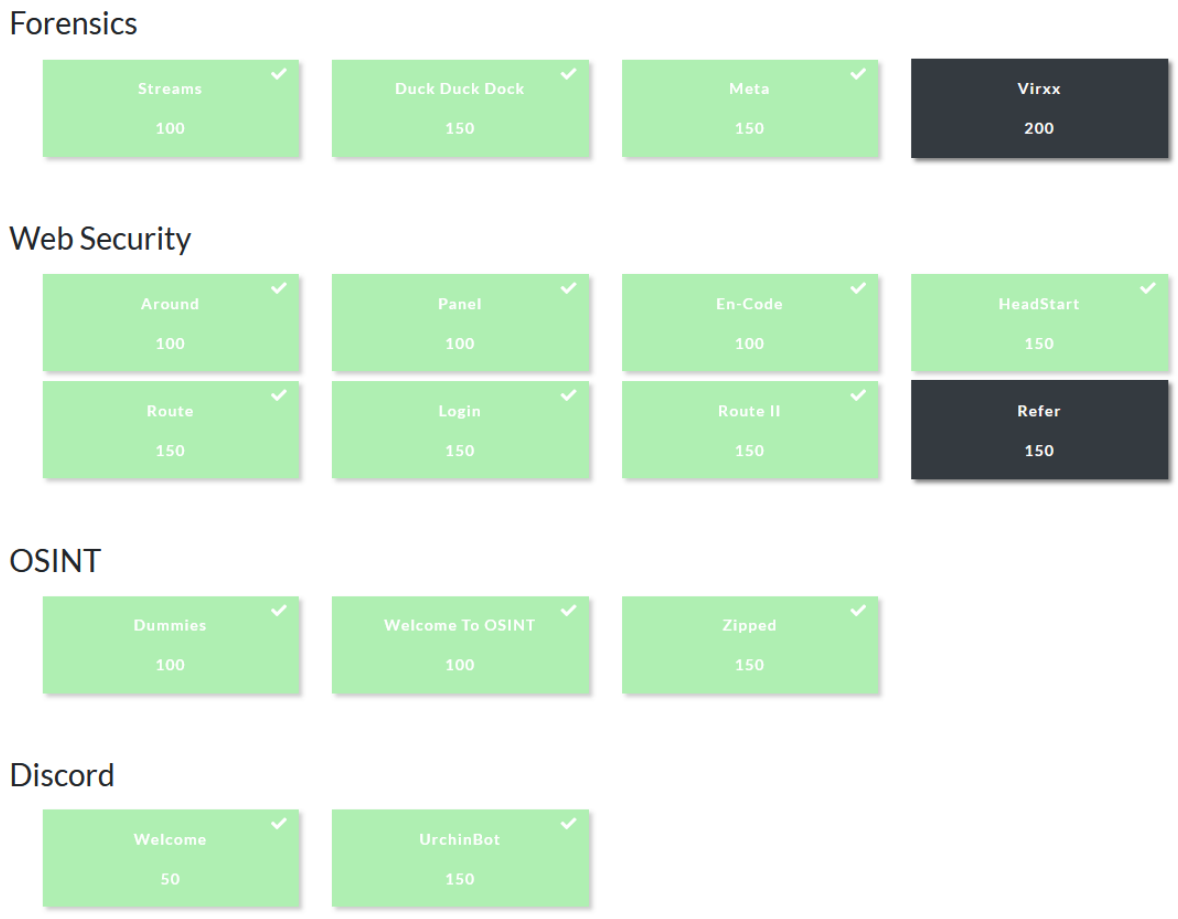
Author: Kharim Mchatta

# Article: URCHINSEC CTF WRITEUP

Date: 3/6/2022

Urchin recently hosted their ctf on 3/5/2022 which was in form of jeopardy. This was an interesting ctf which I would rate it as a fair one due to the fact that the challenges were not complicated. I for the other hand focused solely on the web, forensics, OSINT, and discord challenges.

Bellow are the challenges that I managed to solve, on the web you can see that I didn't do refer because the challenge was discarded by the author due to technical difficulties, for the Virxx I didn't do it simply because I was out of time.

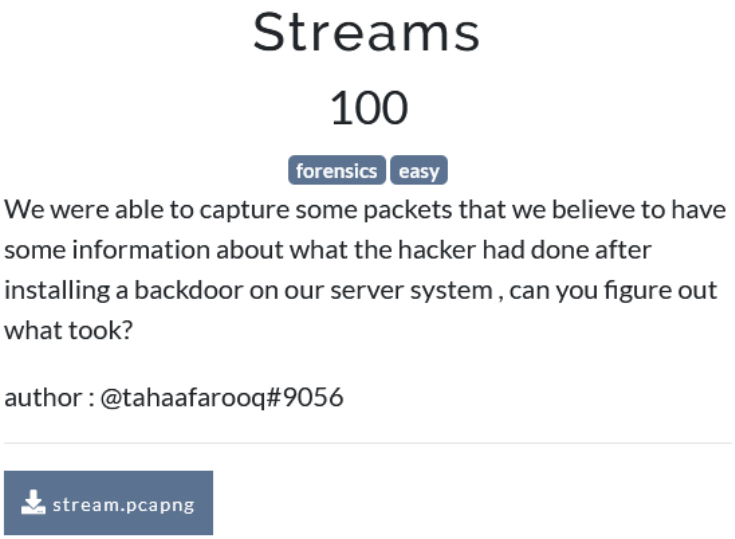


# FORENSICS CHALLENGES

In this category there were 4 challenges and I managed to do 3 out of the 4. Here is my approach on how I solved the challenges in this category

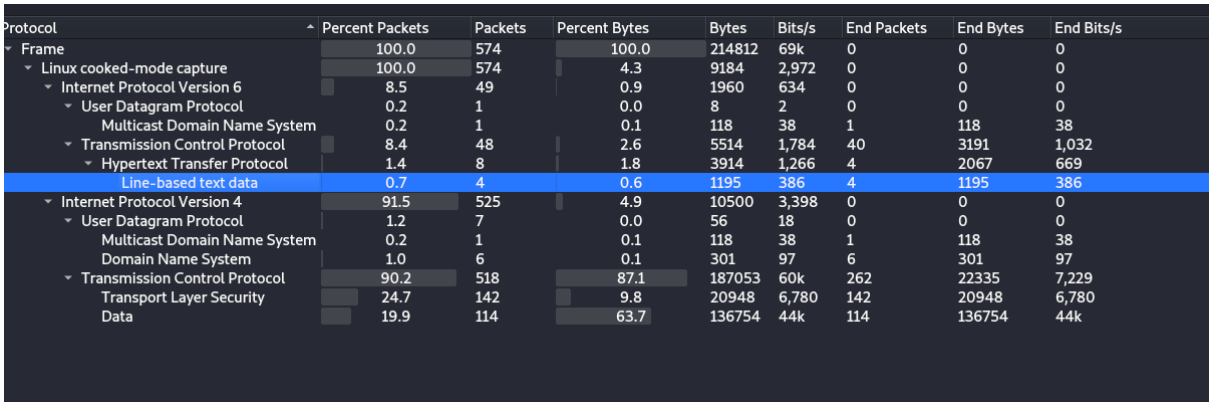
## 1. Streams

This challenge was based on wireshark. Wireshark is a tool that is used to capture and analyse network packets. The challenge had 100 points.



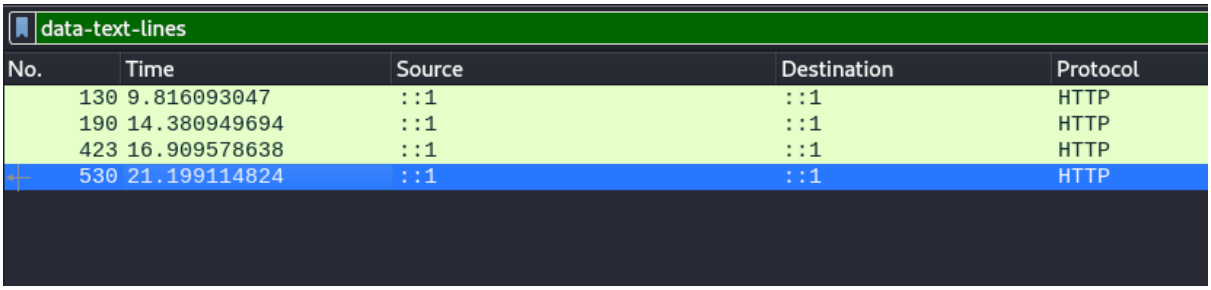

I started by downloading the file on my machine, then opened wireshark in order to analyse the file. Based on the information that we have we are supposed to find out what the hacker did on the system after installing a backdoor.

Opening file, I could see different type of protocols that were captured. On the top bar I clicked on **statistics** then selected **protocol hierarchy** to see the different protocols that are there all together in a tree like structure as shown below.

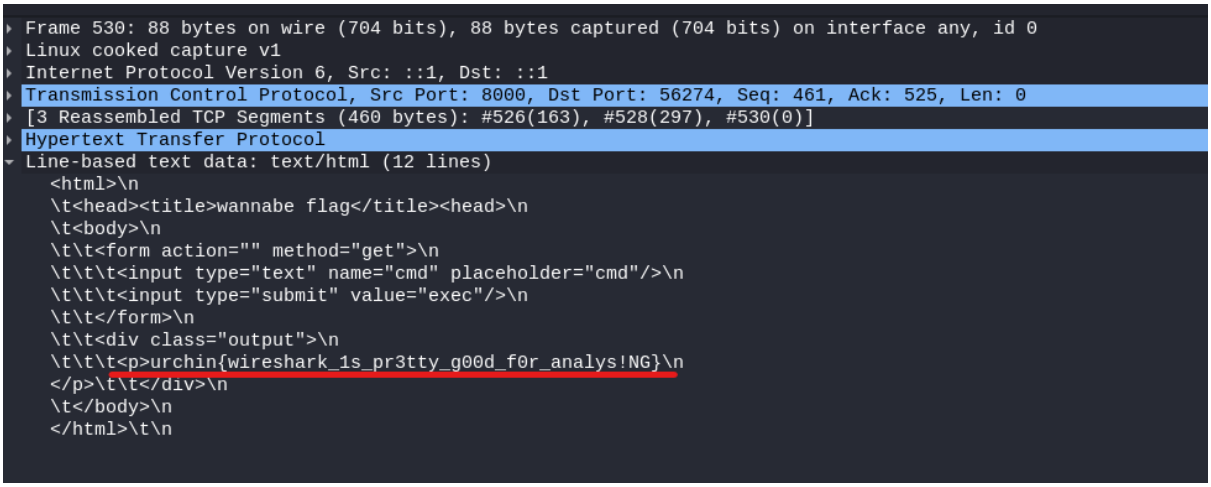


| Protocol | Percent Packets | Packets | Percent Bytes | Bytes | Bits/s | End Packets | End Bytes | End Bits/s |
|---|---|---|---|---|---|---|---|---|
| Frame | 100.0 | 574 | 100.0 | 214812 | 69k | 0 | 0 | 0 |
| Linux cooked-mode capture | 100.0 | 574 | 4.3 | 9184 | 2,972 | 0 | 0 | 0 |
| Internet Protocol Version 6 | 8.5 | 49 | 0.9 | 1960 | 634 | 0 | 0 | 0 |
| User Datagram Protocol | 0.2 | 1 | 0.0 | 8 | 2 | 0 | 0 | 0 |
| Multicast Domain Name System | 0.2 | 1 | 0.1 | 118 | 38 | 1 | 118 | 38 |
| Transmission Control Protocol | 8.4 | 48 | 2.6 | 5514 | 1,784 | 40 | 3191 | 1,032 |
| Hypertext Transfer Protocol | 1.4 | 8 | 1.8 | 3914 | 1,266 | 4 | 2067 | 669 |
| Line-based text data | 0.7 | 4 | 0.6 | 1195 | 386 | 4 | 1195 | 386 |
| Internet Protocol Version 4 | 91.5 | 525 | 4.9 | 10500 | 3,398 | 0 | 0 | 0 |
| User Datagram Protocol | 1.2 | 7 | 0.0 | 56 | 18 | 0 | 0 | 0 |
| Multicast Domain Name System | 0.2 | 1 | 0.1 | 118 | 38 | 1 | 118 | 38 |
| Domain Name System | 1.0 | 6 | 0.1 | 301 | 97 | 6 | 301 | 97 |
| Transmission Control Protocol | 90.2 | 518 | 87.1 | 187053 | 60k | 262 | 22335 | 7,229 |
| Transport Layer Security | 24.7 | 142 | 9.8 | 20948 | 6,780 | 142 | 20948 | 6,780 |
| Data | 19.9 | 114 | 63.7 | 136754 | 44k | 114 | 136754 | 44k |

After seeing all the protocols, the protocol of interest was the HTTP which its data was transmitted in text format which would be easy for us to read. I right clicked on the line based text data, then selected **apply as filter** and then clicked on **selected** and a filter was applied as shown below.

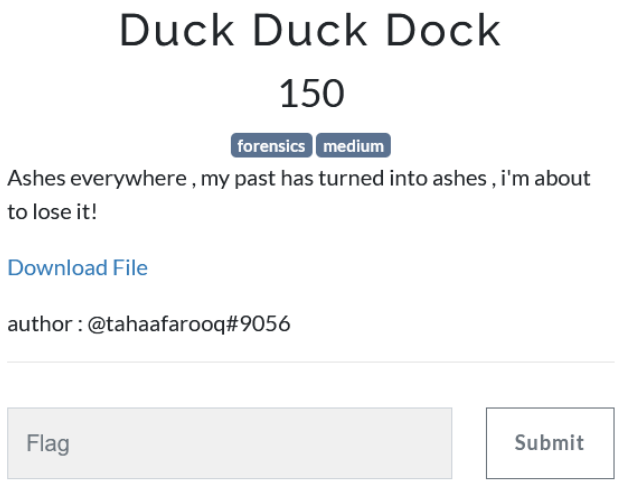



Examining the contents of the first 3 packets they had information which was not useful but on the third packet we had obtained our flag as shown below.

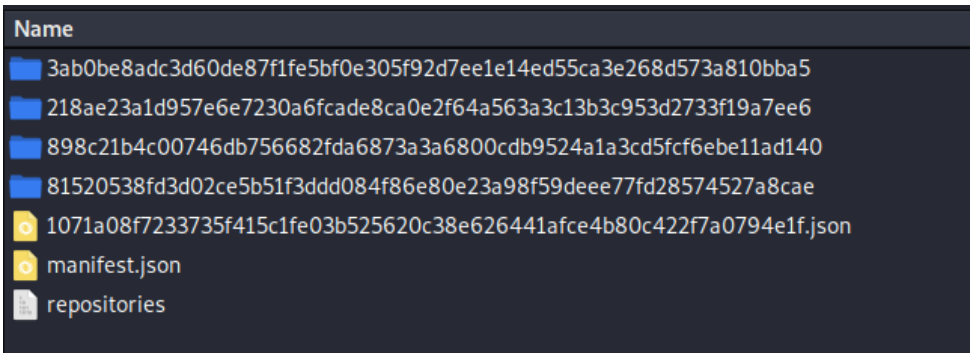


# 2. Duck duck dock

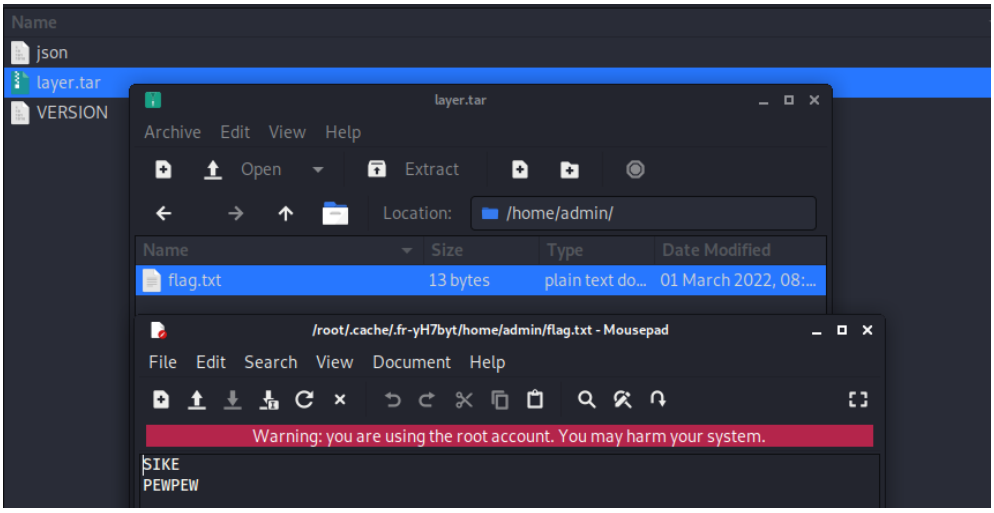
This challenge was about analysing different type of files like bash scripts and json files. This challenge had 150 points.



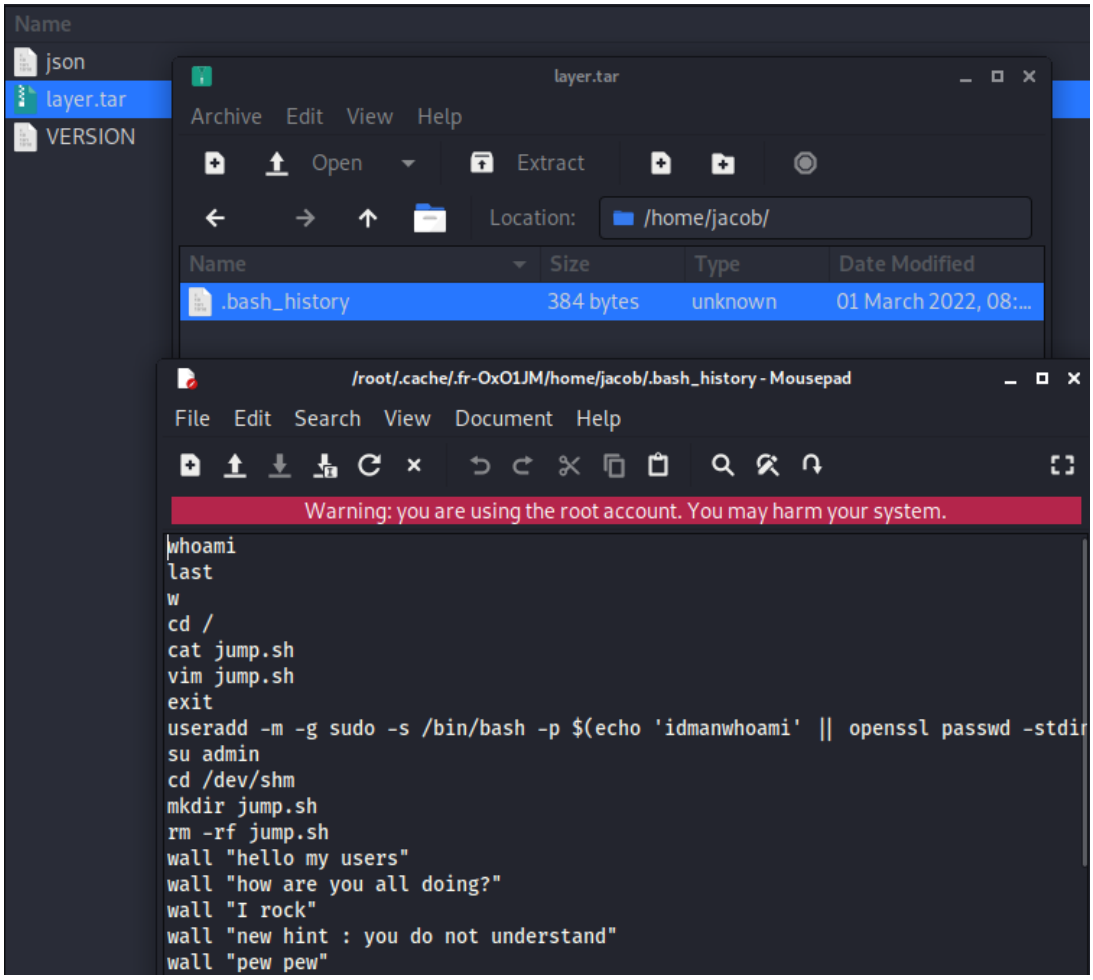

I downloaded the file, and it was zipped, I unzipped it and found different files on it as shown below



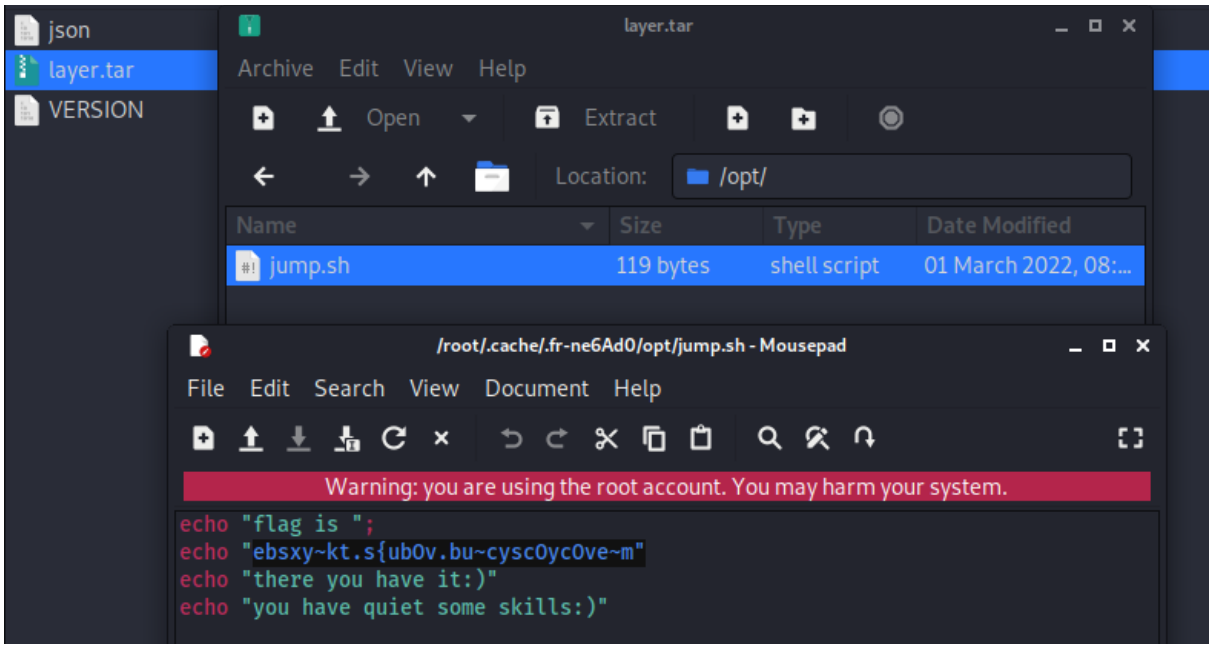

I started with the first file, and I quickly realized at the end of the file that it was a rabbit hole.



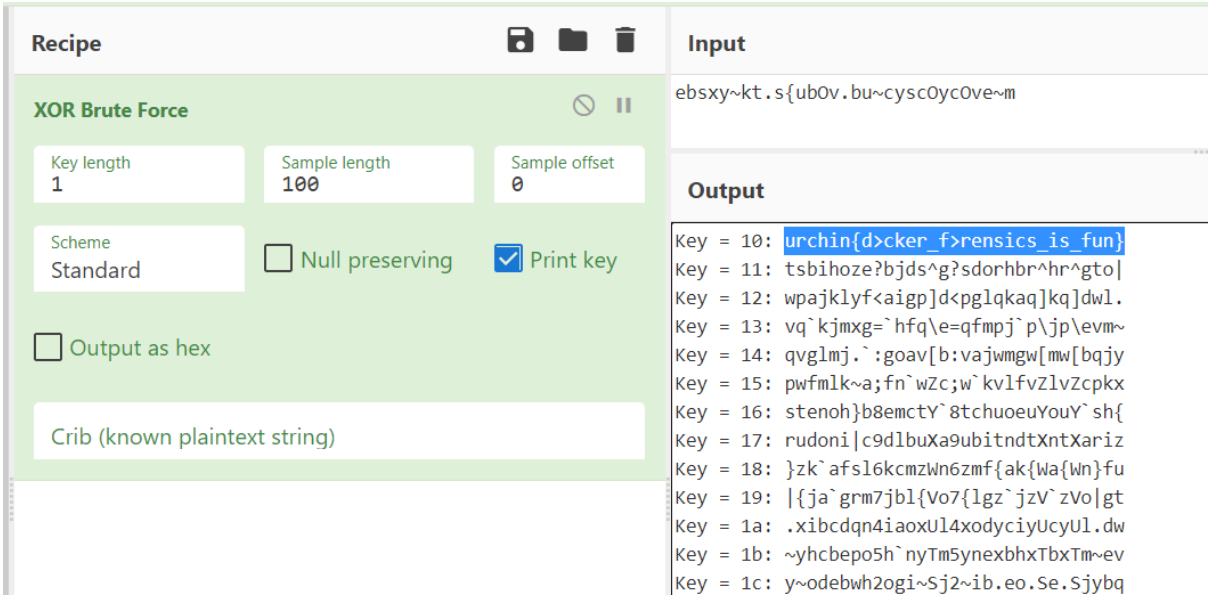
I went to the second file I found a bash script which contained all the commands that were executed previously. Which was still not interesting for me.




I went to the third file and found the flag but unfortunately it was encoded as shown below

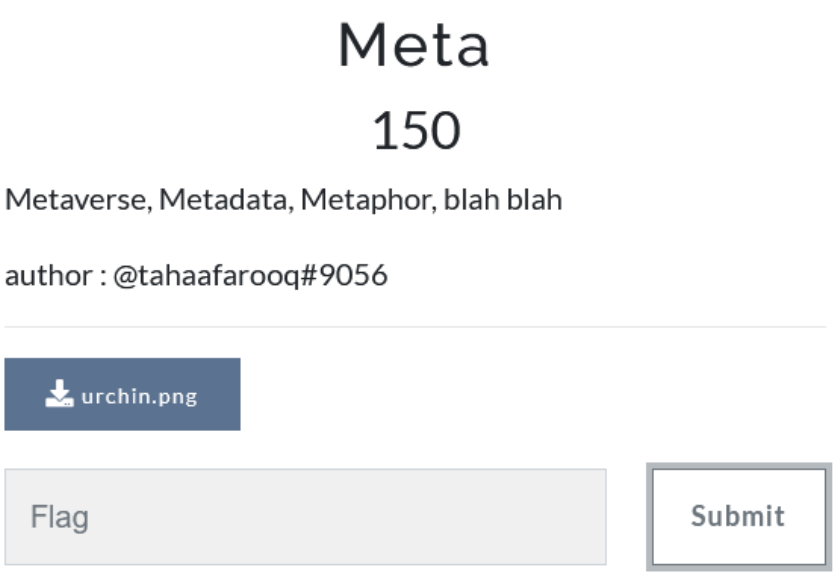


My task now was to decode the string and get the flag in plain text, I took the encoded string to cyberchef and used XOR brute force to decode the string and I found the flag.



# 3. Meta

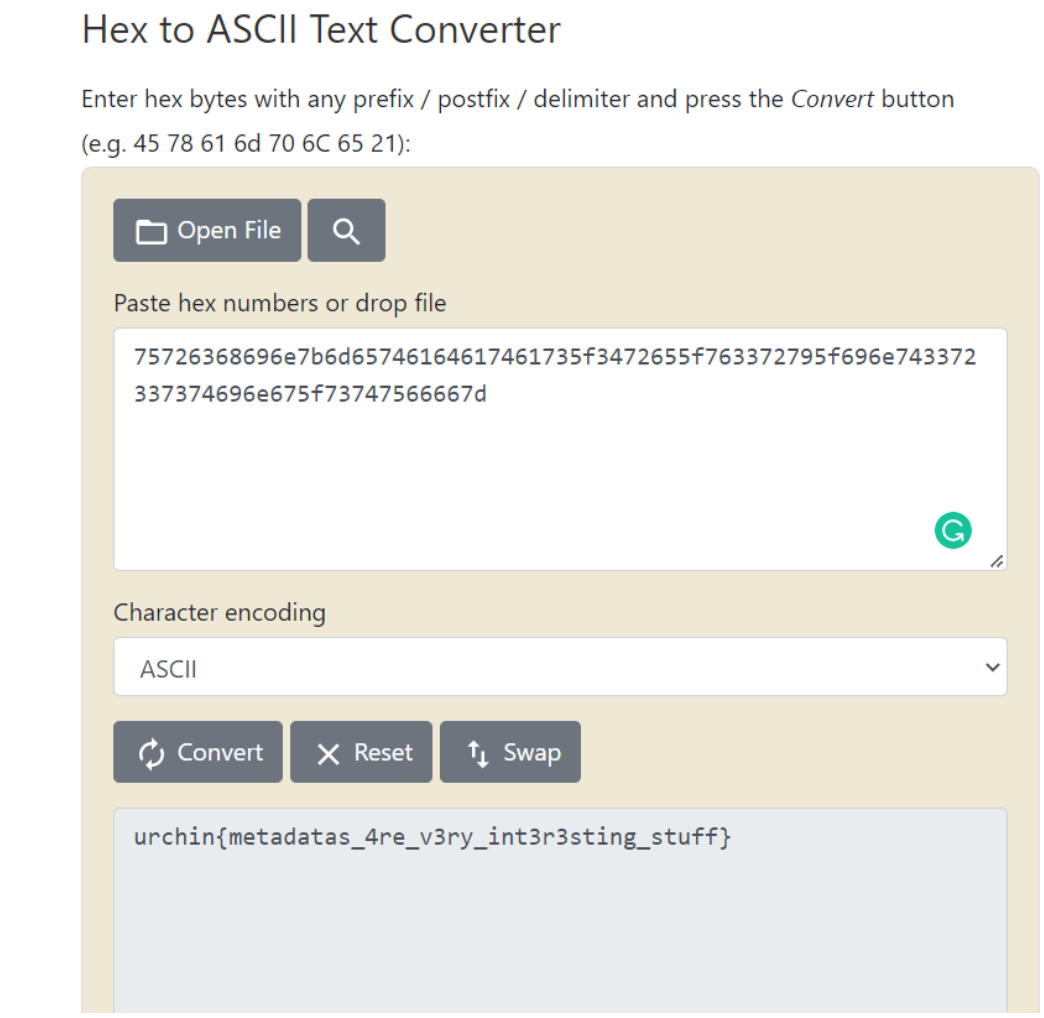
This challenge was a very straight forward challenge, just as the challenge suggests Meta, we are going to have to analyse the metadata of the given image file. This challenge has 150 points.




I downloaded the file, and it was an image of the urchin logo. Examining the metadata using a tool called exiftool we could view the metadata of this image. Going through the meta data under artist I realized there was a long string which was a HEX string.

```
└─# exiftool urchin.png
ExifTool Version Number         : 12.32
File Name                       : urchin.png
Directory                       : .
File Size                       : 43 KiB
File Modification Date/Time     : 2022:03:05 18:47:07-05:00
File Access Date/Time           : 2022:03:05 18:47:08-05:00
File Inode Change Date/Time     : 2022:03:05 18:47:07-05:00
File Permissions                : -rw-rw-rw-
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 300
Image Height                    : 300
Bit Depth                       : 8
Color Type                      : RGB with Alpha
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
Pixels Per Unit X               : 2835
Pixels Per Unit Y               : 2835
Pixel Units                     : meters
Artist                          : 75726368696e7b6d6574616461746 1735f3472655f763372795f696e743723373746 96e675f73747566667d
Image Size                      : 300x300
Megapixels                      : 0.090
```

It was time to decode the string and see what information it has and inserting it on an online decoder tool I managed to get the flag as shown below.




The other method that I could use is through the terminal with the xxd command which is used to decode the hexadecimal file and got the answer as shown below.

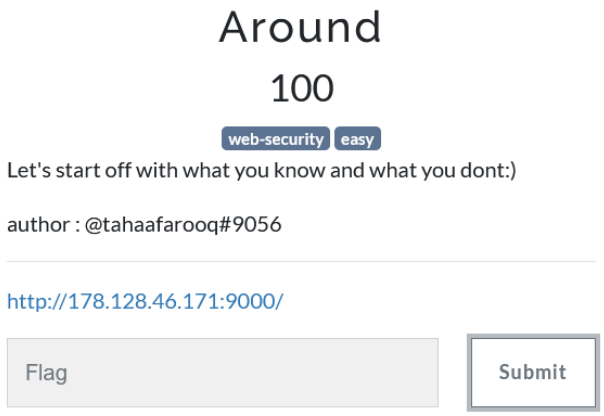
```
└─# echo "75726368696e7b6d65746164617461735f3472655f763372795f696e743372337374696e675f73747566667d" | xxd -r -p
urchin{metadatas_4re_v3ry_int3r3sting_stuff}
```

# WEB SECURITY CHALLENGES

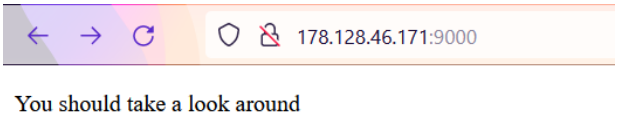
There were a total of 8 challenges, I managed to do 7/8 due to one of the challenges was not working.

## 1. Around

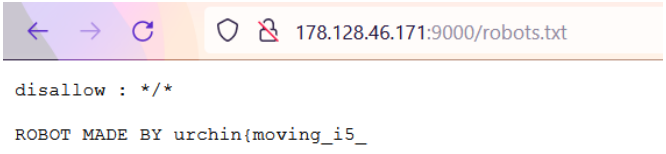
This challenge was very interesting because as the name suggested around, well you had to look around the webpage for the flag because the flag was scattered in different locations. At first, I couldn't connect the dots but I eventually managed to solve the challenge. This challenge had 100 points.




I open up the link, which was provided, and the page had a note which said you should look around.



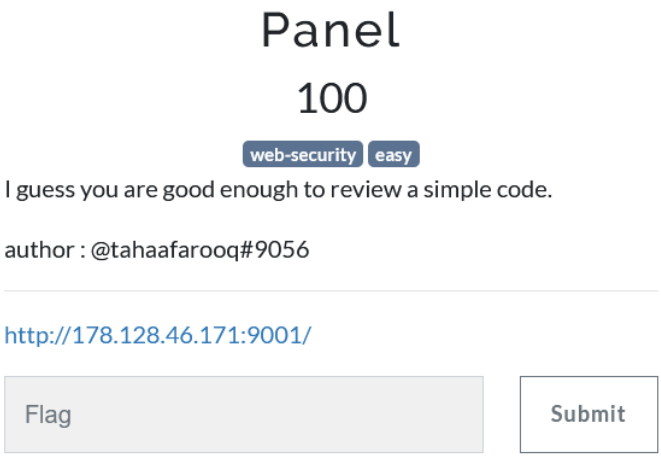

I went and checked robots.txt and found the first bit of the flag



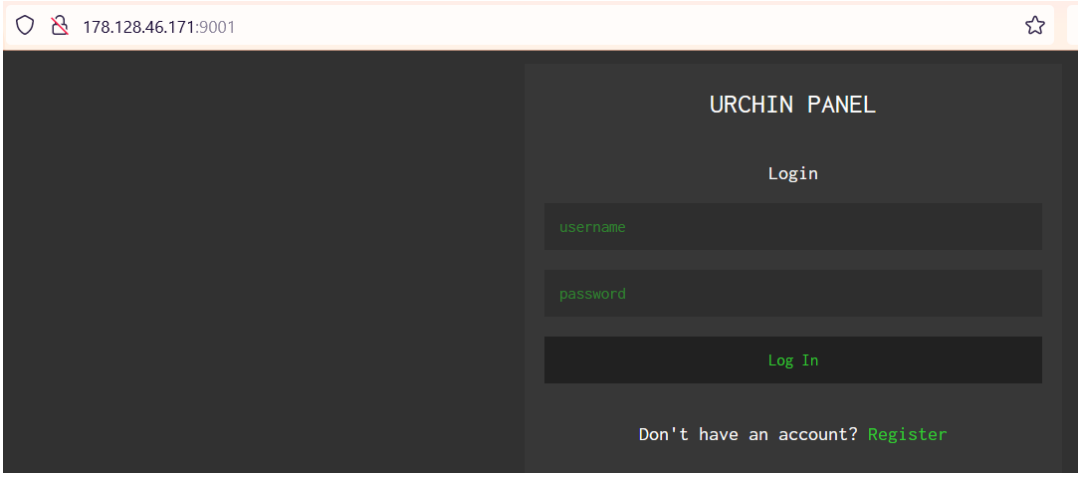

Then next I went and inspected the source code and found the second part of the flag on the JavaScript file as shown below.

# 2. Panel

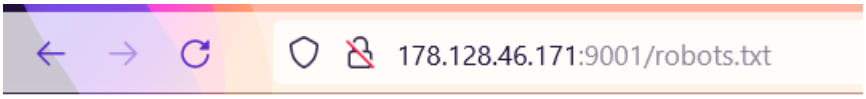
This was quite an interesting challenge, I initially struggled with it by getting into rabbit holes, but I eventually got to the solution. This challenge is 100 points.




Opening the link, we are introduced to a login page where I tried default logins but it didn't work, then I tried registering which also didn't work and went further to review the source code which was also a rabbit hole.

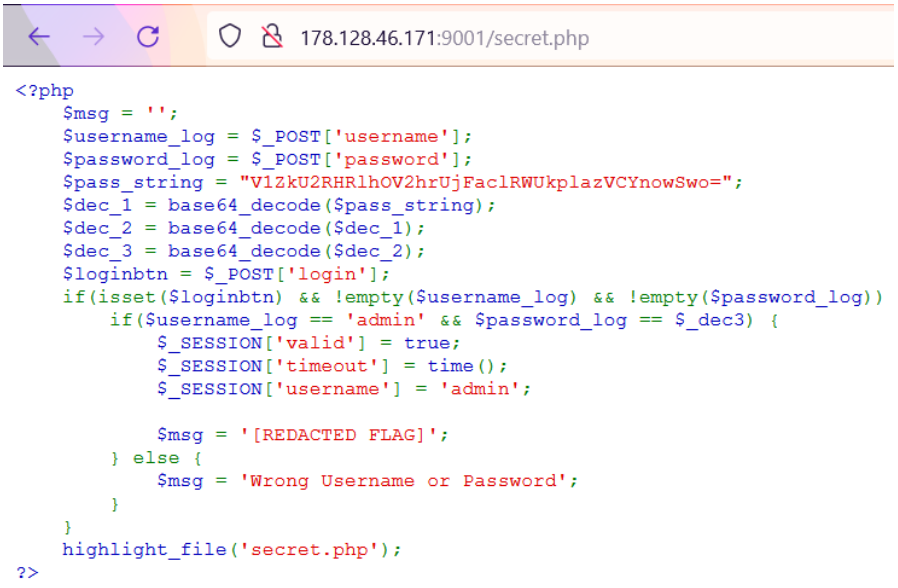



I finally decided to check robots.txt which I had found a file called secret.txt as shown below

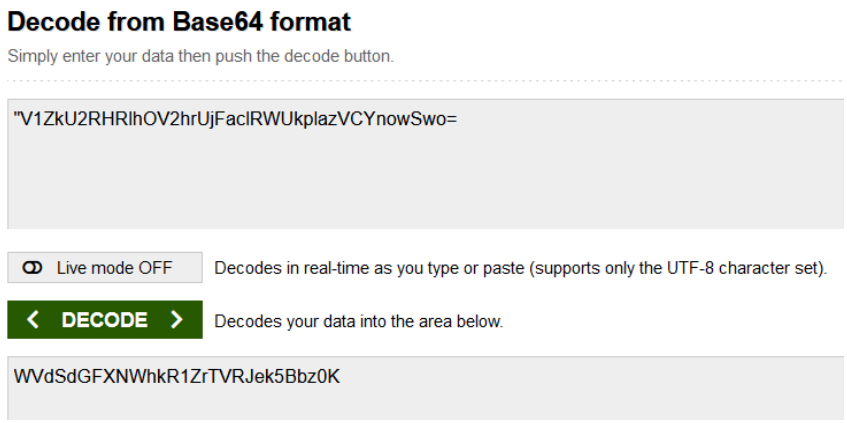



```
disallow : secret.php
disallow : */*
```

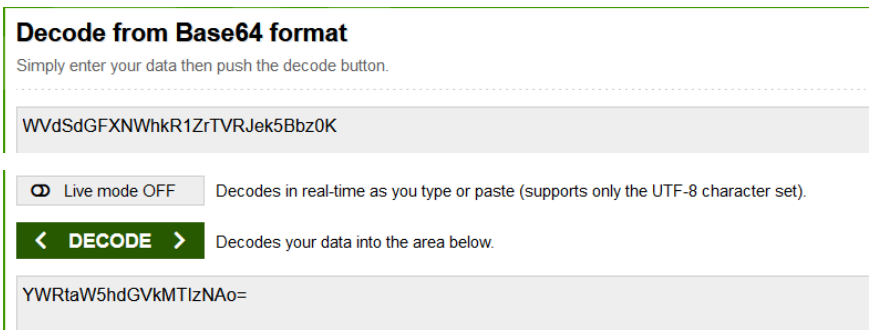
Then I opened the file and found a code which needed to be reviewed. Upon reviewing the code, I realized that the variable pass_string has the encrypted string of the password which was encrypted by base64 3 times so in order to get the password we needed to decode the password string 3 times. But also, we can see that the username is admin.



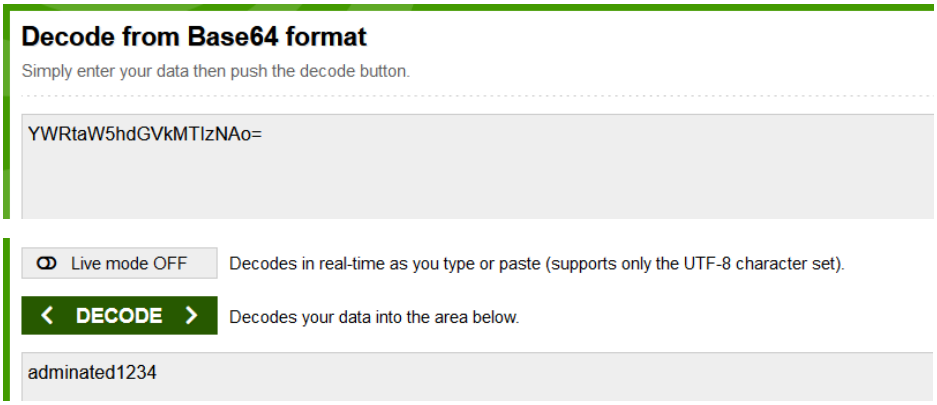

Decoding the string for the first time we get the first base64



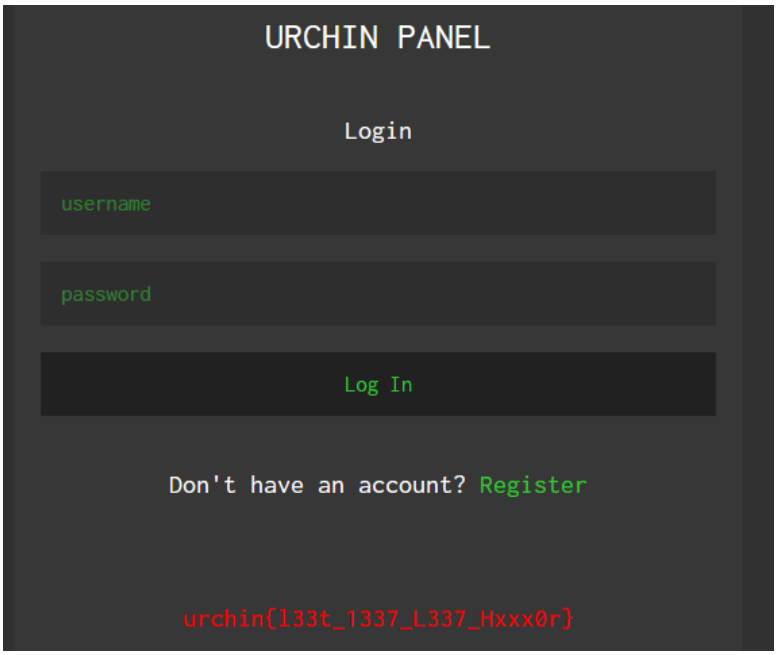

Decoding the base64 again we get another base64



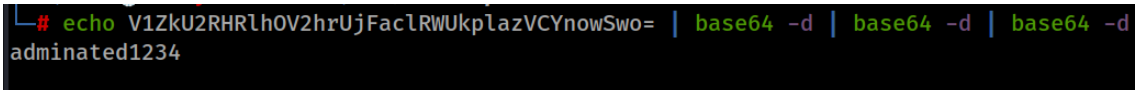
And decoding the last base64 we get the password




Using the username admin and password adminated123 we get flag as shown below




Another method of decoding the base64 is using the command line as shown below

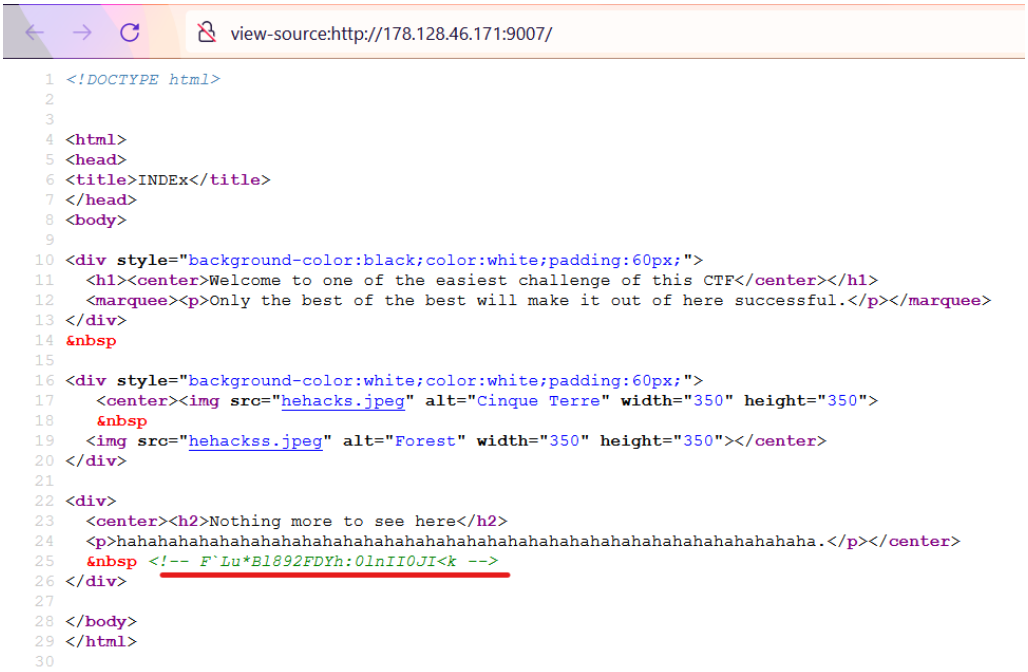


# 3. En-code

This challenge was very easy and straight forward. As the challenge name called encode there was a string which was encoded, and we had to decode it to get the flag. The challenge was 100 points.


Opening the link that we have been provided with we find a static page which doesn't have much to do.


Opening the source code, I found an encoded string as shown below

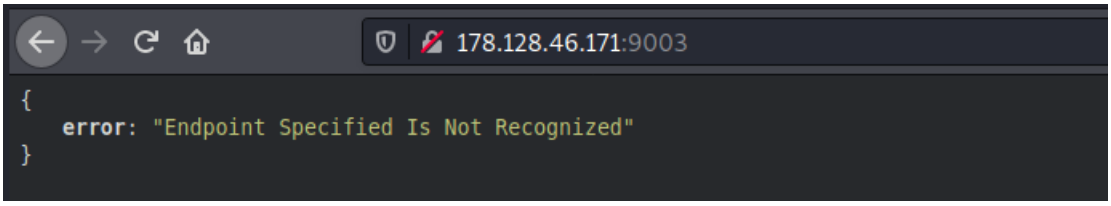



The string was a base 85 which I decoded it using an online tool as shown below and I got the flag.

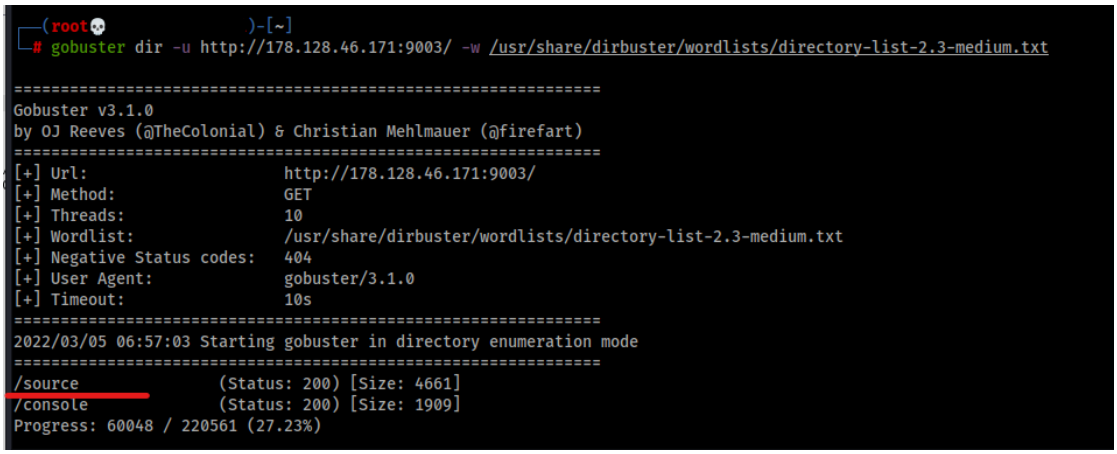
# 4. Headstart

This challenge was very interesting to me because it mainly focused on playing with the http headers in order for you to get the flag. This challenge had 150 points.


Going to the link provided we find a static age which states that it doesn't recognize the endpoint.




Then I had to directory brute force the page to get hidden directories and we found two, source and console.



Console was a rabbit hole and I moved forward to source, and reading the source I realized that for you to get the flag, the header was supposed to be PEWPEW with the directory /getflag

178.128.46.171:9003/source

```
from flask import Flask, request, jsonify, render_template, url_for
from decouple import config

import jinja2_highlight

class MyFlask(Flask):
    jinja_options = dict(Flask.jinja_options)
    jinja_options.setdefault('extensions',[]).append('jinja2_highlight.HighlightExtension')

app = MyFlask(__name__)
flag = config('FLAG')

@app.route('/')
def index():
    data = {'error':'Endpoint Specified Is Not Recognized'}
    return jsonify(data)

@app.route('/getflag', methods=["PEWPEW"])
def getflag():
    if request.method != 'PEWPEW':
        data = {'error':'something went wrong'}
        return jsonify(data)
    else:
        data = {'success':f'{flag}'}
        return jsonify(data)

@app.route('/source')
def source():
    return render_template('index.html')

if __name__ == '__main__':
    app.run()
```

Inserting the header **PEWPEW** and directory **/getflag** and we get the flag as shown below

Request

Pretty Raw Hex \n

```
1 PEWPEW /getflag HTTP/1.1
2 Content-Length: 387
3
4 Host: 178.128.46.171:9003
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
7 Accept-Language: en-US,en;q=0.5
8 Accept-Encoding: gzip, deflate
9 Connection: close
10 Cookie: PHPSESSID=c4b698e125d30e0e250e3e493c2dd438
11 Upgrade-Insecure-Requests: 1
12 Cache-Control: max-age=0
```

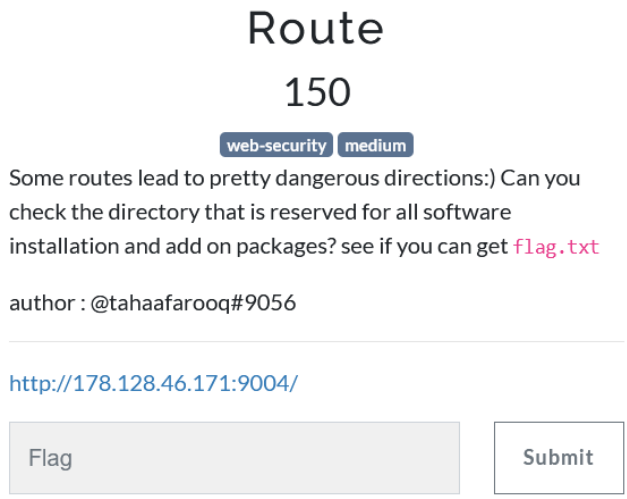
Response

Pretty Raw Hex Render \n

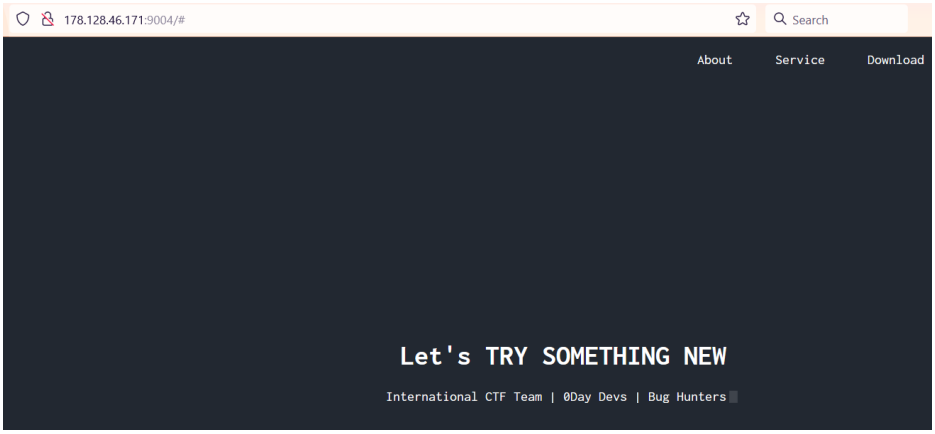
```
1 HTTP/1.0 200 OK
2 Content-Type: application/json
3 Content-Length: 51
4 Server: Werkzeug/2.0.3 Python/3.7.2
5 Date: Sat, 05 Mar 2022 12:28:22 GMT
6
7 {
8   "success":"urchin{m3th0ds_4re_F4scin@ting}"
9 }
```

# 5. Route

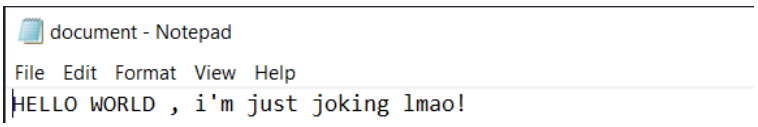
This challenge was all about local file inclusion. This challenge had 150 points.




Opening the link, we find a static web page as shown below



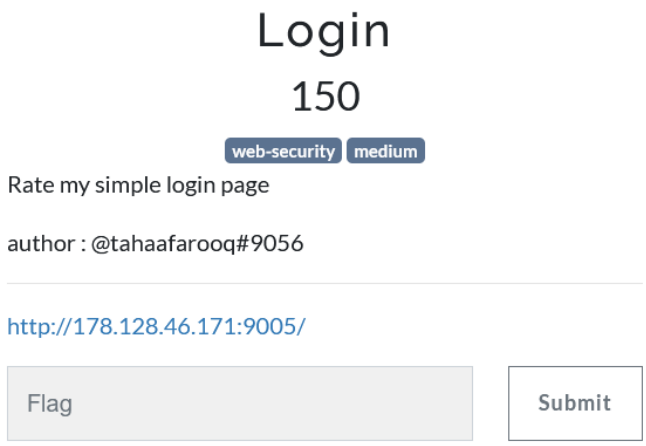

All tabs were not doing anything, but the download button downloaded a txt file which was also another rabbit hole.



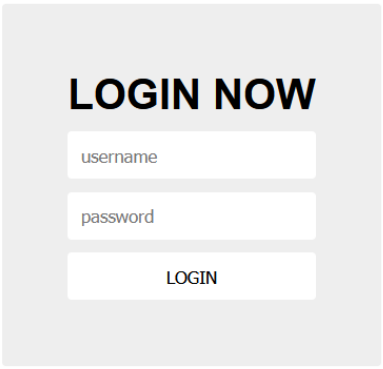

Upon downloading the file I realized that I was specifying the type of file which was being downloaded in the url as shown below **https:ip/download.php?file=xyz.txt** changing the download file to **flag.txt** and I found the flag.

# 6. Login

This was yet another interesting challenge which didn't involve any form of logging in which was actually a rabbit hole, rather It involved a lot of decoding in order to get the flag. This challenge had 150 points



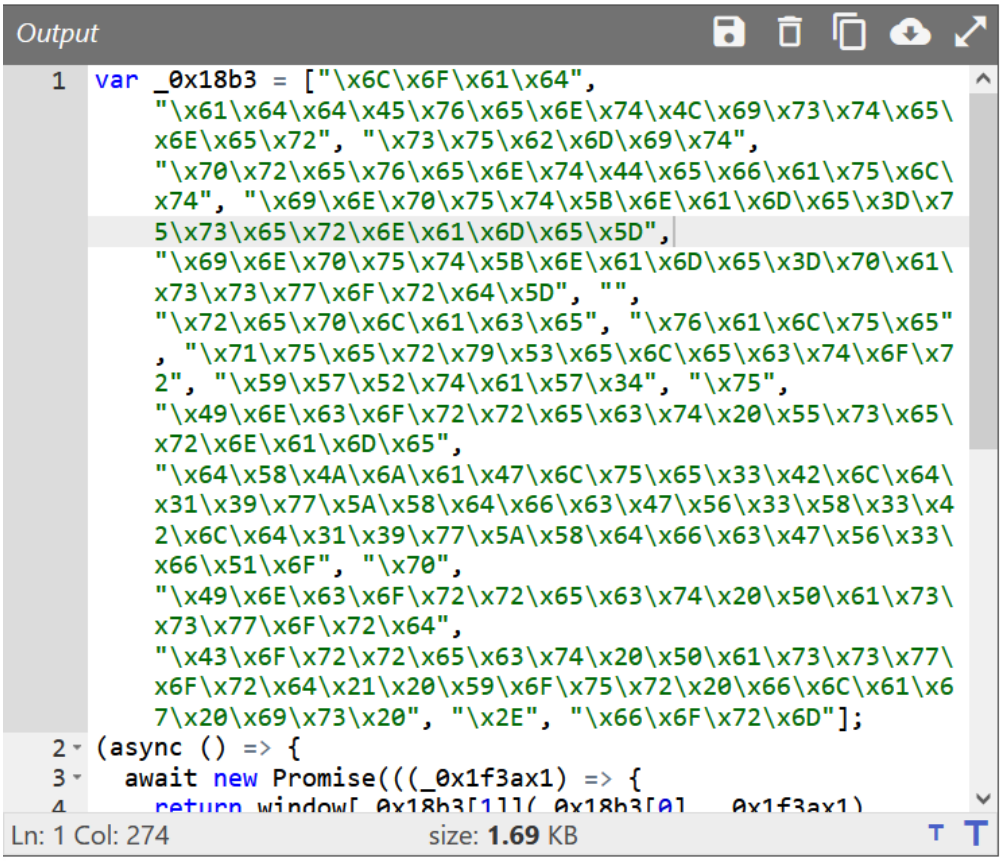

Opening the link provided and there was a login page, I attempted using default credentials It didn't work

Reviewing the source code and I found **main.js** as shown below

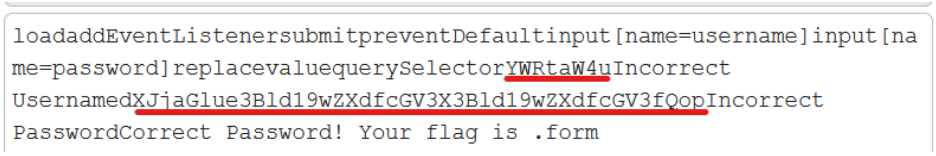

Clicking on main.js and find a long java string as shown below


Inserting the code in java beautifier and this is how the output looked like as shown below.

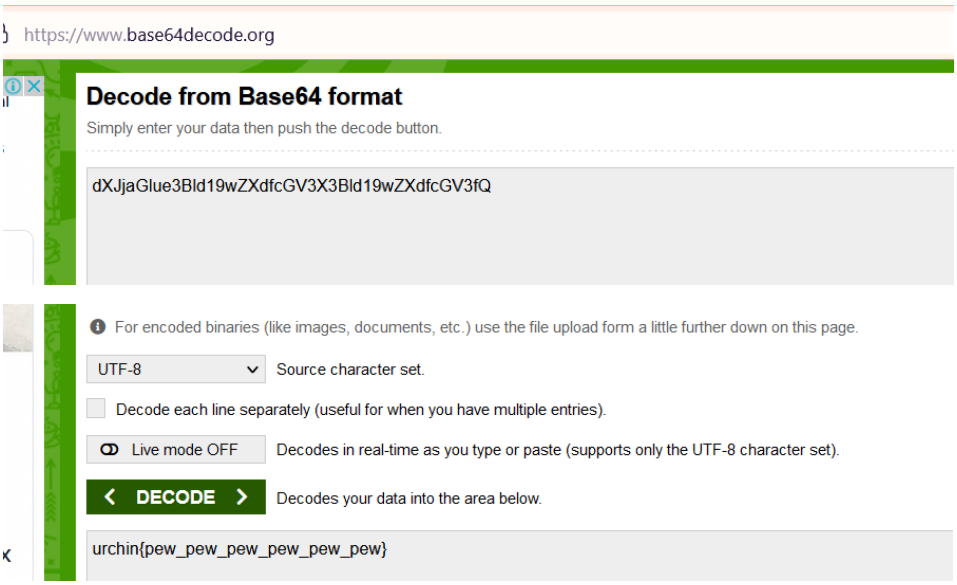


Taking the value's of var _0x18b3 and decoding it we got the below clear text.


We can see on the output there is base64 strings which are underlined as shown below.



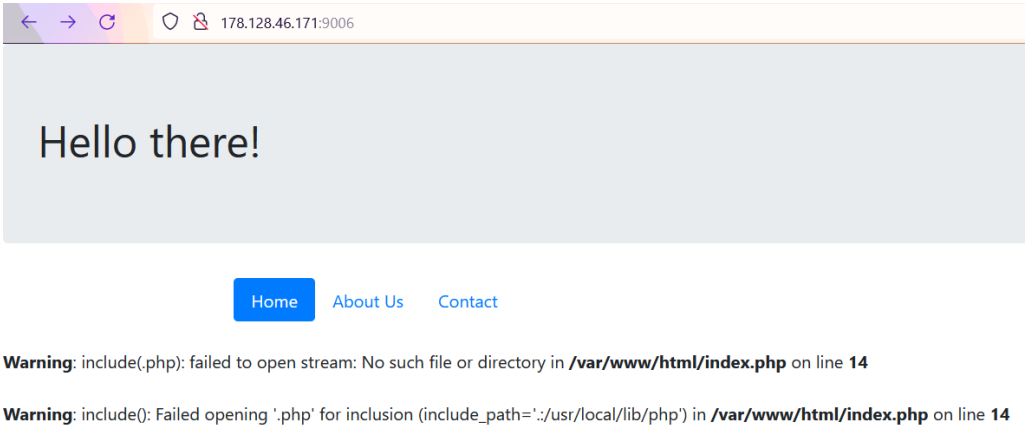

Decoding the first base64 it says admin, decoding the second base64 it displays the flag as shown below



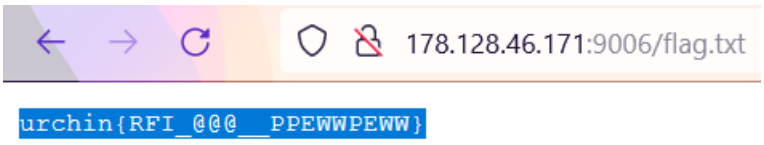
# 7. Route II

This is another challenge which was straight forward. This challenge had 150 points.


Going through the web page I found a static home page as shown below




Then did a directory brute force after not finding anything interesting on the home page, and I found a flag.txt file. Inserting the **flag.txt** on the url, I got the flag.



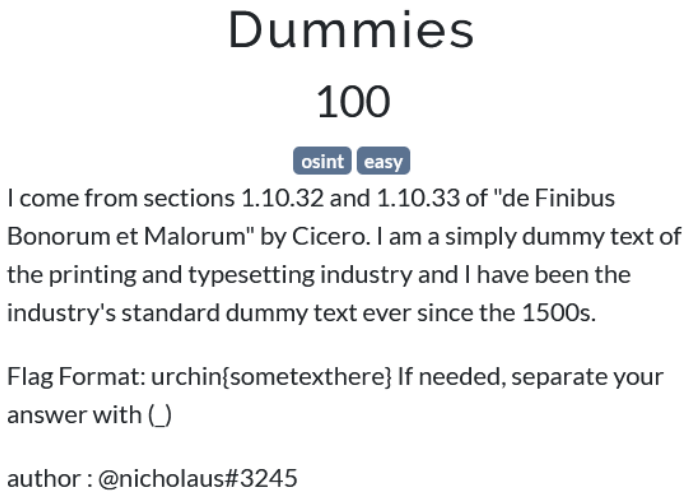

urchin{RFI_@@@__PPEWWPEWW}

# OSINT CHALLENGES

There were a total of 3 challenges in this category which were quite easy except for one which was the zipped challenge, its was not hard for me but it was hard for guys who don't live in Tanzania and will explain why in the tutorial.

## 1. Dummies

Dummies was a very interesting challenge but yet a very simple challenge. This challenge had 100 points.




Google searching the first statement and got hits which indicated the answer to be Lorem Ipsum, submitting the answer indeed it was the answer.

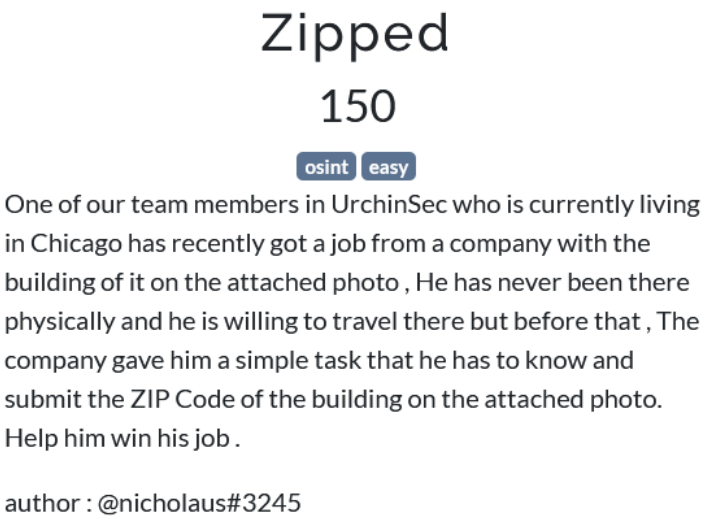
# 2. Welcome to OSINT

This challenge was also another easy straight forward challenge which carried 100 points.


As a cybersecurity expert who has been in the industry for quite some time, I didn't have to google this because the answer is known. The answer was ransomware.

# 3. Zipped

This challenge was another one of my favourite challenge though as I mentioned initially for guys who were outside Tanzania wouldn't be able to solve this hence it would have been considered hard to them. I must say this was a very creative challenge which I really enjoyed solving. The challenge had 150 points.

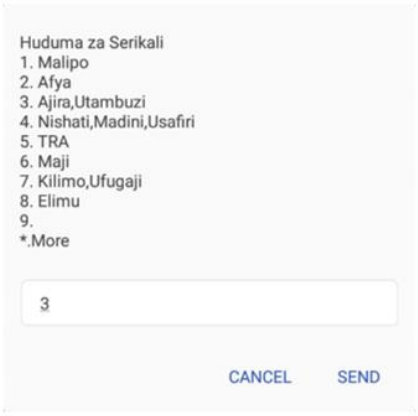


There was a file which we were supposed to download, downloading the file it was a picture of a building which is located in Tanzania, I recognized the building, and it was the building of the HQ of a telecommunication company called Airtel.

There were two methods on how I solved this challenge, the first one being going through a PDF file which I had obtained previously from my friends, going through the pdf I found the flag as shown below. the answer was 14112

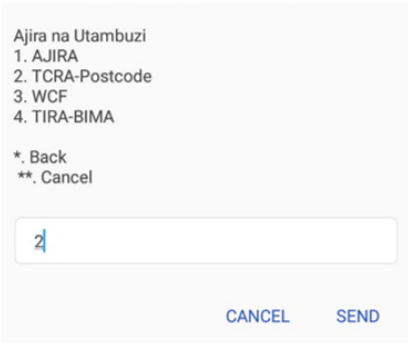
| REGION | POSTCODE | WARD | POSTCODE | MTAA/VILLAGE |
|---|---|---|---|---|
| | | | | Kisutu |
| | | | | Mkunguni |
| | | | | Mkunguni B |
| | | KINONDONI | 14110 | Kumbukumbu |
| | | | | Kinondoni Mjini |
| | | | | Kinondoni Shamba |
| | | | | Ada Estate |
| | | MSASANI | 14111 | Oysterbay |
| | | | | Masaki |
| | | | | Mikoroshoni |
| | | | | Bonde la Mpunga |
| | | | | Makangira |
| | | MIKOCHENI | 14112 | Mikocheni "A" |
| | | | | Mikocheni "B" |
| | | | | Regent Estate |
| | | | | Ally H. Mwinyi |
| | | | | T. P. D. C. |
| | | | | Darajani |

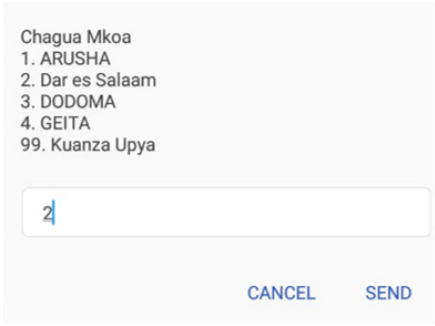
The second method was to dial the code ***152*00#**


FIRST STEP SECOND STEP

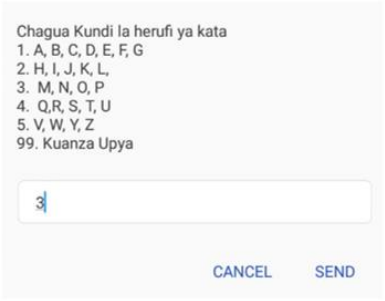
When you dial a menu will popup and then select 3 a further popup will appear




THIRD STEP FOURTH STEP

You will select 2 then select 2 again for dar es salaam since that's where the building is located





FIFTH STEP LAST STEP:

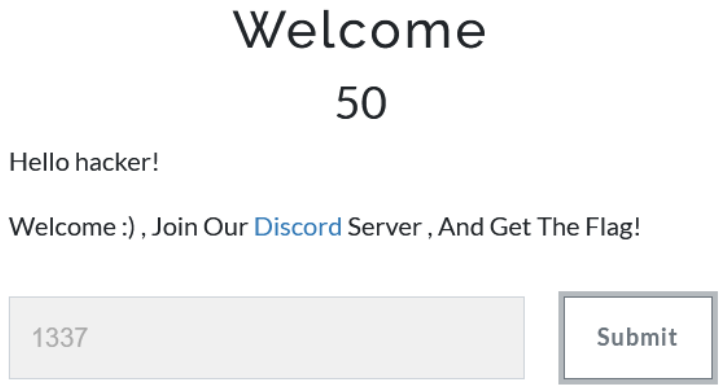
The last step you select M because the building is located in mikocheni and finally you receive the flag on your message.

# DISCORD CHALLENGES

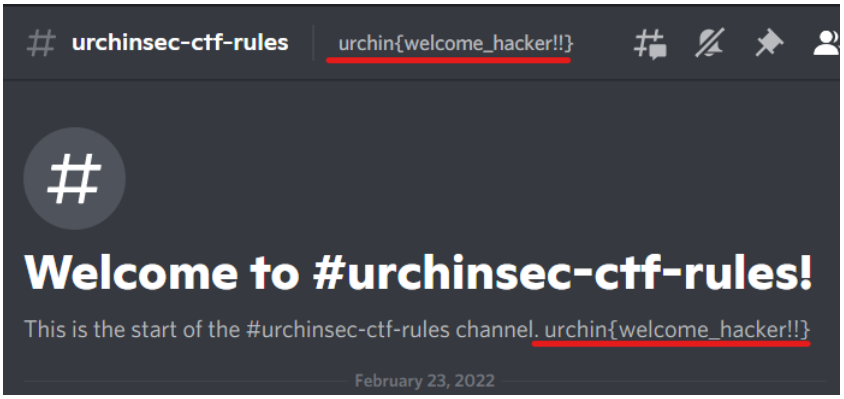
This category involved finding the flag on the discord channel which was very interesting and fun at the same time. There were a total of two challenges in this category

## 1. Welcome

This was the easiest challenge which had a total of 50 points.



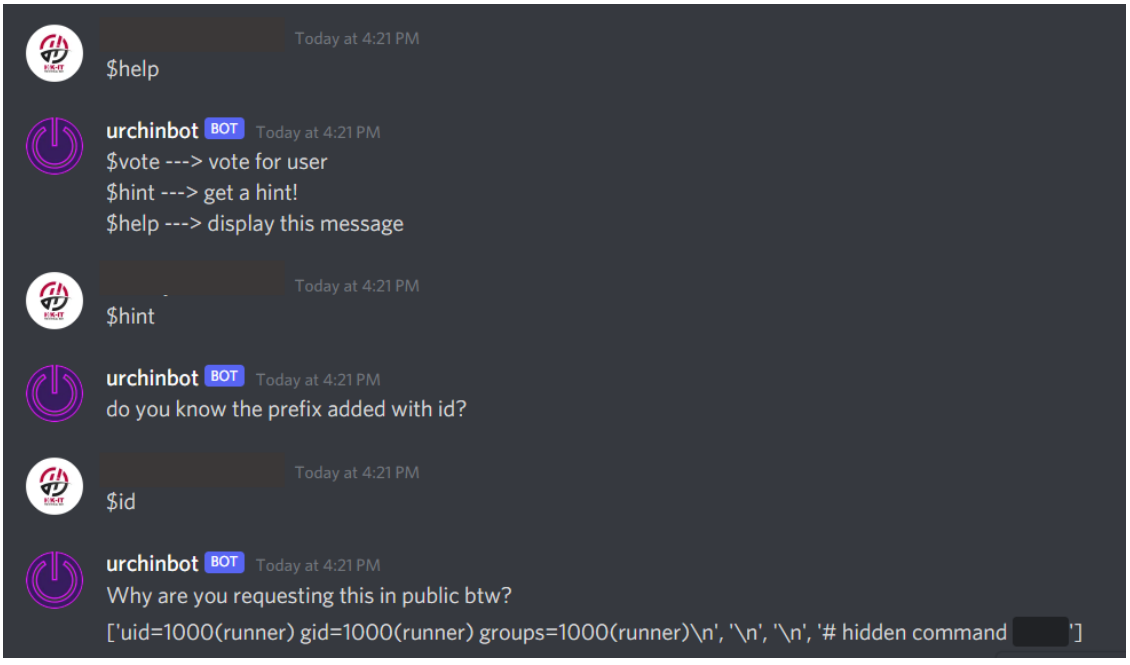

The flag could be found on ctf-rules as shown below.



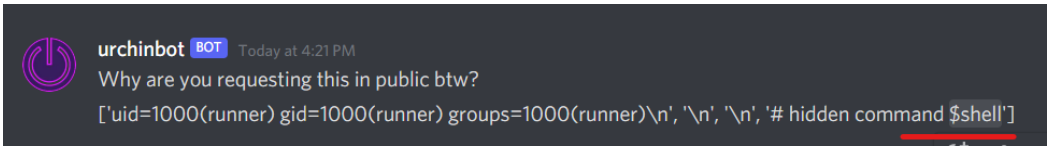
# 2. UrchinBot

This was an interesting challenge too where we had to interact with a bot in order to get the flag. The challenge had a total of 150 points.

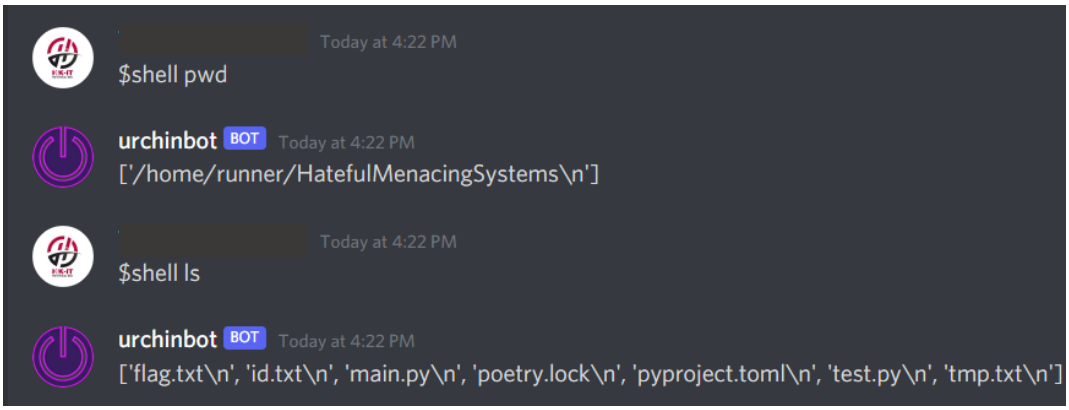

Getting into discord on the **bots-cmd** channel we had to interact with the bot and the first command I had typed was **$help** to see what options I would get. Then I got a list of commands I could run; next I typed the command **$hint** and I got a direct message from the bot. saying whether I know the prefix id? My logic was prefix should be the **$** sign so I just added **$id** as shown below, and I got more information.



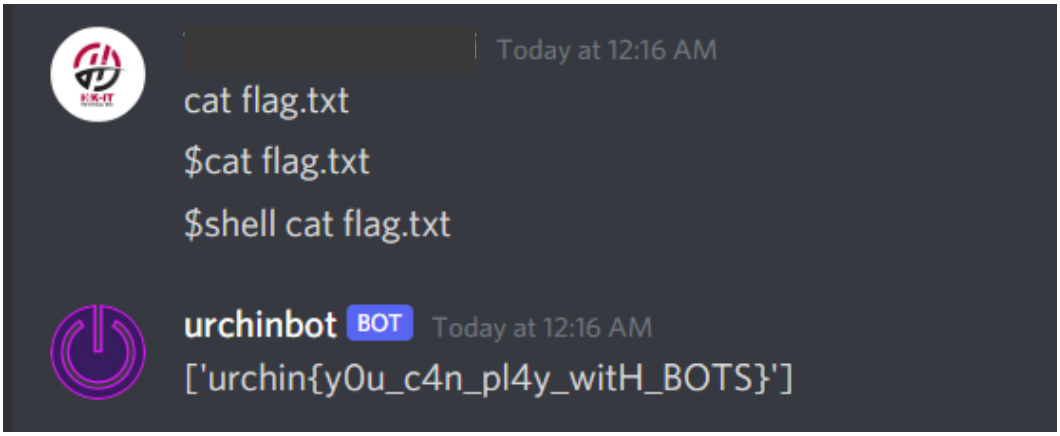
Viewing the hidden text, I saw that it was another command which was **$shell** which I had assumed would allow me to run shell commands.




I started with something small which is the print directory and I saw that it worked, next I listed the files to see what files are there as shown below




Upon seeing the flag, I decided to try and cat the file to get the contents, I did some trials and errors but eventually I managed to read the content of the file flag.txt and got the flag.



The CTF had other challenge categories of binary, reverse engineering and cryptography and two machines which I didn't do because I wanted to focus on the challenges that I did.