RO :P #1

(Return Oriented Programming)

SCP 이예준

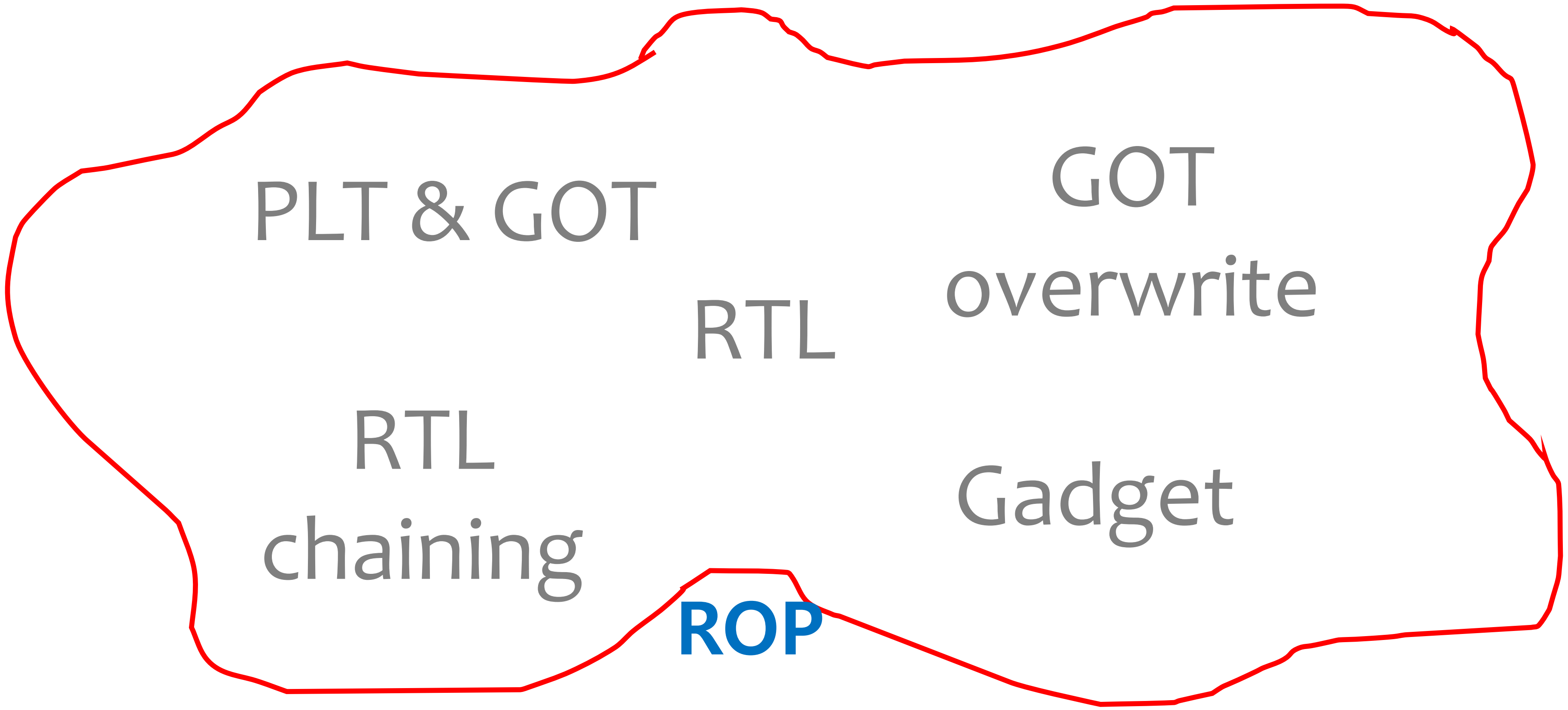
content

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

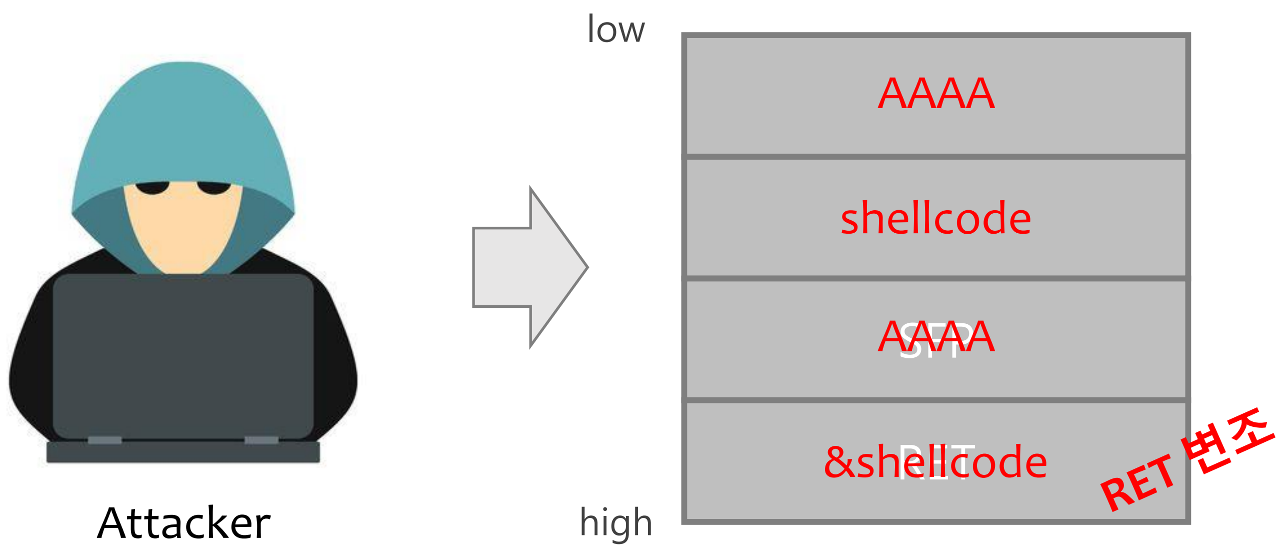
➔ **RET 가지고 프로그래밍**



Memory Protector

NX-bit (Never eXcutable bit)

프로세스 명령어나 코드 또는 데이터 저장을 위한 메모리 영역을 따로 분리하는 CPU의 기술
NX-bit가 적용된 모든 메모리 구역은 데이터 저장을 위해서만 사용



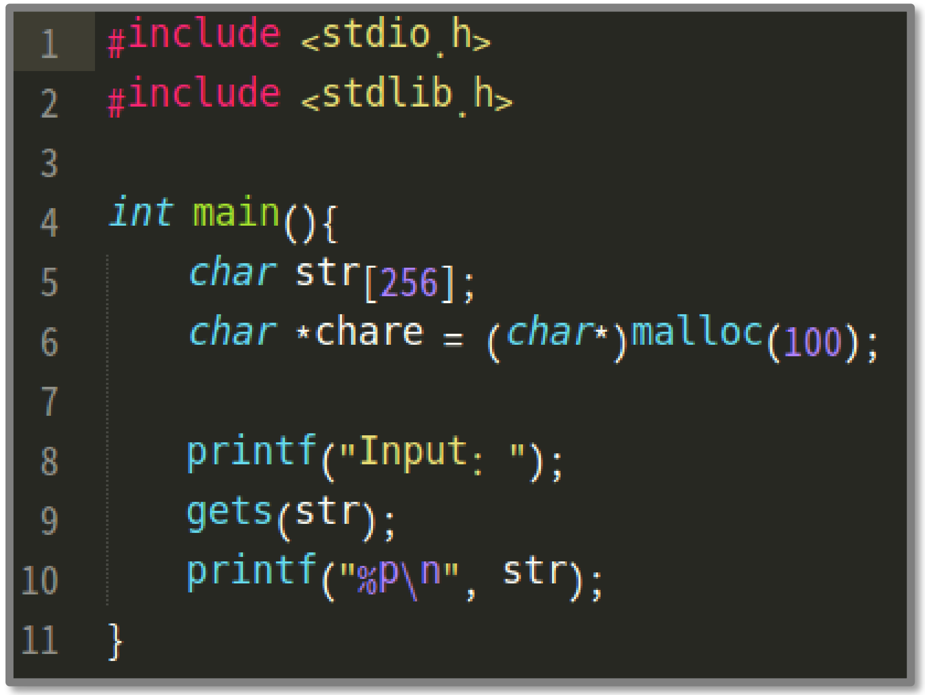
NX-bit enable

실행권한 X

프로그램 예외 처리 후 종료

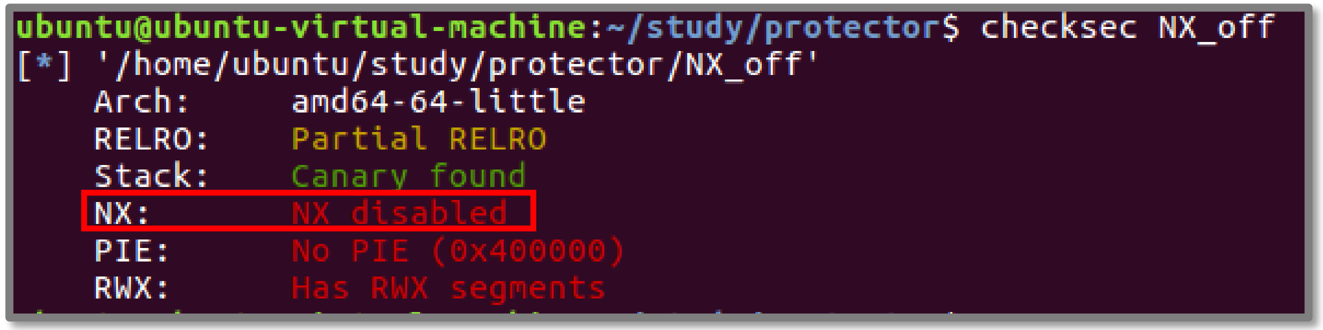
› NX-bit (Never eXcutable bit)

<예제 코드>



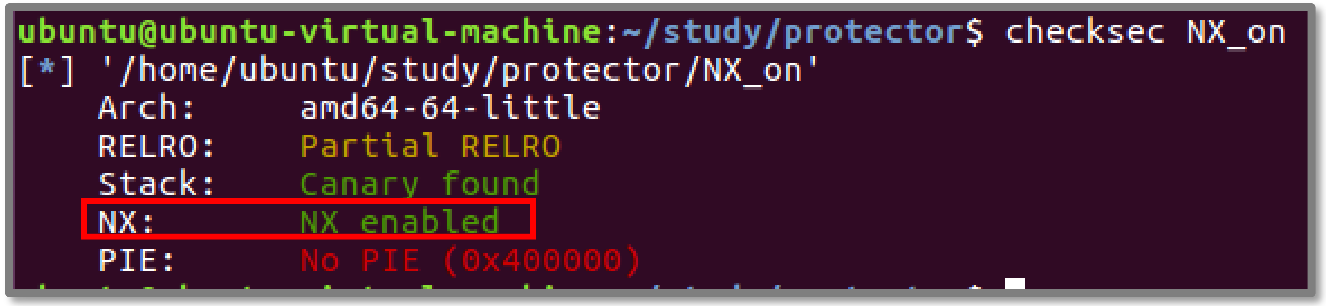
```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    char str[256];
    char *chare = (char*)malloc(100);

    printf("Input: ");
    gets(str);
    printf("%p\n", str);
}
```

<NX disabled>



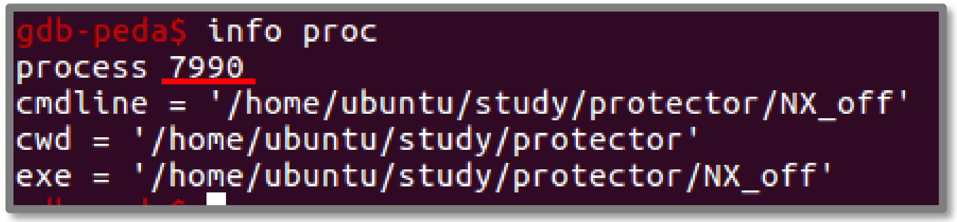
```
ubuntu@ubuntu-virtual-machine:~/study/protector$ checksec NX_off
[*] '/home/ubuntu/study/protector/NX_off'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
```

<NX enabled>



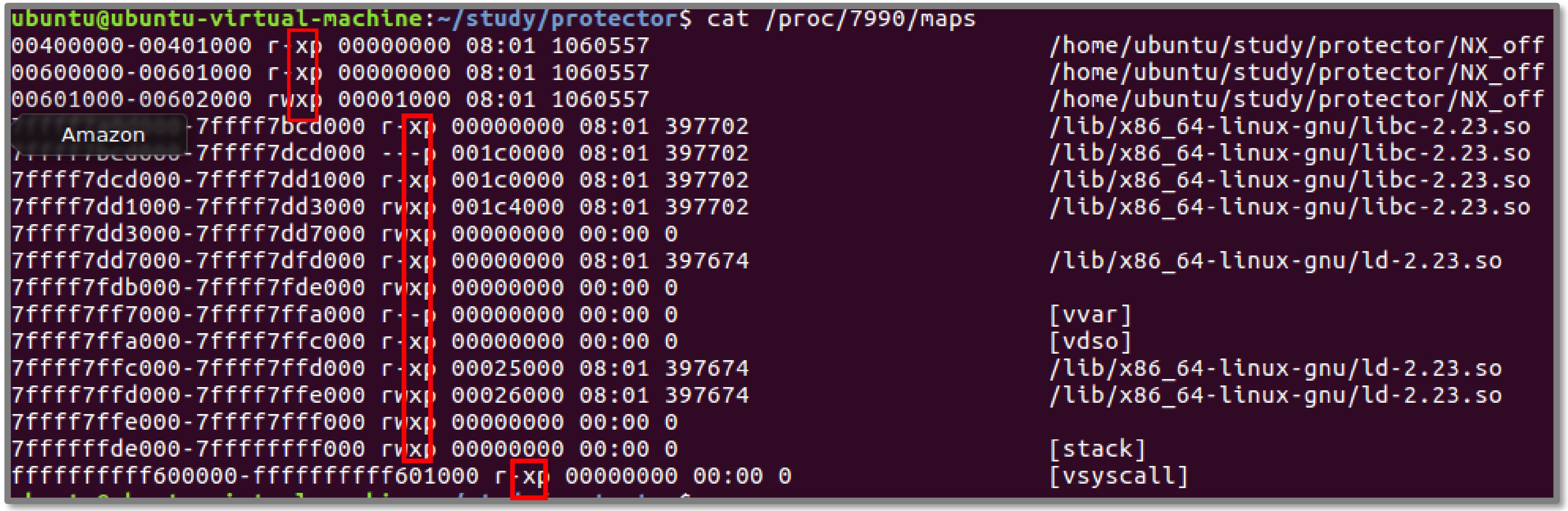
```
ubuntu@ubuntu-virtual-machine:~/study/protector$ checksec NX_on
[*] '/home/ubuntu/study/protector/NX_on'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

› NX-bit (Never eXcutable bit)



```
gdb-peda$ info proc
process 7990
cmdline = '/home/ubuntu/study/protector/NX_off'
cwd = '/home/ubuntu/study/protector'
exe = '/home/ubuntu/study/protector/NX_off'
```

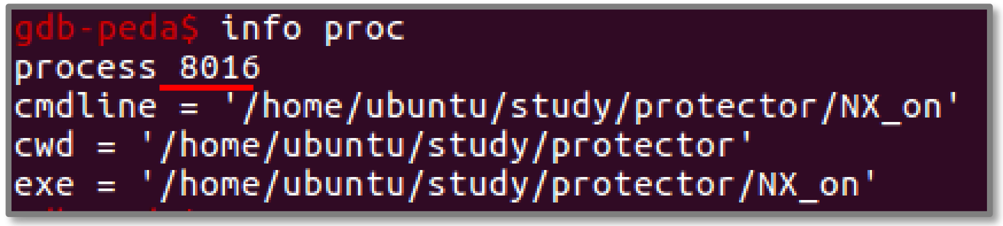
NX_disable



```
ubuntu@ubuntu-virtual-machine:~/study/protector$ cat /proc/7990/maps
00400000-00401000 r-xp 00000000 08:01 1060557                            /home/ubuntu/study/protector/NX_off
00600000-00601000 r-xp 00000000 08:01 1060557                            /home/ubuntu/study/protector/NX_off
00601000-00602000 rwxp 00001000 08:01 1060557                            /home/ubuntu/study/protector/NX_off
             -7ffff7bcd000 r-xp 00000000 08:01 397702                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7bcd000-7ffff7dcd000 ---p 001c0000 08:01 397702                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dcd000-7ffff7dd1000 r-xp 001c0000 08:01 397702                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dd1000-7ffff7dd3000 rwxp 001c4000 08:01 397702                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dd3000-7ffff7dd7000 rwxp 00000000 00:00 0
7ffff7dd7000-7ffff7dfd000 r-xp 00000000 08:01 397674                    /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7fdb000-7ffff7fde000 rwxp 00000000 00:00 0
7ffff7ff7000-7ffff7ffa000 r--p 00000000 00:00 0                          [vvar]
7ffff7ffa000-7ffff7ffc000 r-xp 00000000 00:00 0                          [vdso]
7ffff7ffc000-7ffff7ffd000 r-xp 00025000 08:01 397674                    /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7ffd000-7ffff7ffe000 rwxp 00026000 08:01 397674                    /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7ffe000-7ffff7fff000 rwxp 00000000 00:00 0
7ffffffde000-7ffffffff000 rwxp 00000000 00:00 0                          [stack]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

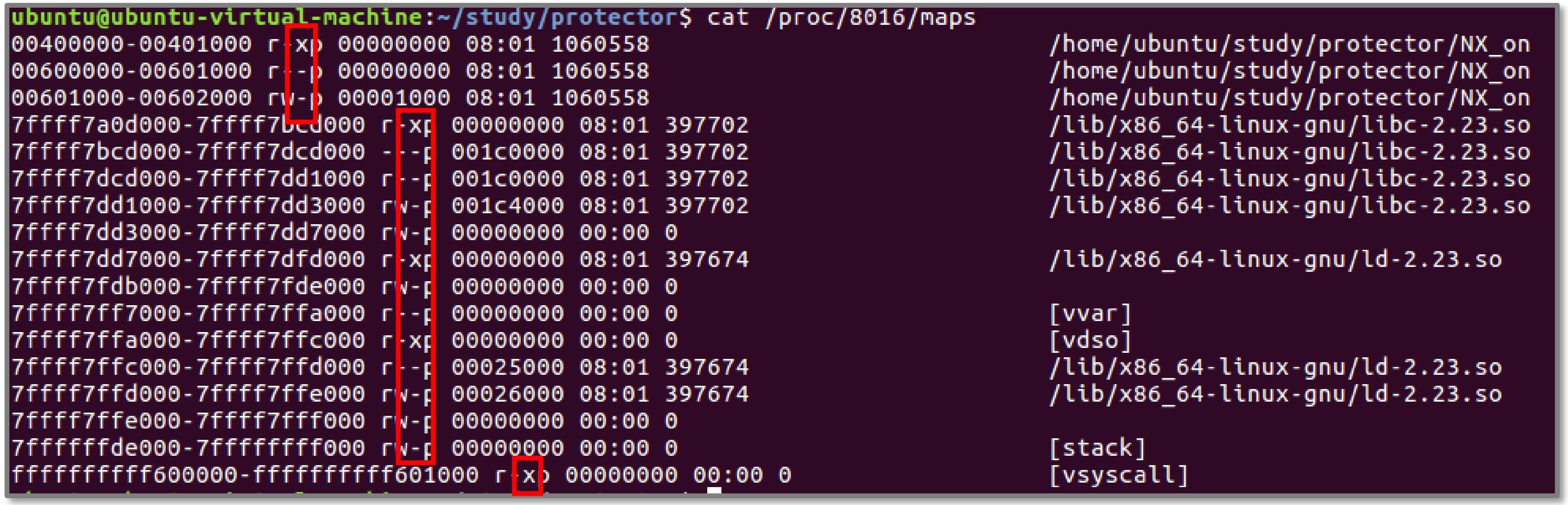
Memory Protector

› NX-bit (Never eXcutable bit)



```
gdb-peda$ info proc
process 8016
cmdline = '/home/ubuntu/study/protector/NX_on'
cwd = '/home/ubuntu/study/protector'
exe = '/home/ubuntu/study/protector/NX_on'
```

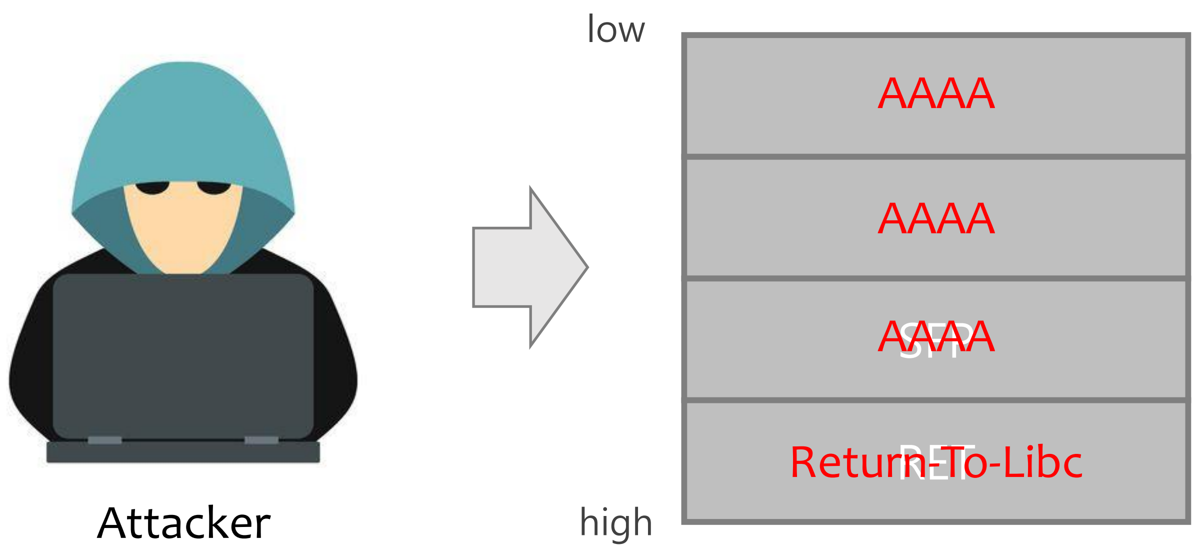
NX_enable



```
ubuntu@ubuntu-virtual-machine:~/study/protector$ cat /proc/8016/maps
00400000-00401000 r-xp 00000000 08:01 1060558                            /home/ubuntu/study/protector/NX_on
00600000-00601000 r--p 00000000 08:01 1060558                            /home/ubuntu/study/protector/NX_on
00601000-00602000 rw-p 00001000 08:01 1060558                            /home/ubuntu/study/protector/NX_on
7ffff7a0d000-7ffff7bcd000 r-xp 00000000 08:01 397702                     /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7bcd000-7ffff7dcd000 ---p 001c0000 08:01 397702                     /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dcd000-7ffff7dd1000 r--p 001c0000 08:01 397702                     /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dd1000-7ffff7dd3000 rw-p 001c4000 08:01 397702                     /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dd3000-7ffff7dd7000 rw-p 00000000 00:00 0
7ffff7dd7000-7ffff7dfd000 r-xp 00000000 08:01 397674                     /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7fdb000-7ffff7fde000 rw-p 00000000 00:00 0
7ffff7ff7000-7ffff7ffa000 r--p 00000000 00:00 0                          [vvar]
7ffff7ffa000-7ffff7ffc000 r-xp 00000000 00:00 0                          [vdso]
7ffff7ffc000-7ffff7ffd000 r--p 00025000 08:01 397674                     /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7ffd000-7ffff7ffe000 rw-p 00026000 08:01 397674                     /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7ffe000-7ffff7fff000 rw-p 00000000 00:00 0
7ffffffde000-7ffffffff000 rw-p 00000000 00:00 0                          [stack]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

ASLR (Address Space Layout Randomization)

스택, 힙, 라이브러리 등의 주소를 랜덤한 영역으로 배치
프로그램이 실행될 때 마다 각 주소 값이 변경

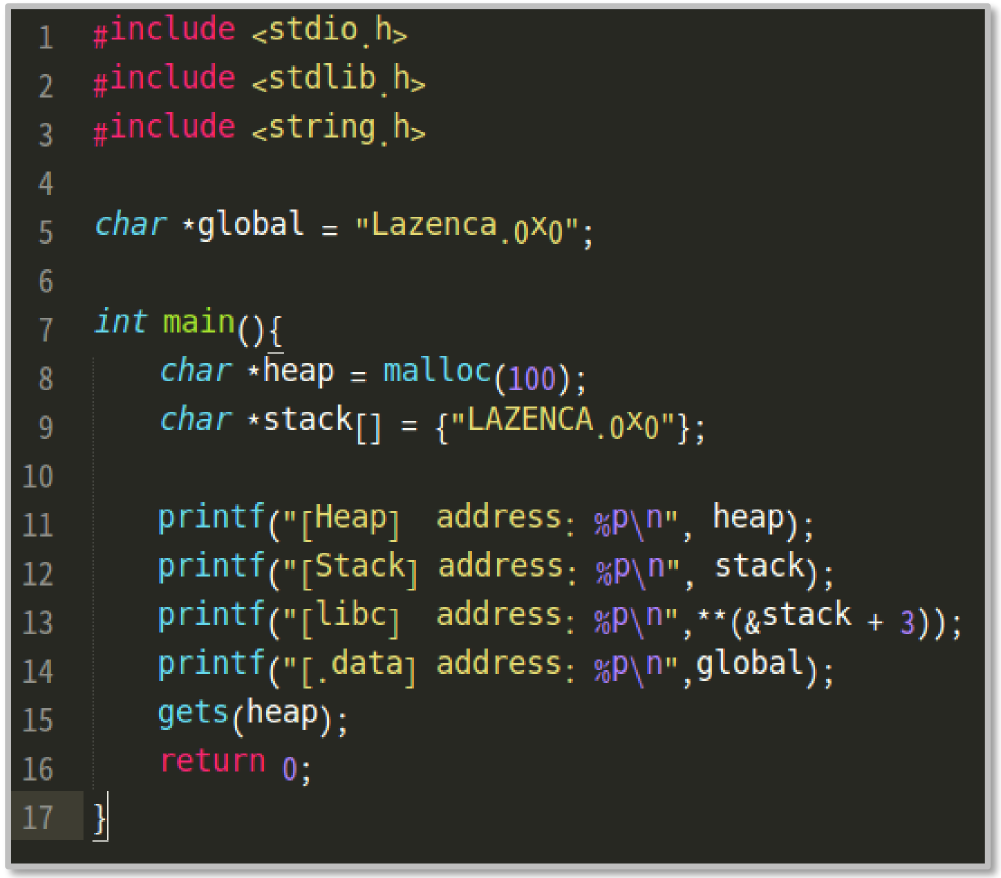


ASLR enable
공유 라이브러리 함수 주소 랜덤

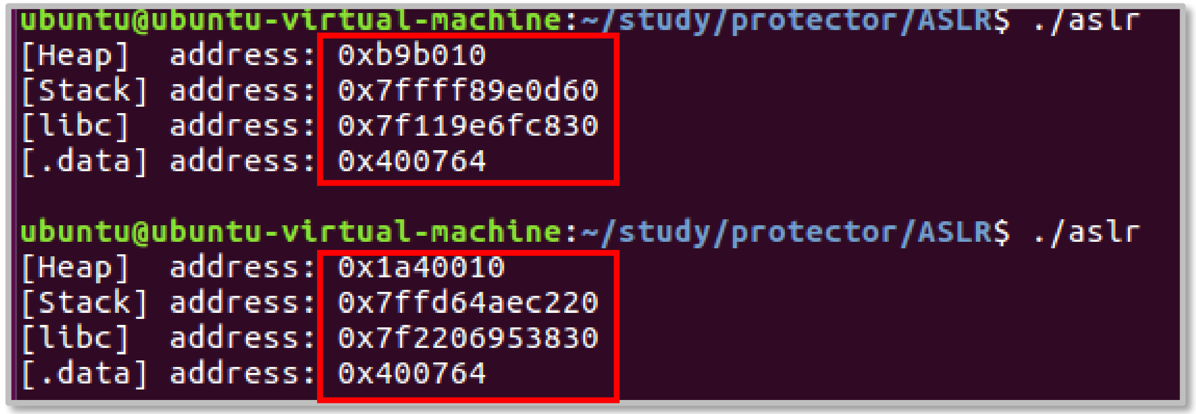
함수 호출 실패

ASLR (Address Space Layout Randomization)

<예제 코드>



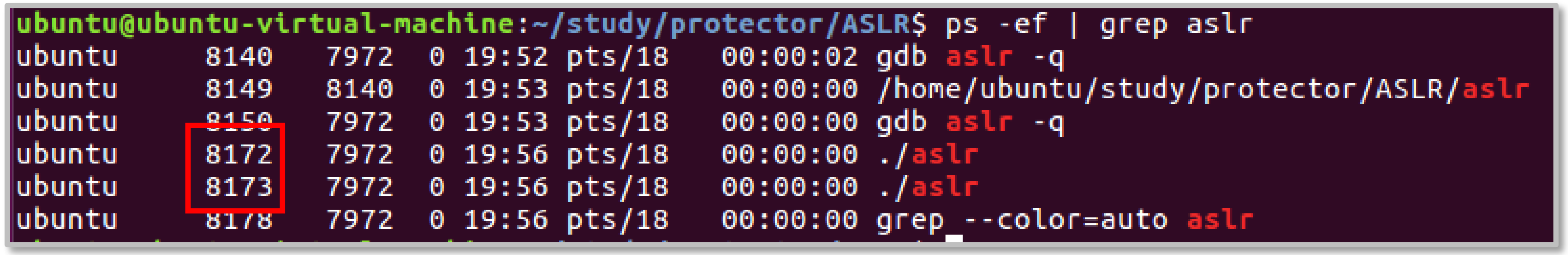
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *global = "Lazenca.0x0";

int main(){
    char *heap = malloc(100);
    char *stack[] = {"LAZENCA.0x0"};

    printf("[Heap]  address: %p\n", heap);
    printf("[Stack] address: %p\n", stack);
    printf("[libc]  address: %p\n",**(&stack + 3));
    printf("[.data] address: %p\n",global);
    gets(heap);
    return 0;
}
```



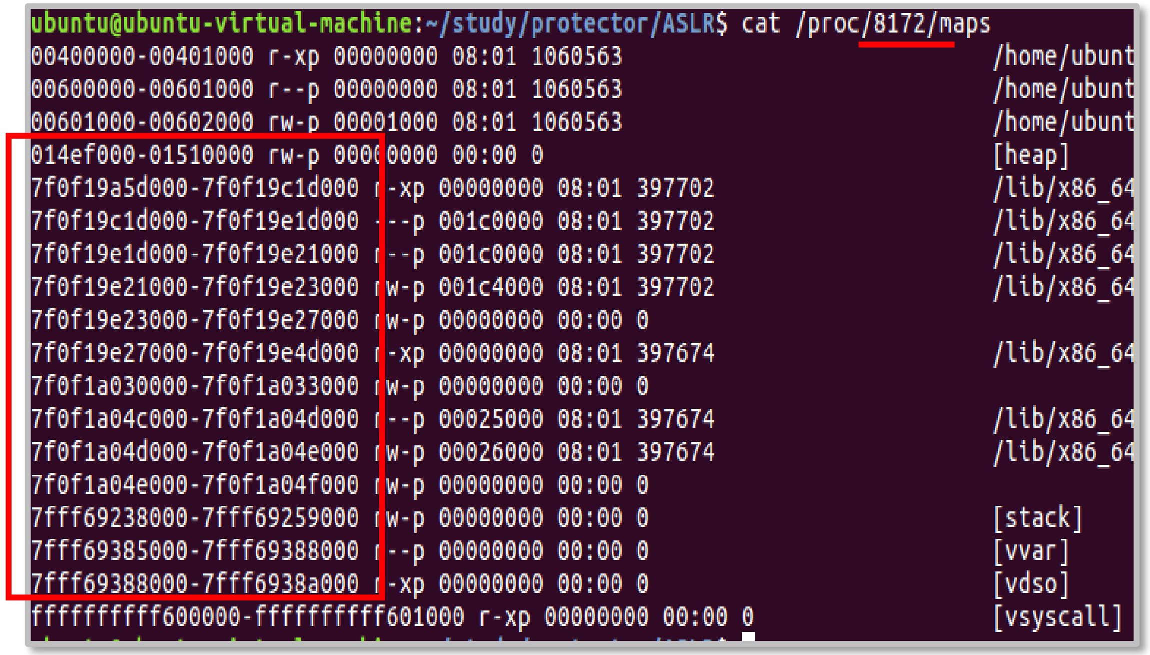
```
ubuntu@ubuntu-virtual-machine:~/study/protector/ASLR$ ./aslr
[Heap]  address: 0xb9b010
[Stack] address: 0x7ffff89e0d60
[libc]  address: 0x7f119e6fc830
[.data] address: 0x400764

ubuntu@ubuntu-virtual-machine:~/study/protector/ASLR$ ./aslr
[Heap]  address: 0x1a40010
[Stack] address: 0x7ffd64aec220
[libc]  address: 0x7f2206953830
[.data] address: 0x400764
```

Memory leak

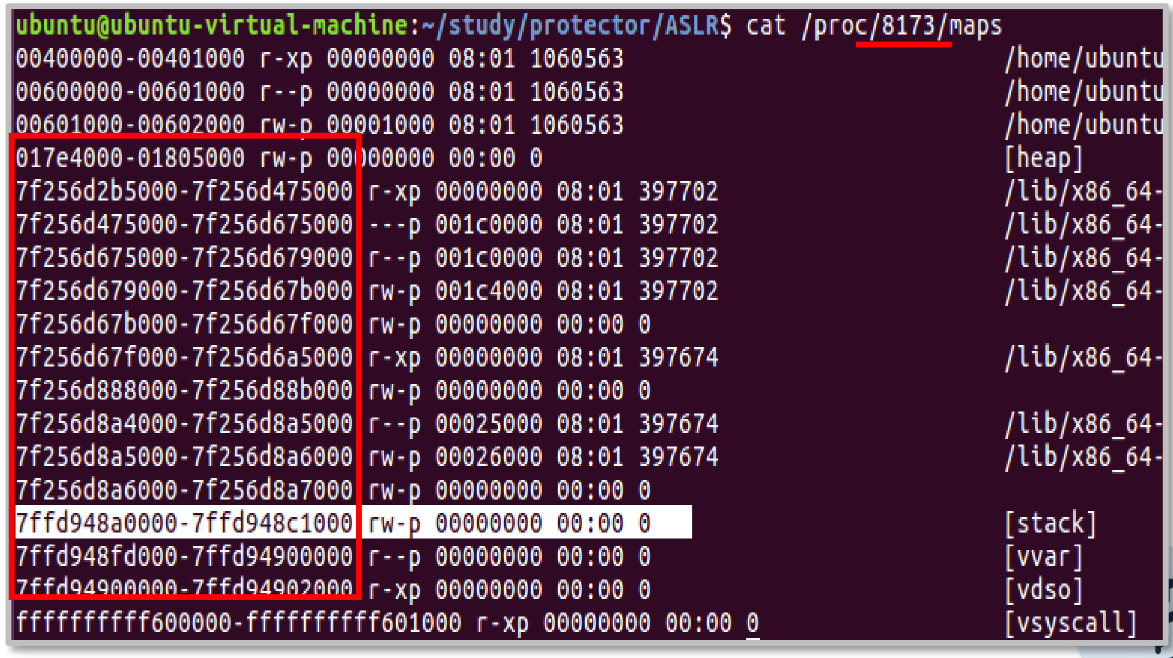
› ASLR (Address Space Layout Randomization)



```
ubuntu@ubuntu-virtual-machine:~/study/protector/ASLR$ ps -ef | grep aslr
ubuntu      8140   7972  0 19:52 pts/18   00:00:02 gdb aslr -q
ubuntu      8149   8140  0 19:53 pts/18   00:00:00 /home/ubuntu/study/protector/ASLR/aslr
ubuntu      8150   7972  0 19:53 pts/18   00:00:00 gdb aslr -q
ubuntu      8172   7972  0 19:56 pts/18   00:00:00 ./aslr
ubuntu      8173   7972  0 19:56 pts/18   00:00:00 ./aslr
ubuntu      8178   7972  0 19:56 pts/18   00:00:00 grep --color=auto aslr
```



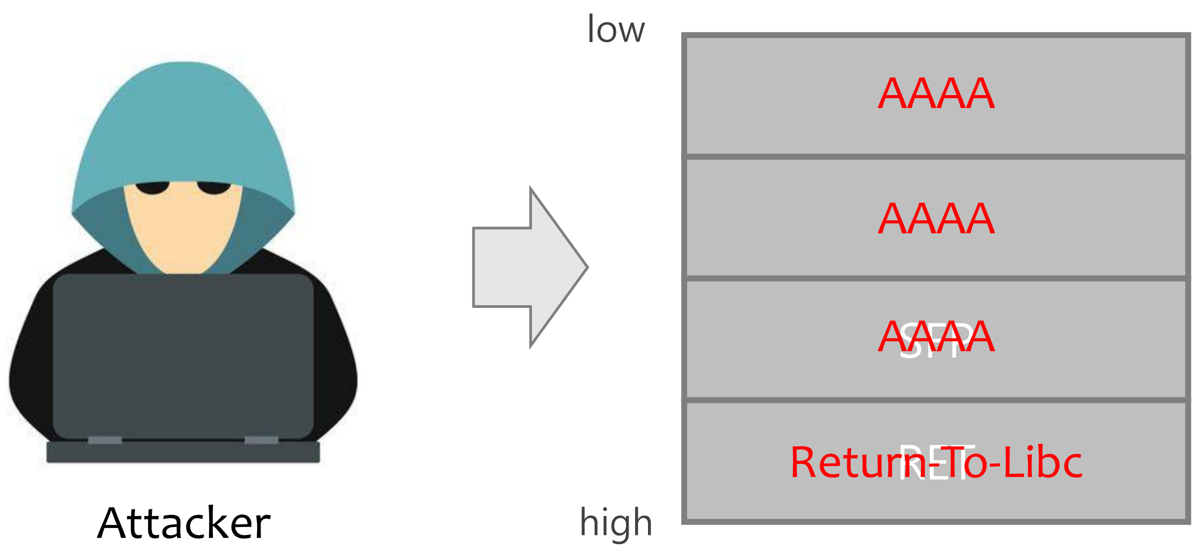
```
ubuntu@ubuntu-virtual-machine:~/study/protector/ASLR$ cat /proc/8172/maps
00400000-00401000 r-xp 00000000 08:01 1060563                    /home/ubunt
00600000-00601000 r--p 00000000 08:01 1060563                    /home/ubunt
00601000-00602000 rw-p 00001000 08:01 1060563                    /home/ubunt
014ef000-01510000 rw-p 00000000 00:00 0                          [heap]
7f0f19a5d000-7f0f19c1d000 r-xp 00000000 08:01 397702             /lib/x86_64
7f0f19c1d000-7f0f19e1d000 ---p 001c0000 08:01 397702             /lib/x86_64
7f0f19e1d000-7f0f19e21000 r--p 001c0000 08:01 397702             /lib/x86_64
7f0f19e21000-7f0f19e23000 rw-p 001c4000 08:01 397702             /lib/x86_64
7f0f19e23000-7f0f19e27000 rw-p 00000000 00:00 0
7f0f19e27000-7f0f19e4d000 r-xp 00000000 08:01 397674             /lib/x86_64
7f0f1a030000-7f0f1a033000 rw-p 00000000 00:00 0
7f0f1a04c000-7f0f1a04d000 r--p 00025000 08:01 397674             /lib/x86_64
7f0f1a04d000-7f0f1a04e000 rw-p 00026000 08:01 397674             /lib/x86_64
7f0f1a04e000-7f0f1a04f000 rw-p 00000000 00:00 0
7fff69238000-7fff69259000 rw-p 00000000 00:00 0                  [stack]
7fff69385000-7fff69388000 r--p 00000000 00:00 0                  [vvar]
7fff69388000-7fff6938a000 r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0          [vsyscall]
```



```
ubuntu@ubuntu-virtual-machine:~/study/protector/ASLR$ cat /proc/8173/maps
00400000-00401000 r-xp 00000000 08:01 1060563                    /home/ubuntu
00600000-00601000 r--p 00000000 08:01 1060563                    /home/ubuntu
00601000-00602000 rw-p 00001000 08:01 1060563                    /home/ubuntu
017e4000-01805000 rw-p 00000000 00:00 0                          [heap]
7f256d2b5000-7f256d475000 r-xp 00000000 08:01 397702             /lib/x86_64-
7f256d475000-7f256d675000 ---p 001c0000 08:01 397702             /lib/x86_64-
7f256d675000-7f256d679000 r--p 001c0000 08:01 397702             /lib/x86_64-
7f256d679000-7f256d67b000 rw-p 001c4000 08:01 397702             /lib/x86_64-
7f256d67b000-7f256d67f000 rw-p 00000000 00:00 0
7f256d67f000-7f256d6a5000 r-xp 00000000 08:01 397674             /lib/x86_64-
7f256d888000-7f256d88b000 rw-p 00000000 00:00 0
7f256d8a4000-7f256d8a5000 r--p 00025000 08:01 397674             /lib/x86_64-
7f256d8a5000-7f256d8a6000 rw-p 00026000 08:01 397674             /lib/x86_64-
7f256d8a6000-7f256d8a7000 rw-p 00000000 00:00 0
7ffd948a0000-7ffd948c1000 rw-p 00000000 00:00 0                  [stack]
7ffd948fd000-7ffd94900000 r--p 00000000 00:00 0                  [vvar]
7ffd94900000-7ffd94902000 r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0          [vsyscall]
```

› ASCII-Armor

공유 라이브러리 상위 주소에 **NULL**(0x00)바이트 삽입 -> 접근 불가

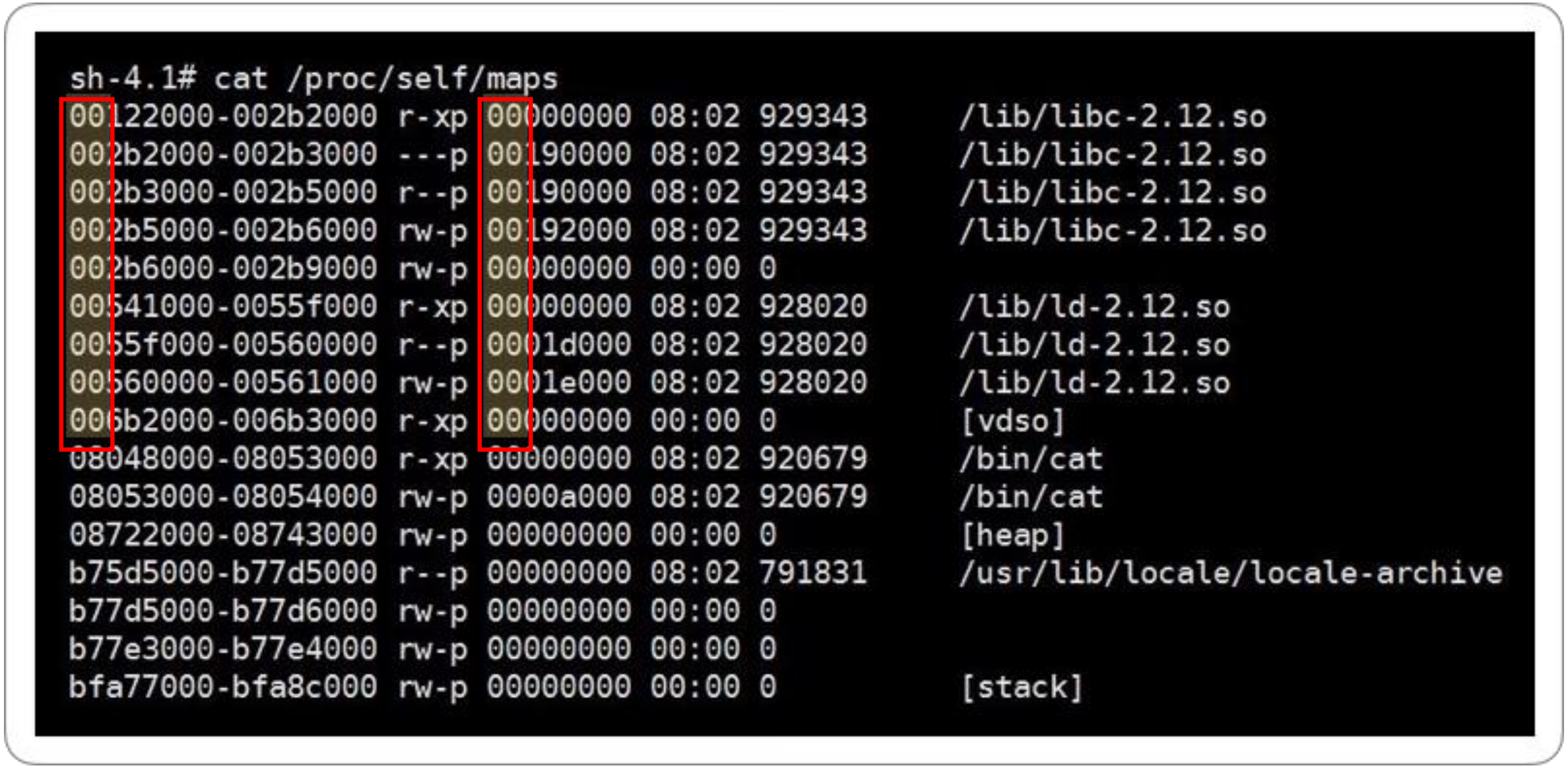


ASCII-Armor enable

문자열 종료(0x00) 인식

함수 호출 실패

› ASCII-Armor

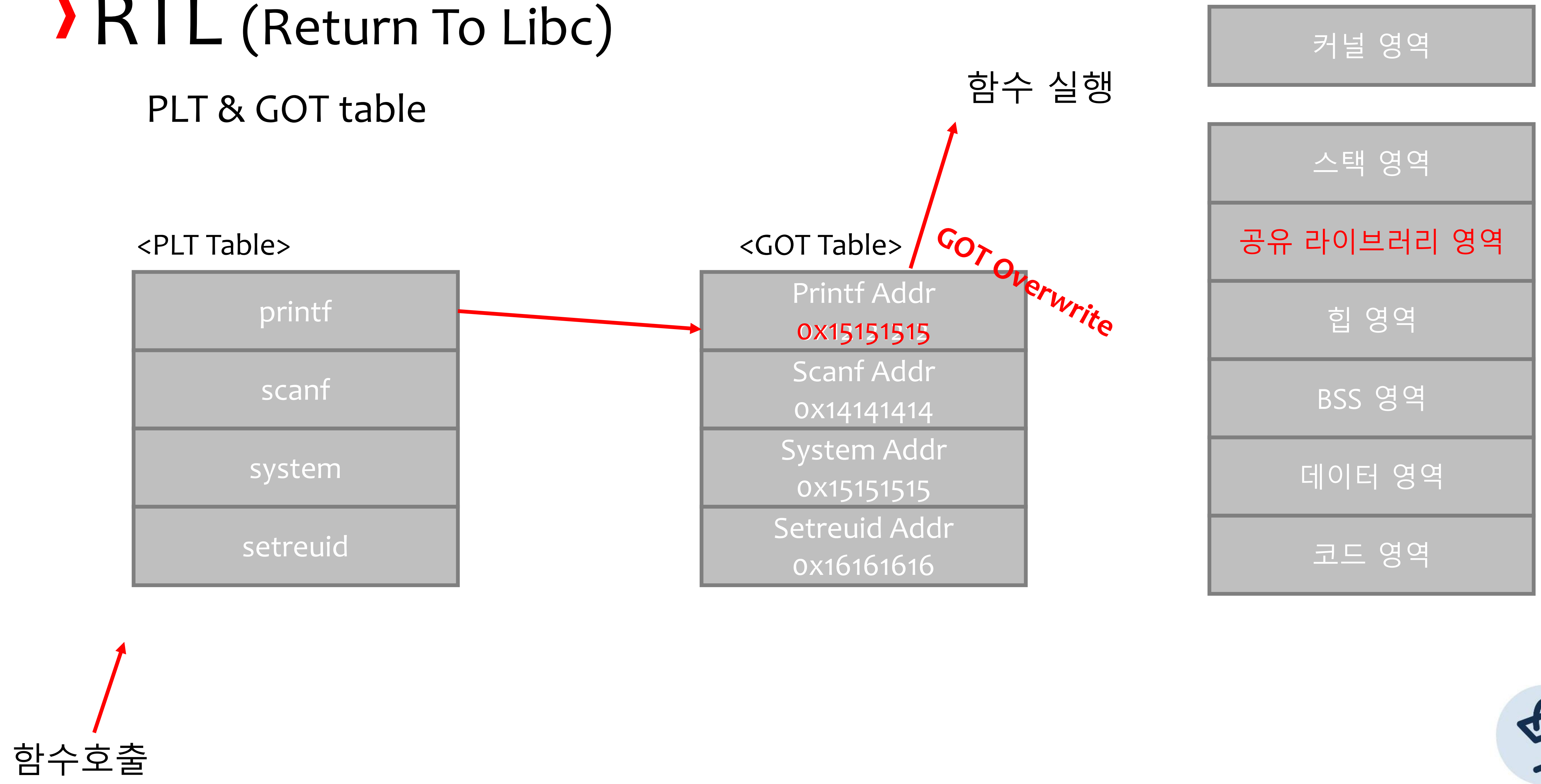


```
sh-4.1# cat /proc/self/maps
00122000-002b2000 r-xp 00000000 08:02 929343     /lib/libc-2.12.so
002b2000-002b3000 ---p 00190000 08:02 929343     /lib/libc-2.12.so
002b3000-002b5000 r--p 00190000 08:02 929343     /lib/libc-2.12.so
002b5000-002b6000 rw-p 00192000 08:02 929343     /lib/libc-2.12.so
002b6000-002b9000 rw-p 00000000 00:00 0
00541000-0055f000 r-xp 00000000 08:02 928020     /lib/ld-2.12.so
0055f000-00560000 r--p 0001d000 08:02 928020     /lib/ld-2.12.so
00560000-00561000 rw-p 0001e000 08:02 928020     /lib/ld-2.12.so
006b2000-006b3000 r-xp 00000000 00:00 0          [vdso]
08048000-08053000 r-xp 00000000 08:02 920679     /bin/cat
08053000-08054000 rw-p 0000a000 08:02 920679     /bin/cat
08722000-08743000 rw-p 00000000 00:00 0          [heap]
b75d5000-b77d5000 r--p 00000000 08:02 791831     /usr/lib/locale/locale-archive
b77d5000-b77d6000 rw-p 00000000 00:00 0
b77e3000-b77e4000 rw-p 00000000 00:00 0
bfa77000-bfa8c000 rw-p 00000000 00:00 0          [stack]
```

Preparation for ROP

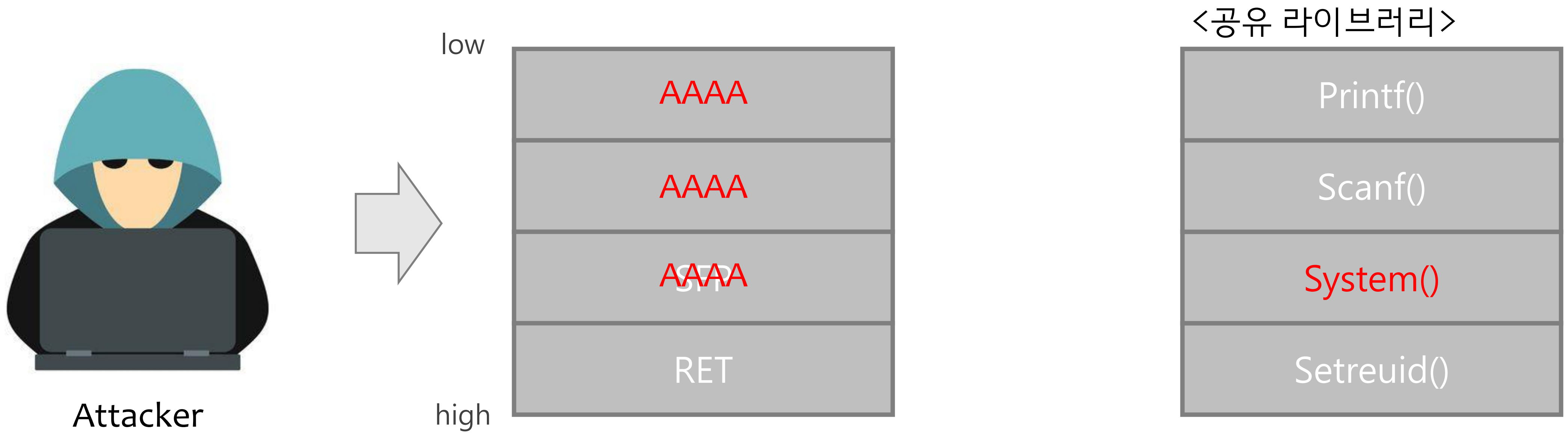
› RTL (Return To Libc)

PLT & GOT table



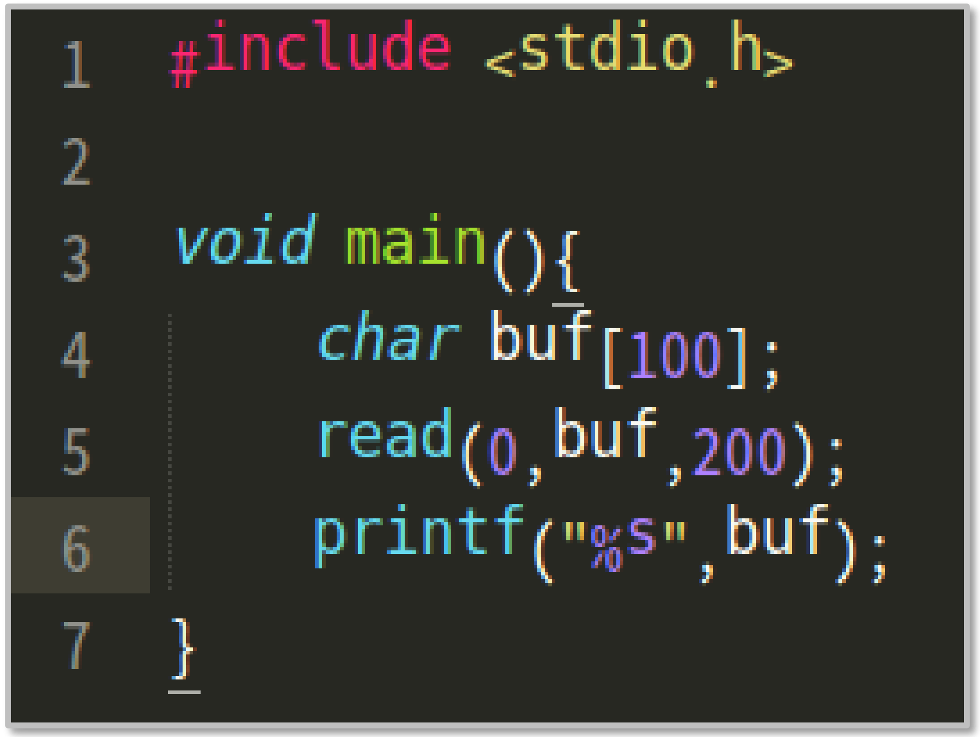
› RTL (Return To Libc)

NX bit를 우회하기 위해 사용되는 공격 기법
메모리에 미리 적재되어 있는 공유 라이브러리를 통해 바이너리에 원하는 함수가 없어도
원하는 함수를 사용할 수 있게 한다.

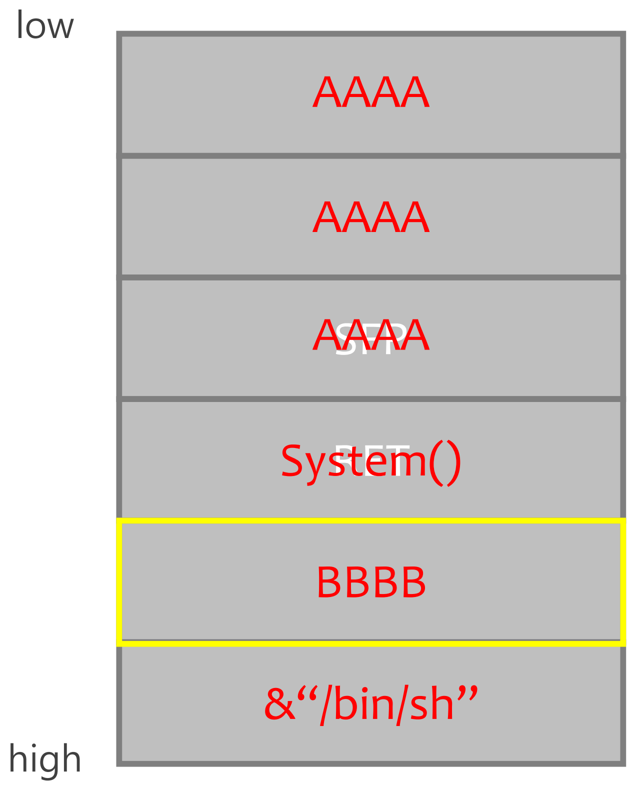


RTL (Return To Libc)

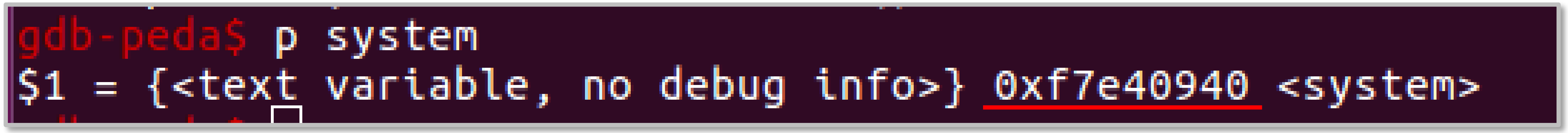
<예제 코드>



```c
#include <stdio.h>

void main(){
	char buf[100];
	read(0,buf,200);
	printf("%s",buf);
}
```



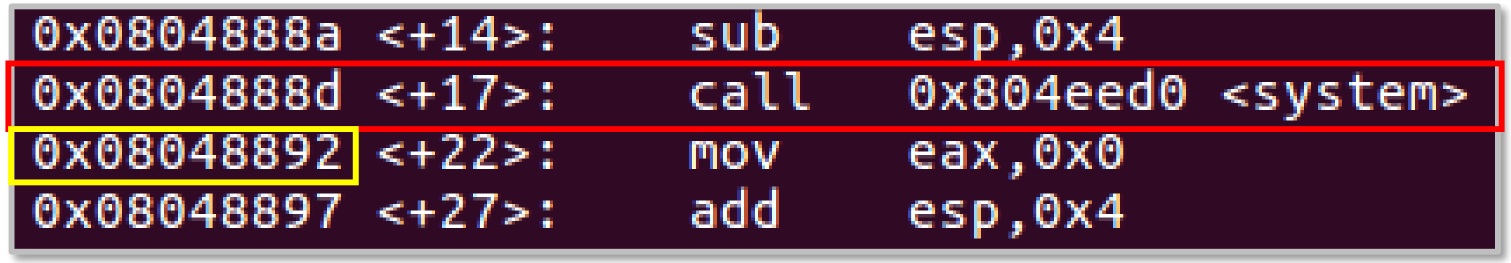
<After break pointer + run>



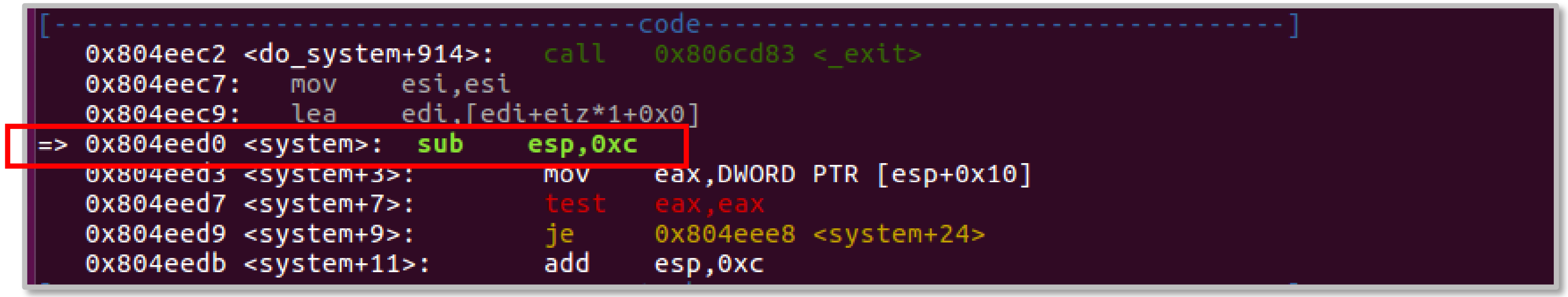
```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e40940 <system>
```

RTL (Return To Libc)

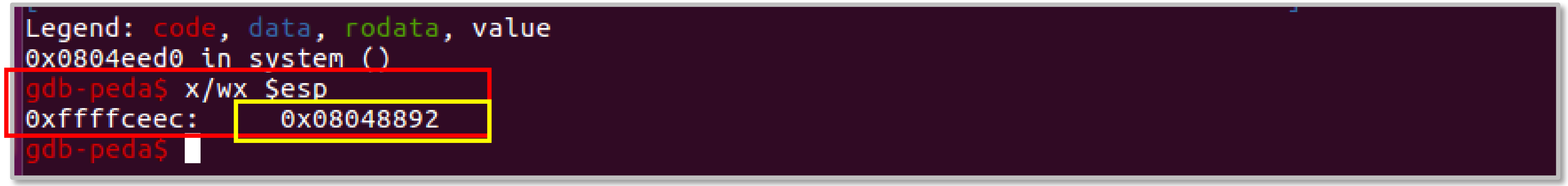
Call = push eip + jmp [func] -> 호출하기 전에 다음으로 실행할 명령어의 주소를 스택에 저장



```
0x0804888a <+14>:     sub    esp,0x4
0x0804888d <+17>:     call   0x804eed0 <system>
0x08048892 <+22>:     mov    eax,0x0
0x08048897 <+27>:     add    esp,0x4
```



```
[-----------------------------------code-----------------------------------]
   0x804eec2 <do_system+914>:   call   0x806cd83 <_exit>
   0x804eec7:   mov    esi,esi
   0x804eec9:   lea    edi,[edi+eiz*1+0x0]
=> 0x804eed0 <system>:  sub    esp,0xc
   0x804eed3 <system+3>:        mov    eax,DWORD PTR [esp+0x10]
   0x804eed7 <system+7>:        test   eax,eax
   0x804eed9 <system+9>:        je     0x804eee8 <system+24>
   0x804eedb <system+11>:       add    esp,0xc
```



```
Legend: code, data, rodata, value
0x0804eed0 in system ()
gdb-peda$ x/wx $esp
0xffffceec:     0x08048892
gdb-peda$ 
```

› RTL (Return To Libc)

Call = push eip + jmp [func] -> 호출하기 전에 다음으로 실행할 명령어의 주소를 스택에 저장

★ **ret** = pop eip -> jmp [func] => 다음으로 실행할 명령어의 주소를 저장하지 않고 함수로 jump

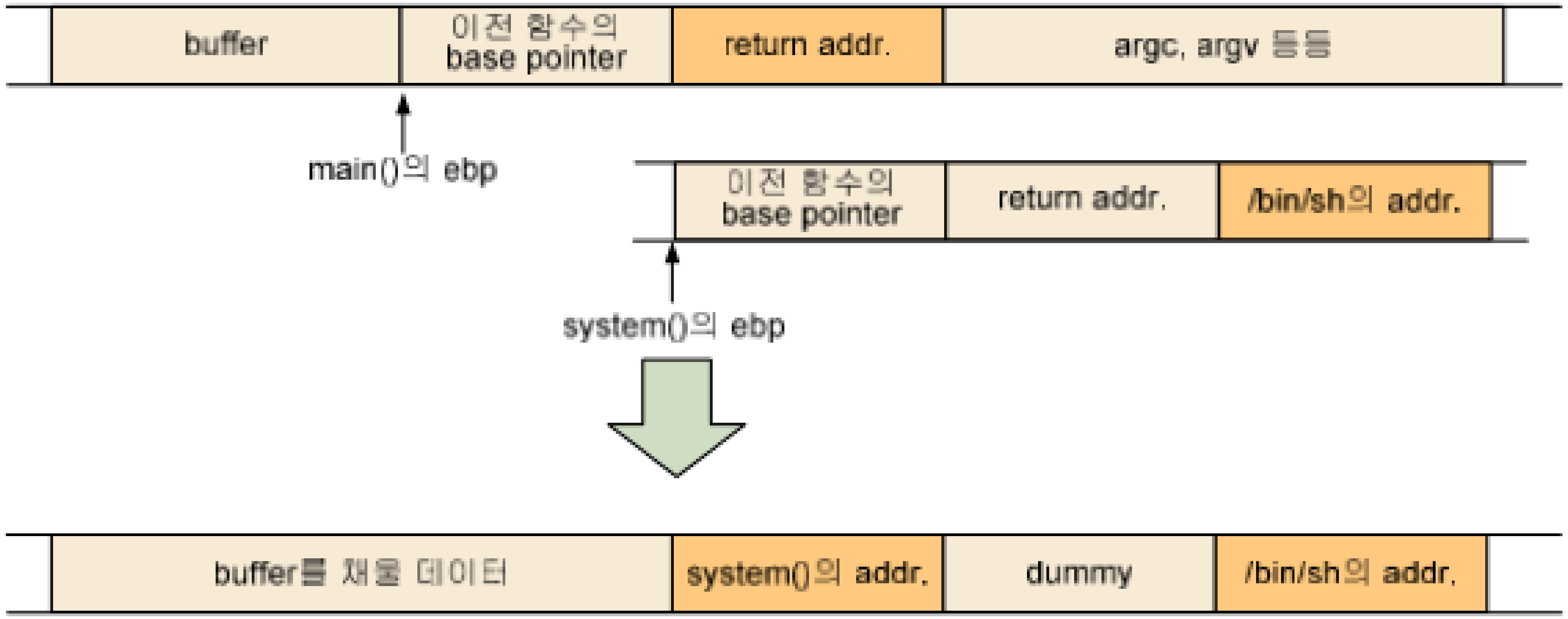


사진 출처 : 달고나 BOF문서

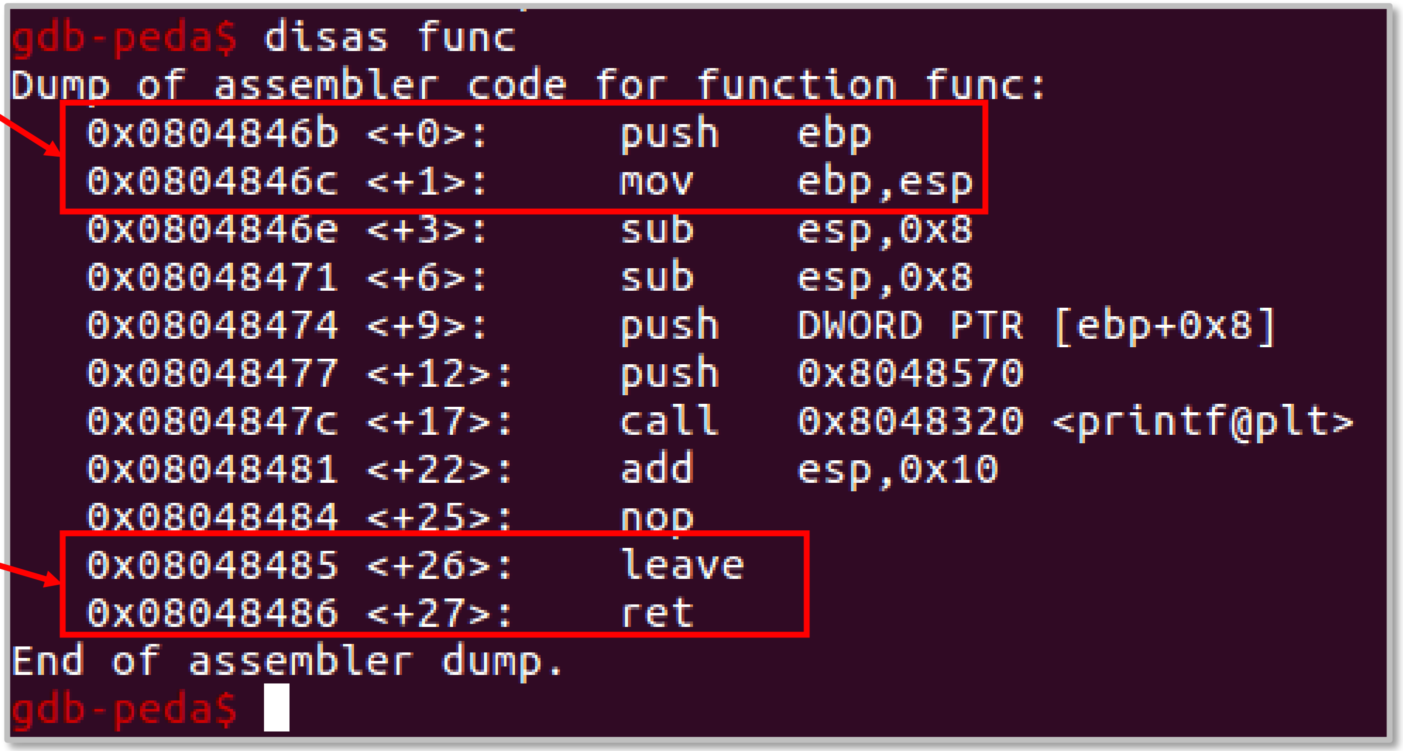
› RTL (Return To Libc)

Function **prologue**

```
push  ebp
mov   ebp, esp
```

Function **epilogue**

```
leave    mov  esp, ebp
         pop  ebp

ret      pop  eip
         jmp  eip
```

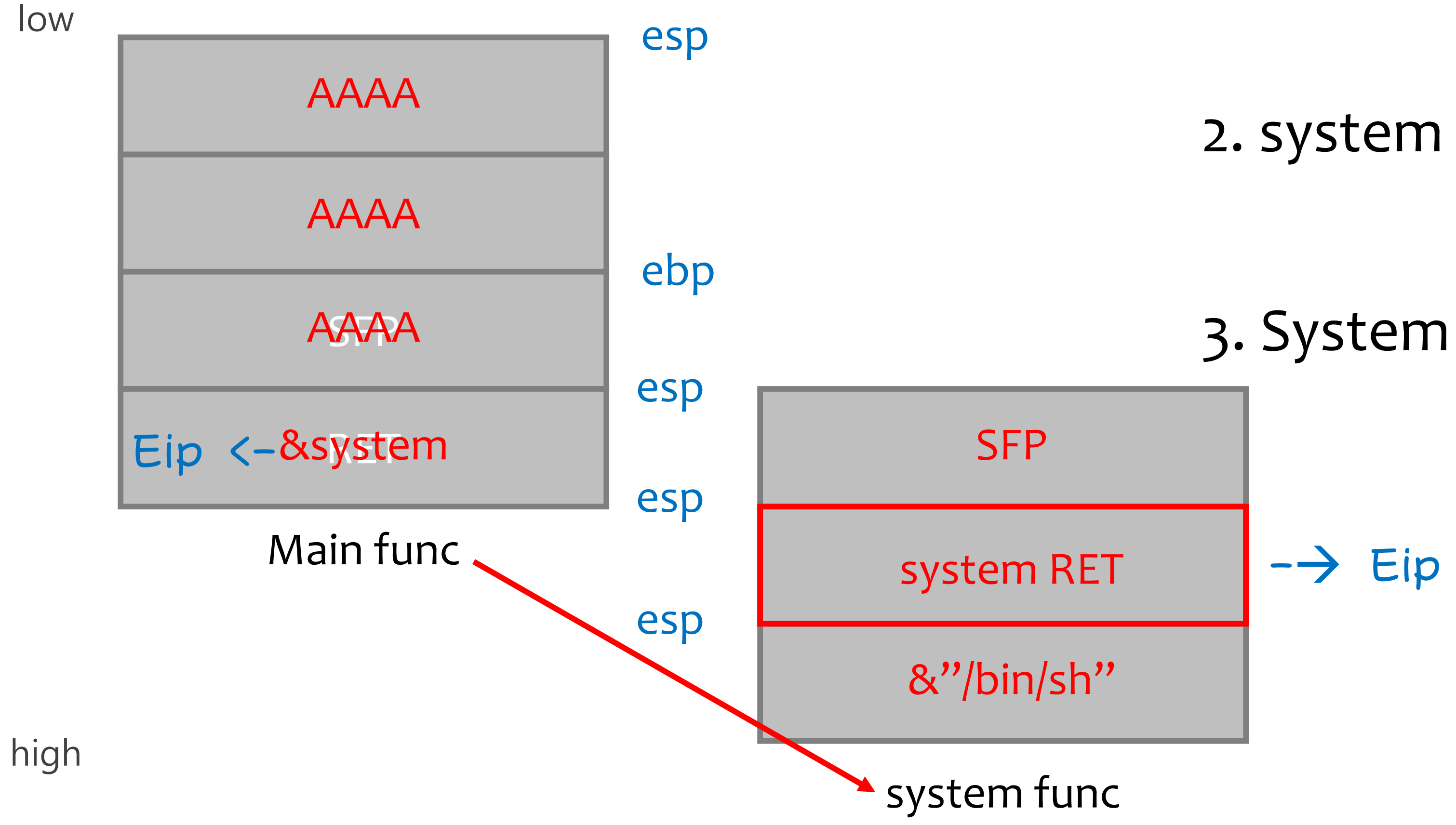


```
gdb-peda$ disas func
Dump of assembler code for function func:
   0x0804846b <+0>:     push   ebp
   0x0804846c <+1>:     mov    ebp,esp
   0x0804846e <+3>:     sub    esp,0x8
   0x08048471 <+6>:     sub    esp,0x8
   0x08048474 <+9>:     push   DWORD PTR [ebp+0x8]
   0x08048477 <+12>:    push   0x8048570
   0x0804847c <+17>:    call   0x8048320 <printf@plt>
   0x08048481 <+22>:    add    esp,0x10
   0x08048484 <+25>:    nop
   0x08048485 <+26>:    leave
   0x08048486 <+27>:    ret
End of assembler dump.
gdb-peda$
```

Preparation for ROP

RTL (Return To Libc)

- PUSH -> esp - 4
- POP -> esp + 4



1. Main epilogue
   - mov esp, ebp
   - pop ebp

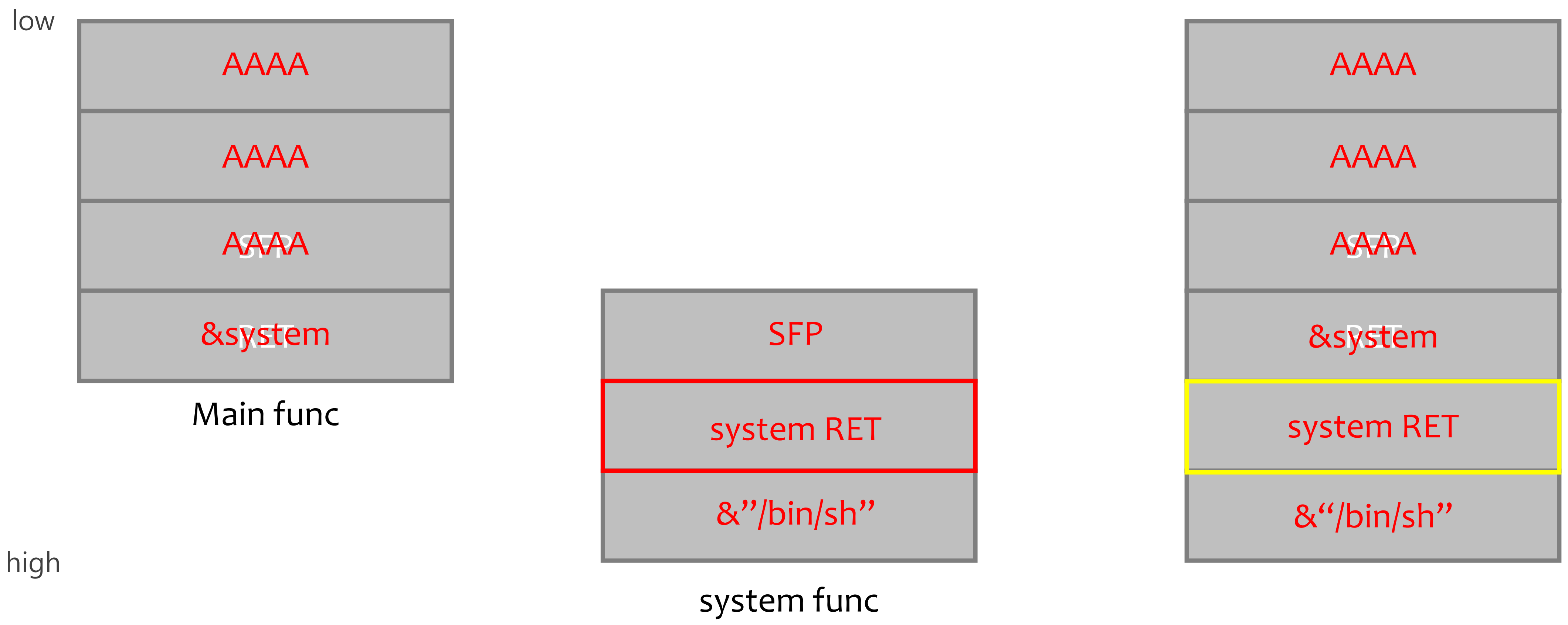
   leave
   - pop eip
   - jmp eip

   ret
2. system prologue
   - push ebp
   - mov ebp, esp
3. System epilogue
   - mov esp, ebp
   - pop ebp

   leave
   - pop eip
   - jmp eip

   ret

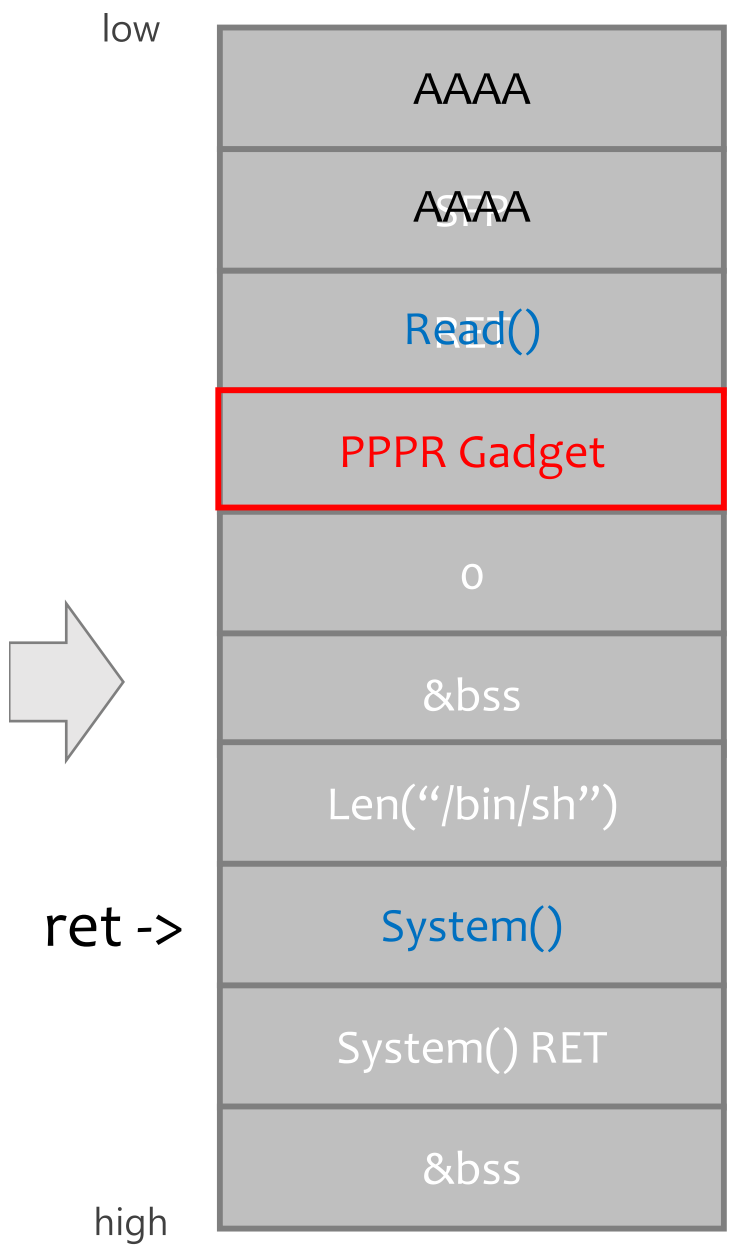
RTL (Return To Libc)



› RTL Chaining

RTL이후 연속으로 RTL을 사용하기 위한 방법
함수 프롤로그와 에필로그의 이해
ESP변조를 위한 Gadget 사용

Attacker



RTL Chaining

Gadget

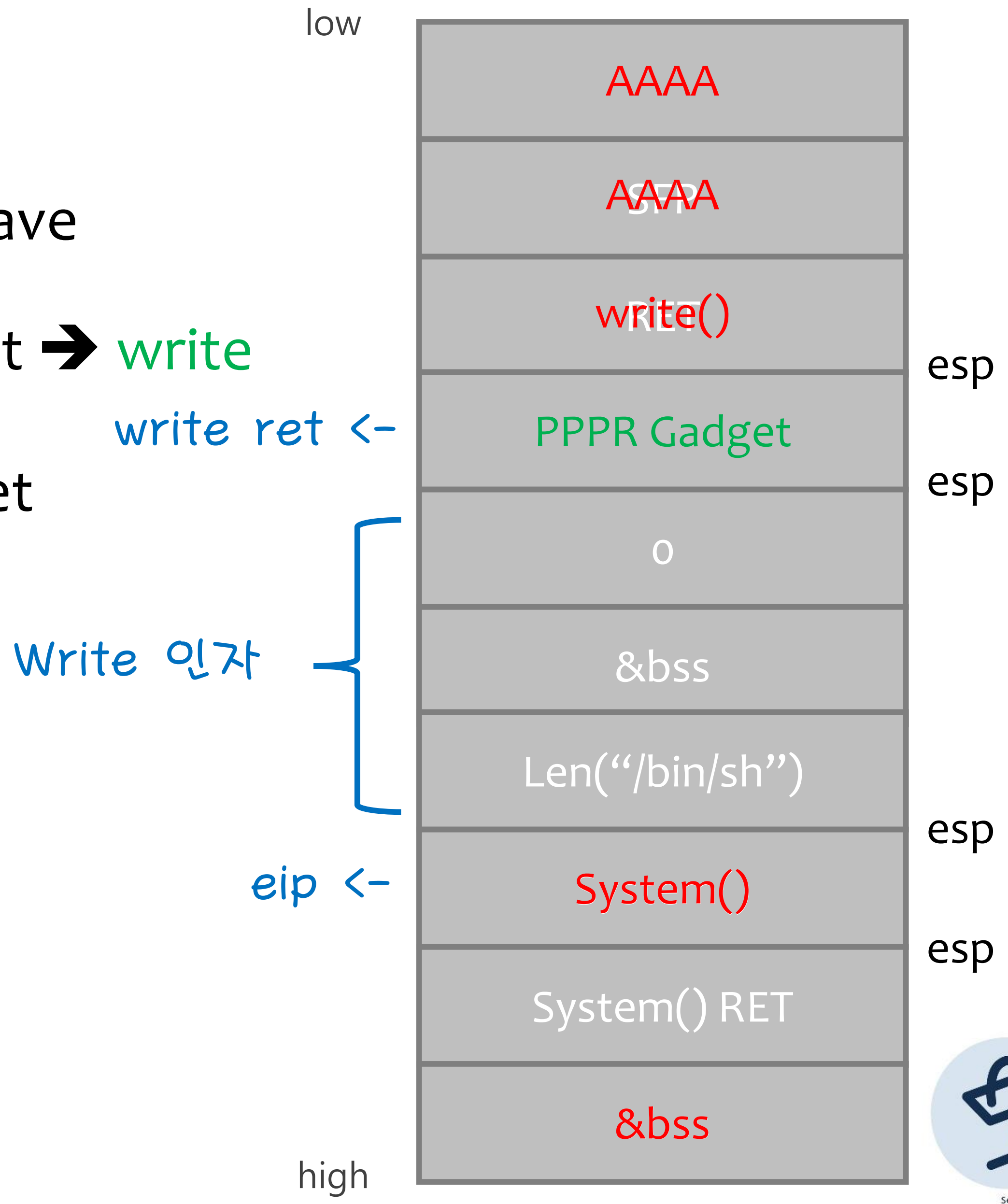
ESP를 조작할 수 있도록
해주는 어셈블리 코드
RTL을 연쇄적으로 사용

POP -> esp+4

RET -> POP eip, jmp eip

System(&bss) → "/bin/sh"

1. main의 leave
2. main의 ret ➔ write
3. write의 ret
4. POP EAX
5. POP EBX
6. POP ECX
7. RET



NEXT

› RO :P #2

(Return Oriented Programming)

QnA