OWASP Juice Shop Exploitation Report

Write-Ups and Security Vulnerability Exploits

Submitted by:

Mohamed Ehab Zaghloul - Student ID: 120220090

Amr Khaled Mohamed - Student ID: 120220236

Abdelrahman Ashraf - Student ID: 120220292

Mohamed khaled yahya - Student ID: 120220081

Omar Khalifa Abdelmonem - Student ID: 120220201

Faculty of Engineering (FOE)

Computer Science and Engineering Department (CSE)

Egypt-Japan University of Science and Technology (E-JUST)

Contents

1 Exploitation 1 (Authentication bypass)

(Mohamed Ehab Zaghloul - 120220090)

1.1 Brief explanation for key concepts:

- Authentication is the process of verifying a user's identity confirming that someone is who they claim to be.
- SQL Injection is a type of web security vulnerability that allows an attacker to interfere with the SQL queries that an application sends to its database.
- JWT stands for JSON Web Token. It's a compact and secure way to transmit data between two parties — most often used for authentication and authorization in web applications.
- Burp Suite is a powerful web application security testing tool, it helps you find and exploit vulnerabilities in websites and web apps.

1.2 Steps for gaining access as the Admin user:

1. First you should enter any email and password as a form of a test or a trial.
2. Then this request will be captured on BurpSuite for further modification.

[H]

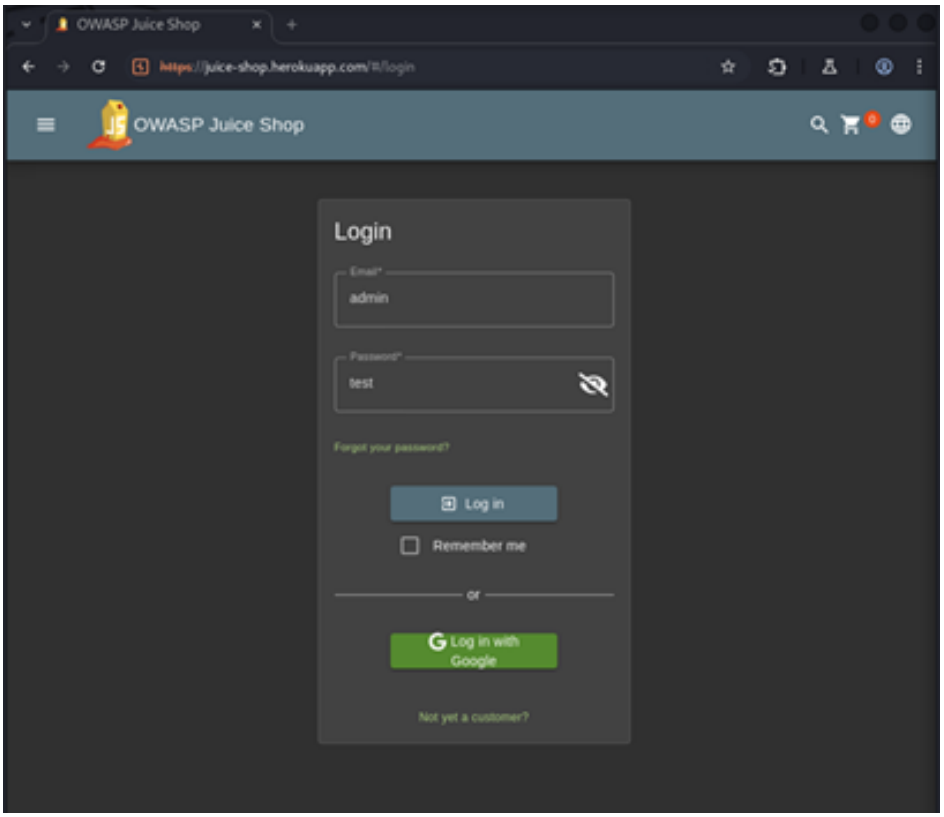

Figure 1: Enter Caption

Here are the captured requests, then we need to forward all of them especially the POST request.

POST request is the request that will be manipulated to gain access to the admin user.

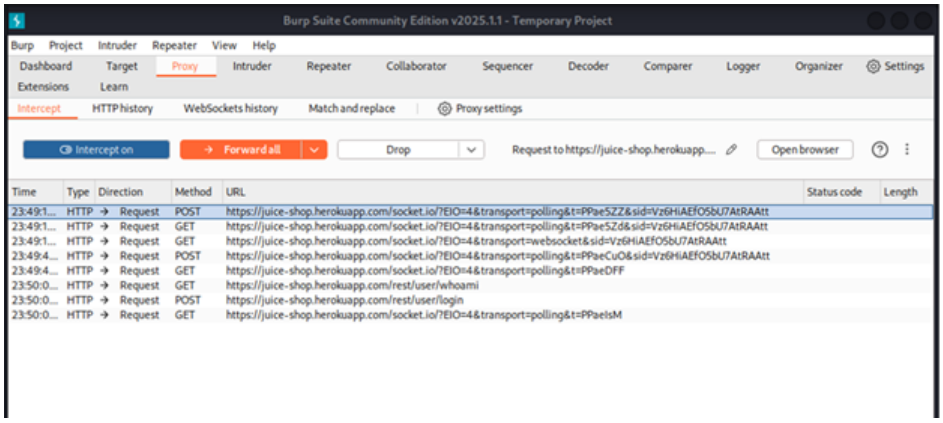

Figure 2: Enter Caption

This POST request will be sent to the repeater where we will manipulate and change a bit in the header to access the user.

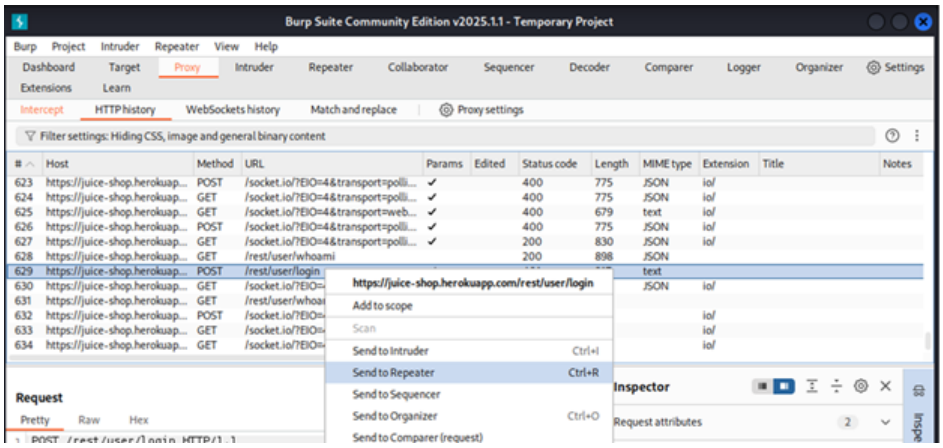

Figure 3: Enter Caption

Here we are at the intruder where we will edit the email part with SQL injection to have a response with a token.

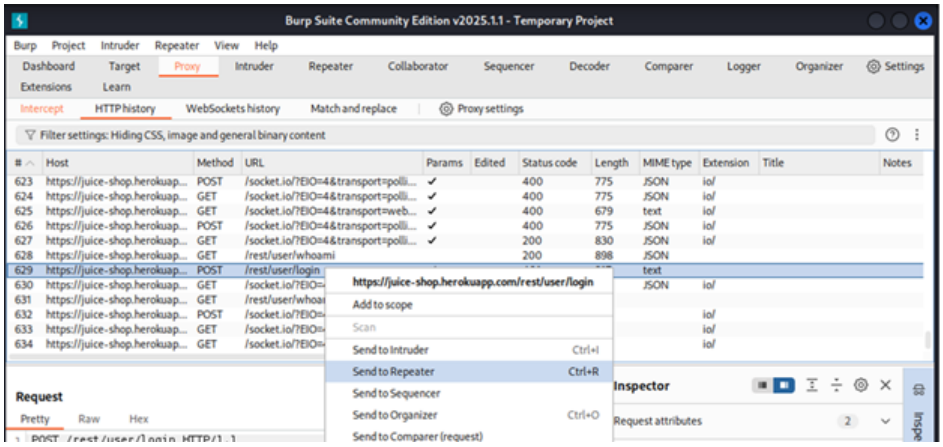

Figure 4: Enter Caption

`"email":"admin' or 1=1 --"` ⇒ That is the part that is modified in the header with SQL injection.

1.3 Here is a breakdown for what is used here:

- `'` : This apostrophe is used for closing the string
- `or 1=1` : Always true, so it bypasses checks
- `--` : Comments out the rest of the SQL (including the password check)

After sending the request with these modification, the response came with the valid token for the log in.

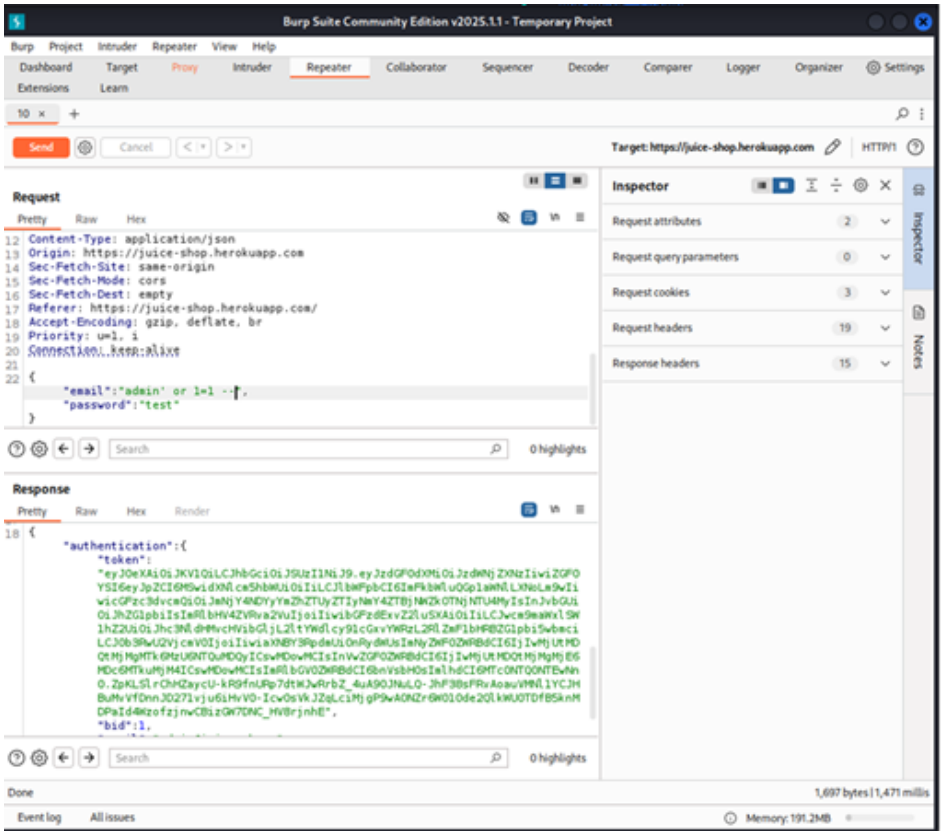

Figure 5: Enter Caption

In the following picture, in the console of the DevTools, we can login using the authentication Token.

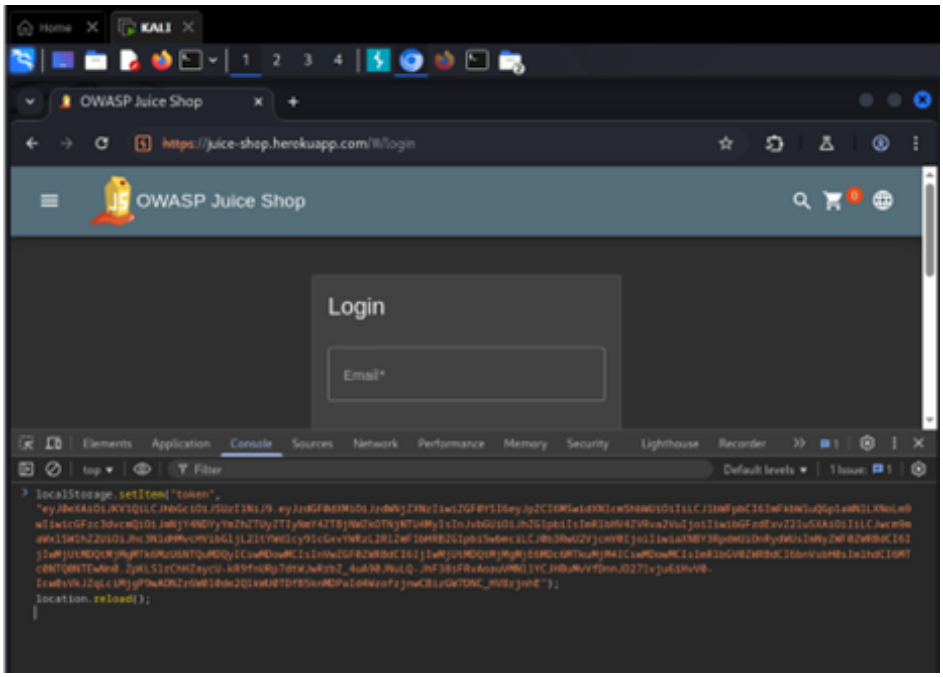

Figure 6: Enter Caption

2 Exploitation 2 (Authentication bypass)

(Mohamed Ehab Zaghloul - 120220090)

- After opening burp suite our packet capturing tool for applying modifications on the headers, I managed to get the domain of juice shop from the highlighted packet.
- Generally almost all email addresses starts with the name of the user and then @ the domain.
- So that is what we will try to do to log in with Bender user.

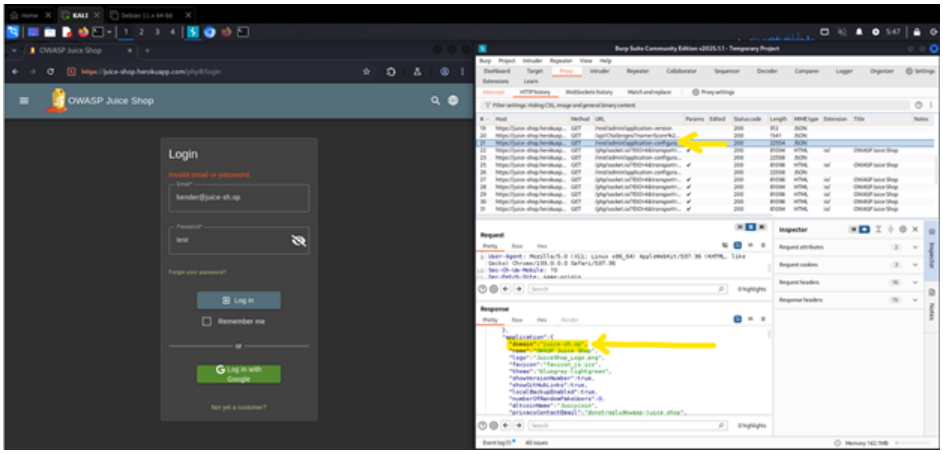

Figure 7: Enter Caption

- After trying this out and used the email as `bender@juice-sh.op`, and typed any password we got an invalid email or password error.
- The previous step was only done to get the header that contains the POST request with `bender@juice-sh.op` and any password.

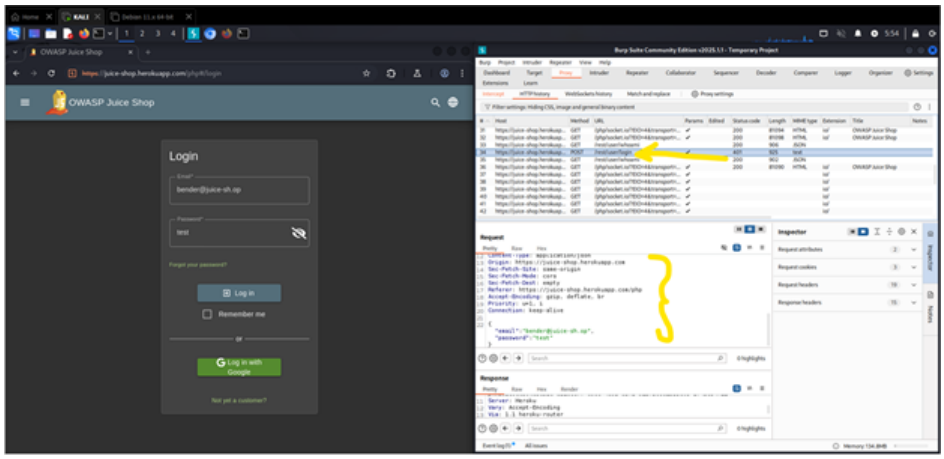

Figure 8: Enter Caption

- After getting the header with the POST request we will try a SQL injection attack on the email part in a trial to log in successfully into the user account.
- We will use a tool called SQLmap for testing SQL vulnerabilities from our Linux Terminal.

2.1 SQLmap

- SQLmap is used for auto detecting and exploiting SQL vulnerabilities
- Command used is `sqlmap -r <file.txt contain POST request header>`
- Then in the header we add an `*` (asterisk in the part we want to add our SQL injections)

- In the following screenshot we created a .txt file with our header using nano.

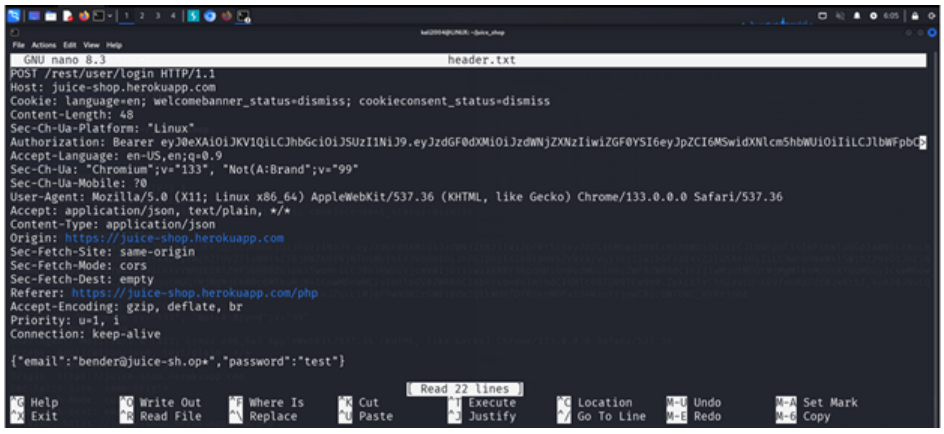

Figure 9: Enter Caption

- Then we will start our attack using SQLmap.

Figure 10: Enter Caption

- In the following screenshot we got an error as the POST request when carried out with an invalid email or password, we get a 401 status code, and this needs to be modified in the command before starting the attack.

Figure 11: Enter Caption

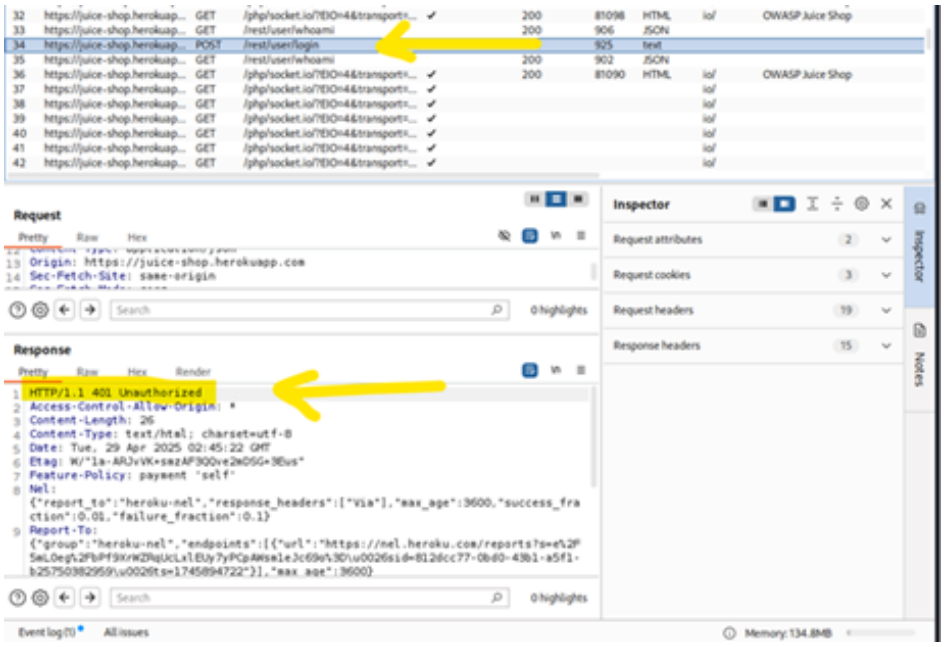

Figure 12: Enter Caption

- After running the command with modified part and ignoring the 401-status code, it was manageable to get the correct credentials and log in successfully.

Figure 13: Enter Caption

3 Exploitation 3 (MC safesearch)

(Mohamed Ehab Zaghloul - 120220090)

3.1 Description:

Briefly here we have a username that is available for a user on juice shop and the target for us, is to find the password.

So, by making a little search about the username MC safesearch, I found a video about a person called Mr. Noodles and the video's subject is about passwords.

At 00:33 second he said the following:

> "But then how do you remember is the question that I get
> I say why not use the first name of your favorite pet?
> Mine's my dog, Mr. Noodles. It don't matter if you know
> because I was tricky and replaced some vowels with zeroes."

So, I deduced that what was meant here is that he changed the vowel 'o' with zeros to be `Mr.N0Odles`, and this is the password that we will give it a try.

- First thing to be done is that we need to get the username of the user which is Mc safesearch, but we need the complete domain.
- To get the domain we can find it in the following header in Burp.

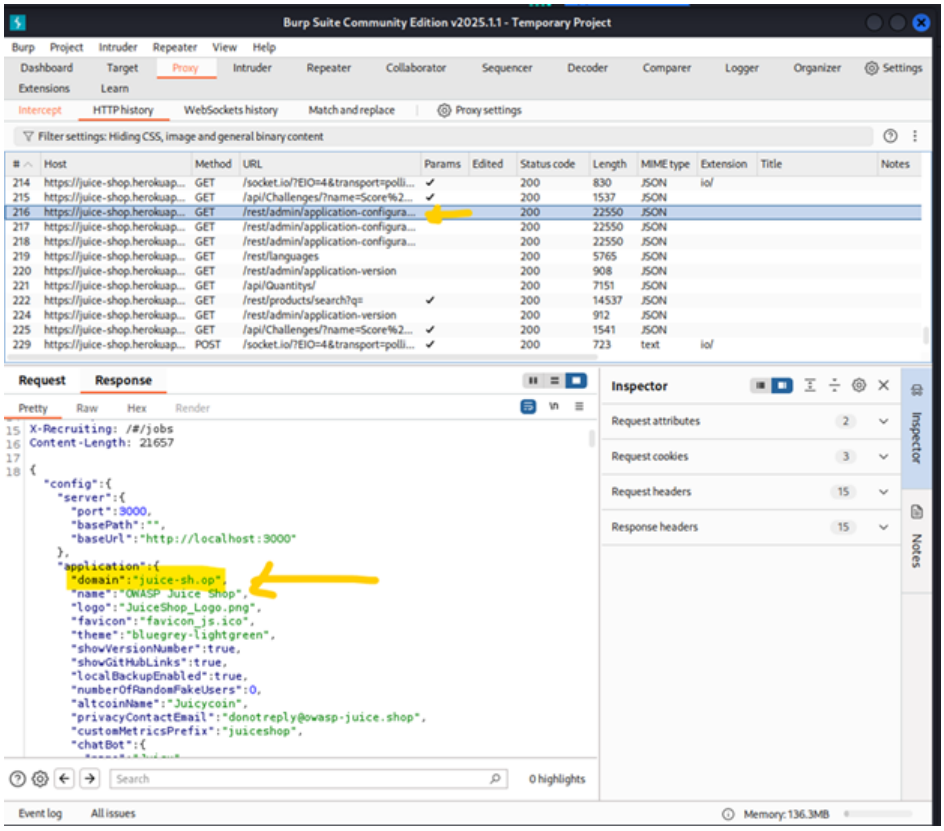

Figure 14: Enter Caption

- Thus we can conclude that the email for this user is `mc.safesearch@juice-sh.op` and the password is `Mr.NOOdles`.

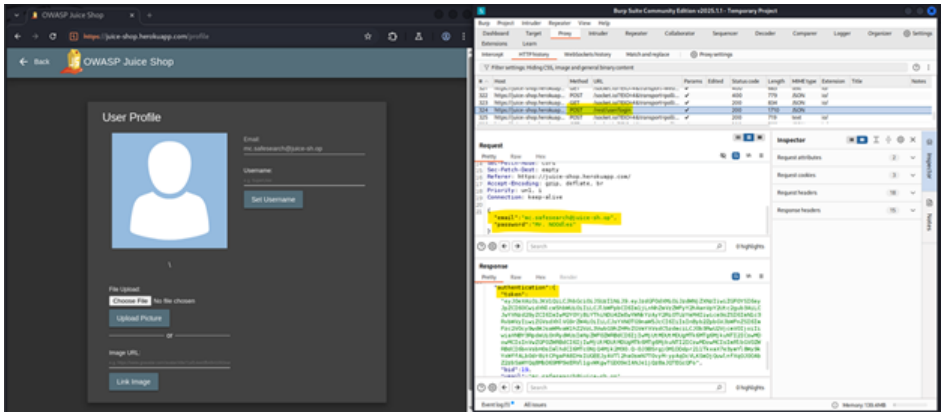

Figure 15: Enter Caption

- Like this we were able to Login with a user by only preforming a simple research with the username Mc Safesearch.
- Lessons learned is that you will need to google and preform simple search before preforming an attack or pen-test.

4 Exploitation 4 (Broken Access Control)

(Mohamed Ehab Zaghloul - 120220090)

4.1 Description:

- Our goal here is to find an access control vulnerability and we found an interesting one.
- Our vulnerability is simply a broken access control where a certain user can manipulate and add items to another user's basket.
- For example, there are 2 users: USER A & USER B

  USER A (hacker): contain a t-shirt in the basket and he wants to add another item which is a banana juice to another user's basket without his permission.

  So, by manipulating an http header request for the server this can be achieved.

  USER B (victom): His added items are the following: a t-shirt and a strawberry juice, but the attacker/hacker added a banana juice.

4.2 Steps

- First things first, we need to make a new user.
- A new user called test was created successfully.

Figure 16: Enter Caption

- Then we will add a product in the basket and go to Burp to catch its header and manipulate it to add another thing to another user's basket and successfully exploit the vulnerability.

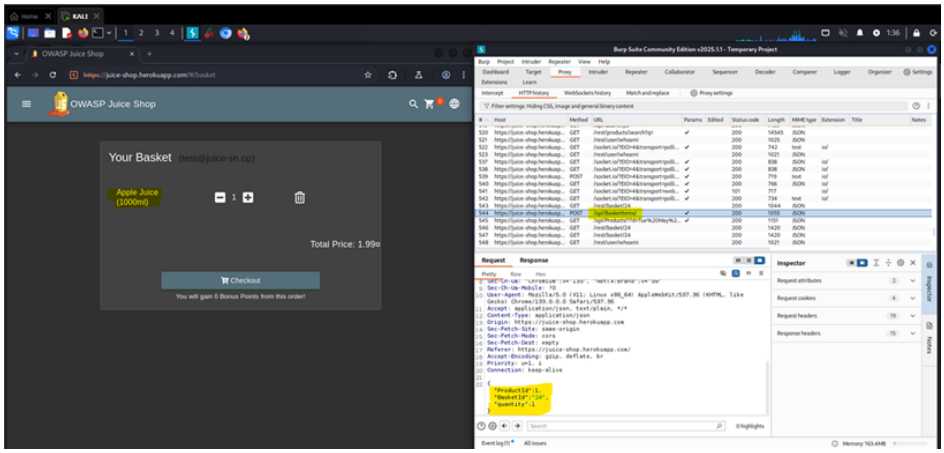

Figure 17: Enter Caption

- Now I will make another new user on another machine to show the exploitation of the vulnerability step by step.
- A new user is created and an item is added to the basket.

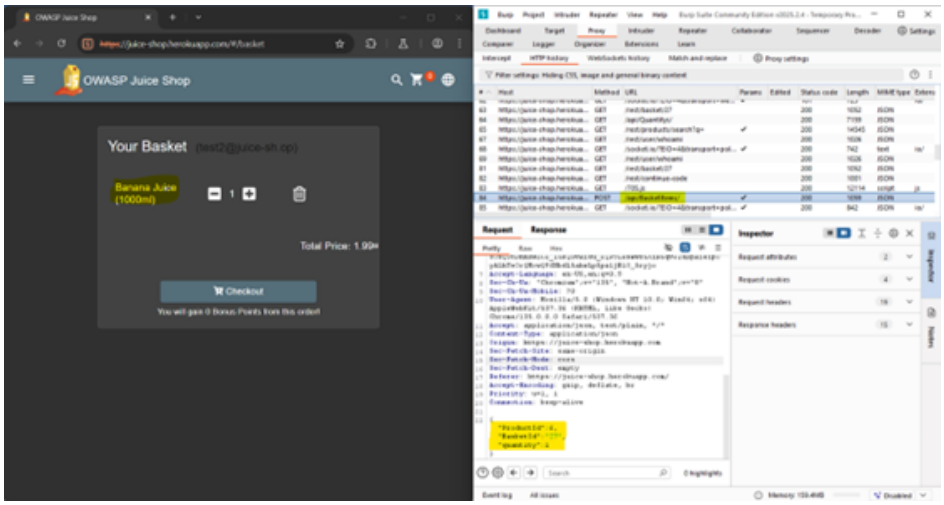

Figure 18: Enter Caption

- Now to make everything clear, we made 2 new users user “test” and another user ”test2”
- User test Basket ID is 24.
- User test2 Basket ID is 27.
- Now we will exploit the vulnerability from user “test” and add an item to user “test2” by manipulating the header of the request.
- From the User test we will add a product in the other user which is the Fruit Press with productid 25.

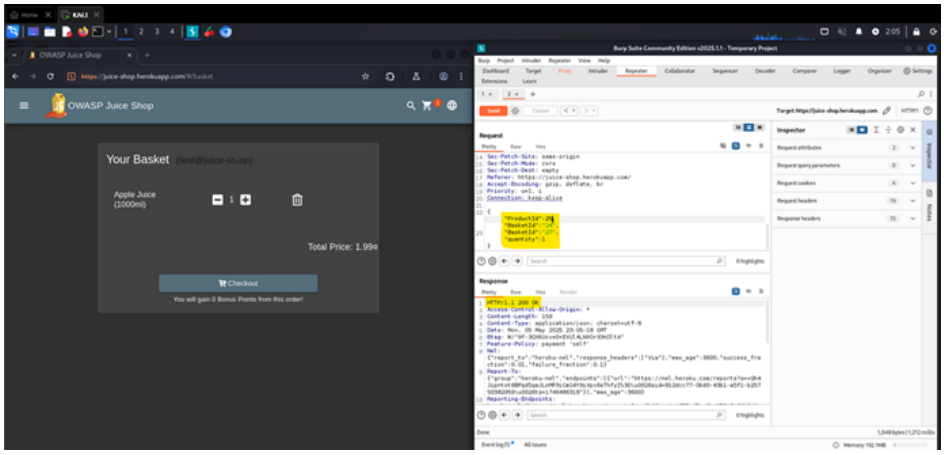

Figure 19: Enter Caption

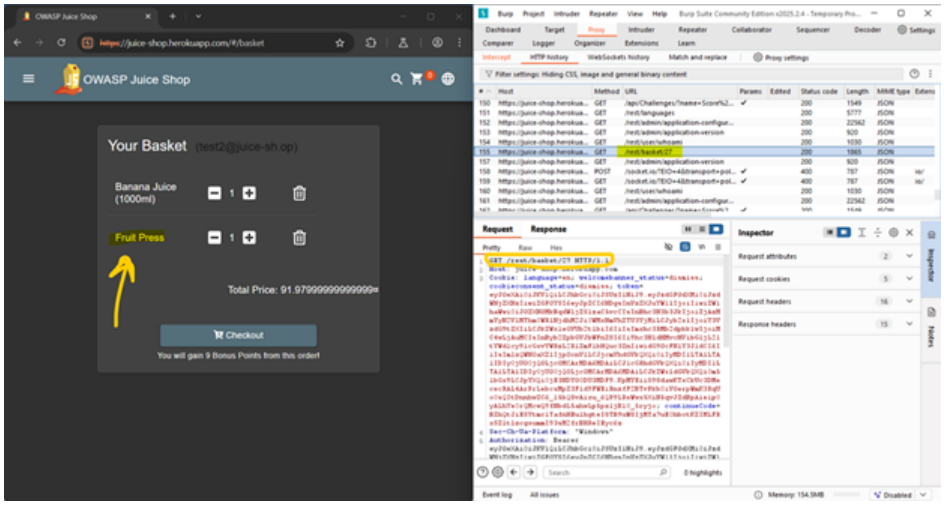

Figure 20: Enter Caption

- Here we go the product is added successfully from user “test” to the other user “test2” and like this we exploited the vulnerability successfully.

5 Exploitation 5 (Forged Review)

(Mohamed Ehab Zaghloul - 120220090)

5.1 Description:

A forged review vulnerability is an access control vulnerability that any user can add a review for himself or any other user by manipulating the http request.

Exploiting this vulnerability have many serious concerns as there will be no privacy for any user and anyone can post any review by any user.

It is also dangerous as SQL injections are possible in this scenario.

- For exploiting this vulnerability, we will be using 2 users, one attacker and the other is the victim.
- In the following image shows the http request of a review made by the attacker user, “Carrot Juice tastes good!!”

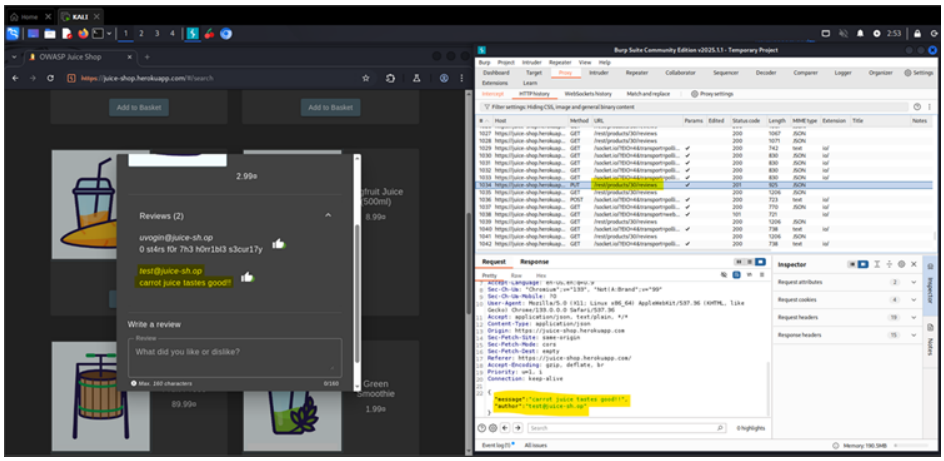

Figure 21: Enter Caption

- By manipulating the author and typing any known user for the attacker and sending the request to the server, you successfully exploited this vulnerability as shown in the following image.

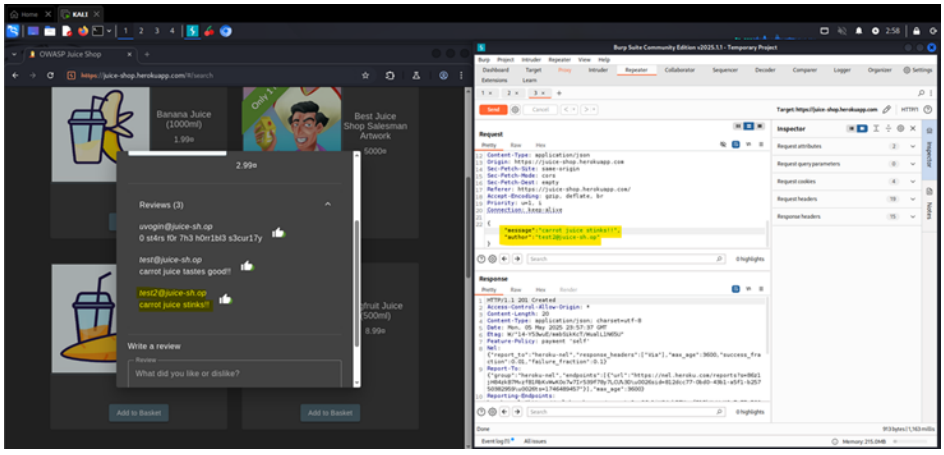

Figure 22: Enter Caption

- Like this we exploited the vulnerability successfully and added a review by another user from my very own user http request.
- Below are the two reviews that were added by the attacker one for his own user and the other for the victim by exploiting vulnerability.

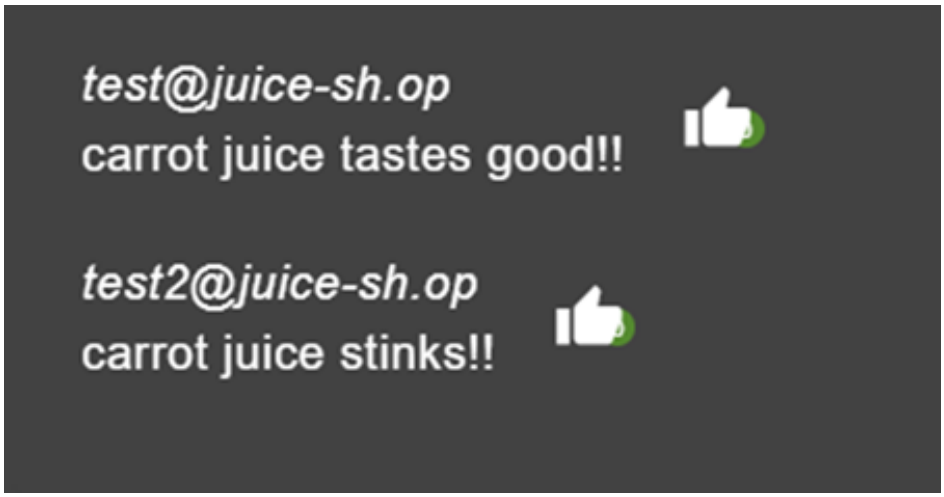

Figure 23: Enter Caption

6 Exploitation 6 (Sensitive Data Exposure)

(Mohamed Ehab Zaghloul - 120220090)

- In this Exploitation our goal is to access confidential files using other existing files by exploiting a vulnerability.
- Now will head over to about us in the website, then press on the terms link in the page.
- After pressing on the link take the packet to the repeater to manipulate it.

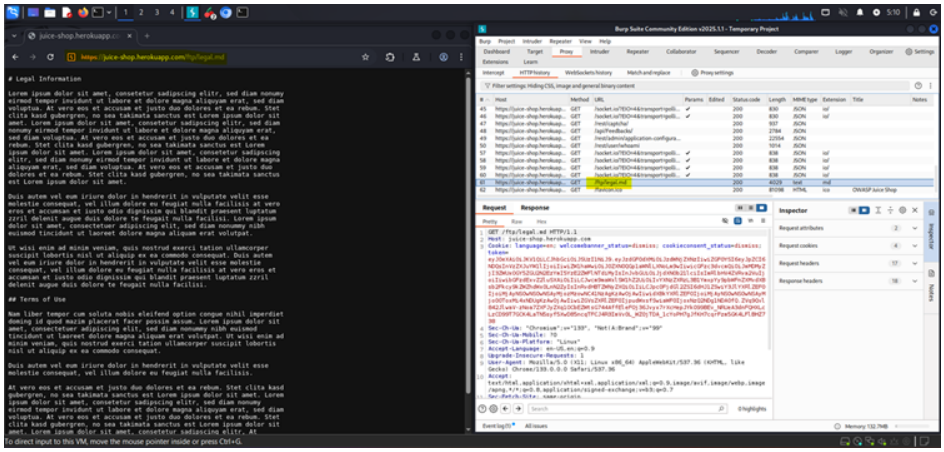

Figure 24: Enter Caption

- After taking the packet header to the intruder, change the file path to by only `GET /FTP/`
- What will happen here is that the server will give us a response with several file paths and confidential ones as in the following pictures.

- The server gave a 200 OK status code and replied

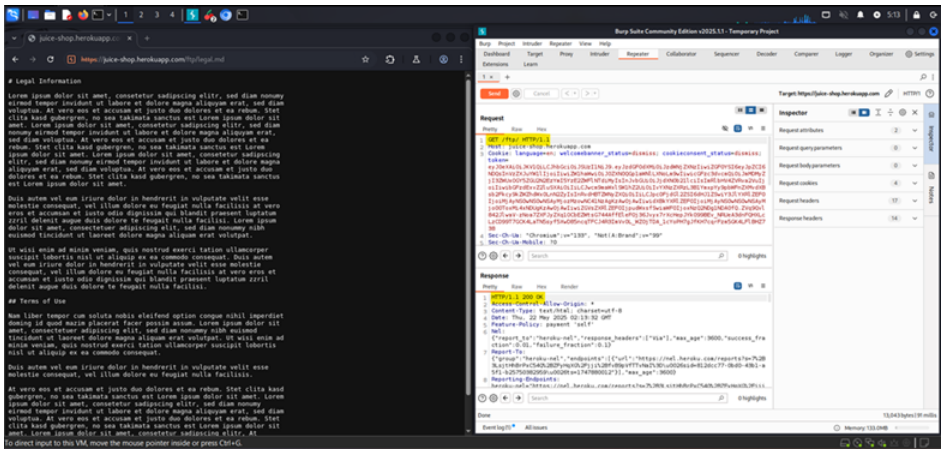

Figure 25: Enter Caption

- Now we will take a look on the rest of the packet to see what we got.
- In the following picture while taking a look on the reply from the server, we found several files, the highlighted ones.

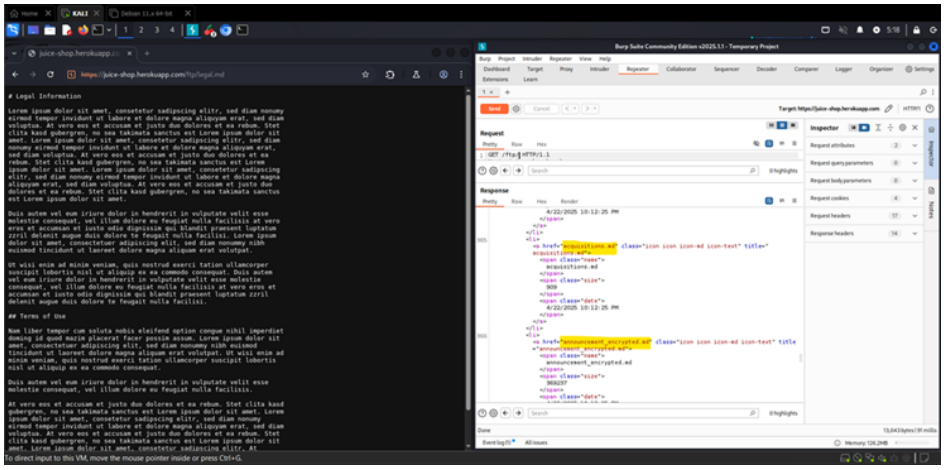

Figure 26: Enter Caption

- By trying to access the file `acquisitions.md`, we fing the following...

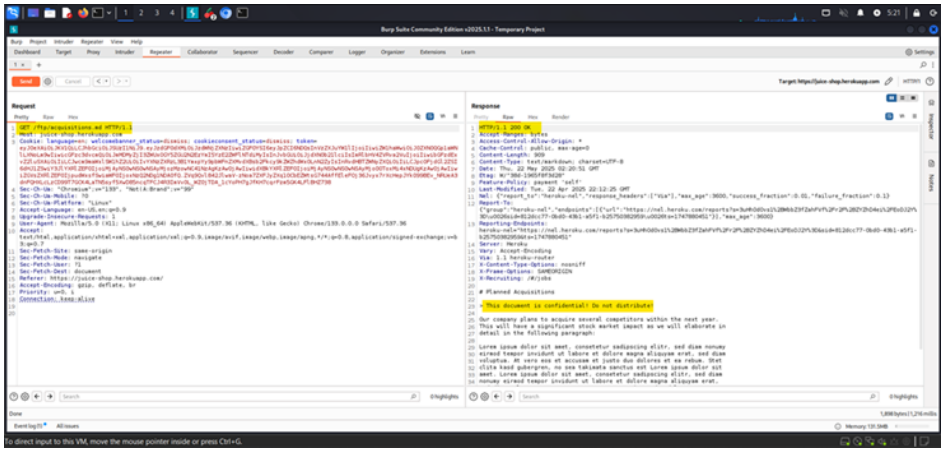

Figure 27: Enter Caption

- We were able to access the file name and the file path that lead us to the content of the file, thus we exploited the vulnerability successfully and reviewed sensitive data.

7 Exploitation 7 (Sensitive Data Exposure)

(Mohamed Ehab Zaghloul - 120220090)

- Now we will exploit a vulnerability that allows the access to ftp directory and accessing sensitive files by adding a script and encoding it to cope with the URL and work.
- Using gobuster we got all the directories that can be accessed directly, from these was the FTP directory which contain in it sensitive files.

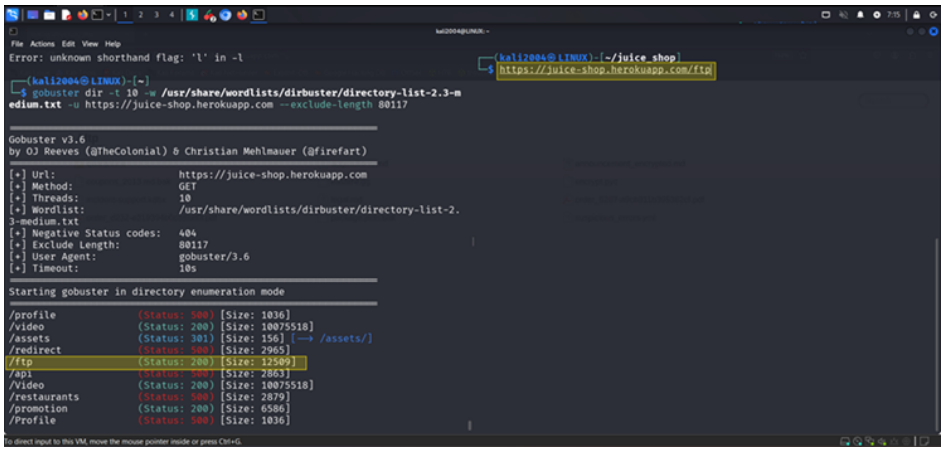

Figure 28: Enter Caption

- Now we will try to access the FTP directory through the search engine.

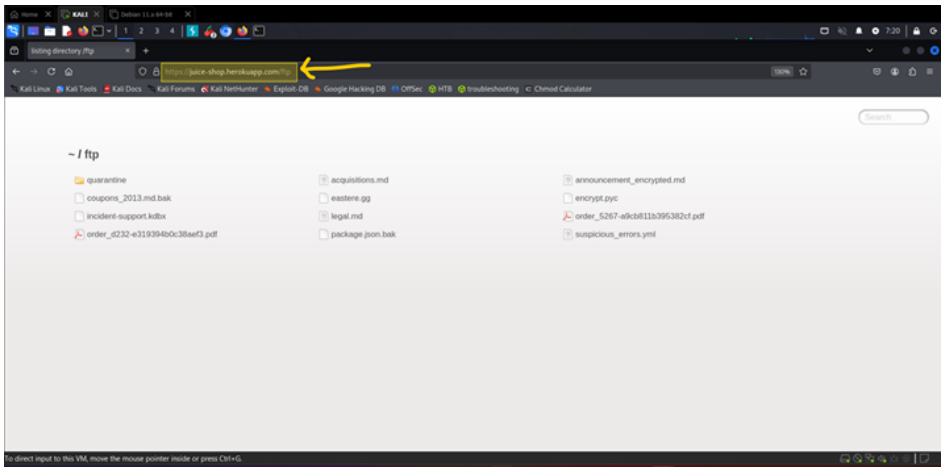

Figure 29: Enter Caption

- And here we go we accessed the FTP directory successfully.
- Now we will try to open any sensitive file to be successfully accessing sensitive information.
- We will try opening a backup file which is `package.json.bak`

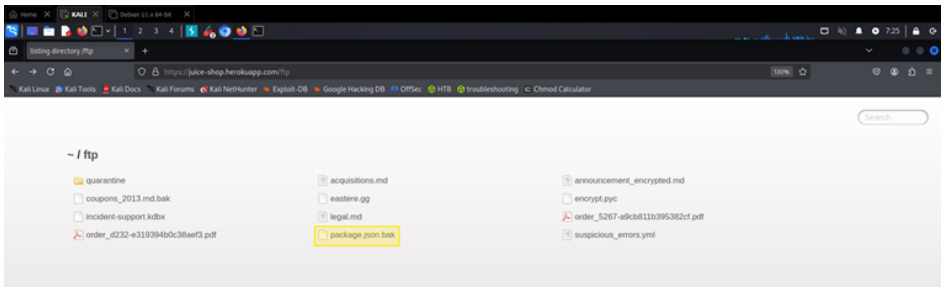

Figure 30: Enter Caption

- While trying to open it I got a 403 error that the .bak extension is not allowed to be opened, so we will use Null byte extension (Null byte poising) and its function is that it simply allows you to open a file with an extension that is not the original extension of the file.

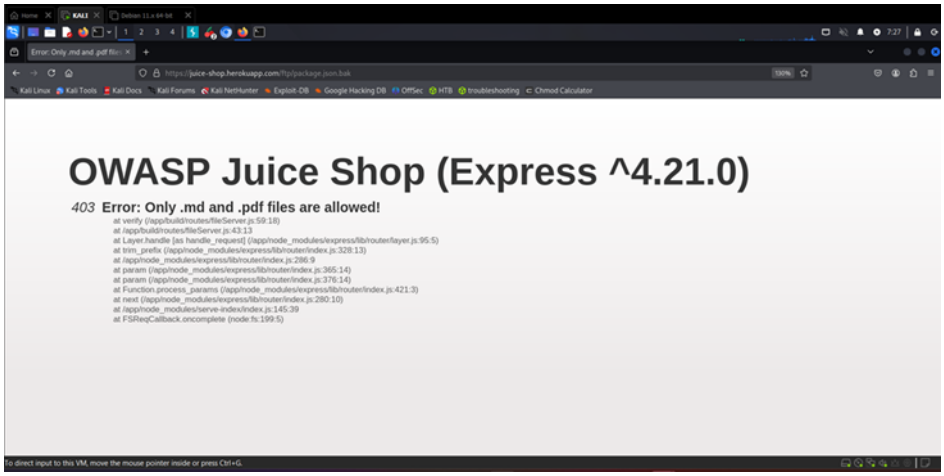

Figure 31: Enter Caption

- Unfortunately, after using the null byte injection, we still got a 400 Bad Request and this is a reason for not encoding the null byte injection
- We will encode it on Burp then try again sending our request to the server.

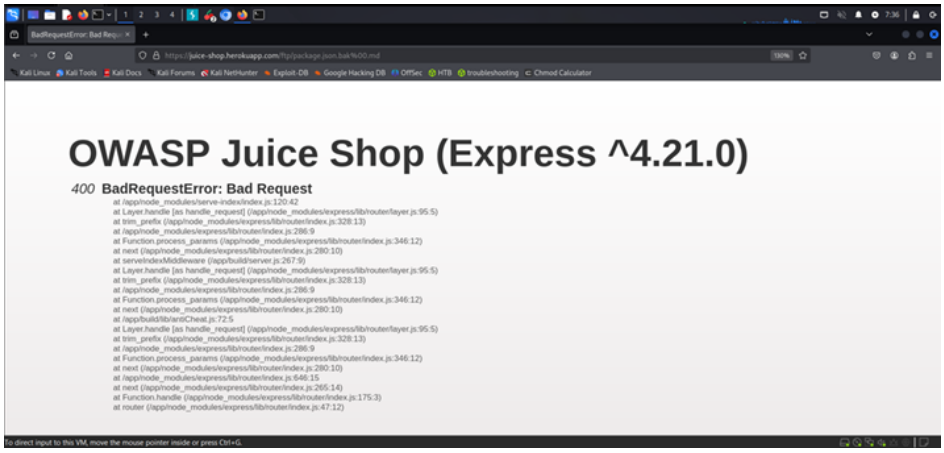

Figure 32: Enter Caption

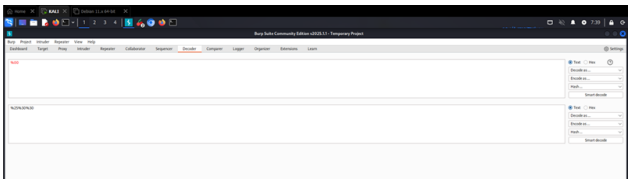

Figure 33: Enter Caption

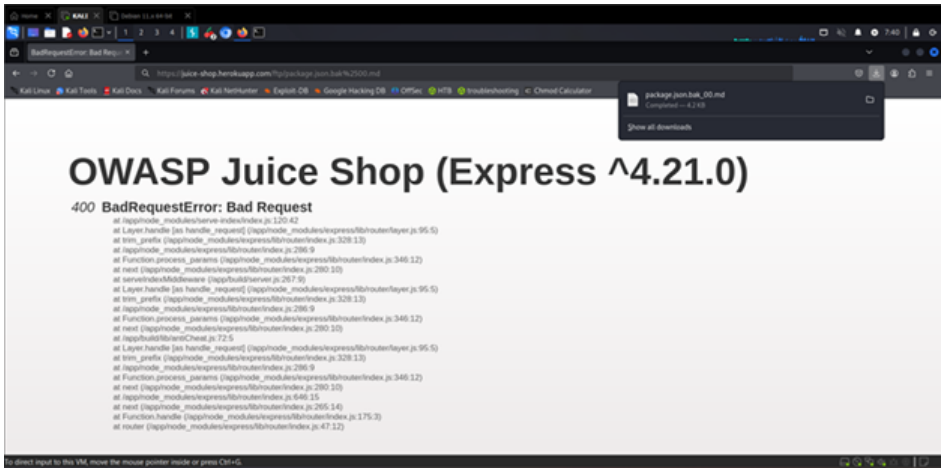

Figure 34: Enter Caption

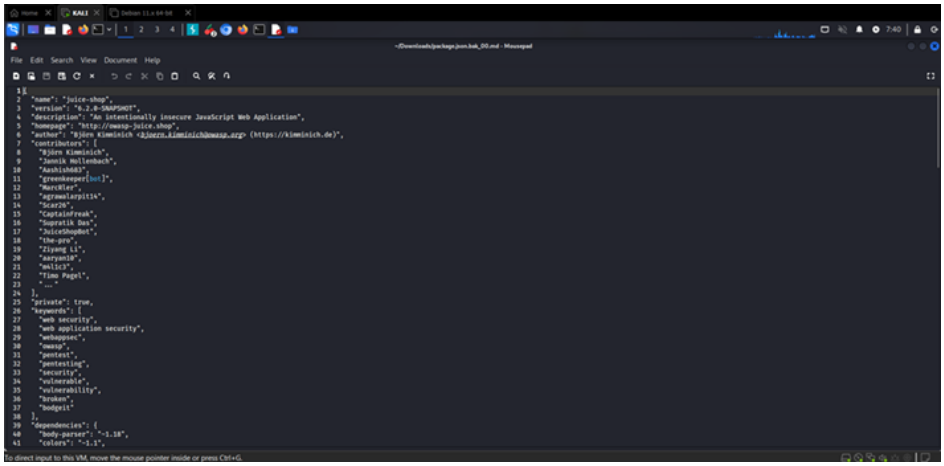

Figure 35: Enter Caption

- Here we go we accessed the backup file after encoding the null byte injection that allow us open the file with .md extension.

8 Exploitation 8 (Sensitive Data Exposure)

(Mohamed Ehab Zaghloul - 120220090)

- First step is that we need to create a new user, add items to the basket, add an address and add the payment information.
- All of the pervious requirements will be done in the following screenshots.

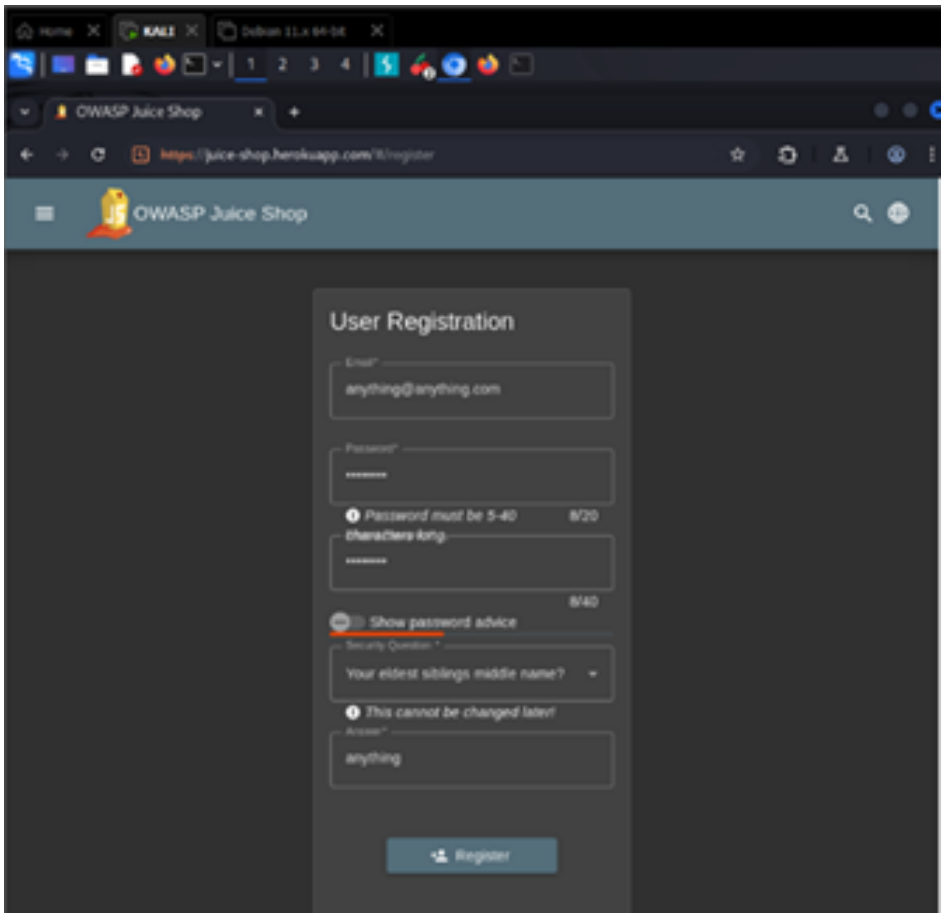

Figure 36: Enter Caption

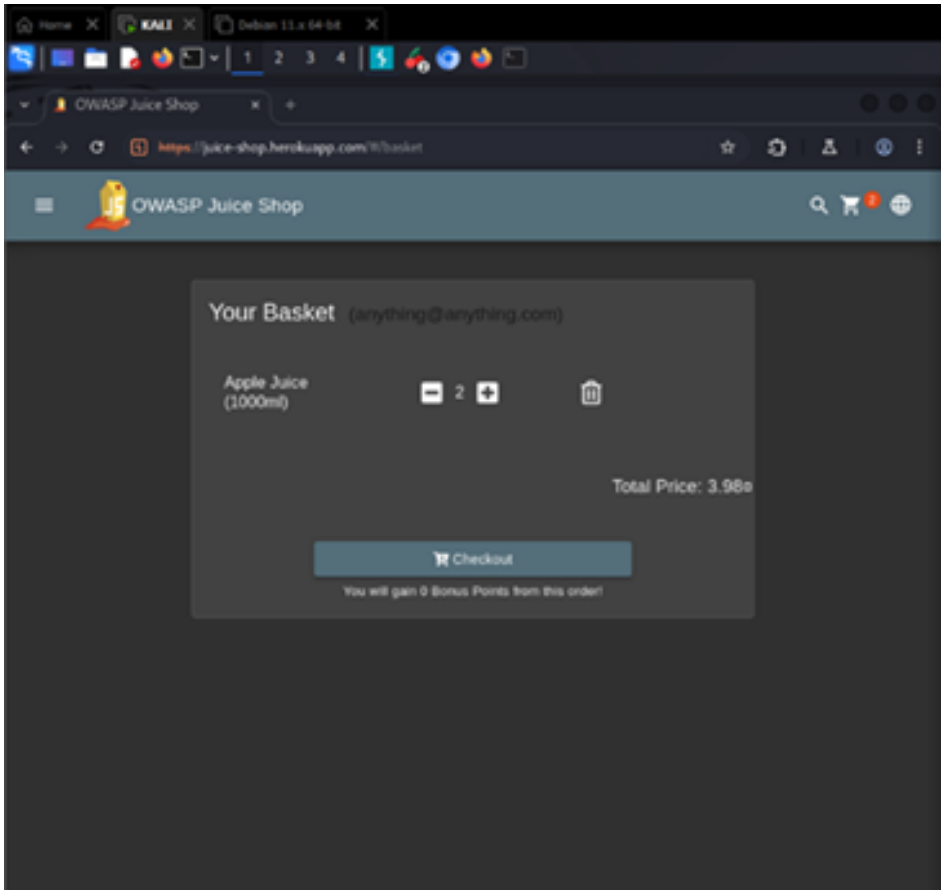

Figure 37: Enter Caption

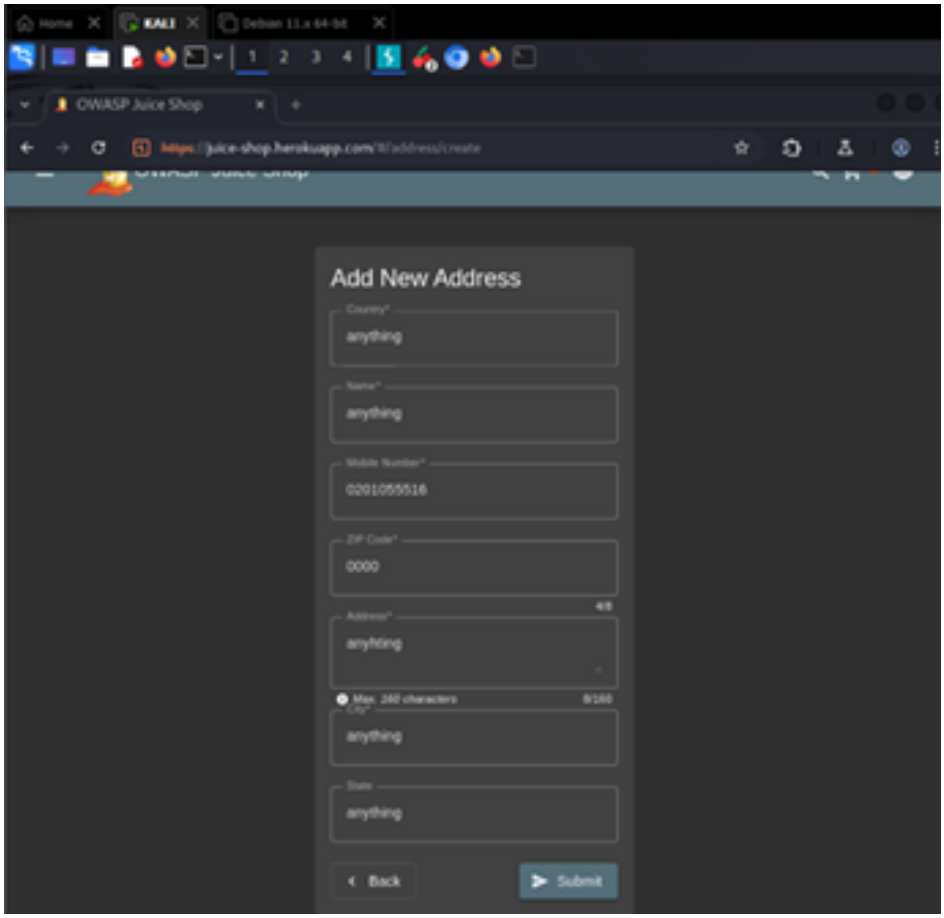

Figure 38: Enter Caption

- Now we are placing our order and it was placed successfully.

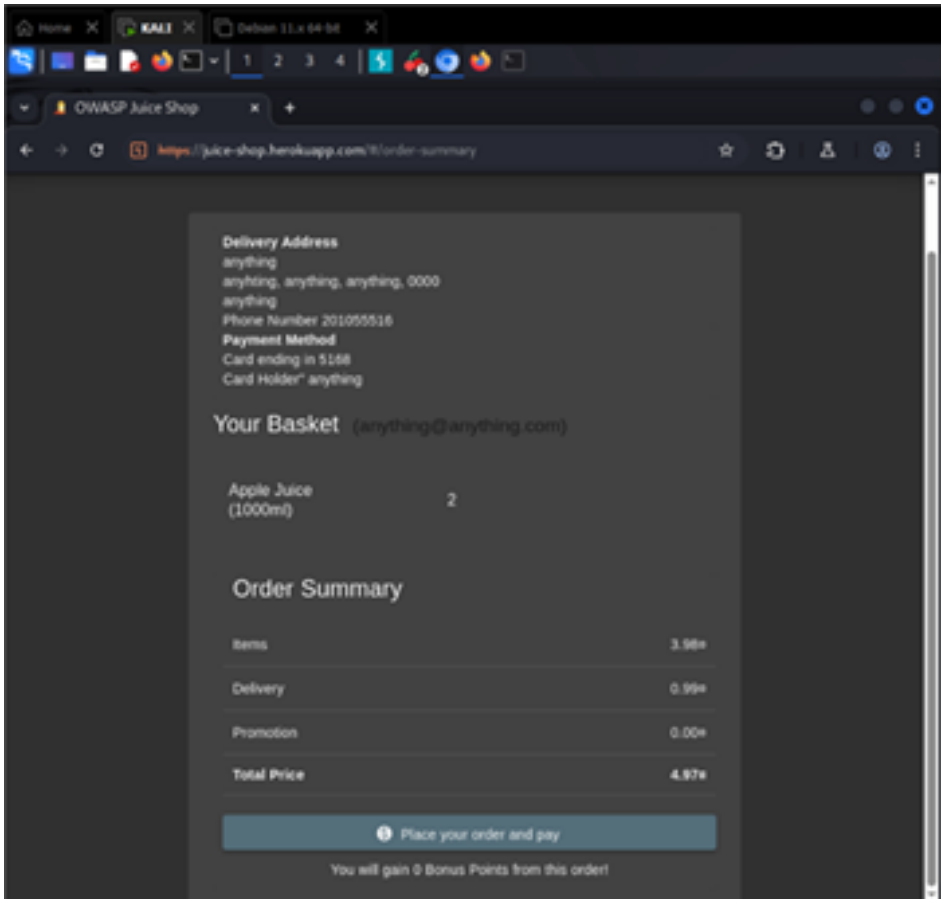

Figure 39: Enter Caption

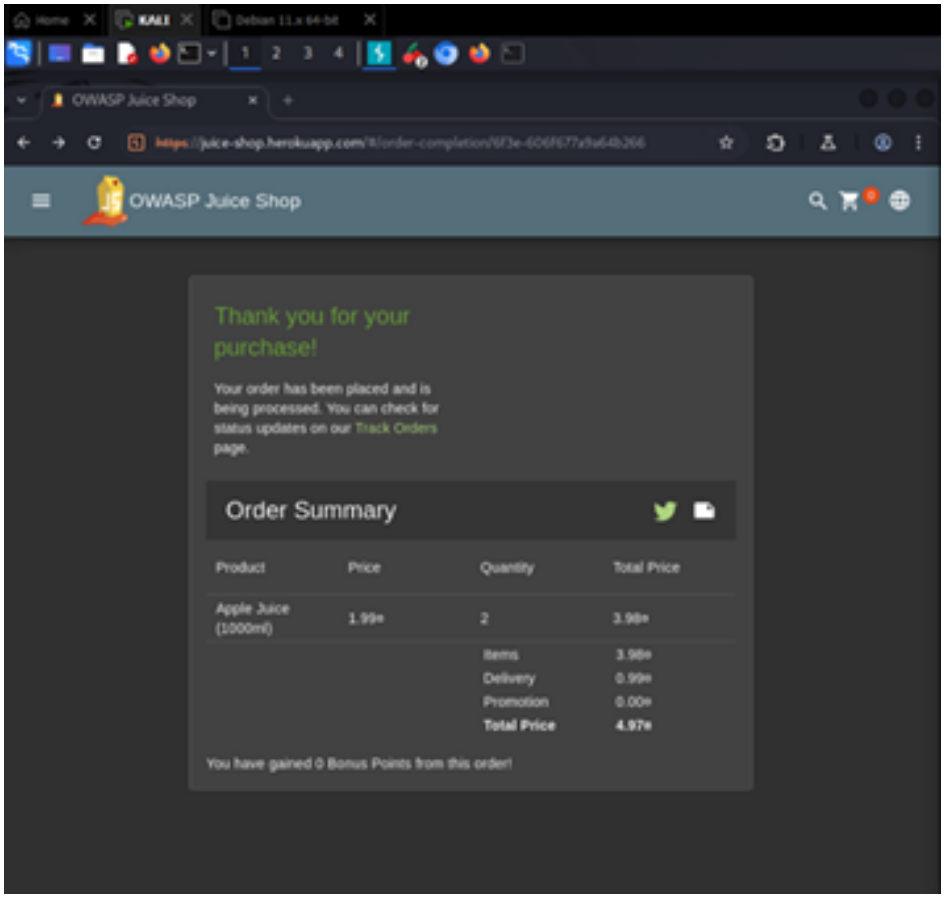

Figure 40: Enter Caption

- Now we will try to track our order and the header of the order traking will be sent too the repeater.

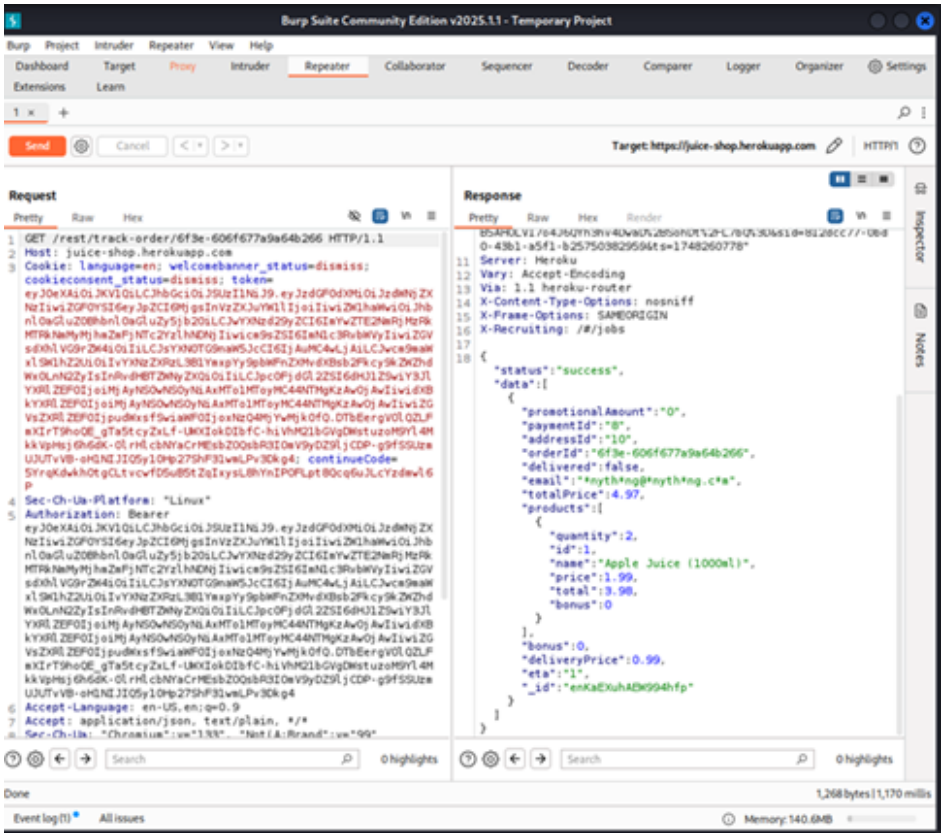

Figure 41: Enter Caption

- In the email that is in the response we can find that all the vowels are replaced by an asterisk and this can be a form of security, but actually we will be able to exploit a vulnerability from this.

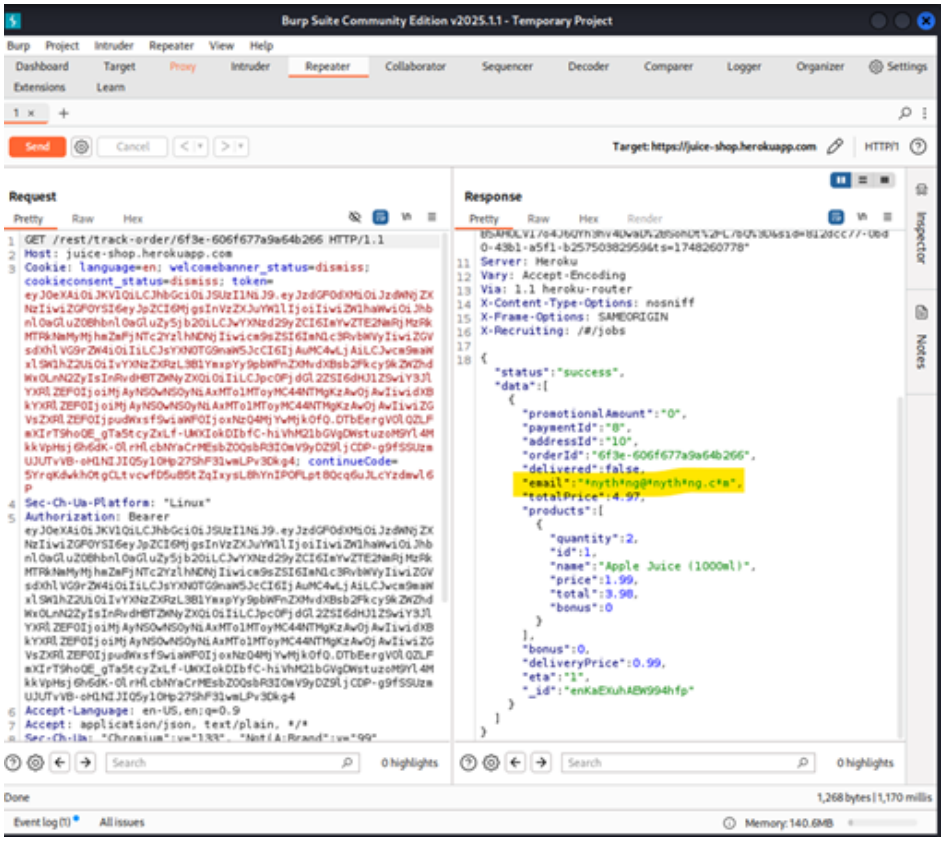

Figure 42: Enter Caption

- If we go back to the website and export our user's data that we created we can extract information about everything the user has done.

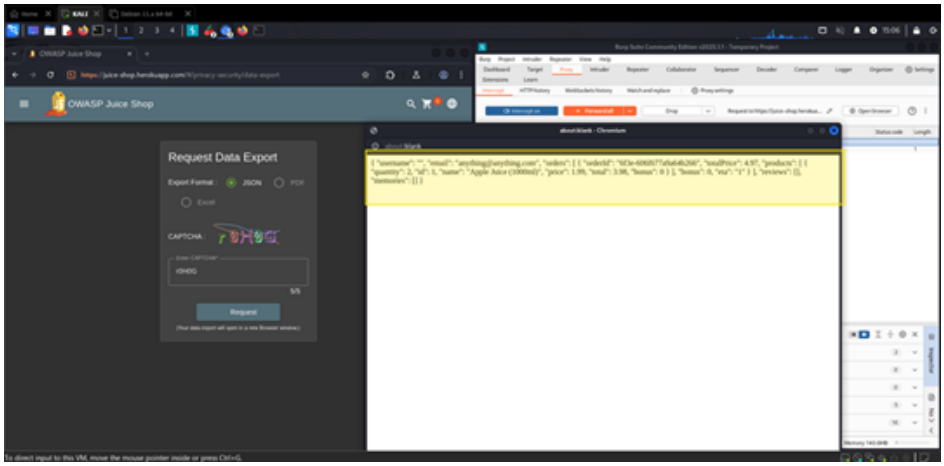

Figure 43: Enter Caption

- Since we extracted the data successfully and the email returned to its normal and was shown completely without any asterisk, we can try a method to exploit this vulnerability.
- We will login by a user that is the same as the admin's email but change only one vowel to see if we can benefit from the vowel replacement and extract information of the admin user.

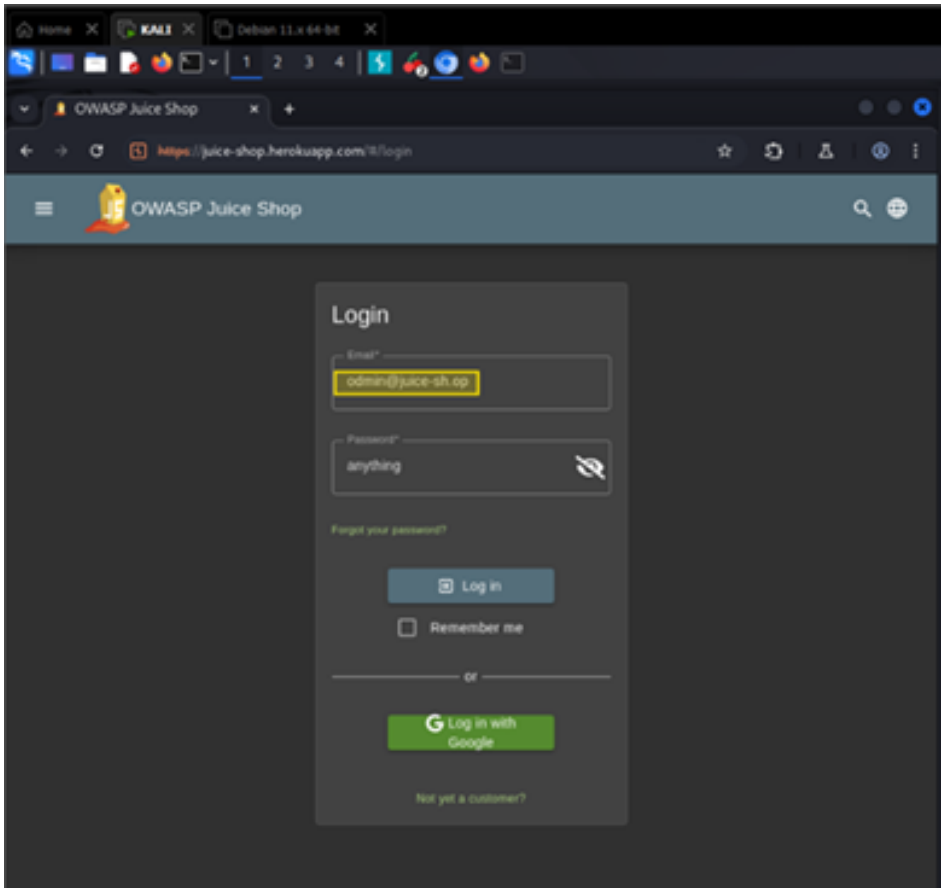

Figure 44: Enter Caption

- Here we go, we revealed the admin information successfully.

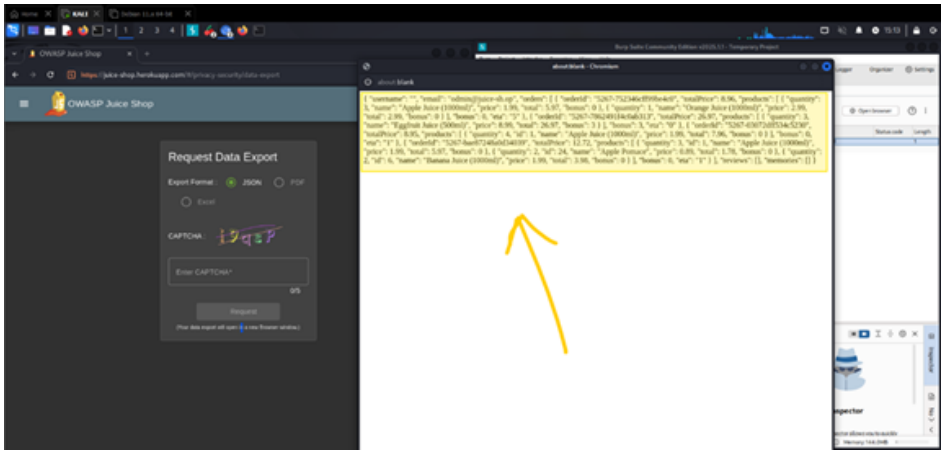

Figure 45: Enter Caption

- Since I did not order anything with odmin user, and the information I have contain several orders, then this makes me sure that this is the admin user.

9 Exploitation 9 (DOM “XSS”)

(Mohamed Ehab Zaghloul - 120220090)

- Now we will try to exploit an XSS (client side vulnrability) which is cross site scripting.
- DOM XSS stands for Document Object Model Cross-Site Scripting.
- The script that we will exploit the vulnerability with is the following script...

¡iframe src=”javascript:alert(’xss’)”¿

- Ø It creates an `<iframe>` element whose src is a JavaScript URL.
- Ø When rendered, it executes the JavaScript code inside the iframe in this case, `alert('xss')`.
- What we will do is that we will apply the script in the search on the top right of the page.

Figure 46: Enter Caption

- When we hit enter, the script is added in the inspector tab of the devtools of the webpage successfully.

Figure 47: Enter Caption

10 Exploitation 10 (Broken Anti-Automation)

(Mohamed Ehab Zaghloul - 120220090)

- In this vulnerability exploitation, what we will do is that we will bypass the captcha check.
- Bypassing the captcha check and sending several requests too the server without being checked could lead to several vulnerabilities and risks that will be discussed in the end of the documentation.
- Now we will start our attack by sending a feedback to the server then taking the http request to the repeater to test it if the captcha ID is used once or can be used several times without any blocking.

[H]

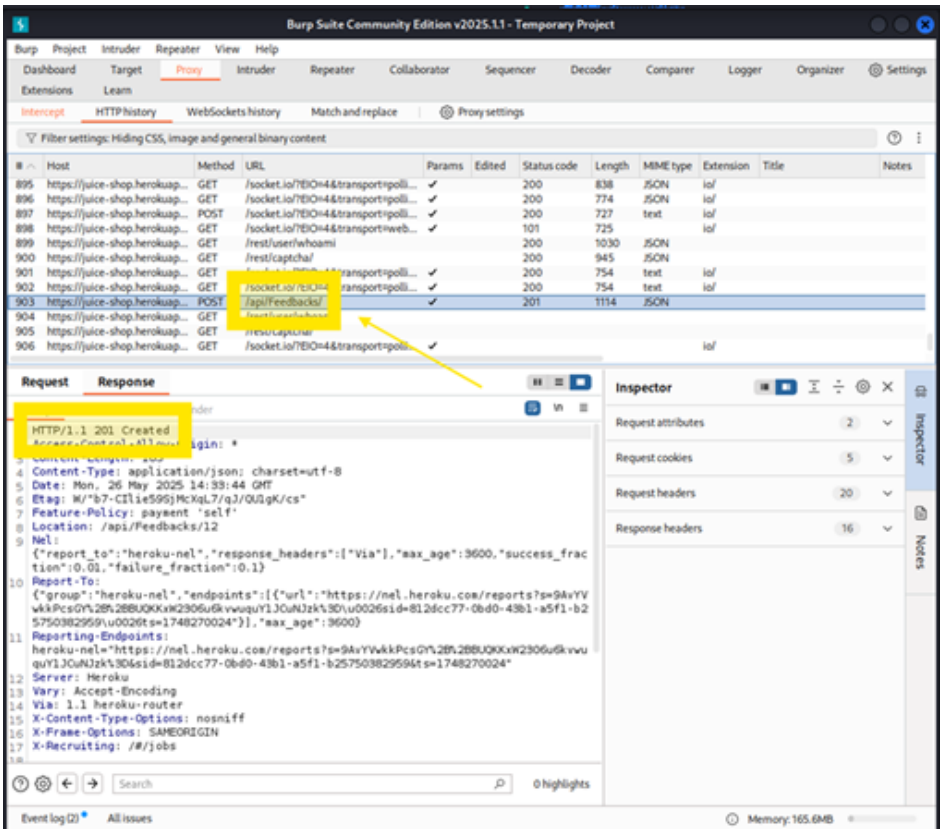

Figure 48: Enter Caption

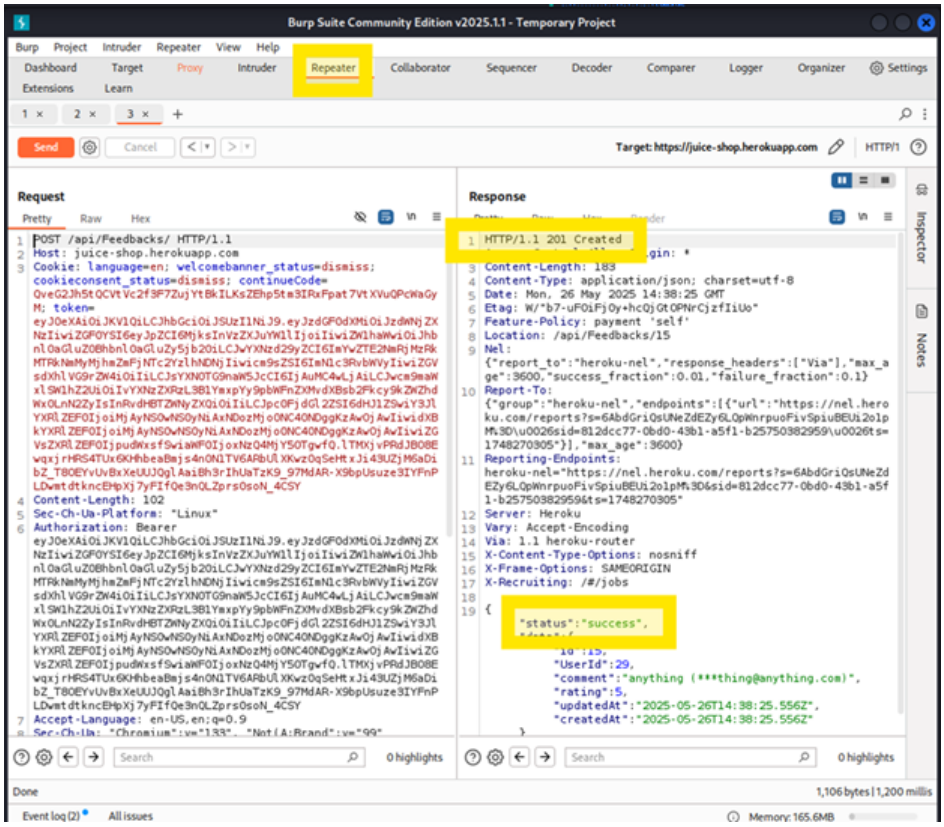

Figure 49: Enter Caption

- Like this we have created the feedback successfully and tested the captcha ID whether it can be used several times or not and we knew from the repeater that we can you it several times and it will be successful.
- Now in the intruder we will send around 18 requests to the server using the null payloads type to test and exploit the vulnerability.

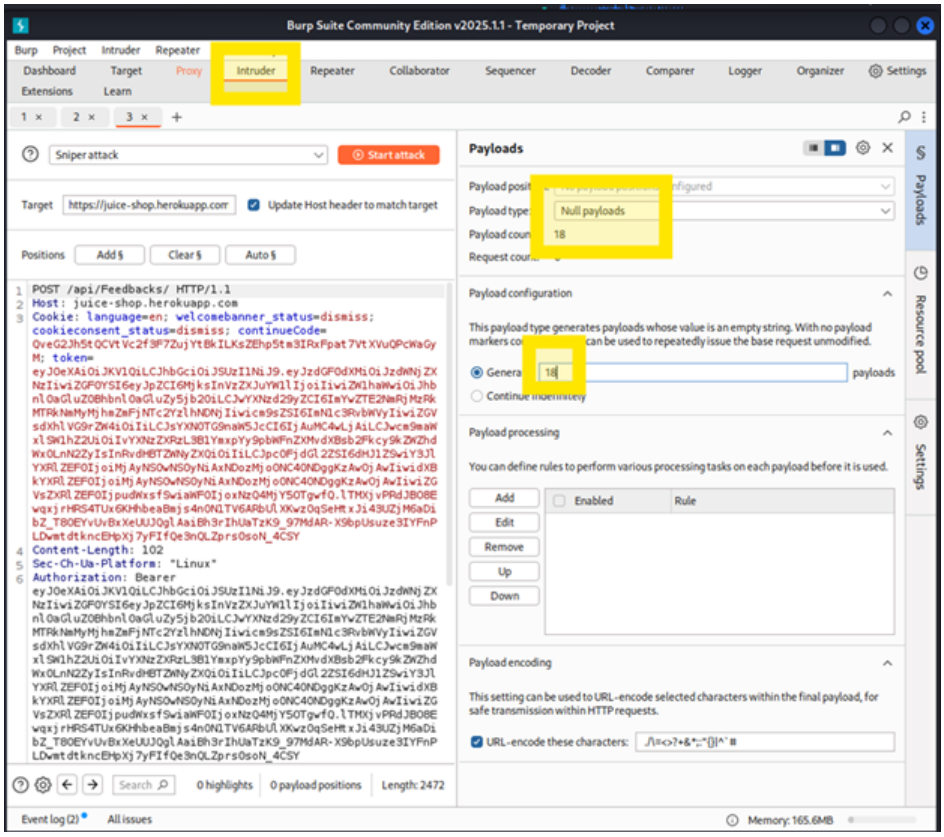

Figure 50: Enter Caption

- Like this we successfully sent 18 feedbacks to the server and all 18 were created. As shown the last feedback was created.

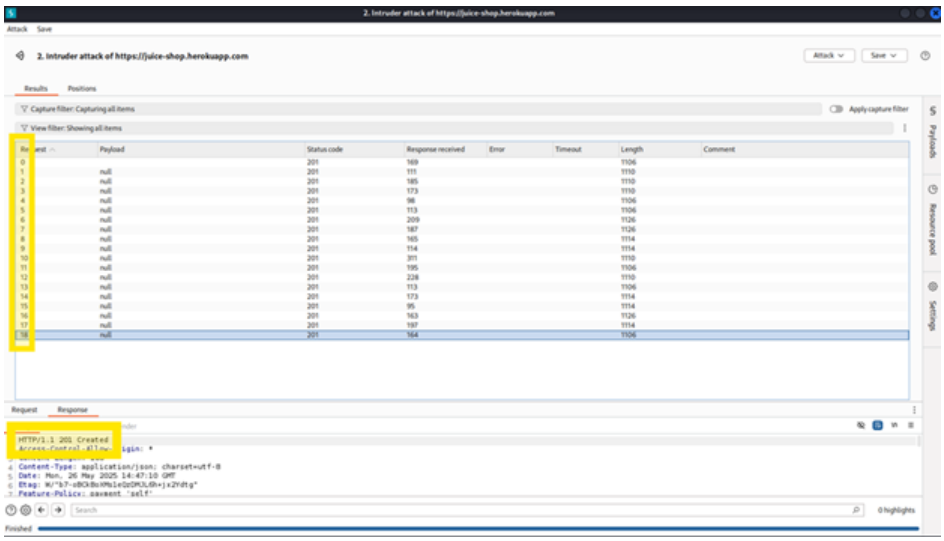

Figure 51: Enter Caption

10.1 So what are the risks that can originate from a vulnerability like this...

- **Automated Spam Submission:**
  - Bots can flood the system with fake feedback, reviews, or comments.
- **Denial of Service (DoS):**
  - Spamming feedback might overwhelm the system or database.
- **Abuse of Business Logic:**
  - If feedback affects ratings, attackers could manipulate customer scores, harming reputation or inflating ratings unfairly.
- **Credential Stuffing / Brute-force Attacks (in other contexts):**
  - Similar bypass could allow attackers to brute-force login forms or abuse APIs.

11 Exploitation 11: Reset Jim's Password

(Amr Khaled Mohamed - 120220236)

In this challenge we are trying to reset one of the user's account called jim and here are the steps that we have gone through to attack this user.

In This step after searching about the users we can attack them, we have known this email of a user and we will start to attack on him and in the figure 2 we have just fone a trivial test in the login page using the email we have just get and tried usring any password.

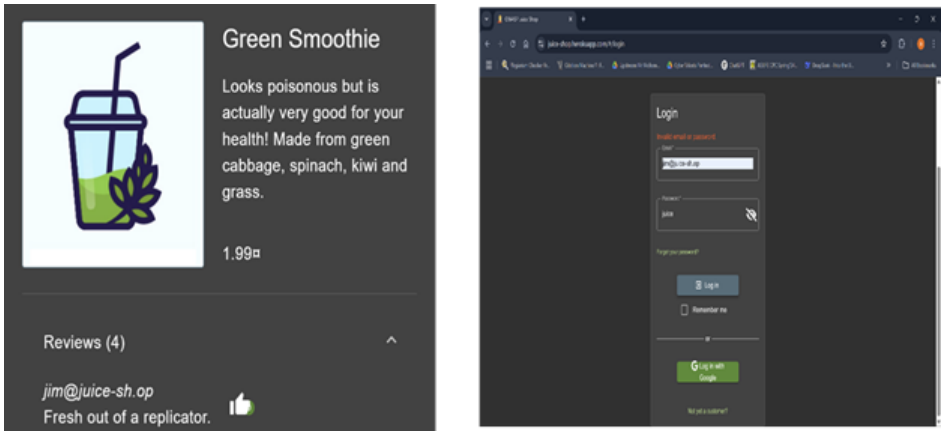

Figure 52: Enter Caption

11.1 Identifying Jim's Identity:

In a comment, Jim quote a line from Star Trek. So, we know that there's maybe a connection to "Star Trek", so we search for "Jim Star Trek". Discover that Jim refers to James Tiberius Kirk from "Star Trek".

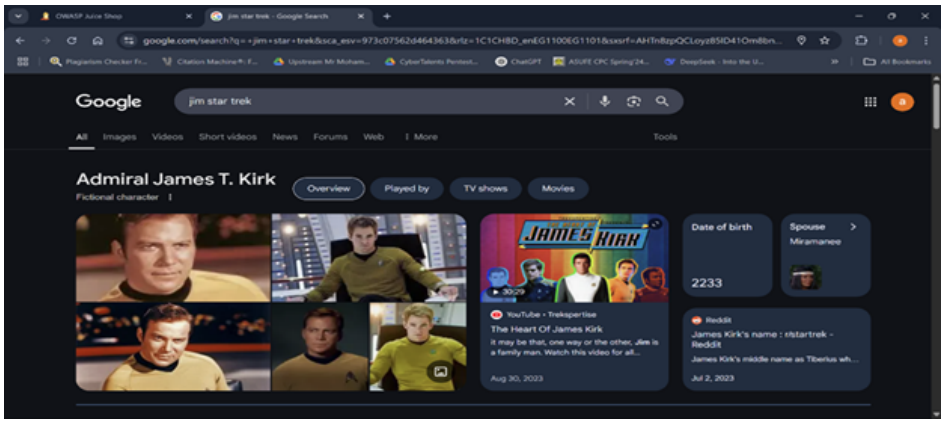

Figure 53: Enter Caption

In this step we are trying to pypass the account by clicking the forget password and here we have found that the website is asking us for answering a question as a step for accessing the account.

George Samuel Kirk so, in this step we tried using different words from his name which is George Samuel Kirk. and the name which was true is Samuel.

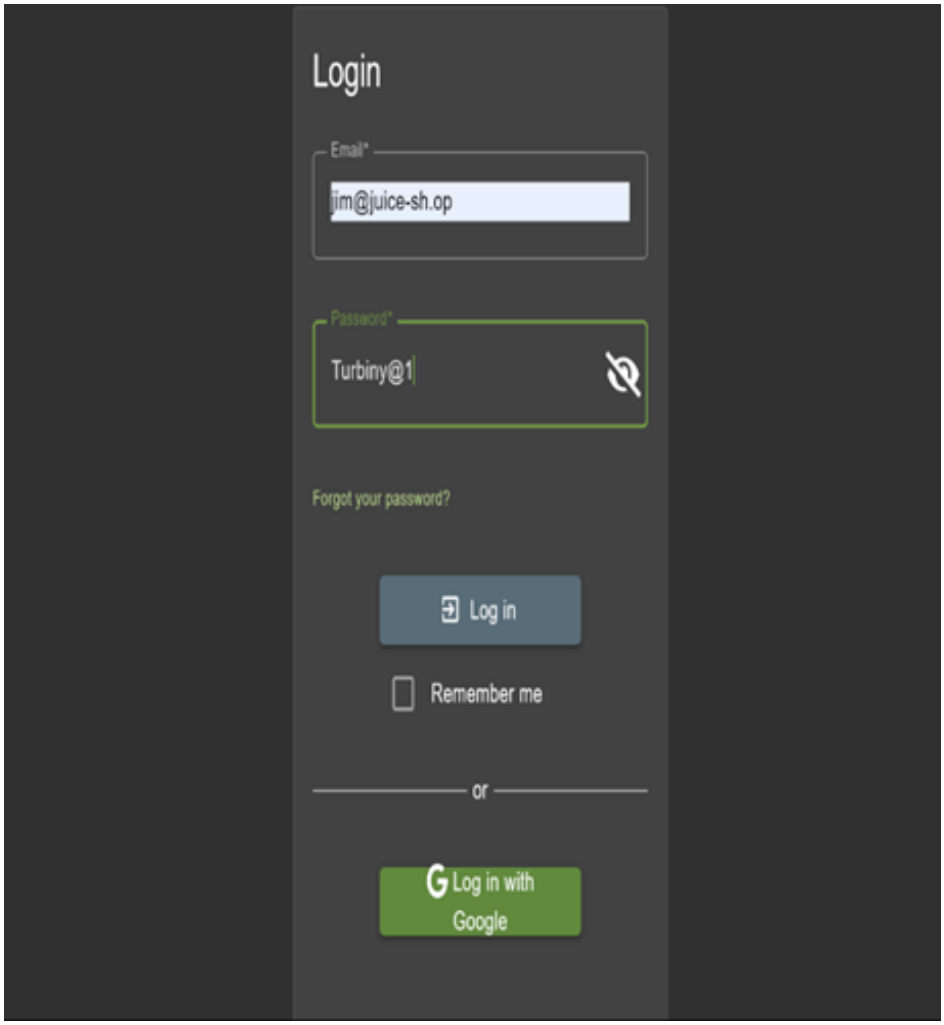

Figure 54: Enter Caption

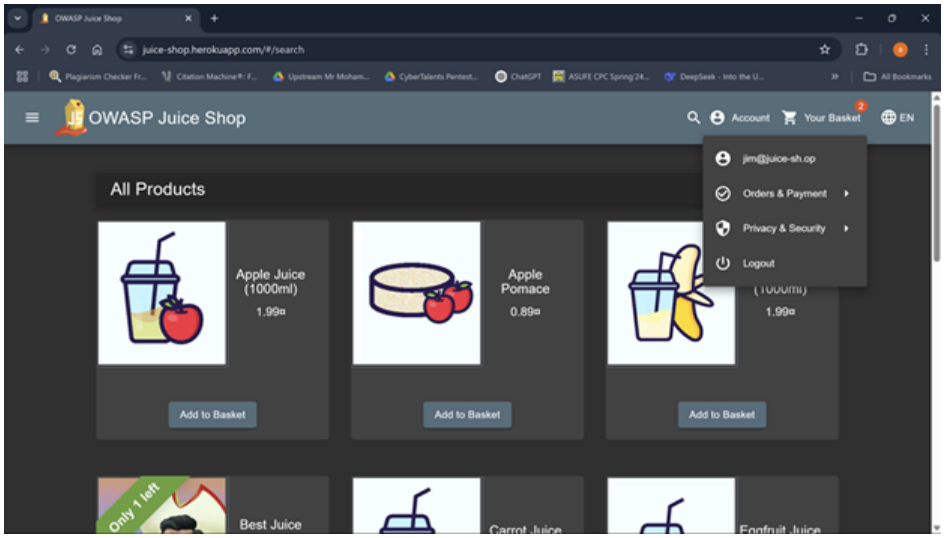

Figure 55: Enter Caption

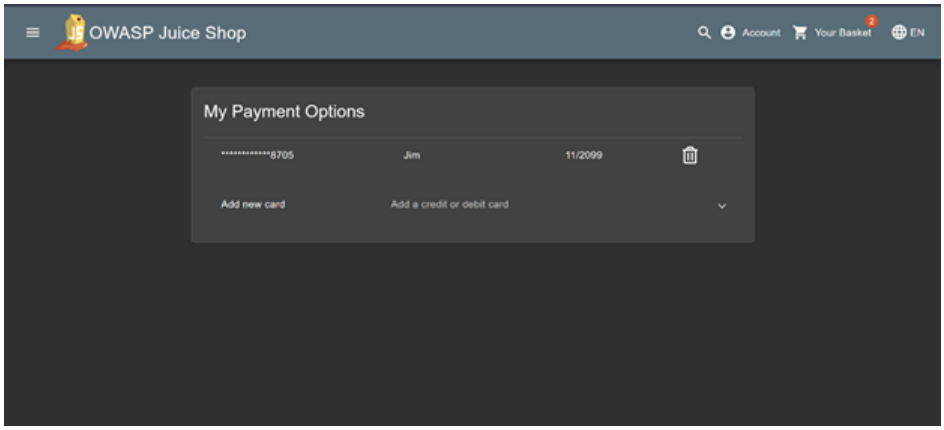

Figure 56: Enter Caption

At this point I have updated the new password of jim and logining into his account. so, I can gain.

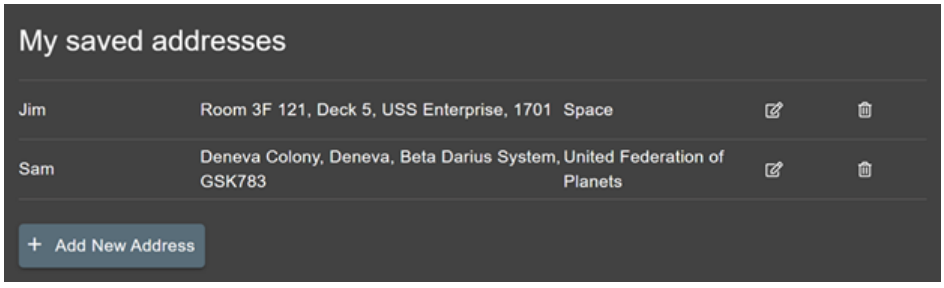

Figure 57: Enter Caption

12 Exploitation 12: Reset Bender's Password

(Amr Khaled Mohamed - 120220236)

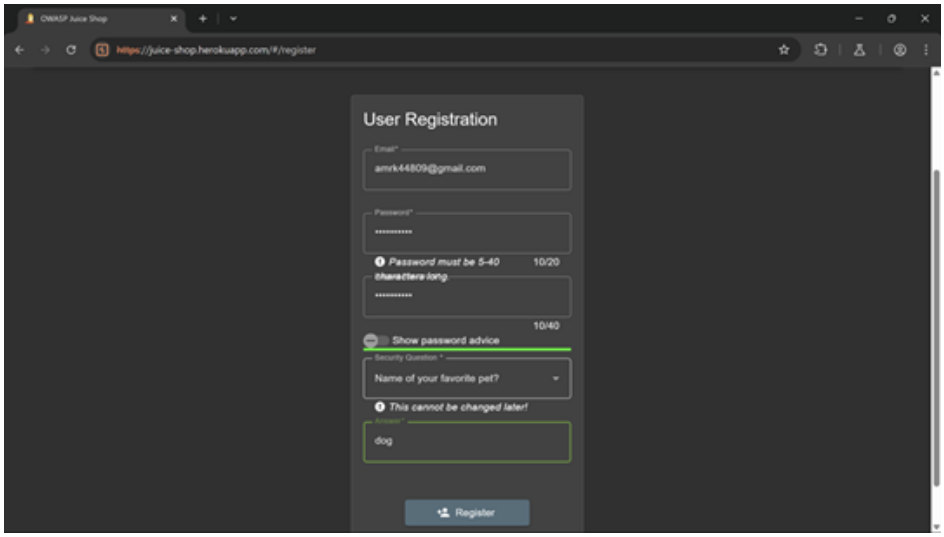

Figure 58: Enter Caption

Frist before we login into the bender's account we wanted to take more advantage of accessing admin account.

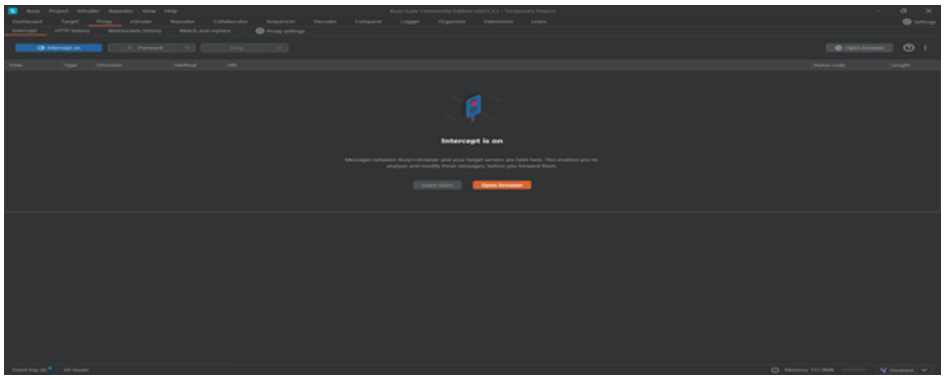

Figure 59: Enter Caption

So, we started creating a new account and before clicking register we have used the proxy in the the burpsuite to intercept the flow of the request.

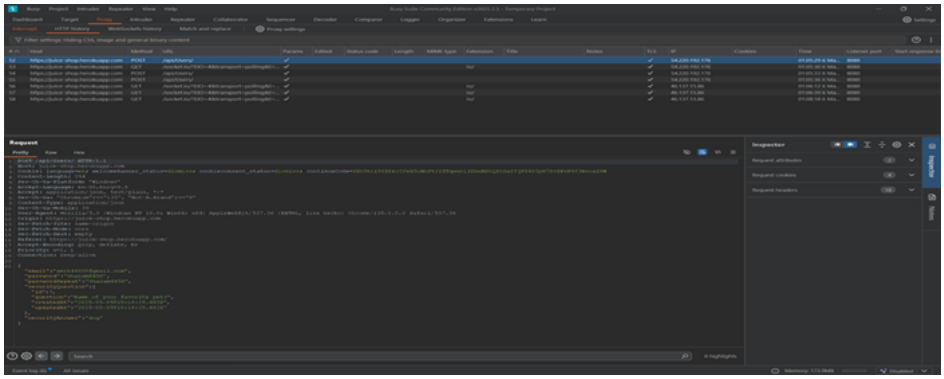

Figure 60: Enter Caption

Here we have captured the post request of creation of the new account.

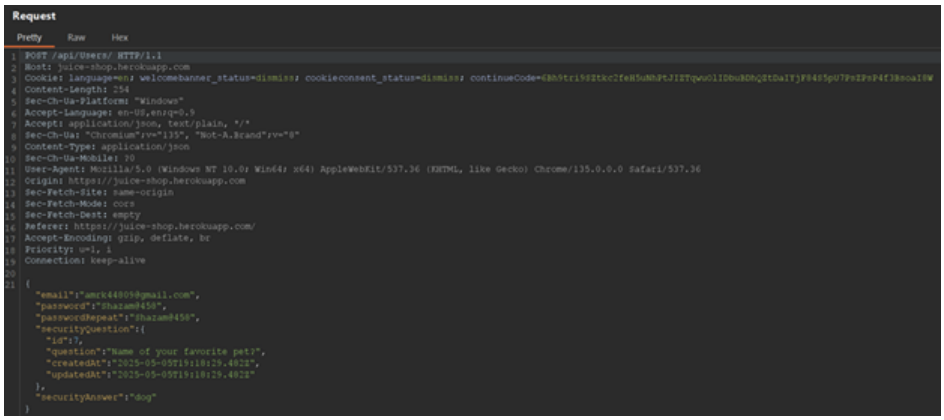

Figure 61: Enter Caption

We have forwarded the request to get the response and appears to us a very important header named role and categorized me as a customer.

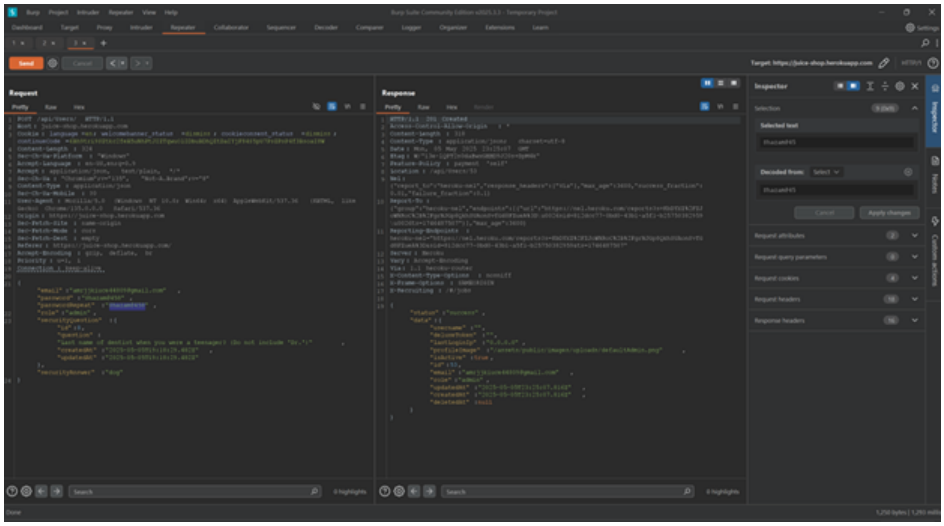

Figure 62: Enter Caption

So, after trying many scenarios it appeared to me that if i have changed the role into admin and forwarding the request i can login by the new account which i have created but as an admin.


```
{
    "status" :"success" ,
    "data" :{
        "username" :"",
        "deluxeToken" :"",
        "lastLoginIp" :"0.0.0.0" ,
        "profileImage" :"/assets/public/images/uploads/defaultAdmin.png" ,
        "isActive" :true ,
        "id" :53,
        "email" :"amrjjkiuce44809@gmail.com" ,
        "role" :"admin" ,
        "updatedAt" :"2025-05-05T23:25:07.816Z" ,
        "createdAt" :"2025-05-05T23:25:07.816Z" ,
        "deletedAt" :null
    }
}
```


Figure 63: Enter Caption

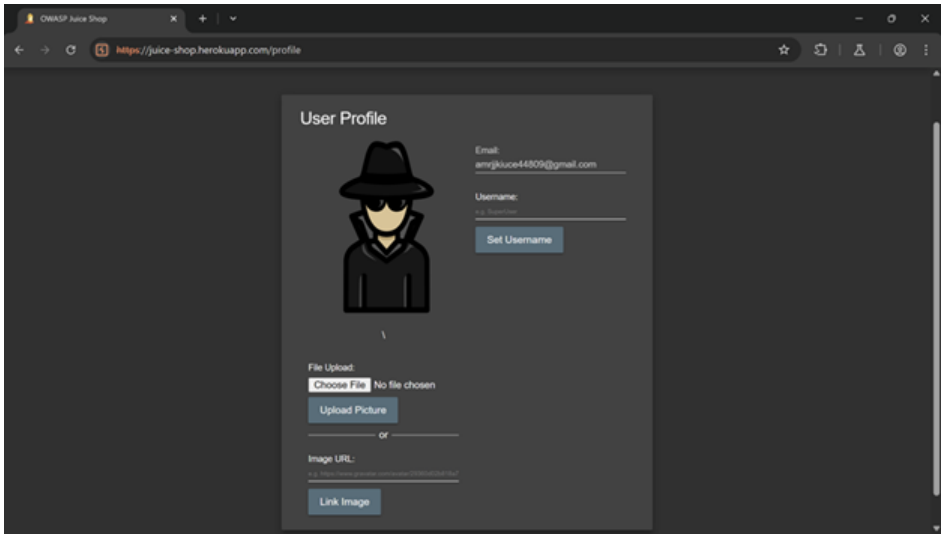

Figure 64: Enter Caption

After i had loginedin as an admin account, i have run into the admin panel to see all of the so, we are done with jim now we have begun with bender account

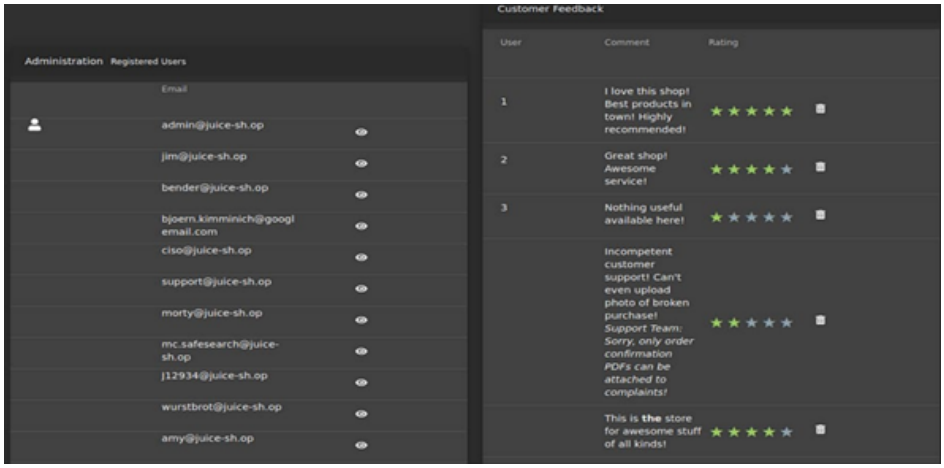

Figure 65: Enter Caption

accounts and the data

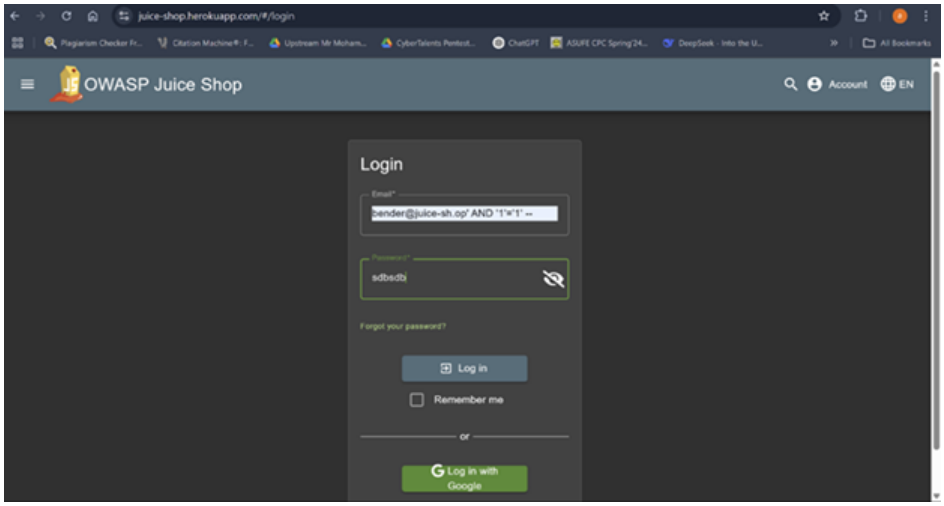

Figure 66: Enter Caption

12.1 Constructing the SQL Injection:

- Modified the entry in the email input field to include an SQL injection payload: `bender@juice-sh.op' AND '1'='1' --`.
- This payload attempts to manipulate the SQL query behind the login form by:
  - Closing off the email string (with a single quote),
  - Adding a condition that always evaluates to true (`AND '1'='1'`),
  - Then commenting out the remainder of the SQL command with `--`.

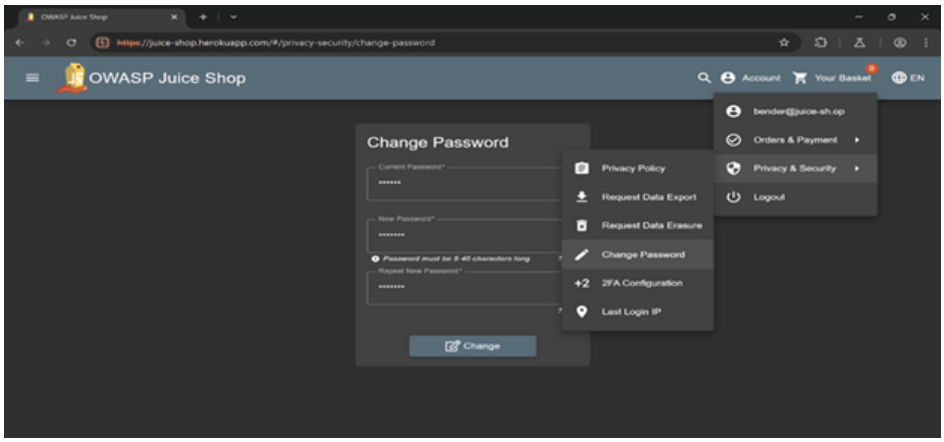

Figure 67: Enter Caption

After that we have open the change password page and typed any password in the current password and the new password with `orange1` and also the repeat.

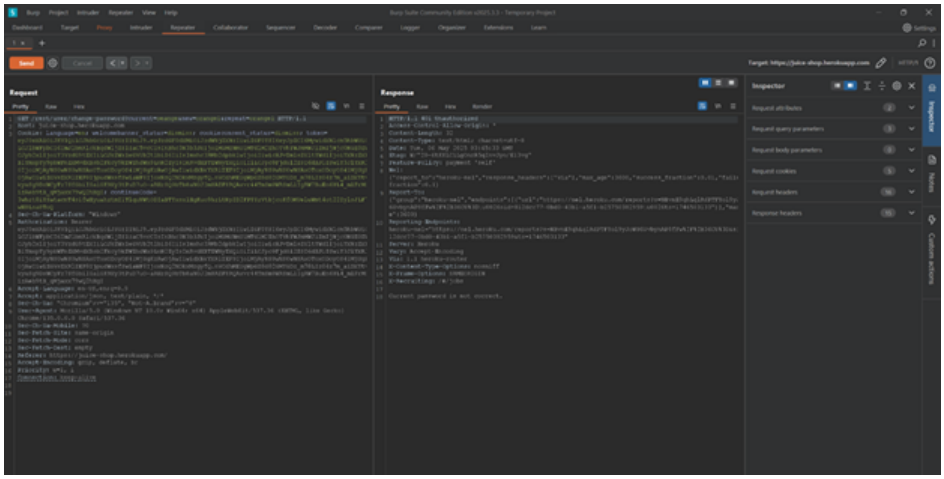

Figure 68: Enter Caption

In this step we have run the burpsuite and activated the proxy to capture the request of changing passwords and as we can see i am unable and unauthorized to change the password.

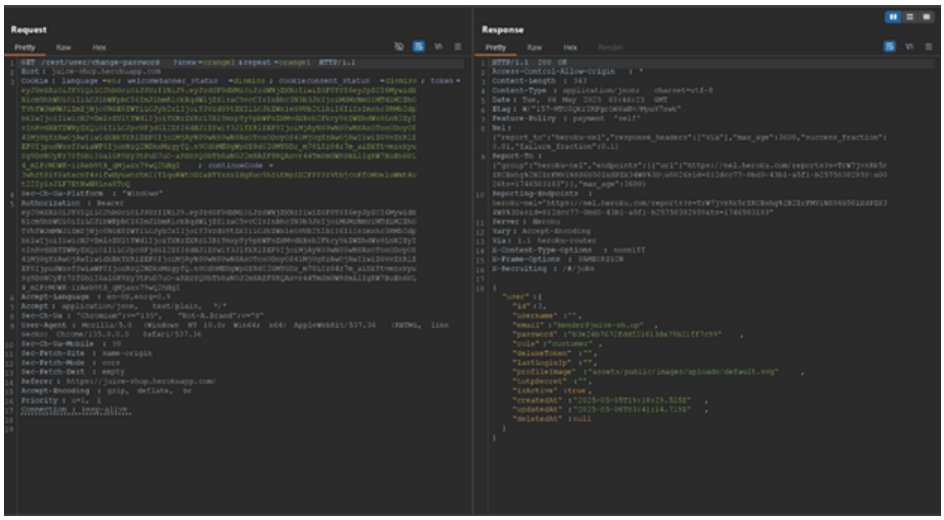

Figure 69: Enter Caption

After trying many several scenarios we have gotten that if we have deleted the current password parameter, i can gain the ability to change the password to the new password i want without even knowing the actual current password.

13 Exploitation 13: Database Schema

(Amr Khaled Mohamed - 120220236)

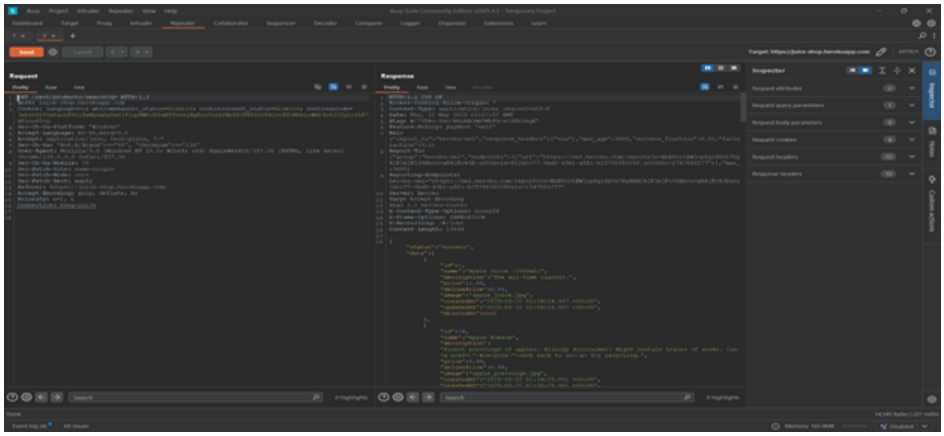

Figure 70: Enter Caption

In the normal, if we typed any product we want in the search bar we will get the product but usually if we typed (‘) after the product like `orange’` it will get us error of no product found we use this technique as the payload starts with the (‘)

After that, we captured the request by using the burpsuite and as u see we get the all the initial data that would appear to us in the website of all products

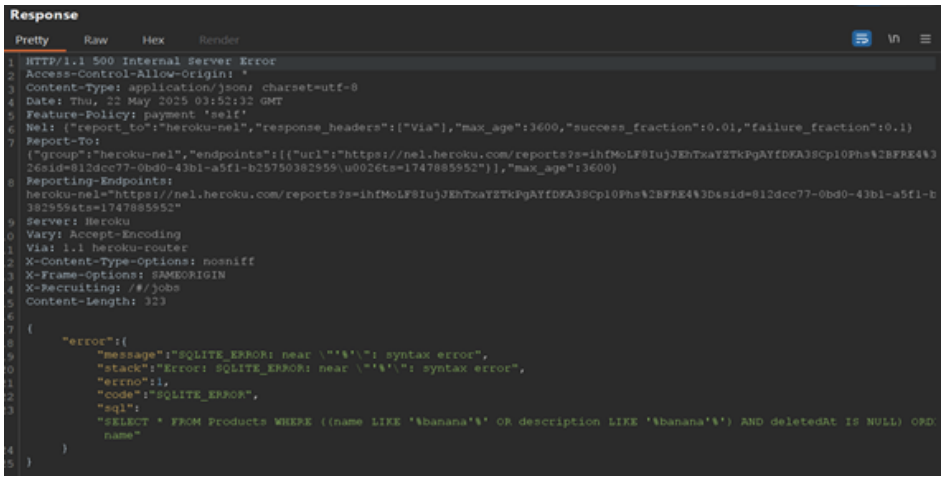

Figure 71: Enter Caption

And as you see, this is error that appeared to me after we typed `banana’`.

But the useful part about the error that it tell us that there are an sql error and also tell us the type of error itself, which is the SQLite error

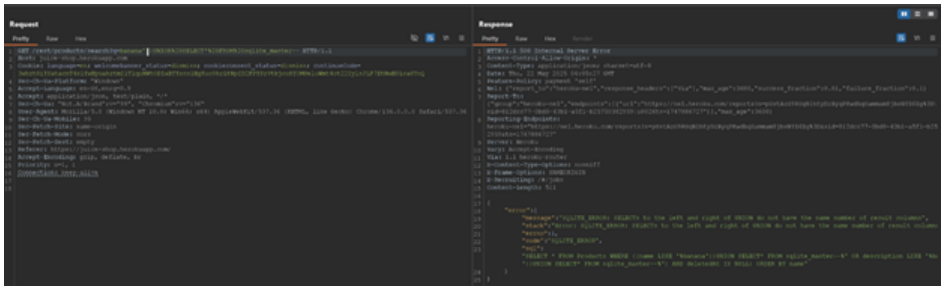

Figure 72: Enter Caption

After that we have use the SQL UNION operator to inject a query that will reveal the database schema. Because the product search results are displayed in a structured format, and it is known that product entries typically have multiple attributes (columns), the SQL injection payload must account for the correct number of columns.

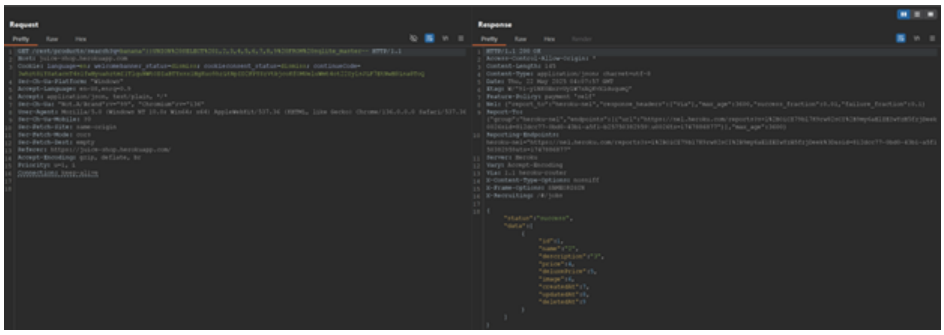

Figure 73: Enter Caption

And also we want to extract the data from the `sql_master` which is a special table in the SQLlite that stores the schema os the data base.

And it appears in the error message that the union don't have the same number of columns

Figure 74: Enter Caption

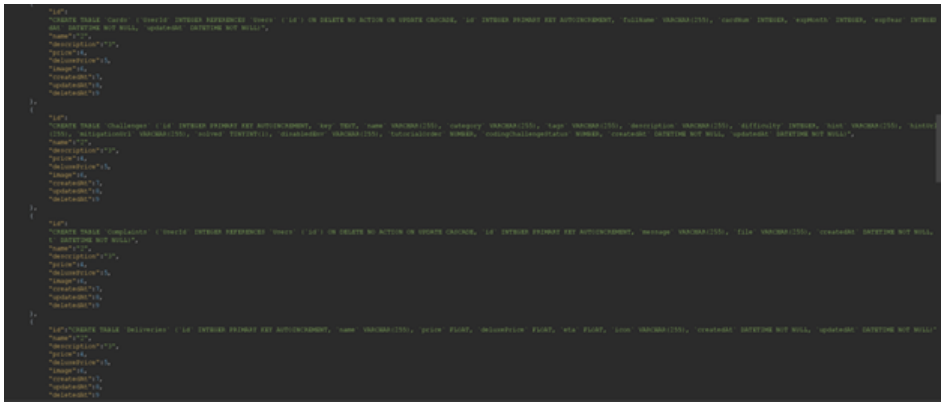

Figure 75: Enter Caption

To align with the expected number of columns in the original query and extract schema information from the `sqlite_schema` table. We have tried values and see if it align or not so we tried values from 1 until we have reached 9

Now we have gotten the table of the product and u can see the nine columns

13.1 Simulate the Login of a Non-existent but Temporary User

In this vulnerability we want to simulate the login of a non-existent user temporarily by using SQL injection

So we first get into the login page and typed any word in the email and password and captured the request of the login

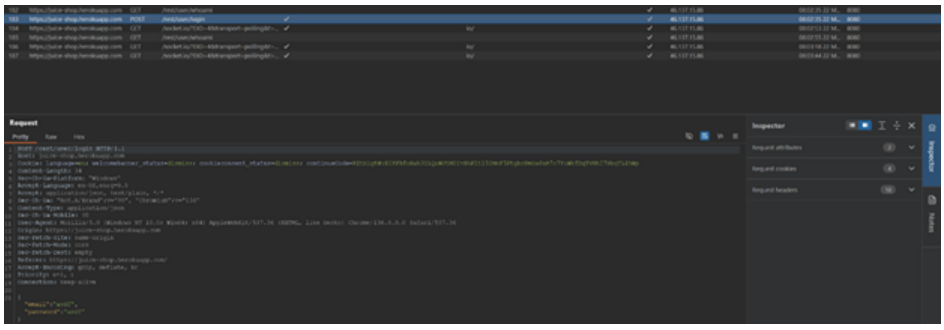

Figure 76: Enter Caption

And as we said above we wanted to test whether is there a sql error or not so we tried to type ’ in the email as u see in the screenshot and it appears that there is an sql error

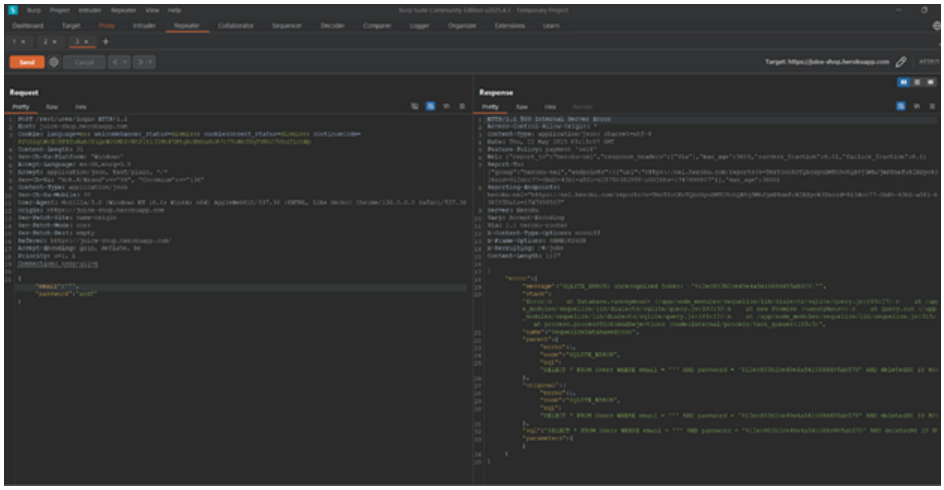

Figure 77: Enter Caption

We will use the tables that we have fotten from our the db schema and we will use the `users` table especially cause this what matters with us now as we want to by the login page

And we will write a payload but we will change the info whatever it is and you can put in the email whatever the email you want

So this is the payload after i changed it

[language=SQL] ' UNION SELECT * FROM (SELECT 1000 as 'id', '' as 'username', 'acc0unt4nt@juice-sh.op' as 'email', 'asdfasdf' as 'password', 'accounting' as 'role', '' as 'deluxeToken', '127.0.0.1' as 'lastLoginIp', 'default.svg' as 'profileImage', '' as 'totpSecret', 1 as 'isActive', '2020-08-30 11:12:13.456 +00:00' as 'createdAt', '2020-08-30 11:12:13.456 +00:00' as 'updatedAt', null as 'deletedAt')—

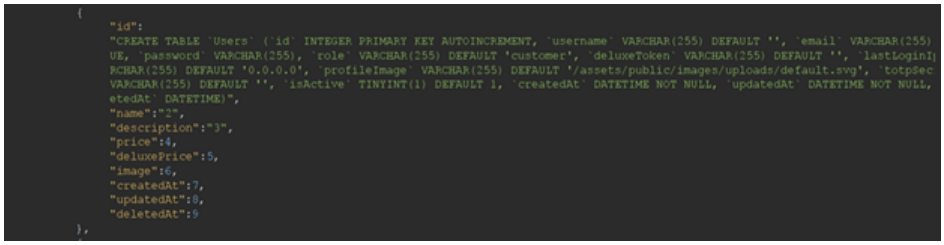

Figure 78: Enter Caption

We observe that the server responded with a JWT token, meaning that he accepted our connection

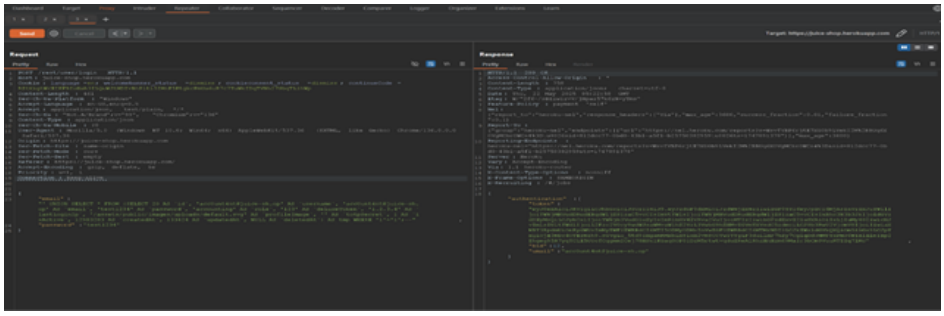

Figure 79: Enter Caption

14 Exploitation Report 14: Privilege Escalation via Role Manipulation

(Amr Khaled Mohamed - 120220236)

Vulnerability Type: Privilege Escalation through Insecure Role Assignment
Target Application: Juice Shop

14.1 Introduction

This report documents a serious privilege escalation vulnerability discovered in the Juice Shop web application. The flaw lies in how the application handles user role assignment during registration. By manipulating the role parameter in the HTTP request, it was possible to create an account with administrative privileges, thus bypassing intended access controls.

14.2 Problem Description

The application relies on a hidden `role` field during account registration, which is sent from the client-side. This is a critical mistake, as any malicious actor using a proxy tool like Burp Suite can intercept and modify this field to escalate privileges (e.g., from `customer` to `admin`). Since the backend fails to validate or restrict the allowed role values, it grants the unauthorized role to the new account.

14.3 Steps to Reproduce

1. **Creating a New Account:** A new user account was initiated through the normal registration form on the Juice Shop.

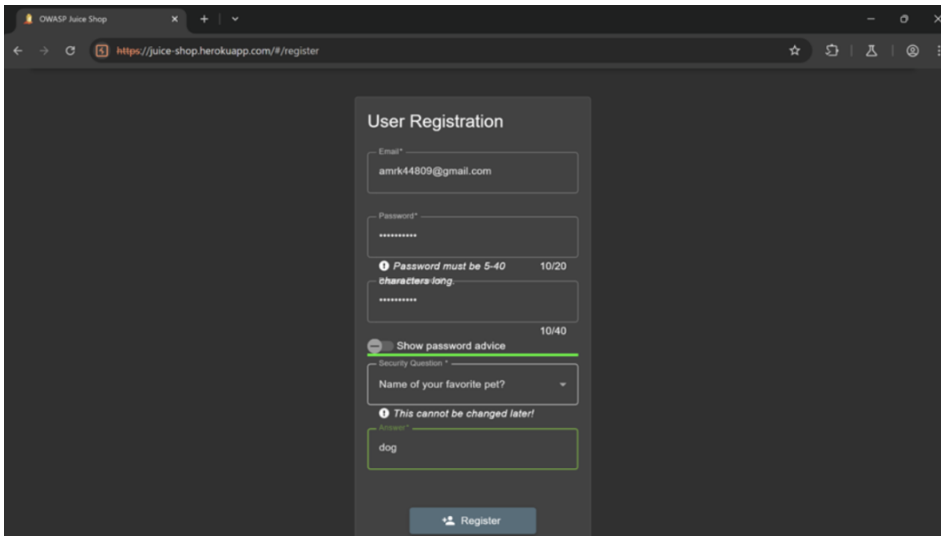

Figure 80: Enter Caption

2. **Intercepting the Request:** Using Burp Suite, the POST request triggered by clicking the Register button was intercepted before being sent to the server.

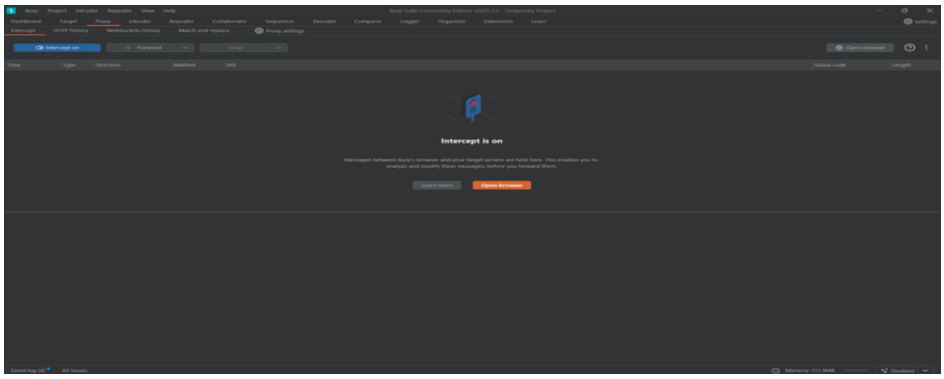

Figure 81: Enter Caption

3. **Modifying the Role:** In the captured request, the field `"role": "customer"` was found in the body or headers. This value was changed to `"role": "admin"`.

Figure 82: Enter Caption

4. **Forwarding the Modified Request:** The altered request was forwarded to the server. The server accepted the request and created the account with admin privileges.

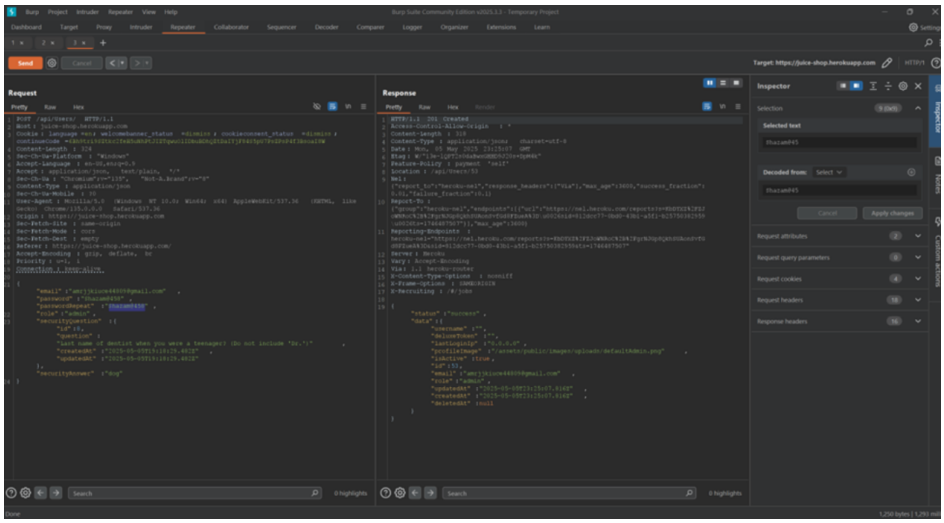

Figure 83: Enter Caption

5. **Logging in as Admin:** Logging in with the newly created account successfully granted access to the admin panel, confirming the account now had administrator rights. Inside the admin panel, access to sensitive data and control features was confirmed.

Figure 84: Enter Caption

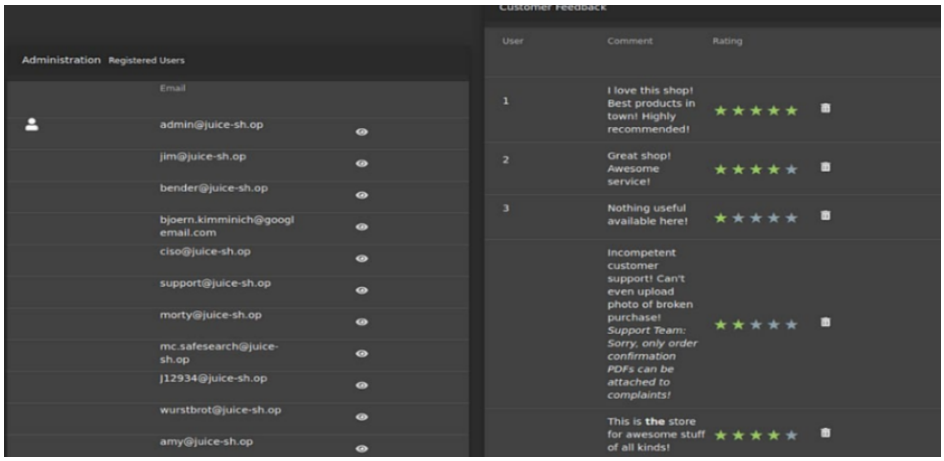

Figure 85: Enter Caption

14.4 Impact

This vulnerability allows any user with basic knowledge of web proxies to:

- Gain unauthorized administrator access.
- View or modify user data and accounts.
- Access sensitive and critical parts of the web application.
- Perform actions that could compromise the integrity and confidentiality of the system.

Severity: *Critical*

14.5 Recommendations

To mitigate this vulnerability, the following actions are recommended:

- Do not allow role assignment from the client side.
- Handle role logic exclusively on the server side, based on predefined rules.
- Validate all incoming data and ignore or sanitize any unexpected fields.
- Implement strict access control policies and enforce them at the server level.
- Perform regular security testing to detect and fix privilege escalation issues.

15 Exploitation Report 15: Unauthorized Password Reset via SQL Injection and Parameter Removal

(Amr Khaled Mohamed - 120220236)

Target Application: Juice Shop
Vulnerability Type: SQL Injection + Insecure Direct Object Reference (IDOR)

15.1 Introduction

This report outlines a compound vulnerability discovered in the Juice Shop application, combining SQL Injection with improper validation of parameters during the password reset process. Through this flaw, an attacker can reset the password of an arbitrary user (in this case, Bender) without knowing the original password.

15.2 Problem Description

The application is vulnerable to:

- SQL Injection during login, allowing unauthorized access to a known user account.
- Improper backend validation during the password change process, where removing the `currentPassword` field bypasses the authentication requirement.

Together, these issues allow a complete compromise of user accounts.

15.3 Steps to Reproduce

1. **SQL Injection to Log in as Bender:**
   - During login, the email field was modified with the following payload: `bender@juice-sh.op' AND '1'='1' --`
   - This payload breaks the SQL query and appends a condition that always evaluates to true.
   - As a result, the attacker is logged in as user Bender.

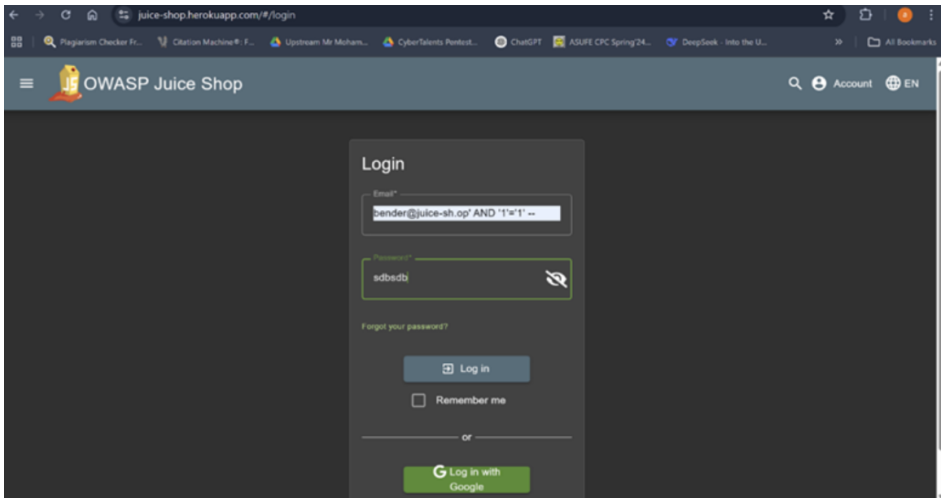

Figure 86: Enter Caption

2. **Attempt to Change Password Normally:**
   - The attacker navigated to the change password page.
   - Attempted to change the password using:
     - Current password: any random input
     - New password: `orange1`
     - Repeat password: `orange1`
   - The server responded with an unauthorized error, since the actual current password is unknown.

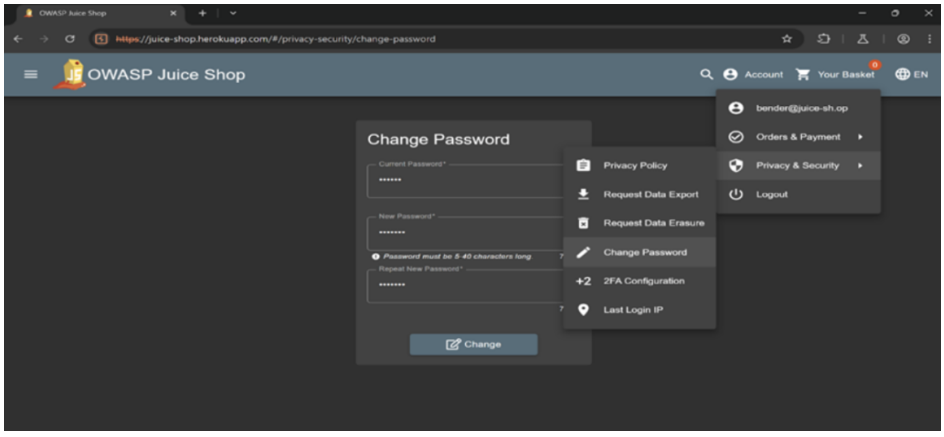

Figure 87: Enter Caption

3. **Bypassing the Current Password Requirement:**
   - Using Burp Suite, the POST request for password change was intercepted.
   - The parameter `currentPassword` was deleted from the request.
   - The modified request was forwarded to the server.
4. **Successful Password Reset:**
   - The server processed the request without verifying the current password.
   - The password was successfully changed to `orange1`, allowing the attacker full control of Bender's account.

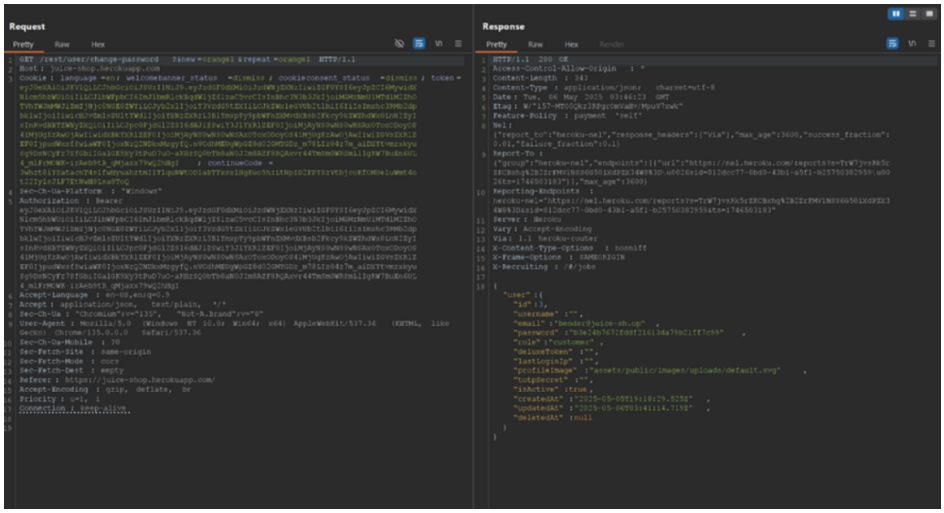

Figure 88: Enter Caption

15.4 Impact

This vulnerability allows attackers to:

- Fully compromise user accounts without needing to know their passwords.
- Escalate their privileges or sabotage target users.
- Permanently lock out users from their accounts by changing their passwords.

Severity: *Critical*

15.5 Recommendations

To prevent this attack, the following security practices should be implemented:

- Sanitize and parameterize all SQL queries to eliminate SQL Injection vulnerabilities.
- Ensure that the `currentPassword` is mandatory and server-validated during any password change.
- Avoid trusting client-side fields. Server logic should enforce all authorization checks independently.
- Implement logging and anomaly detection for unusual account activities.

16 Exploitation Report 16: Revealing Database Schema via SQL Injection in Search Function

(Amr Khaled Mohamed - 120220236)

Target Application: Juice Shop
Vulnerability Type: SQL Injection – Database Schema Enumeration

16.1 Introduction

This report explains how an SQL injection vulnerability was discovered in the search bar of the Juice Shop web application. By strategically injecting SQL payloads and analyzing the resulting error messages, it was possible to enumerate the database schema and identify the structure of backend tables, including the number of columns and their metadata.

16.2 Problem Description

Web applications often allow users to search products using input fields. If the input is not sanitized properly, attackers can inject raw SQL code into the backend query. In this case, the search bar is vulnerable, and improper handling of special characters (like ') exposes the system to SQL injection.

16.3 Steps to Reproduce

1. **Initial Search Input:**
   - A product (e.g., `banana`) was searched using the search bar.
   - To test for SQL injection, an apostrophe was added: `banana'`

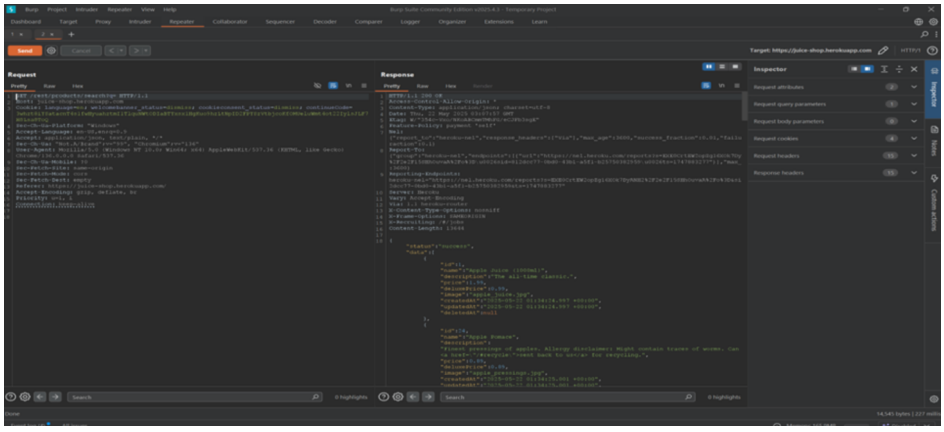

Figure 89: Enter Caption

2. **Observing the Error:**
   - An error was returned: `\SQLite Error: near 'banana'"`, indicating that the backend uses SQLite and does not handle the apostrophe properly.

- This confirms that the input is being executed as part of an SQL query.

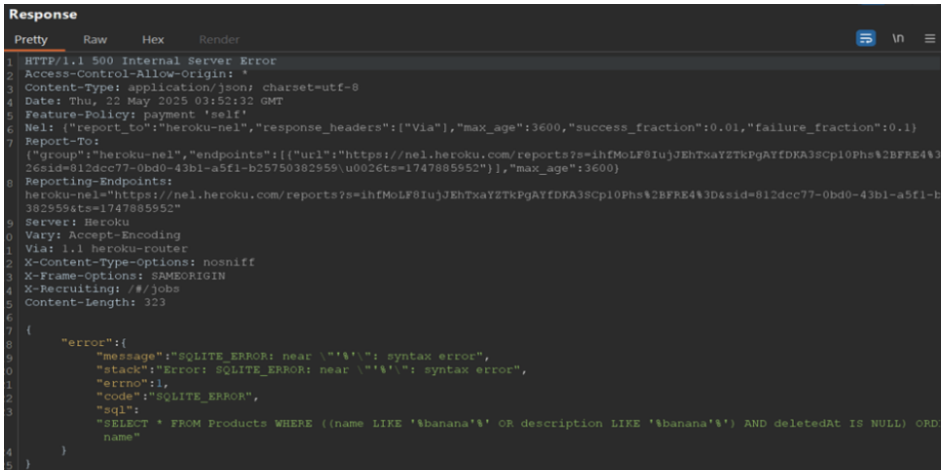

Figure 90: Enter Caption

3. **Intercepting the Request:**

   - Using Burp Suite, the request generated by the search bar was captured.
   - This confirmed that the search input is directly included in the SQL query sent to the server.

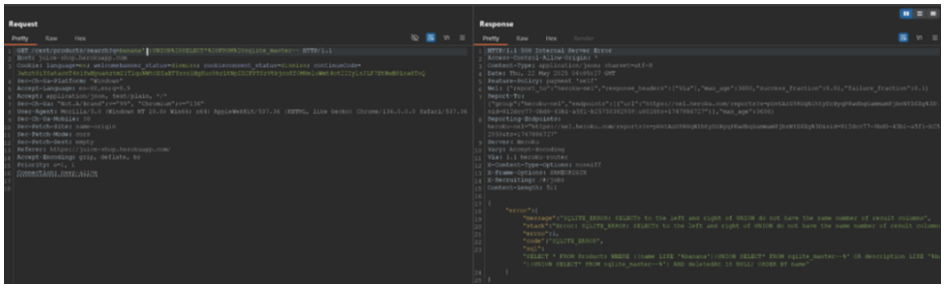

Figure 91: Enter Caption

4. **Crafting a UNION-based SQL Injection:**

   - A UNION SELECT payload was tested to determine the number of columns.
   - The goal was to match the number of columns in the original query so the UNION statement executes without error.
   - Payload format: `' UNION SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL--`
   - The number of `NULL` values was incremented from 1 to 9 until the query executed without error.

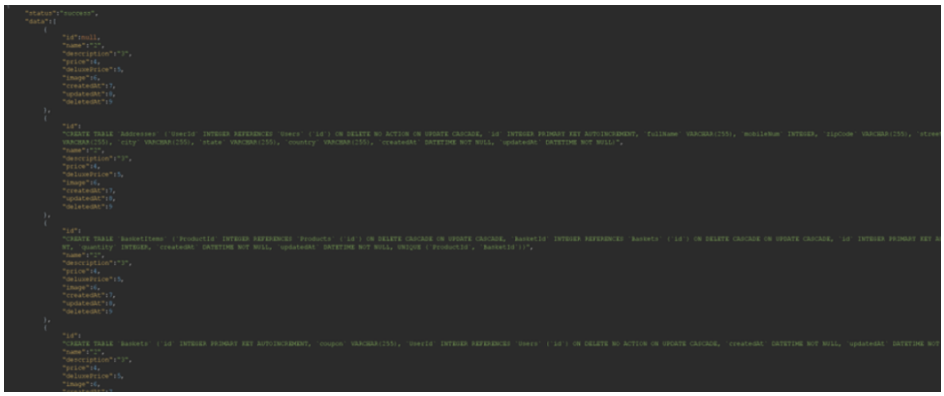

Figure 92: Enter Caption

5. **Extracting the Schema from `sqlite_master`:**

   - SQLite stores schema information in a system table called `sqlite_master` (or `sqlite_schema` in newer versions).
   - A modified payload was used to retrieve table names and structures:
     `' UNION SELECT type, name, tbl_name, rootpage, sql, NULL, NULL, NULL, NULL FROM sqlite_master-`
   - This payload successfully returned the database schema, including the structure of the `products` table.

16.4 Impact

With this vulnerability, an attacker can:

- Fully enumerate database tables and their structures.
- Plan targeted data extraction attacks (e.g., extracting user data, passwords).
- Gain deep insights into the backend logic and potentially escalate attacks further.

Severity: *High*

5. Recommendations

To remediate this vulnerability:

- Sanitize all user inputs using prepared statements (parameterized queries).
- Disable detailed SQL error messages from being shown to end-users.
- Implement input validation and reject suspicious characters or patterns in the search field.
- Use Web Application Firewalls (WAFs) to detect and block SQL injection attempts.

17 Exploitation Report 17: Simulating a Login for a Non-Existent User via SQL Injection

(Amr Khaled Mohamed - 120220236)

Target Application: Juice Shop
Vulnerability Type: SQL Injection – Authentication Bypass via Manual Payload Injection

17.1 Introduction

This report documents a vulnerability in the Juice Shop application that allows an attacker to simulate the login of a non-existent user by crafting a fake identity and injecting it directly into the login SQL query. This is possible due to the lack of proper input sanitization and SQL query parameterization on the login form.

17.2 Problem Description

Authentication should be strictly based on verified, existing user credentials stored in the database. However, due to an SQL injection vulnerability, an attacker can manually construct a virtual user row, inject it into the SQL query, and trick the application into generating a valid JWT token for a user that doesn't actually exist in the database.

17.3 Steps to Reproduce

1. **Identifying the SQL Injection Point:**
   - Navigate to the login page.
   - Enter any placeholder values for email and password (e.g., `test / test`).
   - Use Burp Suite to intercept the POST request for login.
   - Modify the email field to contain a single apostrophe (`'`) to test for errors.
   - The application responds with an SQL syntax error, confirming that the input is injected directly into the SQL query.

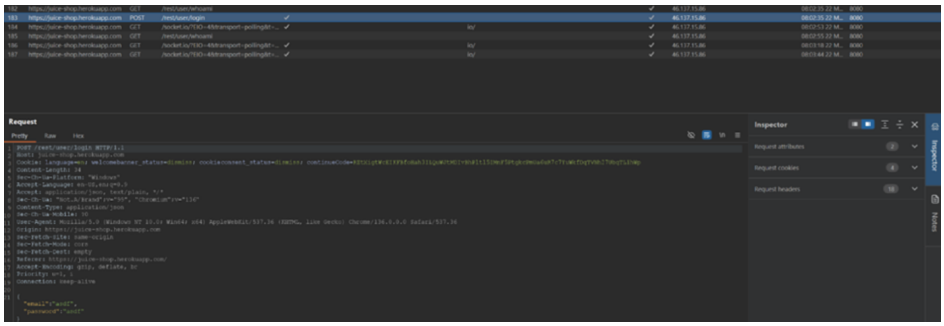

Figure 93: Enter Caption

2. **Crafting the SQL Injection Payload:**
   - From previous exploits (see Exploitation 13), the schema of the `Users` table was obtained.
   - Using this knowledge, a payload was crafted to simulate a login by injecting a complete user record manually.

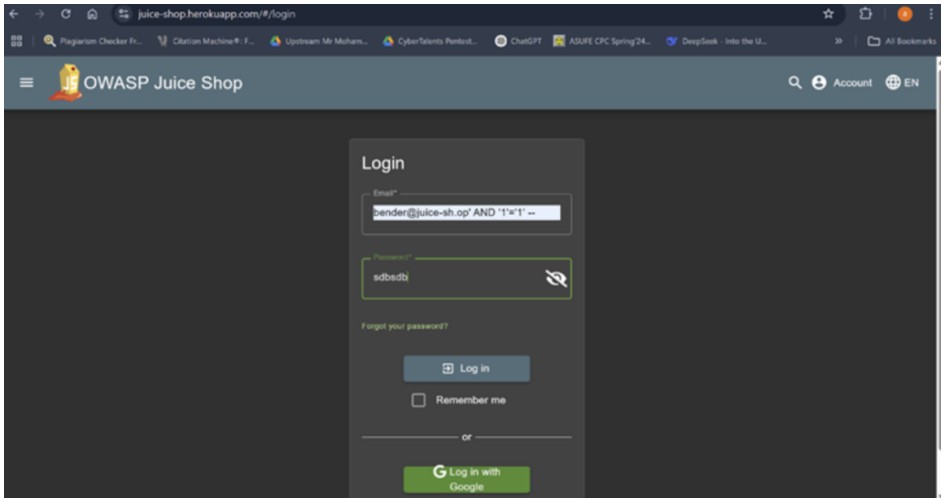

Figure 94: Enter Caption

Injected Email Field:

```
' UNION SELECT * FROM (
  SELECT
    1000 as 'id',
    '' as 'username',
    'acc0unt4nt@juice-sh.op' as 'email',
    'asdfasdf' as 'password',
    'accounting' as 'role',
    '' as 'deluxeToken',
    '127.0.0.1' as 'lastLoginIp',
    'default.svg' as 'profileImage',
    '' as 'totpSecret',
    1 as 'isActive',
    '2020-08-30 11:12:13.456 +00:00' as 'createdAt',
    '2020-08-30 11:12:13.456 +00:00' as 'updatedAt',
    null as 'deletedAt'
)--
```

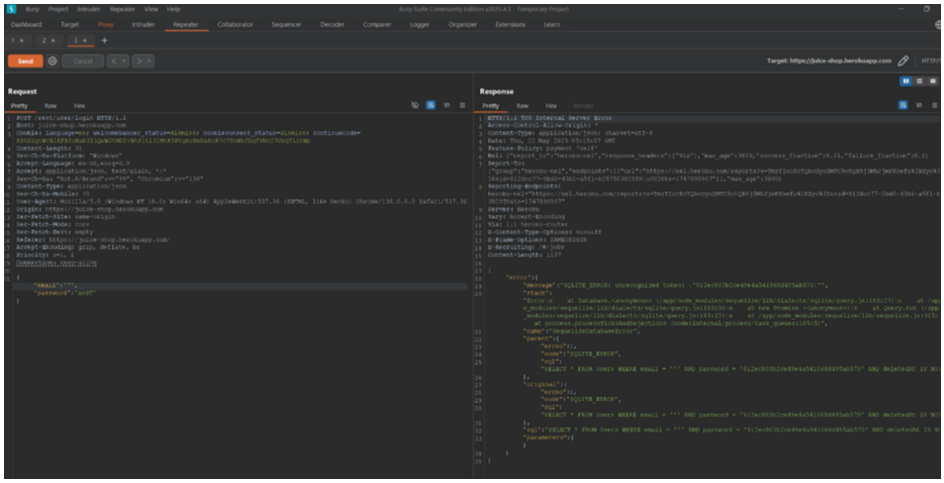

Figure 95: Enter Caption

3. **Sending the Request:**

   - The payload is injected in the email field, while the password can be anything (ignored due to injection).
   - The server executes the UNION query and returns a valid JWT token.

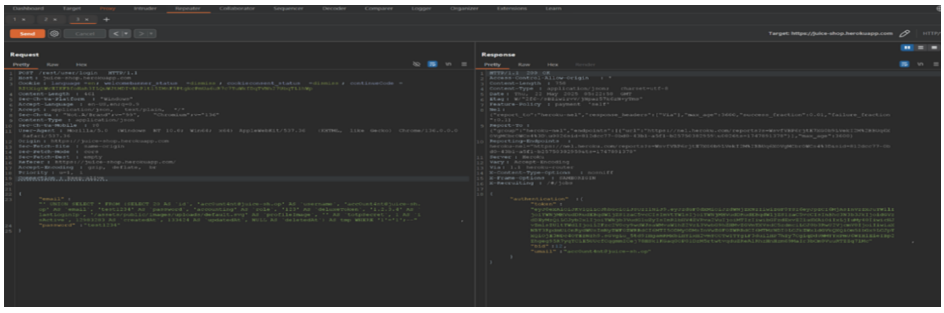

Figure 96: Enter Caption

17.4 Impact

This vulnerability allows attackers to:

- Bypass login authentication and simulate any user identity, including non-existent or privileged users.
- Gain access to resources and dashboards under a forged identity.
- Perform privileged actions if a role like `admin` is assigned in the fake payload.
- Completely undermine the trust and security model of the authentication system.

Severity: *Critical*

Recommendations

To fix this issue:

- Always use parameterized queries (prepared statements) to prevent SQL injection.
- Validate and sanitize all user inputs.
- Never rely on user input to build SQL queries directly.
- Validate issued JWT tokens against real users in the database, ensuring the user exists and is active.

18 Exploitation Report 18: NoSQL Denial of Service (DoS) via Sleep Injection

(Amr Khaled Mohamed - 120220236)

Target Application: Juice Shop
Vulnerability Type: NoSQL Injection – Denial of Service (DoS)

18.1 Introduction

This report documents a NoSQL Denial of Service (DoS) vulnerability discovered in the Juice Shop web application. By exploiting a vulnerable endpoint, it was possible to inject a payload that triggered a sleep operation on the server, effectively causing it to become unresponsive for a specified duration. This simulates a successful application-layer DoS attack.

18.2 Problem Description

Unlike traditional SQL databases, NoSQL databases like MongoDB can be vulnerable to JavaScript-based injections due to their flexible query syntax. If the application fails to properly sanitize inputs, attackers can inject code that runs inside database queries — including commands like `sleep()` which delay processing and tie up server resources.

18.3 Methodology and Steps

1. **Identifying the Vulnerable Endpoint:**
   - During earlier assessments, the following endpoint was identified for handling product reviews: `/rest/products/[productId]/reviews`
   - This endpoint accepts review modifications, making it a good candidate for injection testing.

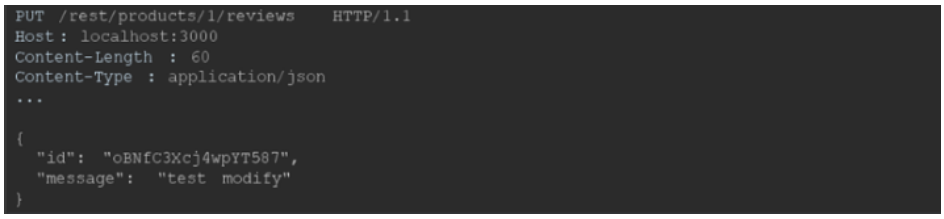

Figure 97: Enter Caption

2. **Intercepting the Review Modification Request:**
   - Navigated to any product page and submitted a review.
   - Used Burp Suite to intercept the request sent when submitting or editing a review.

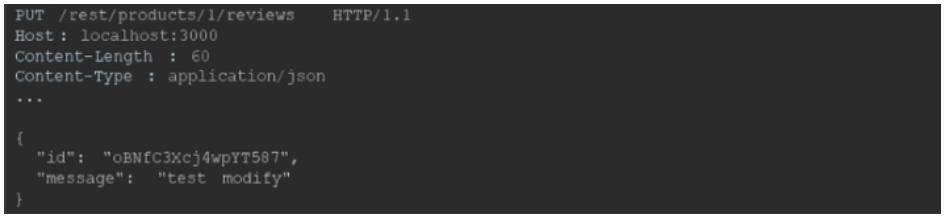

Figure 98: Enter Caption

3. **Analyzing the Request:**
   - The intercepted request contained parameters such as `reviewId`, `message`, and possibly `productId`.
   - These values were found to be vulnerable to injection.
4. **Crafting a Sleep-Based DoS Payload:**
   - Aiming to delay server response using a `sleep()` function.

     Attempt 1: Sleep in JSON body (did not work)
     Injected: `"productId": { "$where": "sleep(20000)" }`
     Result: Server returned an error or ignored the injection.

     Attempt 2: Injecting in the URL
     Manually crafted the URL: `http://localhost:3000/rest/products/sleep(20000)/reviews`

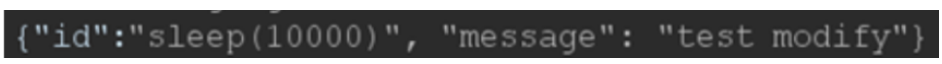

Figure 99: Enter Caption

5. **Executing the Payload:**

- Accessed the crafted URL directly.
- The browser froze for a significant duration.
- The server did not respond immediately, confirming a processing delay.

```
PUT /rest/products/sleep(20000)/reviews HTTP/1.1
Host : localhost:3000
```

Figure 100: Enter Caption

6. **Confirming the DoS Behavior:**
   - Observed server behavior: requests became unresponsive for approximately 20 seconds.
   - This behavior indicated a successful Denial of Service condition, demonstrating that injected `sleep()` functions are executed server-side.

18.4 Impact

This vulnerability allows attackers to:

- Exhaust server resources by forcing long-running operations.
- Disrupt service availability for legitimate users.
- Chain into broader DoS attacks or even RCE if further injection is possible.

Severity: *High*

18.5 Recommendations

To prevent this vulnerability:

- Validate and sanitize all user inputs, especially in API endpoints.
- Disable JavaScript-based query execution in NoSQL queries (e.g., MongoDB `$where`).
- Implement rate limiting and request throttling to reduce DoS impact.
- Use application-level firewalls to block suspicious patterns like `sleep()`.

19 Exploitation Report 19: HTTP Header-Based Reflected XSS via True-Client-IP

(Amr Khaled Mohamed - 120220236)
Target Application: Juice Shop
Vulnerability Type: Reflected Cross-Site Scripting (XSS) via HTTP Header

19.1 Introduction

This report documents a reflected Cross-Site Scripting (XSS) vulnerability discovered in the Juice Shop application. By injecting a malicious payload into the `True-Client-IP` HTTP header, it was possible to trigger JavaScript execution within the admin panel. This exploit leverages the fact that certain HTTP headers are reflected directly into the UI without proper sanitization or encoding.

19.2 Problem Description

The `True-Client-IP` header is used to log the last login IP address of users. This value is reflected in the admin interface at the following path:

```
http://localhost:3000/#/privacy-security/last-login-ip
```

However, the application fails to sanitize this input before displaying it, allowing arbitrary HTML and JavaScript to be injected into the admin panel — a textbook example of reflected XSS via an HTTP header.

19.3 Steps to Reproduce

1. **Identify the Target:**
    - Hypothesis: The last login IP shown in the admin panel may be pulled from an HTTP header such as `True-Client-IP`.
    - Navigated to the panel: `http://localhost:3000/#/privacy-security/last-login-ip`

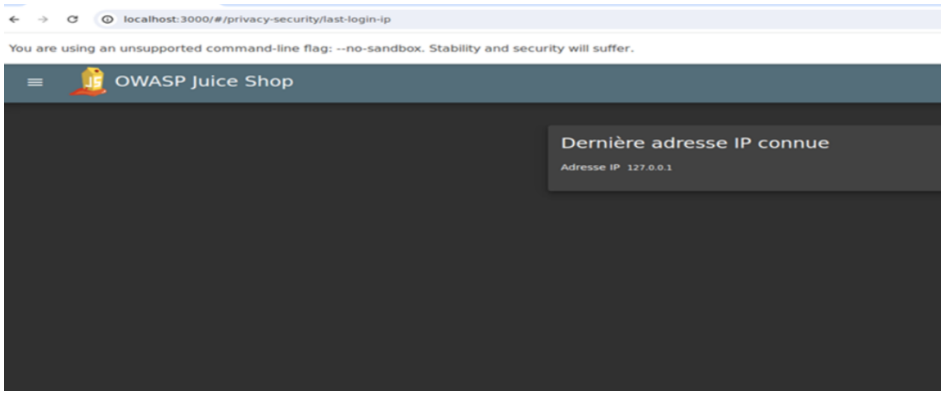

Figure 101: Enter Caption

2. **Intercept the Request:**

   - Performed a logout action while intercepting the request using Burp Suite.
   - Observed request headers to find modifiable IP-related fields.

Figure 102: Enter Caption

3. **Locate the Vulnerable Header:**

   - Found the header: `True-Client-IP: 127.0.0.1`
   - Modified this value to test if it is reflected in the admin panel.

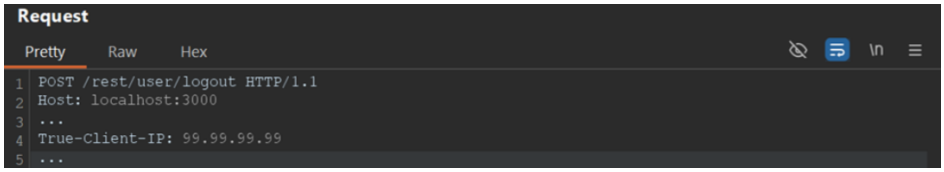

Figure 103: Enter Caption

4. **Verify Reflection:**

   - Changed the header to: `True-Client-IP: 123.123.123.123`
   - Logged in again as admin and visited the IP log page.
   - Confirmed that the new IP was displayed.

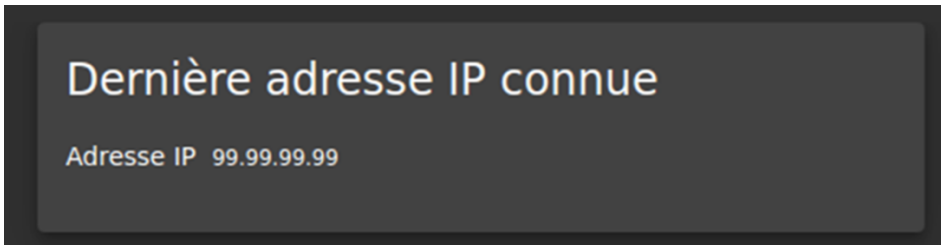

Figure 104: Enter Caption

5. **Craft and Inject the XSS Payload:**

   - Payload: `True-Client-IP: <iframe src="javascript:alert('xss')">`
   - Injected this header using Burp Suite during the logout request.

6. **Observe the Result:**
    - Logged in again as admin and opened the Last Login IP panel.
    - The payload executed and displayed an alert box: `alert('xss')`

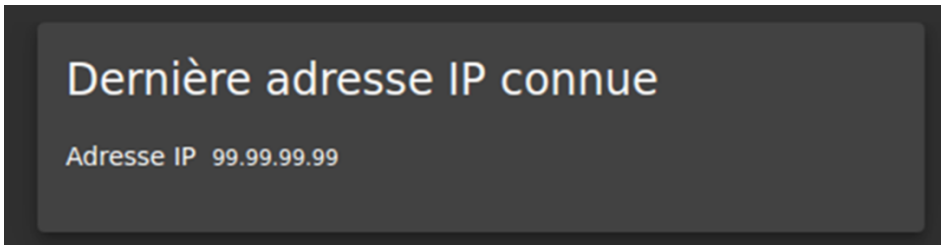

Figure 105: Enter Caption

19.4 Impact

This vulnerability allows an attacker to:

- Execute arbitrary JavaScript in the context of the admin user's browser.
- Steal session tokens, perform unauthorized actions, or inject malicious scripts into the admin interface.
- Target users with phishing attacks, deface the interface, or exfiltrate sensitive data.

Severity: *Critical*

19.5 Recommendations

To mitigate this vulnerability:

- Sanitize and encode all user input, especially data coming from HTTP headers.
- Use Content Security Policy (CSP) to reduce the risk of script execution.
- Escape HTML special characters (`<`, `>`, `"`, `'`, `&`) before rendering user-controlled content.
- Avoid reflecting header values into pages unless strictly necessary and safely handled.

20 Exploitation Report 20: Email Leak via Unauthenticated Access and JSONP Abuse

(Amr Khaled Mohamed - 120220236)
Target Application: Juice Shop
Vulnerability Type: Information Disclosure + JSONP Exploitation

20.1 Introduction

This report documents an email leak vulnerability in the Juice Shop application, caused by improper access control on the `/rest/user/whoami` endpoint. The vulnerability is worsened by the availability of JSONP-style responses, allowing attackers to potentially steal user data via cross-site injection.

20.2 Problem Description

The `/rest/user/whoami` endpoint returns the current user's data, including sensitive details such as the email address. However, the endpoint:

- Does not require authentication.
- Supports a `callback` parameter, allowing JSONP-style responses, which can lead to Cross-Site Data Exfiltration.

20.3 Steps to Reproduce

1. **Identify the Endpoint:**
   - Observed that `/rest/user/whoami` is frequently called when checking login state or viewing user data.
2. **Unauthorized Access:**
   - Used Burp Suite to intercept a legitimate request.
   - Removed the JWT token from the headers and re-sent the request.
   - Server still responded with full user data (including email).
3. **JSONP Injection:**
   - Modified the URL to: `/rest/user/whoami?callback=test`
   - Response returned: `test({ "email": "user@example.com", ... });`

20.4 Impact

This vulnerability allows attackers to:

- Access sensitive user information (email, profile info) without authentication.
- Exploit the callback mechanism to exfiltrate data via malicious websites.
- Launch cross-domain data theft attacks using JSONP.

Severity: *Critical*

20.5 Recommendations

- Require authentication headers for all sensitive endpoints like `/rest/user/whoami`.
- Remove JSONP support unless absolutely necessary.
- Return strict CORS headers and disallow cross-origin access for sensitive endpoints.
- Validate query parameters and avoid blindly wrapping responses in user-defined callbacks.

21 Exploitation Report 21: NoSQL Infiltration

(Amr Khaled Mohamed - 120220236)

21.1 Step 1: Identifying Potential Injection Points

The challenge involves interacting with order-related functionalities, specifically the orders page. The following steps were taken to identify injection points:

- **Access Order History:** Navigate to `http://localhost:3000/#/order-history` (because it's an obvious page for orders) and click on one of the orders.
- **Intercept Requests:** Use Burp Suite to intercept and analyze the requests being made when viewing an order.

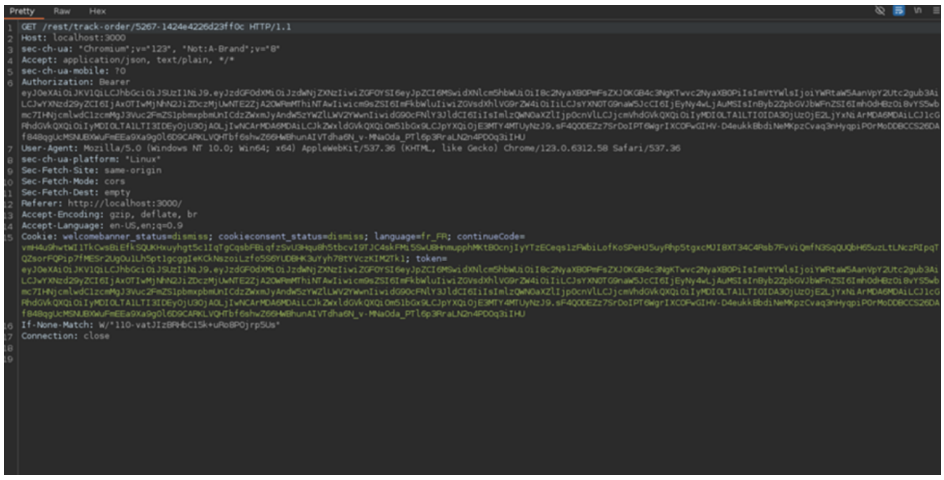

Figure 106: Enter Caption

21.2 Step 2: Analyzing the API Endpoint

By analyzing the intercepted requests, we identified the following endpoint used to fetch order details:

- **Endpoint:** `/rest/track-order/[id]`

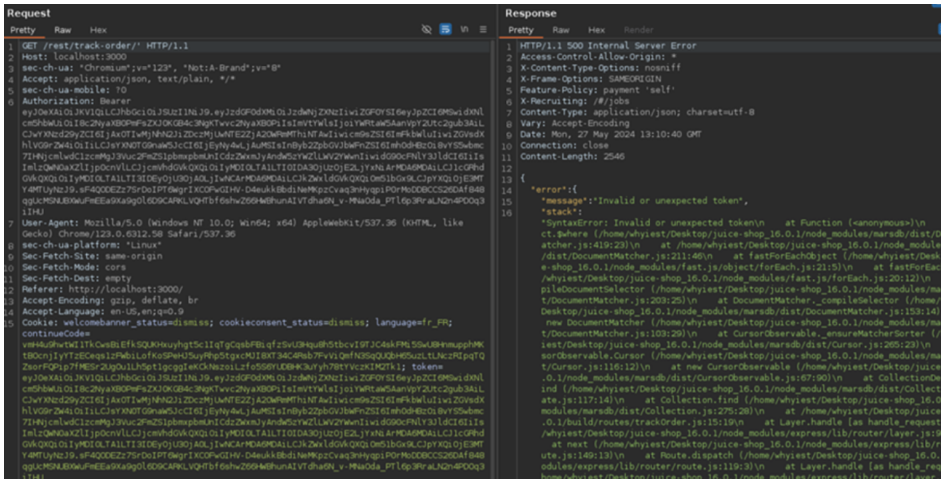

Figure 107: Enter Caption

21.3 Step 3: Testing for NoSQL Injection (Not working yet)

To test for NoSQL injection, we attempted basic payloads:

- **Payload:** `/rest/track-order/'`
- **Result:** Received a 500 Internal Server Error, indicating a potential injection point.
- **Payload:** `/rest/track-order/''`
- **Result:** Received an `Unexpected string` error instead of `Unexpected token`, suggesting that the server interprets our input as part of a NoSQL query.

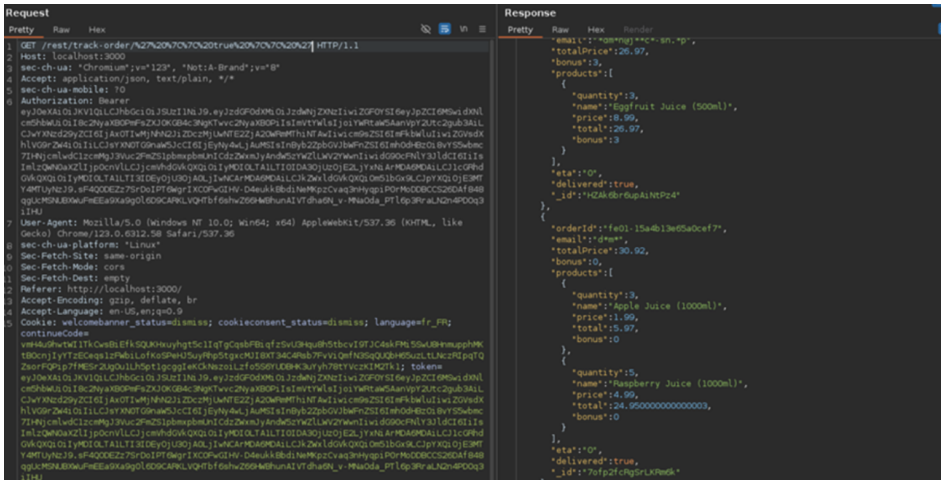

Figure 108: Enter Caption

22 Exploitation Report 22: Access Log (Sensitive Data Exposure)

(Abdelrahman Ashraf - 120220292)

In this demonstration, the hacker targeted the "Access Log" challenge from OWASP Juice Shop, which involves finding and accessing a hidden access log file on the server—a common case of sensitive data exposure.

22.1 Step-by-Step Breakdown

22.1.1 Challenge Goal

Gain access to any access log file on the Juice Shop server.

22.1.2 Tool Used

The attacker used **ffuf** (Fuzz Faster U Fool), a directory brute-forcing tool, to discover hidden folders and files on the website that aren't linked in the frontend.

22.1.3 Approach

- A common wordlist (`common.txt`) was used to brute-force potential folder and file names.
- Initially, many false positives were encountered, as the app redirected all invalid paths to the "All Products" page with the same content length.
- A content-length filter was applied using `-fs 1925` to ignore those common redirects and focus on unusual results.

22.1.4 Discovery

- Hidden folders like `/ftp/`, `/support/`, and `/legal/` were found.
- Under `/support/`, another brute-force revealed a `/logs/` subdirectory.
- Inside `/support/logs/`, a real access log file was found and downloaded.

22.1.5 Outcome

- The access log was successfully accessed, completing the challenge.
- This simulates a real-world vulnerability where sensitive internal files can be accessed due to improper server configuration or lack of access control.

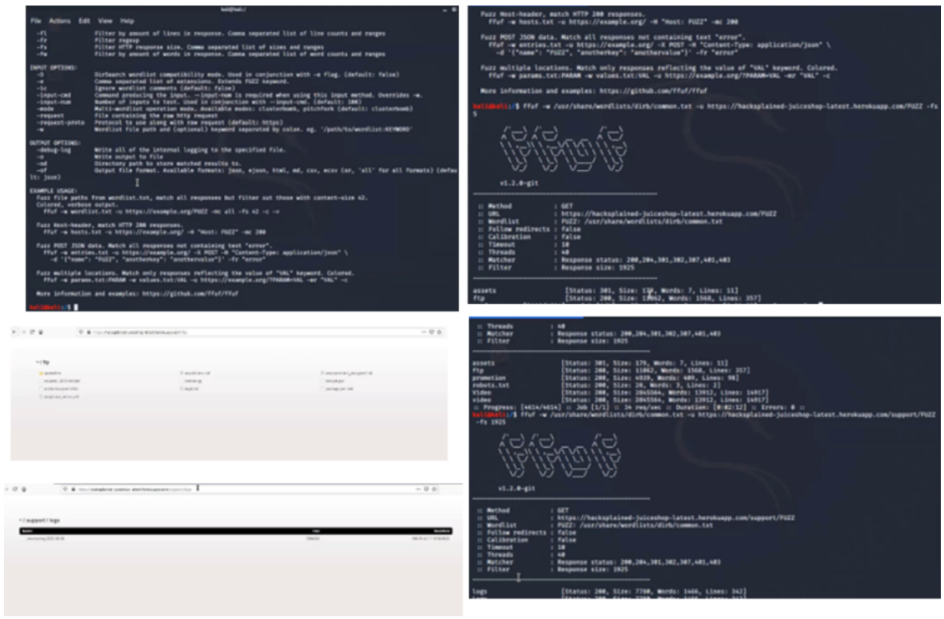

Figure 109: Enter Caption

23 Exploitation 23: Login Amy (Brute Force with a Pattern)

(Abdelrahman Ashraf - 120220292)

23.1 Attack Method – Brute Force with a Pattern

23.1.1 Password Guessing

The attacker guessed that Amy followed a specific password pattern:

- Format: `[Uppercase Letter][Digit][Lowercase Letter]` + series of dots
- Example: `A1a.........`

23.1.2 Burp Suite Setup

- Used Burp Suite's **Intruder** tool in `Cluster Bomb` mode to try all combinations of:
  - Uppercase letters: A–Z
  - Digits: 0–9
  - Lowercase letters: a–z
- The payload attempted thousands of combinations, but the Burp Community Edition proved too slow.

23.1.3 Python Script

- A custom Python script was developed using asynchronous HTTP requests to accelerate the brute-force process.
- The script iterated over all possible 3-character patterns followed by dots.
- Eventually, the Juice Shop backend marked the challenge as **solved**, even though the exact password was not revealed.

23.2 Result – Exploit Success

- The attacker successfully completed the challenge by triggering a login as the user `Amy`.
- However, the actual password was not retrieved, indicating a follow-up task is needed to enhance the script and extract the credentials precisely.

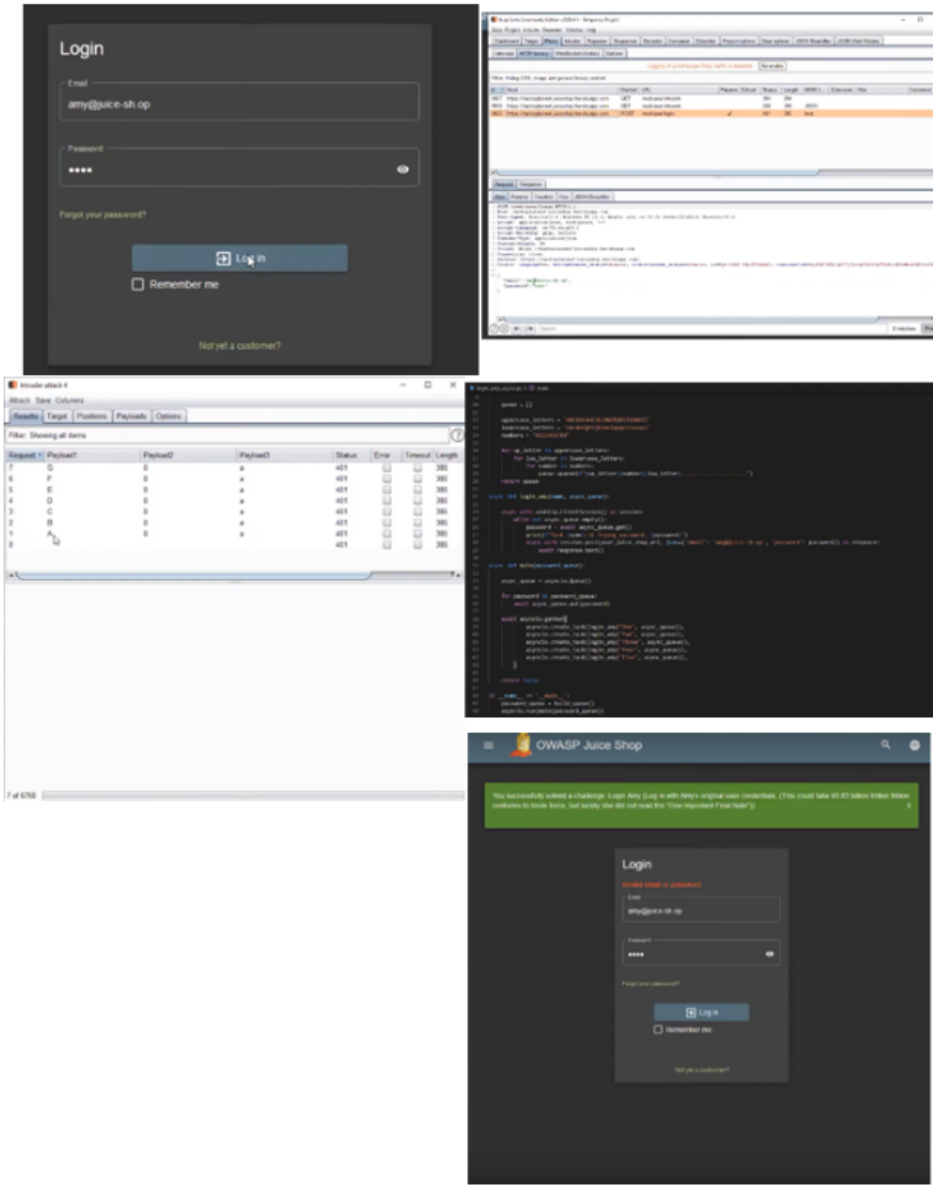

Figure 110: Enter Caption

24 Exploitation 24: Expired Coupon (Improper Input Validation)

(Abdelrahman Ashraf - 120220292)

24.1 Challenge Goal

Redeem a coupon code from a past campaign that has already expired.

24.2 Category

Improper Input Validation

24.3 Attack Method – JavaScript Debugging & Time Manipulation

24.3.1 Inspecting the Source

- Opened the browser's Developer Tools and searched the `main-es2015.js` JavaScript file.
- Found a list of old campaign coupons with specific valid timestamps.

24.3.2 Analyzing Logic

- Located the `applyCoupon()` function in the source code.
- Set a breakpoint and examined how the coupon is validated.
- Found that validation checks if the current client date matches the coupon's `valid-on` date.

24.3.3 Bypassing the Check

- Manually changed the system date and time on the computer to match the expected timestamp (e.g., March 7, 2019).
- Re-submitted the coupon code, triggering successful validation.

24.4 Result – Exploit Success

- The coupon was accepted, giving a 75% discount.
- The challenge was marked as solved in Juice Shop.

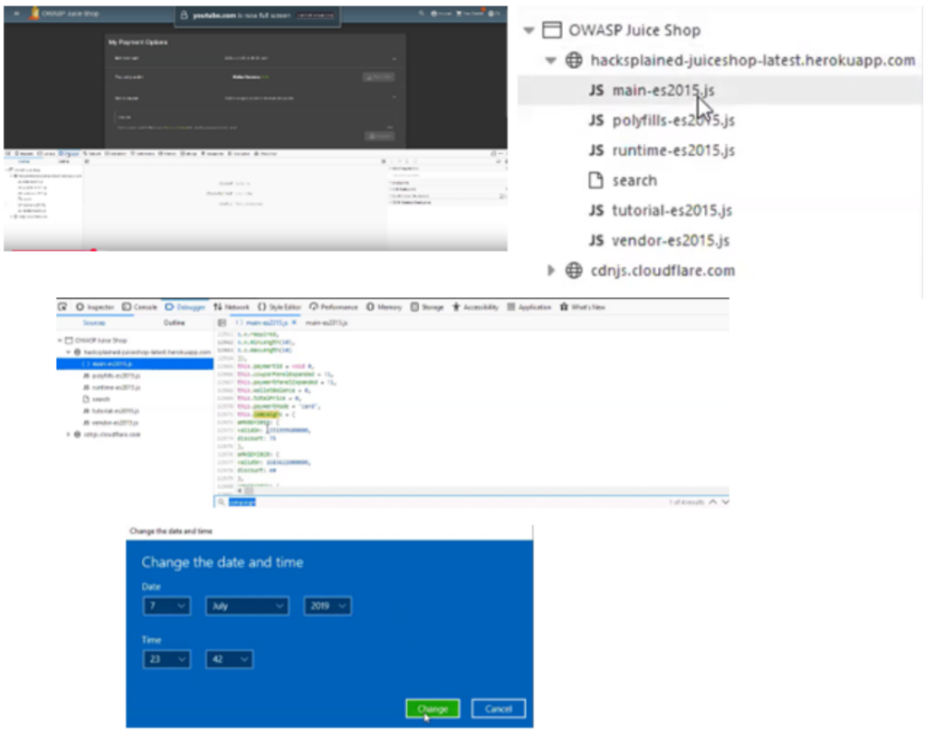

Figure 111: Enter Caption

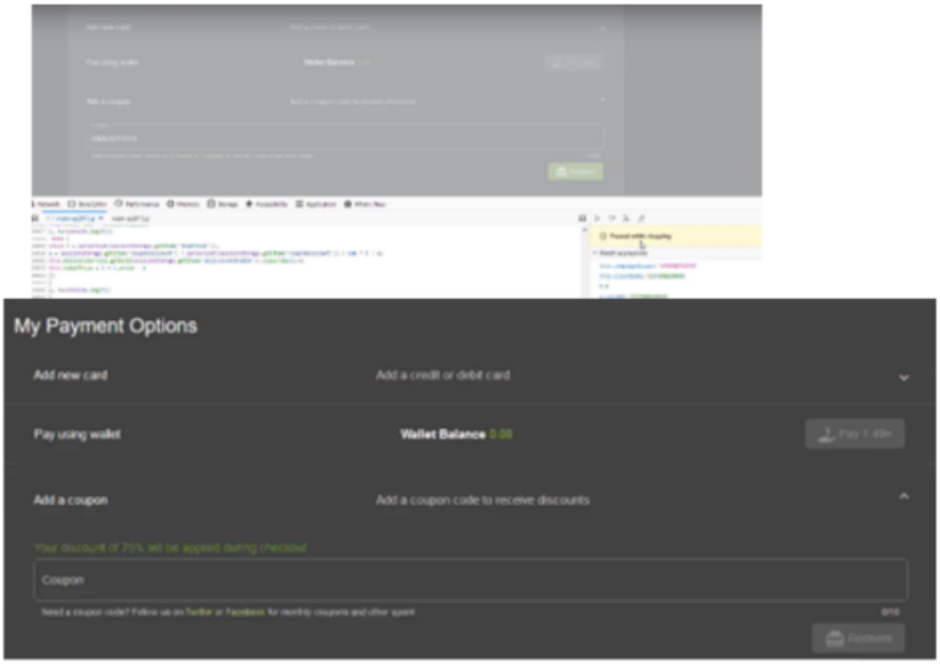

Figure 112: Enter Caption

25 Exploitation 25: Product Tampering (Broken Access Control)

(Abdelrahman Ashraf - 120220292)

25.1 Challenge Goal

Modify the product description of `OSAFT (OS SSL Advanced Forensic Tool)` to redirect the link to `https://owasp.slack.com`.

25.2 Category

Broken Access Control

25.3 Attack Method – Unauthorized PUT Request to Product API

25.3.1 Step 1: Identify Product and Endpoint

- Located the product ID (`id=9`) by searching: `/rest/products/search?q=forensic`.
- Switched to a more flexible endpoint: `/api/Products/9`.

25.3.2 Step 2: Modify Product via API

- Attempted to send a PUT request to update the product.
- Initially failed due to missing headers and excess fields in the request body.
- Simplified the JSON payload to include only the `description` field.
- Added the header: `Content-Type: application/json` to fix the format.

25.3.3 Step 3: Inject New Link

- Replaced the `href` inside the product description with the target URL: `https://owasp.slack.com`.
- Ensured proper escaping of quotes and characters in the HTML string.
- Sent the modified payload successfully.

25.4 Result – Exploit Success

- The product link was successfully tampered with and now redirects to `https://owasp.slack.com`.
- The challenge was marked as solved in Juice Shop.

25.5 Key Lesson

APIs should strictly validate which users can access or modify resources. Allowing unauthenticated users to send PUT or POST requests to sensitive endpoints is a serious security flaw and a textbook example of broken access control.

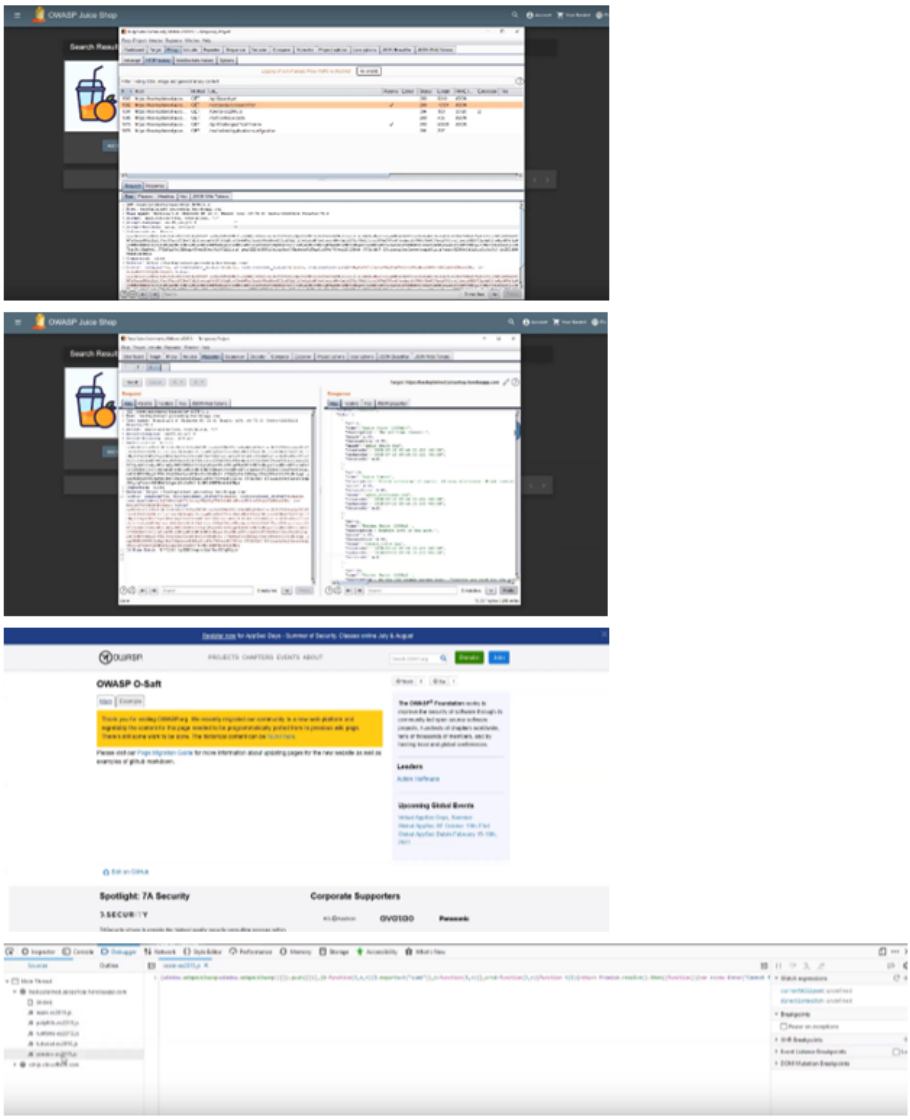

Figure 113: Enter Caption

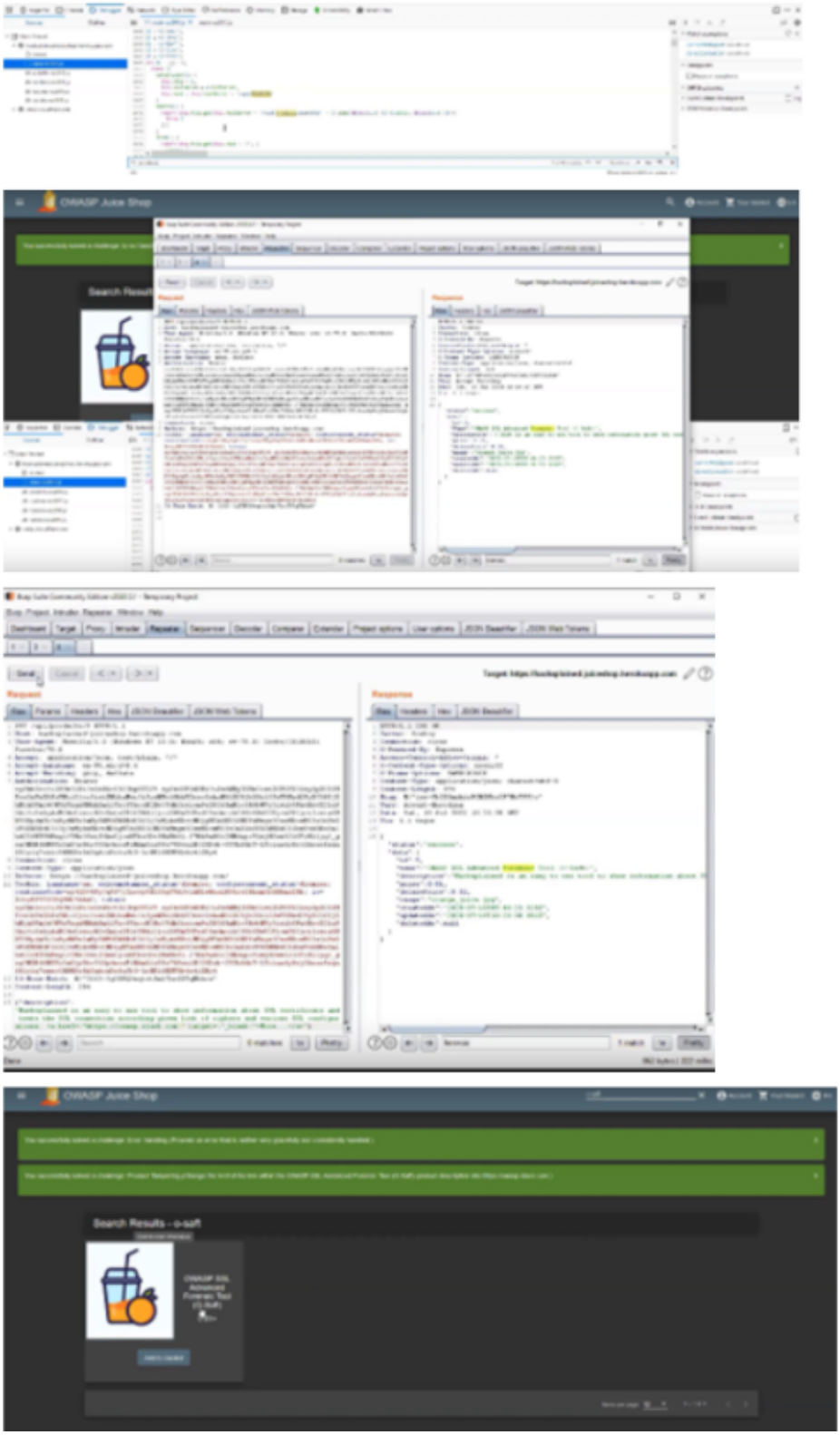

Figure 114: Enter Caption

26 Exploitation 26: Reset Jim's Password (Broken Authentication)

(Abdelrahman Ashraf - 120220292)

26.1 Challenge Goal

Reset the password for the user `Jim` by correctly answering his security question.

26.2 Category

Broken Authentication

26.3 Attack Method – Brute-Forcing Security Question

26.3.1 Step 1: Discover Jim's Email

- Extracted from a product review: `jim@juice-sh.op`

26.3.2 Step 2: Initiate Password Reset

- Used the "Forgot Password" feature to trigger the security question.
- Displayed question: *"Your eldest sibling's middle name"*

26.3.3 Step 3: Brute Force with Burp Suite

- No brute-force protection was in place.
- Used Burp Suite Intruder to send 100 common names as potential answers.
- Found the correct answer: `Samuel`

26.3.4 Step 4: Password Reset

- Submitted a new password using the correct answer.
- Gained full access to Jim's account.

26.4 Result – Exploit Success

- Password reset completed.
- Account takeover successful.
- Challenge marked as solved in Juice Shop.

26.5 Key Lesson

Security questions can be a major vulnerability if:

- The question is too common.
- The answer is easily guessable or public.
- There is no brute-force protection on the input.

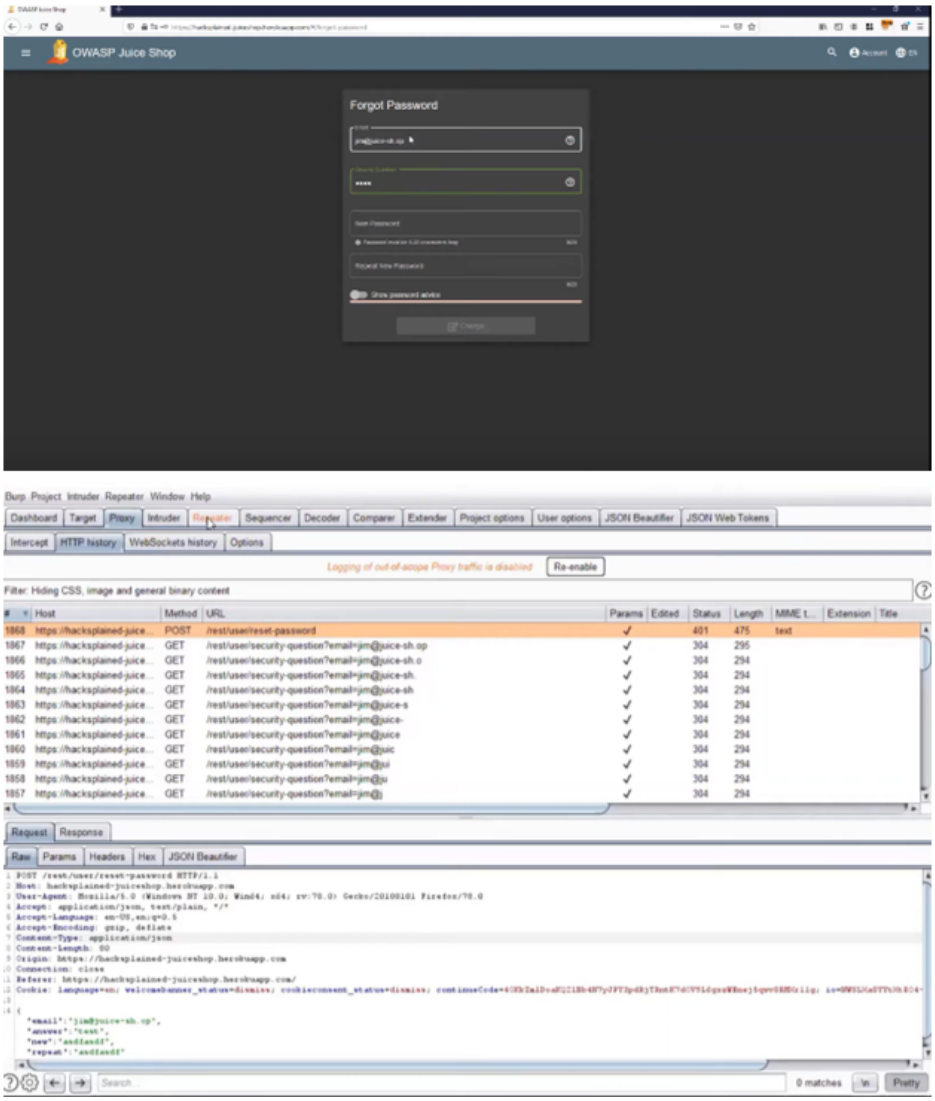

Figure 115: Enter Caption

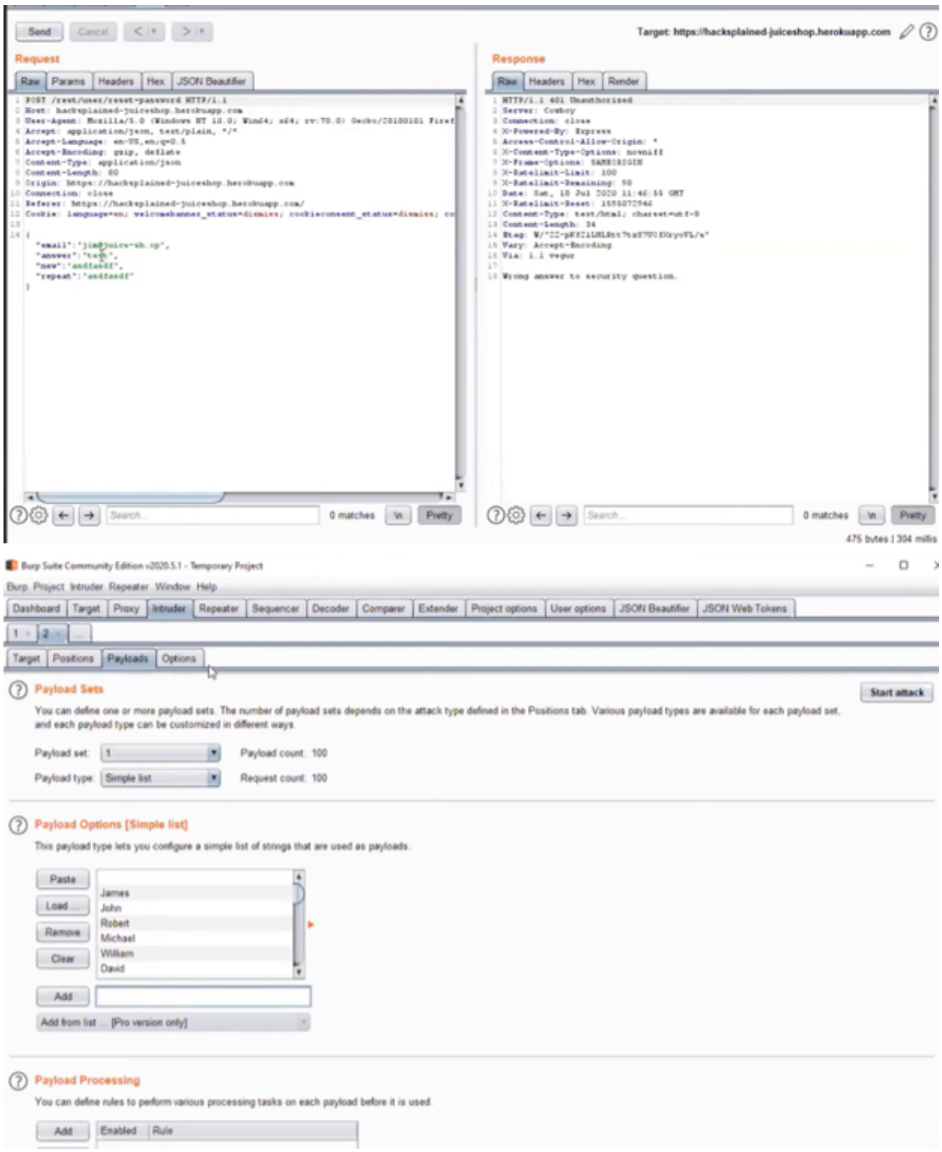

Figure 116: Enter Caption

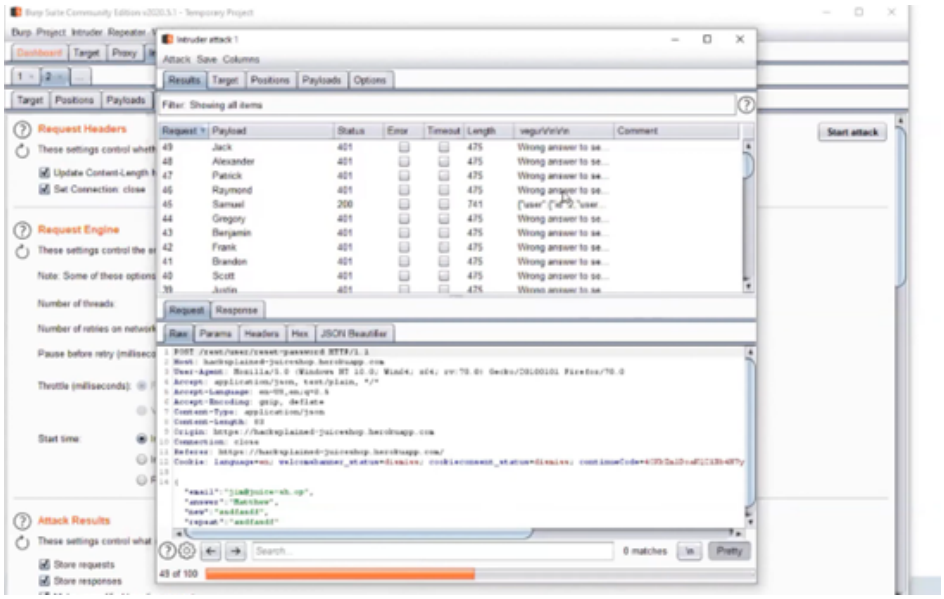

Figure 117: Enter Caption

27 Exploitation 27: Forgotten Developer Backup (Sensitive Data Exposure)

(Abdelrahman Ashraf - 120220292)

27.1 Challenge Goal

Access a forgotten developer backup file located on the server.

27.2 Category

Sensitive Data Exposure

27.3 Attack Method – Null Byte Injection & URL Encoding

27.3.1 Step 1: Locate the Target File

- Found a directory `/ftp/` using fuzzing techniques.
- Inside the directory, located a file named `package.json.bak`.

27.3.2 Step 2: Bypass File Extension Filtering

- Attempting to access the `.bak` file directly failed since Juice Shop allows only `.md` or `.pdf` file types.

- Used a Null Byte Injection (`%00.md`) to trick the system into thinking it's a valid file type: `/ftp/package.json.bak%00.md`

27.3.3 Step 3: Handle Rejection with Proper Encoding

- The raw null byte caused a `400 Bad Request`.
- Applied double URL encoding to encode `%00` as `%2500.md`.
- This bypassed the file extension validation and successfully accessed the file.

27.4 Result – Exploit Success

- The backup file containing sensitive configuration data was successfully opened.
- The challenge was marked as solved in Juice Shop.

27.5 Key Lesson

Security filters must be robust against null byte injections and must properly sanitize all file paths and extensions to prevent unintended file access.

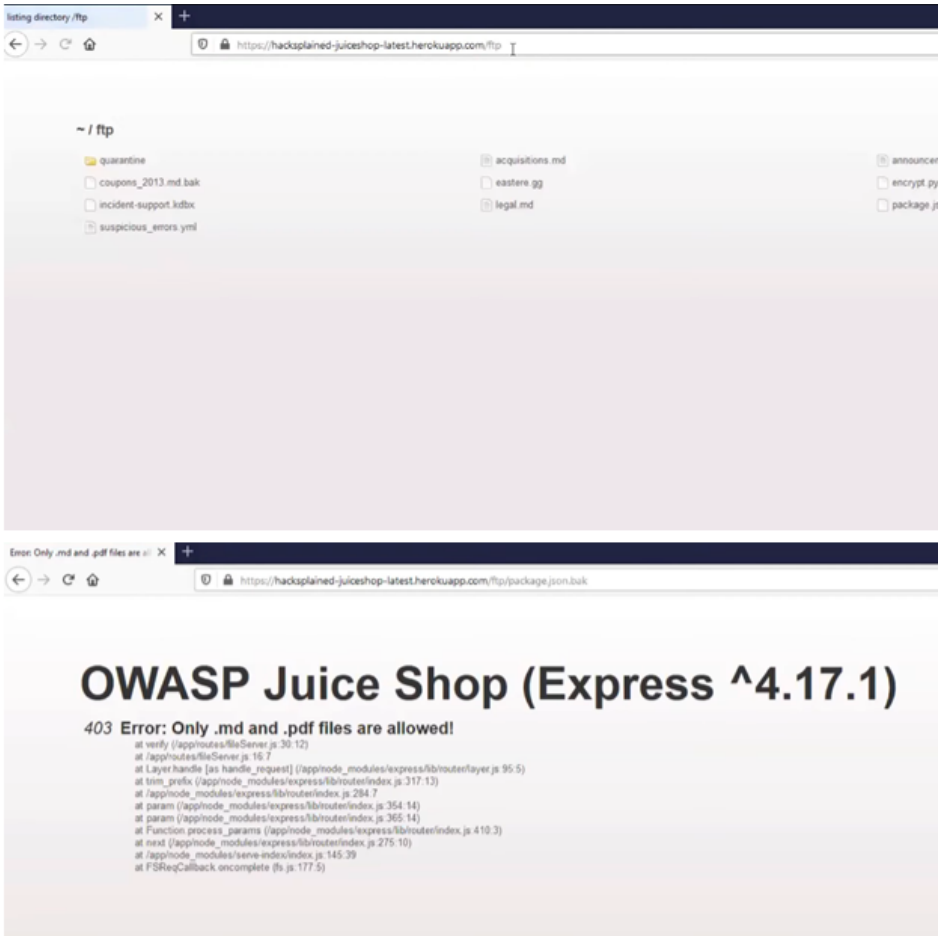

Figure 118: Enter Caption

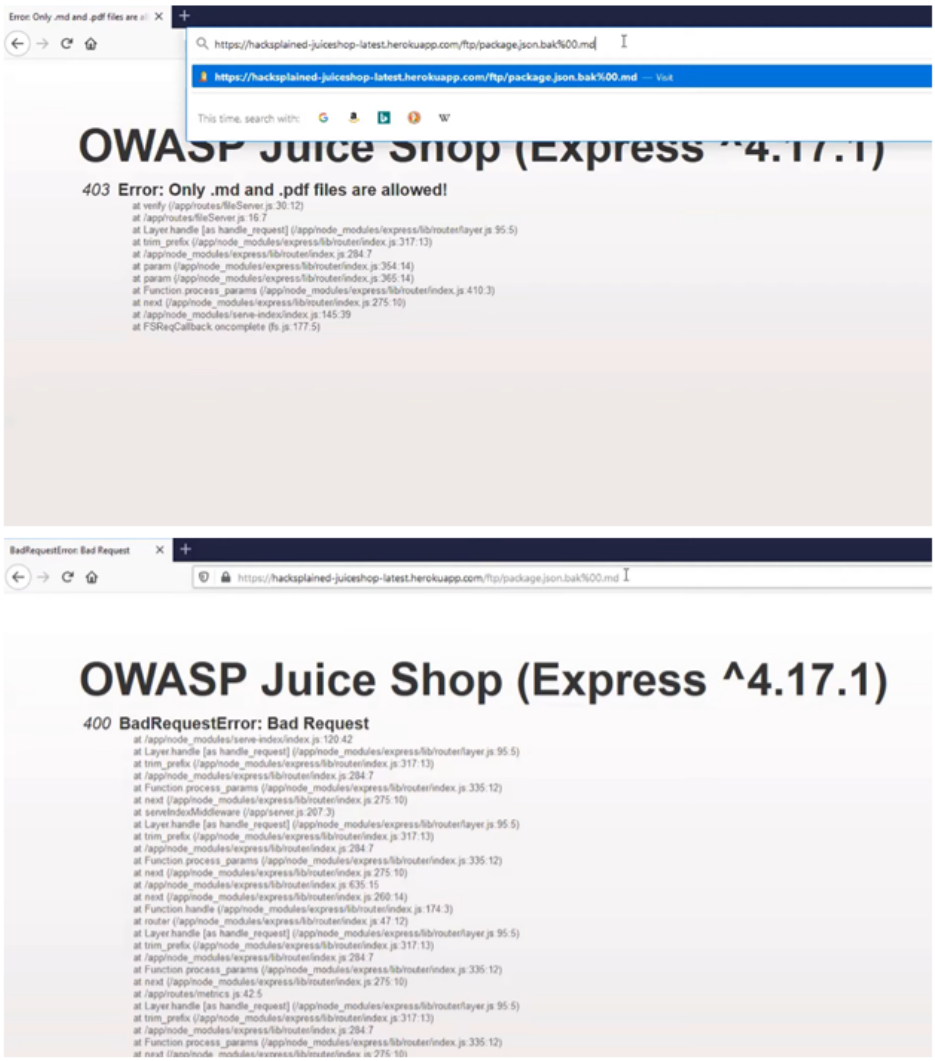

Figure 119: Enter Caption

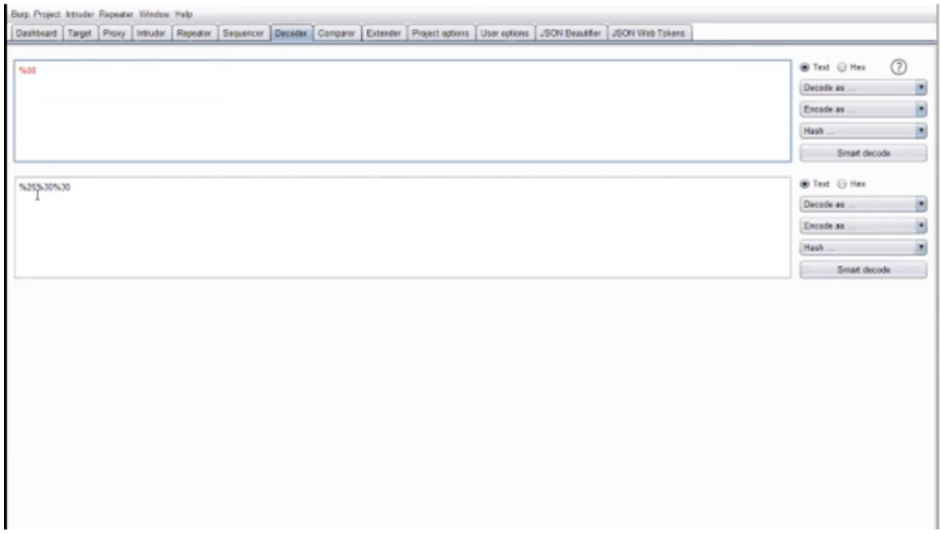

Figure 120: Enter Caption

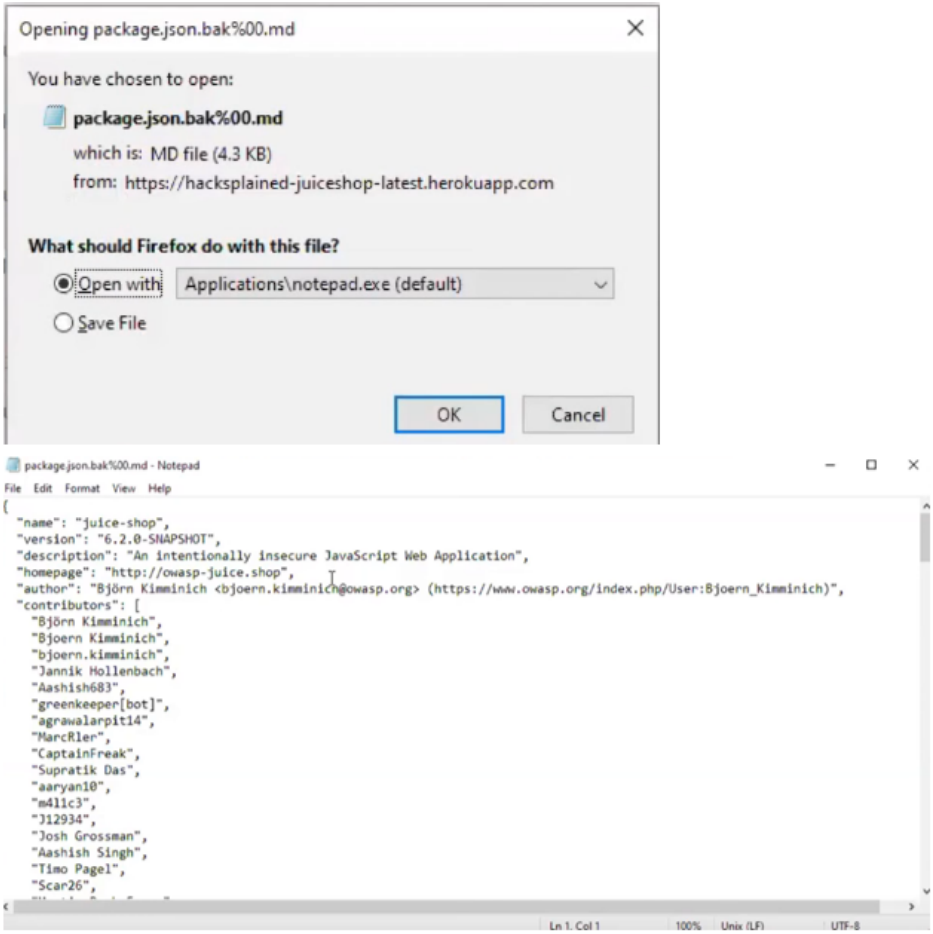

Figure 121: Enter Caption