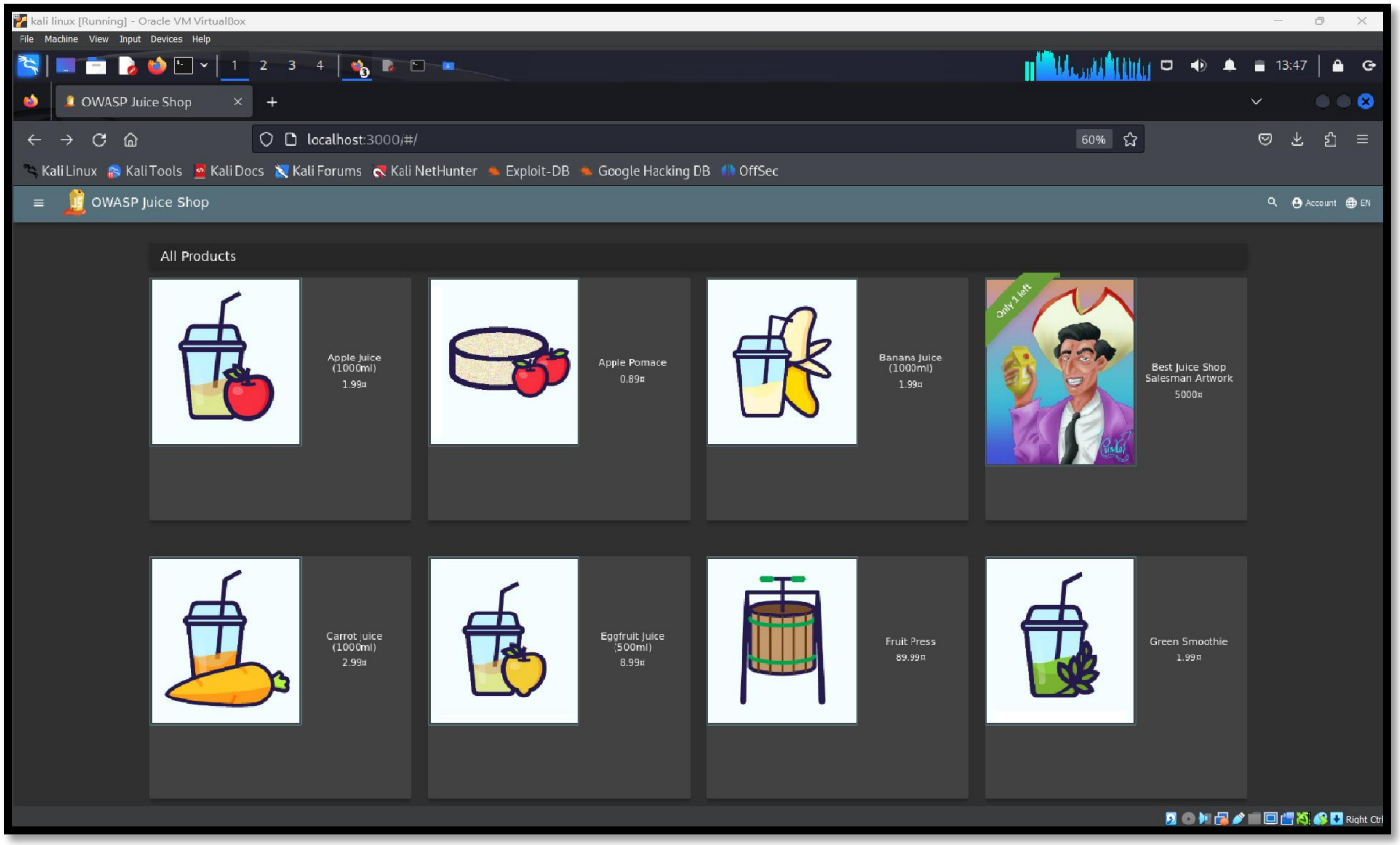
Subject: Security Testing

SENG 8061

Project Part - 03

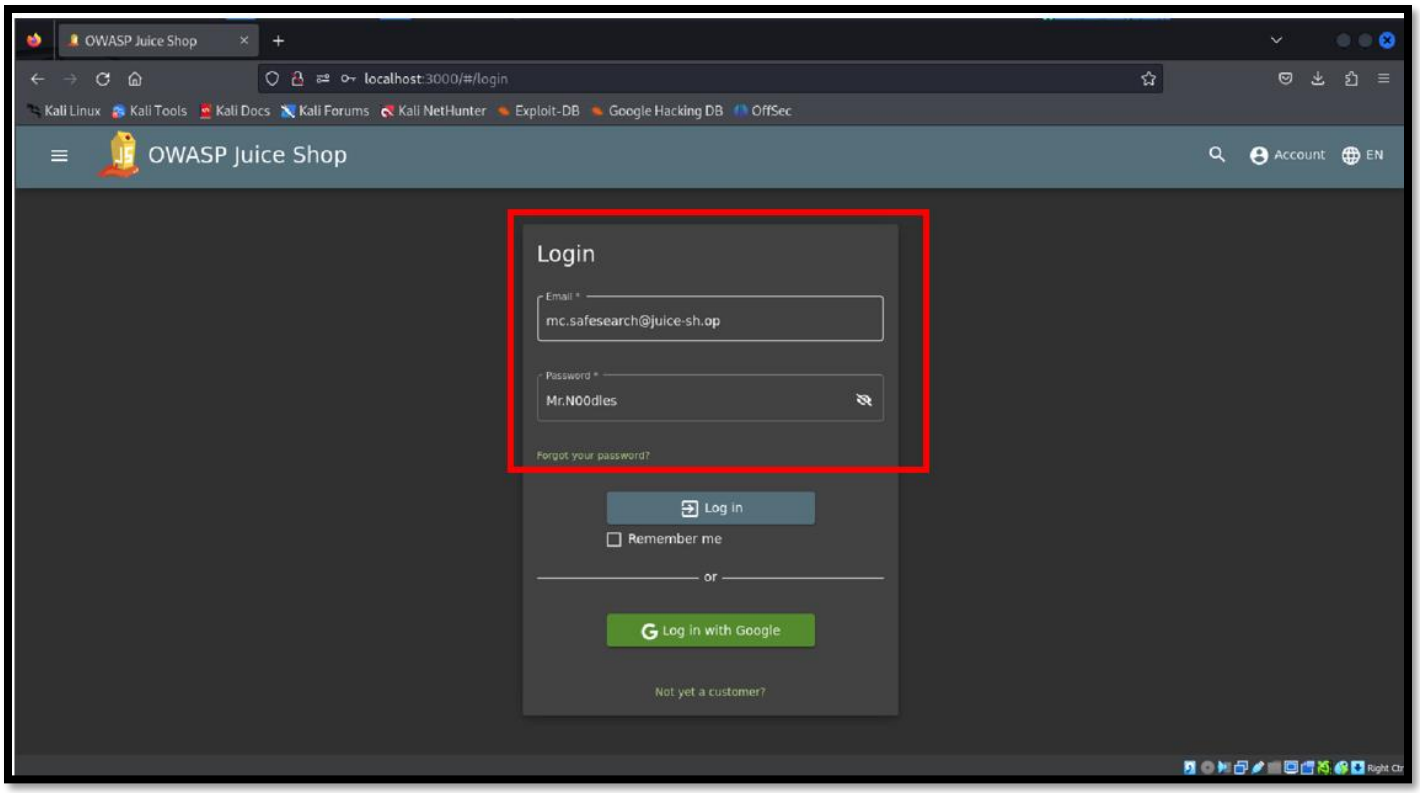
Name	Student ID
Shivani Varu	8941914
Mohammed Rafique	8954785

Screenshot 1: Go to http://localhost:3000/

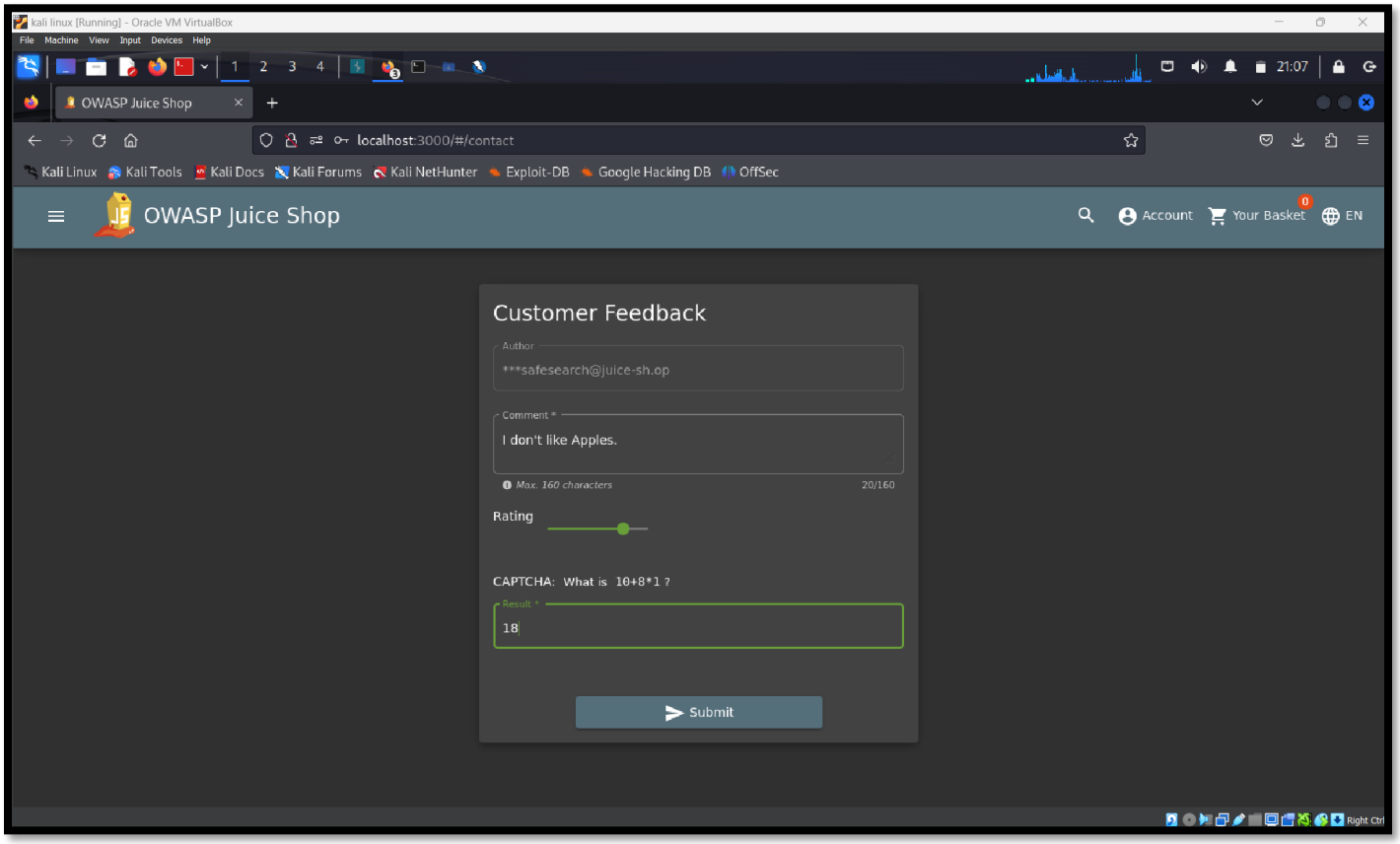


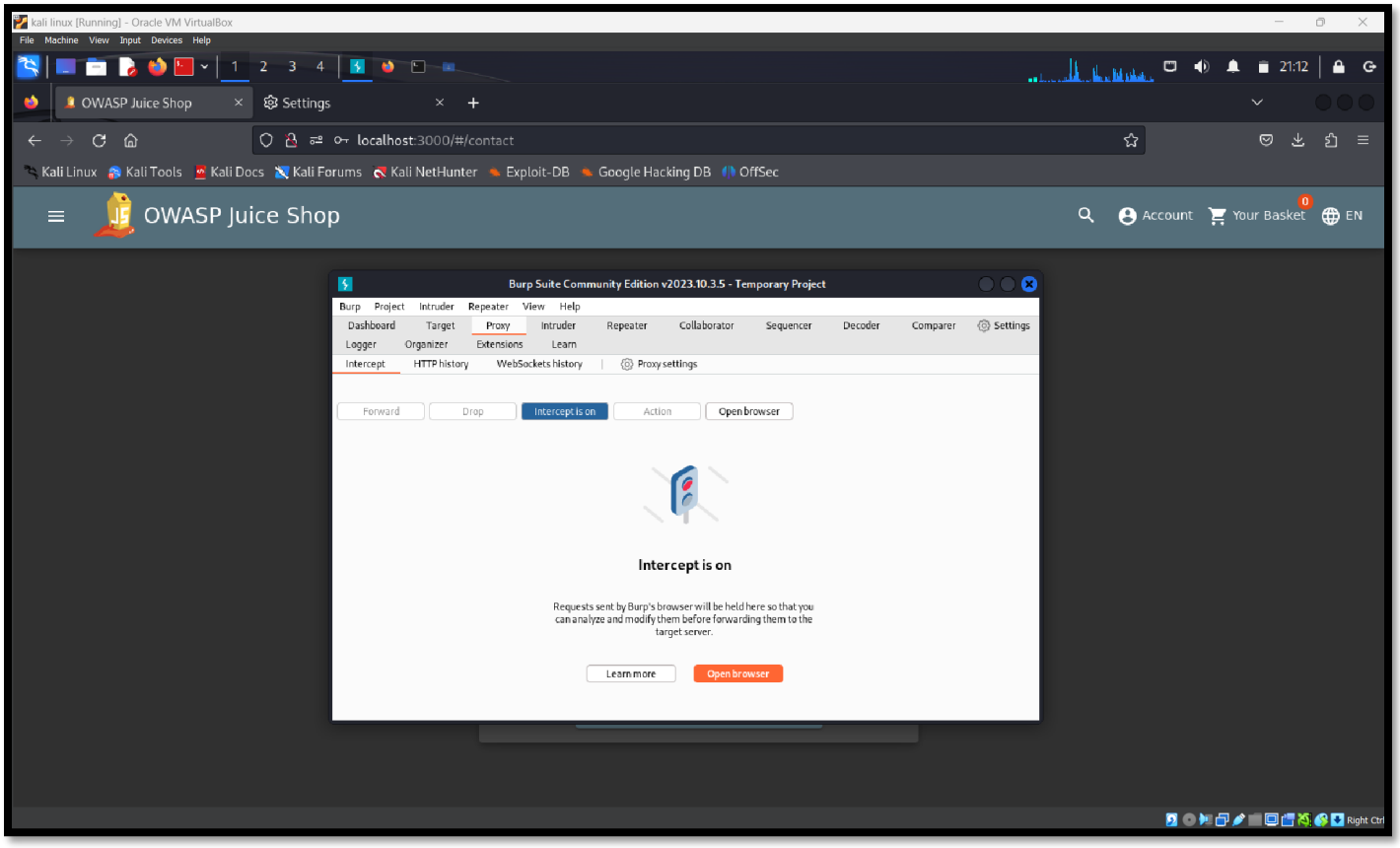
TASK 1
Post some feedback in another user's name

In this challenge, we recalled the email and password of mc. safesearch via project part 2's Task2. We entered his username and password to log in to his account and provide Customer Feedback, which resided in the hamburger menu.

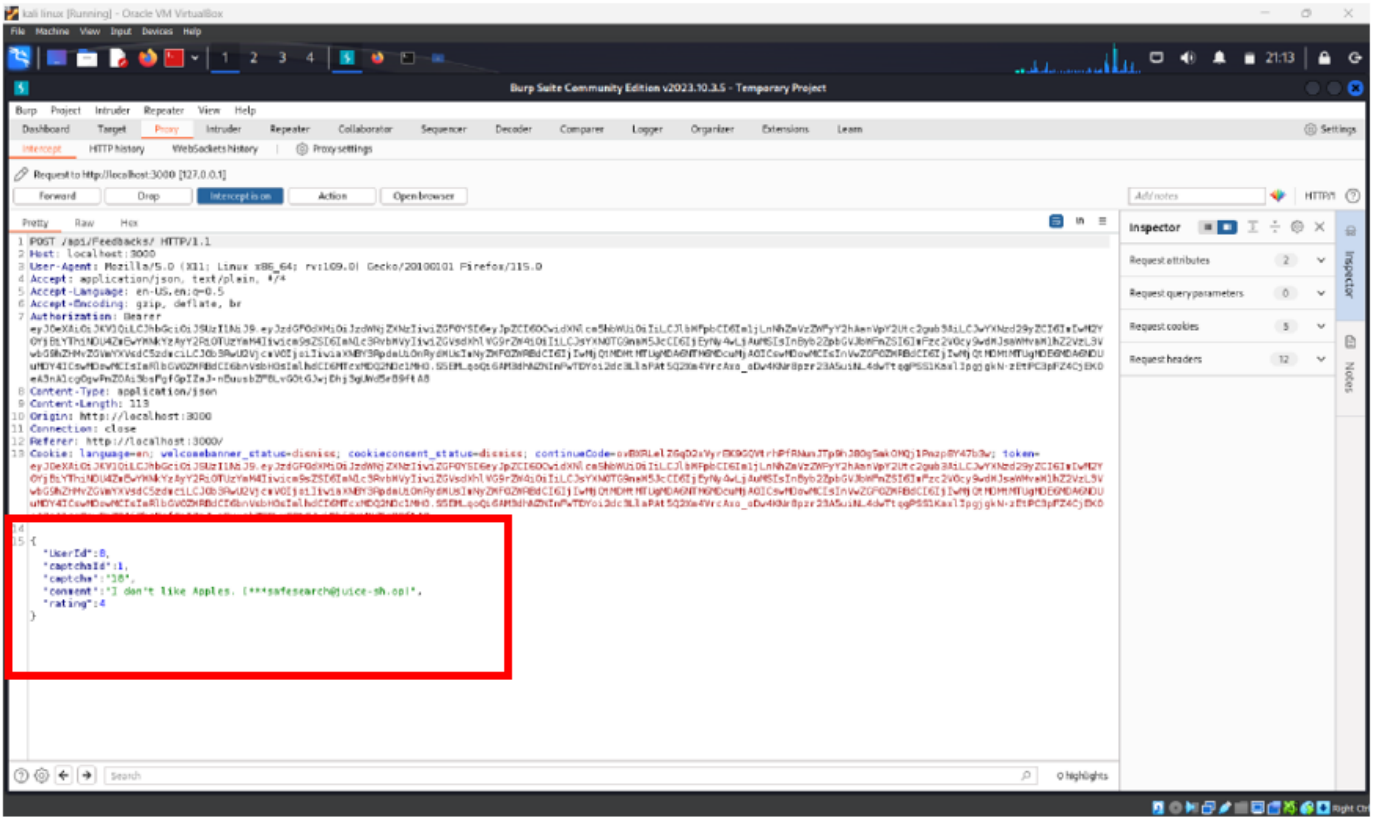


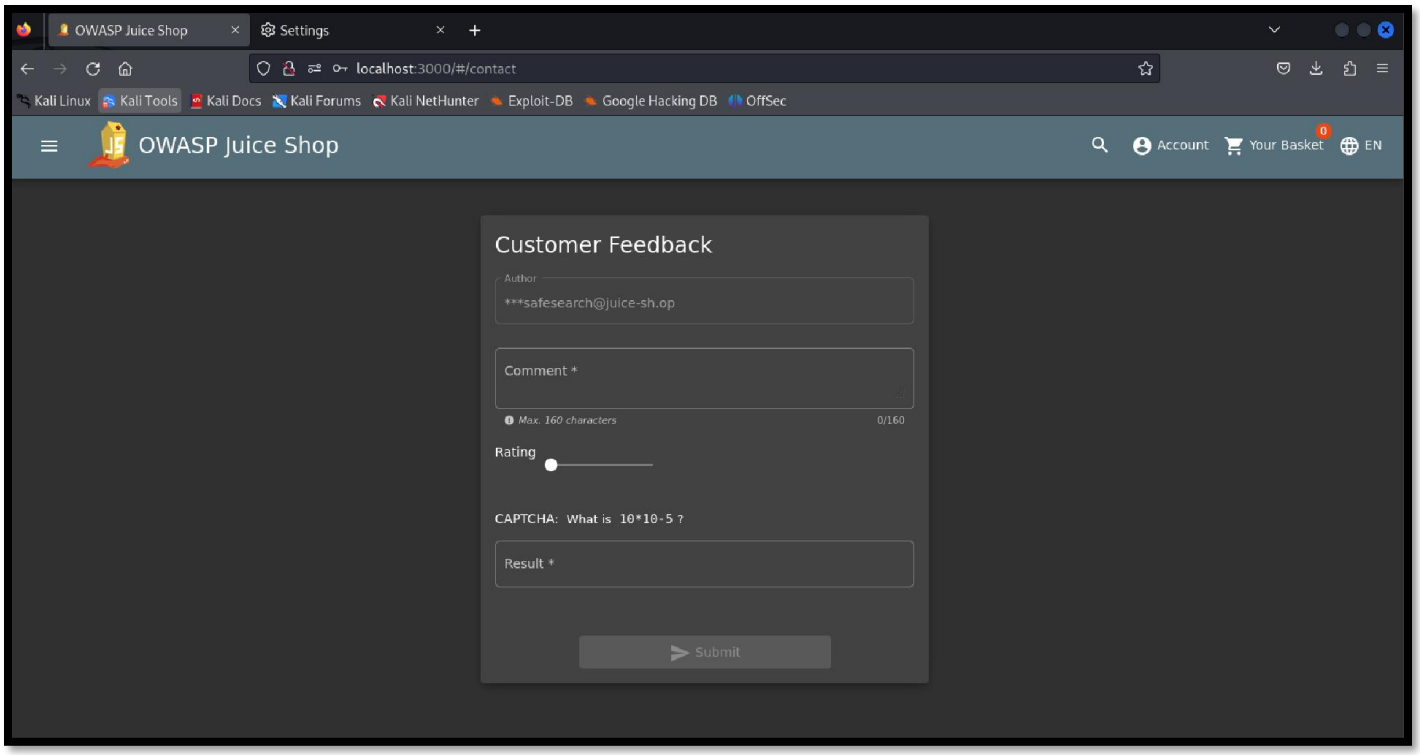
While inspecting the Juice Shop application, we found an option called customer feedback and attempted to enter some feedback. With the help of the Burp tool in Kali, we intercepted the customer feedback request form.



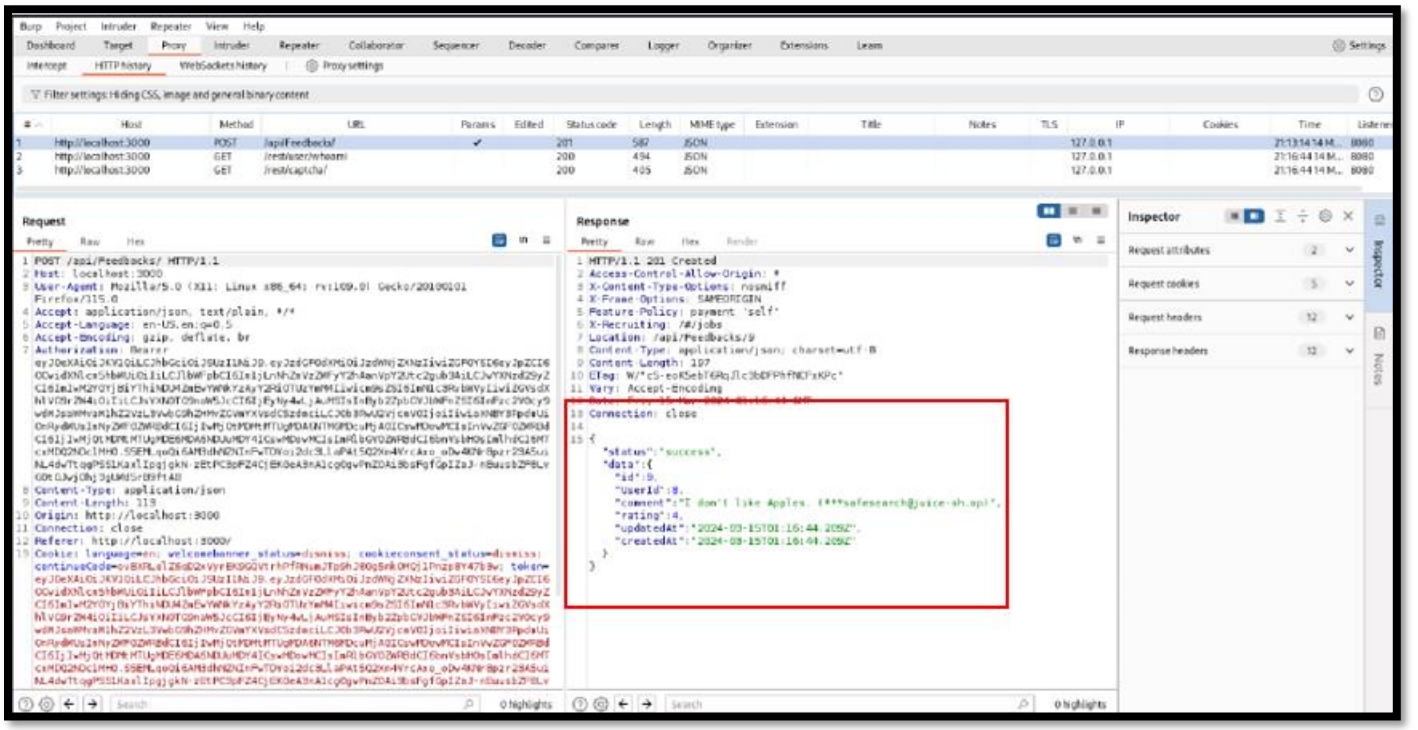


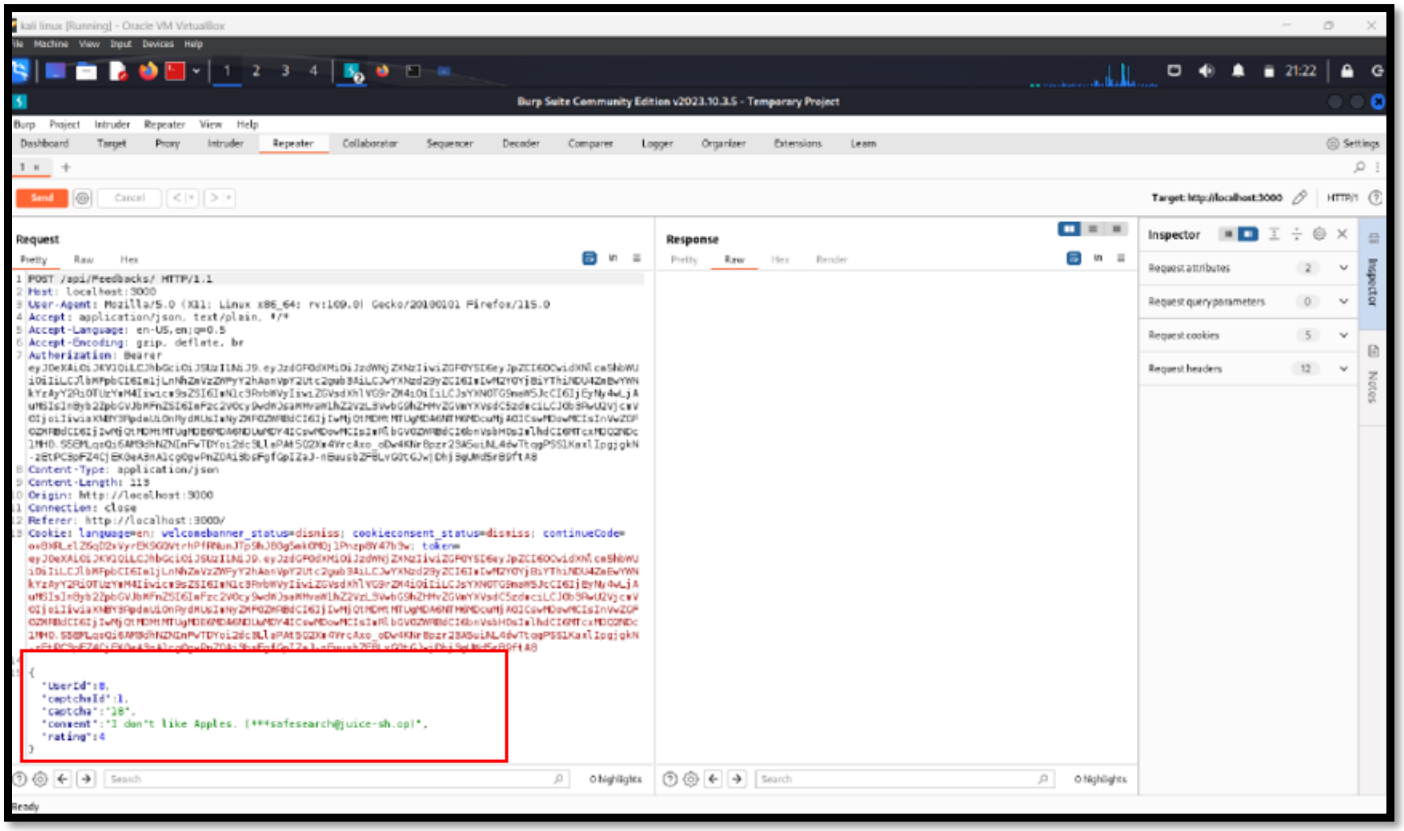
Upon inspecting the request, we discovered a comment id, captcha id, and rating, but nothing was mentioned about the user id.



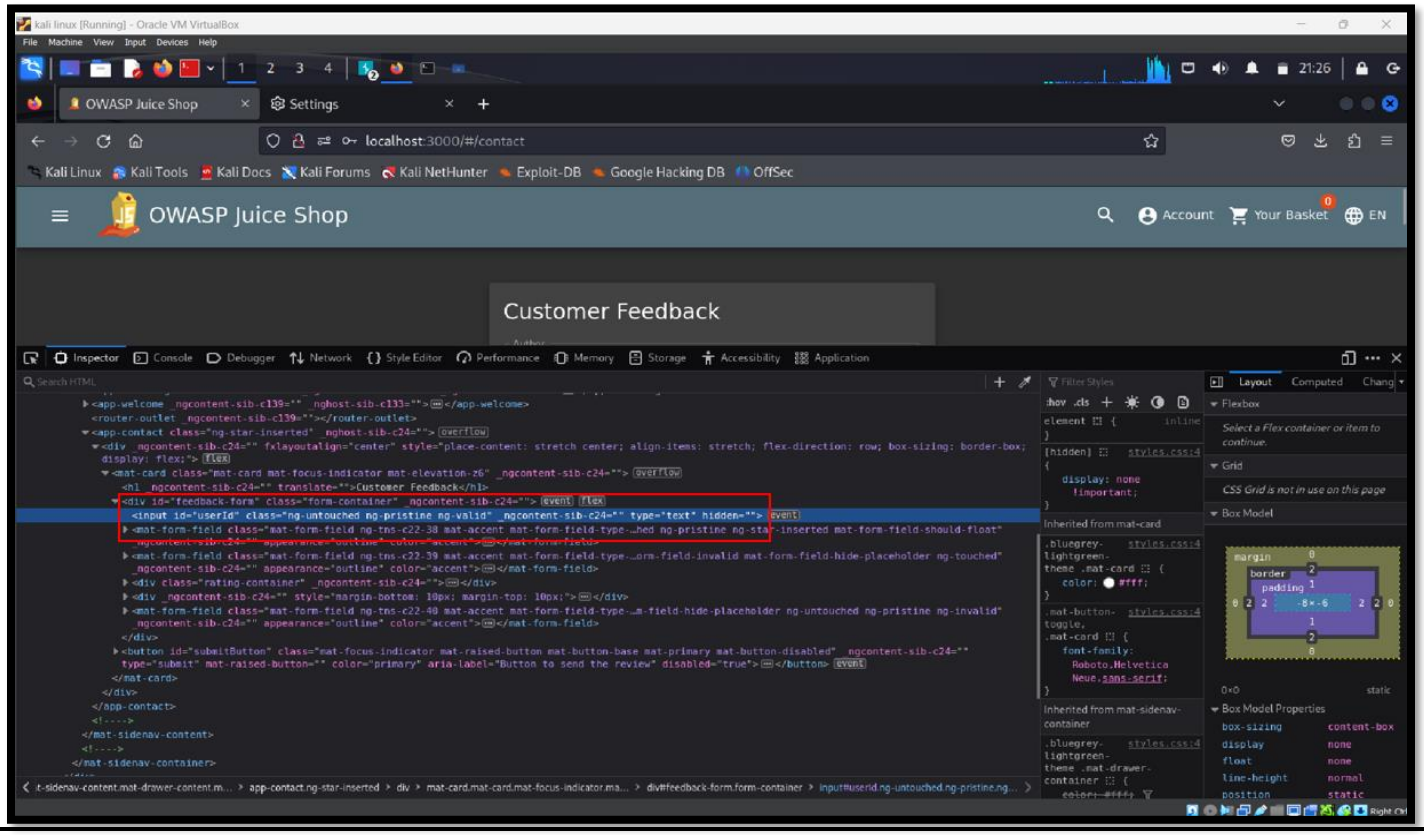


Afterward, we turned off the intercept and submitted the user feedback. We then reopened Burp Suite and intercepted to see the response in the HTTP history. We observed that we received a response status of success.

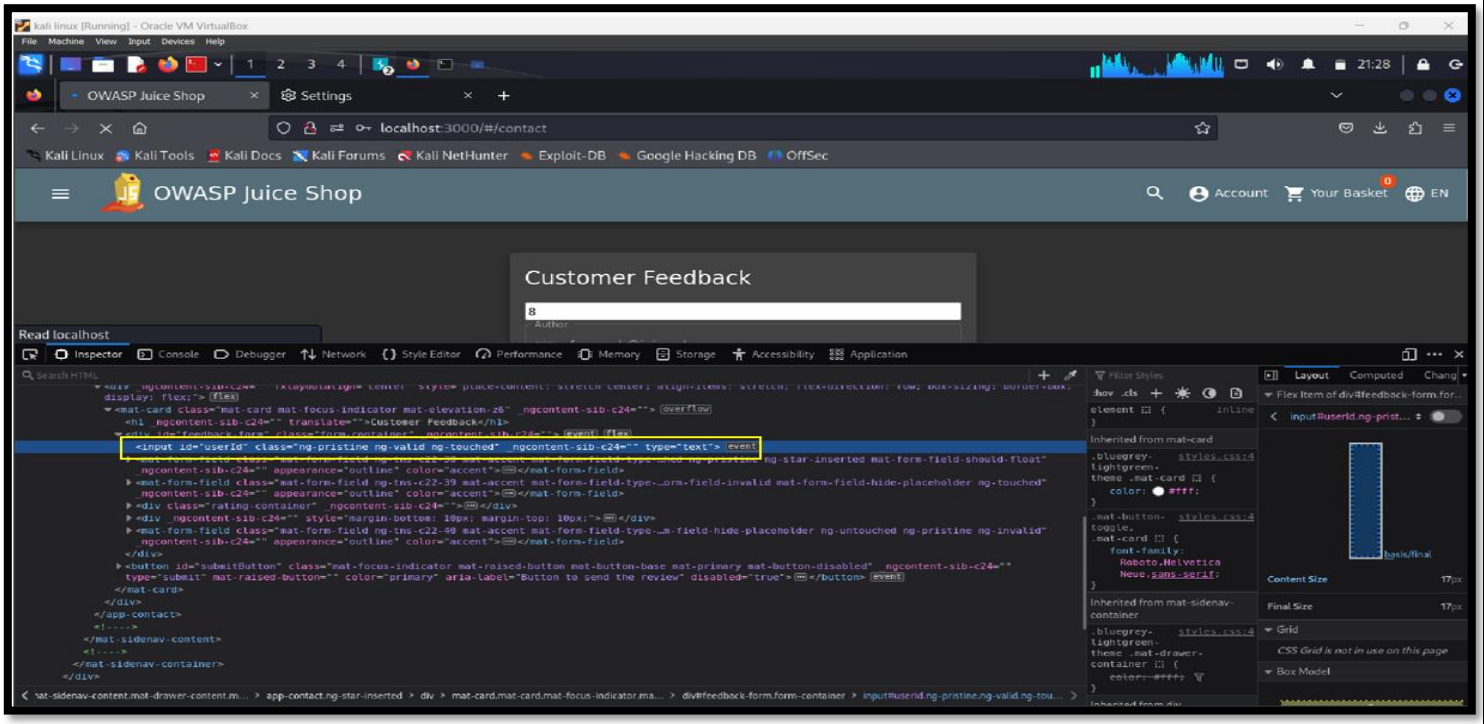




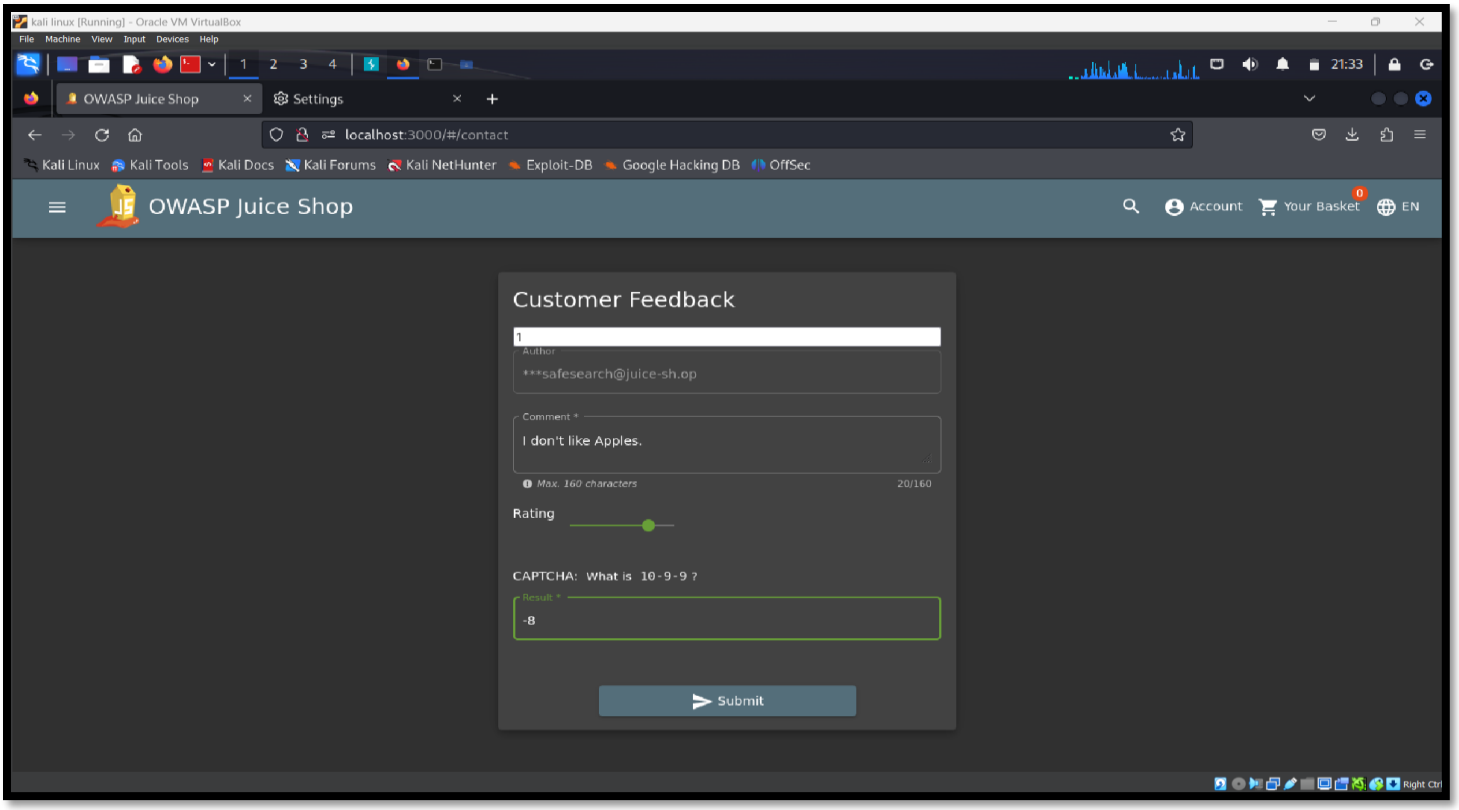
After that, we sent these to repeater. To investigate further, we opened the browser and inspected the feedback form using the developer tools to find something related to the user id. Upon inspection, we noticed that the user ID was included as a parameter in the form submission, and we found a hidden tag, which essentially meant that the input box couldn't be seen in the UI.



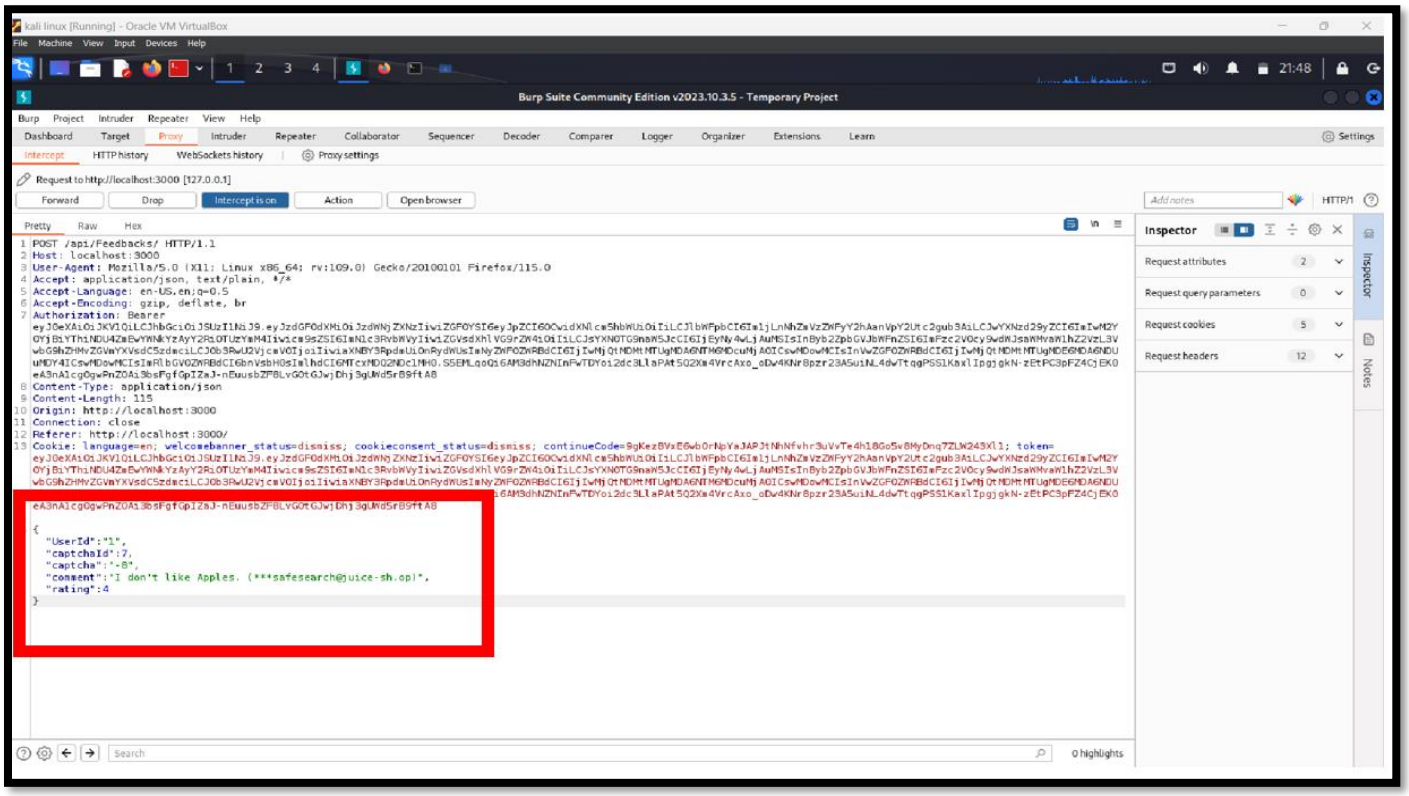
Therefore, we deleted that hidden tag and hit enter to observe the outcome. We observed that an input box popped up, displaying the user ID 8, which we retrieved from BurpSuite. Using the developer tools, we modified the HTML code of the form to replace our own user ID with the ID of another user.



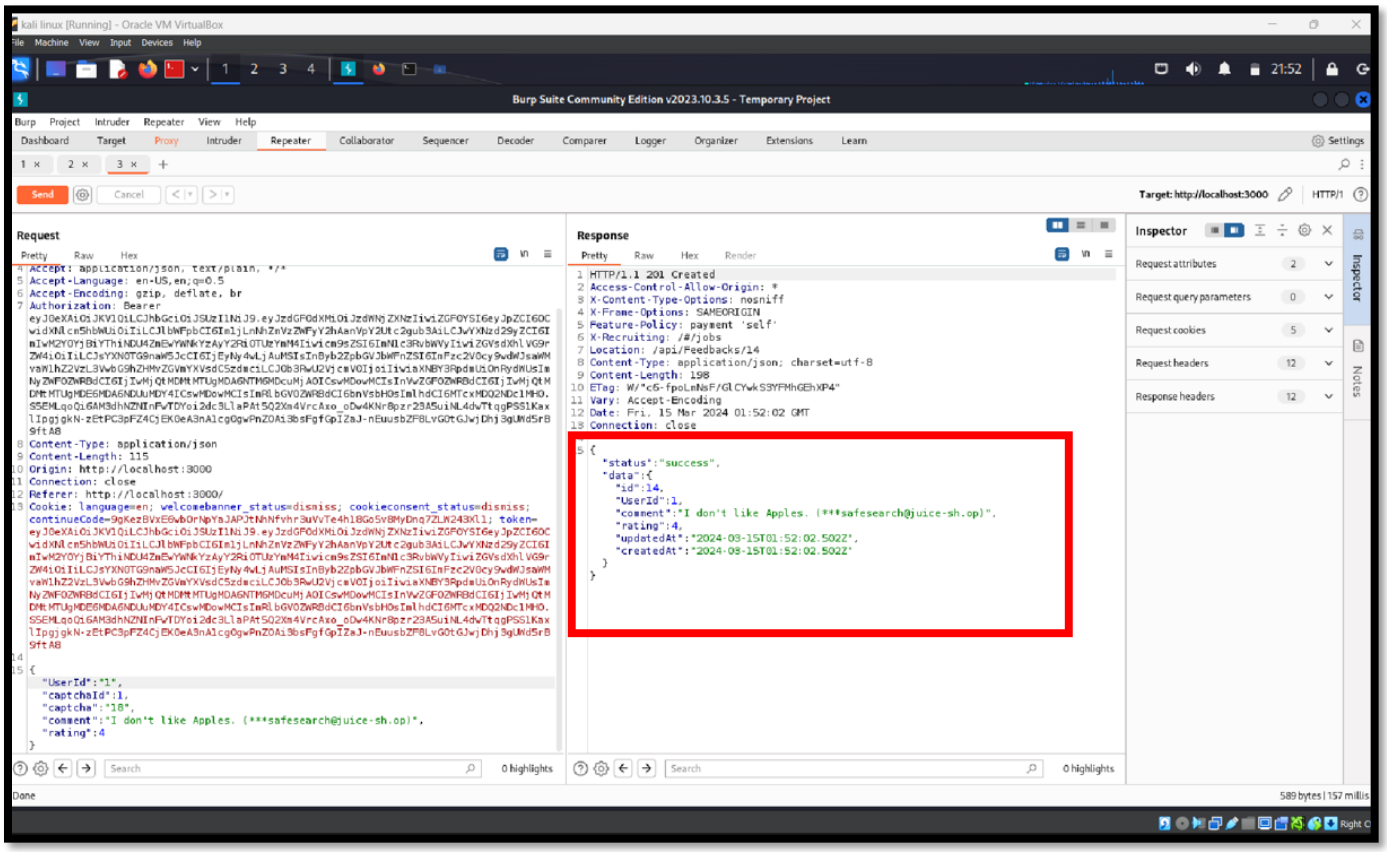
To complete our task, we changed the userid from 8 to 1 to see the result. Thus, we turned off the intercept in BurpSuite, then entered userid as 1, filled the feedback form, and intercepted before submitting the feedback.



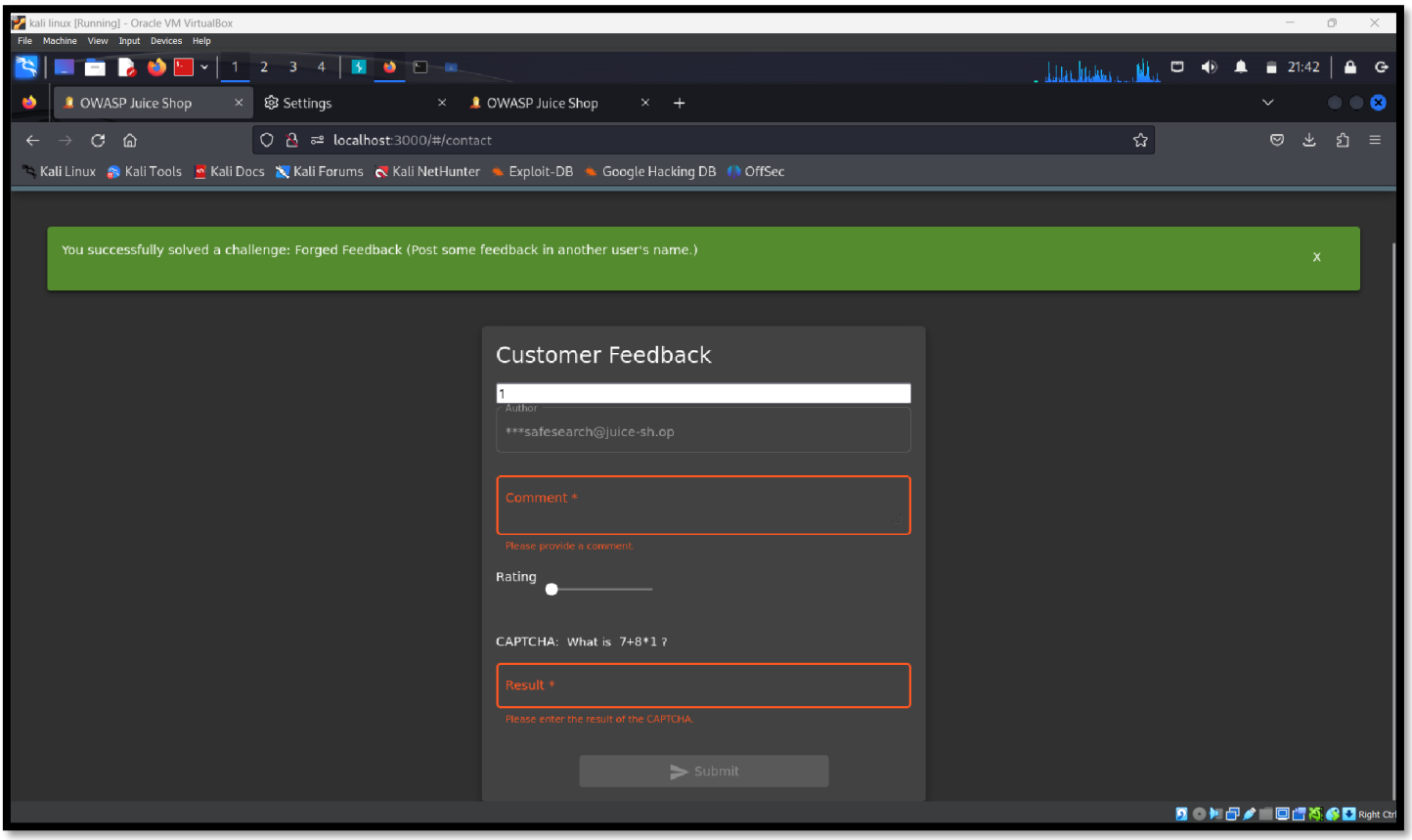
Before submission, we turned on the intercept in BurpSuite, and upon clicking the submit button, we received a response on BurpSuite indicating that the user id was changed. We forwarded that to repeater.



Then, we clicked the send button to see what response status we would get.



We received a response status of success, indicating that we had successfully created new feedback. Finally, we turned off the intercept to check what we would get on the feedback page, confirming that we had successfully solved this challenge by posting feedback under another user's name.



Vulnerabilities Exploited by Task 1:

- Insecure Direct Object Reference (IDOR)
- Lack of Access Control
- Insufficient Input Validation

How Vulnerabilities Can Be Fixed or Addressed:

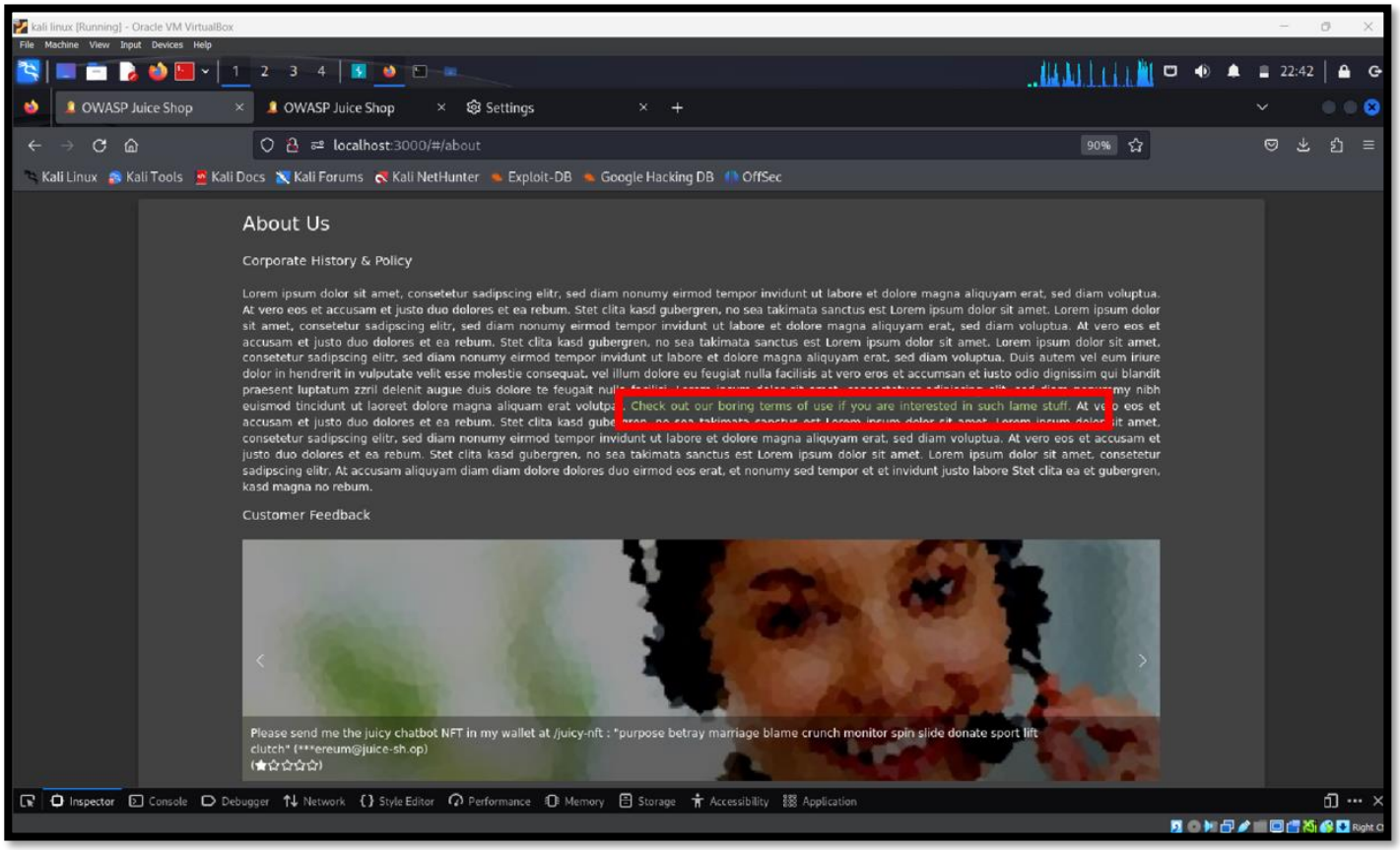
- Implement Proper Access Controls
- Validate User Input
- Use CSRF Tokens
- Secure API Endpoints

Real-World Scenarios:

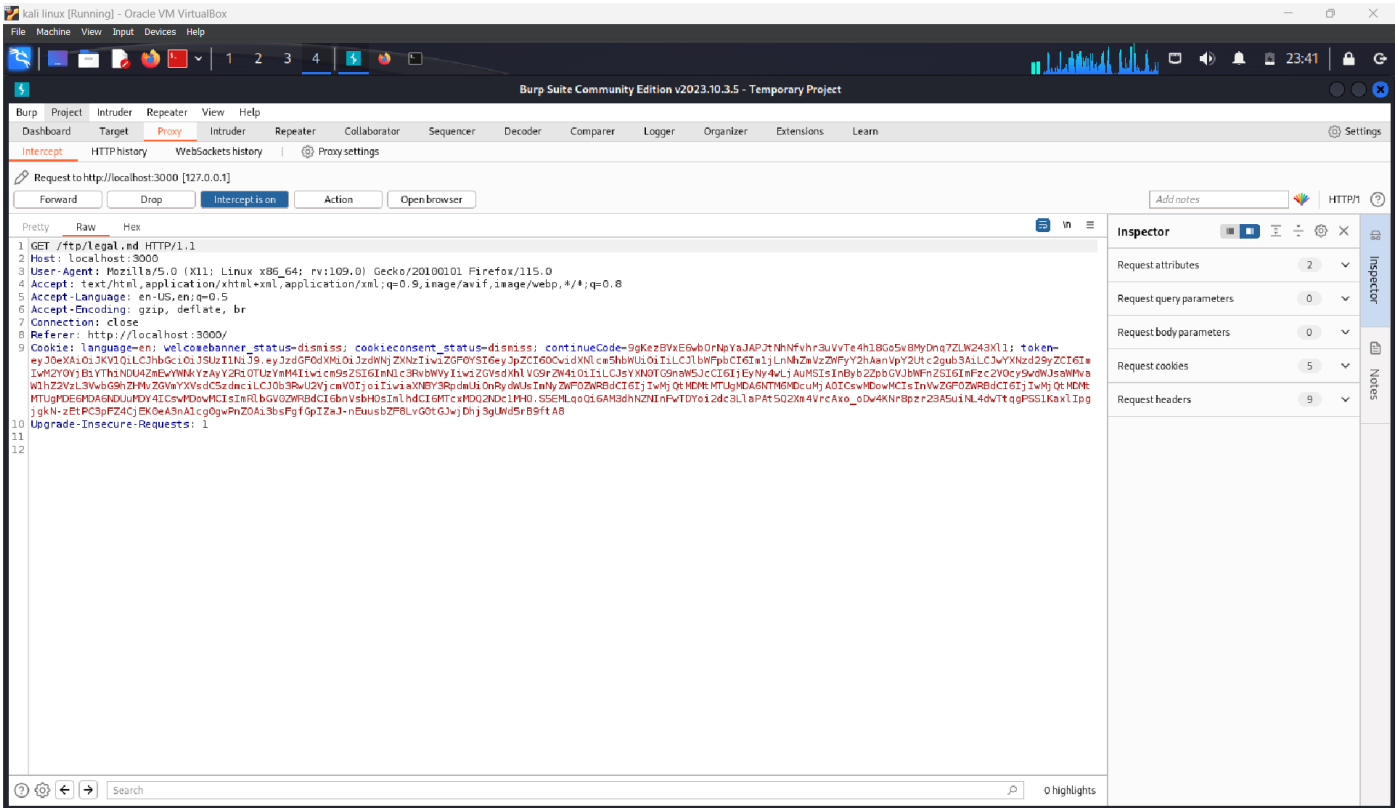
- Exploiting vulnerabilities like IDOR and inadequate access controls can lead to severe consequences.
- Attackers may impersonate users, post malicious content, or manipulate sensitive data.
- This can result in reputation damage, legal consequences, and financial losses for organizations.
- The attacker can steal information from other users and conduct passive reconnaissance.

TASK 2.1
Access a confidential document

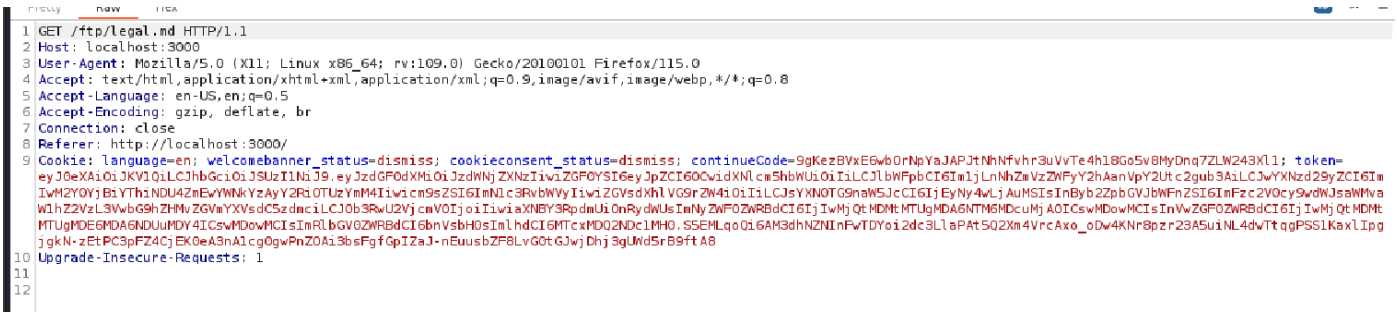
In this challenge, we tried to expose a confidential document in the Juice Shop application. We attempted to search for something useful within the Juice Shop application. An interesting item appeared on the about page of the Juice Shop.



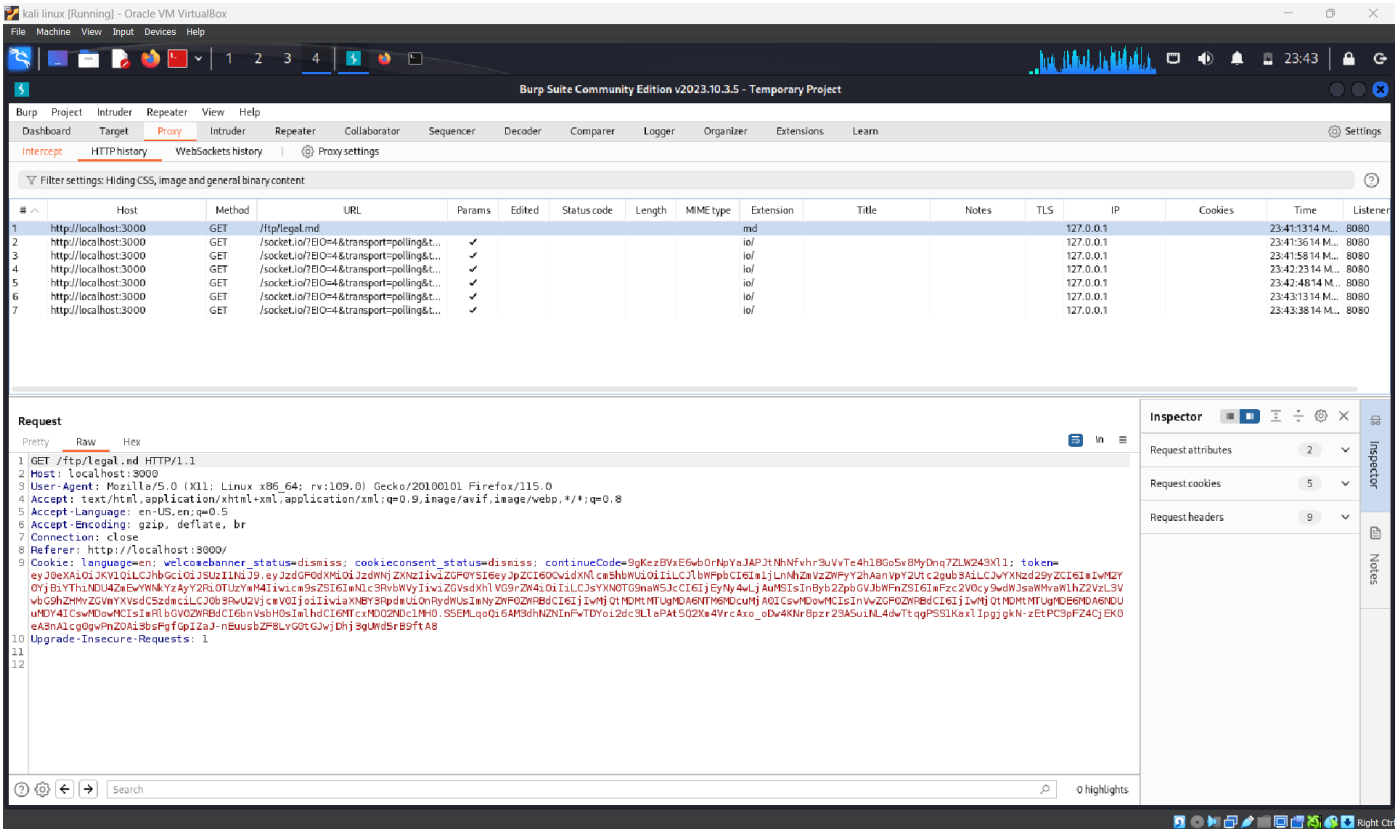
We opened the Burp tool to inspect this link. While inspecting this request, we found a directory named FTP containing a file named legal.md.



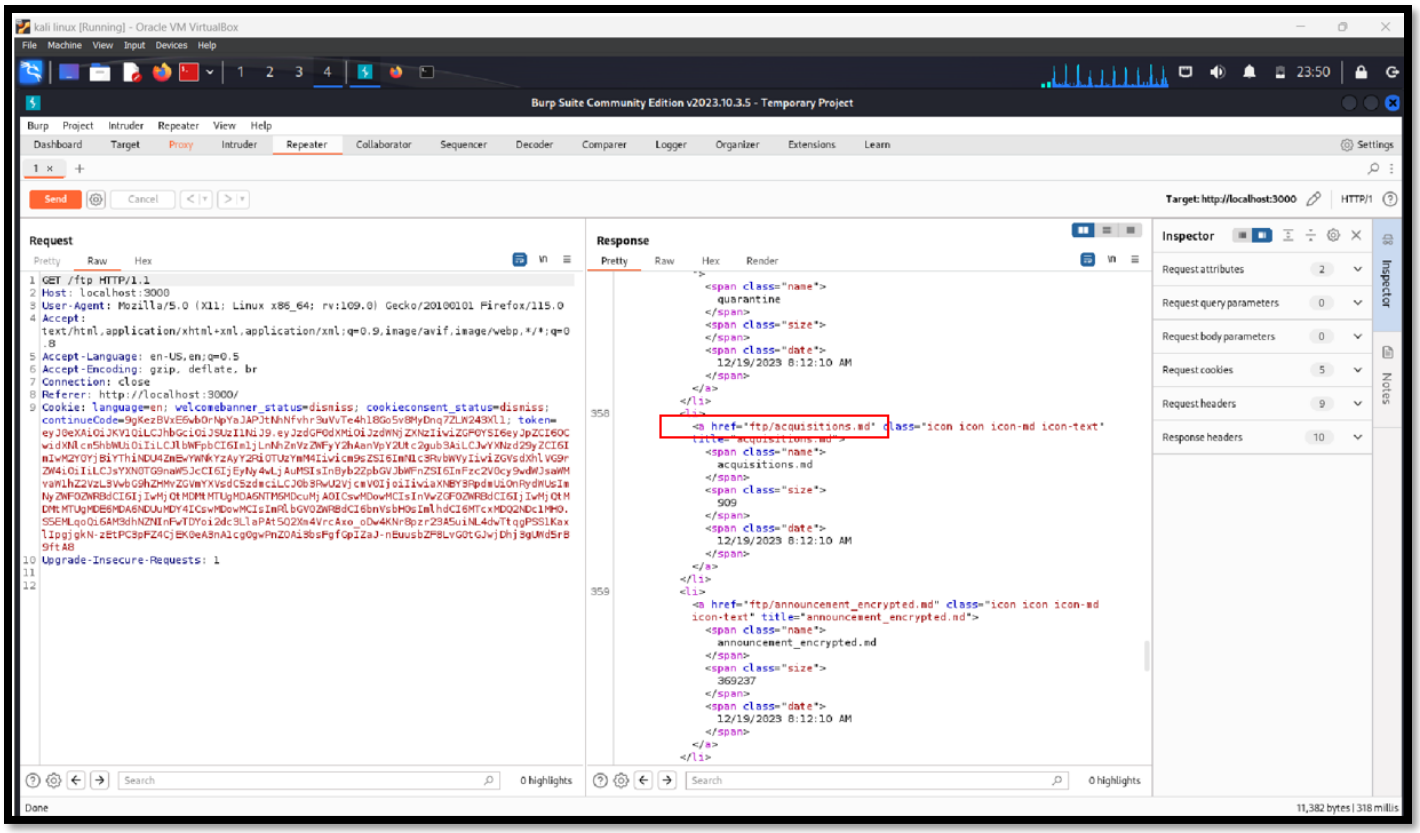
After forwarding the request to the server, we found this file. But we needed to access a confidential file, so we inspected the directory named FTP. We opened the Burp tool to inspect this link.



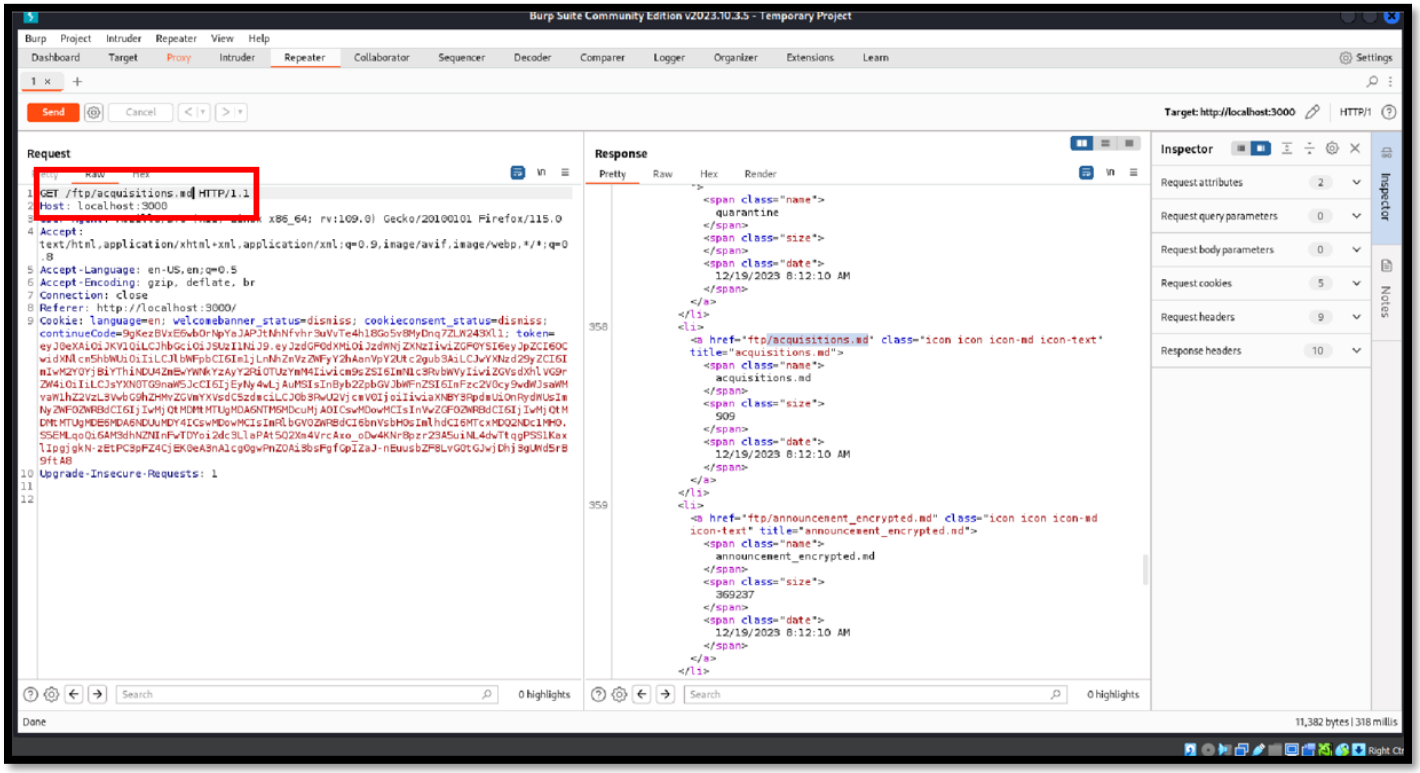
On the HTTP history, we got the URL path, so we found something important for this task. Therefore, we sent this URL path to repeater.



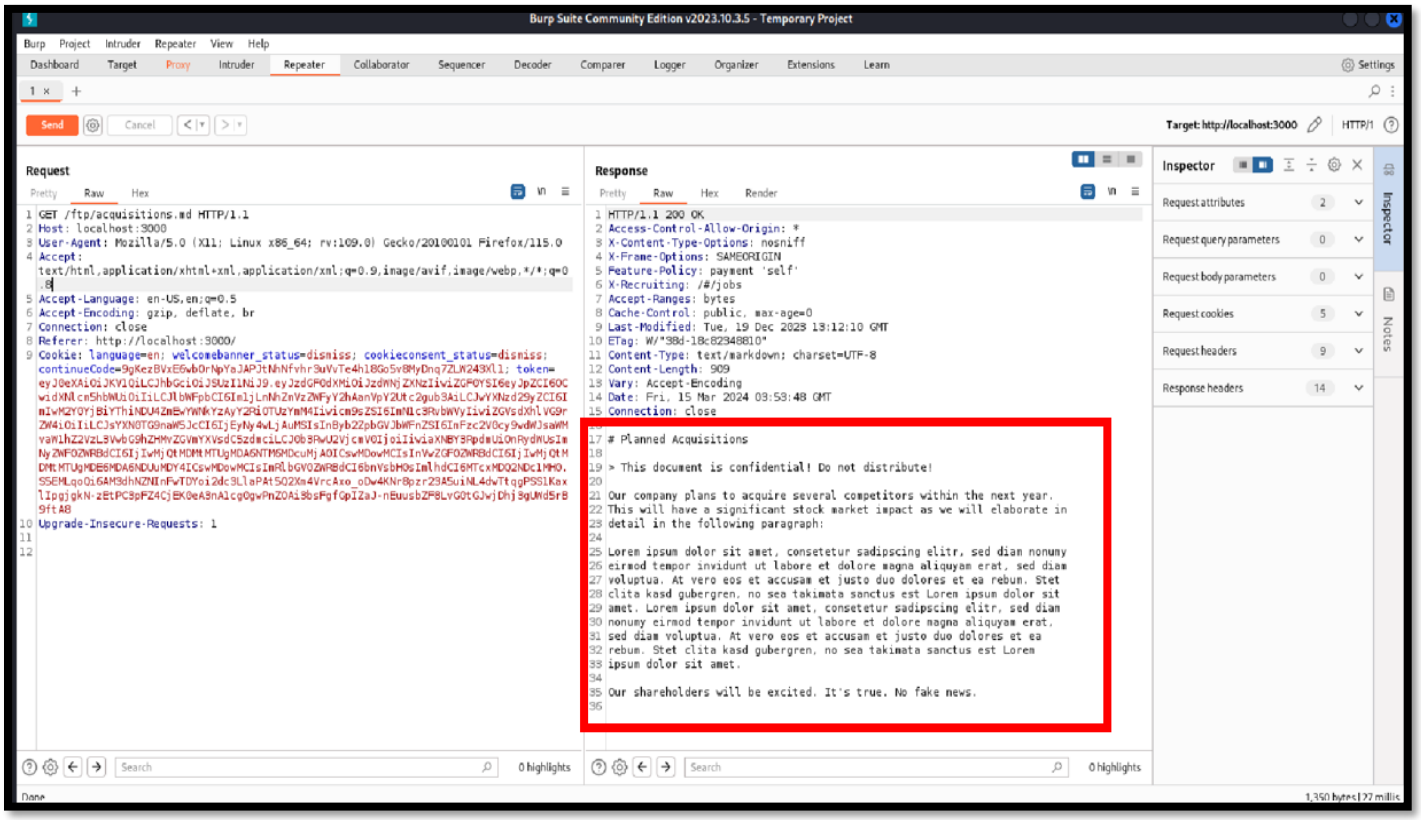
After that, we deleted the legal.md file path from the request and sent that request again. On the right-hand side in response, we got some information. We scrolled through it last. After inspecting the response, we found a file named acquisitions.md.



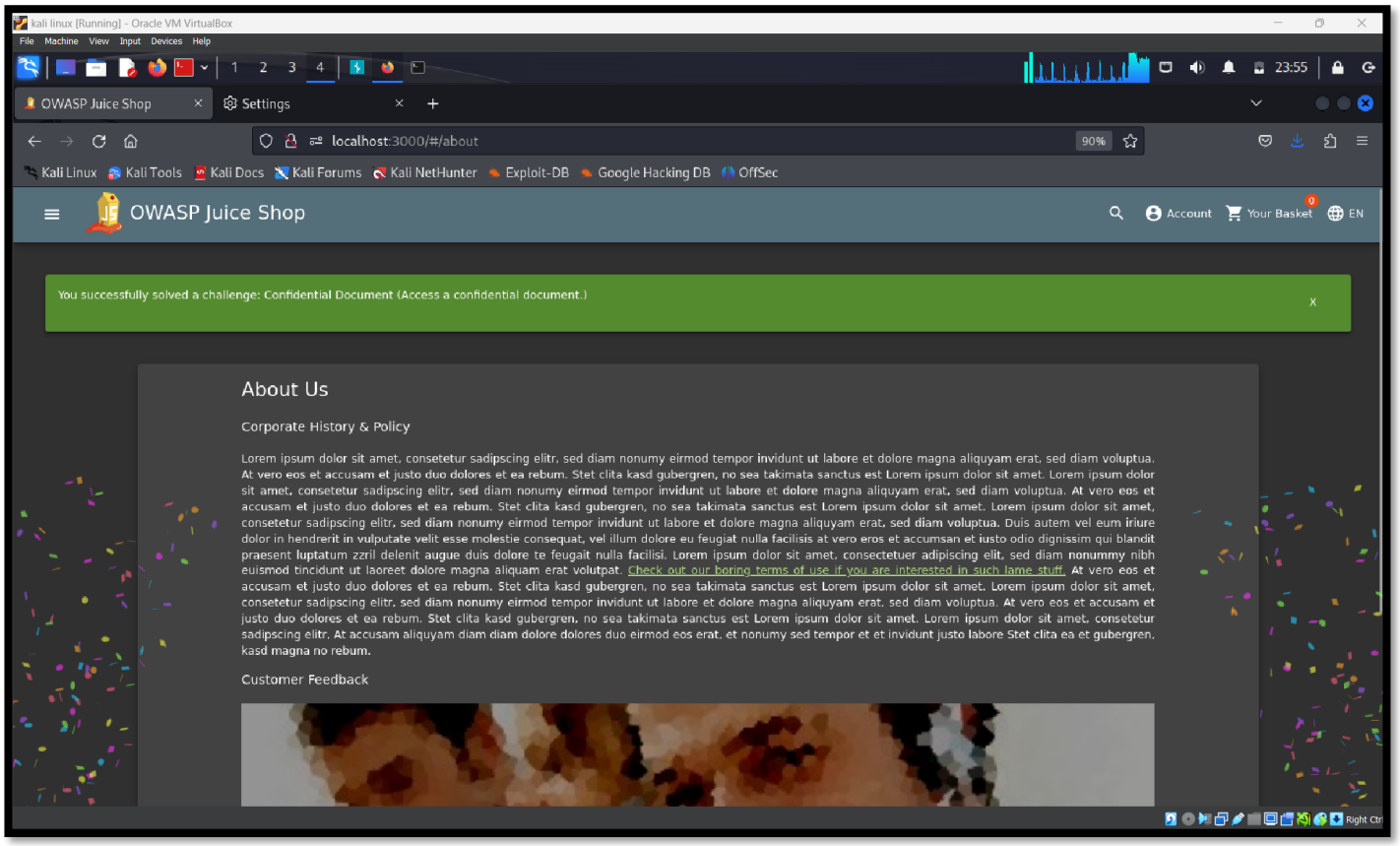
We copied that file path and pasted it on the request.



Afterward, we sent this request again. We went back to the Burp repeater and edited the file to acquisitions.md. After editing, we sent the request again. After checking the response, we found that this document was confidential.



After turning off the interceptor in Burp, we could see that we successfully solved this challenge and accessed the confidential document.



Vulnerabilities Exploited by Task 2.1:

- Listing Lax file directory access rules led to the exploitation of a traversal vulnerability.
- Inadequate authorization checks resulted in illegal access to confidential documents.

How Vulnerabilities Can Be Fixed or Addressed:

- Implement Access Controls
- Validate User Input
- Encrypt Confidential Documents

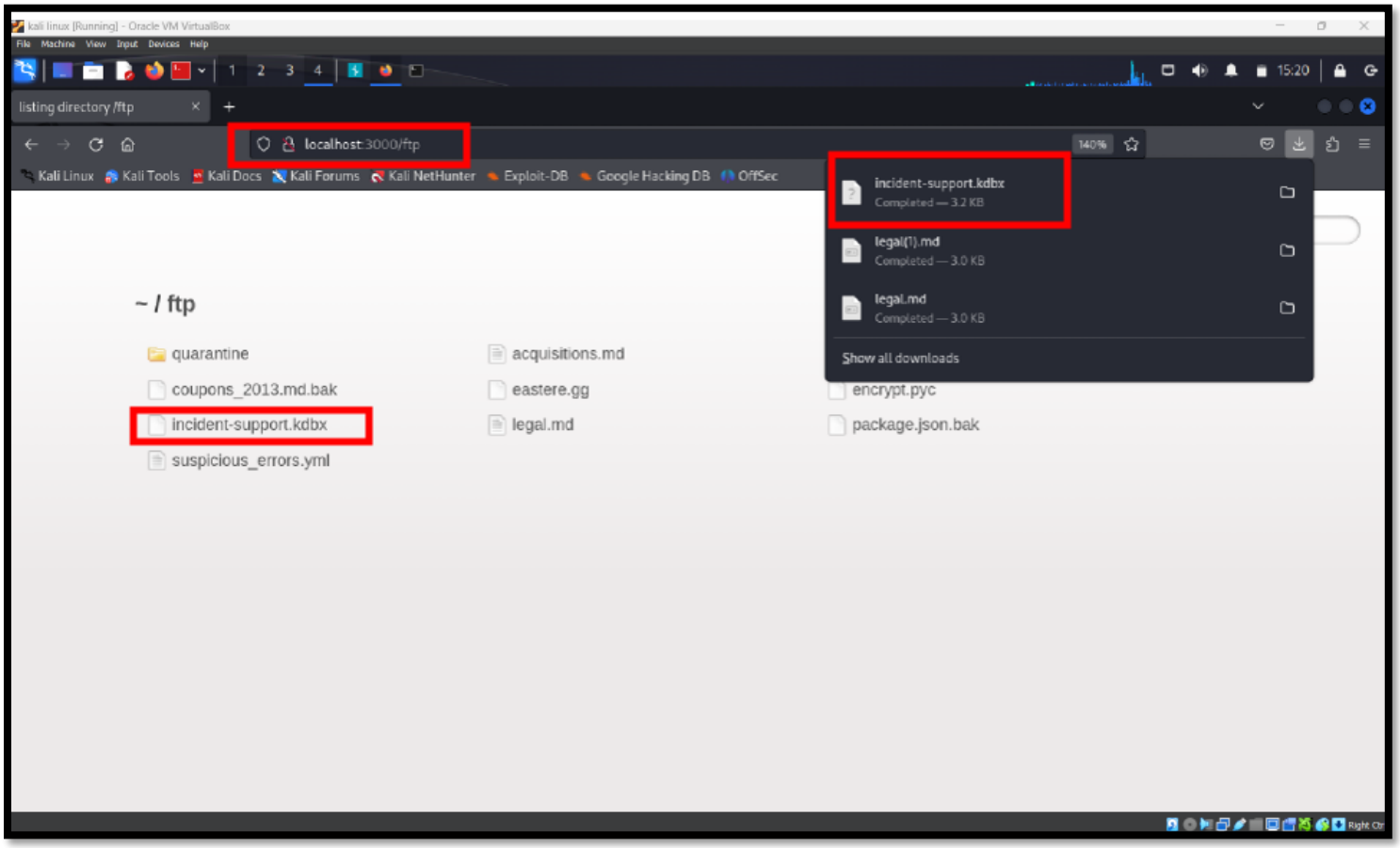
Real-World Scenarios:

- Exploiting directory traversal vulnerabilities can lead to unauthorized disclosure of sensitive information.
- This may include bank records, consumer information, or confidential documents.
- Consequences for the affected organization can include financial losses, loss of trust from customers, and legal penalties.

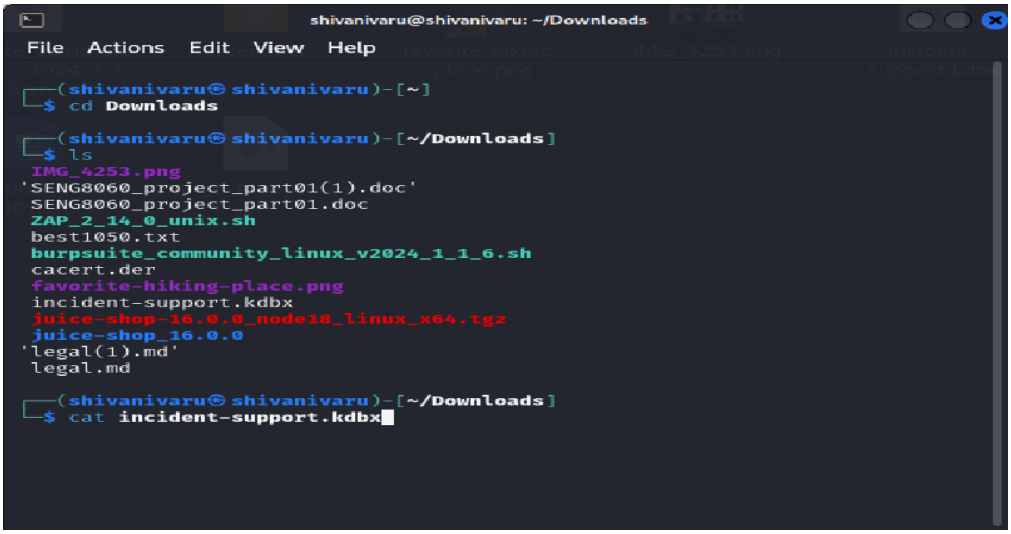
TASK 2.2

Gain access to any access log file of the server

In the previous challenge, we had already accessed the directory called FTP.
In this directory, after inspecting all the files, we found a file called incidentsupport.kdbx. We attempted to

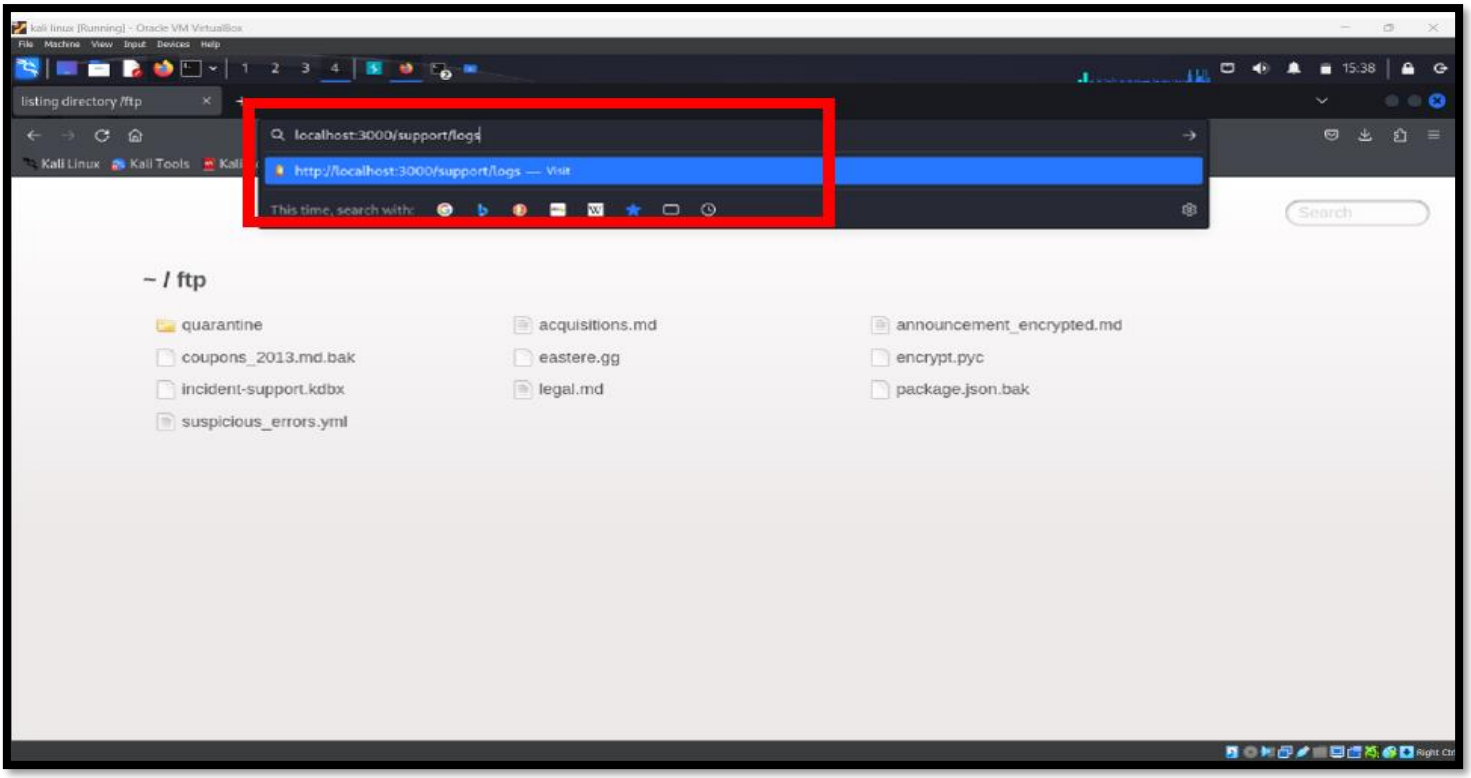


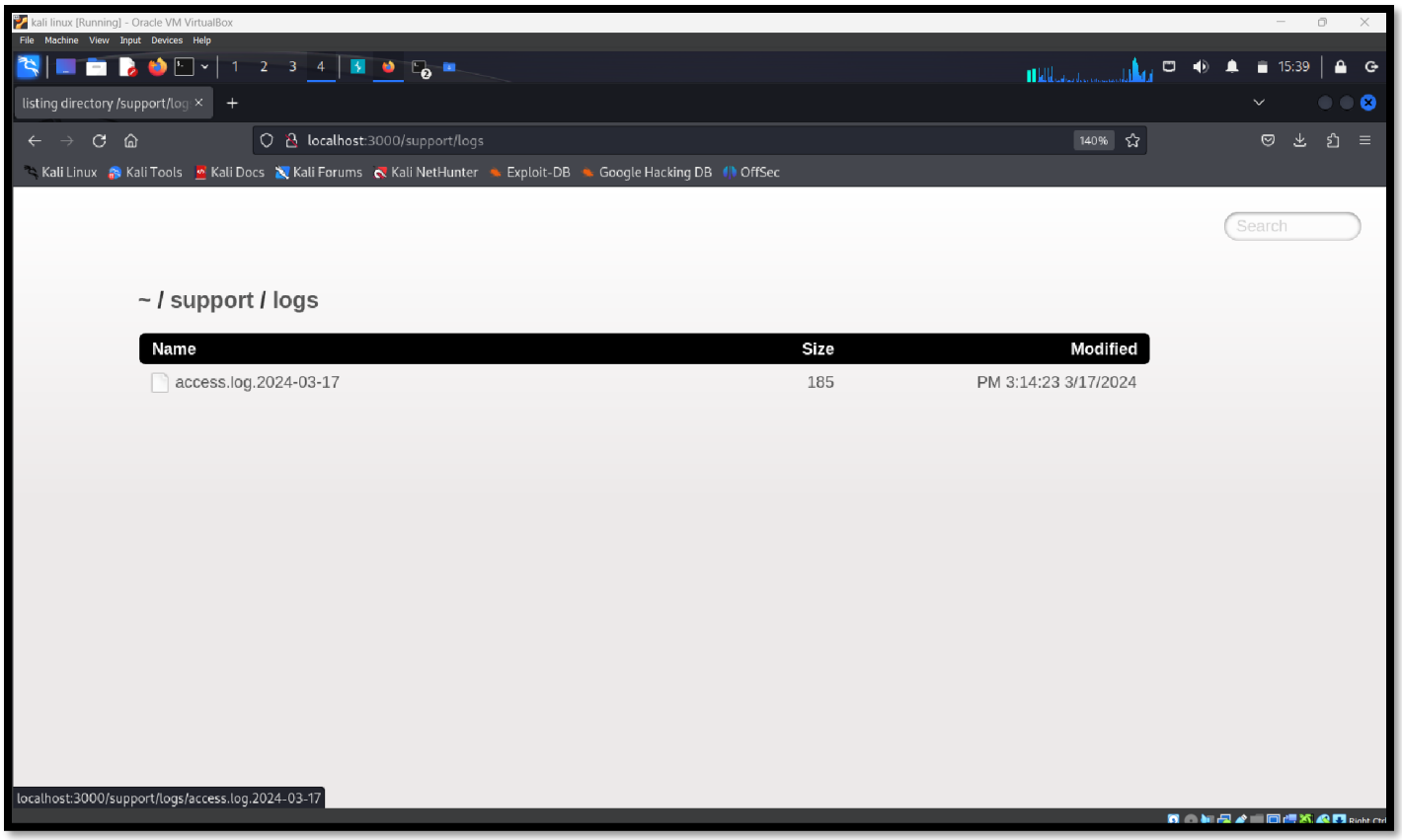
download the file and inspect it. However, no application worked for this type of file.



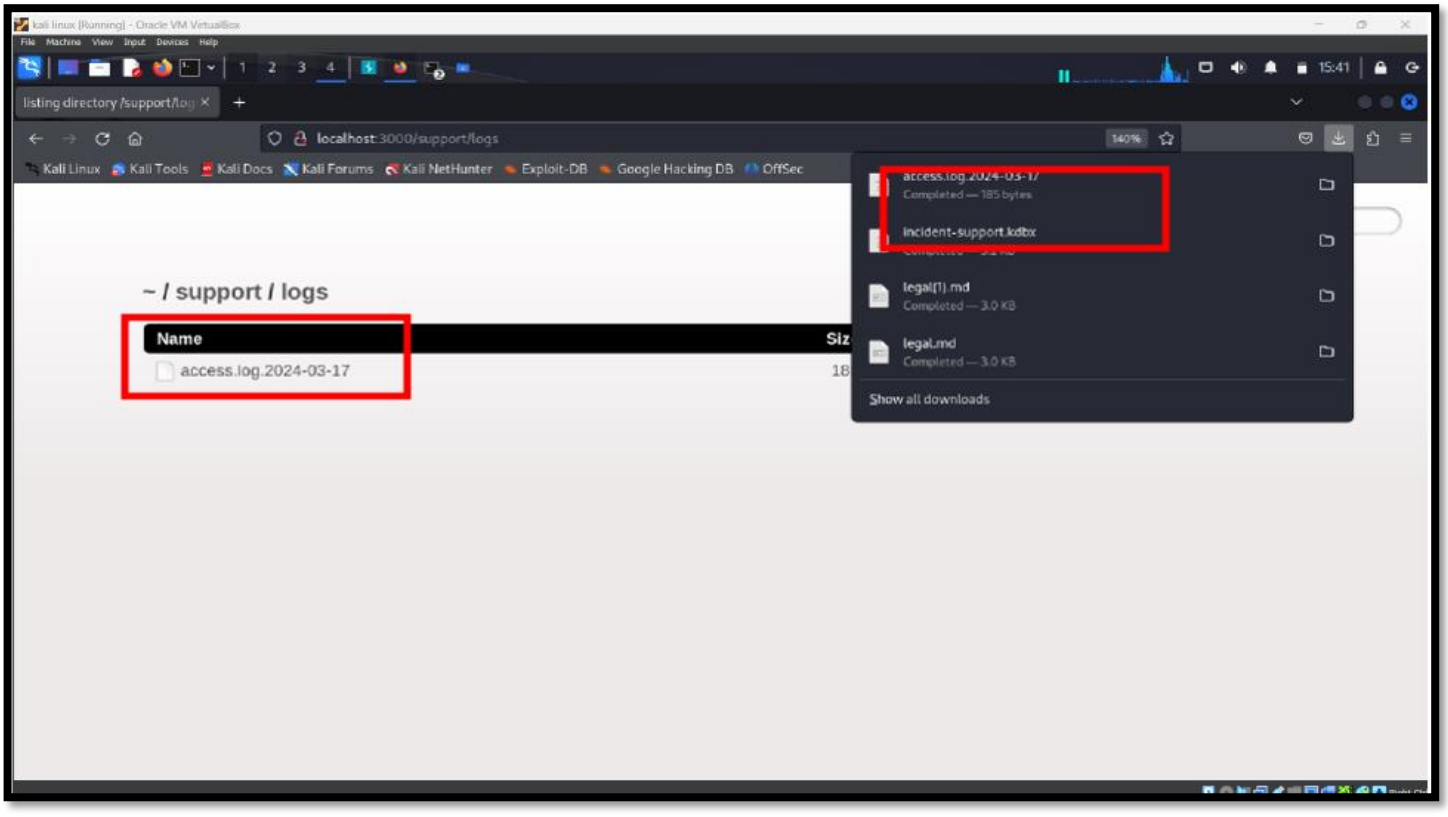
We used the cat command to open this file in the terminal and found its contents.

We then tried to open the support directory and found 1 access log files.

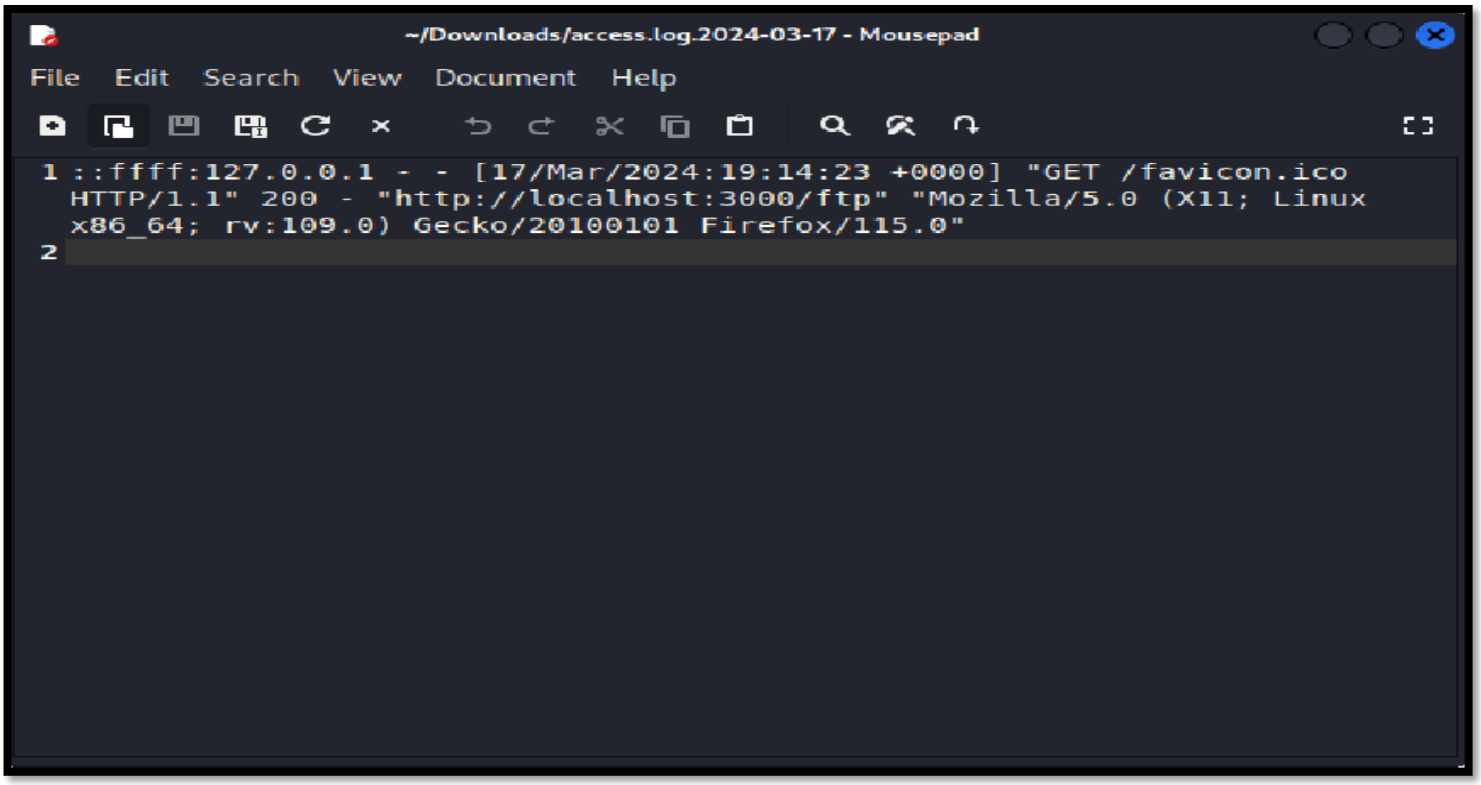




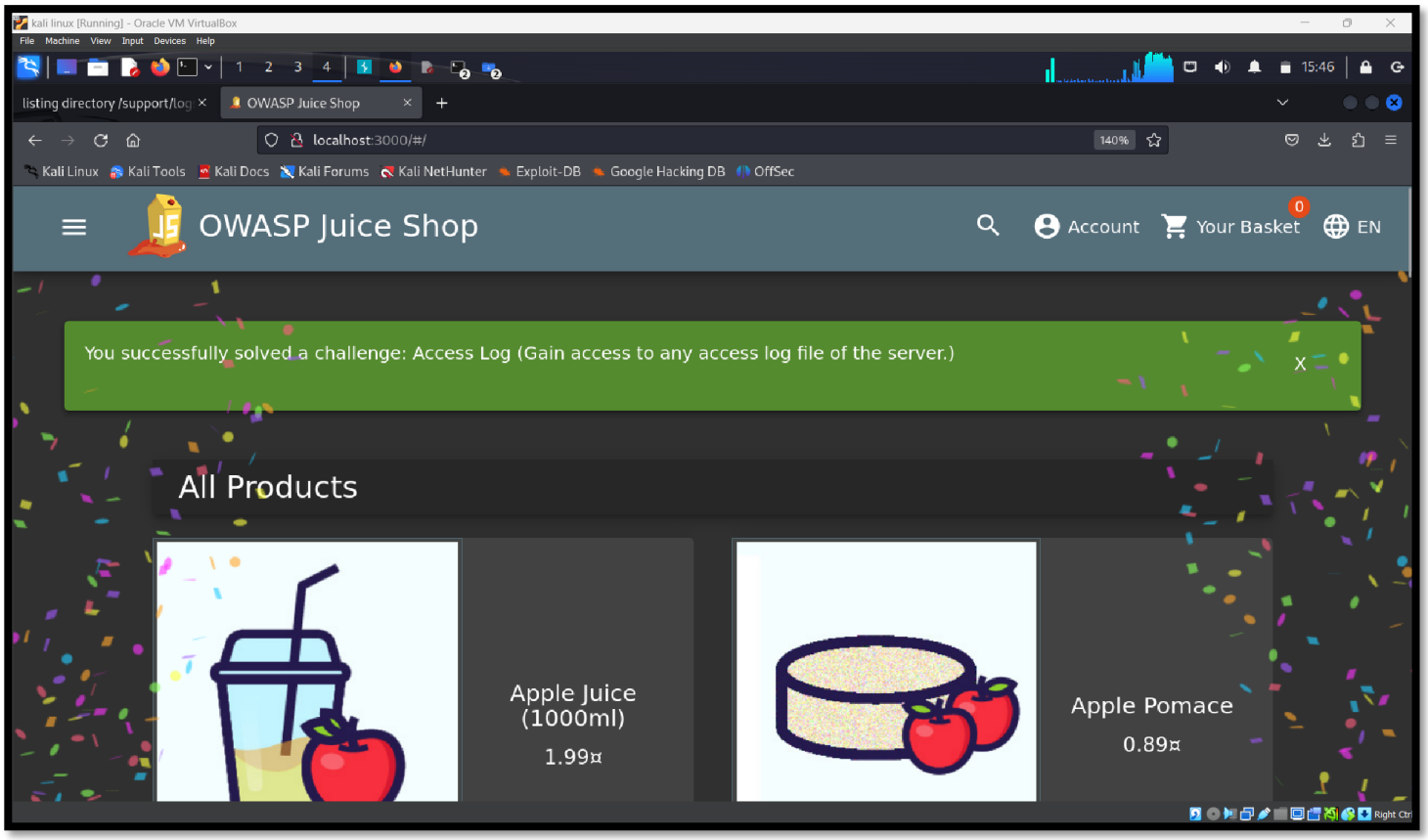
We were able to access and download these files.



Upon downloading the access log files, we found this type of data in them.



We then went back to the OWASP juice shop home page and confirmed that we had successfully solved this challenge in the Juice Shop.



Vulnerabilities Exploited by Task 2.2:

• **Inappropriate file permissions**: Allows illegal access to log files and sensitive data.

• **The lack of file type validation:** Enables potentially sensitive files accessible.

How Vulnerabilities Can Be Fixed or Addressed:

- Strict Access Controls for Sensitive Documents
- Implement strict access controls.
- Use file type validation for approved file viewing/download.
- Protect sensitive documents/log files in secure, encrypted locations.

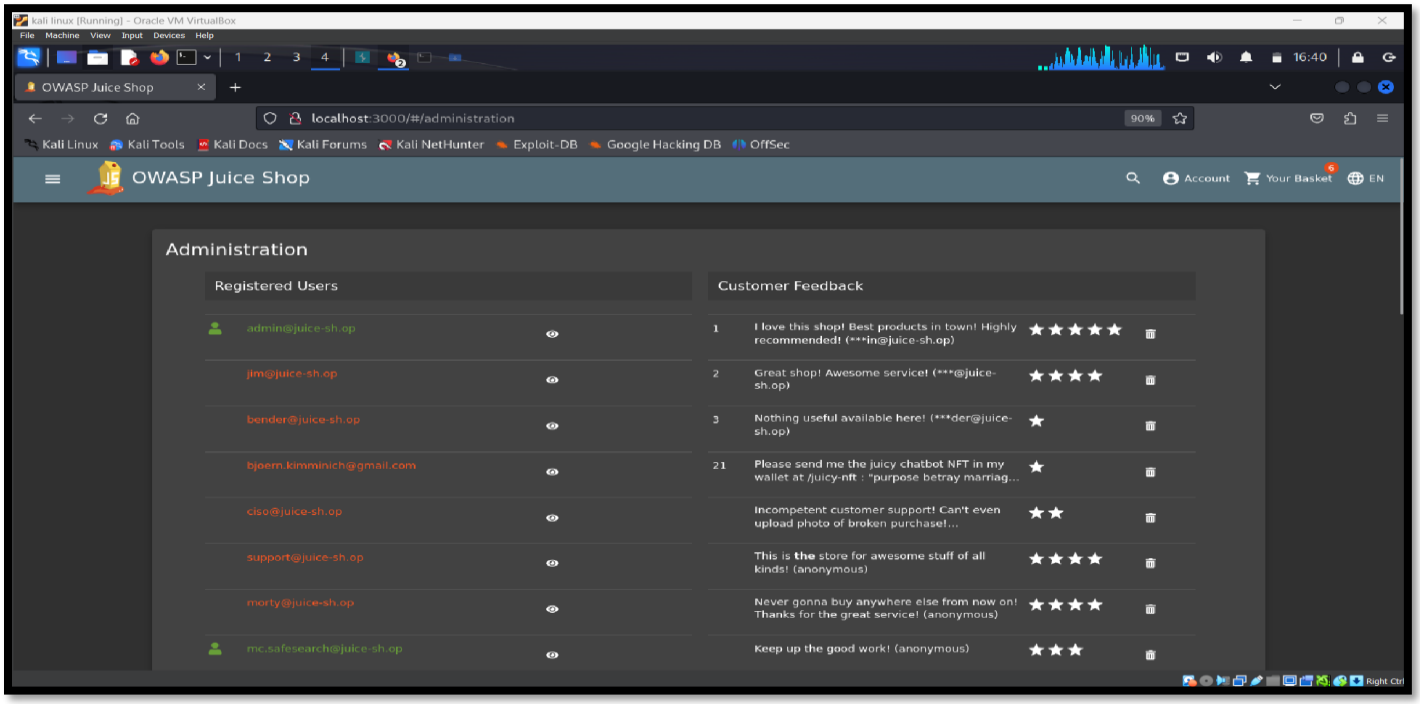
Real-World Scenarios:

- Unauthorized access to confidential documents and log files.
- Potential consequences include data breaches, regulatory fines, and reputational harm.
- Cloud storage service providers may be targets.
- Malicious attackers can bypass proper authorization and access directories.

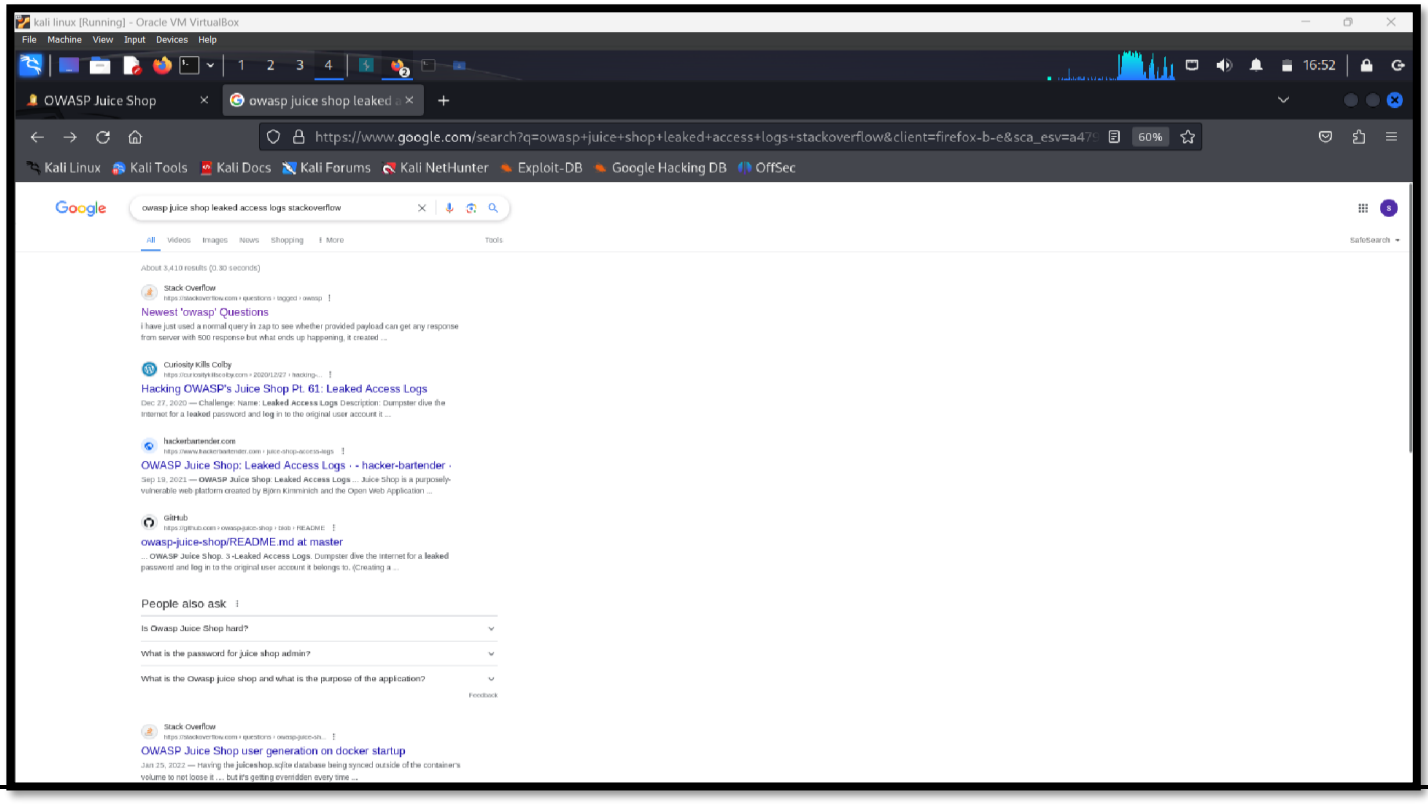
TASK 3

Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to

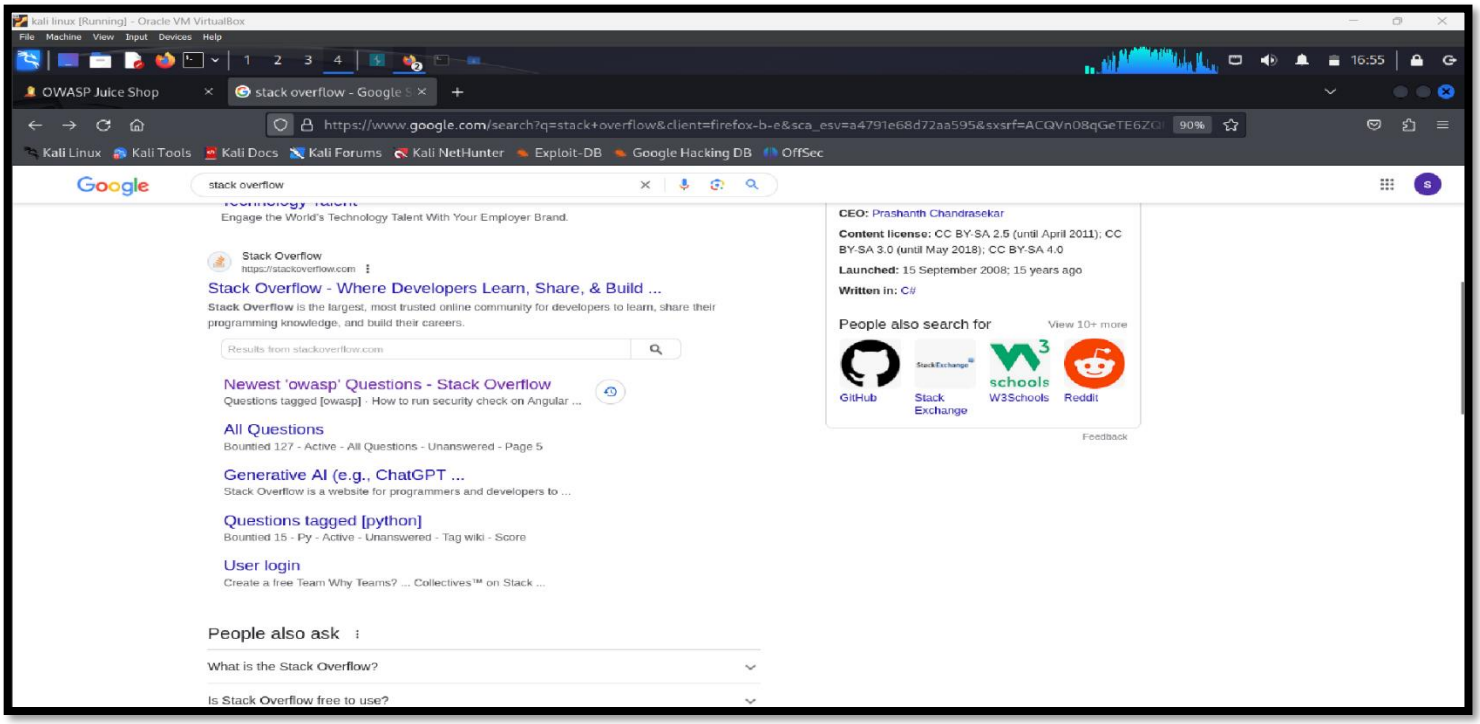
We were logged in to the OWASP Juice Shop with the Administrator's user credentials which were caused by the Sensitive Data Exposure. We entered the administrator's credentials in the "Log in Box by tracking the email address and password.



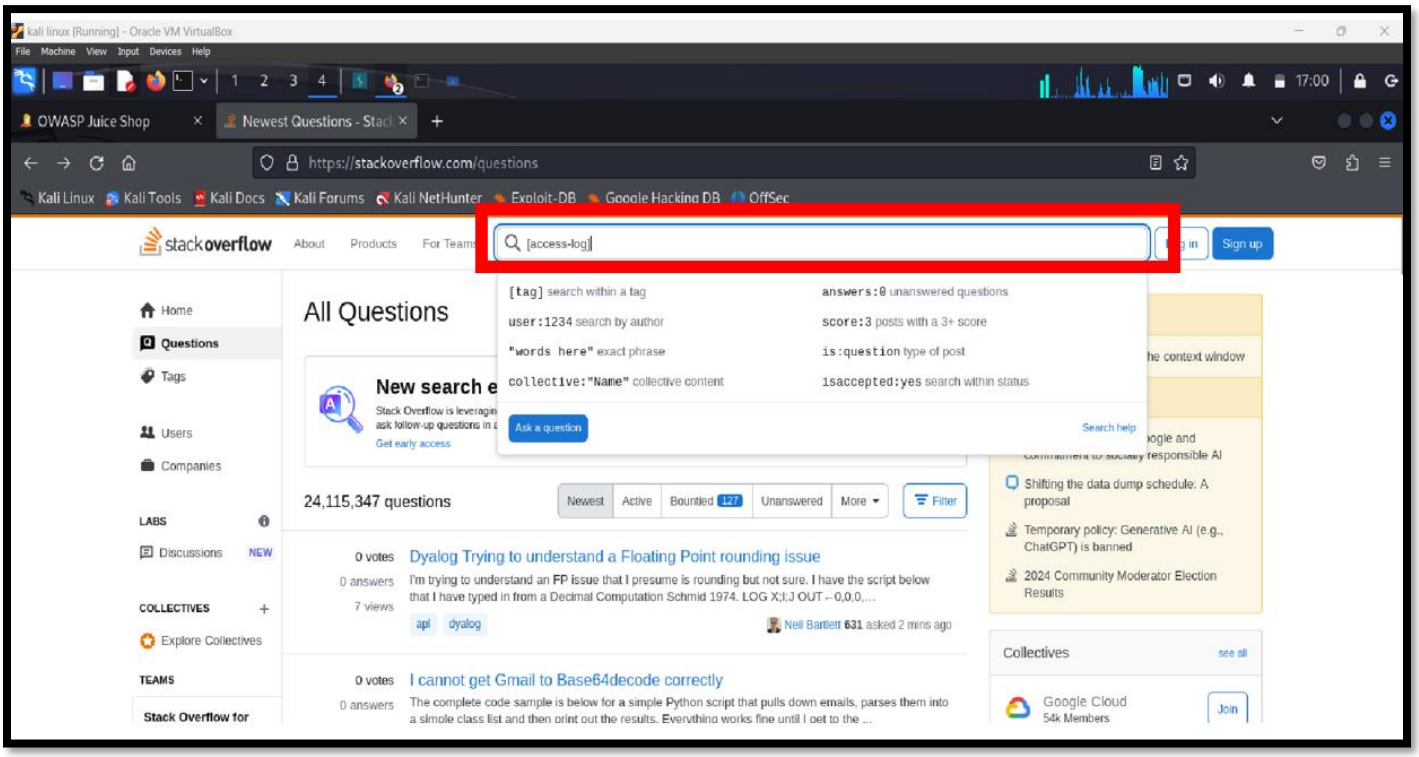
To solve this challenge firstly we searched on Google where we saw a bunch of links related to that but that's not what we were looking for.



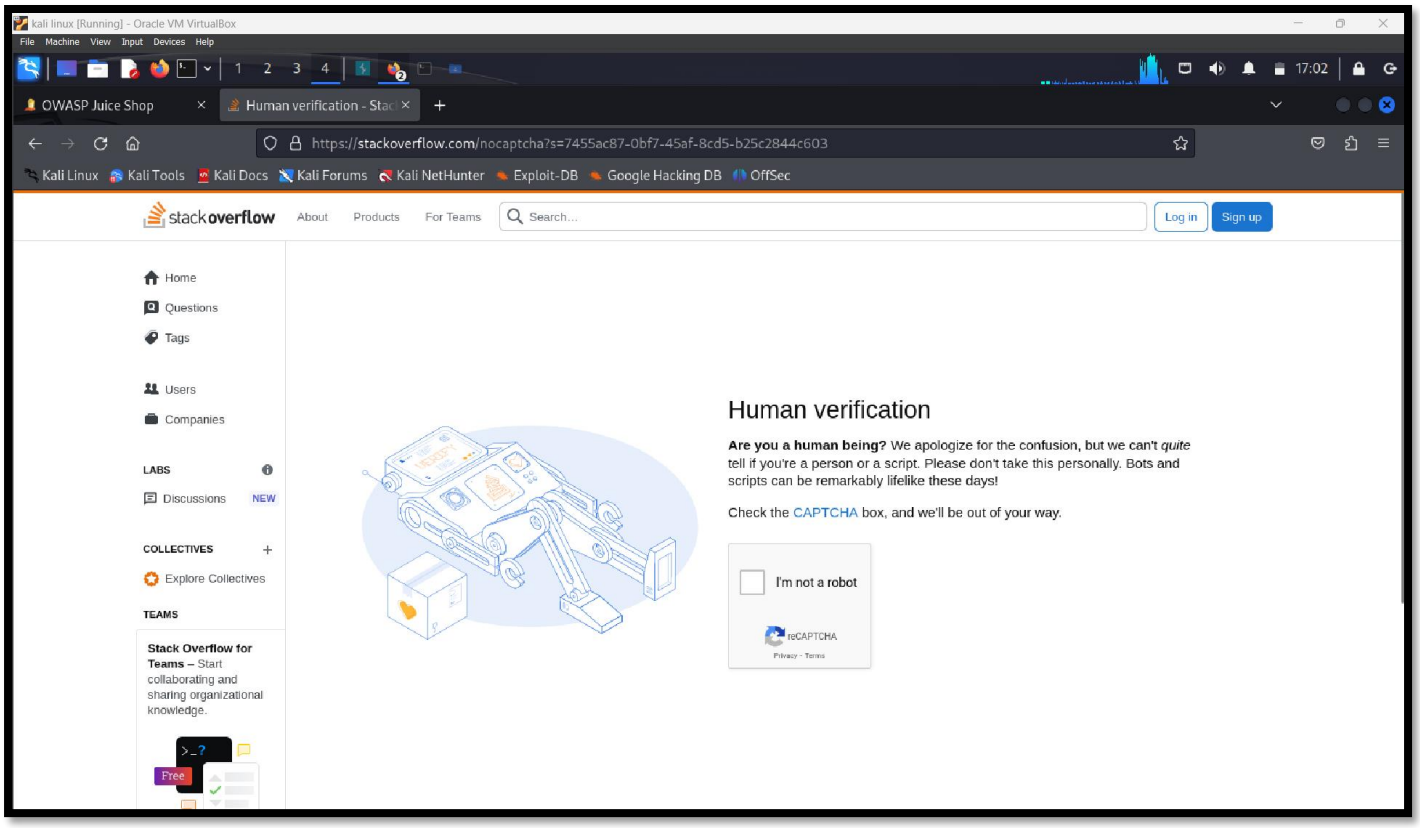
so we did some research on it. We found out that we had to go to a Stack Overflow solution in the Firefox browser.
Then we went to the question, then we searched with the square brackets which happened to be a tag in



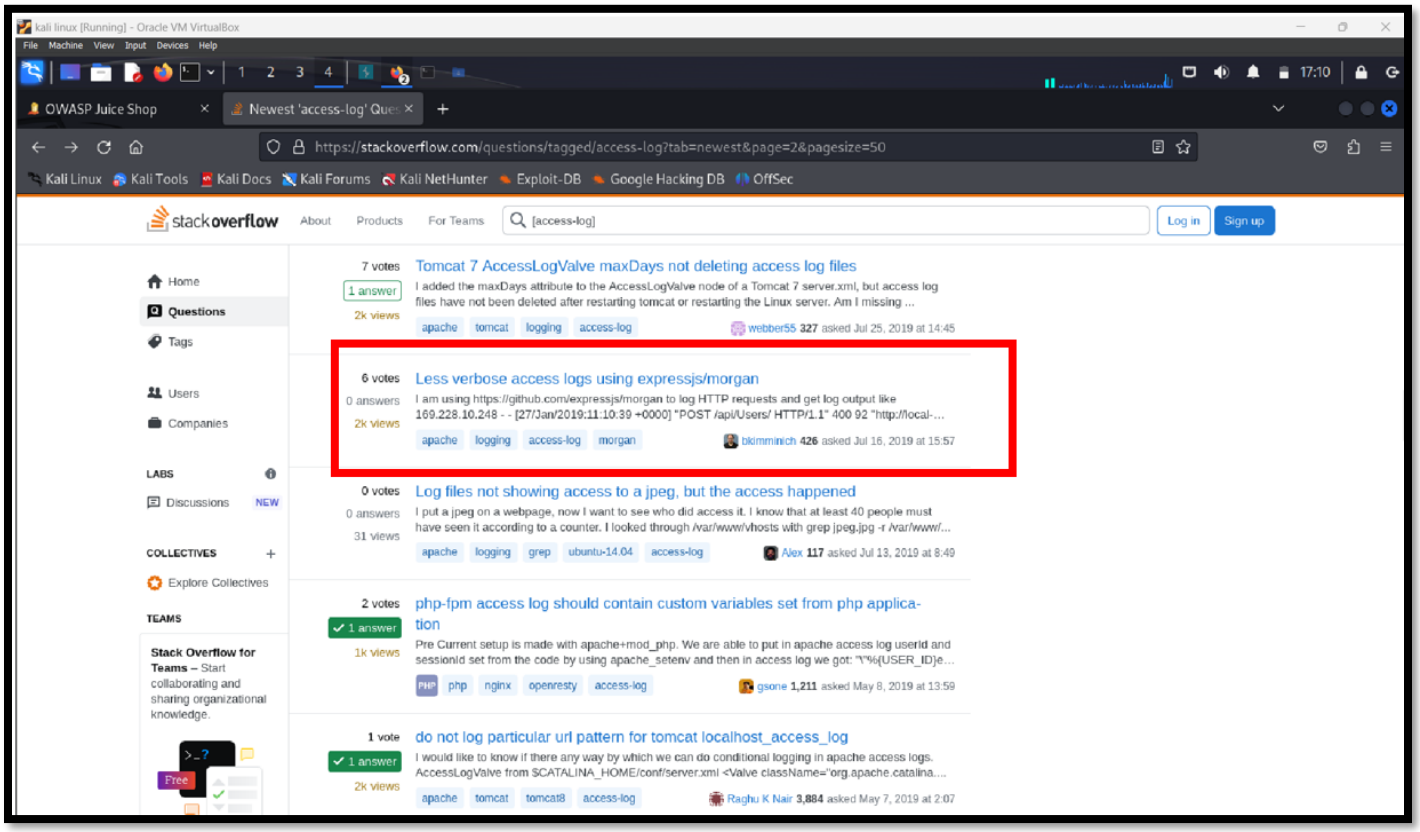
Stack Overflow and we typed "access-log" and then we closed the square bracket. We looked for it.



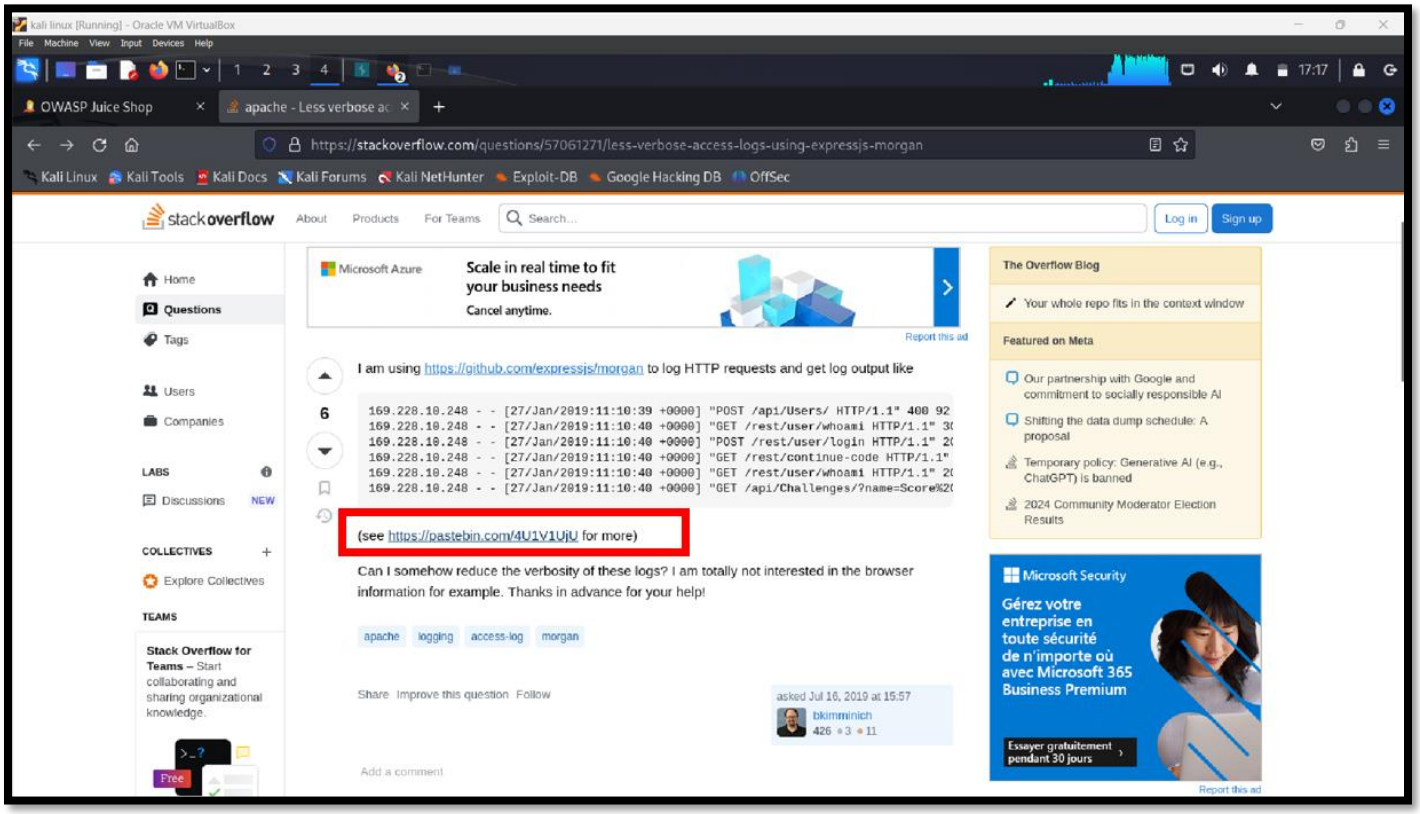
It asked about human verification. We solved that captcha



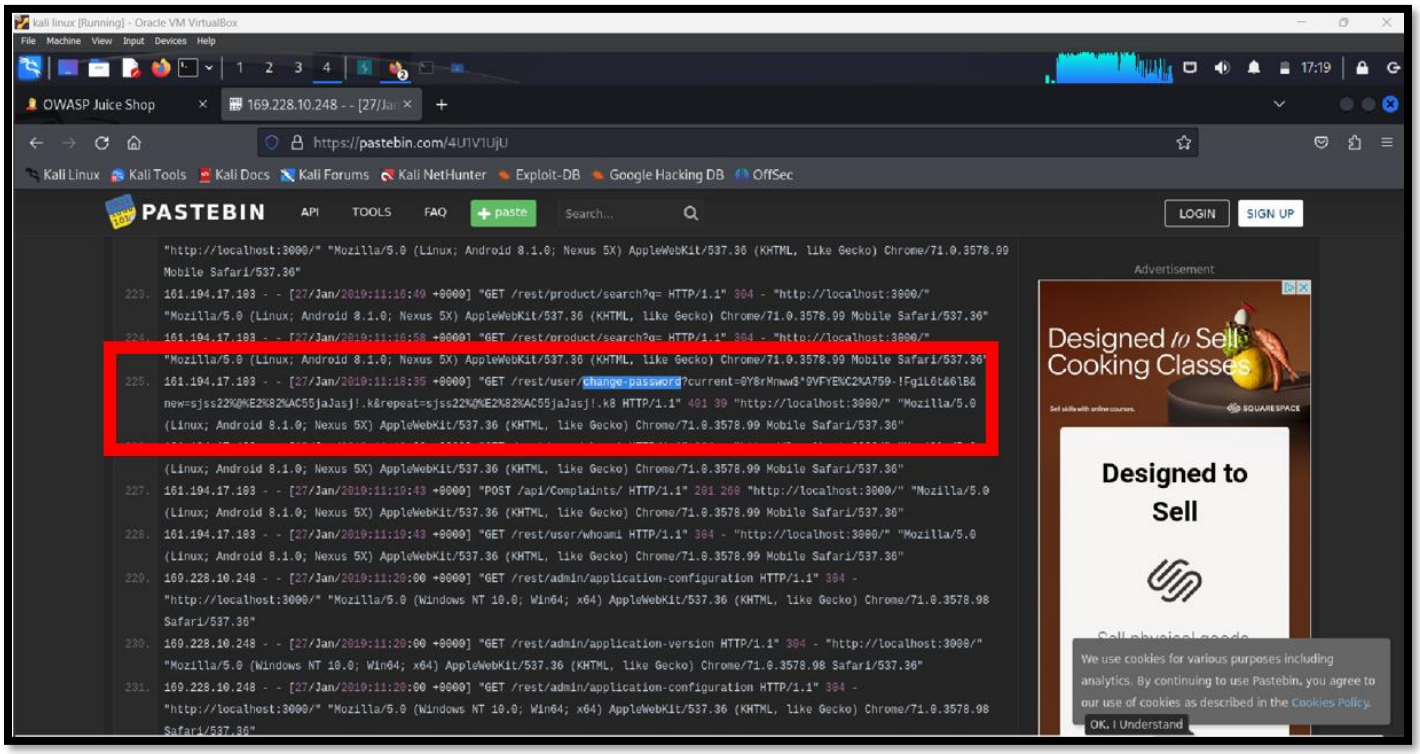
then scrolled through multiple solutions, where we finally found something that was relevant to our task.

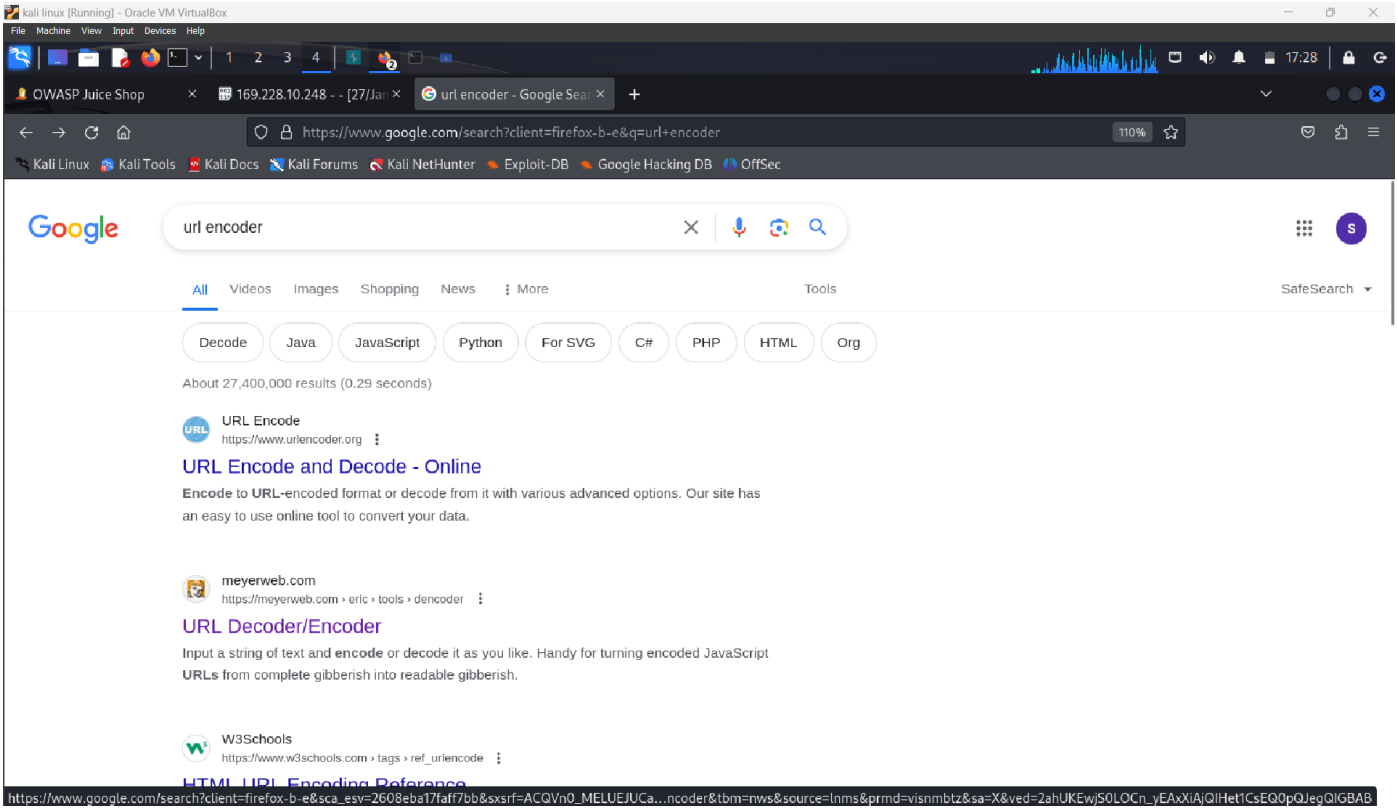


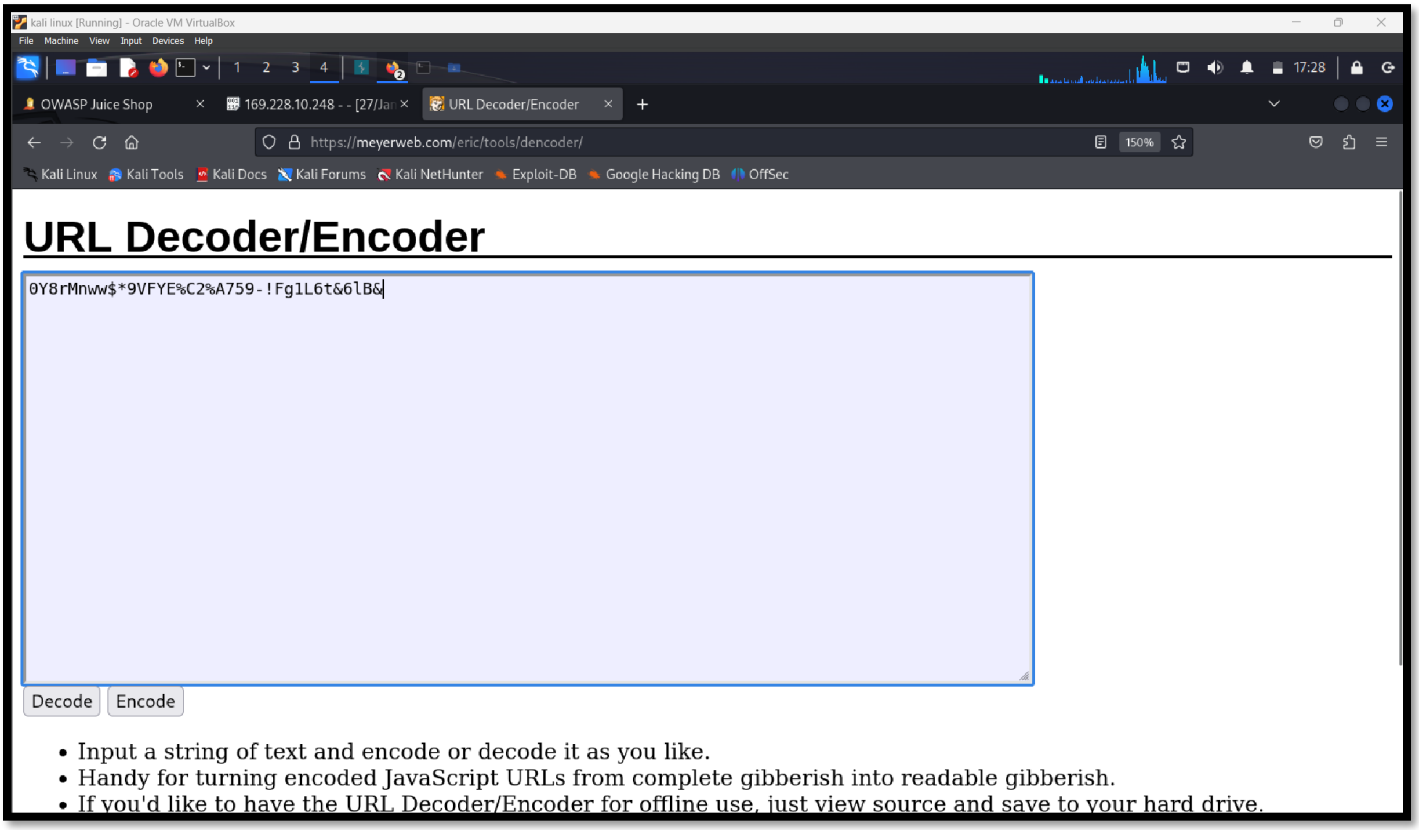
We saw there was localhost:3000 related information so we opened that solution. Then we found a link to Pastebin.



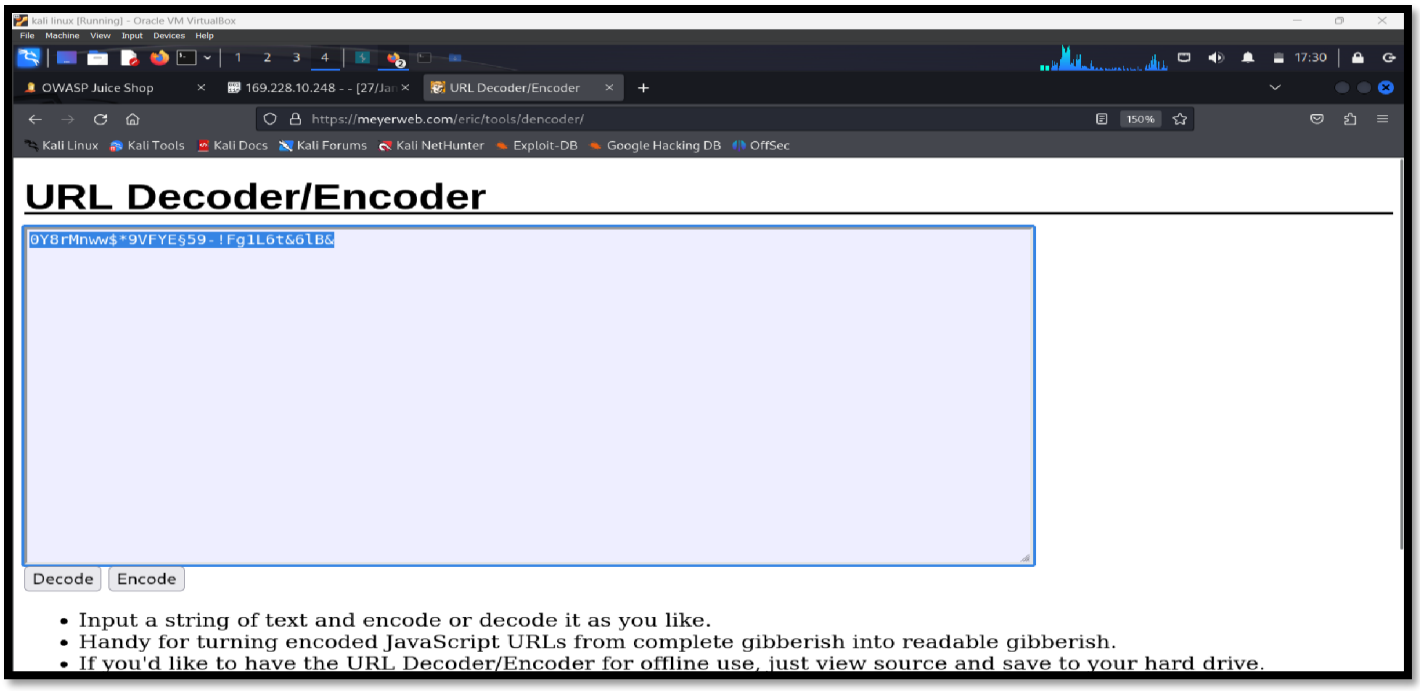
When we went over that link, we found different events and timestamps, each event occurred, so what we were basically looking for in this was log dump, is a keyword is password, so here we found out the password. So, it's invoking a get request for the current password to be this and then the new password would be changed over to this



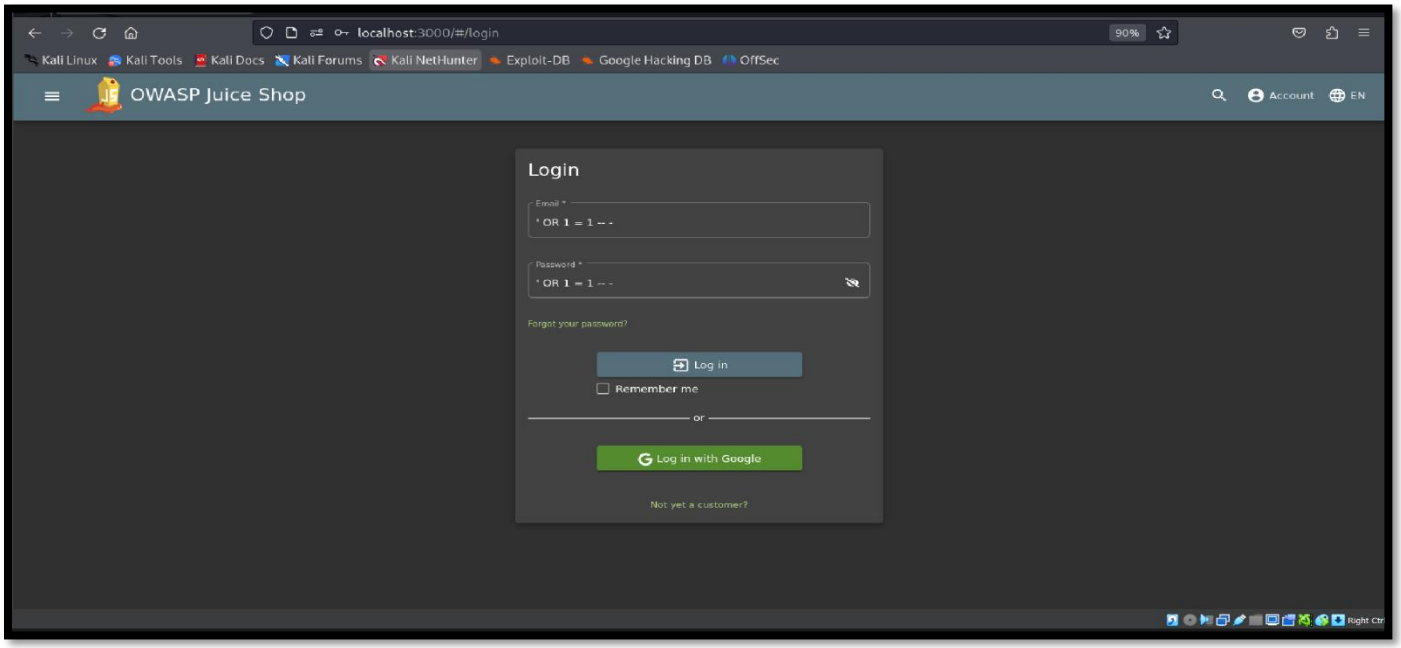




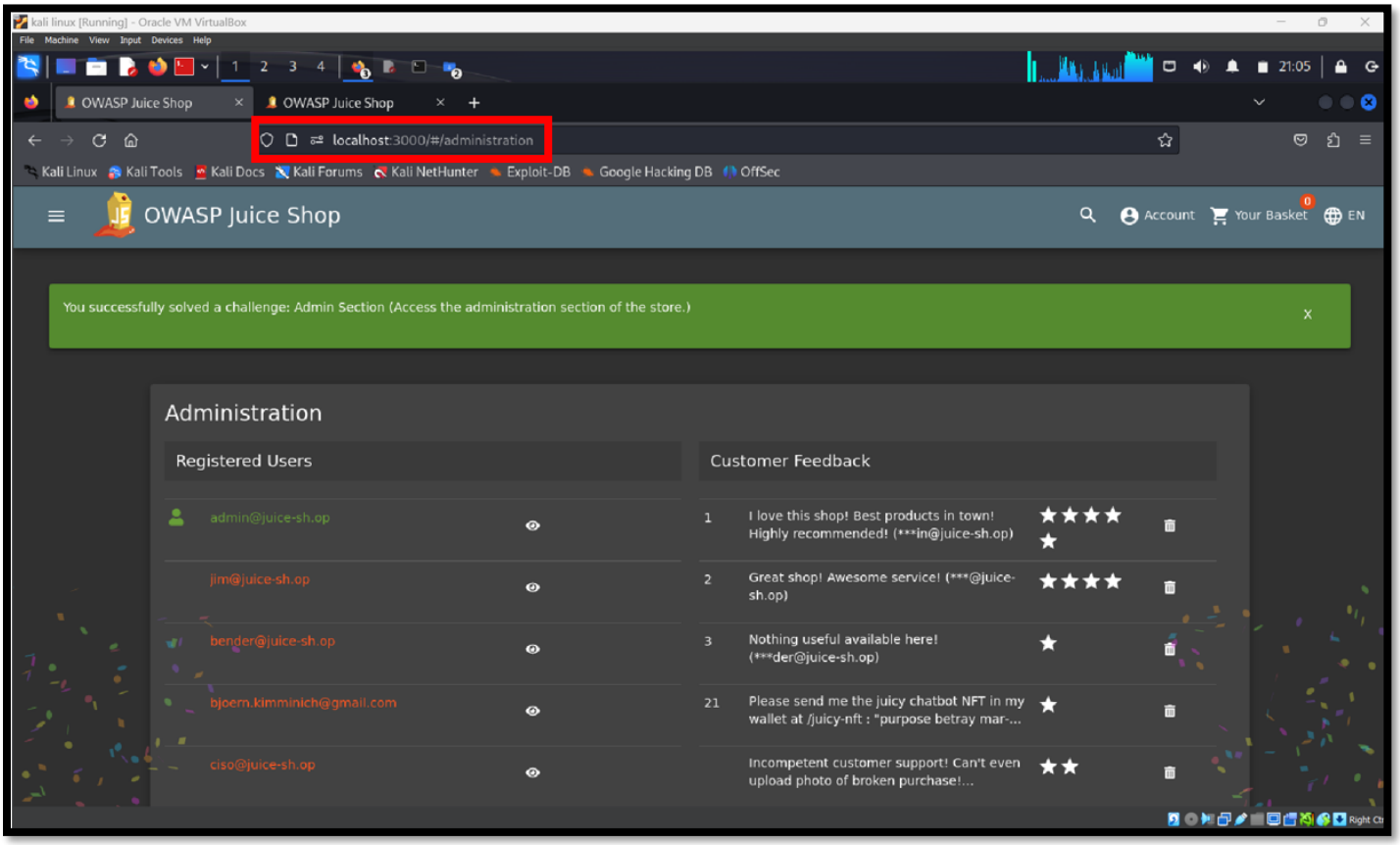
so we copied that password and went to URL encoder/decoder and pasted that. After we clicked on the decode button so finally we got the decoded password.



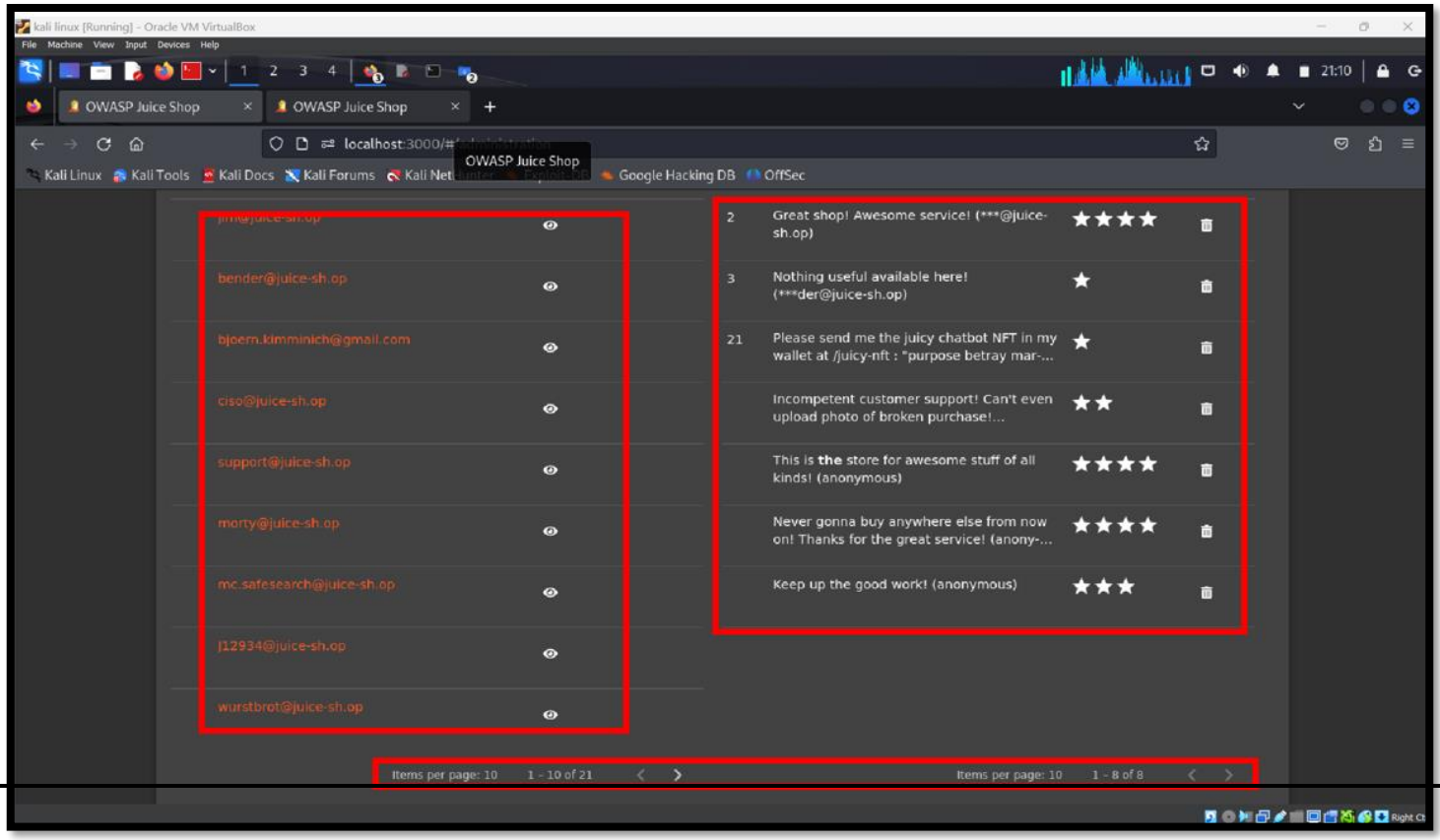
We tried to log in as an administrator by using the SQL injection. We clicked on the account and logged out. We logged in again, on the login page to input the email and password.

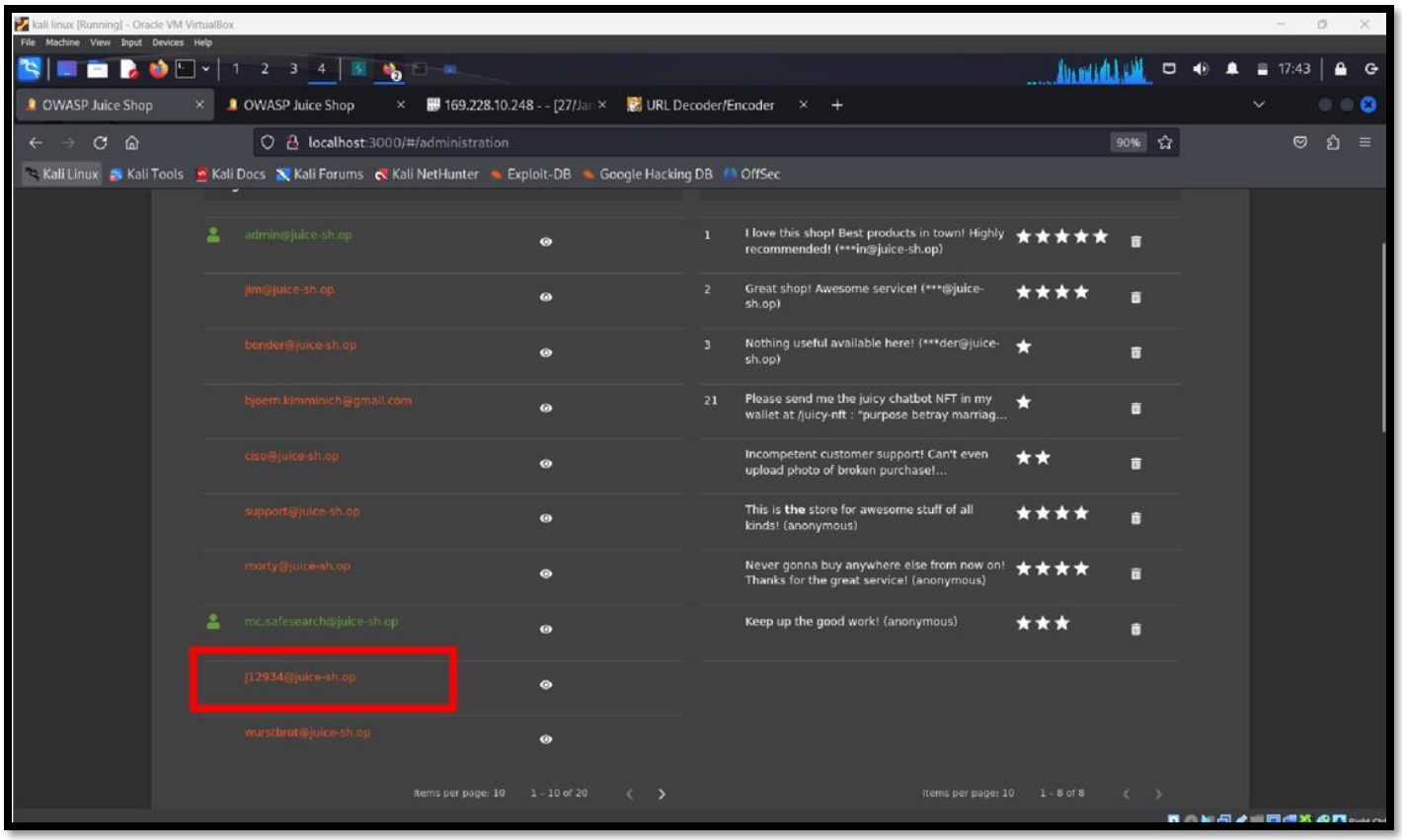


We searched the administration on localhost.

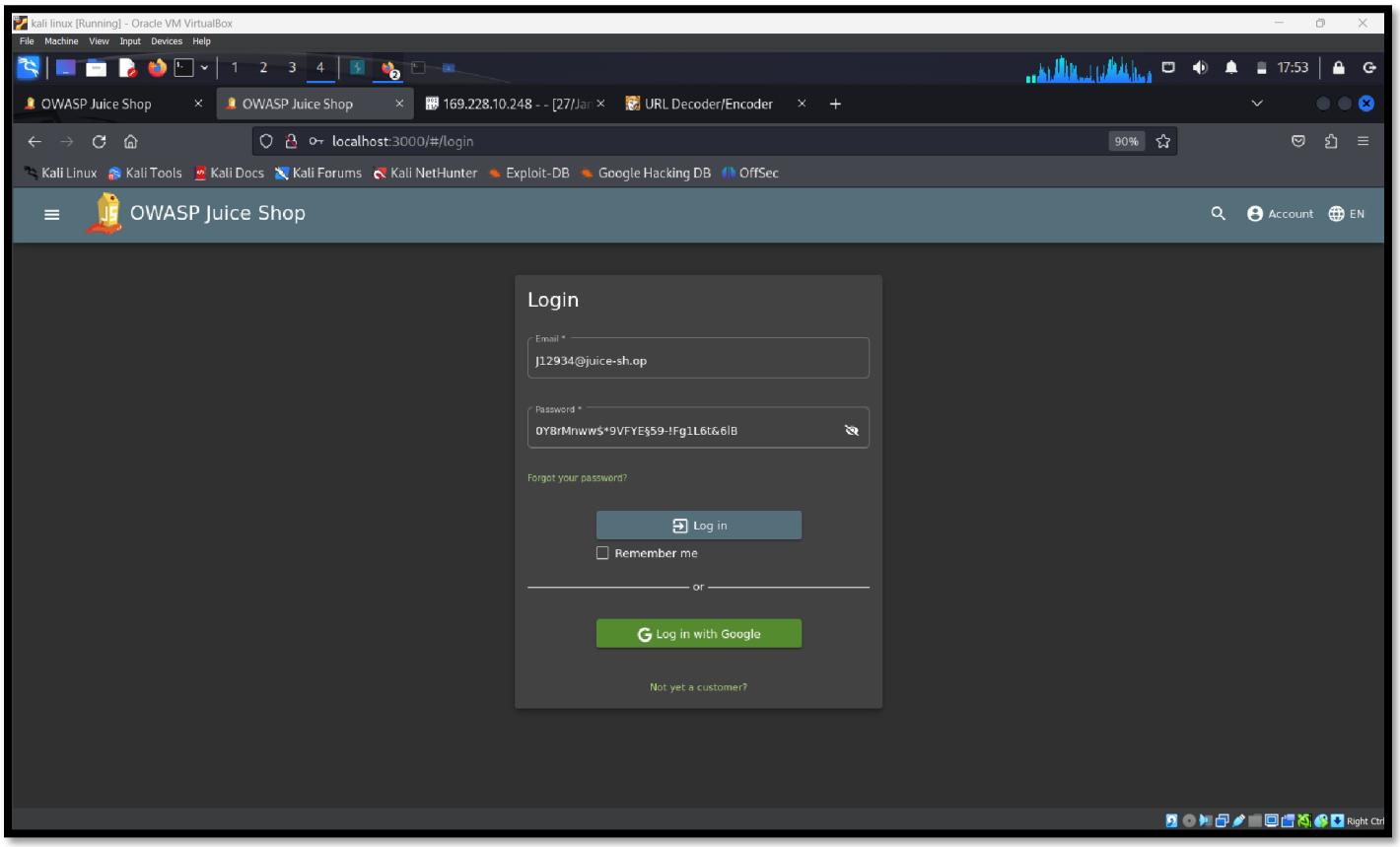


Here we found the list of all registered administrators. All we needed to do was match which user's hashed password was this.

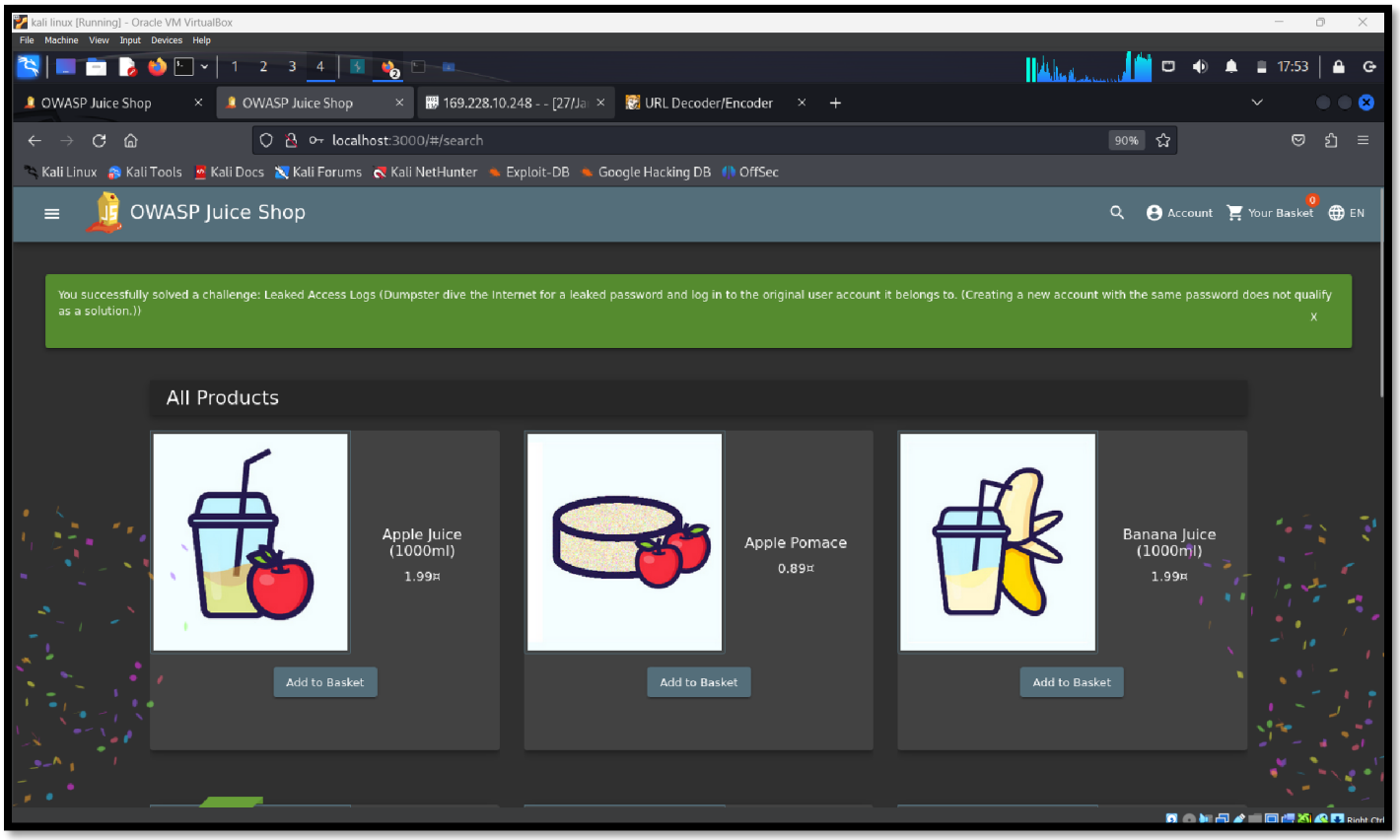




We manually matched one by one all the emails of administrators with that password and the email j12934@juice-sh.op matched with that.



With these credentials we could solve this challenge successfully.



Vulnerabilities' Exploited by Task 3:

- Password exposure due to data breaches/leaks.
- Unauthorized access to user accounts.
- Risk of security breaches for users and companies.
- Hacked passwords made public online.

How Vulnerabilities Can Be Fixed or Addressed:

- Implement strong password standards requiring multi-factor authentication, unique passwords, and frequent password changes.
- Use hashing and encryption methods for secure password storage in databases.
- Regularly monitor user account activity for unauthorized access or questionable behavior.
- Educate users on password security importance and provide instructions for secure password management.
- Use threat intelligence tools to monitor and mitigate user-related password leaks.

Real-World Scenarios:

- Database breaches can lead to leaked user credentials.
- These credentials can be shared online, allowing attackers to access user accounts.
- Attackers can exploit sensitive data, compromise user privacy, and initiate financial fraud, identity theft, or unauthorized transactions.
- Organizations can suffer financial losses, reputational damage, and legal penalties.

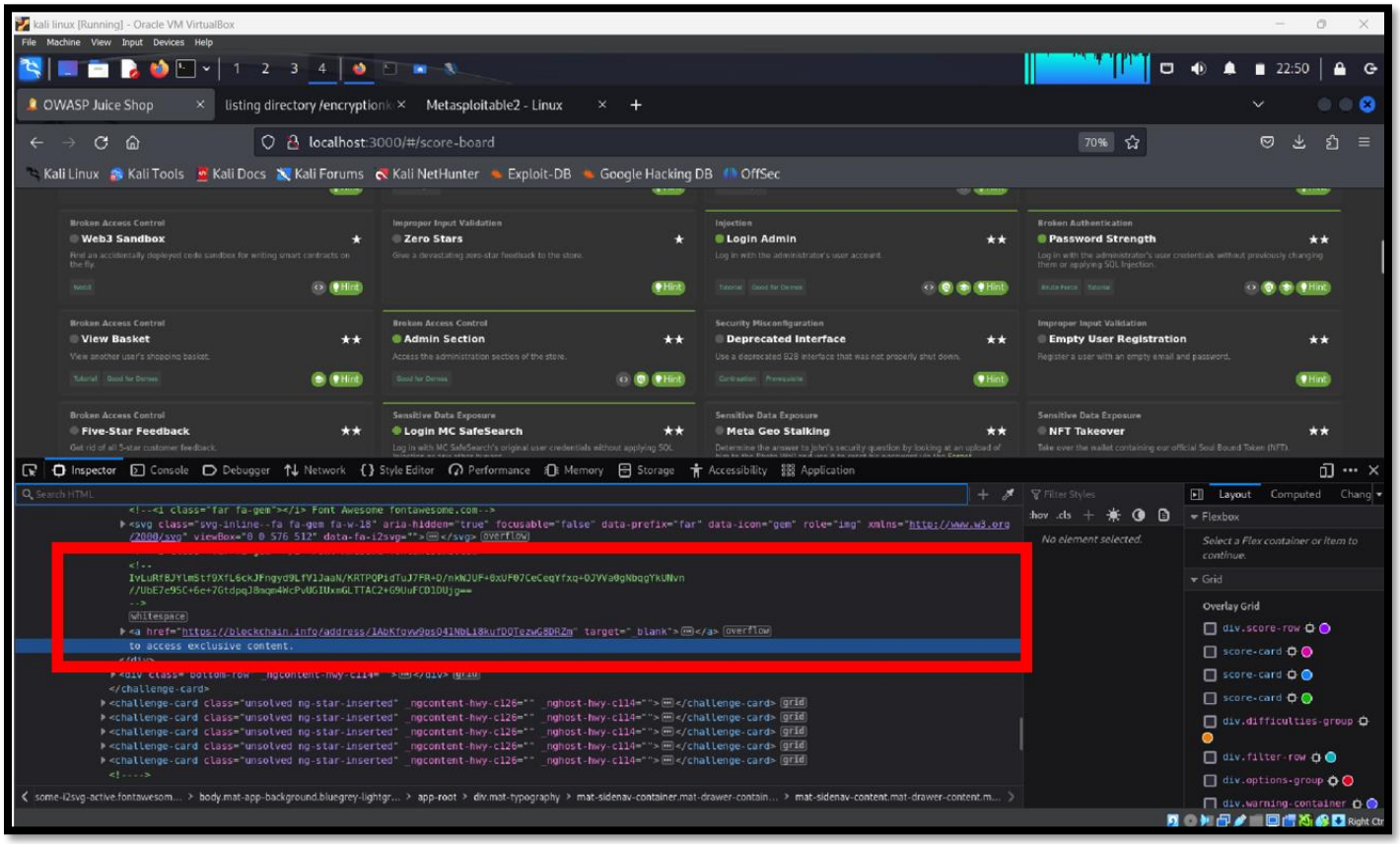
TASK 4

Unlock Premium Challenge to access exclusive content

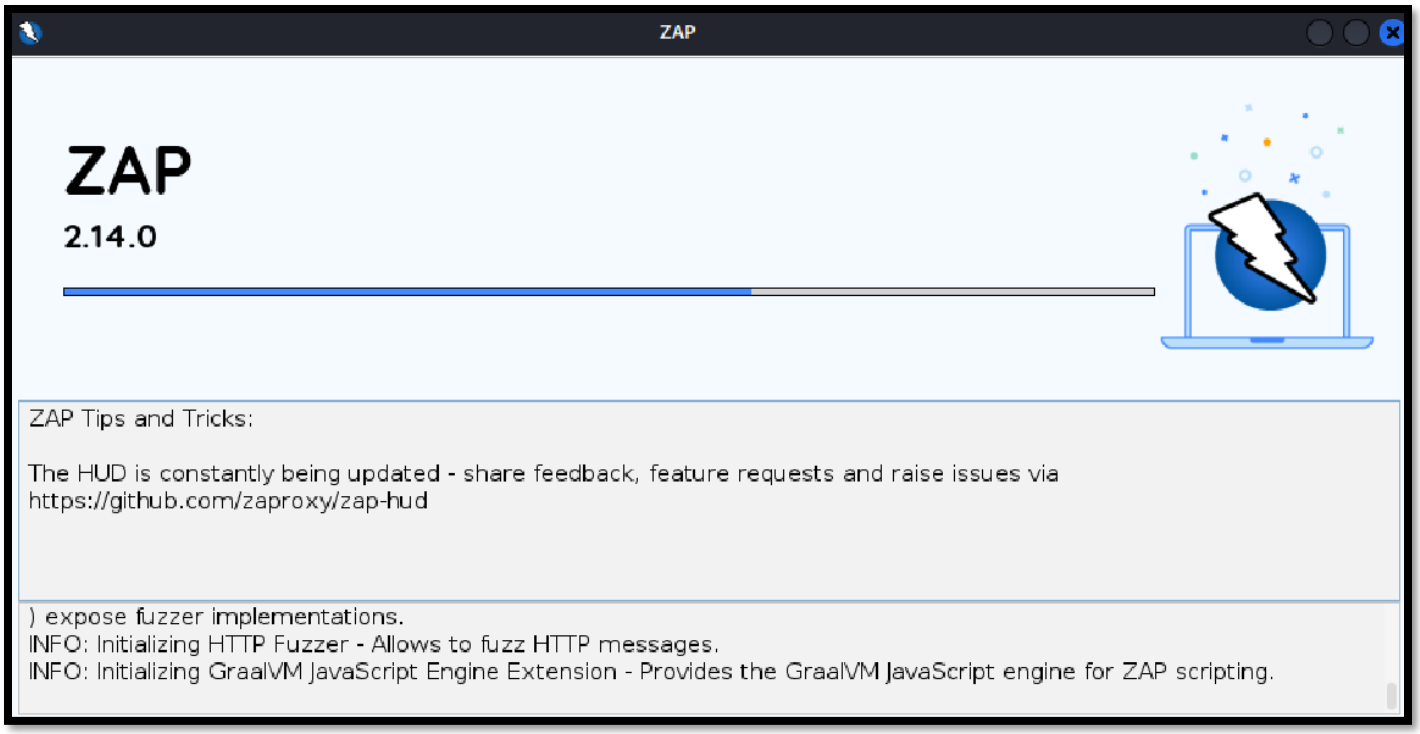
To solve that challenge, we first Inspected the HTML source of the corresponding row in the Score Board table, revealing a HTML comment that was obviously encrypted, which showed like this and in Screenshot:

IvLuRfBJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPidTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+OJVVa0gNbqg YkUNvn//UbE7e95C+6e+7GtdpqJ8m qm4WcPvUGIUxmGLTTAC2+G9UuFCD1DUjg==-->

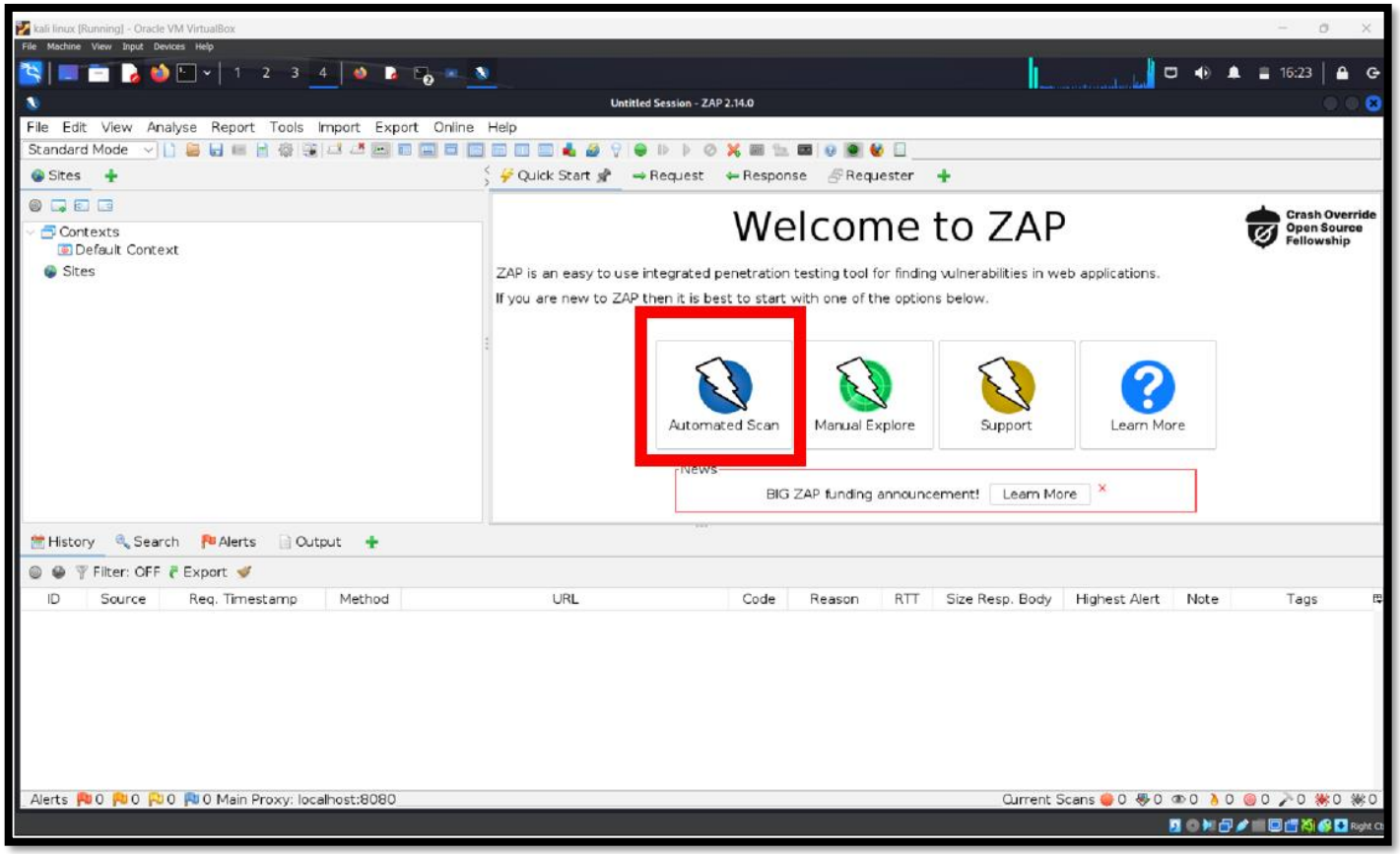
This was a cipher text that came out of an AES-encryption using AES256 in CBC mode.



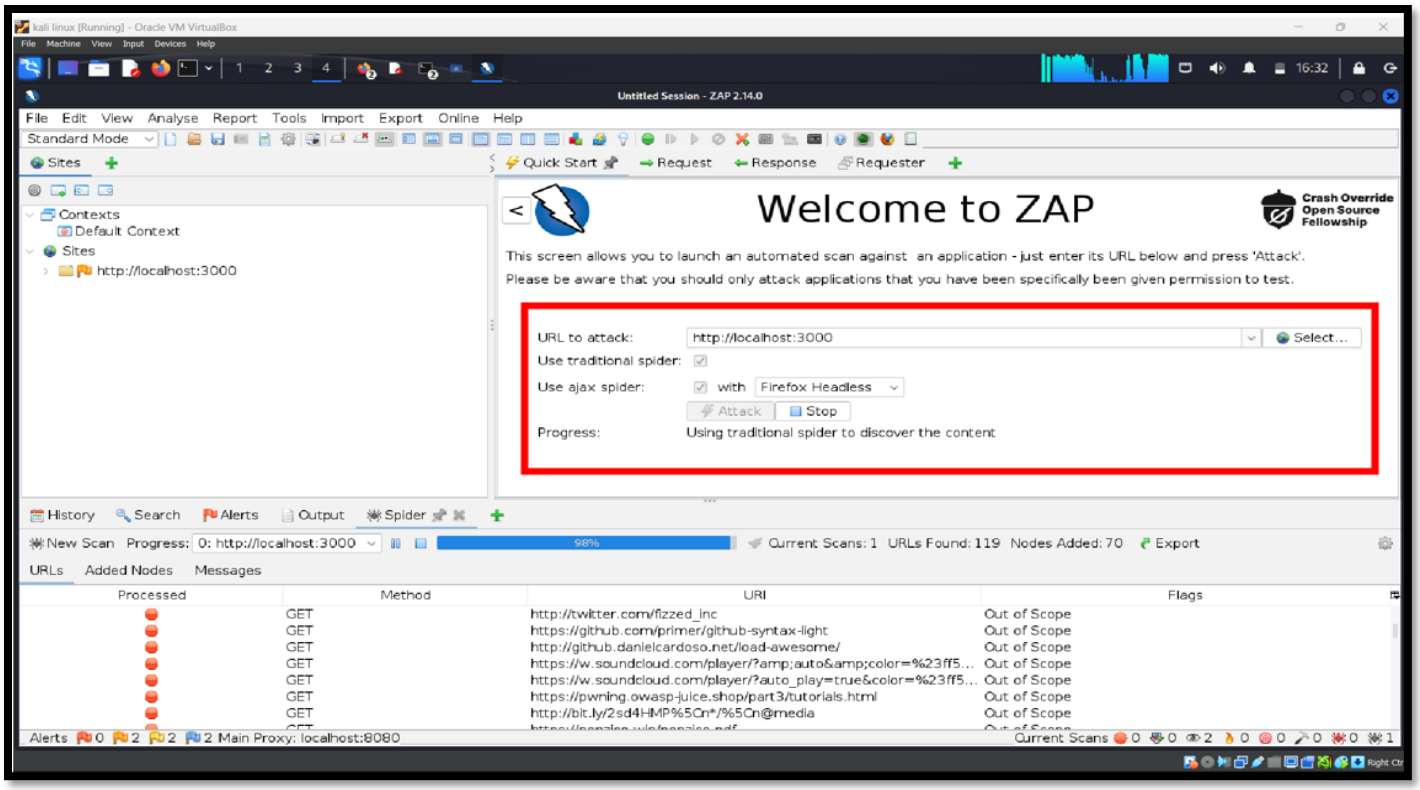
To get the key and the IV, we ran a Forced Directory Browsing attack against the application. We used OWASP ZAP for this purpose.



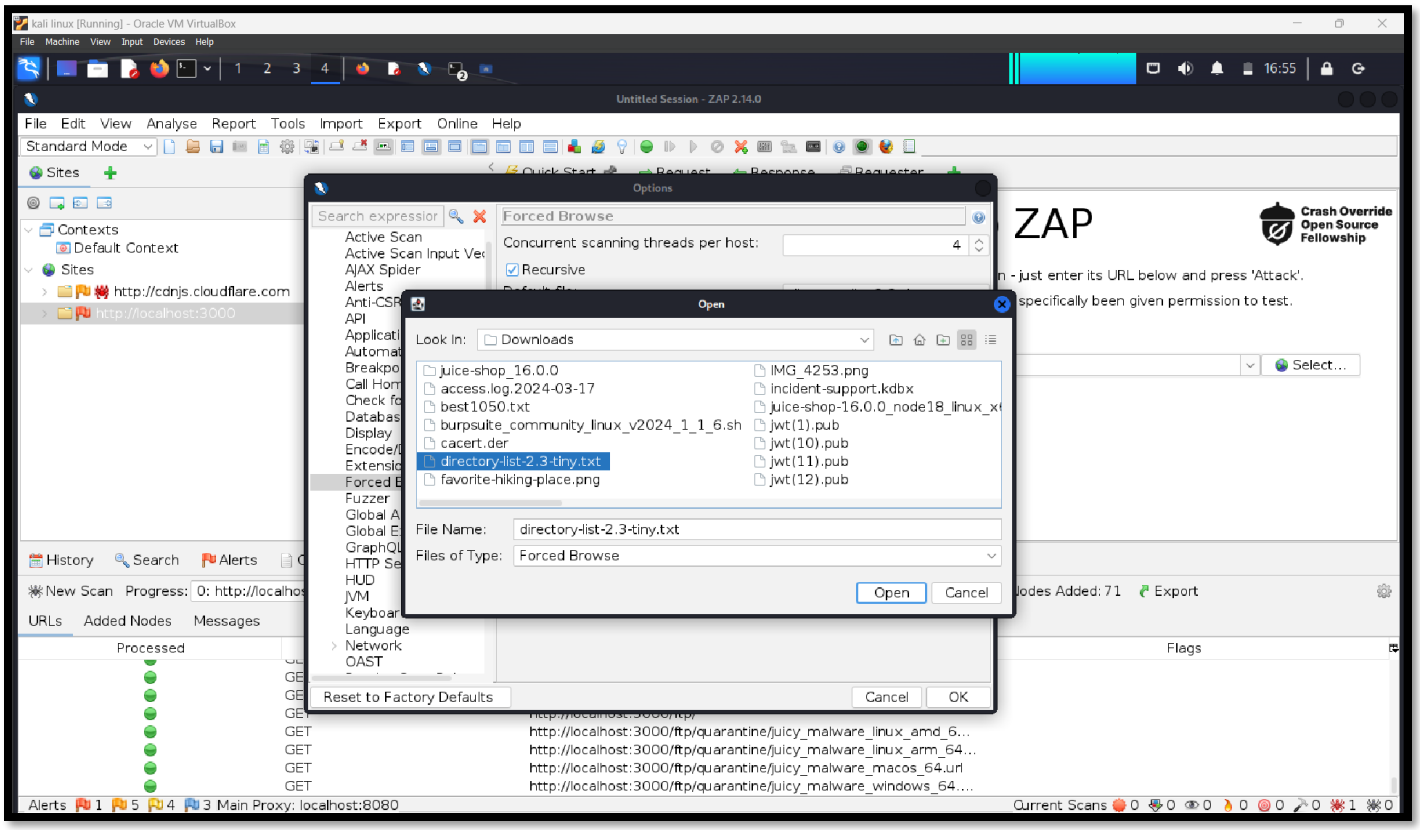
We first launched ZAP, after which we clicked on automated scan.



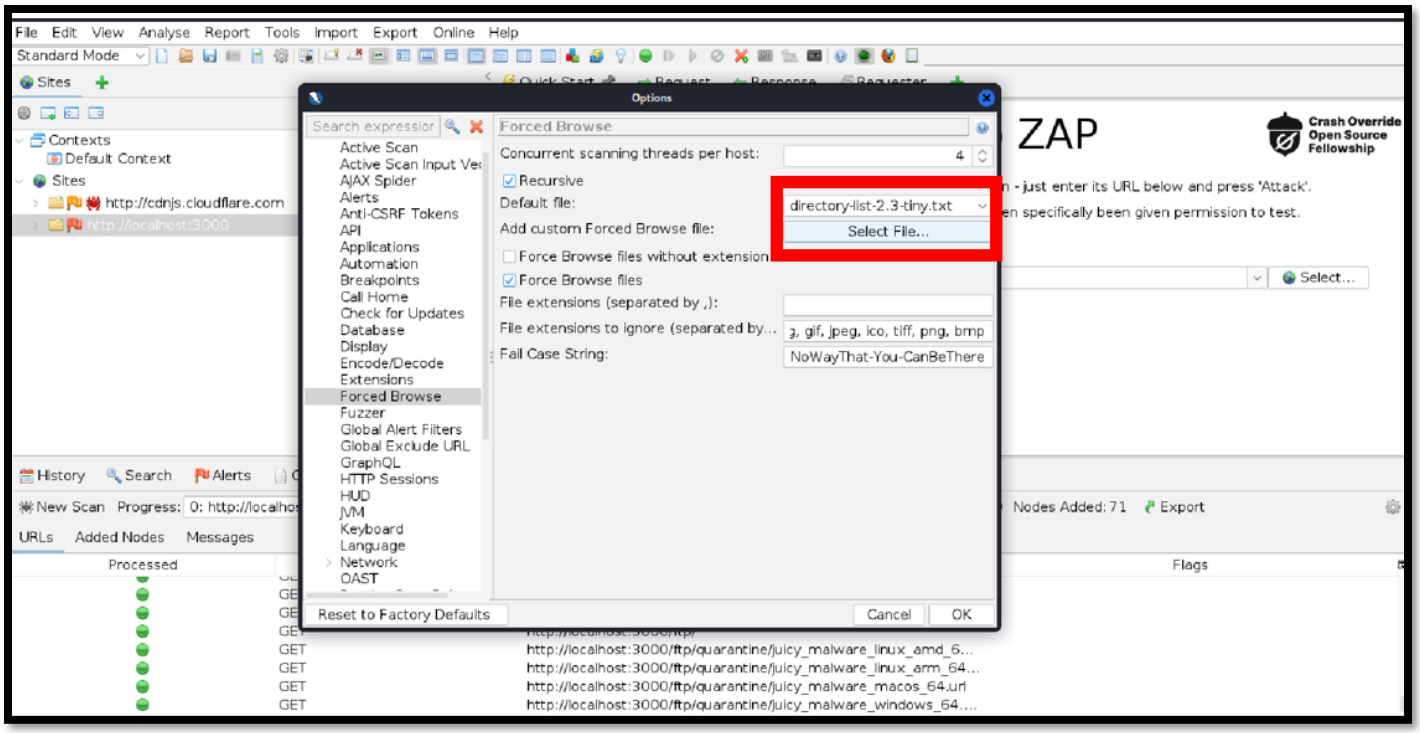
Here, in the URL, we put **localhost:3000** as our target. After that, we selected use traditional spider and used ajax spider and clicked on the attack button.

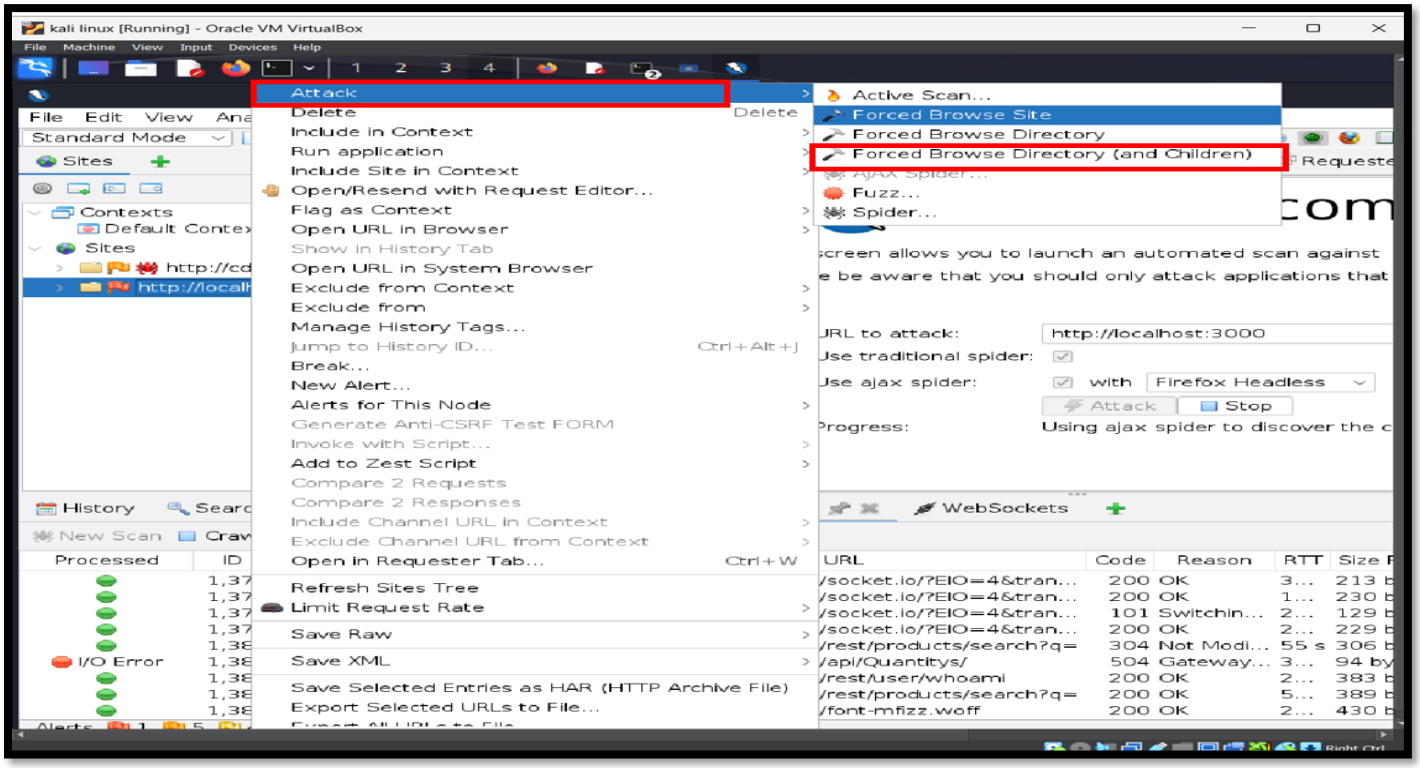


The attack started, and after successfully attacking localhost, we got directory trees on the left side under sites.

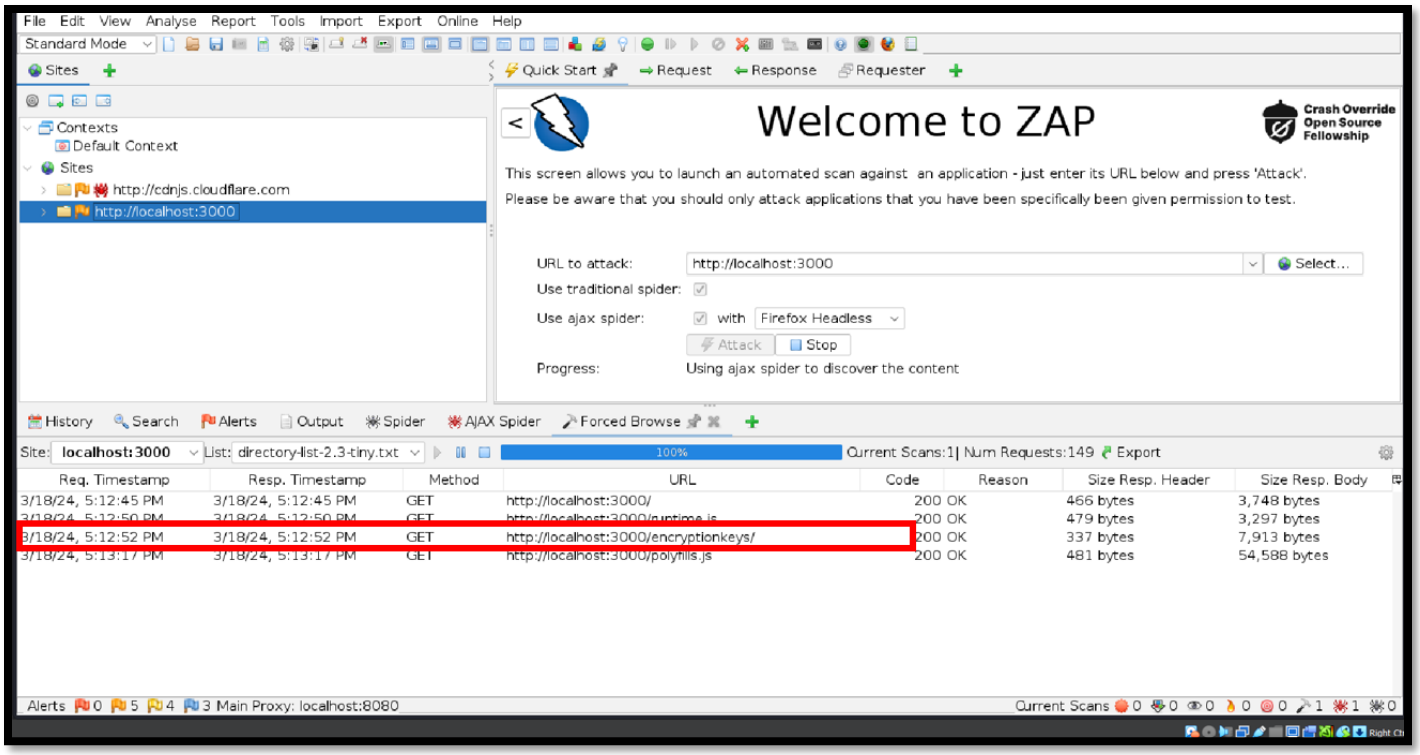


We went to the option menu, which resided under the tools menu, and clicked on forced browse configuration, where we found the option to add a custom wordlist. We downloaded the directory-list-2.3-tiny file from econestoga and added it to this custom wordlist under forced browse configuration.

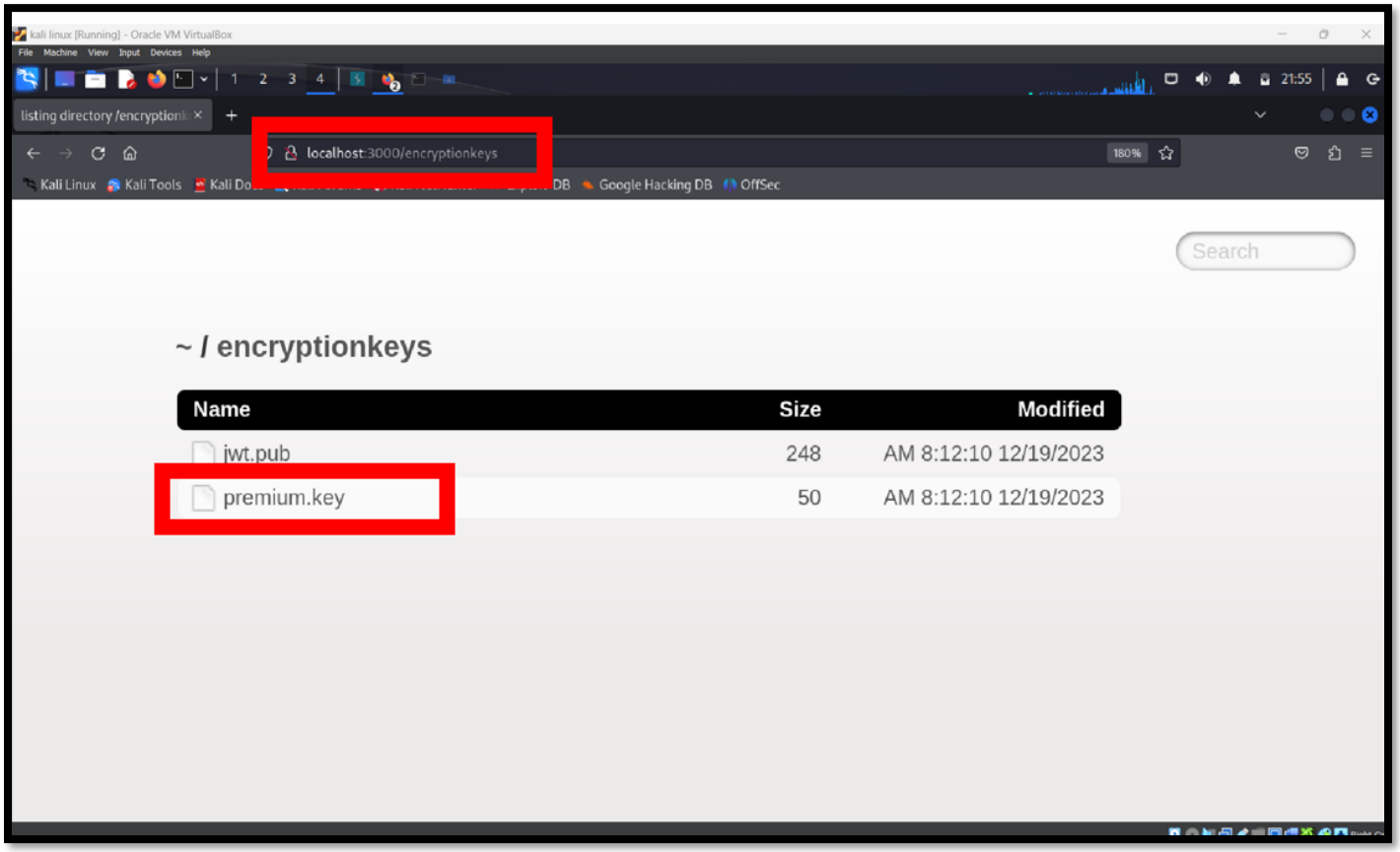




After that, we clicked on forced browser and directory under the attack.

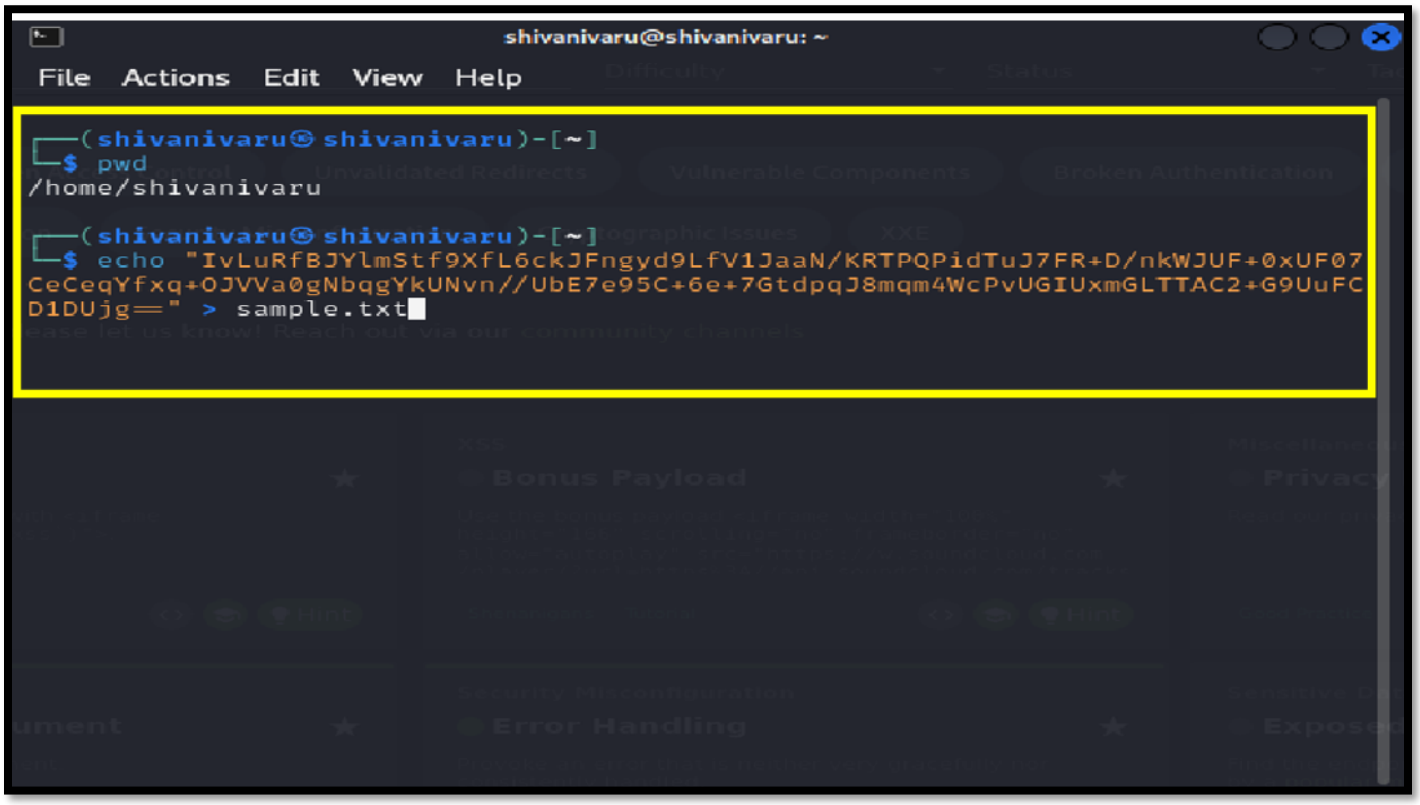


Then, we found http://localhost:3000/encryptionkeys as a browsable directory.

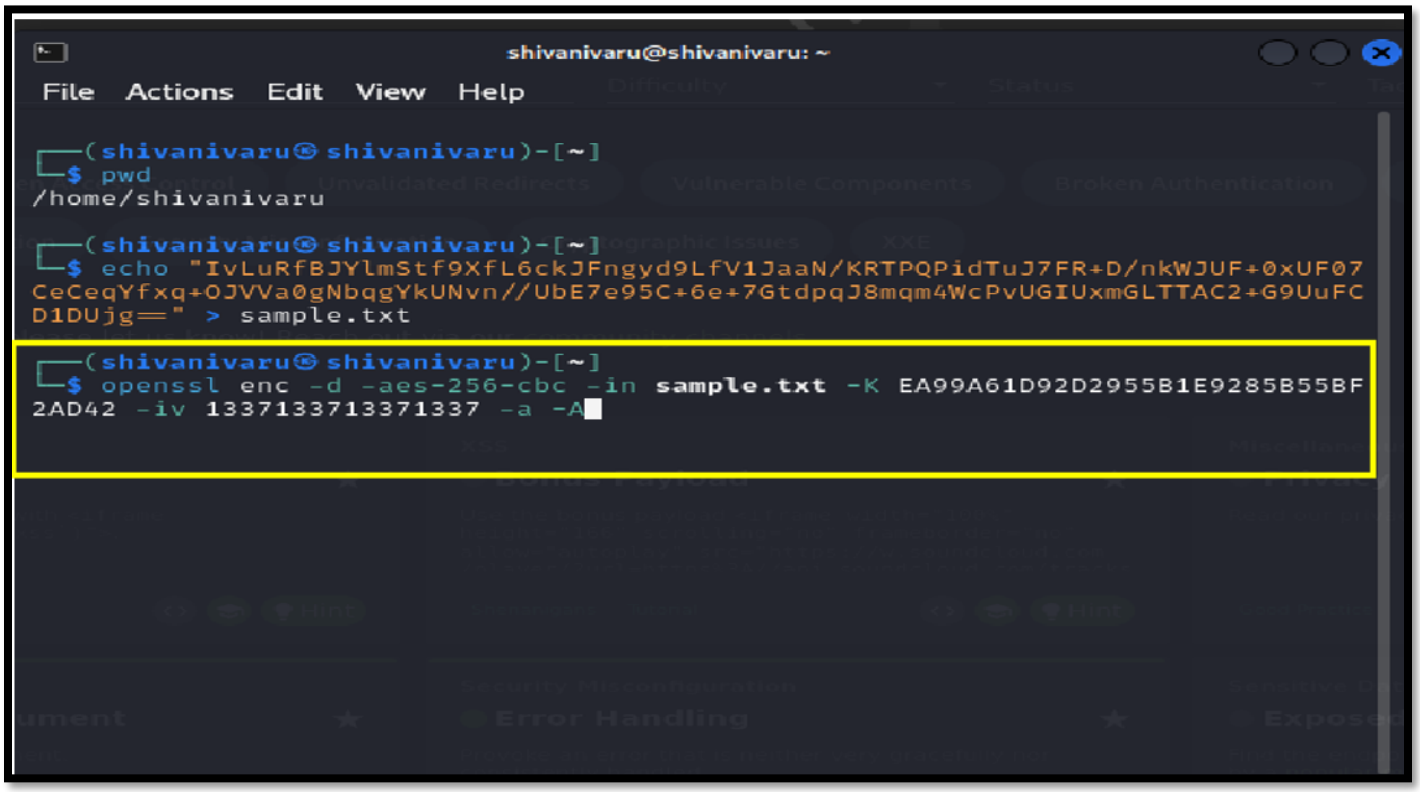


Next, we opened the browser for the encryption keys on localhost and downloaded the premium.key file.

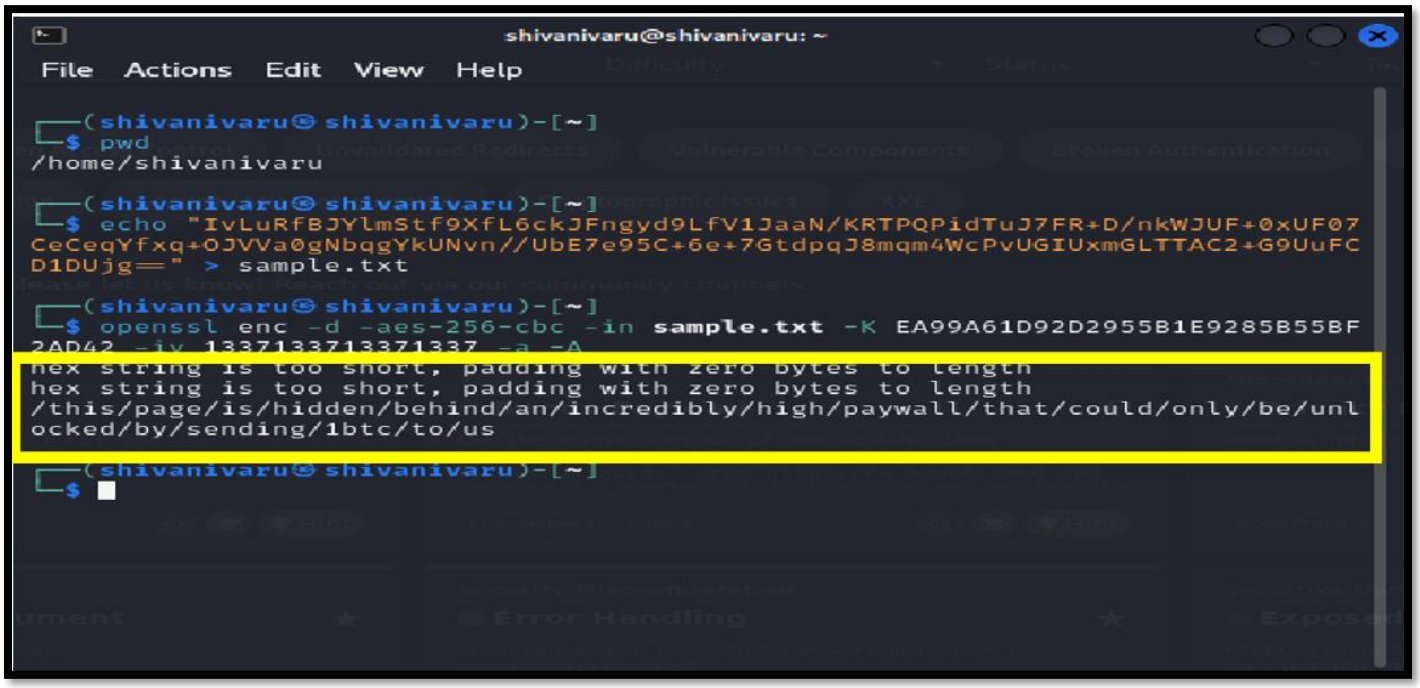
On the desktop, we created a .txt file named sample.txt. Here, first, we checked its current directory using the pwd command to check our current directory. After that, we created a file using the echo command in the terminal and copied the content from the premium.key to the sample.txt file.



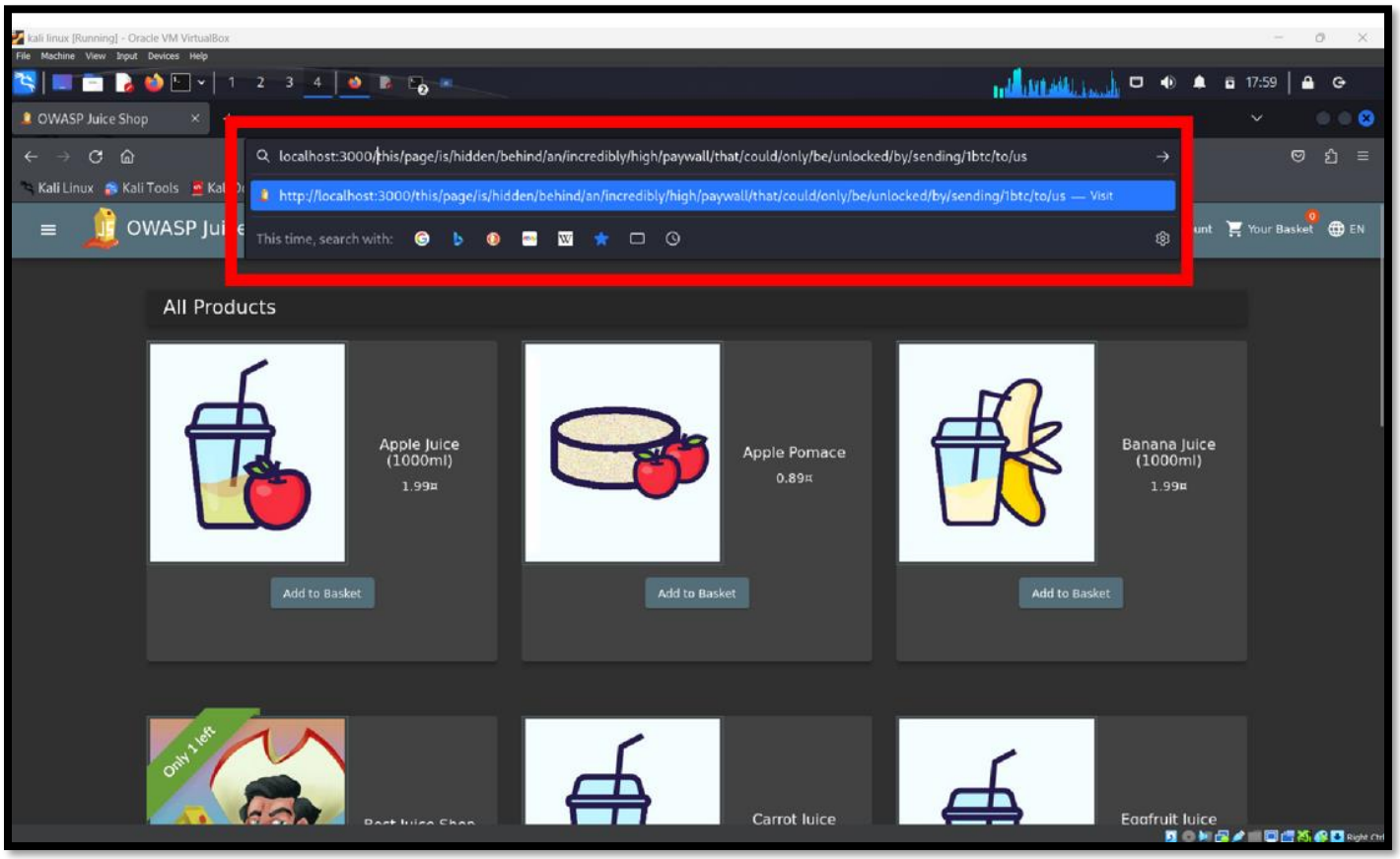
Then, we opened the terminal and used the command openssl with the necessary command to decipher the text as shown in the screenshot.



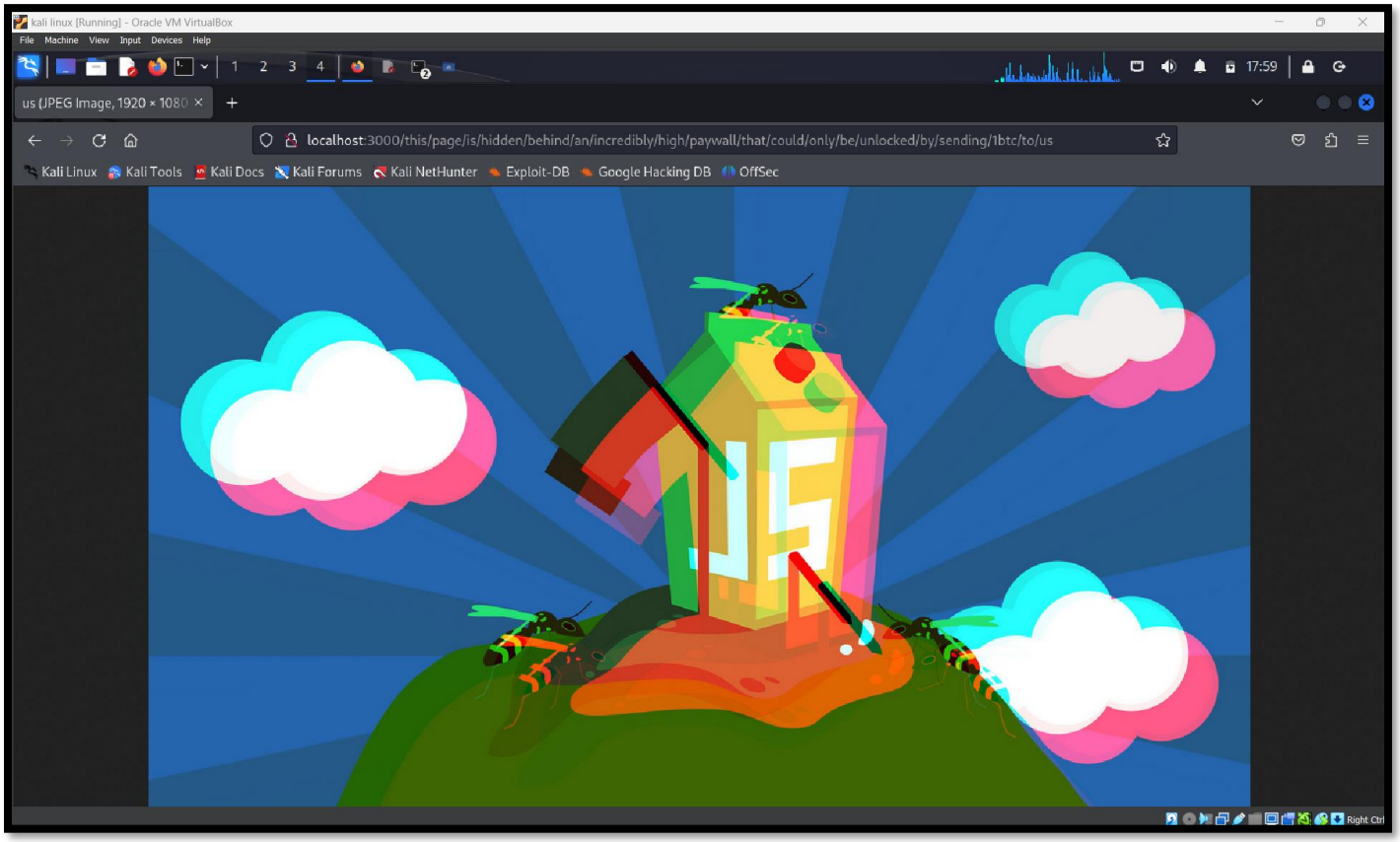
We got the decrypted path where we copied that path.



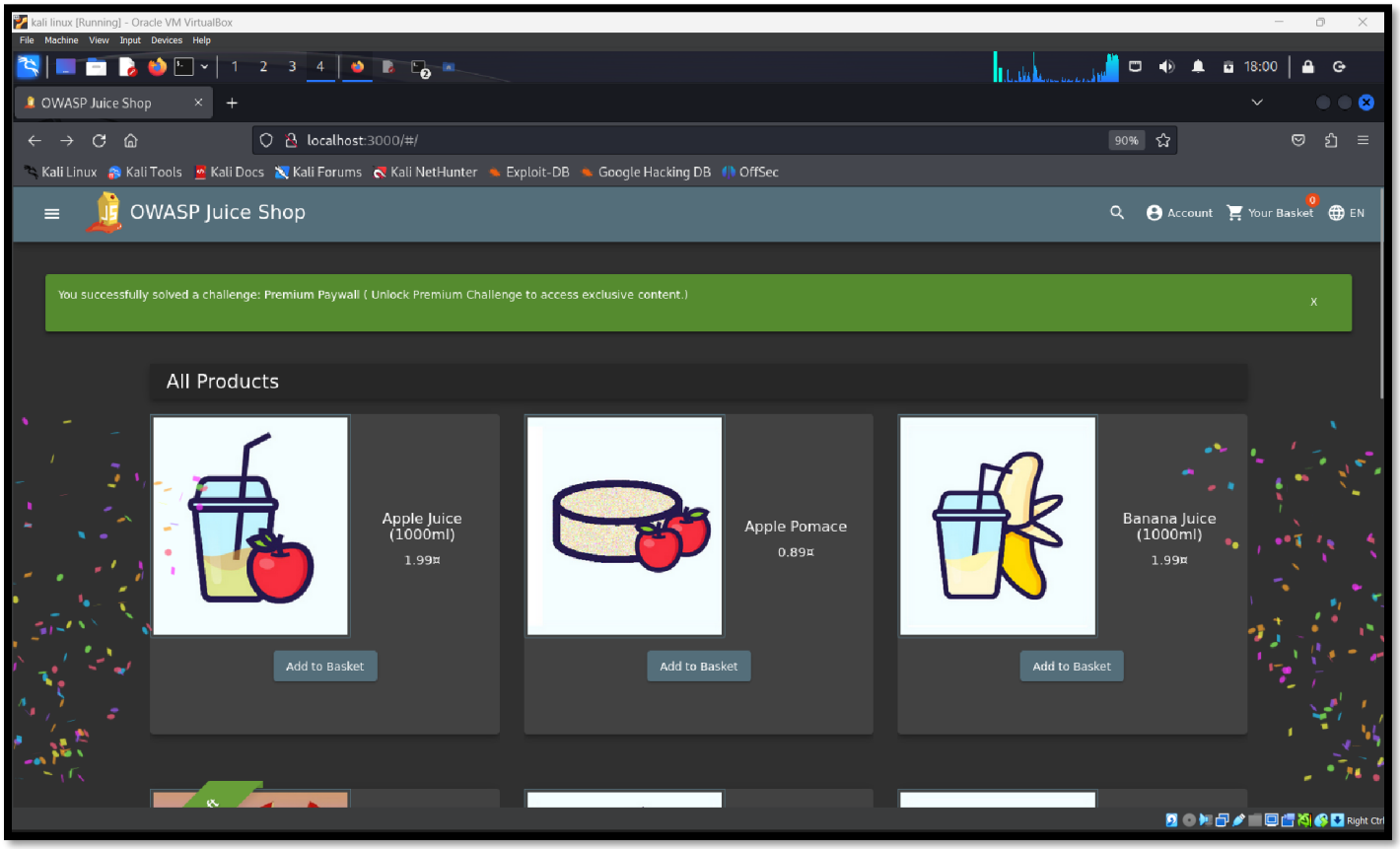
and then we put it on the browser with localhost:3000, and it opened the exclusive content.



We saw some "Marvel at the premium VR wallpaper."



Finally, we solved the challenge.



Vulnerabilities' Exploited by Task 4:

- Inadequate encryption key protection leads to unauthorized access.
- The comment section of an HTML page contains a publicly encrypted premium key, which exposes sensitive data and poses a significant threat.
- Malicious attackers exploit this vulnerability to decrypt sensitive data.
- Forced Browsing Attacks exploit vulnerable local host browsers, allowing open-SSL connections and premium access.key files, enabling decoding of encrypted text like AES encryption.

How Vulnerabilities Can Be Fixed or Addressed:

- Sensitive information should not be exposed, even if encrypted.
- Encryption keys should not be accessible to open SSL connections, as this would compromise security.
- Use industry-standard encryption techniques and secure storage solutions.
- Implement strong access controls, including RBAC and least privilege principles.
- Regularly rotate encryption keys to reduce breach risks.
- Use Multi-Factor Authentication for accessing encryption keys and sensitive cryptographic activities.

Real-World Scenarios:

- Encryption key vulnerabilities lead to unauthorized access to sensitive data.
- Attackers can exploit compromised encryption keys to gain confidential information.
- Consequences include data breaches, financial losses, and damage to reputation.
- Compromised keys enable decryption of encrypted messages/files, compromising privacy.

References

1. URL Decoder/Encoder. (n.d.). https://meyerweb.com/eric/tools/dencoder/

2. https://conestoga.desire2learn.com/d2l/le/content/1002737/viewContent/21555039/View

3. A01 Broken Access Control - OWASP Top 10:2021. (n.d.).

   https://owasp.org/Top10/A01_2021-Broken_Access_Control/

4. OWASP Top Ten 2017 | A3:2017-Sensitive Data Exposure | OWASP Foundation.

   (n.d.). https://owasp.org/www-project-top-ten/2017/A3_2017-

   Sensitive_Data_Exposure

Video Recording Link: