Wrote by

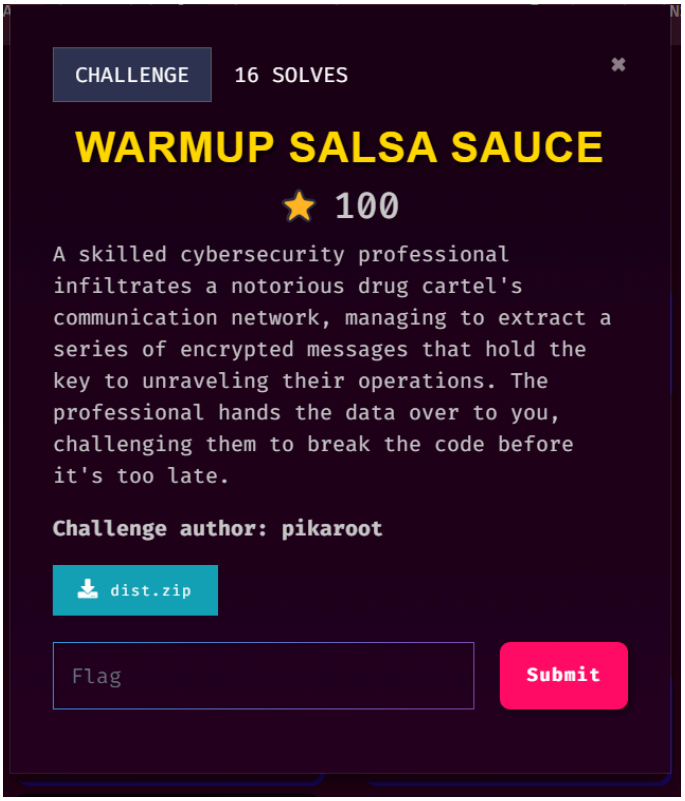
Dark Phoenix

d4rkPho3nix
.geshulin.

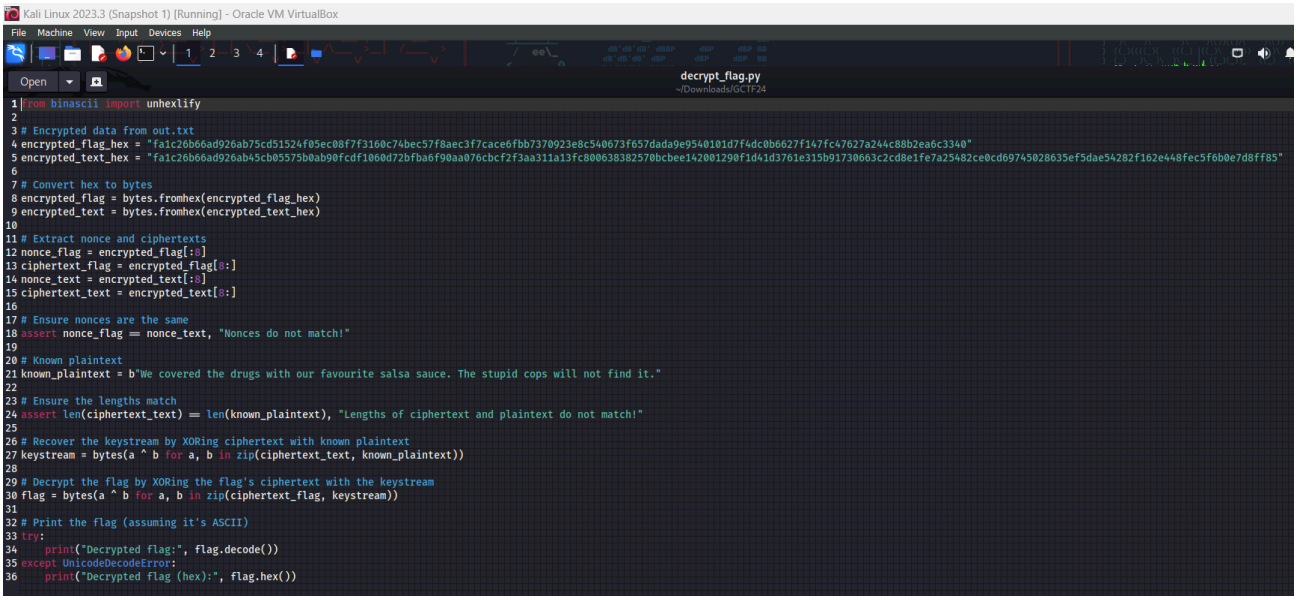
Table of Contents

CRYPTO

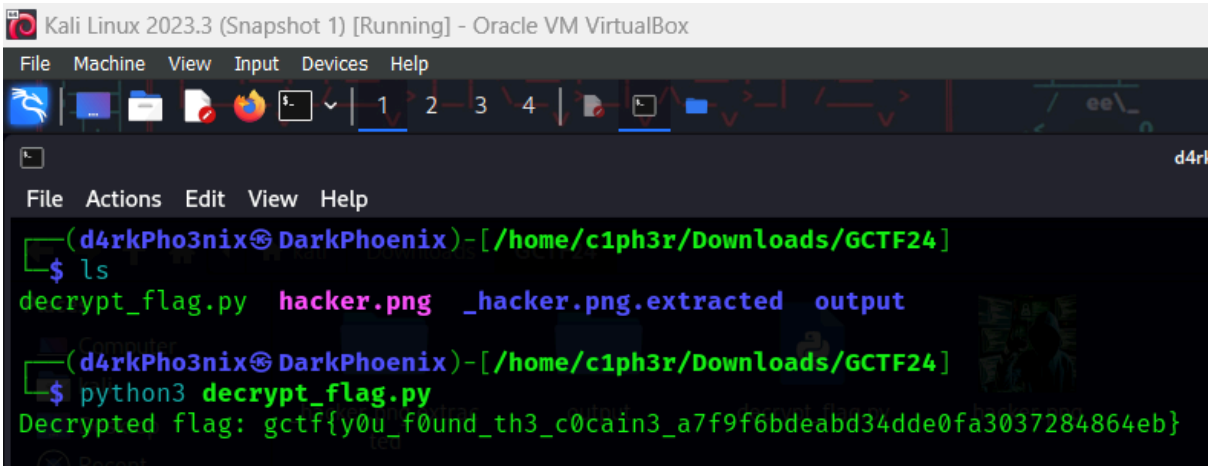
Warmup Salsa Sauce



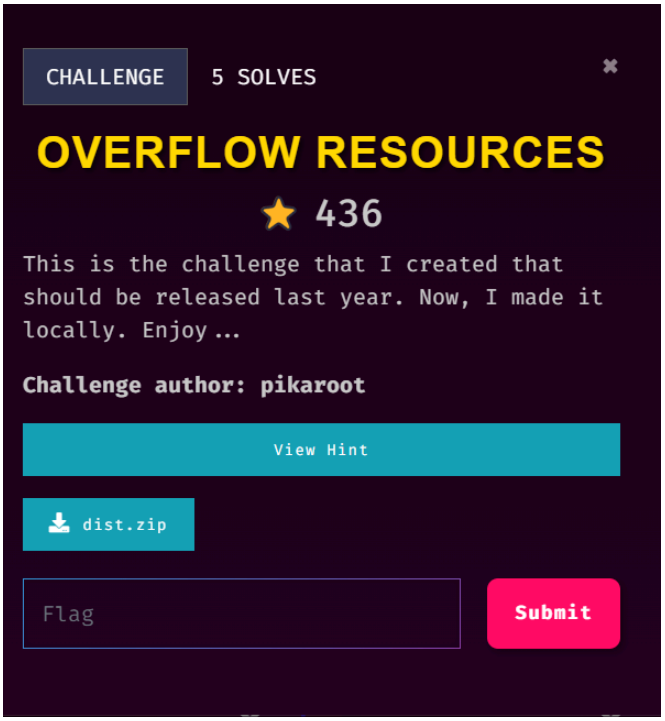
In this challenge, we are given a cryptographic script that encrypts the messages. Here is the Python script we used to solve the challenge and retrieve the flag:



After running the above script, we successfully decrypt the flag.



Overflow Resources



Challenge Overview

In this challenge, we are given a cryptographic script that encrypts two parts of a flag using RSA-like encryption with small exponents. The encryption process randomly selects between two small exponents e = 91 and e = 97, and encrypts either the first part (m_1) or the second part (m_2) of the flag using different RSA moduli (n).

Hints:

- The title of the challenge is "Overflow Resources."
- The hint provided: "I thought it looks Chinese to y'all?" pointed towards an encoding or modular arithmetic concept such as the **Chinese Remainder Theorem (CRT)**, which is common in RSA attacks.

Objective:

We need to recover the full flag by exploiting the small encryption exponents and the multiple ciphertexts generated for the same messages (`m_1` and `m_2`), encrypted under different moduli.

Vulnerabilities

1. **Small Exponents (`e = 91` and `e = 97`)**:
    - The encryption process uses very small values for `e` (91 and 97). Small exponents are vulnerable to certain cryptographic attacks, particularly when the same message is encrypted multiple times with different moduli.
2. **Same Message Encrypted Multiple Times**:
    - The first part of the flag (`m_1`) is encrypted using `e = 91` with different moduli, and the second part (`m_2`) is encrypted using `e = 97`. Since the same message is encrypted with different keys, we can use an attack called **Håstad's Broadcast Attack** to recover the original message.

Attack Explanation

Håstad's Broadcast Attack:

This attack takes advantage of small encryption exponents. When the same message is encrypted using multiple different moduli (`n`) but with the same small exponent (`e`), it is possible to combine the ciphertexts and use the **Chinese Remainder Theorem (CRT)** to recover the original message without needing to factor the moduli.

Steps to Solve the Challenge:

1. **Extract Ciphertexts**:

- The challenge generates multiple ciphertexts for `m_1` and `m_2`. We are provided with a file `out.txt` that contains these ciphertexts in the format `[e, n, c]`, where:
  - `e` is the exponent used for encryption (either 91 or 97).
  - `n` is the RSA modulus.
  - `c` is the ciphertext.
- We will group these ciphertexts based on `e` (either `e = 91` or `e = 97`).

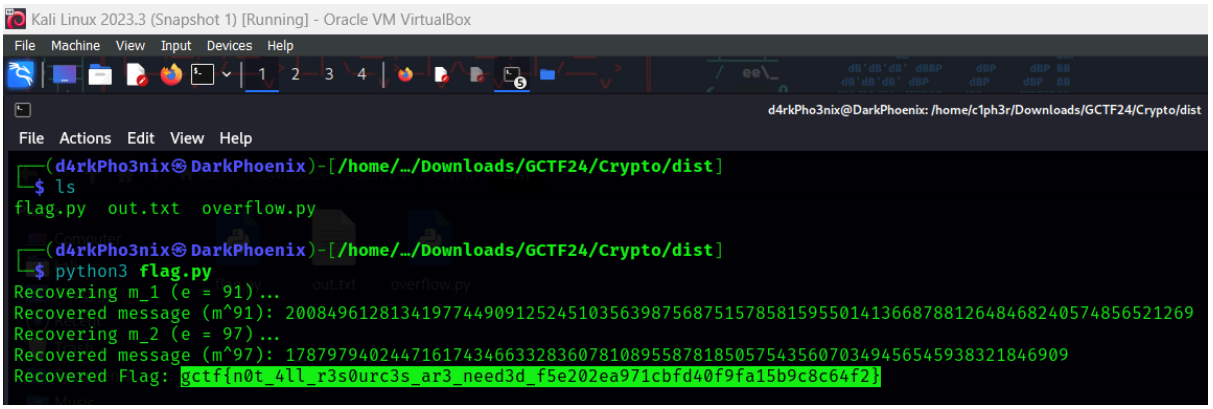
2. **Apply Chinese Remainder Theorem (CRT)**:
   - For each group of ciphertexts (one for `m_1` and another for `m_2`), we can use the **Chinese Remainder Theorem** to combine the ciphertexts into a single system of congruences. This allows us to solve for the encrypted message raised to the power of `e`.
3. **Take the e-th Root**:
   - Once we have the combined ciphertext from the CRT, we can compute the `e`-th root to recover the original message. This gives us the decrypted values of `m_1` and `m_2`.
4. **Reconstruct the Flag**:
   - After recovering both `m_1` and `m_2`, we concatenate them to get the full flag.

Here is the Python script we used to solve the challenge and recover the flag:

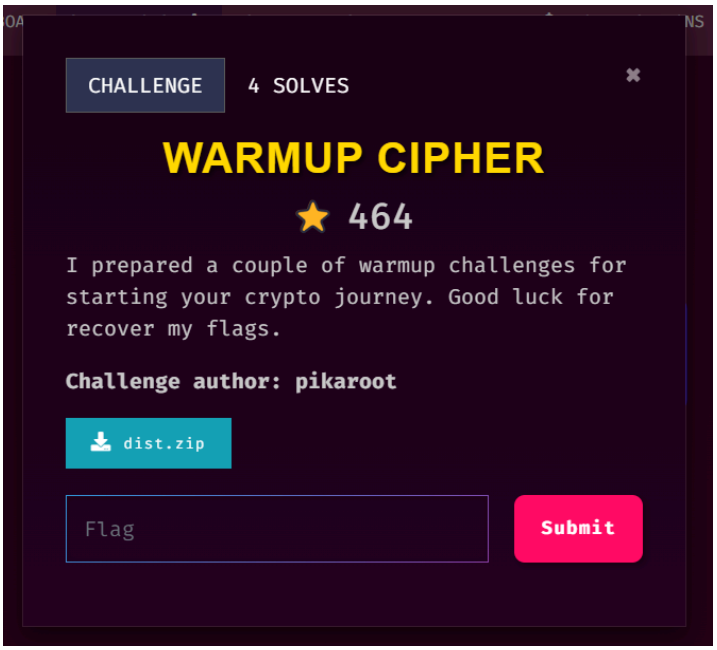
```python
from sympy.ntheory.modular import crt
from gmpy2 import iroot
import re

def read_ciphertexts(file):
    """Reads ciphertexts from out.txt and separates them by e = 91 and e = 97."""
    with open(file, 'r') as f:
        data = f.read()

    # Use regular expression to extract [e, n, c]
    pattern = r"\[(\d+), (\d+), (\d+)\]"
    matches = re.findall(pattern, data)

    ciphertexts_91 = []
    ciphertexts_97 = []

    # Separate ciphertexts based on e value (91 or 97)
    for match in matches:
        e, n, c = int(match[0]), int(match[1]), int(match[2])
        if e == 91:
            ciphertexts_91.append((n, c))
        elif e == 97:
            ciphertexts_97.append((n, c))

    return ciphertexts_91, ciphertexts_97

def recover_message(ciphertexts, e):
    """Applies Håstad's Broadcast Attack to recover the message m."""
    # Extract n and c values
    n_values = [c[0] for c in ciphertexts]
    c_values = [c[1] for c in ciphertexts]

    # Apply Chinese Remainder Theorem (crt) to get combined c and combined n
    combined_c, combined_n = crt(n_values, c_values)

    # Now solve for m: combined_c = m^e
    m, exact = iroot(combined_c, e)  # Taking the e-th root of the combined ciphertext

    if exact:
        print(f"Recovered message (m^{e}):", m)
    else:
        print(f"Failed to recover message for e = {e}")

    return m
```

```python
def main():
    # Read ciphertexts from file
    file = 'out.txt'
    ciphertexts_91, ciphertexts_97 = read_ciphertexts(file)

    # Recover m_1 (with e = 91)
    print("Recovering m_1 (e = 91)...")
    m1 = recover_message(ciphertexts_91, 91)

    # Recover m_2 (with e = 97)
    print("Recovering m_2 (e = 97)...")
    m2 = recover_message(ciphertexts_97, 97)

    # Convert m1 and m2 to bytes and concatenate to get the flag
    # Lengths of m_1 and m_2 are needed to convert them back to bytes (adjust as needed)
    length_of_m1 = 36  # Adjust if needed
    length_of_m2 = len(m2.to_bytes((m2.bit_length() + 7) // 8, byteorder='big'))

    # Combine to form the full flag
    flag = m1.to_bytes(length_of_m1, byteorder='big') + m2.to_bytes(length_of_m2, byteorder='big')

    print("Recovered Flag:", flag.decode('utf-8'))  # Or use other decoding methods if needed

if __name__ == '__main__':
    main()
```

After running the above script, we successfully recover the flag by exploiting the small exponents and multiple ciphertexts.



Warmup Cipher

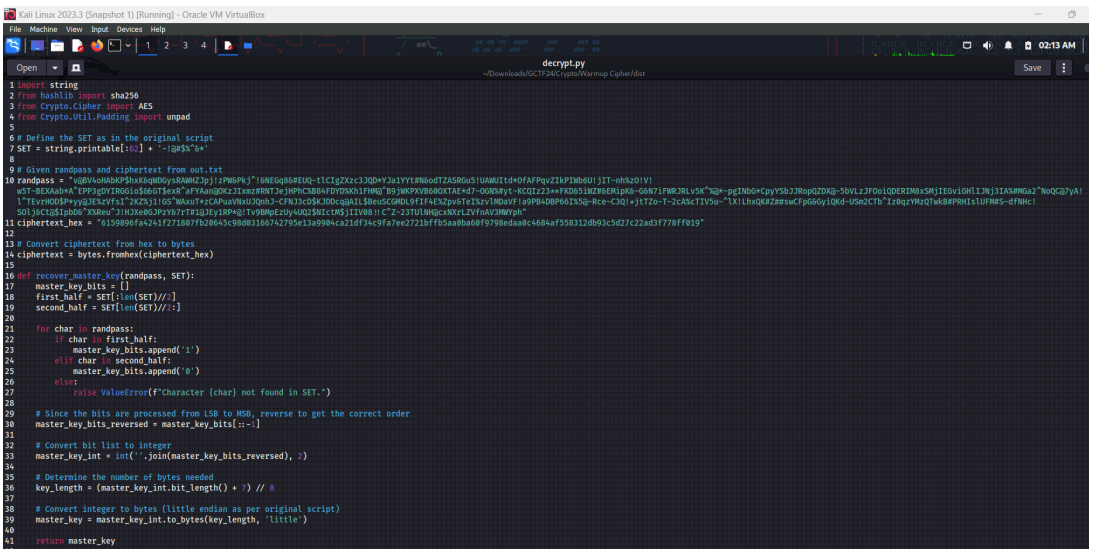


Challenge Description: In this challenge, we were provided with two pieces of information:

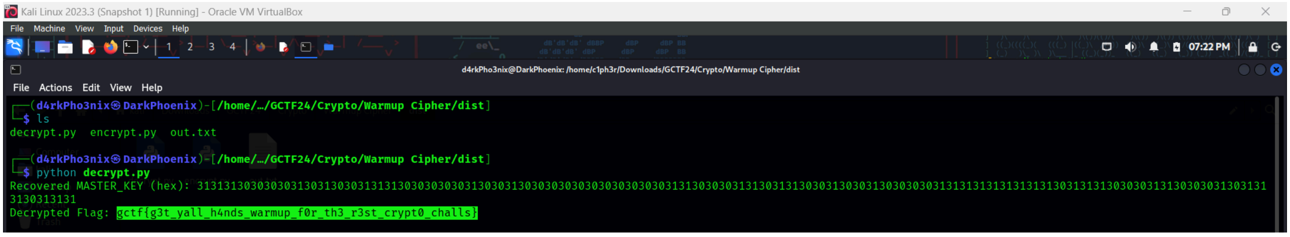
1. A long string labelled randpass, which seems like a random password.
2. An Encrypted Flag, represented as a long string of hexadecimal values.

The goal is to find the flag from the encrypted data.

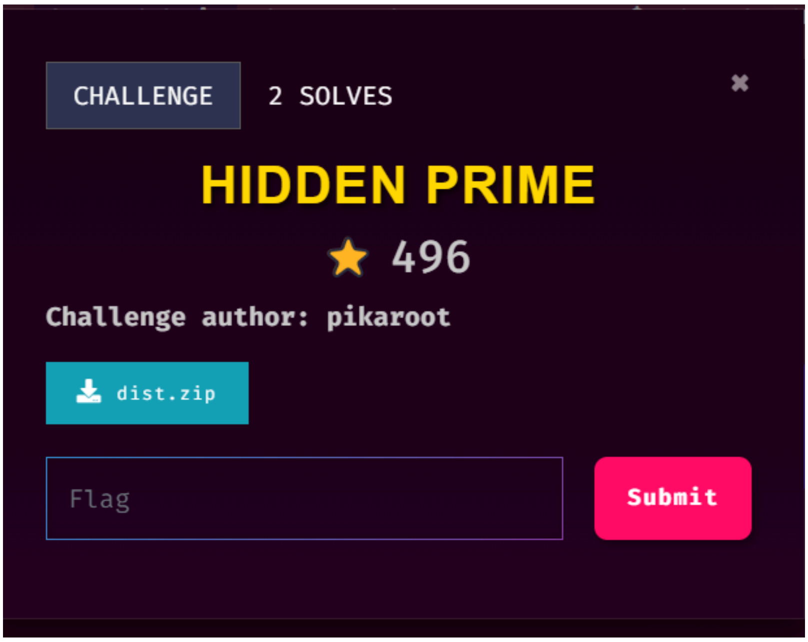
Here is the Python script we used to solve the challenge and recover the flag:



After running the above script, we successfully decrypt the flag.



Hidden Prime

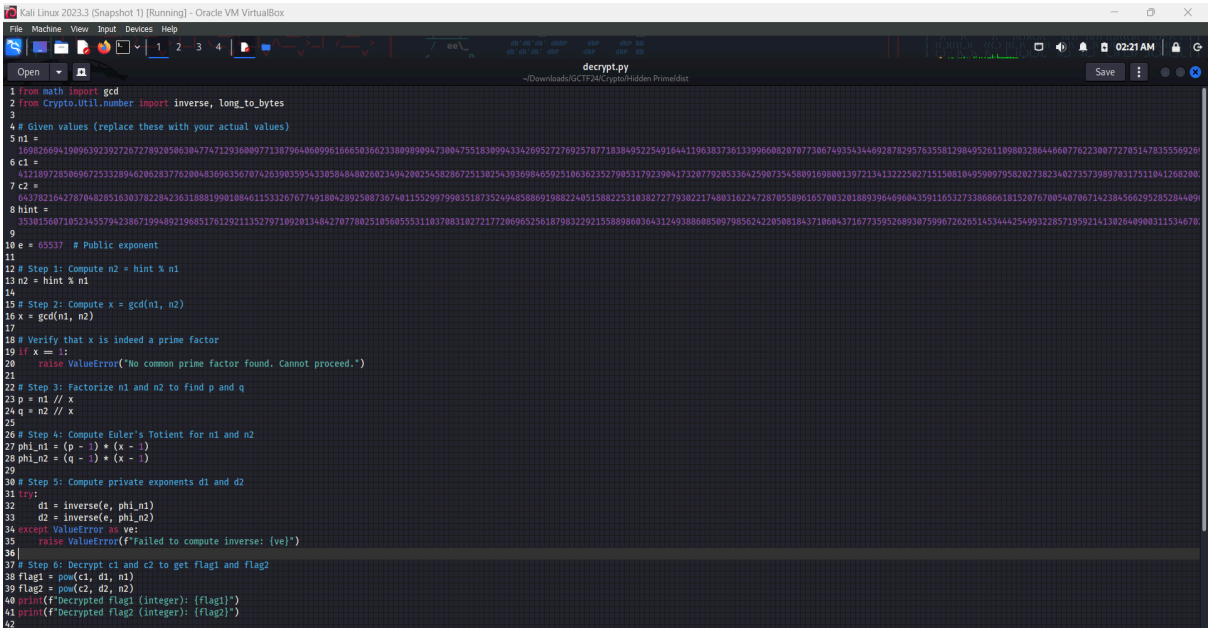


We have been given the following values extracted from out.txt:

- n1: The modulus for the first part of the flag.
- c1: The ciphertext corresponding to flag1.
- c2: The ciphertext corresponding to flag2.
- hint: A value calculated as n1 * E + n2, where E is a random 100-byte number, and n2 is the modulus for the second part of the flag.

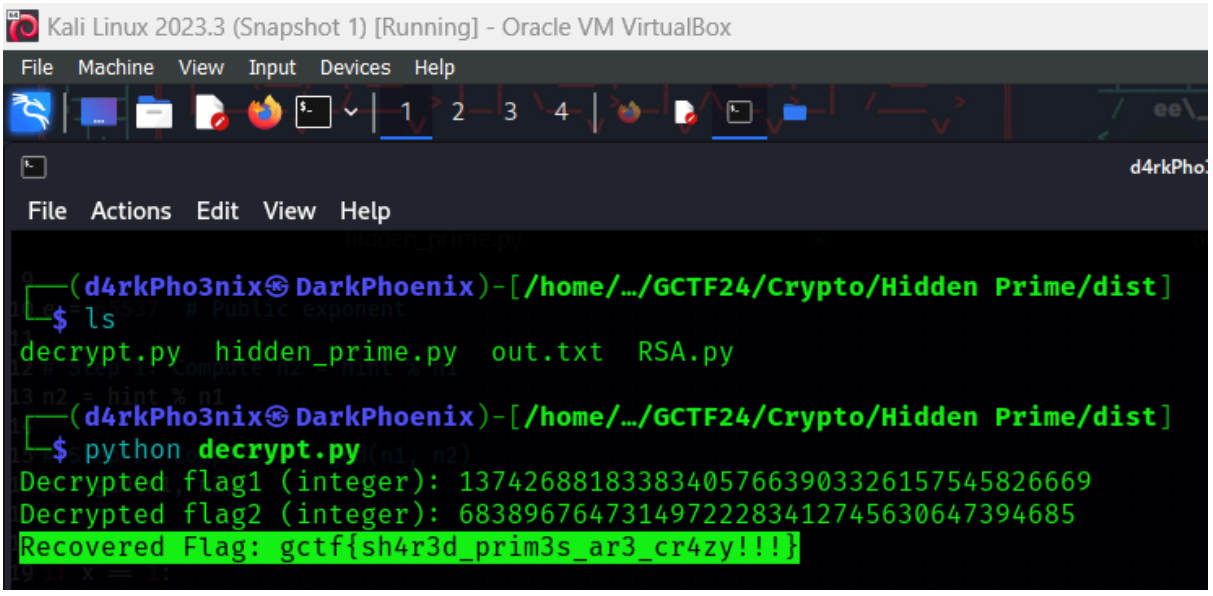
Our goal is to retrieve flag1 and flag2 using these values.

Here is the Python script we used to solve the challenge and recover the flag:



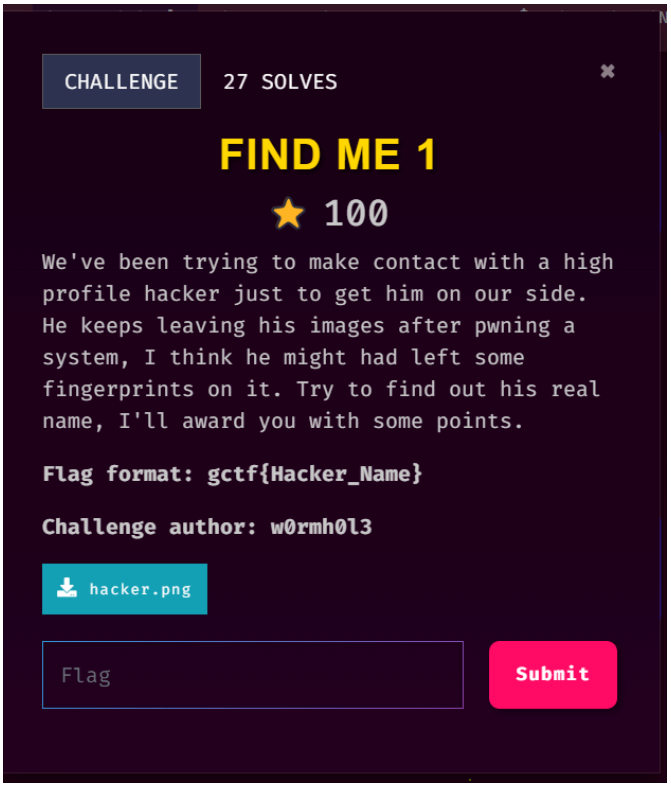

```python
# Step 7: Convert the decrypted integers back to bytes
flag1_bytes = long_to_bytes(flag1)
flag2_bytes = long_to_bytes(flag2)
flag = flag1_bytes + flag2_bytes

try:
    print(f"Recovered Flag: {flag.decode()}")
except UnicodeDecodeError:
    print("Failed to decode the flag. It may contain non-UTF-8 bytes.")
```

After running the above script, we successfully recover the flag.



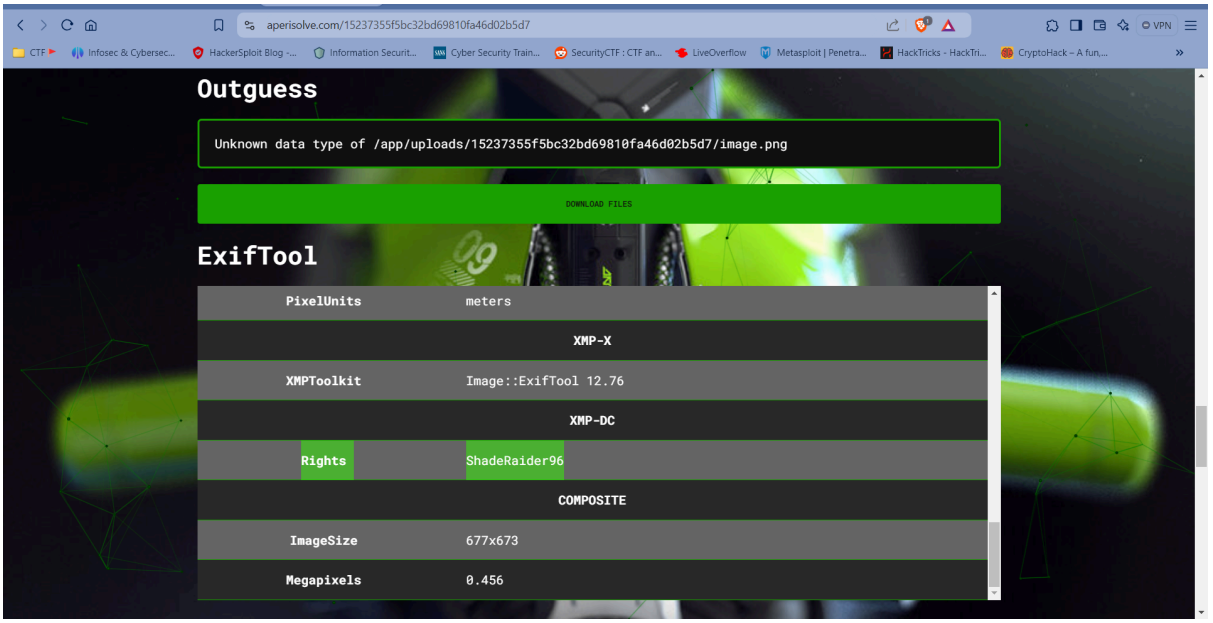
MISC

Find Me 1



We were provided with a file named "hacker.png," which at first glance appeared to be just a regular image.

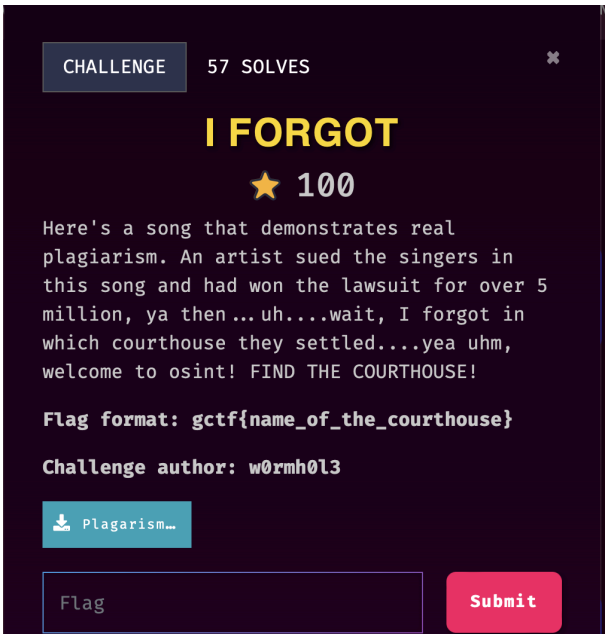
I then used https://www.aperisolve.com/, an online platform for layer analysis of images. It performs various steganography techniques, including zsteg, steghide, outguess, exiftool, binwalk, foremost, and strings. Through this analysis, I discovered a clue that could help identify the real name of the hacker.



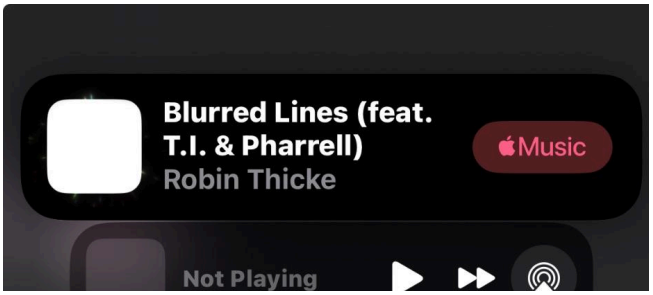
I searched for "ShadeRaider96" across multiple social media platforms and eventually found a matching account on X that fit the description.

Flag: gctf{Paul_Wildenberg}

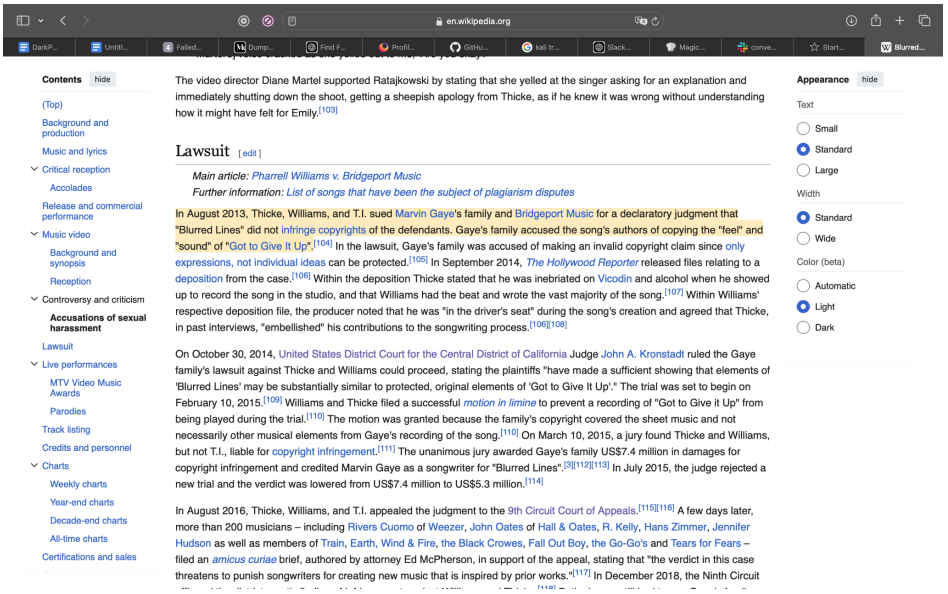
I Forgot



To find the song, I used shazam and found the name of the song and the singer.



This is what I found on google when I searched for the song.

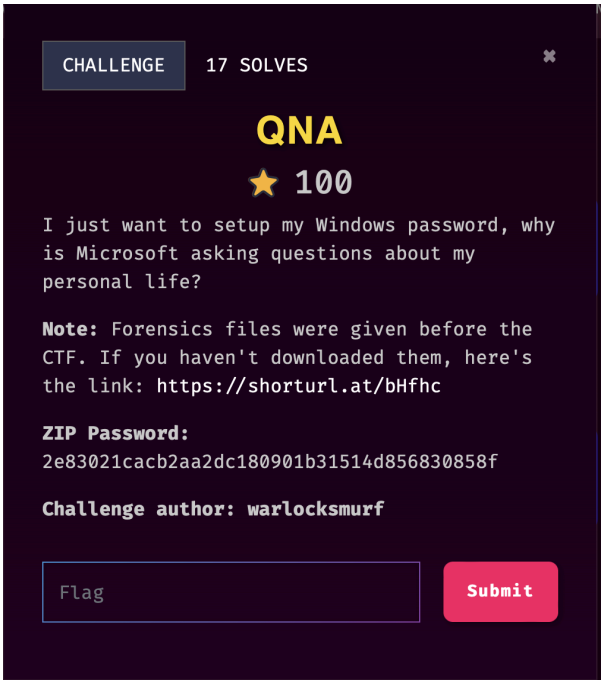


Since the flag is asking for the courthouse name in which they settled the case, I just click “9th Circuit Court of Appeals” to find the actual name of the court house which is United States Court of Appeals for the Ninth Circuit. So the flag would be gctf{United_States_Court_of_Appeals_for_the_Ninth_Circuit}

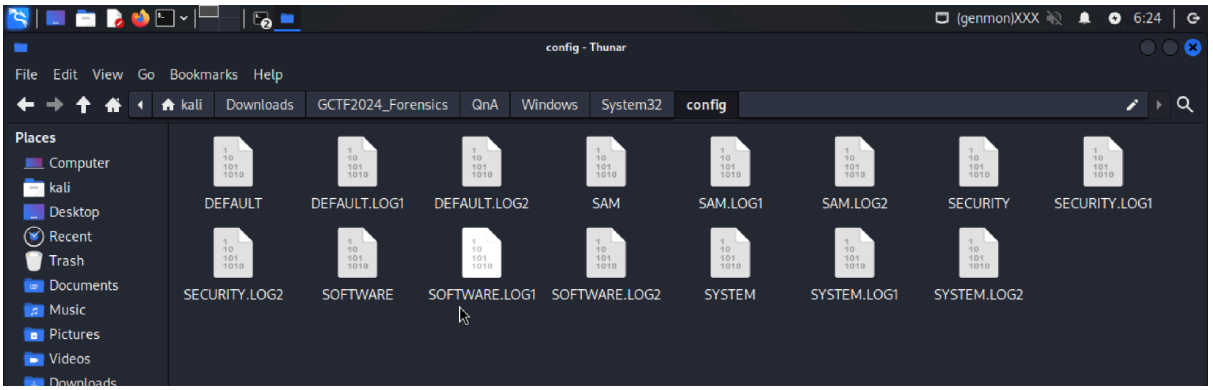
Flag: gctf{United_States_Court_of_Appeals_for_the_Ninth_Circuit}

FORENSIC

QnA



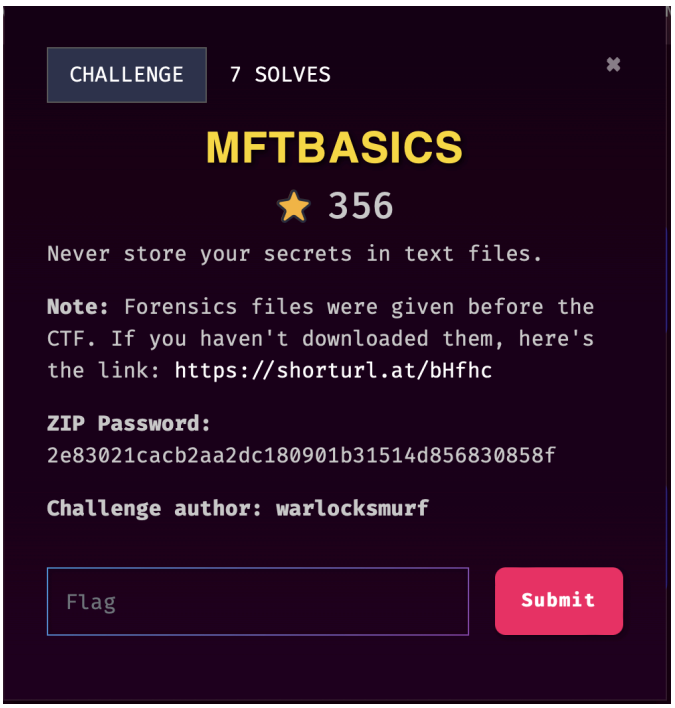
In this directory, I just look for the User's name which is SAM.



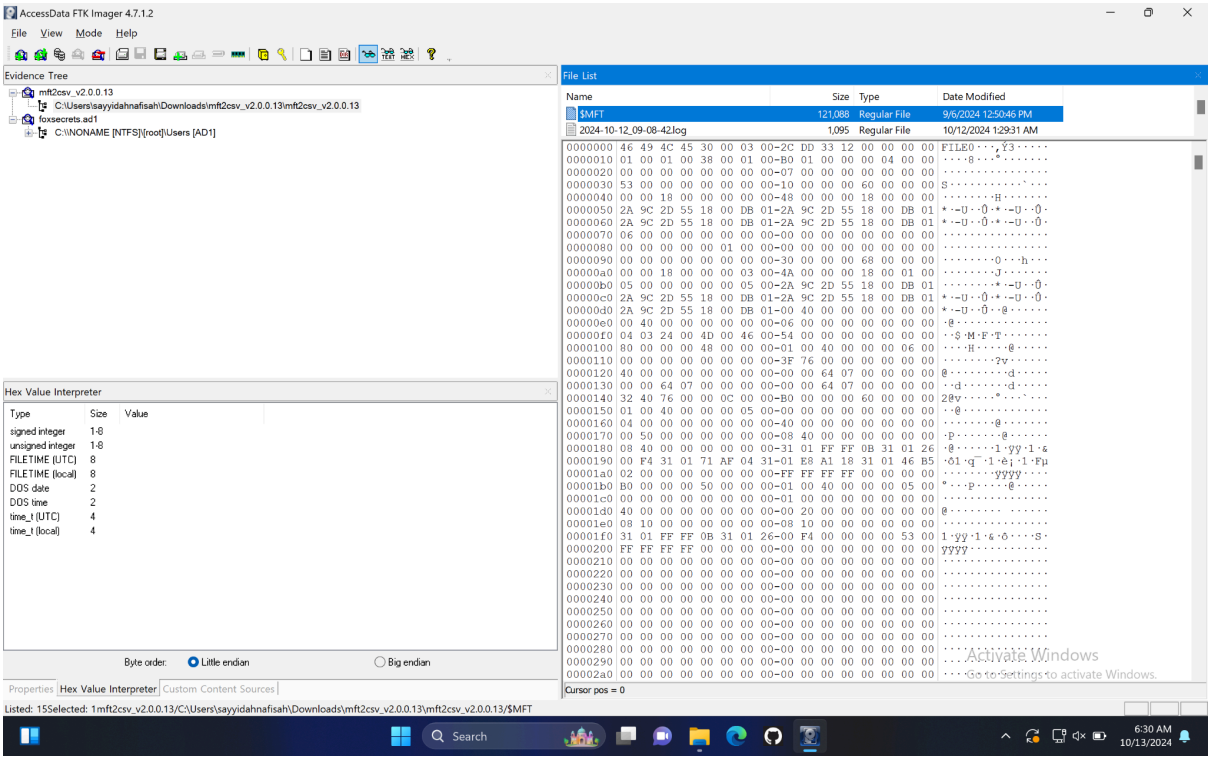
I used the command cat SAM to see the content and finally found the flag which was divided into 3 parts of the questions.

Flag: GCTF{p3rs0nal_s3cr3ts_r3g1stry}

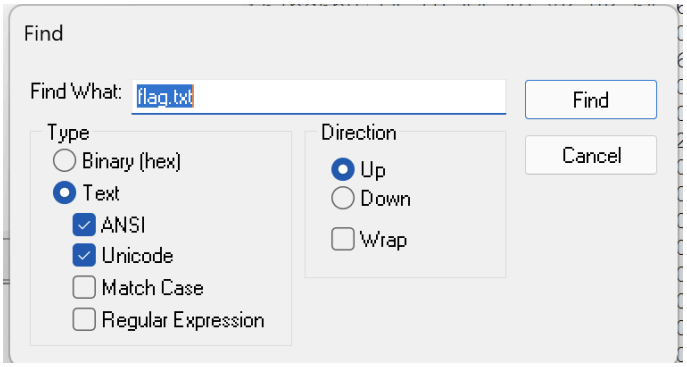
MFTBasics



To perform analysis on the $MFT file, I put it in a folder and opened it using FTK Imager.



Right click and click find to find a keyword for the secret text file which is flag.txt. Just find it until I find the secret code.

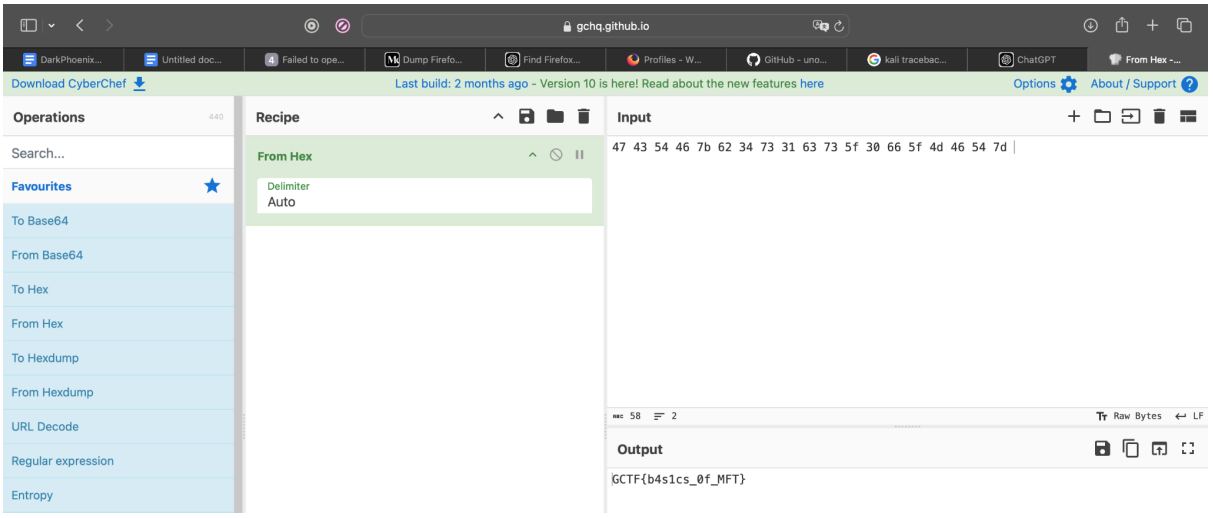


Finally found the secret code:

The secret code is: 47 43 54 46 7b 62 34 73 31 63 73 5f 30 66 5f 4d 46 54 7d 4d

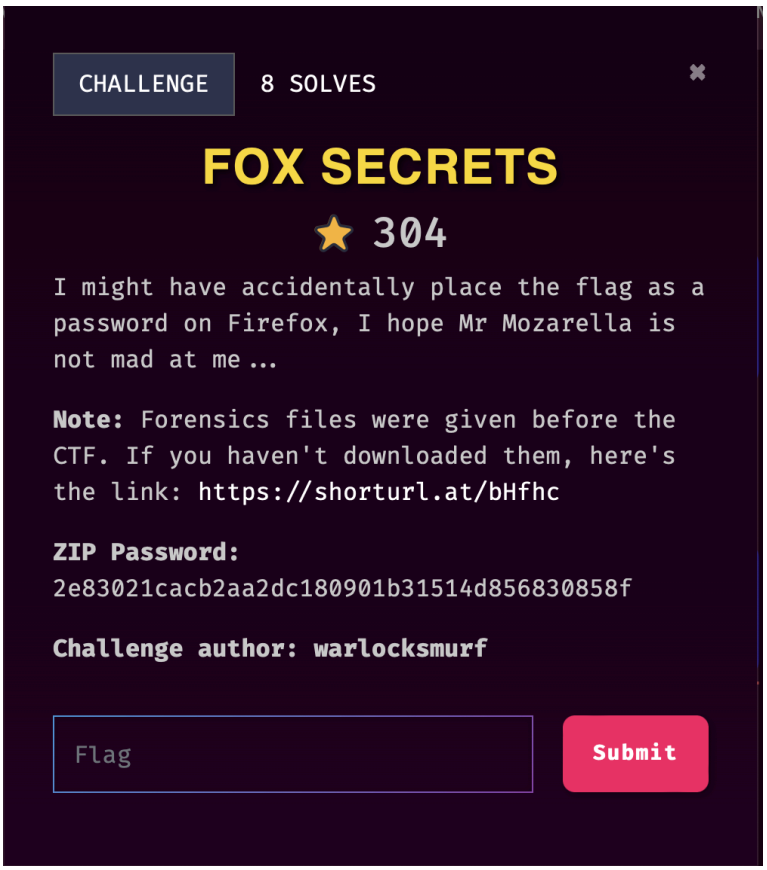
```
6fadcf0 08 03 66 00 6C 00 61 00-67 00 2E 00 74 00 78 00 ··f·l·a·g·.·t·x·
6fadd00 74 00 00 00 00 00 00 00-40 00 00 00 28 00 00 00 t······@···(···
6fadd10 00 00 00 00 00 00 05 00-10 00 00 00 18 00 00 00 ····************
6fadd20 87 8D 61 04 88 6B EF 11-AB D7 00 0C 29 22 8E 27 ··a··kï·«×··)"·'
6fadd30 80 00 00 00 68 00 00 00-00 00 18 00 00 00 01 00 ····h···********
6fadd40 4C 00 00 00 18 00 00 00-54 68 65 20 73 65 63 72 L·******The secr
6fadd50 65 74 20 63 6F 64 65 20-69 73 3A 20 34 37 20 34 et code is: 47 4
6fadd60 33 20 35 34 20 34 36 20-37 62 20 36 32 20 33 34 3 54 46 7b 62 34
6fadd70 20 37 33 20 33 31 20 36-33 20 37 33 20 35 66 20  73 31 63 73 5f
6fadd80 33 30 20 36 36 20 35 66-20 34 64 20 34 36 20 35 30 66 5f 4d 46 5
6fadd90 34 20 37 64 34 64 20 34-FF FF FF FF 82 79 47 11 4 7d4d 4ÿÿÿÿ·yG·
6fadda0 FF FF FF FF 82 79 47 11-00 00 00 00 18 00 00 00 ÿÿÿÿ·yG·******** 
6faddb0 FF FF FF FF 82 79 47 11-00 00 00 00 00 00 00 00 ÿÿÿÿ·yG·********
6faddc0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ****************
```

To decrypt that i just threw it on cyberchef and finally cracked it.



Flag: GCTF{b4s1cs_0f_MFT}

Fox Secrets

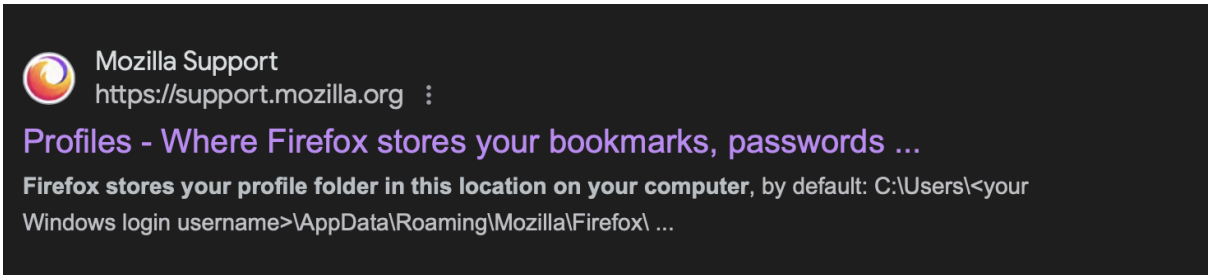


We were given an image file foxsecrets.ad1. To do an analysis on the image file, I opened it using FTK imager.

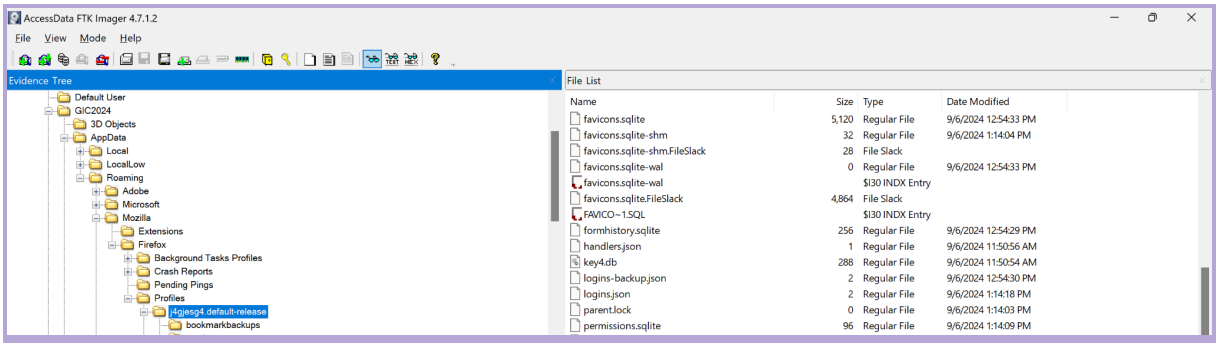
Since the clue said they stored the password on firefox, I must look for key4.db and logins.json according to https://support.mozilla.org/en-US/kb/profiles-where-firefox-stores-user-data,

So I looked for the path of the stored password on mozilla which is:

username\AppData\Roaming\Mozilla\Firefox\Profiles



So to look for the files, I just opened the directories on FTK imager GIC2024\AppData\Roaming\Mozilla\Firefox\Profiles\j4gjesg4.default-release and as shown below, there are key4.db and logins.json in that folder.

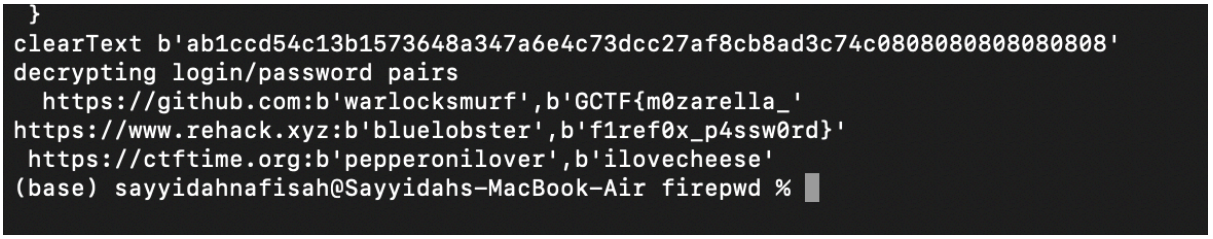


To retrieve the password, I used this website as my reference: https://medium.com/@s12deff/dump-firefox-passwords-with-firepwd-and-firefox-decrypt-65350fd74503

Since we already have key4.db and logins.json, the step to find the passwords are

1. git clone https://github.com/lclevy/firepwd
2. Put the key4.db and logins.json in the same folder as firepwd
3. python3 firepwd.py

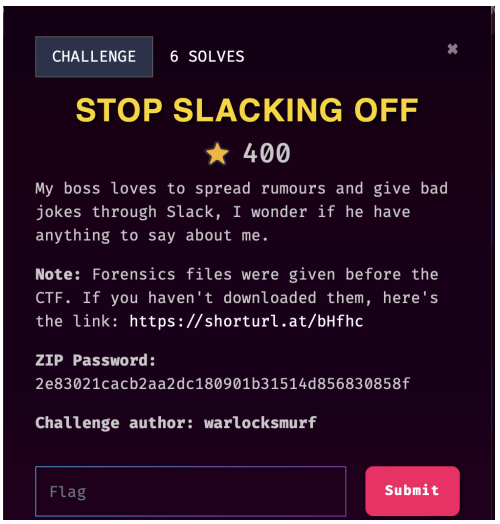
```
zsh: no such file or directory: https://github.com/lclevy/firepwd
(base) sayyidahnafisah@Sayyidahs-MacBook-Air fox secret % git clone https://github.com/lclevy/firepwd

Cloning into 'firepwd'...
remote: Enumerating objects: 88, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 88 (delta 2), reused 3 (delta 0), pack-reused 80 (from 1)
Receiving objects: 100% (88/88), 239.08 KiB | 2.28 MiB/s, done.
Resolving deltas: 100% (41/41), done.
(base) sayyidahnafisah@Sayyidahs-MacBook-Air fox secret % ls
firepwd         key4.db         logins.json
(base) sayyidahnafisah@Sayyidahs-MacBook-Air fox secret % cd firepwd
(base) sayyidahnafisah@Sayyidahs-MacBook-Air firepwd % ls
LICENSE                 logins.json             mozilla_pbe.svg
firepwd.py              mozilla_db              readme.md
key4.db                 mozilla_pbe.pdf         requirements.txt
(base) sayyidahnafisah@Sayyidahs-MacBook-Air firepwd % python3 firepwd
python3: can't open file '/Users/sayyidahnafisah/Desktop/shared/fox secret/firepwd/firepwd': [Errno 2] No such file or directory
(base) sayyidahnafisah@Sayyidahs-MacBook-Air firepwd % python3 firepwd.py
globalSalt: b'b5dbfec66b891e193f516eccaf39209a93634332'
 SEQUENCE {
   SEQUENCE {
     OBJECTIDENTIFIER 1.2.840.113549.1.5.13 pkcs5 pbes2
     SEQUENCE {
       SEQUENCE {
         OBJECTIDENTIFIER 1.2.840.113549.1.5.12 pkcs5 PBKDF2
         SEQUENCE {
           OCTETSTRING b'eaa234484d176f2f091e1a9b2162b550a9874bf9ced92daa19c43e058b1328cf'
           INTEGER b'01'
           INTEGER b'20'
           SEQUENCE {
             OBJECTIDENTIFIER 1.2.840.113549.2.9 hmacWithSHA256
           }
         }
       }
       SEQUENCE {
         OBJECTIDENTIFIER 2.16.840.1.101.3.4.1.42 aes256-CBC
         OCTETSTRING b'c361eb13322fd6004ad463de0ef2'
       }
     }
   }
   OCTETSTRING b'c2cb6eb12879f4c649458e59d634f355'
 }
clearText b'70617373776f72642d636865636b0202'
password check? True
 SEQUENCE {
   SEQUENCE {
     OBJECTIDENTIFIER 1.2.840.113549.1.5.13 pkcs5 pbes2
     SEQUENCE {
       SEQUENCE {
         OBJECTIDENTIFIER 1.2.840.113549.1.5.12 pkcs5 PBKDF2
         SEQUENCE {
           OCTETSTRING b'ae2720e0964ce4beabedba40345712df4234f9ac9cd86e53a08982b65abbbd9c'
           INTEGER b'01'
           INTEGER b'20'
           SEQUENCE {
             OBJECTIDENTIFIER 1.2.840.113549.2.9 hmacWithSHA256
           }
         }
       }
       SEQUENCE {
         OBJECTIDENTIFIER 2.16.840.1.101.3.4.1.42 aes256-CBC
```



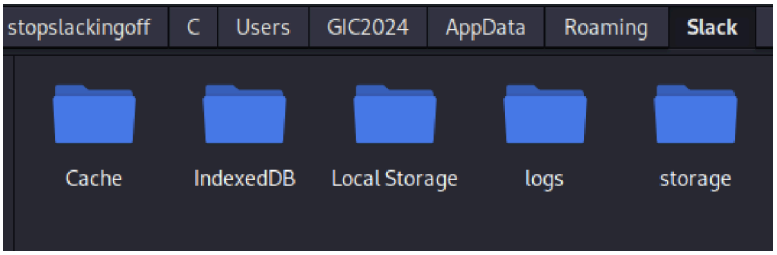
Finally we cracked the password and retrieved the flag after doing all the steps.

Flag:GCTF{m0zarella_f1ref0x_p4ssw0rd}

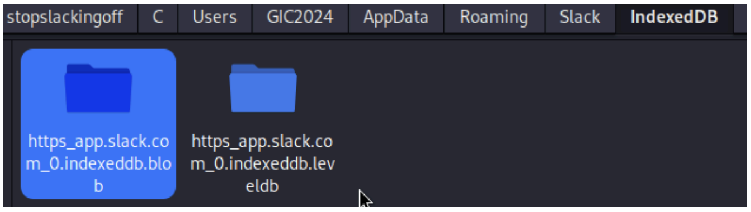
Stop Slacking Off



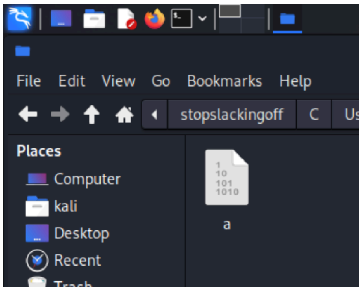
There are 5 folders in the Slack directory. I actually checked and `cat` all the files in those folders.



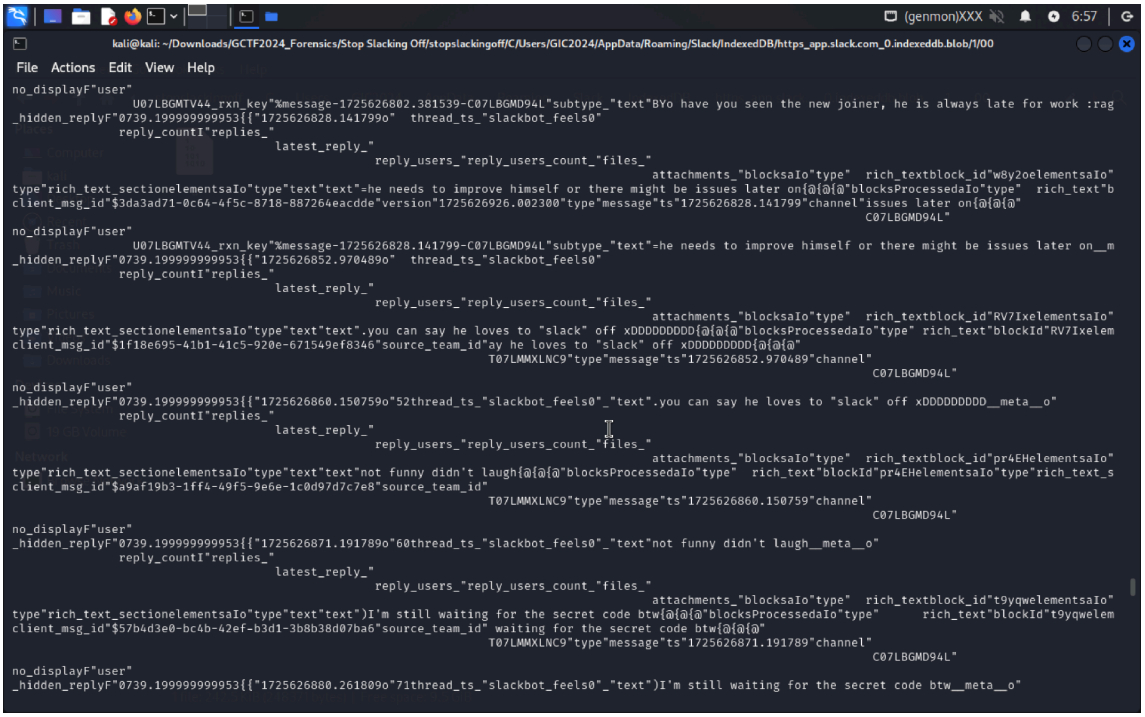
After carefully checked the content of the files, I finally found the conversations on `/Slack/IndexedDB/https_app.slack.com_0.indexeddb.blob`



Below was the only file in that folder and I checked the content using `cat a`.

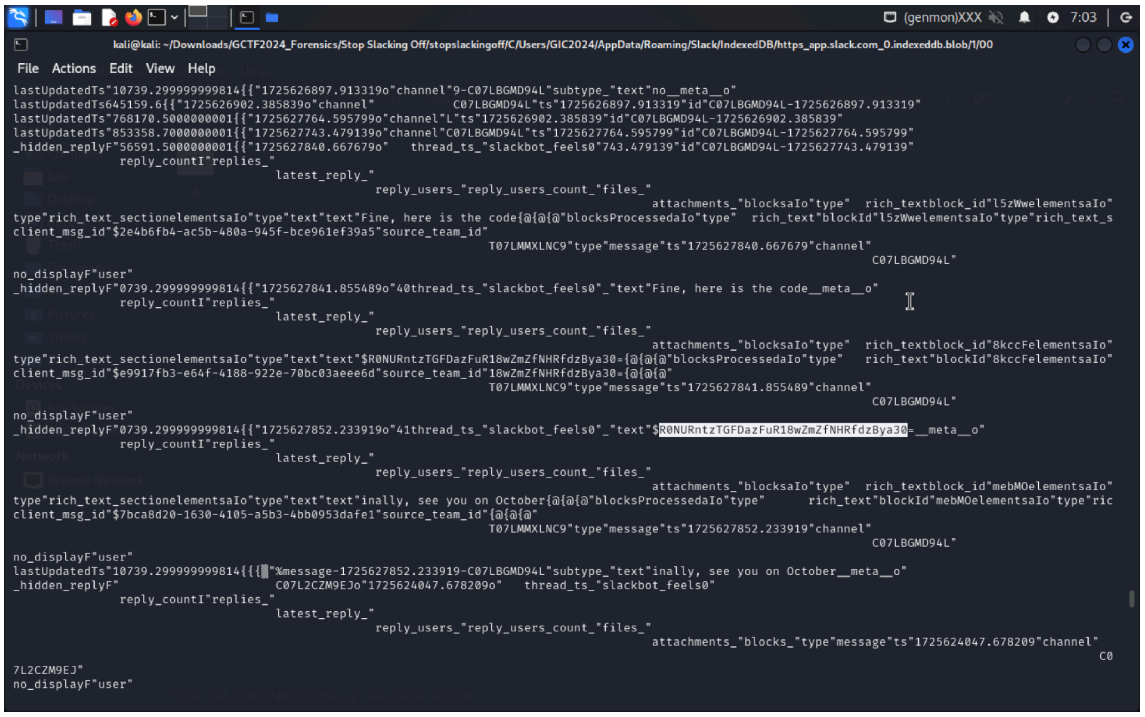


Found the conversation history on slack in that file.

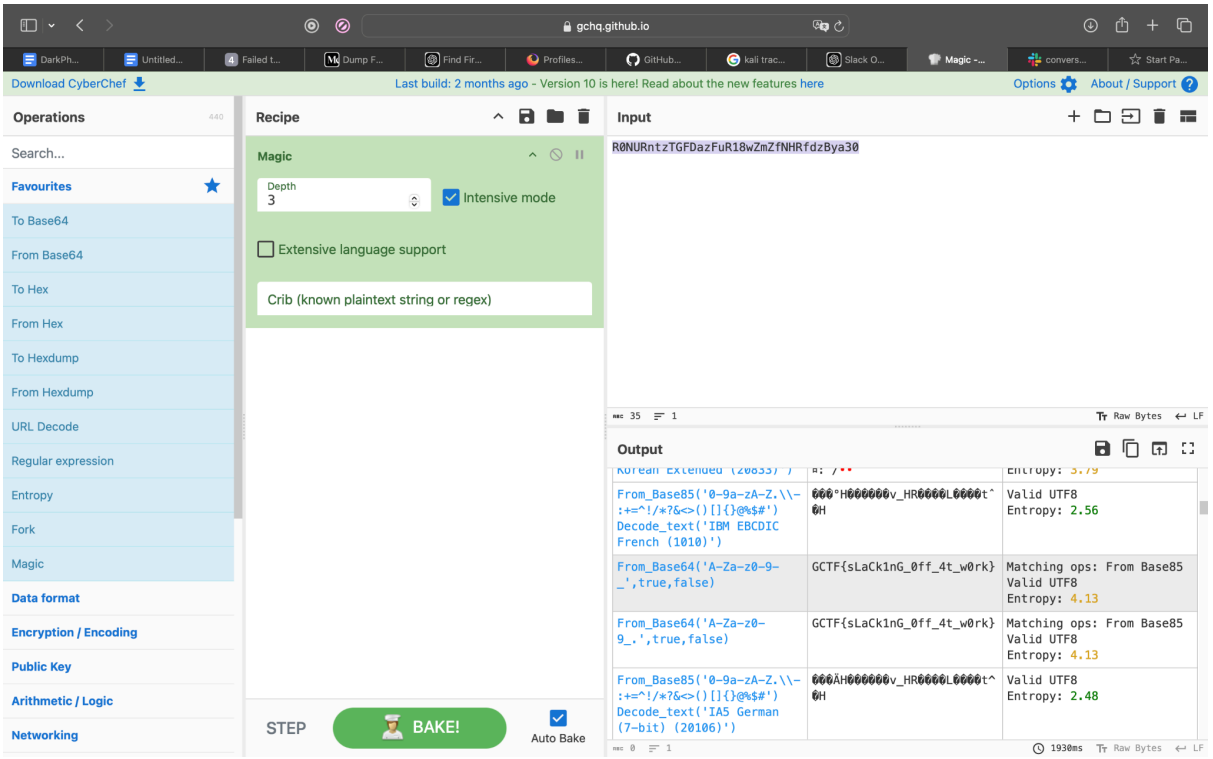


After carefully reading the conversations, I found the secret code which is:

R0NURntzTGFDazFuR18wZmZfNHRfdzBya30



As usual, to decrypt it, I just put in the cyberchef and let it do the work.



Flag: GCTF{sLaCk1nG_0ff_4t_w0rk}