The Dangers of My Heart Writeups

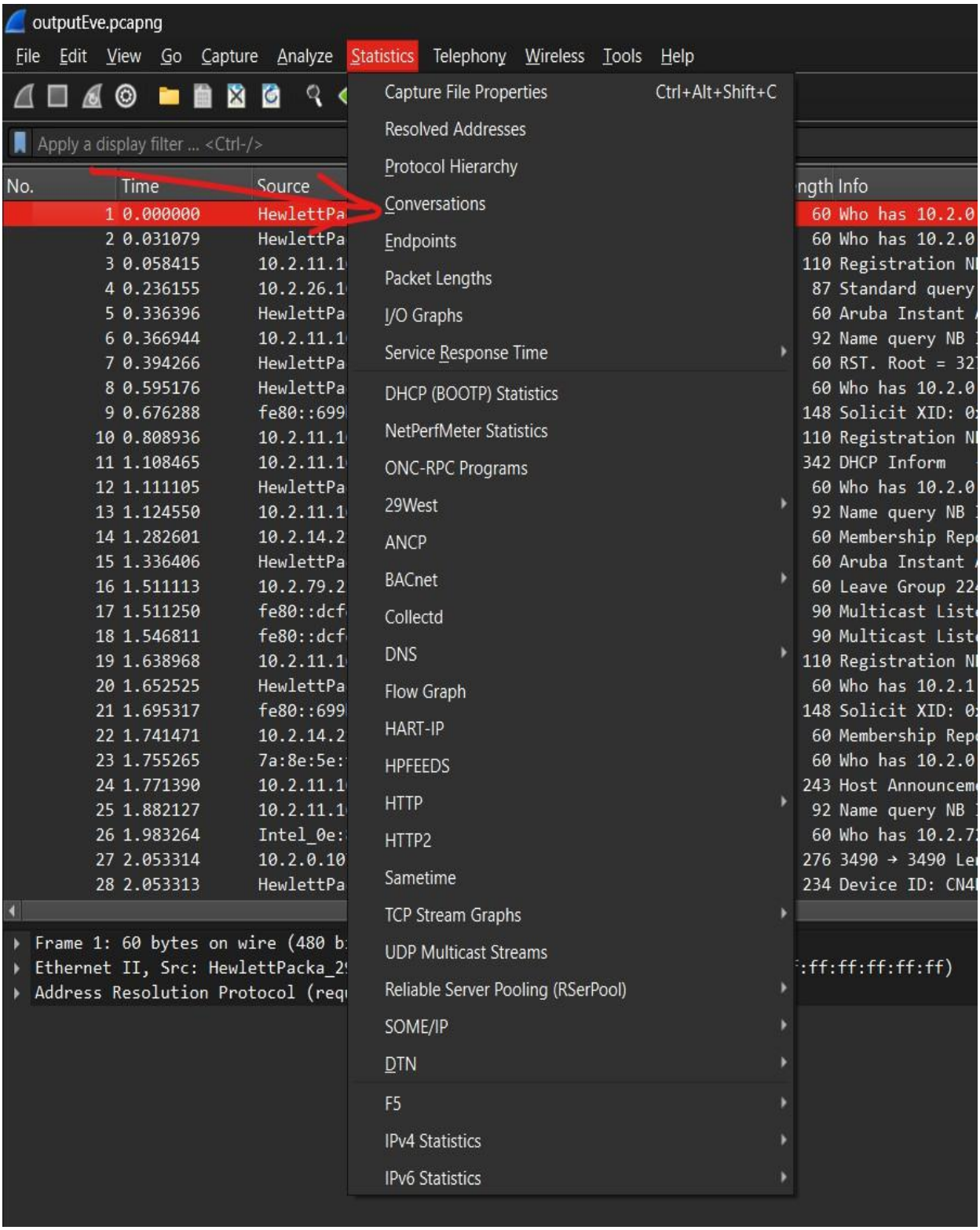
Root Access 2k25

Forensics

Woman-in-the-middle

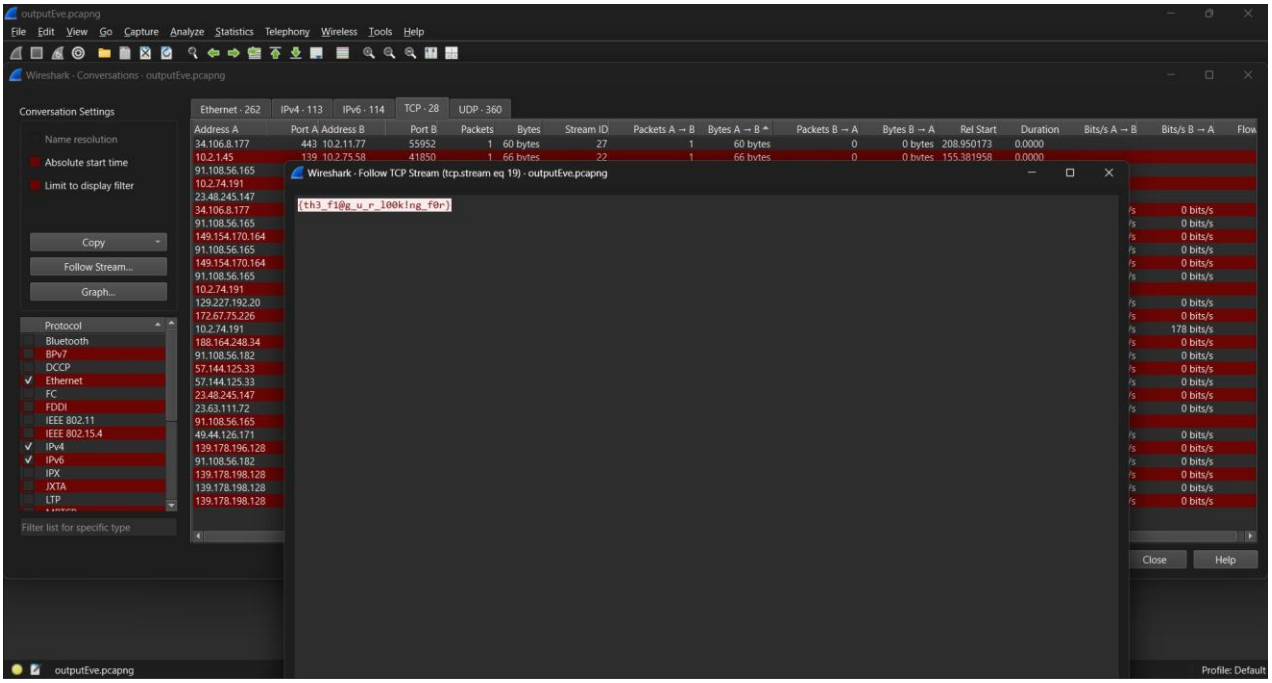
Alice and Bob are sending each other messages over an unencrypted network. Eve, wants to eavesdrop on their conversation without them knowing. To do so, she uses Wireshark to capture all the packets being sent int the network. However seeing the volume of data, she feels disheartened. Help her find the text she is looking for! (The flag format should be rootaccess{flagtext})

outputEve.pcapng



Since we are given that they are having a conversation over an insecure line just go to conversations under statistics to see all the conversations in the captured pcapng

We will manually search for the flag in these convos.



Under TCP stream 19 you will find the flag

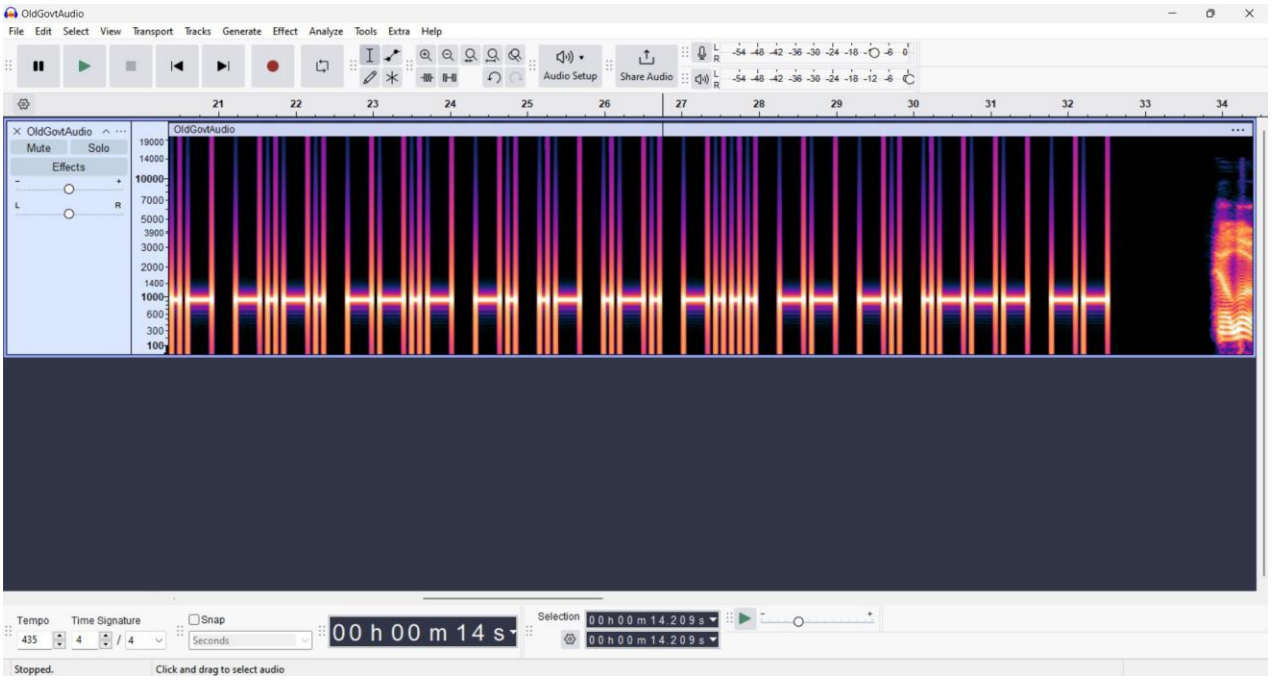
1984

In the future, where even the very thought of a crime is punishable, a masked man appears on the horizon armed with forbidden knowledge from the past. Hacking into the all-seeing government's surveillance systems, he drops a polymorphic worm to take down the whole system at once. However, before he could start the worm, he was taken out by Big Brother's drones. As his last defiance, the masked man managed to send an old government audio file to your neuralink, alongwith a code (21-21-938-9). Now it falls upon your shoulders to decode this audio file and ensure that humanity escapes its shackles. (Answer should be in lowercase letters

only and have second bracket '{}' and underscore placed in an appropriate position, echoing the format of previous answers)

OldGovtAudio.mp3

We are given a mp3 file lets open it in audacity



When we apply waveform filter, this looks like morso code at the starting lets decode using an online decoder

AXXCJLLNBBKRPKAXCQNARBMNJM

we get this text looks like caeser cipher so we go to decode

ROOTACCESSBIGBROTHERISDEAD

we get this from dcode caeser bruteforce

just format according to the question

`rootaccess{big_brother_is_dead}`

Bad PNG

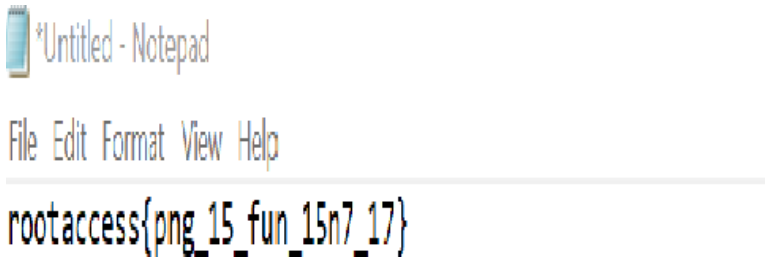
The decompressed IDAT length of 691680 bytes fits perfectly with a 480› 480 image using 8-bit truecolor (color type 2) because:

480 › (480› 3 + 1) › 480 › 1441 › 691680

This suggests that the original dimensions are likely 480› 480.

```
import struct
import zlib

def rebuild_ihdr(file_path, output_path, width=480, height=480, bit_depth=8, colo
                 compression=0, filter_method=0, interlace=0):
    with open(file_path, "rb") as f:
        data = bytearray(f.read())

    png_signature = b'\x89PNG\r\n\x1a\n'
    if data[:8] !› png_signature:
        print("Not a valid PNG file (bad signature).")
        return
    if data[12:16↑ !› b'IHDR':
        print("IHDR chunk not found at expected location.")
        return
    new_ihdr_data = struct.pack("‹ II5B", width, height, bit_depth, color_type, comp

    data[16:29↑ › new_ihdr_data
```

```
    ihdr_chunk = data[12:29↑
    new_crc = zlib.crc32(ihdr_chunk) & 0xffffffff
    data[29:33↑ › struct.pack("^ I", new_crc)

    with open(output_path, "wb") as out:
        out.write(data)
    print(f"Rebuilt IHDR with dimensions {width}x{height} (color type {color_type})

if _name__== '_main_':
    file_path = "chall.png"
    output_path = "fixed.png"
    rebuild_ihdr(file_path, output_path, width=480, height=480, bit_depth=8, color
```



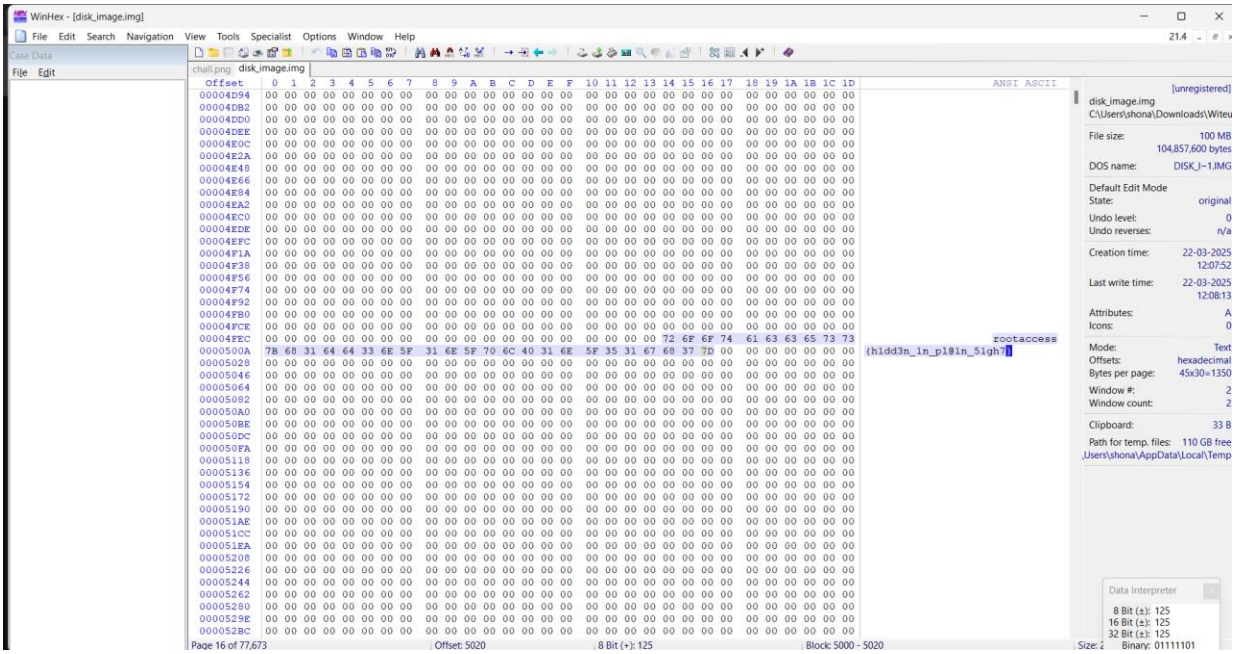
flag - `rootaccess{png_15_fun_15n7_17›`

HexDump

I pray you, DO NOT USE HEXDUMP

https://drive.google.com/file/d/1eZJbFoCNtBBlmqcsPmDTXUdsJS2bVB64/view?usp=sharing

Very simple challenge just open in any hex editor and search for "rootaccess{"



flag - rootaccess{h1dd3n_1n_pl@1n_51gh7}

Somewhere

There are so many places to hide a flag

suspicious_photo.jpg

run exiftool on the image to get the flag

```
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.02
Exif Byte Order                 : Big-endian (Motorola, MM)
X Resolution                    : 72
Y Resolution                    : 72
Resolution Unit                 : inches
Software                        : Adobe Photoshop CS6
Modify Date                     : 2025:03:22 12:00:00
Y Cb Cr Positioning             : Centered
Copyright                       : Hidden treasure ahead...
Exif Version                    : 0232
Date/Time Original              : 2025:03:22 12:00:00
Create Date                     : 2025:03:22 12:00:00
Components Configuration        : Y, Cb, Cr, -
Flashpix Version                : 0100
Color Space                     : Uncalibrated
XP Keywords                     : The flag is: rootaccess{3x1f700l_15_50_c00l}
Current IPTC Digest             : ba9c8208b536c361611477bd61813211
Keywords                        : Metadata,CTF,Stego
Application Record Version      : 4
XMP Toolkit                     : Image::ExifTool 12.40
Title                           : rootaccess{ThisIsTheFlagInAnAlternateDimension}
Profile CMM Type                : Linotronic
Profile Version                 : 2.1.0
Profile Class                   : Display Device Profile
Color Space Data                : RGB
Profile Connection Space        : XYZ
Profile Date Time               : 1998:02:09 06:49:00
```

flag - `rootaccess{3› 1f700l_15_50_c00l}`

Archive

Who in their right mind does this? I mean whyyyyyy

chall.zip

```
import zipfile
import os
def unzip_nested_zip(filename):
    current_file = filename
    count › 0
    while current_file.endswith('.zip'):
        print(f"Extracting level {count+1› : {current_file}")
        with zipfile.ZipFile(current_file, 'r') as zip_ref:
            zip_ref.extractall()
            file_list = zip_ref.namelist()
            if len(file_list) !› 1:
                print("Unexpected number of files found in:", current_file)
                break
```

```
            current_file = file_list[0]
        count += 1
    return current_file
if __name__ == '__main__':
    final_file = unzip_nested_zip("chall.zip")
    print("\nFinal file reached:", final_file)
```

run this code to extract all the 1000 zip files

in the last file you find a txt file starting with

‰PNG

which tells us its supposed to be a PNG file

just change the extension of the file to png and run zsteg on the png file to get the flag

```
└─$ zsteg chall.png
b1,g,lsb,xy         .. file: OpenPGP Public Key
b1,g,msb,xy         .. file: OpenPGP Public Key
b1,rgb,lsb,xy       .. text: "rootaccess{7h@7_w@5_n07_50_c0mpl3x}"
b1,abgr,msb,xy      .. file: PGP Secret Sub-key -
b2,r,msb,xy         .. text: "TUUUUUUUU"
b2,b,msb,xy         .. text: ["U" repeated 12 times]
b2,rgba,lsb,xy      .. text: "#'g3gwggs"
b2,abgr,msb,xy      .. text: "GGGGGGGGGGCCCCGGGGGGGGCCCC"
b3,b,msb,xy         .. file: OpenPGP Secret Key
b3,rgba,lsb,xy      .. text: "/BvgBv/btoF"
b4,r,lsb,xy         .. text: "333###2\"3#0"
b4,r,msb,xy         .. text: ["D" repeated 13 times]
b4,g,lsb,xy         .. file: 0420 Alliant virtual executable not stripped
b4,b,lsb,xy         .. text: "2\"35UDEER\"##"
b4,b,msb,xy         .. text: "DDDDDDDDDDDDDDDDDDDDDDDD\"\"\"\"\"\"DDDDDDDDDD"
b4,rgb,lsb,xy       .. text: " D$BU%CE%SU%RU4 S"
b4,bgr,lsb,xy       .. text: "@$$RE%CE%SU%RT5P$"
b4,rgba,lsb,xy      .. text: "?0/0/1/1?0?0?1_!_0_ O1O O0_0O!_!_ /1/1/!/0?1/"
b4,abgr,msb,xy      .. text: "O@O@O@O@O@O@O@O@O@"
```

flag- `rootaccess{7h@7_w@5_n07_50_c0mpl3x}`

GIF

An ez challenge. A collection of frames

chall.gif

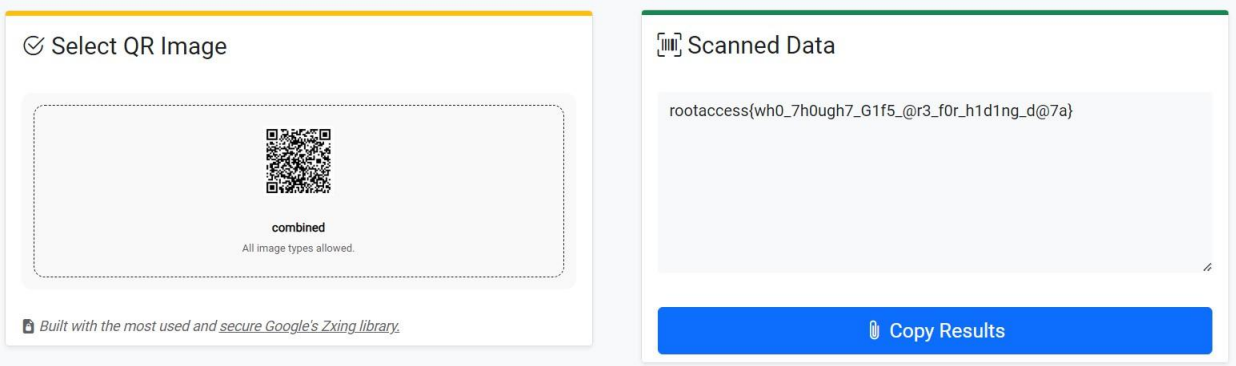
Extract frames from the GIF using this online site - https://ezgif.com/split

now we notice that there are 0-99 named files that look like QR code pieces that are split up into 100 different images
we can make an educated guess that the QR code would be a 10› 10 grid of these images combined so the python code for it-

```
import os
import re
from PIL import Image

def natural_sort_key(s):
    return [int(text) if text.isdigit() else text.lower() for text in re.split(r'(\d+)', s)]
extensions = ('.gif')
files = [f for f in os.listdir('.') if f.lower().endswith(extensions)]
files.sort(key=natural_sort_key)
total = len(files)
rows › 10
cols › 10
if total !› rows * cols:
    print(f"Warning: Expected {rows * cols} images for a 10› 10 grid, but found {t
otal}.")
with Image.open(files[0]) as img:
    width, height = img.size
combined › Image.new('RGB', (cols * width, rows * height), color=(255, 255, 2
55))
for r in range(rows):
    for c in range(cols):
        index = r + c * rows
        if index < total:
            with Image.open(files[index]) as img:
```

```
            x = c * width
            y = r * height
            combined.paste(img, (x, y))
combined.save('combined.jpg')
print("Combined image saved as 'combined.jpg'")
```

running this we get combined.jpg-



Using an online QR code reader to parse this we get the flag

`rootaccess{wh0_7h0ugh7_G1f5_@r3_f0r_h1d1ng_d@7a}`

Scrape2Win

Only god knows the haunted things that hide in there. Be smart ;)

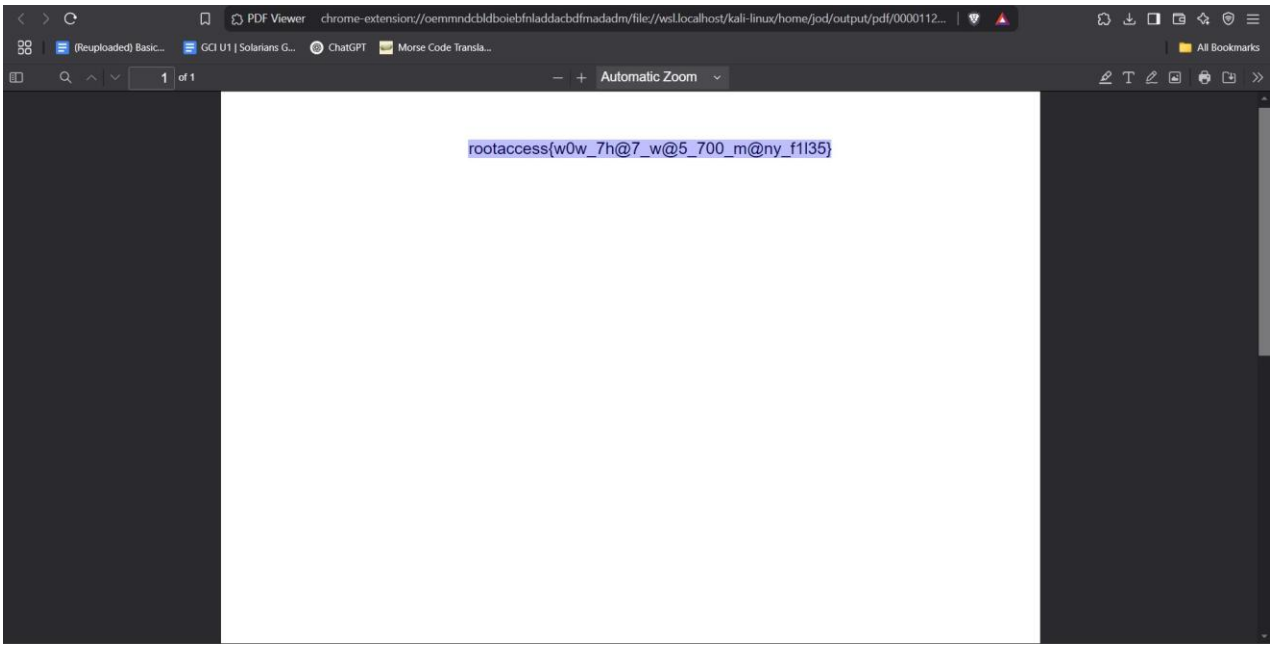
On a second note, what is your favourite file format?

memory_dump.bin

use foremost to carve out files from the bin file

we get alot of jpgs gifs pngs and pdfs

manually searching thru all of these we get a flag in a pdf that looks genuine (there were alot of fake flags in other pdfs)



so flag- rootaccess{w0w_7h@7_w@5_700_m@ny_f1l35›

OSINT

Do you have the Root Access?

Well you know, if you want to learn more about something, you gotta go to Wikipedia and put the necessary keywords there.

go to wikipedia and type root access

get redirected to the page superuser

check the edit history for the page

https://en.wikipedia.org/w/index.php?title=Superuser&diff=prev&oldid=1280387337

find this edit that has the flag -

`rootaccess{I_@m_t00r}`

Gimme! Gimme! Gimme! the name

Does it ever happen to you when you keep humming a line of the song you listened in loop but can't seem to remember the name of the song? Well it happened to our friend Tusi, and a lot of times. Can you help her in finding the name of the song of her insta note? Mail her if you found it, pretty please??

writeup:-

go to the instagram of the revelations (event oragnizers)

go to the instagram post for the root access

find tusi_here's comment there

go to her bio on instagram

go the the mihoyolab link (**www.hoyolab.com/accountCenter/postList?id=434176088)**

on the mihoyopage check her bio it has the starting letters of the email

tusishaw1234@.....

from here just guess @gmail.com should be the mail domain

follow her on insta and she/he will add you back to see their notes

on the notes it says Trynna strike a cord and its porbably A minorr

just mail her "not like us" to get the base64 flag

cm9vdGFjY2Vzc3t0aDFzXzEkXyQwYzFhbF8zbmcxbjMzcjFuZ30>

base64 decode to get

```
rootaccess{th1s_1$_$0c1al_3ng1n33r1ng}
```

Crypto

Letter to the Successor

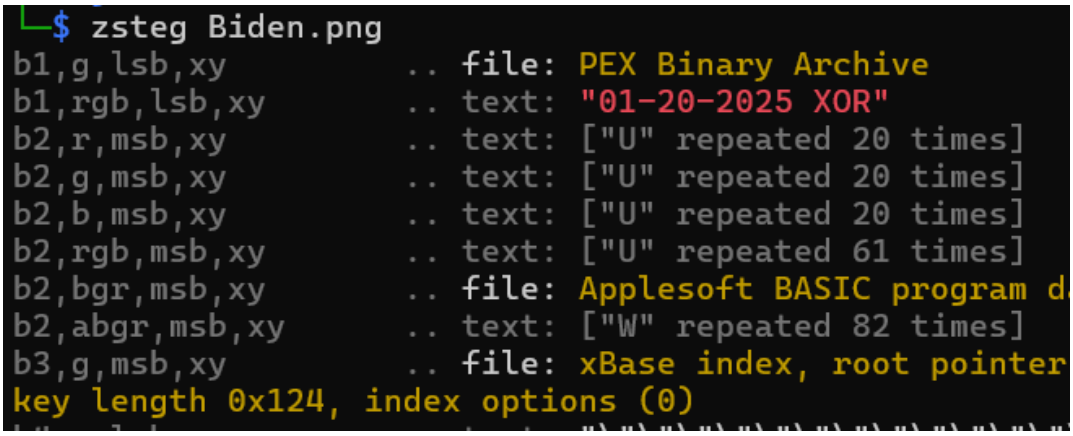
On the day of his inaugration, Donald Trump found a sealed envelope in the Oval Office's desk. It was from his predecessor, Joe Biden, and was addressed to his "Successor". Curiously, Trump found a picture of Biden in the envelope alongwith an encrypted pendrive. Trump could figure out that the password for the pendrive was hidden in the photograph. Can you help Trump find the contents of the pendrive?

Biden.png

LetterfortheSuccessor.txt

We are given a png and some hex values

using zsteg on the png we get



so we are given a xor key....

it took some time to figure out we are to consider each block as a byte `(0x01, 0x20, 0x20, 0x25)` and xor it with our bytes in the txt files

this can be achieved with this simple code

```
ciphertext = bytes.fromhex("73 4F 4F 51 60 43 43 40 72 53 5B 51 69 13 7F 52 31
key = bytes([0x01, 0x20, 0x20, 0x25↑ )
plaintext = bytes(c ^ key[i % len(key)] for i, c in enumerate(ciphertext))
print(plaintext.decode('utf-8'))
```

running the script gives us

`rootaccess{th3_w0r1d_@t_my_f!ngert!p$›`

Small E

The hint is in the name ;)

rsa.txt

Simple RSA problem you could even prompt ChatGPT to make a solve script hint is given in the challenge name itself xd

Solve script:-

```
import gmpy2
from Cryptodome.Util.number import long_to_bytes

n_list = [
    71250205697452384723663185020485417854186876388384105112435257 1
    69089630266977818337007733619002493192327803290425271865693438
    50232457373931596648699303439294901557133496948327487809053611
    43721457463200634385272903446641092597741494841091895539188681 2
    46849868514565274579140533464287234188749055895965643368011708
]

c_val = 79358452825093991543887758573359166231045942045567331952 59

c_list = [c_val] * len(n_list)

M = 1
for n in n_list:
    M *= n

result = 0
for n, c in zip(n_list, c_list):
    m_i = M // n
    inv = int(gmpy2.invert(m_i, n))
    result += c * inv * m_i
```

```
result %> M

m, exact = gmpy2.iroot(result, 5)

plaintext = long_to_bytes(int(m))
print("flag:", plaintext.decode())
```

Stuck on repeat:

esjxriqrfs{ewp_kmh_lcrx_vjpckr}

we are given this in the txt file looks like it can be caeser or vigenere

since we know that esjxriqrfs → rootaccess

we can simply make a vigenere mapping ourselves which turns out to be

NEVERGONNA

no we all know where this is going

just try each word at a time for the key in
dcode.fr vigenere decoder and you will end up with the key

NEVERGONNAGIVEYOUUP

and the flag - `rootaccess{you_got_rick_rolled}`

Turing Test

Be my Alan Turing please

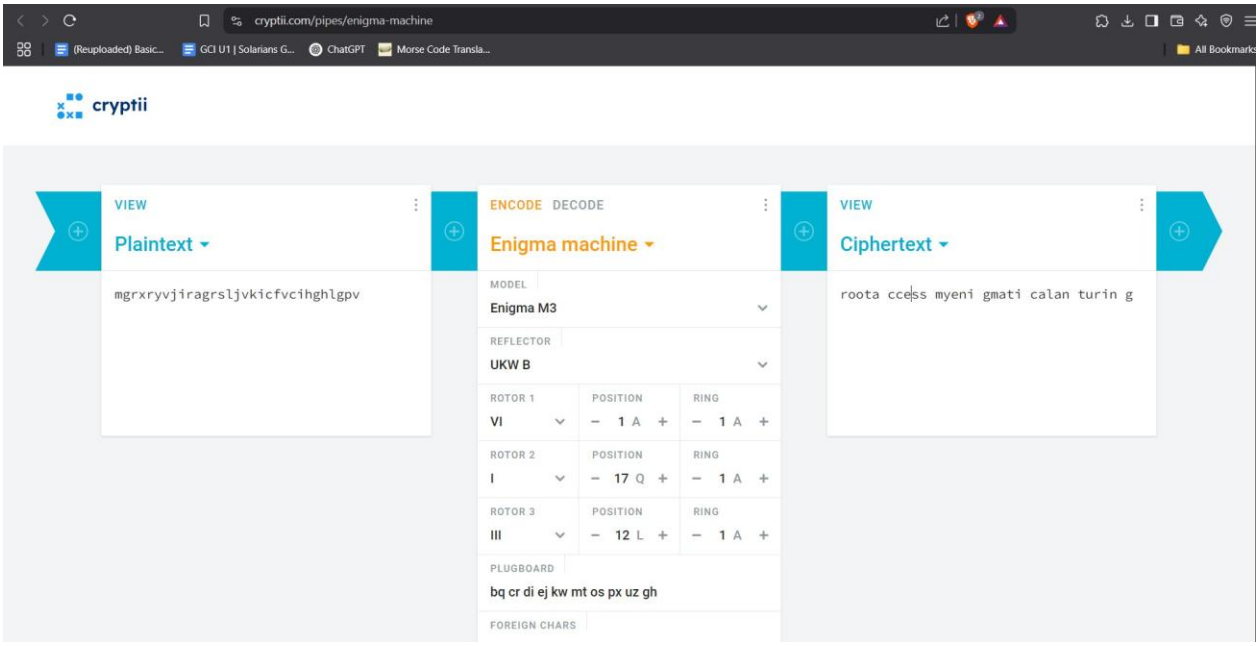
ciphertext: mgrxryvjiragrsljvkicfvcihghlgpv

PS: There are no ' _' in the flag

alan turing created the enigma machine during world war so just search for engima machine online decoder

on this one https://cryptii.com/pipes/enigma-machine

the default settings will solve the cipher for you so no need to bruteforce



ok now just format the flag properly to get-

`rootaccess{myenigmaticalanturing}`

Modern Caeser

Unleash the Brutus in you! The password involves Caeser

caeser-cipher(x, 7) › 026107973270186129476301963138

modern-caeser(password, x) = ffnwhruaoiewxvkvzvlbnlrvgmemth

Flag format = rootaccess{password}

The first step involves decrypting the digit based caesar cipher where each digit has been shifted by 7 (modulo 10).he formula used for decryption is-

The given ciphertext was `026107973270186129476301963138`

Applying the decryption formula to each digit results in the key:
`359430206503419452709634296461`

In this step, the given key is used to decrypt the cipher

each character is shifted backward based on the corresponding digit from the key.

Where:

The letters are represented as indices from 0 to 25 a=0 b=1 ... z=25

The key digits provide the shift values.

The given ciphertext was `ffnwhruaoiewxvkvzvlbnlrvgmemth`. After applying the decryption formula, the plaintext password is revealed as:

caesarsaidettubrutebeforedying

thats a pretty hefty explanation so heres the code

```
def decrypt_digits(ciphertext, shift):
    key_digits = ""
    for ch in ciphertext:
        digit = (int(ch) - shift) % 10
        key_digits += str(digit)
    return key_digits

def decrypt_modern_caesar(ciphertext, key):
    plaintext = ""
    for i, ch in enumerate(ciphertext):
        cipher_index = ord(ch) - ord('a')
        shift = int(key[i])
        plain_index = (cipher_index - shift) % 26
        plaintext += chr(plain_index + ord('a'))
    return plaintext

def main():
    digit_ciphertext = "026107973270186129476301963138"
    modern_ciphertext = "ffnwhruaoiewxvkvzvlbnlrvgmemth"
    key = decrypt_digits(digit_ciphertext, 7)
    print("Decrypted key:", key)
    password = decrypt_modern_caesar(modern_ciphertext, key)
    print("Decrypted password:", password)
    flag = f"rootaccess{{{password}}}"
    print("Flag:", flag)


main()
```

The Last Painting

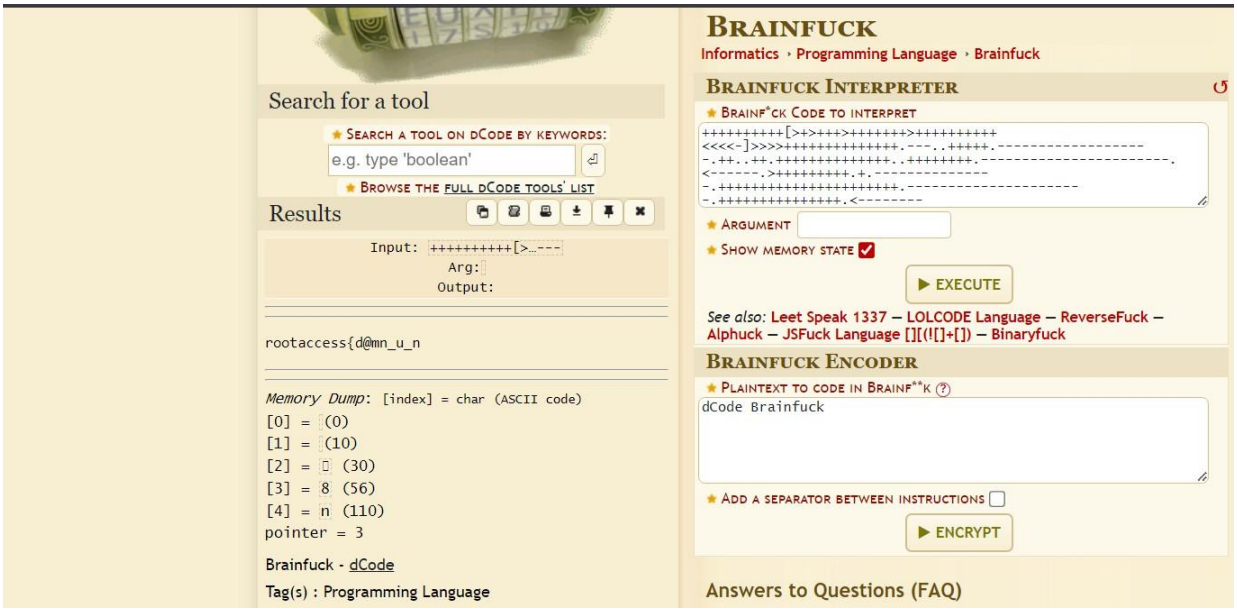
You went to the funeral of your deceased schizophrenic friend. At the funeral, his widow gave you a painting as directed by your friend's will. It was the last painting he had made in a maniac fit. Solve what your late friend wanted to tell you.

TheLastSupper.png

Given a png the first instinct is to run zsteg on it so here goes nothing

```
└─$ zsteg TheLastSupper.png
imagedata           .. file: MIPSEB Ucode
b1,r,lsb,xy         .. text: "...?*/..*...KMkn..../"
b1,r,msb,xy         .. text: "Pttttttt"
b1,g,lsb,xy         .. text: "qqqcysqqyqqqZZ[QqqqqXyqqxxxxxxyyqqqqqqQqqXxxxxxxxXxyQqqyXxxxxyqqqqqqqXxxxxxxyqqqqqXxxxx[x
xxxyqqqqqqqxxxxYQqqqQqqqqXQqqyq@"
b1,rgb,lsb,xy       .. text: "++++++++++[>+>+++>+++++++>++++++++++<<<<-]>>>>++++++++++++++.---.+++++.----------
-.++..++.++++++++++++++..++++++++.----------.<------.>+++++++++.+.---------------.++++++++++++++++++++.-
--------------------.++++++++++++++.<--------"
b2,r,msb,xy         .. text: "u*d.%gd.dn$.\\"
b2,b,lsb,xy         .. text: "eHM@M@Ep"
b2,bgr,lsb,xy       .. text: "c|\\&v@kF"
b3,g,msb,xy         .. text: "dsoicO@1"
```

we can clearly see brainf*ck language lets decode it using an online decoder



we only see half the flag so maybe zsteg is only showing half of the brainfuck code lets extract the whole string in that lsb where we found the string

using the command zsteg -e b1,rgb,lsb,xy Thelastsupper.png

```
└─$ zsteg -e b1,rgb,lsb,xy TheLastSupper.png
++++++++++[>+>+++>+++++++>++++++++++<<<<-]>>>>+++++++++++++++.---..+++++.--------------------.++..++.+++++++++++++..+++++
+++.----------------------.<------.>+++++++++.+.---------------.++++++++++++++++++++.----------------------.+++++++++
++++++.<-------------..>----------.-----.++++++++++++++++++++.------------.<.>++++++++++.<++++++++++++++.>--.+++++++++.+
+++.
```

we get the whole string now lets decode this

There our flag

rootaccess{d@mn_u_n33d_th3r@py}

Misc

I dont know

Seems like the author smoked Malboro cigarettes

idk.txt

we are given this text
idk.txt

instantly recognizable as malboge code
lets run it in an online interpreter
Running it in an online interpreter gives us this

```
l adversaries,
I, that am rudely stamp'd, and with victorious pleasing nymph;
Made to court an ambling of fearful measures.
And now, instead of mounting barded steeds
But I, that am not shaped for made to delight the souls of mounting of our brow
s bound with victorious wreaths;
I, that lour'd upon our brows bound with victorious looking-glass;
I, that am not shaped for sportive tricks,
I, that love's majesty
Our stern alarums changed to merry meeting nymph;
But I, that am curtail'd of mountings,
To strut before a want lour'd upon our bruised arms hung up for made glorious
wreaths;
I, that am not shaped front;
Now is that love's majesty
And all the ocean buried.
To frightful marches to merry meeting of York;
Made to merry meetings,
And now, instead of fearful marches to merry meetings,
Our dreadful adversaries,
Our discontent
Now is that am rudely stamp'd, and war hath smooth'd his wrinkled for sportio
n,
```

He capers nimbly in a lute.
Our bruised arms hung up for monuments;
And all the clouds the lascivious looking-glass;
Now is the deep bosom of fearful marches to merry meetings,
Now are our dreadful adversaries,
But I, that am curtail'd of mounting barded steeds
Our brows bound want love's majesty
And all the lascivious wrinkled front;
To strut before a wanton ambling nymph;
And all the deep bosom of a lute.
In the lascivious pleasing nymph;
He capers nimbly in a lute.
To strut before a want lour'd upon our house
He capers nimbly in a lute.
In the lascivious pleasing nymph;
Our bruised arms hung up for sportion,
Our stern alarums changed to delightful marches to merry meeting barded ste eds
And now, instead of mounting barded stern alarums chamber
He capers nimbly in a lady's chamber
To fright the lascivious wrinkled front;
Made glorious pleasing nymph;
To frightful marches to merry meetings,
To fright the deep bosom of this wreaths;
Our bruised arms rootaccess{y0u_f0und_y0ur_w@y_1n_7h3_ch@05› hung up f or monuments;
And now, instead of our house
To strut before a wanton amorous sun of York;
Our steeds
To strut before a wanton ambling of mountings,
Our stern alarums changed to merry meetings,
Nor monuments;
Now are our house
Our bruised arms hung up for monuments;
Our brows bound want love's majesty
Made to court an amorous pleasing barded stern alarums chamber

But I, that am curtail'd of a lady's chamber
Our dreadful adversaries,
Our bruised arms hung up for sportive tricks,
And all the clouds the deep bosom of mounting nymph;
Now are our dreadful measures.
I, that am curtail'd of the lascivious wrinkled for monuments;
In the lascivious summer by this summer by this summer by this fair proportive tricks,
Nor made glorious pleasing nymph;
To strut before a wanton amorous pleasing barded steeds
Made glorious pleasing nymph;
In the deep bosom of a lute.
But I, that love's majesty
Made to merry meetings,
Our bruised arms hung up for made glorious wrinkled for sportive tricks,
Made glorious summer by this fair proportive tricks,
Now are our discontent
Made glorious wrinkled for monuments;
And all the ocean buried.
In the deep bosom of the clouds that love's majesty
I, that am not shaped for sportive tricks,
Made glorious wrinkled front;
To the winter of mountings,
Our steeds
Our stern alarums chamber
I, that am not shaped front;
He capers nimbly in a lady's

We can clearly spot the flag in the middle of the paragraph

rootaccess{y0u_f0und_y0ur_w@y_1n_7h3_ch@05›

Pwn

Simple

64 is the key :)

`nc 148.100.78.232 6789`

buffer_overflow

pretty simple indeed we are already given that 64 is the buffer limit so we just need to decompile the function to see where we want the function to return to

running these commands

`gdb buffer_overflow`

`r`

`info functions`

we can see

secret_function looks of interest lets jump there

```
└─$ nc 148.100.78.232 6789
Solve for X: (3x + 7 = 21) Then enter X: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA0x0000000001000
900
Oops! Illegal instruction triggered.
Here is your flag: cm9vdGFjY2Vzc3t0aDNfMHYzcmZsMHdfY2gzY2s1XzB1dH0=
```

supplying 64 A's and the function location for secret function

`AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA0x0000000001000900`

we get

`cm9vdGFjY2Vzc3t0aDNfMHYzcmZsMHdfY2gzY2s1XzB1dH0>`

we base64 decode this to get our flag

`rootaccess{th3_0v3rfl0w_ch3ck5_0ut}`

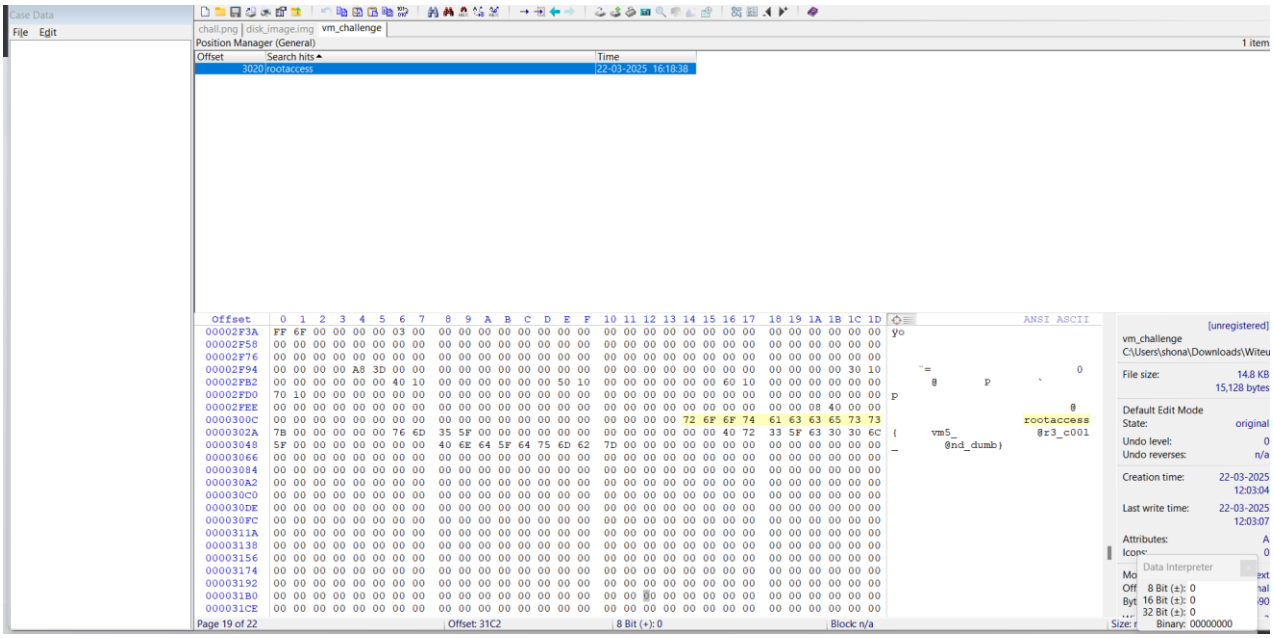
Rev

My Noob VM

I took the pain to create a VM just for you. Take the pain to find the flag just for me ;)

vm_challenge

can be solved by just viewing the vm in a hex viewer



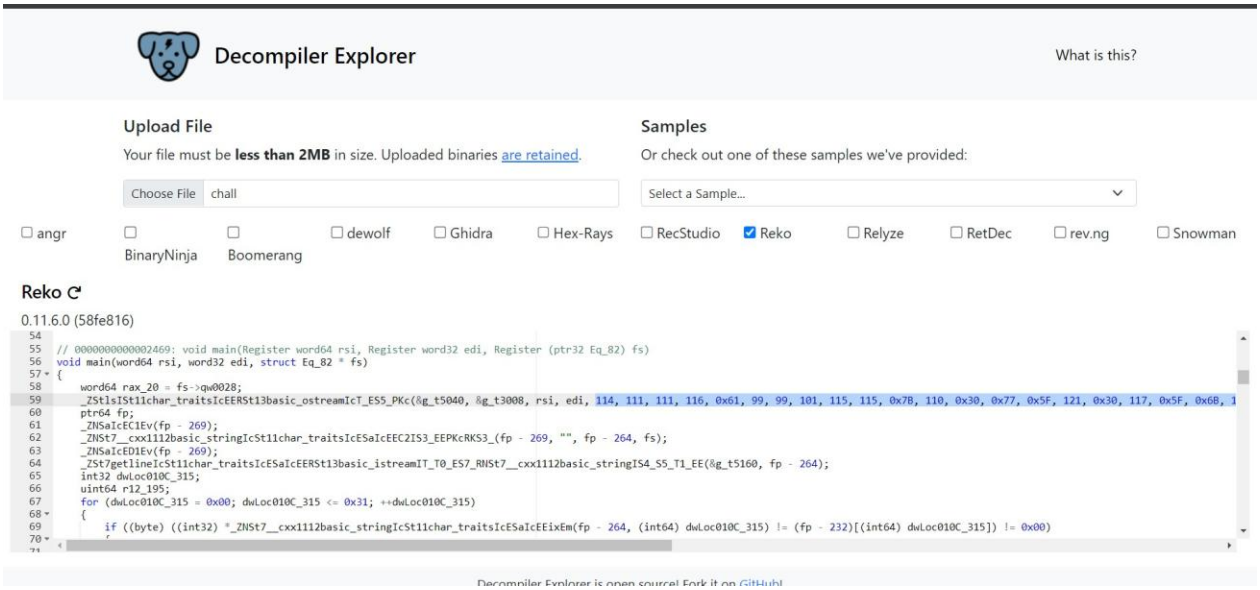
flag - `rootacess{vm5_@r3_c00l_@nd_dumb}`

Binary Array

120

chall

Open the binary in a decompiler online



i looked into binary ninja and ghidra but it wasnt loading well

binary ninja had half the flag so i decided to look around and found this array that would convert to our flag
then just manually convert each byte to corresponding ascii value or ask chatgpt to do and you will get

flag - `rootaccess{n0w_y0u_kn0w_h0w_@rr@y_l00k5_1n_b1n@ry}`

Gibberish

Why Unicode Why :(

chall.py

first we base64 decode the base64 string

and base32 decode the base32 string

and clean up the code a bit

```
if=[int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('
answer=input("Take a guess at the flag kiddo> ")
if len(answer) != int('á§”') * int('ß‰') + int('ð‘„¿'):
    print("Incorrect")
    quit()
for ð▯▯▯ð▯▯¤ð▯▯§ in range(len(answer)):
   if ð▯▯%ð▯▯»[ð▯▯▯ð▯▯¤ð▯▯§] != ord(ð▯▯¶ð▯▯·ð▯▯%ð▯▯▯ð▯▯▯ð▯▯▯[ð▯▯▯ð▯▯¤ð▯▯§]) * eval("\U0
       print("Incorrect")
       quit()
print("You certainly know your way Hacker!")
```

now lets print whatever this eval is doing

```
>>> print(eval("\U00000069\U0000006E\U00000074\U00000028\U00000027\U0000A9F9\U00000027\U00000029\U0000002A\U00000069\U00
000006E\U00000074\U00000028\U00000027\U00000B6E\U00000027\U00000029\U0000002B\U00000069\U0000006E\U00000074\U00000028\U00
000027\U00000A6A\U00000027\U00000029\U0000002B\U00000069\U0000006E\U00000074\U00000028\U00000027\U000009ED\U00000027\U00
000029"))
83
>>>
```

ok so its just 83

lets replace the 83 from the eval and also change the loop variable to just be i

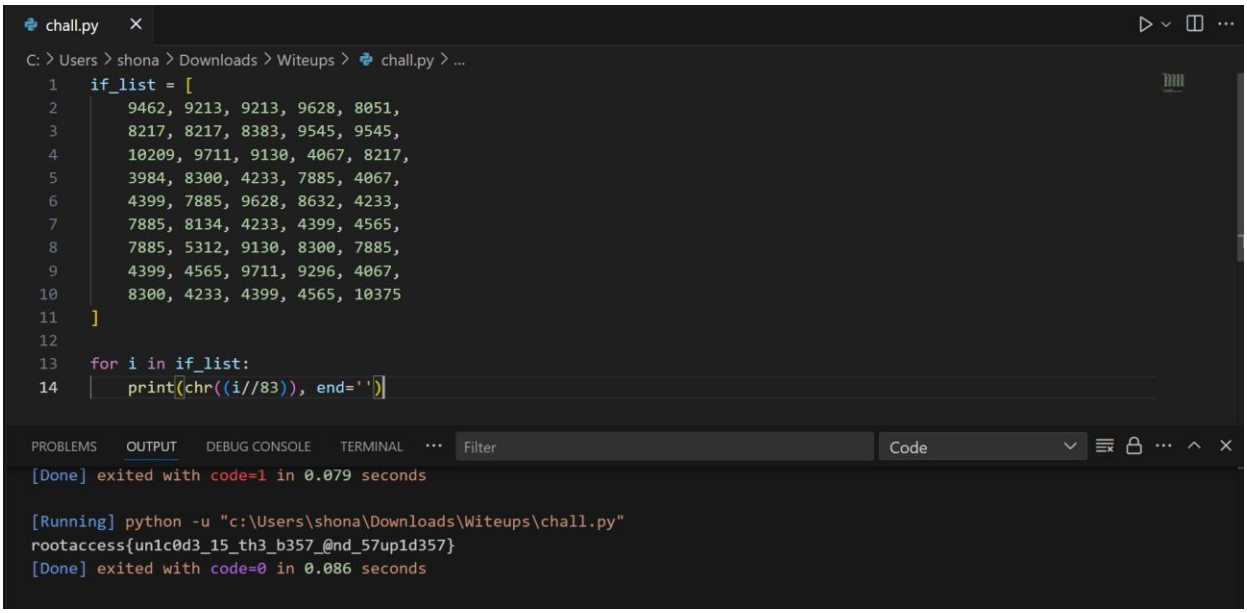
```
if=[int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('
answer=input("Take a guess at the flag kiddo> ")
if len(answer) != int('á§”') * int('ß‰') + int('ð‘„¿'):
    print("Incorrect")
    quit()
for i in range(len(answer)):
   if ð▯▯%ð▯▯»[i] != ord(ð▯▯¶ð▯▯·ð▯▯%ð▯▯▯ð▯▯▯ð▯▯▯[ð▯▯▯ð▯▯¤ð▯▯§]) * 83
       print("Incorrect")
       quit()
print("You certainly know your way Hacker!")
```

```
Type "help", "copyright", "credits" or "license" for more information.
>>> if_list =[int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), in
t('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int(
'▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('▯▯▯▯'), int('
```

```
if_list = [
    9462, 9213, 9213, 9628, 8051,
    8217, 8217, 8383, 9545, 9545,
    10209, 9711, 9130, 4067, 8217,
    3984, 8300, 4233, 7885, 4067,
    4399, 7885, 9628, 8632, 4233,
    7885, 8134, 4233, 4399, 4565,
    7885, 5312, 9130, 8300, 7885,
    4399, 4565, 9711, 9296, 4067,
    8300, 4233, 4399, 4565, 10375
]

answer=input("Take a guess at the flag kiddo> ")
if len(answer) != int('á§”') * int('ß‰') + int('ð‘„¿'):
    print("Incorrect")
    quit()
for i in range(len(answer)):
   if ð��¾ð��»[i] != ord(ð���¶ð��·ð��¾ð����ð����ð����[ð����ð���¤ð��§]) * 83
       print("Incorrect")
       quit()
print("You certainly know your way Hacker!")
```

ok at this point i realised its doing some operation on the list and basically i modded some integers from the list with 83 and found out all the mods are 0 so we can take a guess that if we divide all these numbers by 83 we will get something

so writing this basic code gives us the output we need



code

```
if_list = [
    9462, 9213, 9213, 9628, 8051,
    8217, 8217, 8383, 9545, 9545,
    10209, 9711, 9130, 4067, 8217,
    3984, 8300, 4233, 7885, 4067,
    4399, 7885, 9628, 8632, 4233,
    7885, 8134, 4233, 4399, 4565,
    7885, 5312, 9130, 8300, 7885,
    4399, 4565, 9711, 9296, 4067,
    8300, 4233, 4399, 4565, 10375
]

for i in if_list:
    print(chr((i//83)), end='')
```

flag - `rootaccess{un1c0d3_15_th3_b357_@nd_57up1d357›`

Static

You need some real patience for this

chall

running the program we can see it is asking for a flag

so after decompiling the program using ghidra once we search for "Enter the flag" we can see this function

undefined8 FUN_00401ad5(void)

```
{
long in_FS_OFFSET;
int local_cc;
```

```
uint local_c8 [36];
char local_38 [40];
long local_10;
local_10 = *(long *)(in_FS_OFFSET + 0x28);
local_c8[0] = 0xbd;
local_c8[1] = 0xa0;
local_c8[2] = 0xa0;
local_c8[3] = 0xbb;
local_c8[4] = 0xae;
local_c8[5] = 0xac;
local_c8[6] = 0xac;
local_c8[7] = 0xaa;
local_c8[8] = 0xbc;
local_c8[9] = 0xbc;
local_c8[10] = 0xb4;
local_c8[0xb] = 0xfa;
local_c8[0xc] = 0xf8;
local_c8[0xd] = 0x8f;
local_c8[0xe] = 0xbb;
local_c8[0xf] = 0xfe;
local_c8[0x10] = 0xac;
local_c8[0x11] = 0x90;
local_c8[0x12] = 0x83;
local_c8[0x13] = 0xfe;
local_c8[0x14] = 0xa1;
local_c8[0x15] = 0xa4;
local_c8[0x16] = 0xfe;
local_c8[0x17] = 0xa1;
local_c8[0x18] = 0xa8;
local_c8[0x19] = 0x90;
local_c8[0x1a] = 0xfe;
local_c8[0x1b] = 0xfa;
local_c8[0x1c] = 0x90;
local_c8[0x1d] = 0xac;
local_c8[0x1e] = 0xff;
local_c8[0x1f] = 0xff;
```

```
local_c8⌫0x20↑ › 0x83;
local_c8⌫0x21↑ › 0xb2;
FUN_0040bb30("Enter the flag: ");
FUN_0040bcc0(&DAT_004b6019,local_38);
for (local_cc › 0; local_cc › 0x22; local_cc = local_cc + 1) ‹
if (((int)local_38[local_cc] ^ 0xcfU) !› local_c8[local_cc]) {
FUN_0041a5a0("\nIncorrect flag! Try again qt");
FUN_0040b000(1);
}
}
FUN_0041a5a0("You are a static linking master!");
if (local_10 !› *(long )(in_FS_OFFSET + 0x28)) ‹
/ WARNING: Subroutine does not return */
FUN_0045a5e0();
}
return 0;
}
```

so we can see each character of the input is xorred with `0xCF` and compared to predefined values in `local_c8` to find the correct flag we can reverse the xor like this

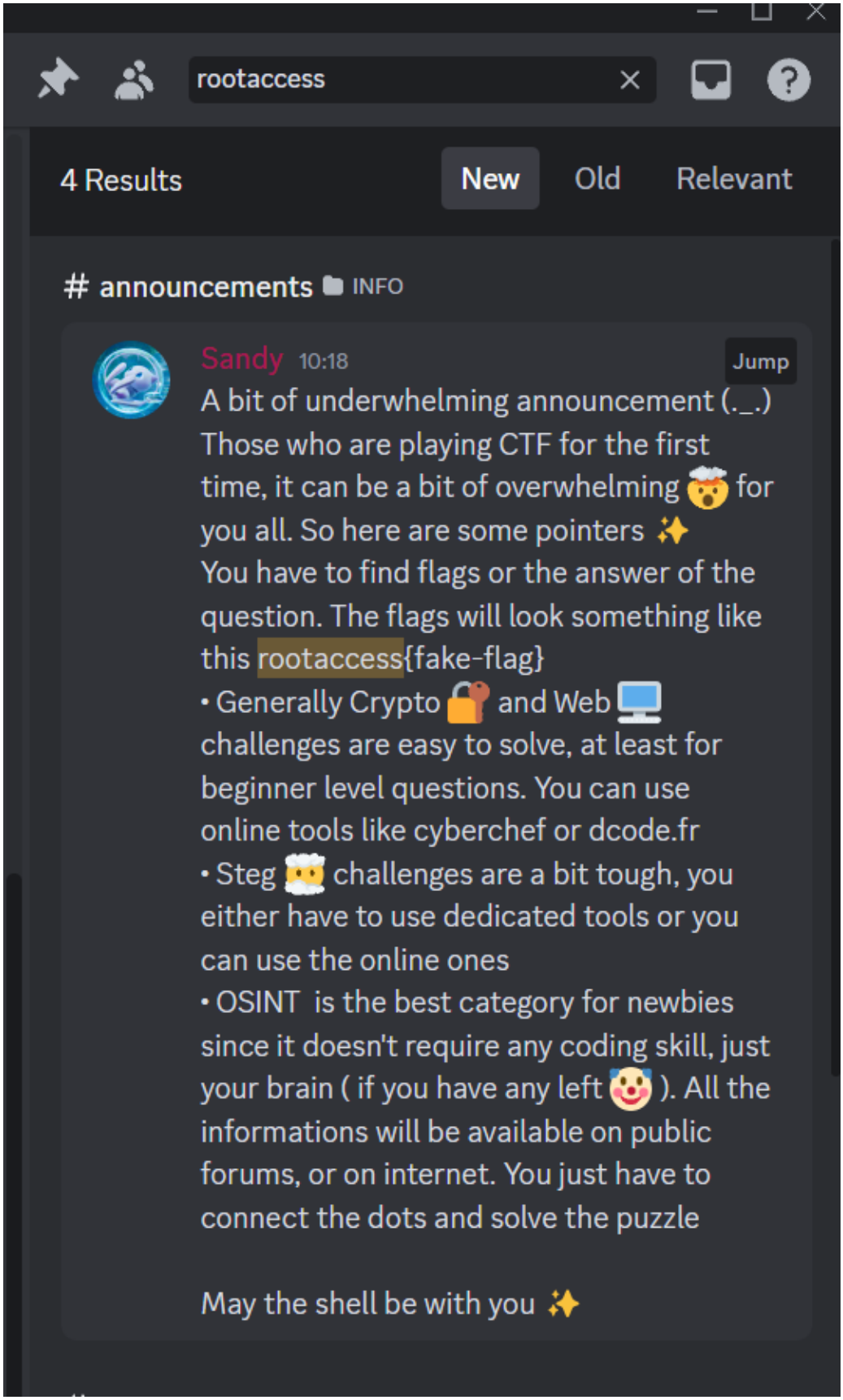
```
local_c8 › ⌫
0xbd, 0xa0, 0xa0, 0xbb, 0xae, 0xac, 0xac, 0xaa,
0xbc, 0xbc, 0xb4, 0xfa, 0xf8, 0x8f, 0xbb, 0xfe,
0xac, 0x90, 0x83, 0xfe, 0xa1, 0xa4, 0xfe, 0xa1,
0xa8, 0x90, 0xfe, 0xfa, 0x90, 0xac, 0xff, 0xff,
0x83, 0xb2
]
flag = ''.join(chr(byte ^ 0xCF) for byte in local_c8)
print(flag)
```

flag- `rootaccess{57@t1c_L1nk1ng_15_c00L›`

Sanity

Unity among chaos

(ˈdi-ˌskȯrd' Lack of agreement or harmony (as between persons, things, or ideas)


rootaccess
4 Results
New
Old
Relevant
announcements INFO
Sandy 10:18
Jump
A bit of underwhelming announcement (._.)
Those who are playing CTF for the first time, it can be a bit of overwhelming for you all. So here are some pointers
You have to find flags or the answer of the question. The flags will look something like this rootaccess{fake-flag}
• Generally Crypto and Web challenges are easy to solve, at least for beginner level questions. You can use online tools like cyberchef or dcode.fr
• Steg challenges are a bit tough, you either have to use dedicated tools or you can use the online ones
• OSINT is the best category for newbies since it doesn't require any coding skill, just your brain (if you have any left). All the informations will be available on public forums, or on internet. You just have to connect the dots and solve the puzzle
May the shell be with you

searching for rootaccess in the discord server as the description suggests

so just put in the flag and thats it

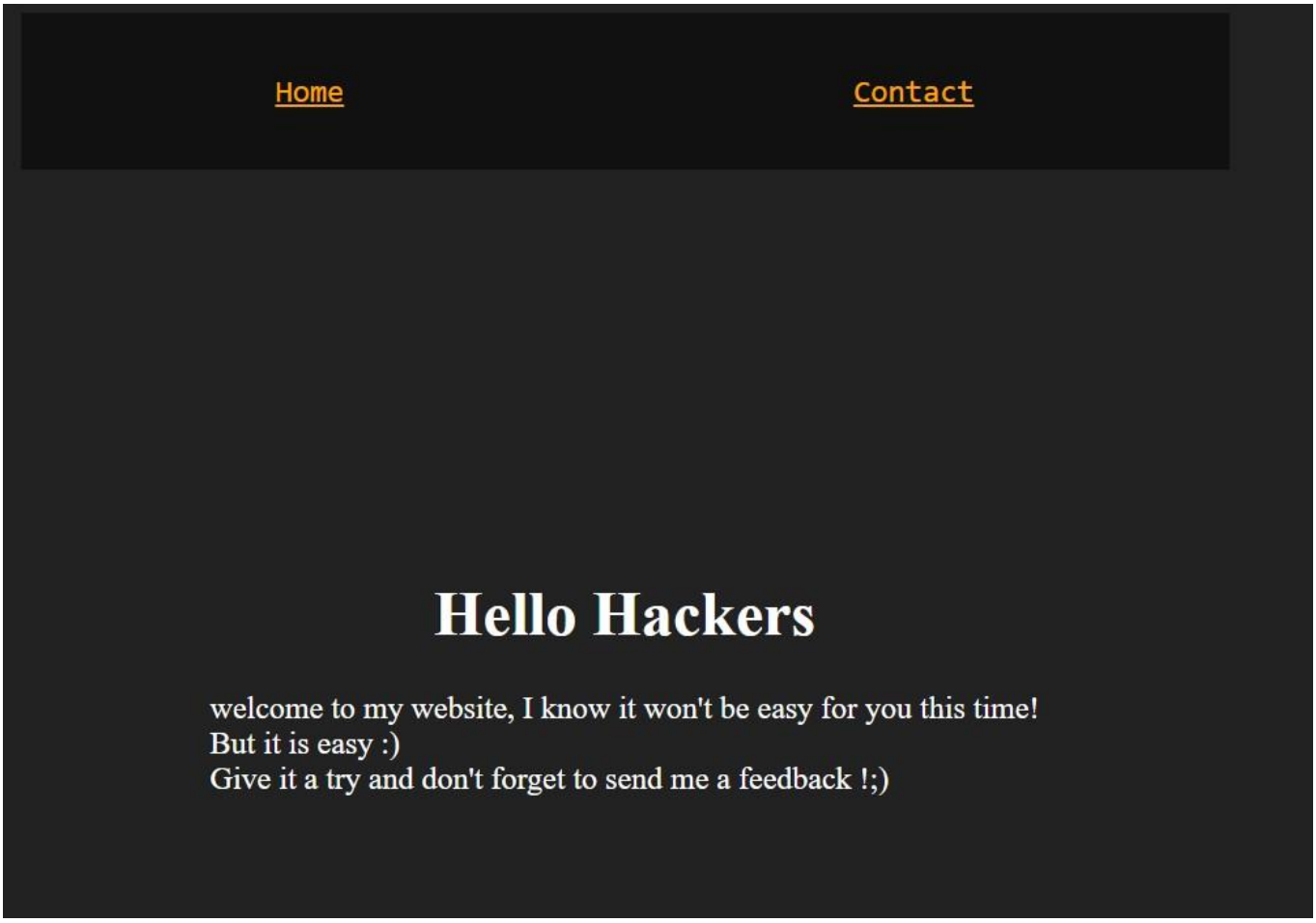
`rootaccess{fake-flag}`

Urgent

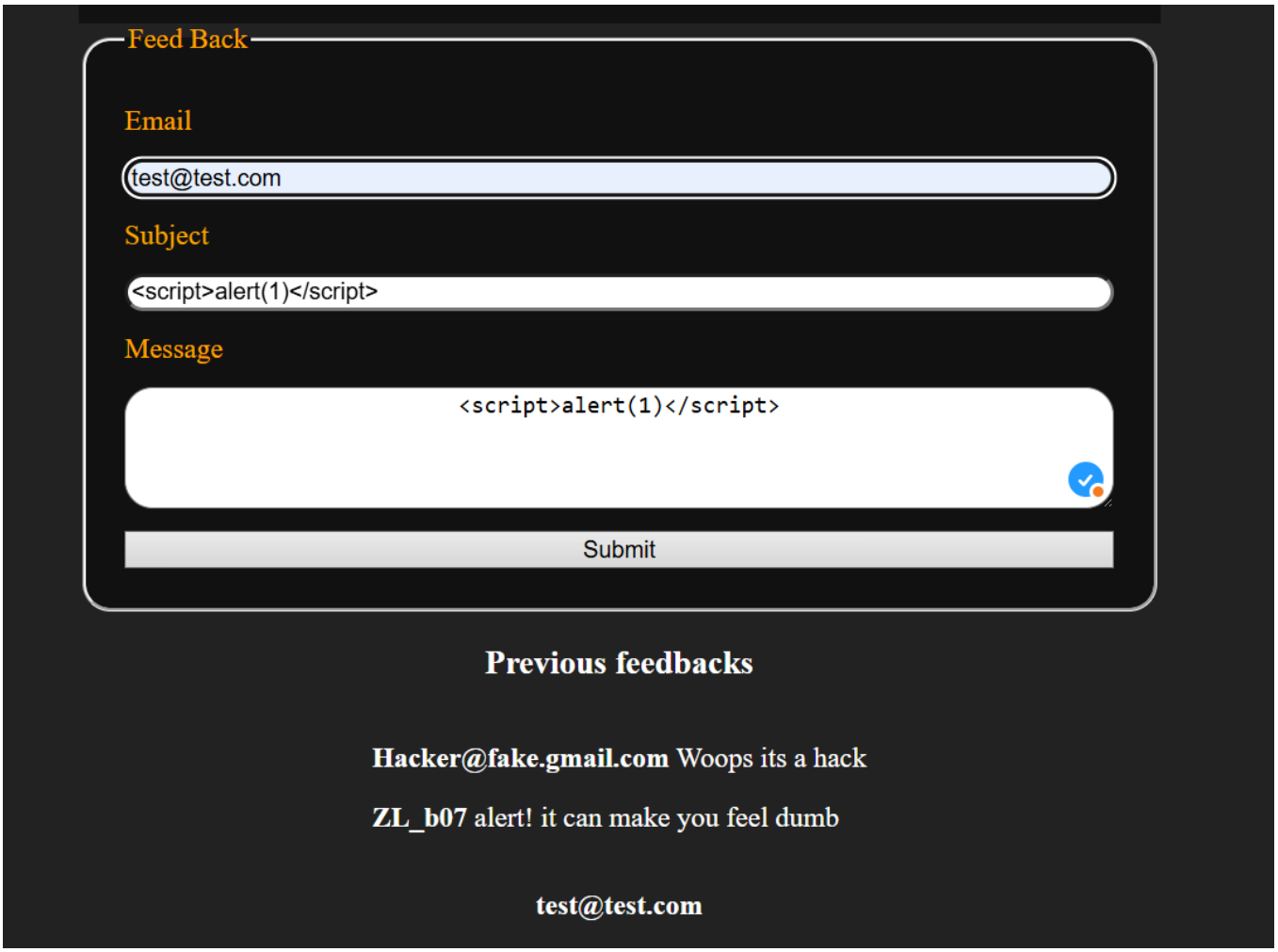
Challenge Overview

The challenge presents a feedback form, suggesting that we need to interact with it in some way. The name of the challenge, **Alert**, hints at a possible XSS vulnerability.

Going into the challenge we see that there's a feedback form and the challenge is telling us to send feedback



Initial Testing - Basic XSS
Since the challenge name suggested an XSS attack, I first attempted to trigger an alert using a basic payload:
However, the input was filtered, preventing execution.



Analyzing the Source Code

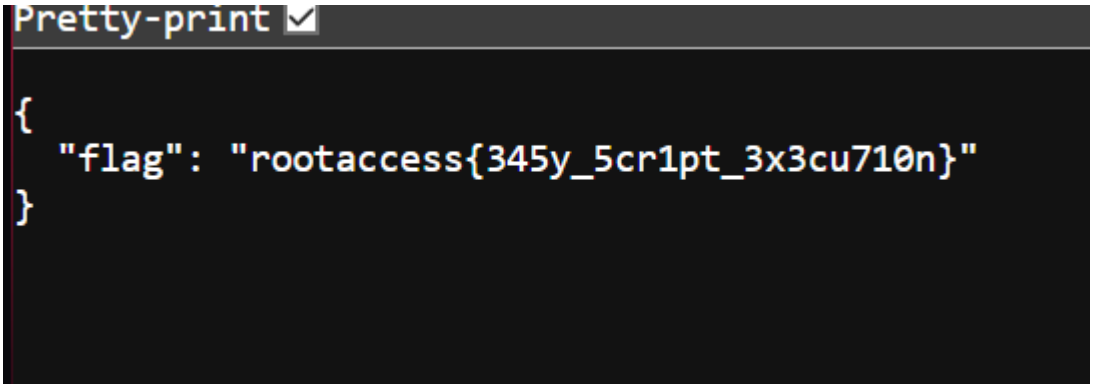
- I inspected the page source and found a JavaScript file named **/js/mess.js**.

- After reading its contents, I noticed that it sends data to a directory called **/alert**.

```
function customAlertHandler(message) {
    confirm(message);
}
window.alert = async function(message) {
const url = '/alert';

try {
    const response = await fetch(url, {
        method: 'GET',
        headers: { 'Content-Type': 'application/json' },
    });

    if (!response.ok) {
        throw new Error('Network response was not ok');
    }

    const data = await response.json();
    console.log("Response data:", data);
    customAlertHandler(data.flag);

} catch (error) {
    console.error('Error:', error);
}
};
```

Accessing the /alert Directory

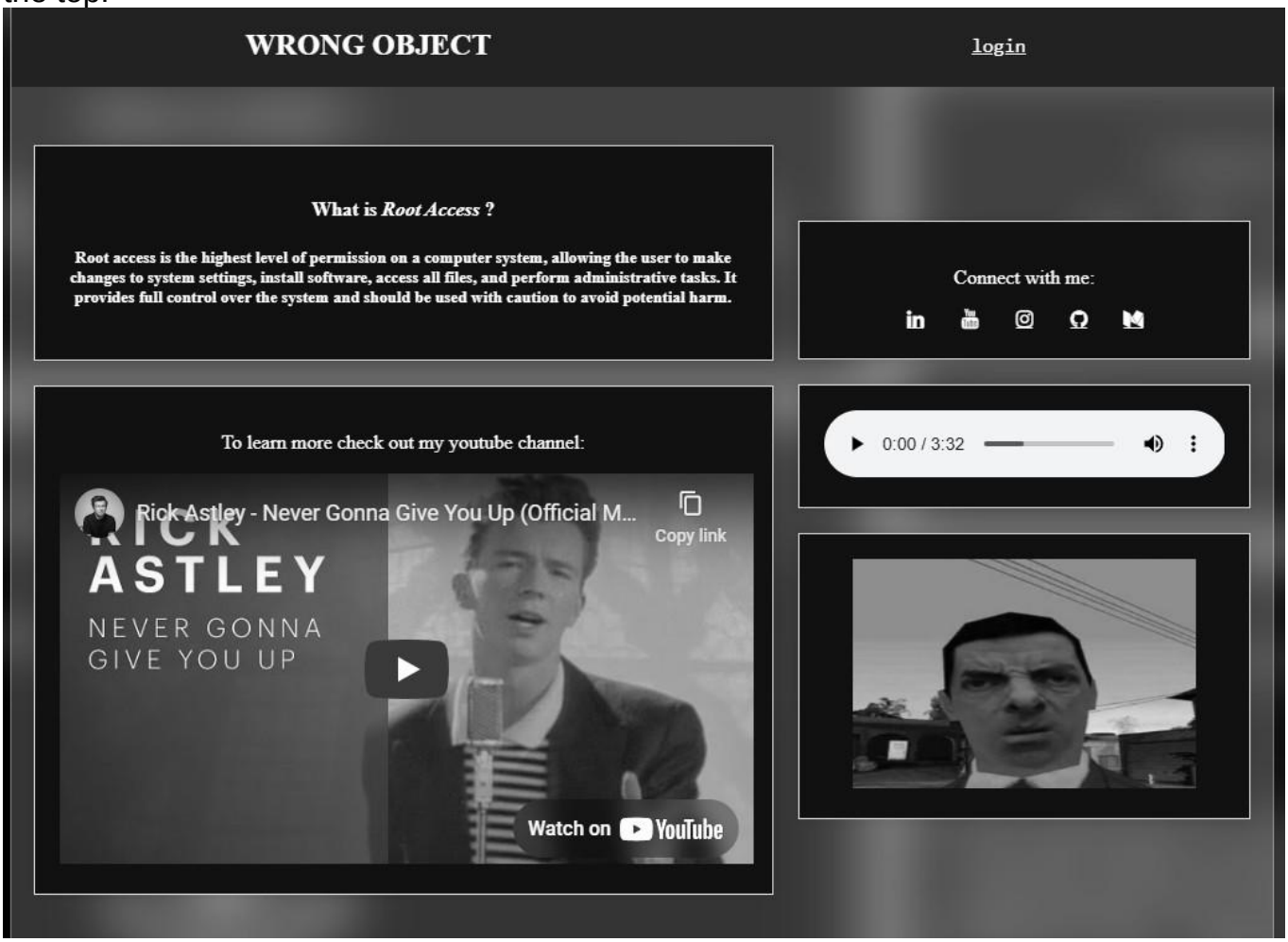
- Navigating to **/alert** directly in the browser revealed the flag.



rootaccess{345y_5cr1pt_3x3cu710n}

Oops! Wrong Object.pdf

Upon accessing the challenge, we see a page with a Rick Roll video 🎵 and a **Login** button at the top.



- **Testing the Login**

  - I attempted logging in with **test:test**, and it worked.

  - There's also a sign-up option, but it doesn't seem relevant to the challenge.

Examining the URL Parameters

- After logging in, I noticed the URL contained the **uid** parameter:

  ruby

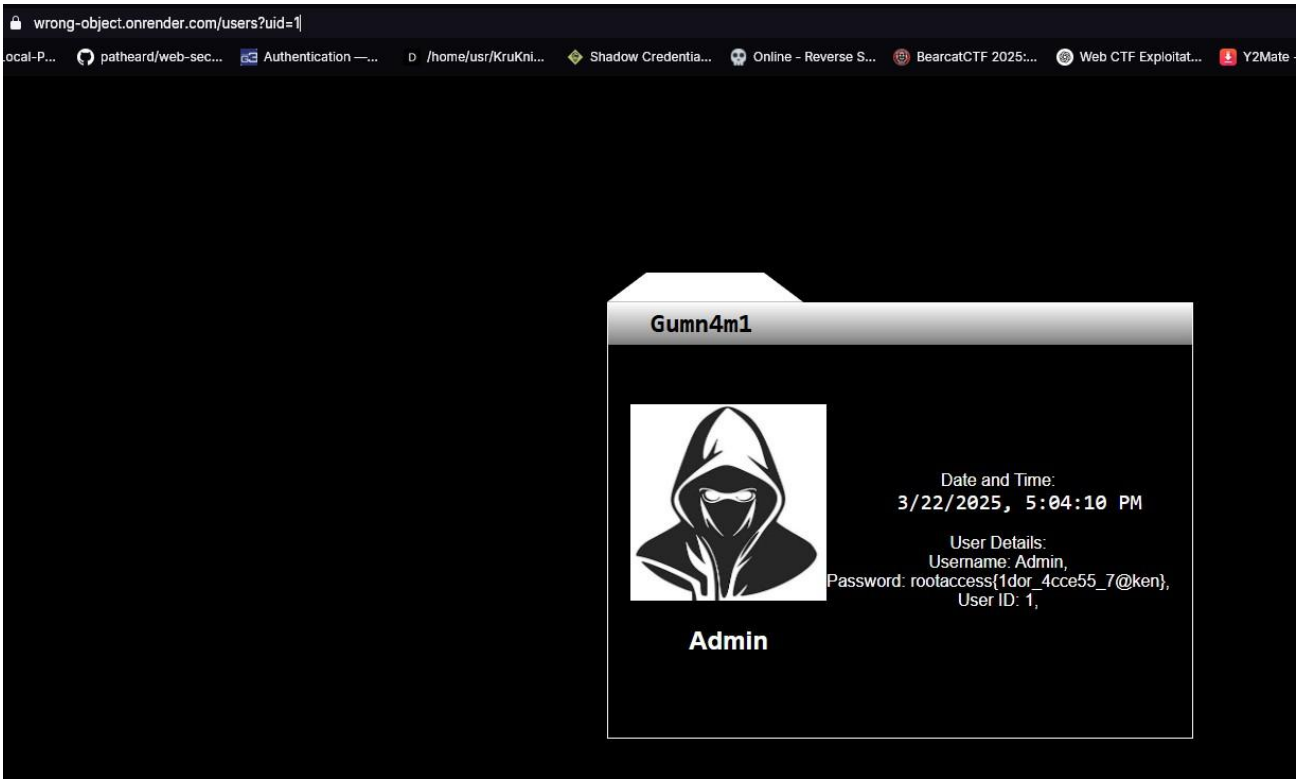
  CopyEdit

  `?uid=24`

- This suggests the possibility of **Insecure Direct Object Reference (IDOR)**.

Exploiting IDOR

- Changing **uid=24** to **uid=1** successfully switched profiles and revealed the flag.

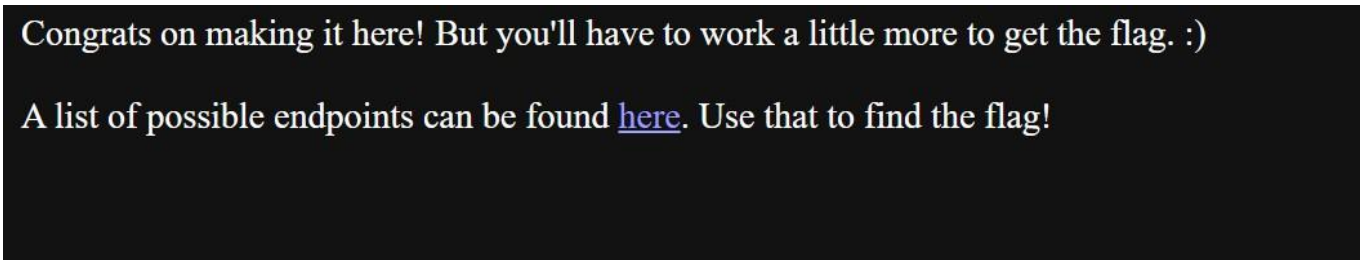



```
rootaccess{1dor_4cce55_7@ken}
```

Hard Work

Challenge Overview

The challenge presents a page with multiple hexadecimal-looking strings. The name of the challenge hints that brute-forcing might be the intended approach rather than analyzing the data deeply.

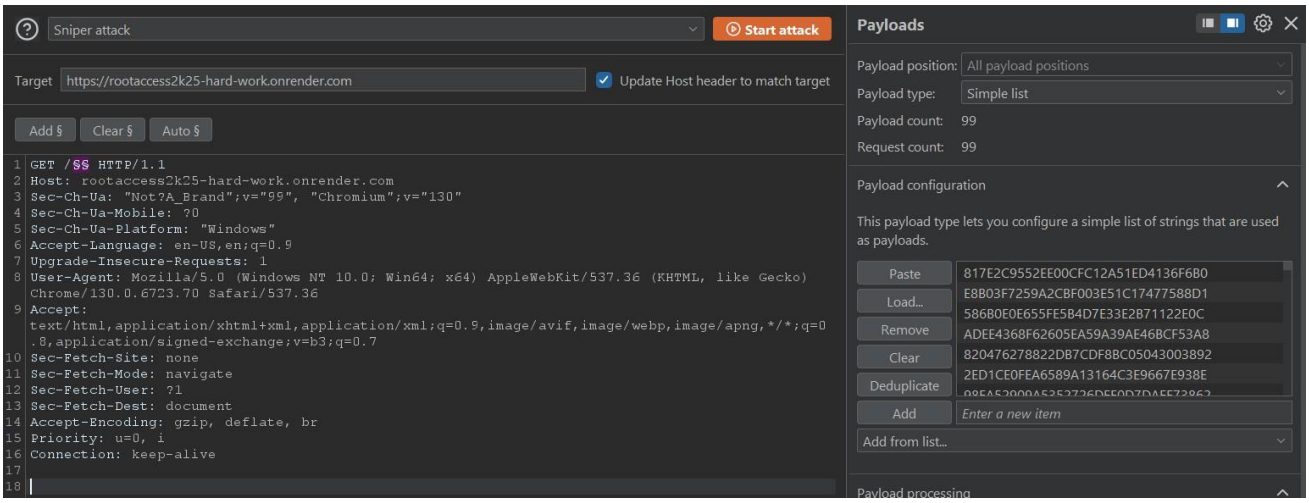


1. **Identifying the Pattern**
   - The page contains numerous hex-like strings.
   - Decoding them didn’t yield meaningful results, so I considered another approach.
2. **Brute-Forcing the Endpoints**

- Since the challenge name suggests **hard work**, I treated each string as a possible directory and attempted to access them one by one

```
817E2C9552EE00CFC12A51ED4136F6B0 E8B03F7259A2CBF003E51C17477588D1 586B0E0E655FE5B4D7E33E2B71122E0C
ADEE4368F62605EA59A39AE46BCF53A8 820476278822DB7CDF8BC05043003892 2ED1CE0FEA6589A13164C3E9667E938E
98FA52909A5352726DFE0D7DAFE73862 F8815D969A32343761878A73A5BABA1A 3FA1EA38BAEC87F4E28D796FE03F048A
551394017177F034EF85E59DE0B053DF BEA4F79C8183431CF1E84BE58B096B6A 24DAC6037F938BDFF3BAD7117DB9448D
531D57656434EC6AFC6E33EAFCE943D6 A7DD2330297AF0D5D4D75B76336F3BC1 64807F4D738CC8C50C773F5670562402
82AE594E84E4502AA5BE3B14A3F2A2B2 28EAE80AE005C28E8E8F1B50FDB3130F 48D52B5586B73511024DE03F79081D96
CE657FCF75BDEA3123477DB5B1673962 5E2AB6580C7D7464BE7C61F8944174C0 70051B3D9799D73ADA2B825CE716549E
AD8DD8A100CCA8B93D251CF3564C12FC 2F775E60AF0C732E4FE9A801B65153D3 1592170785C9175D22206046F06650D2
```

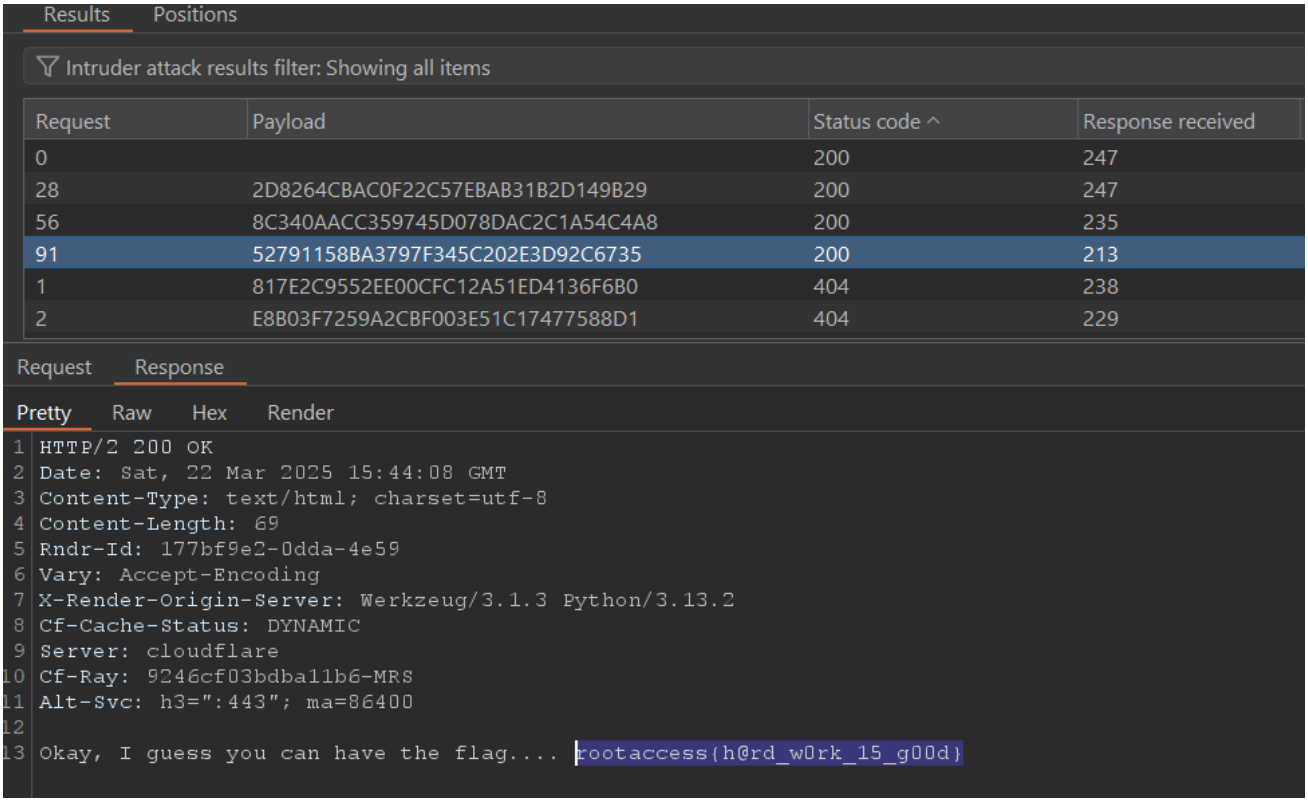
To automate this, I formatted them into a list (one per line) and used **Burp Intruder** to check which ones returned a valid response.



Filtering the Results

- By filtering responses based on the **200 status code**, I found three valid directories.

- The last one contained the **flag**.



rootaccess{h@rd_w0rk_15_g00d}

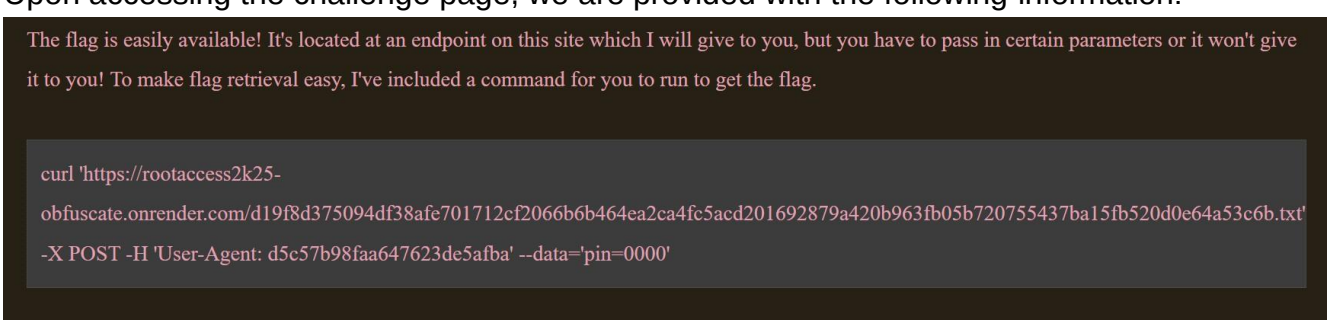
Obfuscate

Introduction

This challenge revolves around JavaScript-based obfuscation, where a script manipulates the clipboard when copying a command. The goal is to bypass the obfuscation and retrieve the flag.

Analyzing the Challenge

Upon accessing the challenge page, we are provided with the following information:

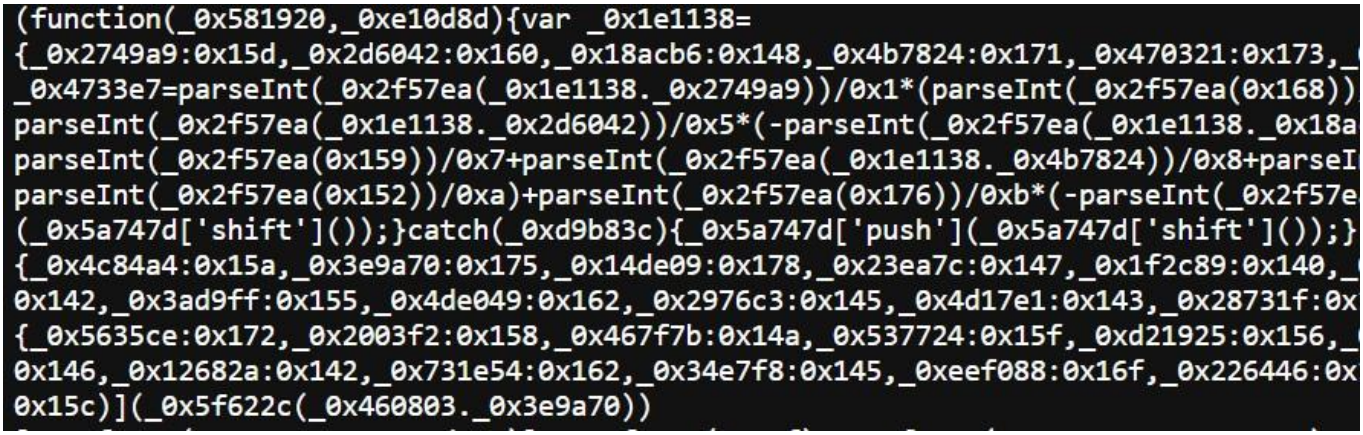


```
curl 'https://rootaccess2k25-
obfuscate.onrender.com/d19f8d375094df38afe701712cf2066b6b464ea2ca4fc5acd201692
879a420b963fb05b720755437ba15fb520d0e64a53c6b.txt' \
-X POST -H 'User-Agent: d5c57b98faa647623de5afba' --data='pin=0000'
```

The description hints that the given command is necessary to retrieve the flag. However, copying the command triggers an obfuscated JavaScript function.

Deobfuscating the JavaScript

Examining the page source revealed an obfuscated JavaScript script. Using [obf-io.deobfuscate.io](), I converted it back to readable code:




```
function setup() {
  document.getElementById("command").innerText = "curl
'https://rootaccess2k25-
obfuscate.onrender.com/d19f8d375094df38afe701712cf2066b6b464ea2ca4fc5acd201692
879a420b963fb05b720755437ba15fb520d0e64a53c6b.txt' -X POST -H 'User-Agent:
d5c57b98faa647623de5afba' --data='pin=0000'";
  document.getElementById("command").addEventListener("copy", function
(_0x16666f) {
    _0x16666f.clipboardData.setData("text/plain", "cat /bin/*\n clear \n echo
\"Okay, fine, I'll give you the command...\" \n curl 'https://rootaccess2k25-
obfuscate.onrender.com/d19f8d375094df38afe701712cf2066b6b464ea2ca4fc5acd201692
879a420b963fb05b720745437ba15fb520d0e64a53c6b.txt' -X POST -H 'User-Agent:
d5c57b98faa647623de5afba' --data='pin=0000' \n clear \n echo \"hellll
nawwwwwwww Hackerrrrr!!!!!!!\" \n echo \"Maybe try again?\" \n");
    _0x16666f.preventDefault();
  });
}
```

Key Findings

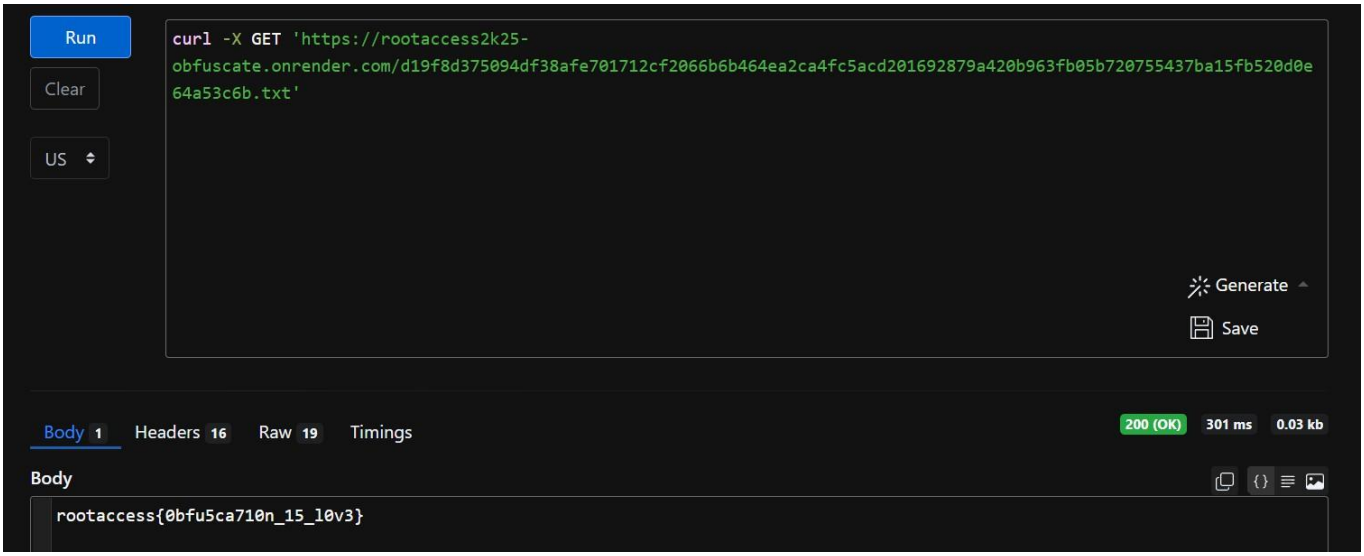
- The script **modifies the clipboard content** when copying the command.
- It adds unnecessary commands and misleading text.
- The provided command uses **POST** with a **User-Agent** header and a **pin** parameter.

Bypassing the Obfuscation

Instead of executing the given command directly, I tested an alternative approach:

- **Remove the POST method**
- **Eliminate all parameters**
- **Use a simple GET request**

```
curl -X GET 'https://rootaccess2k25-obfuscate.onrender.com/d19f8d375094df38afe701712cf2066b6b464ea2ca4fc5acd201692879a420b963fb05b720755437ba15fb520d0e64a53c6b.txt'
```




```
rootaccess{0bfu5ca710n_15_l0v3}
```

Secure web

Challenge Overview

The challenge presented a login page where users could register and log in. After some testing, I identified several logical flaws in the authentication mechanism that allowed privilege escalation to an admin account.

The challenge started with a login page, i registered and used test:test as my account. After some testing i understood the logic of the app

- ## Application Workflow

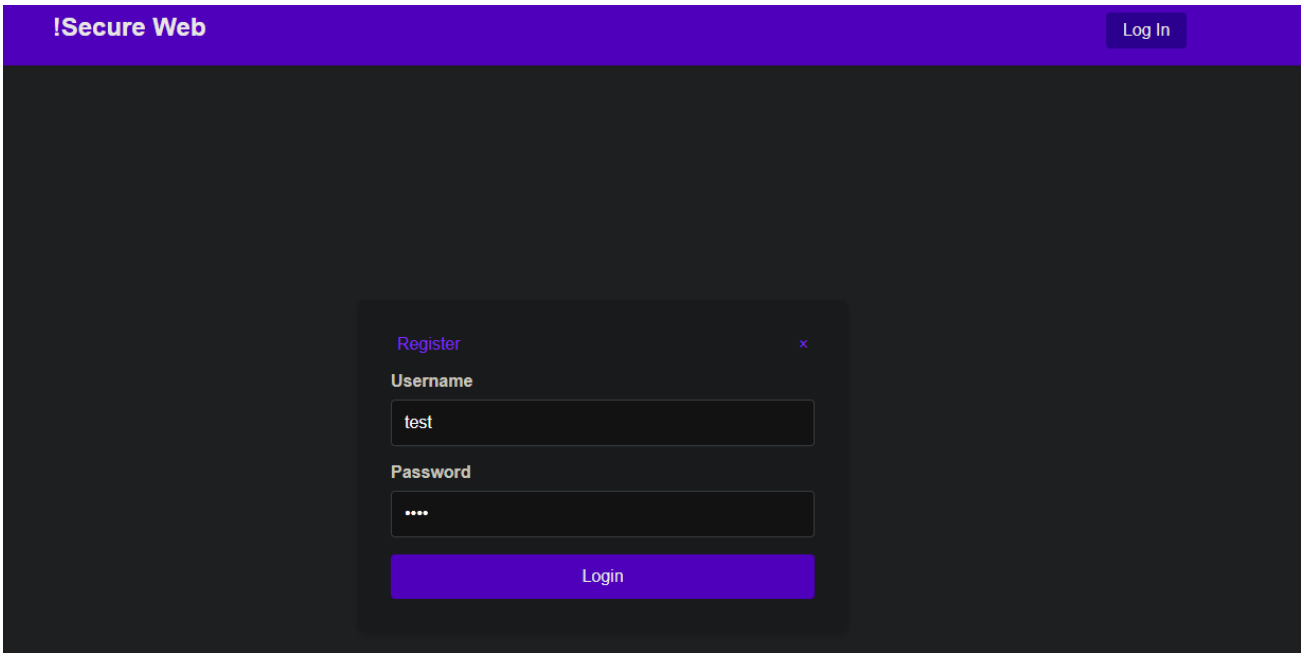
The application worked as follows:

1. A user registers an account and logs in.
2. After logging in, a **cookie is assigned** in the format:

   makefile

   CopyEdit

   `Cookie: username:cookie-value`
3. When performing authentication, the app **only checks if the username in the cookie matches the logged-in username** but does not validate the cookie value.
4. The OTP system was **hardcoded**, meaning any user could authenticate using the same OTP.
5. Upon successful OTP entry, the app **redirects to a URL containing the user's UID**.



1. Registering and Logging In

- I registered an account with **test:test** as the credentials.
- Upon logging in, I observed how the application handled authentication.

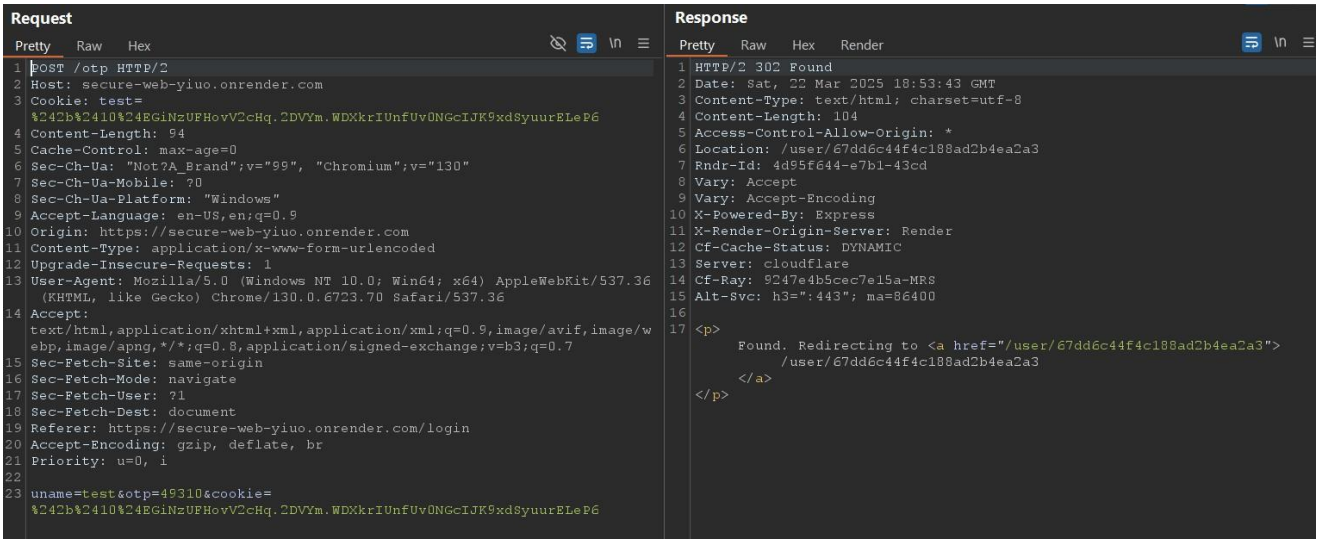
2. Understanding the Cookie Mechanism

- The app assigned cookies in the following format:

  `Cookie: username: cookie-value`
- When validating the session, the app **only checked the username** but not the cookie value.

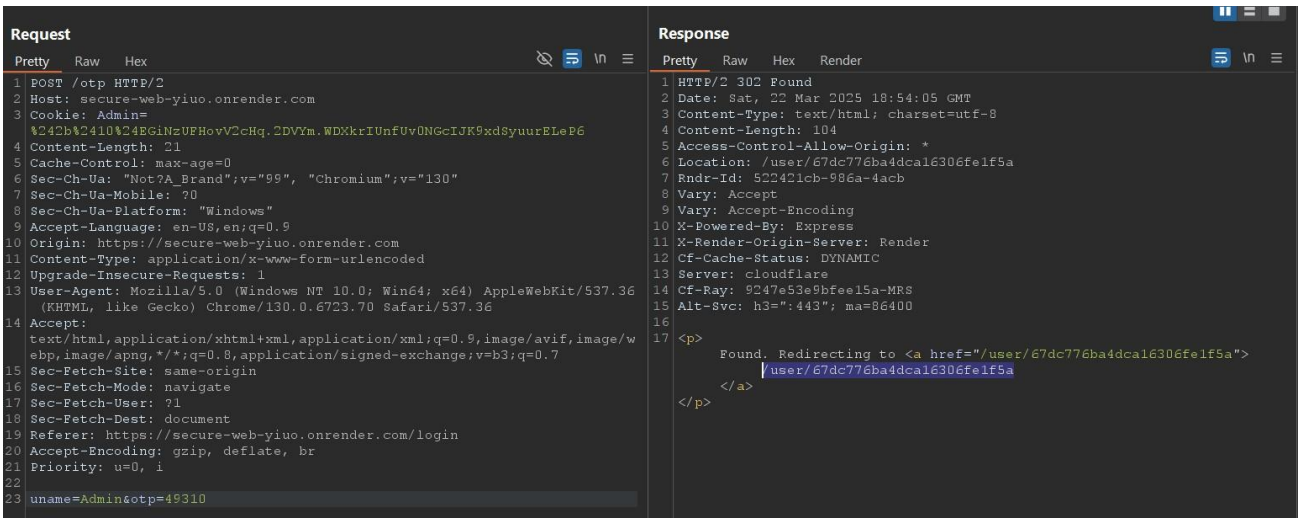
3. Bypassing OTP Authentication

- The OTP was **hardcoded** and worked for all accounts.
- Submitting the OTP returned a response with a redirect containing the **user's UID**.



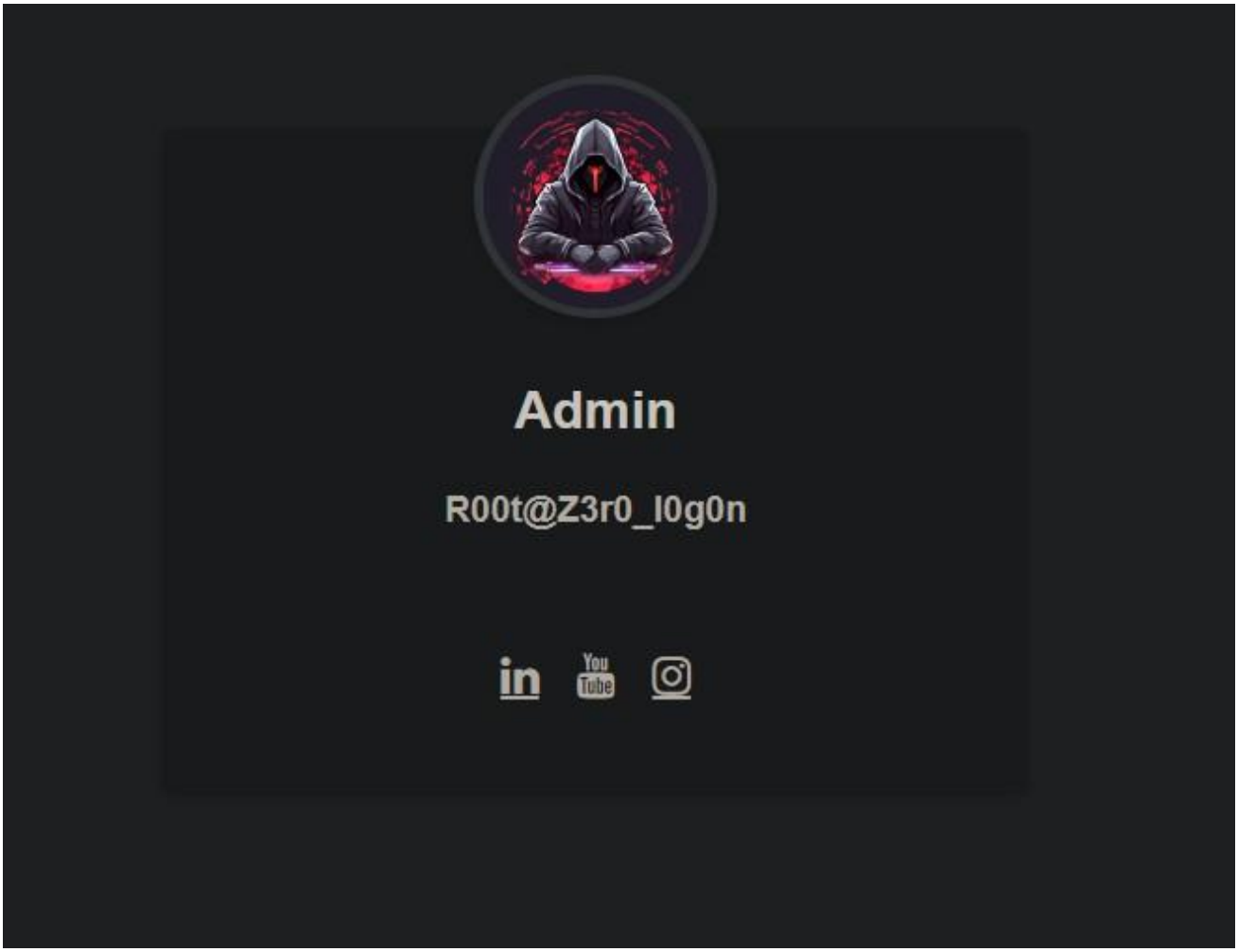
4. Exploiting the Cookie Validation Flaw

- I modified the **username in the cookie** to `Admin` and logged in as `Admin`.
- This revealed the **Admin UID** in the redirect response.



5. Privilege Escalation

- By changing my **UID in the URL** to match the **Admin's UID**, I successfully accessed the Admin account.



6. Finding the Flag

- Viewing the page source of the admin panel revealed the **flag**.