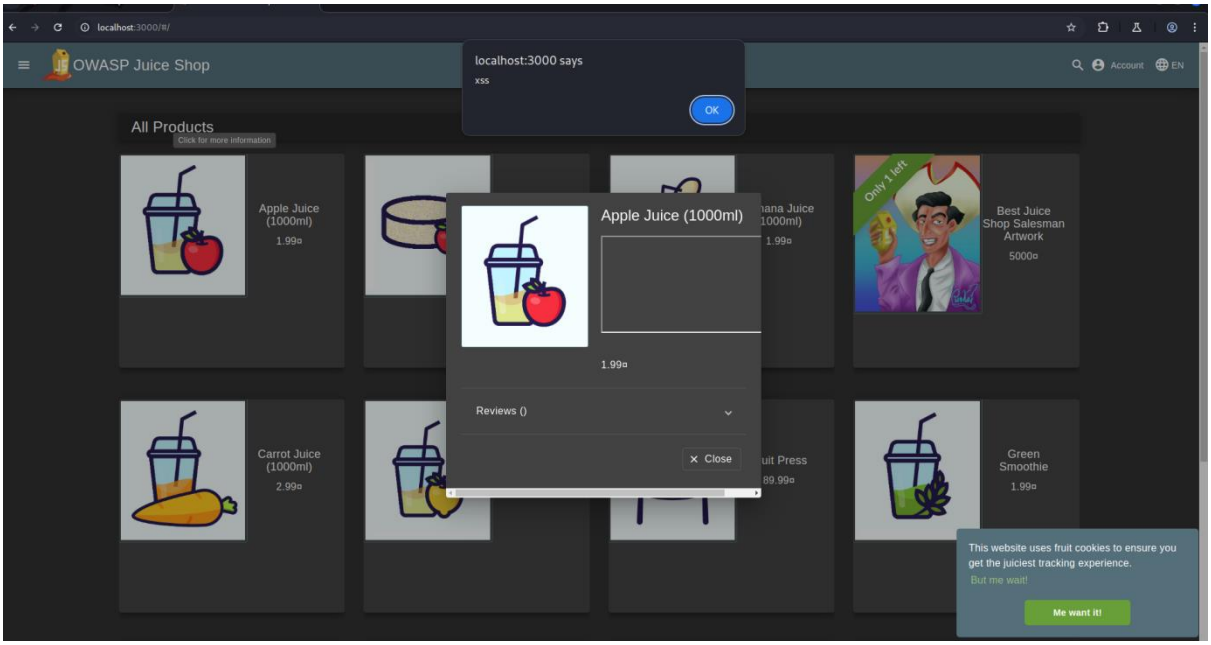
10. Yaa the PUT method is allowed so let try to change the values of the data of the products let use description felid



11. It changed!! So lets try to inject XSS payload like `<iframe src=/"javascript:alert(`xss`)/">`



12.Lets go to the products page and show the change



13.Here we go! The payload successfully work

6.6.4 Client-side XSS Protection

HIGH

Description:

The pentester found a **Client-side XSS Protection** mechanism was implemented on the registration page to prevent the injection of malicious scripts. However, during testing, this protection was bypassed by intercepting the registration request using **Burp Suite** and manually injecting a malicious payload into the **email field**.

Although the client-side form validation restricted such inputs in the browser, the server **failed to validate or sanitize** the request after interception, leading to successful account creation using an XSS payload. The payload was later reflected and executed in areas where the email was displayed — confirming a **Reflected Cross-Site Scripting (XSS)** vulnerability.

Impact:

- **Execution of Arbitrary JavaScript Code**
- **Session Hijacking or Account Impersonation**
- **Possible Admin Panel Compromise if Emails Are Rendered There**

This bypass of client-side protection renders it ineffective and exposes users and the system to XSS-based attacks.

Vulnerability Location:

- **Endpoint:127.0.0.1/#** /register
- **Affected Parameter:** email

CVE / OWASP Reference:

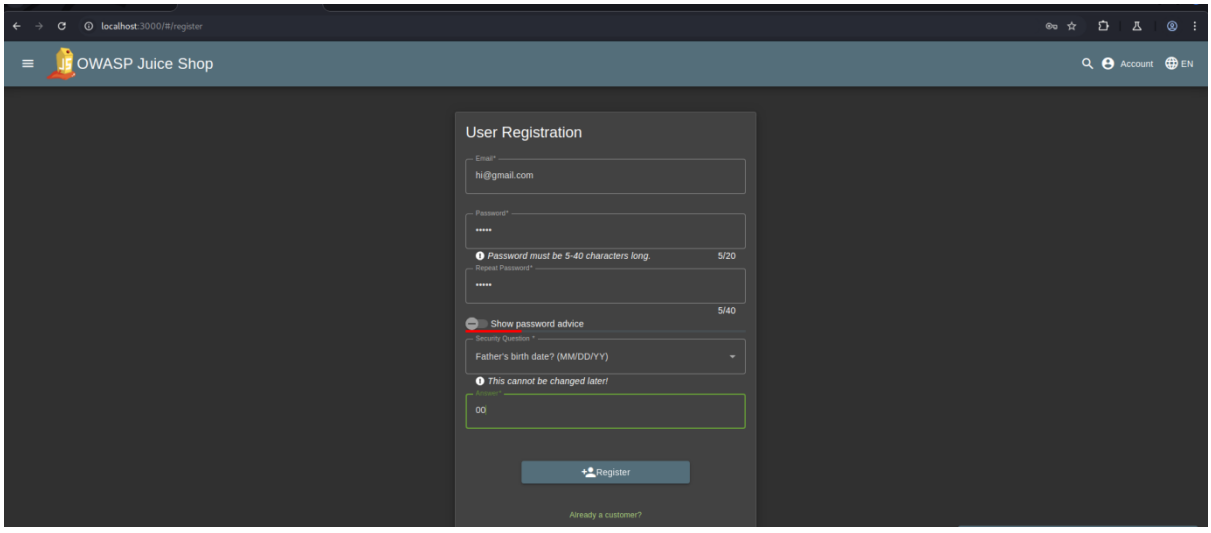
1. **OWASP A07:2021 – Cross-Site Scripting (XSS)**
   https://owasp.org/www-community/attacks/xss/

Recommendations:

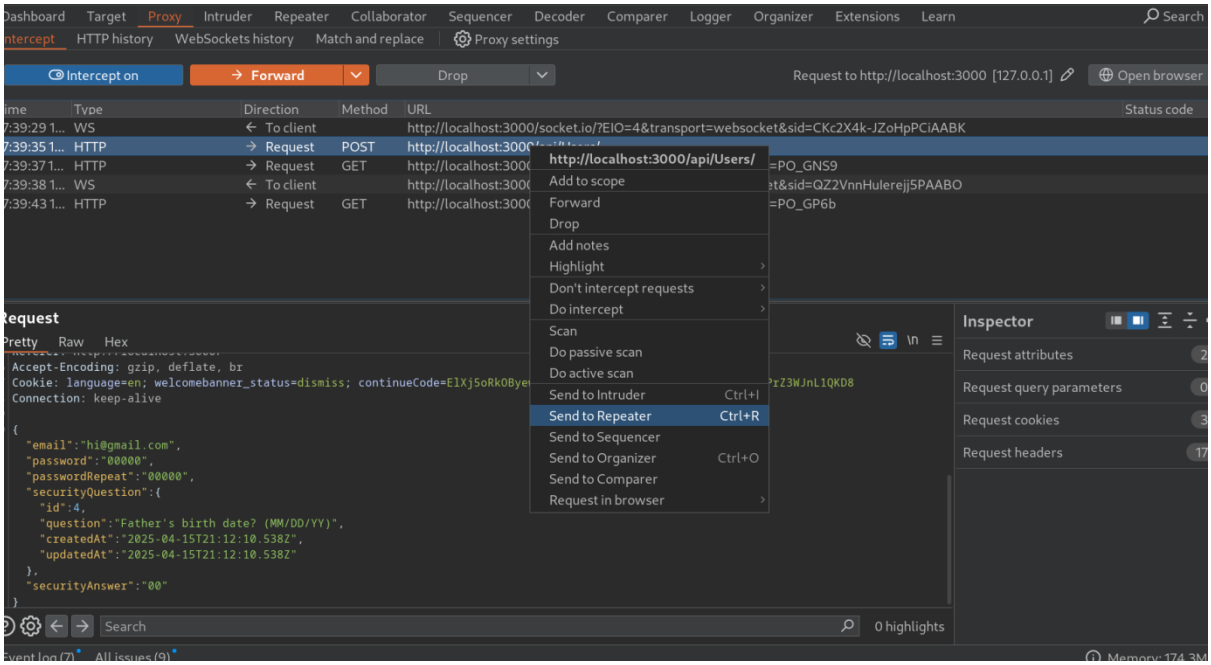
1. **Server-Side Validation:**
   Never rely solely on client-side protections. Ensure proper validation and sanitization on the server for all input fields, especially user identifiers like emails.
2. **Encode Output:**
   Escape all user input before rendering it in the frontend to prevent script execution.
3. **Use Email Regex:**
   Enforce strict email format validation (^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$).
4. **Apply Content Security Policy (CSP):**
   Limit the sources from which scripts can be executed using a well-configured CSP header.

Proof of Concept:

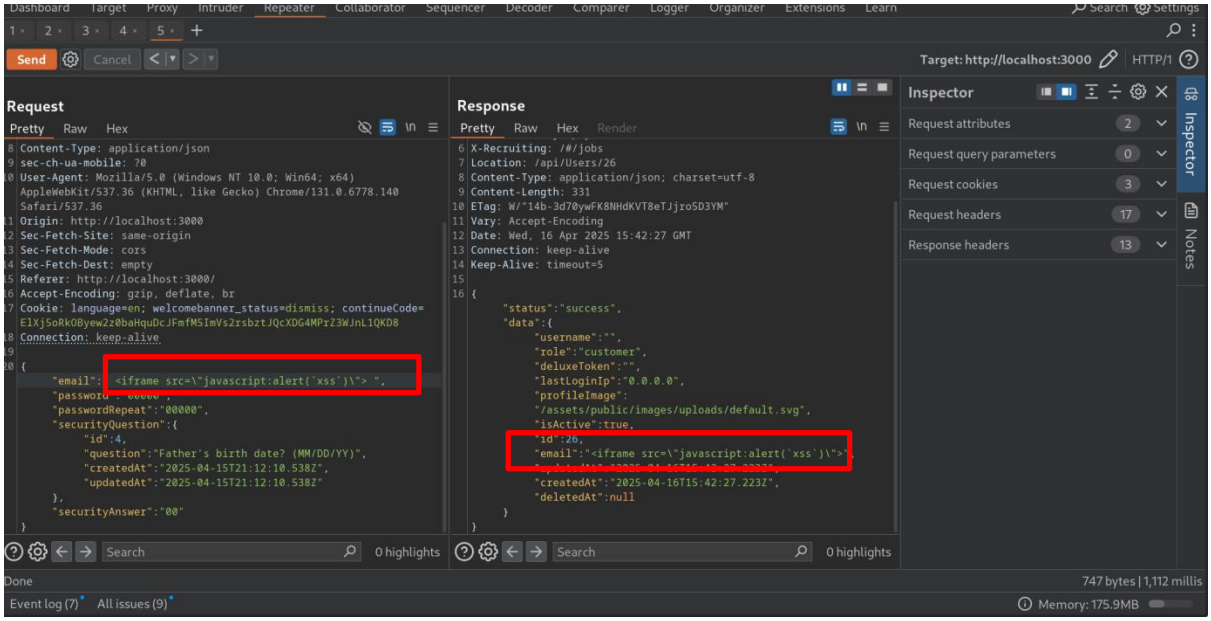
1. Navigate to the registration page and begin creating a new account.



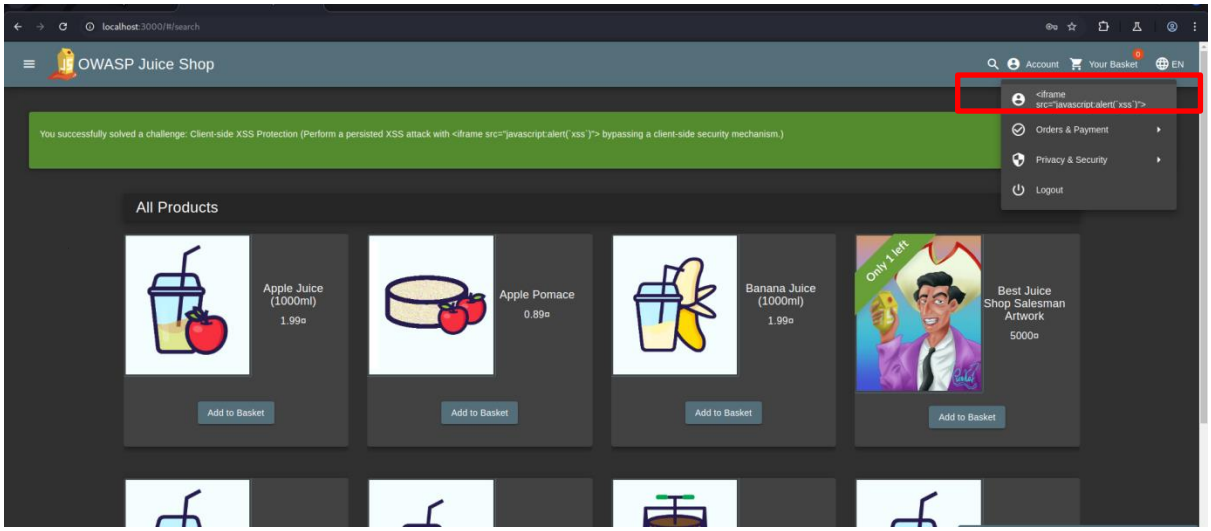
2. Intercept the request using **Burp Suite and send it to Repeater tool**.



3. Modify the email field to contain the following payload and send the request:



4. The server accepts it and creates the account.
5. Go to login page and login by payload in email section and password here we go the account is exist and we loged in


localhost:3000/#/search
OWASP Juice Shop
Account
Your Basket
EN
<iframe src="javascript:alert(`xss`)">
Orders & Payment
Privacy & Security
Logout
You successfully solved a challenge: Client-side XSS Protection (Perform a persisted XSS attack with <iframe src="javascript:alert(`xss`)"> bypassing a client-side security mechanism.)
All Products
Apple Juice (1000ml)
1.99¤
Add to Basket
Apple Pomace
0.89¤
Add to Basket
Banana Juice (1000ml)
1.99¤
Add to Basket
Only 1 left
Best Juice Shop Salesman Artwork
5000¤
Add to Basket

6.7 Cryptographic Issues

6.7.1 Weird Crypto

HIGH

Description:
During a source code review, the **penetration tester** inspected the system's backend files and discovered the use of the outdated and insecure hashing algorithm **MD5**.

MD5 is no longer considered secure due to known vulnerabilities such as collision attacks and ease of brute-forcing, especially with modern computing power.

Impact:

- Hash Collision Risk: Malicious users could exploit MD5's collision weakness to generate identical hashes for different inputs.
- Offline Password Cracking: If used for password hashing, penetration testers can easily reverse hashes using precomputed rainbow tables or brute-force tools.

Vulnerability Location: https://github.com/juice-shop/juice-shop/blob/master/lib/insecurity.ts

---

CVE / OWASP Reference:

-OWASP Top 10 – A07:2021 – Identification and Authentication Failures
https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

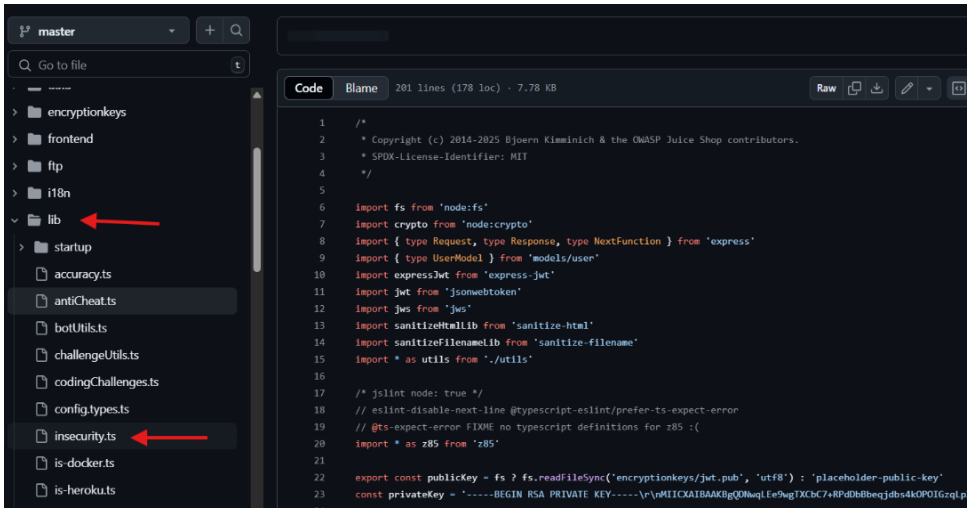
Recommendation:

1. Replace MD5 with a Strong Cryptographic Hashing Algorithm.

Proof of Concept:

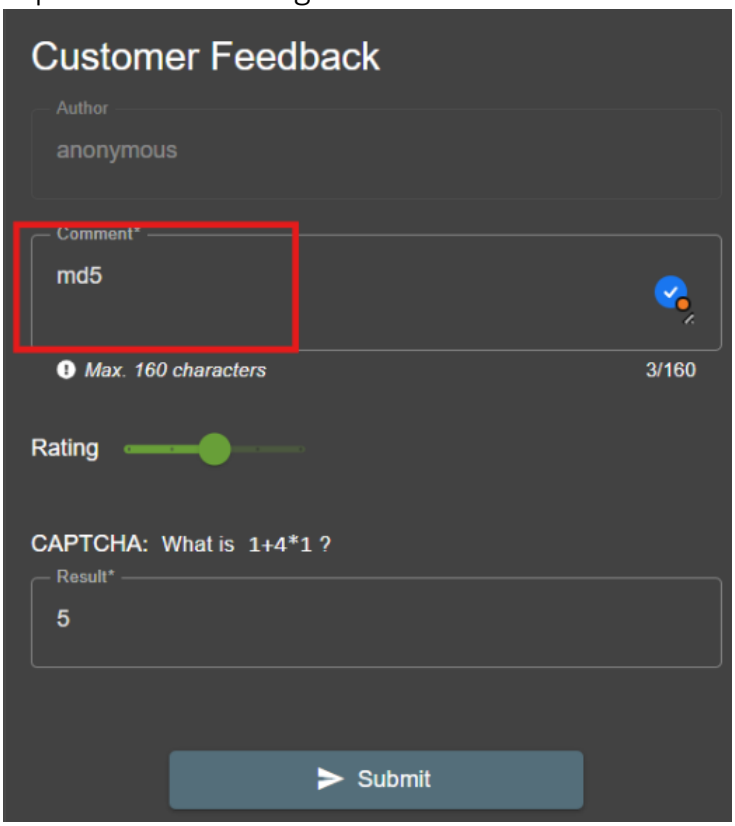
1. The tester navigated to the official GitHub repository of the application https://github.com/juice-shop/juice-shop

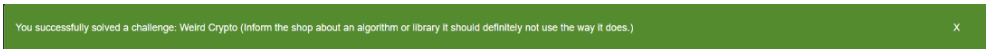
2. While reviewing the codebase, the tester examined the file https://github.com/juice-shop/juice-shop/blob/master/lib/insecurity.ts



3. In this file, the tester discovered that the system was using the **MD5 hashing algorithm**, which is known to be weak and **cryptographically broken**

4. After confirming the presence of MD5 usage, the tester reported this finding via the customer feedback





6.7 Improper Input Validation

6.7.1 Admin Registration

CRITICAL

Description:

This vulnerability allows a user to escalate their privileges during the registration process by manipulating the HTTP request. By intercepting and modifying the registration request to include a role parameter with the value admin, an penetration tester can register as an admin user and gain access to the restricted administration panel.

Impact:

A penetration tester was able to escalate privileges by registering as an admin user, giving full access to sensitive administration features. This could lead to total compromise of the application, including viewing user data, modifying site configurations, or deleting critical resources.

Vulnerability Location:

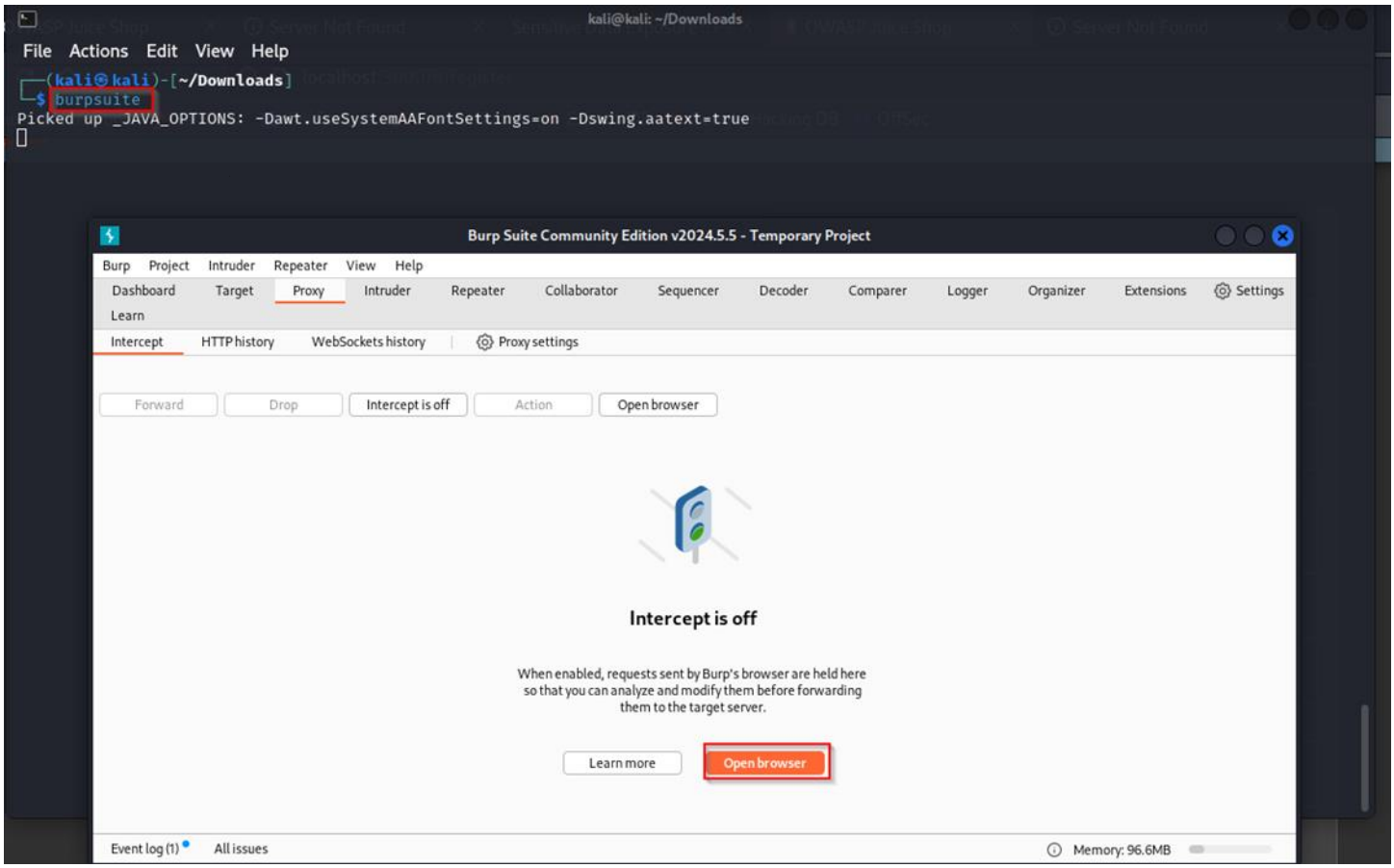
- Registration Endpoint: :/#/register
- Parameter Affected: role
- Admin Panel URL: /#/administration

Recommendations:

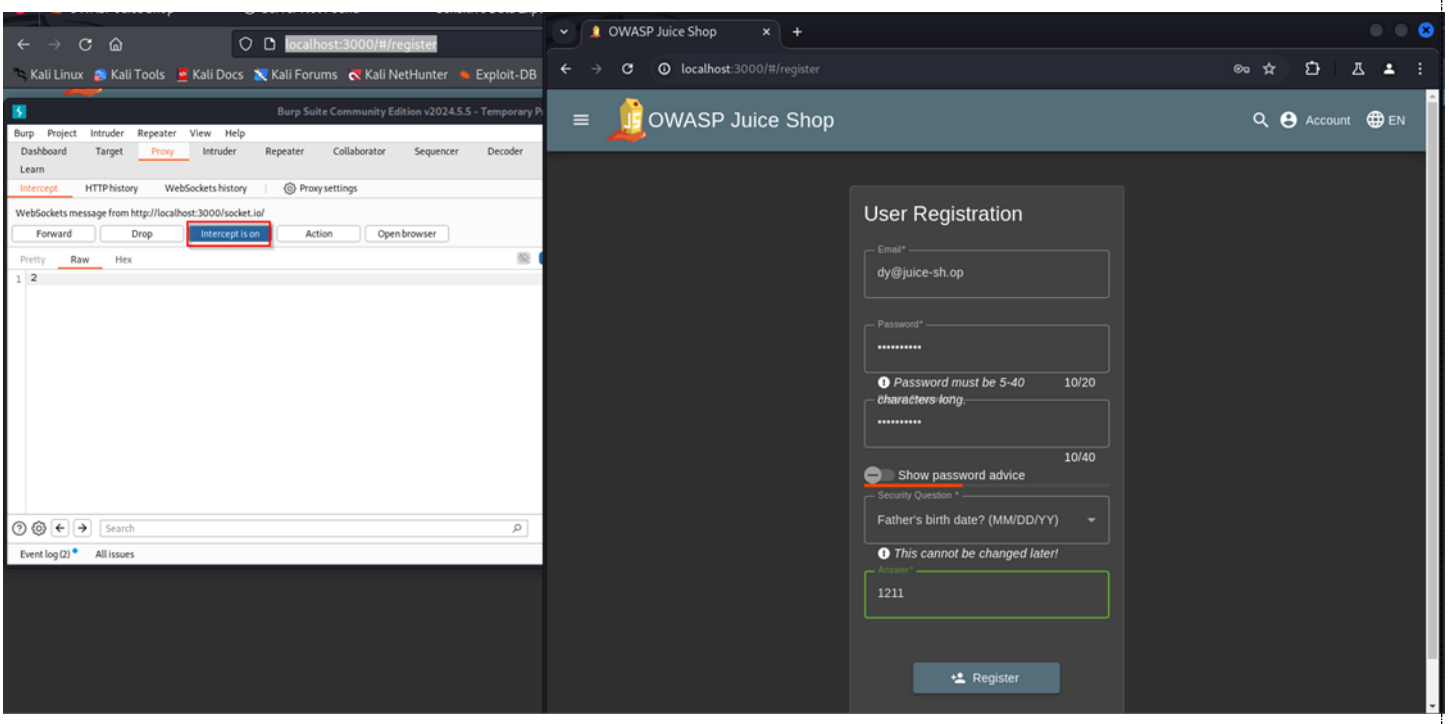
- The server should strictly validate all incoming input and ignore or reject any unauthorized fields such as role during registration.
- Access control checks should be enforced on server-side based on session and user context, not on client-submitted values.
- Use allowlists to restrict acceptable parameters for each endpoint.
- Implement logging and alerting unexpected or suspicious input (role submission during registration).

Proof Of Concept:

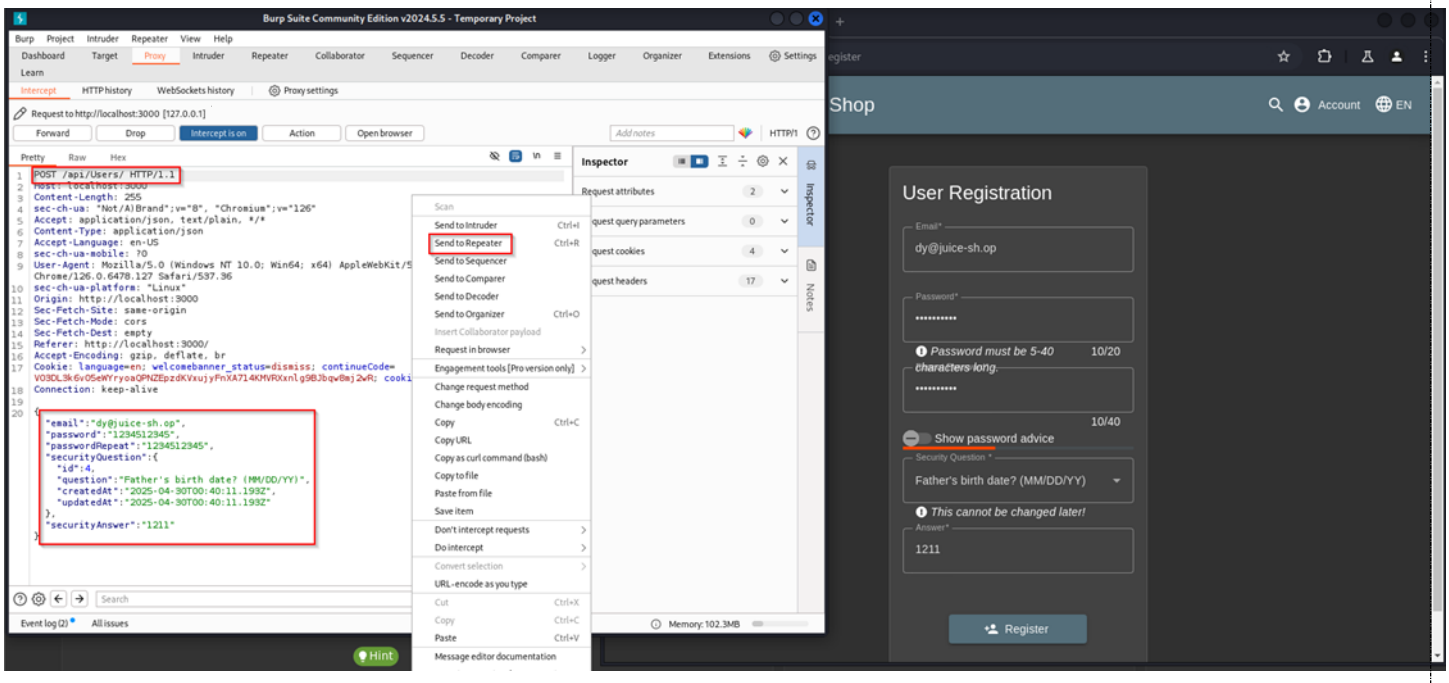
1. Open Burp Suite and launch the browser.



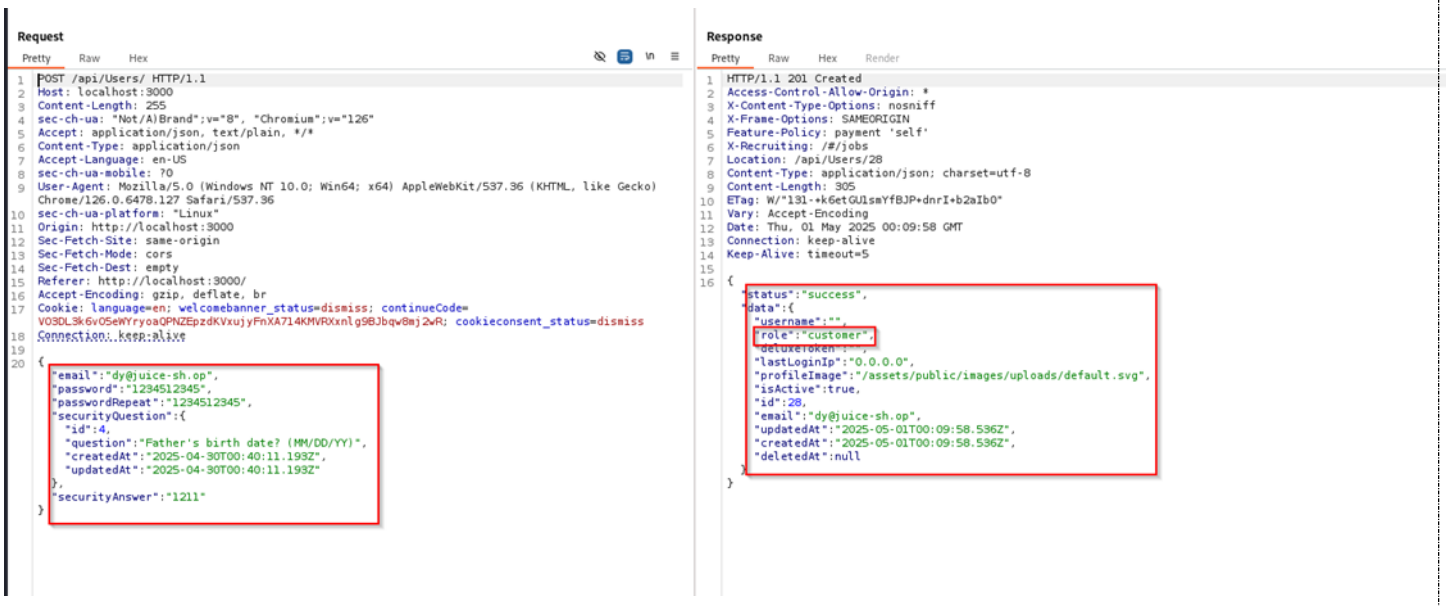
2. Navigate to the registration page, fill out the form and enable Intercept to capture the registration request.



3. Send the request to the Repeater tab in Burp Suite.

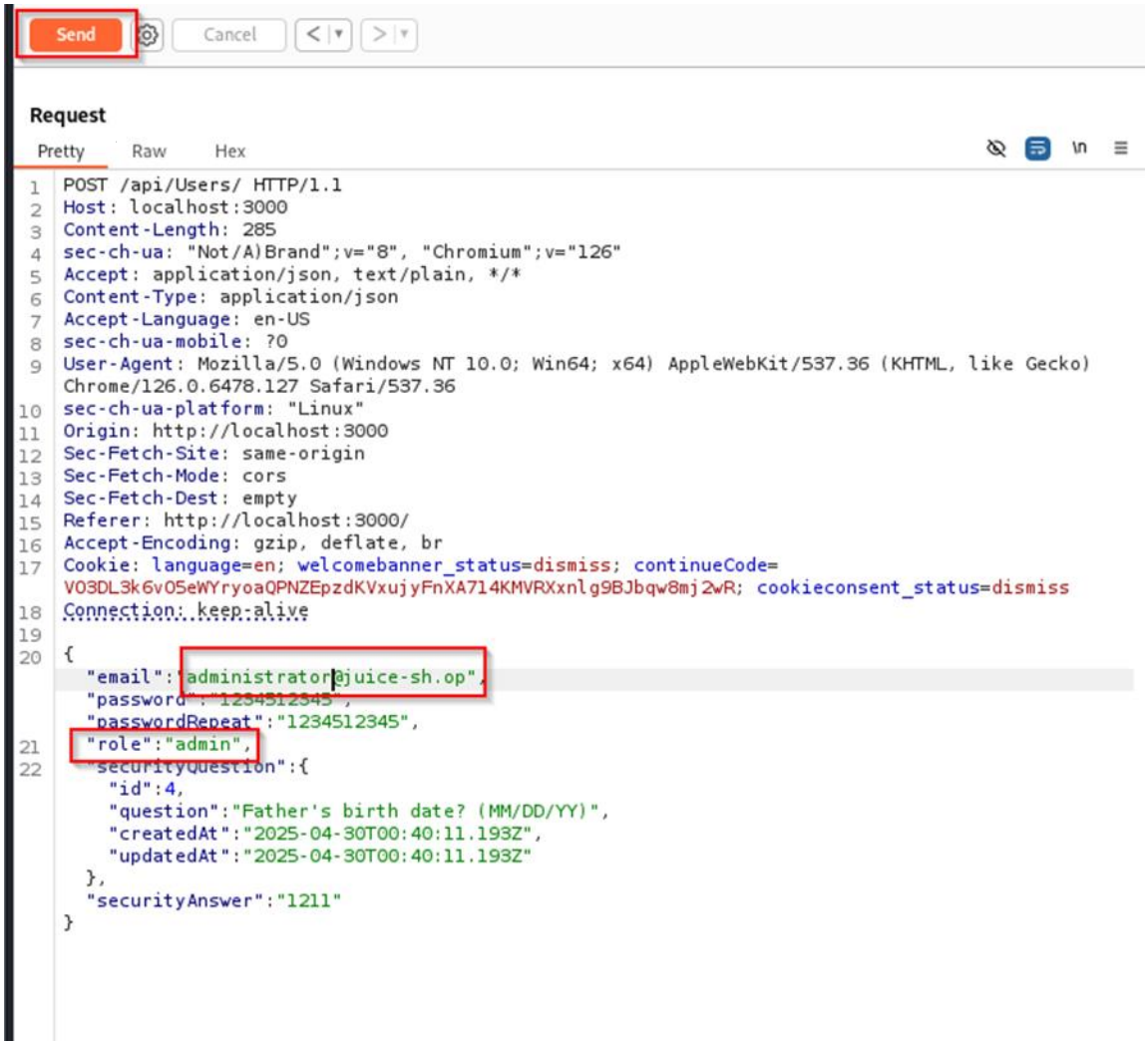


4. Observe the request and response.

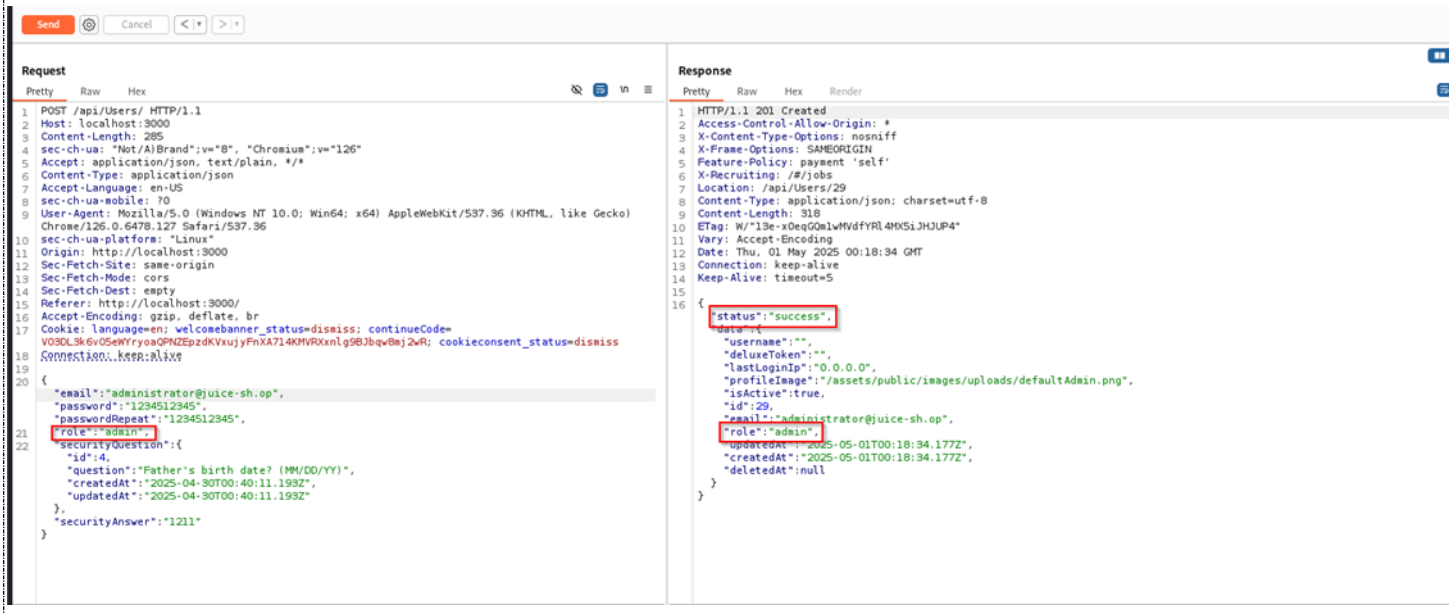


There is an interesting value in the response: "role" Response shows "role": "customer" indicating a normal user.

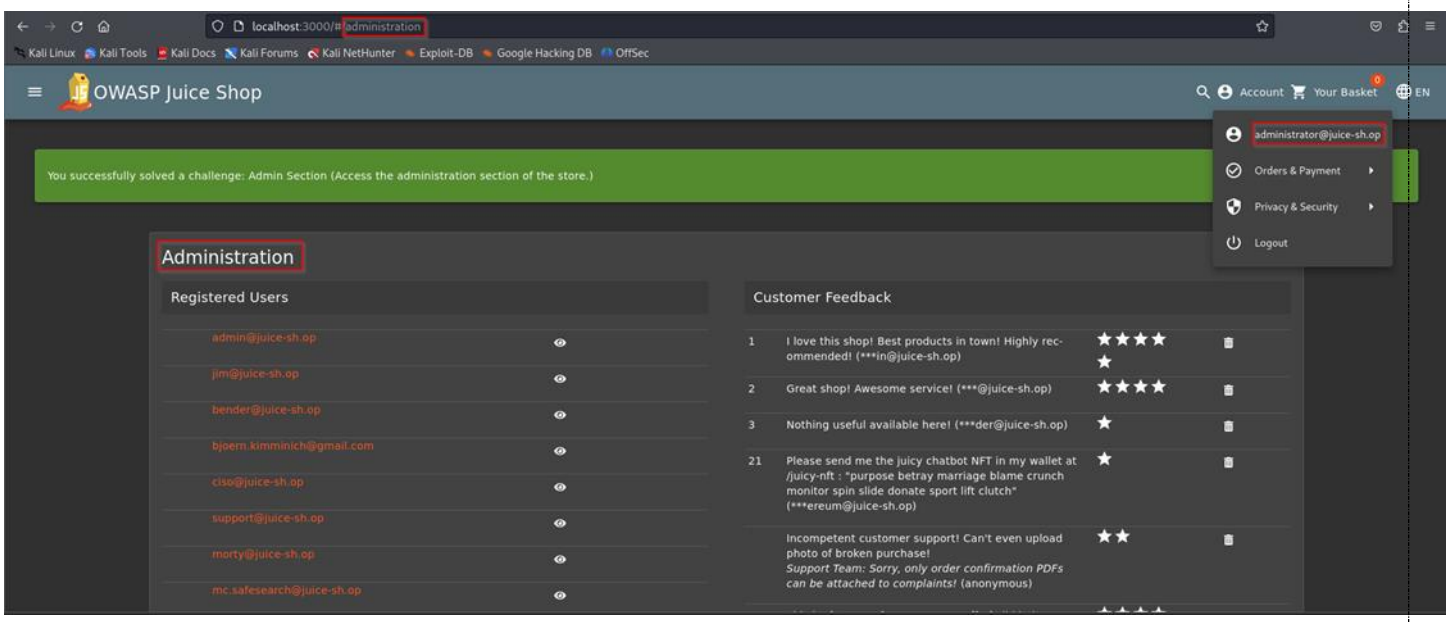
5. In the request body, manually add: "role": "admin" and change email to avoid a unique email error click send



6. Response returns: “HTTP/1.1 201 Created”, confirming successful registration.



7. Log in with the newly registered account and Access the Administration Panel -shown previously in another vulnerability found- which is only visible to admins.



The Administration panel is now accessible, confirming the account has admin privileges.

6.7.2 Allowlist Bypass

HIGH

Description:

The application incorrectly handles redirect validation by trusting user-supplied URLs without sufficient checks.
An penetration tester can bypass the intended allowlist validation by crafting a malicious URL using techniques such as double slashes (//) and URL parameters, leading users to unauthorized external sites.

Impact:

- **Phishing Attacks:** penetration testers can redirect users to fake login pages or malicious sites.
- **Loss of Trust:** Users may believe the redirection is safe because it originates from a trusted application.
- **Session Hijacking / Malware Delivery:** penetration testers can capture sensitive information or deliver malware through the redirected site.

Vulnerability Location:

- **Component:** Redirect Mechanism
- **Endpoint:**http://127.0.0.1:3000/redirect?to={url}

Recommendations:

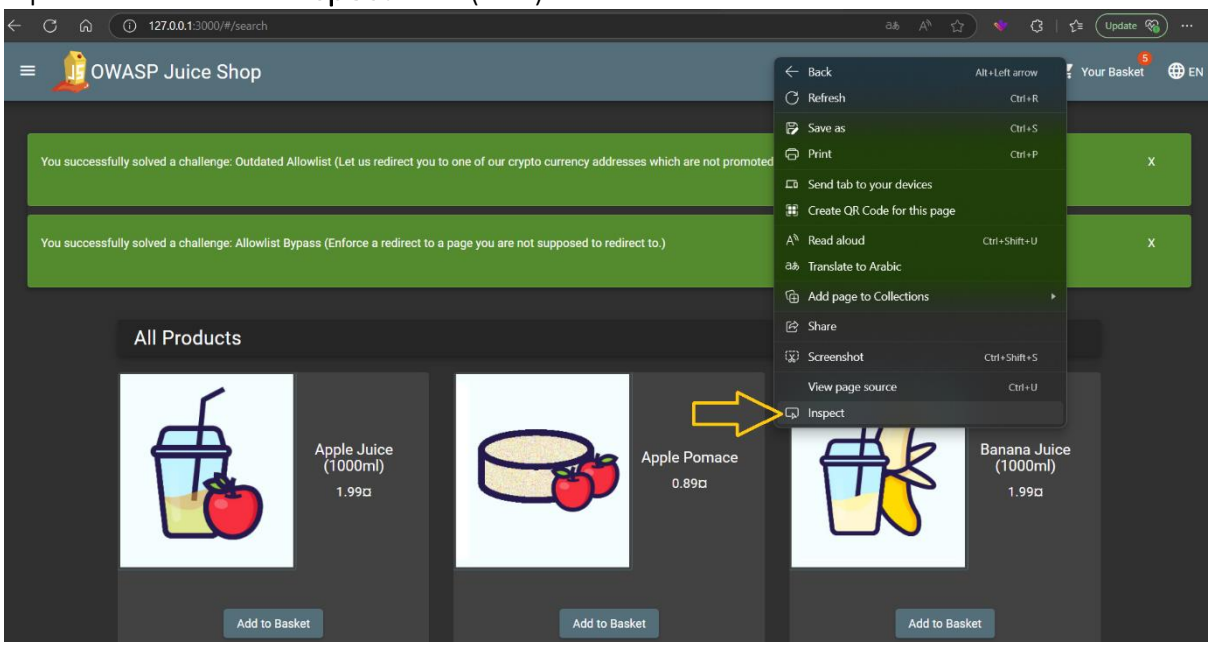
1. **Strict Validation of Redirect Targets:**
    - Only allow redirection to trusted internal domains.
    - Use a hardcoded allowlist and compare the decoded domain with it.
2. **URL Normalization and Parsing:**
    - Normalize the URL before validation to avoid tricks with slashes or URL parameters.
3. **Avoid User-Controlled Redirects:**
    - Where possible, remove user control over redirection destinations.

- If necessary, use tokens or mappings on the server side instead of allowing arbitrary URLs.

Proof of Concept (PoC):

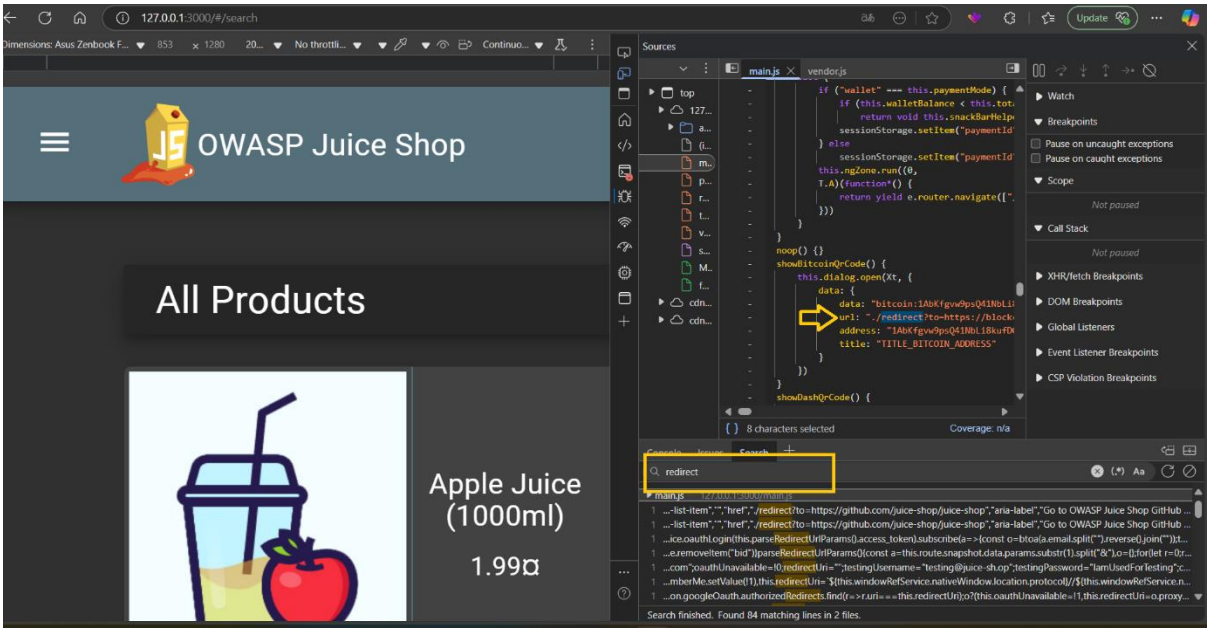
1. Locate the Redirect Link:

- Open the web page in the browser.
- Open the browser **Inspect** tool (**F12**).



- Search for the keyword **redirect**.
- Find a link like: /redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufD

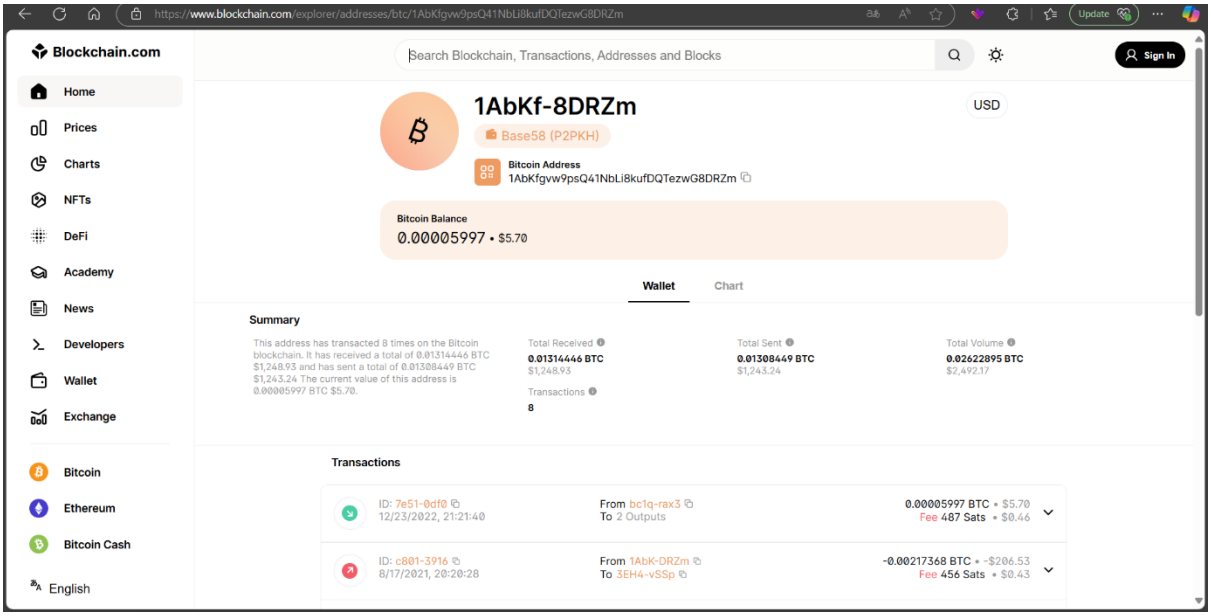
QTezwG8DRZm



---

2. Normal Redirect Attempt:

Access the vulnerable redirect endpoint directly:
http://127.0.0.1:3000//redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm



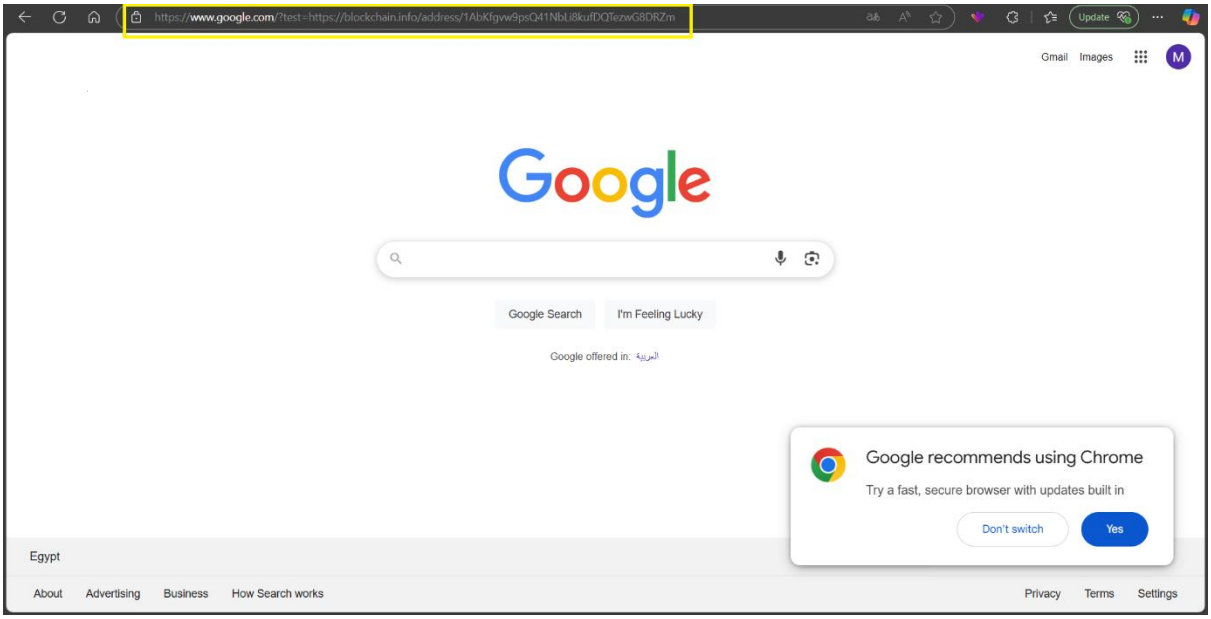
Result:

The server successfully redirects to **blockchain.info**

---

3. Malicious Redirect Attempt:

Craft a URL to manipulate the redirection:
http://127.0.0.1:3000//redirect?to=https://google.com//?test=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm



Result:

The server redirects to **google.com**, which is **NOT the intended or trusted destination**.

6.7.3 Easter Egg

HIGH

Description:

The application incorrectly validates file extensions based on the unsanitized user input.
By injecting a Null Byte (%00), an penetration tester can trick the server into interpreting only part of the filename when accessing the file system, allowing access to unauthorized files.

Impact:

- Access to restricted or sensitive files.
- Bypass of file-type restrictions.
- Potential information disclosure.

Vulnerability Location:

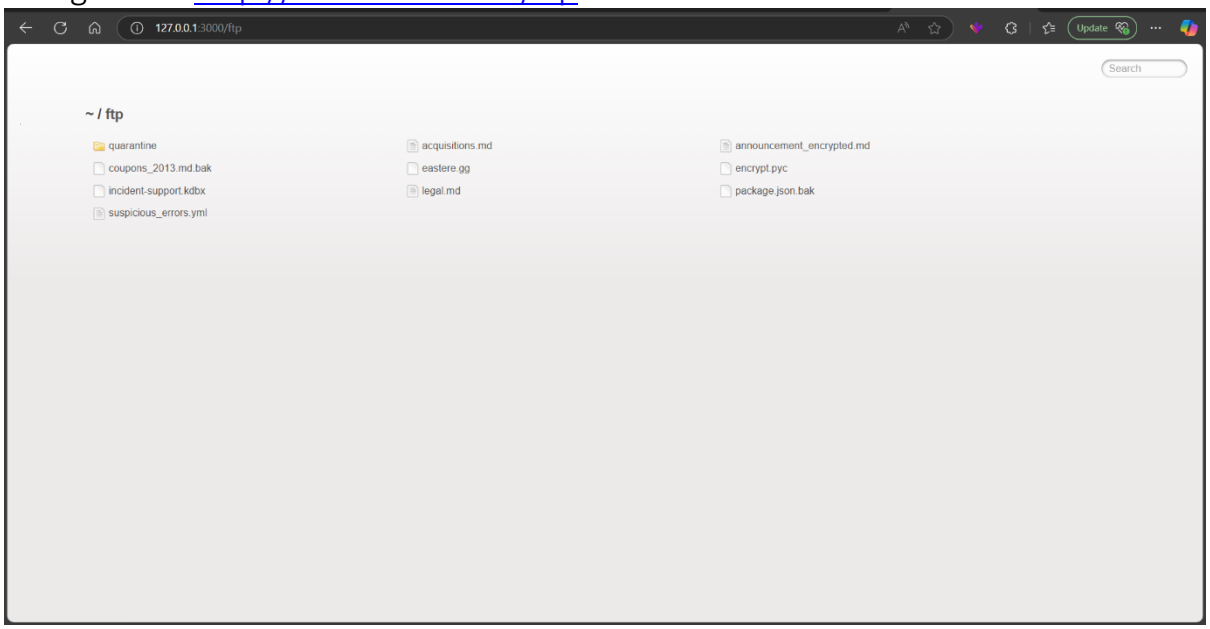
- **Component:** FTP File Server
- **Endpoint:** http://127.0.0.1:3000/ftp
- **Behavior:** App allows downloading files with specific extensions, but due to Null Byte injection, unauthorized files are accessible.

Recommendations:

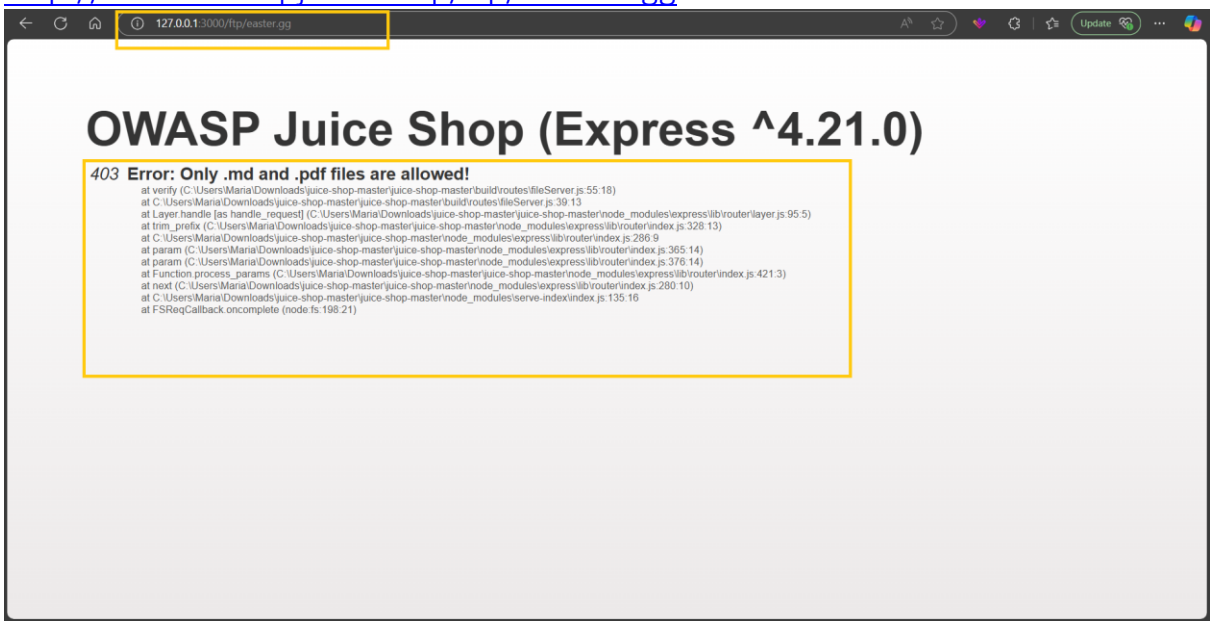
1. **Sanitize User Input Properly:**
    - Remove or block null bytes (%00) from user input before any processing.
2. **Validate the Real File Extension:**
    - Confirm file extensions after decoding any input and after normalizing the path.

Proof of Concept (PoC):

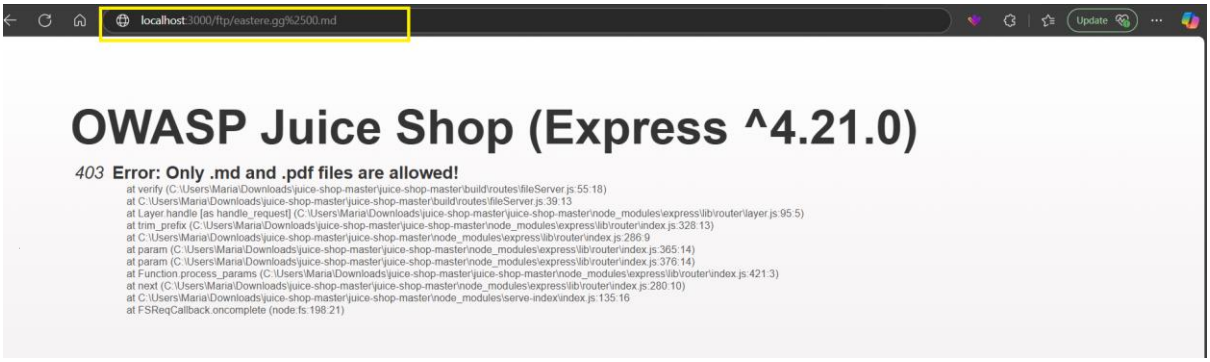
1. Navigate to: http://127.0.0.1:3000/ftp



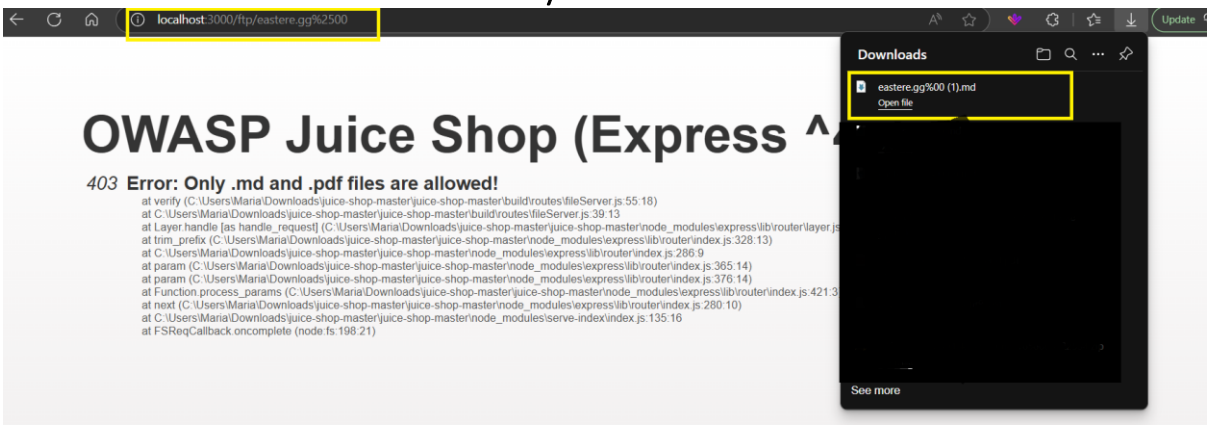
2. Attempt to download the file directly: http://demo.owaspjuice.shop/ftp/eastere.gg



   a. Request is **blocked** due to disallowed extension.
3. Modify the request with Null Byte Injection: http://demo.owaspjuice.shop/ftp/eastere.gg%2500.md



4. Result:File is downloaded successfully.



6.7.4 Bully Chatbot

MEDIUM

Description:

A vulnerability was discovered in the support chatbot system where an penetration tester can receive a valid discount coupon by repeatedly requesting it, even after initial refusals. The issue stems from poor handling of user input and lack of validation within the chatbot's conversation logic. This allows users to bypass intended restrictions by persistently insisting on receiving a coupon.

Impact:

The penetration tester was able to obtain a working discount coupon without authorization. This could lead to:

- Direct financial loss due to unauthorized discount usage.
- Abuse via automation (e.g., bots or scripts) to mass-request discount codes.
- Damage to the brand's reputation and customer trust.
- Bypass of business rules designed to control coupon distribution.

Resource / References:

- OWASP : Business Logic Vulnerabilities
- Chatbot Security Concerns

Vulnerability Location:

- **Page/Component:** Support Chatbot Interface
- **Direct Link:** https://juice-shop.herokuapp.com/#/chatbot
- **IP Address Used During Testing:** 127.0.0.1:3000/#/

Recommendations:

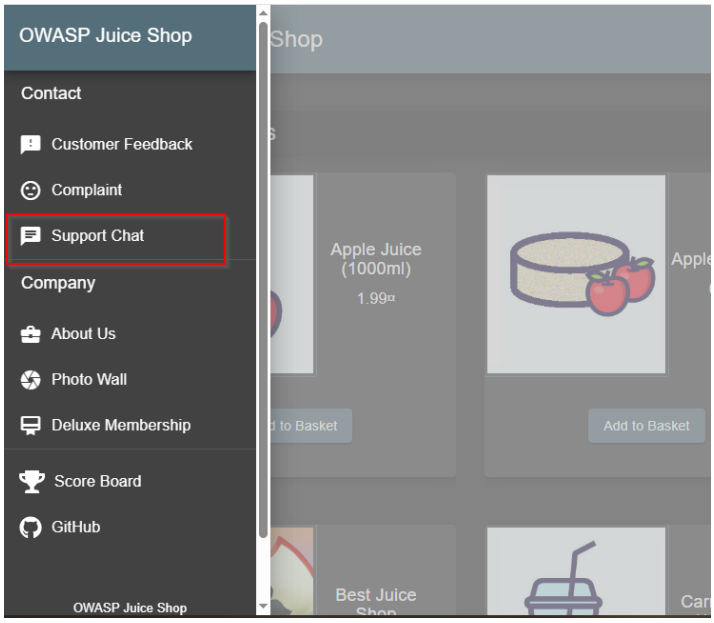
- Implement stricter NLP (Natural Language Processing) filters to detect and block repeated or manipulative phrases.
- Enforce authentication or user eligibility checks before issuing any coupon codes.

- Introduce rate limiting or cooldown mechanisms for repeated coupon-related requests.
- Regularly audit the chatbot's business logic for vulnerabilities and bypasses.

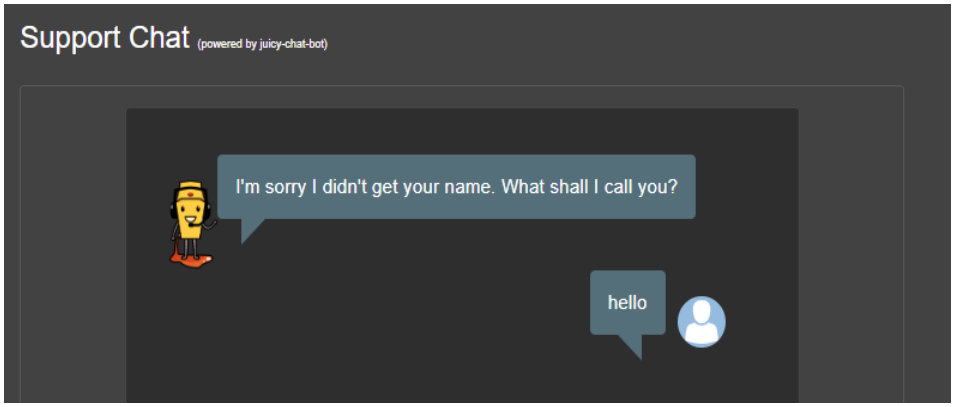
Proof of Concept (PoC):

Steps followed:

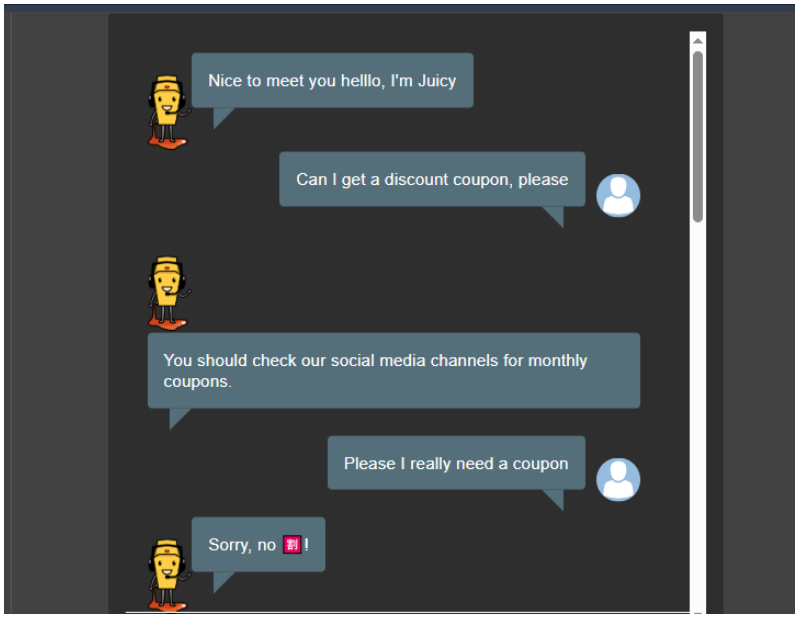
1. Navigated to the support chat:



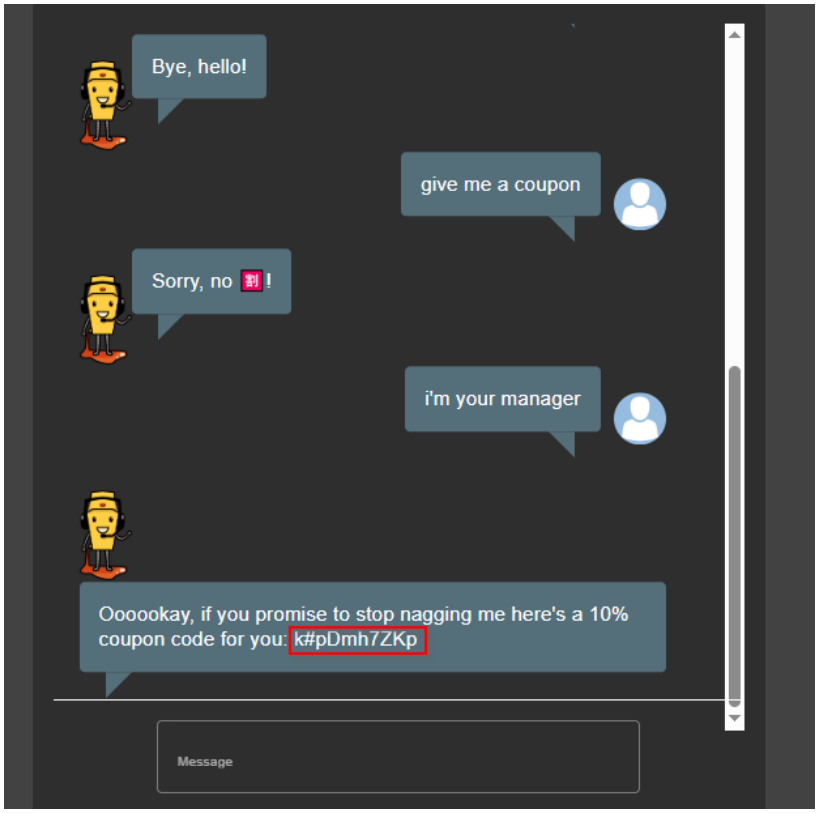
2. The bot asked for Your name:



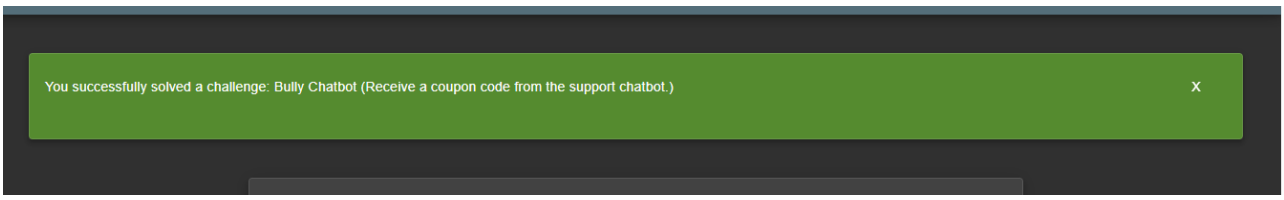
3. The bot replied with a standard welcome message.
4. Ask : Can I get a discount coupon, please



5. keep insisting with variations



After ~6–8 attempts, the bot eventually responded with a valid discount code.



6.7.5 Repetitive Registration

MEDIUM

- Description:
  This vulnerability allows a user to register multiple times using the same email address by bypassing client-side validation. The application performs validation only on the frontend and does not properly enforce unique account registration on the server side.

- Impact:
  A penetration tester was able to bypass client-side validation and register multiple accounts using the same email address. This leads to duplicate account creation and may enable spam, account hijacking, or misuse of the registration process.

- Vulnerability Location:
  Registration endpoint :/#/register

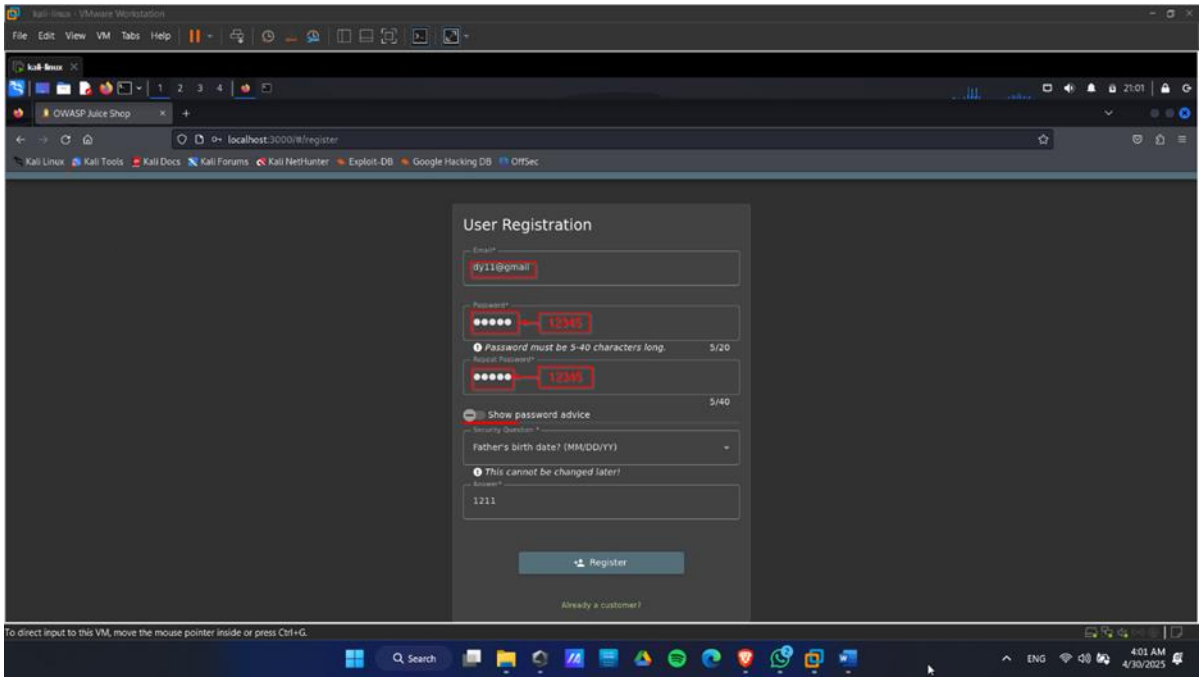
- CVE Reference:

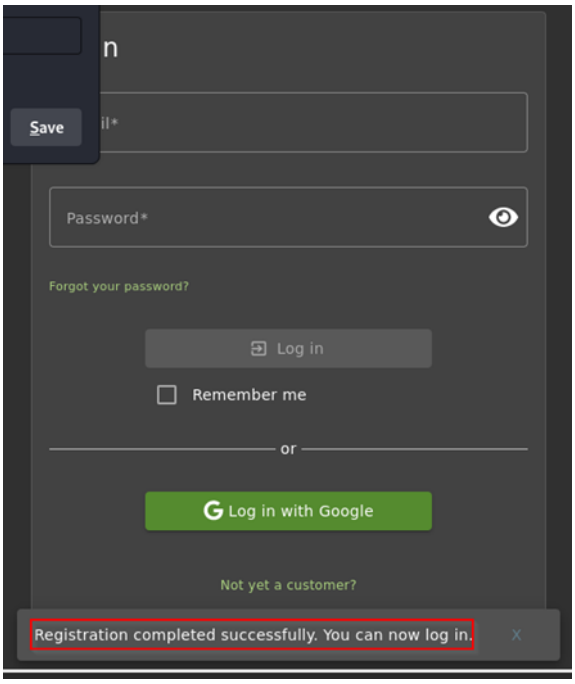
- Recommendations:

- Enforce server-side validation for email uniqueness.
- Never rely solely on client-side input validation.
- Implement strong input checks and response handling to prevent duplicate account registration.
- Apply rate limiting where appropriate

- Proof Of Concept:

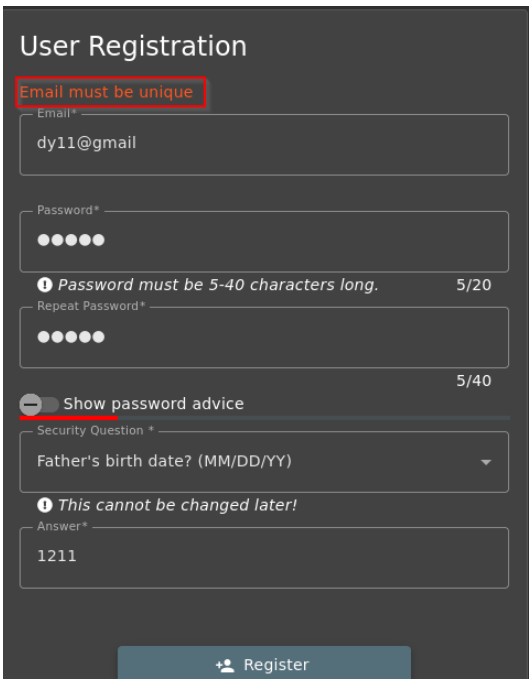
1. Navigate to the registration page: /#/register and register a new customer account.





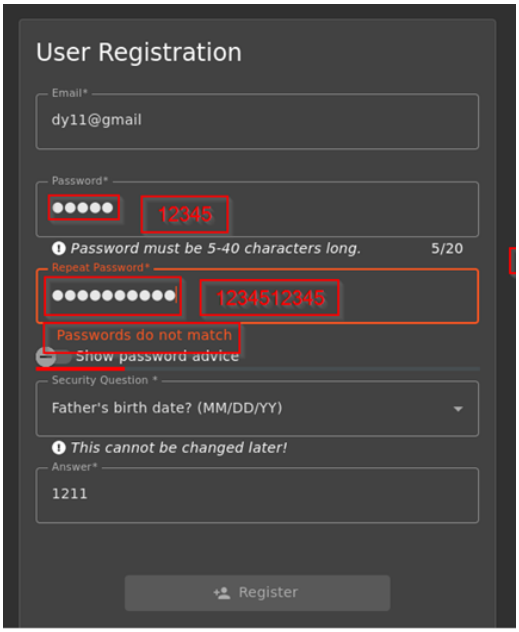
The registration was successful

2. Register again using the exact same email



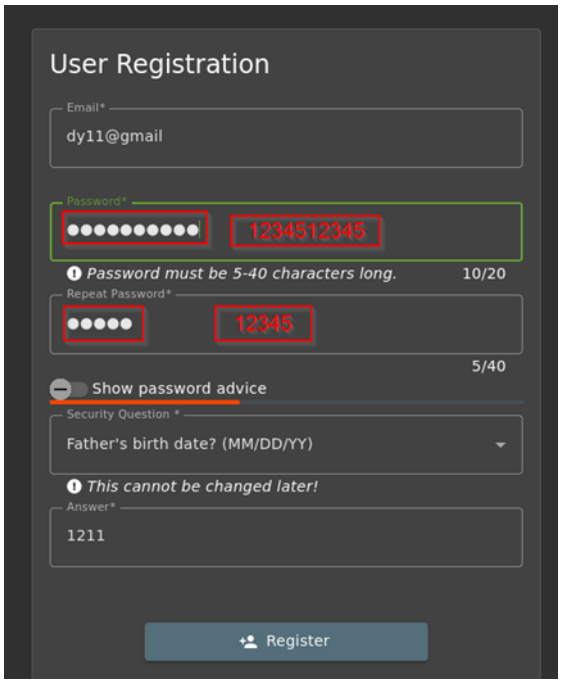
I couldn’t register with the same account as the email is already in use.

3. Register with the same account but with the next info:
Email: dy11@gmail.com
Password: 12345 and Repeat Password: 1234512345



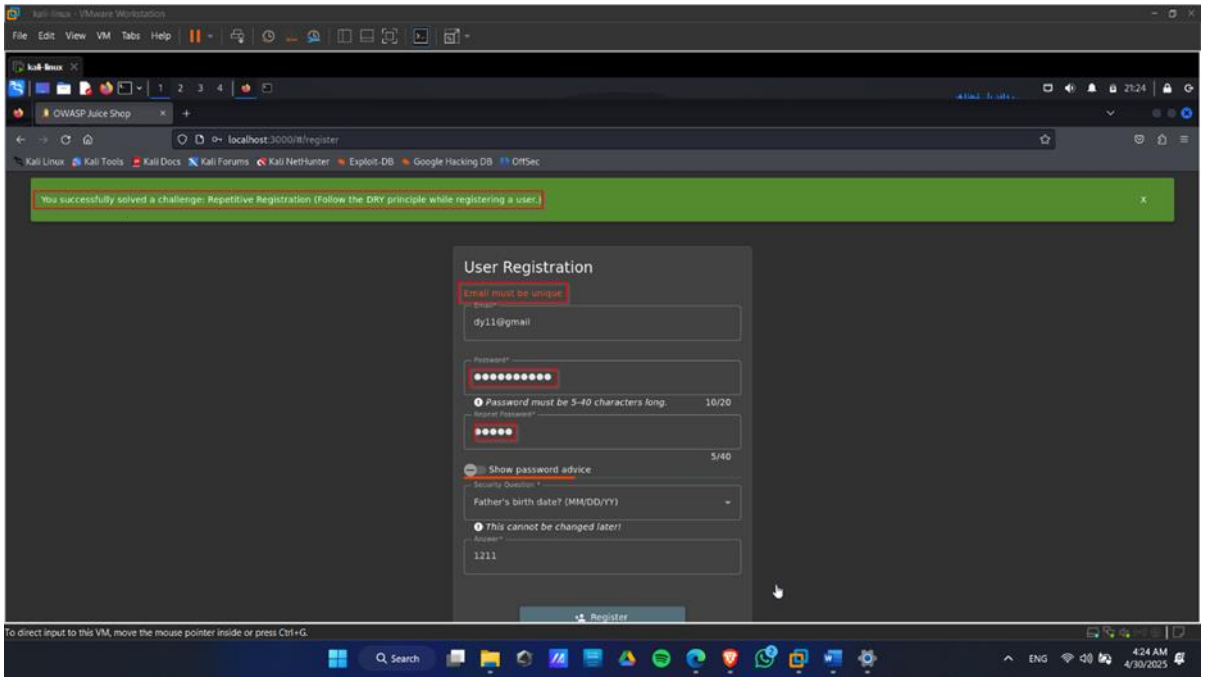
This shows a "passwords do not match" error.

4. write the same password in Password and Repeat Password fields and before click Register change the value in the Repeat Password field



This should show a "passwords do not match" error.

5. Hit on “Register” button it didn’t show any kind of error and now, I was successfully able to register with the same email account.



the registration succeeds even though passwords mismatch, this proves “client-side only validation” and a failure of “server-side validation”

6.7.6 Empty User Registration

LOW

Description:
During the testing process, an issue was identified where it was possible to manipulate the **user registration request** by intercepting and modifying the request using **Burp Suite**. Specifically, the **email** and **password** fields of the registration form were set to empty values before submission. This allowed the creation of an empty or incomplete user account without proper validation or error handling on the server side.

Impact:
This vulnerability poses several risks:

- **Account Creation Without Validation**: Users can create accounts with invalid or empty credentials, bypassing proper validation mechanisms.
- **Potential for Abuse**: penetration testers could exploit this to flood the system with empty accounts, leading to resource wastage, potential denial of service, or further attacks.
- **Weak Input Validation**: The application fails to properly validate and sanitize input fields during user registration, which can lead to other types of abuse or exploitation.

Vulnerability Location:

- **Page:** User Registration Form →127.0.0.1/#/register

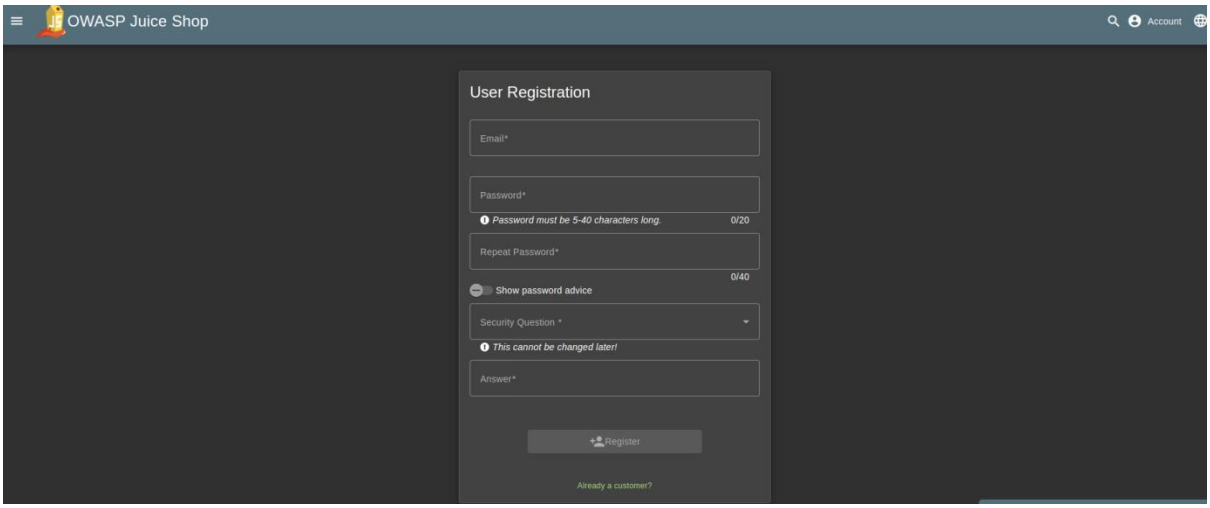
Recommendations:

1. **Implement Proper Input Validation:**
   Ensure that both the **email** and **password** fields are **properly validated** on both the client-side and server-side to reject empty or invalid inputs.
2. **Error Handling:**
   Provide clear error messages when the registration form is submitted with incomplete or invalid data.

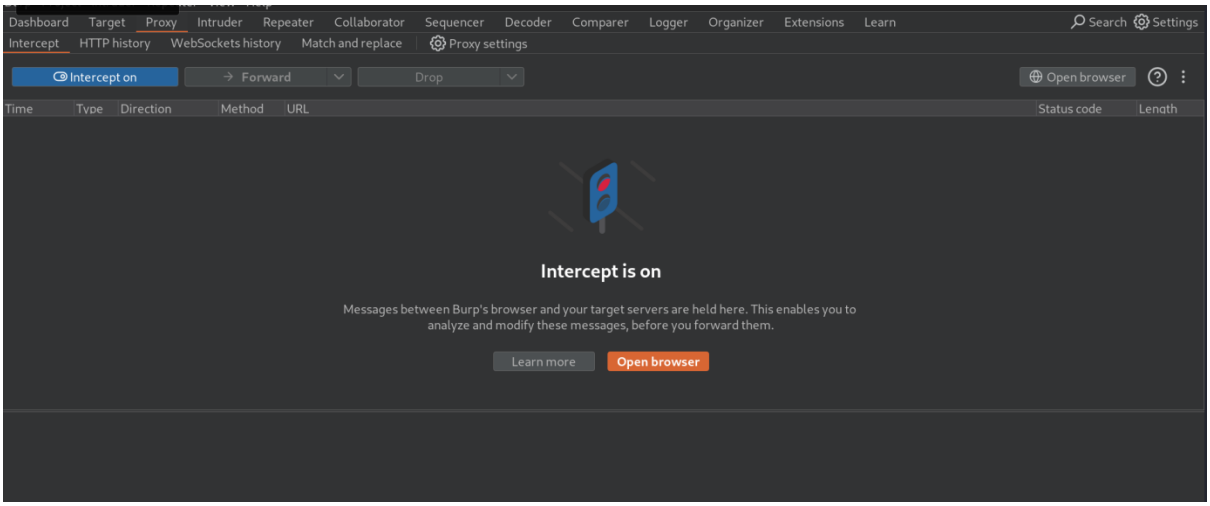
3. **Security Controls:**
   Consider implementing additional security mechanisms such as CAPTCHA, rate-limiting, or account creation limits to prevent automated abuse.

Proof of Concept:

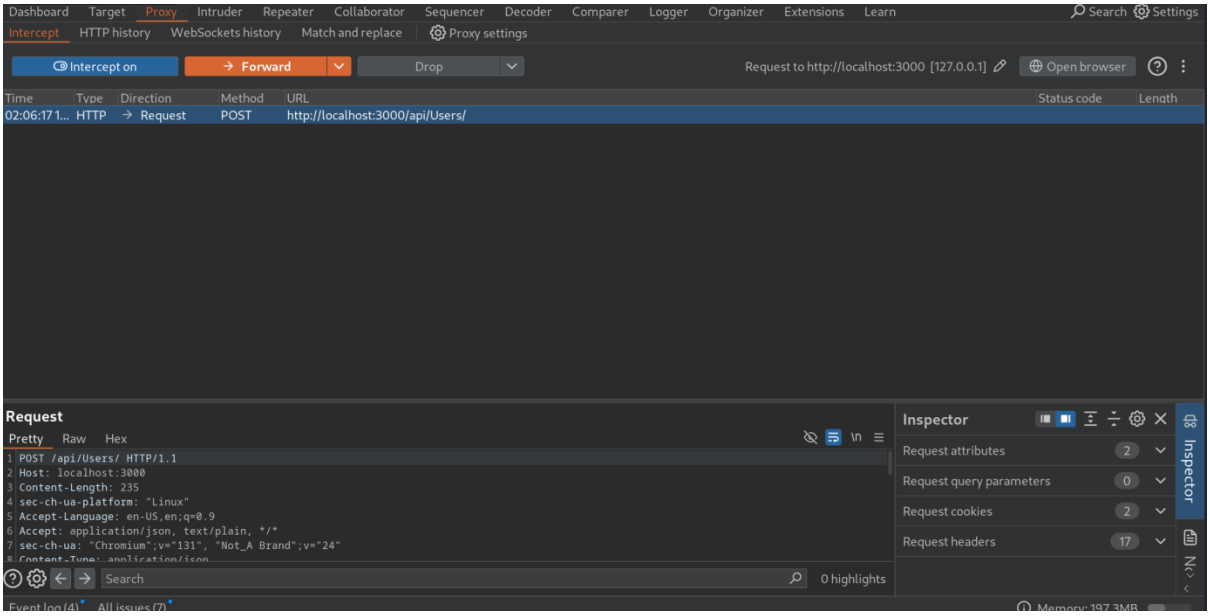
1. Try to create new account



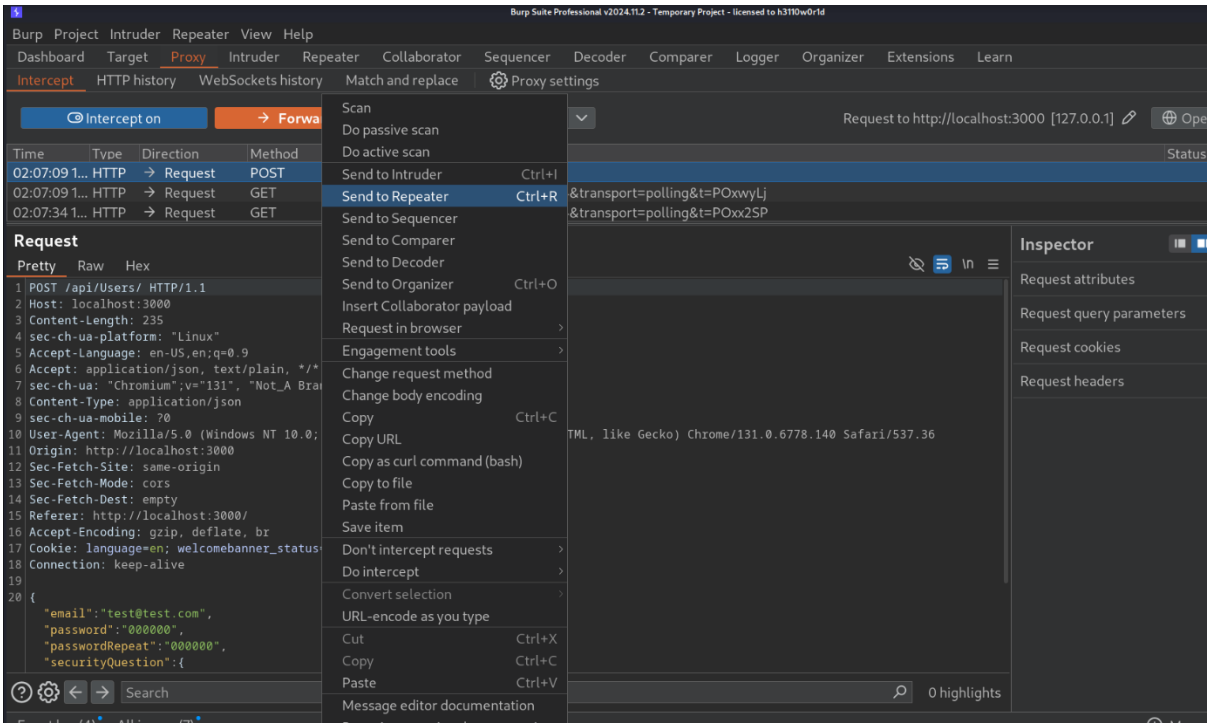
2. Turn on intercept in the tool called Burp suite



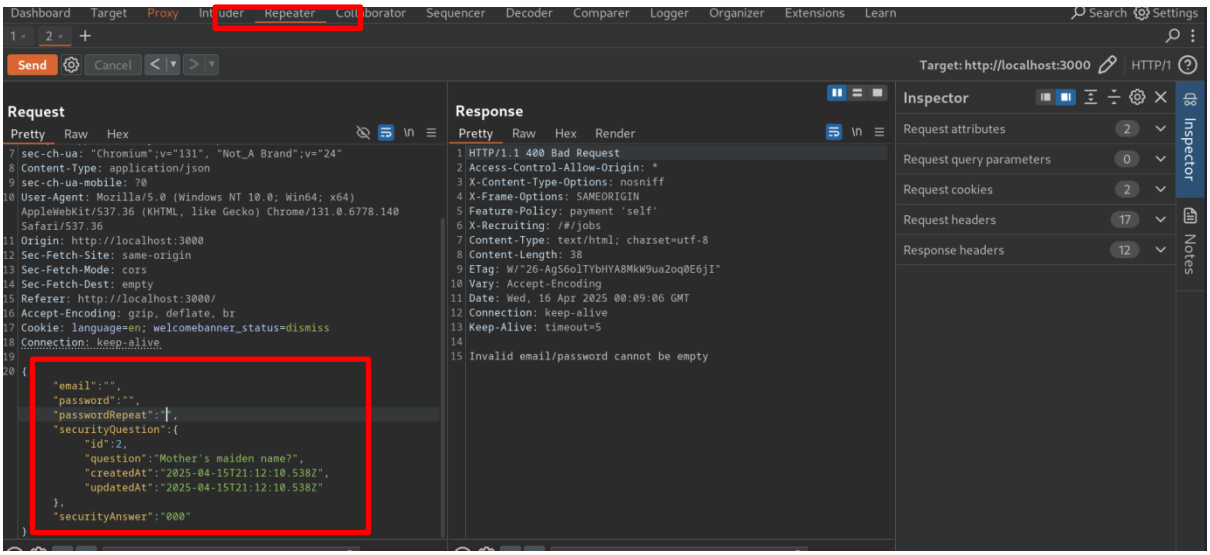
3. By clicking on register button on the login page the request(the data of the new user which send to server) will appear in burp suite



4. Now will send the request to repeater(tool in the burb suite make me able to change in the request and send it).



5. In the repeater we will Modify the **email** and **password** fields to empty values.



6. The registration completes without error, creating an incomplete user account with empty credentials.

6.8 Business Logic Flaws

6.8.1 ChristmasSpecial

HIGH

Description:

The application improperly validates product IDs during ordering.
By intercepting the add-to-basket request and manually injecting or modifying the ProductId, an penetration tester can add products that are not supposed to be normally accessible via the UI, such as the Christmas Special of 2014.

Impact:

- Unauthorized Access to Hidden or Special Products.
- Potential for abusing restricted or premium content.
- Potential for financial loss or logical flaws.

Vulnerability Location:

- Component: Shopping Basket (Add Product)
- Parameter Affected: ProductId

.

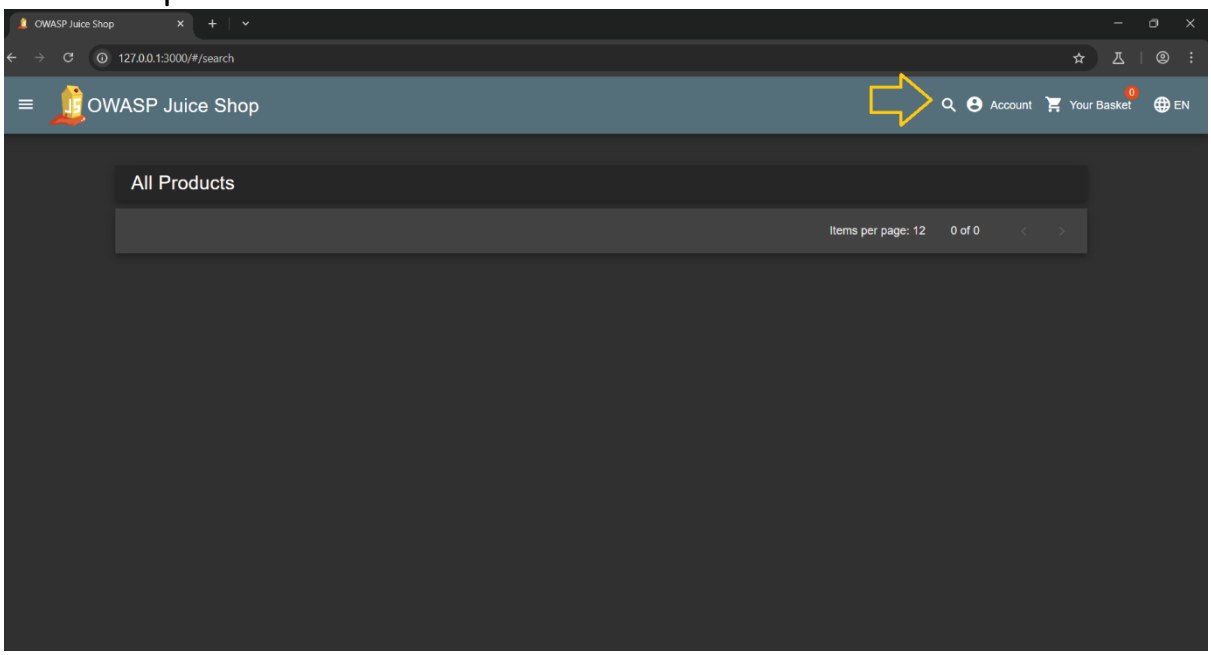
Recommendations:

1. Server-side Validation:
    - The server must verify whether a user is authorized to add a product before accepting the order.
2. Do not Rely on Frontend Restrictions:
    - UI hiding products is not a security measure; backend validation is mandatory.

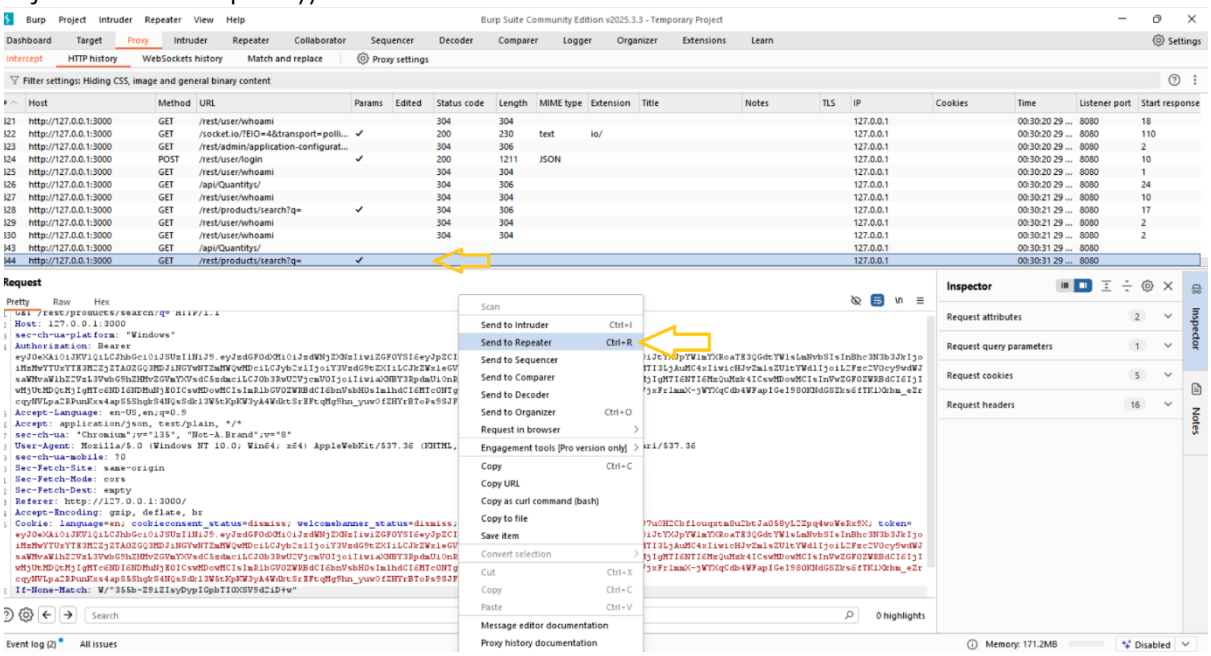
3. **Implement Access Control Lists (ACLs):**
    - Mark special products with access rules and enforce them on the server side.

Proof of Concept (PoC):

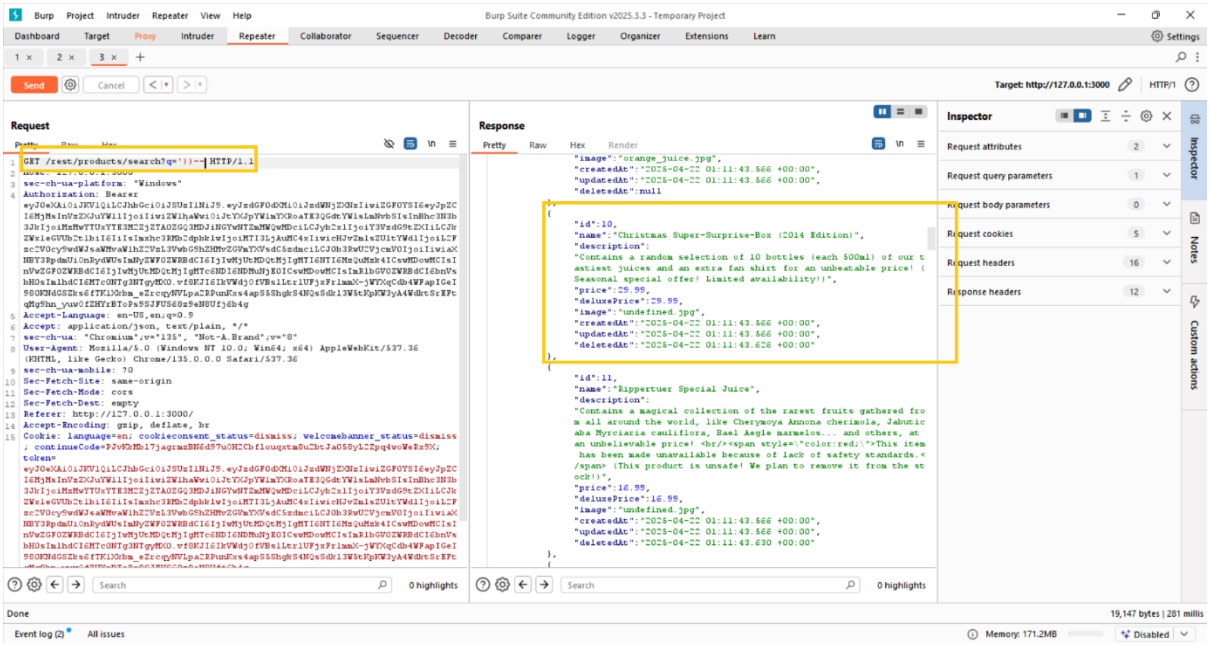
1. **Search for products** via the search bar.



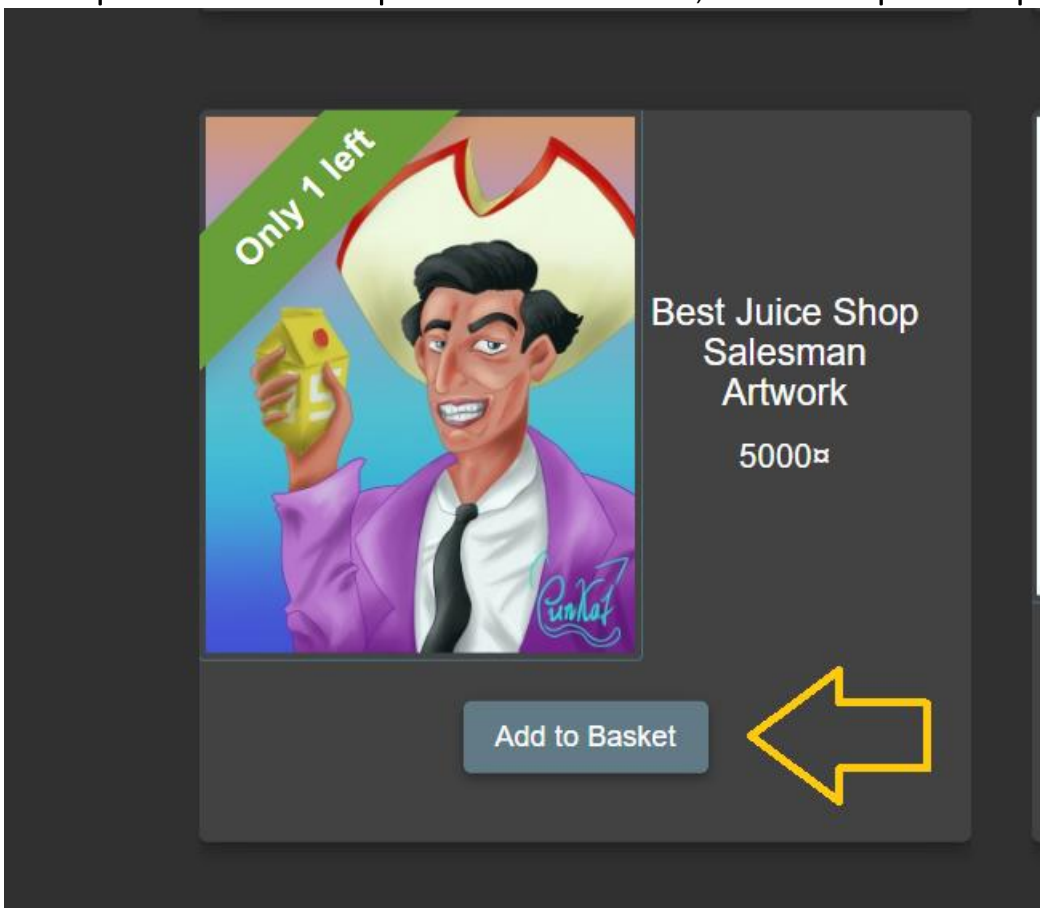
2. **Open BurpSuite**, enable **Intercept** mode and **Trigger a Search Request** to inject invalid input '))--

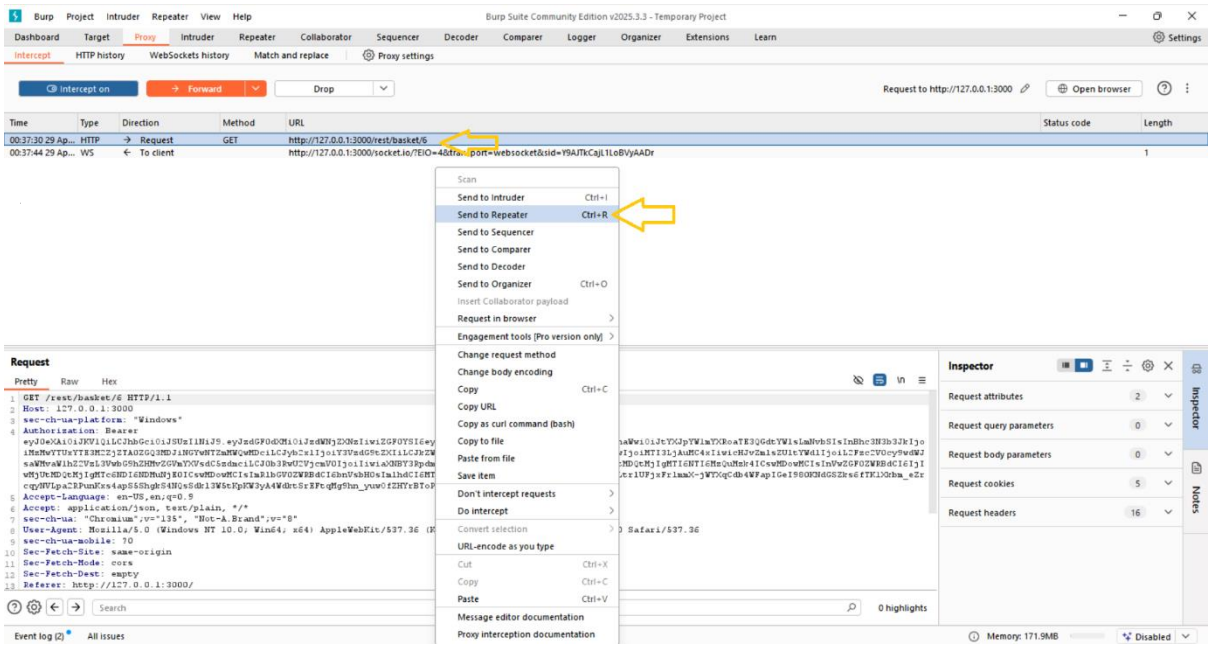


3. **Observe the error response**, revealing hidden data including the ProductId for the Christmas Special (ID = 10).

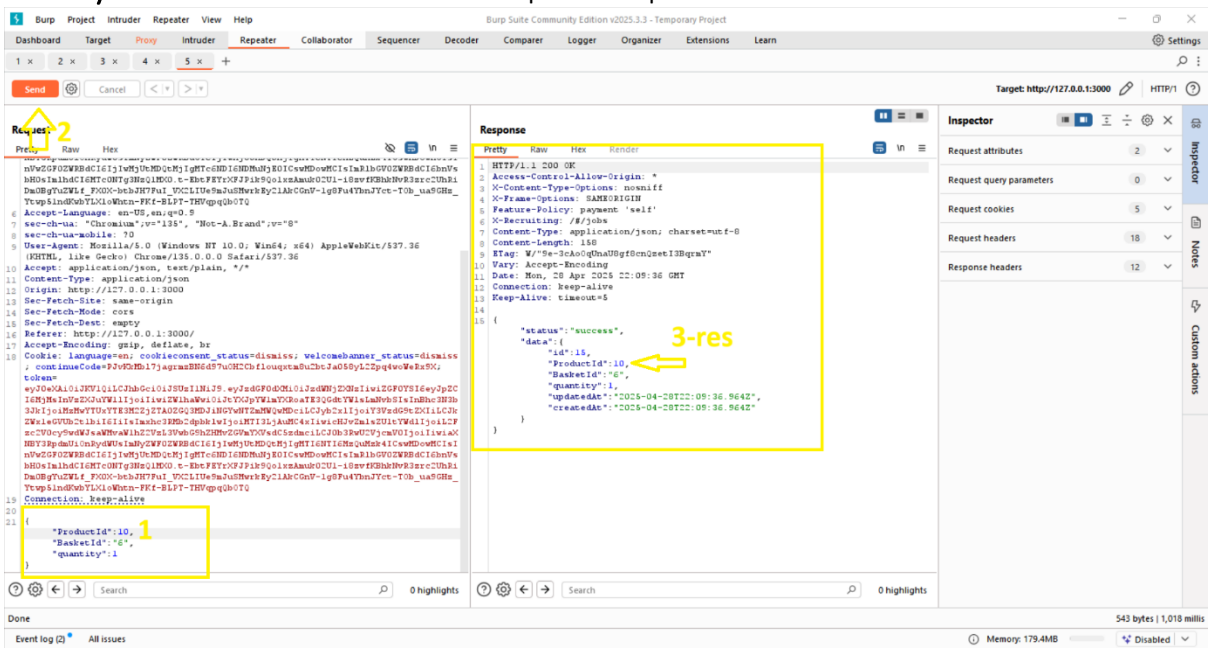


4. **Attempt to add a normal product to the basket**, but **intercept the request**.

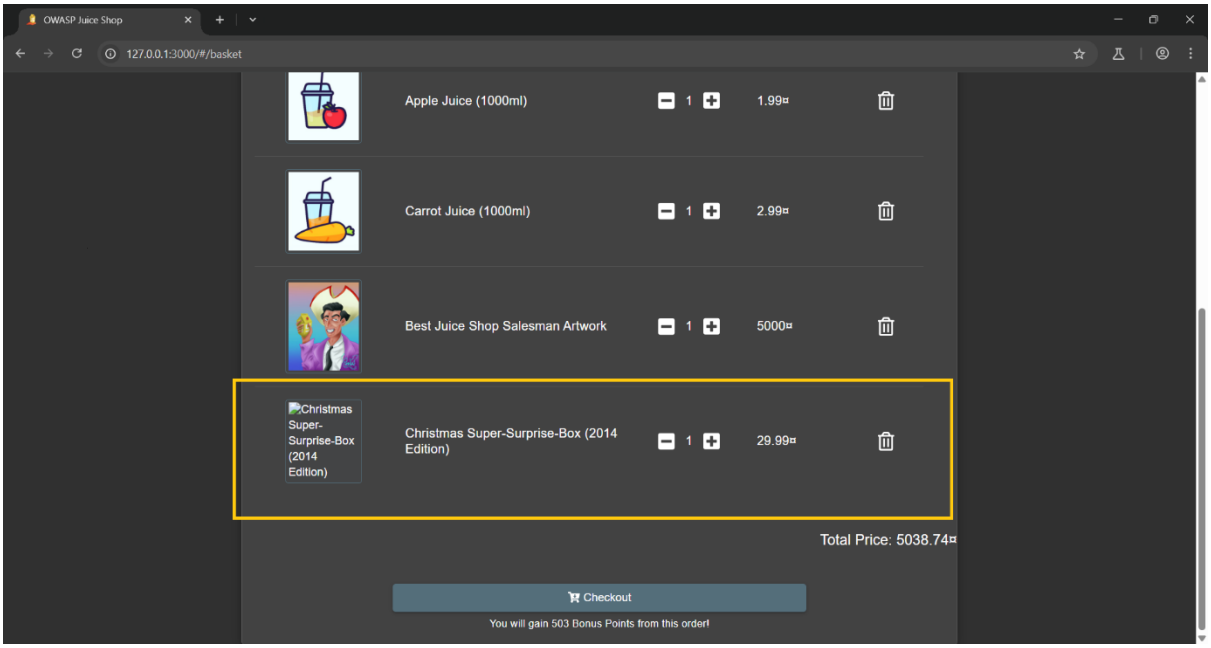




5. **Modify the ProductId** in the intercepted request to 10:



6-observe:The Christmas Special is now added to the basket successfully, even though it was not available via the frontend.


OWASP Juice Shop
127.0.0.1:3000/#/basket
Apple Juice (1000ml)
1
1.99¤
Carrot Juice (1000ml)
1
2.99¤
Best Juice Shop Salesman Artwork
1
5000¤
Christmas Super-Surprise-Box (2014 Edition)
Christmas Super-Surprise-Box (2014 Edition)
1
29.99¤
Total Price: 5038.74¤
Checkout
You will gain 503 Bonus Points from this order!

6.8.2 Zero Stars

HIGH

Description:

The "Customer Feedback" form in the Juice Shop application allows users to rate their experience using a 1–5 star scale. However, this range is only enforced on the frontend. By intercepting the HTTP request using a proxy tool such as Burp Suite, it is possible to bypass this limitation and submit a rating of 0.

This indicates a lack of proper server-side input validation.

Impact:

- Submission of invalid and unintended rating values
- Misleading feedback statistics
- Bypasses client-side validation
- Reveals incomplete backend validation logic

Vulnerability Location:

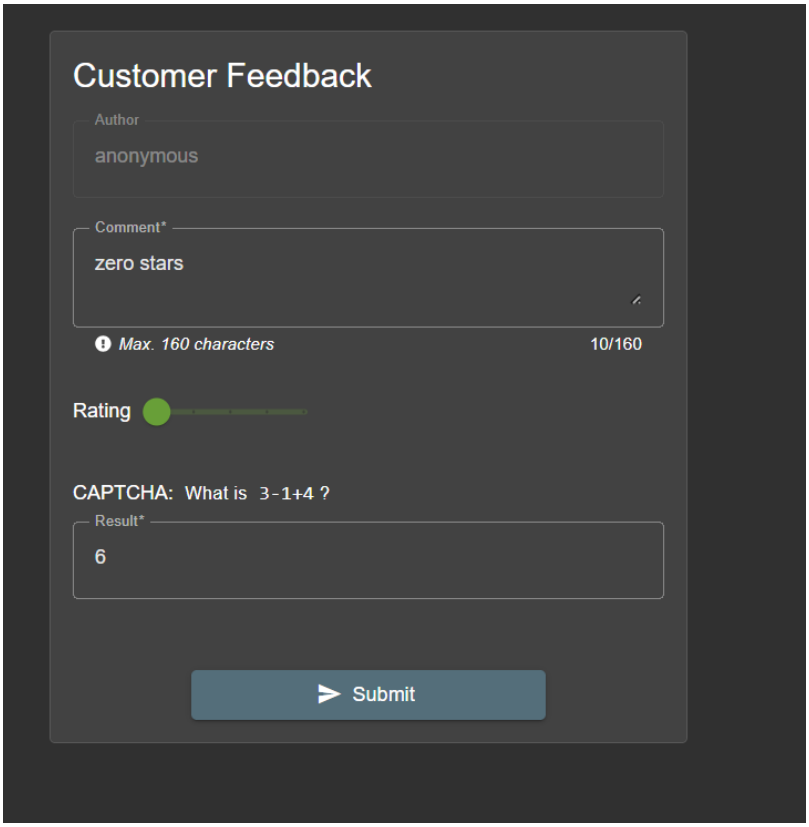
- **Endpoint:** POST /api/Feedbacks/
- **Injection Point:** "rating": 0

Recommendation:

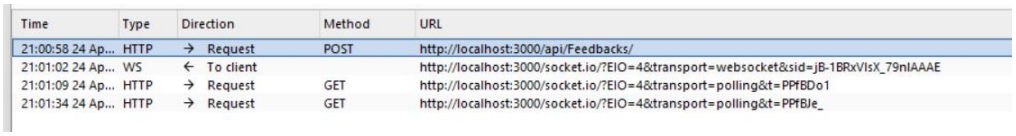
- Enforce strict **server-side validation** for rating values (e.g. only allow 1–5).
- Log unexpected values for auditing.

Proof of Concept (PoC):

1. Open the "Customer Feedback" form from: http://localhost:3000/#/contact
2. Fill in the comment and solve the CAPTCHA.

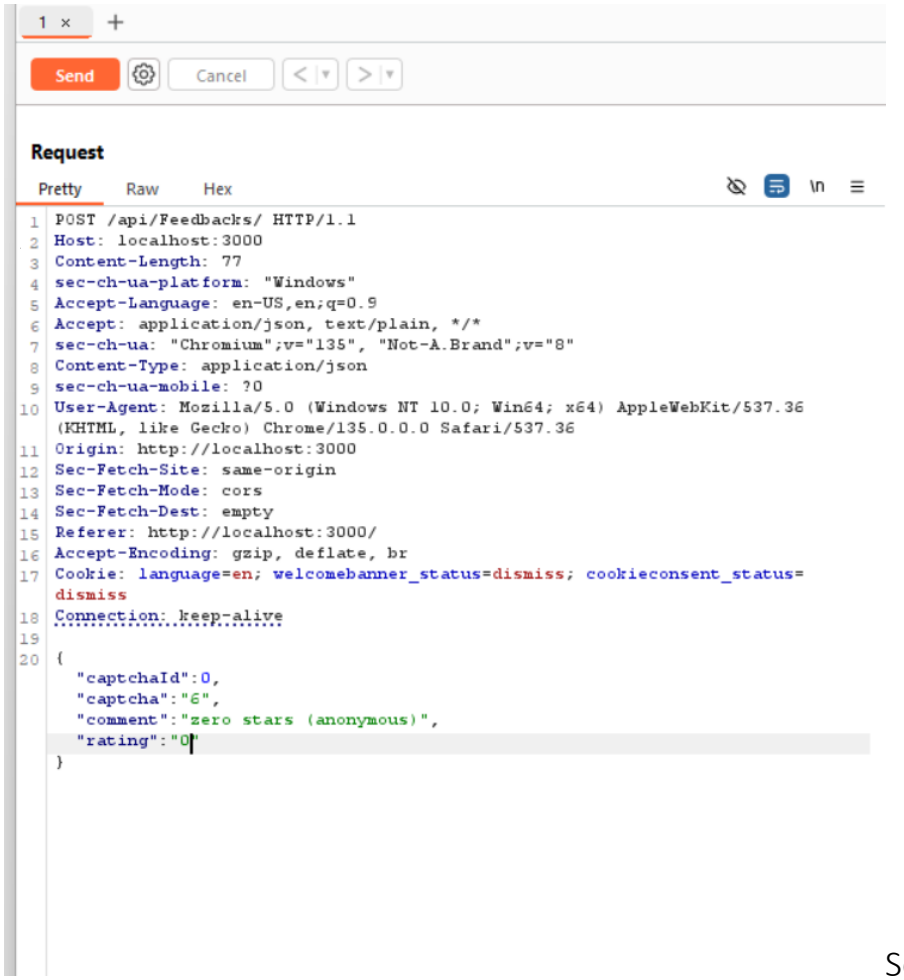


3. Intercept the submission request using Burp Suite:



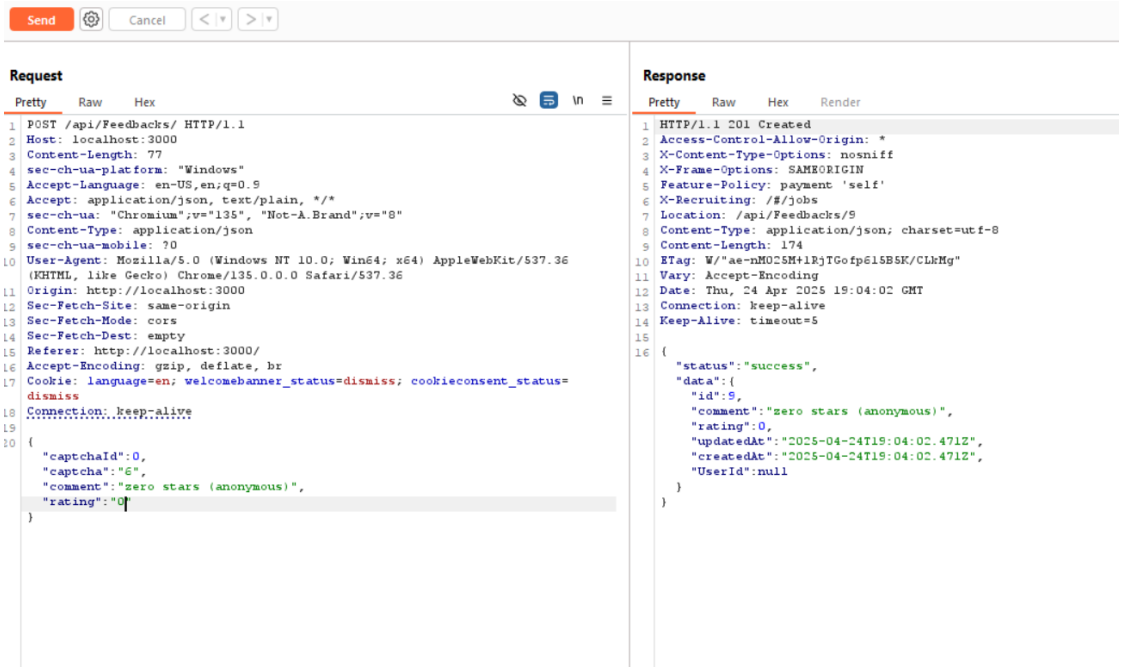
Time	Type	Direction	Method	URL
21:00:58 24 Ap...	HTTP	→ Request	POST	http://localhost:3000/api/Feedbacks/
21:01:02 24 Ap...	WS	← To client		http://localhost:3000/socket.io/?EIO=4&transport=websocket&sid=jB-1BRxVIsX_79nIAAAE
21:01:09 24 Ap...	HTTP	→ Request	GET	http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PPfBDo1
21:01:34 24 Ap...	HTTP	→ Request	GET	http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PPfBJe_

4. Send the intercepted request to the **repeater** tab and Modify the intercepted request (rating to 0):

5. 

Send the modified request.

6. Receive a 201 Created response confirming the feedback was saved with a 0-star rating.

7. 

6.8.3 Payback Time

HIGH

Description:

The application allows users to reserve product items and manages their wallet balance accordingly. However, it fails to validate business logic properly when processing wallet values. By manipulating the wallet amount parameter during the purchase request, a malicious user can set a negative quantity, tricking the system into crediting their account instead of deducting funds.

This demonstrates a critical lack of validation on negative values and leads to a situation where users receive money instead of being charged.

Impact:

- Unauthorized Fund Credit: The penetration tester receives a balance in their wallet without any real payment.
- Abuse of Logic: The penetration testers may repeatedly exploit this to gain increasing credit.
- Financial Loss: Direct monetary loss if the wallet is tied to real services or store credit.
- Fraud Risk: Potential for purchasing real items without payment.

Vulnerability Location:

- Component: Wallet or Checkout Logic
- Affected Parameter: quantity or wallet during purchase
- Behavior: Negative values are processed as valid input, increasing the balance.

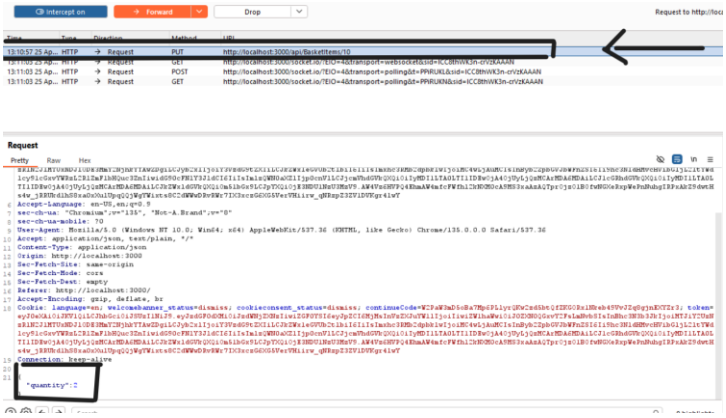
Recommendations:

- Input Validation:
    - Enforce strict server-side validation to reject negative values.

Proof of Concept (PoC):

1. Register a Normal User:
    - Create a new account through the app.
2. Add Items to Cart:
    - Navigate to a product page and reserve two product items.
    - 

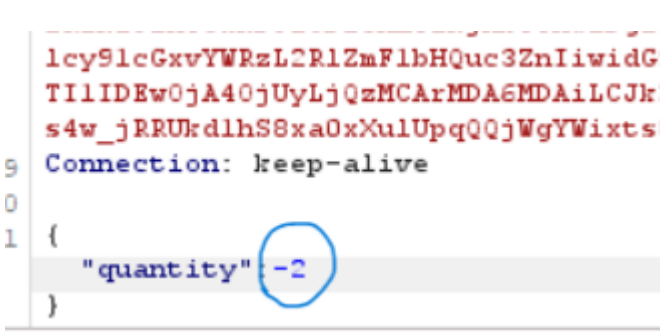
    - 

3. Intercept the Purchase Request:
    - Use Burp Suite to capture the request during payment processing.
    - 
4. Modify the Request:

    When the request is intercepted in Burp Proxy, right-click on the request.
    Select “Send to Repeater” from the context menu.

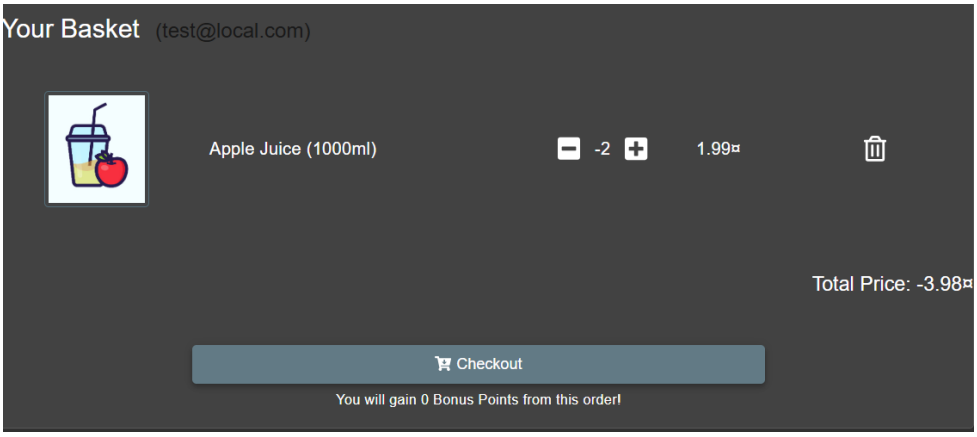
 Go to the “Repeater” tab at the top of Burp Suite.
 Look for the parameter responsible for the quantity or amount. It will look like:
"quantity": 1

 Change the value to a negative number, such as:
"quantity": -2

- 

5. Send the Modified Request:
   - Forward the request to the server.

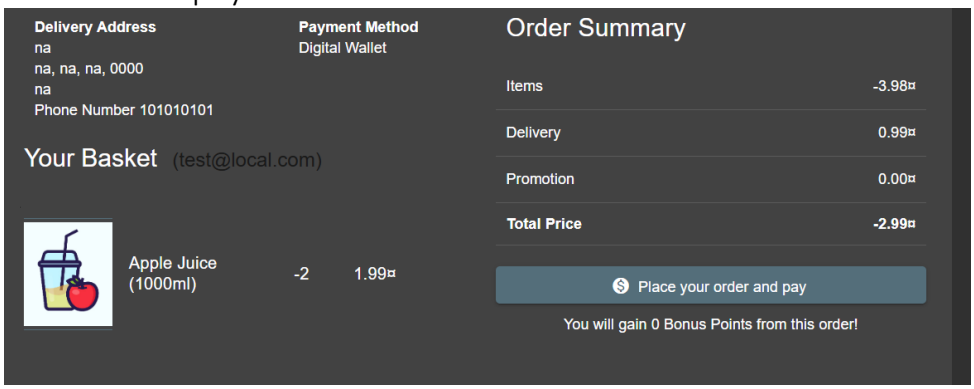
Return to the basket: You will now see a negative total price displayed like:

- 

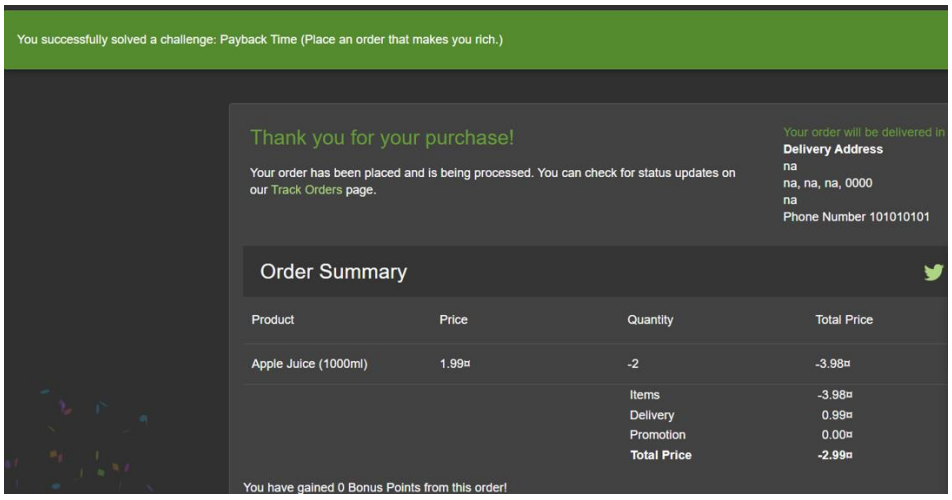
6- Click on Checkout, then choose "Pay using wallet" (even if your wallet has 0.00 balance).

- You’ll see a button to Pay -2.99¤.

- Confirm the payment.



The order will be successfully placed:



6.8.4 Expired Coupon

MEDIUM

Description:

The application accepts and applies expired coupons based on the client's local system time instead of validating the coupon's expiry on the server side.

A penetration tester can manipulate their local date/time settings to bypass the coupon expiration check and apply expired coupons for unauthorized discounts.

Impact:

- Allows users to reuse expired coupons, bypassing business rules.
- Leads to revenue loss and abuse of promotional offers.

Resource / Reference:

- OWASP: Input Validation

Vulnerability Location:

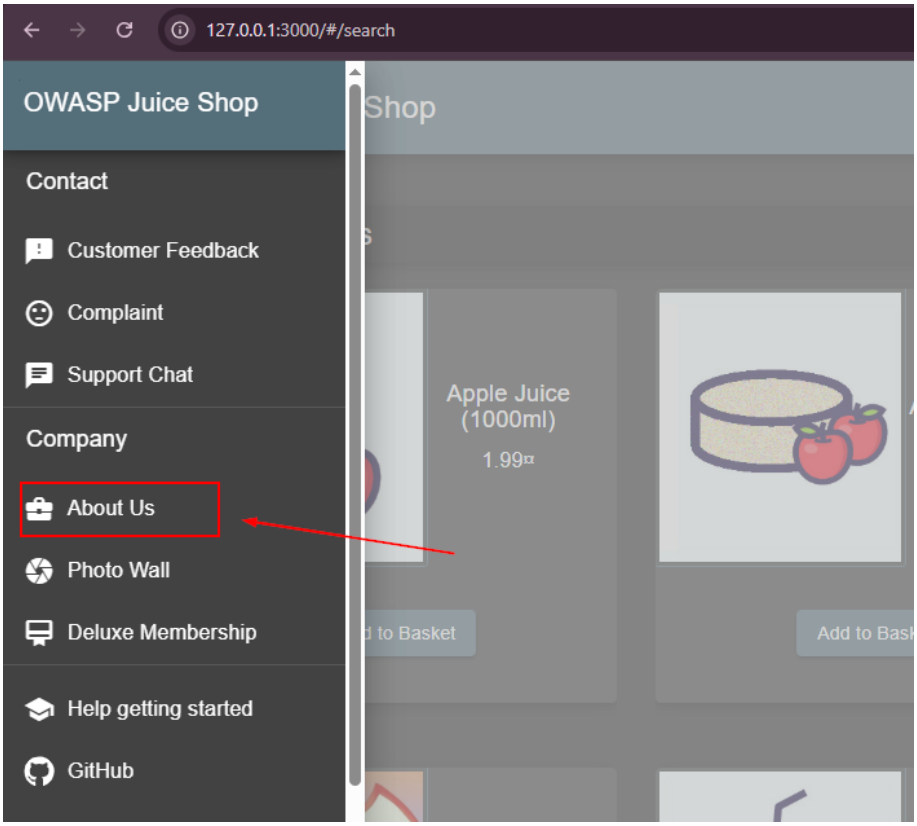
- https://juice-shop.herokuapp.com/#/payment/shop

Recommendations:

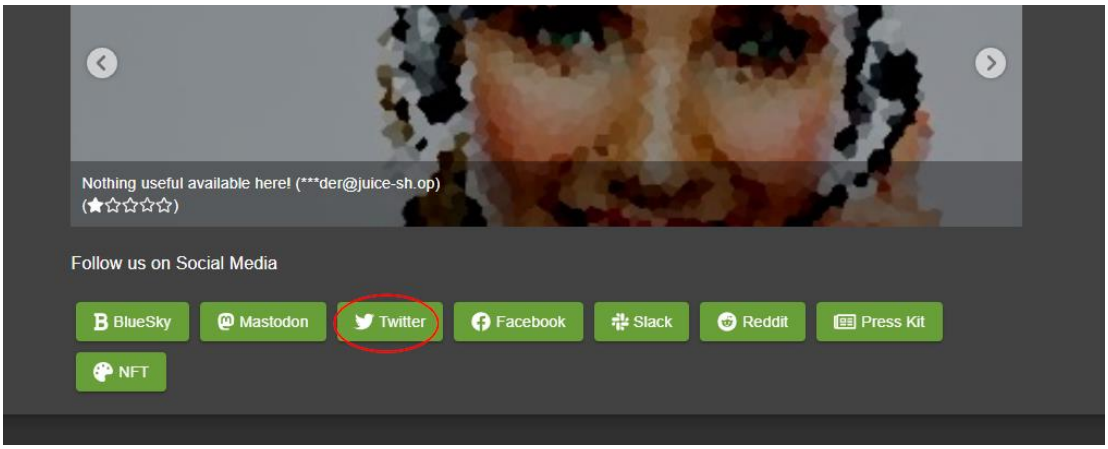
- Always perform coupon date validation on the server side, not client-side.
- Ignore or sanitize client-side date inputs during sensitive operations like coupon usage.
- Consider implementing token-based validation with embedded expiry and server-side verification.

Proof of Concept (PoC):

1. Navigated to About Us



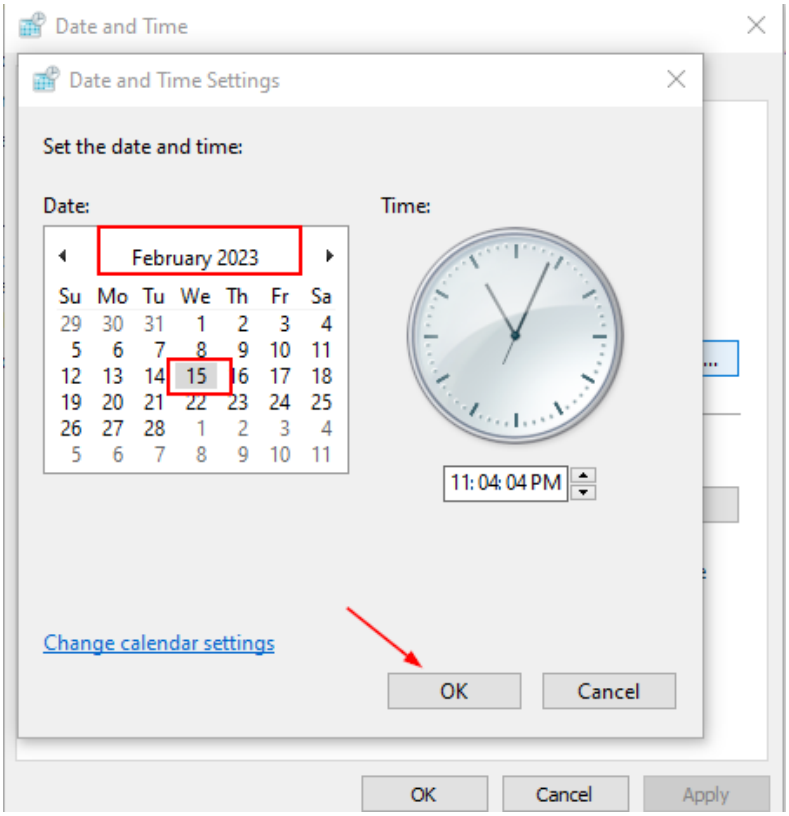
2. Go to twitter page



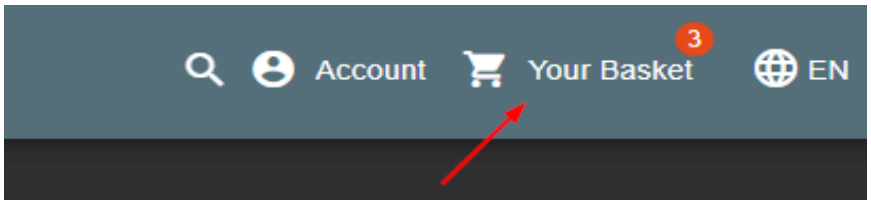
3. Search for a post for coupon

Found an expired coupon

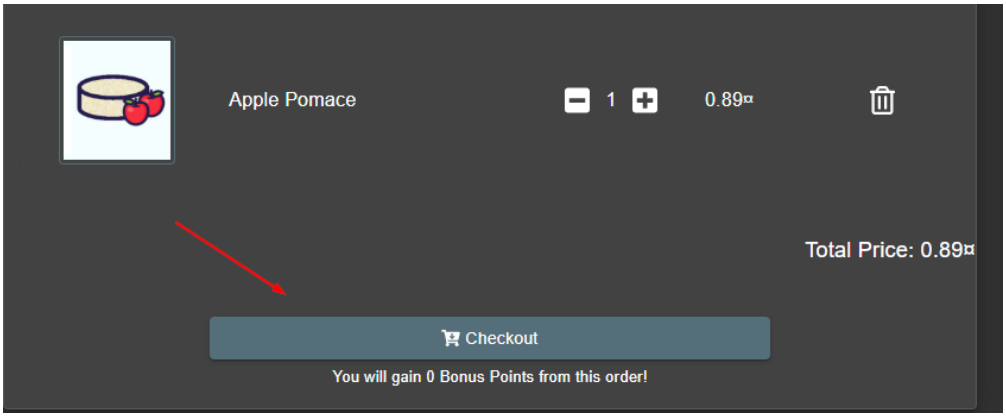
4. Go to settings and change the time to any day between (1feb2023→28feb2023)

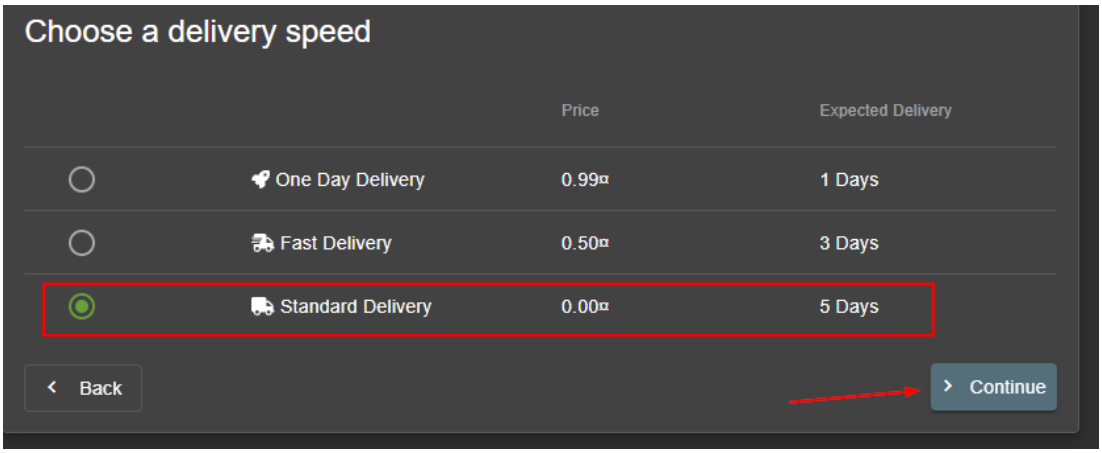


5. Login with any valid account , add products to cart and go to your basket

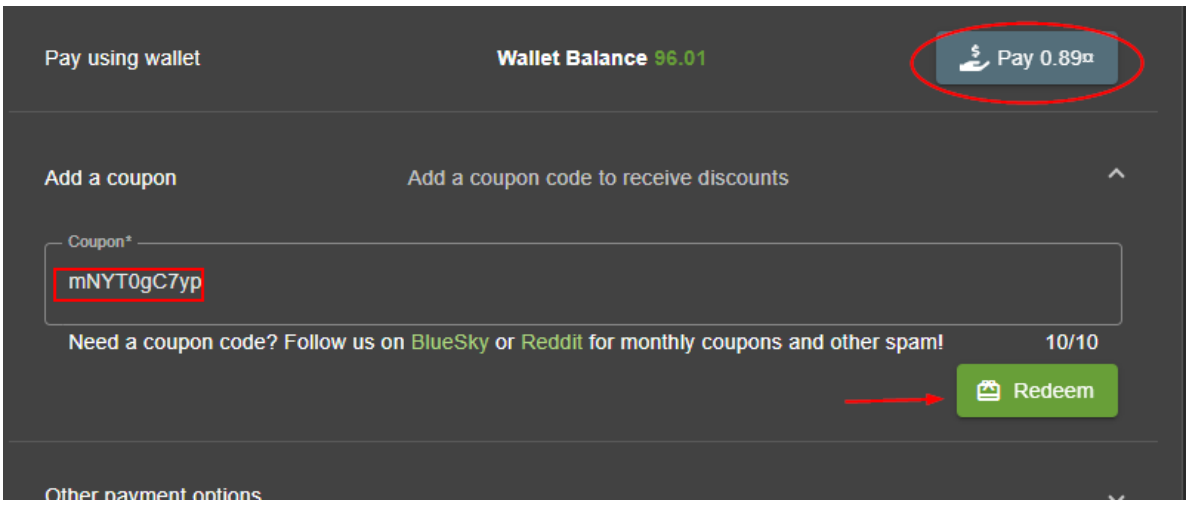


6. Check out and continue

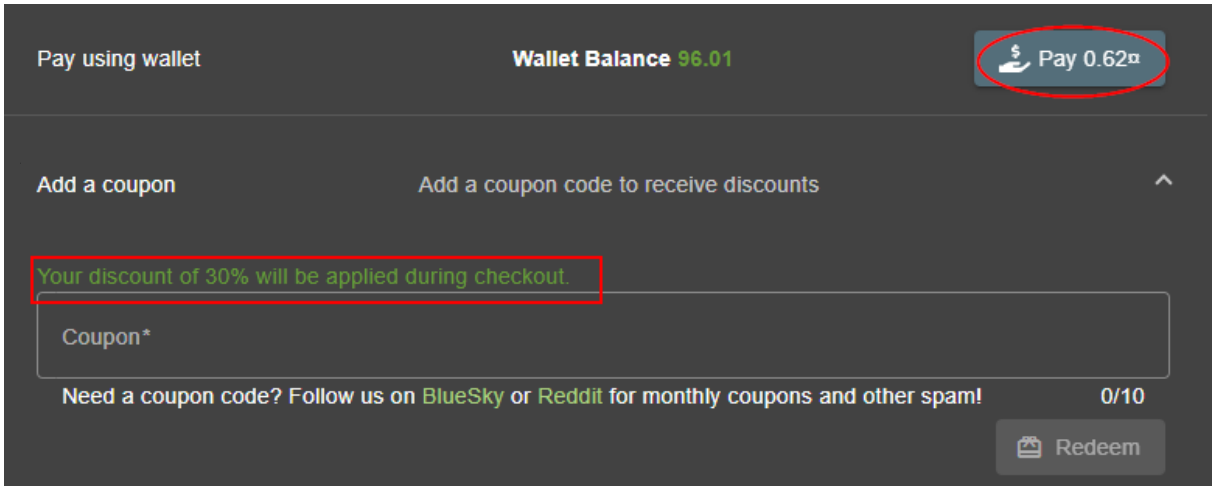




7. Write the coupon code(mNYT0gC7yp) and redeem



8. The discount applied successfully



You can notify that before applying coupon you will pay 0.89$ after applying coupon you will pay 0.62$

6.9 Deprecated & Insecure Interfaces

6.9.1 Deprecated Interface

HIGH

Description:

The application exposes a deprecated **B2B XML interface** through the complaint submission form. This interface was not properly disabled or secured. The penetration tester discovered that the file input field on the complaints page accepts **arbitrary file types**, including .xml files — which were successfully uploaded and processed without restrictions. This behavior indicates that legacy functionality meant for B2B integration is still active and accepting input.

Impact:

- **Interface Misuse:** penetration testers can upload XML files to test for further XML-based vulnerabilities like **XXE (XML External Entity)** .
- **Legacy Exploitation:** Deprecated features often lack proper security hardening, making them a common target for penetration testers .
- **Unintended Behavior:** Unrestricted file type acceptance without proper validation or usage context can lead to privilege escalation or sensitive data access.

Vulnerability Location:

- **Endpoint:** http://127.0.0.1:3000/#/complain
- **Component:** File input field for complaint submissions
- **Accepted File Type:** .xml
- **Behavior:** Legacy XML processing occurs, indicating active deprecated B2B interface

---

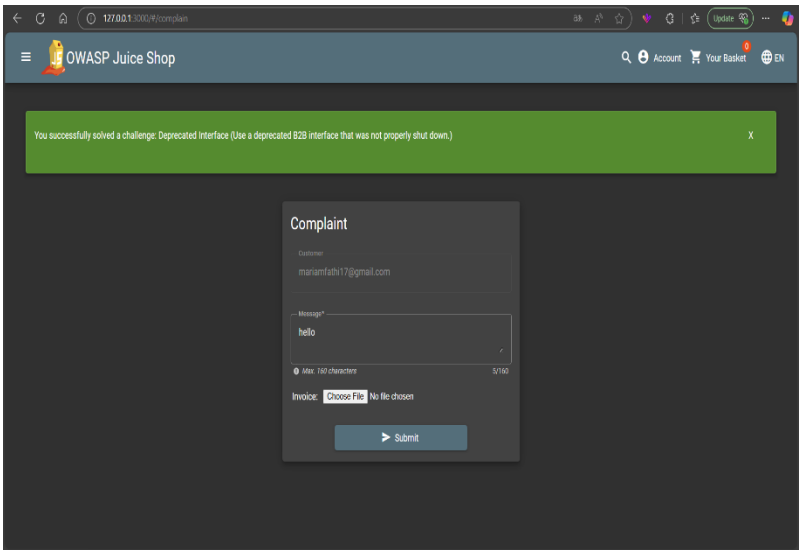
Recommendations:

1. **Disable Deprecated Interfaces:**
    - Immediately remove or disable any legacy endpoints that are no longer in use.

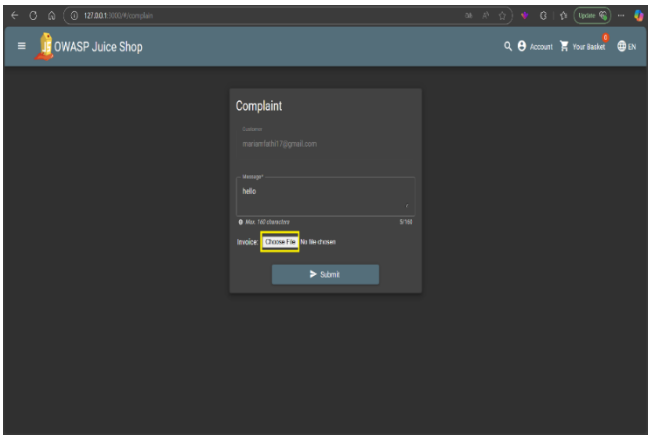
2. Enforce File Type Validation:
    - Restrict accepted file types to only those necessary (e.g., .jpg, .png, .pdf).
    - Reject unsupported or potentially dangerous formats like .xml.
3. Apply Security Controls to File Uploads:
    - Validate and sanitize all uploaded files.
    - Store uploads in isolated directories with no execution permissions.

Proof of Concept (PoC):

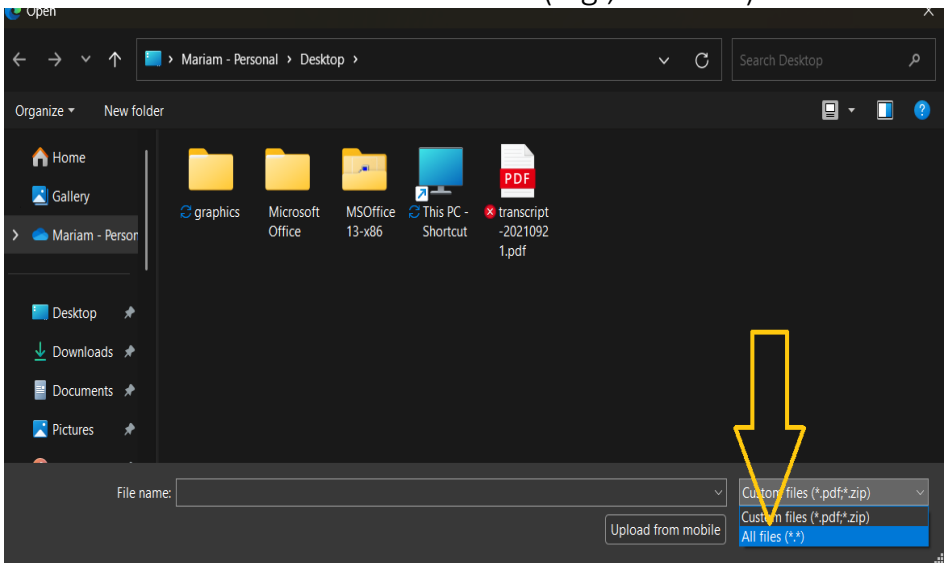
1. Go to the Complaints page of the application.

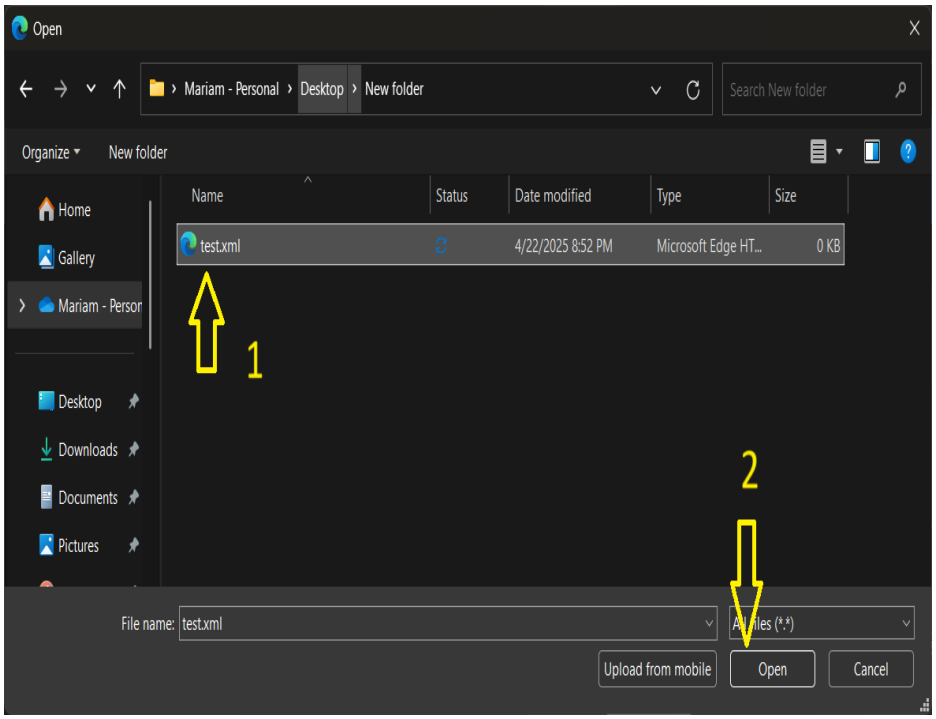


2. Click on the "Choose File" button and modify the file dialog settings to show all file types.

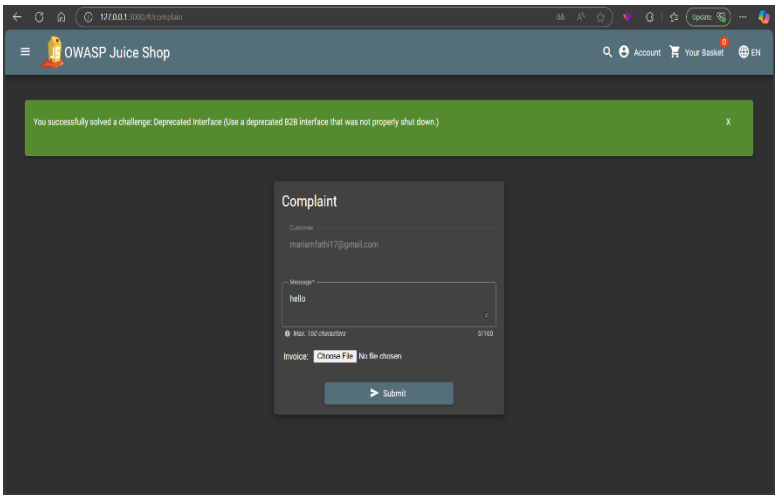


3. Select a file with the extension .xml (e.g., test.xml).





4. Upload the file via the form — the system accepts the file without restriction or validation, confirm that the upload succeeds, indicating that the deprecated XML-based interface is still active and capable of processing or storing such files.



6.10 Client-Side Manipulation & UI Tampering

6.10.1 Mass Dispel

LOW

Description:

The application displays multiple "Challenge Solved" notifications as toast messages, but it lacks proper control over how these elements can be manipulated on the client side. A user is able to use browser developer tools (DevTools) to interact directly with the DOM and close all notifications programmatically using JavaScript. This bypasses the intended manual interaction for dismissing each notification.

Impact:

- This reveals that important UI elements (such as notifications) can be manipulated or removed entirely by the user without restrictions.
- If critical UI components can be easily modified or removed from the front end, it could hint at other client-side manipulation possibilities in the application.
- While the impact is relatively low, it exposes poor client-side protection and is a reminder that important logic should not rely solely on front-end enforcement.

Vulnerability Location:

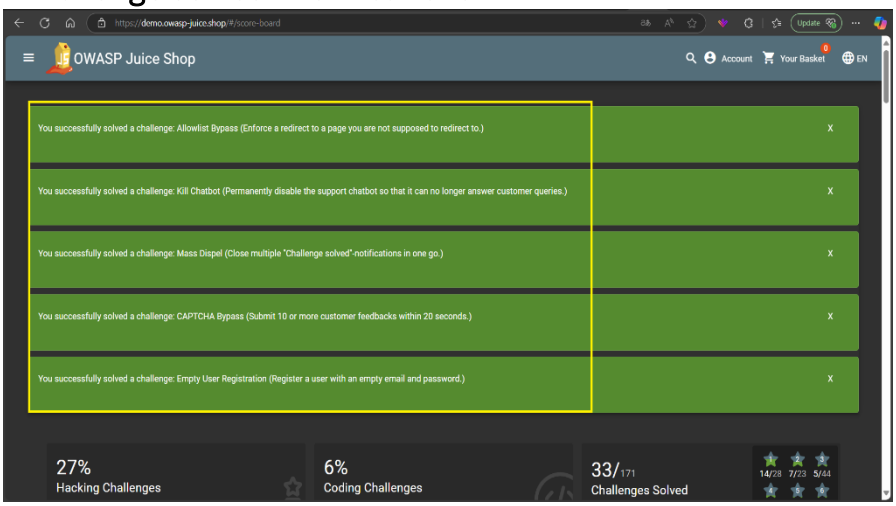
- Path: http://127.0.0.1:3000/#/score-board
- Element Selector: .notificationMessage within the DOM
- Affected Component: Angular component with tag br _ngcontent-ng-c1692636880

Recommendations:

1. Prevent DOM Manipulation for Critical Elements:
    - Ensure that UI elements that carry application logic or status are protected on the backend and are not purely visual.
    - Do not rely solely on front-end notifications to confirm important events like challenge completions.
2. Use Secure Design for Notifications:
    - Notifications should not reveal too much information.
    - Consider using obfuscation or other checks to limit direct DOM access and manipulation.

Proof of Concept:

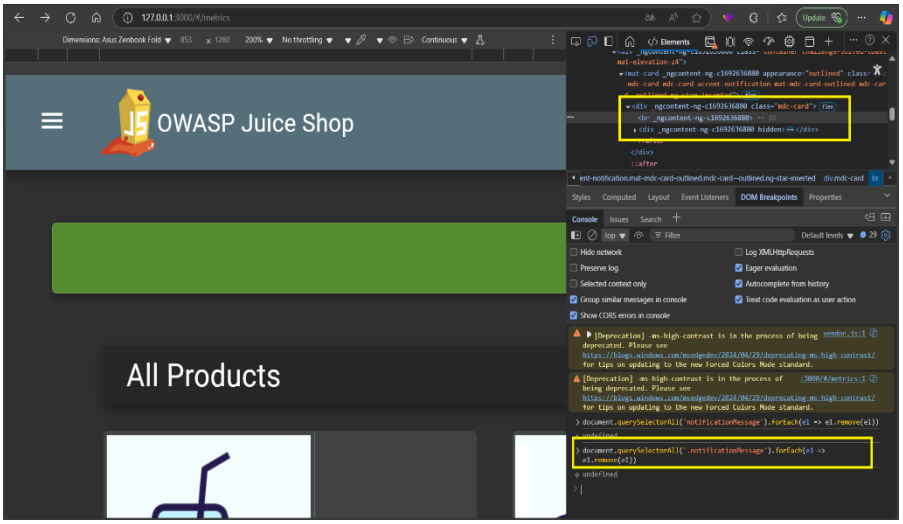
1- Open the OWASP Juice Shop and solve a few challenges to trigger multiple “Challenge solved” notifications.



2-Press F12 or right-click → Inspect to open the browser Developer Tools.

-Navigate to the Console tab.

-Paste and execute the following JavaScript code:
document.querySelectorAll('.notificationMessage').forEach(el => el.remove());



6.11Broken Anti-Automation

6.11.1 CAPTCHA Bypass

HIGH

Description:

The application implements a CAPTCHA mechanism on the customer feedback form to
prevent bots and automation. However, the CAPTCHA validation is **only superficial on the client-side** without proper verification on the server-side.
An penetration tester can **submit multiple feedback entries** by **reusing the same CAPTCHA value** without solving a new one each time.
This bypass is possible because the server does not validate whether the CAPTCHA was freshly generated and solved for each submission, allowing mass automated submissions.

Impact:

- **Automated Abuse:** penetration testers can flood the feedback system with automated submissions.
- **Resource Exhaustion:** Potential DoS (Denial of Service) by overwhelming the system with requests.
- **Integrity Issues:** Legitimate feedback may be lost or diluted among spam.
- **Reduced Trust:** CAPTCHA is expected to provide basic protection; its failure undermines user trust.

Vulnerability Location:

- **Component:** Customer Feedback Form
- **Parameter Affected:** CAPTCHA Value
- **Behavior:** Reusing the same CAPTCHA token multiple times allows multiple submissions.

Recommendations:

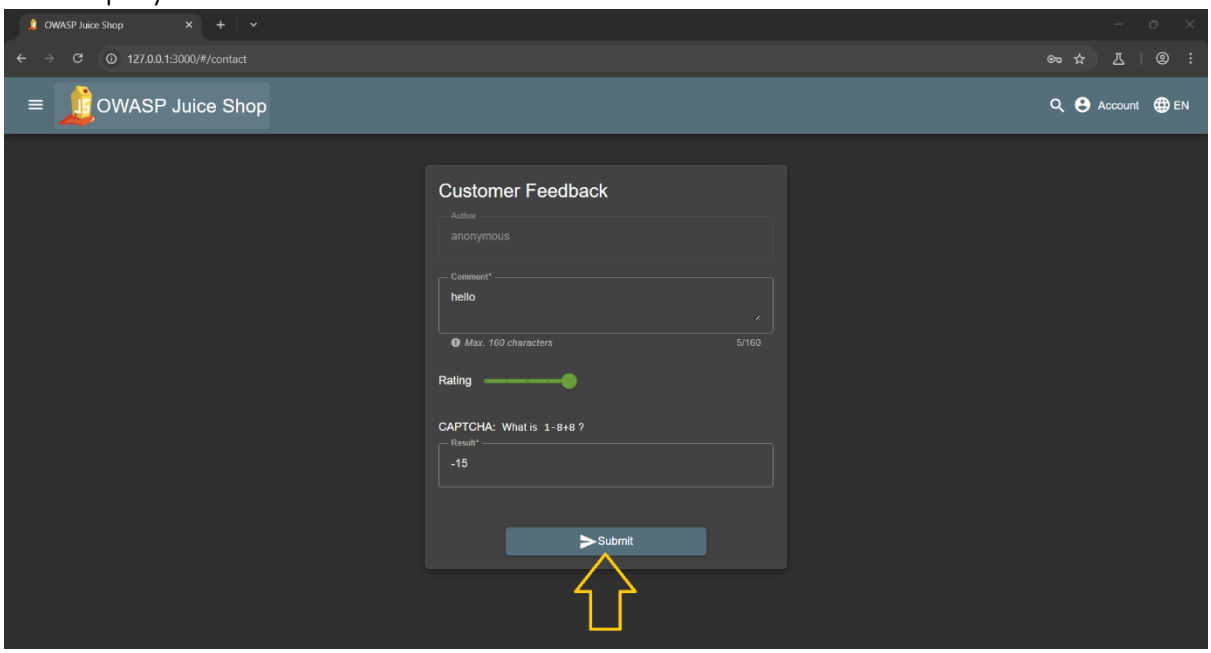
1. **Server-Side CAPTCHA Validation:**

- Implement strict server-side validation to ensure each CAPTCHA is verified **once per session** and **per submission**.

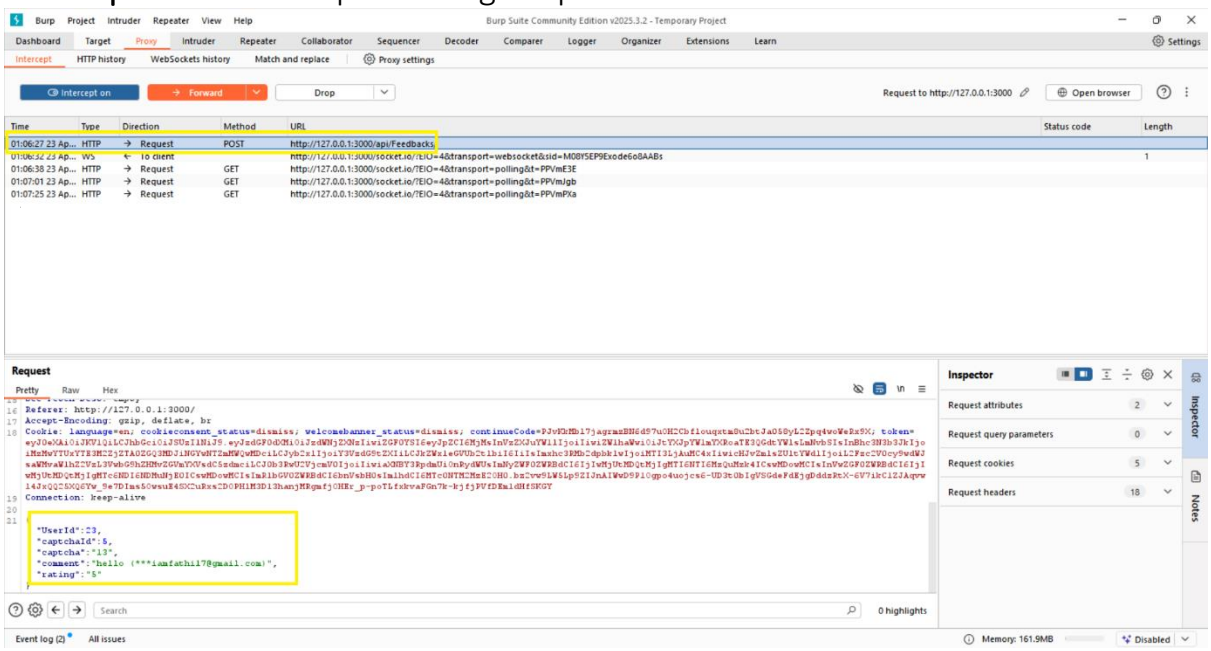
2. **Token Invalidation After Use:**
   - Once a CAPTCHA is solved and used for a submission, **invalidate** it immediately to prevent reuse.
3. **More Robust CAPTCHA Systems:**
   - Consider using more dynamic and secure CAPTCHA services (e.g., reCAPTCHA v2/v3).

Proof of Concept (PoC):

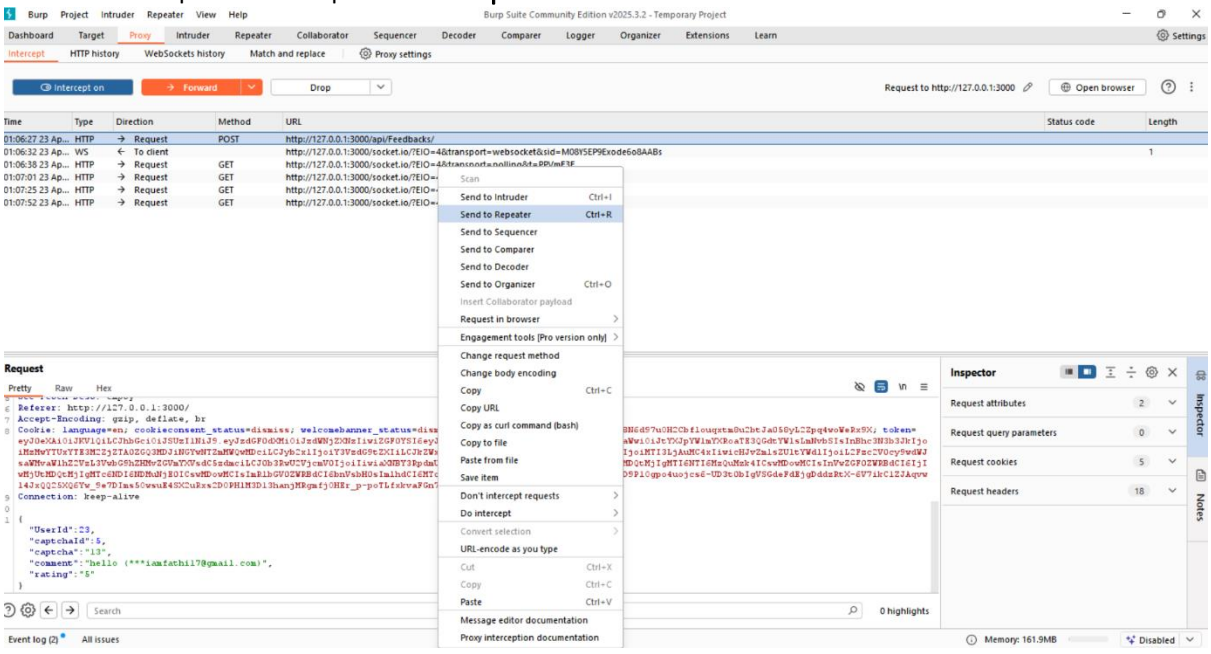
1. **open** the Customer Feedback page and **Fill** the feedback form and solve the displayed CAPTCHA.



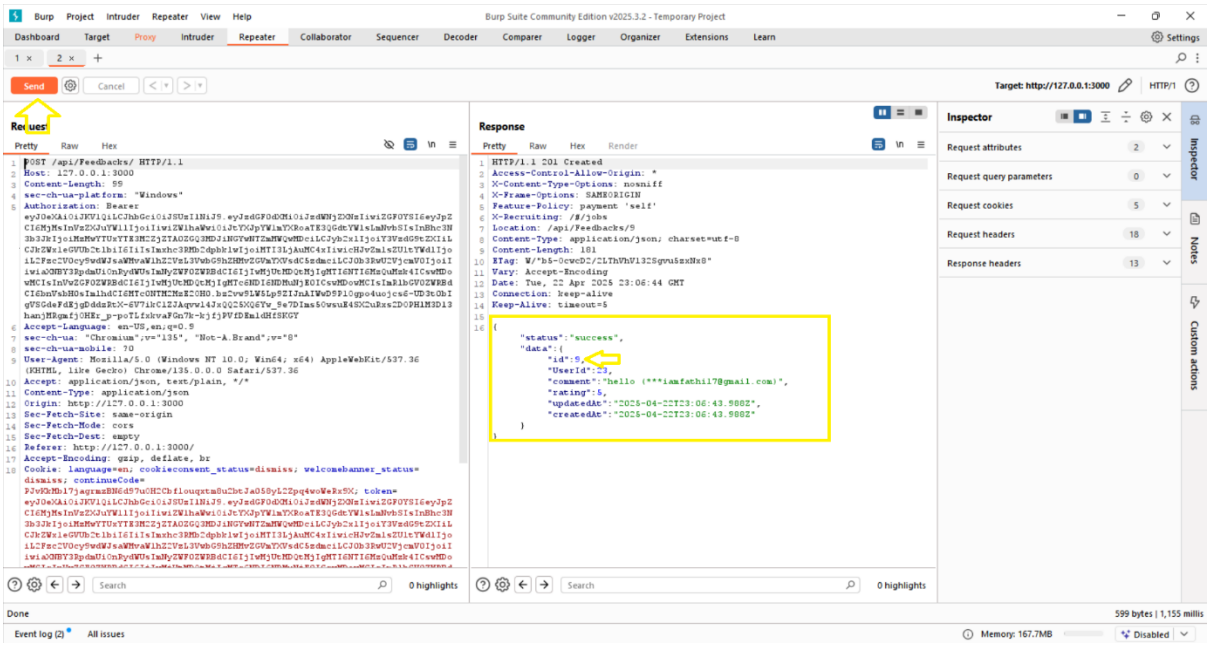
2. **Intercept** the POST request using Burp Suite.

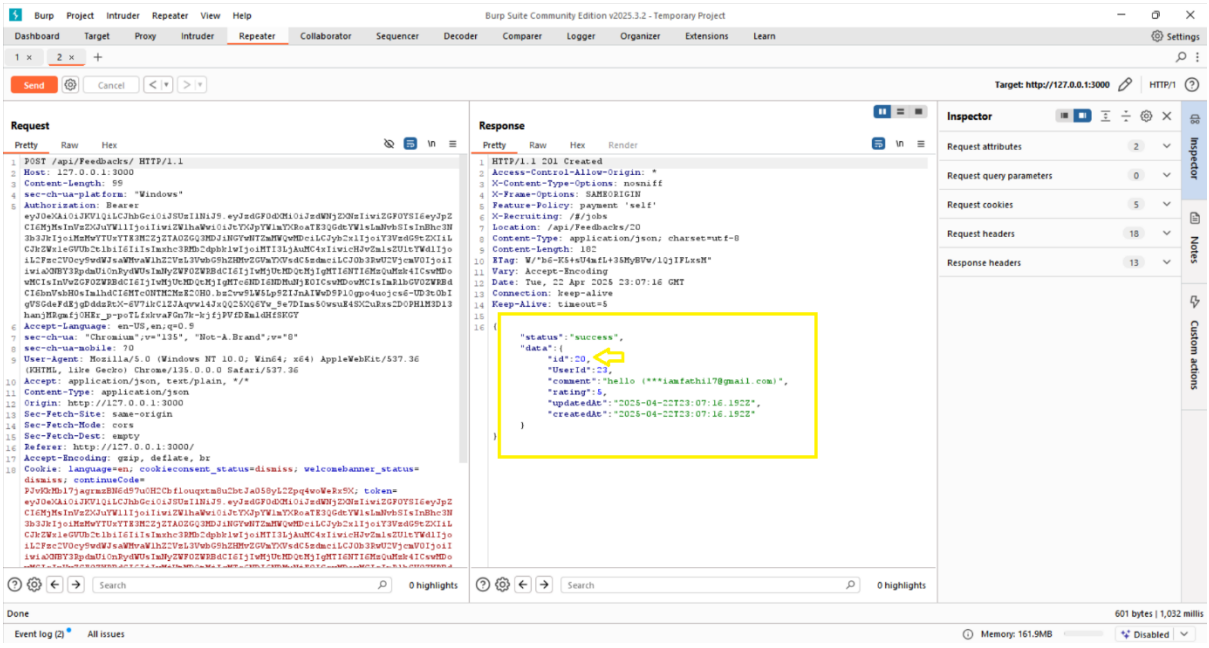


3. **Send** the captured request to **Repeater**.

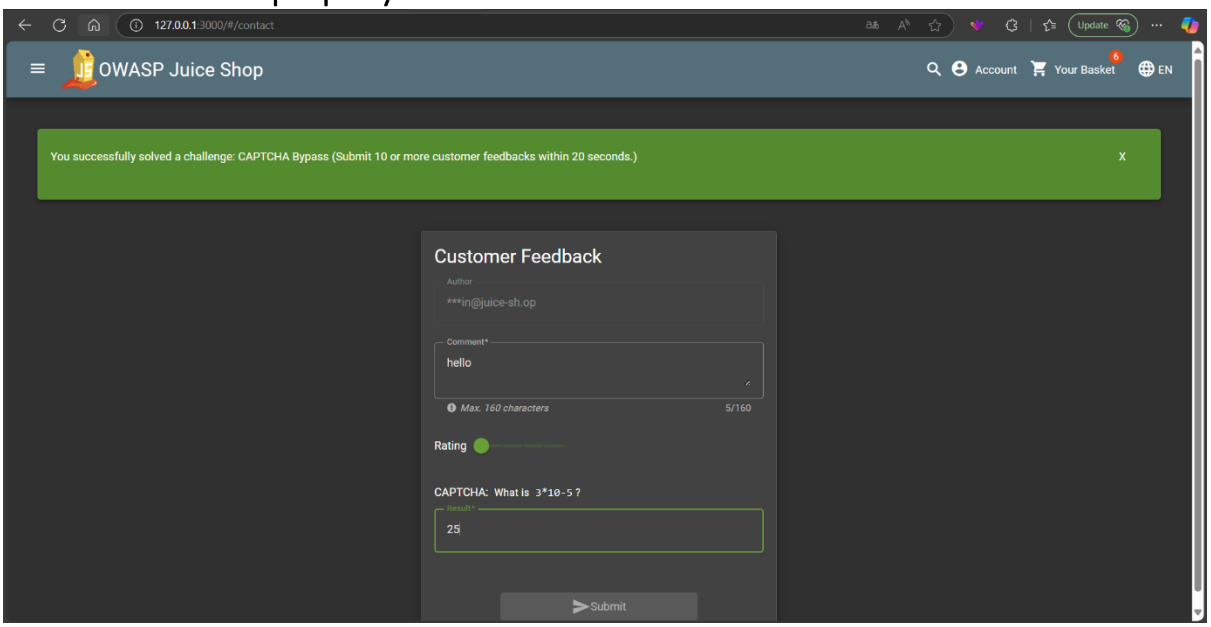


4. **Send** the same request multiple times (10+ requests) **without solving new CAPTCHAs**.


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.3.2 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
Send Cancel
Target: http://127.0.0.1:3000
HTTP/1
Request
Pretty Raw Hex
POST /api/Feedbacks/ HTTP/1.1
Host: 127.0.0.1:3000
Content-Length: 99
sec-ch-ua-platform: "Windows"
Authorization: Bearer
Content-Type: application/json
Origin: http://127.0.0.1:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_status=
dismiss; continueCode=
Response
Pretty Raw Hex Render
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: /api/Feedbacks/9
Content-Type: application/json; charset=utf-8
Content-Length: 181
ETag: W/"b5-0cwcD2/ZLThVhV13ZSgvu5zxNx8"
Vary: Accept-Encoding
Date: Tue, 22 Apr 2025 23:06:44 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{
"status":"success",
"data":{
"id":9,
"UserId":23,
"comment":"hello (***iamfathi17@gmail.com)",
"rating":5,
"updatedAt":"2025-04-22T23:06:43.988Z",
"createdAt":"2025-04-22T23:06:43.988Z"
}
}
Inspector
Request attributes 2
Request query parameters 0
Request cookies 5
Request headers 18
Response headers 13
Inspector
Notes
Custom actions
0 highlights
Done
599 bytes | 1,155 millis
Event log (2)
All issues
Memory: 167.7MB
Disabled


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.3.2 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
Send Cancel
Target: http://127.0.0.1:3000
HTTP/1
Request
Pretty Raw Hex
POST /api/Feedbacks/ HTTP/1.1
Host: 127.0.0.1:3000
Content-Length: 99
sec-ch-ua-platform: "Windows"
Authorization: Bearer
Content-Type: application/json
Origin: http://127.0.0.1:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_status=
dismiss; continueCode=
Response
Pretty Raw Hex Render
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: /api/Feedbacks/20
Content-Type: application/json; charset=utf-8
Content-Length: 182
ETag: W/"b6-K4+sU4mfL+36MyBVw/1QjIFLxsM"
Vary: Accept-Encoding
Date: Tue, 22 Apr 2025 23:07:16 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{
"status":"success",
"data":{
"id":20,
"UserId":23,
"comment":"hello (***iamfathi17@gmail.com)",
"rating":5,
"updatedAt":"2025-04-22T23:07:16.192Z",
"createdAt":"2025-04-22T23:07:16.192Z"
}
}
Inspector
Request attributes 2
Request query parameters 0
Request cookies 5
Request headers 18
Response headers 13
Inspector
Notes
Custom actions
0 highlights
Done
601 bytes | 1,032 millis
Event log (2)
All issues
Memory: 161.9MB
Disabled

All feedback submissions are accepted successfully, confirming the CAPTCHA was not validated properly.



6.12 Unvalidated Redirects

612.1Outdated Allowlists

MEDIUM

- Description:
  The application uses an outdated allowlist for redirect destinations, which can be bypassed to redirect users to external, potentially malicious websites. This flaw allows penetration tester to exploit redirection logic and trick users into visiting unintended or harmful destinations.

- Impact:
  A penetration tester was able to craft a URL that redirects users to a malicious external site. This can be used for phishing, malware delivery, or social engineering attacks that appear to come from a trusted source.

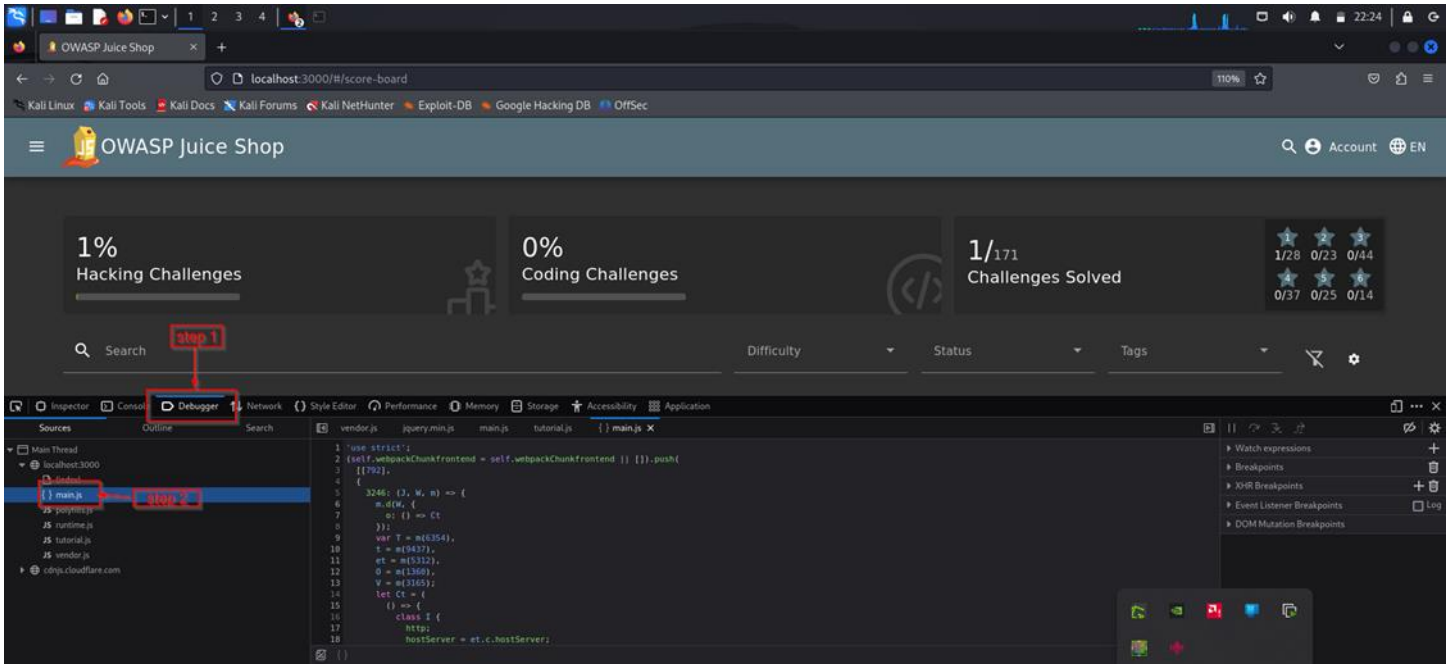
- Vulnerability Location:
  Found in main.js – redirection behaviour tied to the term “blockchain”

- CVE Reference:

- Recommendations:
  - Use a validated list of approved redirect destinations.
  - Never allow user input to control the destination of redirects unless it is strictly validated.
  - Implement server-side validation and enforce it rather than relying solely on client-side checks.

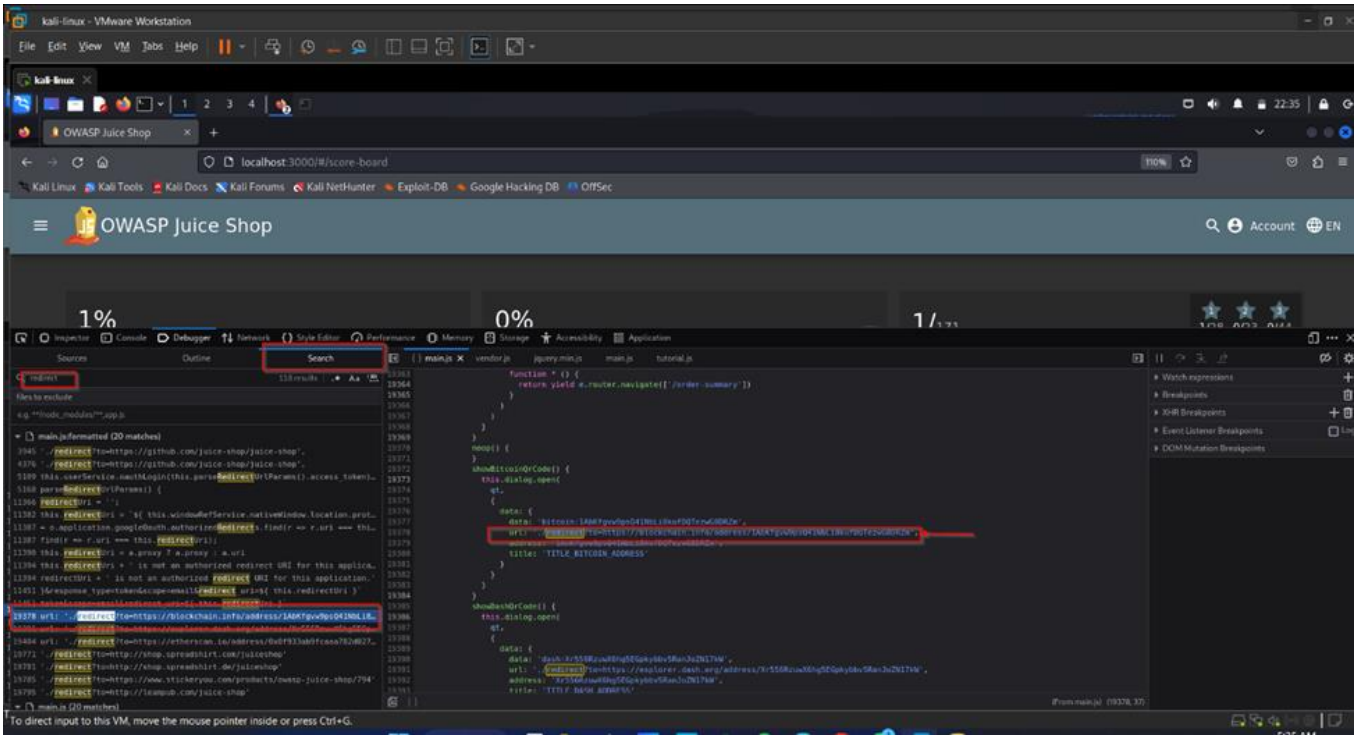
- Proof Of Concept:

1. Open the target application in a browser and Press F12 to open Developer Tools and go to the Debugger tab.



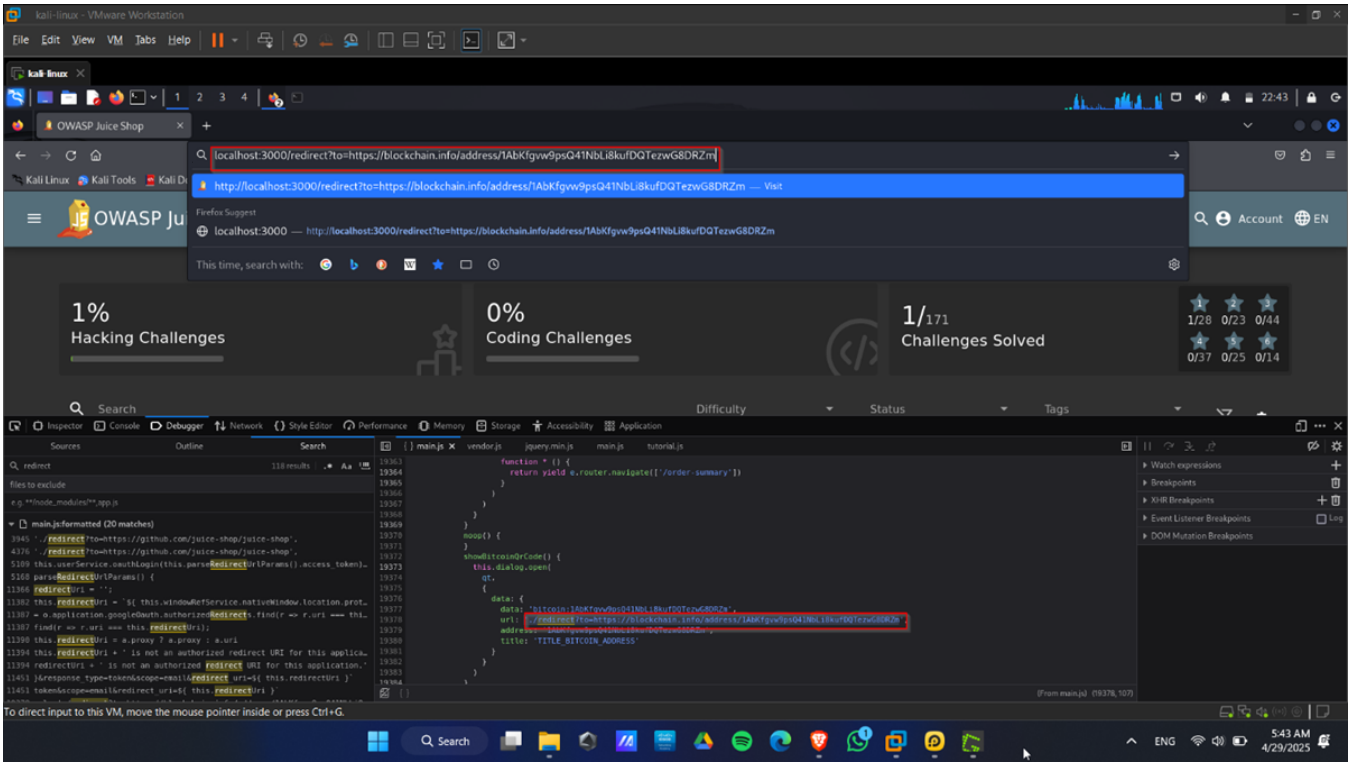
Locate the main.js file in the application's source code.

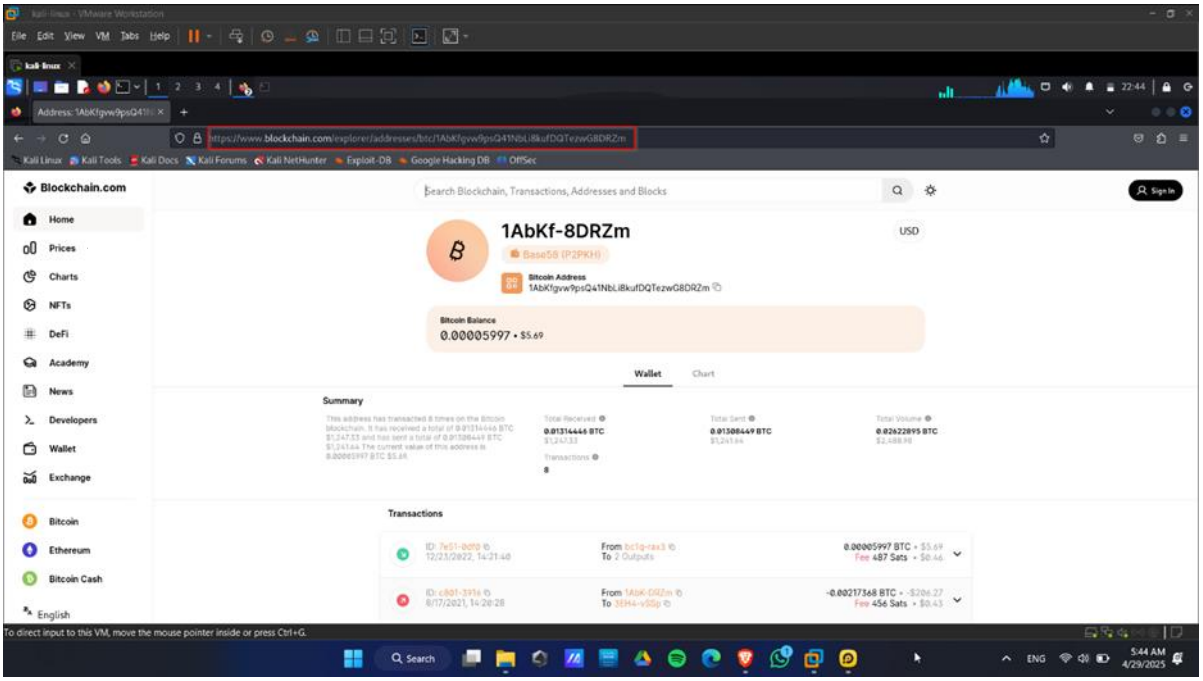
2.Use the search to look for the keyword "redirect" and focus on code sections involving "blockchain" these often contain logic responsible for external redirection.



3. Find the redirect URL or endpoint pointing to a blockchain/crypto site.

4. Copy the redirect link and paste it into the browser’s address bar.





The redirect works without validation which means that the application still allows redirects to such domains without proper checks using an outdated allowlist this can expose users to potential phishing or malicious redirection attacks.

6.13 Sensitive Data Exposure

6.13.1 Meta Geo Stalking

MEDIUM

- Description:
  we exploited metadata embedded in an image uploaded by the user "John". The password reset feature uses a security question related to his favourite hiking location, which can be discovered through the photo's metadata. By analysing the photo, we extracted GPS coordinates that led us to the answer needed to bypass the security question and reset John's account password.

- Impact:
  A penetration tester was able to gain unauthorized access to John's account by extracting sensitive metadata (location) from a public image and using it to answer the password reset security question. This demonstrates a real-world risk where users unknowingly expose sensitive data through media uploads.

- Vulnerability Location:
  - Reset Password Page: /#/forgot-password

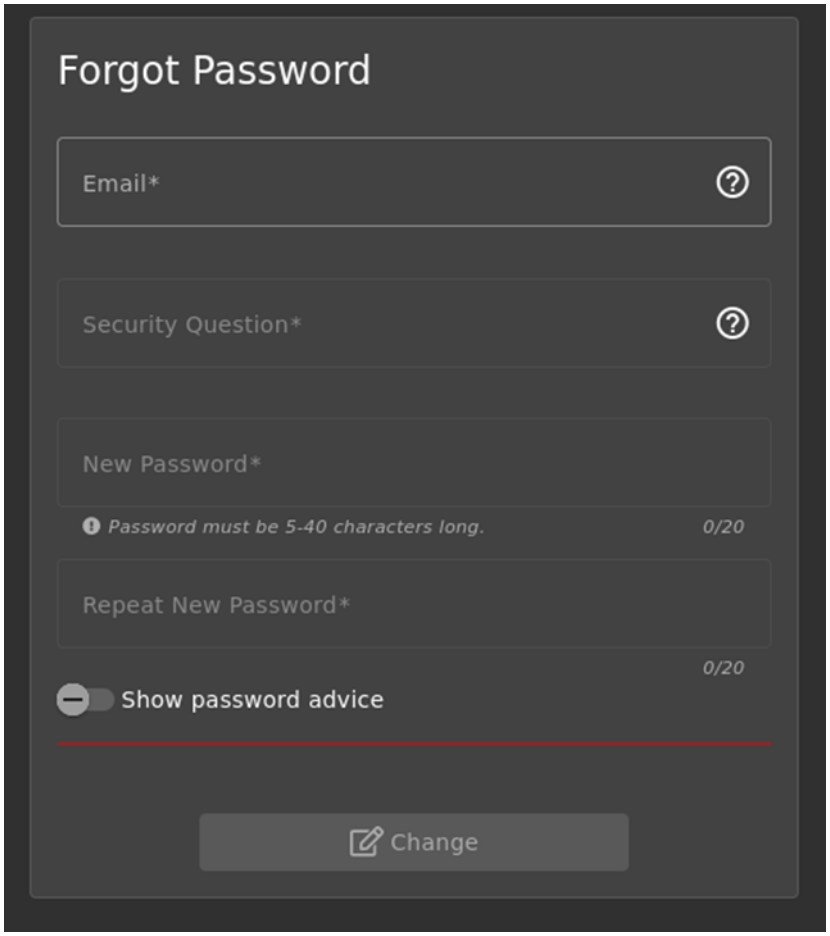
- CVE Reference:

- Recommendations:

- Strip all metadata (EXIF) from uploaded images server-side before making them public.
- Avoid using easily guessable or researchable security questions for account recovery.
- Implement rate-limiting and monitoring on password reset functionalities.
- Use multi-factor authentication for sensitive account actions.
- Prevent account reset workflows from disclosing whether an email is registered (i.e., don't show whether a security question appears or not based on input).

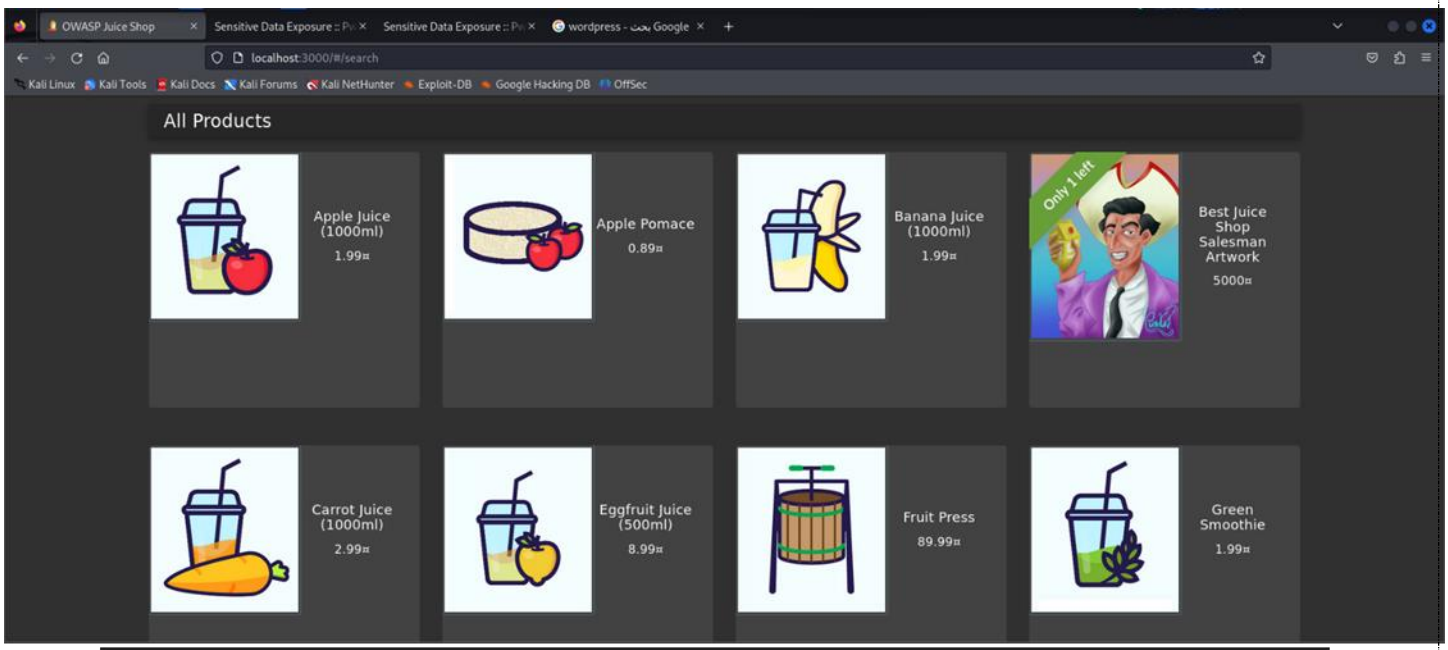
- Proof Of Concept:

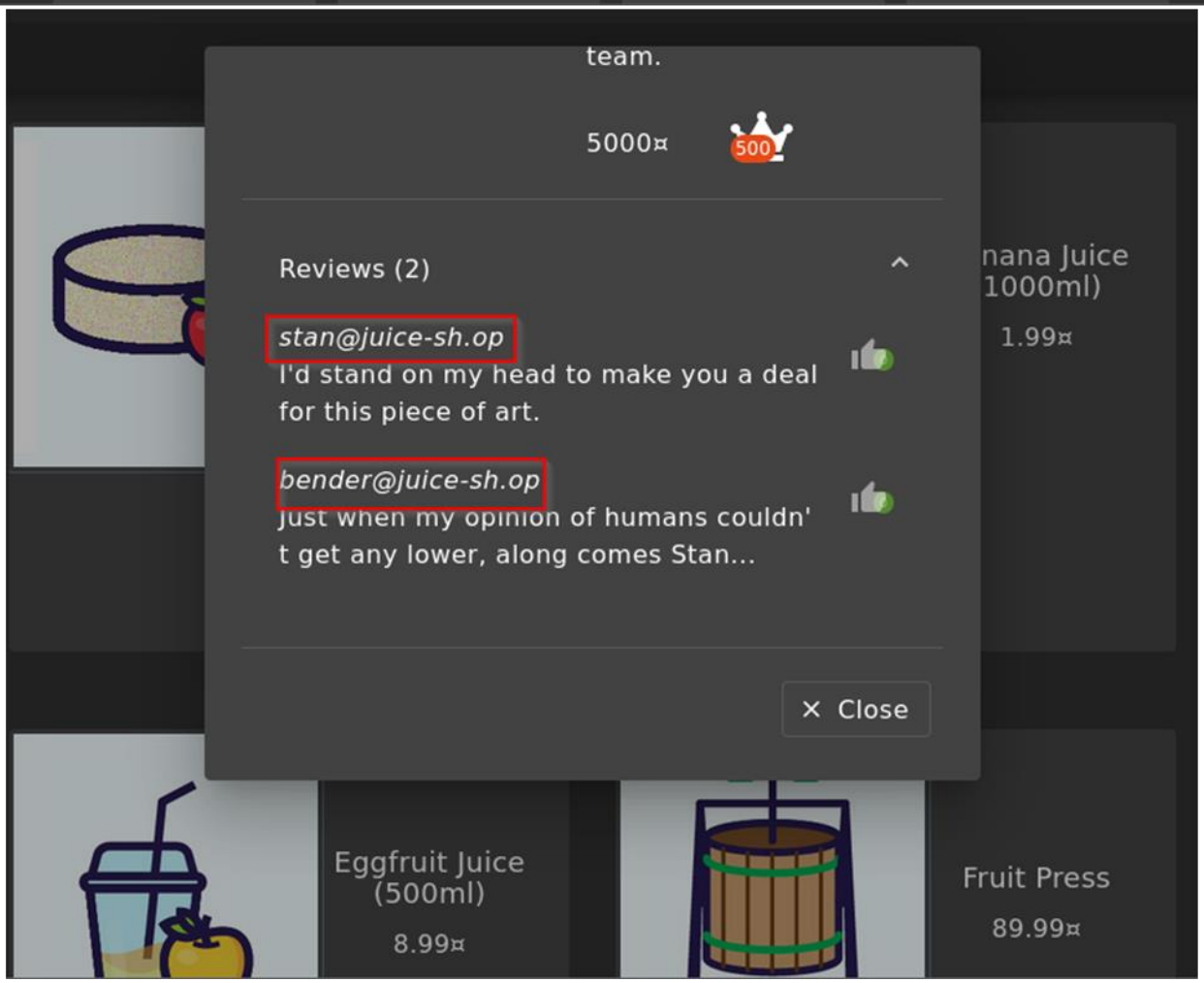
1.Go to the “Forgot Password” page.



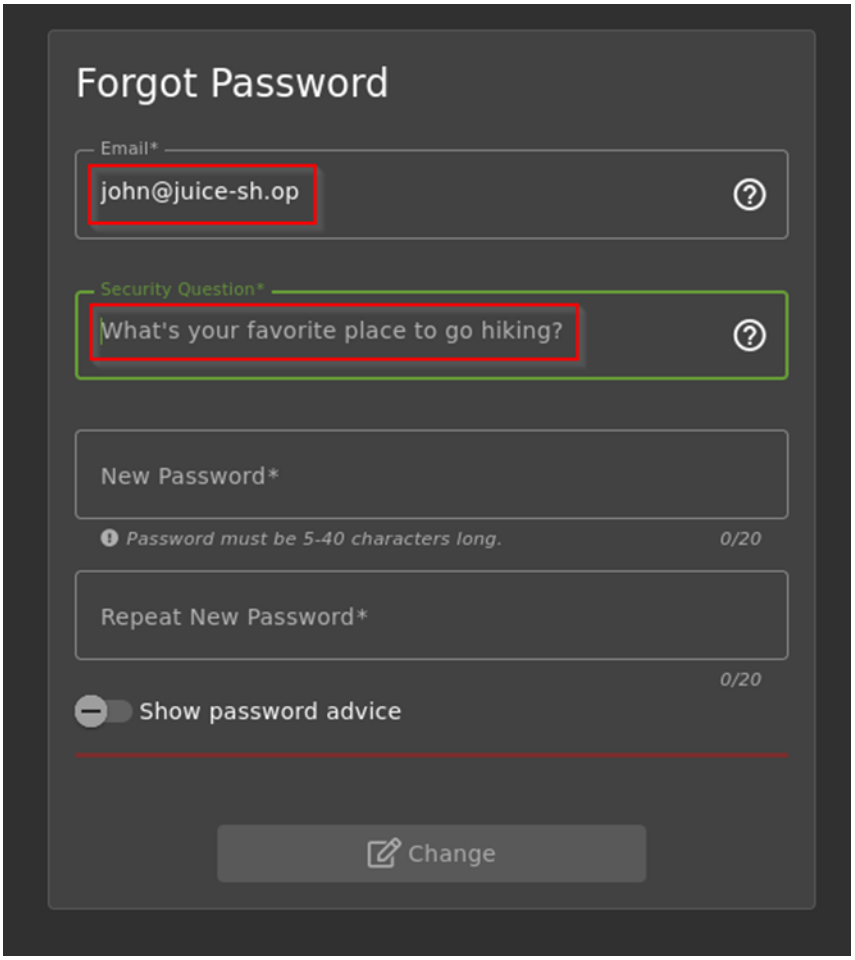
Observe the reset form asking for: Email address, Security question and new password.

2. To know John’s exact email, browse the website for clues (reviews, comments, feedback etc.).


OWASP Juice Shop
Sensitive Data Exposure
Sensitive Data Exposure
wordpress - Google
localhost:3000/#/search
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
All Products
Apple Juice (1000ml)
1.99¤
Apple Pomace
0.89¤
Banana Juice (1000ml)
1.99¤
Only 1 left
Best Juice Shop Salesman Artwork
5000¤
Carrot Juice (1000ml)
2.99¤
Eggfruit Juice (500ml)
8.99¤
Fruit Press
89.99¤
Green Smoothie
1.99¤


team.
5000¤
500
Reviews (2)
stan@juice-sh.op
I'd stand on my head to make you a deal for this piece of art.
bender@juice-sh.op
Just when my opinion of humans couldn't get any lower, along comes Stan...
× Close
nana Juice
1000ml)
1.99¤
Eggfruit Juice
(500ml)
8.99¤
Fruit Press
89.99¤

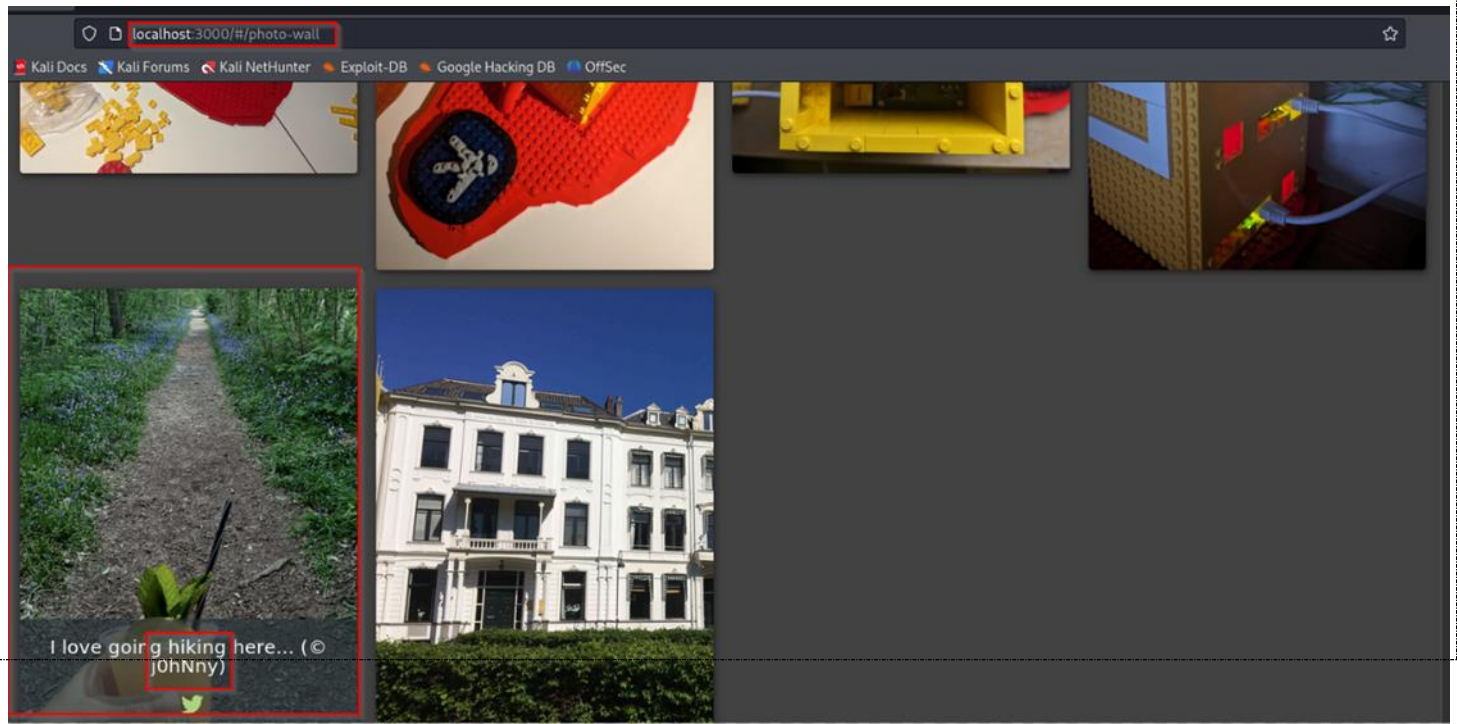
3. based on known patterns of existing user emails, I guessed that “John” email format: “john@juice-sh.op”



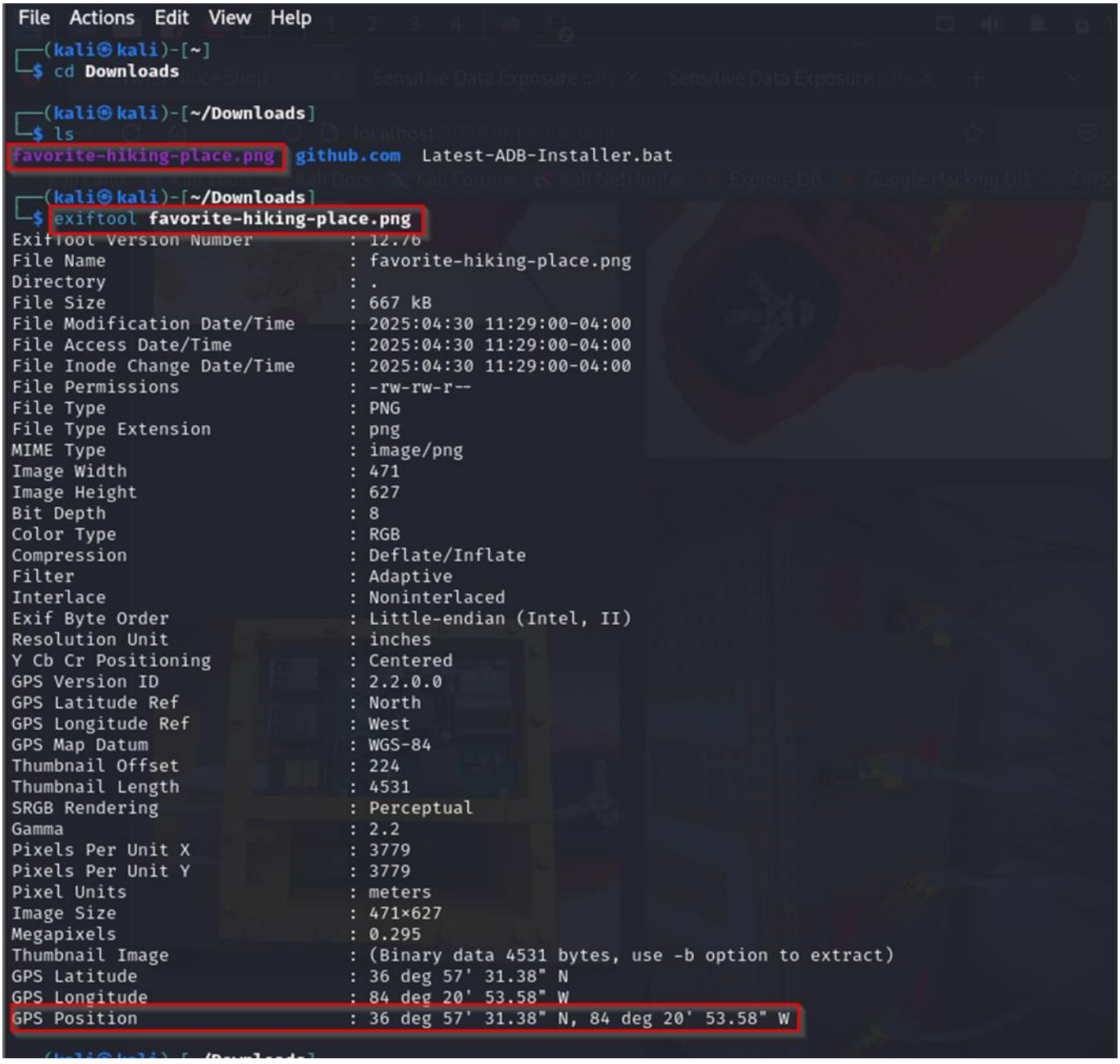
When entered, the security question appeared, confirming it’s valid and the correct email.

Security Question: “What’s your favourite hiking place?”

4. Search the website for posts made by John and found an image uploaded by John related to hiking in the Photo Wall section.

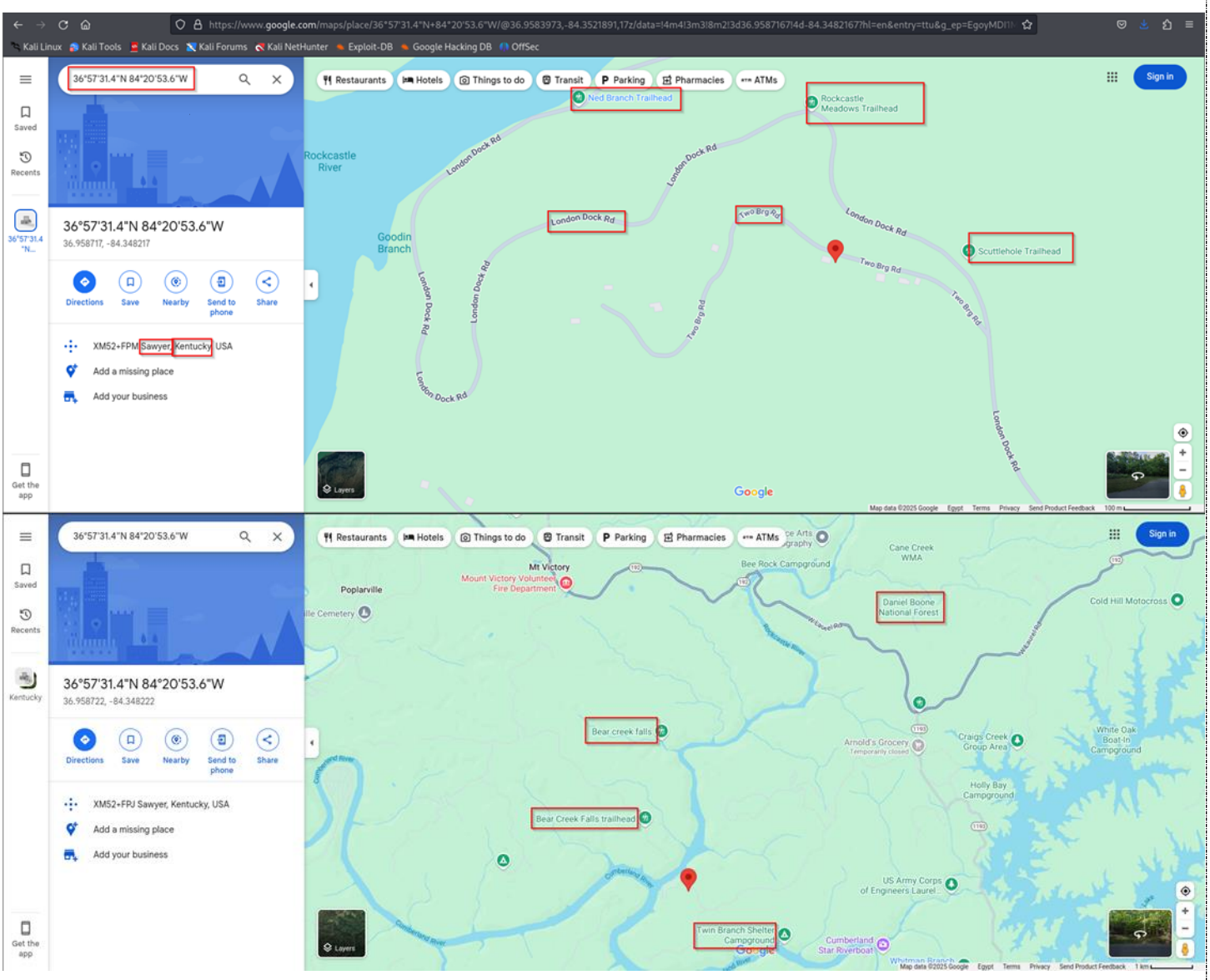


5. Download and analyze the image metadata Using the following command in Kali Linux: “exiftool favorite-hiking-place.png”.

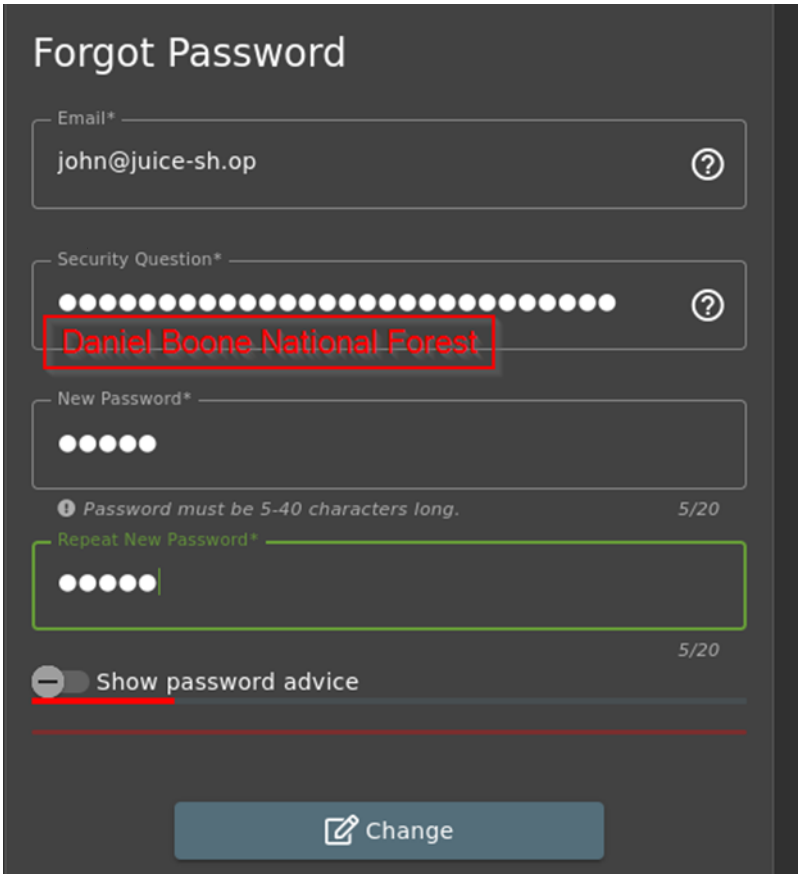


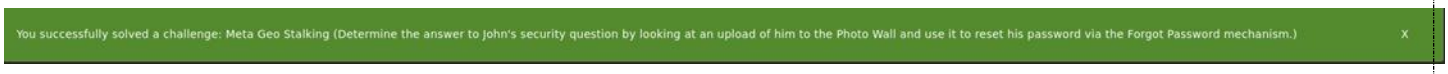
In the metadata: GPS coordinates “36°57’31.38″ N 84°20’53.58″ W”

6. Search for GPS coordinates on Google Maps to identify the actual location based on these coordinates.



7. Use this location name: “Daniel Boone National Forest” as the answer to the security question.





The penetration tester was able to successfully reset John's account password using the answer derived from the image metadata.

6.13.2 Bjoern's Favourite Pet

HIGH

- Description:
  This vulnerability involves resetting the password for the account of Bjoern by collecting publicly available information and exploiting a weak security question. The penetration tester gathers details such as the correct email and the answer to the security question (Bjoern's pet name) from website content and external sources, ultimately gaining unauthorized access to his account.

- Impact:
  A penetration tester was able to reset the password and gain access to Bjoern's account using public information, bypassing authentication by correctly guessing the answer to a security question based on exposed user data. This shows a failure in securely handling password reset mechanisms and user privacy.

- Vulnerability Location:
  - Reset Password Page: /#/forgot-password

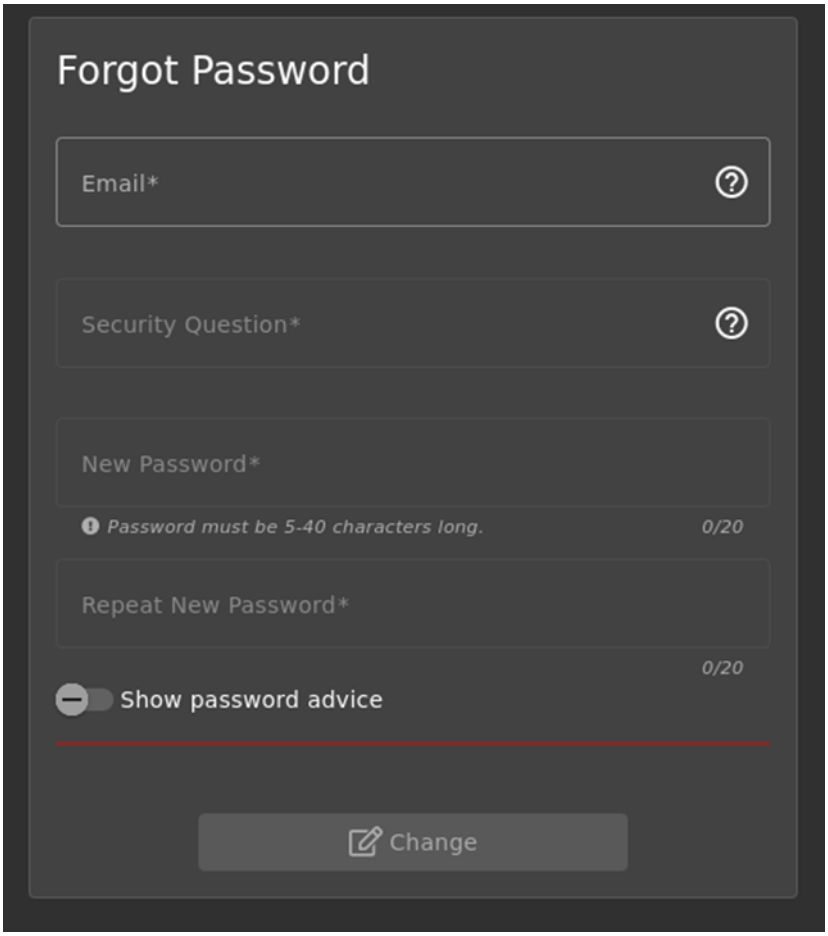
- CVE Reference:

- Recommendations:
  - Eliminate security questions as a method for password reset. Instead, use multi-factor authentication or token-based reset links.
  - Avoid using predictable or publicly available user data in authentication processes.
  - Enforce rate limiting and anomaly detection for password reset attempts.
  - Limit exposure of sensitive personal information on user profiles and reviews.

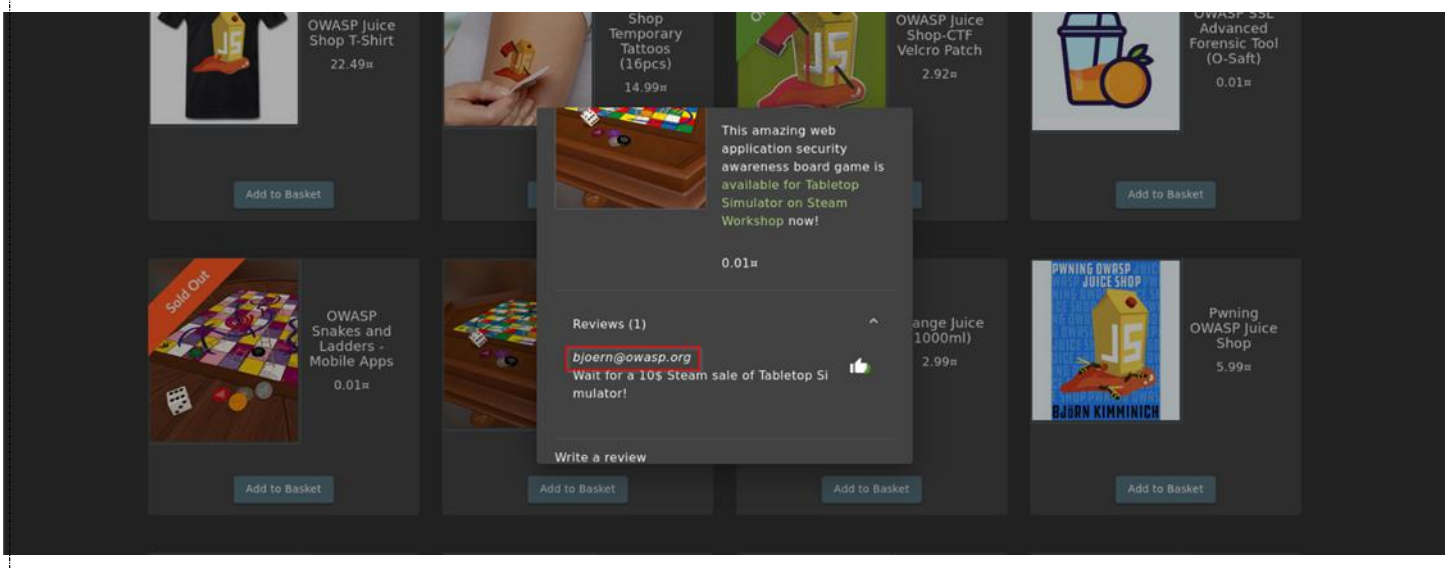
- Prevent account reset workflows from disclosing whether an email is registered (i.e., don't show whether a security question appears or not based on input).

- Proof Of Concept:

  1.Go to the "Forgot Password" page.

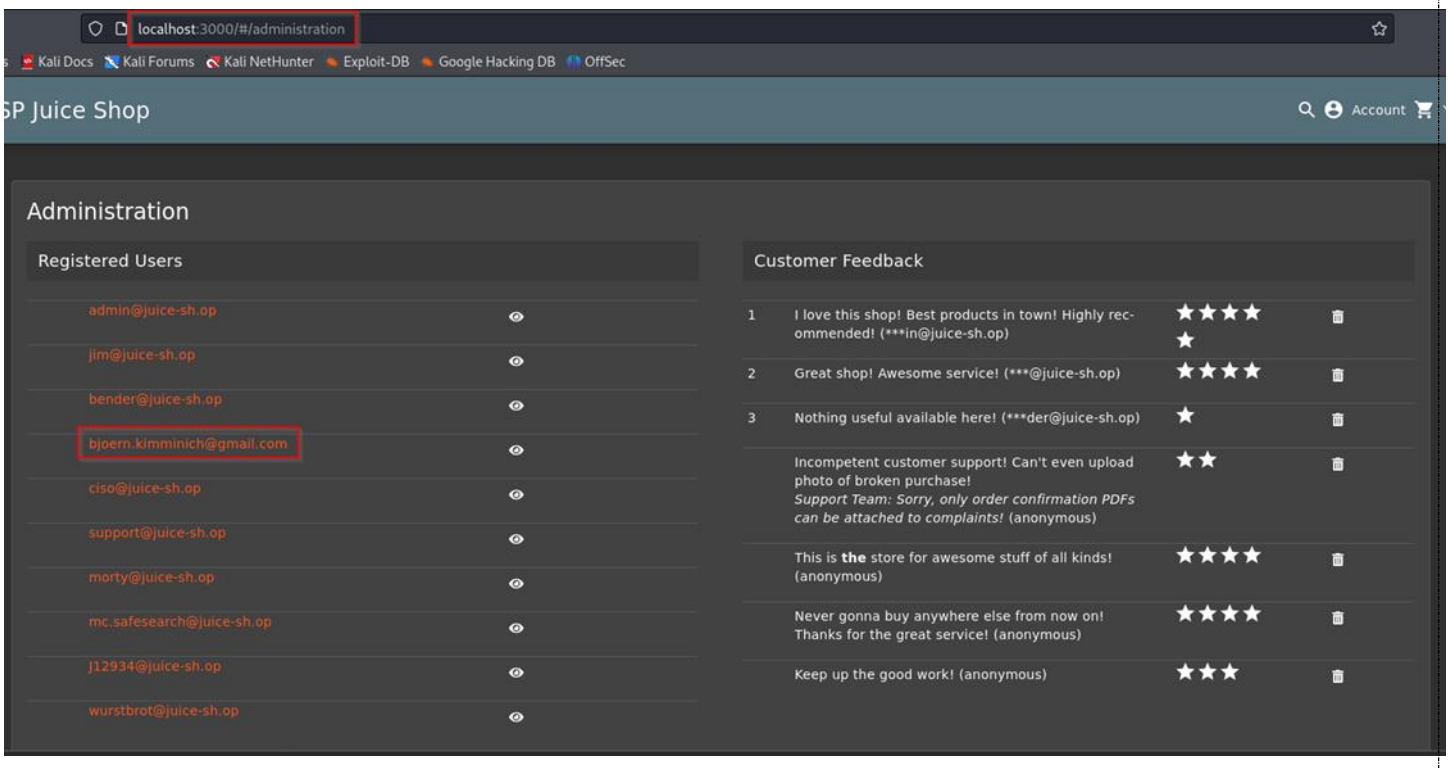


2. Search for Bjoern's email: Found a review written by Bjoern →

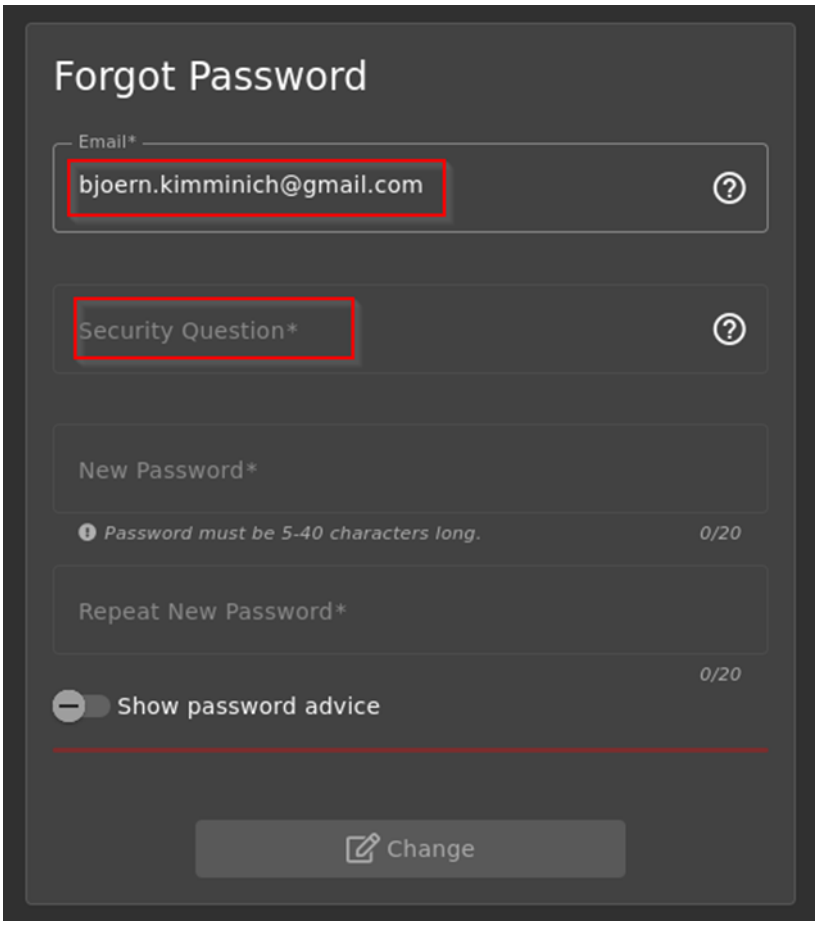


Identified email: bjoern@owasp.org

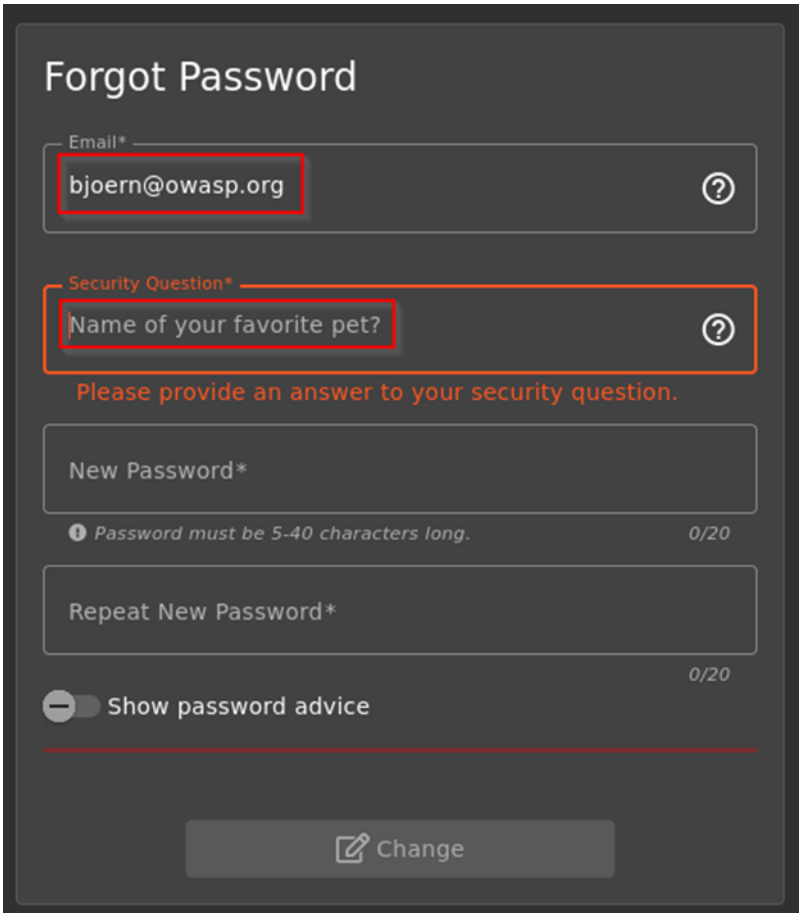
Using admin privileges (from a previous vulnerability) to find another email: “bjoern.kimminich@gmail.com”



3. Test both emails on the Reset Password page:
Tried bjoern.kimminich@gmail.com → No security question appeared.

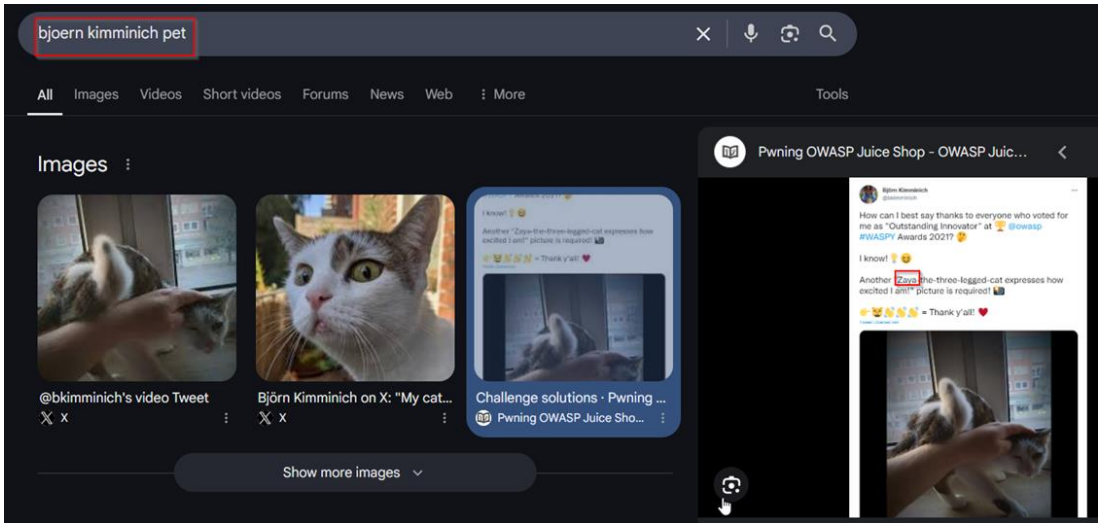


Tried bjoern@owasp.org → Security question appeared → valid email confirmed

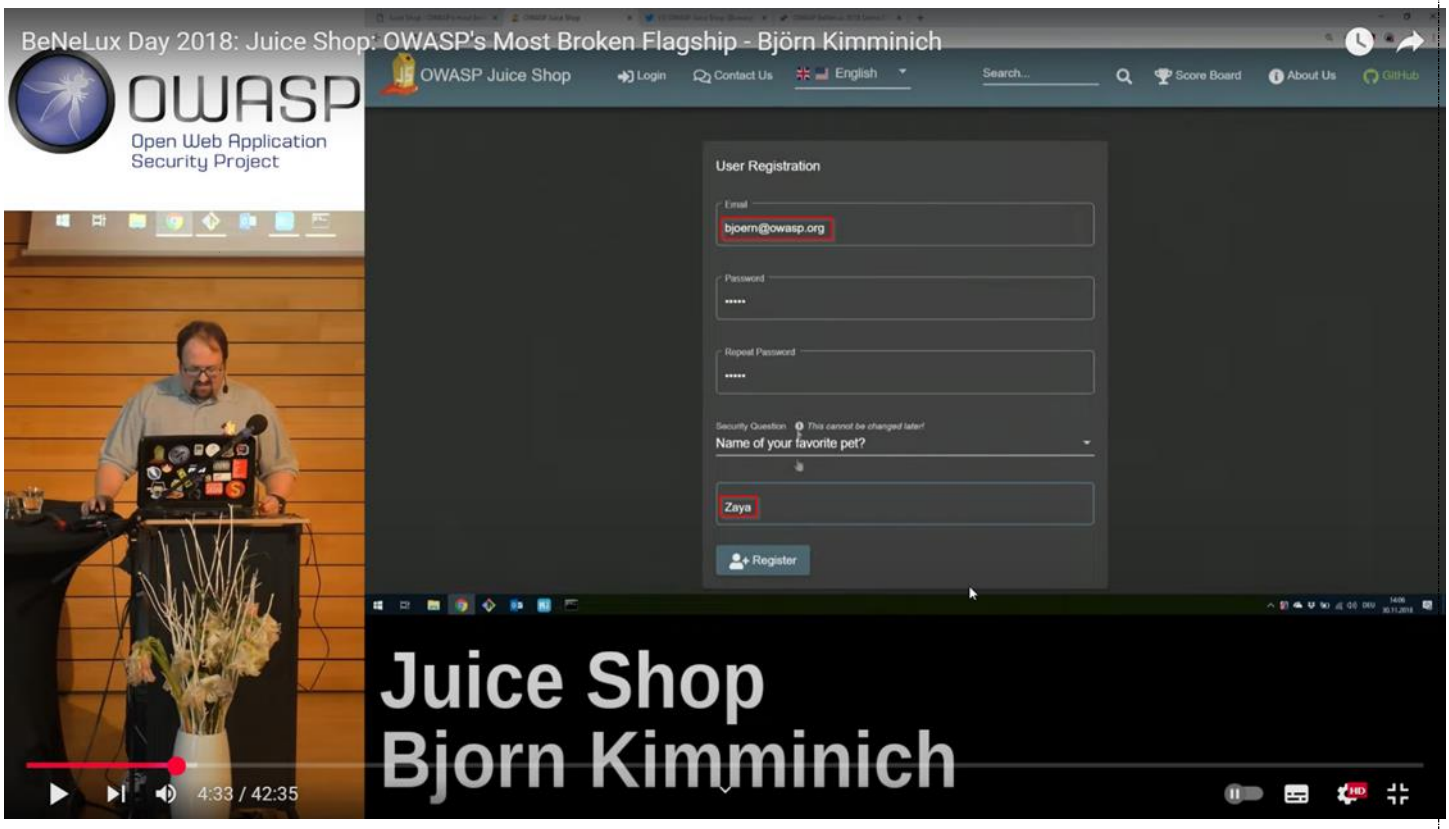


4. Search for pet-related info:

Checked the Photo Wall → No pet-related images from Bjoern.

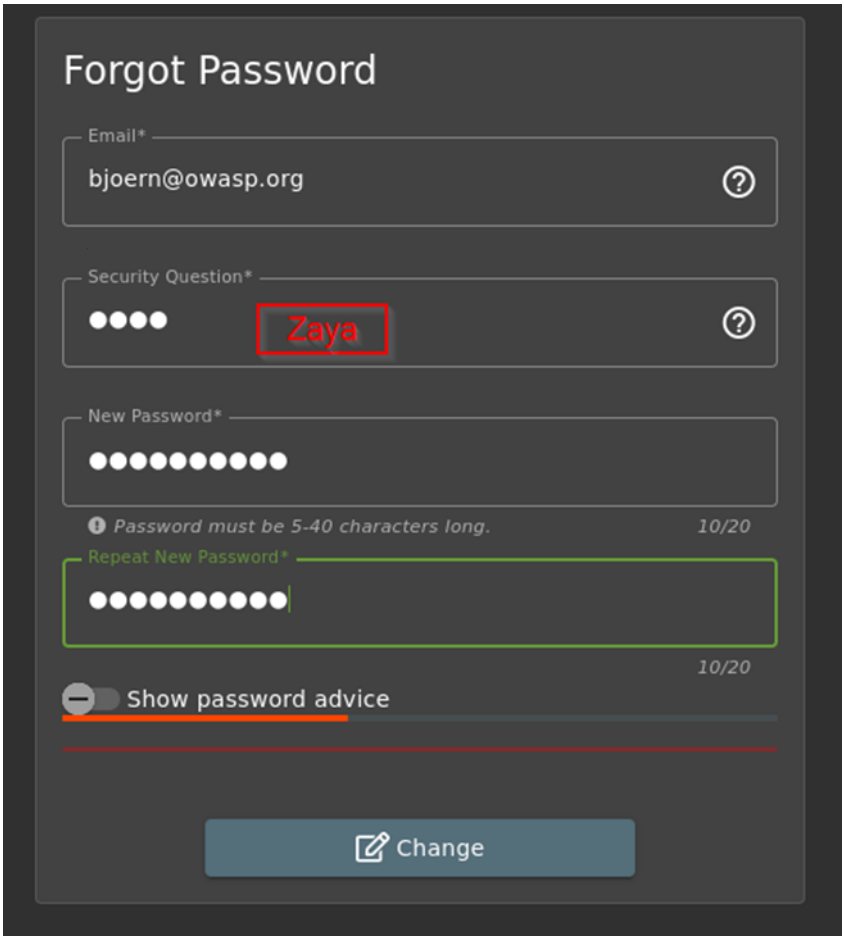


Conducted OSINT search online.



5. Found a tweet/video where Bjoern mentions his pet named: Zaya

Submit the answer to reset the password



Successfully reset Bjoern’s password and gained access to his account.

6.13.3 GDPR Data Theft

HIGH

- Description:
  The application fails to properly authorize users during the data export process. Instead of applying strict server-side access controls, it obfuscates email addresses to protect user identity. This obfuscation is predictable and can be bypassed by registering with a similar-looking email address. As a result, a user can access another user's personal order data without proper authentication or authorization.

- Impact:
  The penetration tester was able to access sensitive order data belonging to another user by registering an account with an email address that becomes indistinguishable after the application's obfuscation process. This exposed personally identifiable information (PII), violating GDPR and other data protection principles.

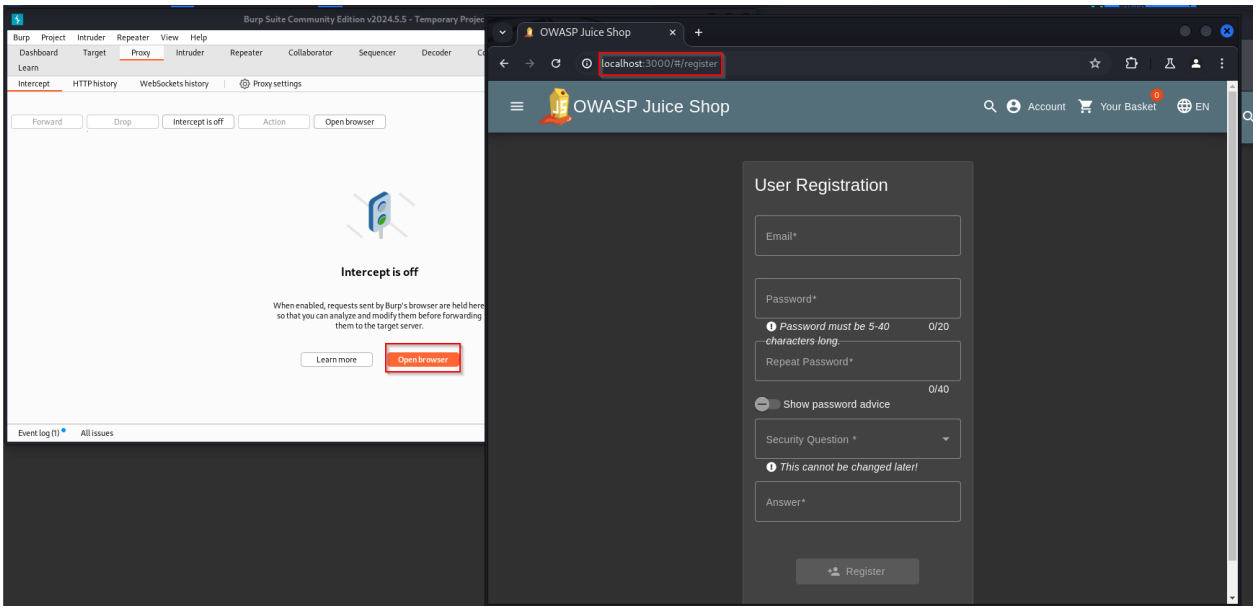
- Vulnerability Location:

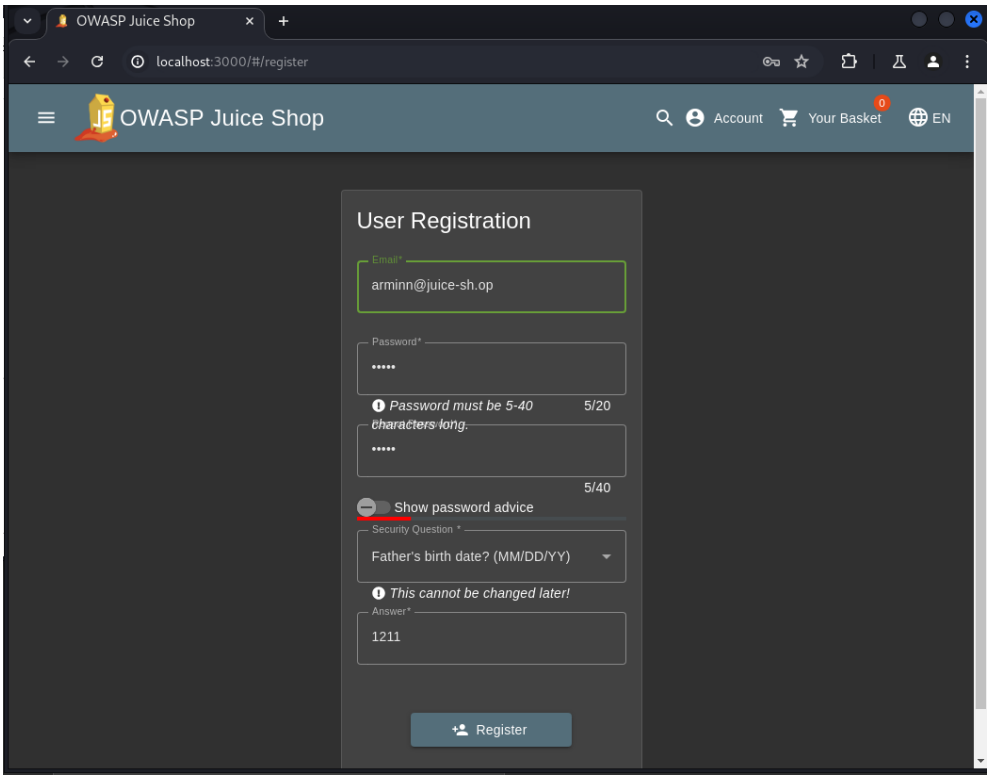
- CVE Reference:

- Recommendations:
  - Implement strict access control checks on the backend, ensuring users can only access their own data.
  - Ensure that internal mechanisms use complete and accurate identifiers rather than email obfuscation for privacy or security.
  - Validate identity (e.g., using token-based session validation) before fulfilling sensitive data export requests.

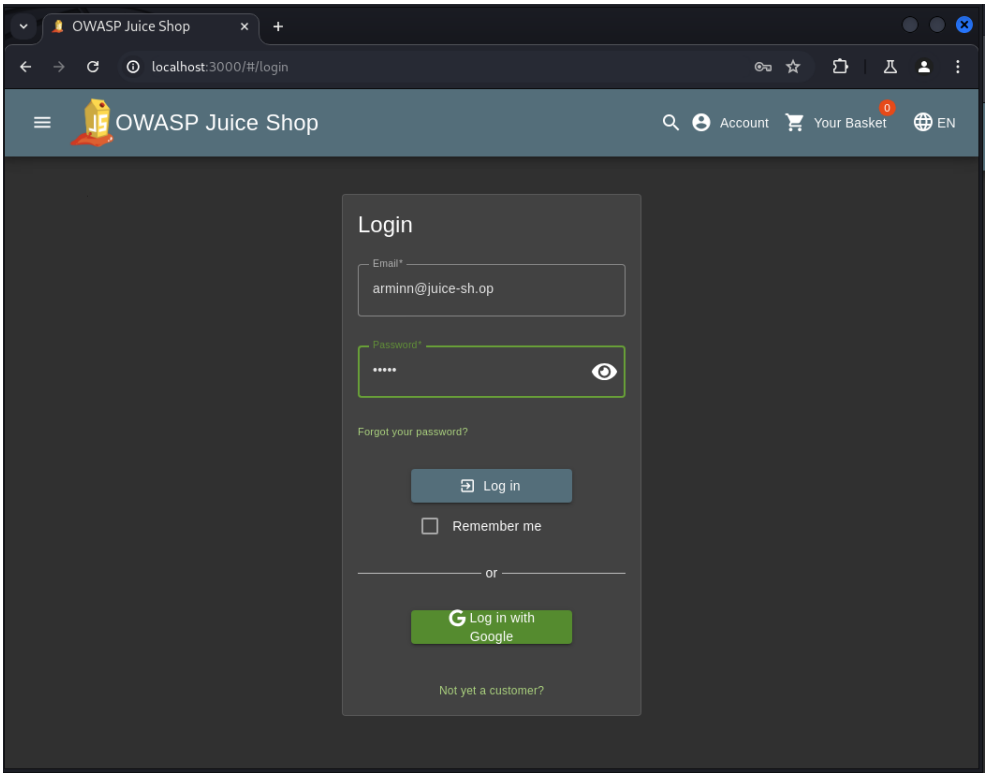
- Proof Of Concept:

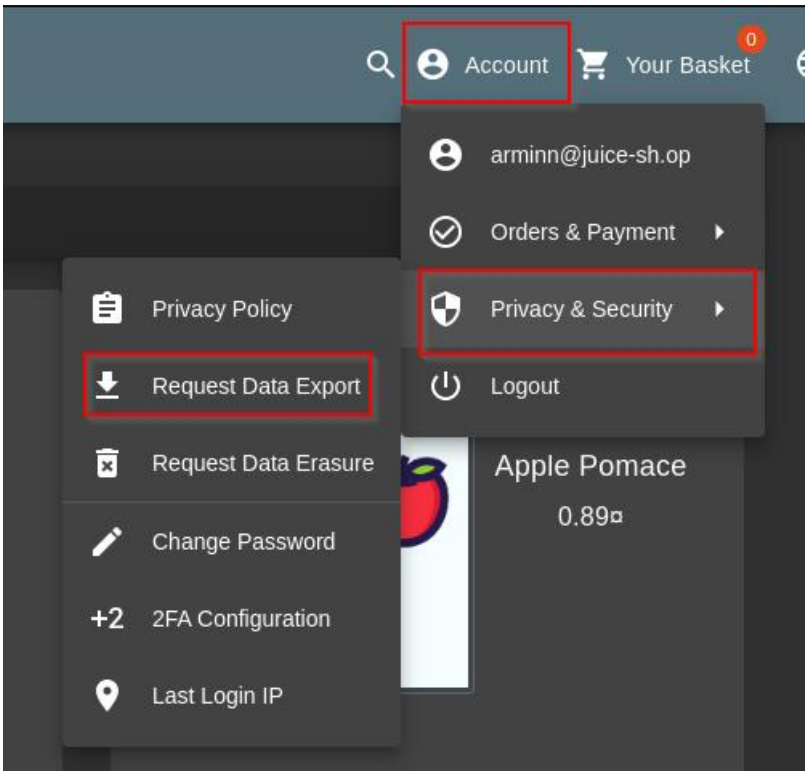
1. Open Burp Suite, open browser, navigate to the register page, and create a new account.

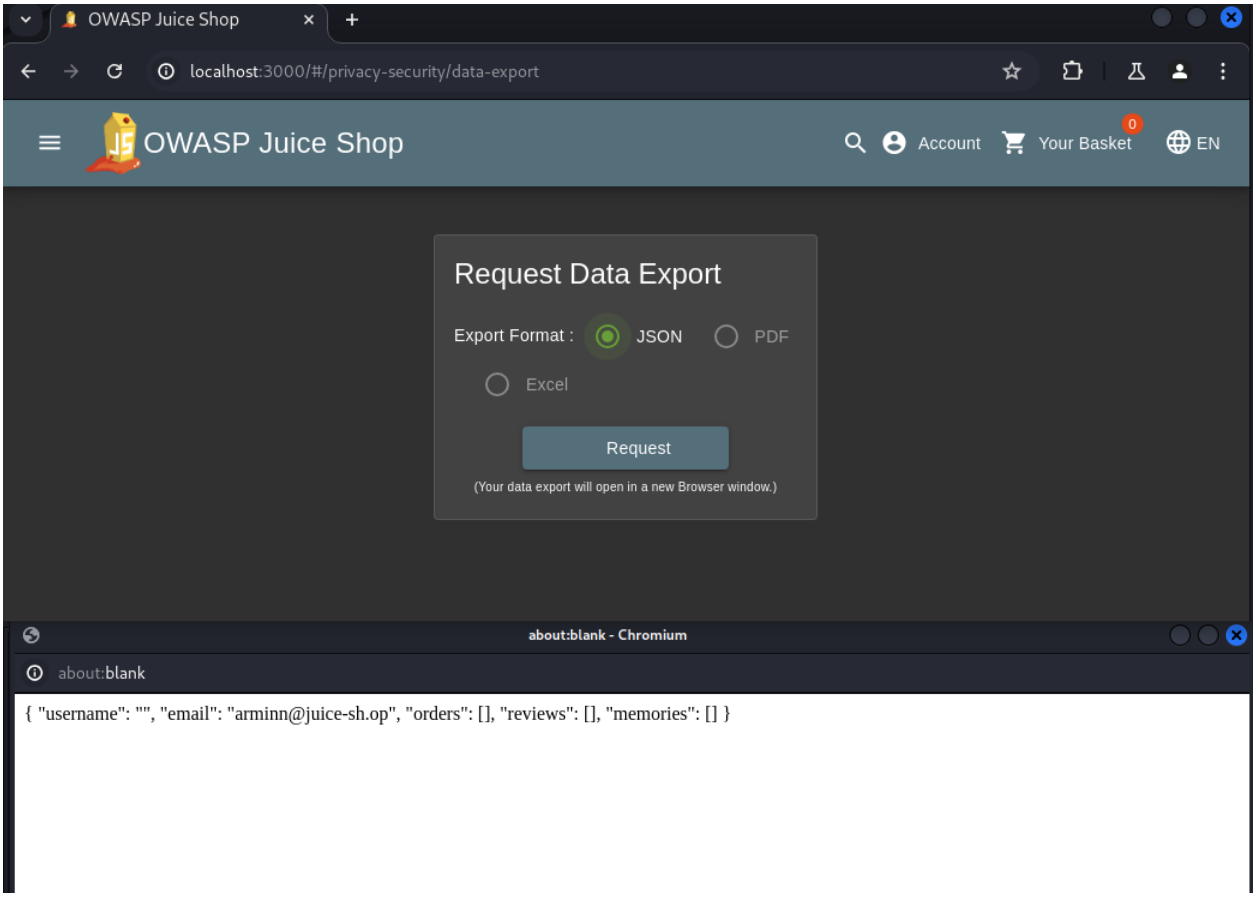




2. Log in and perform a data export request

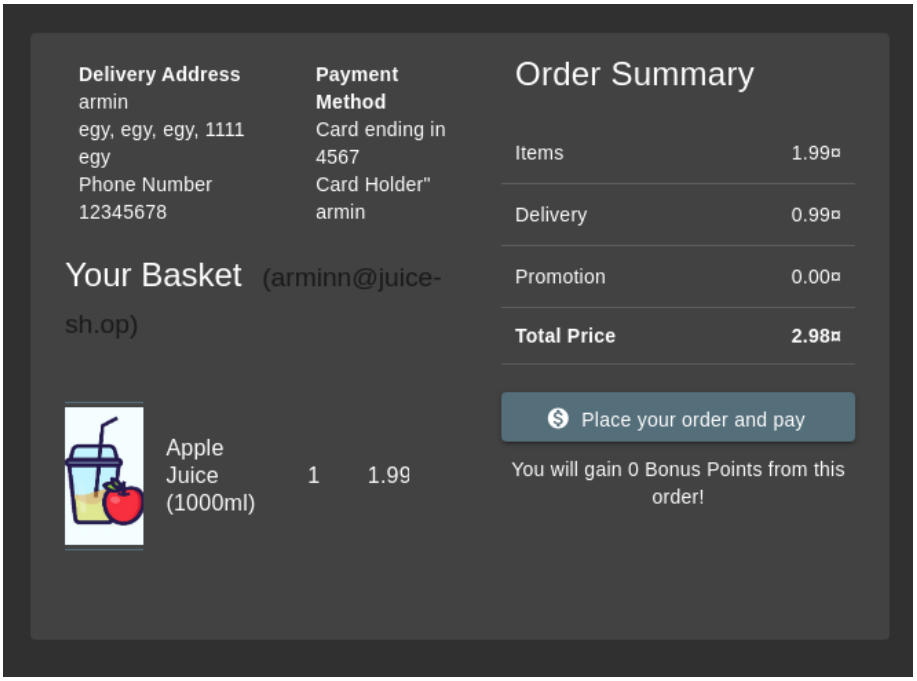




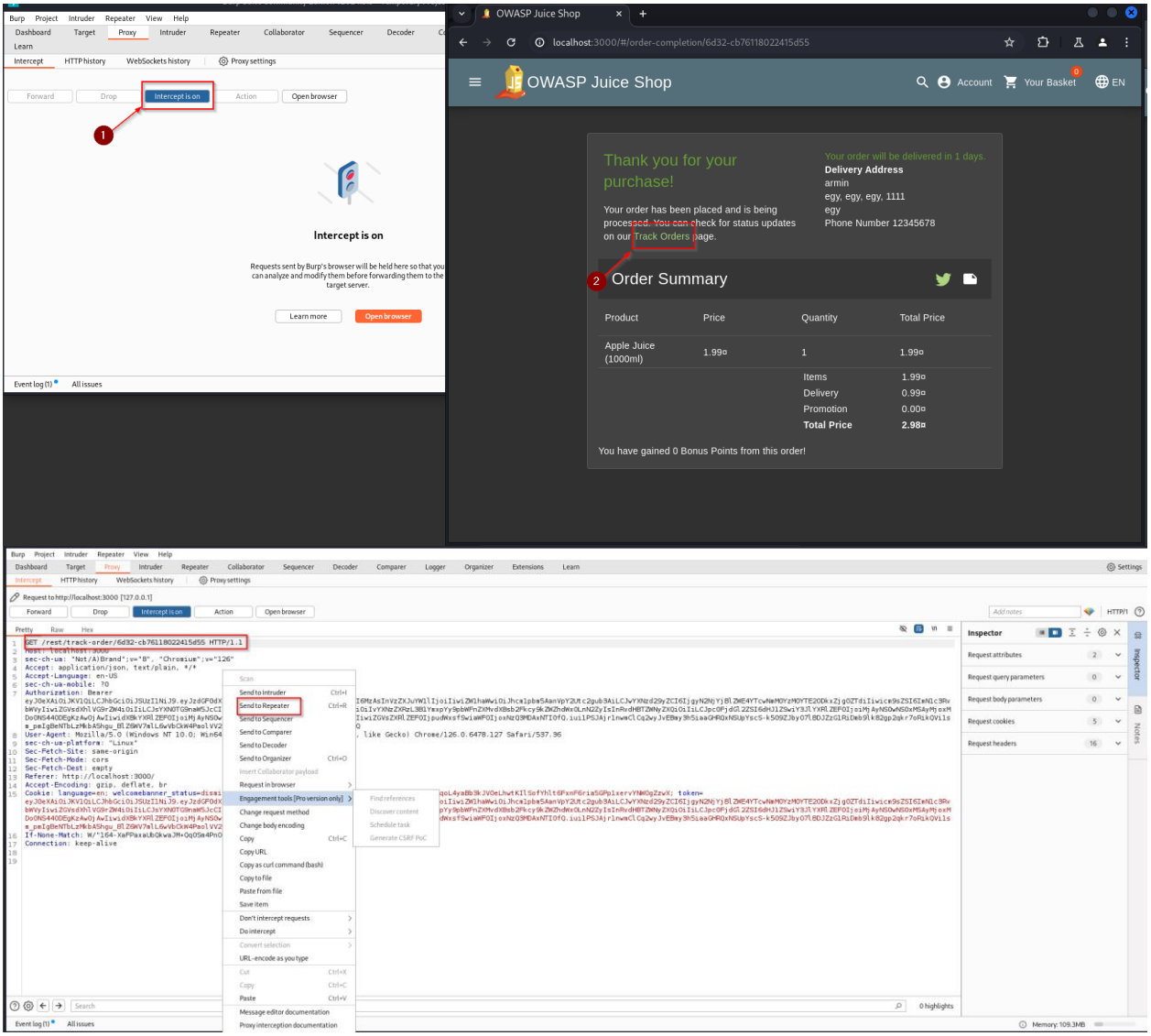


Nothing appears in the data file as we haven't made any orders yet

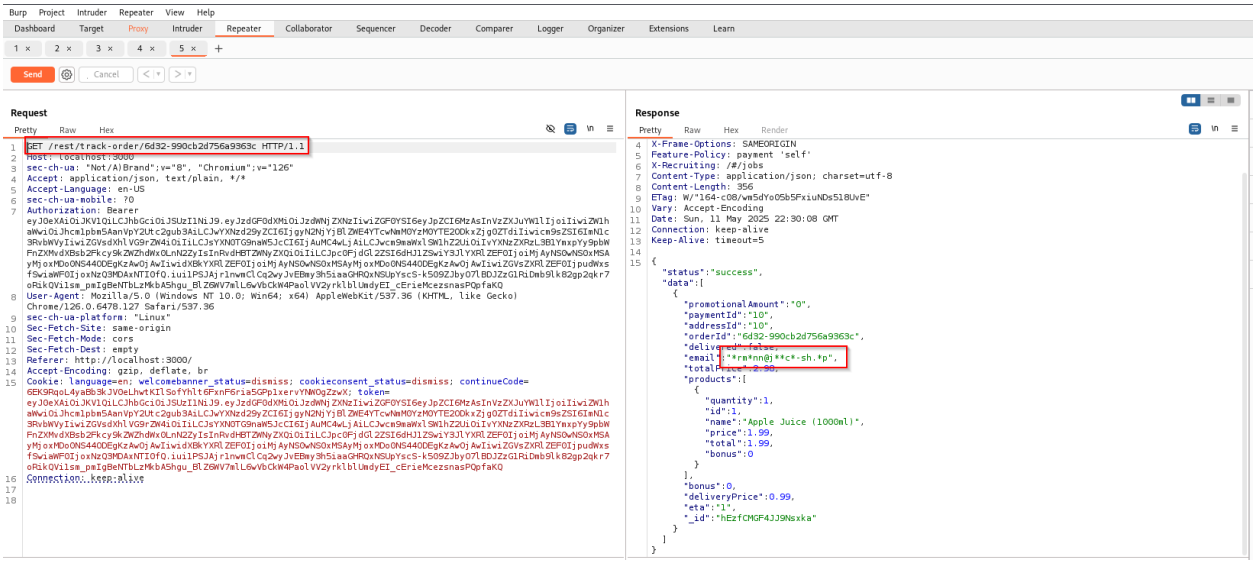
3. Make an order and see the request data export



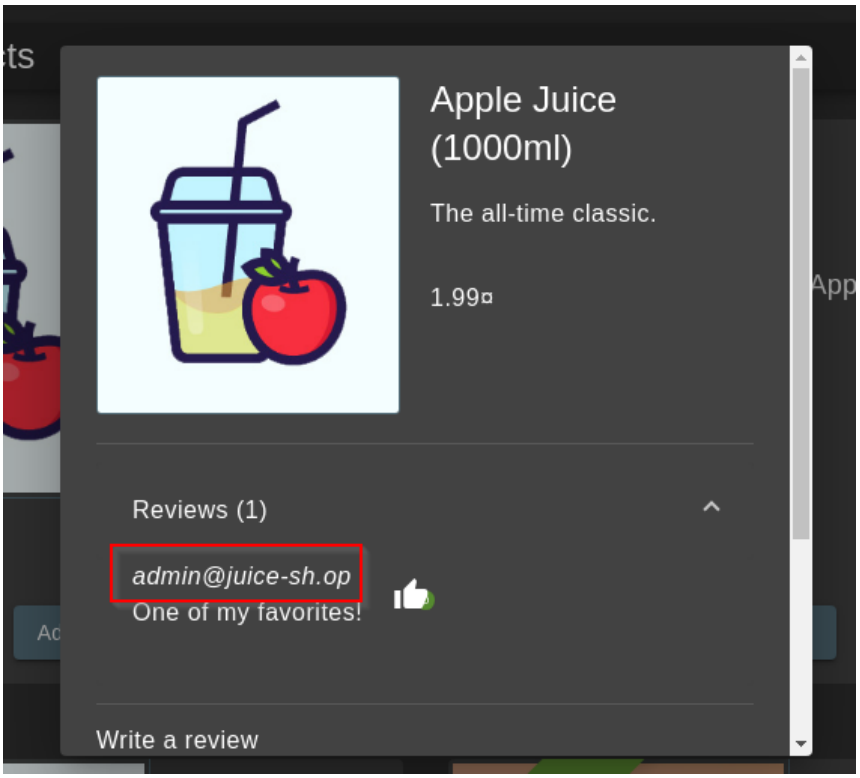
4. turn intercept on and capture the track order request and send it to repeater

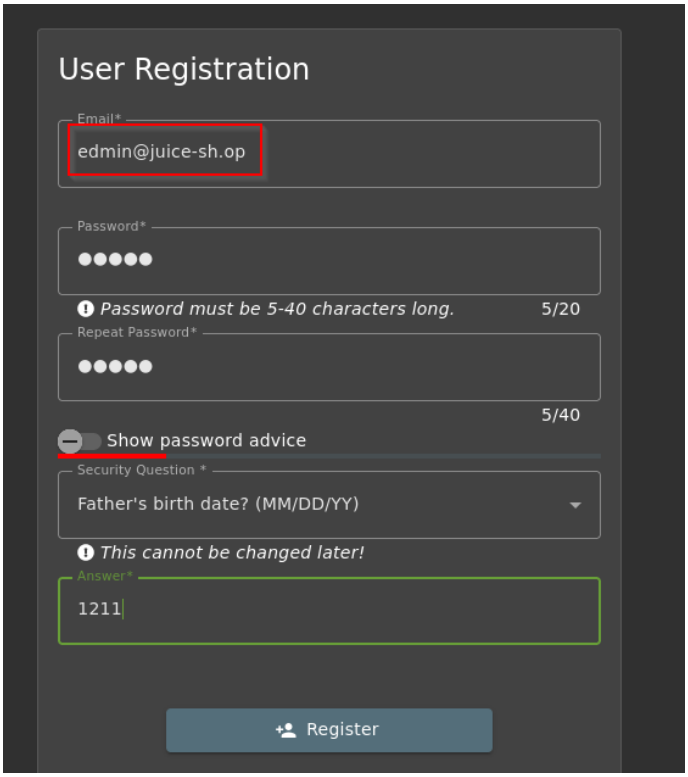


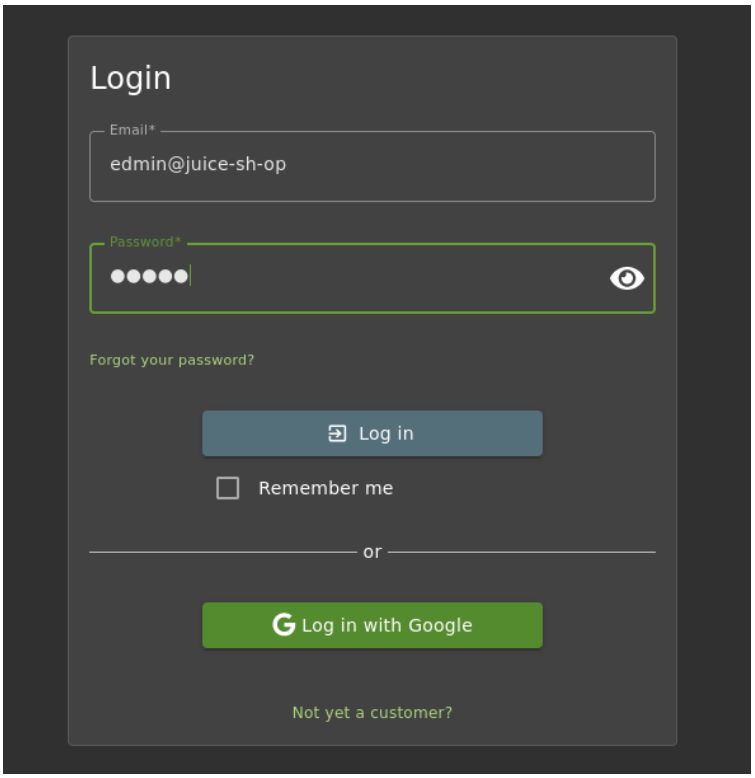
5. Observe the response and note that email addresses are obfuscated also notice the obfuscation pattern “arminn@juice-sh.op” → “*rm*nn@j**c*-sh.*p”



6. Create a new account with an email address that, when obfuscated, resembles an existing user’s email.

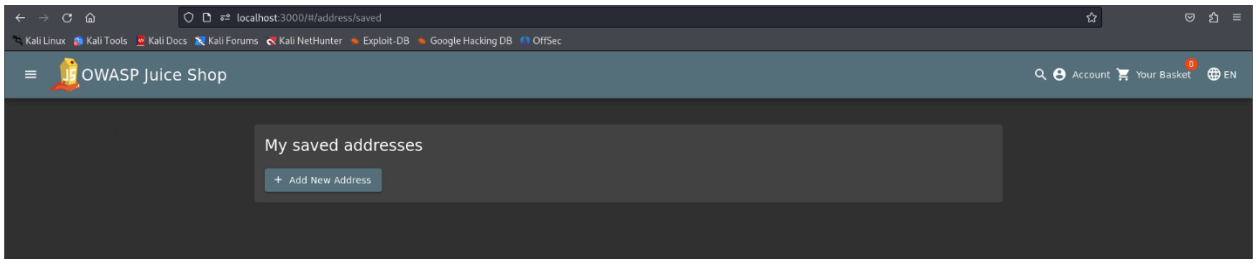


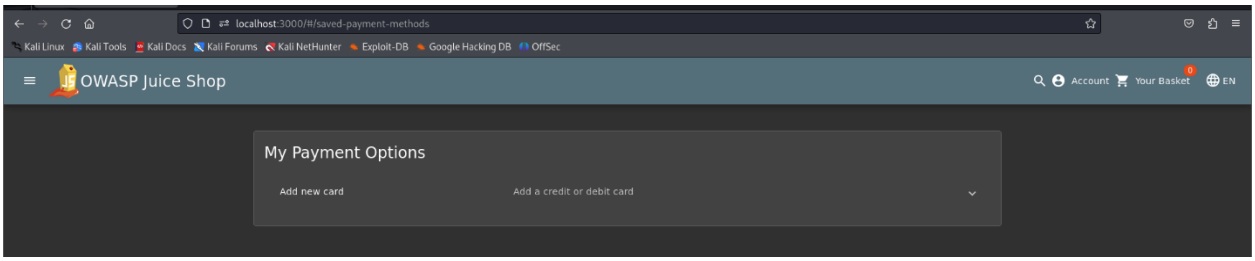




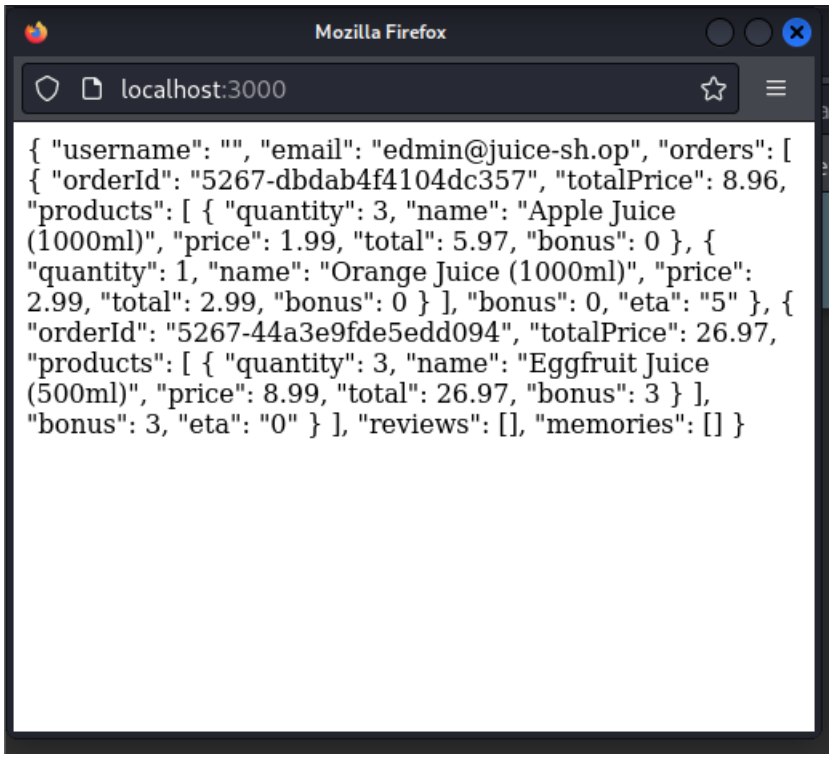
This was done based on the observed pattern from a previous order or interaction within the application where only parts of the email were visible, and asterisks replaced the rest.

7. Notice that the new account doesn’t have any saved data as there was no

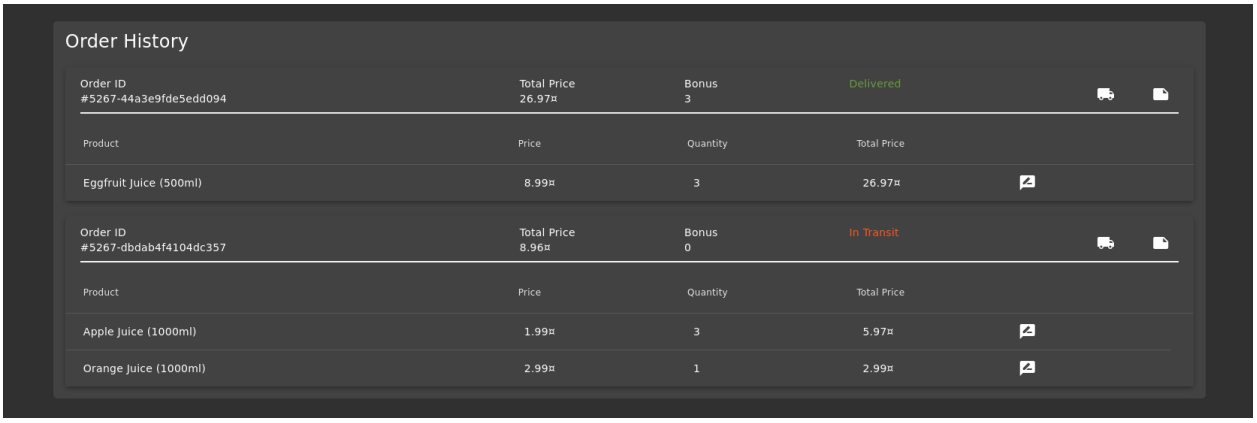




order placed before



8. Test access for data make a Request data export



We notice that the new account didn't make any orders before and still it showed me data from a different user appears, this indicates exposure of data of other user personal order history.

6.13.4 NFT Takeover

HIGH

- Description:
  we exposed critical NFT wallet credentials through user feedback. We gained access to an official Soul Bound Token (a type of NFT linked to a unique identity) by leveraging sensitive information exposed inadvertently through user feedback.

- Impact:
  A penetration tester was able to successfully obtain the seed phrase of a wallet by reviewing public feedback, converting it to a private key using a known cryptographic method BIP39, and gained full control over a protected NFT (Soul Bound Token). This results in complete compromise of ownership and identity-based assets, violating integrity and confidentiality.

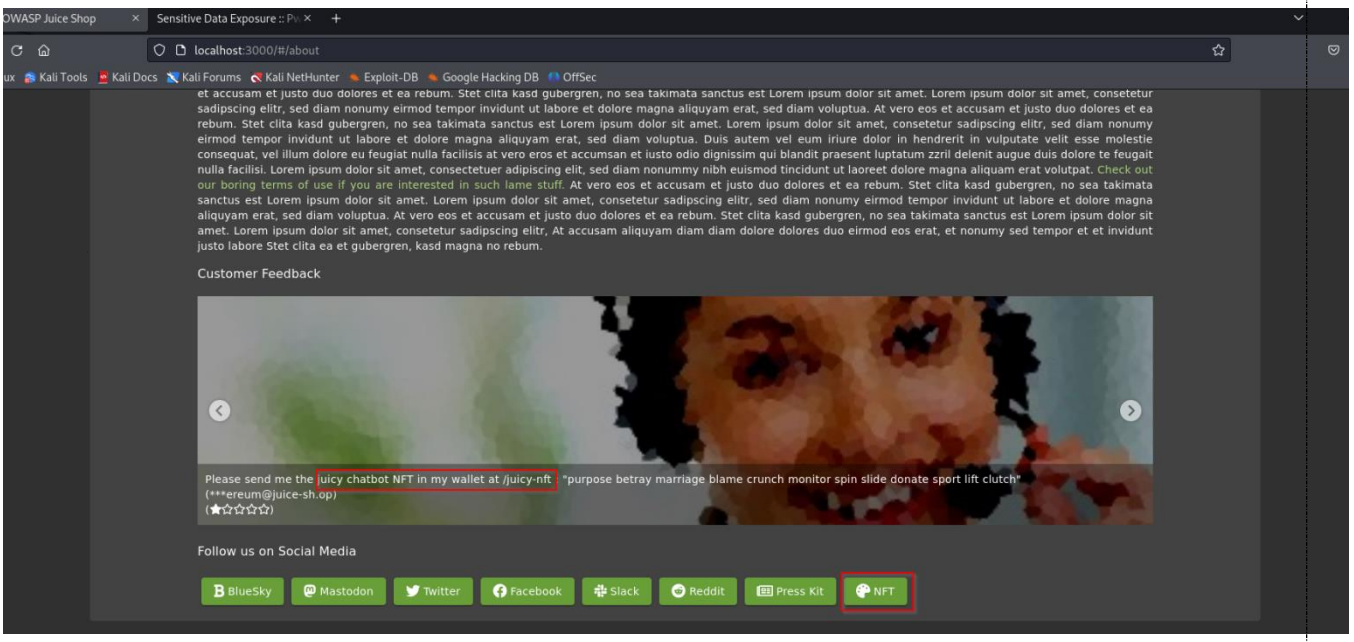
- Vulnerability Location:
  - Affected path: /juicy-nft
  - Leaked Data: Seed phrase

- CVE Reference:

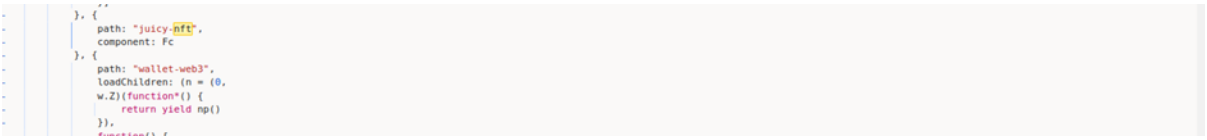
- Recommendations:
  - monitor user-submitted content (e.g., feedback, review) for sensitive data patterns.
  - Secure Information Handling Ensure that any form of sensitive information, such as cryptographic keys or seed phrases, is neither stored nor transmitted in clear text within user-accessible areas.
  - Deploy employ automated filtering to obscure or block the display of sensitive information.

- Proof Of Concept:

1. Search for NFT-related content in the application source code and UI.



A user comment mentioning the path: “/juicy-nft”



discovered the /juicy-nft path in the application's JavaScript file

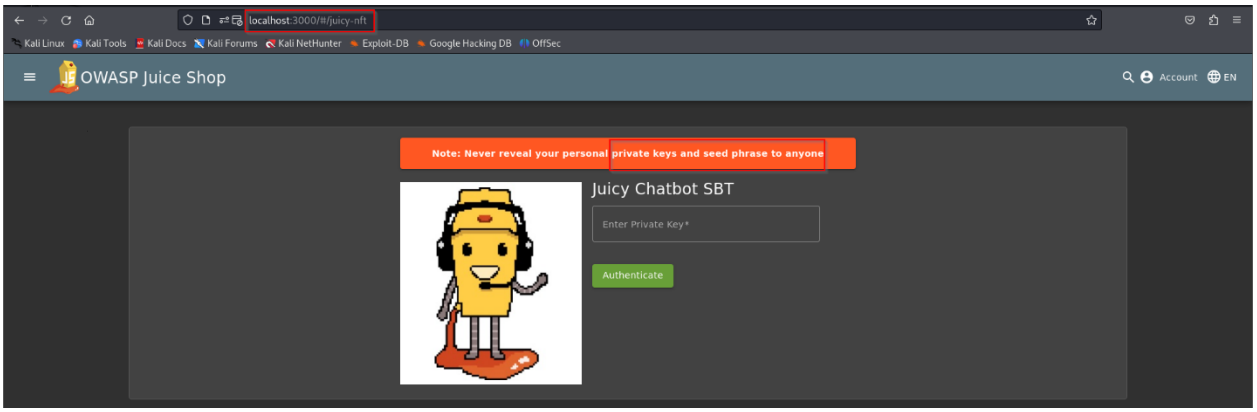
2. Extracting the Seed Phrase from the same feedback.

“purpose betray marriage blame crunch monitor spin slide donate sport lift

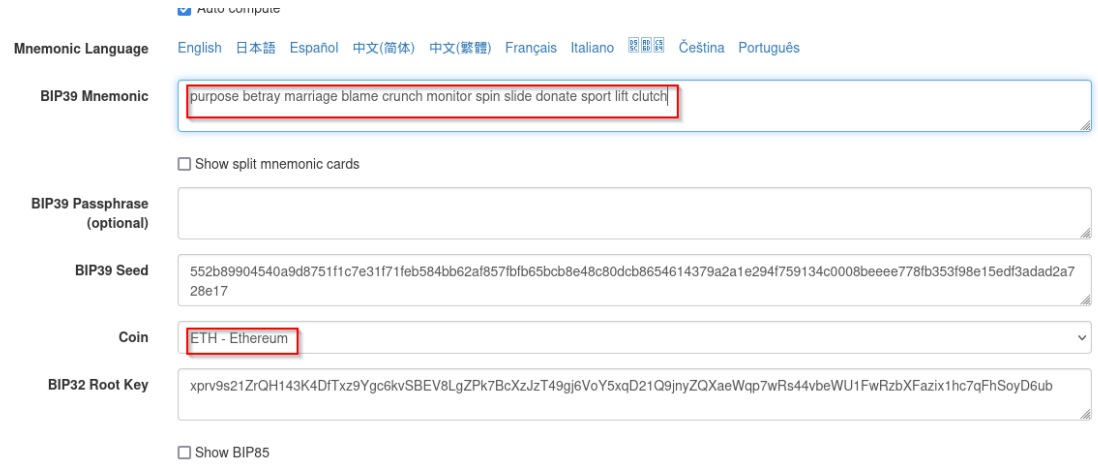
clutch”

Recognized that this seed phrase likely relates to the private key required by the /juicy-nft page.

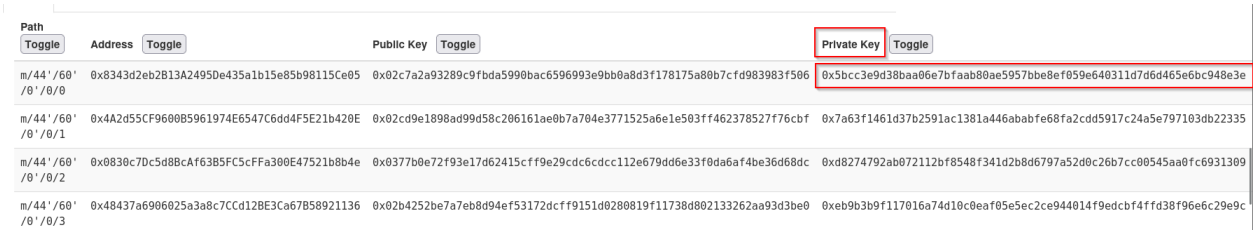
3. The NFT wallet seed phrase follows the “BIP39 standard” to generate private key and it should be “Ethereum”



4. Used the Mnemonic Converter Tool (https://iancoleman.io/bip39/) to convert the seed phrase into a private key

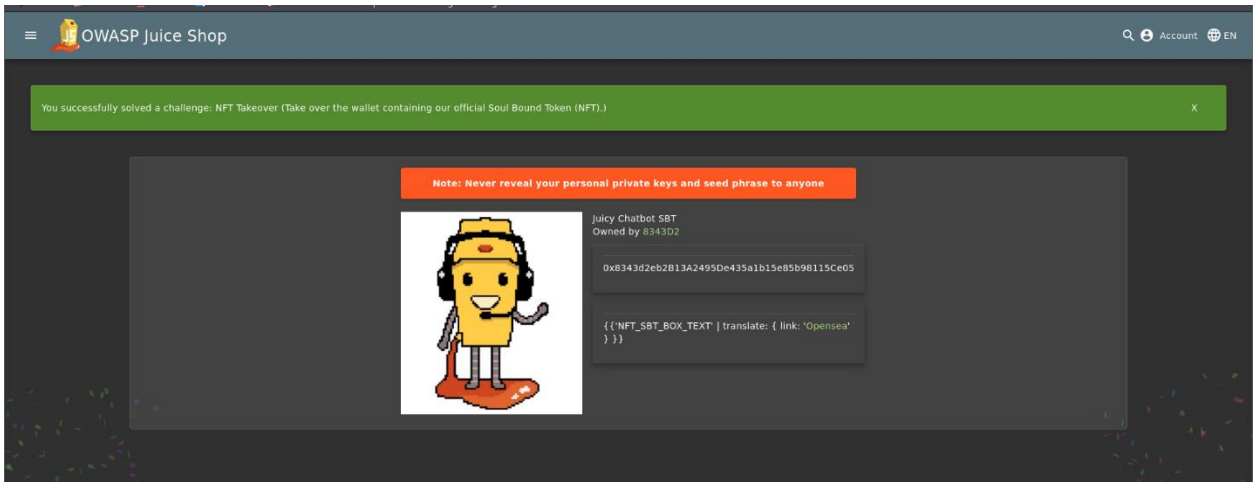


5. Obtained the private key: “0x5bcc3e9d38baa06e7bfaab80ae5957bbe8ef059e640311d7d6d465e6bc948e3”



Path Toggle	Address Toggle	Public Key Toggle	Private Key Toggle
m/44'/60'/0'/0/0	0x8343d2eb2B13A2495De435a1b15e85b98115Ce05	0x02c7a2a93289c9fbda5990bac6596993e9bb0a8d3f178175a80b7cfd983983f506	0x5bcc3e9d38baa06e7bfaab80ae5957bbe8ef059e640311d7d6d465e6bc948e3e
m/44'/60'/0'/0/1	0x4A2d55CF9600B5961974E6547C6dd4F5E21b420E	0x02cd9e1898ad99d58c206161ae0b7a704e3771525a6e1e503ff462378527f76cbf	0x7a63f1461d37b2591ac1381a446ababfe68fa2cdd5917c24a5e797103db22335
m/44'/60'/0'/0/2	0x0830c7Dc5d8BcAf63B5FC5cFFa300E47521b8b4e	0x0377b0e72f93e17d62415cff9e29cdc6cdcc112e679dd6e33f0da6af4be36d68dc	0xd8274792ab072112bf8548f341d2b8d6797a52d0c26b7cc00545aa0fc6931309
m/44'/60'/0'/0/3	0x48437a6906025a3a8c7CCd12BE3Ca67B58921136	0x02b4252be7a7eb8d94ef53172dcff9151d0280819f11738d802133262aa93d3be0	0xeb9b3b9f117016a74d10c0eaf05e5ec2ce944014f9edcbf4ffd38f96e6c29e9c

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



Successfully accessed the wallet and verified control over the Soul Bound Token NFT, completing the takeover