

Girls in CTF 2024



Empowering the Next Generation of Cybersecurity Enthusiasts

Wrote by

Dark Phoenix

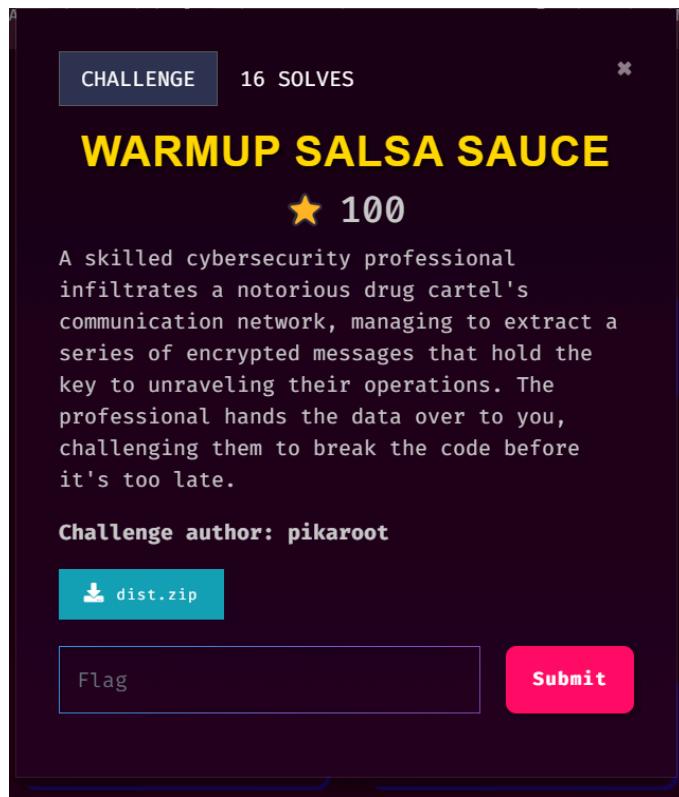
d4rkPho3nix
.geshulin.

Table of Contents

CRYPTO.....	3
Warmup Salsa Sauce.....	3
Overflow Resources.....	4
Warmup Cipher.....	8
Hidden Prime.....	9
MISC.....	11
Find Me 1.....	11
I Forgot.....	13
FORENSIC.....	15
QnA.....	15
MFTBasics.....	16
Fox Secrets.....	18
Stop Slacking Off.....	21

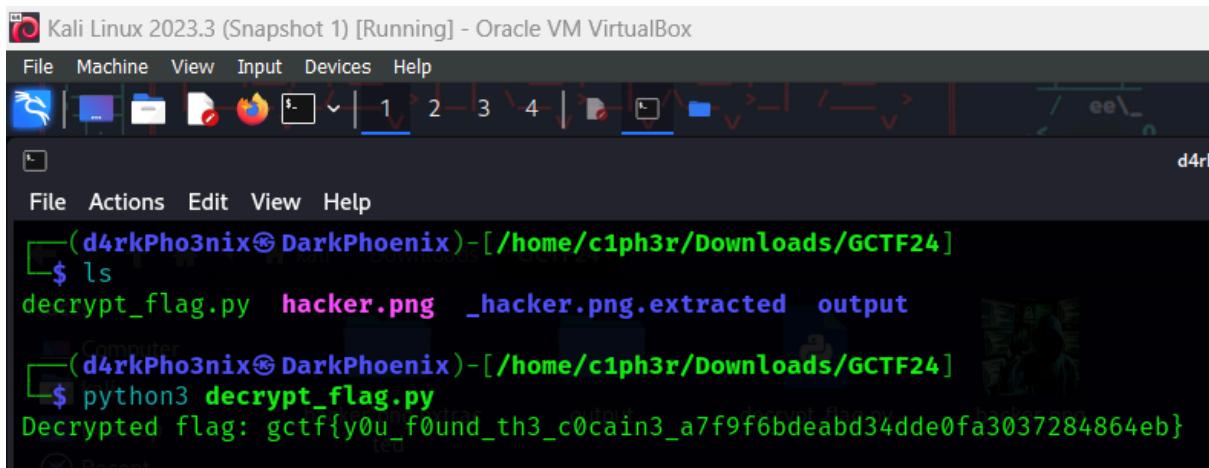
CRYPTO

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.

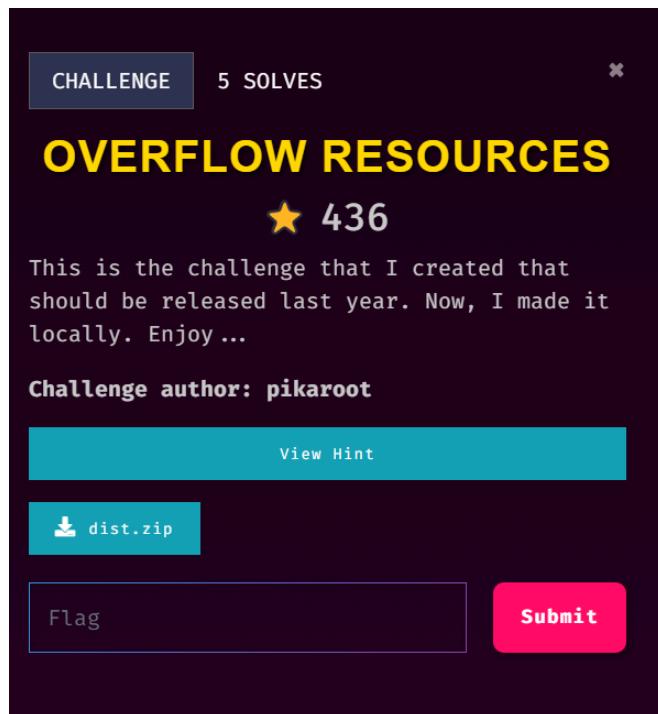


Kali Linux 2023.3 (Snapshot 1) [Running] - Oracle VM VirtualBox

```
(d4rkPho3nix㉿DarkPhoenix)-[~/home/c1ph3r/Downloads/GCTF24]
$ ls
decrypt_flag.py  hacker.png  _hacker.png.extracted  output

(d4rkPho3nix㉿DarkPhoenix)-[~/home/c1ph3r/Downloads/GCTF24]
$ python3 decrypt_flag.py
Decrypted flag: gctf{y0u_f0und_th3_c0cain3_a7f9f6bdeabd34dde0fa3037284864eb}
```

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:

```

1|from sympy.ntheory.modular import crt
2|from gmpy2 import iroot
3|import re
4|
5|def read_ciphertexts(file):
6|    """Reads ciphertexts from out.txt and separates them by e = 91 and e = 97."""
7|    with open(file, 'r') as f:
8|        data = f.read()
9|
10|    # Use regular expression to extract [e, n, c]
11|    pattern = r"[(\d+), (\d+), (\d+)]""
12|    matches = re.findall(pattern, data)
13|
14|    ciphertexts_91 = []
15|    ciphertexts_97 = []
16|
17|    # Separate ciphertexts based on e value (91 or 97)
18|    for match in matches:
19|        e, n, c = int(match[0]), int(match[1]), int(match[2])
20|        if e == 91:
21|            ciphertexts_91.append((n, c))
22|        elif e == 97:
23|            ciphertexts_97.append((n, c))
24|
25|    return ciphertexts_91, ciphertexts_97
26|
27|def recover_message(ciphertexts, e):
28|    """Applies Håstad's Broadcast Attack to recover the message m."""
29|    # Extract n and c values
30|    n_values = [c[0] for c in ciphertexts]
31|    c_values = [c[1] for c in ciphertexts]
32|
33|    # Apply Chinese Remainder Theorem (crt) to get combined c and combined n
34|    combined_c, combined_n = crt(n_values, c_values)
35|
36|    # Now solve for m: combined_c = m^e
37|    m, exact = iroot(combined_c, e) # Taking the e-th root of the combined ciphertext
38|
39|    if exact:
40|        print(f"Recovered message (m^{e}):", m)
41|    else:
42|        print(f"Failed to recover message for e = {e}")
43|
44|    return m
45|

```

```

45
46 def main():
47     # Read ciphertexts from file
48     file = 'out.txt'
49     ciphertexts_91, ciphertexts_97 = read_ciphertexts(file)
50
51     # Recover m_1 (with e = 91)
52     print("Recovering m_1 (e = 91)...")
53     m1 = recover_message(ciphertexts_91, 91)
54
55     # Recover m_2 (with e = 97)
56     print("Recovering m_2 (e = 97)...")
57     m2 = recover_message(ciphertexts_97, 97)
58
59     # Convert m1 and m2 to bytes and concatenate to get the flag
60     # Lengths of m_1 and m_2 are needed to convert them back to bytes (adjust as needed)
61     length_of_m1 = 36 # Adjust if needed
62     length_of_m2 = len(m2.to_bytes((m2.bit_length() + 7) // 8, byteorder='big'))
63
64     # Combine to form the full flag
65     flag = m1.to_bytes(length_of_m1, byteorder='big') + m2.to_bytes(length_of_m2, byteorder='big')
66
67     print("Recovered Flag:", flag.decode('utf-8')) # Or use other decoding methods if needed
68
69 if __name__ == '__main__':
70     main()

```

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

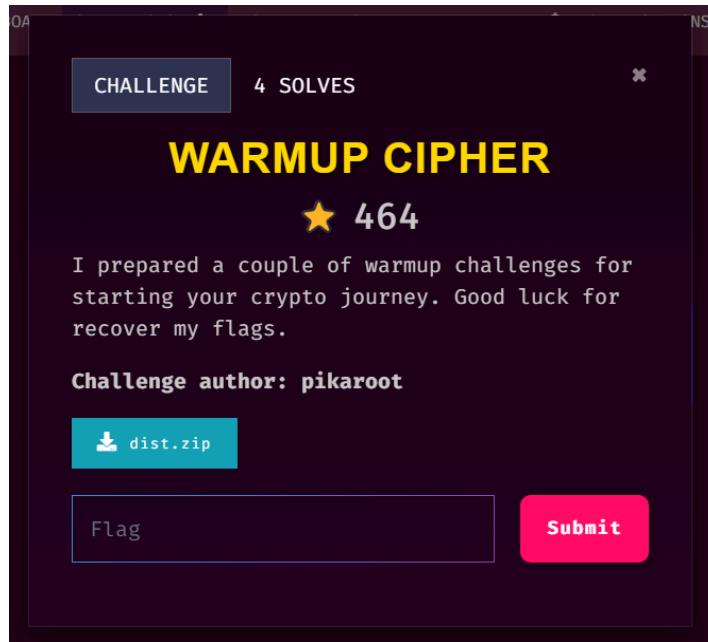
The screenshot shows a terminal window titled "Kali Linux 2023.3 (Snapshot 1) [Running] - Oracle VM VirtualBox". The terminal output is as follows:

```

File Machine View Input Devices Help
File Actions Edit View Help
(d4rkPho3nix㉿DarkPhoenix)-[~/Downloads/GCTF24/Crypto/dist]
$ ls
flag.py out.txt overflow.py
(d4rkPho3nix㉿DarkPhoenix)-[~/Downloads/GCTF24/Crypto/dist]
$ python3 flag.py
Recovering m_1 (e = 91)...
Recovered message (m^91): 200849612813419774490912524510356398756875157858159550141366878812648468240574856521269
Recovering m_2 (e = 97)...
Recovered message (m^97): 178797940244716174346633283607810895587818505754356070349456545938321846909
Recovered Flag: gctf{not_4ll_r3s0urc3s_ar3_need3d_f5e202ea971cbfd40f9fa15b9c8c64f2}

```

Warmup Cipher



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

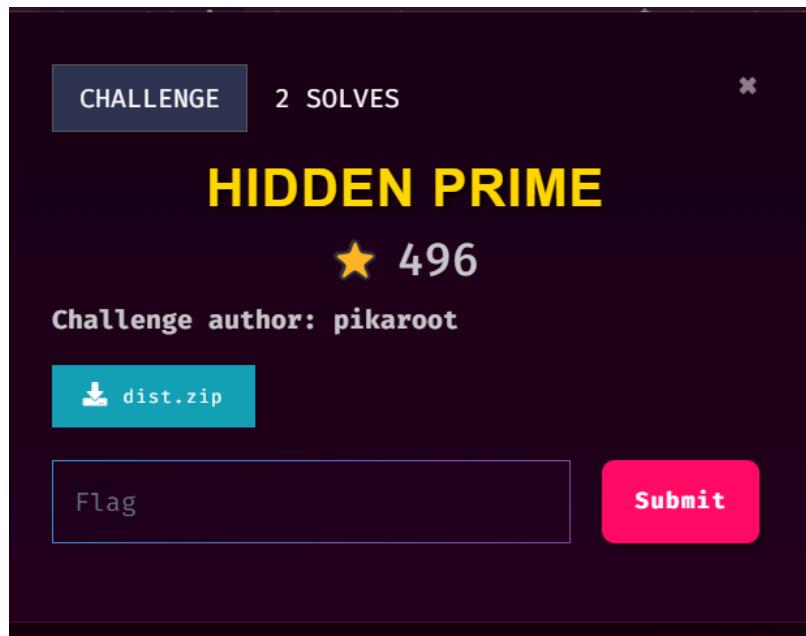
1. A long string labelled **randpass**, which seems like a random password.
 2. An **Encrypted Flag**, represented as a long string of hexadecimal values.

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](#):

- n_1 : The modulus for the first part of the flag.
 - c_1 : The ciphertext corresponding to `flag1`.
 - c_2 : The ciphertext corresponding to `flag2`.
 - hint: A value calculated as $n_1 * E + n_2$, where E is a random 100-byte number, and n_2 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:

```

1 From math import gcd
2 from Crypto.Util.number import inverse, long_to_bytes
3
4 # Given values (replace these with your actual values)
5 n1 =
6 c1 =
7 x =
8 hint =
9
10 e = 65537 # Public exponent
11
12 # Step 1: Compute n2 = hint % n1
13 n2 = hint % n1
14
15 # Step 2: Compute x = gcd(n1, n2)
16 x = gcd(n1, n2)
17
18 # Verify that x is indeed a prime factor
19 if x == 1:
20     raise ValueError("No common prime factor found. Cannot proceed.")
21
22 # Step 3: Factorize n1 and n2 to find p and q
23 p = n1 // x
24 q = n2 // x
25
26 # Step 4: Compute Euler's Totient for n1 and n2
27 phi_n1 = (p - 1) * (x - 1)
28 phi_n2 = (q - 1) * (x - 1)
29
30 # Step 5: Compute private exponents d1 and d2
31 try:
32     d1 = inverse(e, phi_n1)
33     d2 = inverse(e, phi_n2)
34 except ValueError as ve:
35     raise ValueError(f"Failed to compute inverse: {ve}")
36
37 # Step 6: Decrypt c1 and c2 to get flag1 and flag2
38 flag1 = pow(c1, d1, n1)
39 flag2 = pow(c2, d2, n2)
40 print(f"Decrypted flag1 (integer): {flag1}")
41 print(f"Decrypted flag2 (integer): {flag2}")
42

```

```

42
43 # Step 7: Convert the decrypted integers back to bytes
44 flag1_bytes = long_to_bytes(flag1)
45 flag2_bytes = long_to_bytes(flag2)
46 flag = flag1_bytes + flag2_bytes
47
48 try:
49     print(f"Recovered Flag: {flag.decode()}")
50 except UnicodeDecodeError:
51     print("Failed to decode the flag. It may contain non-UTF-8 bytes.")
52

```

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

```

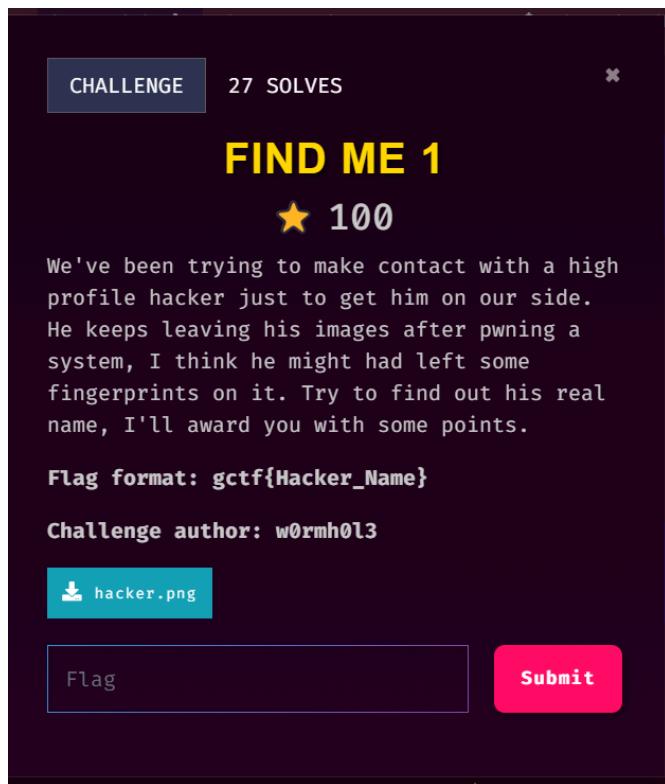
d4rkPho3nix@DarkPhoenix:[/home/.../GCTF24/Crypto/Hidden Prime/dist]
$ ls
decrypt.py  hidden_prime.py  out.txt  RSA.py

d4rkPho3nix@DarkPhoenix:[/home/.../GCTF24/Crypto/Hidden Prime/dist]
$ python decrypt.py
Decrypted flag1 (integer): 137426881833834057663903326157545826669
Decrypted flag2 (integer): 68389676473149722283412745630647394685
Recovered Flag: gctf{sh4r3d_prim3s_ar3_cr4zy!!!}

```

MISC

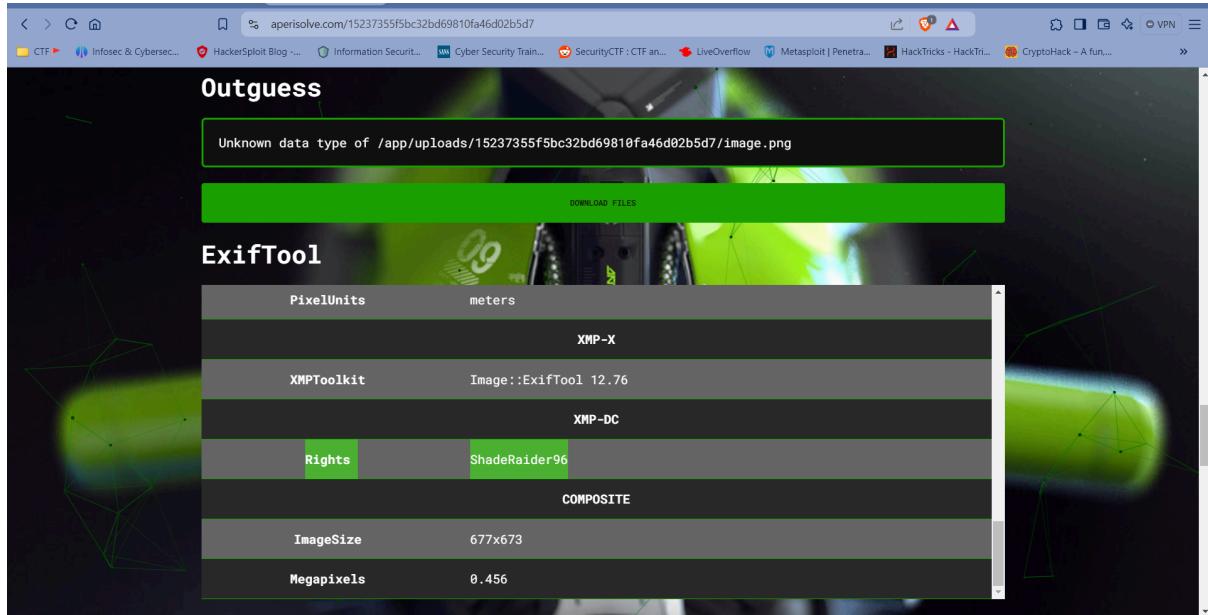
Find Me 1



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.

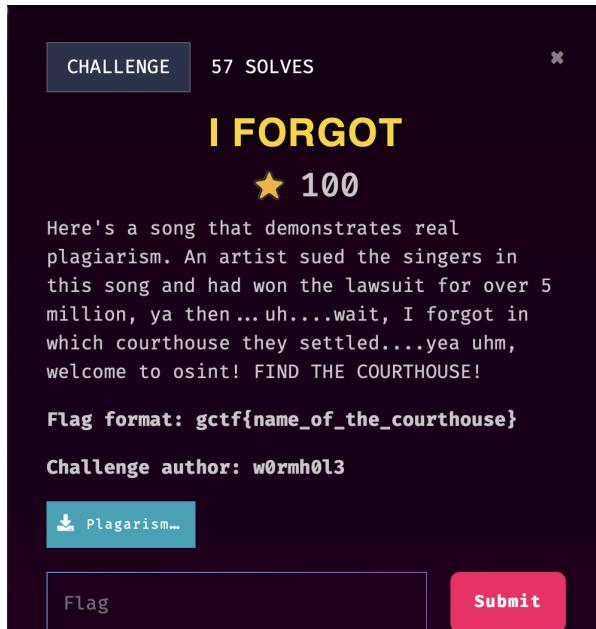


I searched for "ShadeRaider96" across multiple social media platforms and eventually found a matching account on X that fit the description.

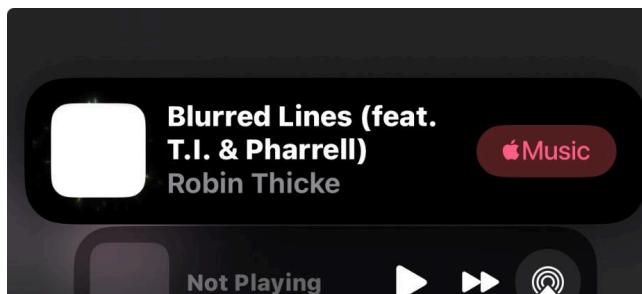


Flag: gctf{Paul_Wildenberg}

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.

The video director Diane Martel supported Ratajkowski by stating that she yelled at the singer asking for an explanation and immediately shutting down the shoot, getting a sheepish apology from Thicke, as if he knew it was wrong without understanding how it might have felt for Emily.^[103]

Lawsuit [edit]

Main article: *Pharrell Williams v. Bridgeport Music*
Further information: *List of songs that have been the subject of plagiarism disputes*

In August 2013, Thicke, Williams, and T.I. sued Marvin Gaye's family and *Bridgeport Music* for a declaratory judgment that "Blurred Lines" did not infringe copyrights of the defendants. Gaye's family accused the song's authors of copying the "feel" and "sound" of "Got to Give It Up".^[104] In the lawsuit, Gaye's family was accused of making an invalid copyright claim since only expressions, not individual ideas can be protected.^[105] In September 2014, *The Hollywood Reporter* released files relating to a deposition from the case.^[106] Within the deposition Thicke stated that he was inebriated on *Vicodin* and alcohol when he showed up to record the song in the studio, and that Williams had the beat and wrote the vast majority of the song.^[107] Within Williams' respective deposition file, the producer noted that he was "in the driver's seat" during the song's creation and agreed that Thicke, in past interviews, "embellished" his contributions to the songwriting process.^{[108][109]}

On October 30, 2014, United States District Court for the Central District of California Judge John A. Kronstadt ruled the Gaye family's lawsuit against Thicke and Williams could proceed, stating the plaintiffs "have made a sufficient showing that elements of 'Blurred Lines' may be substantially similar to protected, original elements of 'Got to Give It Up.' The trial was set to begin on February 10, 2015.^[109] Williams and Thicke filed a successful *motion in limine* to prevent a recording of "Got to Give It Up" from being played during the trial.^[110] The motion was granted because the family's copyright covered the sheet music and not necessarily other musical elements from Gaye's recording of the song.^[110] On March 10, 2015, a jury found Thicke and Williams, but not T.I., liable for copyright infringement.^[111] The unanimous jury awarded Gaye's family US\$7.4 million in damages for copyright infringement and credited Marvin Gaye as a songwriter for "Blurred Lines".^{[112][113]} In July 2015, the judge rejected a new trial and the verdict was lowered from US\$7.4 million to US\$5.3 million.^[114]

In August 2016, Thicke, Williams, and T.I. appealed the judgment to the 9th Circuit Court of Appeals.^{[115][116]} A few days later, more than 200 musicians – including Rivers Cuomo of Weezer, John Oates of Hall & Oates, R. Kelly, Hans Zimmer, Jennifer Hudson as well as members of Train, Earth, Wind & Fire, the Black Crowes, Fall Out Boy, the Go-Go's and Tears for Fears – filed an *amicus curiae* brief, authored by attorney Ed McPherson, in support of the appeal, stating that "the verdict in this case threatens to punish songwriters for creating new music that is inspired by prior works."^[117] In December 2018, the Ninth Circuit

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}

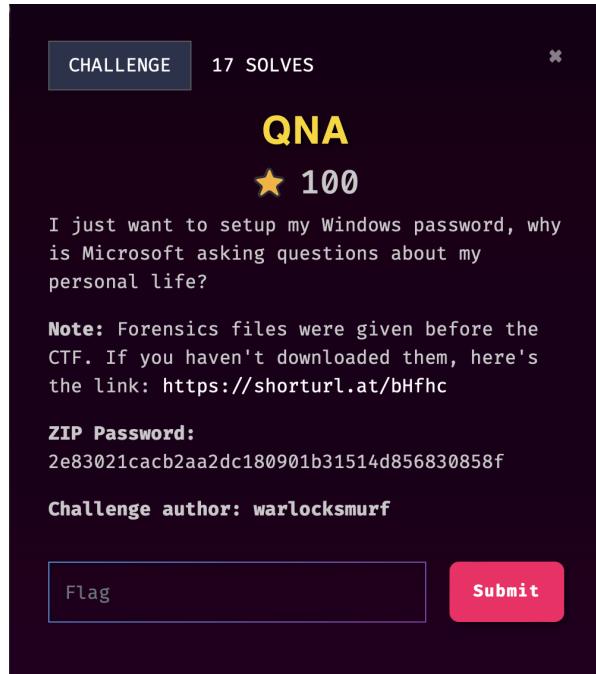


The screenshot shows a web browser window with the URL en.wikipedia.org in the address bar. The main content is the Wikipedia article for the "United States Court of Appeals for the Ninth Circuit". The page title is "United States Court of Appeals for the Ninth Circuit". Below the title, there are tabs for "Article" and "Talk". The main text of the article states: "The United States Court of Appeals for the Ninth Circuit (in case citations, 9th Cir.) is the U.S. federal court of appeals that has appellate jurisdiction over the". A callout box highlights the text "United States Court of Appeals for the Ninth Circuit". The browser interface includes a search bar, a sidebar with navigation links like "Contents", "History", and "Criticism", and various browser extensions at the top.

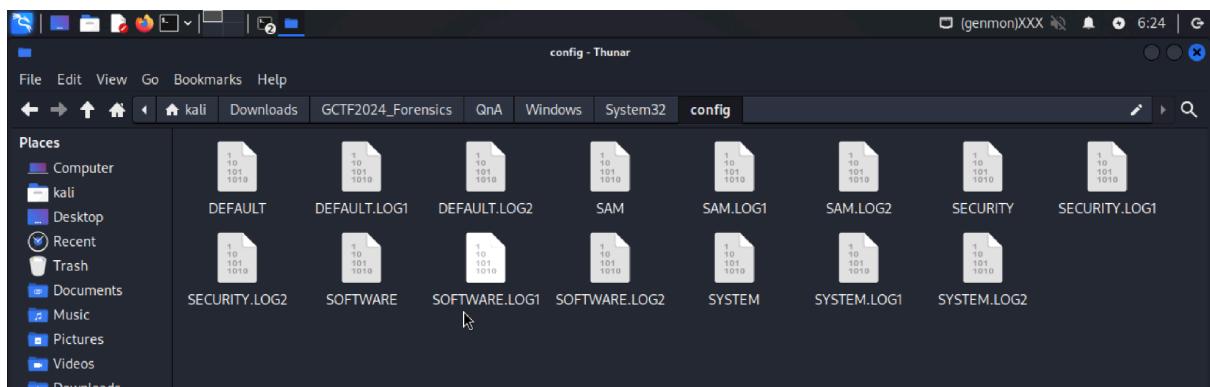
Flag: gctf{United_States_Court_of_Appeals_for_the_Ninth_Circuit}

FORENSIC

OnA



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

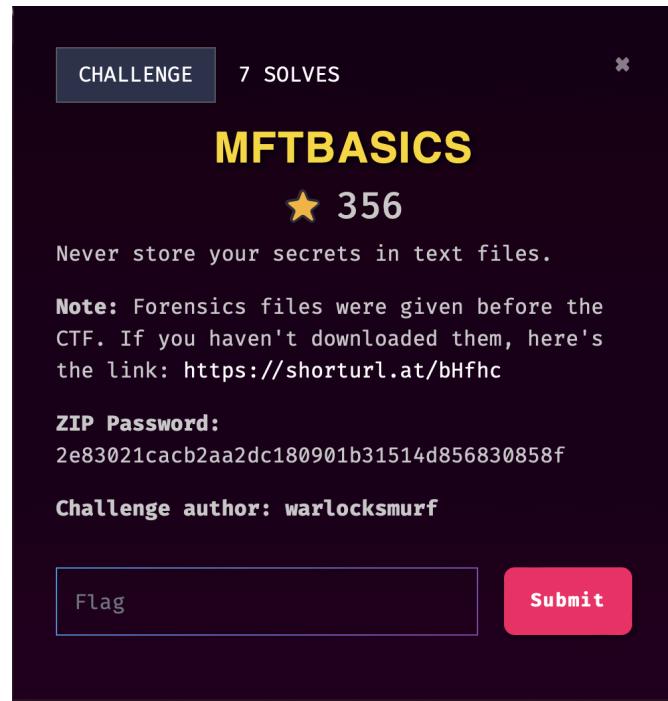


I used the command cat SAM to see the content and finally found the flag which was divided into 3 parts of the questions.

```
Administrator have complete and unrestricted access to the computer/domain♦♦M♦K♦♦♦♦nk #♦♦♦♦`  
♦♦♦♦♦♦H  
♦♦♦♦Users X@♦♦{'version':1,'questions':[{'question':'What was your first pet s name?','answer':'GCTF{p3rs0nal}'},{  
'question':'What s the name of the city where you were born?','answer':'_s3cr3ts_'},{'question':'What was your childhood nickname?','answer':'r3g1stry'}]}♦♦I♦!♦g♦*♦!  
T=♦wYyyv♦e}♦2♦*p_♦2♦*P♦*R♦*Yf*Is+♦pw♦*♦*♦*env-*C*(cpv*♦2♦*9♦*j1*♦z♦*♦  
HUV,♦*♦*G*3*8{♦*♦*♦*-*1yVf*M7*♦*j*♦*z*Q*♦*1*{♦*I*♦*f*♦}*s*♦*W*♦*o*er:g*♦*V*]♦*yC*♦*a*G]*♦*  
-♦*18b*J*♦*W*♦*t*♦*x*♦*K*♦*M*♦*3*=J*♦*2*E  
-♦*o*er:♦*s*♦*h*♦*e*♦*A*?X*}♦*s*♦*C*♦*e*♦*c*♦*Y*?*E
```

Flag: GCTF{p3rs0nal_s3cr3ts_r3g1stry}

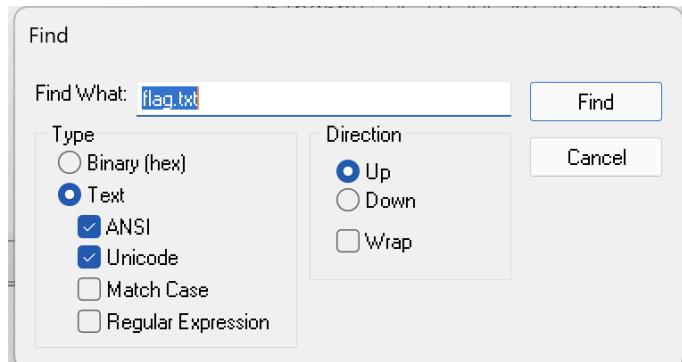
MFTBasics



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

The screenshot shows the FTK Imager interface. The Evidence Tree pane shows two entries: 'mft2csv_v2.0.0.13' and 'foxsecrets.aid'. The File List pane shows a single file named '2024-10-12-09-08-42.log' with a size of 1,095 bytes and a date modified of 10/12/2024 12:31 AM. The main area displays the raw hex and ASCII data of the '\$MFT' file. The Hex Value Interpreter pane at the bottom shows the raw binary data. The status bar at the bottom right indicates 'Activate Windows'.

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:

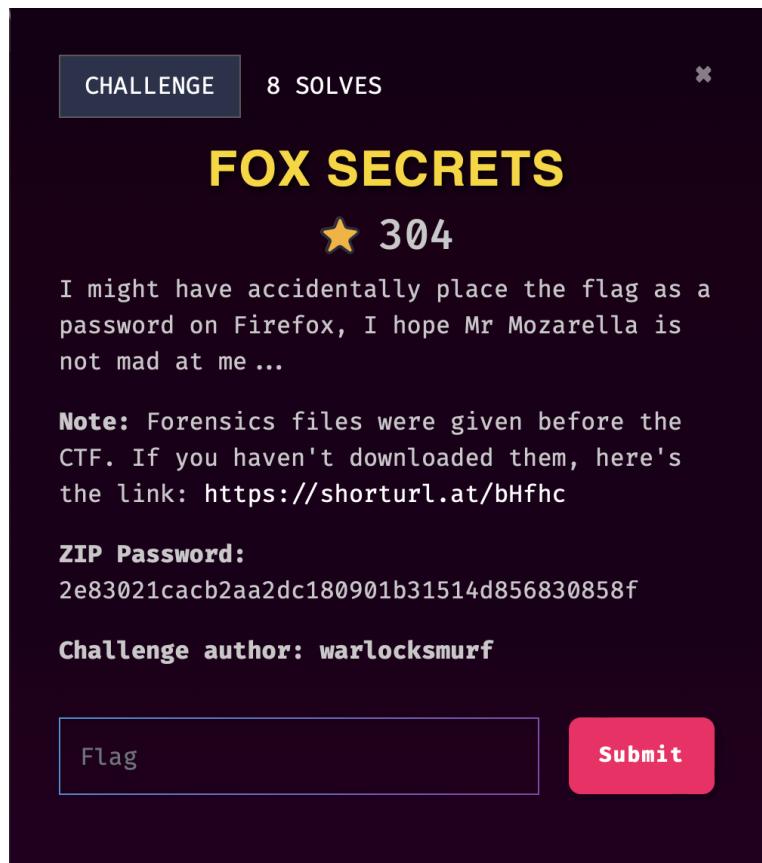
The secret code is: 47 43 54 46 7b 62 34 73 31 63 73 5f 30 66 5f 4d 46 54 7d 4d

Address	Hex Value	Decoded Value	Description
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..ki..<>..)"..	
6fadd30	80 00 00 00 68 00 00 00-00 00 18 00 00 00 01 00h.....	
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 82 79 47 11-00 00 00 00 18 00 00 00	ÿÿÿÿ·yG.....	
6faddb0	FF FF FF 82 79 47 11-00 00 00 00 00 00 00 00 00	ÿÿÿÿ·yG.....	
6faddc00	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

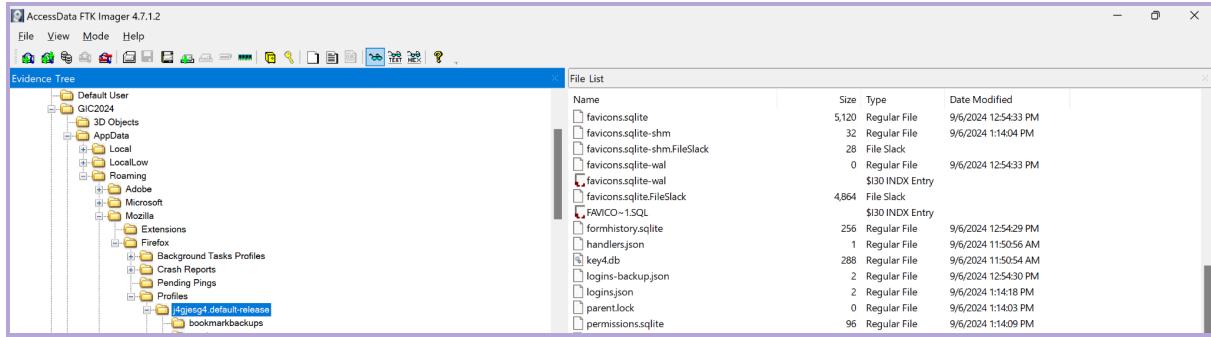


Mozilla Support
<https://support.mozilla.org> :

Profiles - Where Firefox stores your bookmarks, passwords ...

Firefox stores your profile folder in this location on your computer, by default: C:\Users\<your Windows login username>\AppData\Roaming\Mozilla\Firefox\ ...

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

```

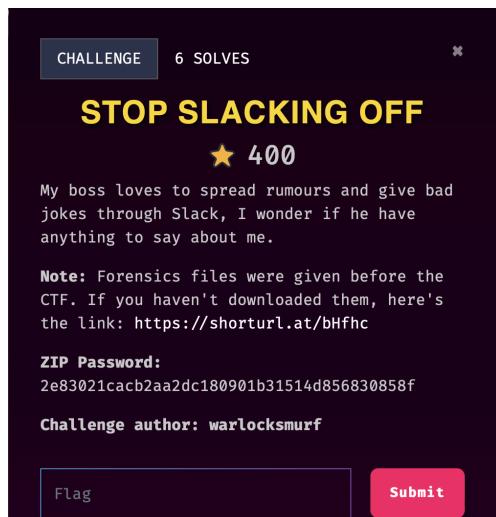
sh: no such file or directory: https://github.com/lclevy/firepwd
(base) sayyidahnafishah@Sayyidahs-MacBook-Air fox secret % git clone https://github.com/lclevy/firepwd
Cloning into 'firepwd'...
remote: Enumerating objects: 88, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 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) sayyidahnafishah@Sayyidahs-MacBook-Air fox secret % ls
firepwd    key4.db    logins.json
(base) sayyidahnafishah@Sayyidahs-MacBook-Air fox secret % cd firepwd
(base) sayyidahnafishah@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) sayyidahnafishah@Sayyidahs-MacBook-Air firepwd % python3 firepwd
python3: can't open file '/Users/sayyidahnafishah/Desktop/shared/fox secret/firepwd/firepwd': [Errno 2] No such file or directory
(base) sayyidahnafishah@Sayyidahs-MacBook-Air firepwd % python3 firepwd.py
globalsalt: b'b5dbfec66b891e19f3d16eccaf39209a93d4332'
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'ea234484d176f2f091e1a9b2162b550a9874bf9ced92daa19c43e058b1328cf'
        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'c361eb1332fd6004ad463de0ef2'
    }
  }
}
OCTETSTRING b'c2cb6eb12879f4c649458e59d634f355'
}
clearText b'70617373776f72642d636865636b0282'
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'ae2720e0964ce4beabedba40345712df4234f9ac9cd86e53a08982b65abbb9c'
        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
}

```

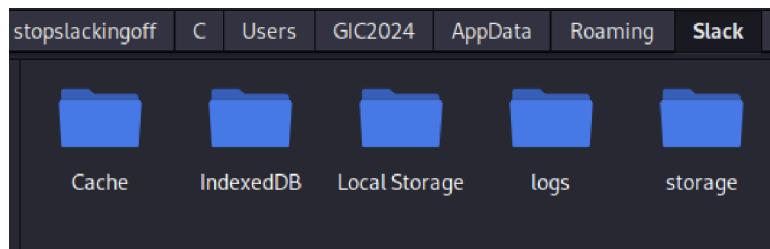
Finally we cracked the password and retrieved the flag after doing all the steps.

Flag:GCTF{m0zarella_f1ref0x_p4ssw0rd}

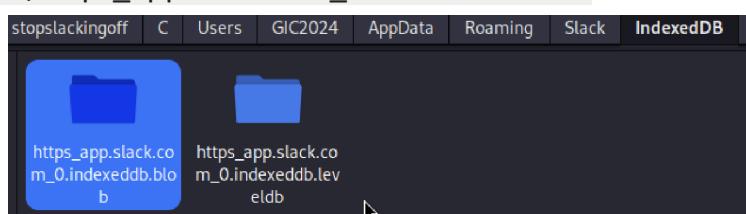
Stop Slacking Off



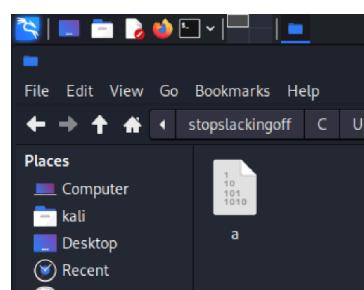
There are 5 folders in the Slack directory. I actually checked and cat all the files in those folders.



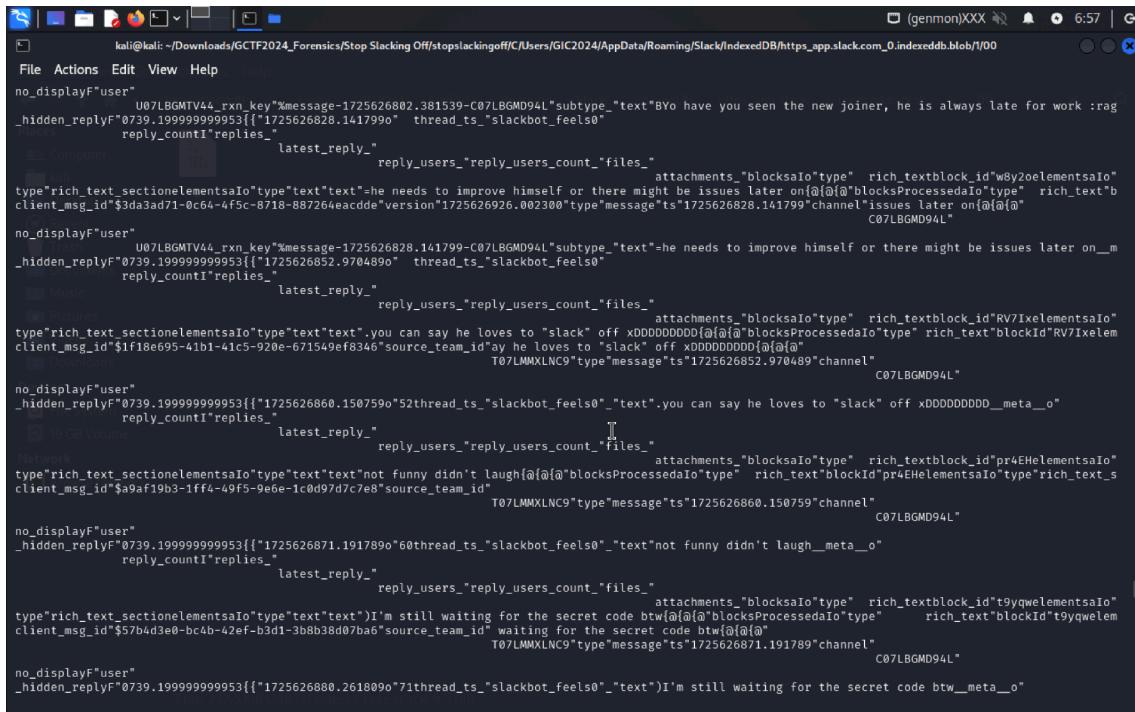
After carefully checked the content of the files, I finally found the conversations on /Slack/IndexedDB/https_app.slack.com_0.indexeddb.blob



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.

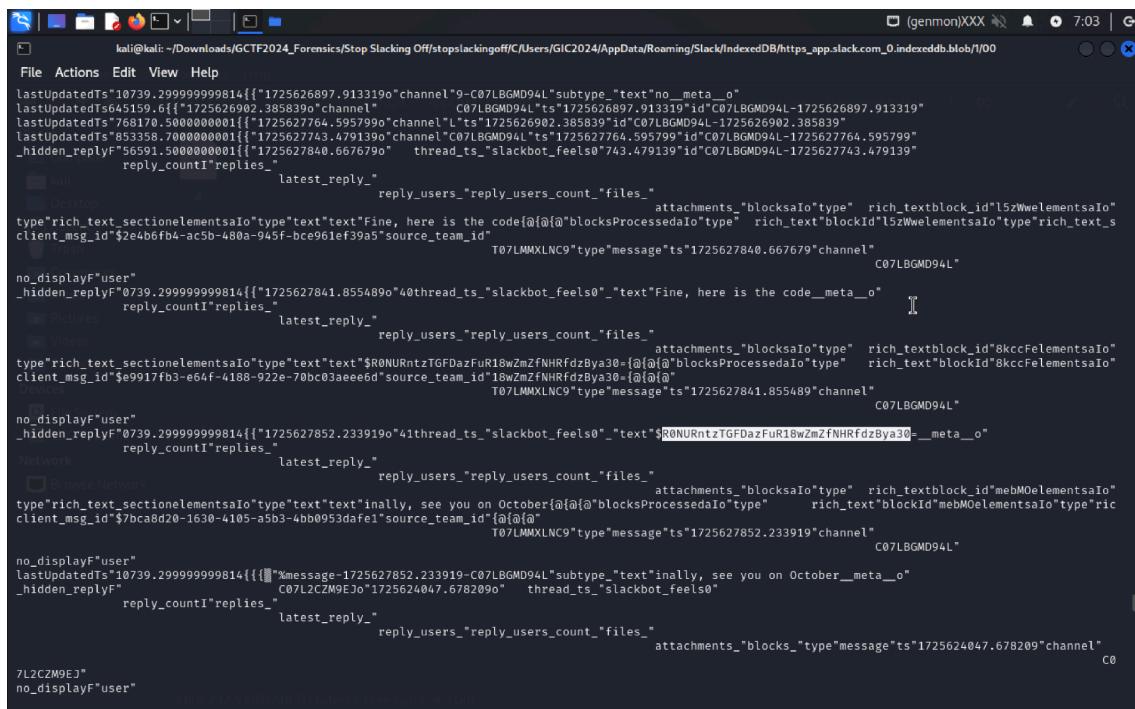


```
kali㉿kali: ~/Downloads/GCTF2024_Forensics/Stop Slacking Off/stopslackingoff/C/Users/GIC2024/AppData/Roaming/Slack/IndexedDB/https_app.slack.com_0_indexeddb.blob/f/00
File Actions Edit View Help
no_displayF"user"
    U07LBGMTV44_rxn_key"%message-1725626802.381539-C07LBGMD94L"subtype_"text"BYo have you seen the new joiner, he is always late for work :rag
_no_hidden_replyF"0739.199999999953[{"1725626828.1417990" "thread_ts_"slackbot_feels0"
    reply_countI"replies_"
        latest_reply_
            reply_users_"reply_users_count_files_"
                attachments_"blocksA0"type" rich_textblock_id"w8y2oelementsA0"
type"rich_text_sectionelementsA0"type"text"text"=he needs to improve himself or there might be issues later on@{@{@blocksProcessedA0"type" rich_text"b
client_msg_id"$3da3d71-0c64-4fc-8718-887264eacdde"version"1725626926.00230"message"ts"1725626828.141799"channel"issues later on@{@{@
_c07LBGMD94L"
no_displayF"user"
    U07LBGMTV44_rxn_key"%message-1725626802.381539-C07LBGMD94L"subtype_"text"=he needs to improve himself or there might be issues later on_m
_no_hidden_replyF"0739.199999999953[{"1725626852.9704890" "thread_ts_"slackbot_feels0"
    reply_countI"replies_"
        latest_reply_
            reply_users_"reply_users_count_files_"
                attachments_"blocksA0"type" rich_textblock_id"RV7IxelementsA0"
type"rich_text_sectionelementsA0"type"text"text"=you can say he loves to "slack" off xDDDDDDDDD@{@{@blocksProcessedA0"type" rich_text"blockId"RV7Ix elem
client_msg_id$1f18e695-41b1-4c5-920e-671549ef8346"source_team_id"ay he loves to "slack" off xDDDDDDDDD@{@{@
T07LMMXLNC9"type"message"ts"1725626852.970489"channel"
C07LBGMD94L"
no_displayF"user"
_no_hidden_replyF"0739.199999999953[{"1725626860.1507590"52"thread_ts_"slackbot_feels0"_"text".you can say he loves to "slack" off xDDDDDDDDD__meta_o"
    reply_countI"replies_"
        latest_reply_
            reply_users_"reply_users_count_files_"
                attachments_"blocksA0"type" rich_textblock_id"pr4EHelementsA0"
type"rich_text_sectionelementsA0"type"text"text"not funny didn't laugh@{@{@blocksProcessedA0"type" rich_text"blockId"pr4EHelementsA0"type"rich_text_s
client_msg_id$a9af19b3-1ff4-49f5-9e6e-1c0d97d7c7e8"source_team_id"
T07LMMXLNC9"type"message"ts"1725626860.150759"channel"
C07LBGMD94L"
no_displayF"user"
_no_hidden_replyF"0739.199999999953[{"1725626871.1917890"60"thread_ts_"slackbot_feels0"_"text"not funny didn't laugh__meta_o"
    reply_countI"replies_"
        latest_reply_
            reply_users_"reply_users_count_files_"
                attachments_"blocksA0"type" rich_textblock_id"t9yqwelementsA0"
type"rich_text_sectionelementsA0"type"text"text"=I'm still waiting for the secret code btw@{@{@blocksProcessedA0"type" rich_text"blockId"t9yqwelem
client_msg_id$57b4d3e0-bc4b-42ef-b3d1-3bb38d07ba6"source_team_id" waiting for the secret code btw@{@{@
T07LMMXLNC9"type"message"ts"1725626871.191789"channel"
C07LBGMD94L"
no_displayF"user"
_no_hidden_replyF"0739.199999999953[{"1725626880.2618090"71"thread_ts_"slackbot_feels0"_"text")I'm still waiting for the secret code btw__meta_o"
    reply_countI"replies_"
        latest_reply_
            reply_users_"reply_users_count_files_"

```

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

RONURntzTGF DazFuR18wZmZfNHRfdzBya30



```
kali㉿kali: ~/Downloads/GCTF2024_Forensics/Stop Slacking Off/stopslackingoff/C/Users/GIC2024/AppData/Roaming/Slack/IndexedDB/https_app.slack.com_0_indexeddb.blob/f/00
File Actions Edit View Help
no_displayF"user"
lastUpdatedTs"10739.29999999814[{"1725626897.9133190"channel"9-C07LBGMD94L"subtype_"text"no__meta_o"
lastUpdatedTs"10739.29999999814[{"1725626902.3858390"channel"
C07LBGMD94L"ts"1725626897.913319"id"C07LBGMD94L-1725626897.913319"
lastUpdatedTs"768170.5000000001[{"1725627764.5957990"channel"1"ts"1725627764.595799"id"C07LBGMD94L-1725627764.595799"
lastUpdatedTs"768170.5000000001[{"1725627743.4791390"channel"1"ts"1725627743.479139"id"C07LBGMD94L-1725627743.479139"
_no_hidden_replyF"5691.5000000001[{"1725627840.6676790" "thread_ts_"slackbot_feels0"743.479139"id"C07LBGMD94L-1725627743.479139"
    reply_countI"replies_"
        latest_reply_
            reply_users_"reply_users_count_files_"
                attachments_"blocksA0"type" rich_textblock_id"l5zWwelementsA0"
type"rich_text_sectionelementsA0"type"text"text"Fine, here is the code@{@{@blocksProcessedA0"type" rich_text"blockId"l5zWwelementsA0"type"rich_text_s
client_msg_id$2e4b6fb4-ac5b-480a-945f-bce961ef39a5"source_team_id"
T07LMMXLNC9"type"message"ts"1725627840.667679"channel"
C07LBGMD94L"
no_displayF"user"
_no_hidden_replyF"0739.29999999814[{"1725627841.8554890"40"thread_ts_"slackbot_feels0"_"text"fine, here is the code__meta_o"
    reply_countI"replies_"
        latest_reply_
            reply_users_"reply_users_count_files_"
                attachments_"blocksA0"type" rich_textblock_id"8kccFelementsA0"
type"rich_text_sectionelementsA0"type"text"text">$RONURntzTGF DazFuR18wZmZfNHRfdzBya30=@{@{@blocksProcessedA0"type" rich_text"blockId"8kccFelementsA0"
client_msg_id$9917fb3-e64f-4188-922e-70bc03aeee6"source_team_id"18wZmZfNHRfdzBya30=@{@{@
T07LMMXLNC9"type"message"ts"1725627841.855489"channel"
C07LBGMD94L"
no_displayF"user"
_no_hidden_replyF"0739.29999999814[{"1725627852.2339190"41"thread_ts_"slackbot_feels0"_"text">$RONURntzTGF DazFuR18wZmZfNHRfdzBya30=__meta_o"
    reply_countI"replies_"
        latest_reply_
            reply_users_"reply_users_count_files_"
                attachments_"blocksA0"type" rich_textblock_id"mebMoelementsA0"
type"rich_text_sectionelementsA0"type"text"text"inally, see you on October@{@{@blocksProcessedA0"type" rich_text"blockId"mebMoelementsA0"type"ric
client_msg_id$7bca8d20-1630-4105-a5b3-4bb0953dafe1"source_team_id">@{@{@
T07LMMXLNC9"type"message"ts"1725627852.233919"channel"
C07LBGMD94L"
no_displayF"user"
lastUpdatedTs"10739.29999999814[{"1725627852.233919-C07LBGMD94L"subtype_"text"inally, see you on October__meta_o"
_no_hidden_replyF" C07LB2CZM9EJ"0725624047.678209" "thread_ts_"slackbot_feels0"
    reply_countI"replies_"
        latest_reply_
            reply_users_"reply_users_count_files_"
                attachments_"blocksA0"type" message"ts"1725624047.678209"channel"
C07LB2CZM9EJ"
7L2CZM9EJ"
no_displayF"user"

```

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

The screenshot shows the CyberChef interface with the following details:

- Input:** R0NURntzTGFdazFuR18wZmZfNHRfdzBya30
- Recipe:** Magic (Depth 3, Intensive mode checked)
- Output:** Four rows of decoded text:
 - From_Base85('0-9a-zA-Z.\\";:+=~!/*?&>()[]{}@%\$#') Decode_text('IBM EBCDIC French (1010)')
 - From_Base64('A-Za-z0-9_+',true,false)
 - From_Base64('A-Za-z0-9_+',true,false)
 - From_Base85('0-9a-zA-Z.\\";:+=~!/*?&>()[]{}@%\$#') Decode_text('IAS German (7-bit) (20106)')Each row includes hex dump, raw bytes, entropy, and validation information (Valid UTF8, Entropy: 2.56, Matching ops: From Base85, etc.).
- Buttons:** STEP, BAKE!, Auto Bake

Flag: GCTF{sLaCk1nG_0ff_4t_w0rk}