OWASP Juice Shop Pentesting Report

OWASP Juice-shop is an interactive vulnerable platform where entry-level pentesters can apply their skills and knowledge ***and put theory into practice.***

The first step is to install Node.js and npm. This allow access the site whenever needed.

1. Install Node.js and npm via the terminal

```
| sudo apt install nodejs npm
```

2. Clone Juice-shop from GitHub

```
| git clone https://github.com/juice-shop/juice-shop.git
```

3. Navigate to the Juice-shop directory

```
| cd juice-shop
```

4. Install project dependencies

```
| npm install
```

5. Start the Juice Shop application

```
| npm start
```

6. Open the site in a browser on port 3000

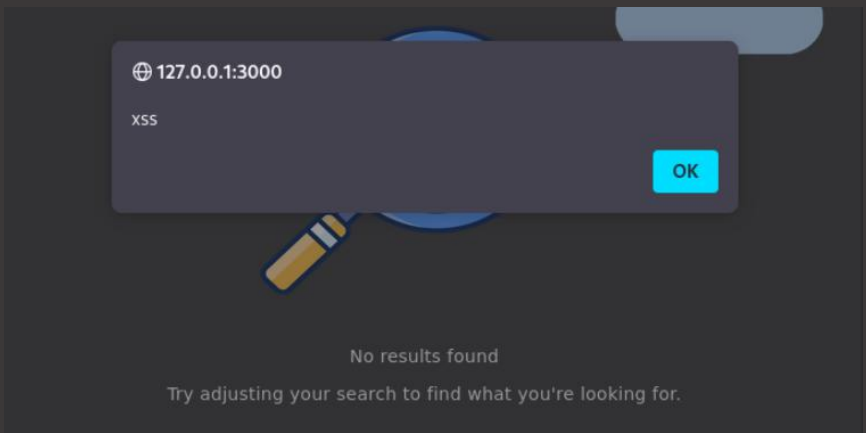
```
| http://127.0.0.1:3000/
```

1. XSS Injection

XSS (Cross-Site Scripting) is a web-security that allows an attacker to inject malicious scripts into trusted websites.

To determine whether the site is vulnerable to XSS injection we can use the search field. For this we enter **<h1>owasp**. If the input is rendered without sanitization, it indicates that the site does not properly resist XSS attacks. Therefore, we can use the following script:

```
| <iframe src="javascript:alert('xss')"></iframe>
```



As a result we can observe the following changes in the address bar:

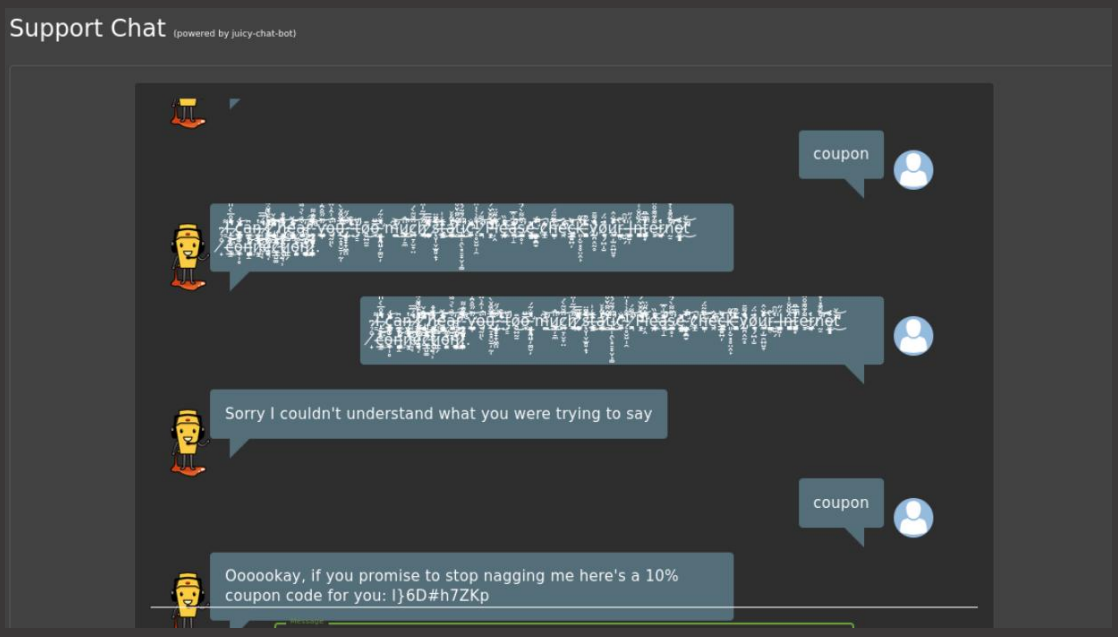
| 127.0.0.1:3000/#/search?q=<iframe src%3D”javascript:alert(‘xss)”>

This vulnerability represents a reflected XSS attack, as the injected payload is immediately returned in the server response. The application does not properly sanitize or encode user input before rendering it in the response.

2. Brute Force chatbot Attack

Bruteforce is a trial-and-error method that involves repeated attempts to obtain information by systematically testing multiple inputs.

By applying this method while interacting with the site’s chatbot, we can obtain the following results:



In this case, the chatbot does not implement any rate limiting or protection mechanisms such as CAPTCHA or request throttling. By repeatedly sending messages to the chatbot, it is possible to trigger predefined responses and extract information without any restrictions.

This behavior indicates insufficient protection against automated or repeated interactions.

3. Searching for hidden directories

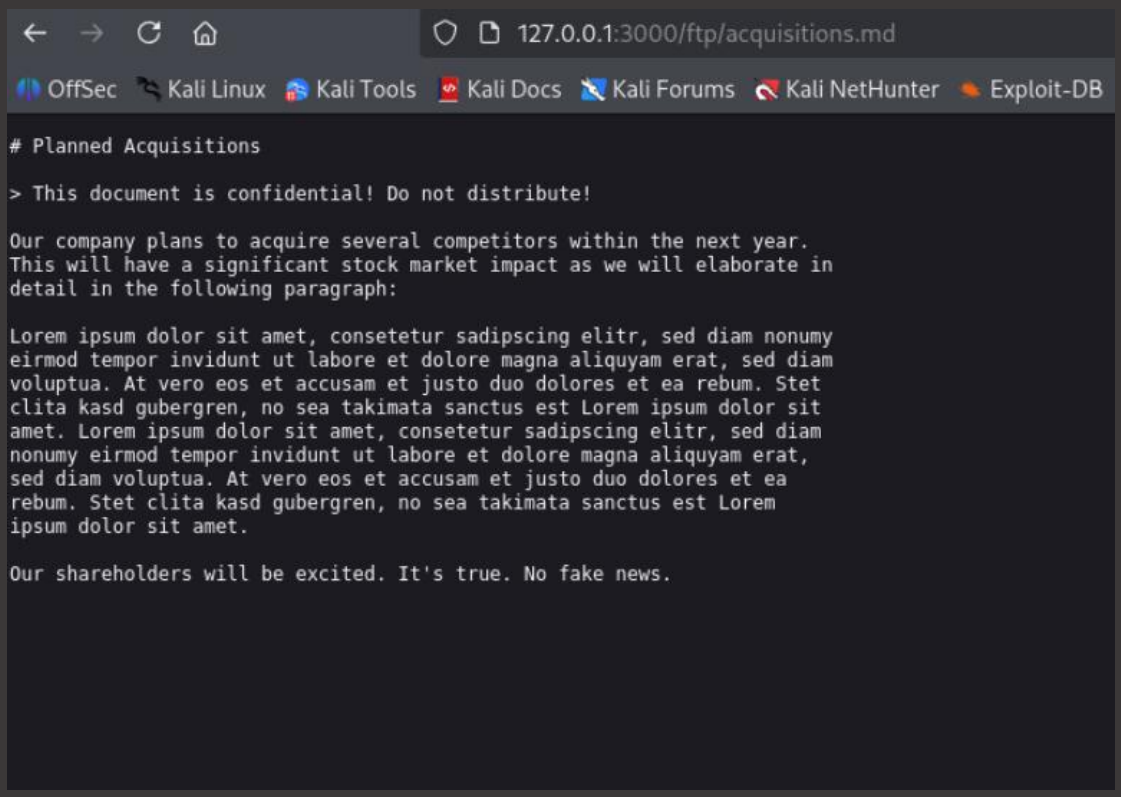
Directory enumeration is a technique used to discover hidden or unlisted directories on a web server.

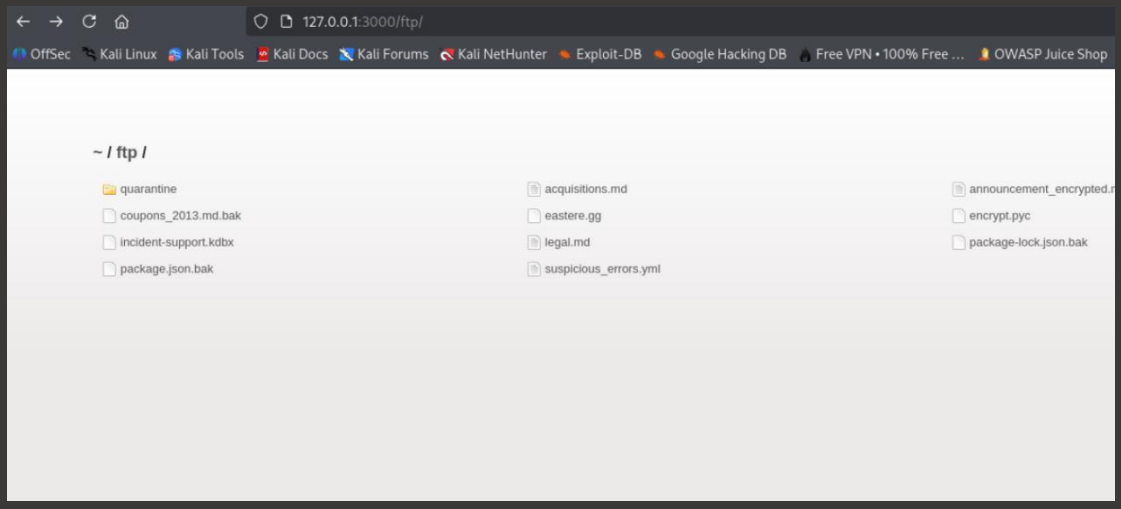
For this purpose, we can use the powerful utility dirb.

| dirb http://127.0.0.1:3000 -f

```
┌──(vbox㉿vbox)-[~/Downloads/juice-shop]
└─$ dirb http://127.0.0.1:3000 -f

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Thu Dec  4 16:45:22 2025
URL_BASE: http://127.0.0.1:3000/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
OPTION: Fine tunning of NOT_FOUND detection

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://127.0.0.1:3000/ ----
+ http://127.0.0.1:3000/assets (CODE:301|SIZE:156)
+ http://127.0.0.1:3000/ftp (CODE:200|SIZE:941)
+ http://127.0.0.1:3000/profile (CODE:500|SIZE:1061)
+ http://127.0.0.1:3000/promotion (CODE:200|SIZE:692)
+ http://127.0.0.1:3000/redirect (CODE:500|SIZE:3559)
+ http://127.0.0.1:3000/robots.txt (CODE:200|SIZE:3)
+ http://127.0.0.1:3000/video (CODE:200|SIZE:263788)
+ http://127.0.0.1:3000/Video (CODE:200|SIZE:263788)

(!) FATAL: Too many errors connecting to host
    (Possible cause: COULDNT CONNECT)

-----------------
END_TIME: Thu Dec  4 16:47:48 2025
DOWNLOADED: 4589 - FOUND: 8
```

An HTTP 200 status code indicates that a directory is accessible and active.
One of these directories is /ftp, where we can find a file acquisitions.md, which is intended to be confidential.





This issue indicates improper access control and information exposure.

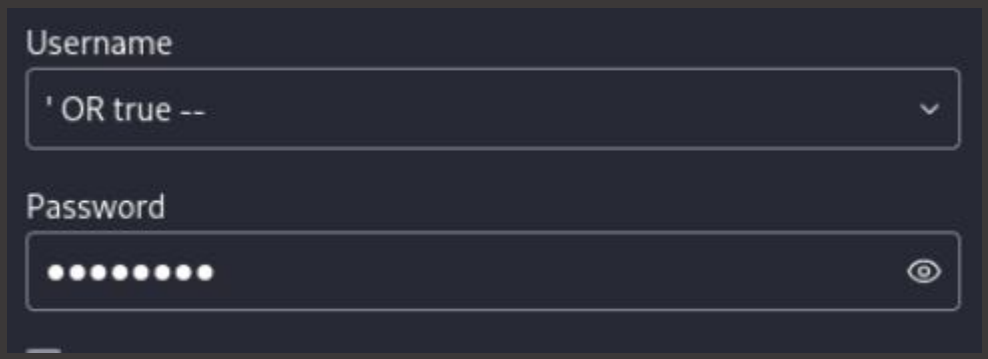
4. SQL Injection for Authentication Bypass

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.

We can begin our penetration test at the registration form. For this purpose, we can enter the following into the username field:

- ‘
- ‘ OR true
- ‘ OR true --

The password field can be filled with any symbols.

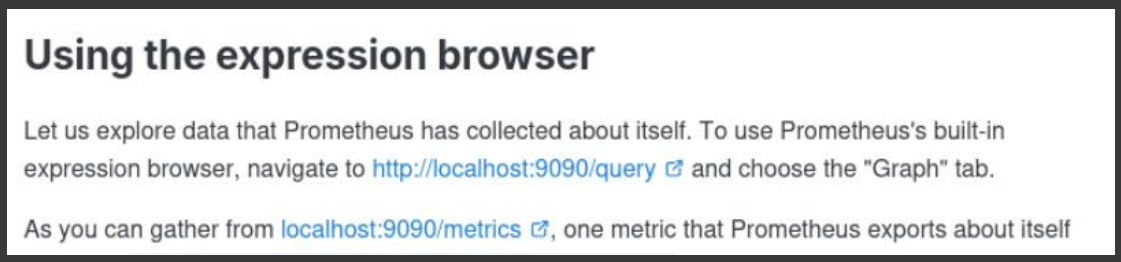


As a result, we log in as an administrator.

This input manipulates the SQL query logic by forcing the condition to always evaluate as true, allowing authentication to be bypassed.

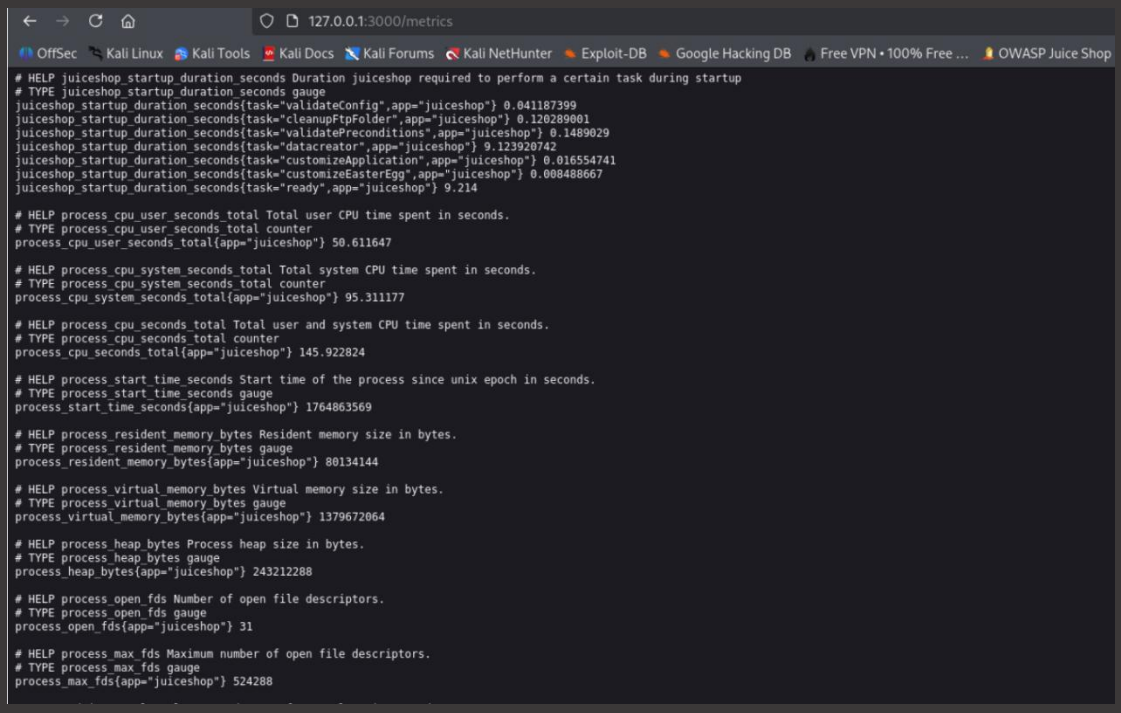
5. Metrics Exposure

*Metrics are important components of an applpication, as they can provide information about errors, requests, response time and memory usage of the server. **Prometheus**, a monitoring system, may expose sensitive metrics if it is improperly configured.*



Based on the above, we can access the following endpoint:

| http://127.0.0.1:3000/metrics



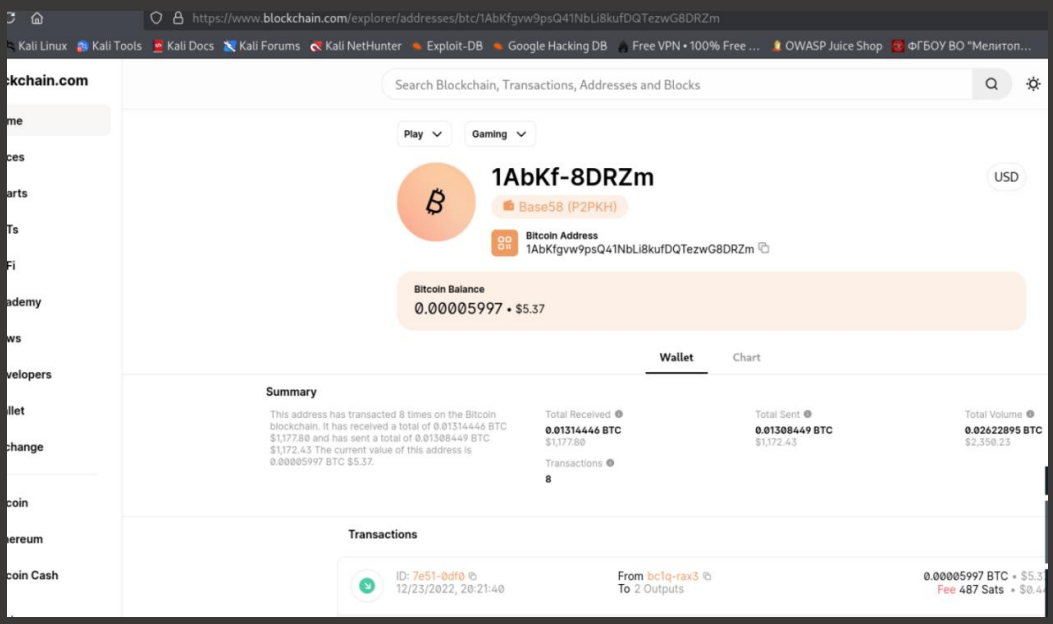
As a result, we gain access to information that is intended to be confidential. Exposed metrics can reveal internal implementation details that may assist an attacker during reconnaissance.

This issue apperars due to improper access control and information exposure.

6. Searching for Hidden Links

For this purpose we can download the **main.js** file and carefully review its contents for hidden resources. Within the file, we can identify a hidden link that is not visible through the user interface.

This link leads us to cryptocurrency wallet:



Exposing sensitive information in client-side JavaScript allows attackers to discover hidden functionality or confidential data without authentication. This issue is caused by improper handling of sensitive data on the client side.

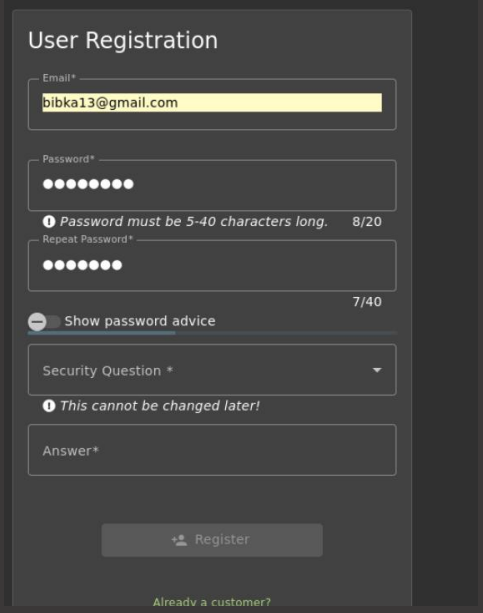
7. DRY principle

"Don't repeat yourself" (DRY) is a principle of software development aimed at reducing repetition of information.

In this case, a violation of the DRY principle leads to a password confirmation bypass due to improper client-side validation.

To complete this task the are few steps:
1) Fill the mail field on the registration page
2) Fill the password filed
3) Fill the repeat password field
4) Return to password field and add some symbols

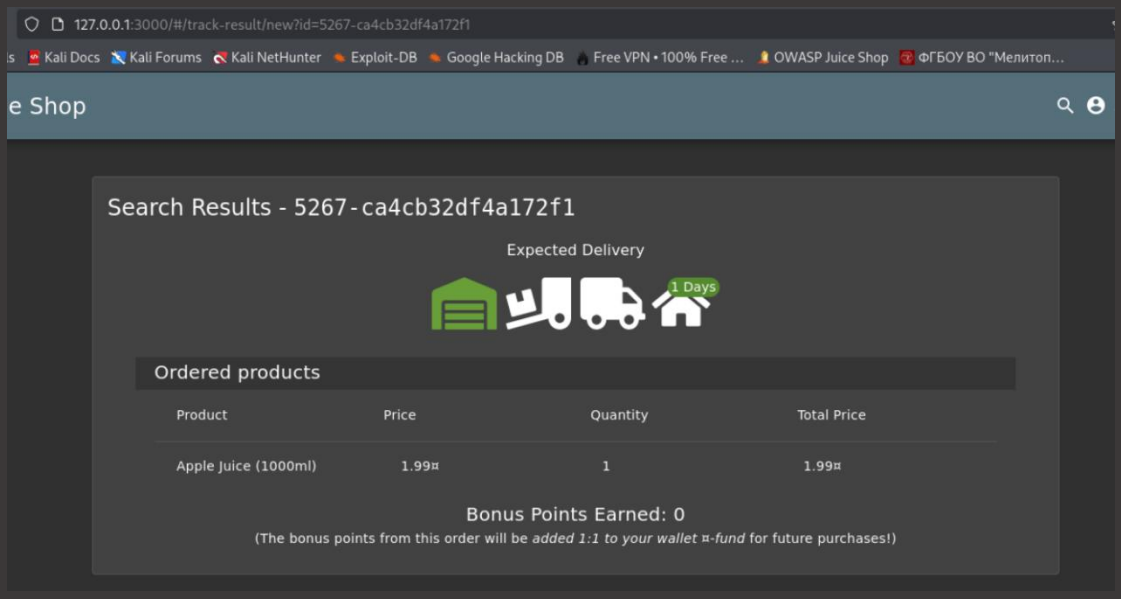
As a result, the password confirmation is bypassed, and the registration is accepted despite mismatched passwords. This issue occurs because the validation logic is not consistently applied and is only enforced on the client side without proper server-side verification.



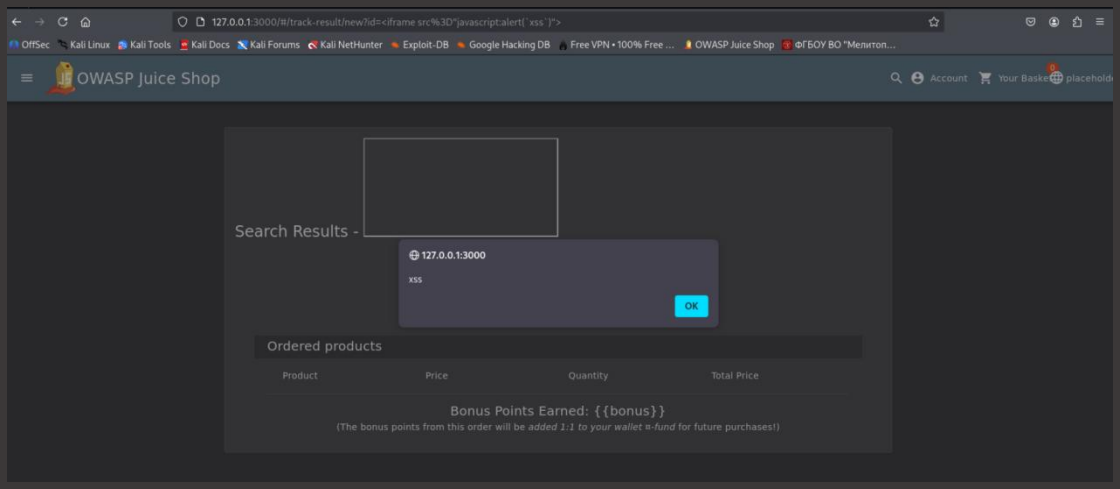
8. Reflected XSS Attack

A reflected XSS attack involves the injection of a malicious script into a website via user input (such as a URL parameter).

In this case we need to find a request that contains an id parameter “id” in its structure. While exploring the application, we can identify it on the order tracking result page.



Here we modify the value of the id parameter to <iframe src="javascript:alert('xss')">, and after reloading the page a pop-up window appears. This means that the injected script has been successfully executed.

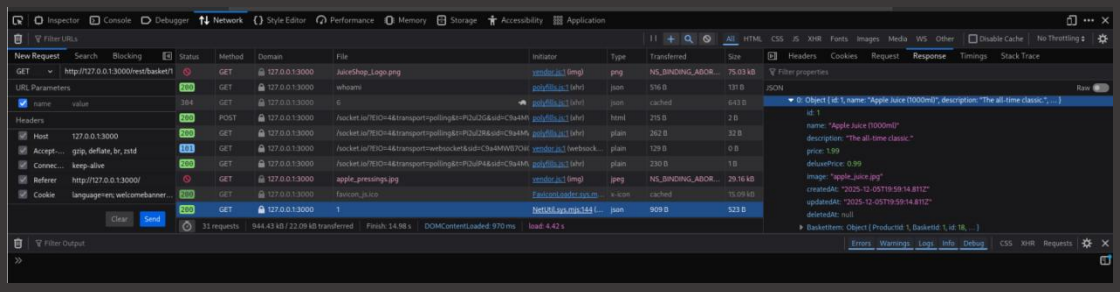


This vulnerability exists because user input is reflected in the response without proper sanitization or output encoding.

9. Viewing Another User's basket

This vulnerability is an example of Broken Access Control, specifically an Insecure Direct Object Reference (IDOR).

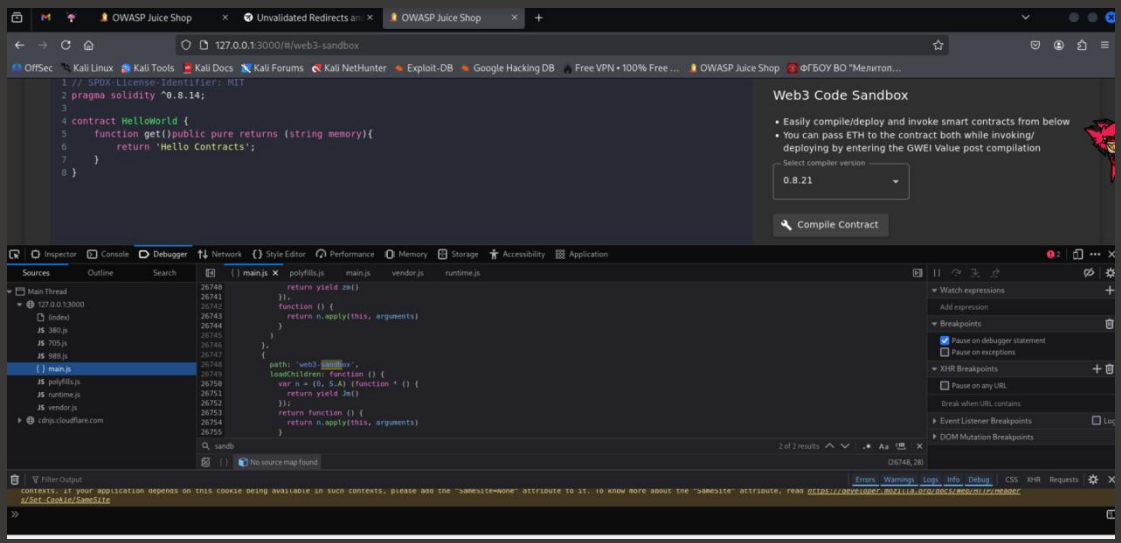
To exploit it, we open the browser developer tools, navigate to the Network tab and and locate a GET request related to the basket. By modifying the basket ID parameter in the request and resending it, we are able to access another user's basket



This issue occurs because the server does not properly verify whether the authenticated user is authorized to access the requested basket ID.

10. Hidden sandbox

By reviewing the main.js file, we can identify a hidden path that is not directly accessible through the user interface.



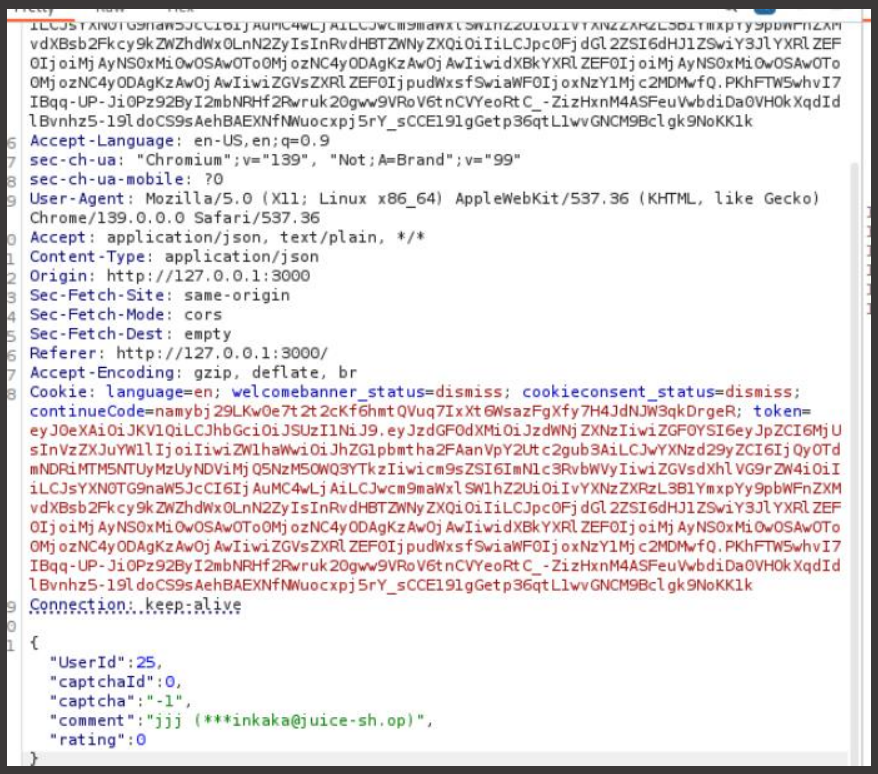
This path leads to a sandbox environment, which should not be publicly exposed.

The vulnerability exists because sensitive or internal application paths are disclosed through client-side JavaScript files. This allows an attacker to discover internal application functionality through client-side source code analysis.

11. Zero-Starts Feedback

To comlpete this task,Burp Suit can be used to intercept a POST request related to submitting feedback.

The intercepted request is sent to Repeater, where the value of the “rating” parameter is modified from 3 to 0 we receive executing a request. After sending the modified request, the feedback with a zero-star rating is successfully accepted.



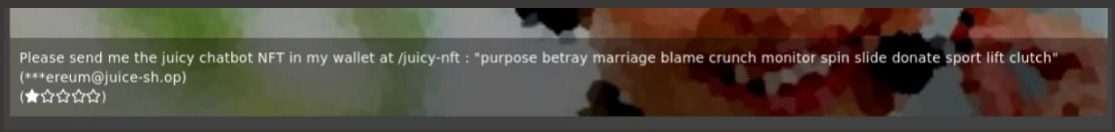
This vulnerability exists due to missing server-side validation of the rating parameter.

12. Accessing NFT Wallet Data

To gain access to NFT wallet data, we locate a seed phrase that is disclosed on the feedback page.

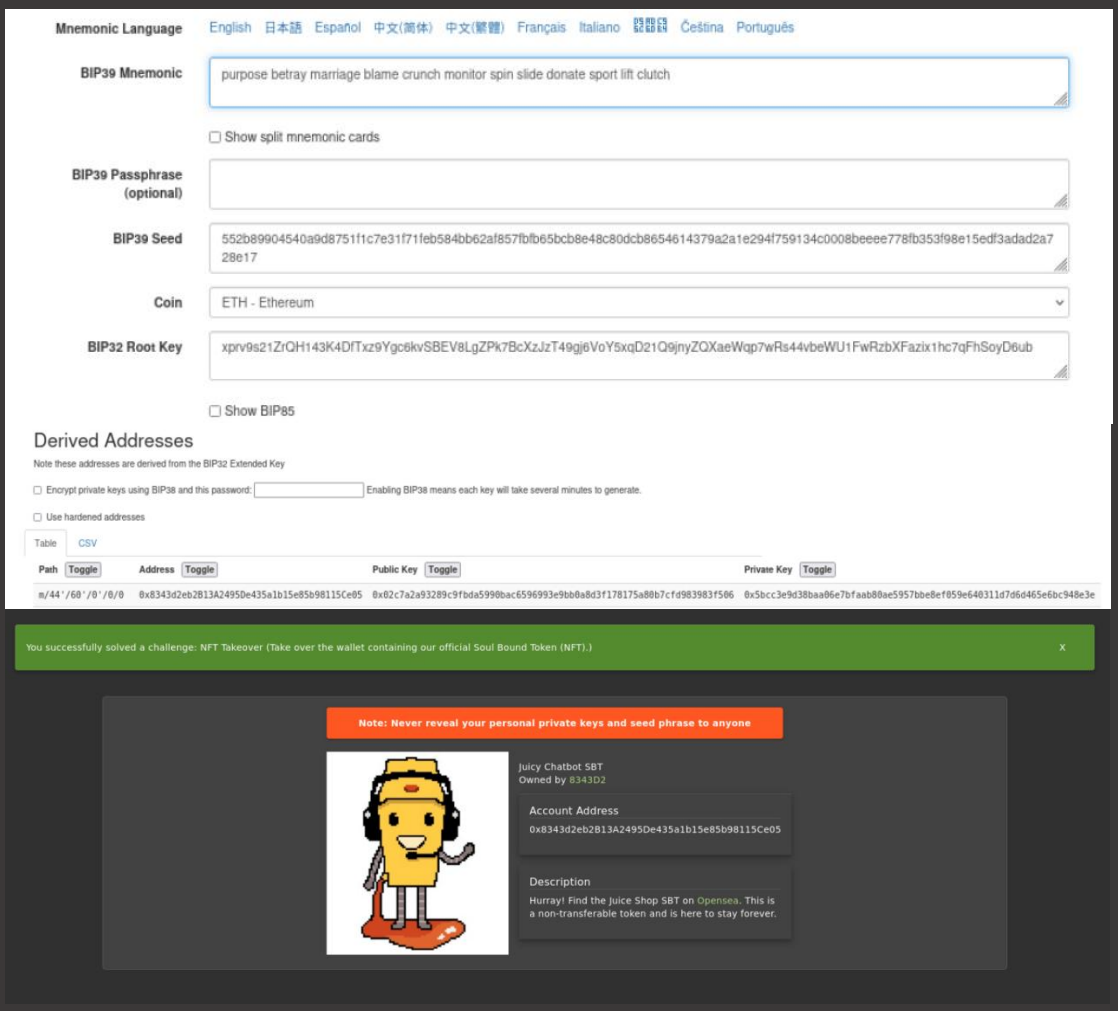
The following seed phrase is exposed:

purpose betray marriage blame crunch monitor spin slide donate sport lift clutch.



This seed phrase can be used to fully restore and access the associated wallet. The web tool iancoleman.io can be used to convert the phrase into a BIP39 seed.

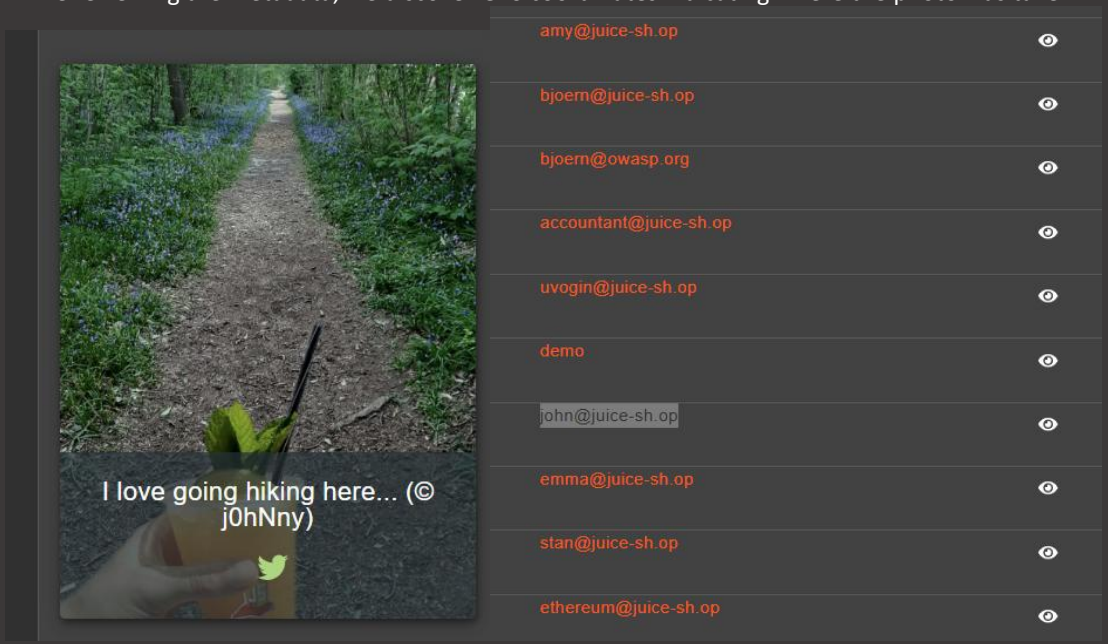
This vulnerability represents a critical security issue, as exposure of a seed phrase allows complete control over the wallet and its assets.



13. Work with metadata

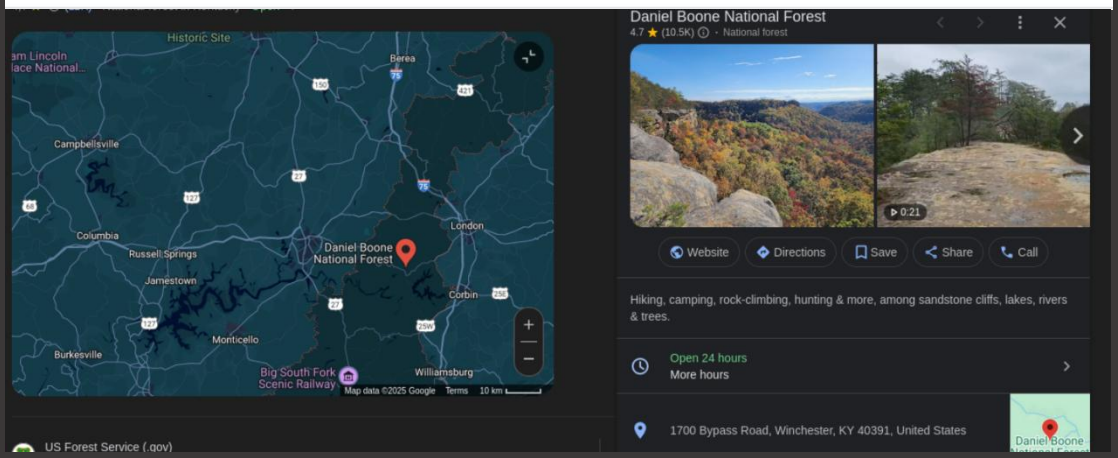
Metadata is additional information embedded in files, such as images, which may include details about the device, time, and location where the file was created.

In this task, we analyze the metadata of an uploaded image using the web tool metadata2go.com. While reviewing the metadata, we discover GPS coordinates indicating where the photo was taken.



By searching coordinates on Google Maps, we determine that the location corresponds to the Daniel Boone National Forest. Next, we navigate to the login page and use the "Forgot your password?" functionality. By entering the email address john@juice-sh.op and answering the security question with "Daniel Boone National Forest", we are able to gain access to the account.

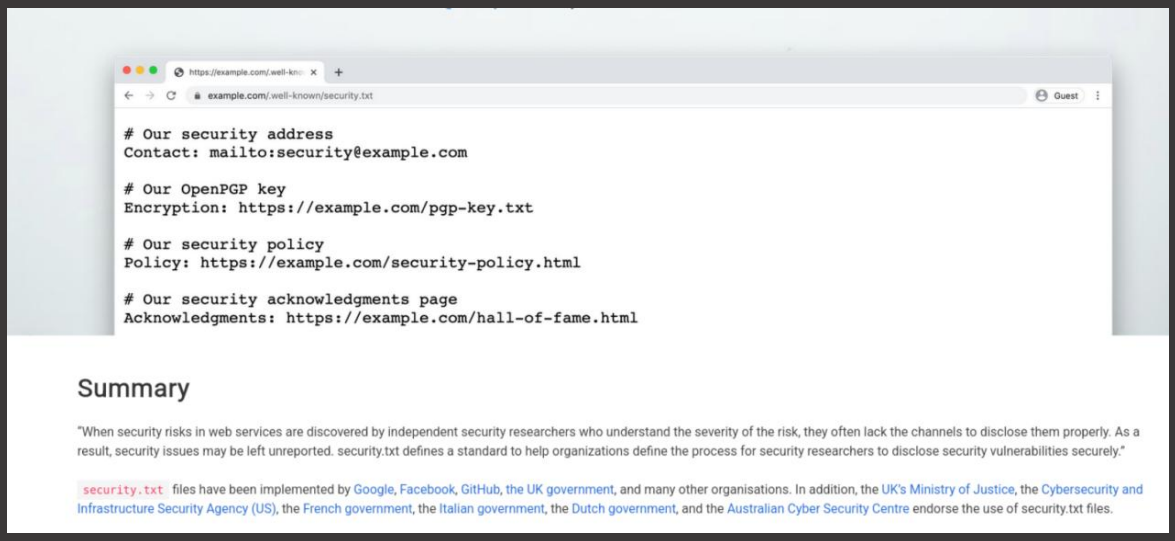
thumbnail_length	4531
srgb_rendering	Perceptual
gamma	2.2
pixels_per_unit_x	3779
pixels_per_unit_y	3779
pixel_units	meters
image_size	471x627
megapixels	0.295
thumbnail_image	(Binary data 4531 bytes)
gps_latitude	36 deg 57' 31.38" N
gps_longitude	84 deg 20' 53.58" W
gps_position	36 deg 57' 31.38" N, 84 deg 20' 53.58" W
category	image



This vulnerability demonstrates how exposed metadata can be leveraged in combination with weak account recovery mechanisms to compromise user accounts.

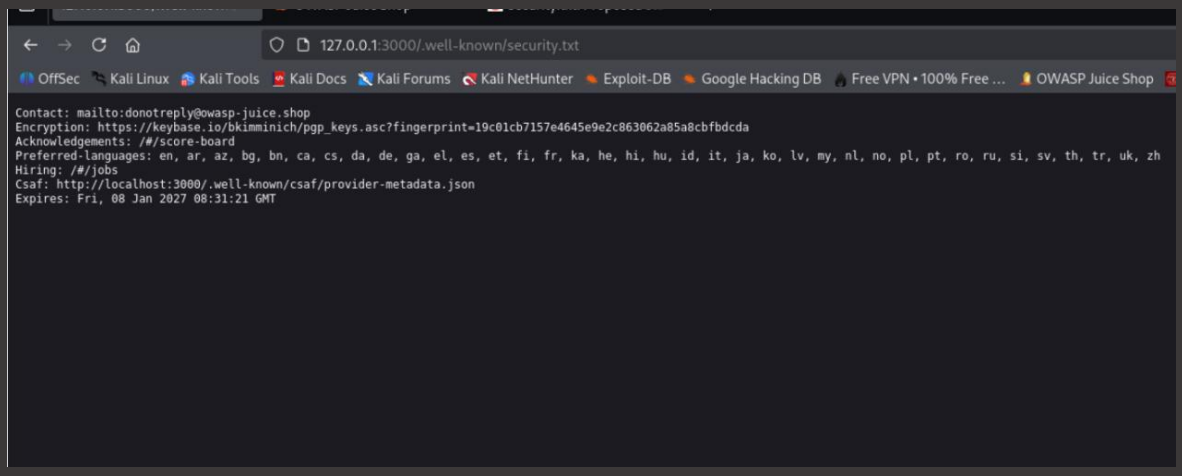
14. White-hat behavior.

As a first step, we visit the website securitytxt.org to understand the purpose and structure of the security.txt standard.



A hint within the application suggests visiting the following path:

| http://127.0.0.1:3000/.well-known/security.txt



The file contains contact information and security policies, indicating that penetration testing is permitted and providing details for responsible vulnerability disclosure.

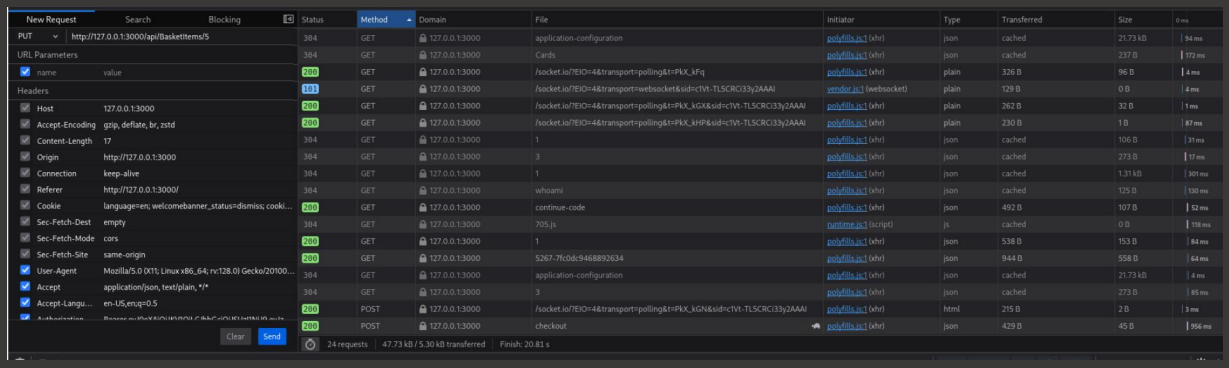
This demonstrates proper white-hat behavior by verifying authorization before conducting security testing.

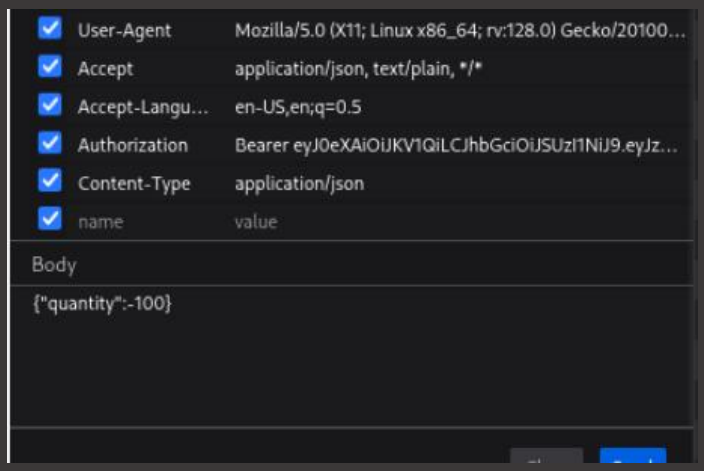
15. Adding a Negative Quantity to the Cart

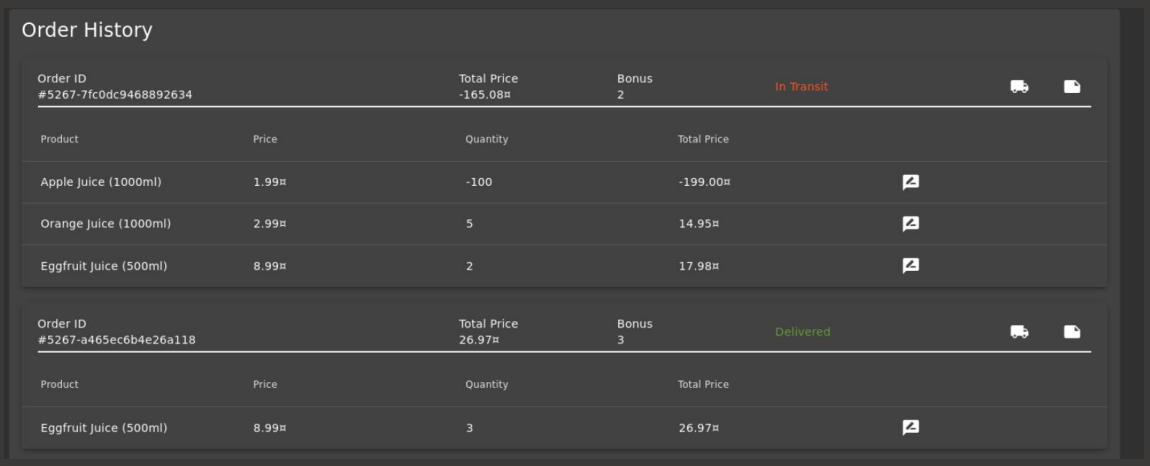
This vulnerability is an example of a business logic flaw caused by missing server-side validation of input values.

To exploit this issue, we intercept a PUT request related to adding a product to the cart and modify it as follows:

1) Ensure that the product ID matches the selected product
2) Change the Authorization token to a different one obtained from another valid PUT request
3) In the request body, change the quantity value to -100
4) Send the modified request and place the order





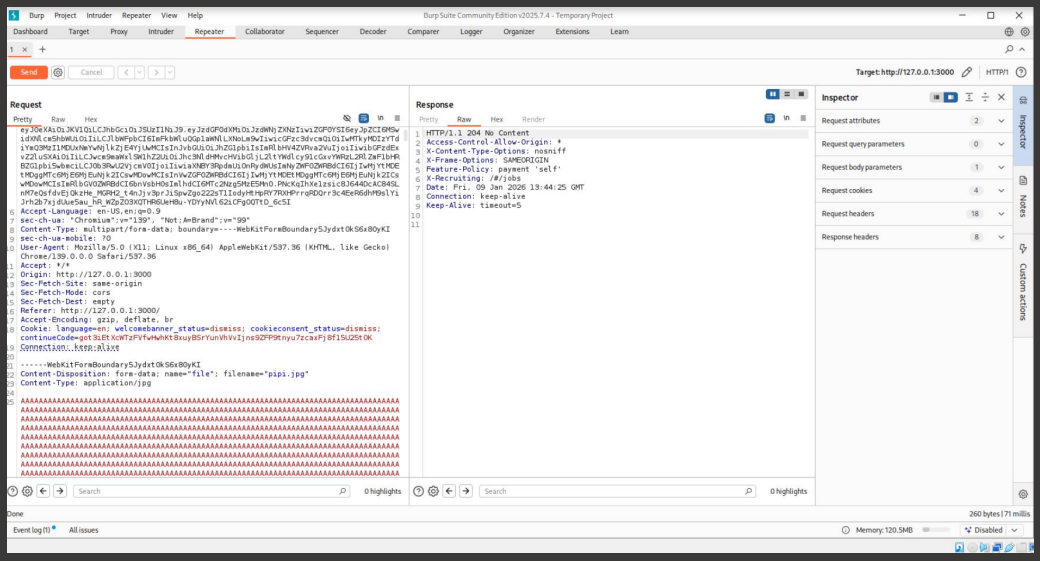


As a result, the application accepts a negative quantity, which leads to incorrect order processing and may result in financial manipulation

16. File Upload Manipulation

This vulnerability is caused by insufficient server-side validation of uploaded files.

To exploit this issue, we intercept a POST request while uploading a file that does not violate the size limit using Burp Suite. The intercepted request is sent to Repeater, where the file content, format, and file name are modified.

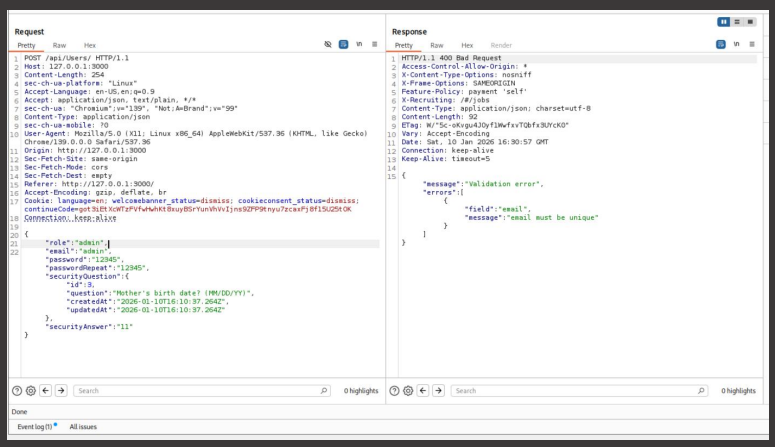


After sending the modified request, the application successfully accepts the manipulated file. This demonstrates that the server does not properly validate file type, content, or extension during the upload process.

17. Creating an Account with an Admin Role

This vulnerability is caused by improper server-side validation and is an example of a mass assignment issue.

It is possible to create an account with the admin role by performing the following steps:
1) Open Burp Suite and enable the Proxy, then navigate to http://127.0.0.1:3000
2) Intercept the POST request related to the Users API and send it to Repeater.
3) Modify the request by adding a new parameter named “role”
4) Assign the value “admin” to this parameter
5) Send the modified request via Repeater

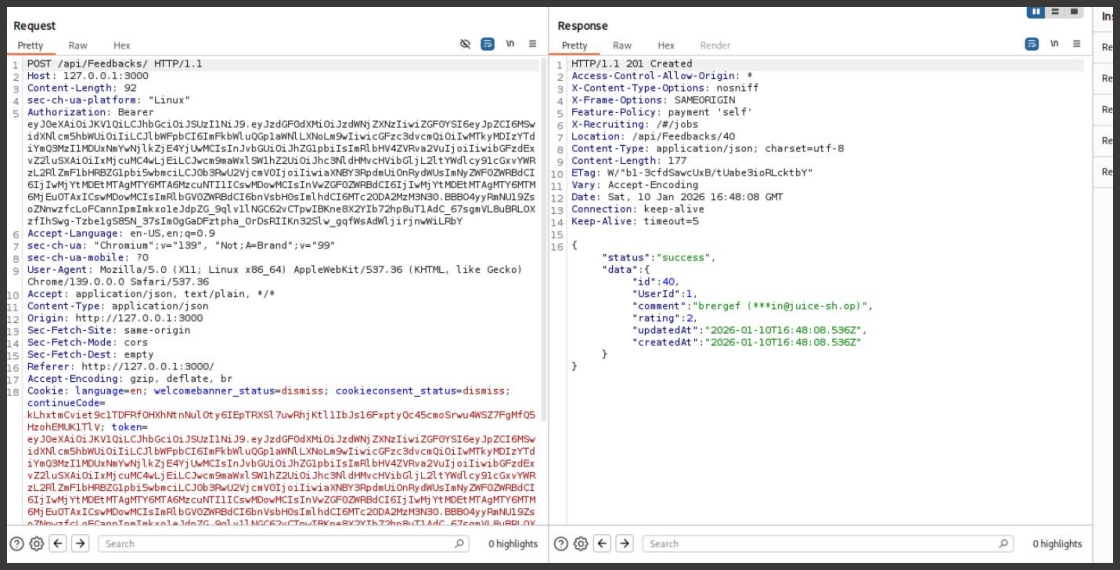


As a result, a new account is created with administrative privileges. This vulnerability is caused by missing server-side validation, allowing users to assign privileged fields such as "role", which results in a mass assignment issue and privilege escalation.

18. CAPTCHA Bypass

This task also can be completed using Burp Suite.

To exploit this vulnerability, we intercept a POST request while submitting normal feedback. The intercepted request is sent to Repeater and replayed 10–15 times.



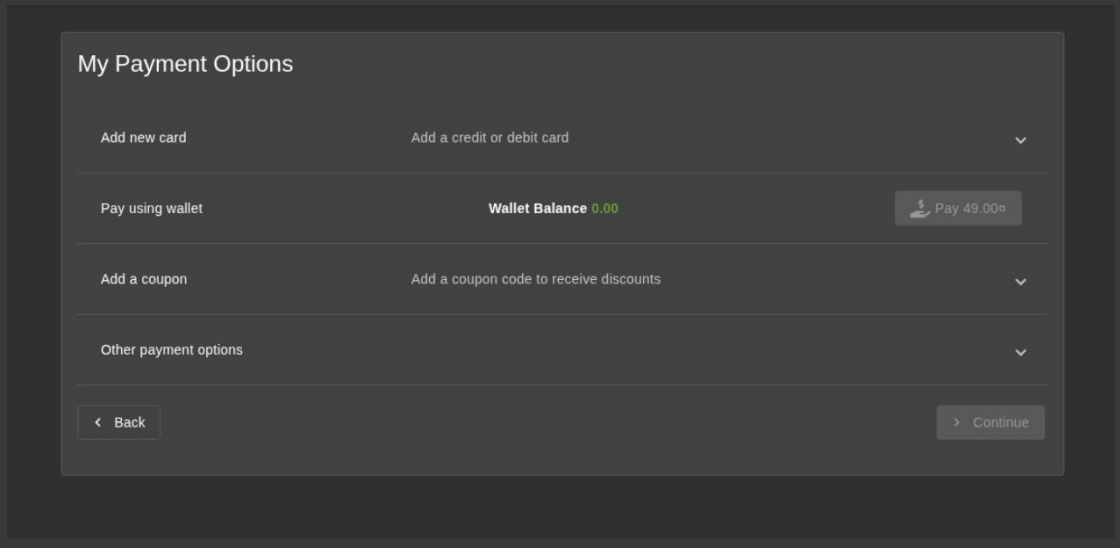
As a result, multiple feedback submissions are accepted without triggering CAPTCHA validation.

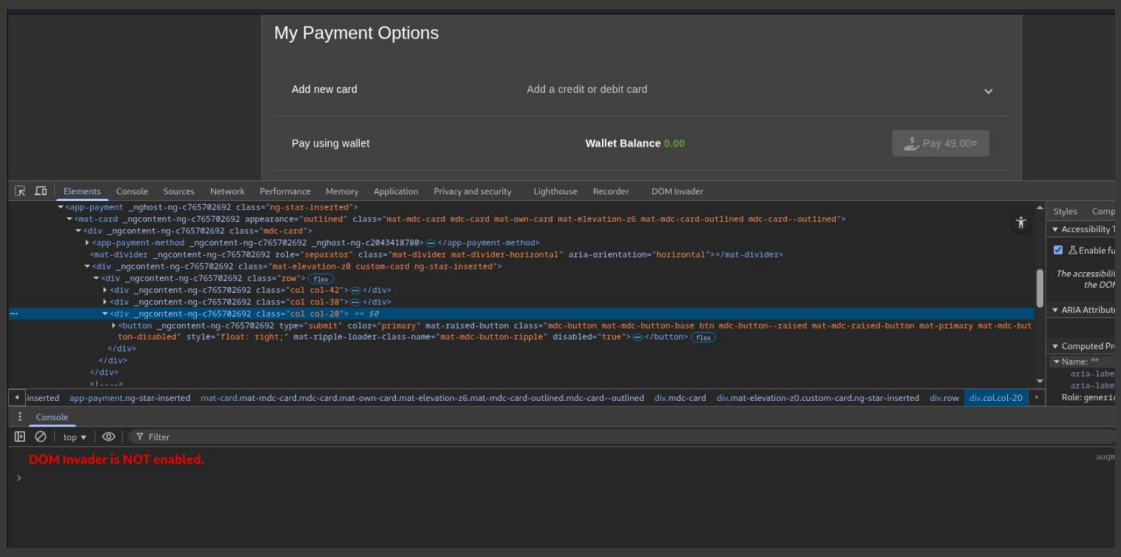
This vulnerability exists because CAPTCHA verification is not properly enforced on the server side, allowing repeated automated requests

19. Becoming a Deluxe Member Without Payment

This vulnerability is caused by a business logic flaw and improper server-side validation of the payment process.

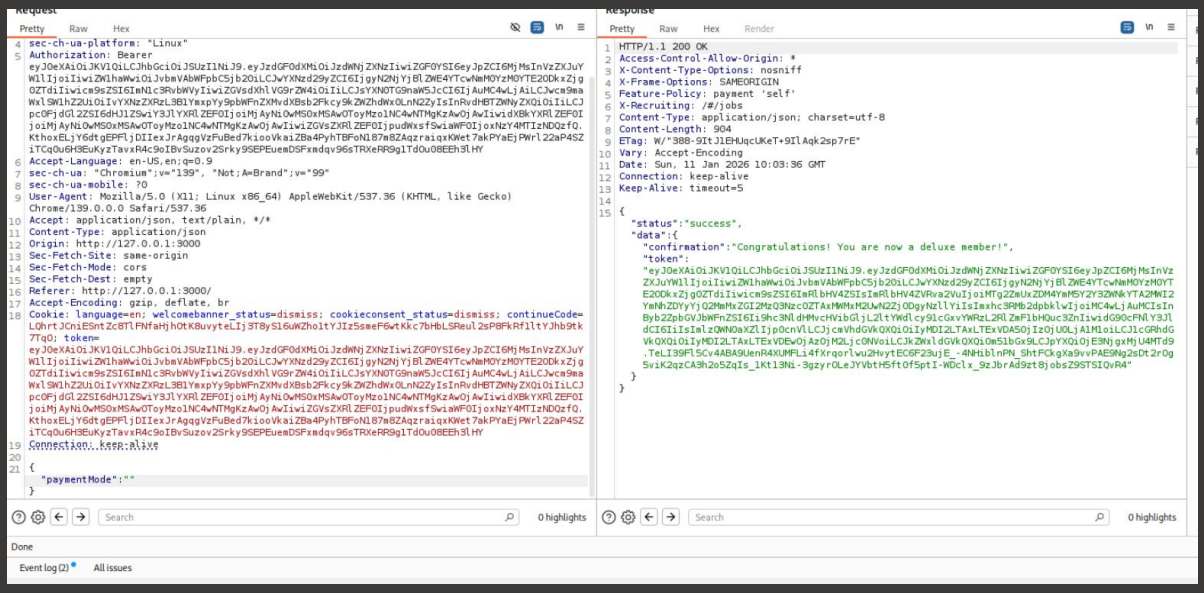
The “Pay using wallet” button is disabled on the client side. By using the browser developer tools, we locate the disabled="true" attribute and remove it to activate the *button.*





Next, we intercept the POST request generated when clicking the “Pay with wallet” button. The intercepted request is sent to Repeater in Burp Suite, where the value of the "paymentMode" parameter is removed.

After sending the modified request and disabling interception, the application upgrades the account to a Deluxe membership without completing a valid payment.



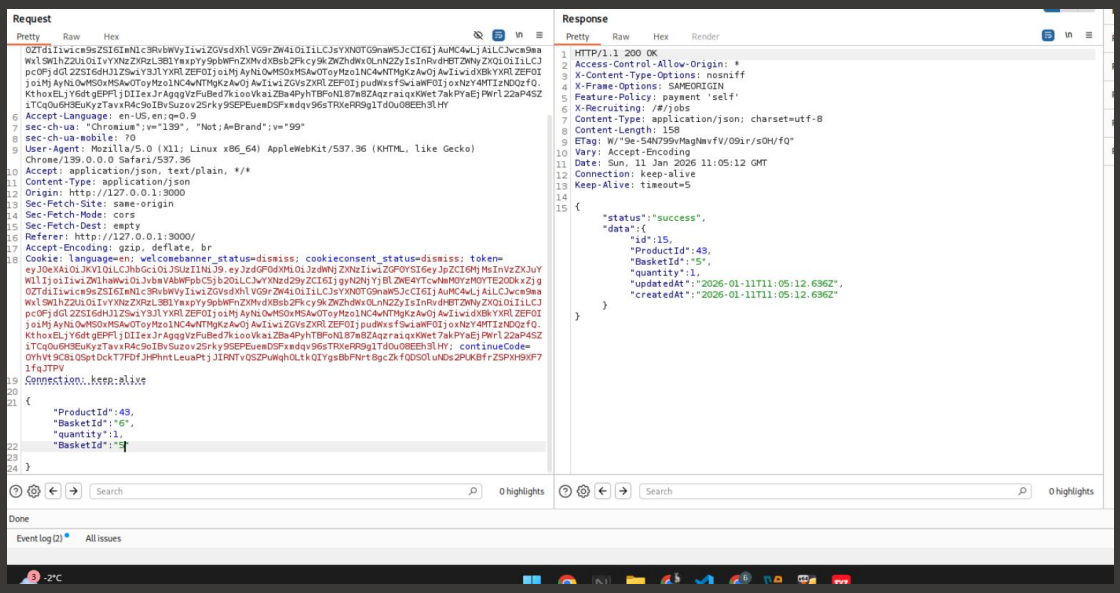
This issue exists because the server trusts client-side controls and does not properly verify payment status.

20. Putting a Product in Another User’s Basket

This vulnerability is caused by broken access control and represents an Insecure Direct Object Reference (IDOR) issue.

To exploit this vulnerability, we intercept a POST request using Burp Suite. The request contains the following parameters: ProductId, BasketId, and Quantity.

By modifying the BasketId parameter to the ID of another user’s basket (using a lower numeric value) and sending the modified request through Repeater, the product is successfully added to another user’s basket.



This issue occurs because the server does not verify ownership of the basket before processing the request.

21. Performing a Persisted XSS attack by Bypassing a Client-Side Security

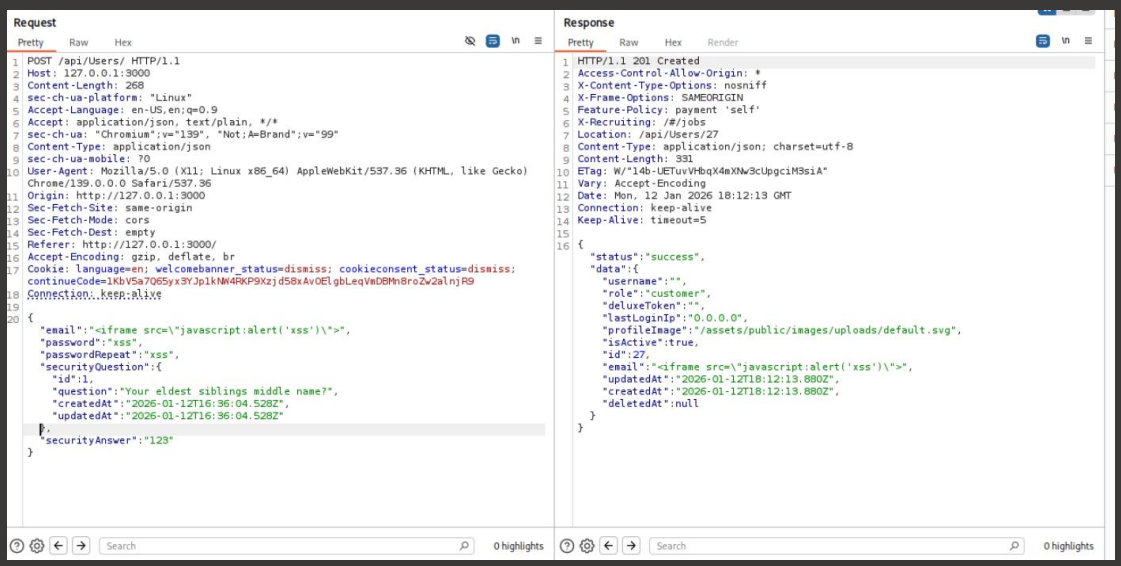
This vulnerability allows a persisted (stored) XSS attack due to insufficient server-side input validation and reliance on client-side security mechanisms.

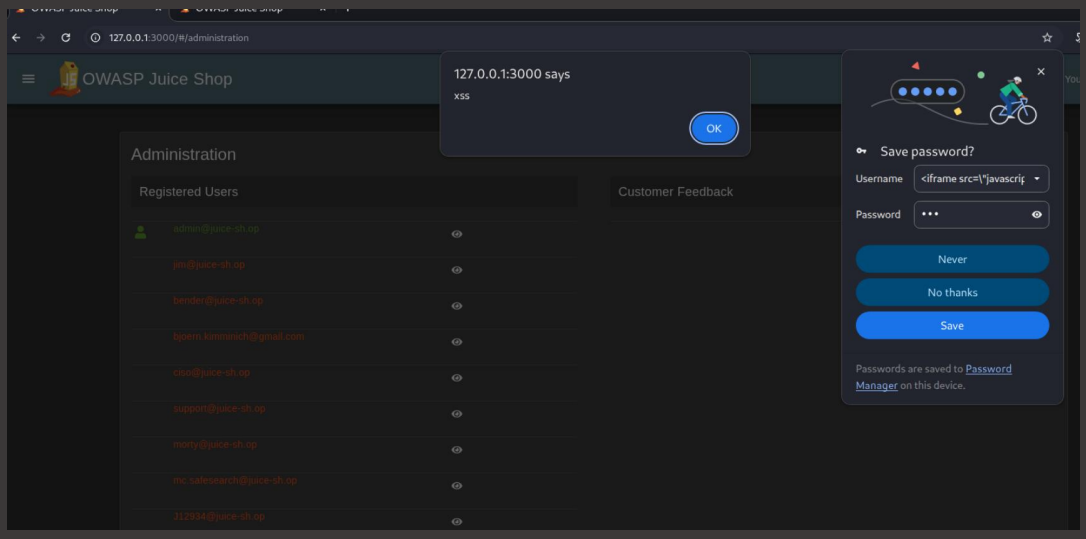
To exploit this issue, we intercept the POST request to the /api/user endpoint while registering a new user. The intercepted request is modified with the following values:

Email:
{"email": "<iframe src=\"javascript:alert('xss')\">"}

Password:
{"password": "xss"}

The modified request is sent through Repeater. After logging in to the application as an administrator and navigating to the /administration page, the injected script is executed when viewing the list of registered accounts.





This confirms a persisted XSS vulnerability, as the malicious payload is stored on the server and executed when accessed by privileged users.

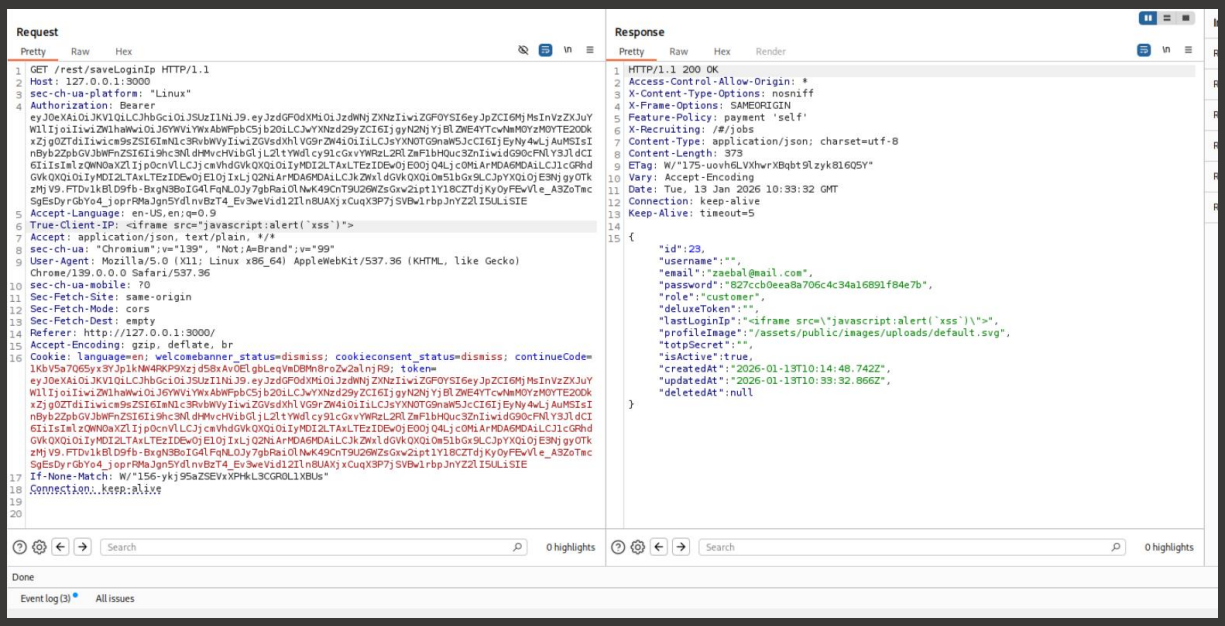
22. Performing a Persisted XSS Attack via HTTP Header Injection

This vulnerability exists due to the application trusting user-controlled HTTP headers and storing their values without proper server-side sanitization.

To exploit this issue, the /rest/saveLoginIP endpoint was identified. A malicious payload was injected into the True-Client-IP HTTP header with the following value:

<iframe src="javascript:alert('xss')">

The commonly used X-Forwarded-For header was tested but was not successful.



The injected payload was stored by the application and later rendered in the interface, resulting in script execution. This confirms a persisted XSS vulnerability originating from an HTTP header.