Vulnerability Assessment and Penetration Test

Report for OWASP Juice Shop

TEAM Members :

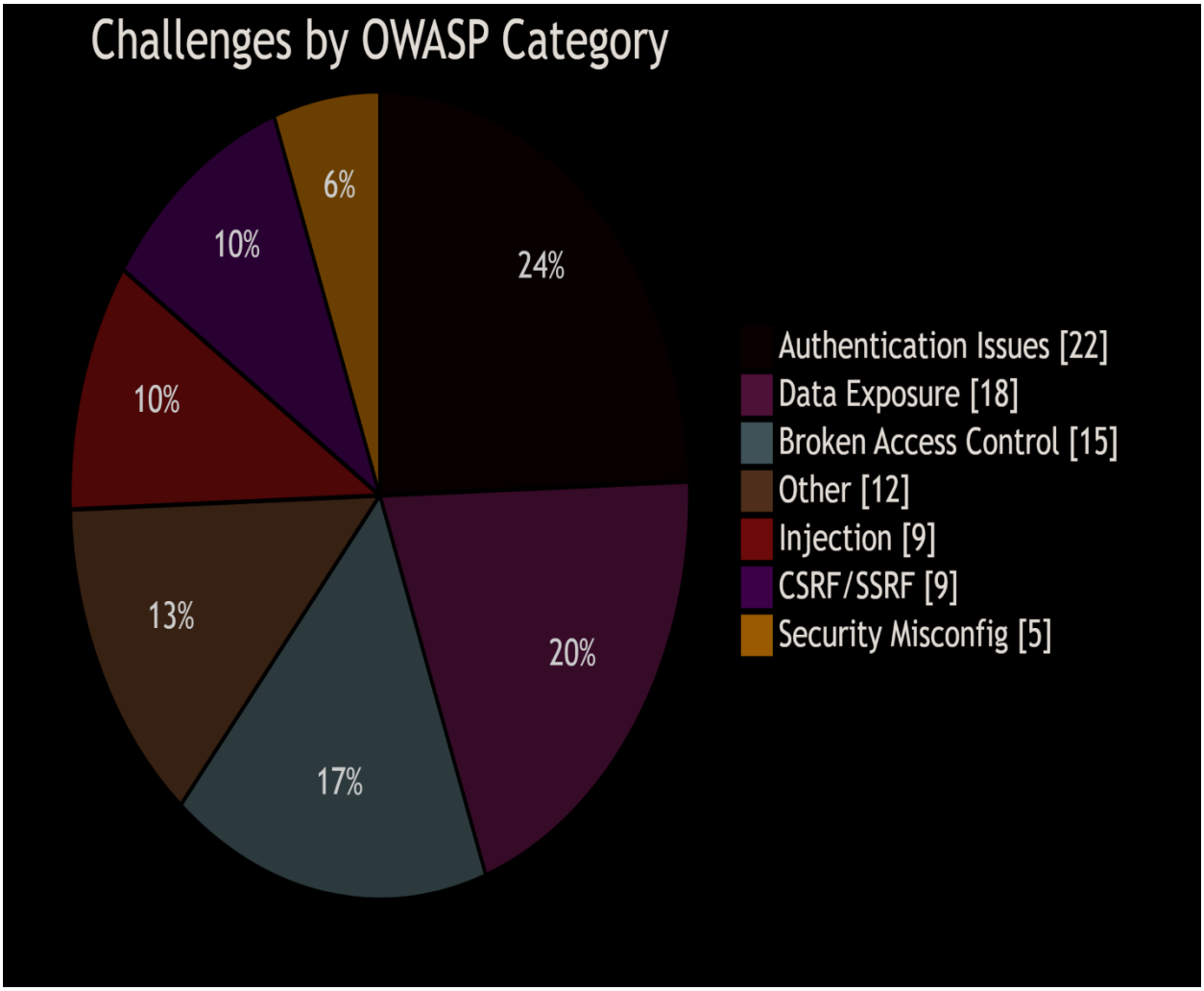
Habiba hosam eldin salah

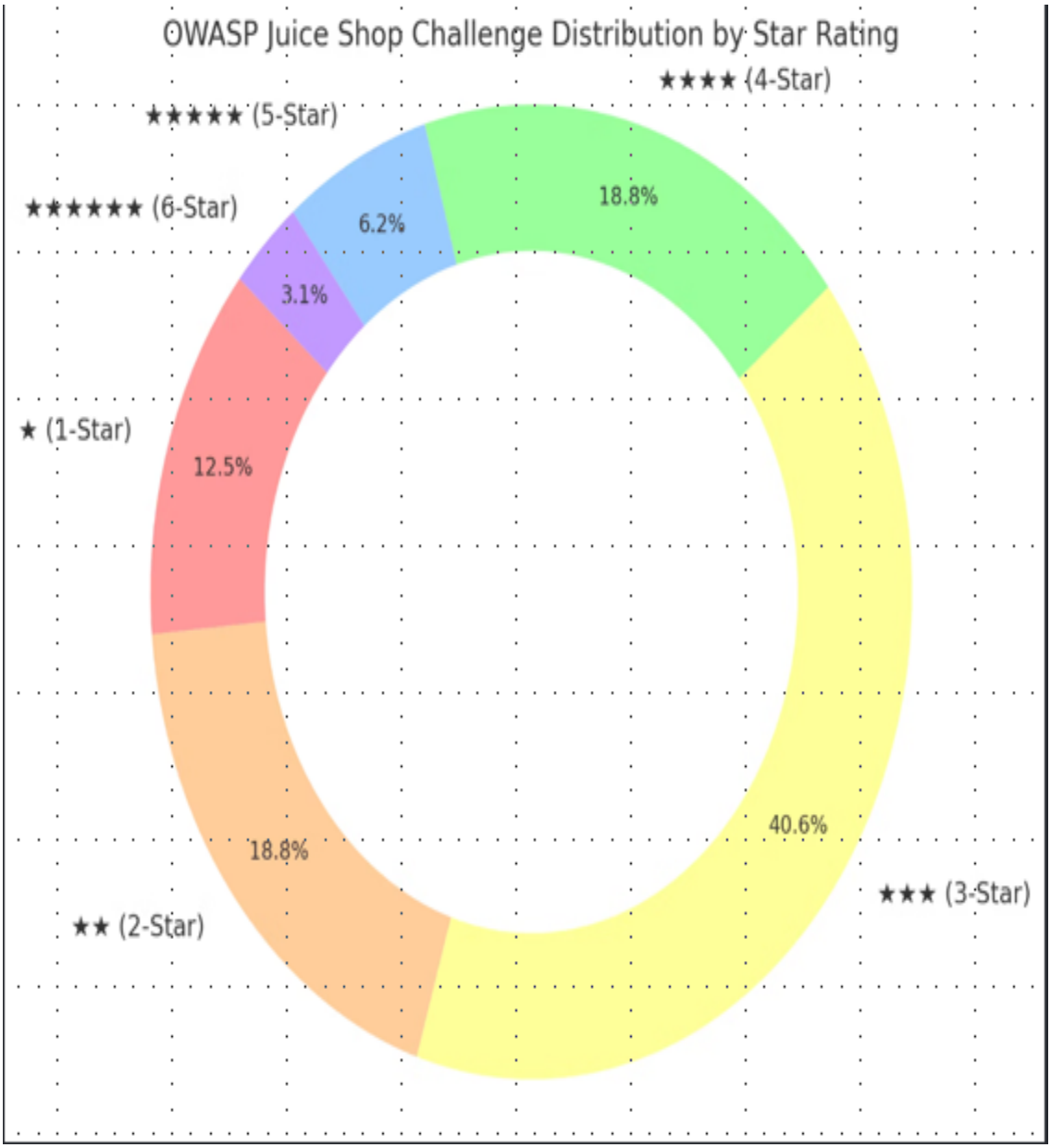
Sohaila Mohammed Ahmed Mohammed

Mohamed Ayman Ahmed

Mohamed abdelbasit

Table of Contents


Challenges by OWASP Category
24%
20%
17%
13%
10%
10%
6%
Authentication Issues [22]
Data Exposure [18]
Broken Access Control [15]
Other [12]
Injection [9]
CSRF/SSRF [9]
Security Misconfig [5]


OWASP Juice Shop Challenge Distribution by Star Rating
★★★★ (4-Star)
★★★★★ (5-Star)
★★★★★★ (6-Star)
★ (1-Star)
★★ (2-Star)
★★★ (3-Star)
18.8%
6.2%
3.1%
12.5%
18.8%
40.6%

1 - Executive Summary

SQL Injection

- **Login Form**
  Description: Malicious SQL code is injected into login inputs to bypass authentication.
  Impact: Unauthorized access to user or admin accounts.
- **Search Query**
  Description: SQL injection through search input allows manipulation of queries.
  Impact: Data leakage, modification, or deletion.

Reflected XSS

- **Search Functionality**
  Description: User input is reflected in the response without sanitization.
  Impact: Executes JavaScript in the user's browser, leading to session theft or phishing.

Broken Access Control

- **File Transfer Endpoint**
  Description: Insecure endpoint allows unauthorized file transfer operations.
  Impact: Unauthorized file uploads/downloads, leading to data breach.
- **Sensitive Data in Source Code**
  Description: Hardcoded credentials or API keys exposed in client-side code.
  Impact: Attackers can misuse services or access sensitive systems.
- **Viewing Basket**
  Description: Users can view others' shopping baskets by manipulating parameters.
  Impact: Privacy violations and potential data theft.
- **Admin Section**
  Description: Unrestricted access to the admin panel.
  Impact: Full control over system configuration and data.
- **IDOR in Basket ID**
  Description: Direct object reference in basket ID without access control.
  Impact: Unauthorized viewing or modification of other users' data.
- **IDOR in UserID**
  Description: Changing userID parameter gives access to other users' information.
  Impact: Identity theft or data manipulation.
- **Product Review Views**
  Description: Reviews are visible without proper access checks.
  Impact: Unauthorized viewing of internal or private user feedback.

- **Email Leak**
  Description: Emails are exposed via insecure endpoints or views.
  Impact: Spam, phishing, or targeted attacks.
- **Login MC Safesearch**
  Description: Restricted feature accessible without proper login validation.
  Impact: Unauthorized use of sensitive features.
- **Access Log**
  Description: Access to system logs without proper restrictions.
  Impact: User activity exposure and potential privacy breaches.
- **Empty User Registration**
  Description: User registration allowed with missing or invalid data.
  Impact: Creation of fake or incomplete accounts.

Broken Authentication

- **Reset by Security Question**
  Description: Weak or guessable security questions allow password resets.
  Impact: Account takeover.
- **Change Password**
  Description: Password can be changed without full verification.
  Impact: Unauthorized control over user accounts.
- **Weak Password Policy**
  Description: System allows short or easily guessable passwords.
  Impact: Increased risk of brute-force attacks.
- **GDPR**
  Description: Failure to secure user data as required by GDPR.
  Impact: Legal penalties and loss of user trust.
- **CAPTCHA Bypass**
  Description: CAPTCHA can be bypassed or is missing.
  Impact: Automated attacks like spam or brute-force login.

Security Misconfiguration

- **Complaint Section**
  Description: Poor access controls in complaints interface.
  Impact: Unauthorized access or editing of complaint records.
- **Error Handling**
  Description: Detailed system errors are shown to users.
  Impact: Information disclosure that aids attackers.
- **Upload File**
  Description: Insecure file upload feature lacks validation.
  Impact: Malware upload and remote code execution.

- **Upload Type**
  Description: File type restrictions not enforced properly.
  Impact: Uploading of executable or malicious files.

CSRF

- **Profile Update Endpoint**
  Description: No CSRF protection on profile updates.
  Impact: Attackers can modify user profiles without their consent.

Sensitive Data Exposure

- **FTP Directory Access**
  Description: FTP folders accessible without authentication.
  Impact: Sensitive file exposure or leakage.
- **File Download Endpoint**
  Description: Insecure download endpoint leaks private files.
  Impact: Unauthorized data access or theft.
- **GDPR Data Theft**
  Description: Personal data is exposed or not properly protected.
  Impact: Breach of GDPR regulations, leading to fines.
- **Meta Geo Stalking**
  Description: Metadata in uploads reveals user location.
  Impact: User tracking or stalking risks.

Improper Input Validation

- **Deluxe Fraud**
  Description: Manipulated input gives unauthorized benefits.
  Impact: Financial loss or system abuse.
- **Repetitive Registration**
  Description: Lack of checks allows repeated registrations.
  Impact: System overload or misuse of promotions.
- **Payback Time**
  Description: Exploitable refund/payment logic due to bad validation.
  Impact: Fraudulent refunds or revenue loss.
- **Admin Registration**
  Description: Insecure registration allows privilege escalation.
  Impact: Unauthorized admin account creation.

- **Poison Null Byte**
  Description: Null byte used to bypass string termination or path validation.
  Impact: File upload or access control bypass.

SSRF

- **Image Upload Feature**
  Description: External URLs in image uploads trigger internal requests.
  Impact: Internal service access or data extraction.

Miscellaneous

- **Security Policy**
  Description: Absence or poor implementation of security policies.
  Impact: Increased exposure to a wide range of vulnerabilities.
- **Supply Chain Attack**
  Description: Infected or malicious dependencies used in the system.
  Impact: Compromise via trusted third-party software.

2. Technical findings

1) SSRF – image upload feature

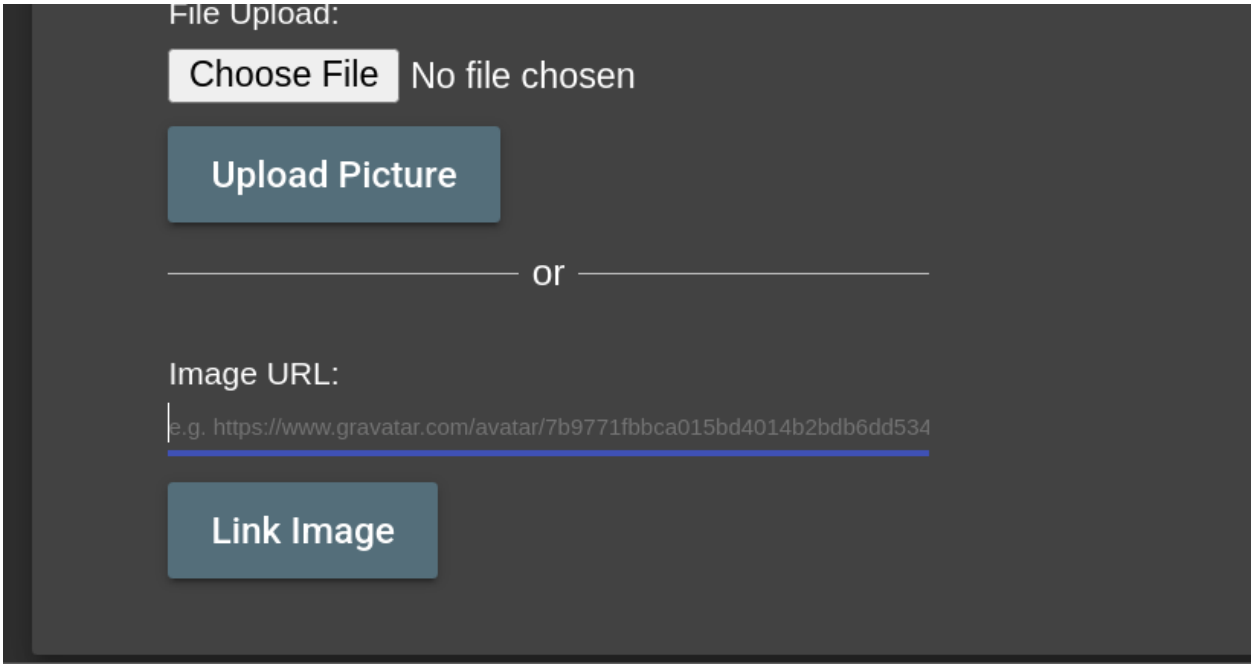
Severity: Critical

Description:
The application is vulnerable to a Server-Side Request Forgery (SSRF) attack due to improper validation of user-supplied URLs in the image upload functionality. This allows an attacker to manipulate the server into making unauthorized requests to internal or external resources, potentially exposing sensitive data or enabling further exploitation

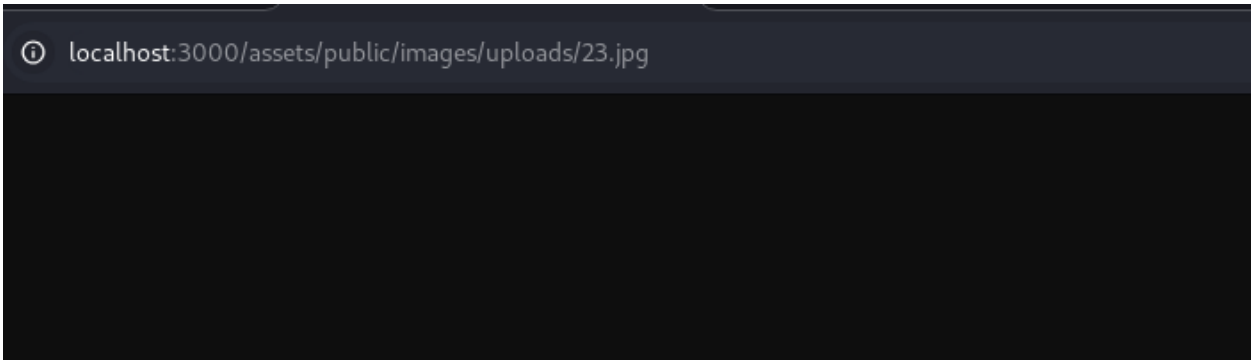
Affected Component:
Image upload feature (specifically the "Image URL" input field) that fetches images from user-provided URLs.

Steps to Reproduce (in the vulnerable application):

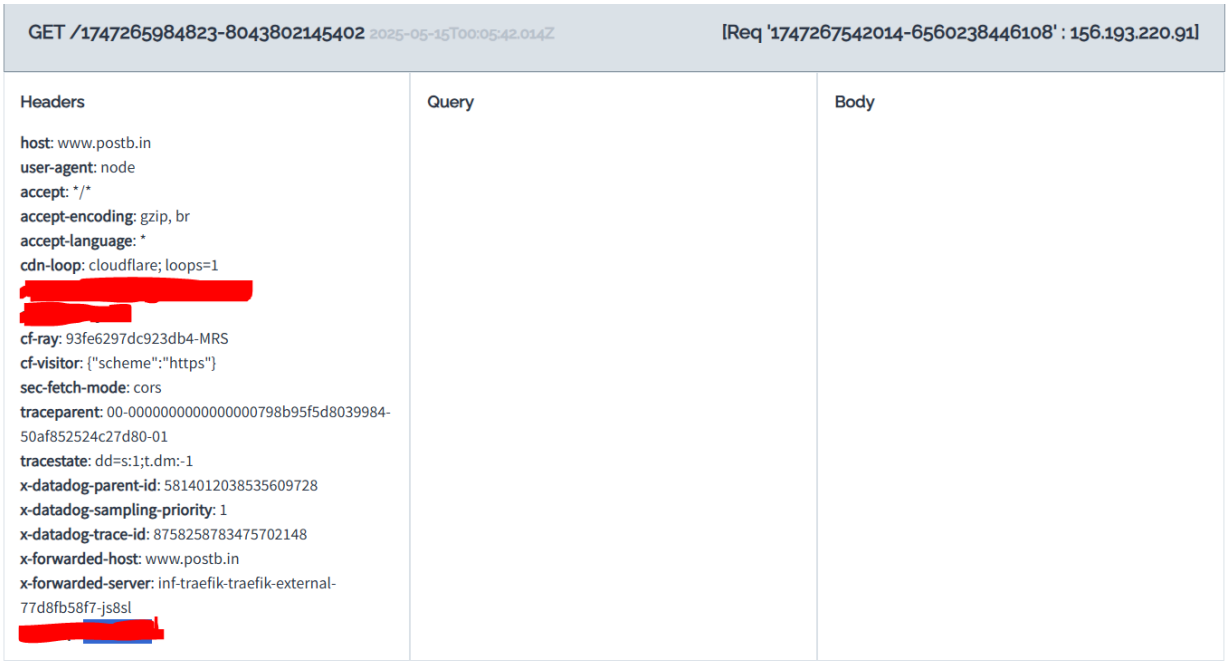
1. Access the application and navigate to the image upload page.



2. Upload an image file (e.g., resulting in a src like assets/public/images/uploads/23.jpg at localhost:3000).



3. Locate the "Image URL" input field, which allows entering a URL to fetch an image.
4. Obtain a unique PostBin URL (e.g., https://postb.in/123456) from the PostBin service.
5. Enter the PostBin URL (https://postb.in/123456) into the "Image URL" field.
6. Click the "Link Image" button.
7. Check the PostBin dashboard to confirm the server sent a request to the provided URL, as evidenced by logged headers (e.g., user-agent, cf-ray).



Impact:

• Unauthorized access to internal services (e.g., http://localhost:8080/admin or http://127.0.0.1).

• Potential extraction of sensitive data from internal resources or cloud metadata endpoints (e.g., http://169.254.169.254/latest/meta-data/ on AWS).

• Ability to bypass network restrictions or firewalls, enabling reconnaissance or further attacks.

• Possible exploitation of misconfigured internal systems, leading to full server compromise.

2) Broken access control – email leak

Severity: Critical

Description:

Broken Access Control occurs when an application fails to enforce proper authorization or authentication checks, allowing unauthorized users to access restricted resources or data. The /usr/whoami?callback={parameter} endpoint returns sensitive user data when the callback parameter is modified (e.g., to -admin`), without verifying the requestor's authentication or authorization status, allowing unauthorized access to any user's data.

Affected Component:

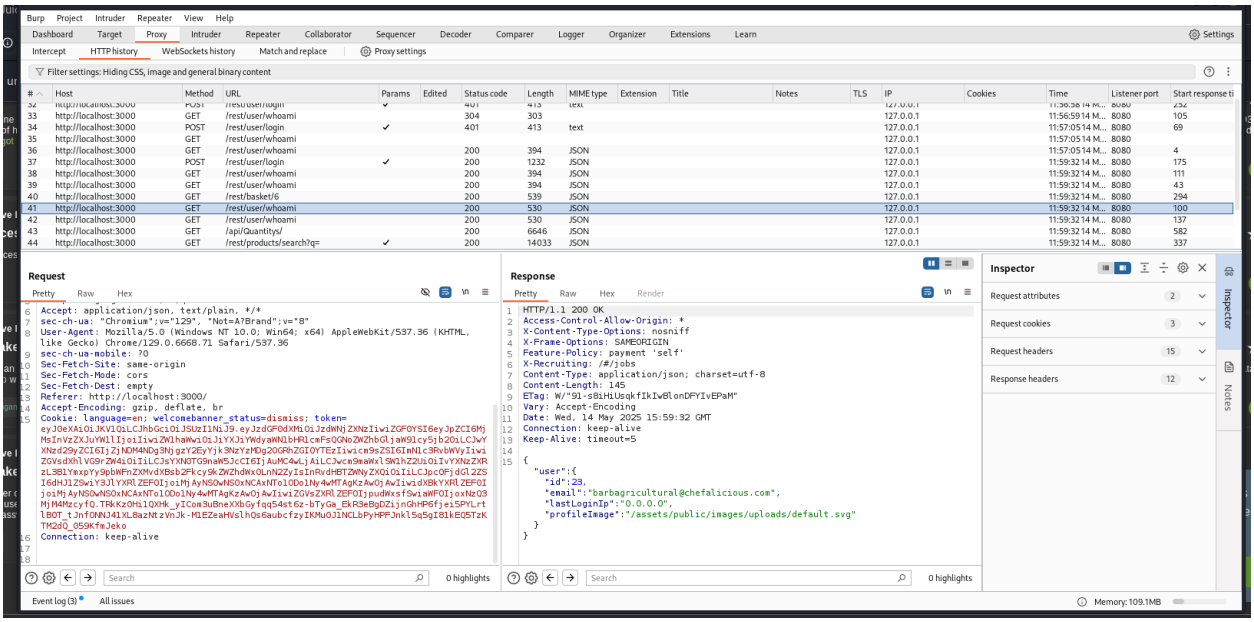
User information endpoint

/usr/whoami?callback={parameter}

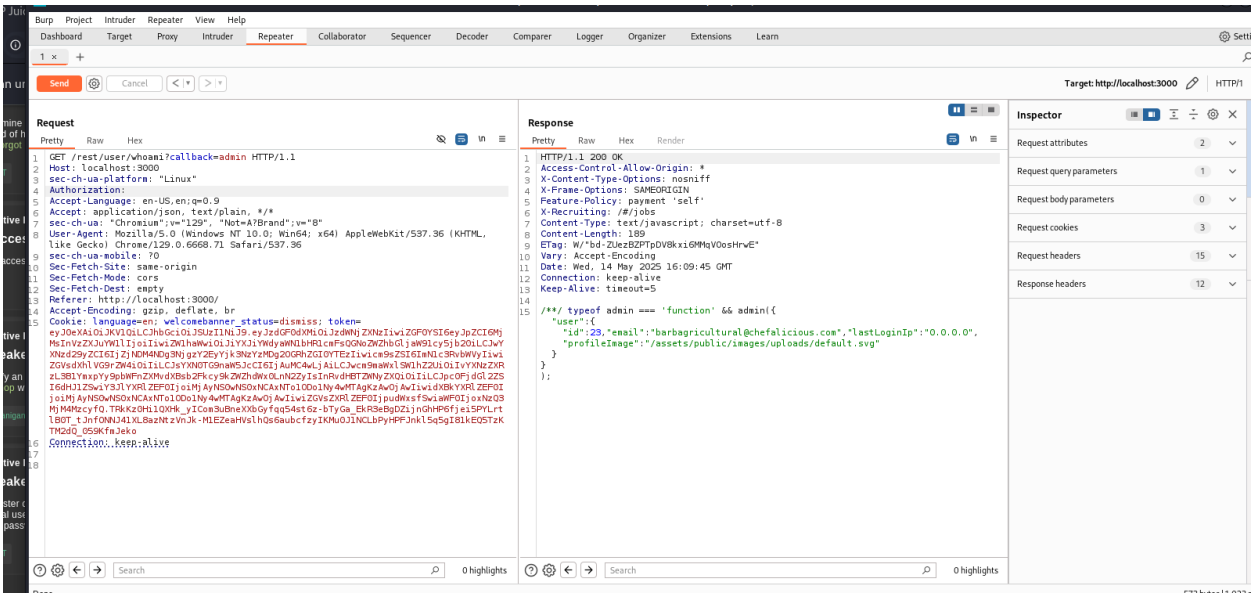
The endpoint fails to validate authentication credentials or user permissions when the callback parameter is provided, exposing sensitive user data to unauthenticated requests.

Steps to Reproduce (in a vulnerable test environment):

1. Access the application and log in with any valid user account.
2. Intercept the HTTP request to the /usr/whoami endpoint using a proxy tool (e.g., Burp Suite).



3. Modify the request by changing the endpoint to /usr/whoami?callback=admin.
4. Remove any Authorization Bearer token or session cookies from the request.



5. Send the modified request using a repeater tool.
6. If vulnerable, the response will return sensitive data associated with the user logged in, without any authentication checks.

Impact:

- Unauthorized access to any user's sensitive data, including personal details or credentials.
- Potential for account takeover by accessing data of privileged users (e.g., admins).
- Exposure of confidential information, enabling further attacks like phishing or fraud.
- Complete bypass of access controls, compromising the application's security.

3) Broken authentication – Reset password

Severity: Critical

Description:

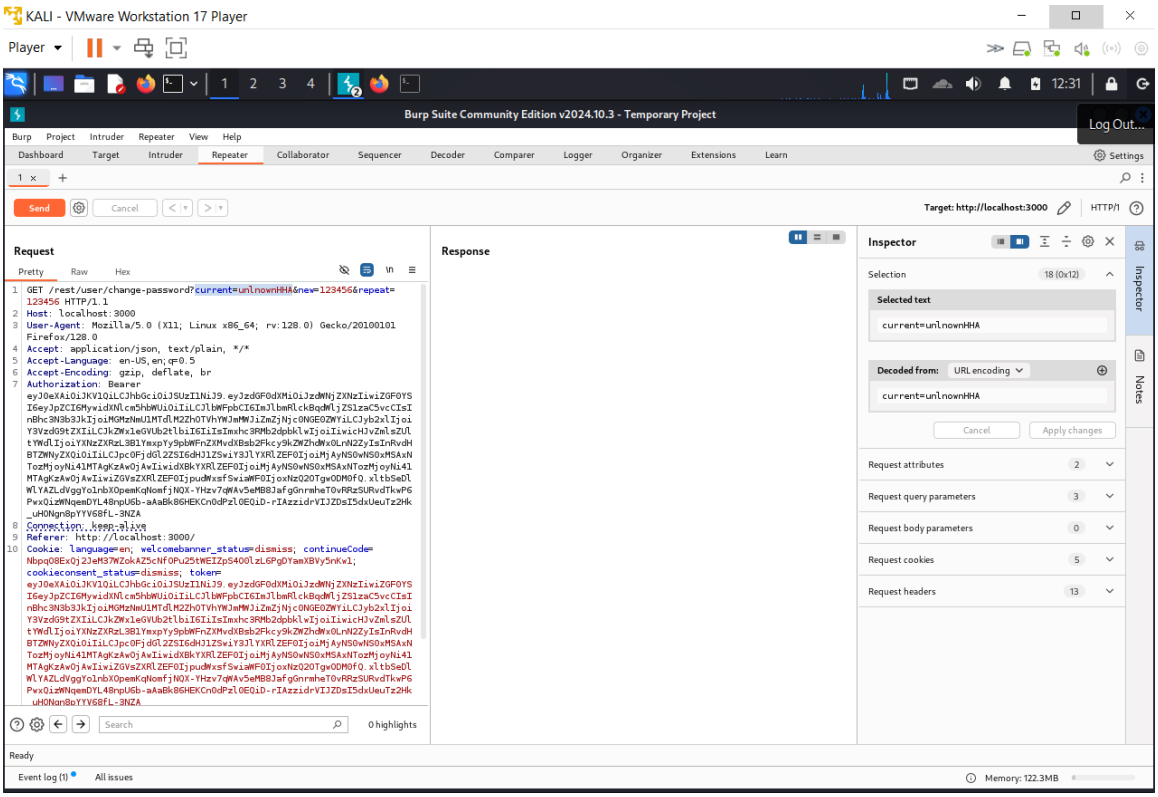
The application allows authenticated users to change the password of their own account without verifying the current password. This weakness can be exploited to change the password of any known user by manipulating the password change request. In this case, the attacker successfully changed the password for the user bender@juice-sh.op without knowing the original password, resulting in full account takeover.

Affected Component:

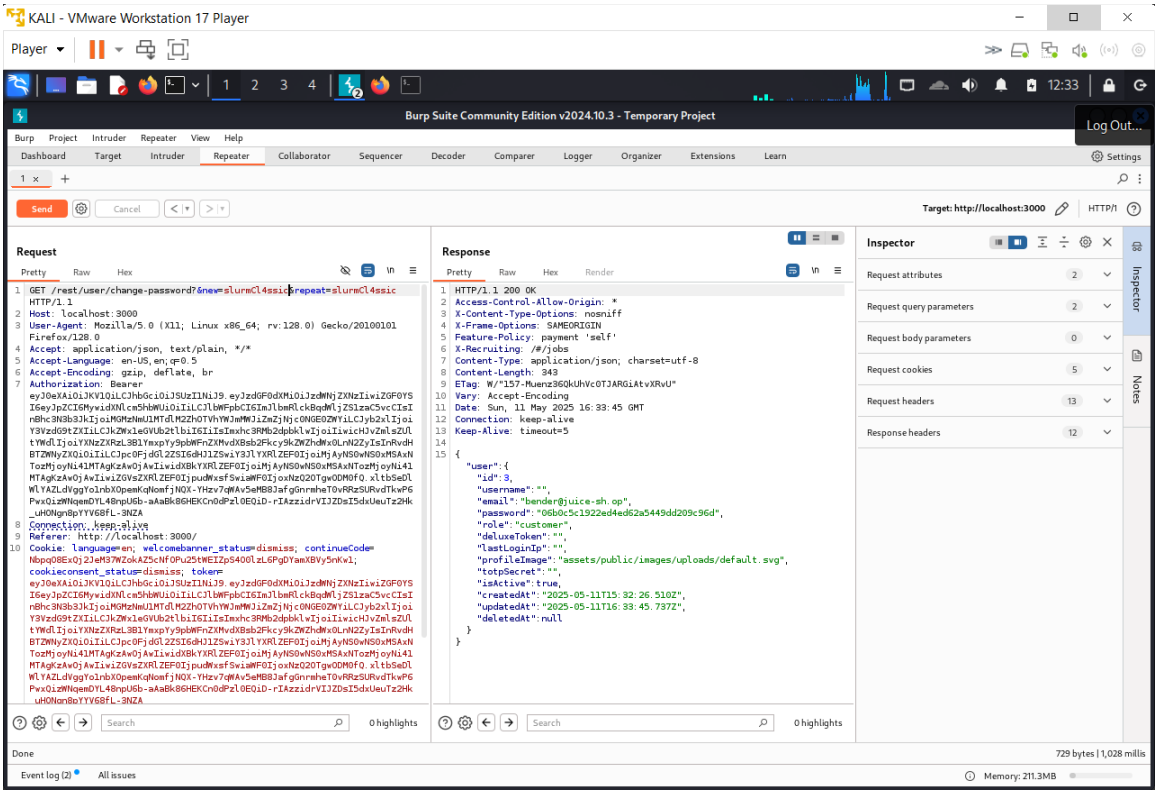
/api/Users/change-password – Password change endpoint

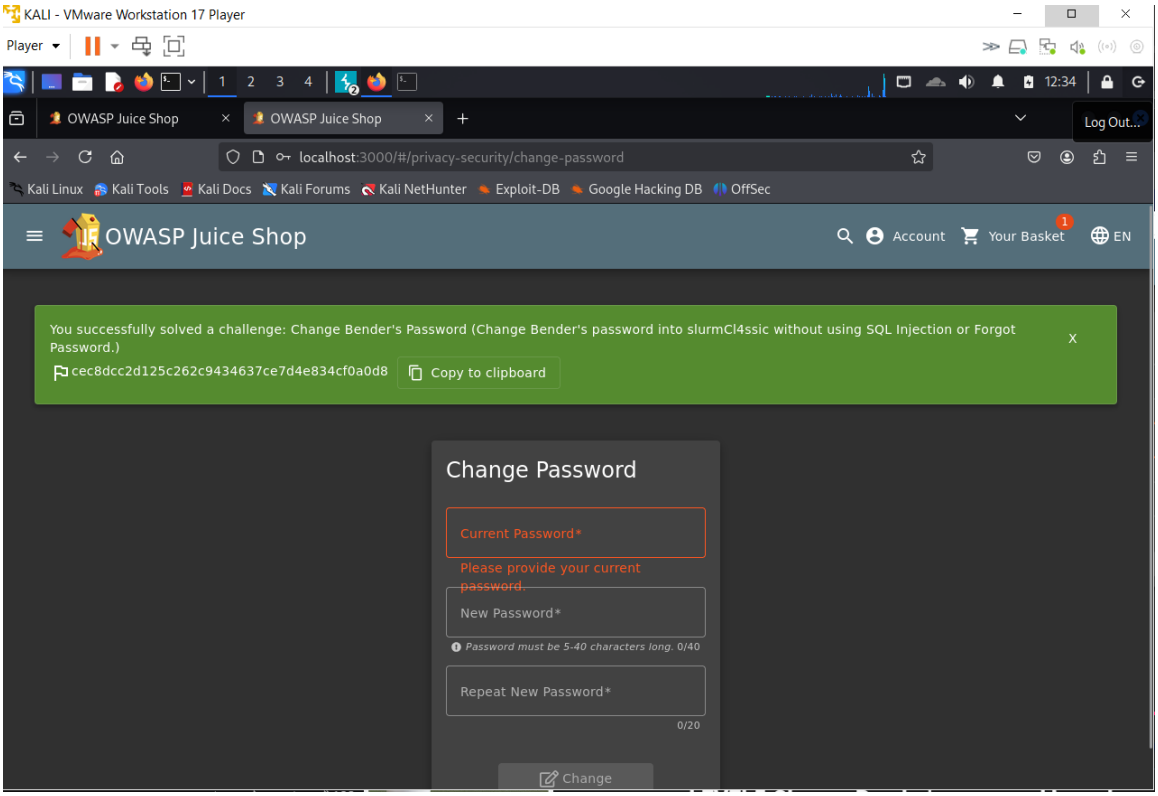
Steps to Reproduce:

1. Logged in with a low-privileged user account.
2. Navigated to the "Change Password" form and intercepted the request using Burp Suite.
3. Observed the following request structure:



4 - Removed the "current" password field entirely and modified the "new" field to:





4) Supply chain attack

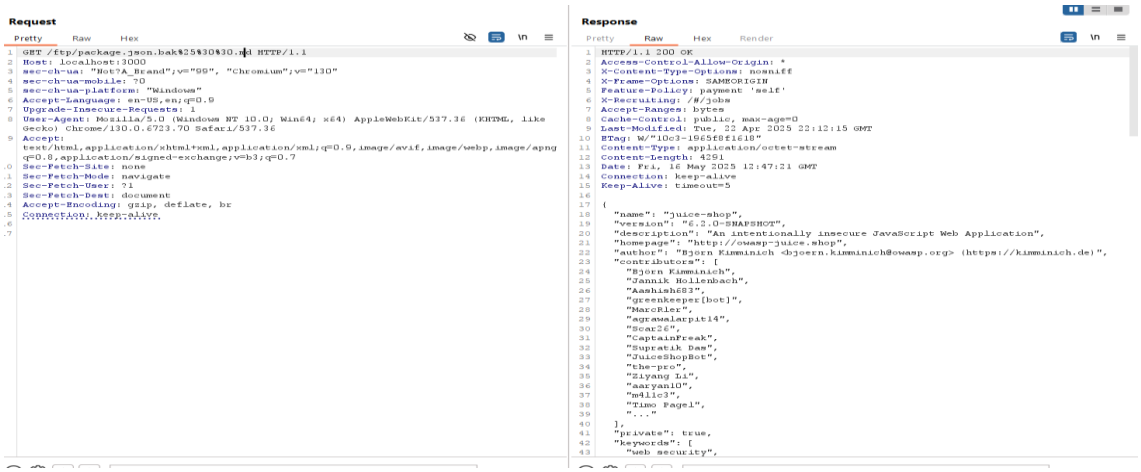
Severity: Critical

Description:

The package.json.bak file revealed a development dependency, eslint-scope@3.7.2, which was compromised in a real-world supply chain attack in 2018. The malicious version included code to steal npm tokens and enable remote code execution, posing a significant risk to developer credentials.

Steps to Reproduce:

1- Access package.json.bak using the Poison Null Byte bypass (http://localhost:3000/ftp/package.json.bak%00.md).



2- Identify eslint-scope@3.7.2 in the devDependencies section.

ESLint-scope Vulnerability (CVE-2018-3728)

Overview

The ESLint-scope vulnerability (CVE-2018-3728) was a security issue discovered in the `eslint-scope` package, which is a dependency of ESLint, a popular JavaScript linting utility.

Vulnerability Details

- **Type**: Arbitrary code execution via malicious npm package
- **Affected versions**: All versions prior to 3.7.2
- **CVE ID**: CVE-2018-3728
- **Disclosure date**: March 2018

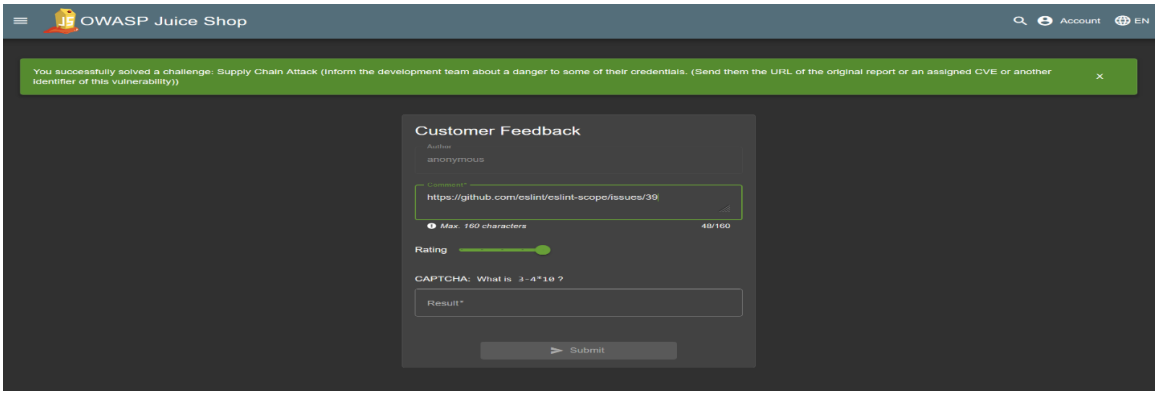
Impact

The vulnerability allowed attackers to execute arbitrary code by exploiting the way `eslint-scope` handled certain npm packages. Specifically:

- Malicious npm packages could modify the `eslint-scope` package during installation
- This could lead to arbitrary code execution on systems where vulnerable versions were installed

3- Research the package and confirm its compromise (e.g., via https://github.com/eslint/eslint-scope/issues/39).

4- Report the vulnerability to the development team via the contact form at http://localhost:3000/#/contact, including the package name, version, risk, and a reference URL.



impact:

Erosion of trust in the software supply chain, as malicious dependencies can propagate to production.

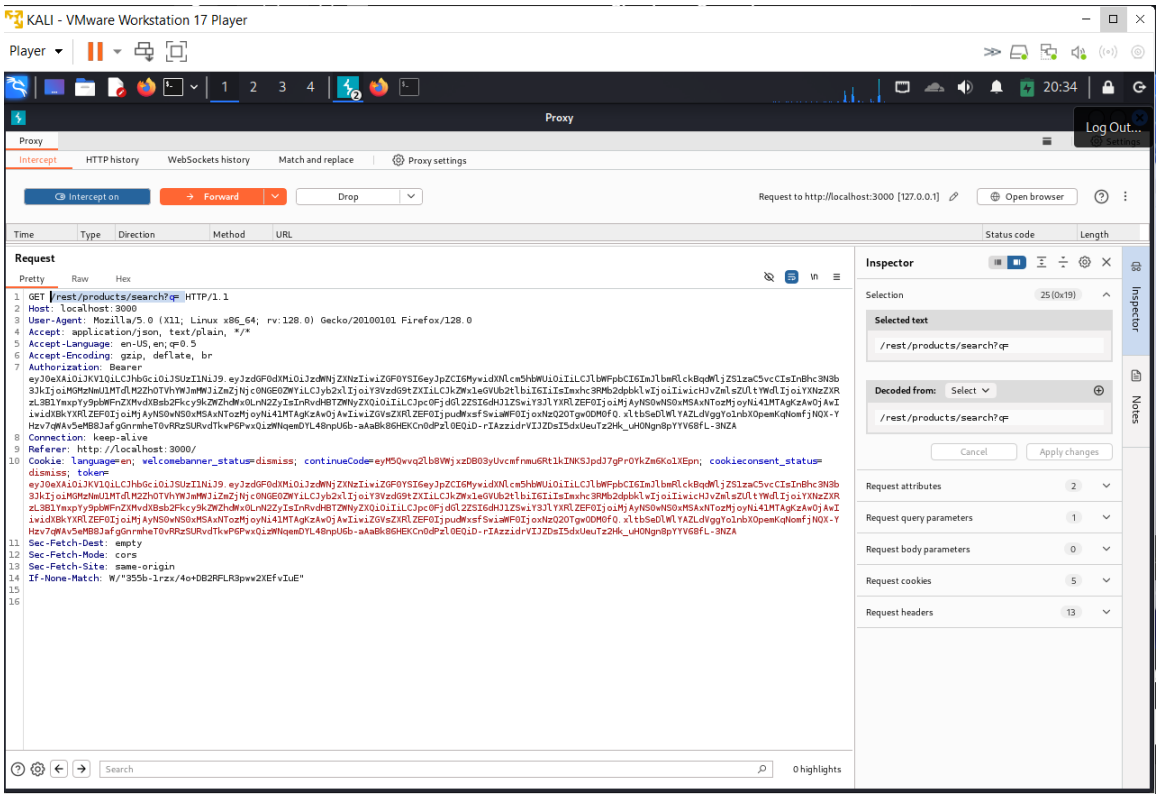
5) SQL injection - search quey – 3

Severity: critical

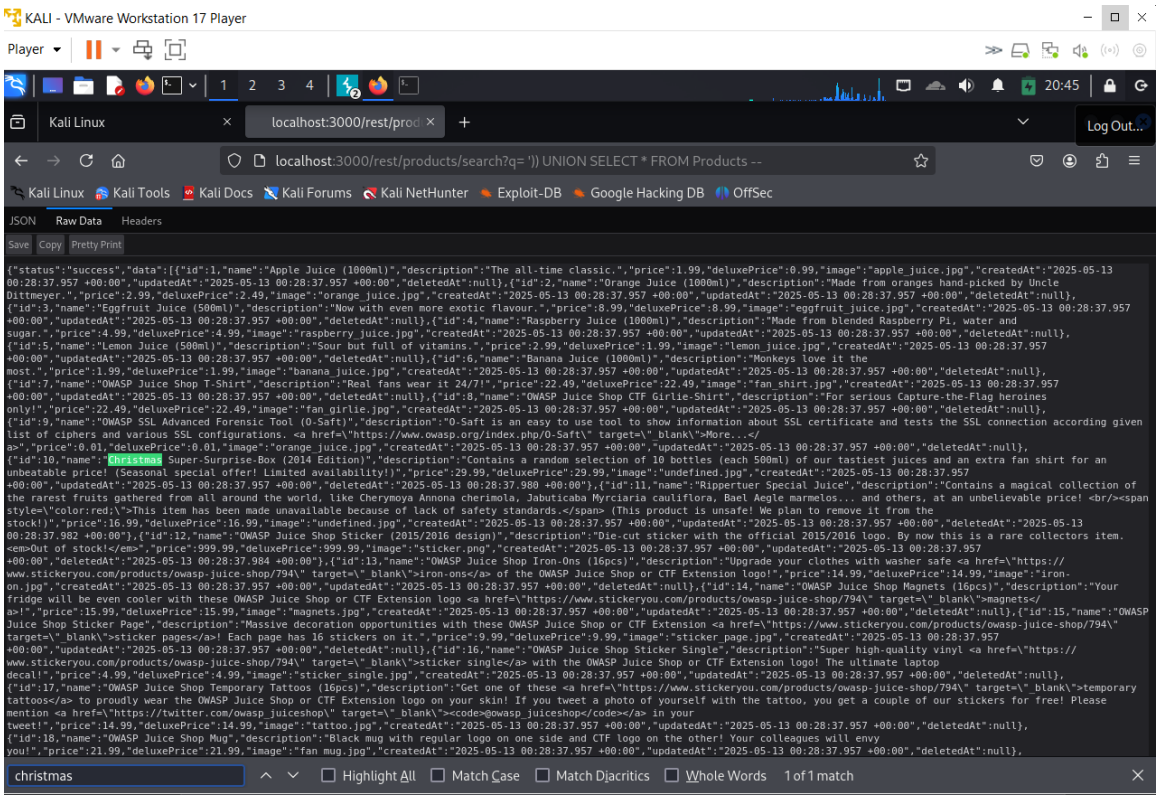
Description:
A SQL Injection vulnerability was identified in search query allows users to inject union query to dump all the data of the products using that I found the id of , and inject various sql payloads that leads to dump all the data of the users or database schema

Steps to Reproduce attack 1 :

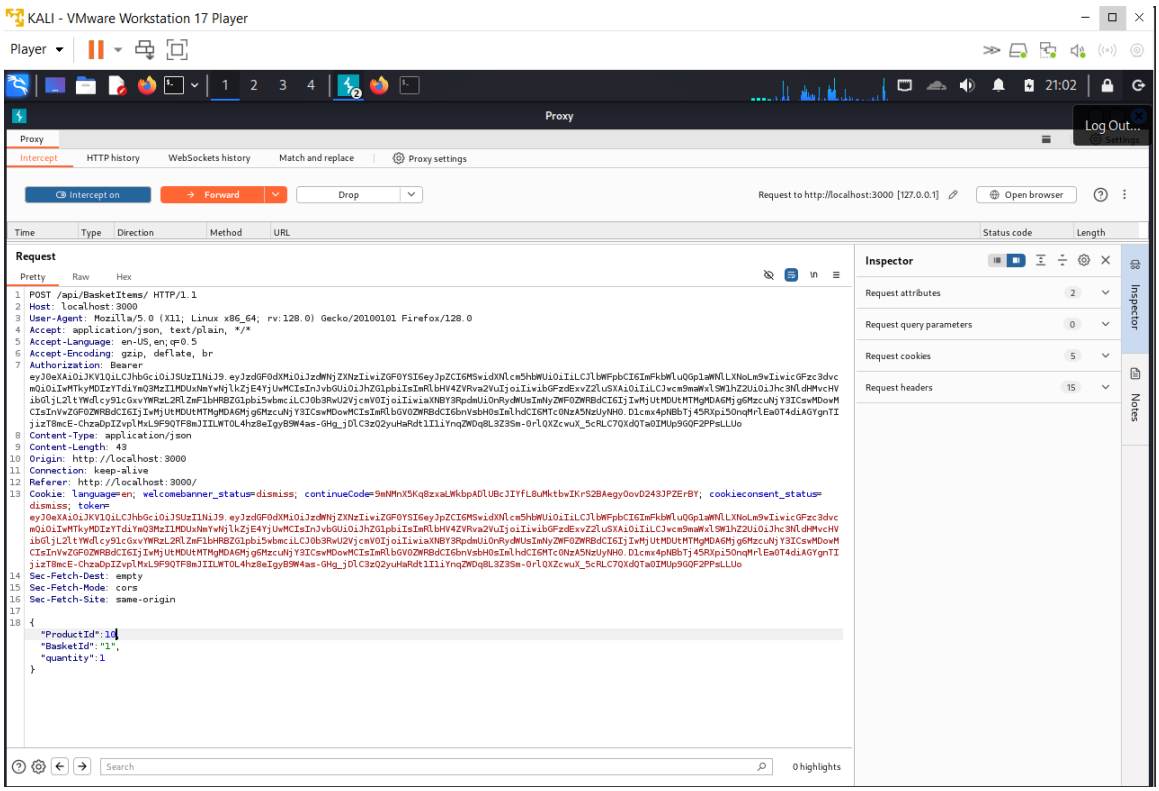
1. Discovered the search query param :



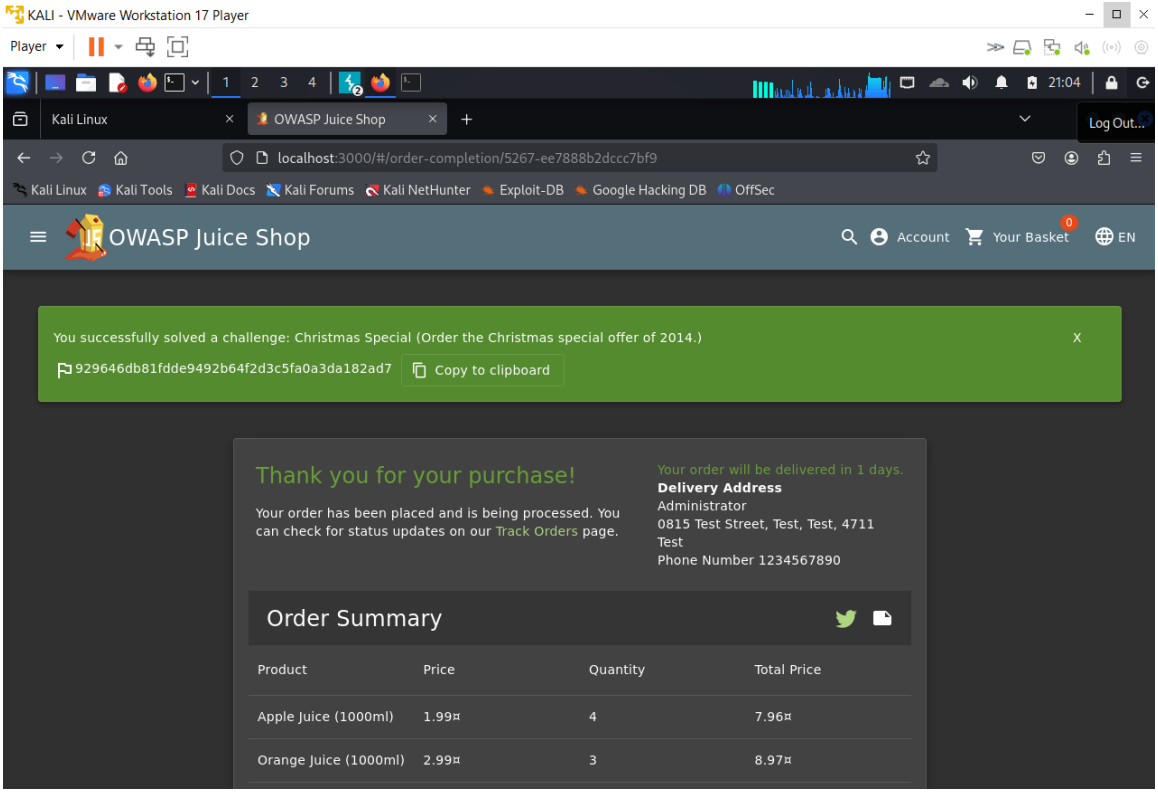
2. Inject the payload .

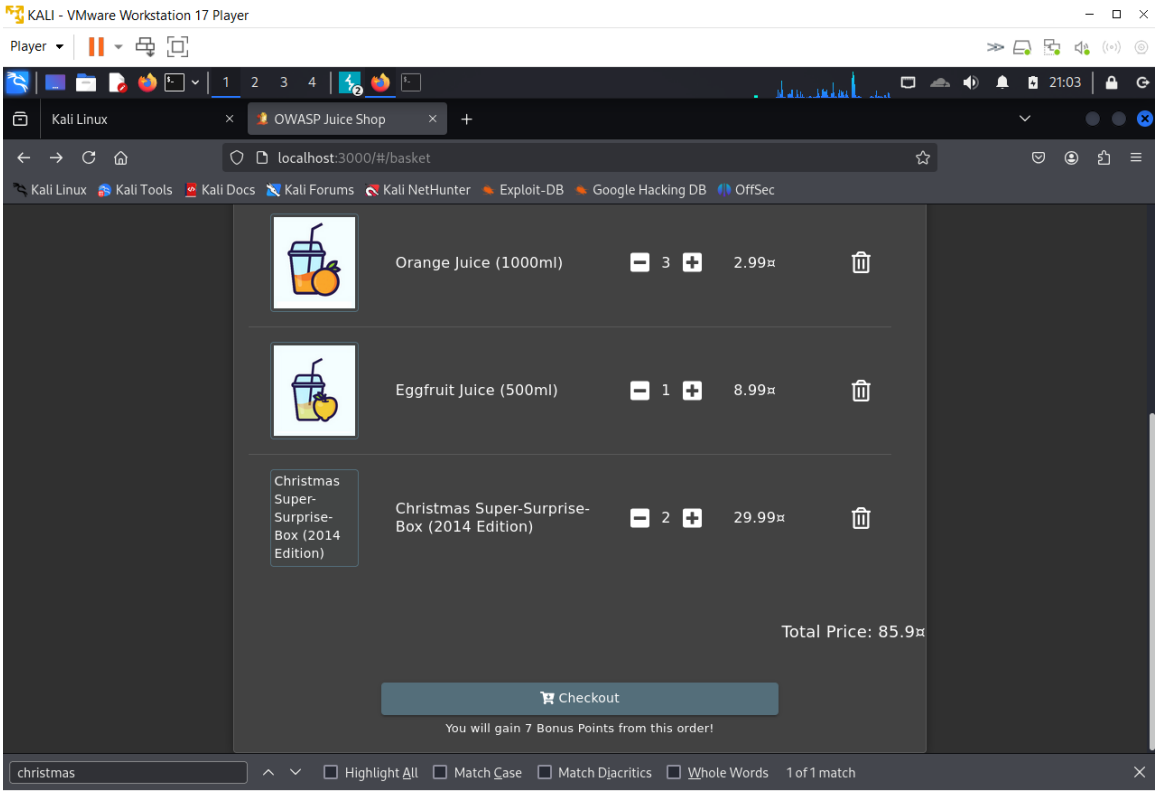


3. Use the found data in the order request :



4. Successfully the product was added





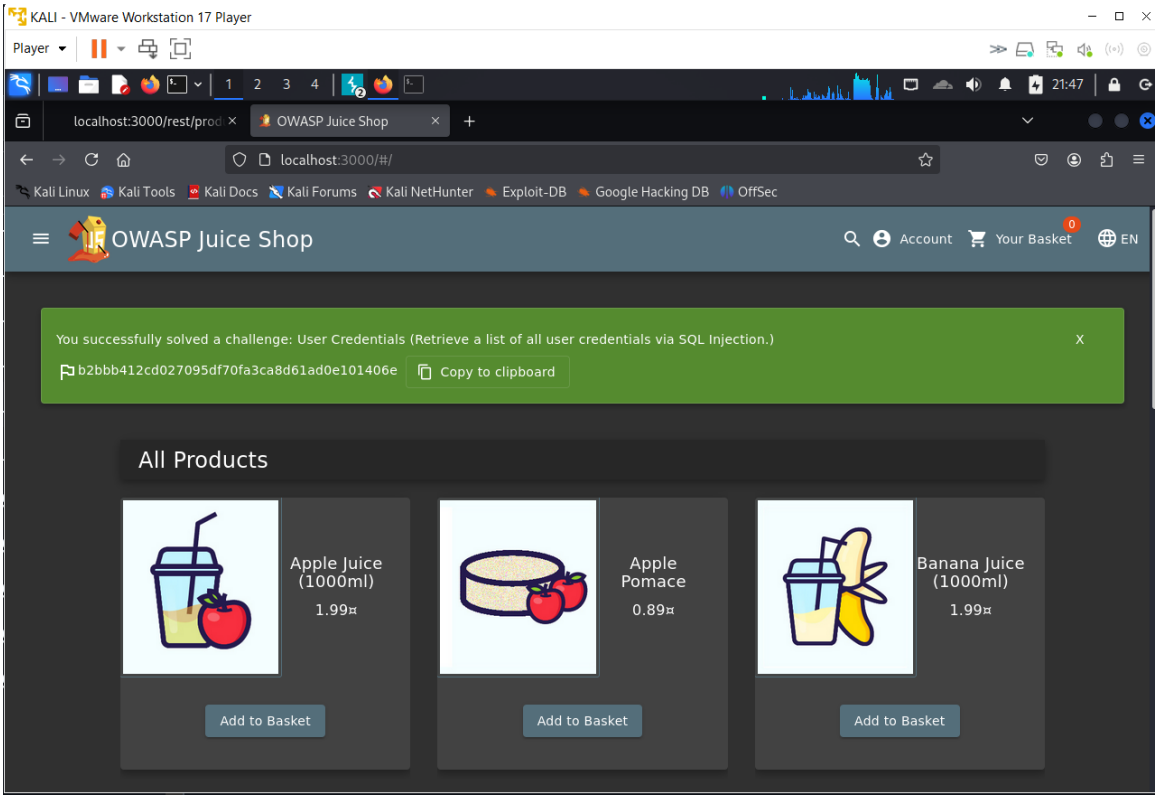
Steps to Reproduce attack 2 :

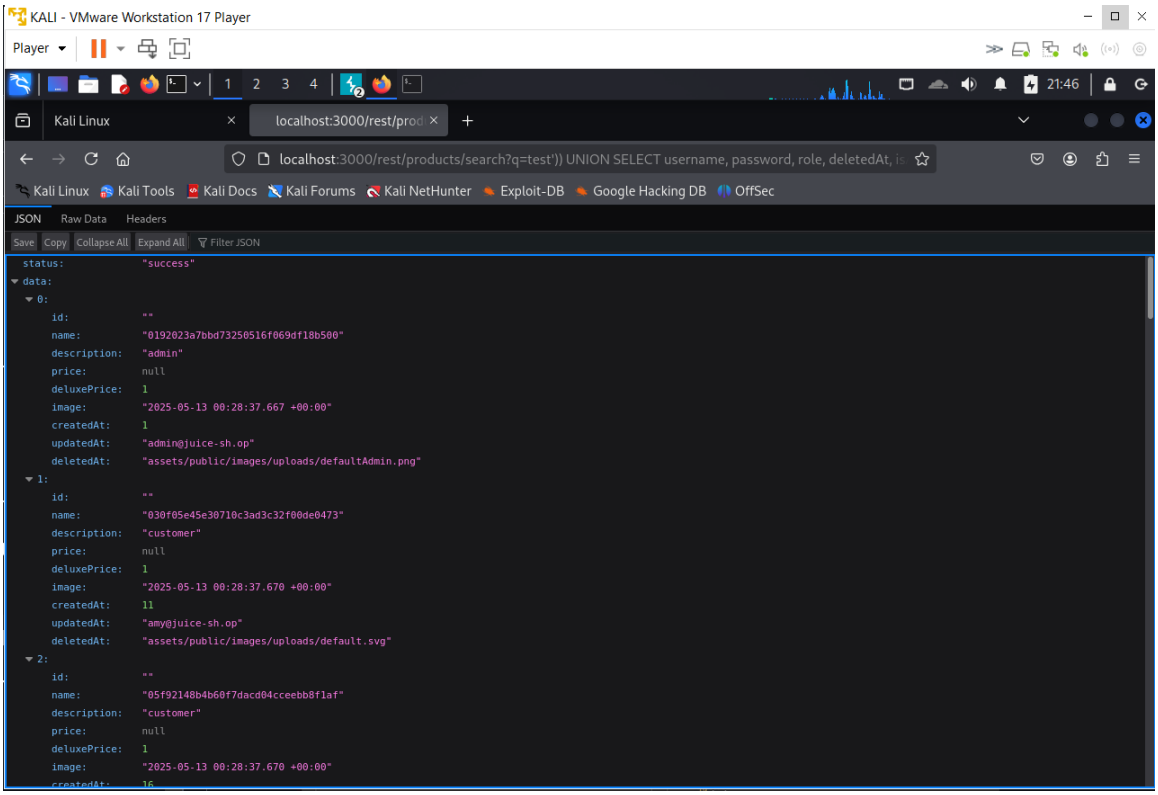
1- go to / rest/products/search?q=

2- tried **'OR 1=1 --** and it bypassed , there is a logic problem
3 – tried **'UNION SELECT * FROM USERS --** it gives

4- inject this payload test')) UNION SELECT username, password, role, deletedAt, isActive, createdAt, id, email, profileImage FROM USERS

5 – boom ! all users data



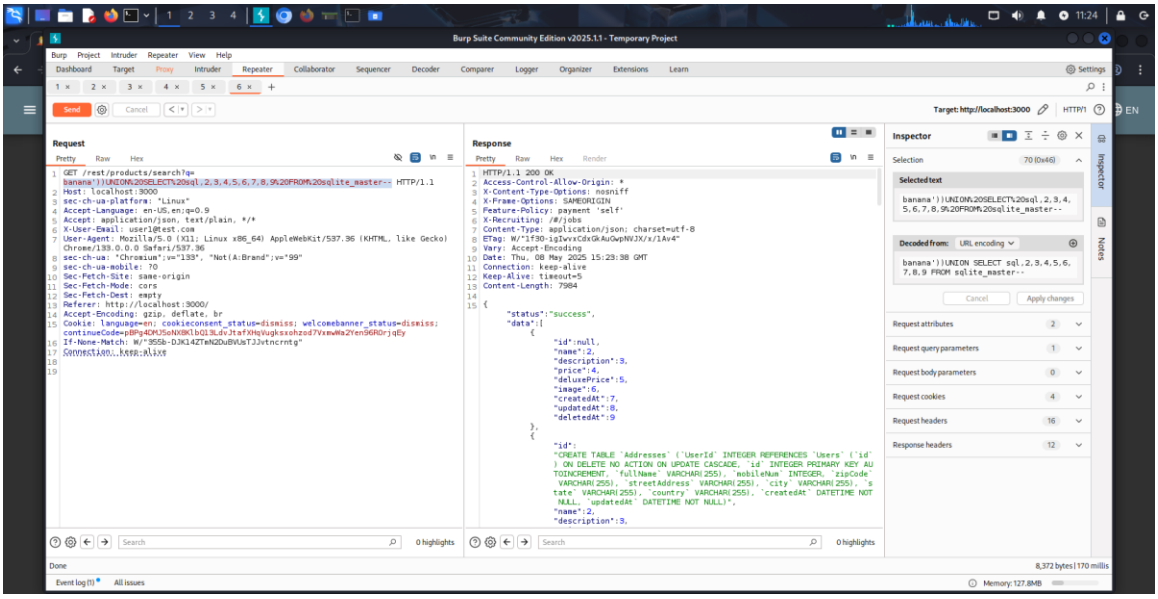


Steps to Reproduce:

1. Intercepted the search request using **Burp Suite**.
2. Modified the q parameter to include the following payload:

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

3. Forwarded the modified request.



4. The server responded with the following JSON output that included part of the SQL schema:

```
{
 "id": "CREATE TABLE \"Addresses\"...",
 "name": 2,
 "description": 3,
 ...
}
```

This output contained full SQL CREATE TABLE statements, including table names like Users, Addresses, and all column definitions.

Impact :

- Attacker gains visibility into internal database structure (Users, Addresses, etc.).
- Can identify relationships and sensitive fields (e.g., password, email, mobileNum).
- Facilitates targeted SQLi or data extraction attacks.
- Compromises database confidentiality and increases risk of full exploitation.

6) Broken access control – access log

Severity: Critical

Description:

The application improperly exposes system access logs through an unprotected /support/logs endpoint. By modifying the URL path from

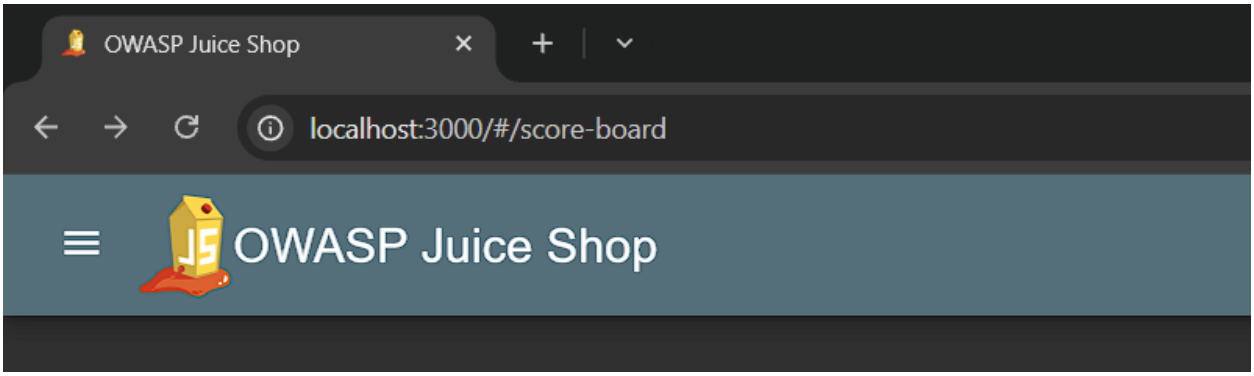
/score-board to /support/logs, attackers can view raw server logs containing sensitive operational data without authentication.

Affected Component:

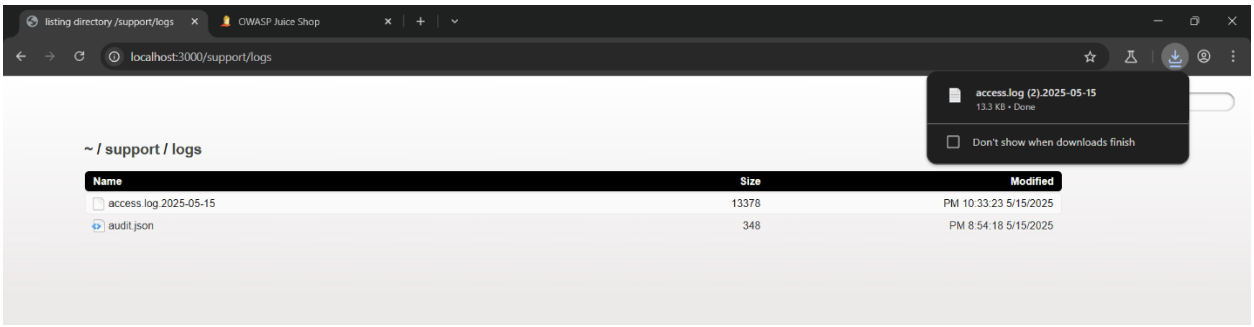
Unprotected endpoint: http://localhost:3000/support/logs

Steps to Reproduce:

1. While authenticated, navigate to: http://localhost:3000/#/score-board

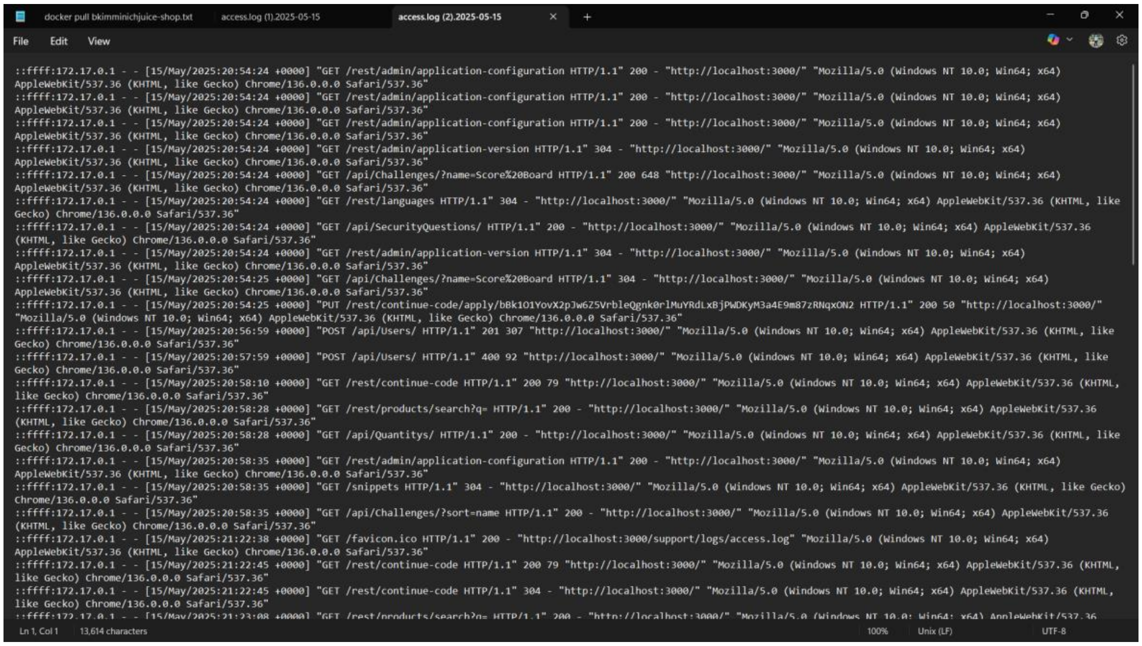


2. Manually modify the URL to: http://localhost:3000/support/logs



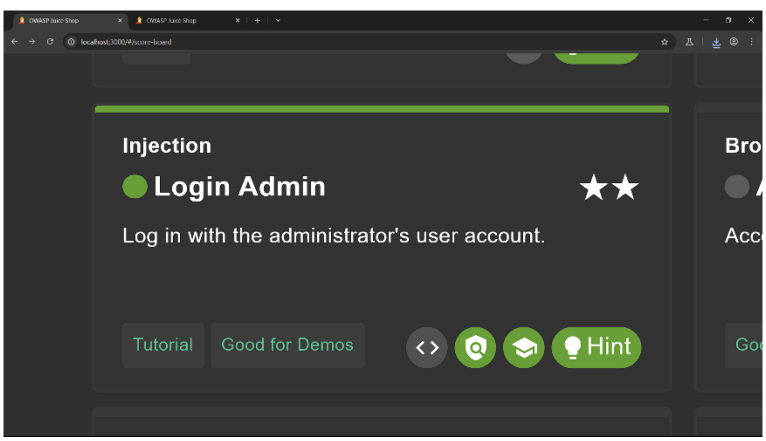
3. Observe the server responds with:

- Full access logs in plaintext



- Timestamps of all requests
- User IP addresses
- HTTP methods and endpoints accessed
- Session tokens (if logged)

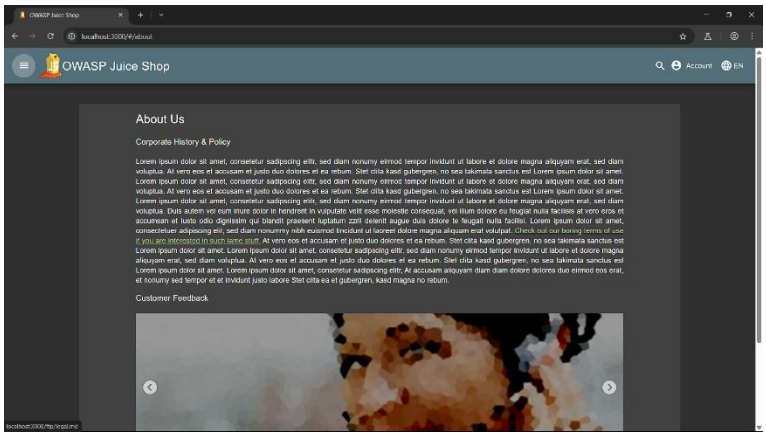
4.

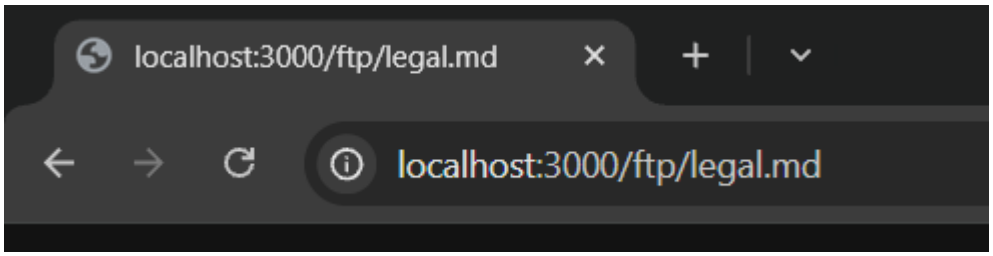


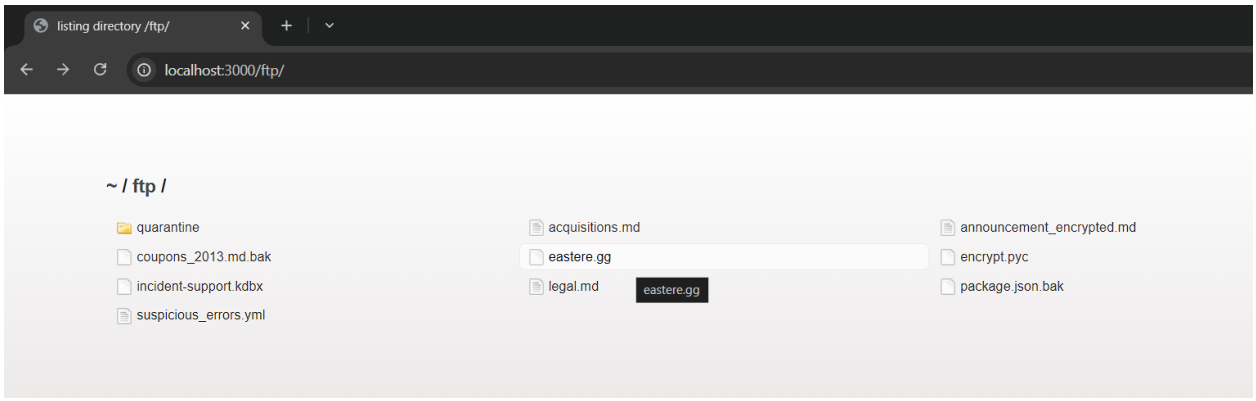
7) Improper input validation – Poison null byte

Severity: High

- **Description :** The application fails to sanitize null bytes (%00) in file paths, allowing attackers to bypass extension-based security controls. By appending %00.md to a restricted filename (e.g., eastere.gg), the system incorrectly validates the file extension, enabling unauthorized access to blocked files.
- **Exploitation :**
  1. Initial Access:
     Navigated to About Us → Clicked "Legal Notice" (http://localhost:3000/ftp/legal.md)
     Modified URL to list FTP directory: http://localhost:3000/ftp/



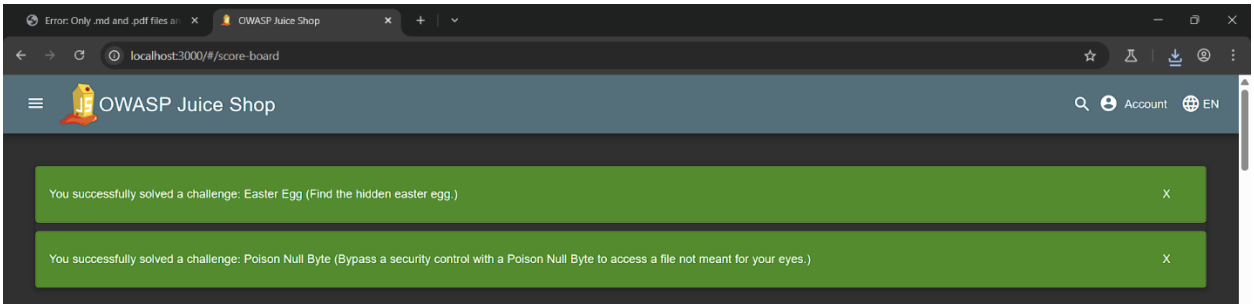




  2. Identify Target File: Discovered restricted file eastere.gg (hidden "Easter Egg" challenge file)

3. Bypass Extension Filter:
   Direct access blocked: http://localhost:3000/ftp/eastere.gg → Error: "Only .md and .pdf files allowed" Injected null byte:

Result: Successfully retrieved eastere.gg contents



8) Sensitive data exposure – file download endpoint

Severity : High

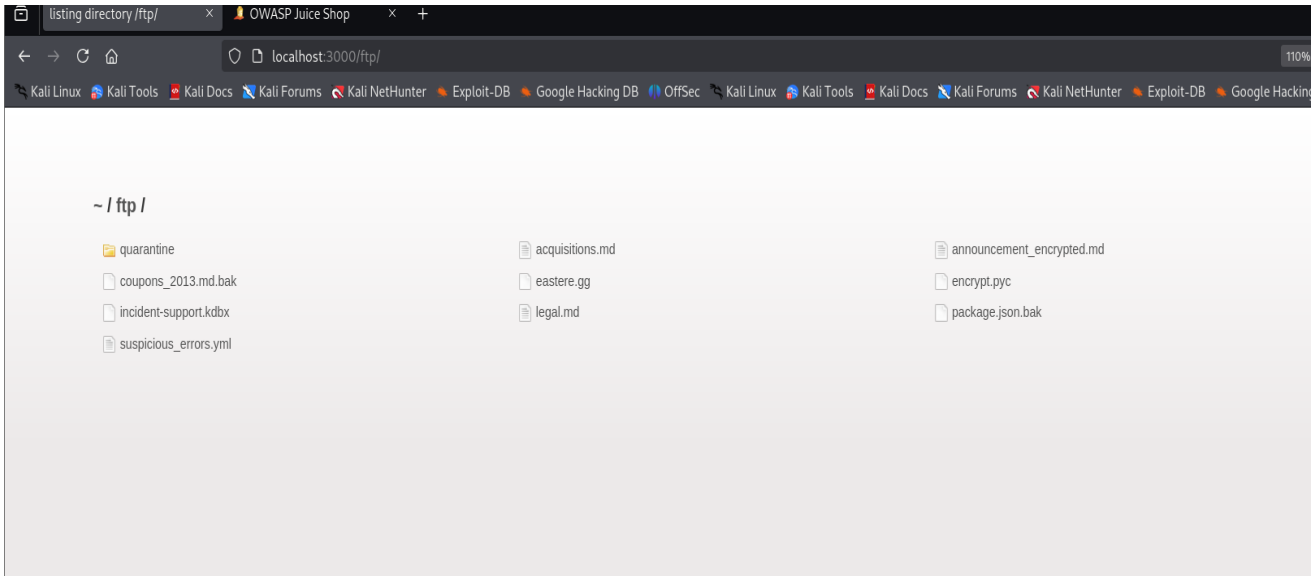
Description:
Null Byte Injection is a technique that exploits how some backend systems handle null characters (`%00`) to bypass input validation — often tricking the server into treating part of a file name or input as complete while ignoring the rest. This is commonly used to bypass file extension filters or access restricted files.

Affected Component:
File download endpoint
`/ftp/:file`

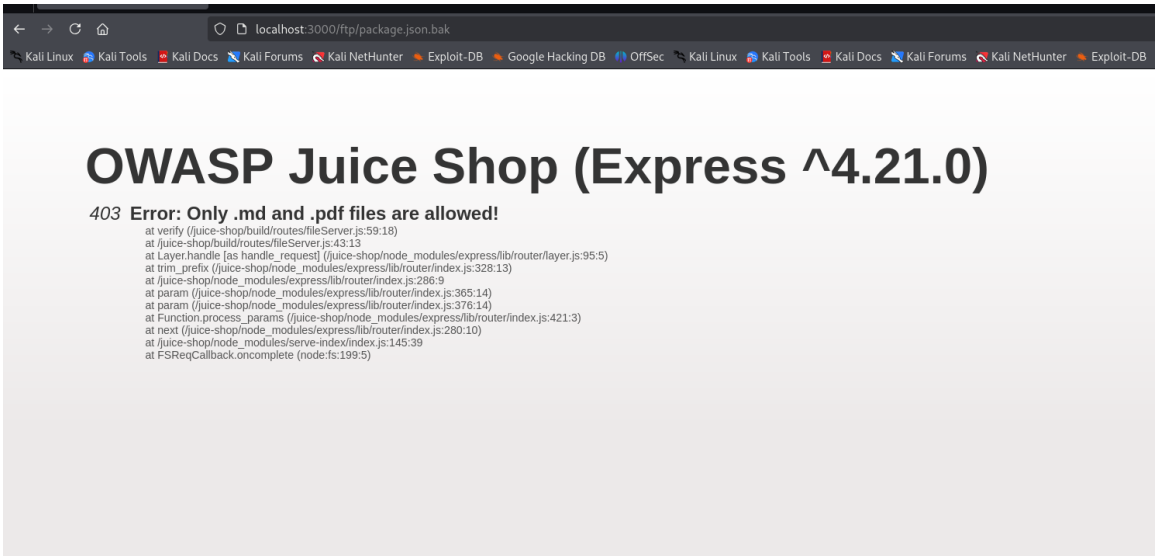
This route checks for allowed file extensions (e.g., `.md`, `.pdf`) but may mishandle encoded null bytes in some configurations.

Steps to Reproduce (in OWASP Juice Shop):

1. Go to:
2. `ftp/package.json.bak%2500.md`



3. The `%2500` is double-encoded and gets decoded by the server into `%00` (null byte).
4. If the server is vulnerable, it sees the full string as `.md`, but actually opens `package.json.bak`.



Impact:

- Bypasses file extension restrictions.
- Allows access to sensitive backup files (e.g., `.bak`, `.old`, `.config`).
- Can lead to exposure of credentials, app secrets, or source code.

So what is null byte injection?

Null byte injection is a technique where an attacker inserts a special null character (`%00`) to trick the server into cutting off a string early. It's used to bypass security checks, like file extension filters, by making the server think the file ends before the real extension.

Here's what we do:

We try to trick the server by writing the file name like this:

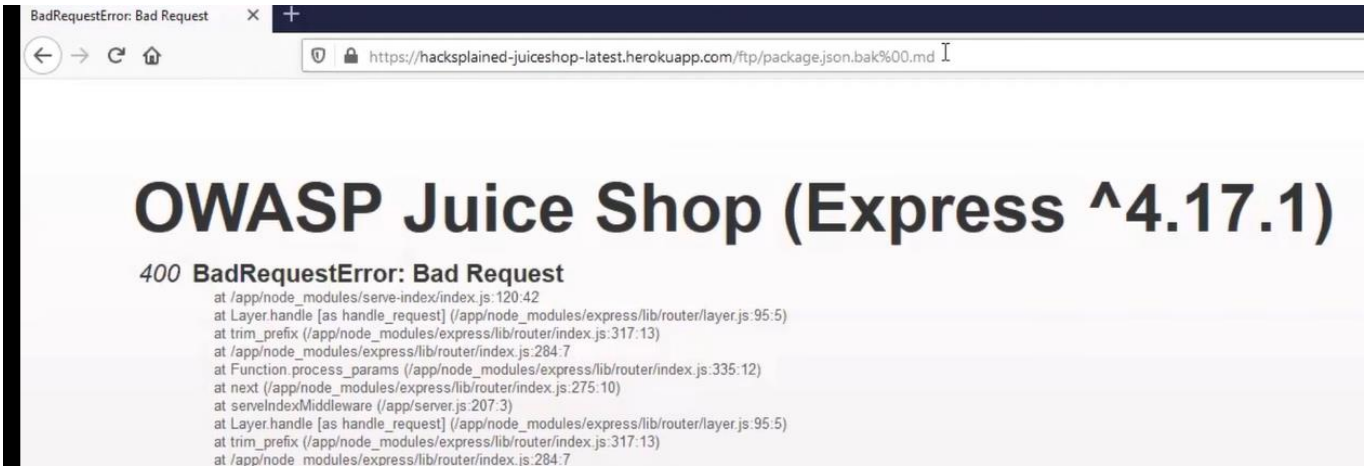
```
/ftp/package.json.bak%00.md
```

Let's explain it step by step:

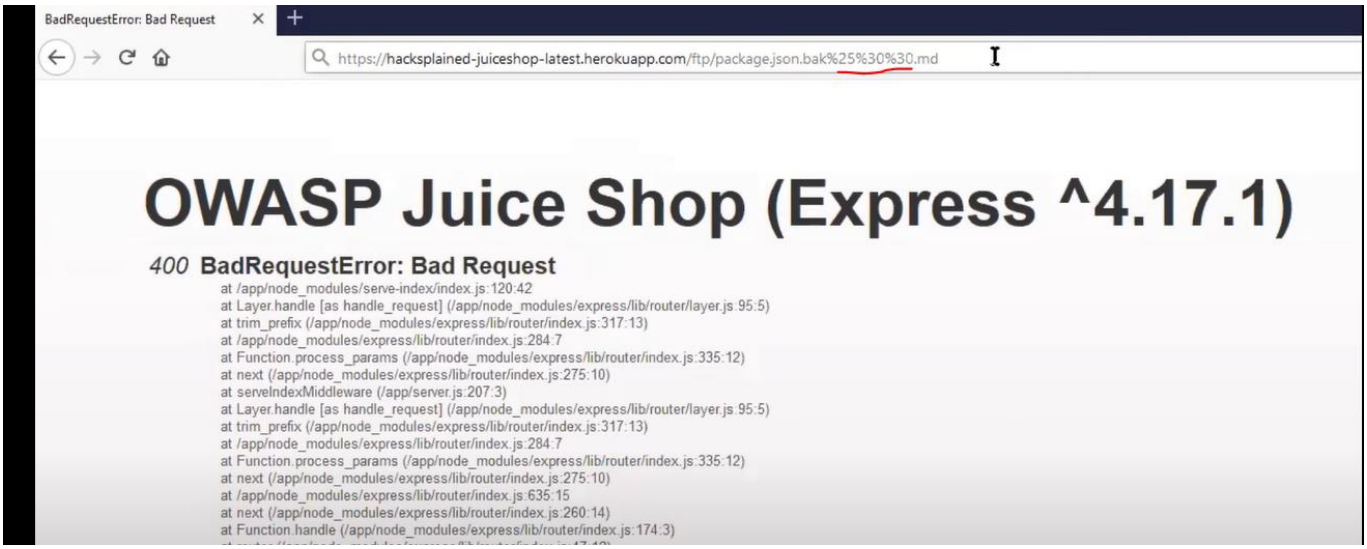
- `%00` is a **null byte**, and in some systems, it means "the end of the string".
- So when the server checks the file name, it sees:
- `package.json.bak%00.md → Ends with .md? Yes! ✓`
- But when the system **opens the file**, it may stop reading at `%00`, so it actually reads:
- `package.json.bak`

So the server **thinks it's a `.md` file**, but really it opens the **.bak file**!

But there was an error when we tried this:



Showing bad request so lets try to give it encoded in the url:



And now it works:

9) Sensitive data exposure – GDRP Data theft

Severity : High

- **Description**

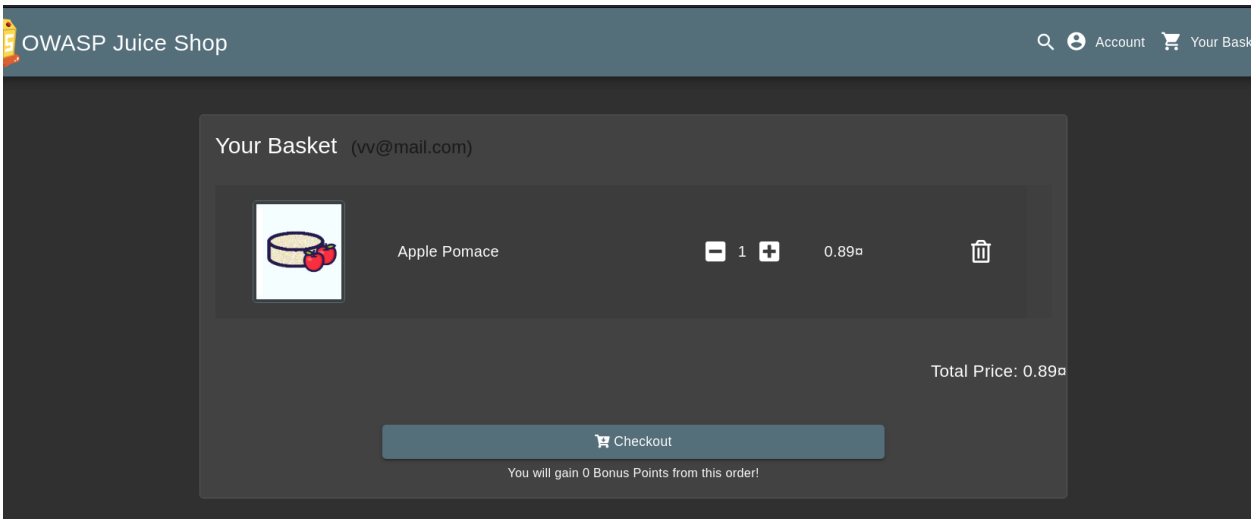
Sensitive Data Exposure occurs when an application fails to adequately protect sensitive information, such as user email addresses. In OWASP Juice Shop, the /rest/track-order endpoint exposes user email addresses in a JSON response by replacing vowels with asterisks (*), which can be easily reversed to reveal the original email. Additionally, when exporting data via the "Privacy and Security" settings, sensitive user details are exposed. This flawed obfuscation also allows an attacker to register a similar email (e.g., odmin@juice-sh.op for admin@juice-sh.op) and potentially access admin data, completing the "Privacy Policy Inspection" challenge.

- Affected Component:

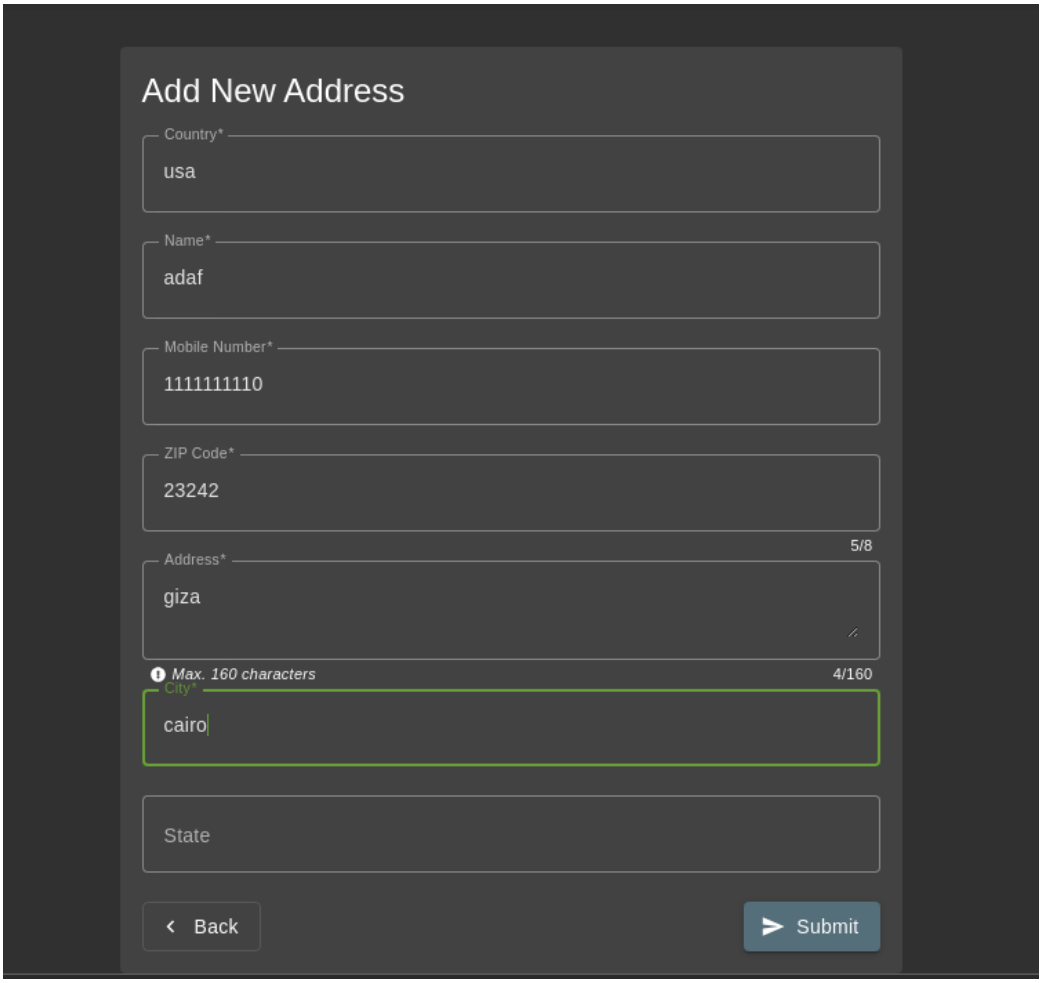
Track order endpoint
/rest/track-order
The endpoint exposes obfuscated email addresses in JSON responses, and the "Privacy and Security" data export feature reveals sensitive user details without proper protection.

Steps to Reproduce (in OWASP Juice Shop):

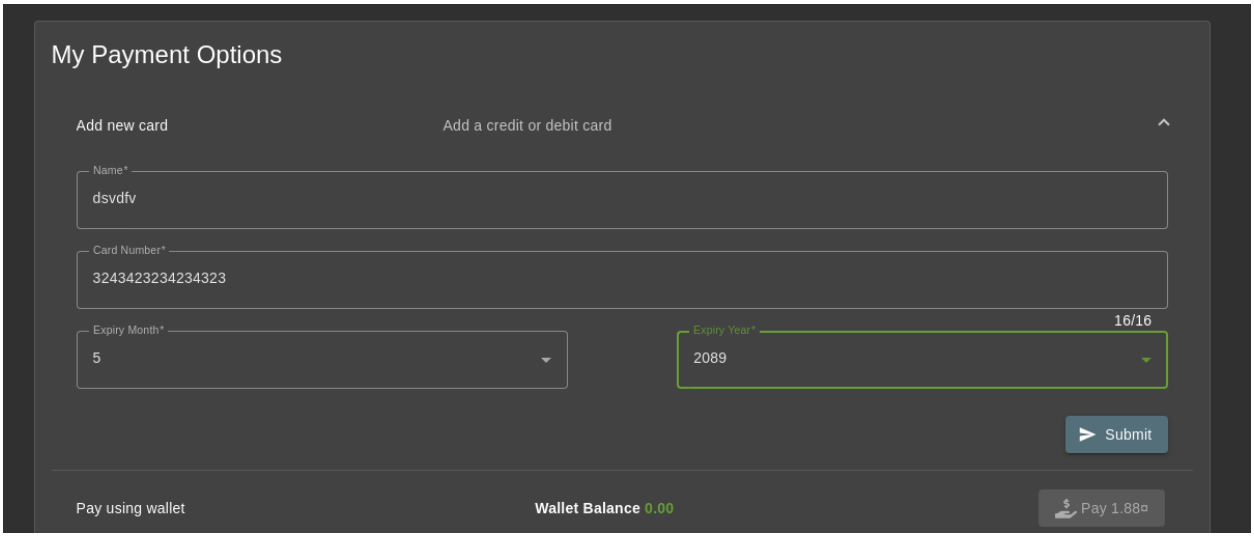
1. Register a new user at http://localhost:3000/#/register with an email (e.g., testuser@juice-sh.op) and password (e.g., Pass123!).
2. Log in with the newly created account at http://localhost:3000/#/login.
3. Add any product (e.g., "Apple Juice (1000ml)") to the basket.



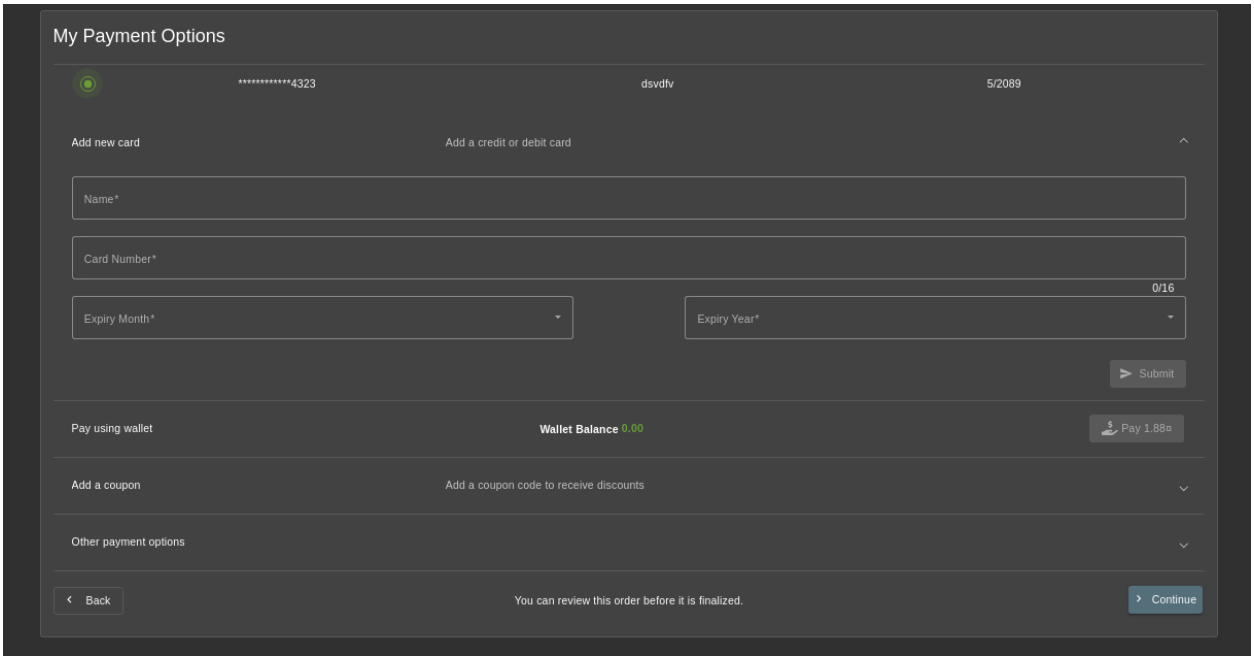
4. Proceed to checkout and add a delivery address (e.g., "123 Main St, City, Country").



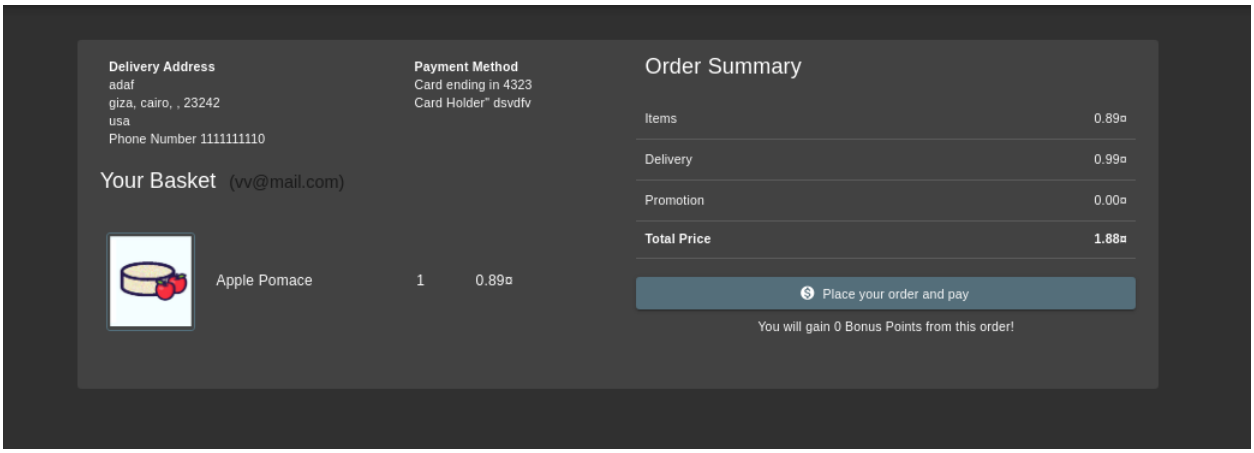
5. Select "One Day Delivery" as the delivery option.



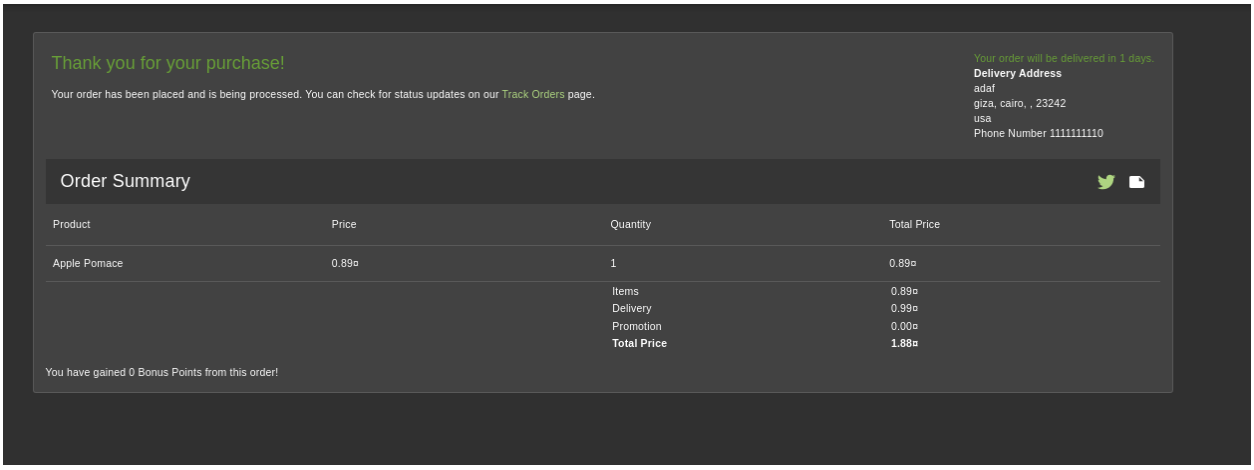
6. Add a new payment card (e.g., card number: 1234 5678 9012 3456, expiry: 12/26, CVV: 123) and select it for payment.



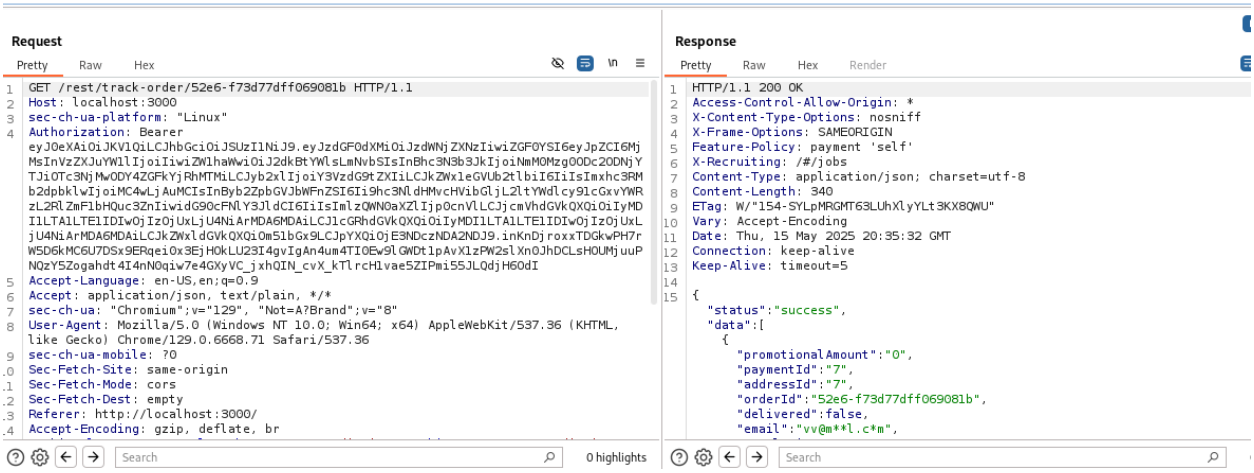
7. Place the order by clicking "Place Order".



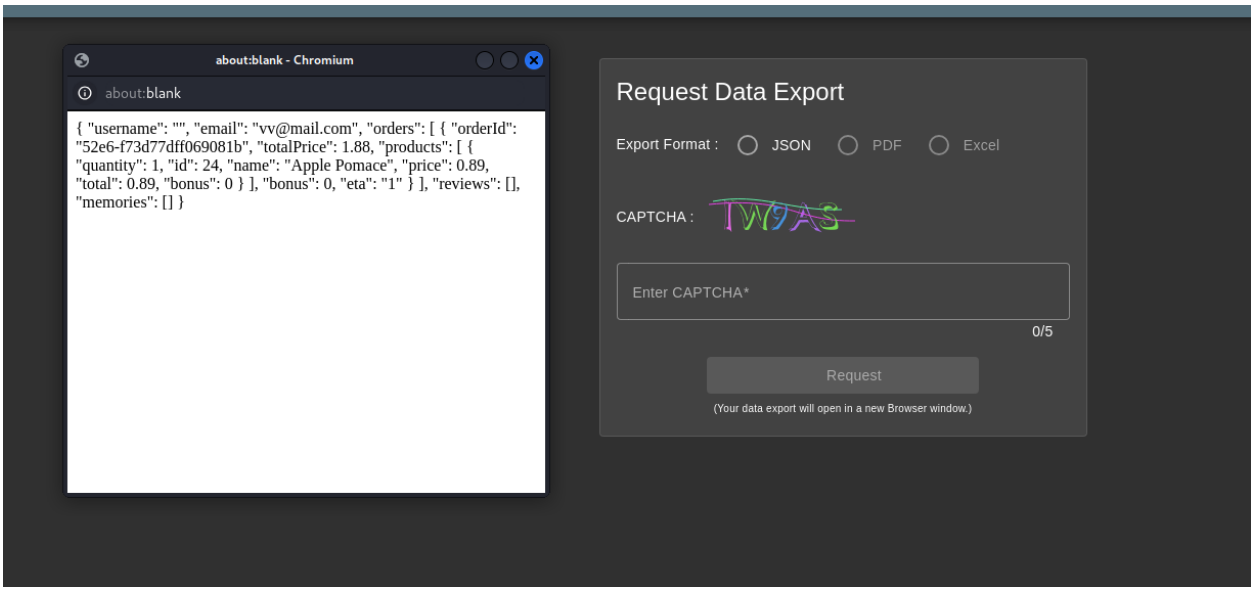
8. Navigate to "Track Order" to view the order details.



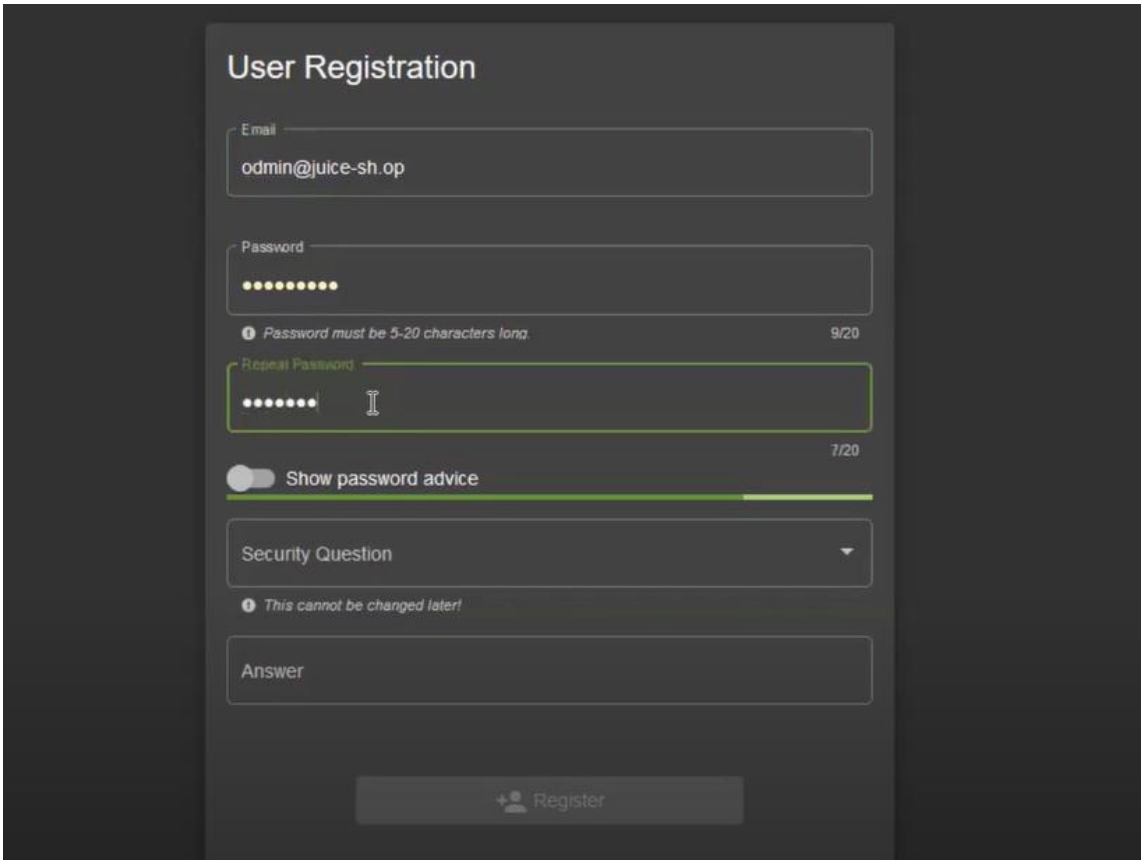
9. Intercept the /rest/track-order request using a proxy tool (e.g., Burp Suite) and examine the JSON response, which shows the email with vowels replaced by asterisks (e.g., t*st*s*r@j**c*-sh.*p for testuser@juice-sh.op).



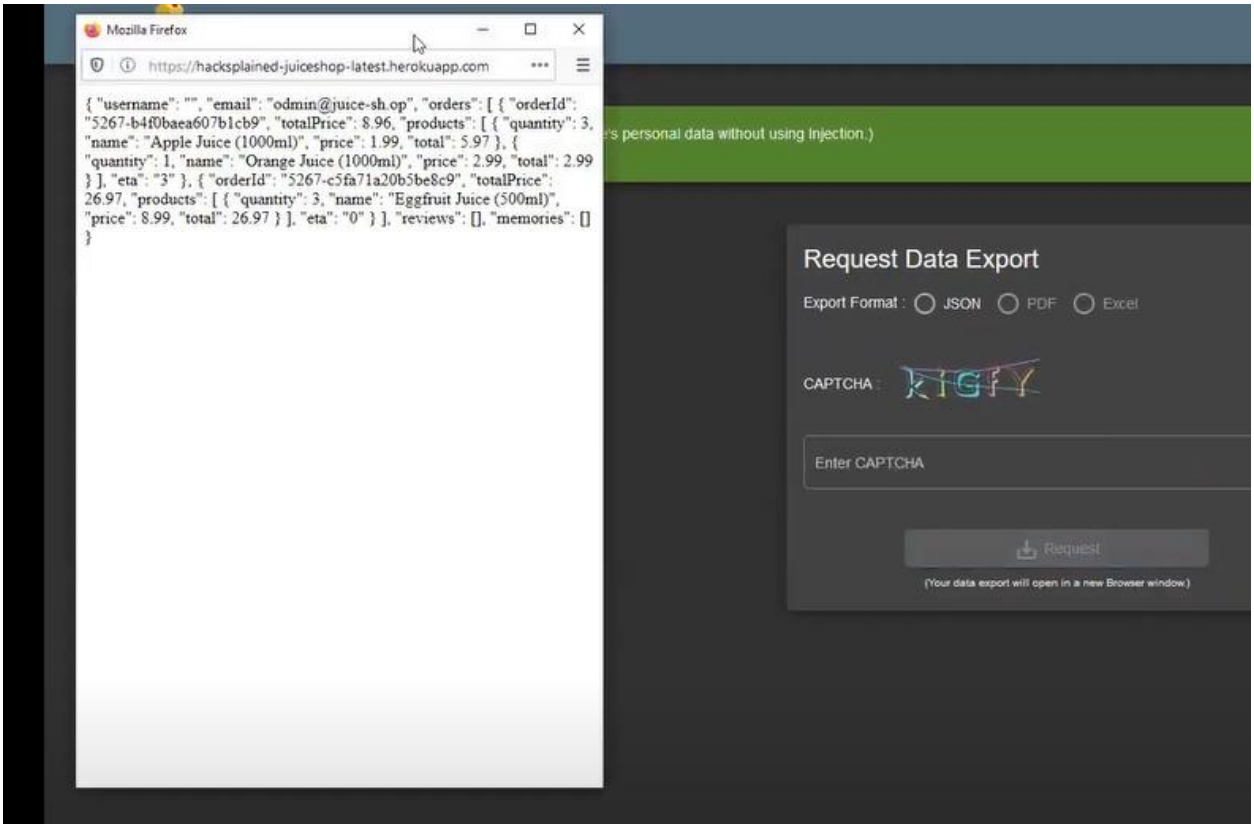
10. Export user data via the "Privacy and Security" settings, revealing sensitive details like the full email, address, and order history.



11. Register a new user with a similar email to the admin's (e.g., odmin@juice-sh.op for admin@juice-sh.op) by mimicking the obfuscation pattern (*dm*n@j**c*-sh.*p).



12. Log in with this account and export data again, gaining access to the admin's sensitive information, such as email and order details, completing the "Privacy Policy Inspection" challenge.



Impact:

• Exposure of sensitive user data, including email addresses, addresses, and order details, through the data export feature.

• Potential reconstruction of obfuscated emails due to predictable vowel replacement, leading to privacy breaches.

• Unauthorized access to admin data by registering a similar email, exploiting the obfuscation flaw.

• Increased risk of account impersonation or targeted attacks using exposed user information.

10) Broken access control – file transfer endpoint

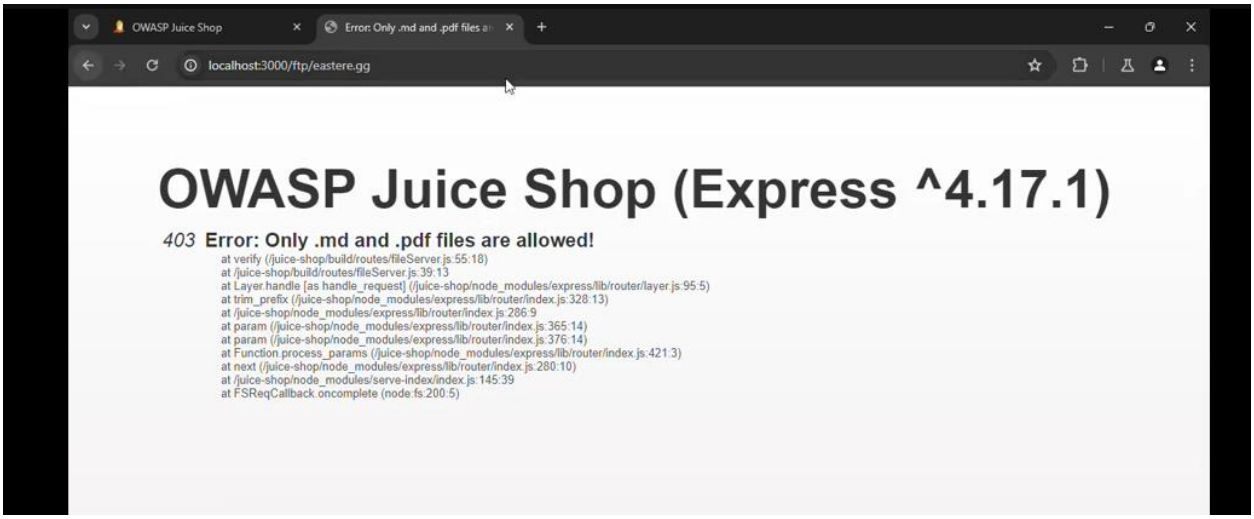
-Severity :High

- **Description:**
Broken Access Control occurs when an application fails to properly restrict access to resources, allowing unauthorized users to bypass intended security measures. In OWASP Juice Shop, the /ftp endpoint is vulnerable to null byte injection, enabling the download of restricted files. This exposes a Base64-encoded string that, after decoding and ROT13 decryption, reveals a hidden endpoint. Accessing this endpoint redirects to a restricted page, demonstrating a significant access control flaw that violates the application's security model.

- **Affected Component:**
File transfer endpoint
/ftp
The endpoint lacks proper validation, allowing null byte injection to bypass file type restrictions and access hidden content, including an encoded path to a restricted page.

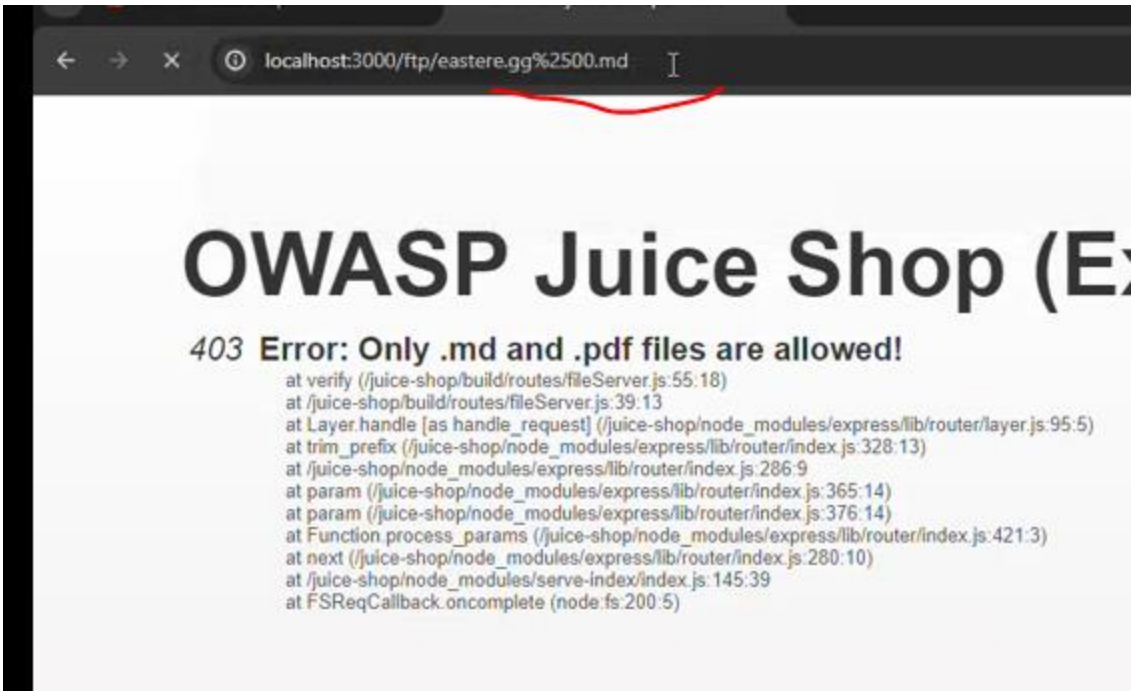
Steps to Reproduce (in OWASP Juice Shop):

1. Navigate to the /ftp endpoint:

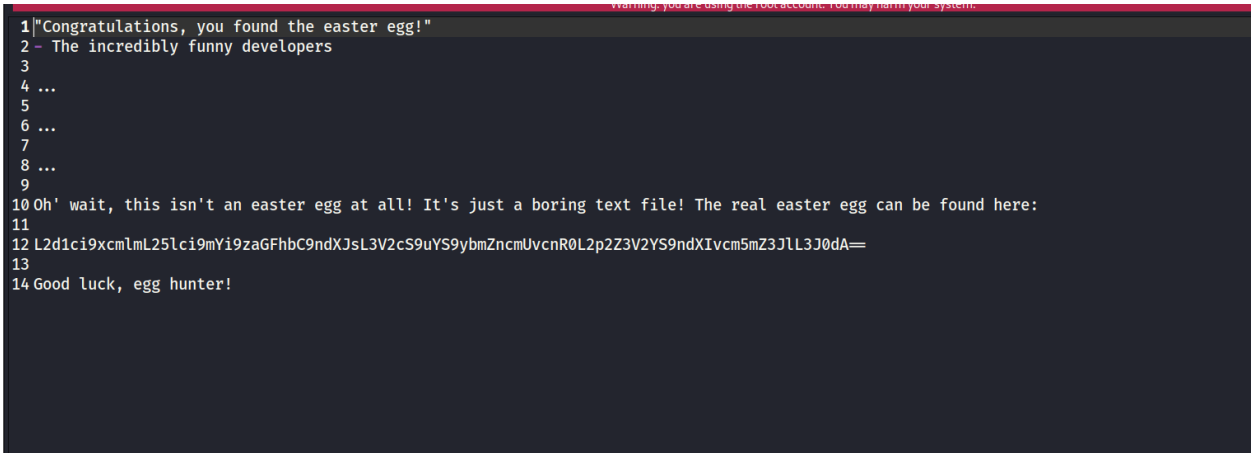


http://localhost:3000/ftp

2. Use a null byte injection technique by appending %00 to a file request, e.g., easteregg.gg%00.md.



3. Download the file, which contains a Base64-encoded string: L2d1cm9mZm5lcmZic2hhYWxndXJ2bmFtcmZncmV0dGpndnVhZ3VybWZvcmV0dA==.



4. Decode the Base64 string using a decoder tool, yielding: /gur/offner/fbshaal/gur/vna/mrfgret/tjgqva/gur/mforett.

Decode from Base64 format

Simply enter your data then push the decode button.

L2d1ci9xcmlmL25lci9mYi9zaGFhbC9ndXJsL3V2cS9uYS9ybmZncmUvcnR0L2p2Z3V2YS9ndXIvcm5mZ3JlL3J0dA

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

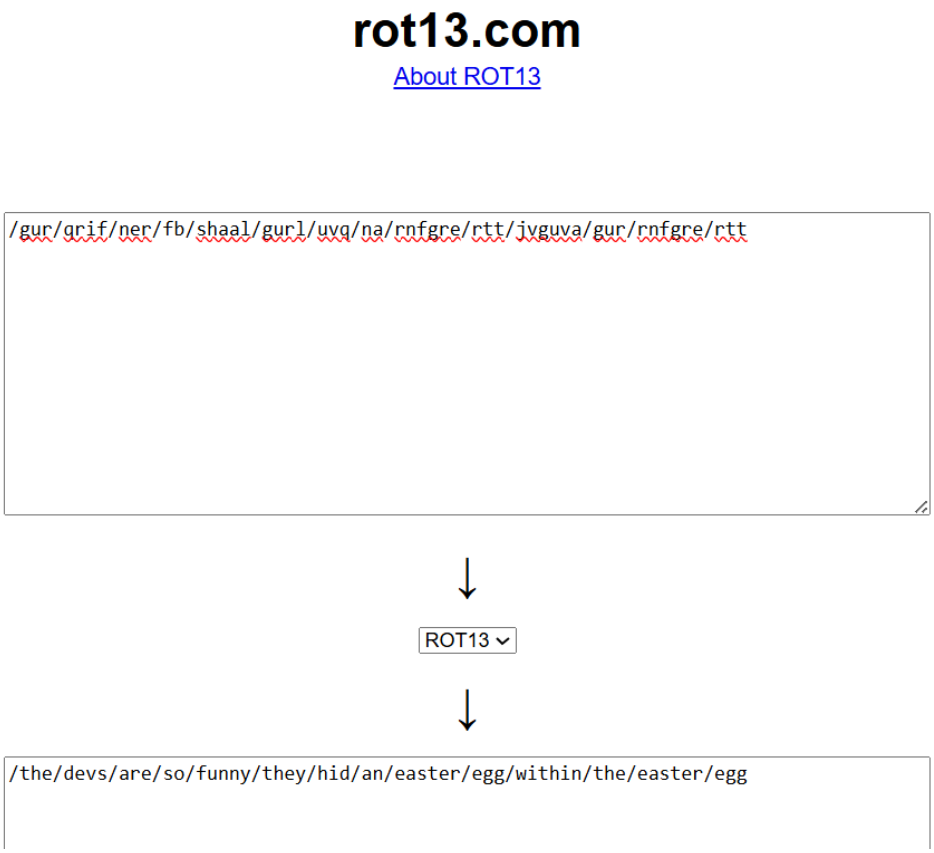
Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

DECODE Decodes your data into the area below.

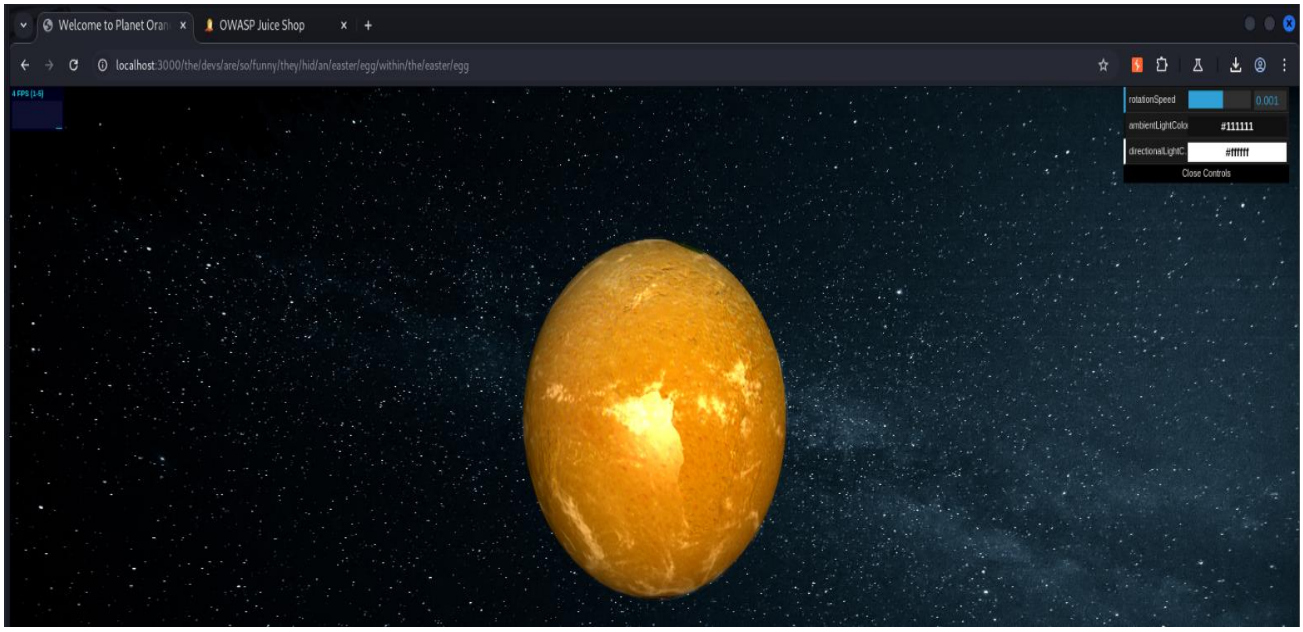
/gur/qrif/ner/fb/shaal/gurl/uvq/na/mfgre/rtt/jvguva/gur/mfgre/rtt

5. Recognize this as a ROT13-encrypted string and decode it to: /the/devs/are/so/funny/they/hid/an/easter/egg/with/the/easter/egg.



6. Insert the decoded path into the URL: http://localhost:3000/the/devs/are/so/funny/they/hid/an/easter/egg/with/the/easter/egg.

The page redirects to a restricted area, confirming the challenge: "You successfully solved a



challenge: Nested Easter Egg (Apply some advanced cryptanalysis to find the real easter egg!)."

Impact:

- Unauthorized access to hidden content, potentially exposing sensitive information or system details.
- Bypassing of file type restrictions, enabling exploitation of restricted resources.
- Increased risk of further attacks if additional hidden endpoints are uncovered.
- Undermines the application's access control integrity, affecting overall security.

11) SQL injection – login form – 3

Severity: High

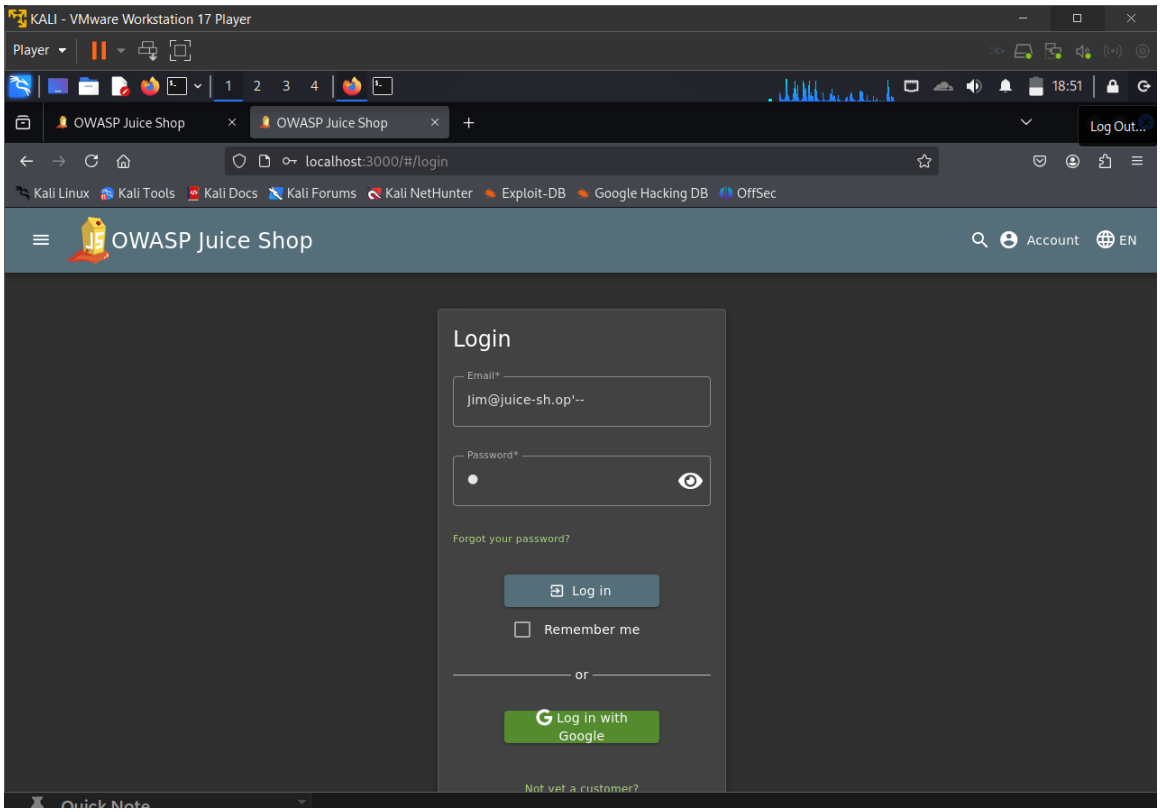
Description:

A SQL Injection vulnerability was identified in the login form of the application. This allows an attacker to bypass authentication mechanisms and potentially gain unauthorized access to user accounts, including administrative ones.

Affected Component:
/login form

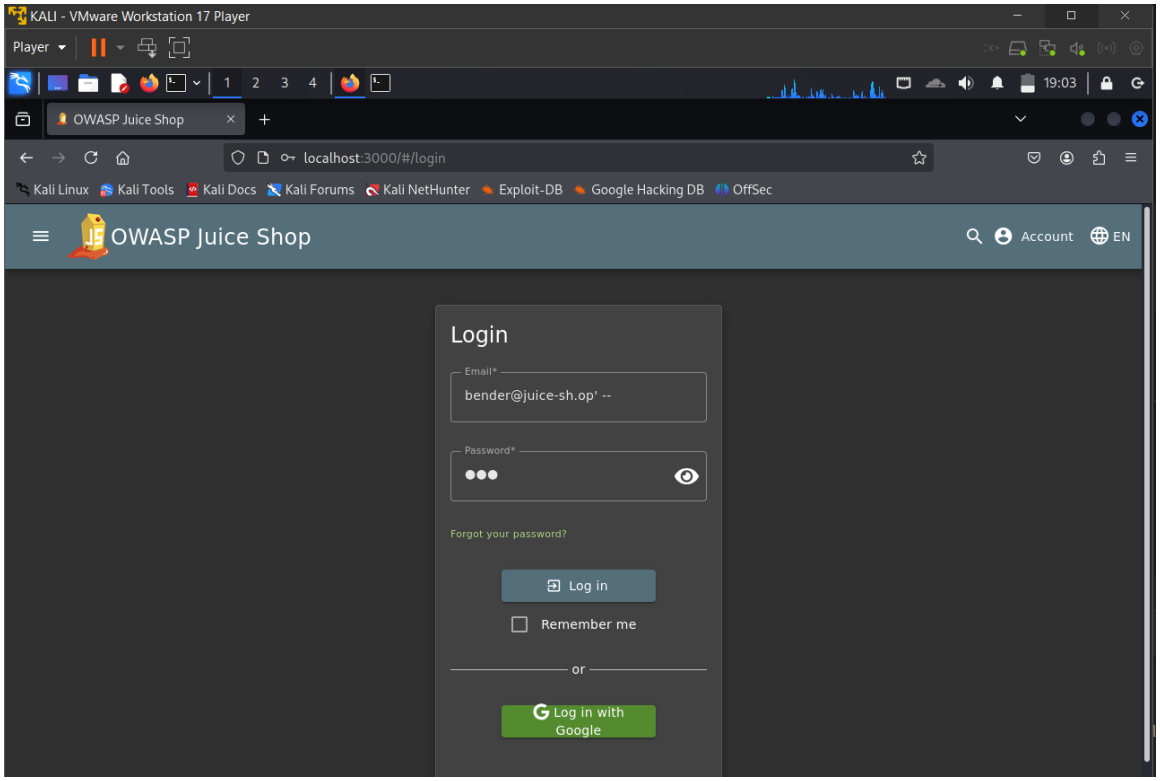
Steps to Reproduce:

a. Discovered the admin's email via recon in blog posts and product comments.
b. Navigated to the login page.

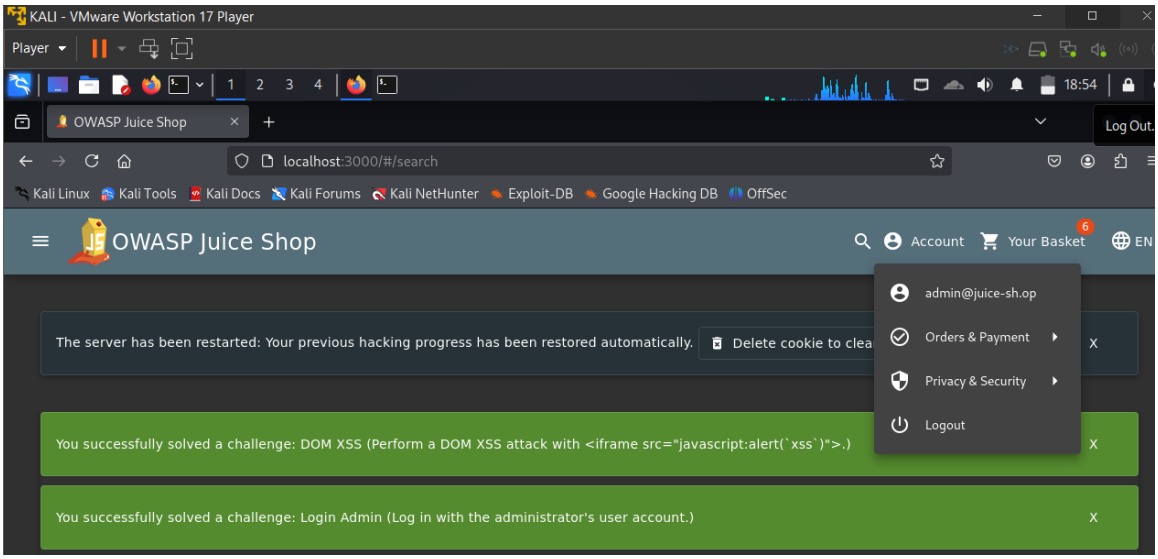


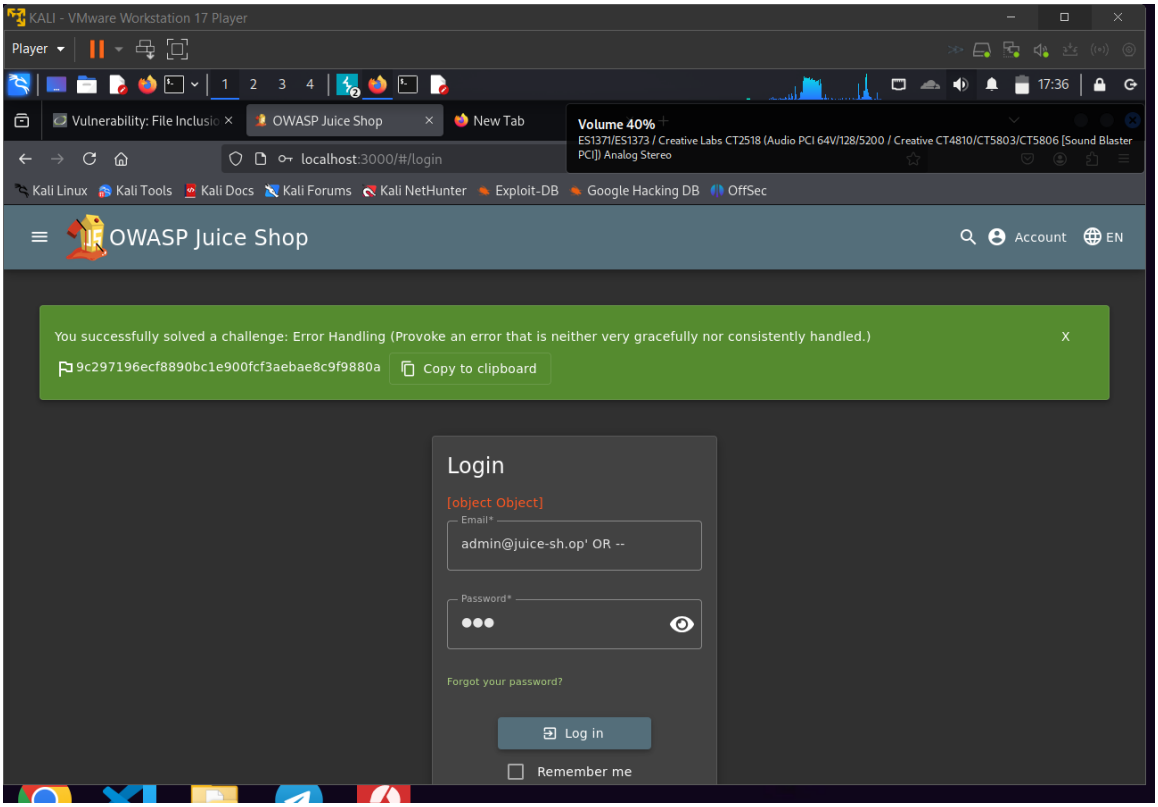
c. Entered the following payload in the **password** field:

' OR '1'='1 –



d. Successfully bypassed the login mechanism and was logged in as the admin user.




KALI - VMware Workstation 17 Player
Player
Vulnerability: File Inclusio
OWASP Juice Shop
New Tab
localhost:3000/#/login
Volume 40%
ES1371/ES1373 / Creative Labs CT2518 (Audio PCI 64V/128/5200 / Creative CT4810/CT5803/CT5806 [Sound Blaster PCI]) Analog Stereo
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
OWASP Juice Shop
Account
EN
You successfully solved a challenge: Error Handling (Provoke an error that is neither very gracefully nor consistently handled.)
9c297196ecf8890bc1e900fcf3aebae8c9f9880a
Copy to clipboard
Login
[object Object]
Email*
admin@juice-sh.op' OR --
Password*
Forgot your password?
Log in
Remember me

Impact:
An attacker can gain full access to the application as an administrator without valid credentials. This can lead to full compromise of user data, application settings, and potential lateral movement within the internal infrastructure.

12) Broken access control – IDOR in the basket ID

Severity: High

Description

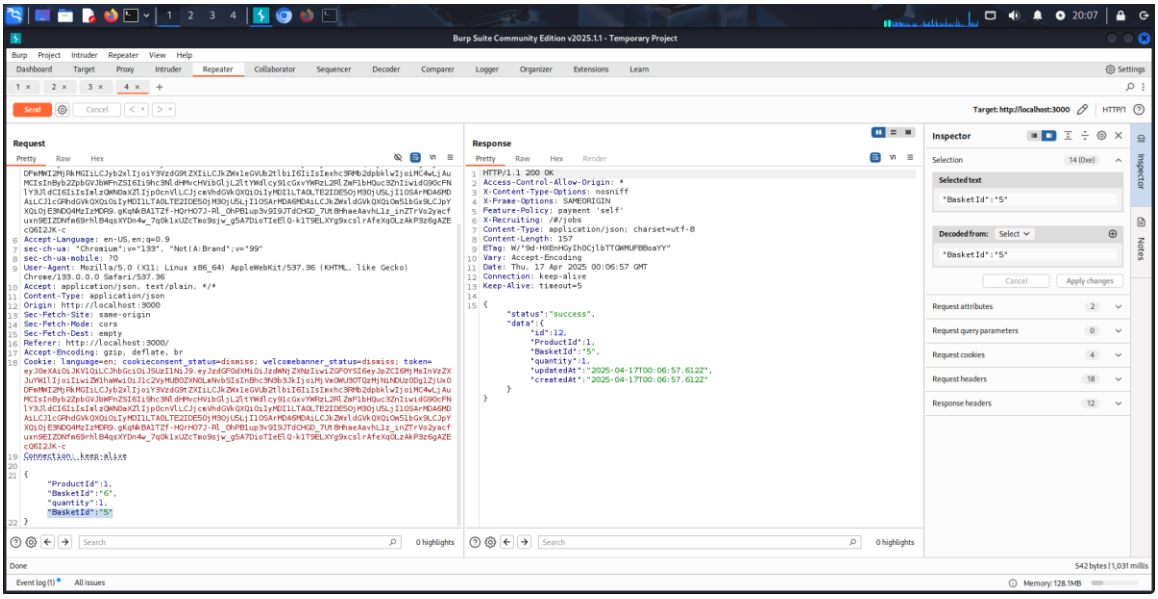
The application allows an authenticated user to manipulate another user's shopping basket by modifying the BasketId in a request. This happens due to missing server-side authorization checks, which should verify that the user has permission to modify the specified basket.

Affected Component

PUT /api/BasketItems/:id

Steps to Reproduce

1. Logged in as a normal user (e.g., user@example.com).
2. Opened **Burp Suite** and intercepted a request after adding a product to the basket. The following request was observed:
3. In Burp Suite, modified the BasketId in the request to a different value, assuming it belongs to another user.
4. Forwarded the following malicious request:



5. The server accepted the request and added the product to **another user's basket**, without any ownership validation.

Impact

- An attacker can modify the contents of another user's shopping basket.
- This could lead to:
    - Confusion or disruption of purchases.
    - Abuse of discounts or promotions.
    - Potential fraudulent behavior or customer manipulation.
- It represents a clear violation of data integrity and access control principles.

13) Broken access control – userID

Severity: heigh

Description

The application allows users to submit feedback that appears to be from another user by tampering with a hidden userId field in the client-side HTML form. Since the server **does not validate that the userId matches the authenticated user**, this allows attackers to forge feedback in the name of any other user, including admin accounts.

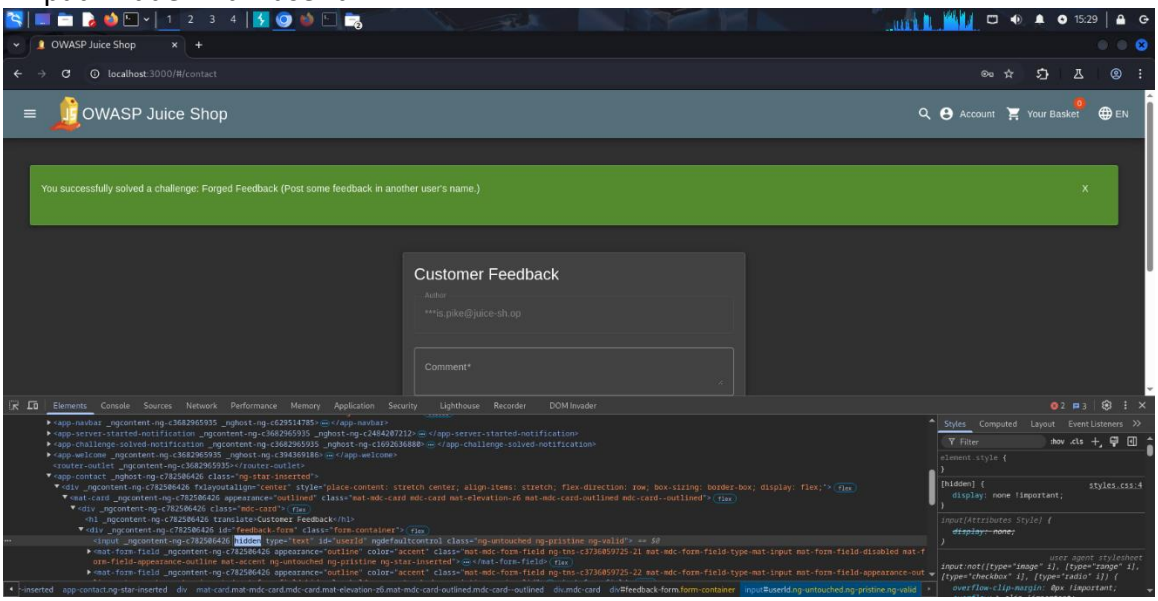
Affected Component

POST /api/Feedbacks

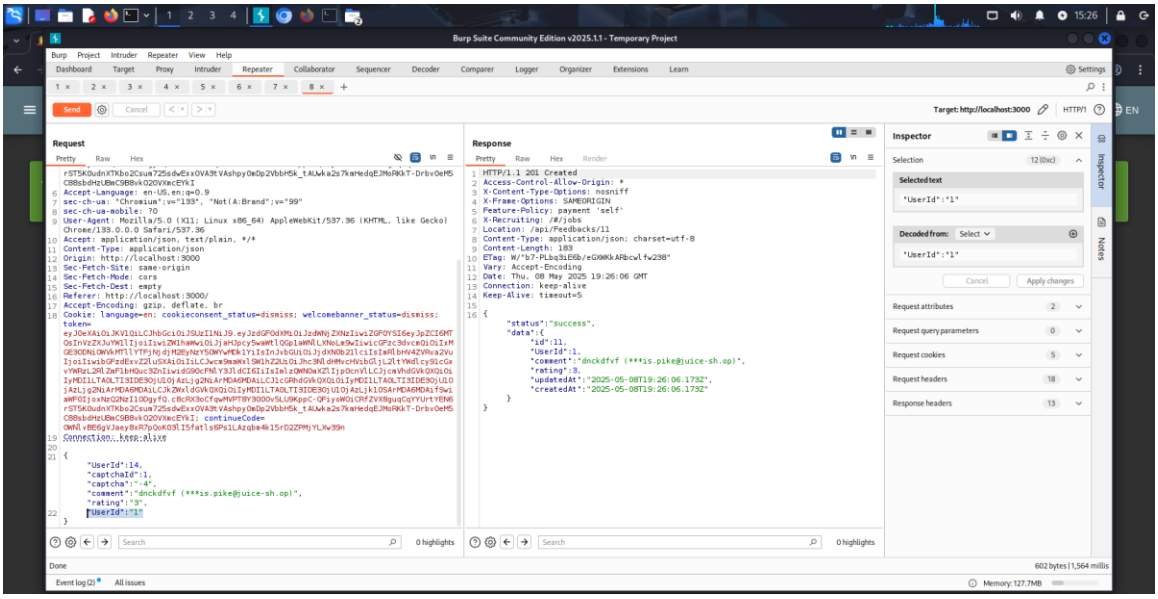
Steps to Reproduce

1. Logged in as a normal user (e.g., user@example.com).
2. Navigated to the "Customer Feedback" page and opened **Developer Tools (F12)** to inspect the form.
3. Observed a hidden input field in the form:

   html
   <input "hidden" id="userId" >



4. Changed the value of userId to 1, which corresponds to the admin user (or another known user ID), directly in the browser using the **Elements** tab or by intercepting the request in **Burp Suite**.
5. Submitted the form with a feedback message. The resulting request looked like:



6. The server accepted the request and the feedback appeared to be submitted by user ID

Impact

- An attacker can impersonate other users by submitting feedback on their behalf.
- This breaks **authentication integrity** and can:
    - Falsely attribute statements to trusted users (e.g., admins).
    - Damage reputation or create misleading trust signals. Be used for social engineering or internal manipulation

14) Broken access control – product review views

Severity: Medium

Description:

The application allows users to submit product reviews. However, it does not properly validate or restrict the message and author fields in the request body. This enables an attacker to **forge reviews using any author name**, including impersonating other users or administrators, thus violating trust and allowing manipulation of public-facing content.

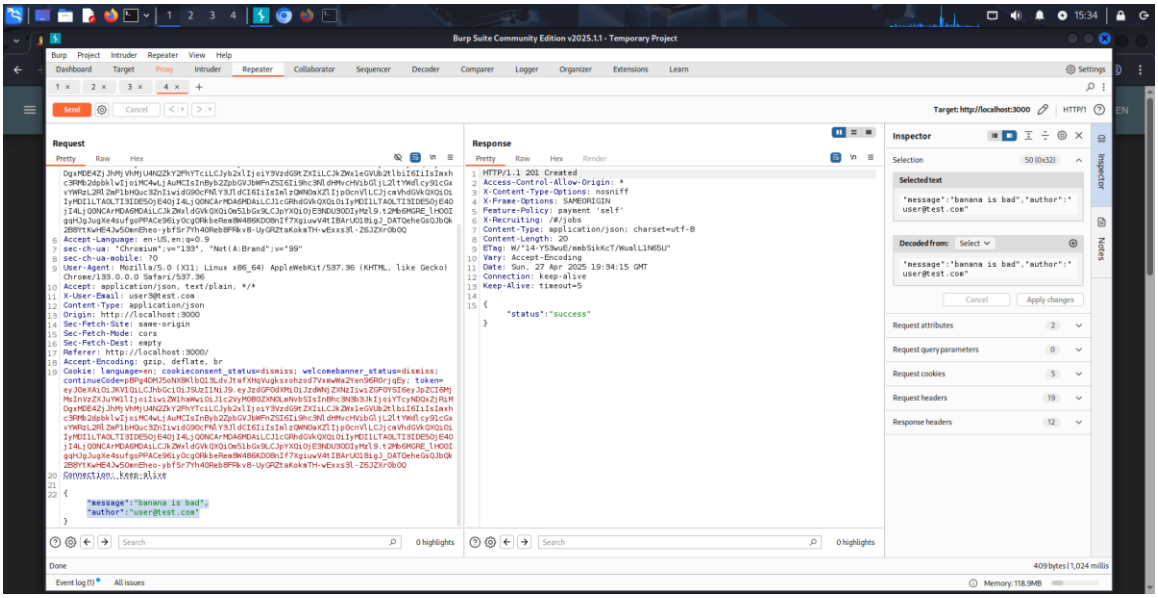
Affected Component:

POST /api/ProductReviews

Steps to Reproduce:

1. Logged in as a normal user (e.g., user@juice-sh.op).

2. Opened a product page and used **Burp Suite** or browser developer tools to intercept the **review submission request**.
3. Modified the JSON request body before it was sent:



4. Sent the modified request to the server.
5. The review was successfully posted, **displaying the author name as "admin"**, even though the logged-in user was not admin.

Impact:

- **Impersonation**: Users can post content under any name, potentially impersonating admins or other users.
- **Reputation manipulation**: Attackers can forge positive or negative reviews to manipulate product trust.
- **No server-side validation**: Indicates lack of integrity controls for critical user-generated content.

15) Improper input validation – Deluxe fraud

Severity : High

Description:

Improper Input Validation occurs when an application relies on client-side controls and fails to enforce server-side validation, allowing attackers to bypass payment restrictions. In this case, the /payment/deluxe endpoint in OWASP Juice Shop permits a Deluxe Membership to be

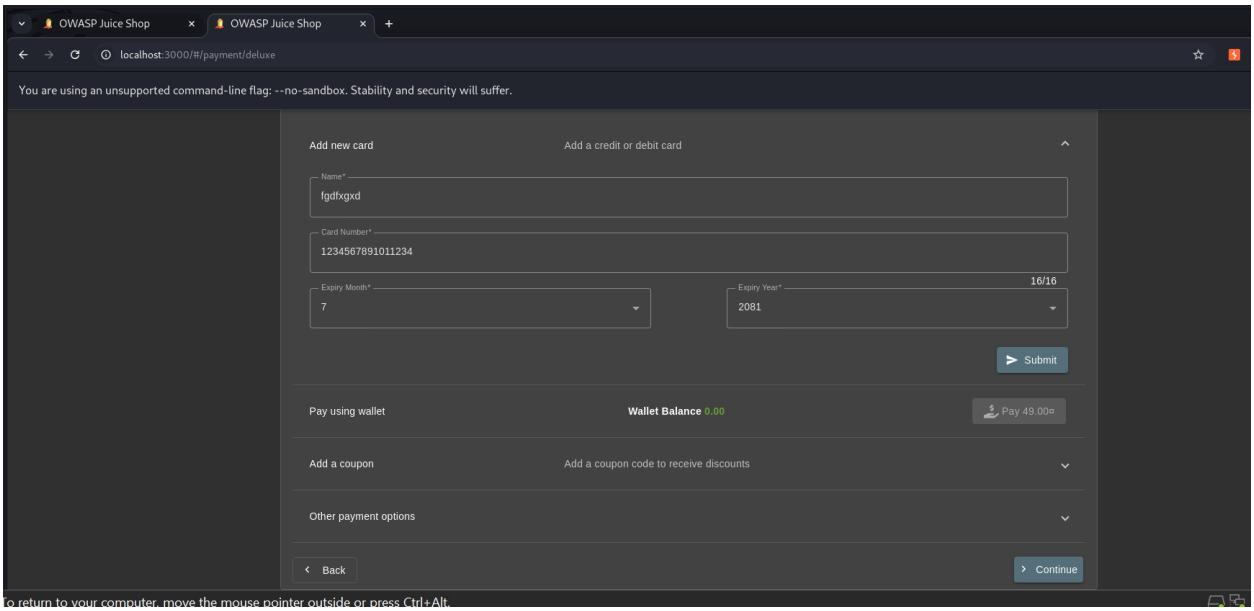
obtained by manipulating the disabled "Pay" button and removing the wallet parameter from the intercepted request, enabling fraudulent upgrades without payment.

Affected Component:
Payment processing endpoint
/payment/deluxe
The endpoint does not validate payment legitimacy on the server-side and processes requests even when the wallet parameter is removed, bypassing payment requirements.

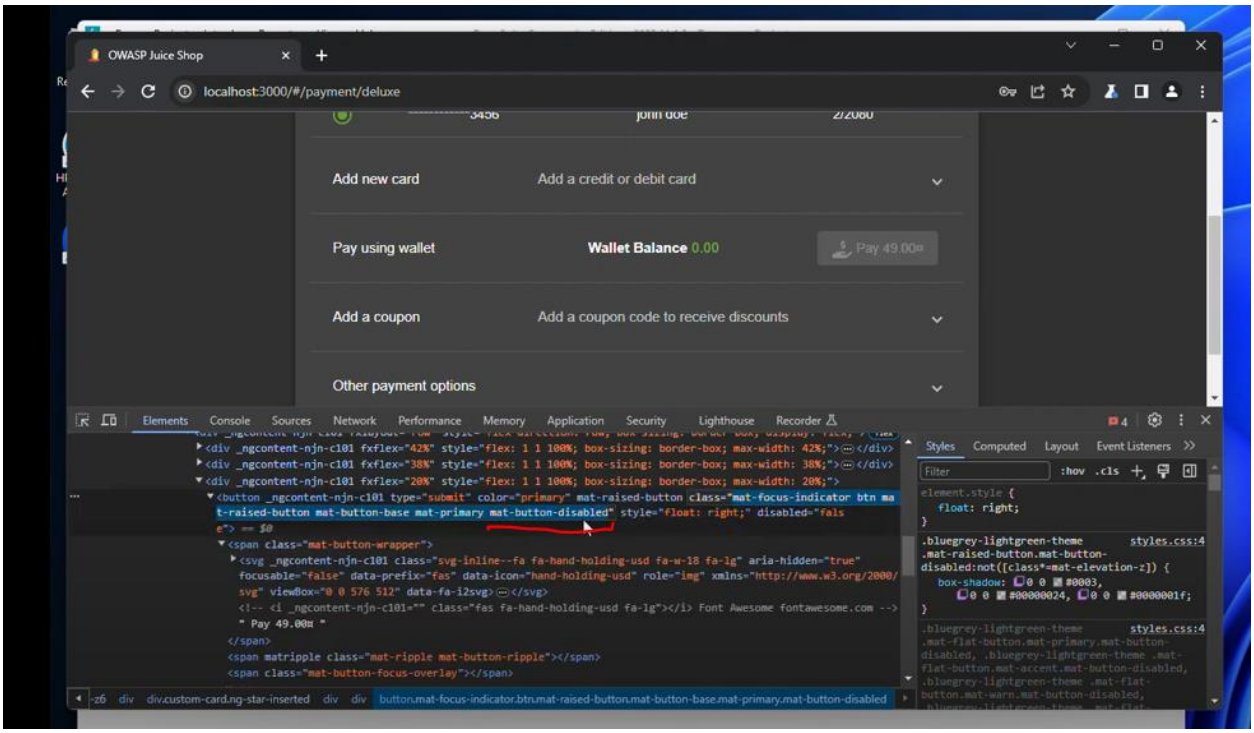
Steps to Reproduce (in OWASP Juice Shop):

1. Navigate to the payment page:
   http://localhost:3000/#/payment/deluxe
2. Enter new card information (e.g., card number: 1234567890123456, expiry: 12/25, CVV: 123).
3. Right-click the "Pay" button and select "Inspect Element" to view its HTML.
4. Locate the button's class (e.g., mat-button-disabled) and disabled attribute.
5. Modify the HTML to remove mat-button-disabled and set disabled="false".
6. Click the now-enabled "Pay" button and intercept the request using a proxy tool (e.g., Burp Suite).
7. In the intercepted request, remove the wallet parameter (e.g., change wallet=... to an empty value or remove it entirely).
8. Forward the modified request.
9. If vulnerable, the response confirms a successful Deluxe Membership upgrade without payment, as shown in the success message.

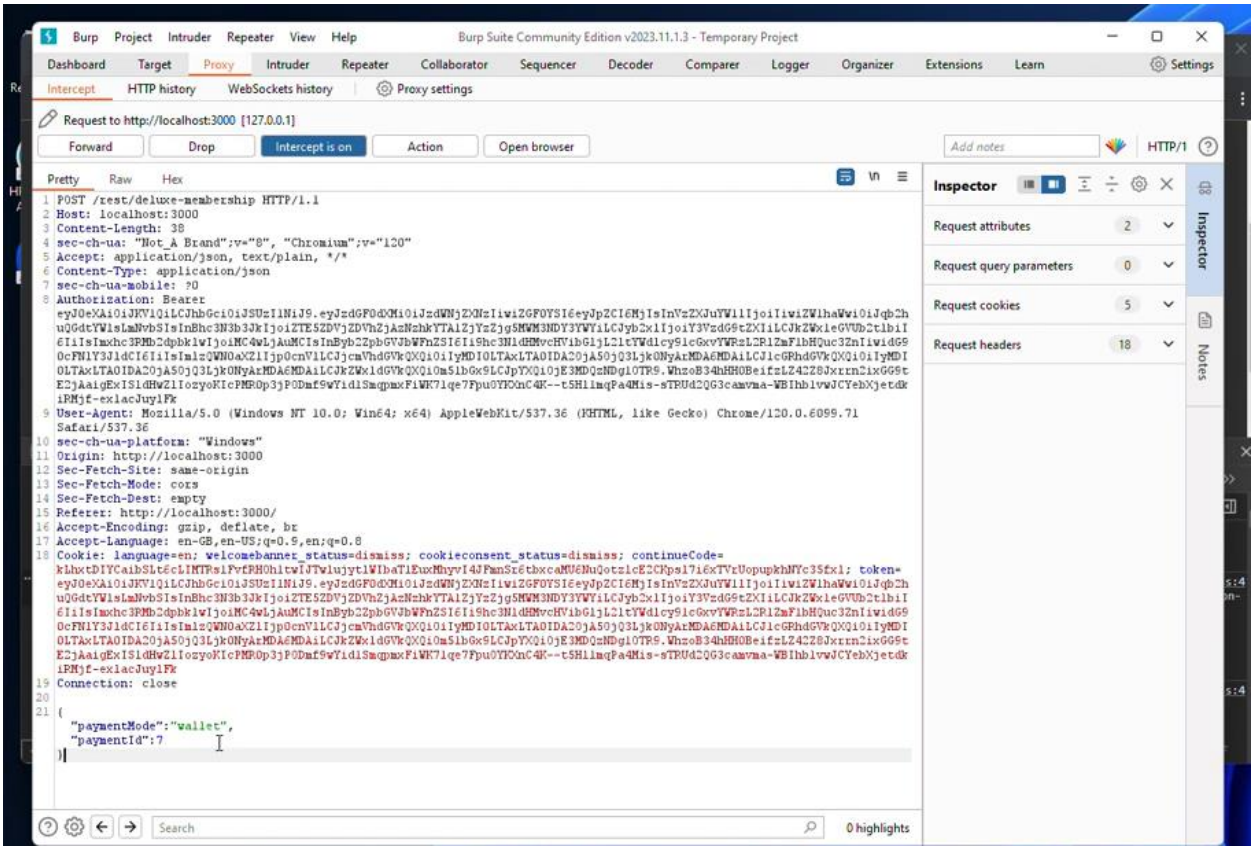
Impact:
• Unauthorized access to premium features (e.g., Deluxe Membership) without payment.



• Financial loss for the application provider due to fraudulent upgrades.



• Potential exploitation by attackers to scale the attack across multiple accounts.



• Undermines the payment system's integrity, affecting user trust.

16) Security Misconfiguration – file size

Severity: high

Description :

The application's file upload functionality relies on client-side validation, allowing attackers to bypass size and type restrictions by modifying requests through Burp Suite. During testing, we successfully uploaded a prohibited 150KB .txt file despite apparent client-side restrictions (100KB limit).

Step-by-Step Exploitation:

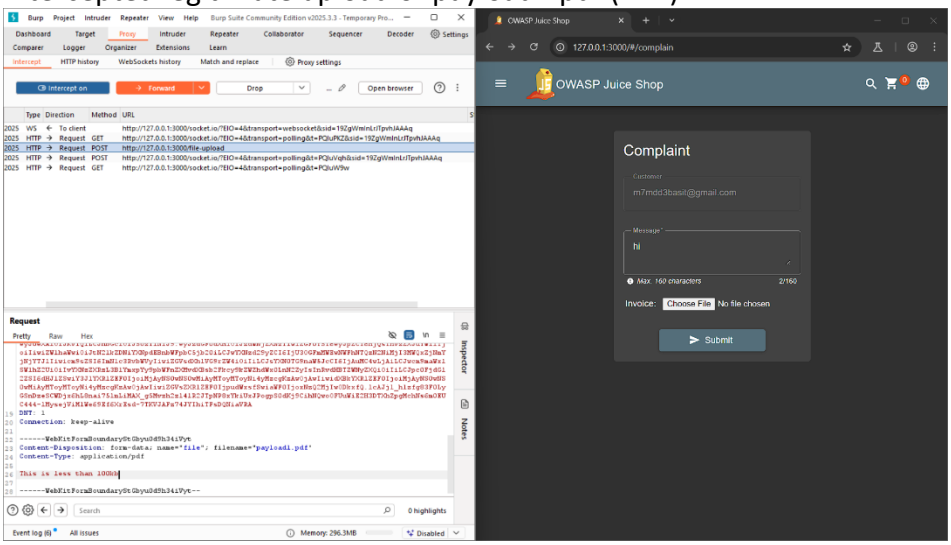
1- Payload Preparation (PowerShell/Python):

```
>>> with open('payload1.txt', 'w') as file:
...     file.write('This is less than 100kb')
...
23
>>> with open('payload1.txt', 'w') as file:
...     file.write('This is less than 100kb')
...
```

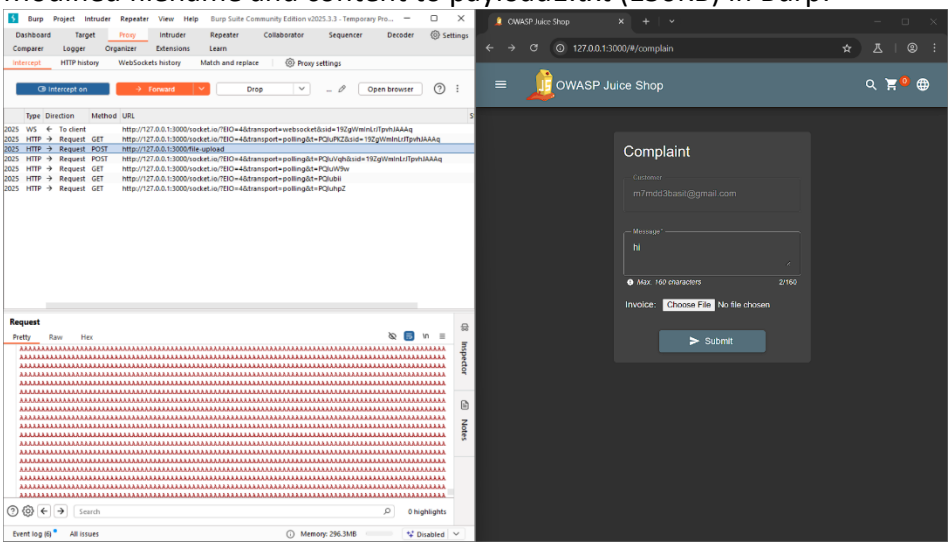
payload1.pdf	5/2/2025 11:42 PM	Chrome PDF Docu...	1 KB
payload2.txt	5/2/2025 11:43 PM	Text Document	150 KB

2- Interception & Tampering:

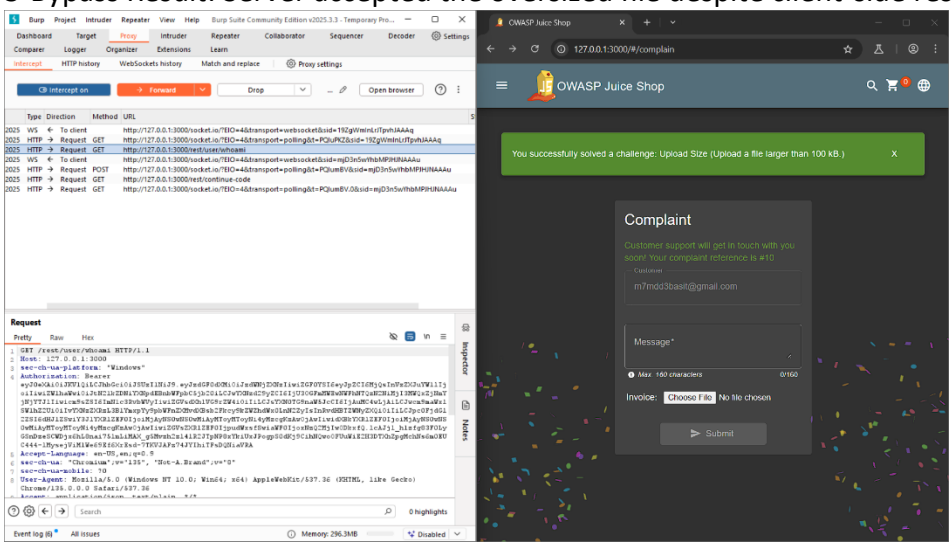
- Intercepted legitimate upload of payload1.pdf (1KB)



- Modified filename and content to payload1.txt (150KB) in Burp:



3-Bypass Result: Server accepted the oversized file despite client-side restrictions



17) Security Misconfiguration – upload type

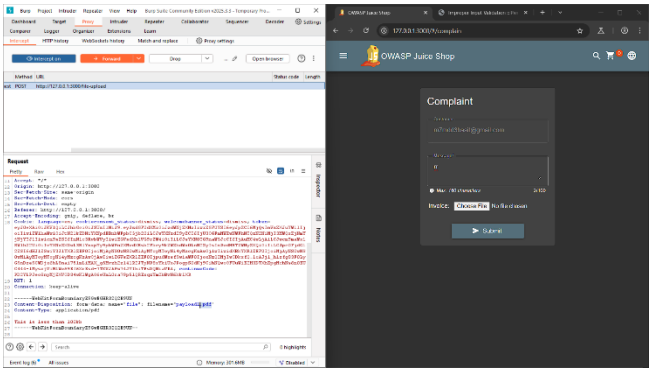
Severity: high

Description :

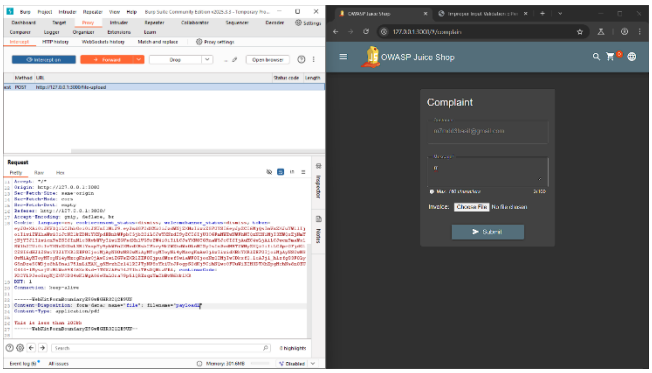
The application's reliance on client-side file type validation allows attackers to upload prohibited file types (e.g., .txt) by intercepting and modifying requests in Burp Suite. During testing, we bypassed PDF-only restrictions to upload a .txt file despite frontend controls.

Exploitation Steps :

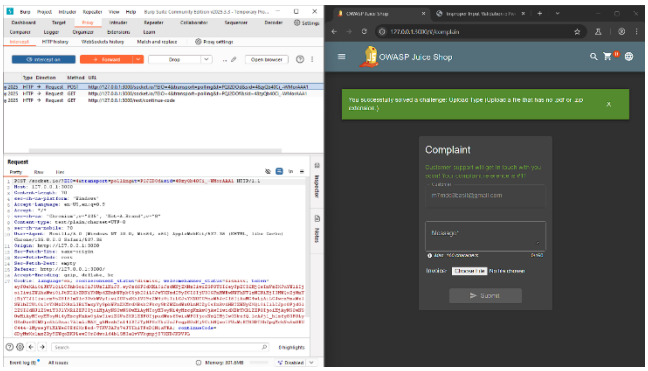
1-upload the file



2- Modified the Content-Disposition and Content-Type headers:



- **Confirmed successful**



18) Improper input validation – payback time

Severity: High

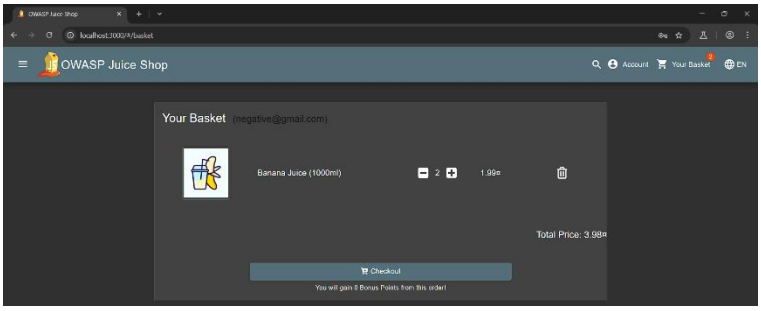
Description :

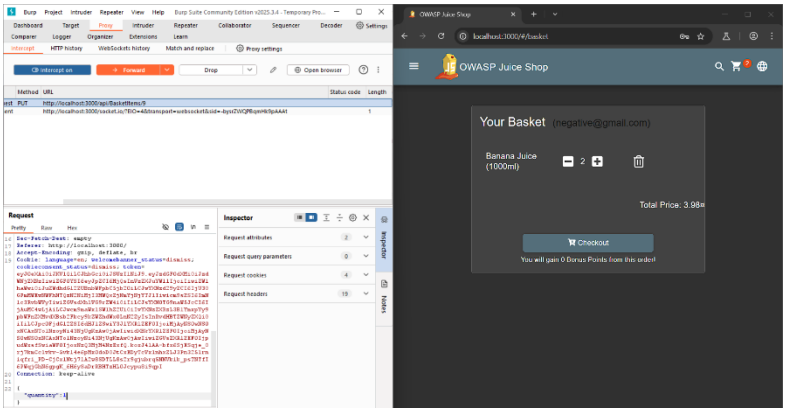
The application fails to validate product quantity values during checkout, allowing attackers to submit negative quantities (e.g., -10000) through request tampering. This flaw credits the attacker's account instead of debiting it, enabling financial fraud.

Exploitation :

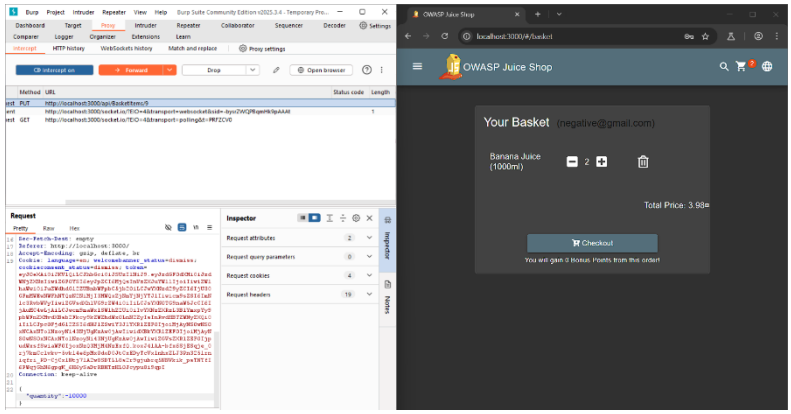
Steps 1 - Legitimate Order Placement:

Add a product (e.g., Banana Juice) to cart with quantity 1 Initiate checkout





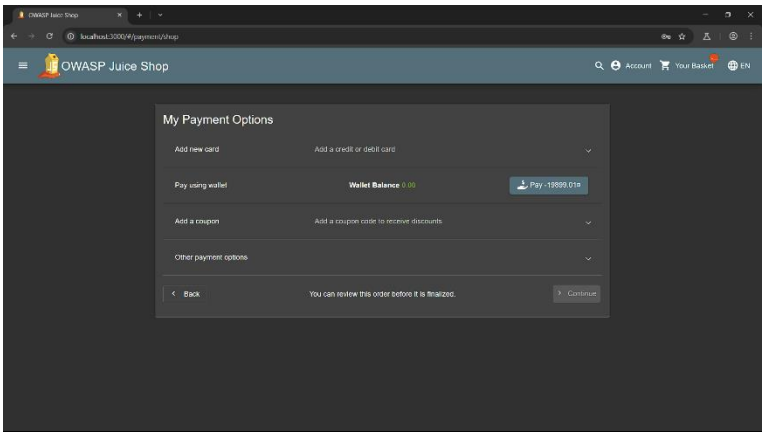
Step 2 - 2. Request Tampering: intercept the request in Burp Suite and send it .

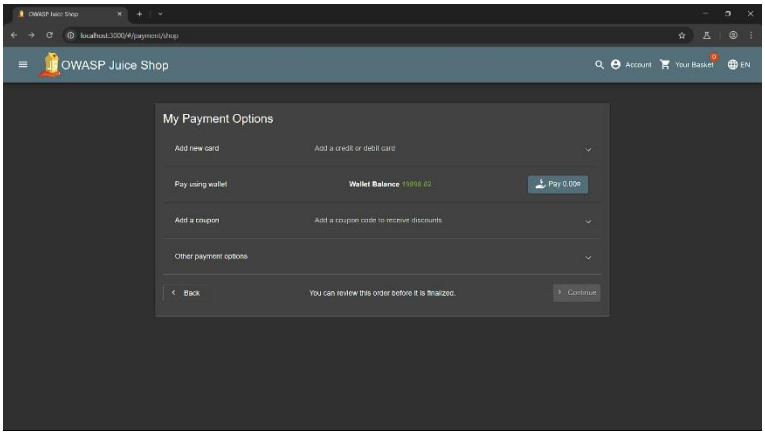


Exploit Result:

- System processes negative quantity as a refund
- Attacker's balance increases by 10000 * product_price

- Challenge "Payback Time" solved



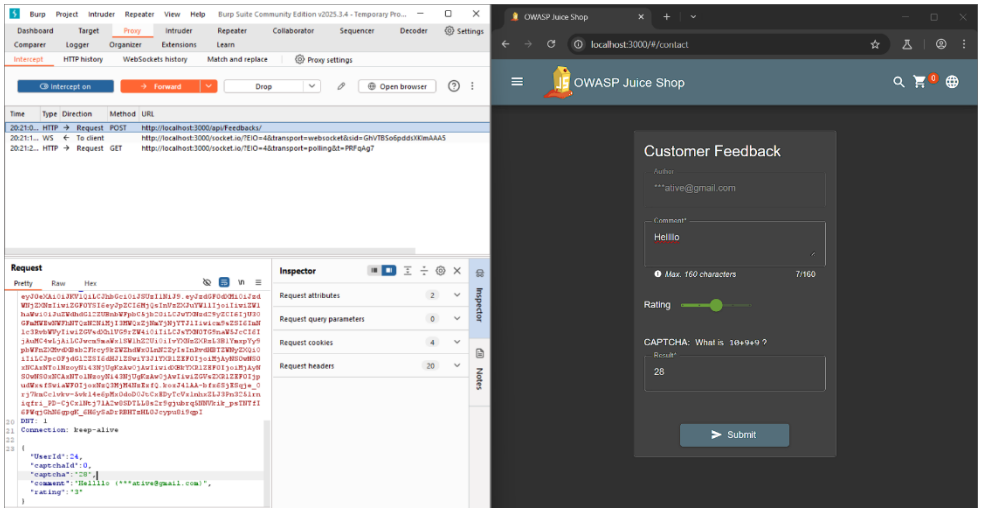


19) Broken authentication – CAPTCHA Bypass

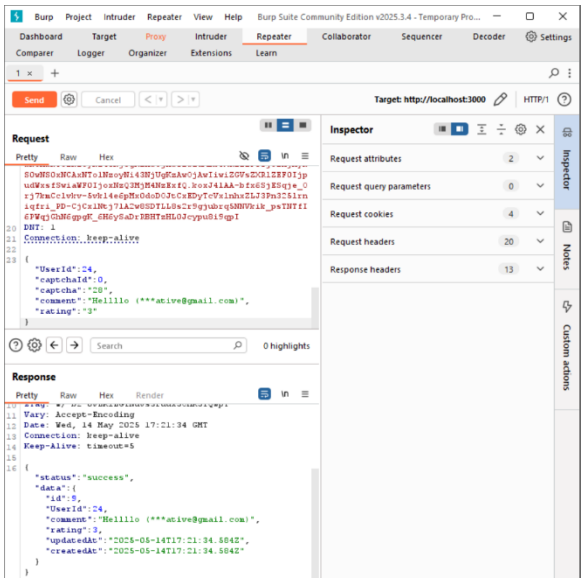
Severity: high

- **Description :** The application's CAPTCHA mechanism fails to prevent automated submissions, allowing attackers to bypass rate limiting by replaying the same feedback request multiple times through Burp Repeater. This enables spam attacks and potential data corruption.
- **Exploitation Steps :**
  1. **Legitimate Feedback Submission**:

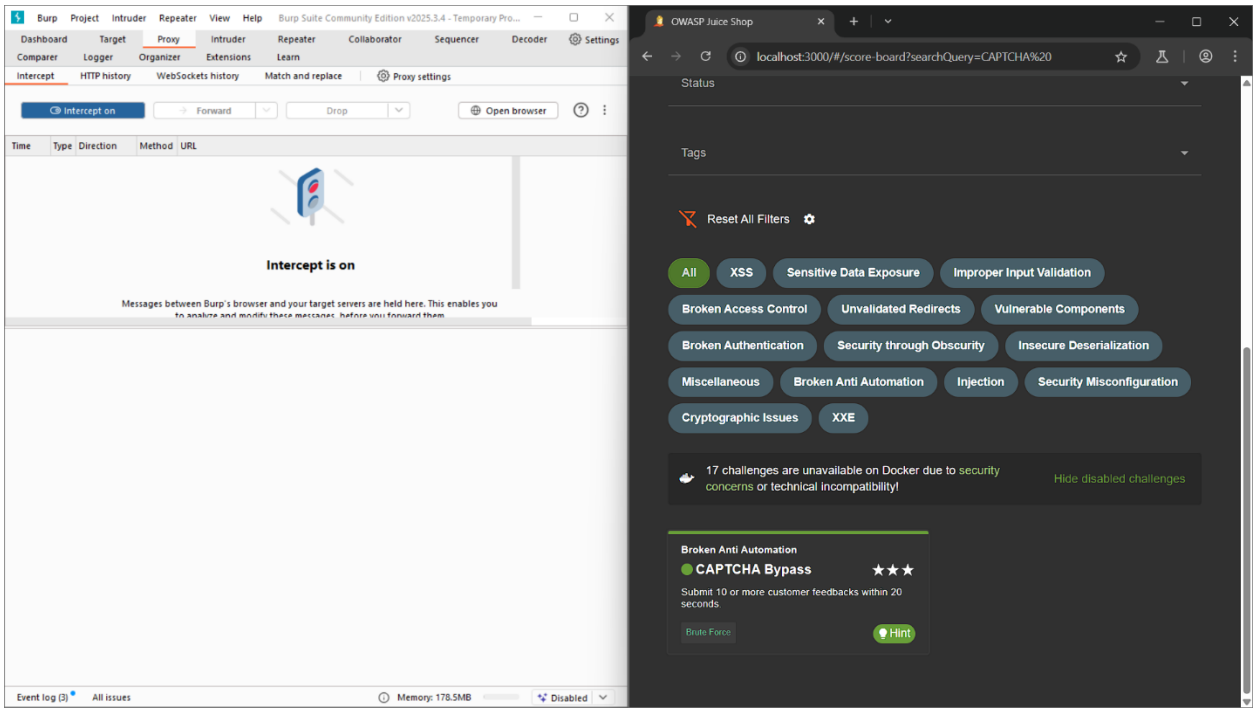
Submit a feedback form with valid CAPTCHA



2. **Intercept the request in Burp Suite** : Request Replay Attack
   Sent 10+ identical requests within 20 seconds via Burp Repeater



3. **Verification:**



20) Improper input validation – Admin registration

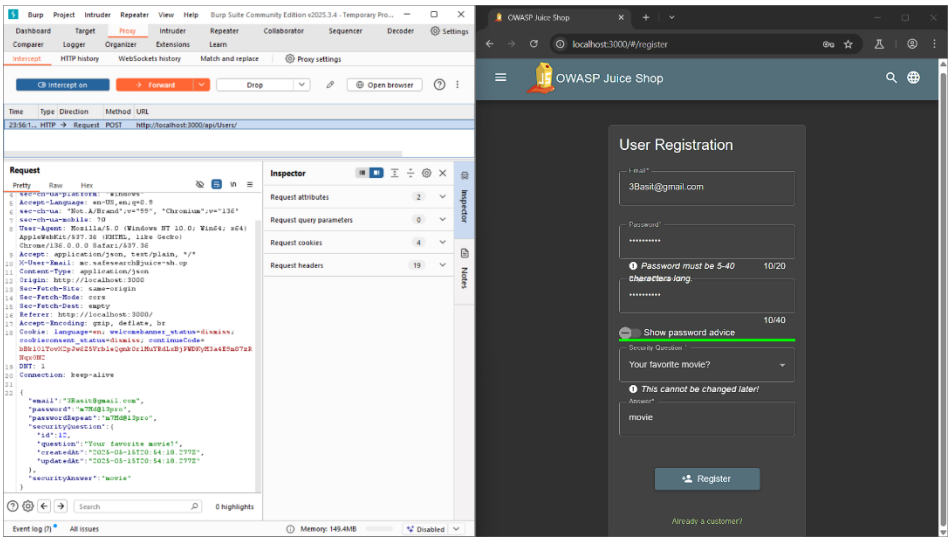
Severity: high

Description:

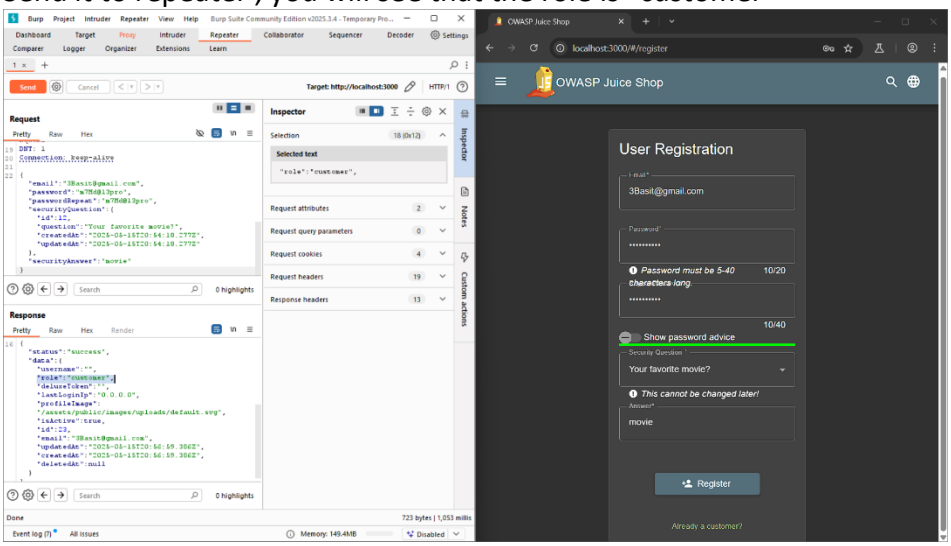
The application's user registration functionality improperly validates input, allowing attackers to self-assign administrator privileges by manipulating the registration request. This flaw bypasses the intended role-based access control (RBAC) system, enabling vertical privilege escalation.

Steps to procedures :

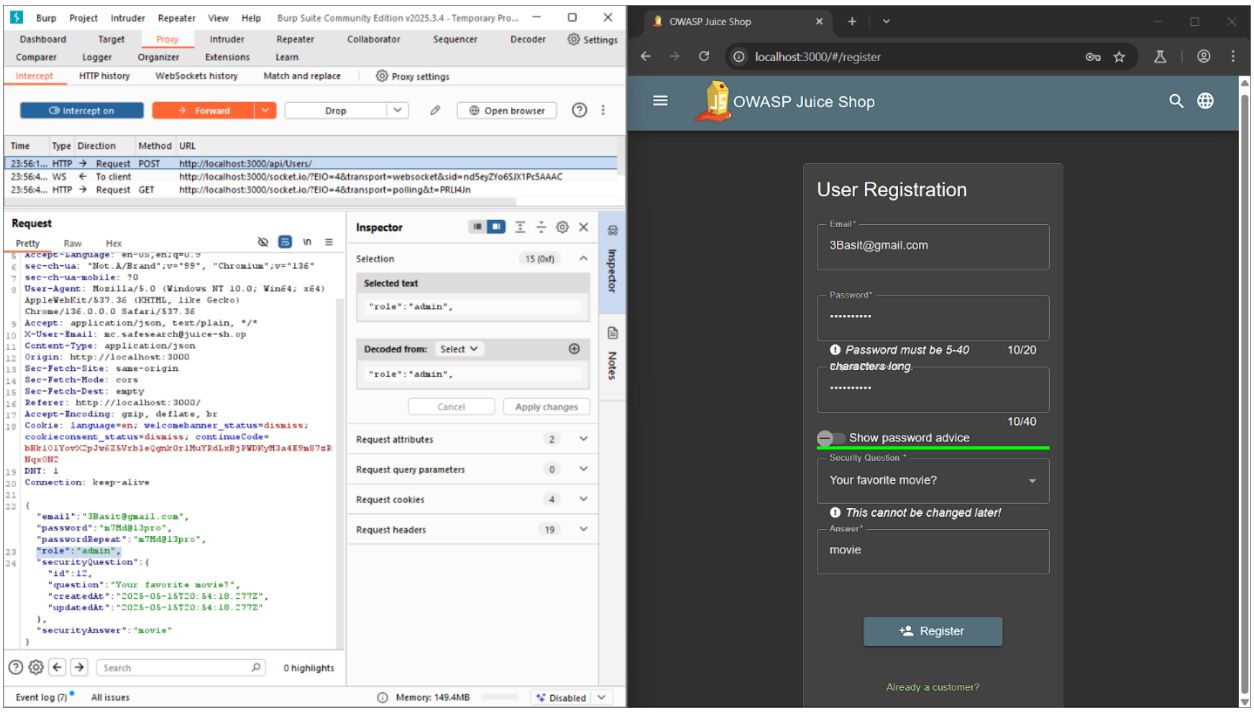
Intercept the registration request using Burp Suite or browser developer tools :



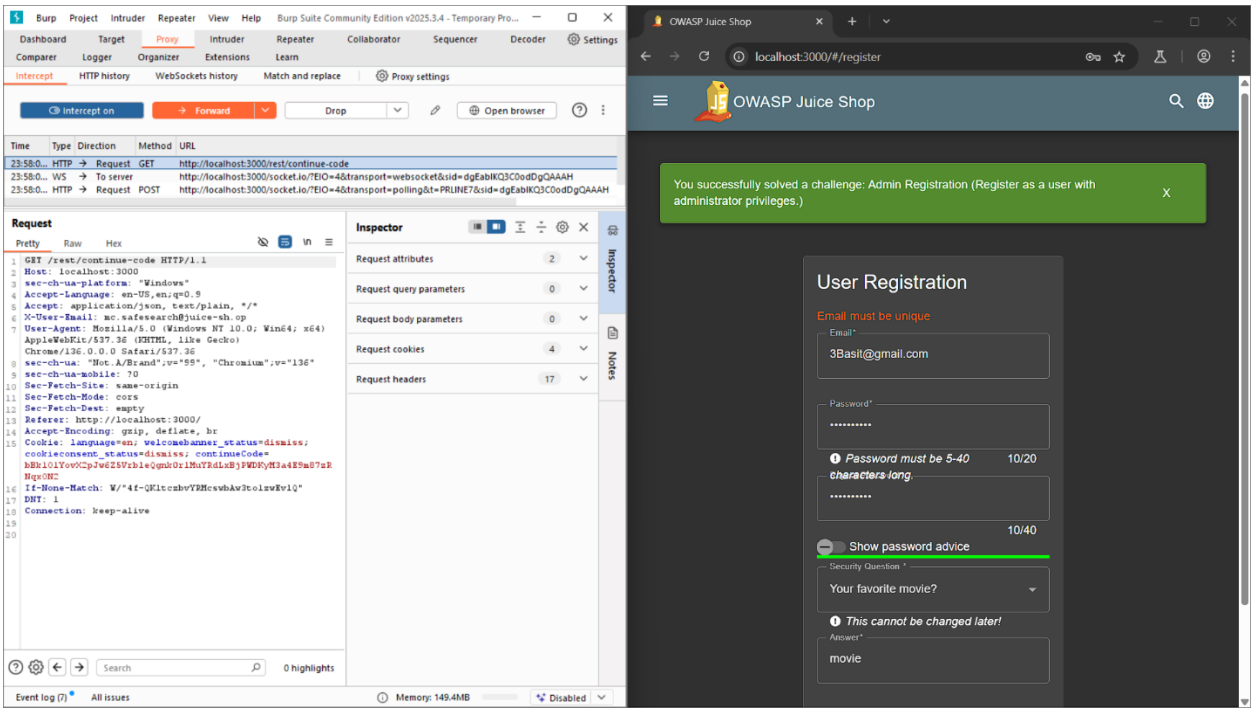
- Send it to repeater , you will see that the role is “customer”



- So before sending it I will change it’s role to “admin”



-



21) Reflected XSS in search functionality

Severity: Medium

Description:
A **reflected XSS** vulnerability was discovered in the search functionality of the web application. The input provided in the q parameter of the search endpoint is directly reflected in the HTML response without proper sanitization or encoding, allowing attackers to execute arbitrary JavaScript in the context of a victim's browser.

Affected Component:
/search?q=

Steps to Reproduce:

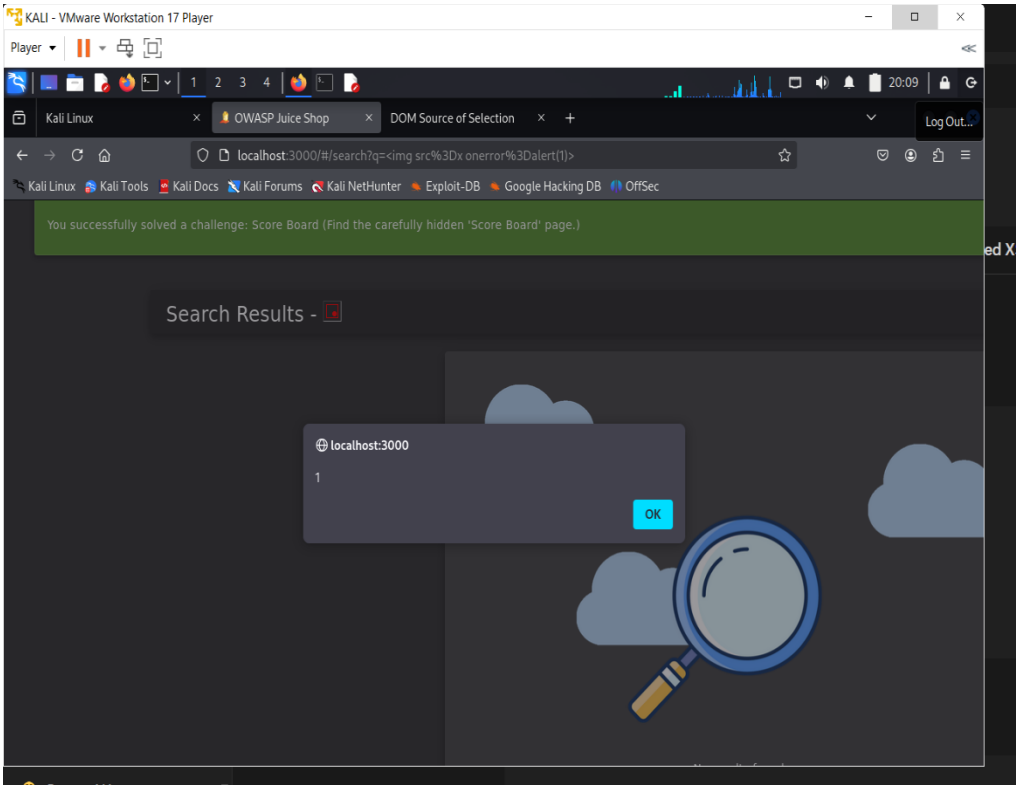
1. Navigated to the following URL:

   https://target-site.com/search?q=<img src=z onerror=alert(1)>

2. Observed the browser executing the payload, triggering a JavaScript alert(1) popup.
3. Confirmed the input is reflected in the page source without encoding.

- **Impact**:
  An attacker could trick users into clicking a malicious link (e.g., via email, social media, or phishing). If a victim accesses the link, JavaScript code can execute in their browser, allowing for:
  1. Session hijacking
  2. Defacement
  3. Redirection to malicious sites
  4. Phishing of credentials
- **Evidence**:
  1. URL tested:

     javascript
     CopyEdit
     /search?q=<img src=z onerror=alert(1)>



22) Broken access control -- viewing basket

Severity: medium

Description

A **Broken Access Control** vulnerability exists in the "View Basket" feature of OWASP Juice Shop. The application fails to properly verify ownership of the shopping basket when accessing it via its ID. This allows any authenticated user to view another user's basket by simply altering the Basket ID in the request, without any server-side authorization enforcement.

Affected Component

GET /api/Basket/:id

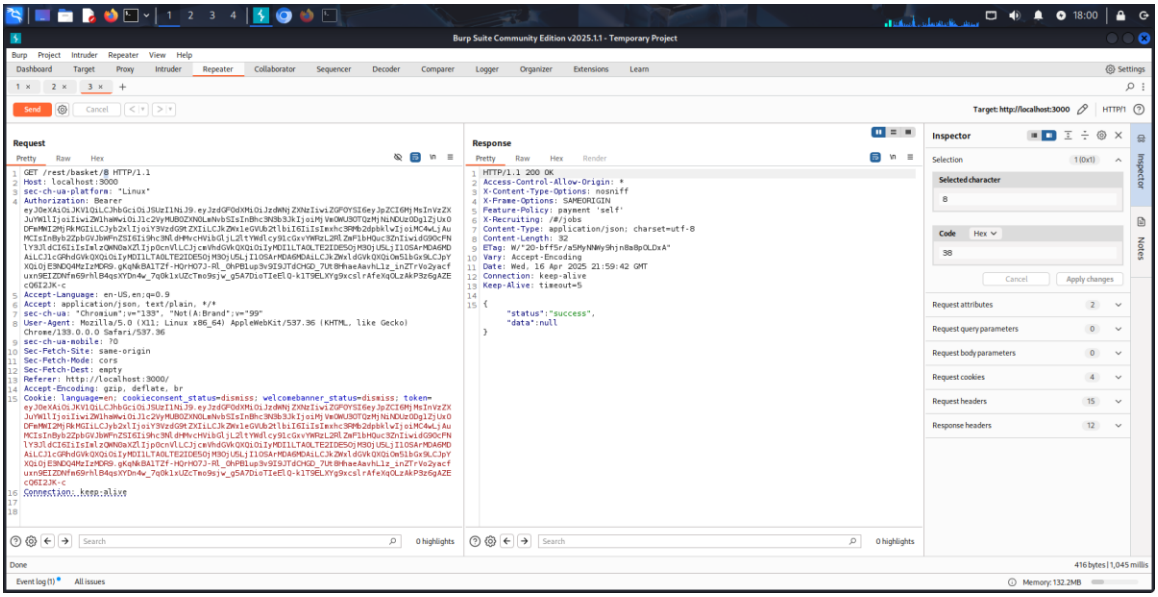
Steps to Reproduce

1. Logged into the Juice Shop with a valid user account.
2. Navigated to the /basket page and intercepted the HTTP request using **Burp Suite**.
3. Observed the following request:

http
GET /api/Basket/3 HTTP/1.1
Host: localhost:3000

4. Modified the Basket ID to 8:

   http
   GET /api/Basket/8 HTTP/1.1



5. The server responded with the contents of **another user's basket** without verifying that I had permission to access it.

Impact

An attacker can view the contents of **any user's basket** by modifying the ID parameter, exposing private information such as selected products and user behavior. This violates data confidentiality and, if combined with other vulnerabilities, may lead to **privilege escalation or full account compromise**.

23) Broken access control – empty user Registration

Severity: Medium

Description

Improper Input Validation occurs when applications fail to enforce input checks. In OWASP Juice Shop, the /api/Users endpoint allows user registration without validating critical fields like

email and password. Attackers can intercept requests, remove these fields, and create empty user accounts, bypassing essential validation and completing the *"Empty User Registration"* challenge.

Affected Component

User registration endpoint /api/Users
The endpoint lacks server-side validation for required fields, enabling incomplete account creation.

Steps to Reproduce (OWASP Juice Shop)

1. **Navigate to registration page**:
   http://localhost:3000/#/register
2. **Register a valid user**:
   - Email: emptyreg@email.com
   - Password: Pass123!
   - Security Question: "Mother's maiden name?"
   - Answer: Smith
3. **Intercept the request** (e.g., Burp Suite):

   text
   POST /api/Users HTTP/1.1
   {
    "email": "emptyreg@email.com",
    "password": "Pass123!",
    "role": "customer"
   }

4. **Resend the request** via Repeater → 400 Bad Request (email exists).
5. **Modify the request** by removing email and password:

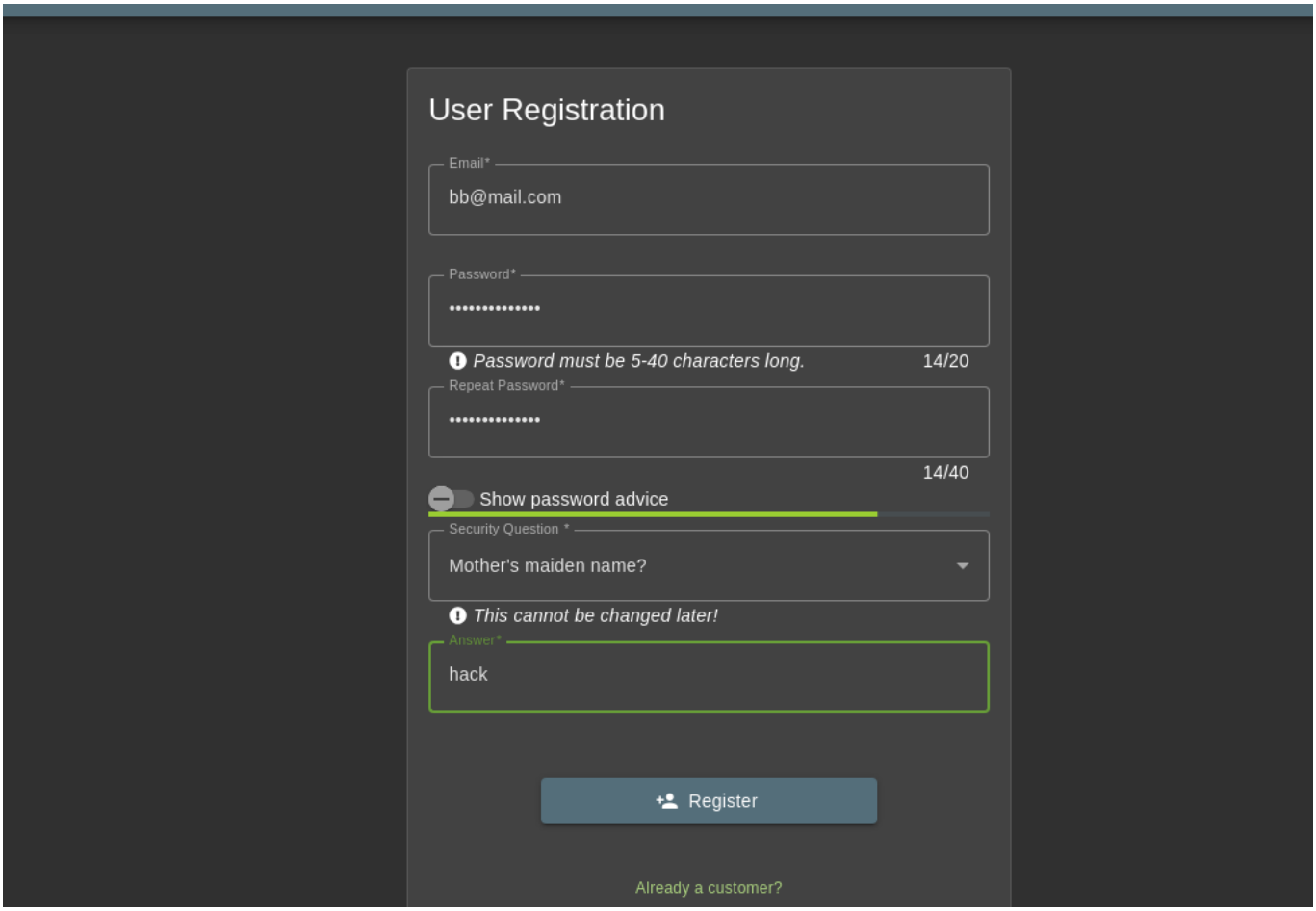
   text
   POST /api/Users HTTP/1.1
   {
    "role": "customer"
   }

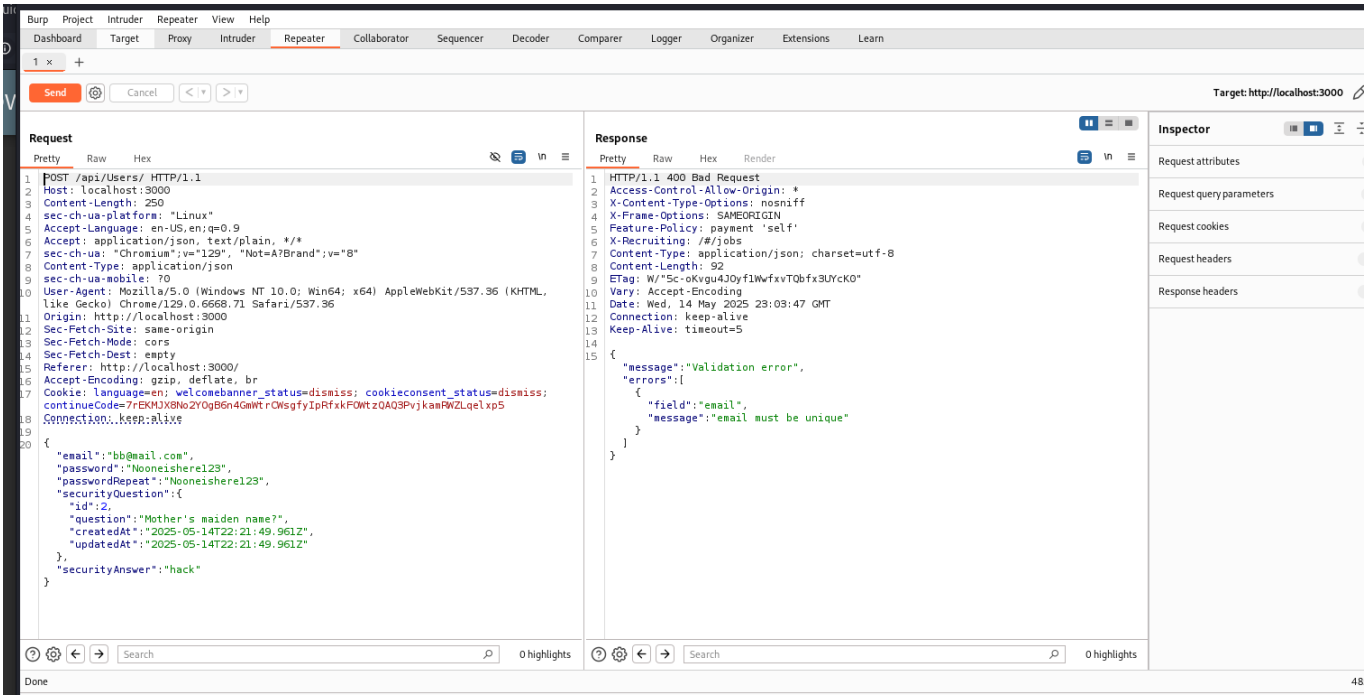
6. **Confirm vulnerability**: Server responds 201 Created, creating an empty account and solving the challenge.

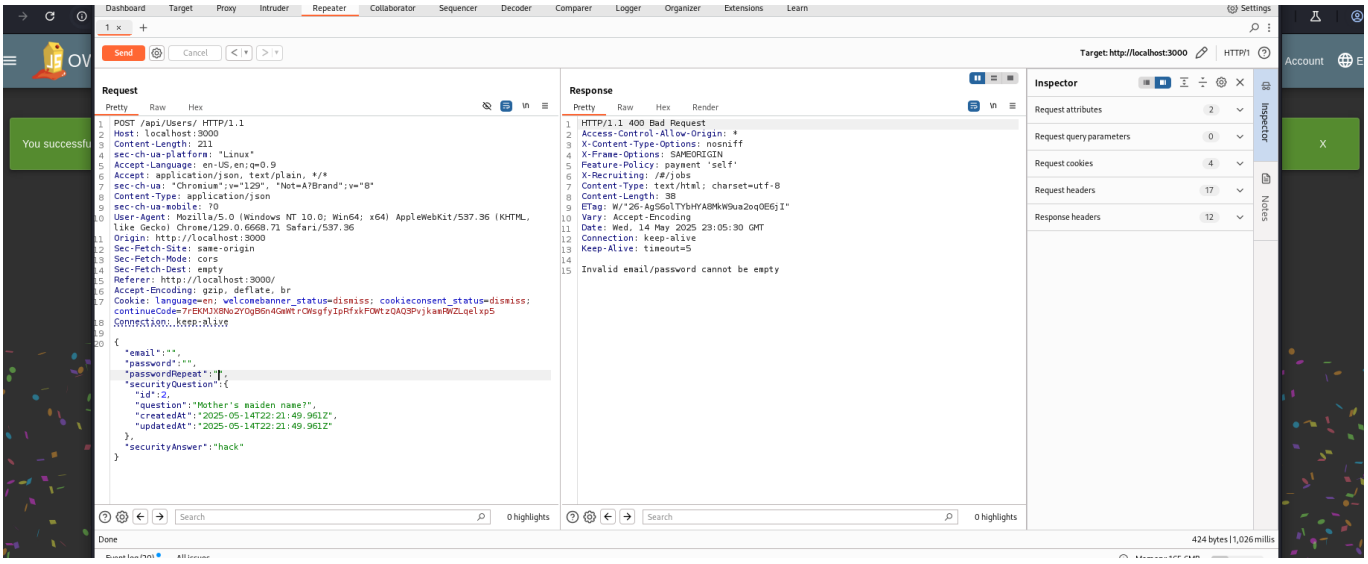
Impact

- Invalid accounts disrupt user management.

- Bypasses registration validation integrity.
- Enables anonymous/untraceable account creation.
- Risks system errors from incomplete data.







You successfully solved a challenge: Empty User Registration (Register a user with an empty email and password.)

24) Broken authentication – reset a forgot password by security

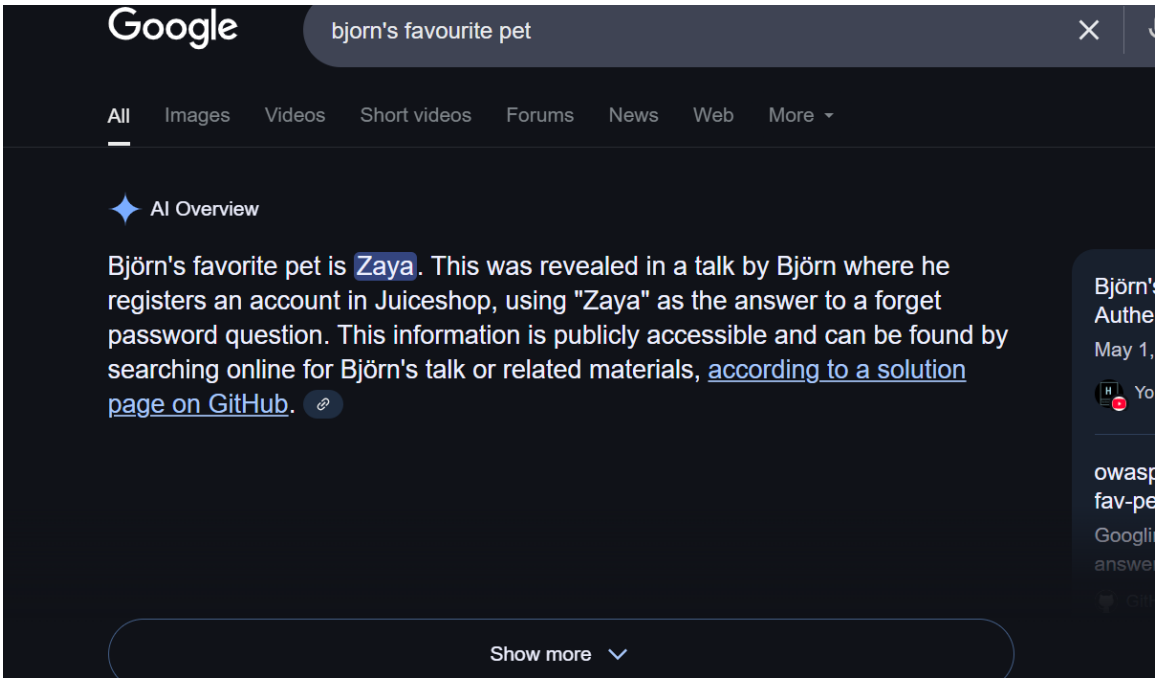
Severity :Medium

Description:
Broken Authentication occurs when an application fails to adequately protect user credentials, allowing unauthorized password resets. In OWASP Juice Shop, the Forgot Password mechanism for Bjoern's account (bjoern@owasp.org) is vulnerable due to insufficient validation of security question answers. By researching publicly available information, specifically identifying Bjoern's favorite pet as "zaya" from a related video, an attacker can reset his password without proper authentication, completing the "Bjoern's Favorite Pet" challenge.

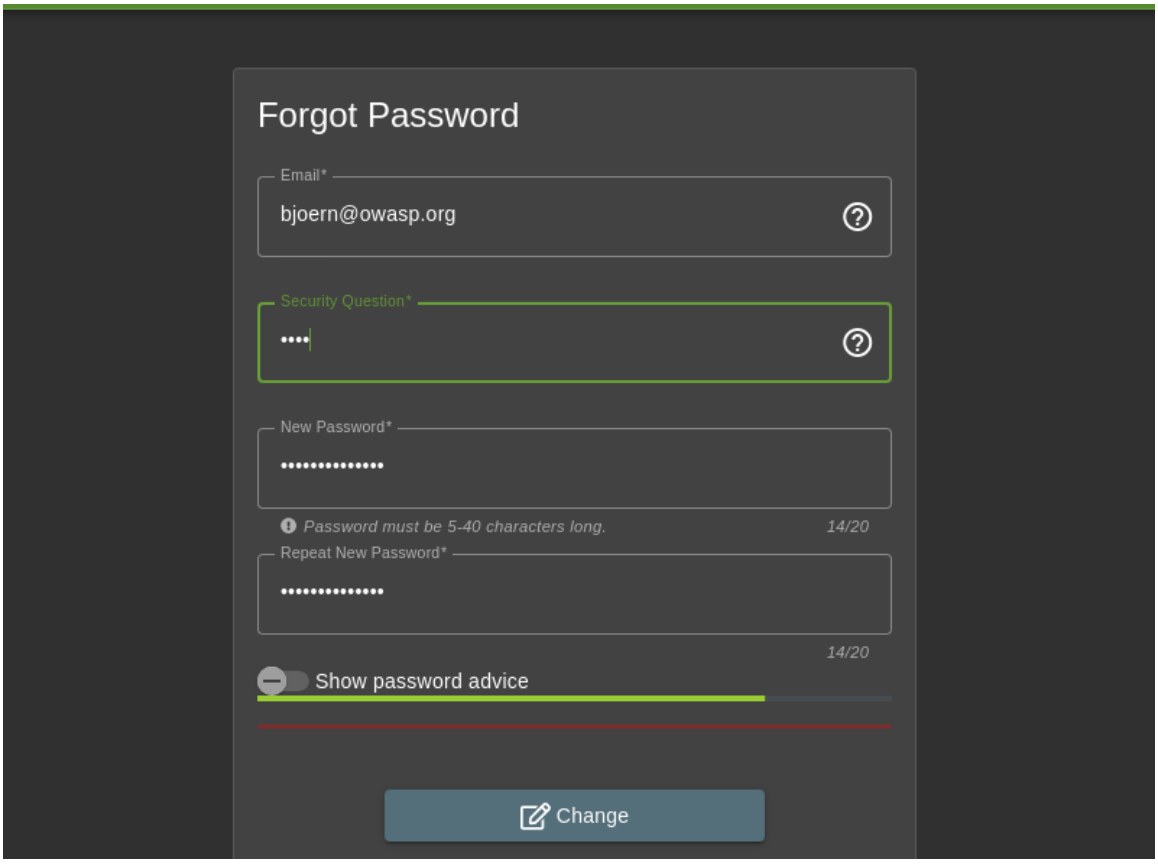
Affected Component:
Forgot Password endpoint
/forgot-password
The endpoint does not enforce strong authentication or restrict access to security question answers, enabling password resets with easily obtainable information.

Steps to Reproduce (in OWASP Juice Shop):

1. Navigate to the Forgot Password page:
   http://localhost:3000/#/forgot-password
2. Enter Bjoern's email address: bjoern@owasp.org.
3. Search for information about Bjoern, discovering from a video that his favorite



4. pet is "zaya," which is the required answer for his security question.
5. Submit the Forgot Password request with the email and answer "zaya" to the security question.
6. Receive a password reset token or confirmation, allowing the attacker to set a new password for Bjoern's account.



7. The challenge is marked as solved: "You successfully solved a challenge: Bjoern's Favorite Pet."

Impact:
- Unauthorized access to Bjoern's account due to weak authentication controls.
- Exposure of account takeover risks if security questions are publicly deducible.
- Potential compromise of other user accounts with similar weak authentication mechanisms.
- Undermines trust in the password recovery process, affecting overall security.

25) Broken access control – Admin section – 2

Severity: medium

Description

A **Broken Access Control** vulnerability exists in the "Admin Section" of OWASP Juice Shop. The application fails to enforce proper role-based access control, allowing any authenticated user to directly access the administrator dashboard by navigating to its known URL path, without validating whether the user has admin privileges.

Affected Component

GET /administration

Steps to Reproduce

1. Logged into the Juice Shop using a **standard (non-admin) user** account.
2. Manually accessed the admin panel by entering the URL directly in the browser:

   https:// http://localhost:3000/#/administration

3. The server responded with HTTP 200 OK, and the full admin dashboard was rendered, despite the user having no admin privileges.
4. Admin-only functionalities like **user management**, **review control**, and **application settings** were accessible.

Impact

This vulnerability allows an attacker with any valid user account to escalate privileges and gain unauthorized **administrator-level access**. Potential consequences include:

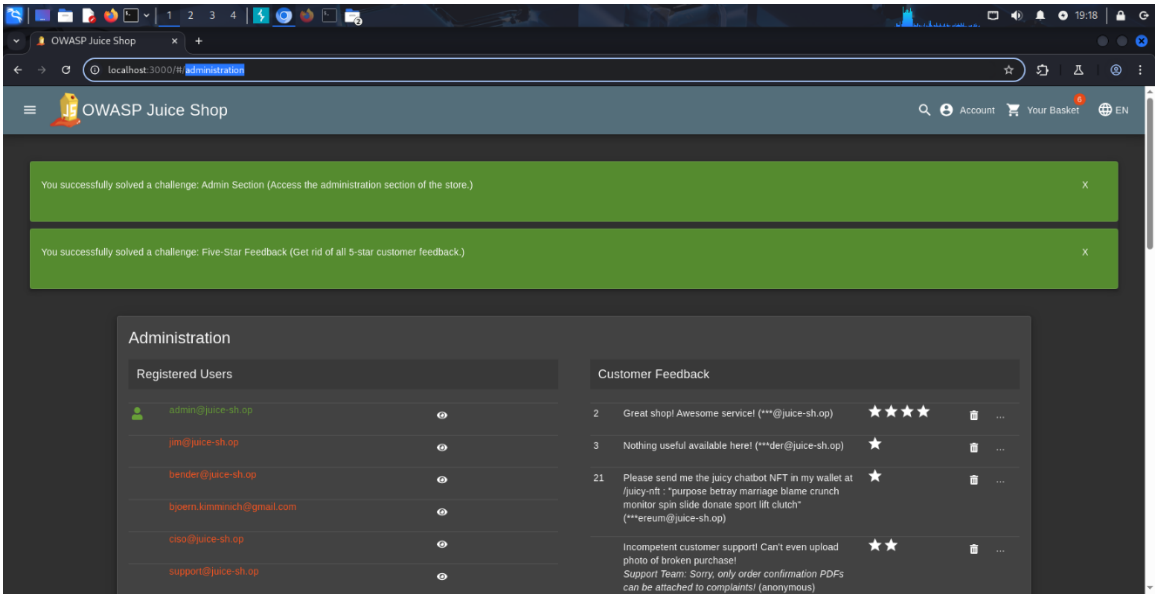
- Viewing or modifying sensitive user data
- Managing product reviews
- Making configuration-level changes to the application
- Chaining with other vulnerabilities for deeper system compromise

This is a clear case of **Vertical Privilege Escalation**, and due to the extent of control gained, the severity is **Critical**.

Remediation

- Implement strict **role-based access control (RBAC)** on the backend to validate each request to /administration.
- Do not rely on frontend routing or visibility toggles to enforce access rights.
- Server-side logic should check the user's role from the session or token and **deny access** unless the role is explicitly admin.
- Return **HTTP 403 Forbidden** for unauthorized access attempts.

Poc



26) Broken access control – sensitive data in the source code

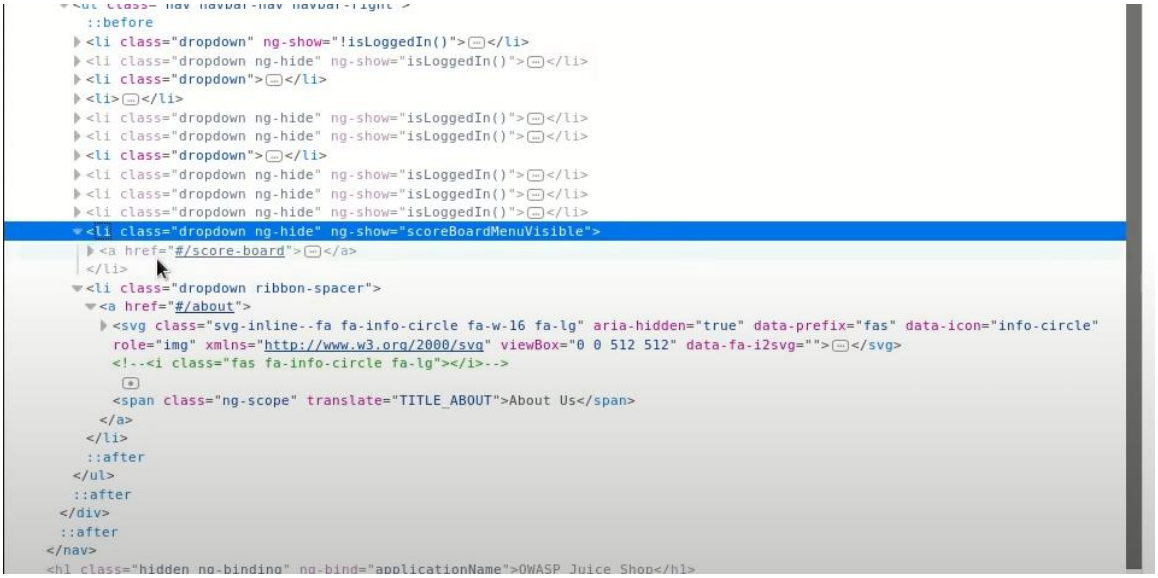
Severity: low

Description

A **Broken Access Control** vulnerability exists in the OWASP Juice Shop. The application allows users to inspect the page of the front so anyone can find the hidden path of the scoreboard .

PoC :

1 – got to website dashboard
2 – right click -> go to view page source code
3 – you will find a hidden url that leads to score board



27) Broken authentication – meta geo stalking

Severity: High

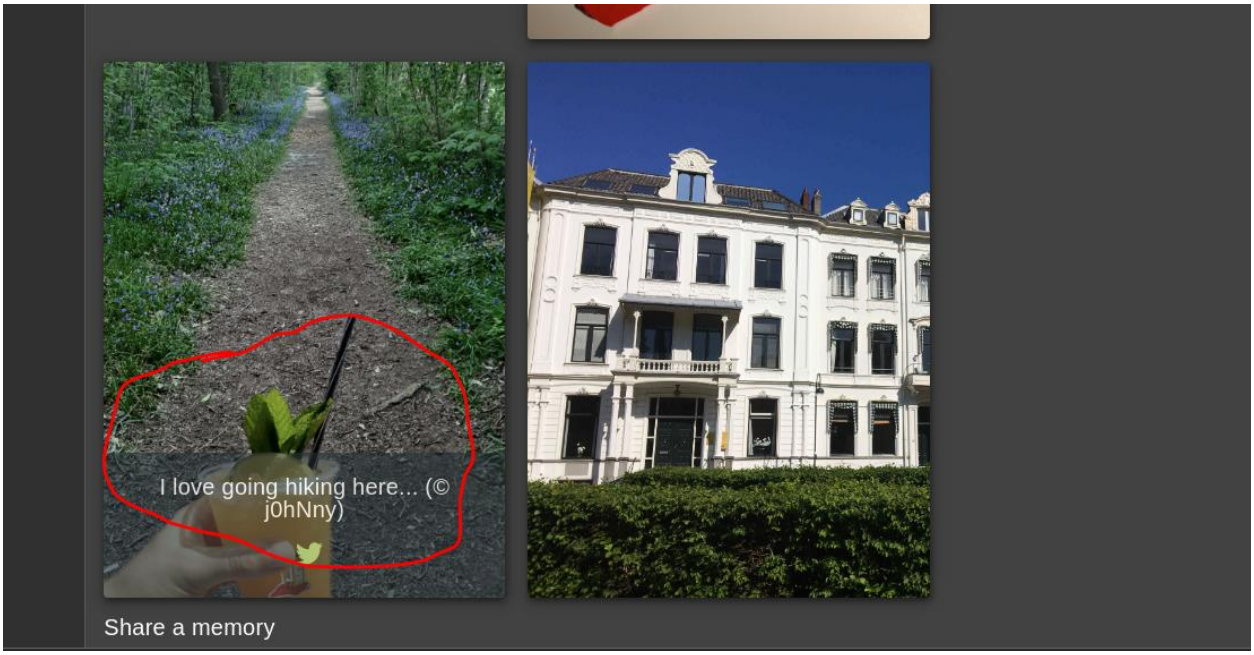
Description:
A Meta Geo Stalking vulnerability exists in OWASP Juice Shop's Photo Wall and Forgot Password features. The Photo Wall exposes images with embedded GPS metadata, revealing the user's location. By extracting this metadata, an attacker can determine the answer to a user's security question (e.g., "What's your favorite place to go hiking?"). Using the default domain name (@juice-sh.op), the attacker can access the Forgot Password functionality, input the victim's email (e.g., john@juice-sh.op), retrieve the security question, and reset the password, leading to a full account takeover. This completes the "Meta Geo Stalking" challenge.

Affected Components:

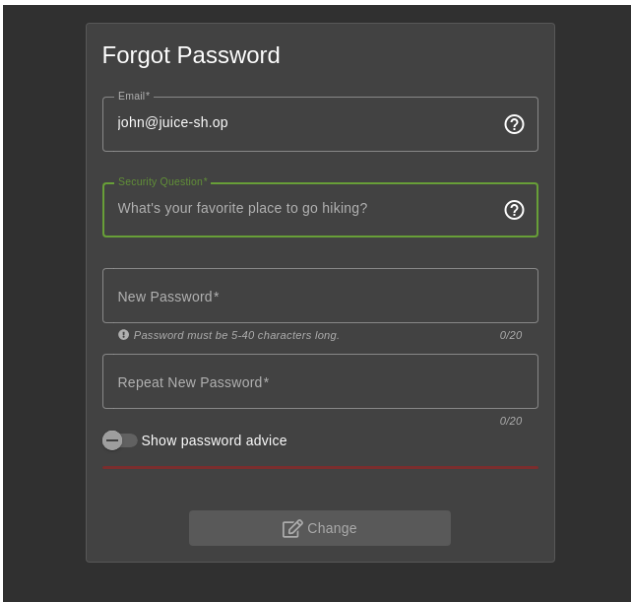
- **Photo Wall**: Exposes images with unstripped GPS metadata, revealing user locations.
- **Forgot Password Functionality**: Allows retrieval of security questions with predictable email domains, enabling account takeover.

Steps to Reproduce (in OWASP Juice Shop):

1. Navigate to the "Photo Wall" section at http://localhost:3000/#/photo-wall.



2. Identify a photo uploaded by the target user (e.g., a photo with the caption "I love going hiking here ... (@johnny)") and save the image (e.g., favorite-hiking-place.png).
3. Note that the default email domain for users in OWASP Juice Shop is @juice-sh.op, so the target's email is likely john@juice-sh.op.
4. Navigate to the "Forgot Password" section at http://localhost:3000/#/forgot-password.

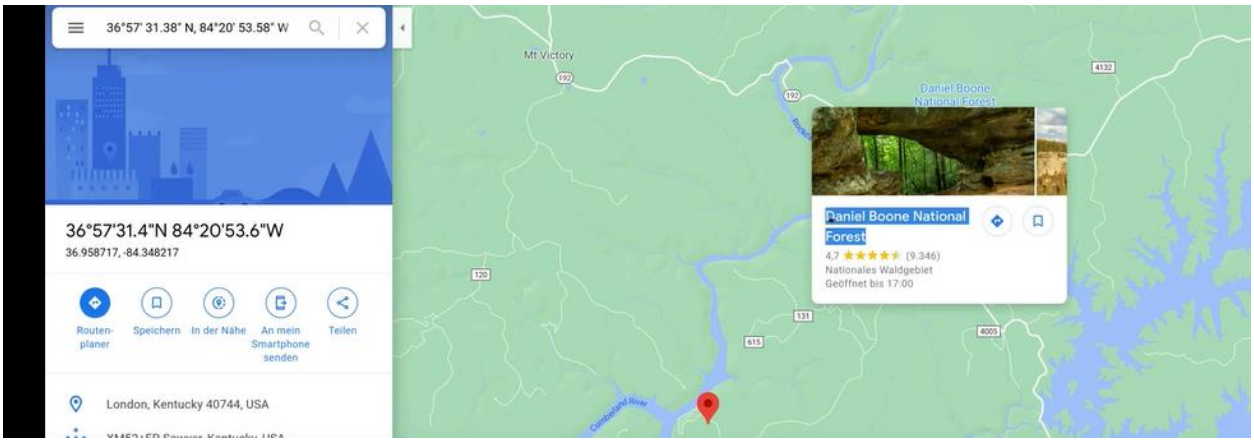


5. Enter the target's email (john@juice-sh.op) to retrieve the security question: "What's your favorite place to go hiking?".
6. Use an EXIF metadata extraction tool (e.g., ExifTool) to analyze the downloaded image.

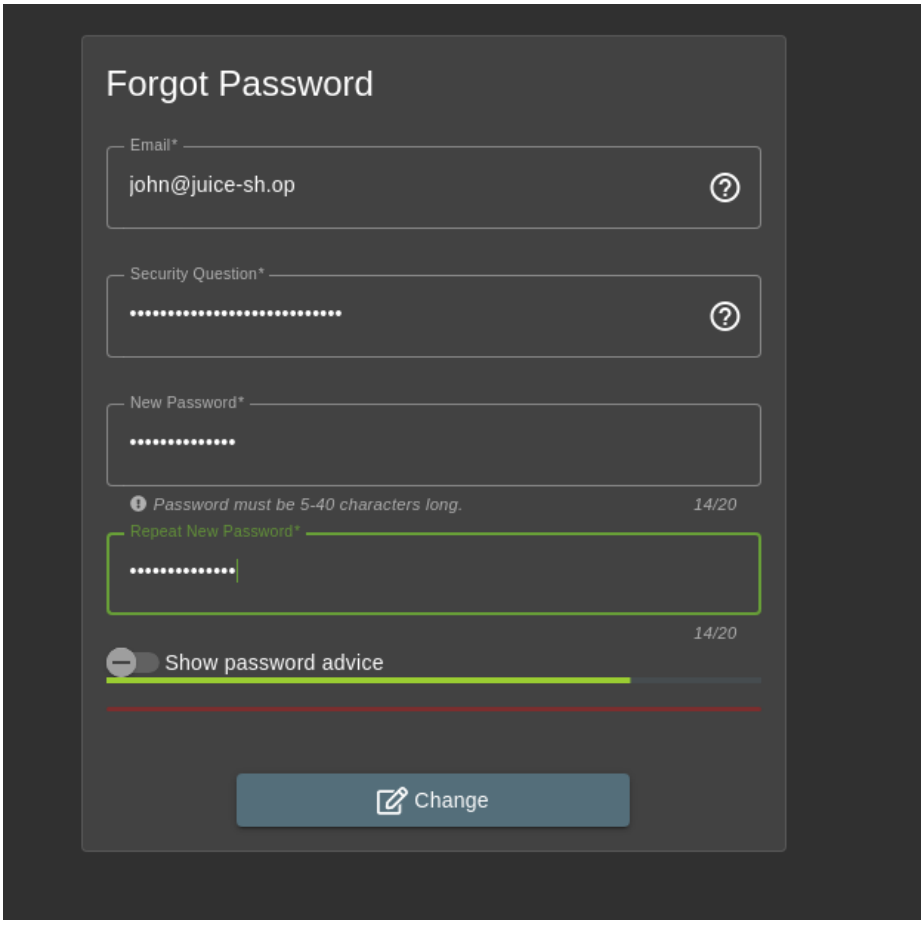
7. Extract the GPS coordinates from the image metadata (e.g., 36°57'31.4" N 84°20'53.6" W).

```
┌──(root㉿kali)-[~/Desktop]
└─# exiftool favorite-hiking-place.png
ExifTool Version Number         : 13.00
File Name                       : favorite-hiking-place.png
Directory                       : .
File Size                       : 667 kB
File Modification Date/Time     : 2025:05:15 17:53:37-04:00
File Access Date/Time           : 2025:05:15 17:53:41-04:00
File Inode Change Date/Time     : 2025:05:15 17:53:41-04:00
File Permissions                : -rw-r--r--
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 471
Image Height                    : 627
Bit Depth                       : 8
Color Type                      : RGB
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
Exif Byte Order                 : Little-endian (Intel, II)
Resolution Unit                 : inches
Y Cb Cr Positioning             : Centered
GPS Version ID                  : 2.2.0.0
GPS Latitude Ref                : North
GPS Longitude Ref               : West
GPS Map Datum                   : WGS-84
Thumbnail Offset                : 224
Thumbnail Length                : 4531
SRGB Rendering                  : Perceptual
Gamma                           : 2.2
Pixels Per Unit X               : 3779
Pixels Per Unit Y               : 3779
Pixel Units                     : meters
Image Size                      : 471×627
Megapixels                      : 0.295
Thumbnail Image                 : (Binary data 4531 bytes, use -b option to extract)
GPS Latitude                    : 36 deg 57' 31.38" N
GPS Longitude                   : 84 deg 20' 53.58" W
GPS Position                    : 36 deg 57' 31.38" N, 84 deg 20' 53.58" W
```

8. Input the GPS coordinates into Google Maps to identify the location (e.g., Daniel Boone National Forest, London, Kentucky 40744, USA).



9. Return to the Forgot Password page, enter the security question answer (Daniel Boone National Forest), and set a new password (e.g., NewPass123!).
10. Submit the password reset by clicking "Change".



11. Observe the success message: "You successfully solved a challenge: Meta Geo Stalking (Determine the answer to John's security question by looking at uploaded metadata in the Photo Wall and use it to reset his password via the Forgot Password mechanism.)".

Impact:

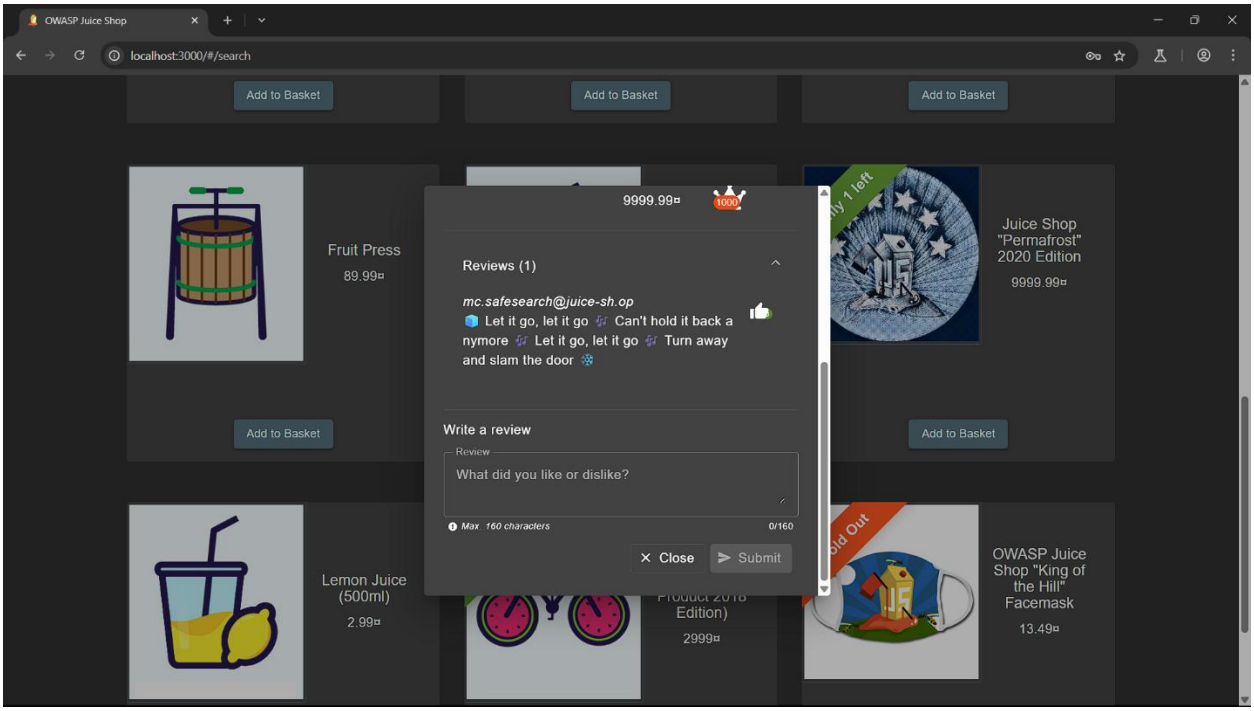
- Exposure of sensitive user location data through unstripped GPS metadata in uploaded images.
- Full account takeover by exploiting predictable email domains and security question answers derived from metadata.
- Potential for further unauthorized actions, such as accessing sensitive user data or impersonating the victim.
- Violation of privacy best practices, increasing the risk of targeted attacks (e.g., physical stalking or social engineering).

28) Broken access control – login MC Safesearch

Severity: Medium

Description:

The application allows logging in as the user **MC SafeSearch** using default credentials without requiring SQL Injection or bypass techniques. The credentials were discovered through:

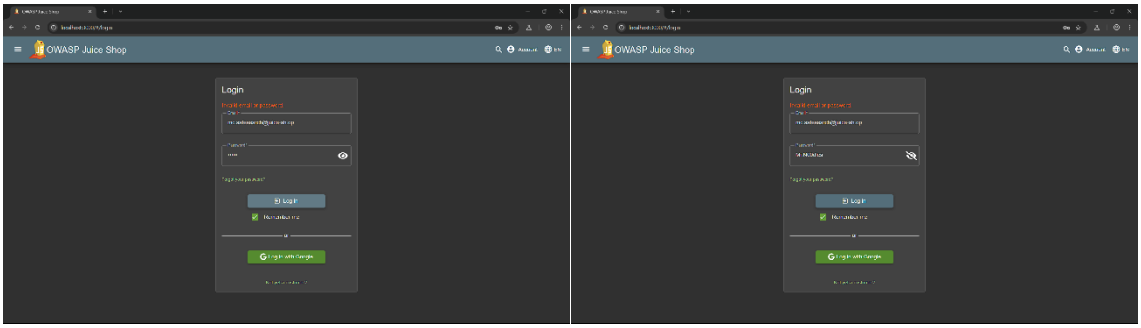


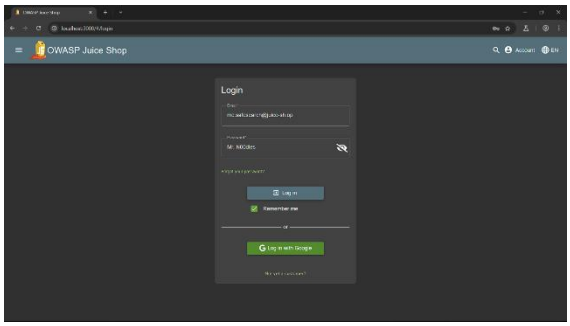
29) **Product Review Analysis**: A review by mc.safesearch@juice-sh.op contained lyrics from *Frozen*'s "Let It Go" (Let it go, let it go ... ❄), hinting at a weak password themed around the song (e.g., LetItGo - LetItGo123 – Frozen – Elsa - MrN00dles).

30) Enter the credentials:

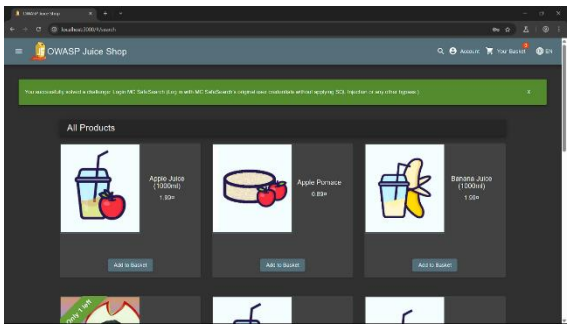
Email: mc.safesearch@juice-sh.op

Password: trying different ones!





31) And finally



32) Broken authentication – GDPR

Severity: High

Description:

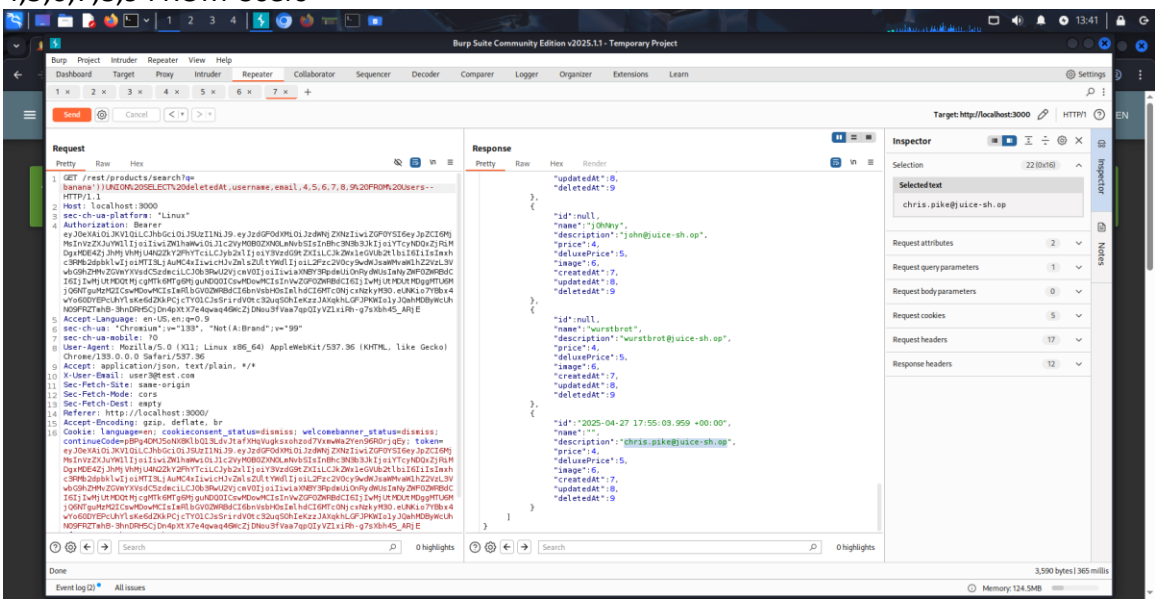
A previously registered user identified by email chris.pike@juice-sh.op is still **retrievable through SQL Injection**, even though the account is **marked as deleted** via the deletedAt field. This proves the system does **not fully erase or anonymize user data** in accordance with GDPR Article 17 (Right to erasure / right to be forgotten).

Steps to Reproduce:

1. Challenge solved: Database Schema.

2. Authenticated as a valid user.
3. Performed SQL Injection via:

   GET /rest/products/search?q=banana')) UNION SELECT deletedAt, username, email, 4,5,6,7,8,9 FROM Users--



Impact:

- The user chris.pike is marked as deleted but their **personally identifiable information (PII)** remains accessible.
- Violates **GDPR Article 17**, putting the organization at legal risk.
- An attacker can enumerate and expose deleted users' data.
- Users lose trust in the platform's privacy guarantees.

33) Broken auth – Password strength

Severity: High

Description:

The login form of the application does not implement protection mechanisms against brute-force attacks. Combined with the fact that the **administrator account uses a weak default password**, this allows attackers to perform credential stuffing using common password lists to successfully gain administrative access.

Affected Component:

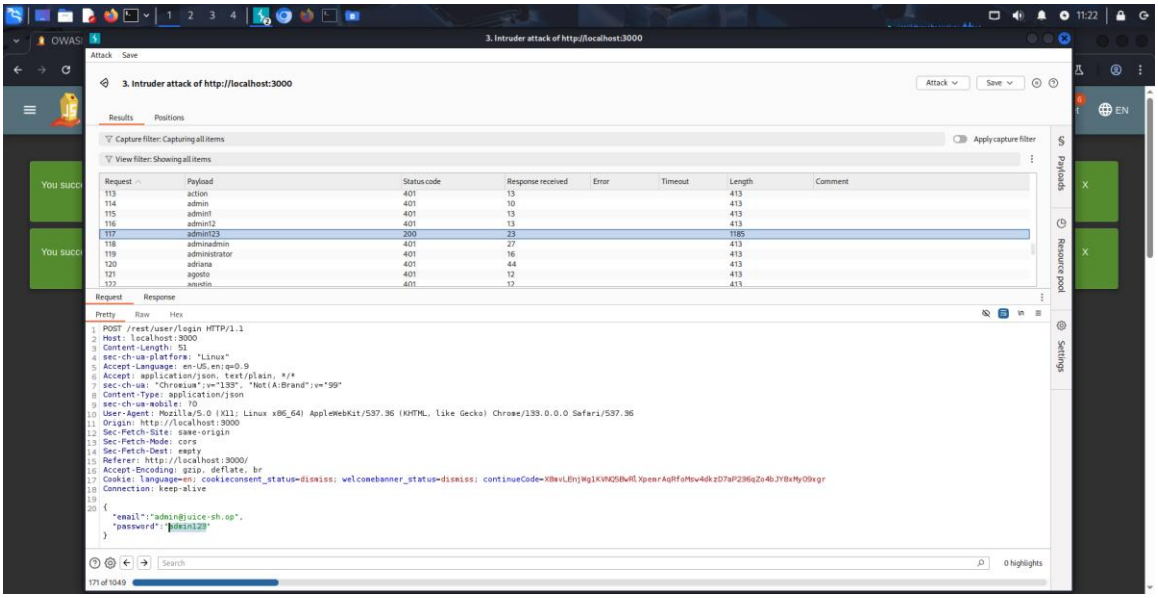
/login form – authentication endpoint

Steps to Reproduce:

1. Identified the admin email (admin@juice-sh.op) from a previous challenge (recon via comments/posts).
2. Opened Burp Suite and captured a valid login request with the known email.
3. Sent the request to **Intruder**, configured it to **fuzz the password field** using the payload list from:

   Daniel Miessler's SecLists:
   passwords/common-credentials/best1050.txt
   (https://github.com/danielmiessler/SecLists)

4. Started the attack.



5. One of the requests received a **200 OK** response with an authentication token in the body, indicating a successful login.
6. Verified that the login led to the **admin dashboard**, confirming full compromise.

Impact:

- **Full administrative access** to the application without needing to exploit any injection or misconfiguration.
- Enables the attacker to:
  - View and delete user data.
  - Access internal functions (e.g., feedback management, user roles).
  - Perform lateral movement within the app or infrastructure.

34) Security Misconfiguration – Complaint section

Severity: Medium

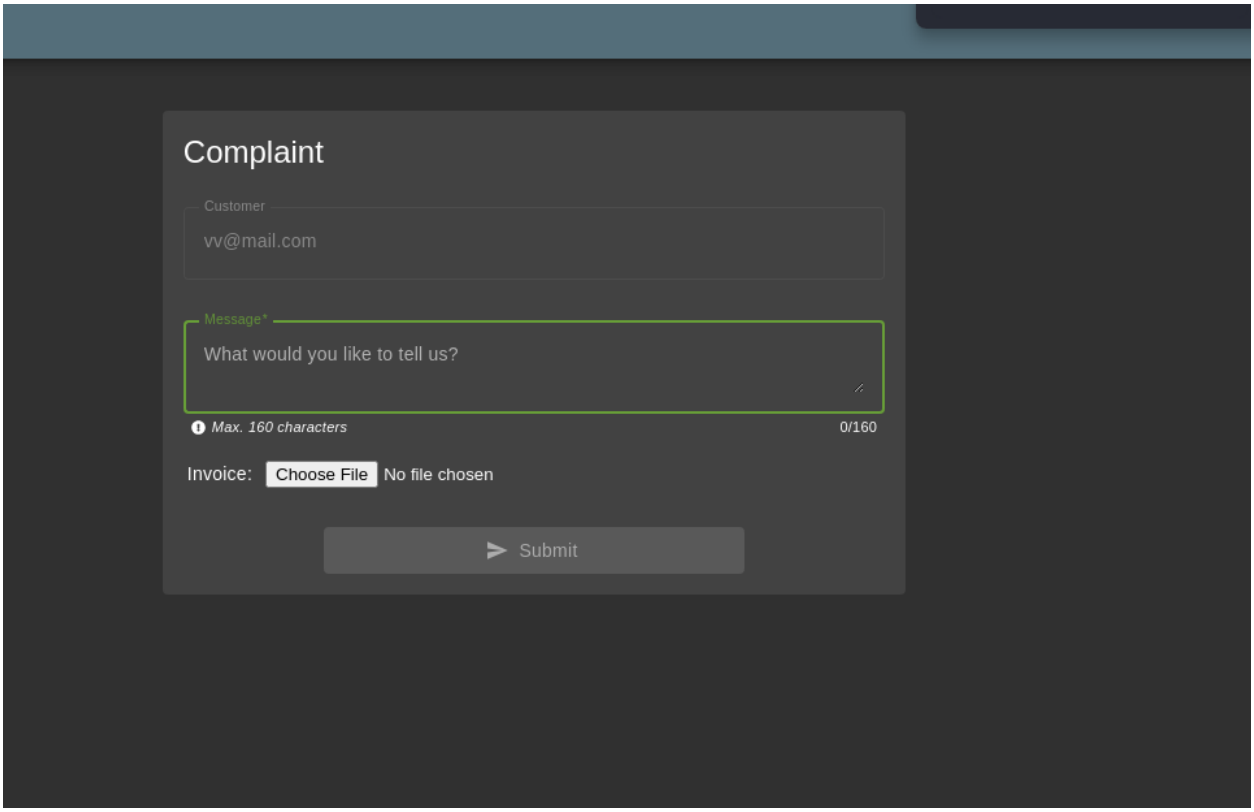
Description:
A Security Misconfiguration vulnerability exists in OWASP Juice Shop's complaint section, where the file upload feature uses a deprecated interface. The UI restricts uploads to PDF and ZIP files, but the backend accepts XML files due to a misconfiguration in the validation logic. This discrepancy allows attackers to upload potentially malicious XML files, which could lead to XML External Entity (XXE) attacks or other exploits, completing the "Deprecated Interface" challenge.

Affected Component:

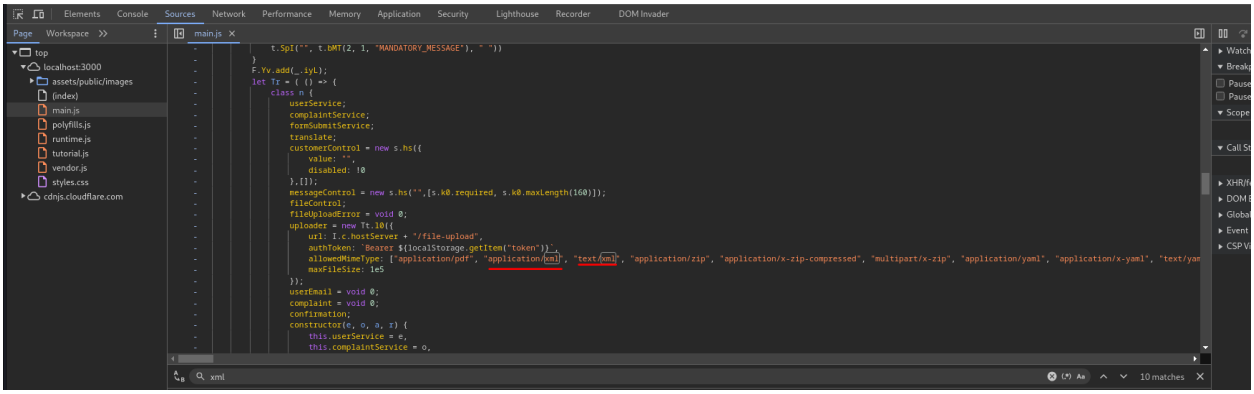
- **Complaint Section File Upload**: The file upload feature in the complaint section (/complaints) improperly validates file types, allowing XML files despite UI restrictions.

Steps to Reproduce (in OWASP Juice Shop):

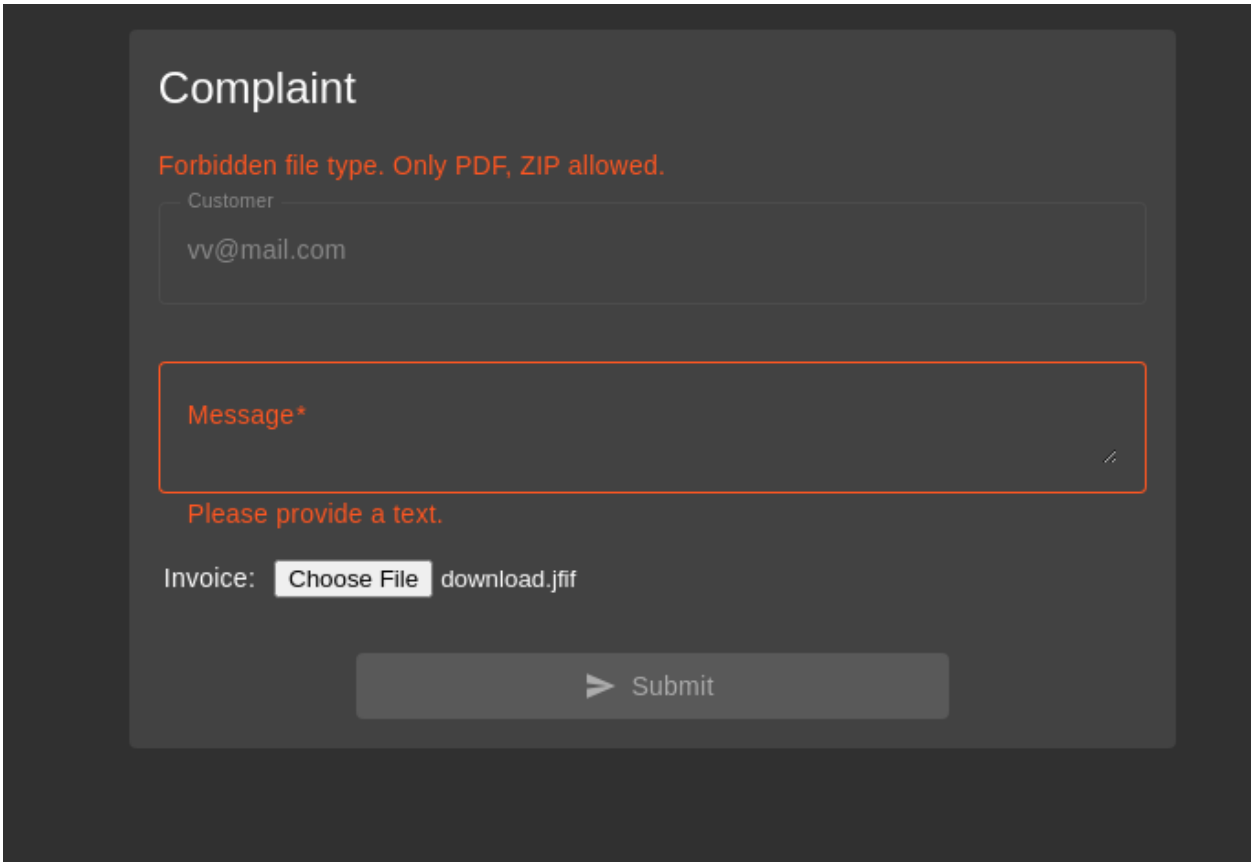
1. Navigate to the "Complaint" section at http://localhost:3000/#/complain.
2. Enter a customer email (e.g., vvc@gmail.com) and a message (e.g., "fgd").
3. Attempt to upload a file by clicking "Choose File" in the "Invoice" section.



4. Observe the file picker dialog, which indicates only PDF and ZIP files are allowed (e.g., dropdown shows "PDF (*.pdf)" and "ZIP (*.zip)").



5. Inspect the page source code by right-clicking and selecting "Inspect" or using a browser's developer tools (e.g., Chrome DevTools).
6. Locate the JavaScript file (e.g., main.js) and search for file upload validation logic.



7. Identify that the backend accepts XML files (e.g., application/xml is listed in the accepted MIME types alongside application/pdf and application/zip).

8. Create or select an XML file (e.g., test.xml) and attempt to upload it by selecting "All Files" in the file picker and choosing the XML file.
9. Submit the complaint by clicking "Submit".

10. Observe the success message: "You successfully solved a challenge: Deprecated Interface (Use a deprecated B2B interface that was not properly shut down.)", confirming the XML file was accepted.

Impact:

- Bypassing file type restrictions allows attackers to upload XML files, which could be exploited for XML External Entity (XXE) attacks, potentially leading to data exfiltration or server-side request forgery (SSRF).
- Use of a deprecated interface increases the attack surface, as outdated components may have unpatched vulnerabilities.
- Potential for further exploitation if uploaded XML files are processed unsafely by the backend.
- Violation of secure file upload best practices, undermining user trust and application integrity.

35) Security Misconfiguration – Error handling

Severity: low

Description:

The application reveals **detailed error messages** containing stack traces and internal implementation details when certain invalid requests are made. This information can be used by an attacker to **enumerate backend technologies**, understand business logic, and **identify potential entry points** for further exploitation.

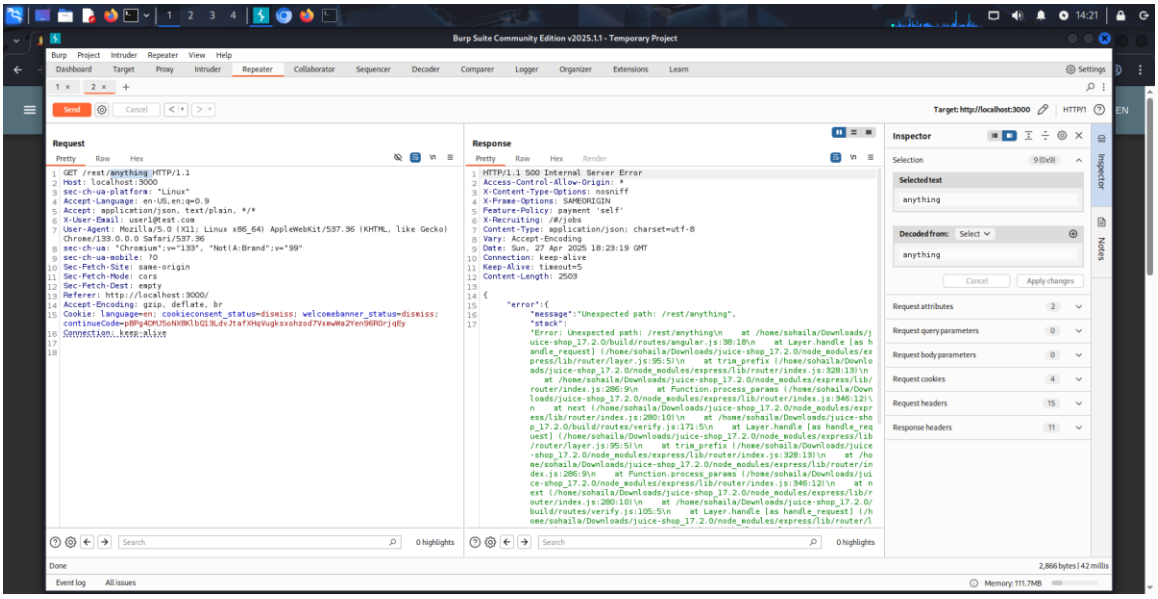
Affected Component:

Multiple endpoints – error responses when sending malformed requests or triggering edge cases

Steps to Reproduce:

1. Send the following HTTP request:

   GET /rest/anything HTTP/1.1



2. The application responded with a **500 Internal Server Error** and displayed a **stack trace** showing:
    - Internal class names
    - File paths
    - Node.js error messages
    - Possibly database errors (in some cases)
3. This confirmed the app does not properly sanitize or suppress internal errors.

Impact:

- Provides **insight into the backend stack**, including libraries, error types, and file structure.
- Can assist attackers in crafting targeted attacks (e.g., exploiting known vulnerabilities in specific packages).
- May expose sensitive logic or misconfigurations.

36) CSRF – Profile update endpoint

Severity: High

Description:

The /profile endpoint is vulnerable to Cross-Site Request Forgery (CSRF). The application fails to validate the origin of the request and lacks proper anti-CSRF mechanisms (e.g., CSRF tokens or SameSite cookie attributes). This allows an attacker to craft a malicious HTML page that forces a logged-in victim to unknowingly change their profile data—such as the username—without their consent, resulting in unauthorized actions being performed on their behalf.

Affected Component:
POST /profile – Profile update endpoint

Steps to Reproduce:

1. Created a malicious HTML file named csrf.html with the following content:

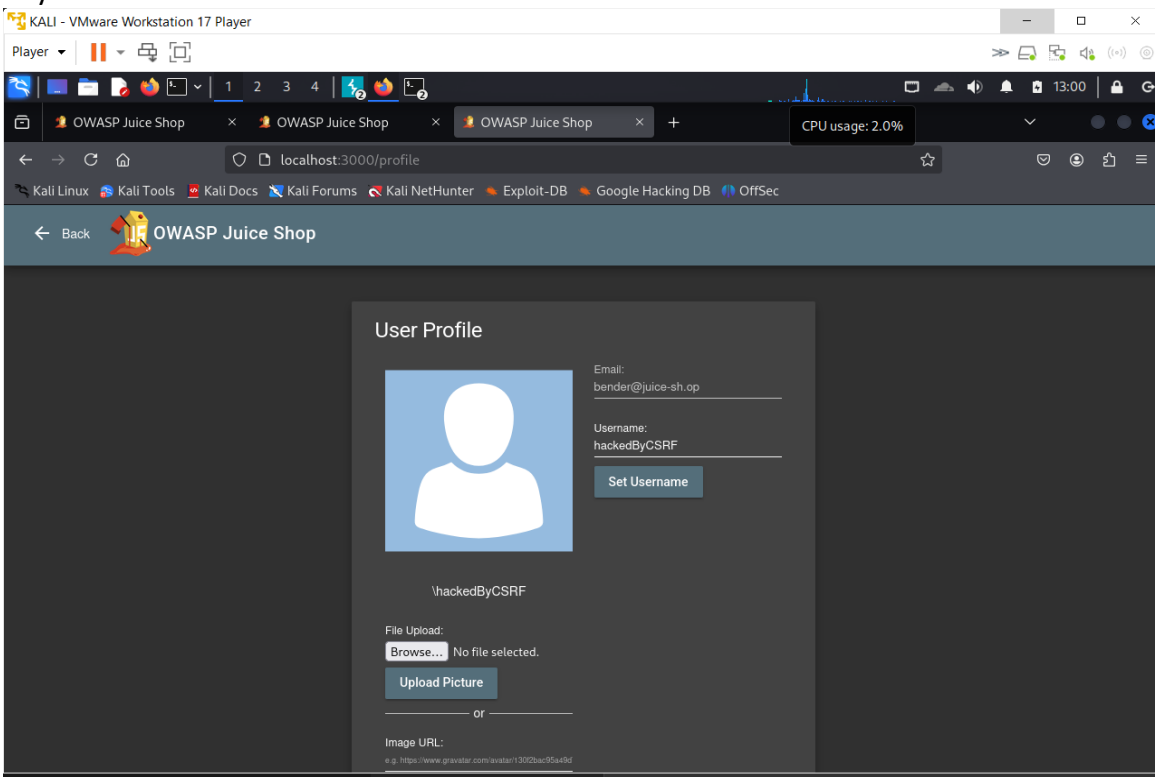
   ```
   <!DOCTYPE html>
   <html>
    <body onload="document.forms[0].submit()">
     <form action="http://localhost:3000/profile" method="POST">
      <input type="hidden" name="username" value="hackedByCSRF">
     </form>
    </body>
   </html>
   ```

2. Hosted the file using Python's built-in web server:

   python3 -m http.server 8000

3. Sent the link (http://localhost:8000/csrf.html) to a victim logged into the target site.
4. When the victim accessed the link, the browser automatically sent a POST request to http://localhost:3000/profile using the victim's session cookies.

5. The server accepted the request and updated the victim's username without requiring any verification or CSRF token.



37) Sensitive data exposure – FTP directory access

Severity: High

Description:
A Sensitive Data Exposure vulnerability exists in OWASP Juice Shop due to improper FTP server configuration. The "About Us" page contains a hyperlink (ftp/legal.md) that, when hovered over, reveals an FTP directory path (/ftp). Accessing this path exposes the FTP directory structure, including a sensitive file (acquisitions.md), which contains confidential business acquisition plans. This misconfiguration allows unauthorized access to sensitive documents, potentially leading to competitive disadvantage or legal issues.

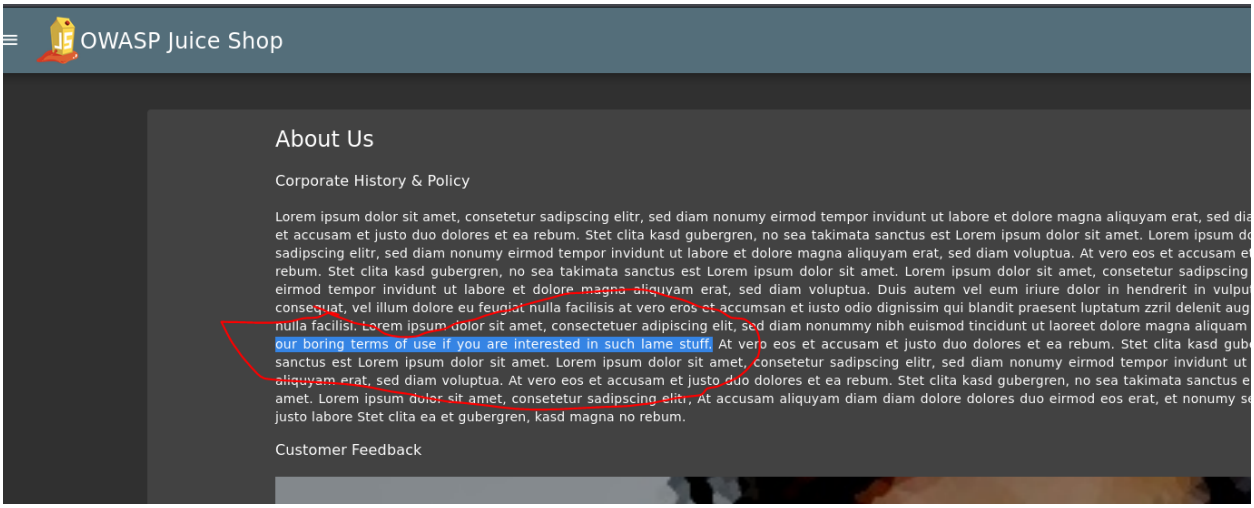
Affected Component:

- **FTP Directory Access**: The FTP server middleware exposes the /ftp directory and its contents, including sensitive files like acquisitions.md.

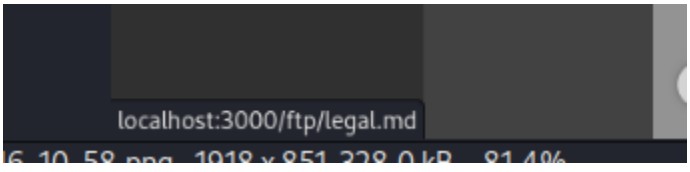
Steps to Reproduce (in OWASP Juice Shop):

1. Navigate to the "About Us" section at http://localhost:3000/#/about.

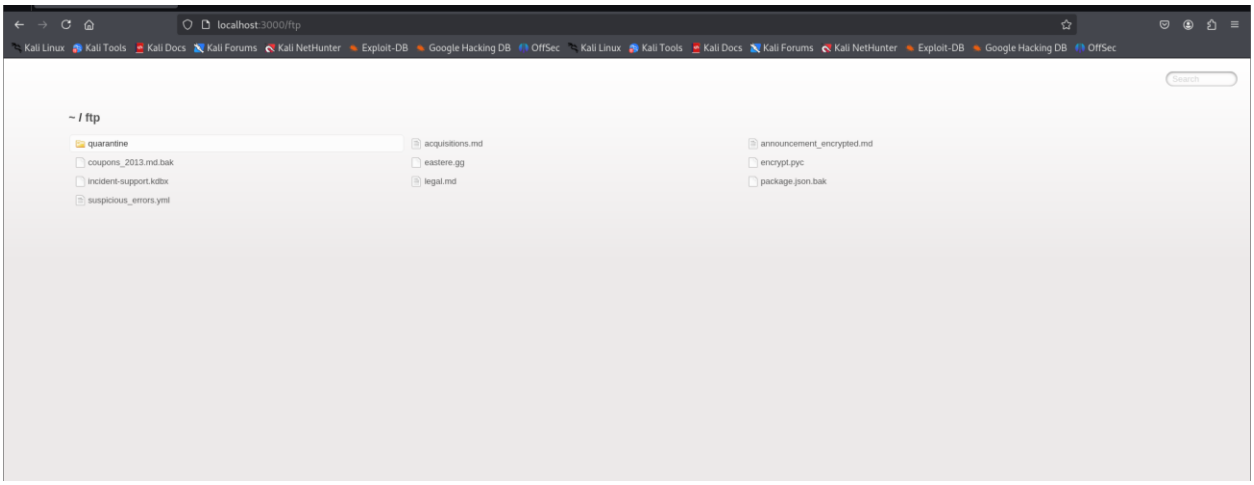
2. Locate the hyperlink text in the "Corporate History & Policy" section that references ftp/legal.md.



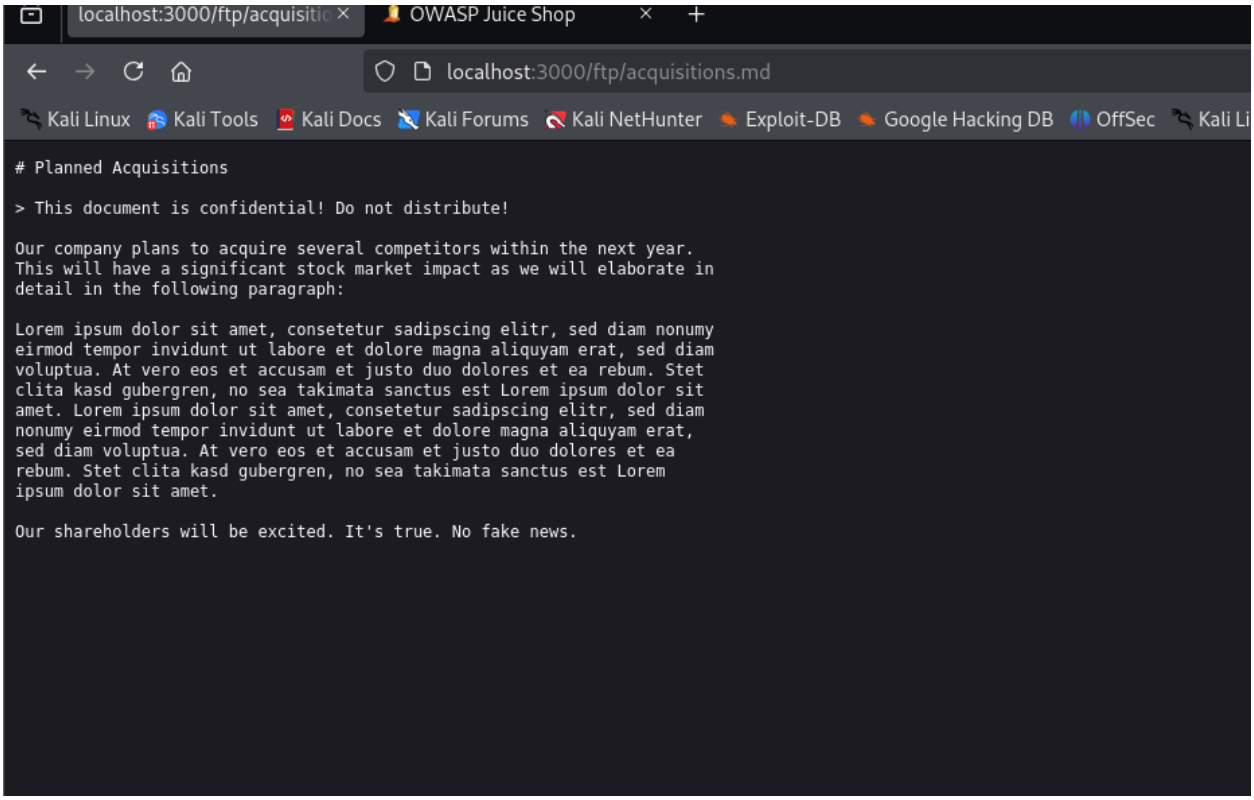
3. Hover over the hyperlink to reveal the URL (http://localhost:3000/ftp/legal.md).



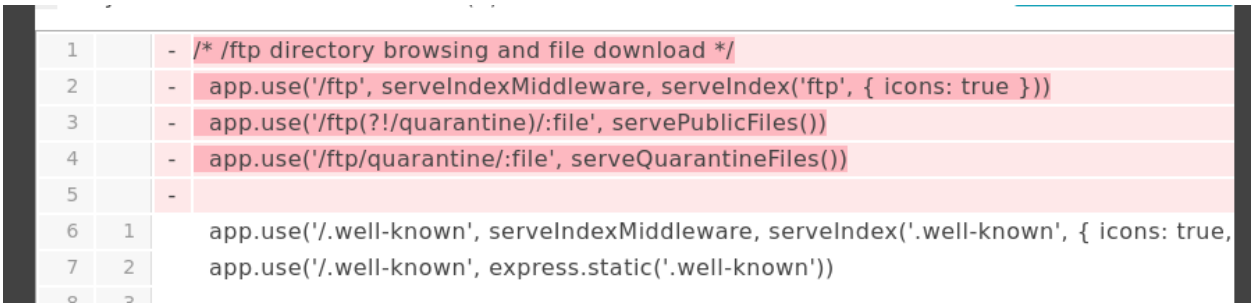
4. Modify the URL by removing /legal.md to access the FTP directory directly (http://localhost:3000/ftp/).



5. Observe the exposed directory structure, which includes files like acquisitions.md, legal.md, and others.



6. Access the acquisitions.md file by navigating to http://localhost:3000/ftp/acquisitions.md.



7. View the contents of acquisitions.md, which reveals sensitive acquisition plans (e.g., "Our company plans to acquire several competitors within the next year, this will have a significant stock market impact as we will elaborate in detail in the following paragraph").

Impact:

- Unauthorized access to sensitive business documents, such as acquisition plans, which could lead to competitive disadvantage if leaked to competitors.
- Potential financial loss due to stock market manipulation based on the exposed acquisition details.
- Legal and regulatory risks due to the exposure of confidential business strategies.
- Erosion of customer and stakeholder trust due to poor data protection practices.

38) Improper input validation – Repetitive registration

Severity: Medium

Description:
The DRY (Don't Repeat Yourself) principle, which mandates that every piece of knowledge should have a single, authoritative representation within a system, is violated in the OWASP Juice Shop registration process. This violation manifests as an Improper Input Validation vulnerability where the /register endpoint relies on client-side validation to ensure "Password" and "Repeat Password" fields match, without enforcing this check on the server-side. The lack of a centralized validation mechanism allows users to bypass the intended DRY-compliant workflow by manipulating the form after an initial mismatch, enabling registration with inconsistent passwords and solving the "Repetitive Registration" challenge.

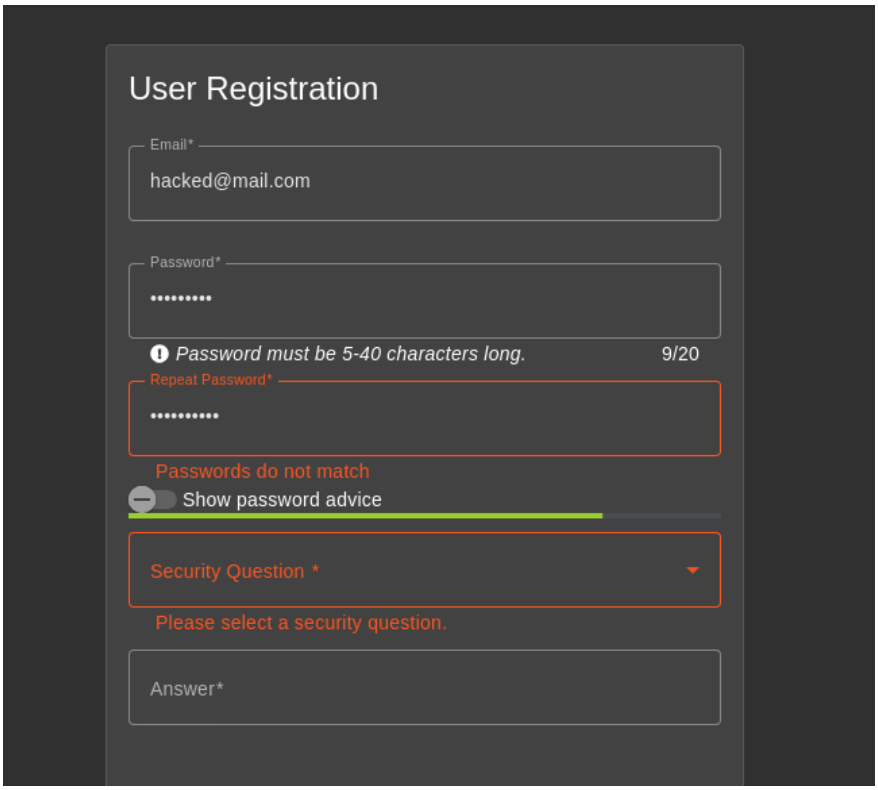
- Affected Component:
Registration endpoint
/register
The endpoint fails to adhere to the DRY principle by duplicating validation logic between client and server, with the server lacking an authoritative validation step to ensure password consistency.

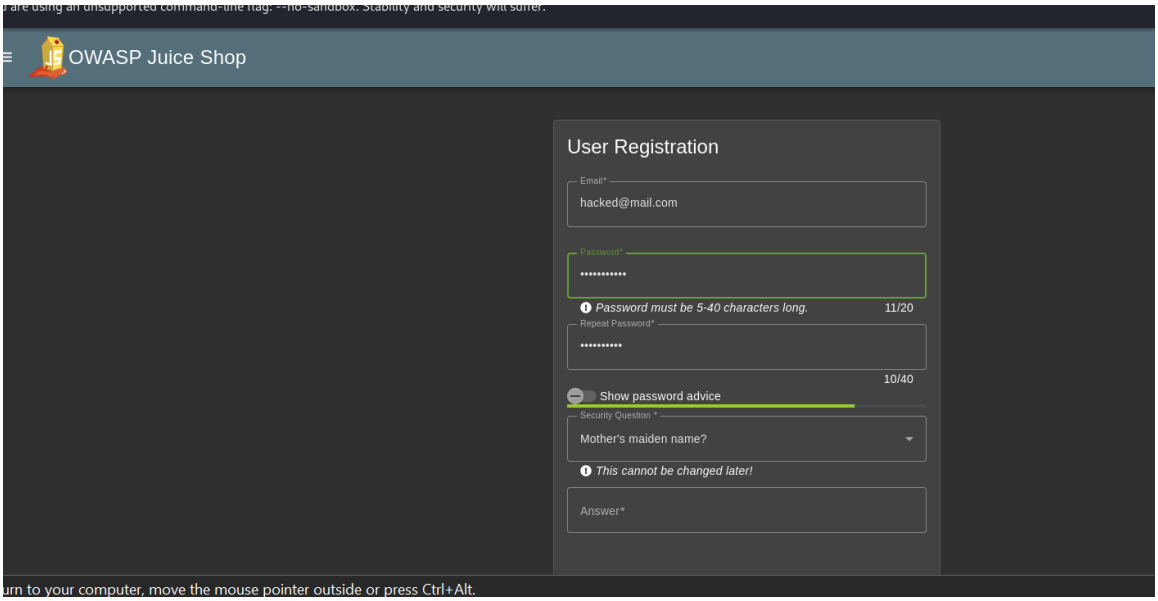
Steps to Reproduce (in OWASP Juice Shop):

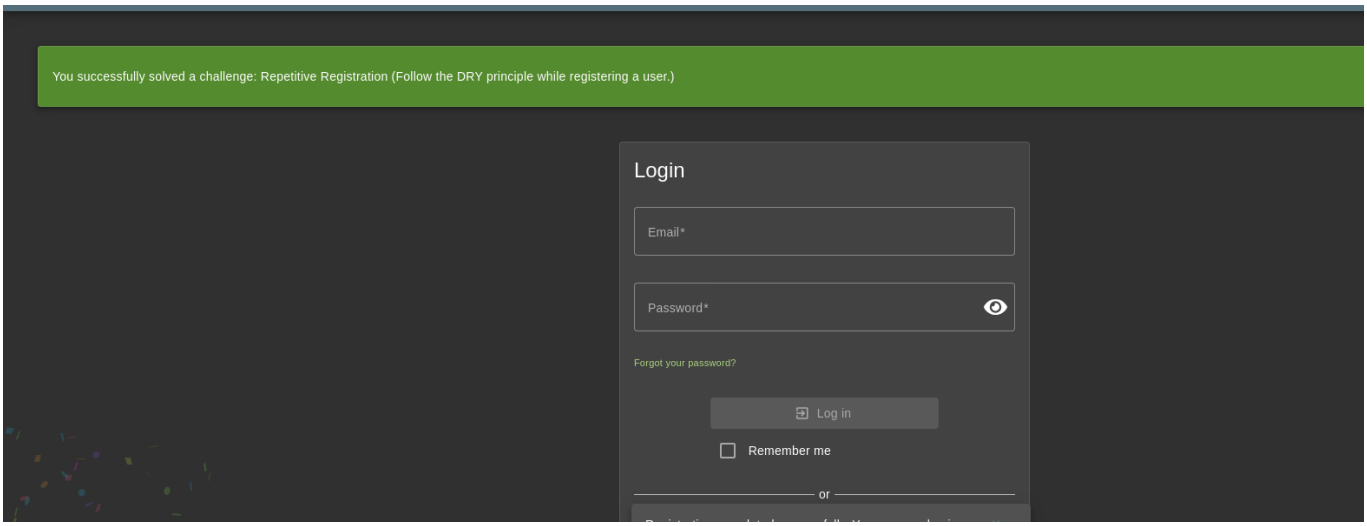
1. Navigate to the registration page: http://localhost:3000/#/register
2. Enter an email (e.g., hacked@email.com).
3. Enter a password (e.g., Pass123!) that meets the 5-40 character requirement.
4. Enter a mismatched repeat password (e.g., Pass456!).
5. Observe the client-side DRY violation warning: "Passwords do not match."
6. Modify the first password field to match the repeat password (e.g., change Pass123! to Pass456!).
7. Select a security question (e.g., "Mother's maiden name?") and provide an answer (e.g., Smith).
8. Submit the form.
9. If vulnerable, the server accepts the registration without re-validating the inputs, completing the "Repetitive Registration" challenge with the message: "You successfully solved a challenge: Repetitive Registration (Follow the DRY principle while registering a user)."

Impact:

• Bypassing of password validation due to the DRY principle violation, leading to potential inconsistencies in user accounts.
• Increased risk of user errors or unintended password usage, undermining the single-source-of-truth concept central to DRY.
• Potential exploitation by attackers to manipulate registration workflows where DRY is not enforced.
• Erodes trust in the registration process due to inconsistent application of the DRY principle.


User Registration
Email*
hacked@mail.com
Password*
Password must be 5-40 characters long.
9/20
Repeat Password*
Passwords do not match
Show password advice
Security Question *
Please select a security question.
Answer*


OWASP Juice Shop
User Registration
Email*
hacked@mail.com
Password*
Password must be 5-40 characters long.
11/20
Repeat Password*
10/40
Show password advice
Security Question *
Mother's maiden name?
This cannot be changed later!
Answer*


You successfully solved a challenge: Repetitive Registration (Follow the DRY principle while registering a user.)
Login
Email*
Password*
Forgot your password?
Log in
Remember me
or

3 - Conclusion

The OWASP Juice Shop assessment reveals a broad spectrum of vulnerabilities ranging from low to critical severity. While some issues present minor risks, several critical and high-severity challenges pose significant threats to the security, privacy, and integrity of your application and data.

Addressing these vulnerabilities is essential to safeguard against potential data breaches, unauthorized access, and compliance violations. Prioritizing remediation efforts on critical and high-severity issues will dramatically improve your security posture and reduce the risk of exploitation.