# RTL Chaining

# *RTL(Return To Library)*

메모리에 미리 적재되어 있는 공유 라이브러리를 이용해,

바이너리에 원하는 함수가 없어도

**공유 라이브러리에서 원하는 함수를 사용할 수 있는 공격 기법**

# Why use RTL ??

```
int main( int argc, char *argv[] )
{
    char str[256];

    setreuid( 3092, 3092 );
    strcpy( str, argv[1] );
    printf( str );
}
```

strcpy 함수의 취약점을 이용해 쉘코드를 리턴주소에 삽입하여 공격이 가능하다.

**DEP 기법?**

버퍼 오버플로우로 인한 메모리 영역의 침범이 발생하더라도,

해당 영역에서 실행을 방지함으로써 공격을 방어하는 기법

**NX-bit 기법?**

가상 메모리의 최소 단위인 페이지 기존에 읽고 쓰는 권한 이외에

실행 권한에 대해서 체크하여 메모리 공격으로부터 시스템을 보호하는 기법

**** 쉘코드 실행 불가 ****

# *Why use RTL ??*

```
int main()
{
    char buf[20];
    gets(buf);
    printf("%s\n", buf);
}
```

**권한 변경을 할 수 있는 setreuid 함수가 필요하다.**

**쉘을 얻어오기 위해 system 함수가 필요하다.**

# *Function Epilogue*

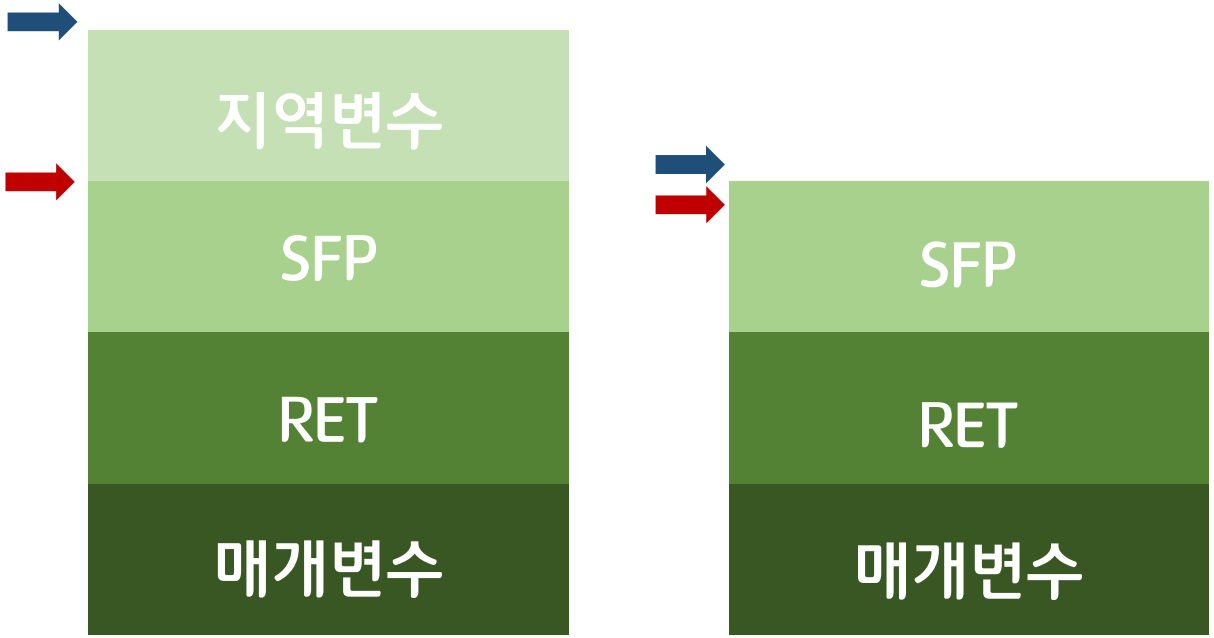
System() 호출

call 0x8048321 <SYSTEM>
add esp, 0x10

esp
ebp

① leave

①-1 mov esp, ebp



# *Function Epilogue*

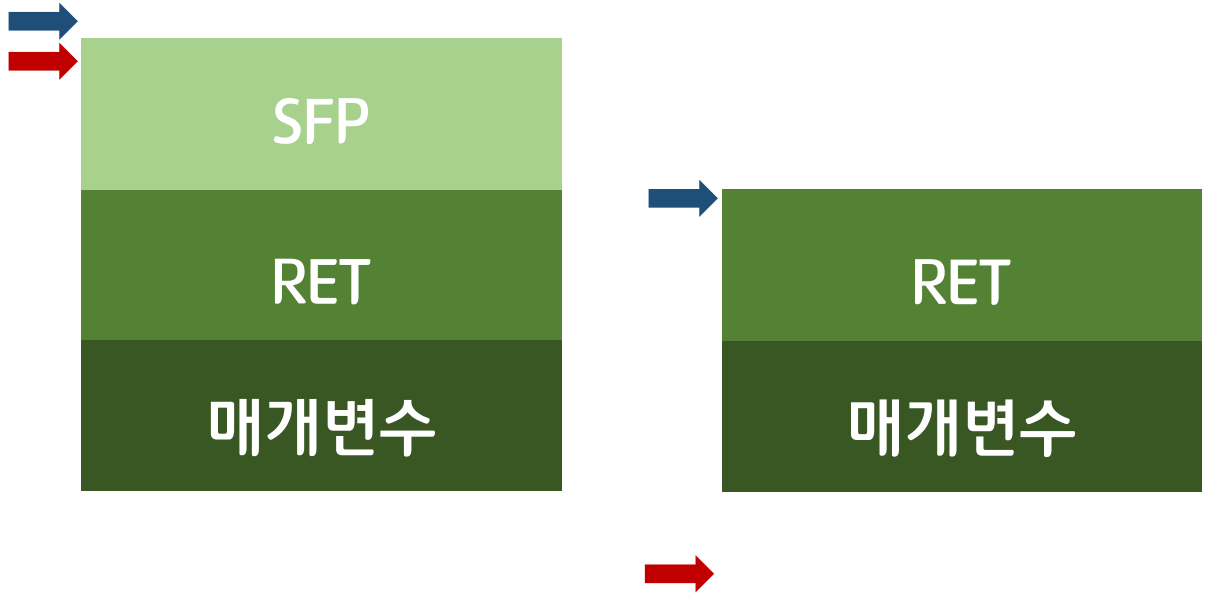
System() 호출

call 0x8048321 <SYSTEM>
add esp, 0x10

➡ esp
➡ ebp

① leave

①-2 pop ebp



# *Function Epilogue*

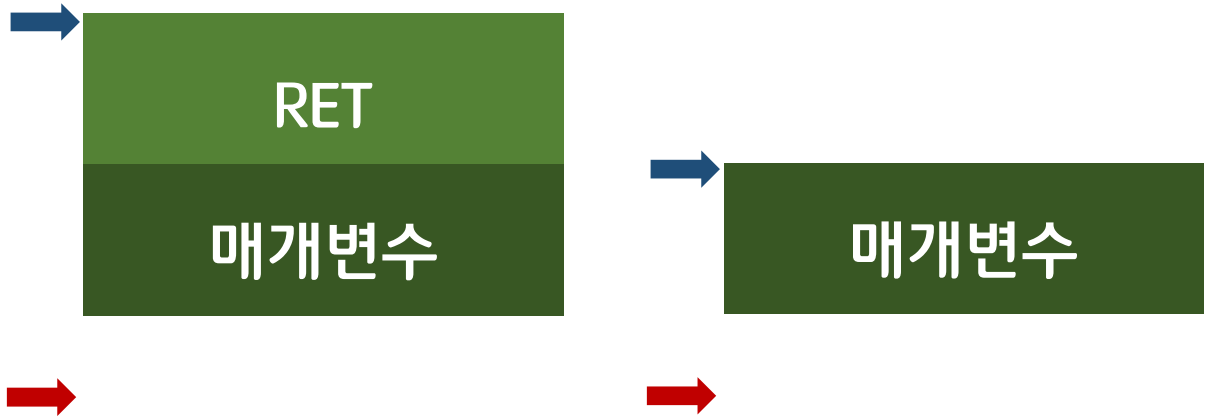
System() 호출

call 0x8048321 <SYSTEM>
add esp, 0x10

➡ esp
➡ ebp

② ret

pop eip



# RTL(Return To library)

System() 호출

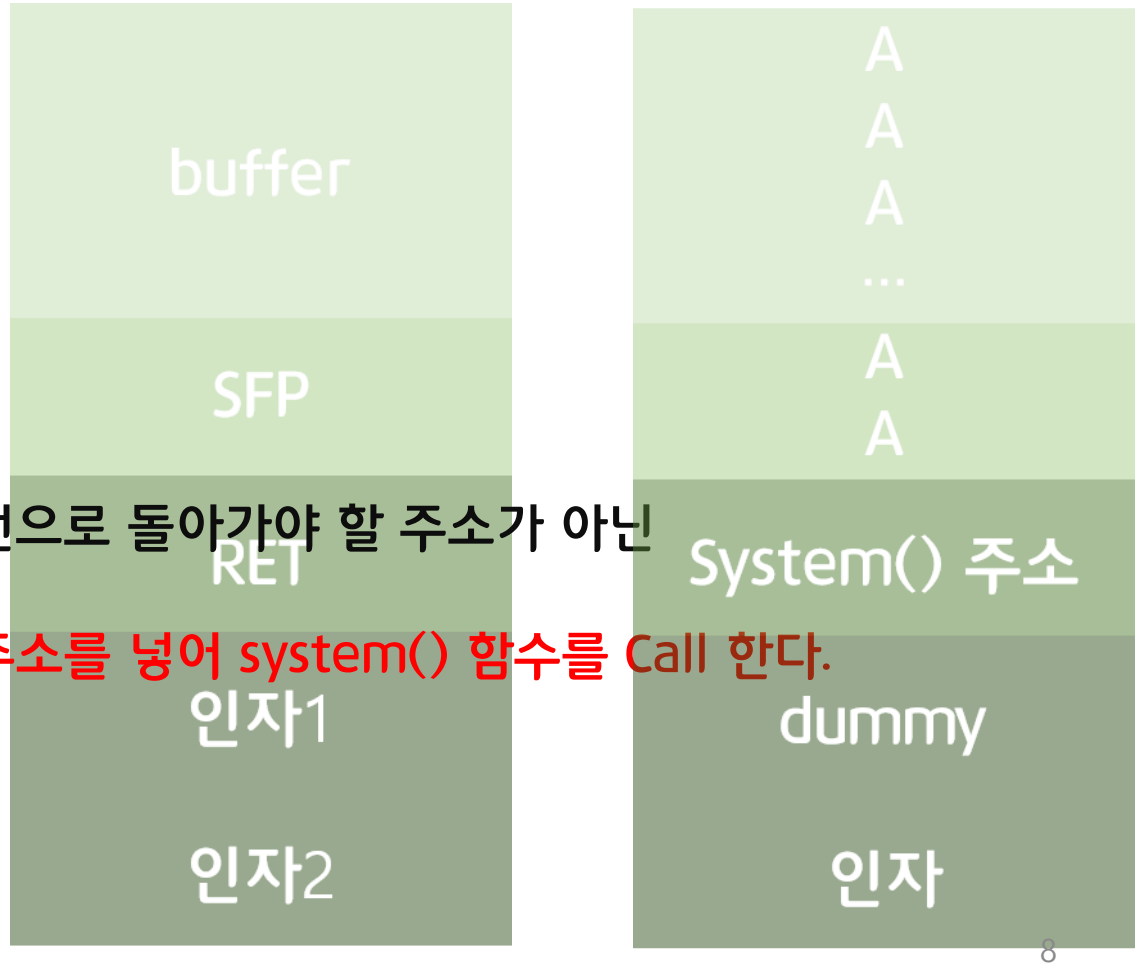
call 0x8048321 <SYSTEM>
add esp, 0x10

➡ esp
➡ ebp

gets 함수나 strcpy 함수 등 취약한 함수를 사용하여 오버플로우를 일으킨다.

1) sfp까지 값을 채운 후,

2) ret에 스택프레임의 호출 전으로 돌아가야 할 주소가 아닌 system 함수가 위치하는 주소를 넣어 system() 함수를 Call 한다.



# RTL Chaining

```c
int main()
{
    char buf[20];
    gets(buf);
    printf("%s\n", buf);
}
```

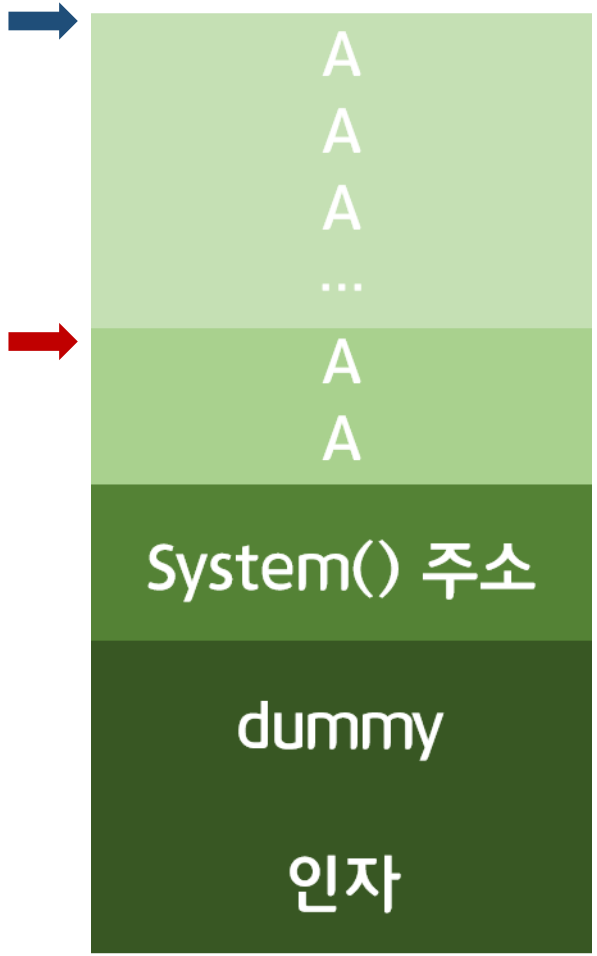
**여러 개의 함수가 필요할 때 사용하는 기법**

# RTL Chaining

System() 호출

call 0x8048321 <SYSTEM>
add esp, 0x10

esp
ebp




System 함수 시작

# RTL Chaining

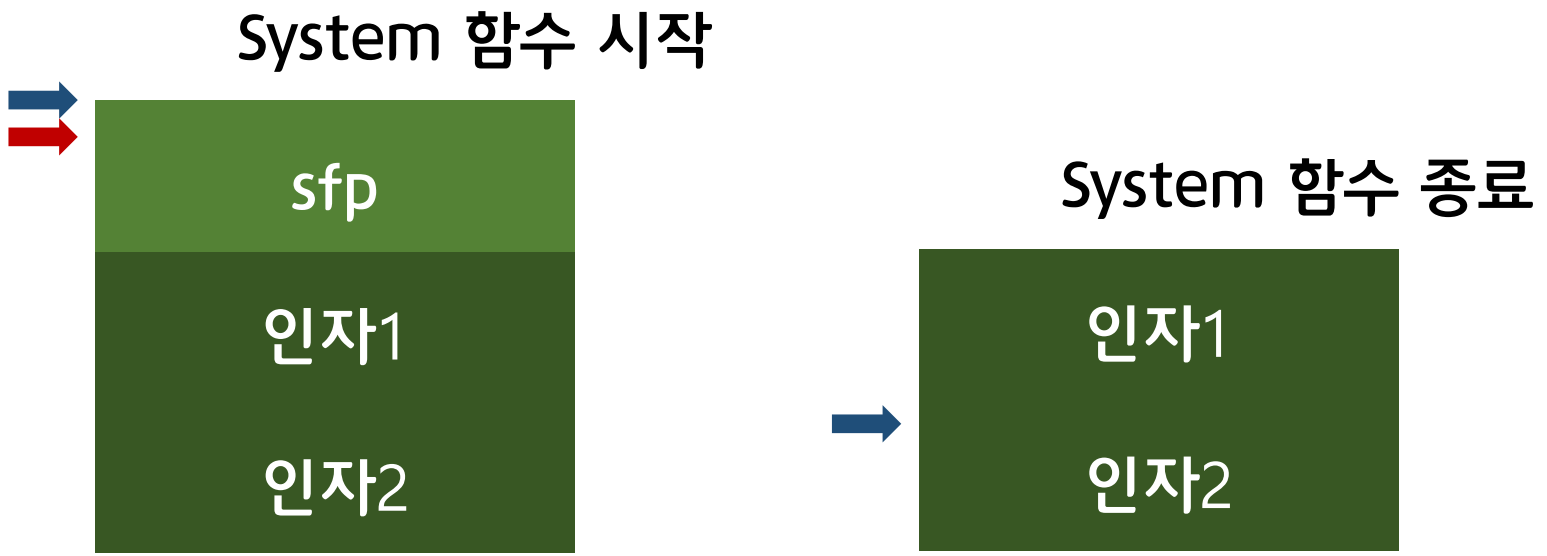
System() 호출

call 0x8048321 <SYSTEM>
add esp, 0x10

esp

ebp

인자1에 system()가 끝나고 실행될 주소를 넣는다.



# RTL Chaining

payload


함수가 호출될 때
현재 esp+8 위치에 있는 인자 값을 가져와서
해당 함수의 스택 프레임이 형성된다 !

# *gadget*

어셈블리어 명령어 pop과 ret로 구성된 것

[PR Gadget]

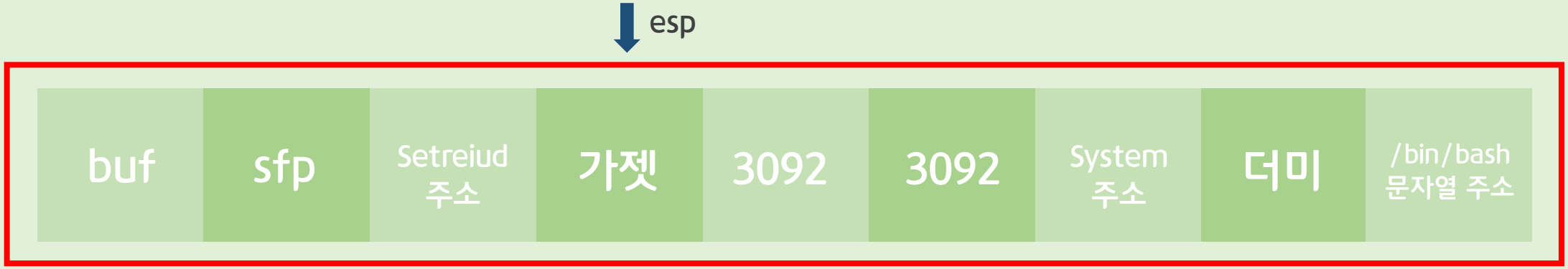
pop ret

[PPR Gadget]

pop pop ret

...

# *gadget*




<Setreuid가 끝나고 ret을 하기 직전>

1. 현재 인자가 2개 있기 때문에 pop pop ret 명령어가 위치하는 주소를 가젯에 넣는다.

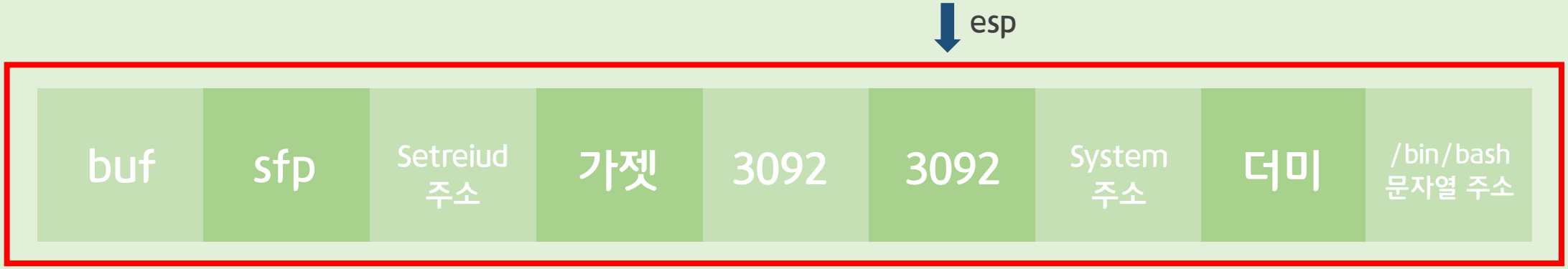
2. ret가 수행되면, 현재 가젯 즉, pop pop ret 명령어 주소를 eip에 넣고 esp를 하나 증가시킨다.

# *gadget*


<Setreuid가 끝나고 ret을 하기 직전>

3. 다음 실행할 명령어는 pop pop ret이므로, esp가 가리키고 있는 첫번째 인자를 pop한다.

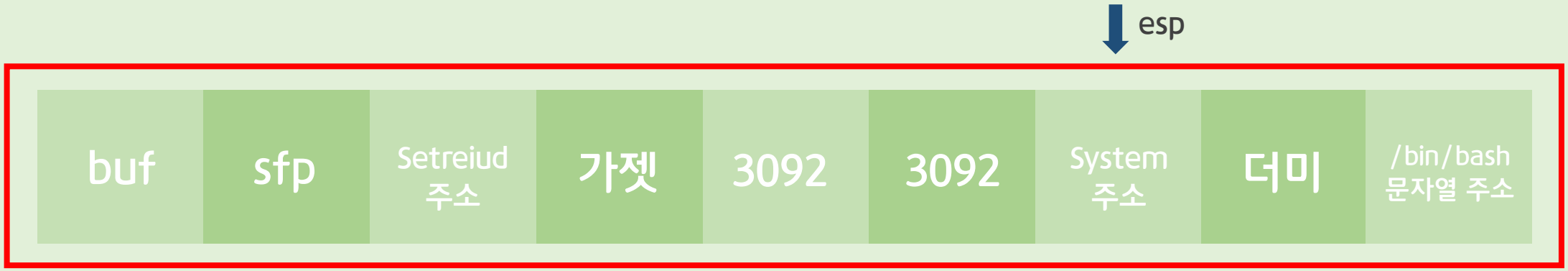
# *gadget*




<Setreuid가 끝나고 ret을 하기 직전>

4. 그 다음 인자를 pop한다.

# gadget




<Setreuid가 끝나고 ret을 하기 직전>

5. pop pop ret의 마지막인 ret에 의해 system 함수가 호출되게 된다.

6. system 함수가 호출될 때 esp + 8에 /bin/bash 문자열의 주소가 들어있으므로 이를 인자로 가지고 들어간다.

7. 더이상 다른 함수를 호출 할 필요가 없으므로 더미 값에는 아무 값이나 넣어도 상관없다.

# Q & A