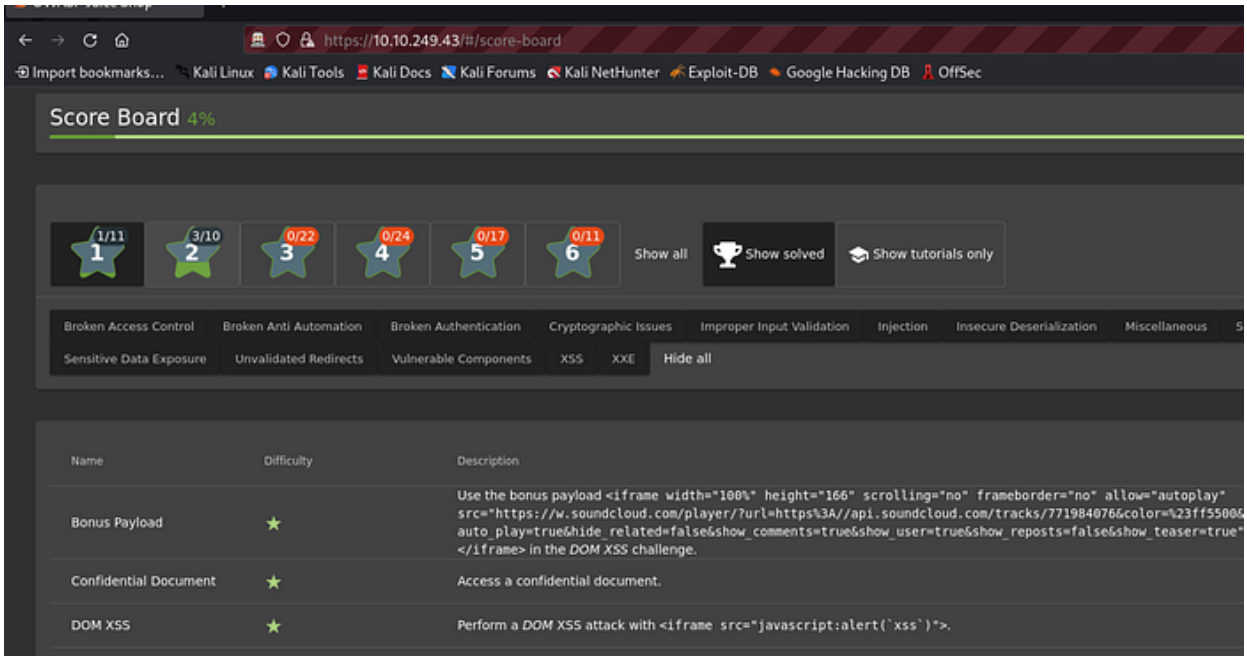
## #0 Access the /#score-board/page

Last task of the room is very simple. We will browse the hidden score board page. Let's type it on the URL.

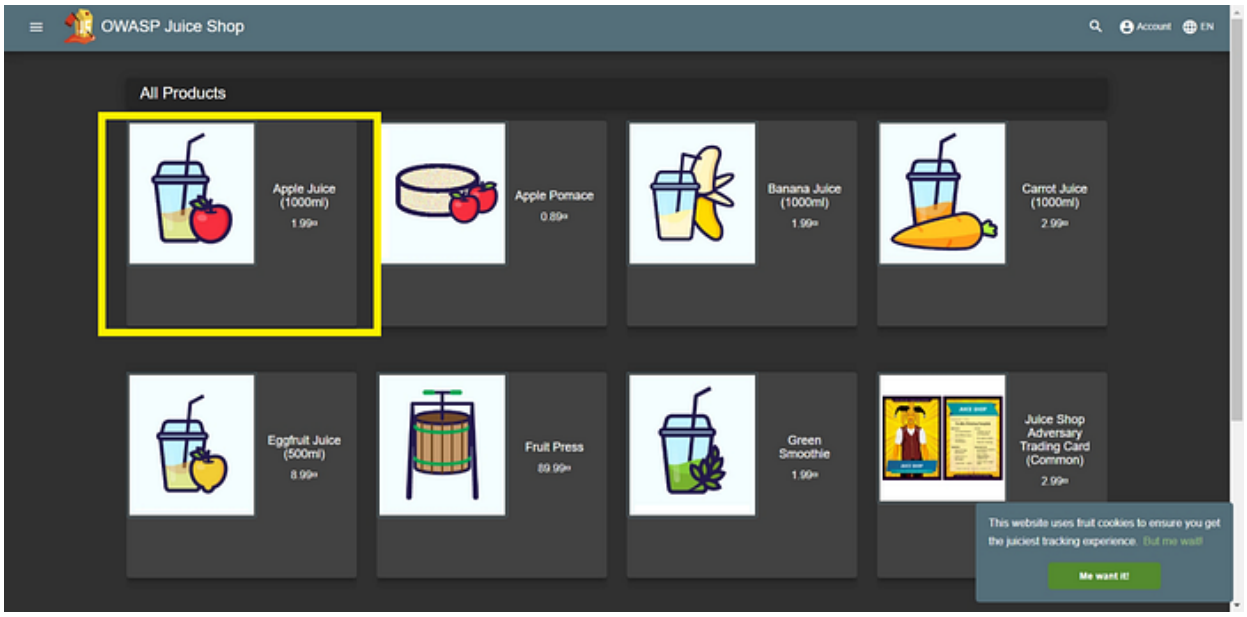



The score board certainly provide hints on discovering every weakness, techniques and even some entertaining bonus. There's a good reference for doing all of the tasks given by this web app. We could visit the website of Pwning OWASP Juice Shop.

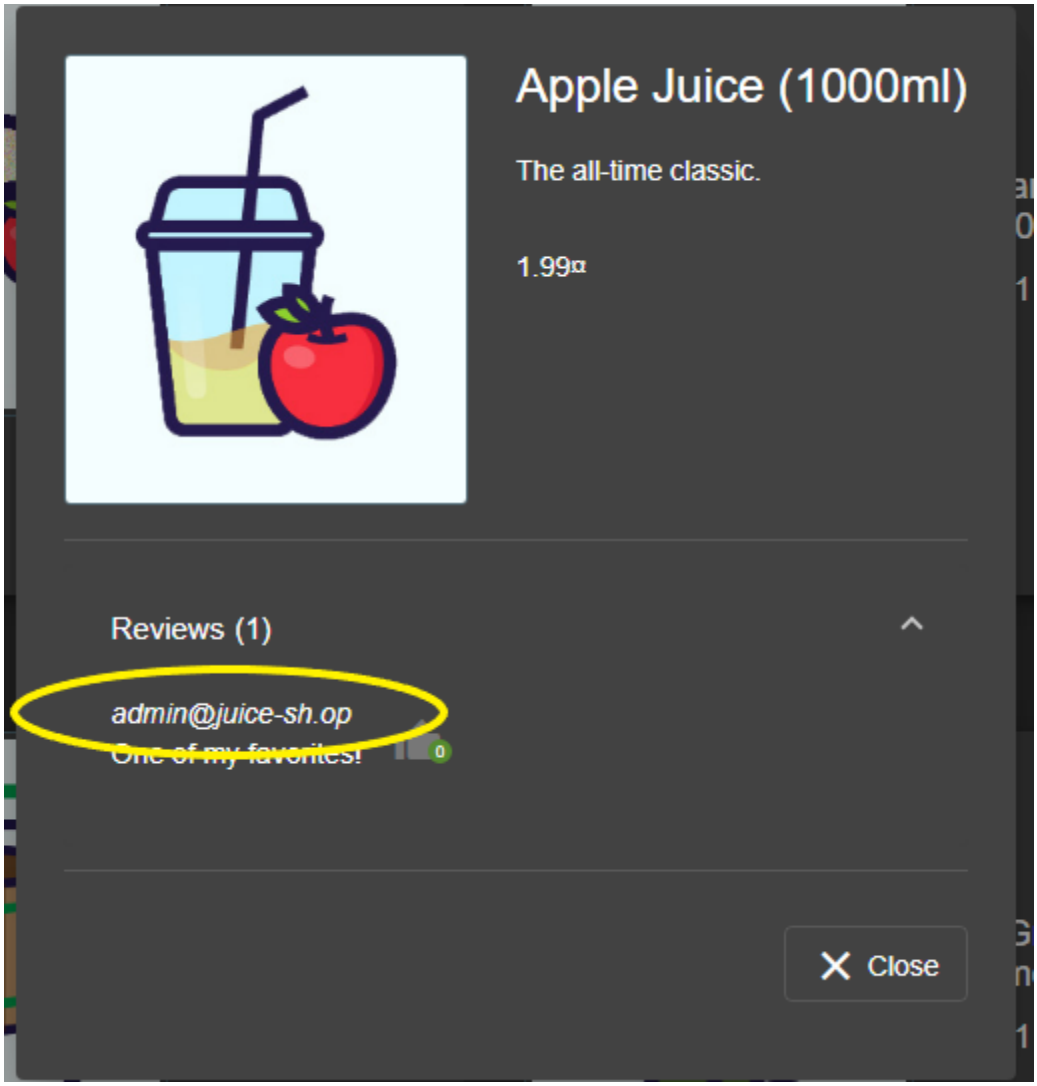
## #1 What's the administrator's email address ?

This one is a simple one because we only need to look all of the products reviewed on the home page. It is recommended to view each of the products on the page, there might be other useful information we could find.

The very first product on the page is the “Apple Juice”.



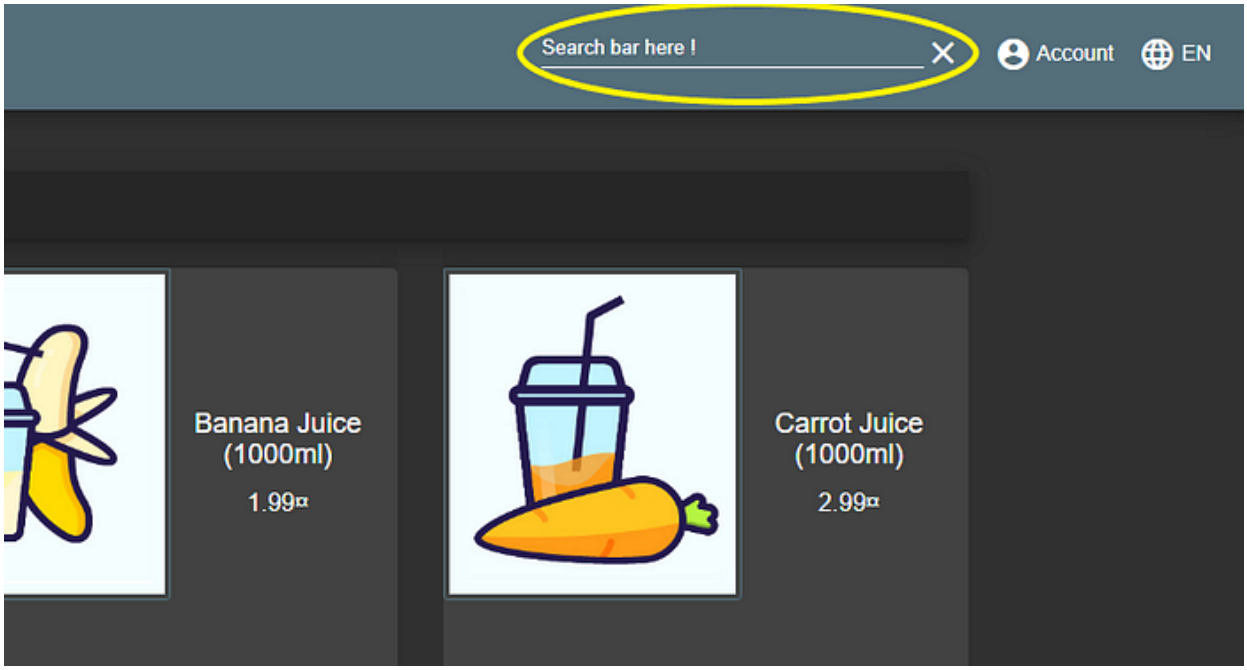

Viewing inside the detail of the product, there is a “Reviews” section. Click the dropdown button and the reviews should be listed.

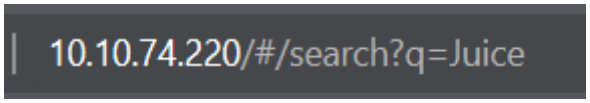



**Answer : admin@juice-sh.op**

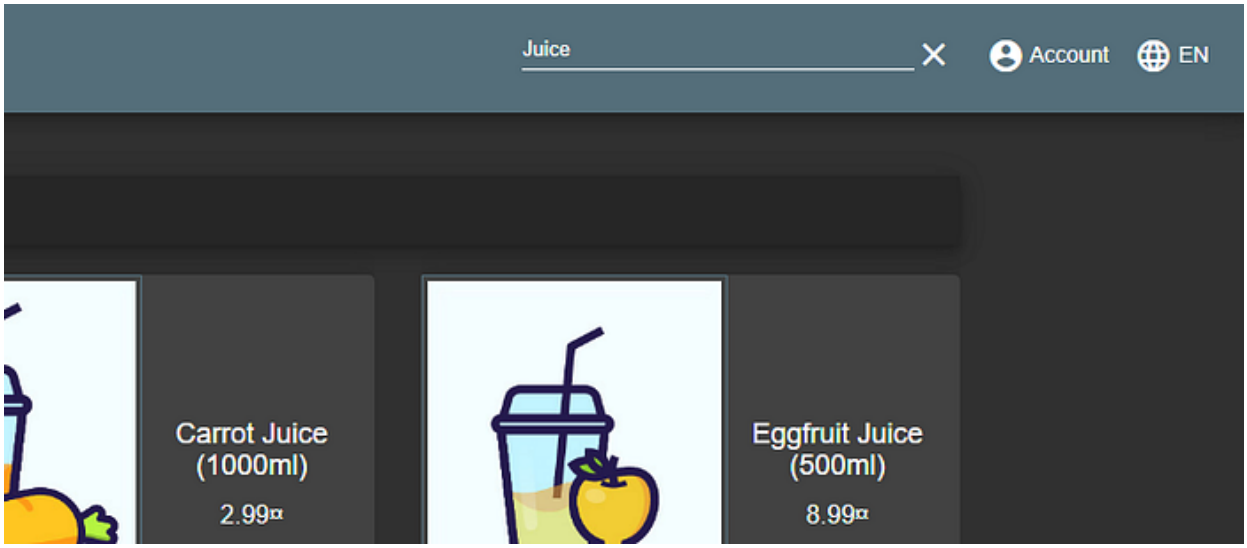
### #2 What parameter is used for searching ?

Searching should involve inputting something into a search bar and in this case is no different. The search bar is located on the top right, just right beside the “Account” dropdown button.




Click it and begin inputting any product names. On our URL, there will be updates on the parameter of the request URL immediately after we enter our inputted strings into it.




Parameters on the URL is identified by a question mark (?) located after the URL path and an equal sign (=) after them.

**Answer : q**

### #3 What show does Jim reference in his review ?

If we look up a product with the name of “Green Smoothie”, a reviewer named Jim mentioned the word “replicator”. Just like in the platform’s description guide, we could google the word. The result is a fictional device, capable of synthesizing many things like food or tools. Further information shown that this fictional device appear on a TV show namely Star Trek — The Next Generation.


https://makeagif.com/i/UabBmB

What is funny about the product is that it is made partly with grass. Perhaps in the future, sentient beings eventually consider grass edible😏(?).

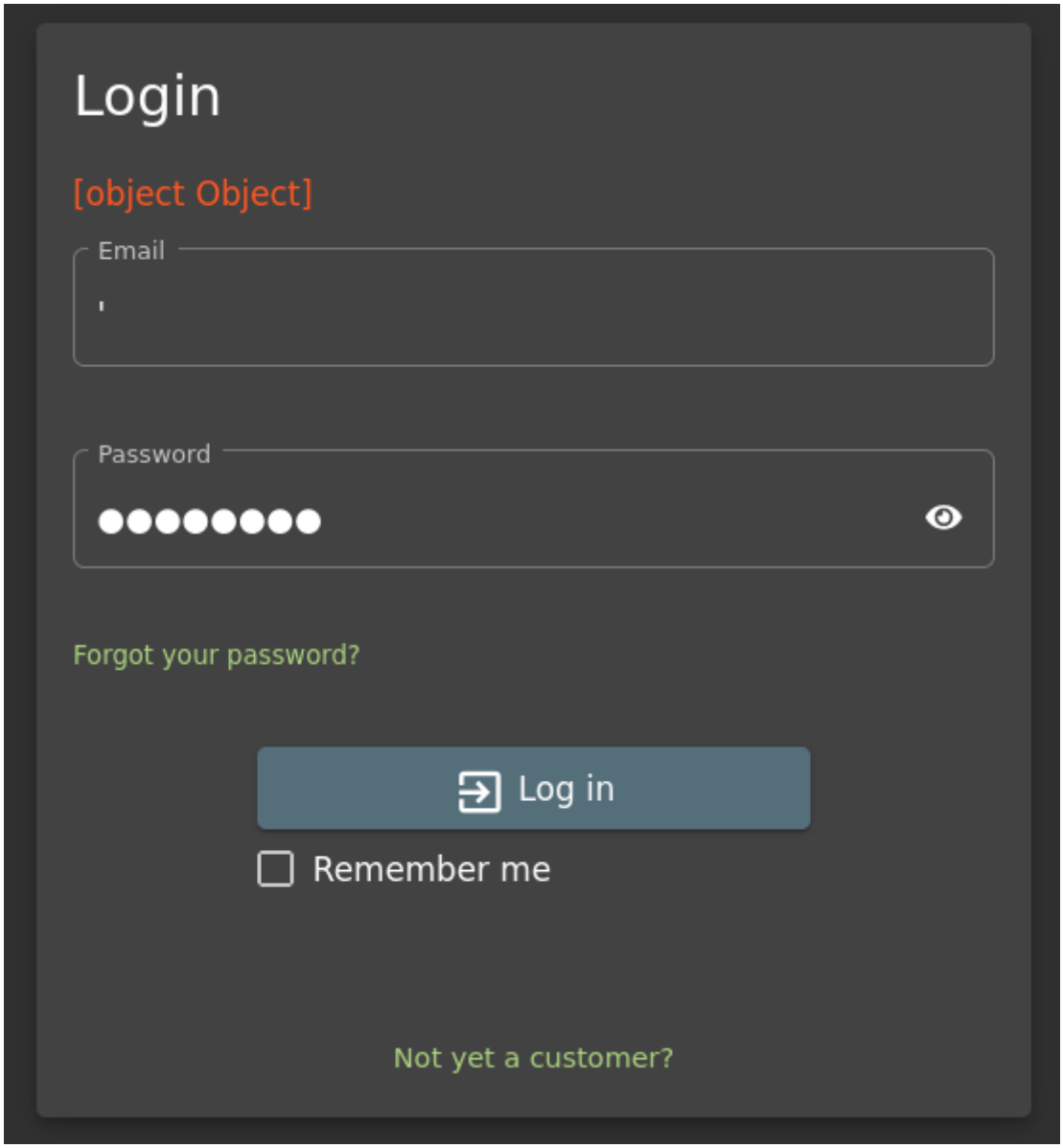
**Answer : Star Trek**

### #4 Log into the Administrator account !

Let's move to the login page by clicking the "Account" dropdown button. On the page, there will be input bar for email and password. Before checking for any SQLi vulnerabilities, we could check first if any of the input bar has some kind of strange behaviour when inputted with single (') or double quotes ("). If the results received

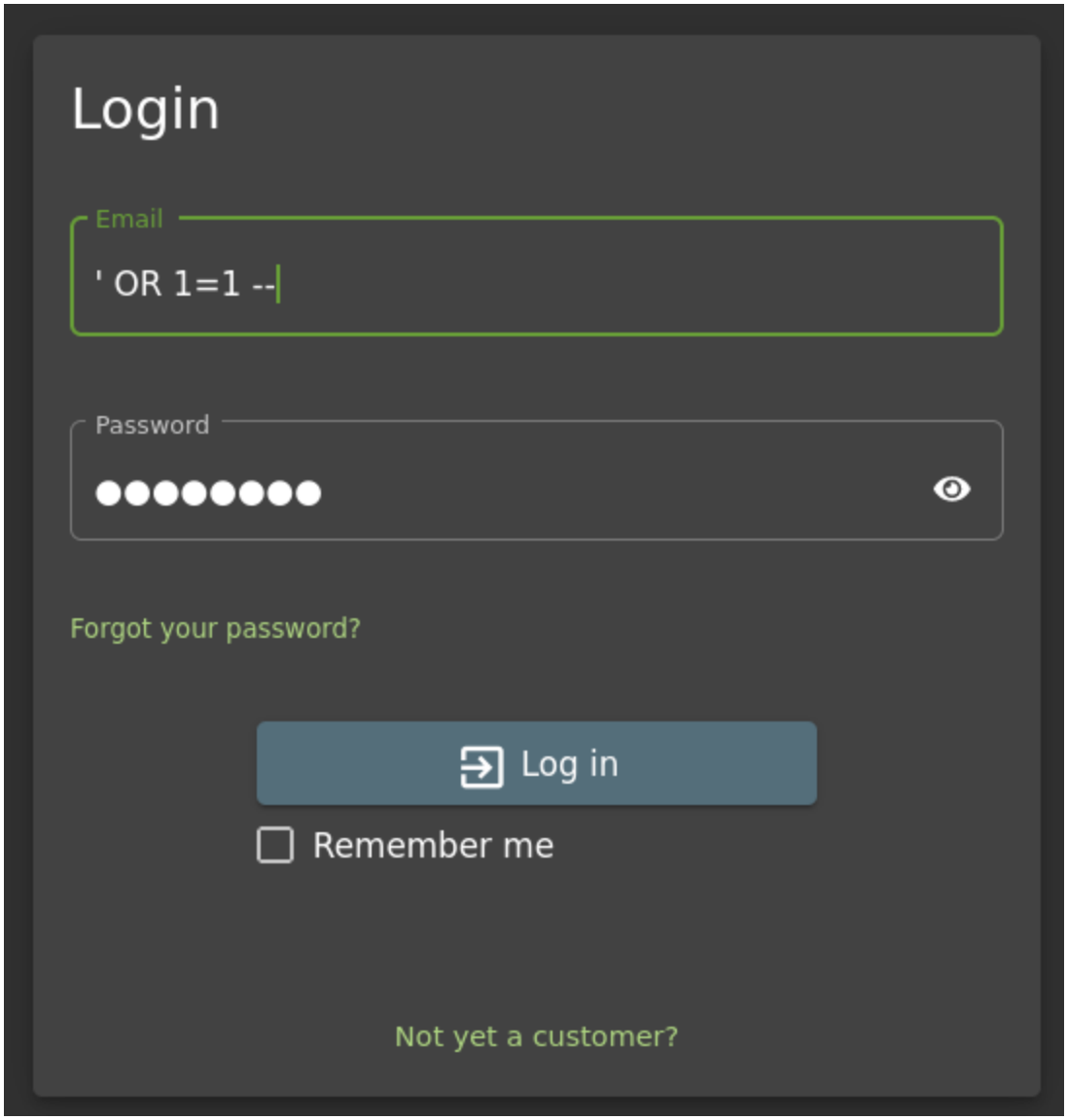
expected strange behaviour, then there's a probability that SQLi vulnerabilities exists.

In this test, the expected strange behaviour exists. We could confirm the input bar could be vulnerable to SQLi.

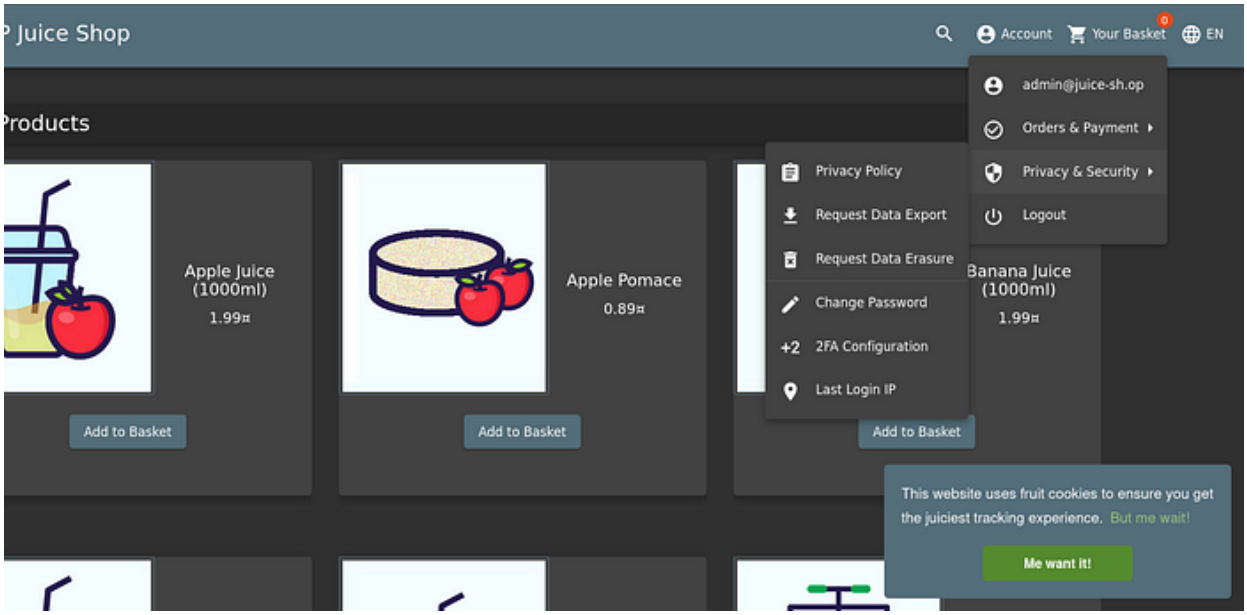



Returned a strange error: [Object Object]

Next, let's try inputting the query given on the platform manually and click the log in button.




We successfully logged in with an administrator account.

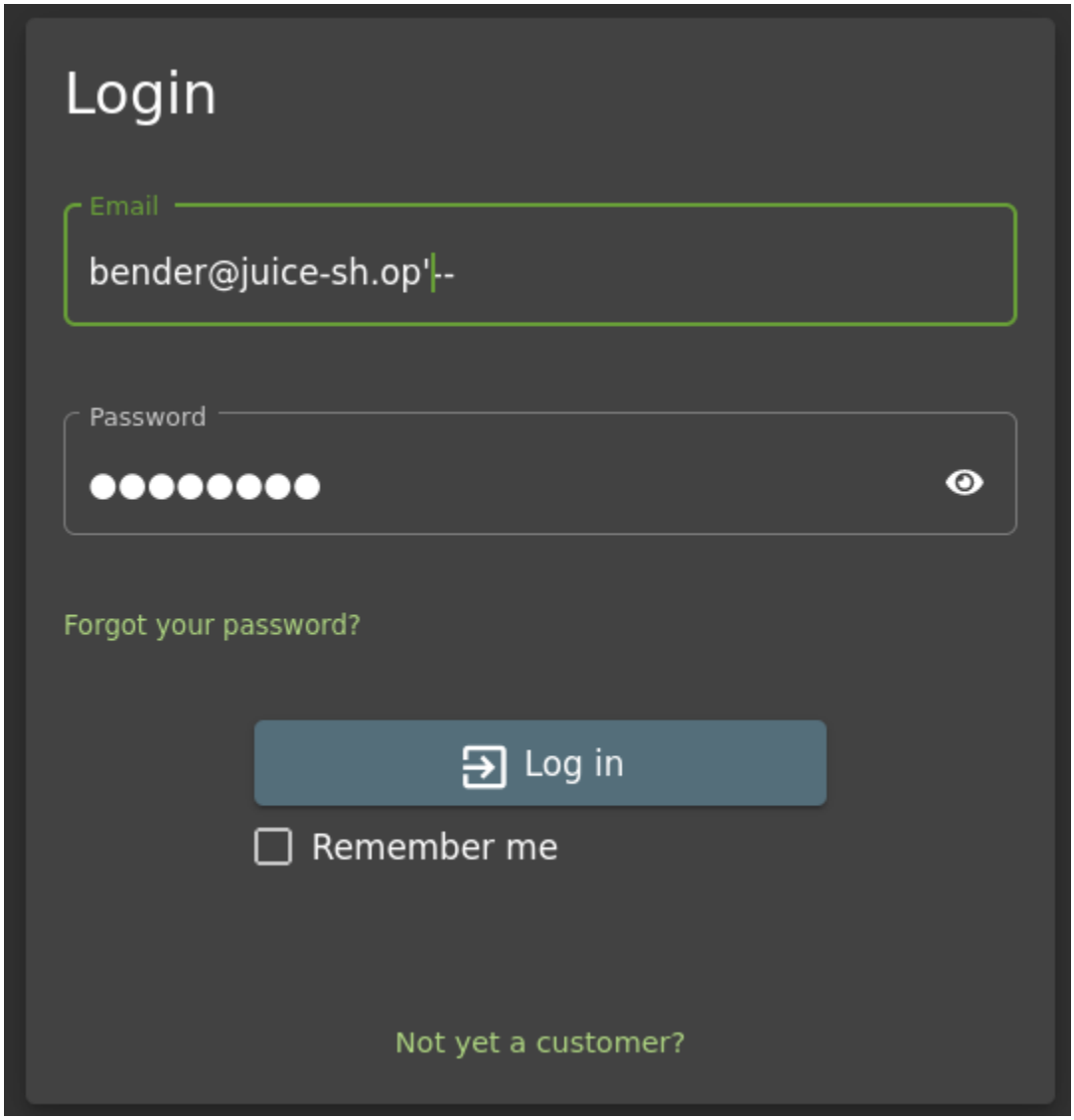



The very reason why this injection works explained correctly on the platform. The statement produce a True value and any restrictions will be commented. To be exact, the entire SQL query should be like this :

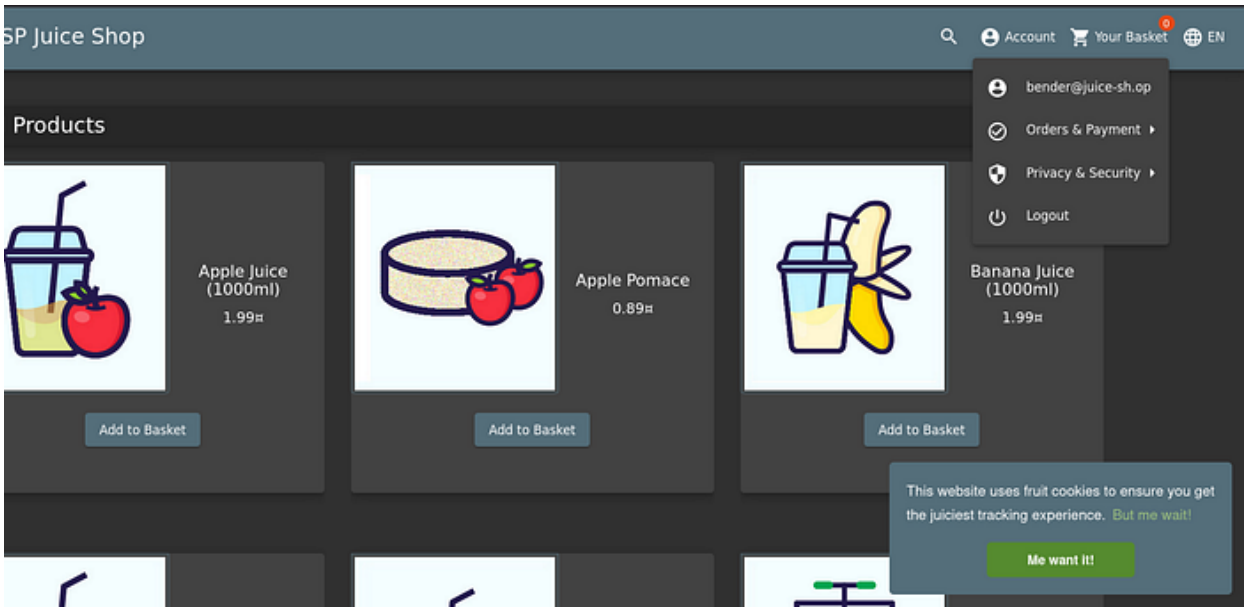
```
SELECT user_id FROM users_table WHERE email_address = '' OR 1=1--' AND password = 'thepassword';
```

Notice the colors of the query. The query starting from ‘AND’ and beyond changed into green, marking the rest of the query is actually a comment. As of the result, the query only selecting an ‘email_address’ which has the value of True in the database’s table.

**#5 Log into the Bender account !**




The same technique applied when logging in using another email address.




On the platform, it is also explained very well why we don't need to use the '1=1' statement as the email address is True in a value sense. The final SQL query that we expect should be like this :

```
SELECT user_id FROM users_table WHERE email_address =
'bender@juice-sh.op'--' AND password = 'thepassword';
```

We could notice again that the last part of the query is commented, thus the query only asks for an email address.

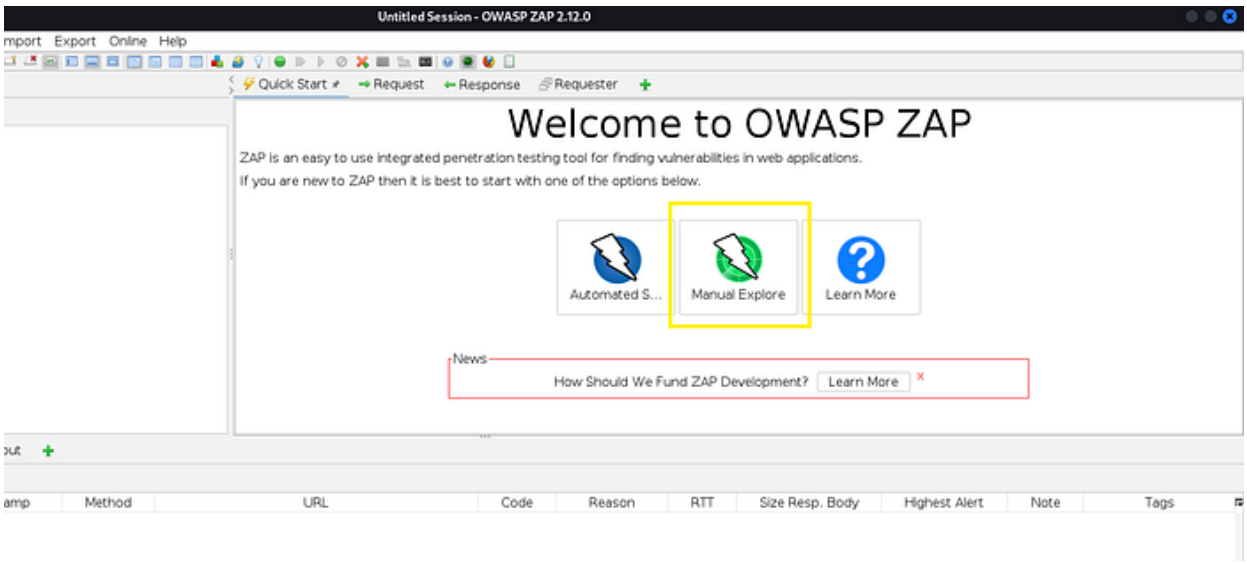
### #6 Bruteforce the Administrator account's password

We may have the access of high privilege account, but we still don't know the password yet. In order to get the password, we could try the very basic way of bruteforcing through the login form. On the platform, the tool that is used is **Burpsuite**. However, using this tool with Community Edition license will only hold back the attack performance. Hence, I've discovered another alternative that is good enough and has GUI just like **Burpsuite**. The tool is **OWASP — Zap**, a web app scanner that is free and open-source.


https://www.zaproxy.org/

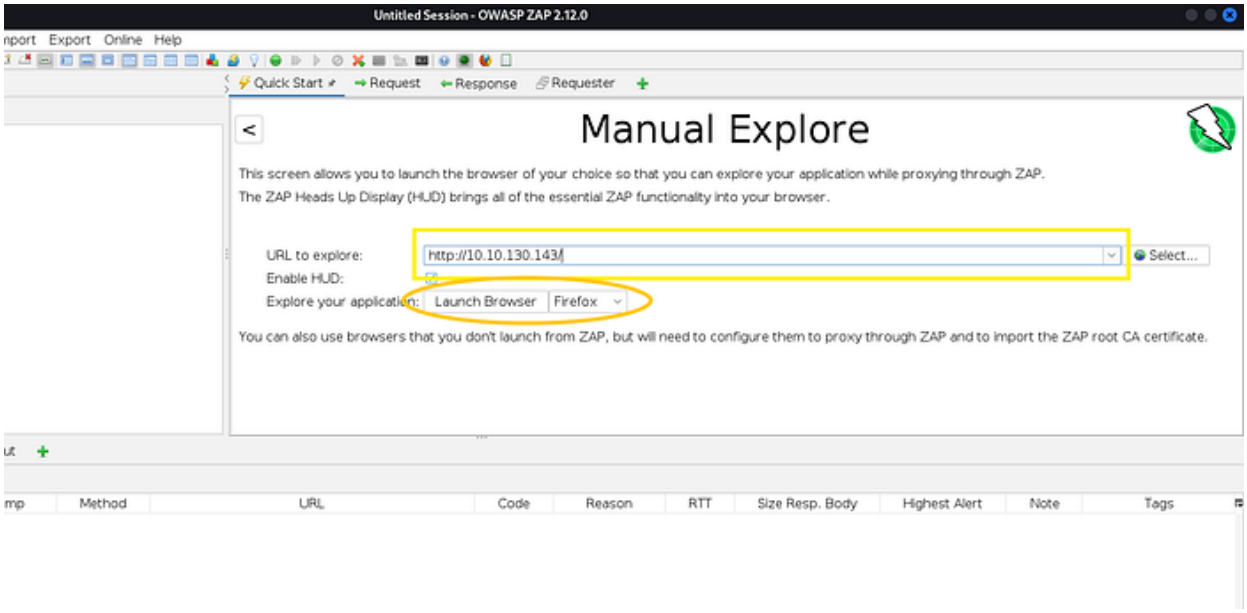
Here are the steps of the usage :

1. The very first start of the tool in the quick start tab, click “Manual Explore”



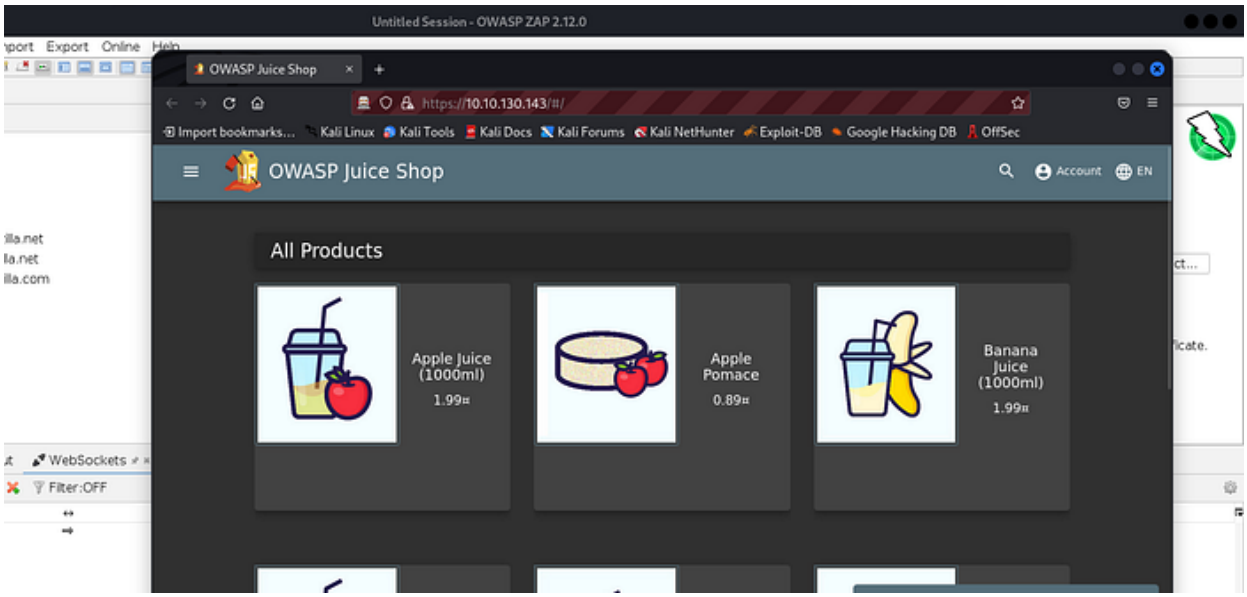

Quick Start Tab

2. Inside of the manual explore option, we will provide the “URL to explore” and select our browser for exploring. After that, click “Launch Browser”.



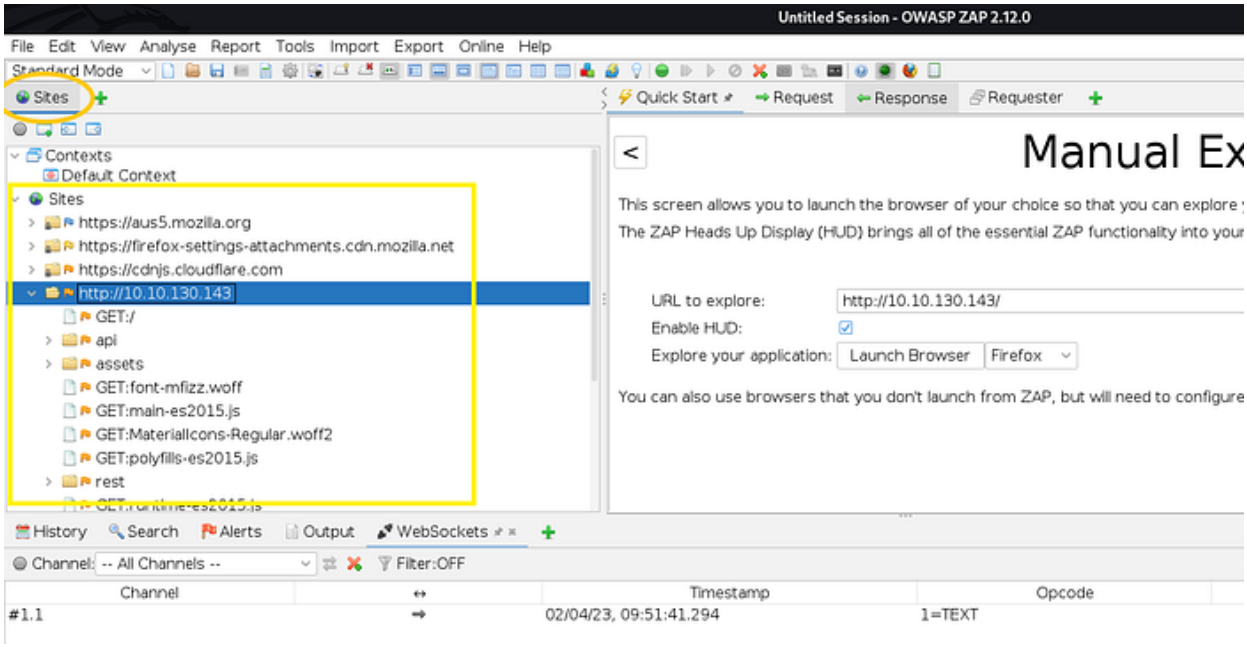

Manual Explore Option

3. A browser that is monitored by **Zap** will then be launched and we could explore the web app freely just like on a normal browser.

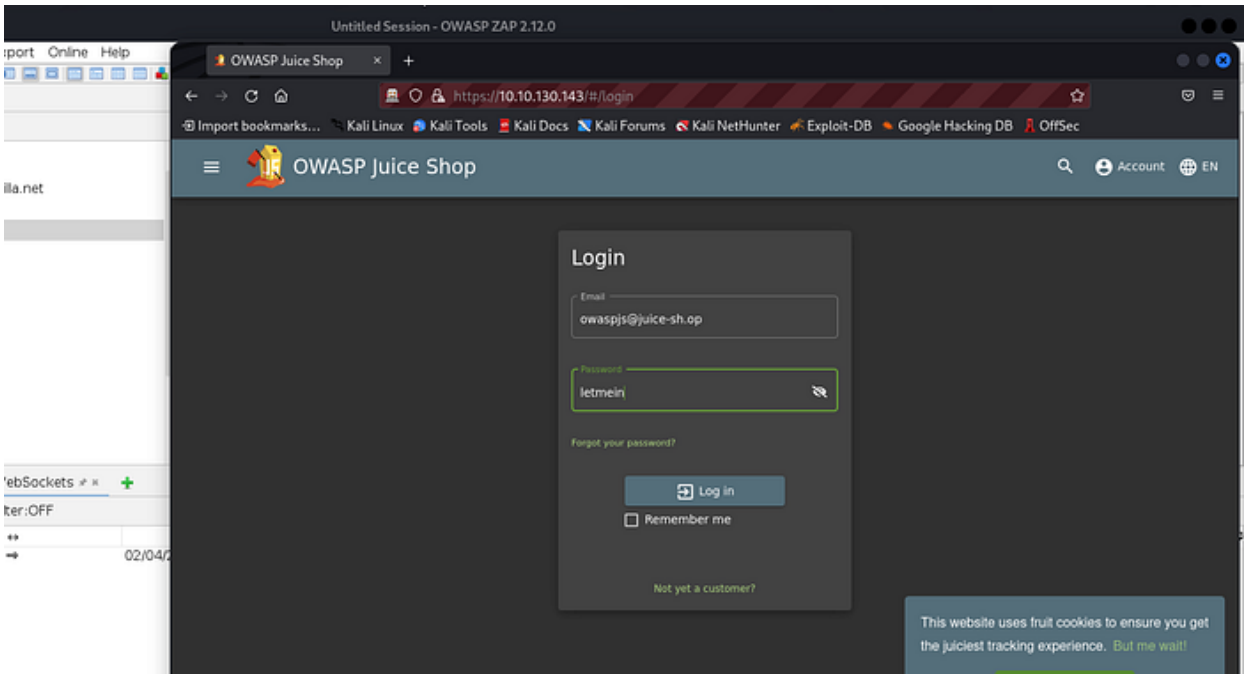



Monitored Browser

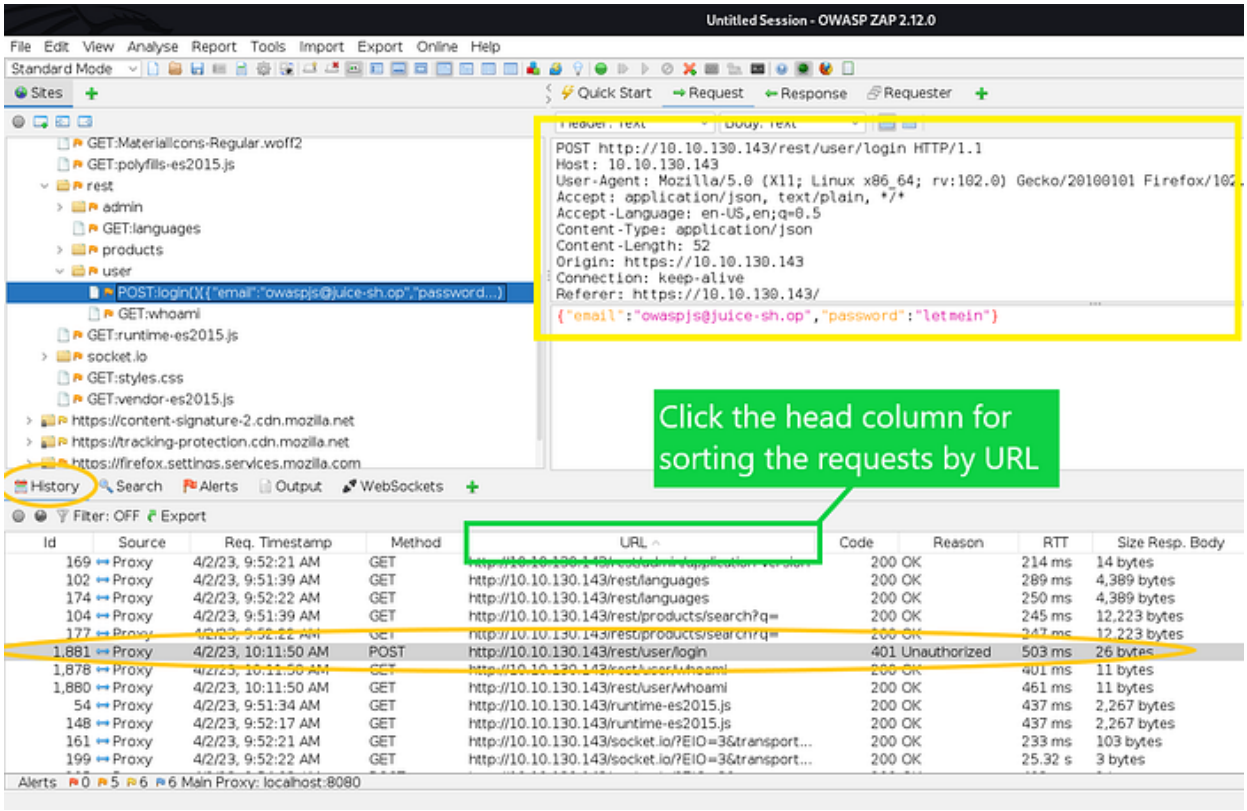
4. Before we head into the login form, we could see all requests that are recorded by **Zap** in the “Sites” tab.




5. Let’s head into the login form and try logging in with random credentials just for discovering what is the request URL (Note: this part of the step is similar when we try intercepting request in Burpsuite, but we won’t hold the request just like Burpsuite does. Instead, we will send the request normally). After that, we look for the request that was used when we were logging in with our random credentials.

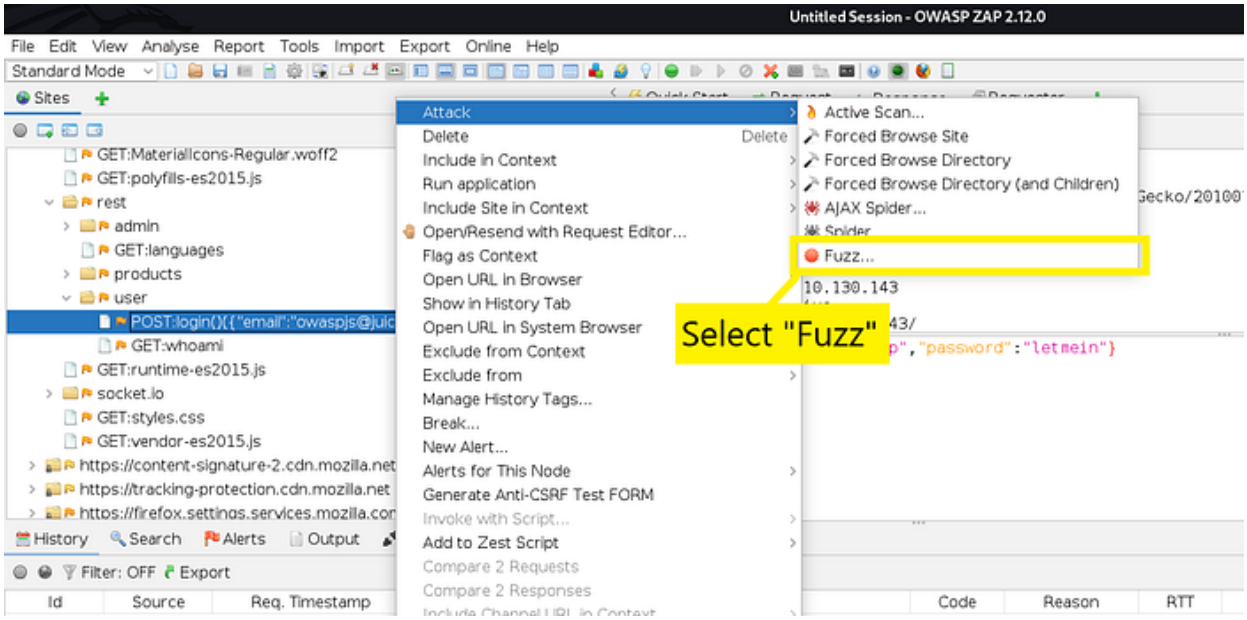



Random Credential Login



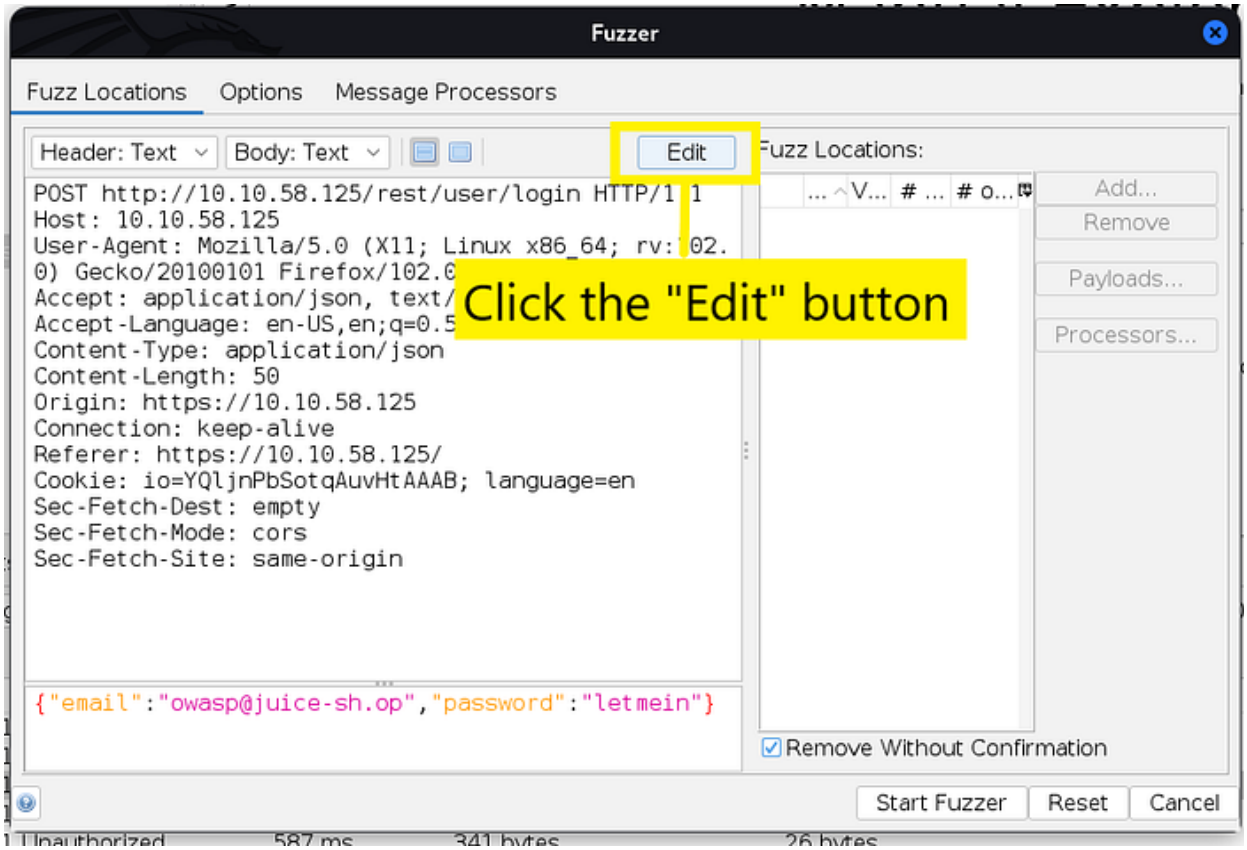

Request Discovered

6. We now know the request used for logging in. Next, we will launch our attack by ‘Fuzzing’ with the request. Right click the request inside the user’s request lists (Inside the “Sites” tab).

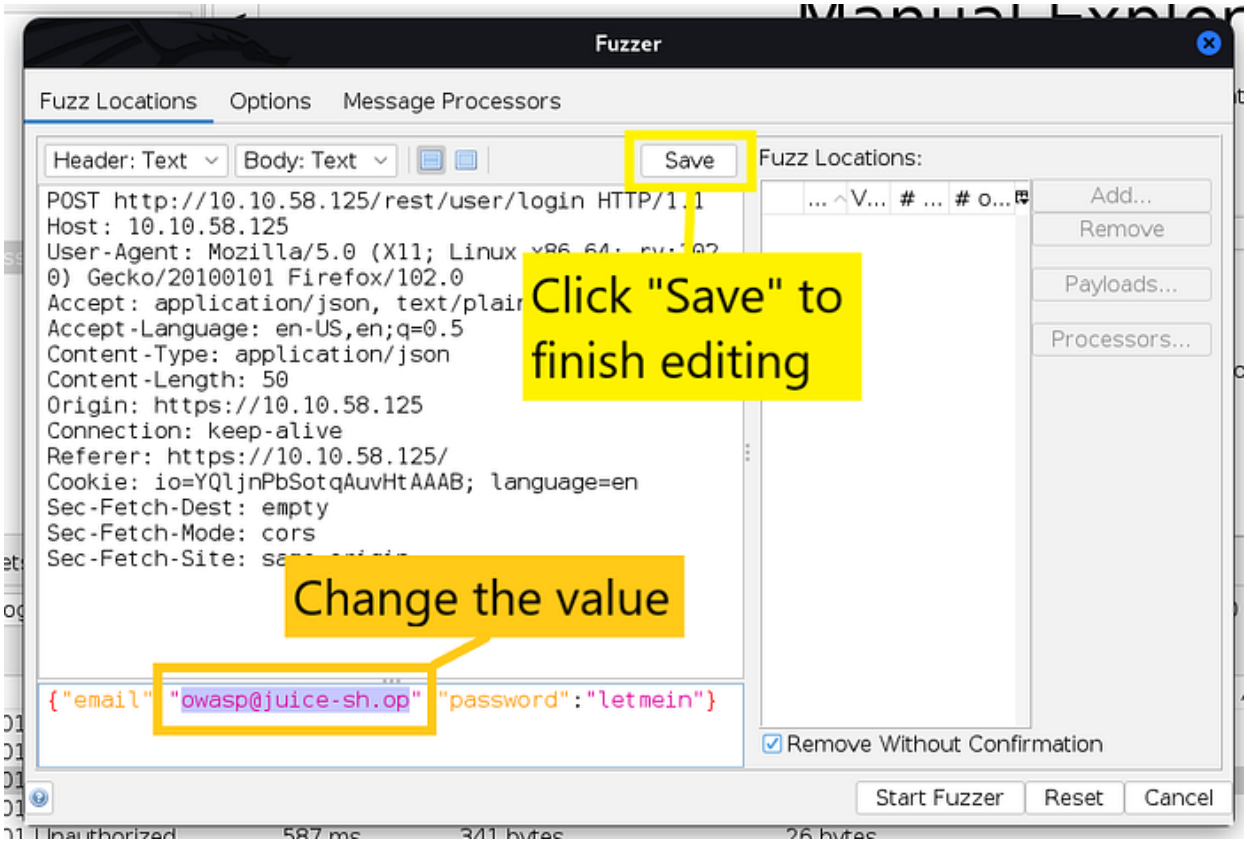


Selecting the request

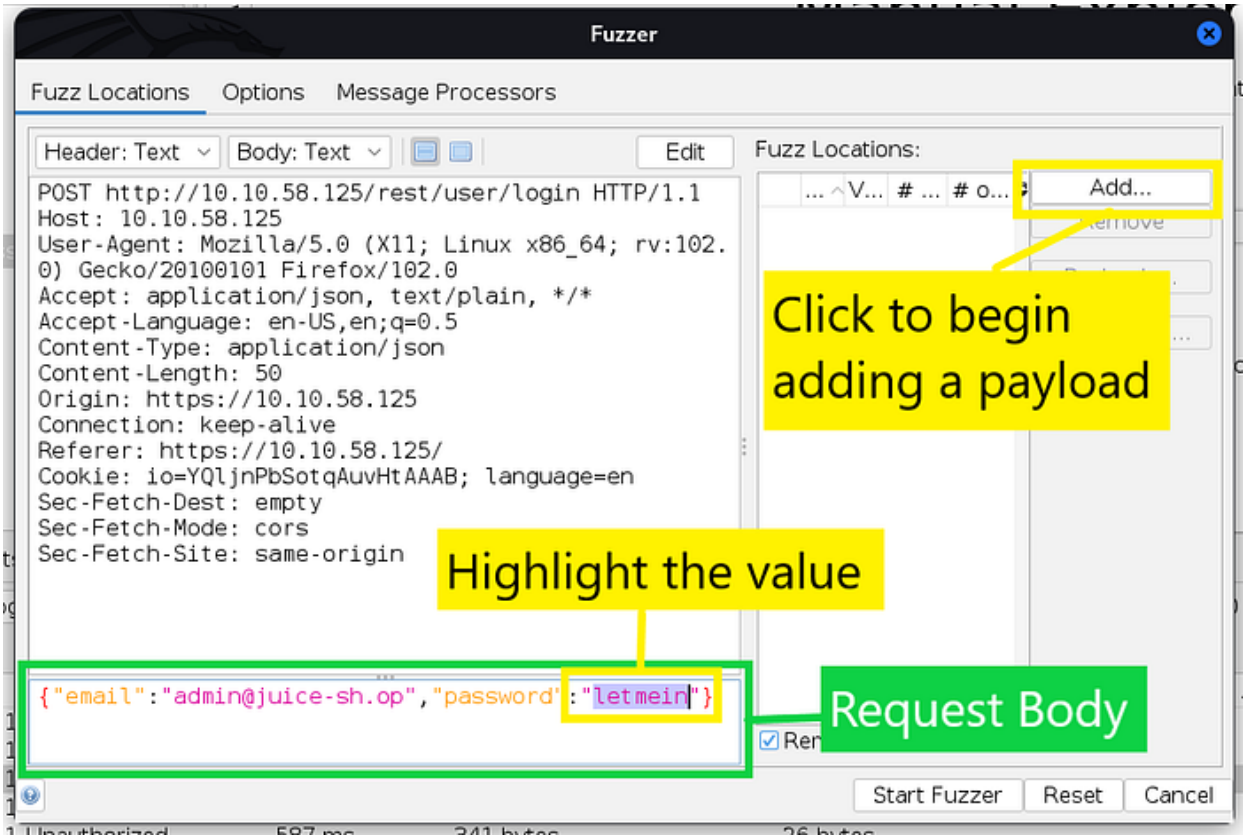
7. The Fuzzer window will pop up and we will select our the fuzz location. Our request is a POST method, so we have a JSON type body request located under the request headers. Click the “Edit” button to start editing our payload. Change the email parameter’s string value with the correct administrator email address. Click the “Save” button to finish editing. Next, highlight the string value of the ‘password’ parameter and click the “Add” button to add a wordlist for fuzzing the request.




Fuzzer Window

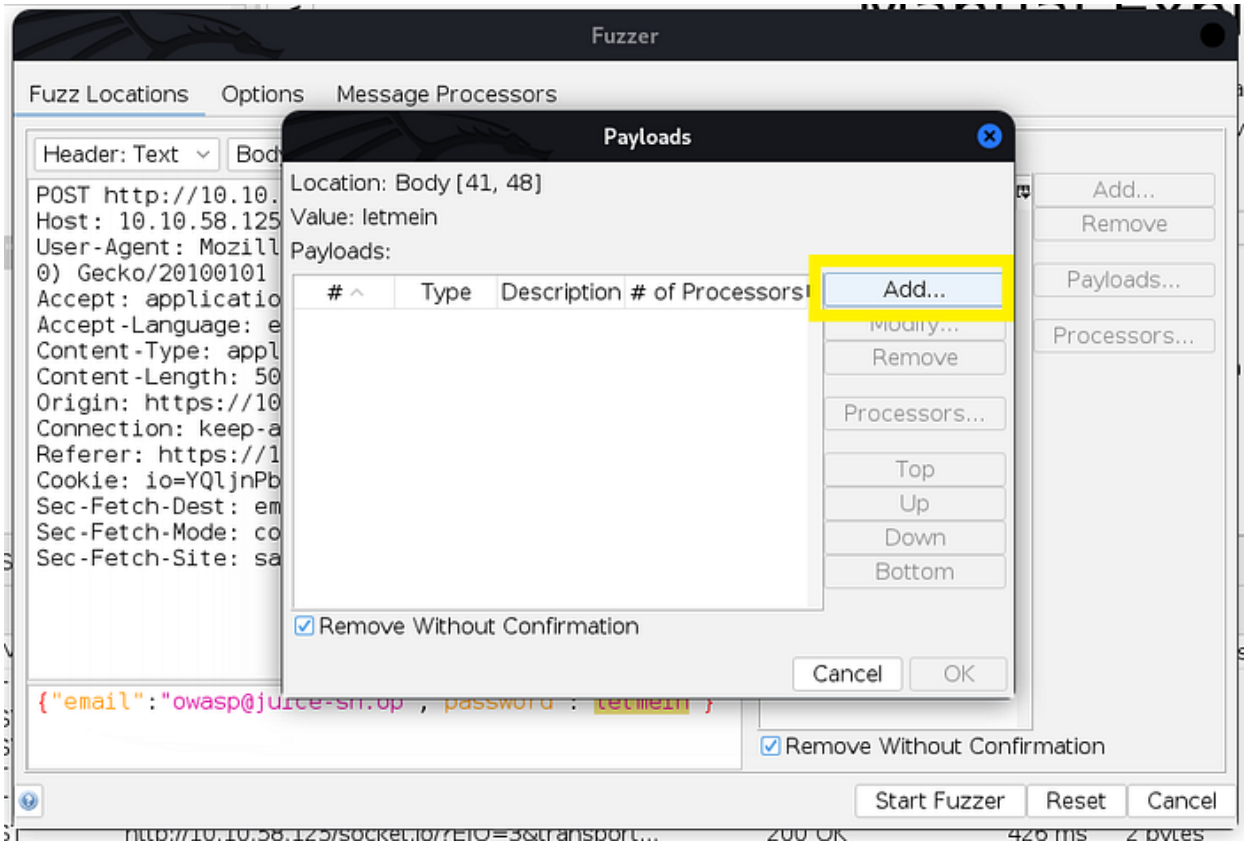



Editing value

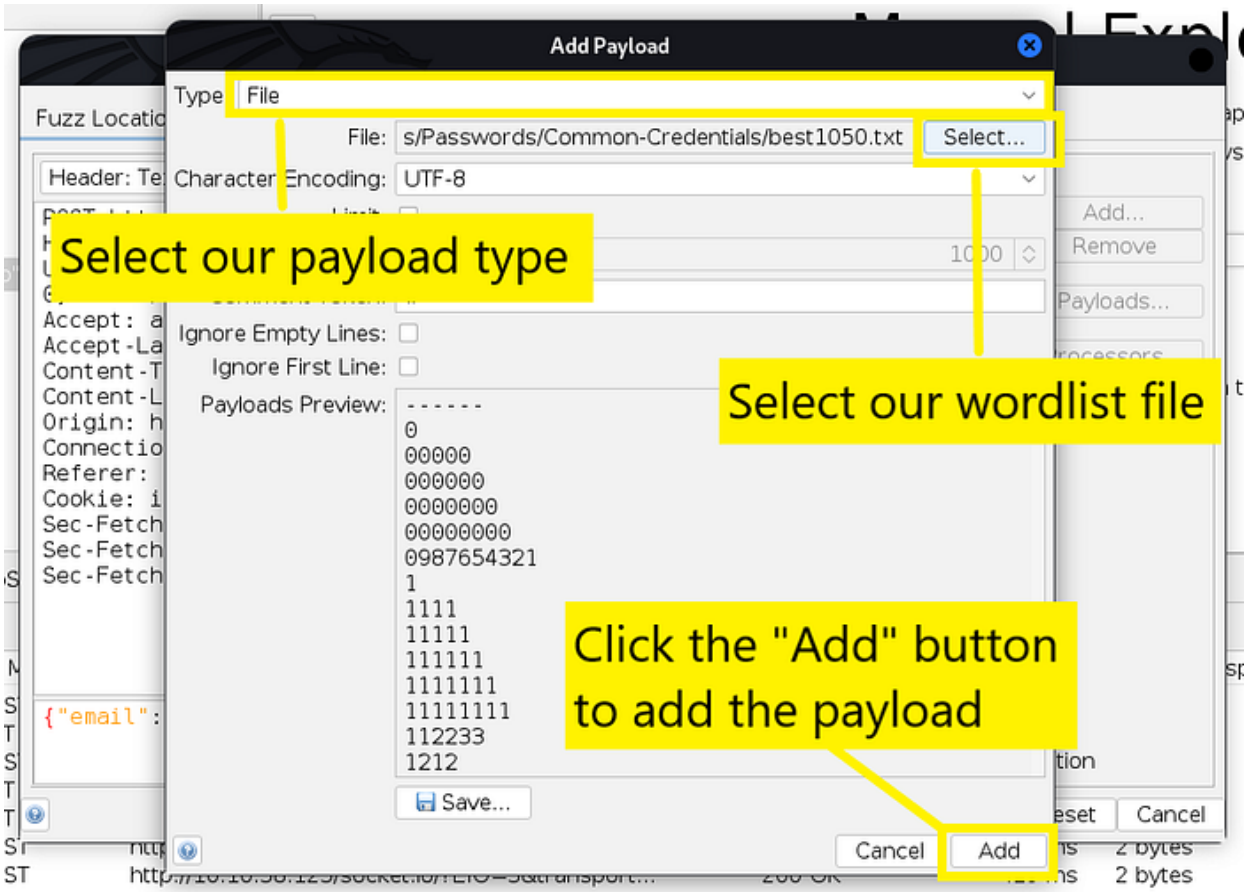


Begin adding a payload

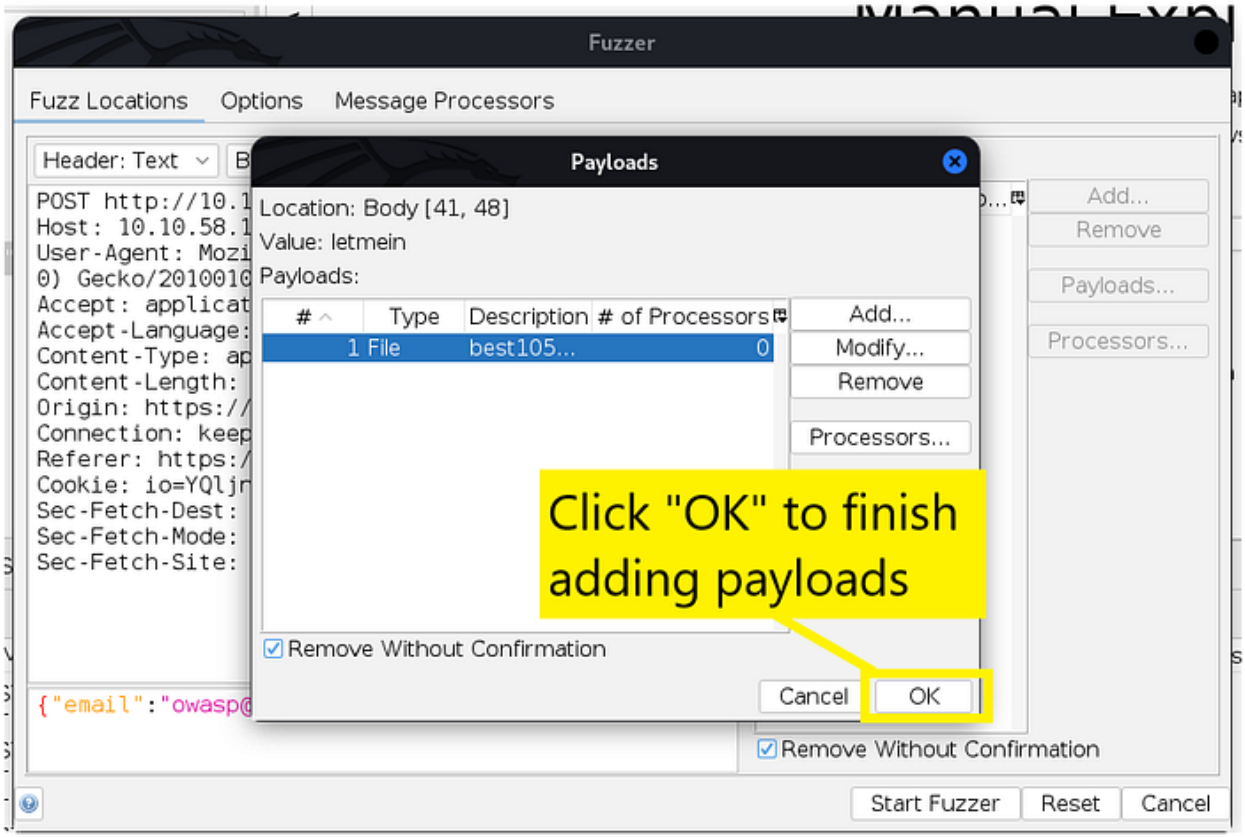
8. The Payload window will pop up and we click the “Add” button to select our wordlist file. We will use the wordlist that is recommended by the task. Click the dropdown button and select ‘File’ for the payload type. Next, click the “Select” button to choose the file.




Payloads Window

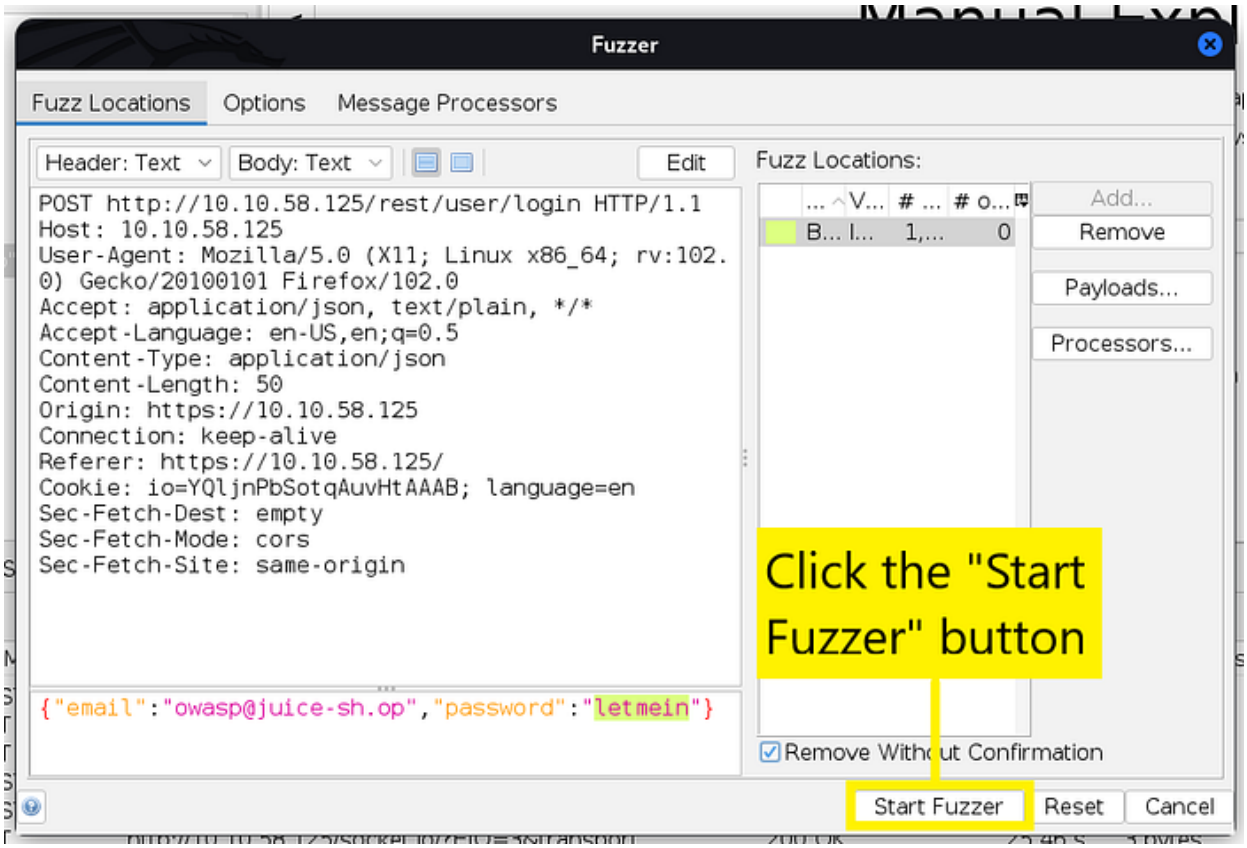



Adding payloads



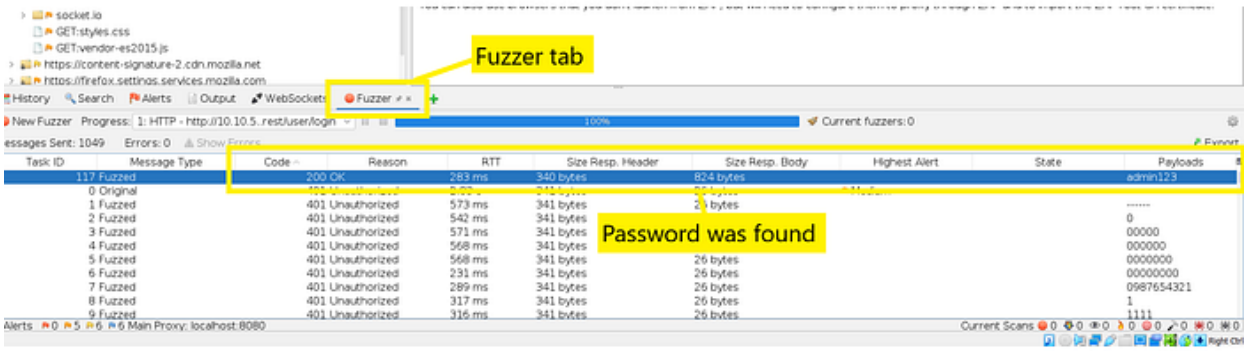

Finish adding payloads

9. After adding our payload, we could start fuzzing the request. Click the “Start Fuzzer” button in order to begin the fuzzing.



Start Fuzzer

10. Our “History” tab then will be switched to the “Fuzzer” tab, marking that the fuzzing has started. We could click the ‘Code’ head column to sort the response codes. In this case, we will sort the response codes ascendingly. After it finished, we could see the correct password that was returned with response code of 200. The password is: **admin123**

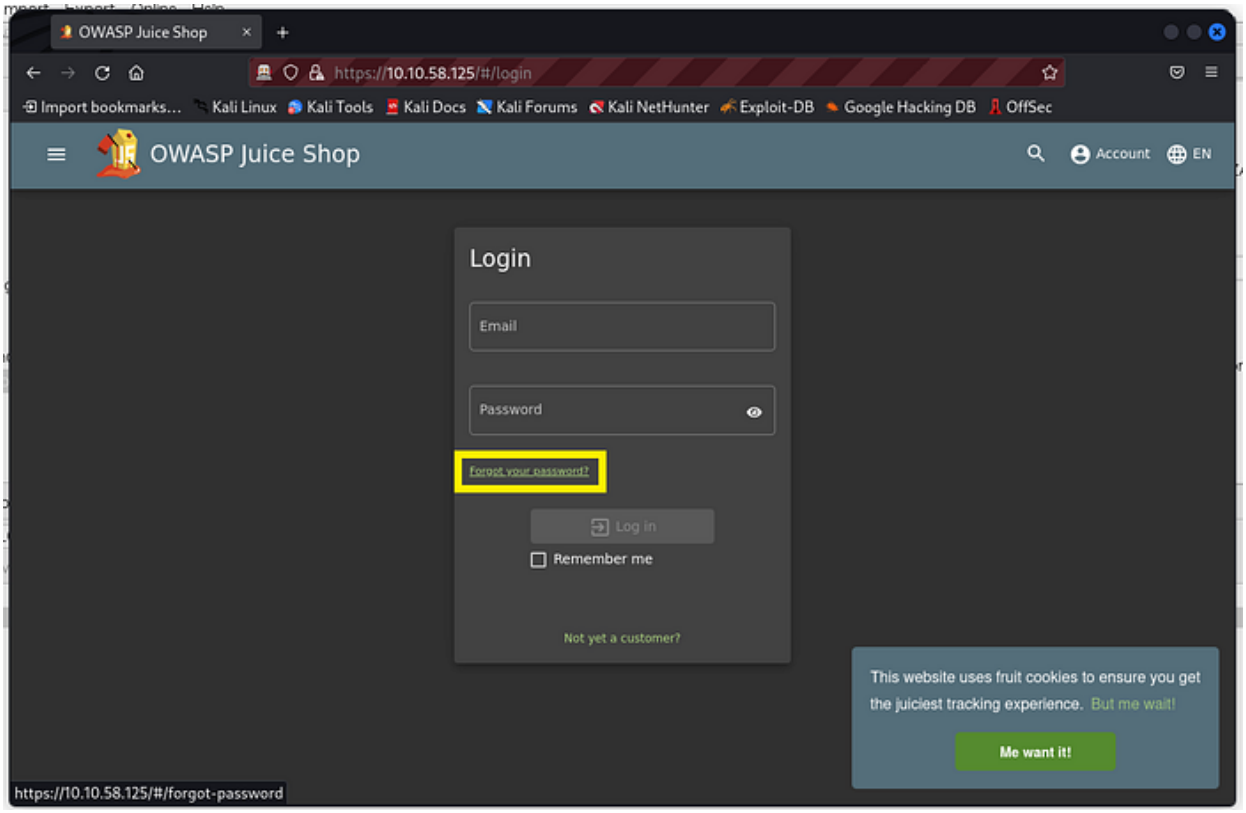



Password was found !

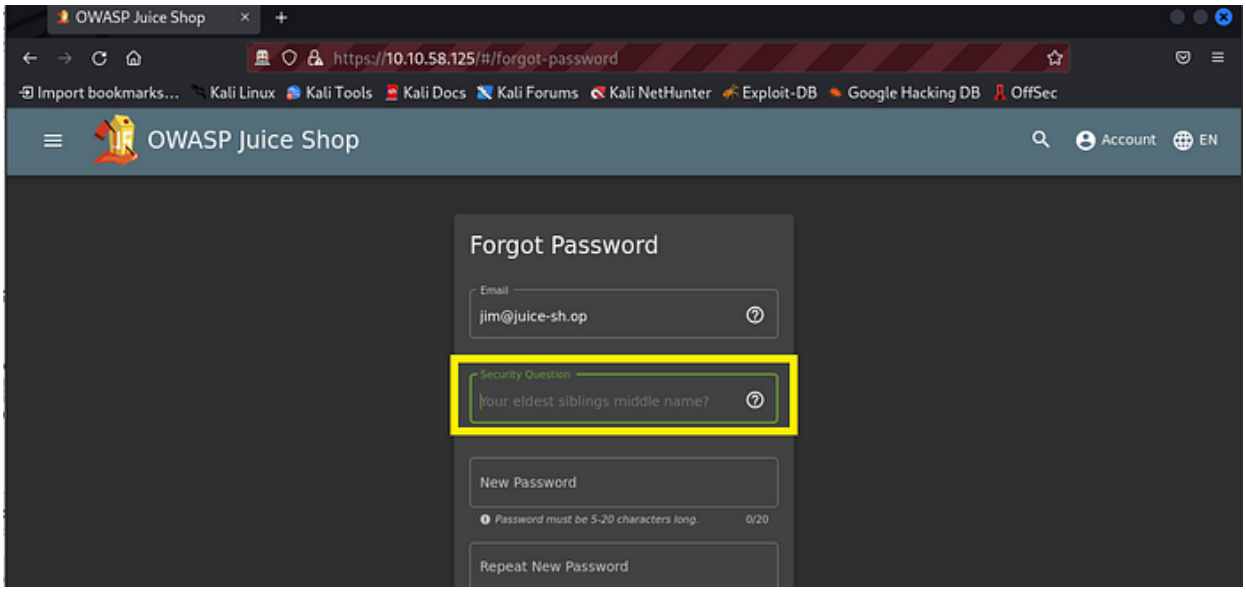
We could enter this credential into the login form and we should successfully logged in with an administrator account.

## #7 Reset Jim's password!

On this task, the exploitation is very simple and straightforward. We will head into the login form, click the underlined words of "Forgot your password" under the input bar of password. The form would be changed into 'Forgot Password' form. Next, we enter Jim's email address on the email input bar and the input bar of 'Security Question' would show the question as the input hint. The question is: "your eldest siblings middle name?".

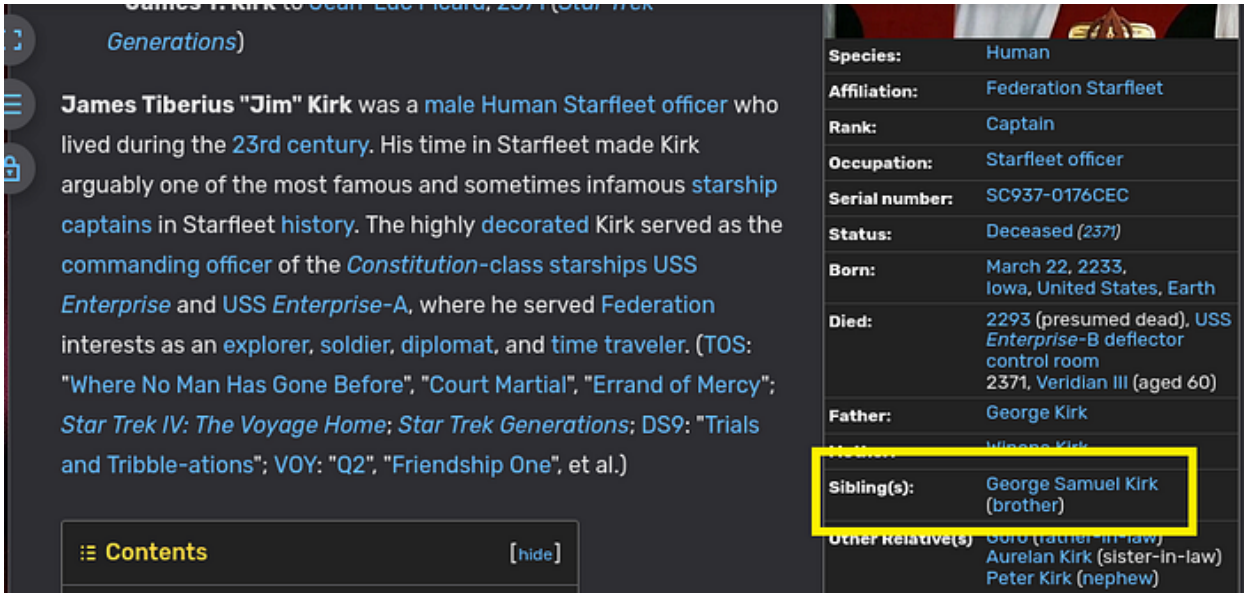



Forgot your password ?



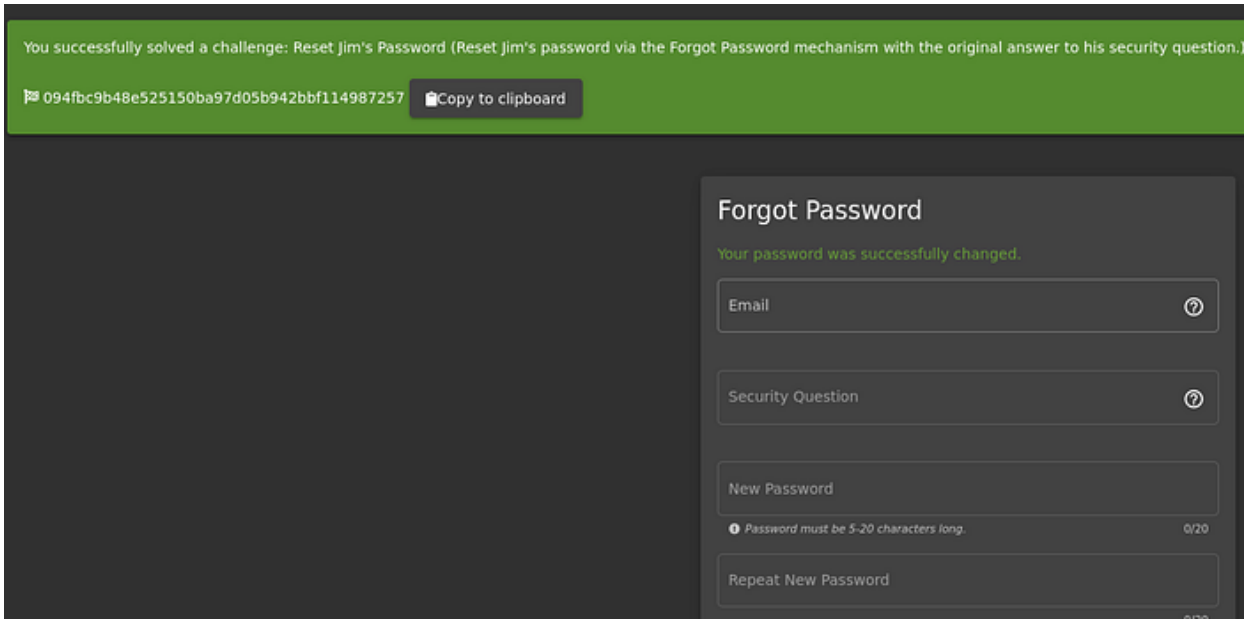

Forgot Password form

Since this question directly related to Star Trek, we could search the information about the shows on the internet. I prefer that the search engine do most of the heavy lifting this time so I used these keywords: “jim star trek siblings”. The results gave me a couple of websites that could lead to the information. I chose that appears to be an official fandom website for the show and the answer lies on the character’s profile information.



Fandom Webpage

Next, we enter all the information we need and input any password that we like.




## #8 Access the Confidential Document!

Before we actually know if there are any directory sites that shouldn't be accessed by the public, we must enumerate the web app directories. The reason in my opinion is that, most of the time directories that shouldn't be accessed publicly obviously are not put on a hyperlink or a click-to button to access.

In this case, we could follow along on how an ftp directory is accessible just like on the task's guide or we could do something else a little bit

different. I prefer the later and we will use other tools to keep it easy to follow along.

Let's use **Gobuster**. An open source tool built using Go programming language. This CLI based tool is powerful and lightweight at the same time.

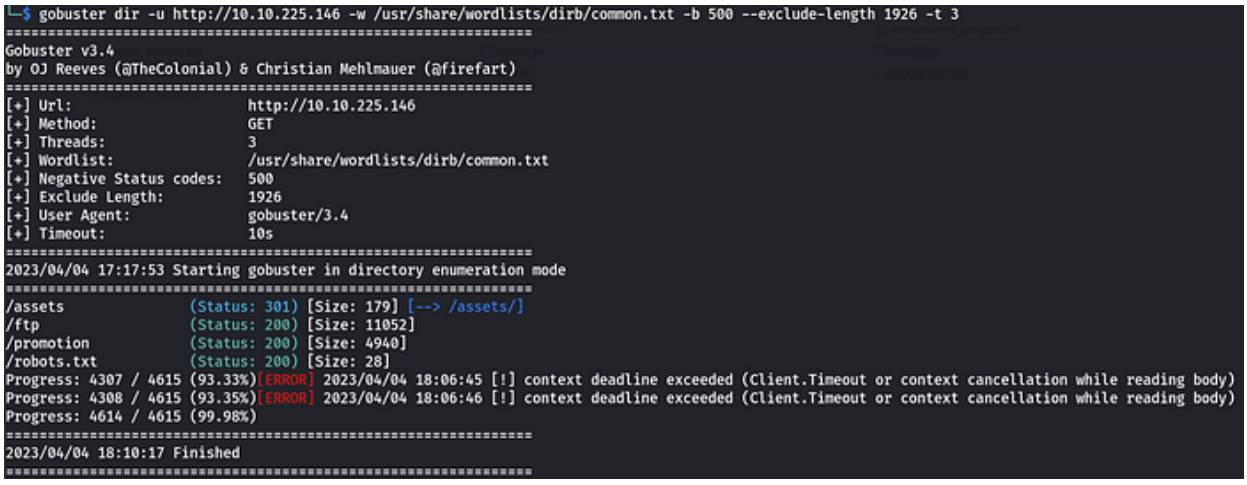
Here is the one liner of the command:

```
gobuster dir -u [WEB_ADDRESS] -w [WORDLIST_FILE] --exclude-length 1926 -b 500 -t 3
```

The command is explained here:

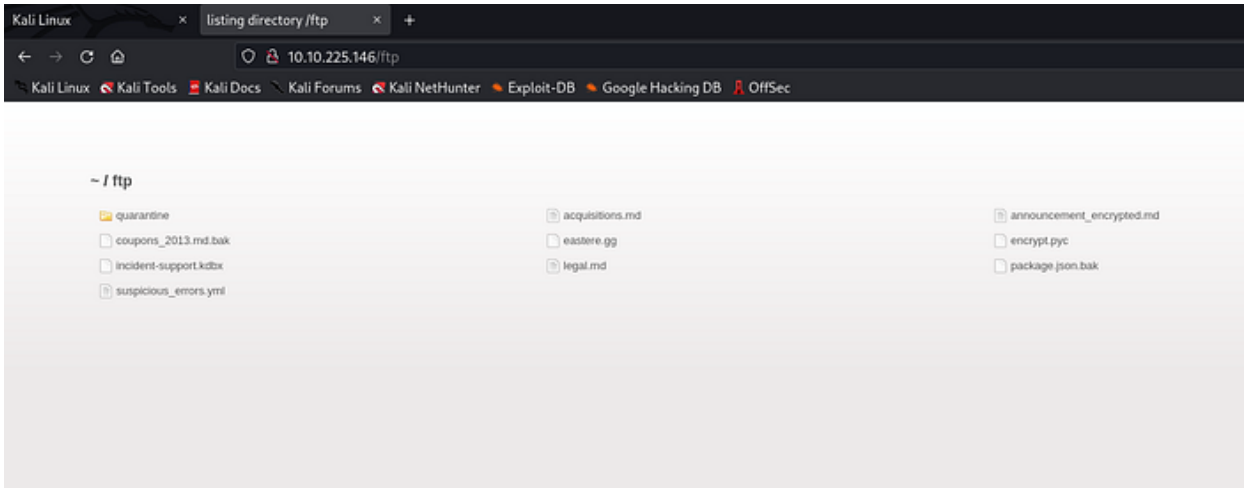
- **dir** : tool's command for using directory enumeration mode.
- **-u** : address/URL flag.
- **-w** : wordlist file flag.

- **-b** : blacklist flag, for excluding responses with particular status code. The status code of 500 is the default respond for non-existent page of the web app.
- **--exclude-length** : exception flag, for excluding responses with particular length. In this case, there is a certain response that would block our enumeration.
- **-t** : thread flag, for tuning up the attack performance. We use only three to avoid error because of too many requests at the same time.



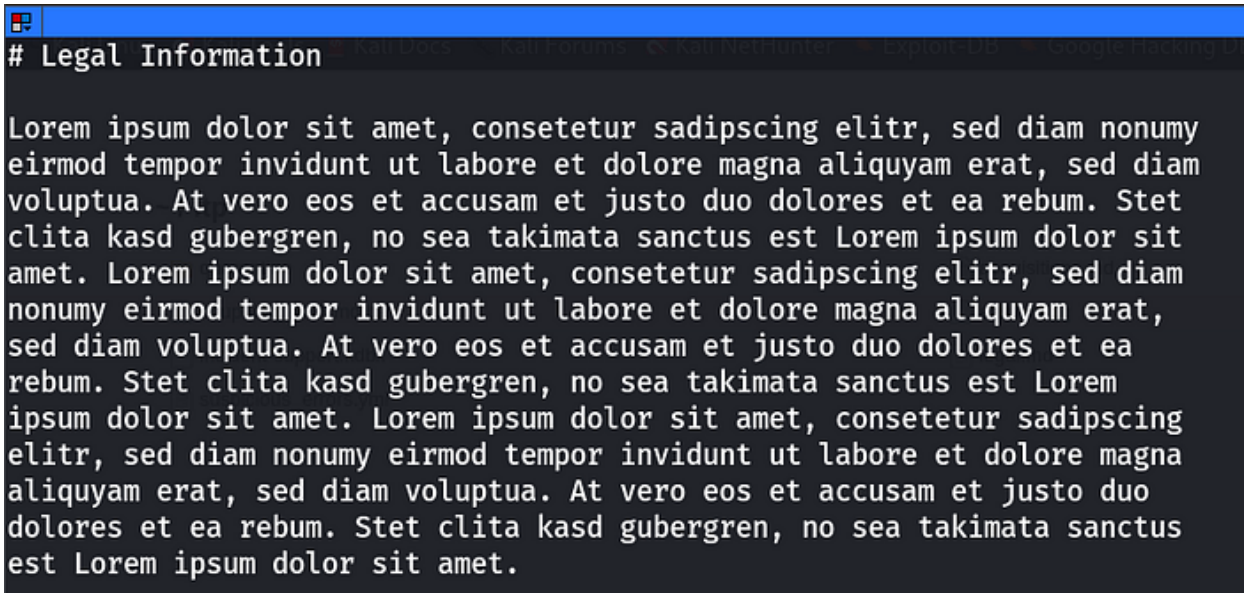

Gobuster

After the enumeration had finished, we could visit pages that returned with status code of 200. In this case, we're going to visit the *ftp* page.




FTP directory

There are many interesting files on this *ftp* directory, but there is one that would provide us important information about the company. It's on the "acquisitions.md". Let's download it and see what is inside the file.

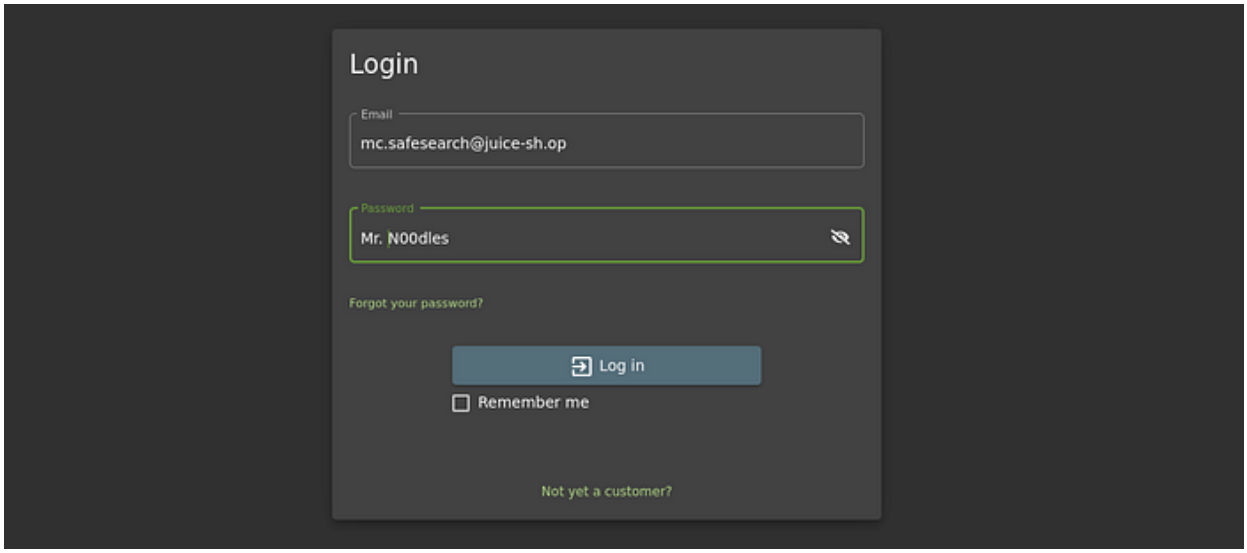



Turns out, it's just an example file. However, the structure of the writings indicate that this document should be confidential.

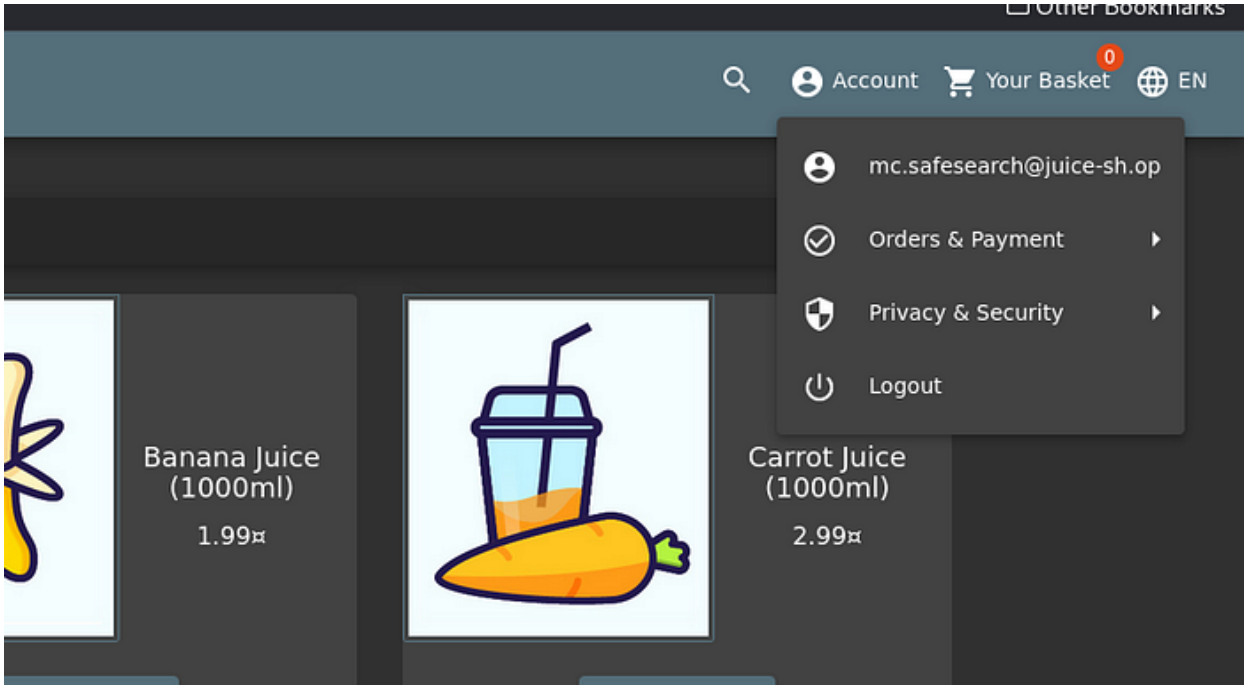
## #9 Log into MC SafeSearch's account!

This task probably an entertaining one because the song itself actually provide some good advice on keeping your password safe (Be careful with those shoulder surfers 😉). Nevertheless, we're trying to guess both his username and password. Just like what is written on the guide, on the lyric it's mentioned that his password is Mr. Noodles but the vowels was replaced with zeros. We know now his password is

actually “Mr. N**o**odles”. However, what would be the username ? Since the email address has a structure of **@juice-sh.op**, we could safely assume that the email address is : **mc.safesearch@juice-sh.op**




Login Form — MC. Safe Search

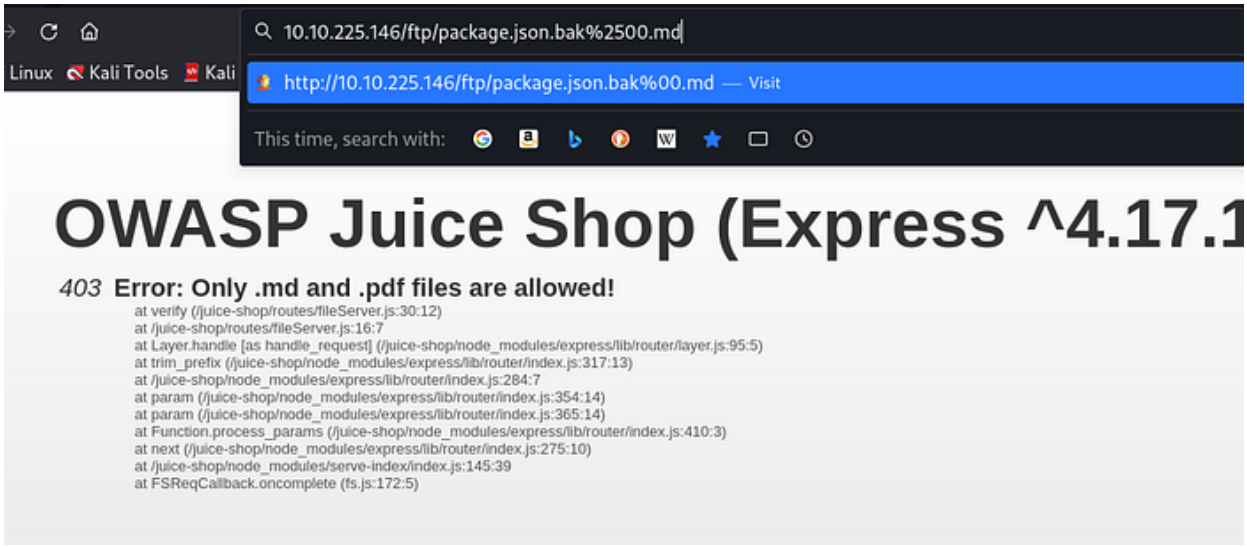



#10 Download the Backup file!

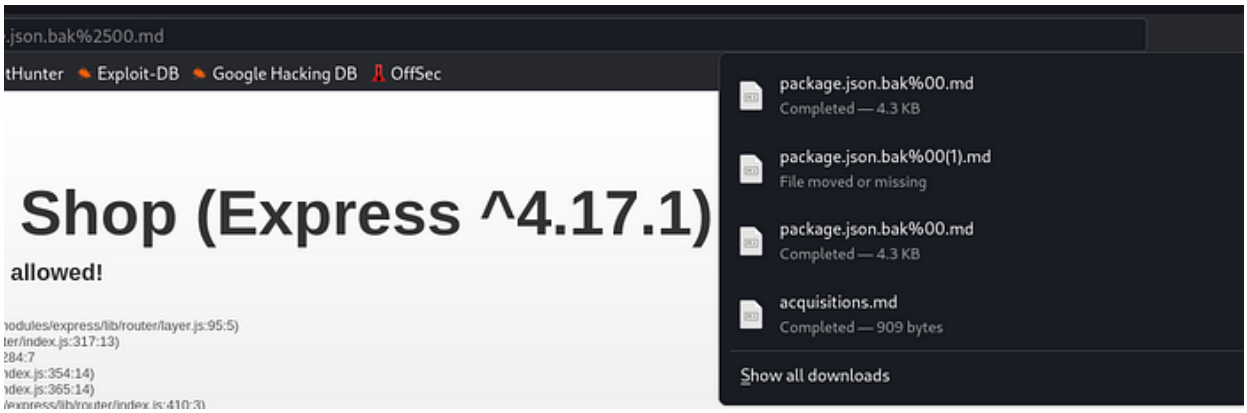
Bypassing a sanitized input requires some knowledge of how things works. In this case, the URL is sanitized so that we could only download files that only have extensions such as '.md' or '.pdf'. The URL however could be replaced with html encodings and so, we could replace some or add an encoded strings. On the guide, we will add a null terminator that would allow us to bypass the download sanitation.

Why it works ? The simple explanation that I could give right now is that when a URL request is read by a function (a function on a programming language), the sanitized request would read every characters and with no exception the null character. In this exploit, we add a percentage character encoding and two zeroes (%2500). When both are put together, the first encoding character would be read and that is “%25”. This encoding character would be translated as percentage by the function. Next, because the percentage and the two zeroes are paired together the function would read them again as another encoding character. This, in particular would be the null character.

Let’s try this exploit.




Bypassing restriction

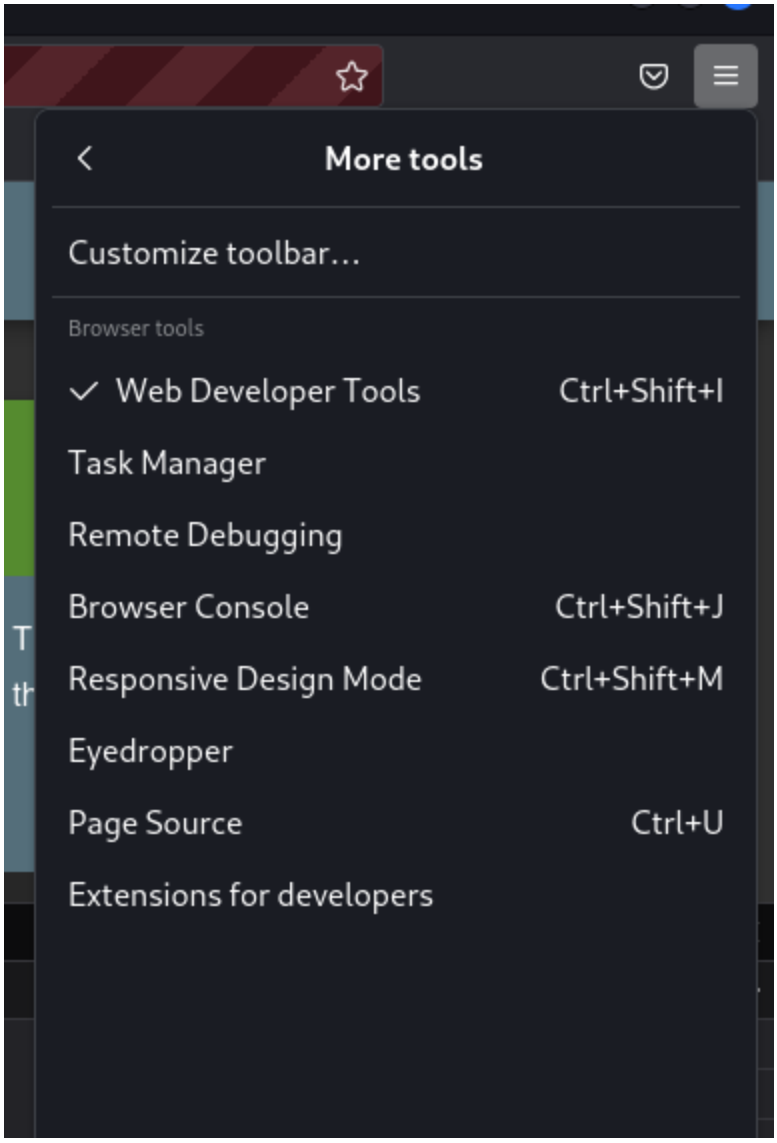



File downloaded

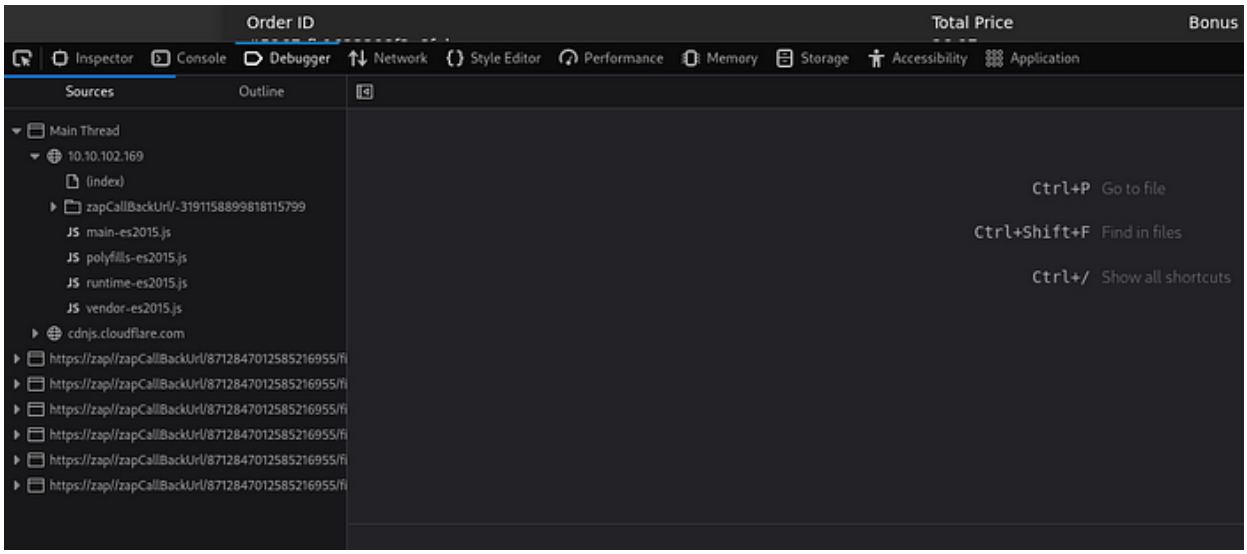
As we expected, we downloaded the file.

## #11 Access the administration page!

On this task, we are going to find pages that are available to be visited but no links provided on the site's page. The easiest way of finding this not so hidden page is checking through the app's runtime. Browsers have dev tools with debugger to check the app is running. We can access the dev tools by clicking the browser's application menu. In this demo, I used Firefox. Accessing the dev tools, I simply clicked the application's menu then 'More Tools'.

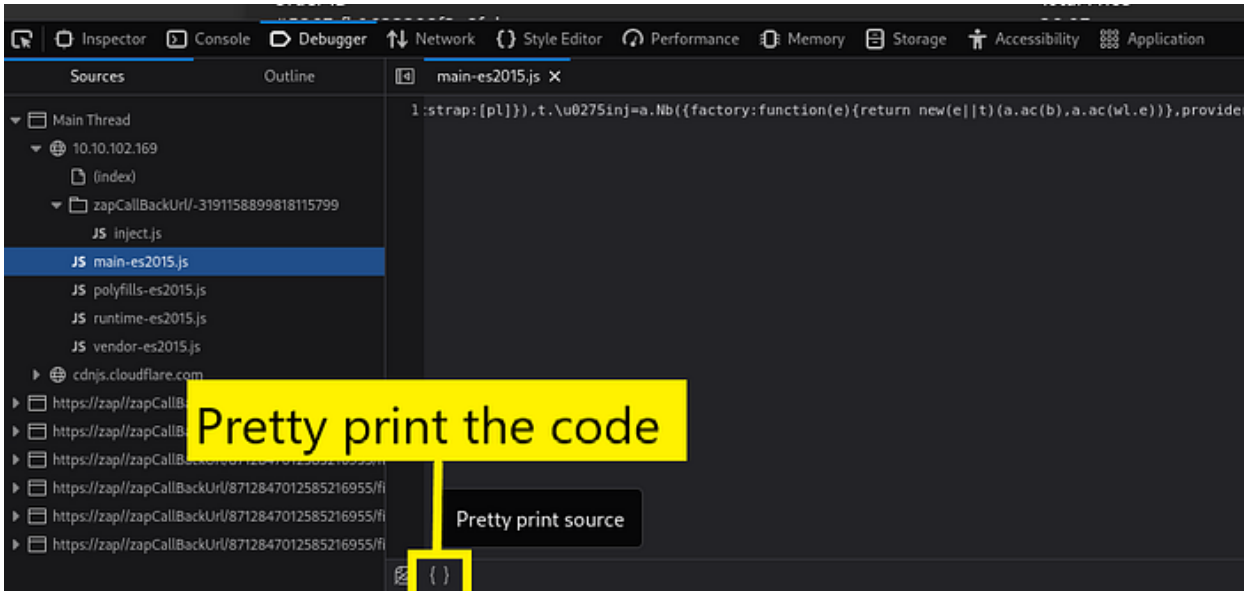



Firefox’s Application Menu

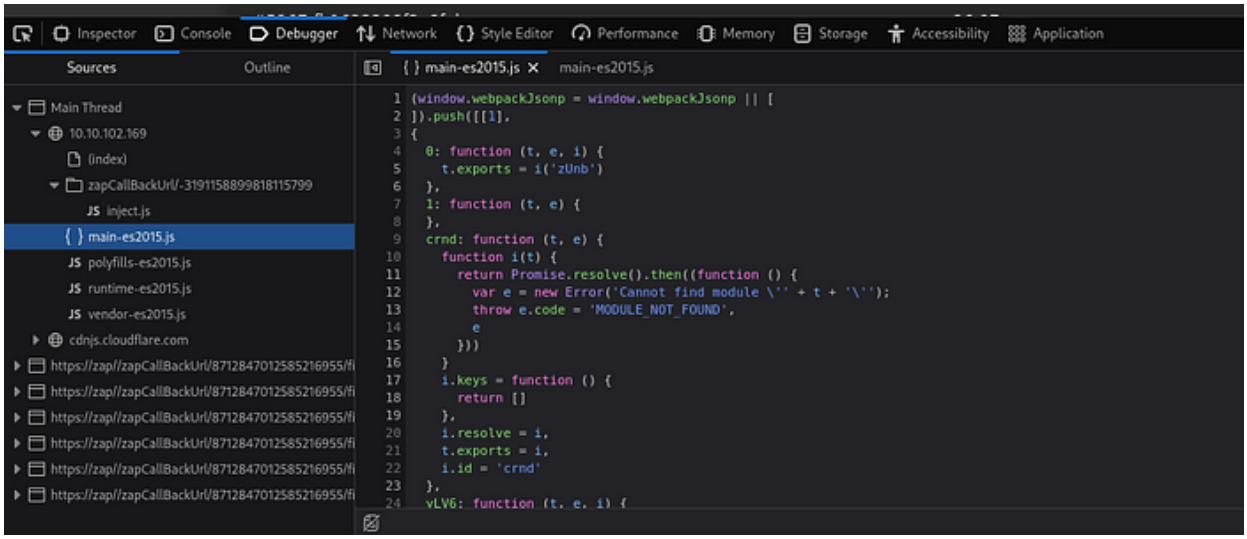


Dev Tools — Debugger

We’re going to check any related ‘.js’ files using the debugger. In this case, there is one interesting file that has name “main” on its prefix. The code inside the file is rearranged so that the app could be loaded quicker. In order to read the code in a more appropriate manner, we prettify the code by clicking the “Pretty Print”.



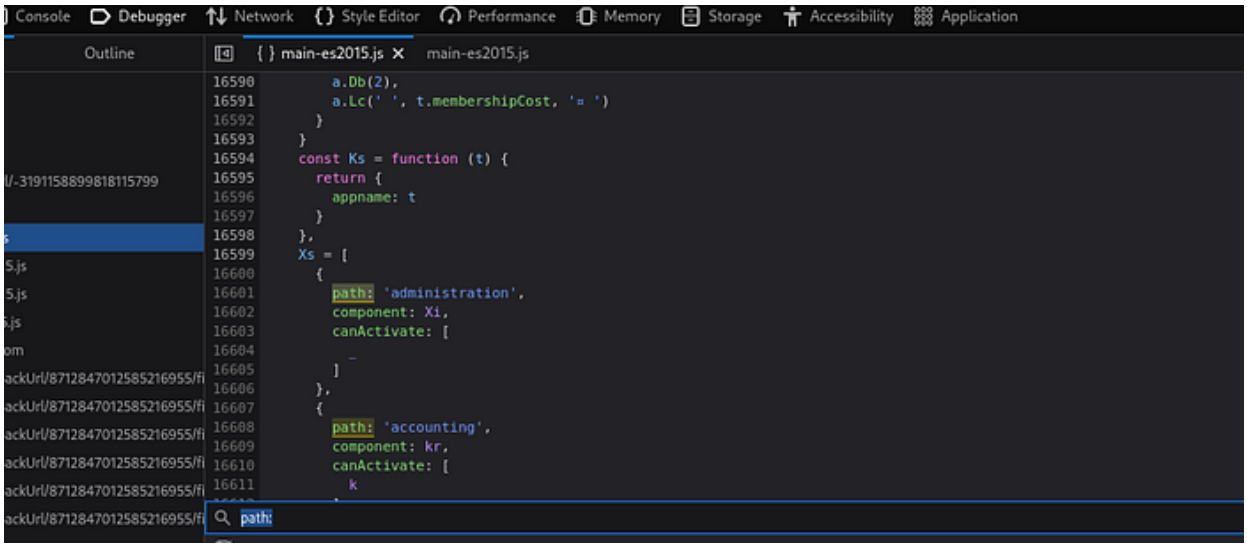

Pretty Print




Source Prettified

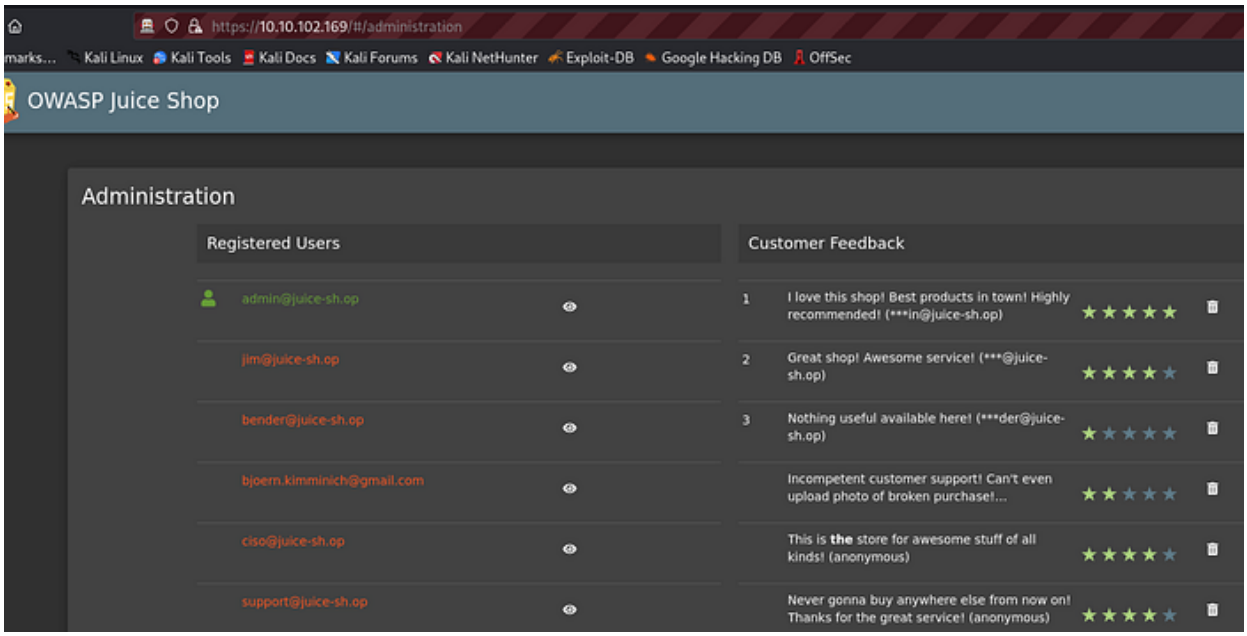
After all of the code are prettified, we could search for words containing ‘path’. We could do that by pressing both CTRL+F and type

in the words. It would return a lot of results searching this way, so in order to get a better result we refine our searching pattern. Type in “path:” on the search bar and we could see code blocks that are meant for directing web pages.




“path:”

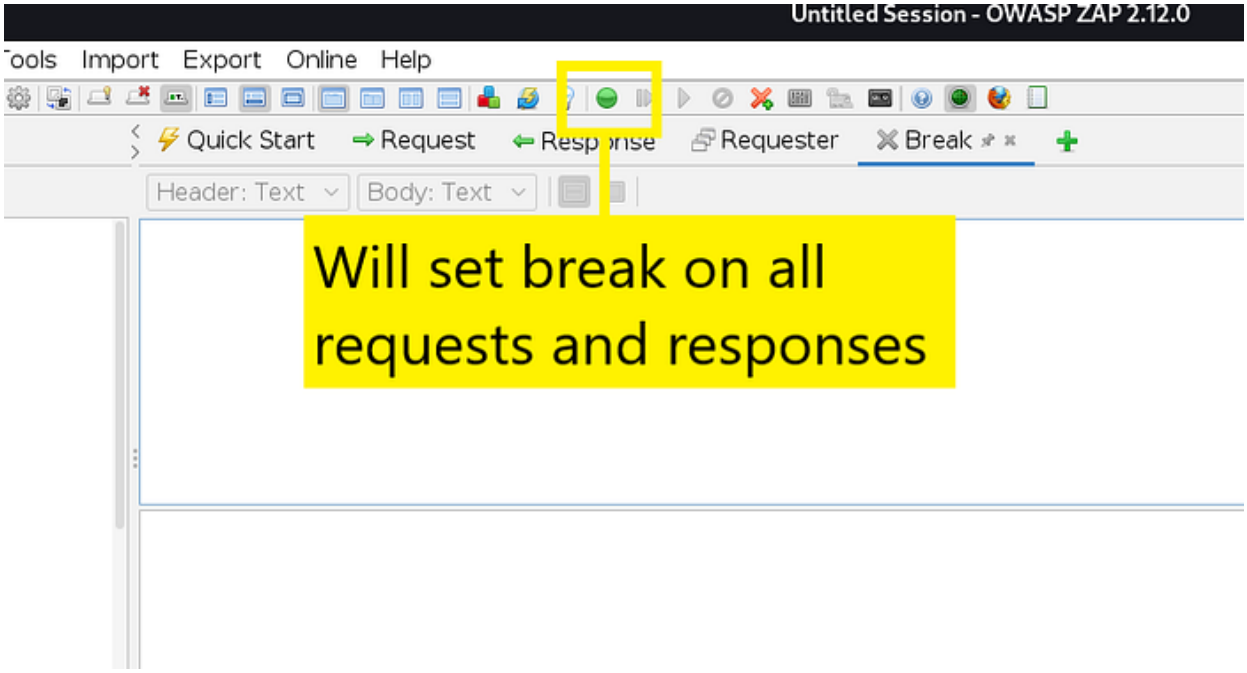
As for our target, there’s an administration page that we could visit upon logging in with an administrator account. Type ‘administration’ on the page URL and we would be greeted with a list of customer’s feedback.




## #12 View another user's shopping basket!

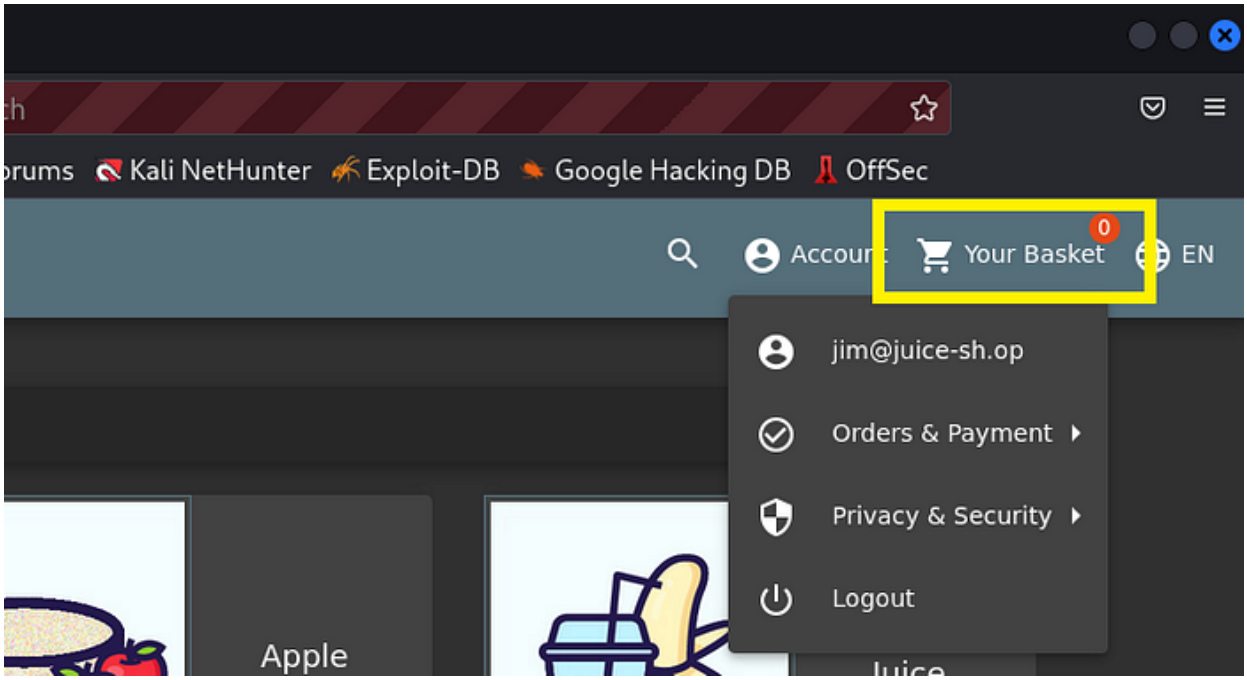
In this task, we're going to view shopping basket that could be viewed by any user that has logged the app in. We're going to use **Zap** again to intercept requests and manipulate them.

First, let's login using Jim's account. Next, before we're going to view Jim's shopping basket we're going to intercept the request. Click the green round button (set breakpoints) to 'break' requests. Once clicked, it would turn to red and we're ready to intercept requests.



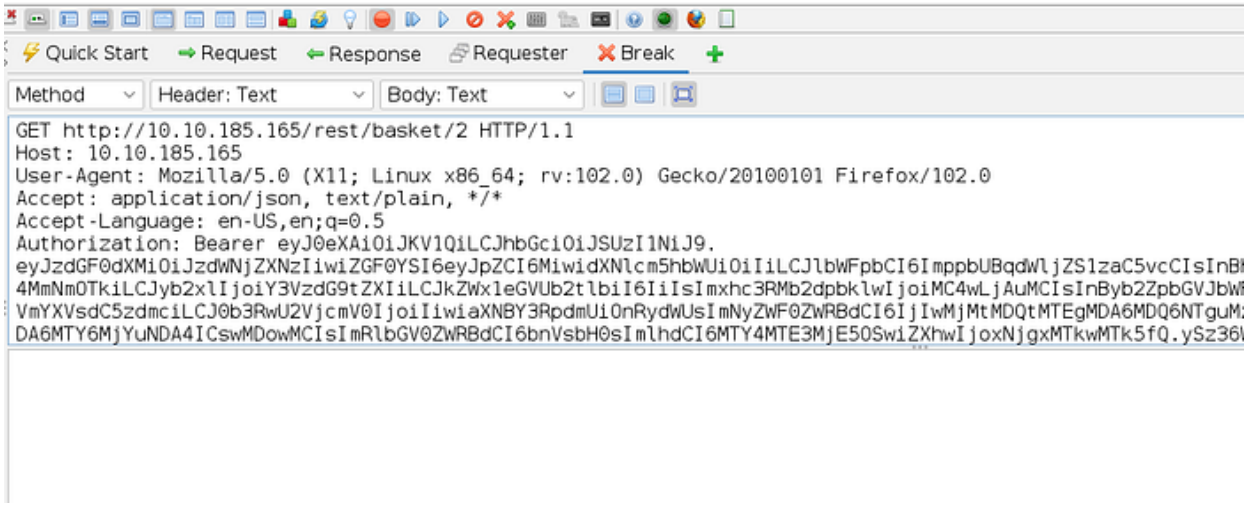

Break points

Unto the next step, we're going to view Jim's shopping basket. Click the 'Your Basket' icon.



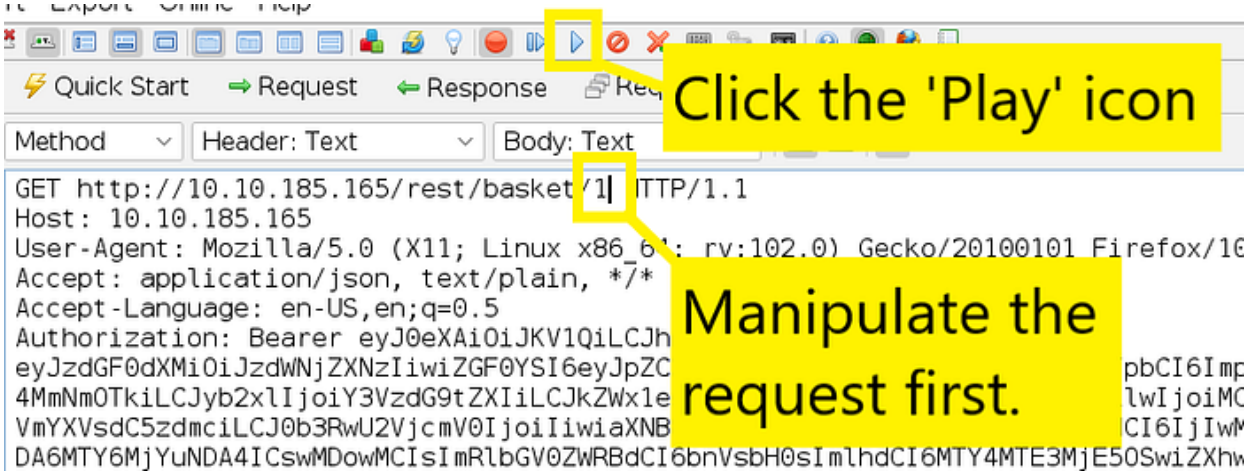

Your Basket cart

Now, we could view our ‘break’ request on **Zap**.



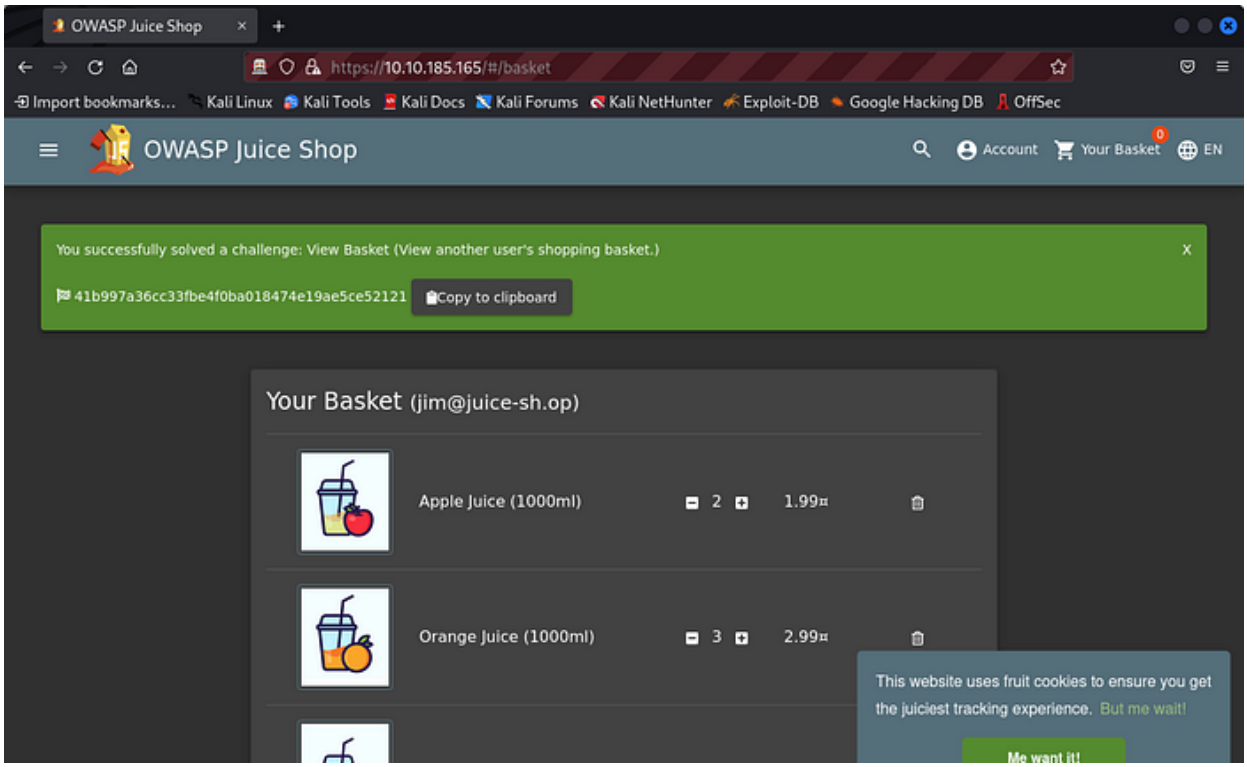

Intercepted request

On this intercepted request, we could manipulate it by simply changing the numbers. Next, we send the request by clicking the ‘play’ icon.



Manipulated request

As expected, we could see the other’s list of shopping basket.



Admin's shopping basket

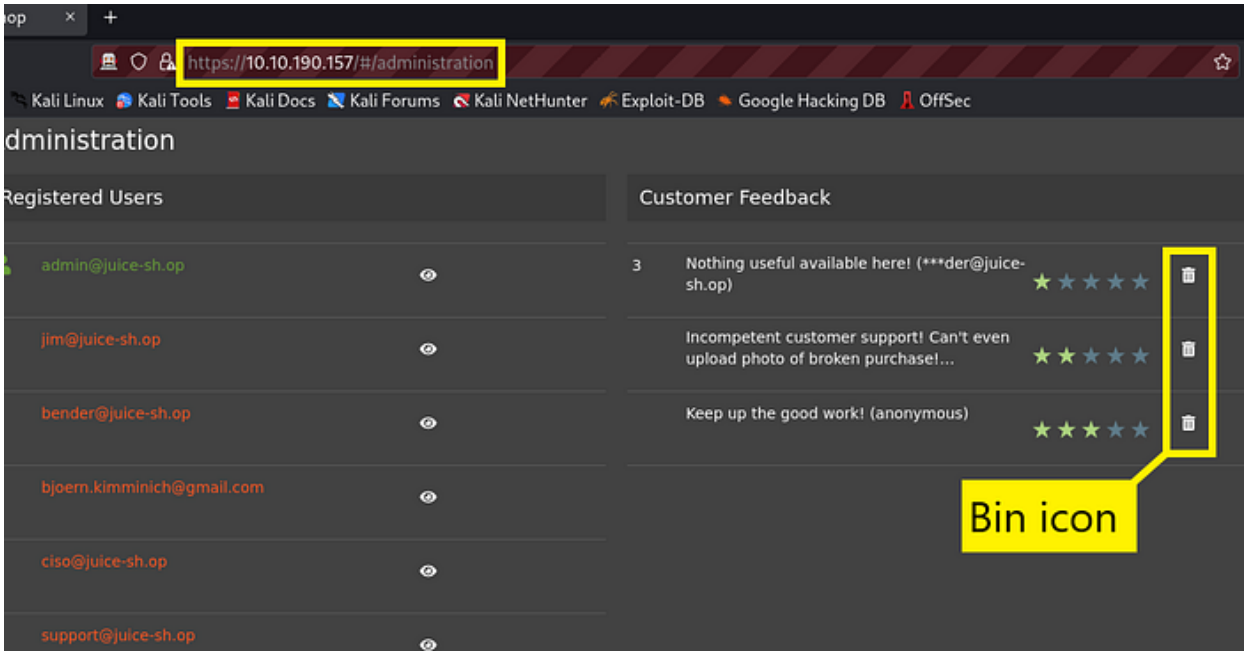
The reason why this request works is that there's a broken authorization on the request. There's a particular name for this type of vulnerability, it's called **Broken Object Level Authorization** (OWASP API Security Top 10). The request should've been checked for the correct permission to view the data that was provided. One of the simplest way to do this is by providing authorization token or any permission identifier on all of the shopping basket lists that linked with their accounts. With this way, any requests that attempt to view

shopping basket that doesn't belong to its ID would not be possible by cross checking both the account's unique authorization token with the basket list's.

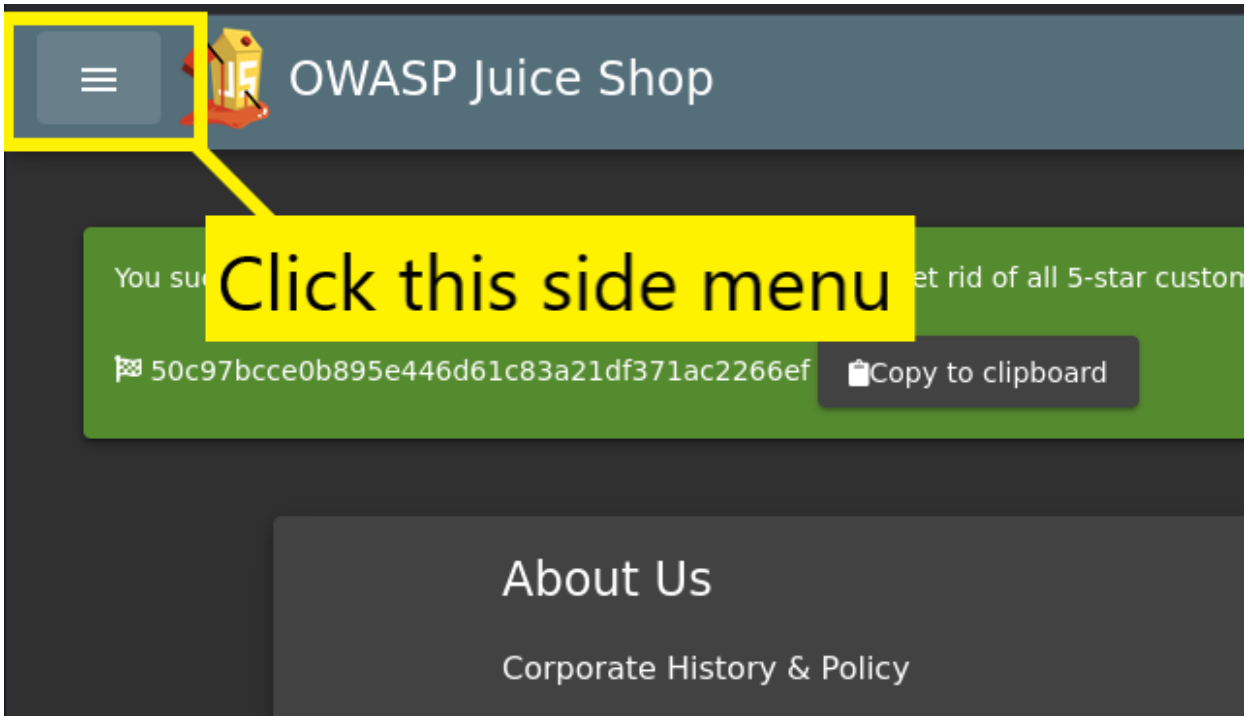
## #13 Remove all 5-star reviews!

This task should really not be replicated in a real case settings (There would be written rules on the contract, be vigilant), but for the sake of fun in this practice let's remove all of the five star reviews ! In fact, let's remove all four stars and above reviews 🙃.

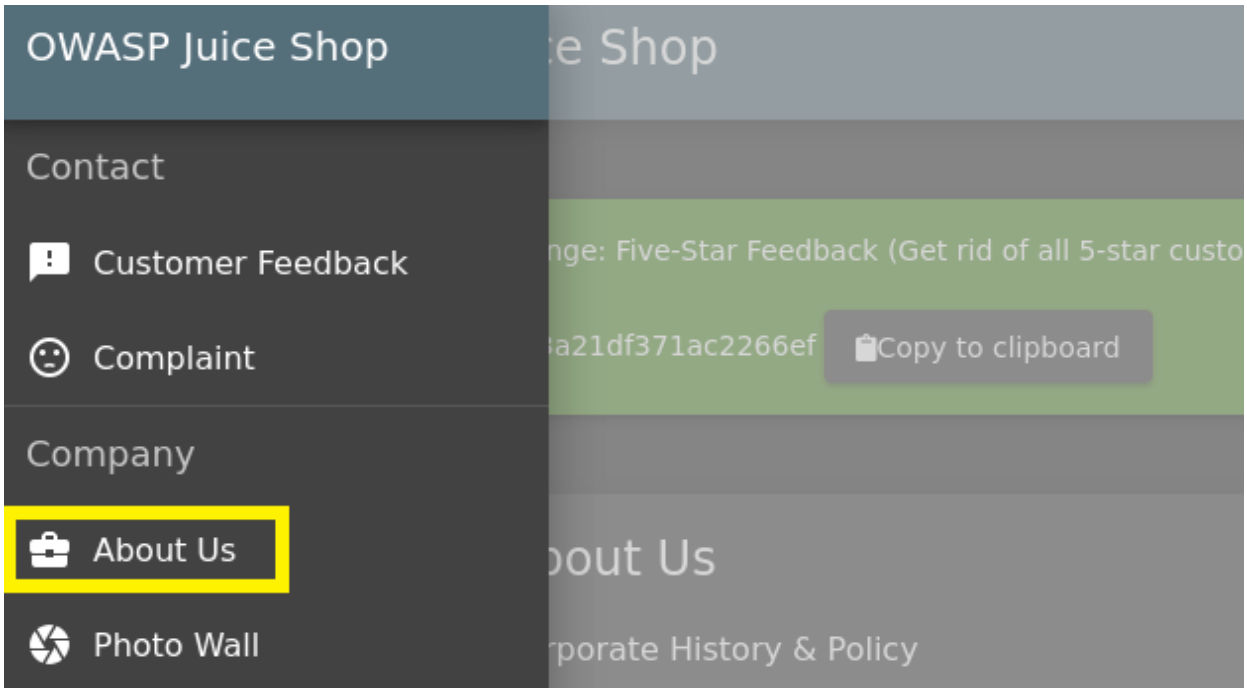
Let's log out from our previous account, then log in with an administrator account. Next, we would head to the administration page by typing 'administration' on the URL and we delete the reviews by clicking the bin icon.



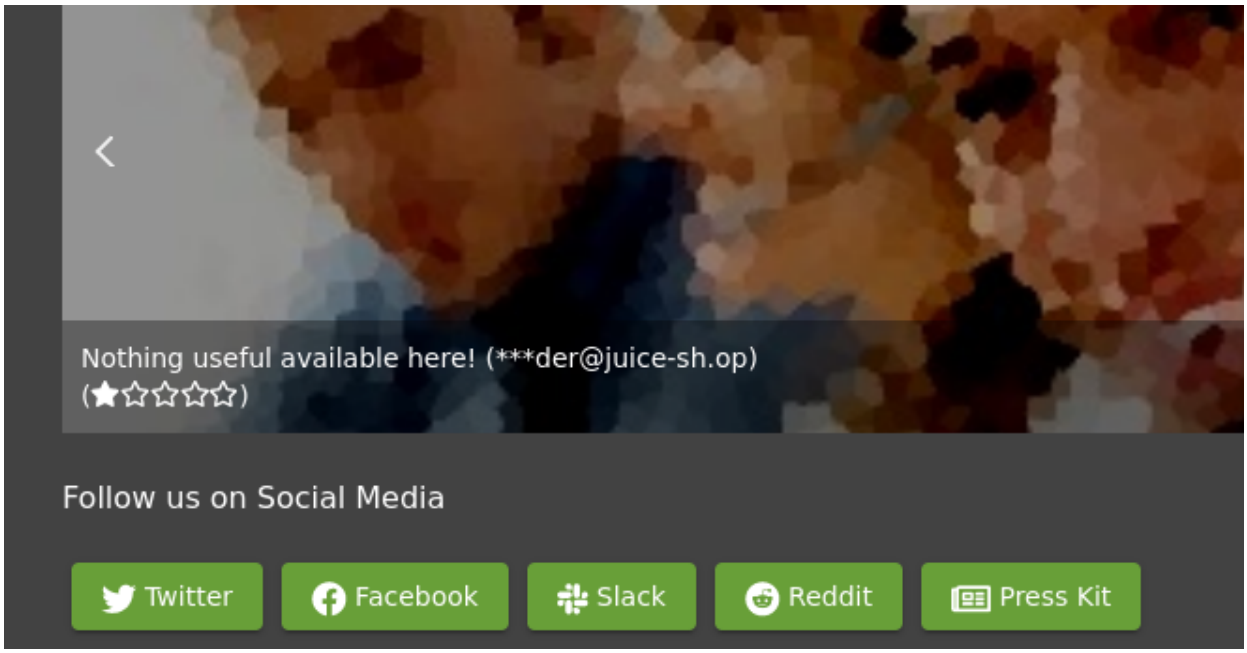

On the ‘About Us’ page, we could see all of the reviews. Click the side menu then click the ‘About Us’ inside the menu list.


OWASP Juice Shop
Click this side menu
50c97bcce0b895e446d61c83a21df371ac2266ef
Copy to clipboard
About Us
Corporate History & Policy

Side menu


OWASP Juice Shop
Contact
Customer Feedback
Complaint
Company
About Us
Photo Wall
Copy to clipboard

‘About Us’

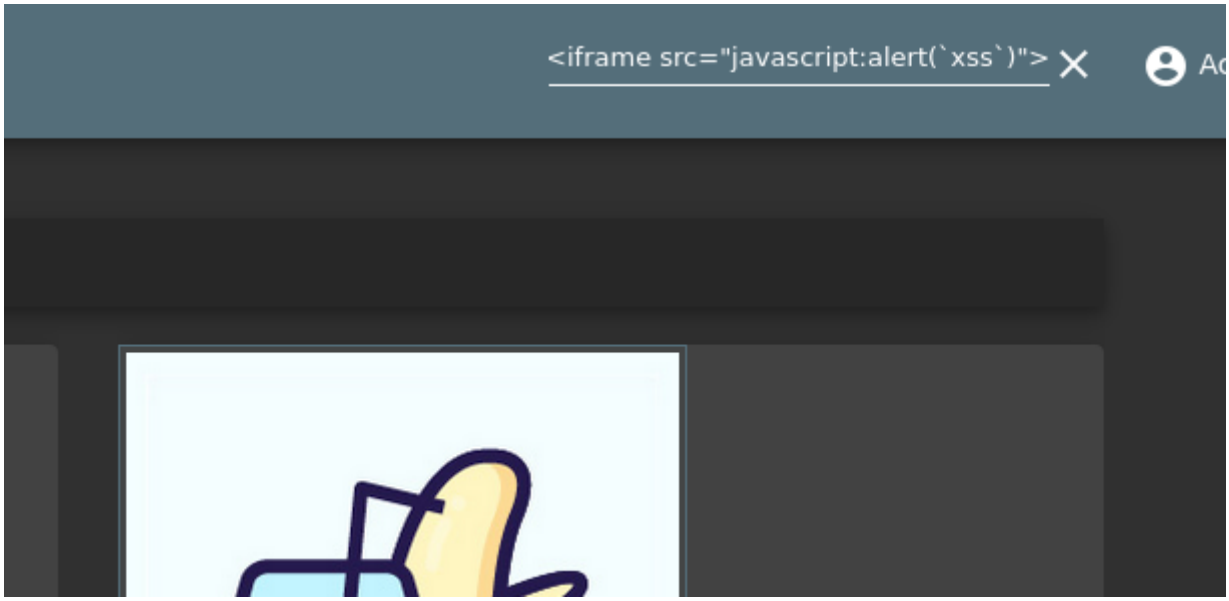



## #14 Perform a DOM XSS!

To perform this DOM XSS, we simply type in an html element that would invoke a Javascript alert. In order for this to work, write the payload exactly like this below :

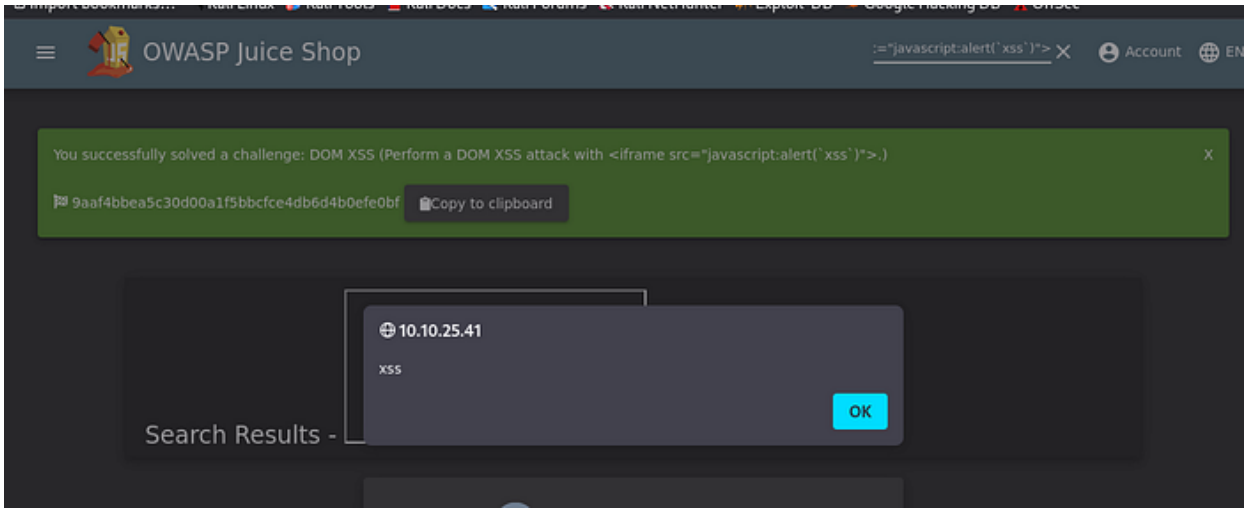
```
<iframe src="javascript:alert(`xss`)">
```

Let’s try this exploit on the search bar. Type the payload in then press Enter to run the exploit.



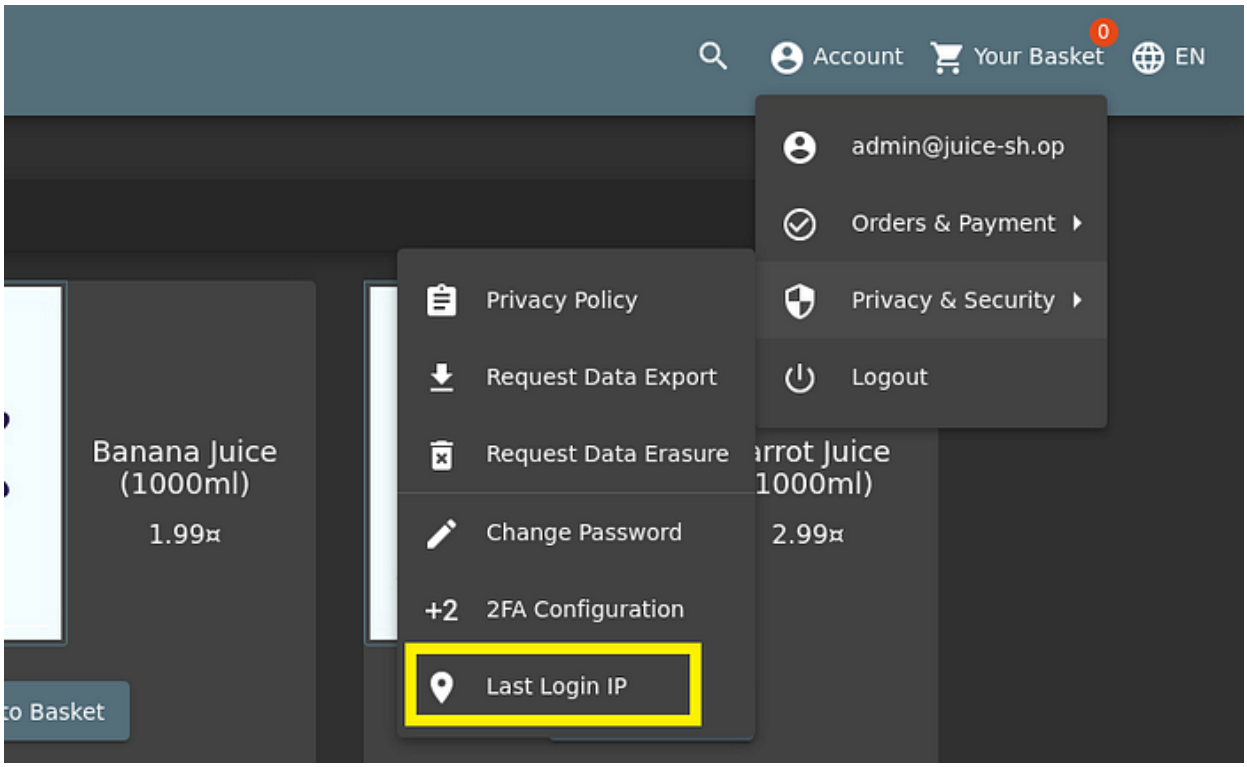

Search bar, iframe DOM XSS

The exploit works and we received a flag.



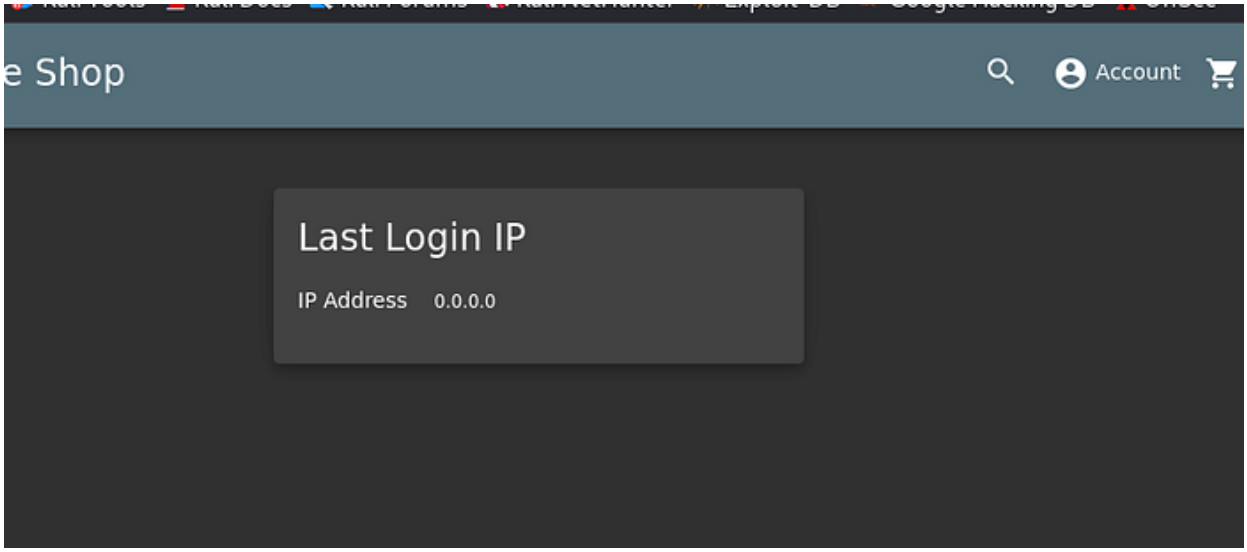

## #15 Perform a persistent XSS!

In order to perform a persistent XSS, we must be logged in with an administrator account. Next, we head to the last login page.

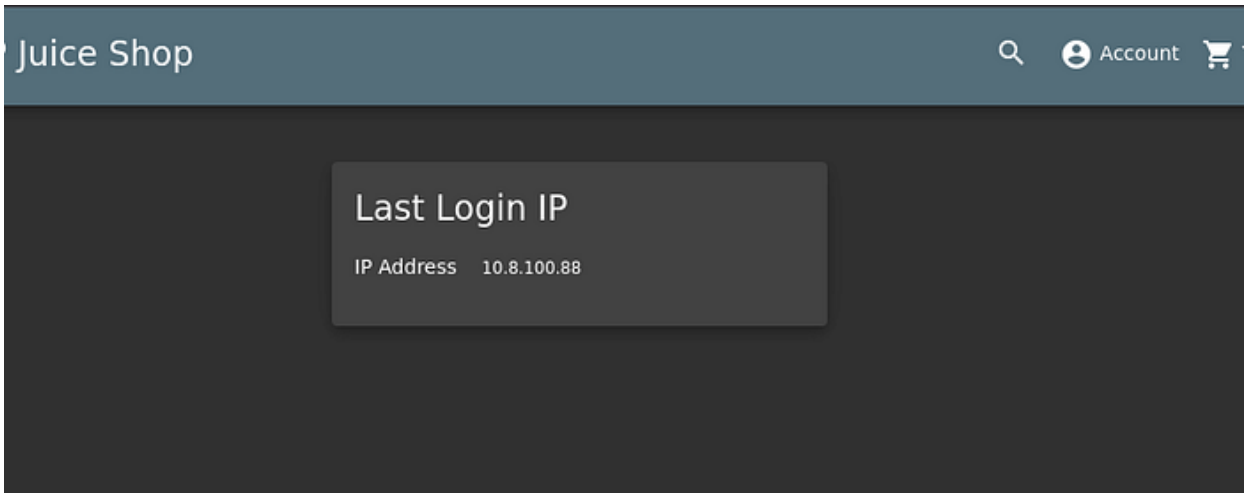


Privacy & Security dropdown menu

On the page we will be shown with IP address of 0.0.0.0 if we log in for the first time. After we try logging out then logging in with the same account, we will have our last IP address we used for logging in.



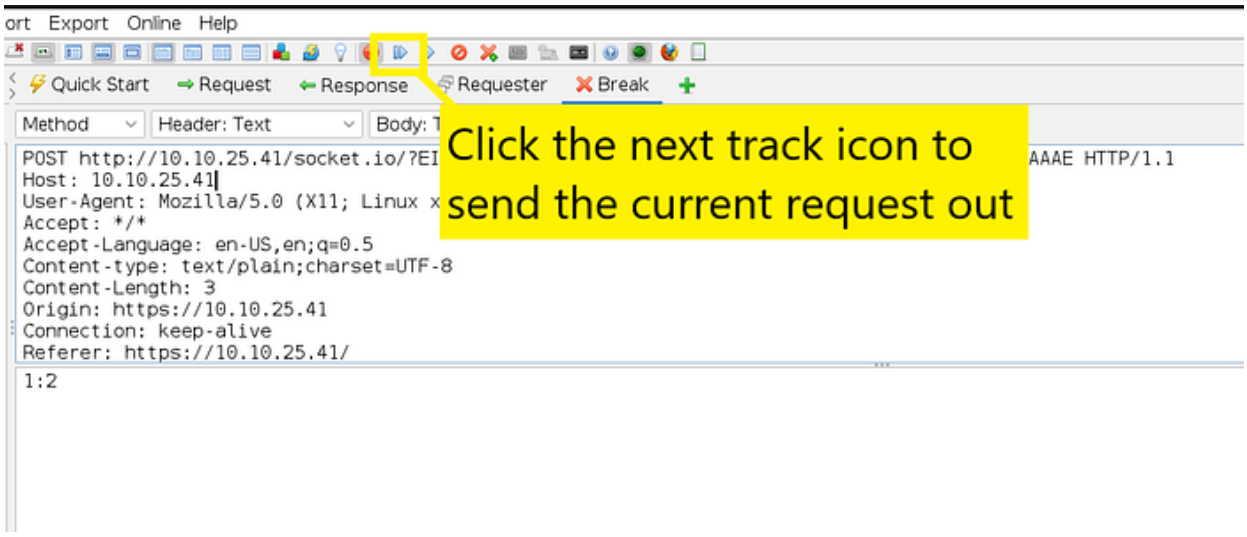

Last Login IP — Not recorded



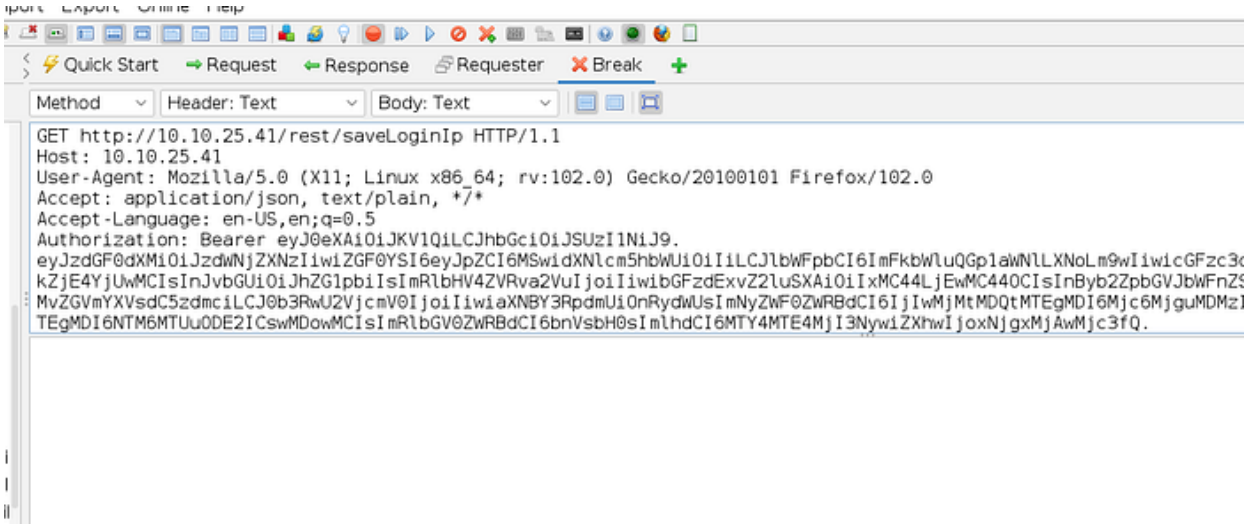

Last Login IP — Recorded

Let's intercept the last login IP request and manipulate it in order to get persistent XSS. Head to our **Zap** and click the green round icon to intercept. Next, we log out from our current account. After that, we

could see our intercepted requests in the ‘Break’ tab. If the current request is not what we’re looking for, we could send the current request and view the next by clicking the next track icon. If we missed the request, we could simply repeat the process by turning off the intercepts, quickly log back in, re-intercept our requests and log out.



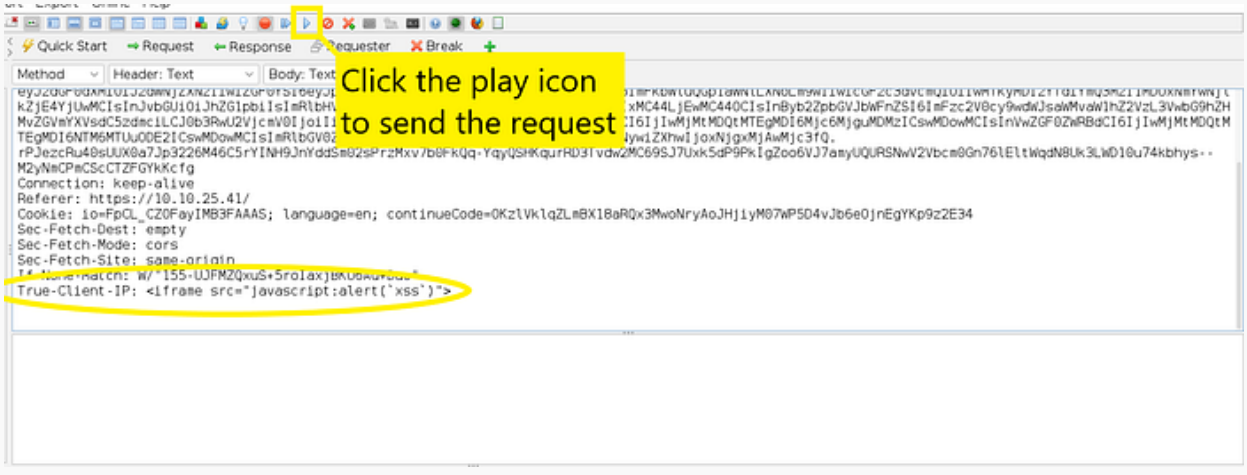

Next request



Intercepted request

After we intercepted the request, we then edit the request's header by directly editing inside the 'Break' tab. We would add a 'True-Client-IP' header and set the value with previous XSS payload. Put the header on the very below of the request's header. Again, type the payload in exactly like below :

```
True-Client-IP: <iframe src="javascript:alert(`xss`)">
```

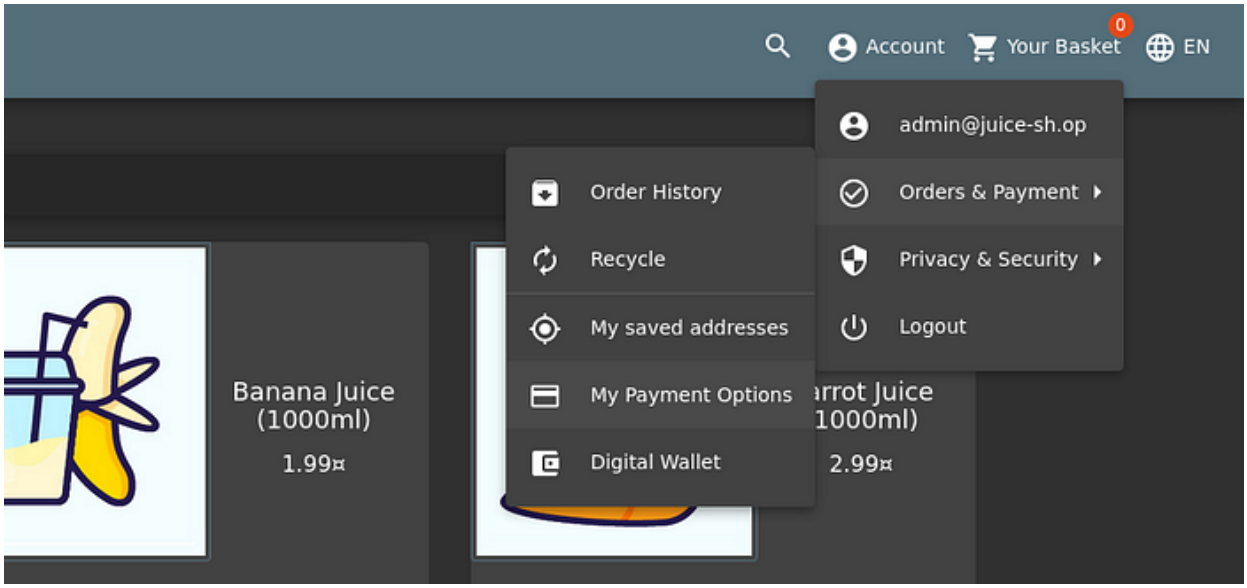


Payload on the request's header

We successfully injected the persistent XSS by logging in back with the same account and head to the last login IP page.

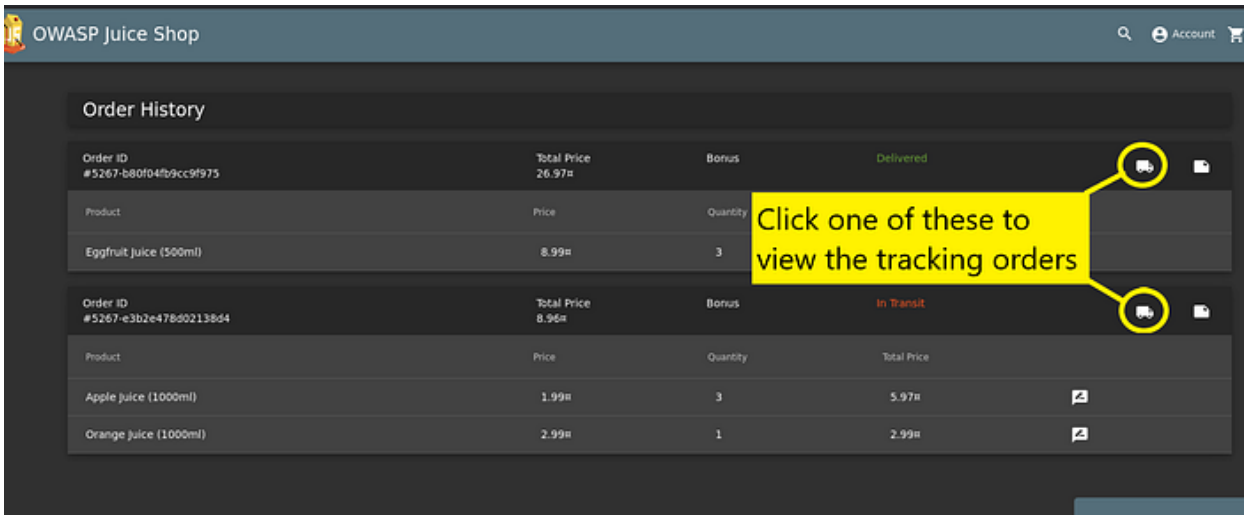
## #16 Perform a reflected XSS!

In this task, we're going to perform a reflected XSS by manipulating other request in the order tracking page. We will intercept the order tracking request and begin manipulating the request's parameter. Let's head to the 'Order History' page and turn on the intercepts.



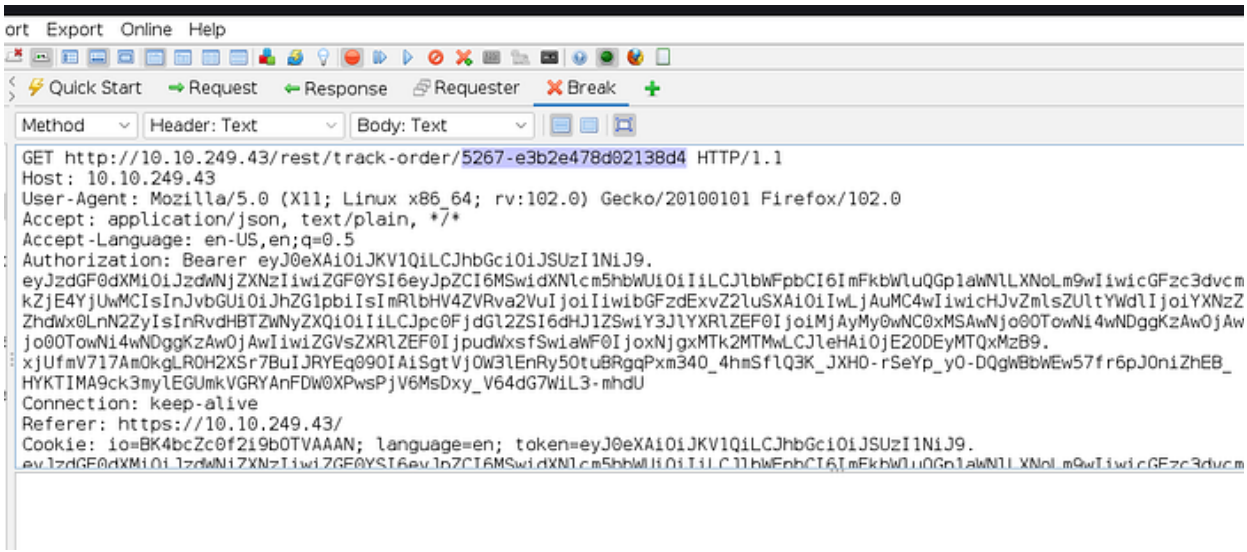

Orders & Payment menu list

On the 'Order History' page we're going to check for order tracks. Click the truck icon on any of the history list.



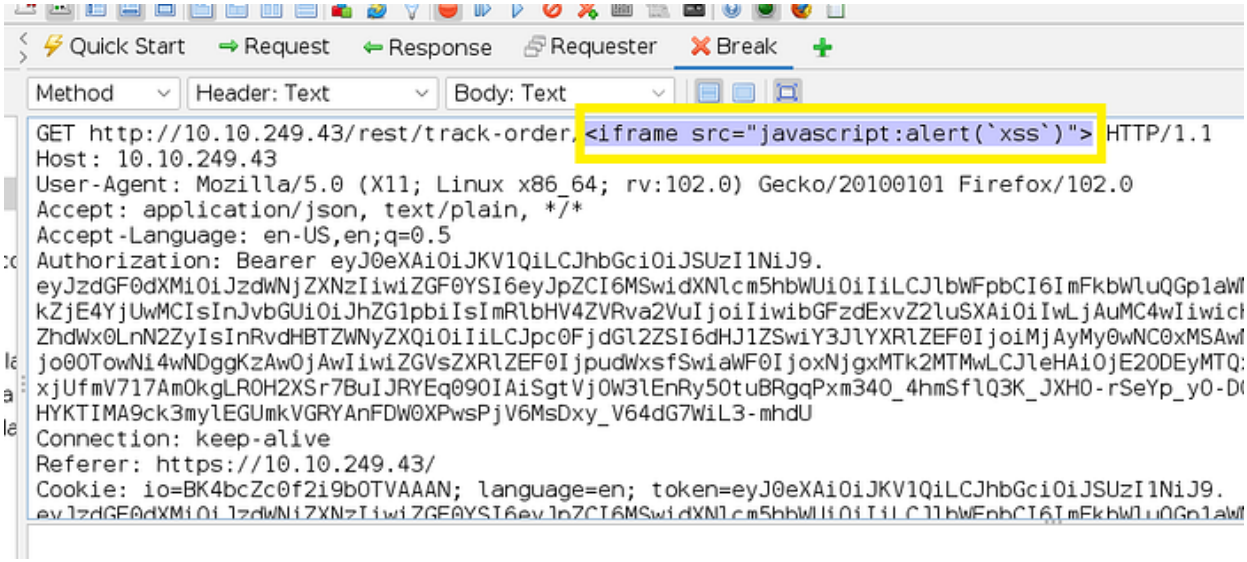

Order History Page

Next, we view our intercepted requests.



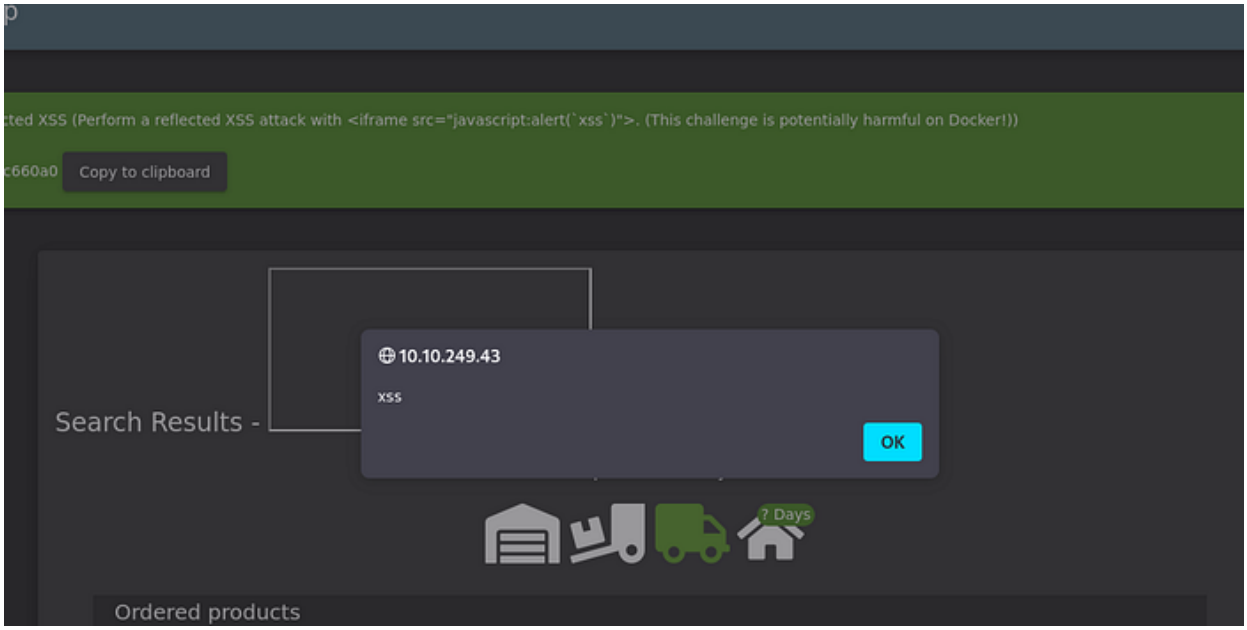

Track-Order API

From here, we could change the tracking ID with the XSS payload and send the request.




API manipulated

We exploited the app using reflected XSS successfully




Reflected XSS

There’s a reason why we cannot execute the same thing by typing the payload on the URL. The URL sadly would be encoded with HTML encoding before the request is executed. Therefore, the payload won’t work because it would only be read as a string not as a script.