Solomon's Hand - **Contents**

1. All you can overflow (200pts)

Tools needed

- strings

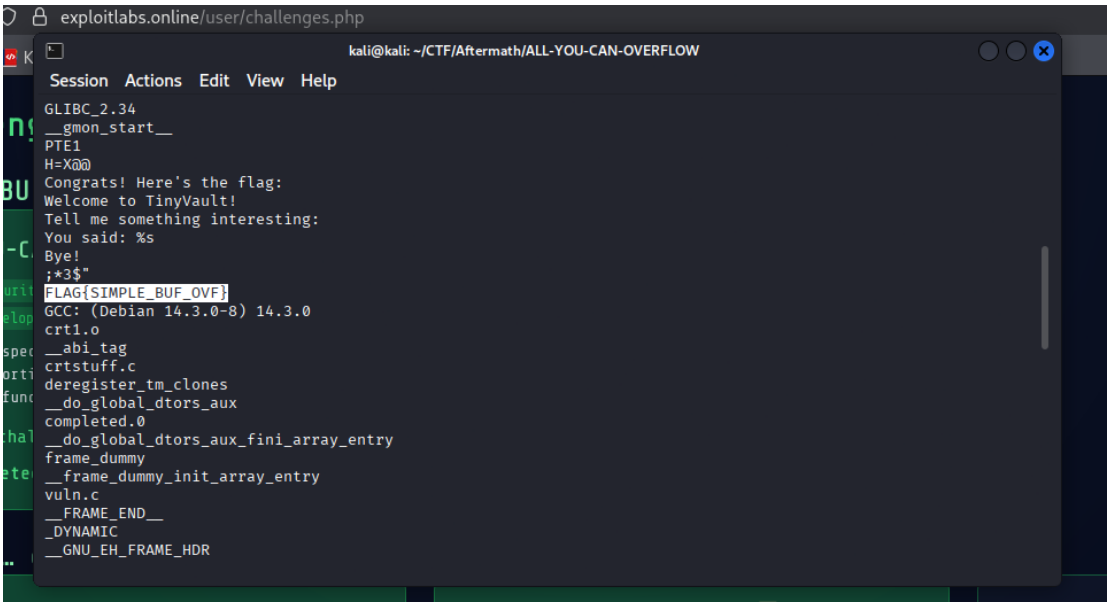
1) The challenge implies a buffer overflow. First let's run the executable and see it's behavior.

chmod +x vuln

./vuln

2) It asks for an input and closes. So the challenge is directing us for a buffer overflow. However, the flag could be a plain string. So let's check the strings of the file and see if there is a flag.

trings vuln

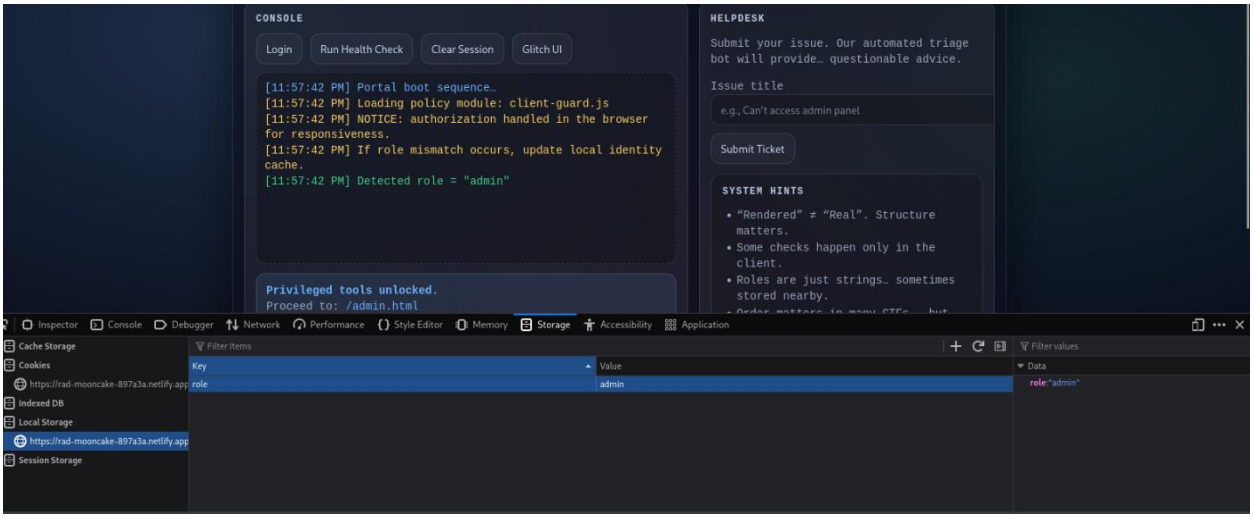


3) Indeed the flag is there and completely bypassed the buffer overflow step.

2. When a browser Decide who you are (150 pts)

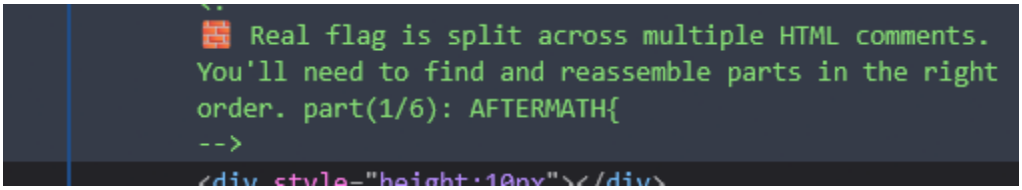
When you go to the challenge website it speaks about roles. More often it is done with sessions and local storage. Check them to see what it reveals

2) Indeed it is localstorage. Change it to user and it should update the website. Then head to /admin.html for the flag



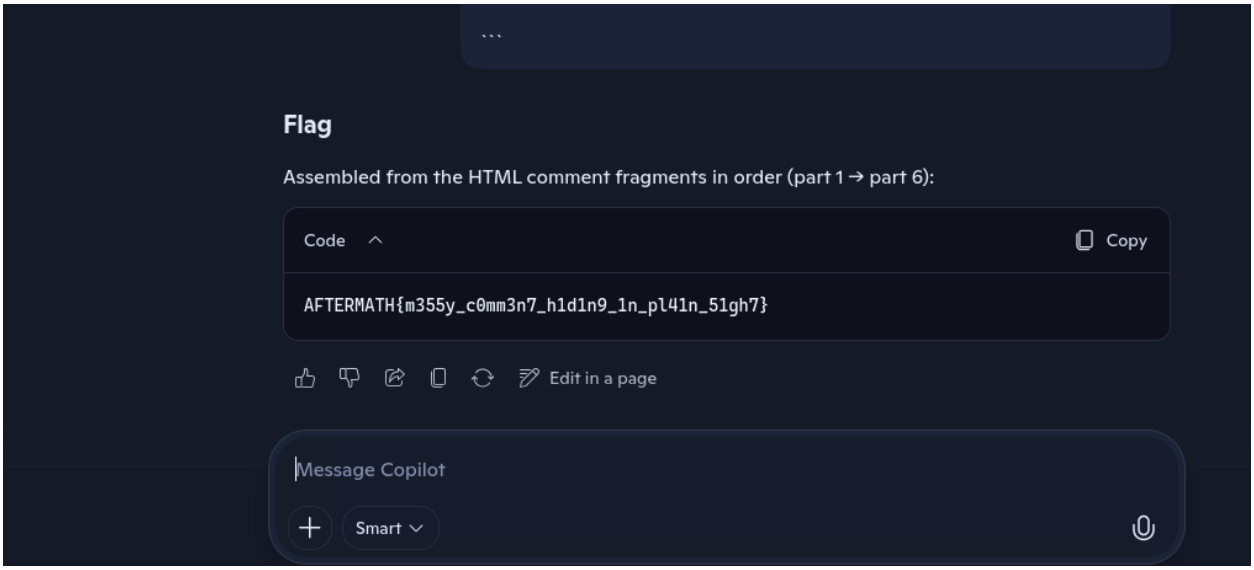
3. The Illusion of Order (70 pts)

1. For this challenge you are given the source code of the website
2. Scouring through the code you find lots of random garbage but also parts of the flag scattered about

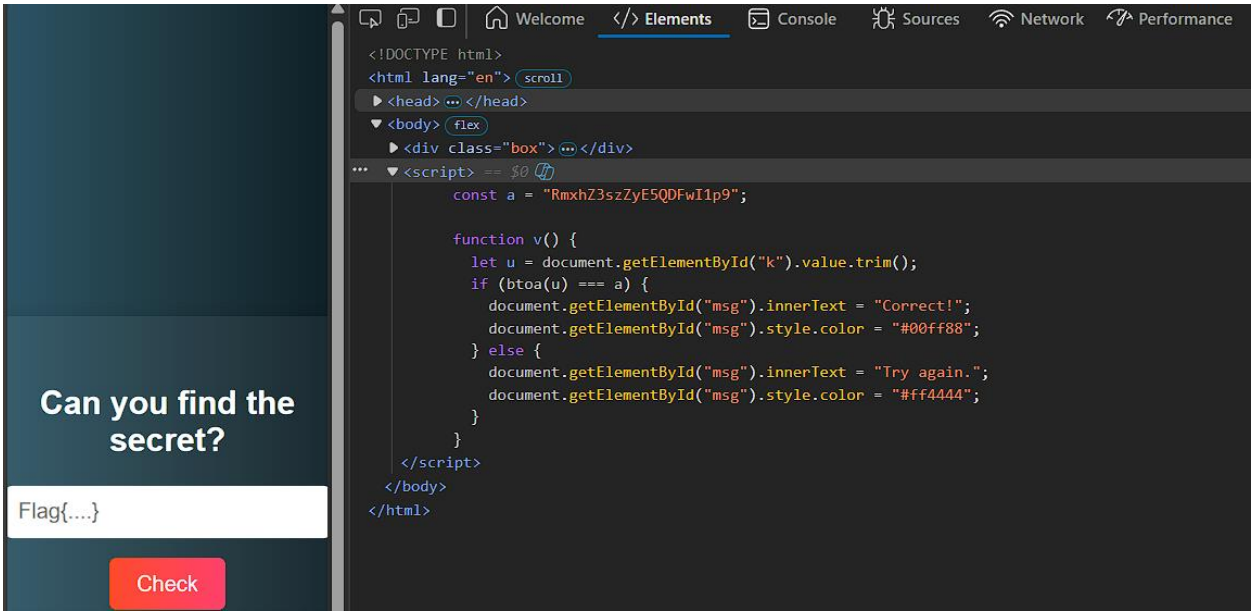


Putting together these parts gives us the full flag

2) Paste the source code into you preferred AI tool and get the flag.

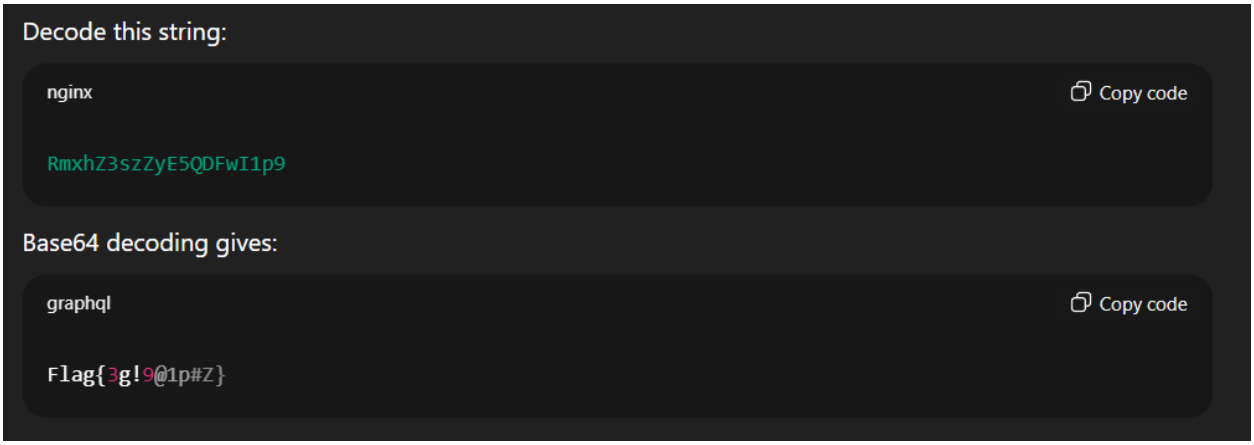


4. Key in the Shadows (50 pts)



First I have went to the inspect mode iof the given html file and there has been a line added where it has mentioned a const a with an encrypted file and then later on it seemed to be a base64 code

And afterputting it to a decoder ive gotten



The insert bar indicates that the way the flag should be uploaded is Flag{code} hence the 3g!9@p#Z}

That has been captured as the flag.

5. Admin's Lazy Day (30pts)

https://glittering-gaufre-a43545.netlify.app/

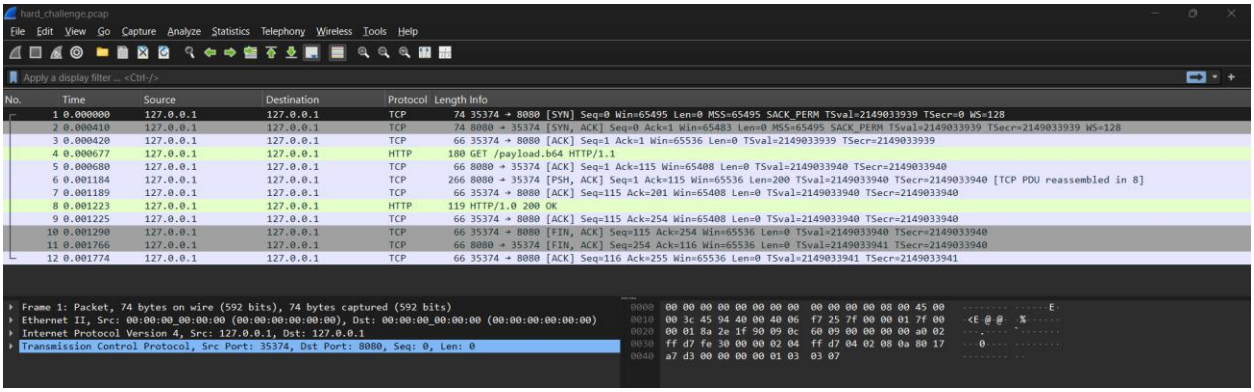
As the first step I have visited this website and the inspection tool was blocked, the description indicated that the flag is hidden in the source code (As they mentioned lazy) ,

I have clicked, **SHIFT+U** to get the source code and in the last bit, the flag was visible.

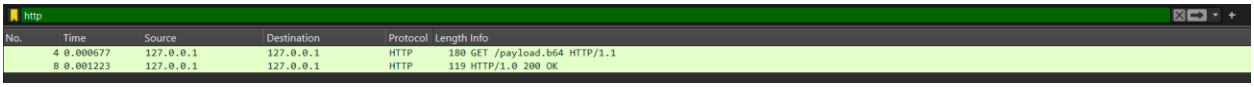
```
        <label for="password">Password:</label>
        <input type="password" id="password" placeholder="Enter password" required>

        <input type="submit" value="Enter the Gates">
    </form>

    <div class="flag" id="flag">CTF{sql_injection_success}</div>
</div>

<script>
    document.getElementById('loginForm').addEventListener('submit', function(e) {
        e.preventDefault();
        const username = document.getElementById('username').value.trim();
        const password = document.getElementById('password').value.trim();
```

6. Deep packet secrets (100pts)

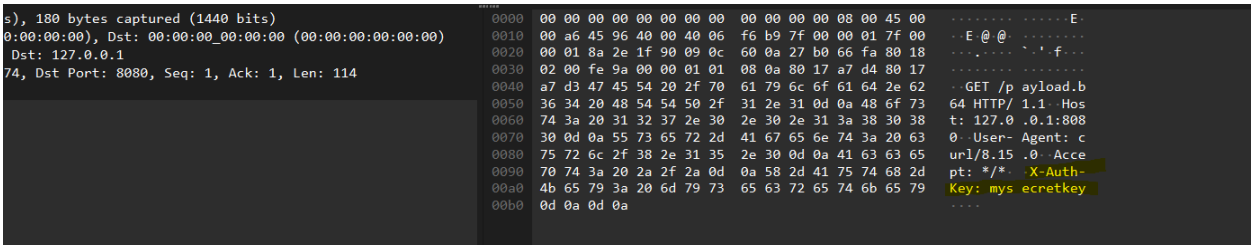
First, I have opened the wireshark file revealing this,



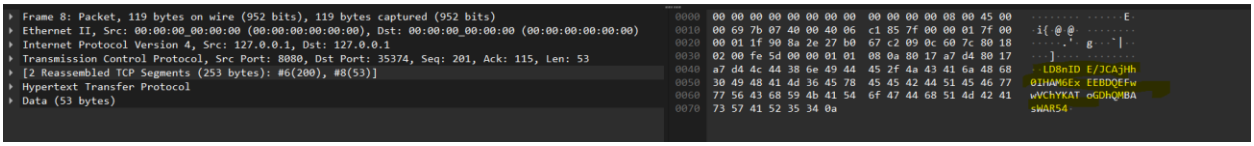
This challenge hides data in **application-layer traffic**, so we narrow it down. The reason being The challenge hint says *local transfer*



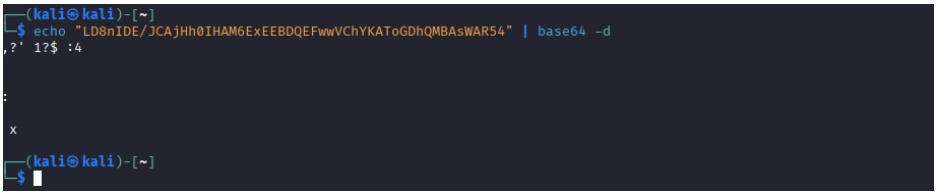
And once I clicked it and it expanded the request headers, it shows the a x-auth-key being **"mysecretkey".**



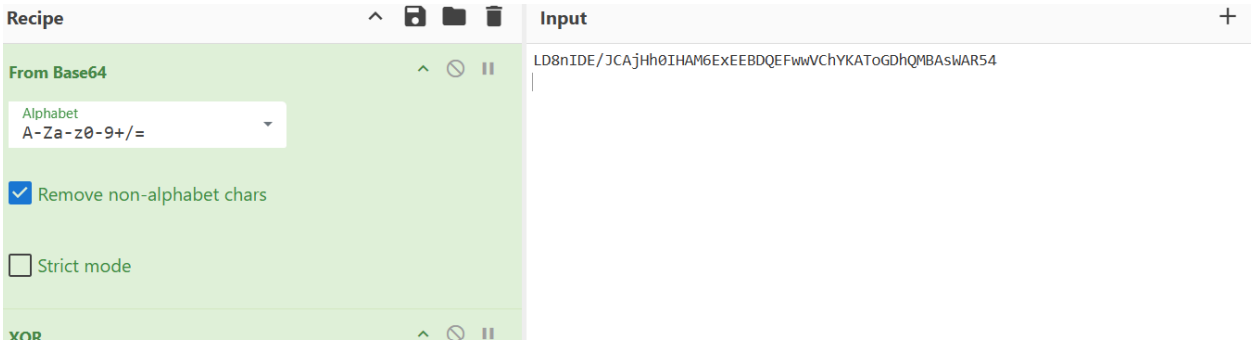
With that noted down we have moved to the next packet revealing,



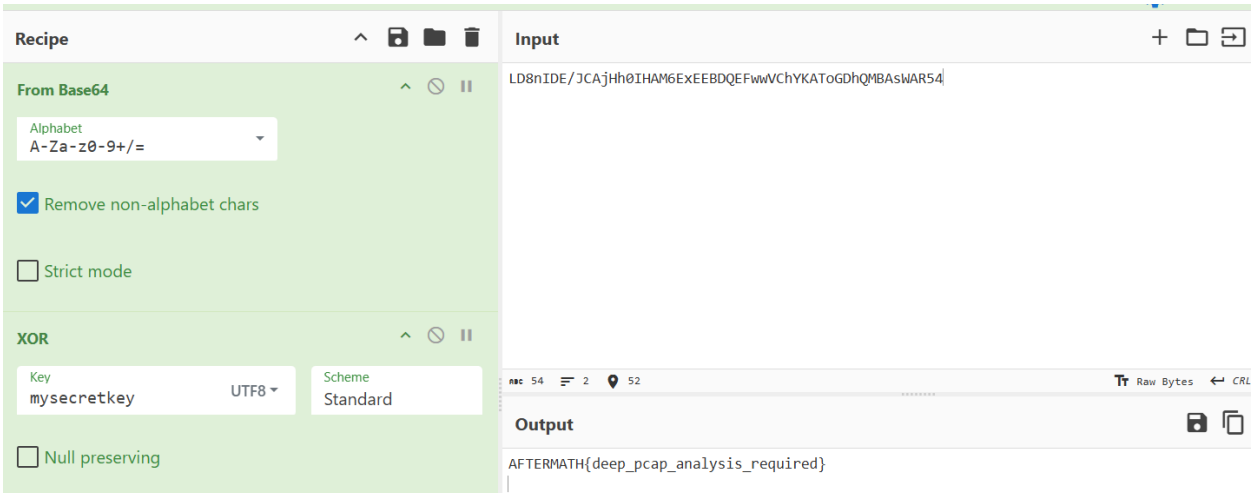
An encrypted message, which seems to be a base64 encrypted message and when it has been decoded, it happens to look unreadable



Which happens to be a good sign because that have proven its only one layer secured,and I have started to decrypt it from Base64

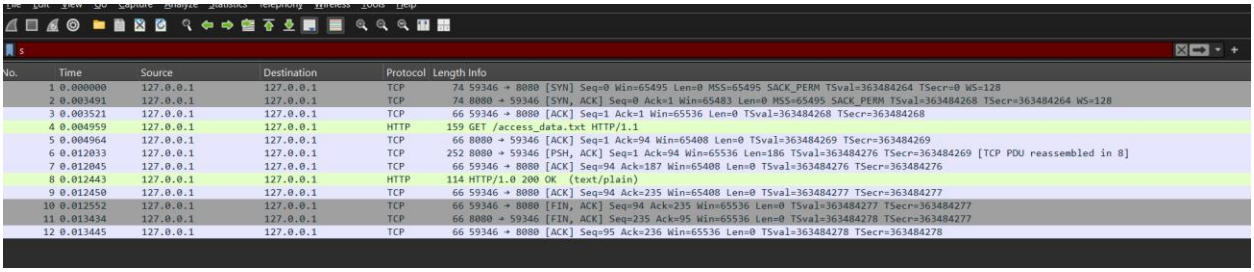


Then as I have found a code security code, I have added an xor panel thus adding the security key there



Which revealed the flag.

7. Whispers in the Stream (30pts)



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	74	59346 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=363484264 TSecr=0 WS=128
2	0.003491	127.0.0.1	127.0.0.1	TCP	74	8080 → 59346 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=363484268 TSecr=363484264 WS=128
3	0.003521	127.0.0.1	127.0.0.1	TCP	66	59346 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=363484268 TSecr=363484268
4	0.004959	127.0.0.1	127.0.0.1	HTTP	159	GET /access_data.txt HTTP/1.1
5	0.004964	127.0.0.1	127.0.0.1	TCP	66	8080 → 59346 [ACK] Seq=1 Ack=94 Win=65408 Len=0 TSval=363484269 TSecr=363484269
6	0.012033	127.0.0.1	127.0.0.1	TCP	252	8080 → 59346 [PSH, ACK] Seq=1 Ack=94 Win=65536 Len=186 TSval=363484276 TSecr=363484269 [TCP PDU reassembled in 8]
7	0.012045	127.0.0.1	127.0.0.1	TCP	66	59346 → 8080 [ACK] Seq=94 Ack=187 Win=65408 Len=0 TSval=363484276 TSecr=363484276
8	0.012443	127.0.0.1	127.0.0.1	HTTP	114	HTTP/1.0 200 OK (text/plain)
9	0.012450	127.0.0.1	127.0.0.1	TCP	66	59346 → 8080 [ACK] Seq=94 Ack=235 Win=65408 Len=0 TSval=363484277 TSecr=363484277
10	0.012552	127.0.0.1	127.0.0.1	TCP	66	59346 → 8080 [FIN, ACK] Seq=94 Ack=235 Win=65536 Len=0 TSval=363484277 TSecr=363484277
11	0.013434	127.0.0.1	127.0.0.1	TCP	66	8080 → 59346 [FIN, ACK] Seq=235 Ack=95 Win=65536 Len=0 TSval=363484278 TSecr=363484277
12	0.013445	127.0.0.1	127.0.0.1	TCP	66	59346 → 8080 [ACK] Seq=95 Ack=236 Win=65536 Len=0 TSval=363484278 TSecr=363484278

First, I have opened the **easy.pcap** file with **wireshark**.

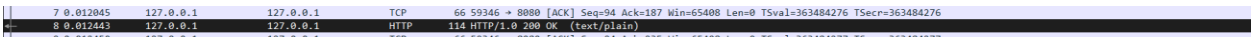
And as the hint indicates,

“Even the loudest chatter can hide secrets”,*“Somewhere in the flow of data”* Hints us that the packet is hidden inside the packet payloads. Hence we focus on streams, but not headers.

Then in order to find the packet files containing the flag I have clicked **Ctrl + F** and the small header pops up

I have set the Find by to **String** and the search in **Packet Bytes** and the character has been set to **ASCII** and then searched for **{**

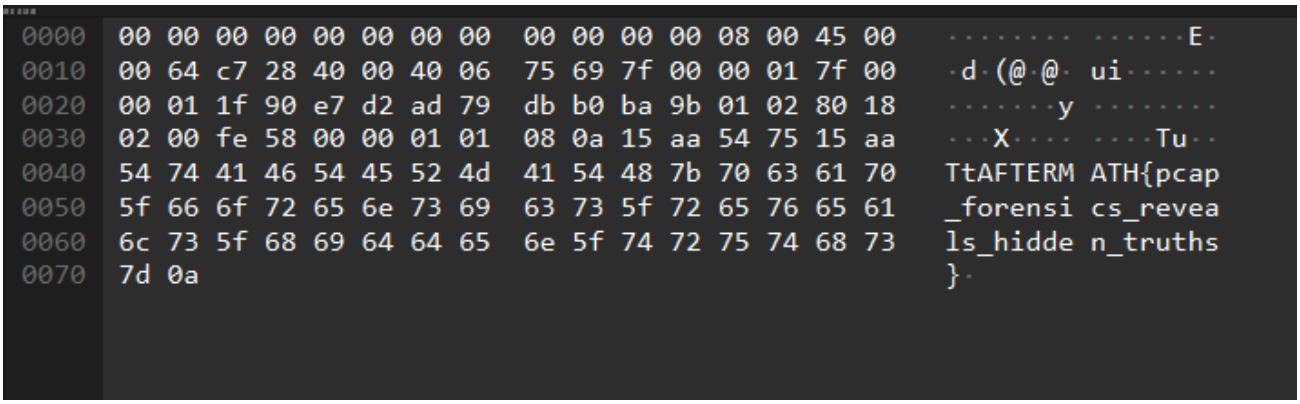
Once the find button is clicked,



7	0.012045	127.0.0.1	127.0.0.1	TCP	66 59346 → 8080 [ACK] Seq=94 Ack=187 Win=65408 Len=0 TSval=363484276 TSecr=363484276
8	0.012443	127.0.0.1	127.0.0.1	HTTP	114 HTTP/1.0 200 OK (text/plain)

It has opened this file which indicates the flag is inside this

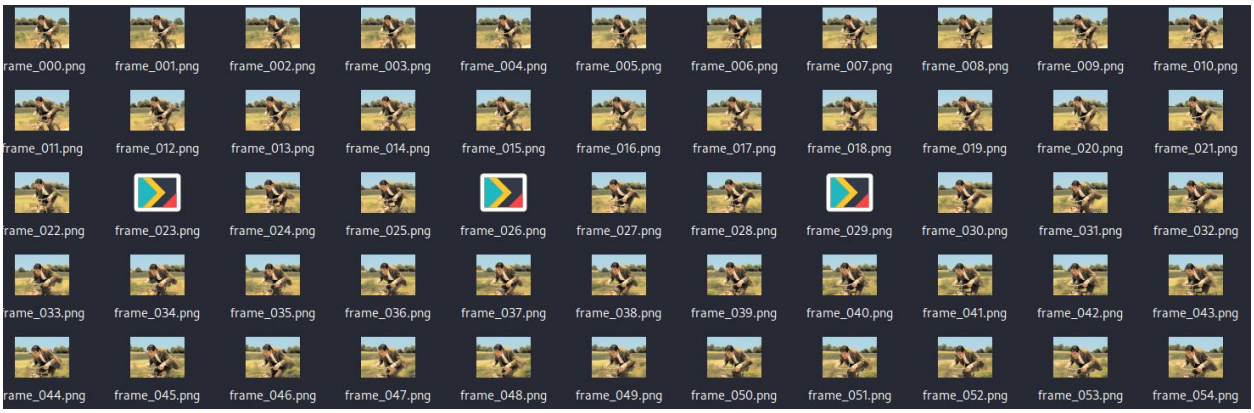
Once it has been checked it has revealed the flag as this



```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  00 64 c7 28 40 00 40 06  75 69 7f 00 00 01 7f 00   ·d·(@·@· ui·····
0020  00 01 1f 90 e7 d2 ad 79  db b0 ba 9b 01 02 80 18   ·······y ········
0030  02 00 fe 58 00 00 01 01  08 0a 15 aa 54 75 15 aa   ···X···· ····Tu··
0040  54 74 41 46 54 45 52 4d  41 54 48 7b 70 63 61 70   TtAFTERM ATH{pcap
0050  5f 66 6f 72 65 6e 73 69  63 73 5f 72 65 76 65 61   _forensi cs_revea
0060  6c 73 5f 68 69 64 64 65  6e 5f 74 72 75 74 68 73   ls_hidde n_truths
0070  7d 0a                                             }·
```

8. Frame by Frame, Color by Color (200pts)

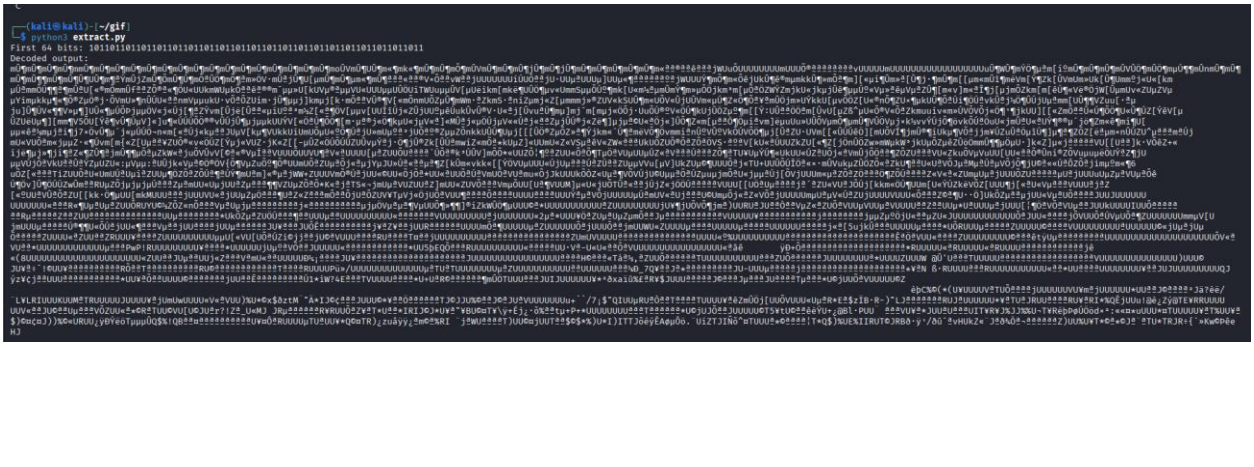
First I have divded the the gif into frames,



Then I have tried to make a python file to crack the information

```
>>>
>>> gif = Image.open("challenge.gif")
>>> gif.seek(0)
>>>
>>> pixels = list(gif.getdata())
>>> pixels[:20]
[121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121, 121]
>>>
```

Which has led me to this



9. Fragments of a Lost Image (70pts)

Tools needed

- **base64**
- **binwalk**
- **zip2john**
- **john**

1) Download the file and analyze it. The characters seem oddly familiar to some code of an encoding. It is unusual for large files to be encoded so it could also be an image. This is also evidenced by the header. Run the following command to attempt to decode it. for rapid identification and AI tool can help with it.

base64 -d <file.txt> > mystery.jpg

2) After running some stegtools it didn't reveal much so let's try binwalk to see if there's any hidden files

binwalk -e mystery.jpg

```
1159  base64 -d encoded-20251219165134.txt > mystery.jpg
1160  ls
1161  thunar
1162  binwalk -e mystery.jpg
1163  ls
```

3) Indeed there is a zip file. Attempting to open it shows that it is password protected. We can solve that by using john the ripper. First convert it to a hash and run a bruteforce to reveal the password.

zip2john 260D.zip > hash.txt

john hash.txt <wordlist>

4) The password is revealed as ```**welcome123**```. Unzip the contents and you'll be presented with two text files.

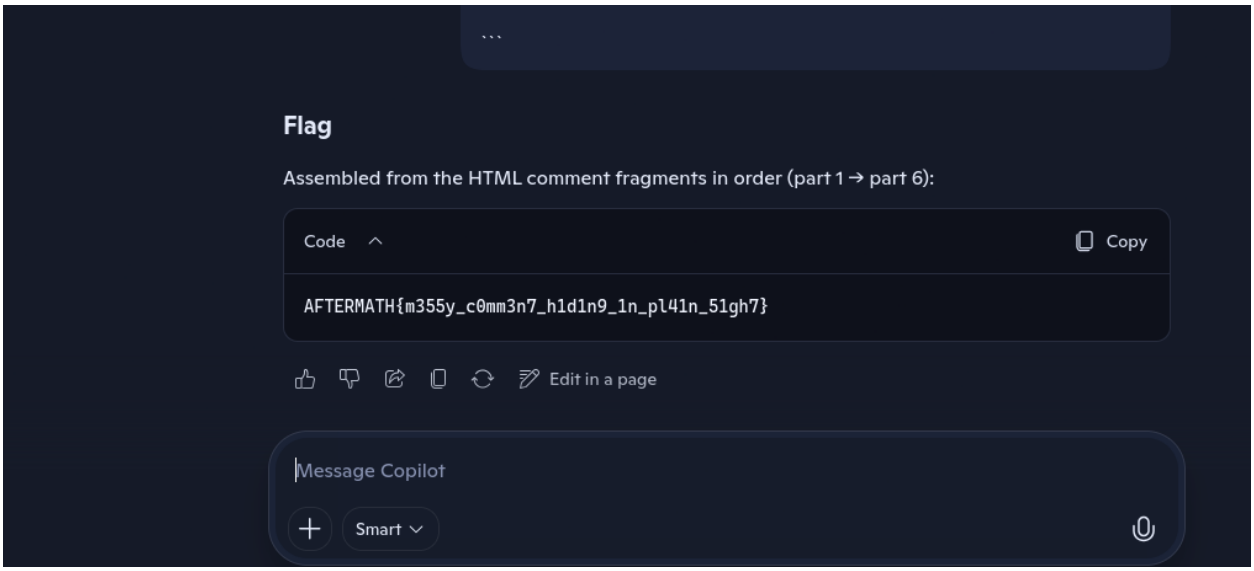
5) Reveal the contents of ```**hidden.txt**``` from my past experience, it appears to be brainfuck code. Copy the contents and paste it into an online brainfuck interpreter, ```**copy.sh/brainfuck**```

```
┌──(kali㉿kali)-[~/CTF/Aftermath/fragmentsoflostimage/_mystery.jpg.extracted]
└─$ zip2john 26D6.zip > hash.txt
john hash.txt --wordlist=/usr/share/wordlists/rockyou.txt

ver 1.0 efh 5455 efh 7875 26D6.zip/next_step.txt PKZIP Encr: 2b chk, TS_chk, cmplen=54, decmplen=42, crc=60
D63F19 ts=B185 cs=b185 type=0
ver 2.0 efh 5455 efh 7875 26D6.zip/hidden.txt PKZIP Encr: TS_chk, cmplen=103, decmplen=381, crc=A33B0469 ts
=B215 cs=b215 type=8
NOTE: It is assumed that all files in each archive have the same password.
If that is not the case, the hash may be uncrackable. To avoid this, use
option -o to pick a file at a time.
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
welcome123       (26D6.zip)
1g 0:00:00:00 DONE (2025-12-21 01:29) 33.33g/s 546133p/s 546133c/s 546133C/s havana..cocoliso
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Illusion of Order

1) For this challenge you're given the sourcecode of the website. AI tools is best for this challenge because it can recognize patterns better than us.

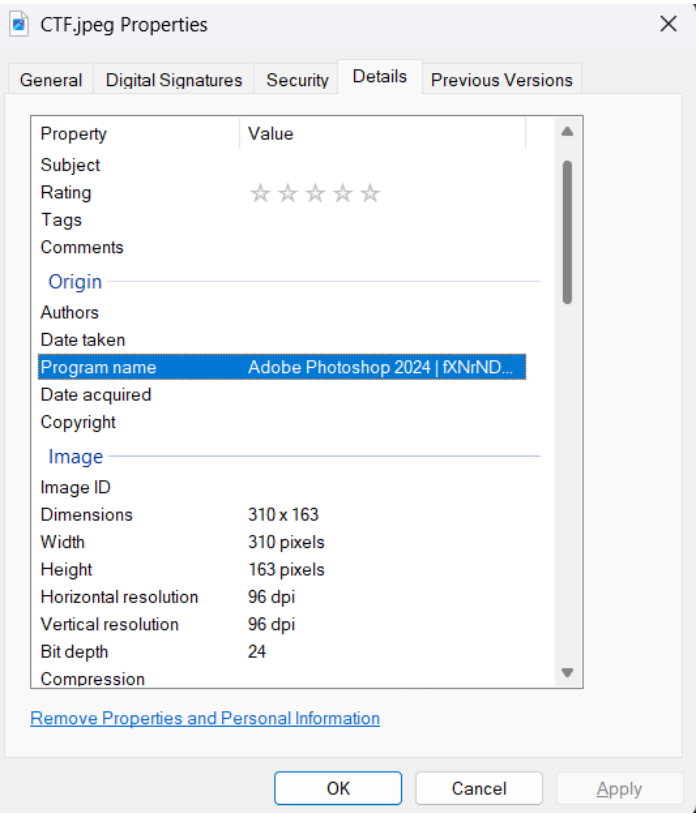
2) Paste the source code into you preferred AI tool and get the flag.



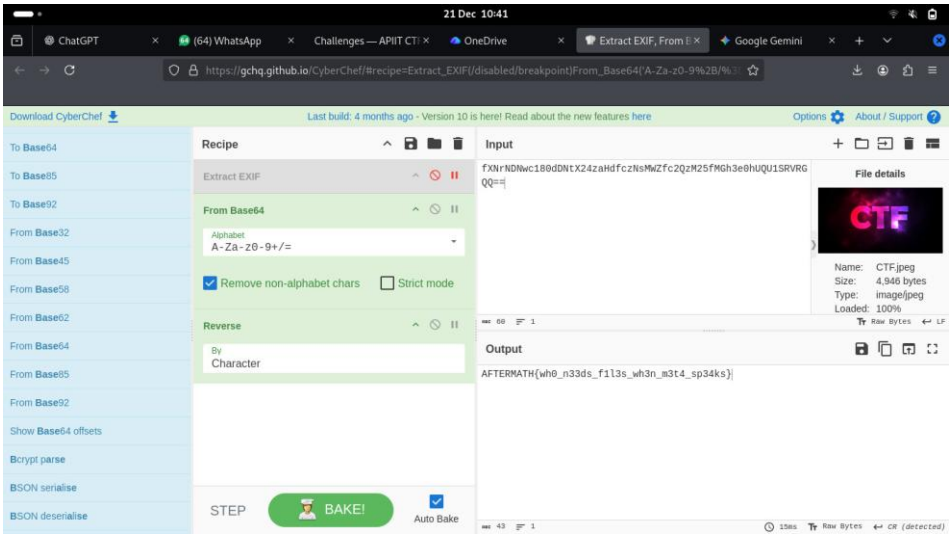
10. Echoes After the AfterMath (50pts)

For this challenge we are given an image, by instinct we check the metadata first.

Here we find a base64 Encoded string in the program name field



3.Simply converting the base64 back gives us the flag



11. Nothing to See Here (90 pts)

Tools used:

- Strings
- objdump

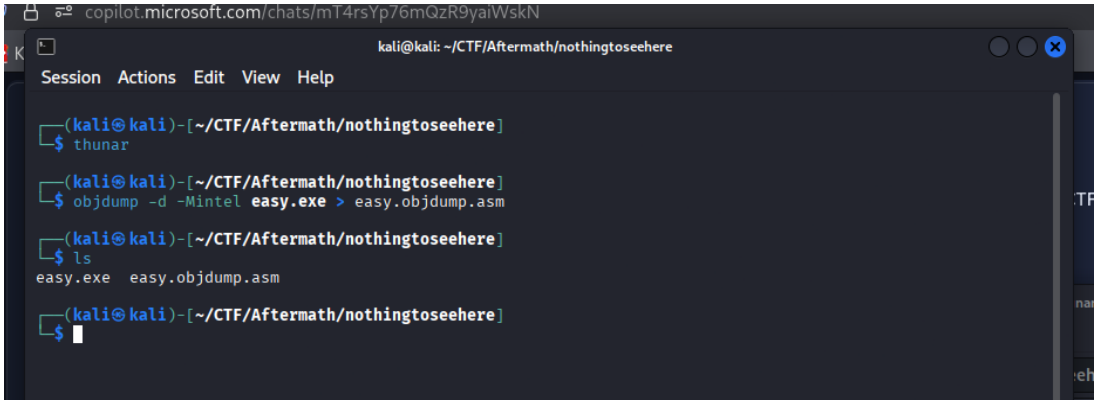
1) It's a windows executable, first let's try strings to see if the flag is there like the last challenge

strings easy.exe | grep FLAG

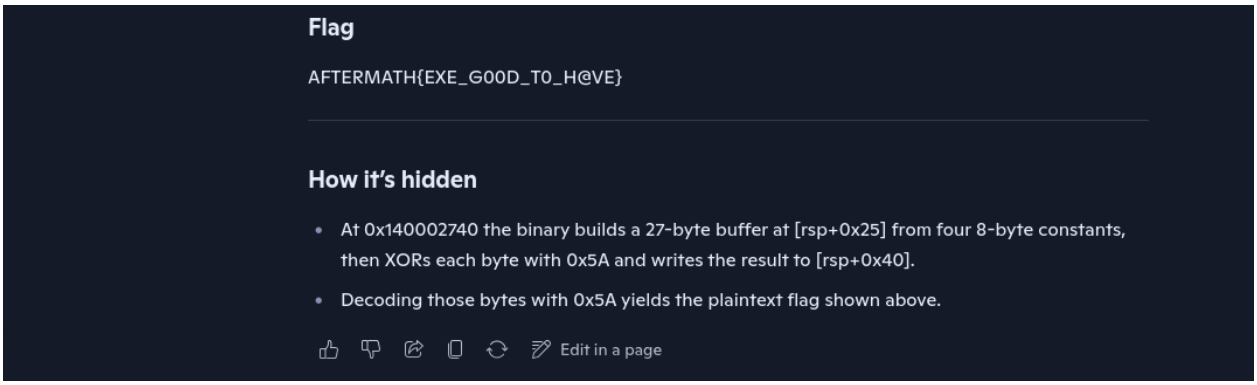
strings easy.exe | grep AFTERMATH

2) No luck, then we must get the decompiled code of it. And run it through an AI tool to identify the flag. Leveraging AI is the best because it can detect patterns and decode complex code.

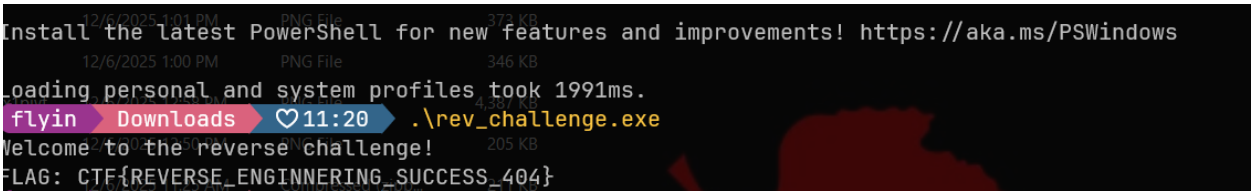
objdump -d -Mintel easy.exe > easy.objdump.asm



3) Paste it into your preferred AI tool and you should get the flag



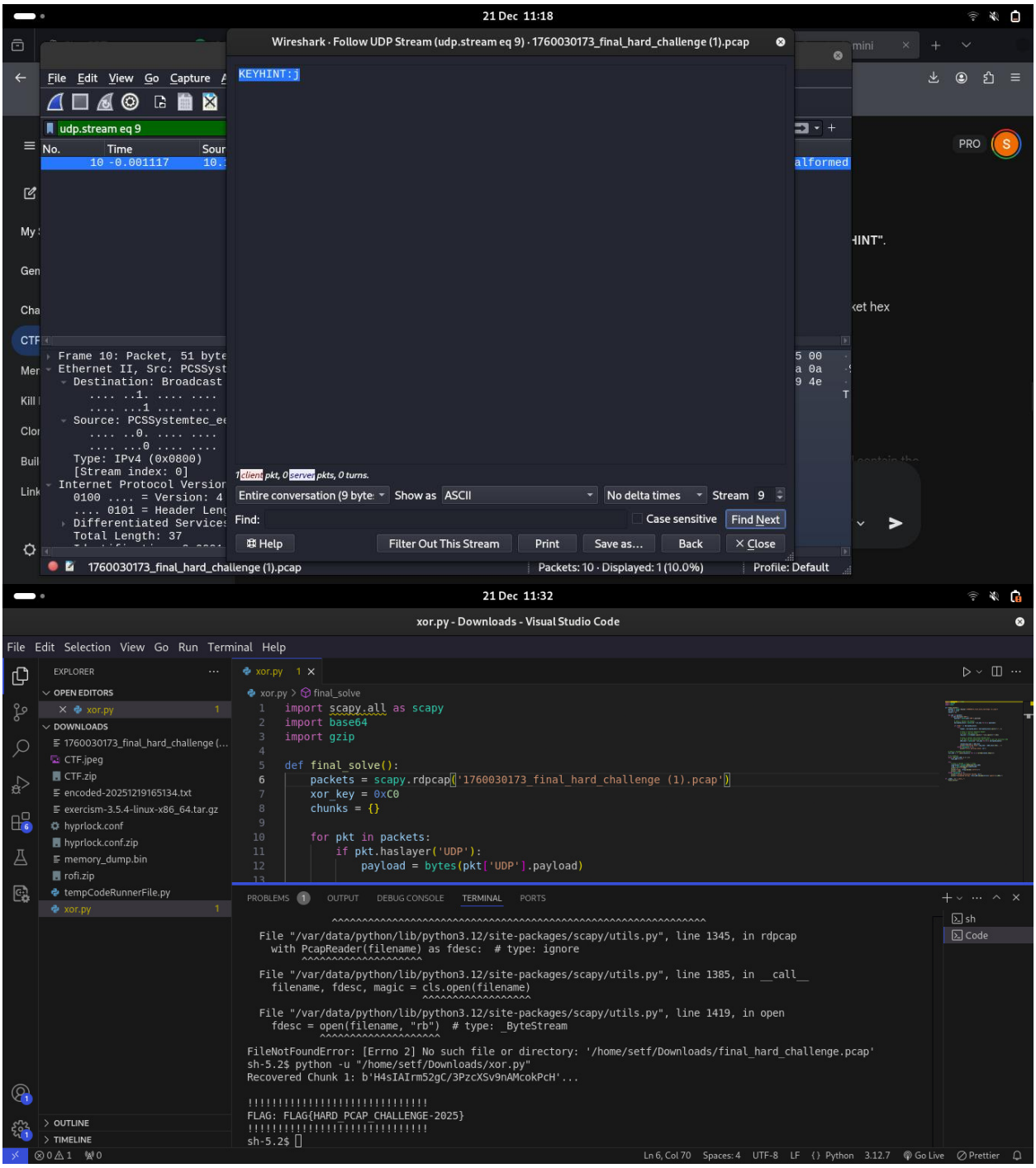
12. Foreing Execution (40 pts)



The challenge description talks about foreign environments and talks about *“if you have the right wine to sip* 🍷🧩*.”* referring to Wine on linux which you can use to run windows native applications.

However since I was on windows i simply ran the program normally which gave me the flag

13. The XOR Files: A Tale of Obfuscation (250 pts)



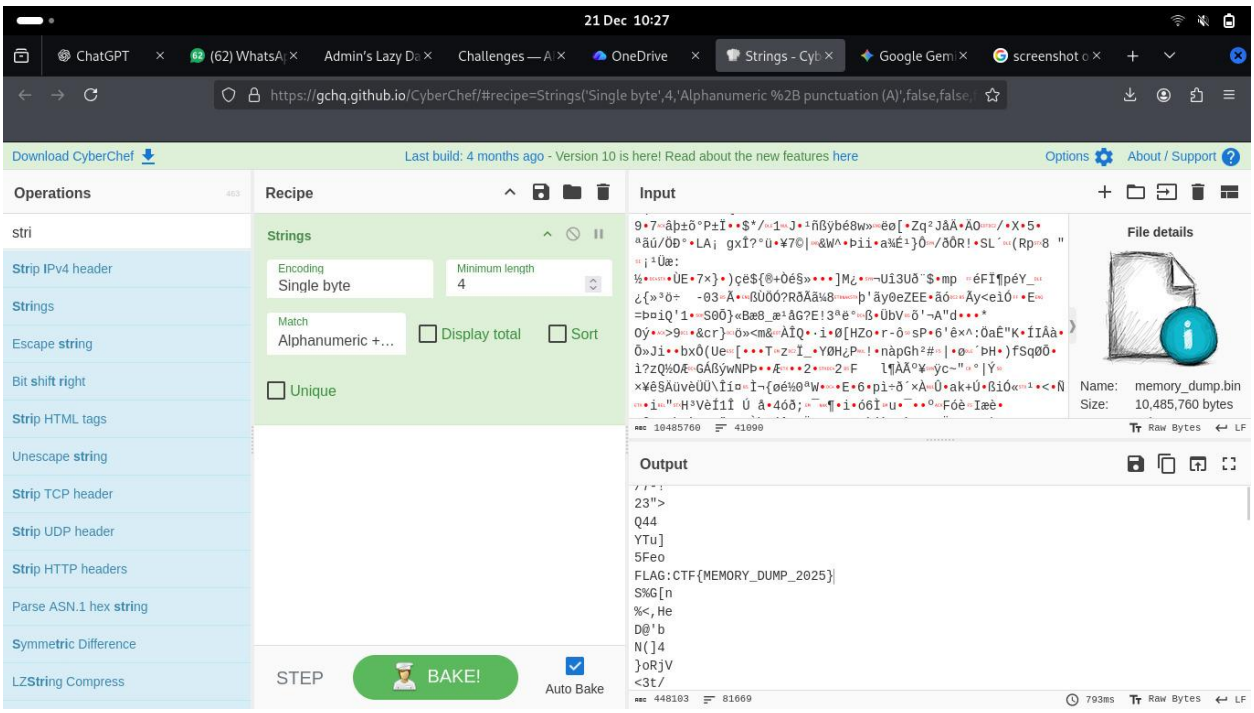
Python Code:

```
import scapy.all as scapy
import base64
import gzip

def final_solve():
```

```
    packets = scapy.rdpcap('1760030173_final_hard_challenge (1).pcap')
    xor_key = 0xC0
    chunks = {}

    for pkt in packets:
        if pkt.haslayer('UDP'):
            payload = bytes(pkt['UDP'].payload)

            # Step 1: Reveal the Header
            decrypted_block = bytes([b ^ xor_key for b in payload])

            if b"IDX:" in decrypted_block:
                try:
                    header, encrypted_data = decrypted_block.split(b"|", 1)

                    # Step 2: Extract Sequence Number
                    # Format: IDX:num/total
                    seq_num = int(header.split(b":")[1].split(b"/")[0])

                    # Step 3: Reveal the actual Base64 text
                    # We XOR the encrypted_data part again to get the
plaintext B64
                    b64_text = bytes([b ^ xor_key for b in encrypted_data])

                    chunks[seq_num] = b64_text
                    print(f"Recovered Chunk {seq_num}: {b64_text[:30]}...")
                except Exception as e:
                    print(f"Error parsing chunk: {e}")

    # Step 4: Assemble and Process
    full_b64 = b"".join(chunks[i] for i in sorted(chunks.keys()))

    # Fix Padding
    while len(full_b64) % 4 != 0:
        full_b64 += b"="

    try:
        # Step 5: Base64 -> Gzip -> Flag
        compressed = base64.b64decode(full_b64)
        flag = gzip.decompress(compressed)
        print("\n" + "!"*30)
        print(f"FLAG: {flag.decode('utf-8')}")
        print("!"*30)
    except Exception as e:
        print(f"\nFinal extraction failed: {e}")
        print(f"Assembled String:
{full_b64.decode(errors='ignore')[:100]}")

if __name__ == "__main__":
    final_solve()
```

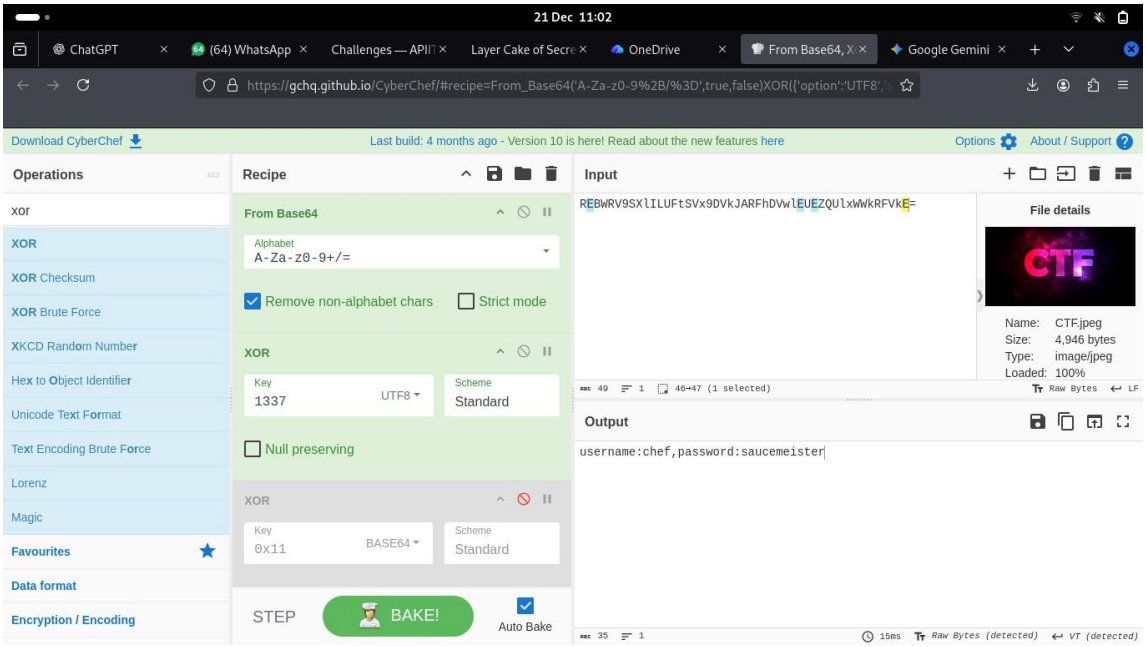
14. Memory Matters (60pts)

First we inspect the binary file for strings

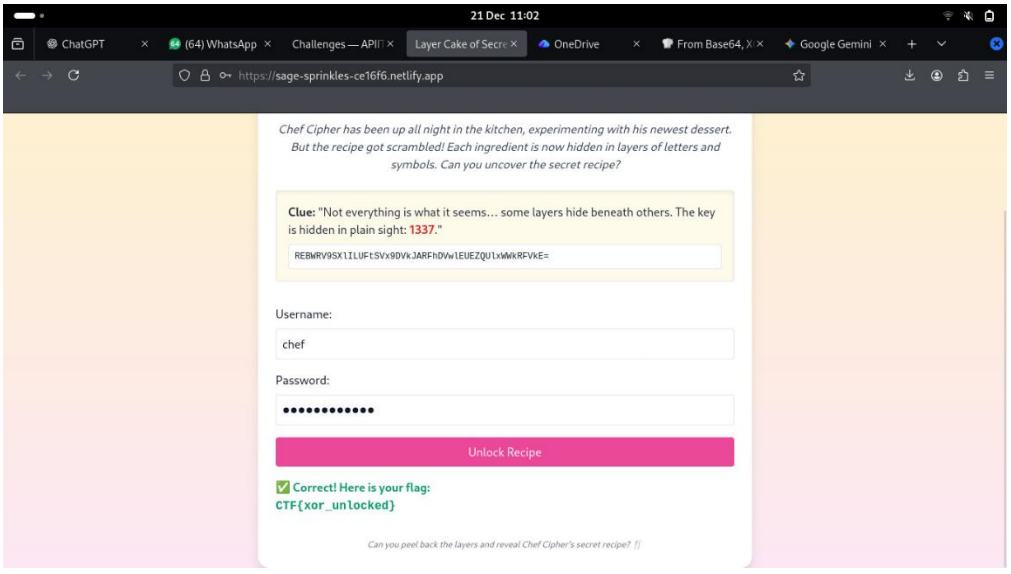


15. Layer Cake of Secrets (60pts)

1. For this challenge the site has give us a key which is 1337 and a base4 string,

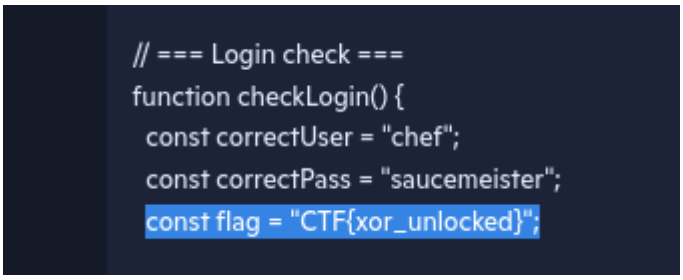


2. Based on the assumption this is a XOR key we can use to decode the base64 string we enter it into a decoder



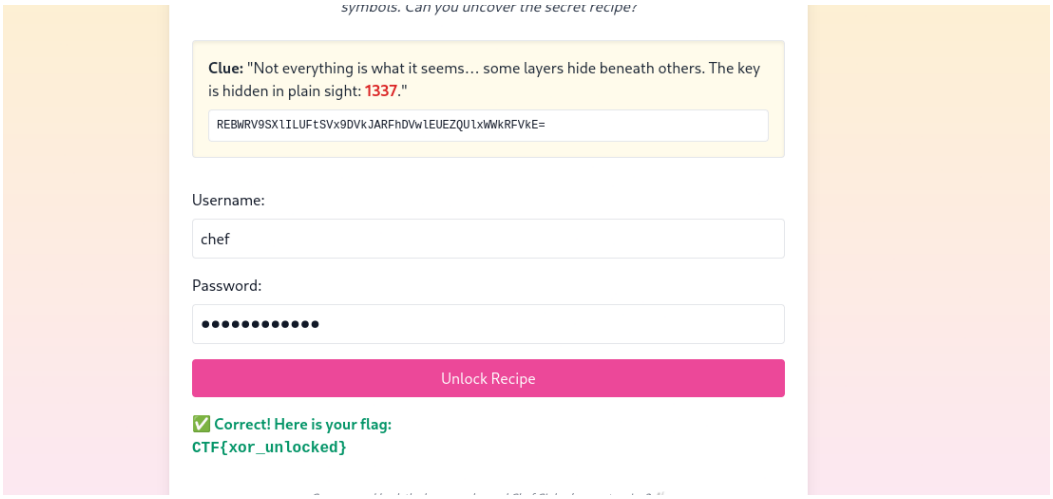
Alternative Approach

1) The site blocks dev tools and right click. We can try bypassing this with an external site to view the source code like ```https://www.view-page-source.com/```



```
// === Login check ===
function checkLogin() {
  const correctUser = "chef";
  const correctPass = "saucemeister";
  const flag = "CTF{xor_unlocked}";
```

2) In the javascript we can see the credentials and the flag. Submit that and we get the flag



16. Login Is Just a Distraction (50 pts)

The first hint that tipped us off was the Domain title being caesar login challenge, referring to the caesar cipher where the letters of a message are shifted by a certain number, the challenge is then finding that exact offset.

Even though the letters are shifted the words retain the same number of letter so the attack pattern was to find commonly grouped words that have the have recognizable lengths and then unshift the bits till the word makes see if the entire message has been decoded

We did this assuming that “td l ” was “is a” and when the message was shifted by exactly 11 the message appeared giving us the username and the password.