

시스템 해킹 교육

Part 3. 바이너리 분석 심화

교육 구성

INDEX

-
- 0. 저번 회차 실습 풀이
 - 1. Hand-Ray
 - 2. Bomb Lab
 - 3. 분석 툴
-

실습 푸셨나요?

지역변수 조작하기

실습 풀이

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int first = 0x12345678;
6     char buf[64];
7
8     gets(buf);
9
10    if(first == 0x87654321)
11        printf("clear!\n");
12    else
13        printf("try again~\n");
14
15    return 0;
16 }
```

지역변수 조작하기 실습 풀이

```
gdb-peda$ pd main
```

```
Dump of assembler code for function main:
```

```
0x08048456 <+0>:  push    ebp
0x08048457 <+1>:  mov     ebp,esp
0x08048459 <+3>:  push    ebx
0x0804845a <+4>:  sub     esp,0x44
0x0804845d <+7>:  call    0x8048390 <__x86.get_pc_thunk.bx>
0x08048462 <+12>: add     ebx,0x1b9e
0x08048468 <+18>: mov     DWORD PTR [ebp-0x8],0x12345678
0x0804846f <+25>: lea     eax,[ebp-0x48]
0x08048472 <+28>: push    eax
0x08048473 <+29>: call    0x8048300 <gets@plt>
0x08048478 <+34>: add     esp,0x4
0x0804847b <+37>: cmp     DWORD PTR [ebp-0x8],0x87654321
0x08048482 <+44>: jne     0x8048495 <main+63>
0x08048484 <+46>: lea     eax,[ebx-0x1ad0]
0x0804848a <+52>: push    eax
0x0804848b <+53>: call    0x8048310 <puts@plt>
0x08048490 <+58>: add     esp,0x4
0x08048493 <+61>: jmp     0x80484a4 <main+78>
0x08048495 <+63>: lea     eax,[ebx-0x1ac9]
0x0804849b <+69>: push    eax
0x0804849c <+70>: call    0x8048310 <puts@plt>
0x080484a1 <+75>: add     esp,0x4
0x080484a4 <+78>: mov     eax,0x0
0x080484a9 <+83>: mov     ebx,DWORD PTR [ebp-0x4]
0x080484ac <+86>: leave
0x080484ad <+87>: ret
```

```
End of assembler dump.
```

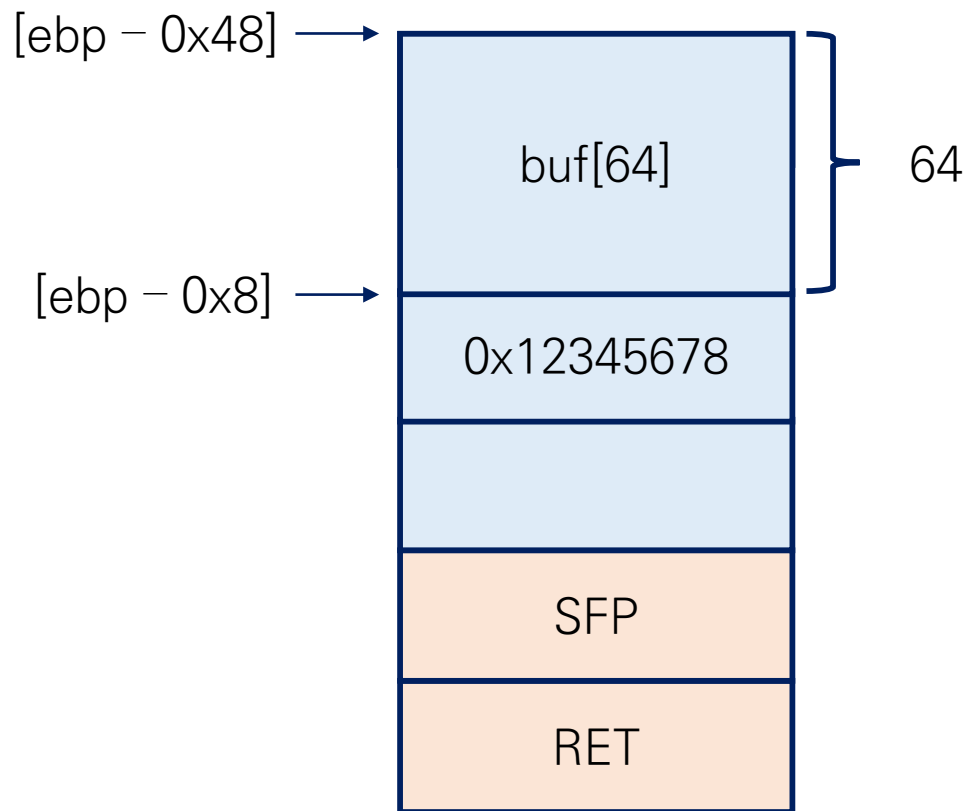
```
gdb-peda$
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int first = 0x12345678;
6     char buf[64];
7
8     gets(buf);
9
10    if(first == 0x87654321)
11        printf("clear!\n");
12    else
13        printf("try again~\n");
14
15    return 0;
16 }
```

지역변수 조작하기

실습 풀이

```
gdb-peda$ pd main
Dump of assembler code for function main:
0x08048456 <+0>:    push    ebp
0x08048457 <+1>:    mov     ebp,esp
0x08048459 <+3>:    push    ebx
0x0804845a <+4>:    sub     esp,0x44
0x0804845d <+7>:    call    0x8048390 <__x86.get_pc_thunk.bx>
0x08048462 <+12>:   add     ebx,0x1b9e
0x08048468 <+18>:   mov     DWORD PTR [ebp-0x8],0x12345678
0x0804846f <+25>:   lea     eax,[ebp-0x48]
0x08048472 <+28>:   push    eax
0x08048473 <+29>:   call    0x8048300 <gets@plt>
0x08048478 <+34>:   add     esp,0x4
0x0804847b <+37>:   cmp     DWORD PTR [ebp-0x8],0x87654321
0x08048482 <+44>:   jne     0x8048495 <main+63>
0x08048484 <+46>:   lea     eax,[ebx-0x1ad0]
0x0804848a <+52>:   push    eax
0x0804848b <+53>:   call    0x8048310 <puts@plt>
0x08048490 <+58>:   add     esp,0x4
0x08048493 <+61>:   jmp     0x80484a4 <main+78>
0x08048495 <+63>:   lea     eax,[ebx-0x1ac9]
0x0804849b <+69>:   push    eax
0x0804849c <+70>:   call    0x8048310 <puts@plt>
0x080484a1 <+75>:   add     esp,0x4
0x080484a4 <+78>:   mov     eax,0x0
0x080484a9 <+83>:   mov     ebx,DWORD PTR [ebp-0x4]
0x080484ac <+86>:   leave
0x080484ad <+87>:   ret
End of assembler dump.
gdb-peda$
```



지역변수 조작하기

실습 풀이

```
minibeef@argos-edu:~/19_system_edu/lecture1$ (python -c 'print "A"*64 + "\x21\x43\x65\x87"') | ./prac1  
clear!
```

Hand-Ray

Hand-Ray란?

Dump of assembler code for function main:

```
0x0804842d <+0>:  push    ebp
0x0804842e <+1>:  mov     ebp,esp
0x08048430 <+3>:  sub     esp,0x10
0x08048433 <+6>:  mov     DWORD PTR [ebp-0x8],0x3
0x0804843a <+13>: mov     DWORD PTR [ebp-0x4],0x4
0x08048441 <+20>: mov     eax,DWORD PTR [ebp-0x4]
0x08048444 <+23>: mov     edx,DWORD PTR [ebp-0x8]
0x08048447 <+26>: add     eax,edx
0x08048449 <+28>: mov     DWORD PTR [esp+0x4],eax
0x0804844d <+32>: mov     DWORD PTR [esp],0x80484e0
0x08048454 <+39>: call    0x80482e0 <printf@plt>
0x08048459 <+44>: mov     eax,0x0
0x0804845e <+49>: leave
0x0804845f <+50>: ret
```

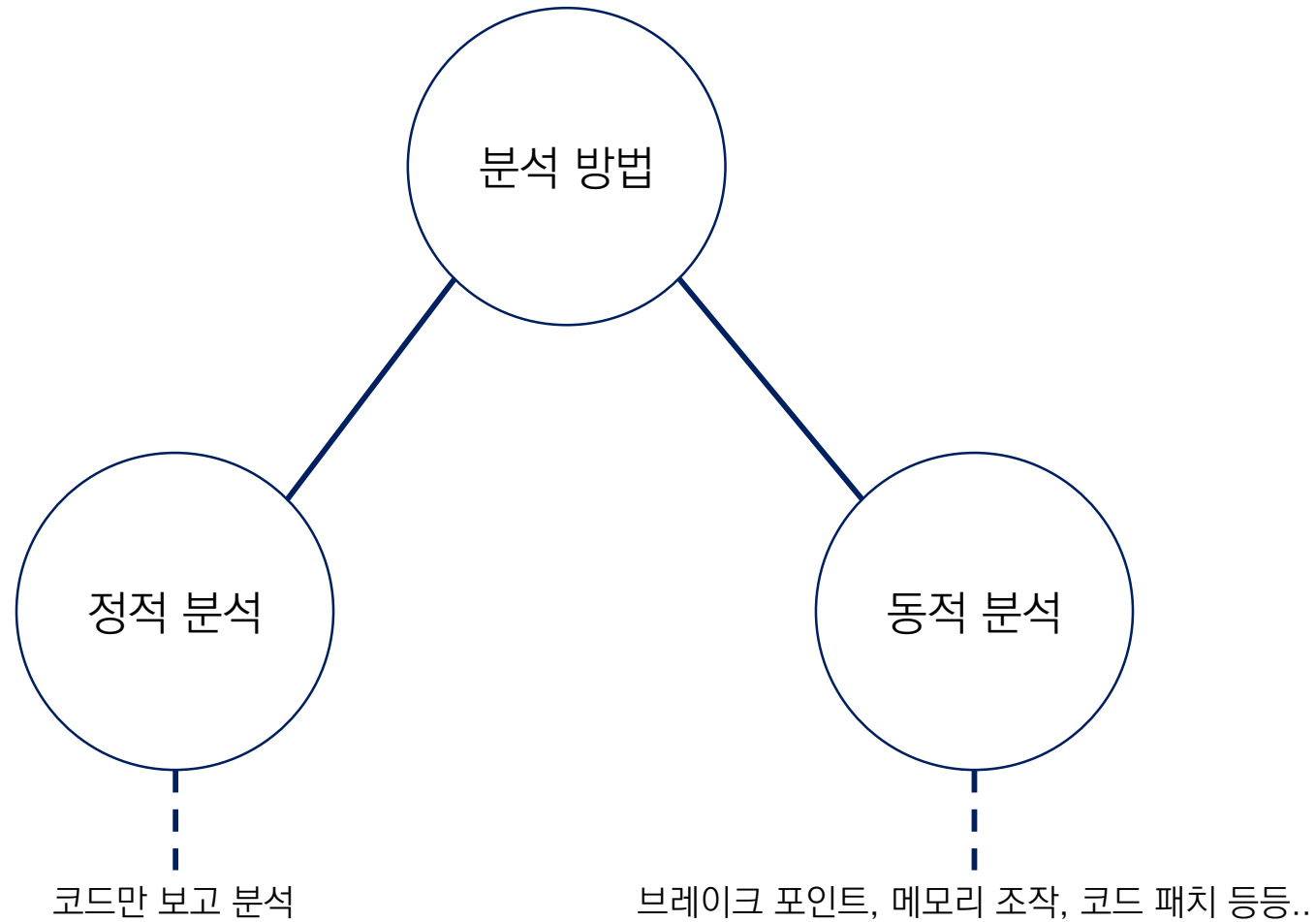


```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a = 3;
6     int b = 4;
7
8     printf("%d", a + b);
9
10    return 0;
11 }
```

어셈블리 코드를 보고 C 코드(혹은 의사 코드)로 변환하는 분석 테크닉,
1회차 교육에서 언급했듯이 어셈블리를 보고 분석하는 능력은 중요한 역량 임

Hand-Ray

Hand-Ray란?

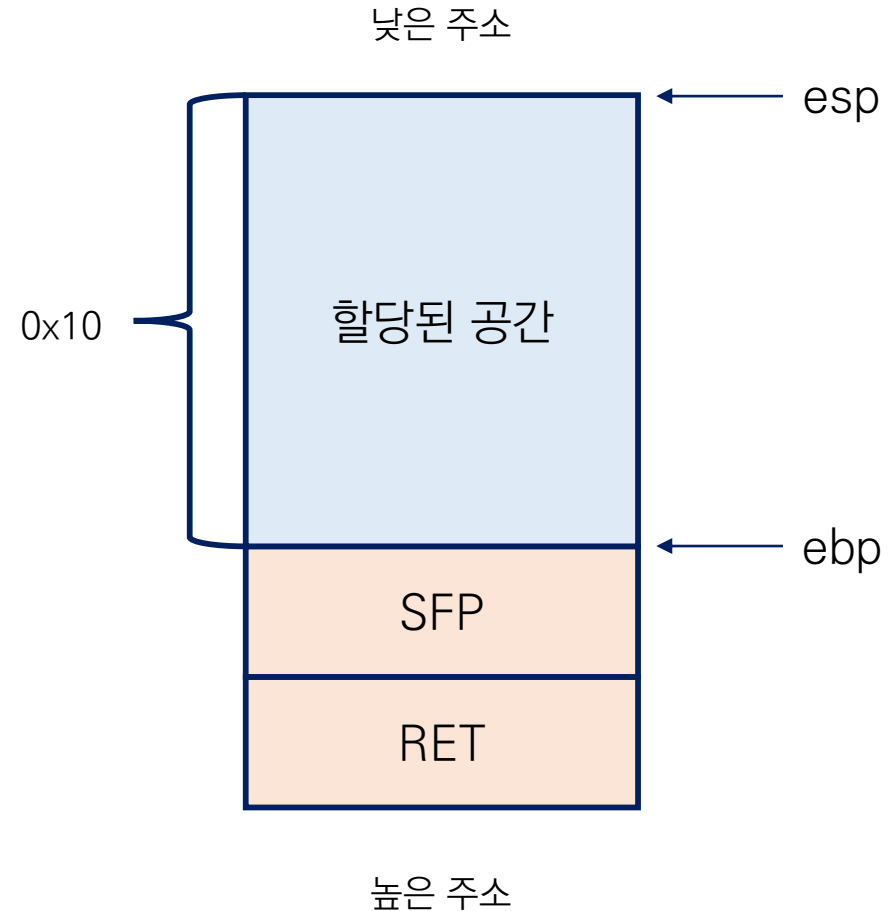


Hand-Ray

Hand-Ray란?

Dump of assembler code for function main:

```
0x0804842d <+0>:  push    ebp
0x0804842e <+1>:  mov     ebp,esp
0x08048430 <+3>:  sub     esp,0x10
0x08048433 <+6>:  mov     DWORD PTR [ebp-0x8],0x3
0x0804843a <+13>:  mov     DWORD PTR [ebp-0x4],0x4
0x08048441 <+20>:  mov     eax,DWORD PTR [ebp-0x4]
0x08048444 <+23>:  mov     edx,DWORD PTR [ebp-0x8]
0x08048447 <+26>:  add     eax,edx
0x08048449 <+28>:  mov     DWORD PTR [esp+0x4],eax
0x0804844d <+32>:  mov     DWORD PTR [esp],0x80484e0
0x08048454 <+39>:  call    0x80482e0 <printf@plt>
0x08048459 <+44>:  mov     eax,0x0
0x0804845e <+49>:  leave
0x0804845f <+50>:  ret
```



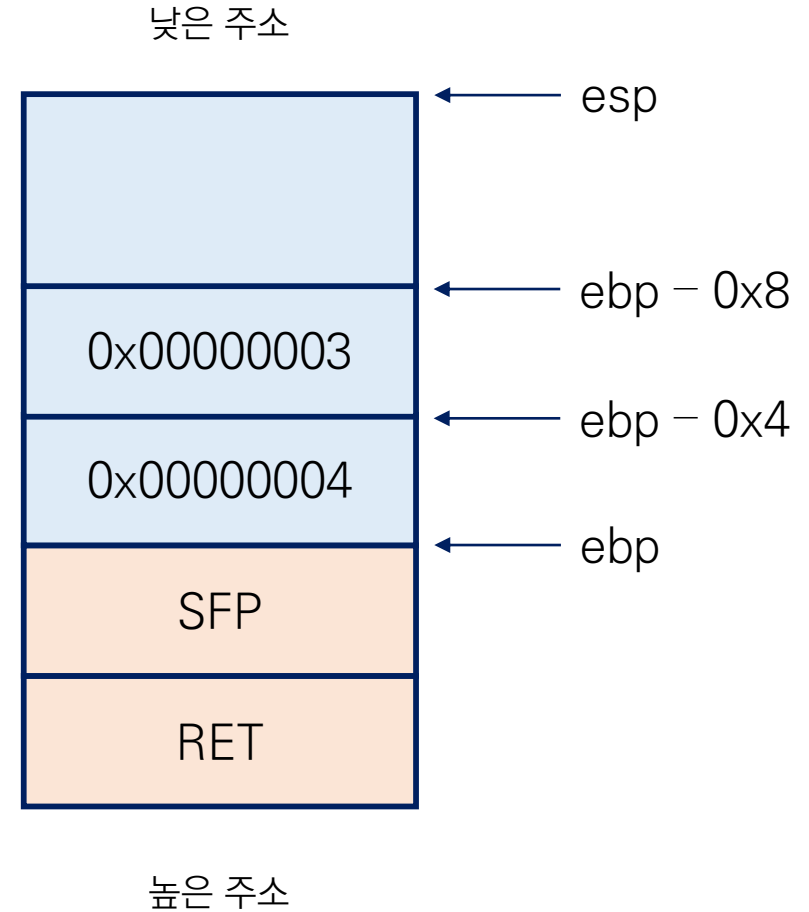
변수가 들어갈 공간을 확보 한다(프롤로그 이후 sub 명령 => 변수 선언일 경우가 유력)

Hand-Ray

Hand-Ray란?

Dump of assembler code for function main:

```
0x0804842d <+0>:  push    ebp
0x0804842e <+1>:  mov     ebp,esp
0x08048430 <+3>:  sub     esp,0x10
0x08048433 <+6>:  mov     DWORD PTR [ebp-0x8],0x3
0x0804843a <+13>:  mov     DWORD PTR [ebp-0x4],0x4
0x08048441 <+20>:  mov     eax,DWORD PTR [ebp-0x4]
0x08048444 <+23>:  mov     edx,DWORD PTR [ebp-0x8]
0x08048447 <+26>:  add     eax,edx
0x08048449 <+28>:  mov     DWORD PTR [esp+0x4],eax
0x0804844d <+32>:  mov     DWORD PTR [esp],0x80484e0
0x08048454 <+39>:  call    0x80482e0 <printf@plt>
0x08048459 <+44>:  mov     eax,0x0
0x0804845e <+49>:  leave
0x0804845f <+50>:  ret
```



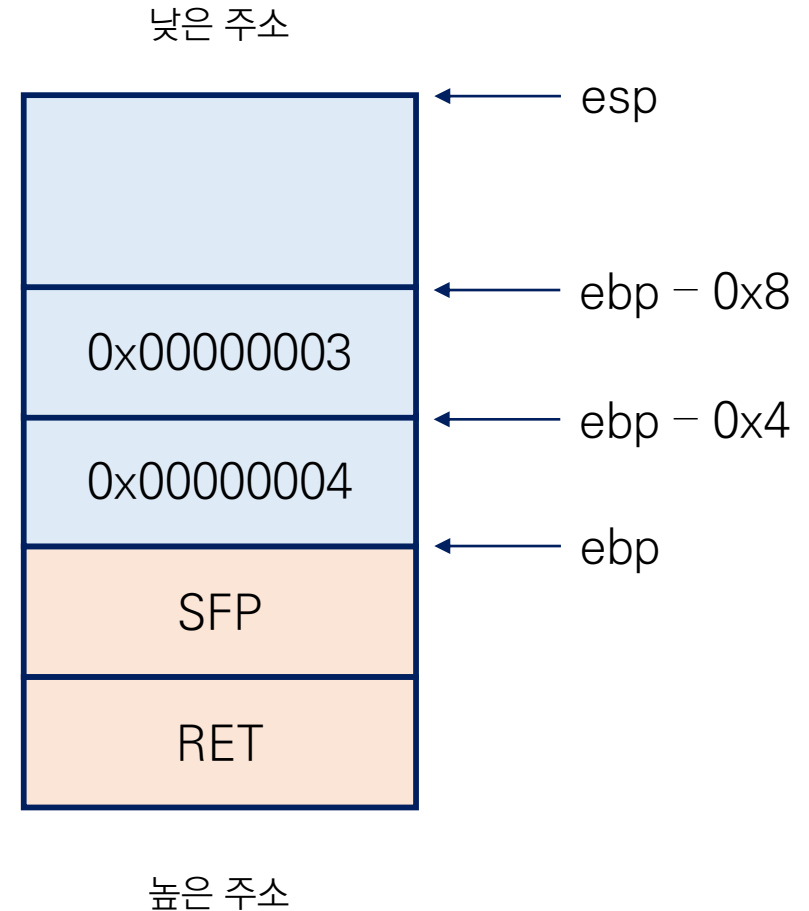
두 변수에 int값을 각각 할당(스택에 저장)

Hand-Ray

Hand-Ray란?

Dump of assembler code for function main:

```
0x0804842d <+0>:  push    ebp
0x0804842e <+1>:  mov     ebp,esp
0x08048430 <+3>:  sub     esp,0x10
0x08048433 <+6>:  mov     DWORD PTR [ebp-0x8],0x3
0x0804843a <+13>:  mov     DWORD PTR [ebp-0x4],0x4
0x08048441 <+20>:  mov     eax,DWORD PTR [ebp-0x4]
0x08048444 <+23>:  mov     edx,DWORD PTR [ebp-0x8]
0x08048447 <+26>:  add     eax,edx
0x08048449 <+28>:  mov     DWORD PTR [esp+0x4],eax
0x0804844d <+32>:  mov     DWORD PTR [esp],0x80484e0
0x08048454 <+39>:  call    0x80482e0 <printf@plt>
0x08048459 <+44>:  mov     eax,0x0
0x0804845e <+49>:  leave
0x0804845f <+50>:  ret
```



연산을 하기 위해 두 값을 레지스터 eax, edx에 복사하고, add 연산을 통해 eax가 7이 된다.

Hand-Ray

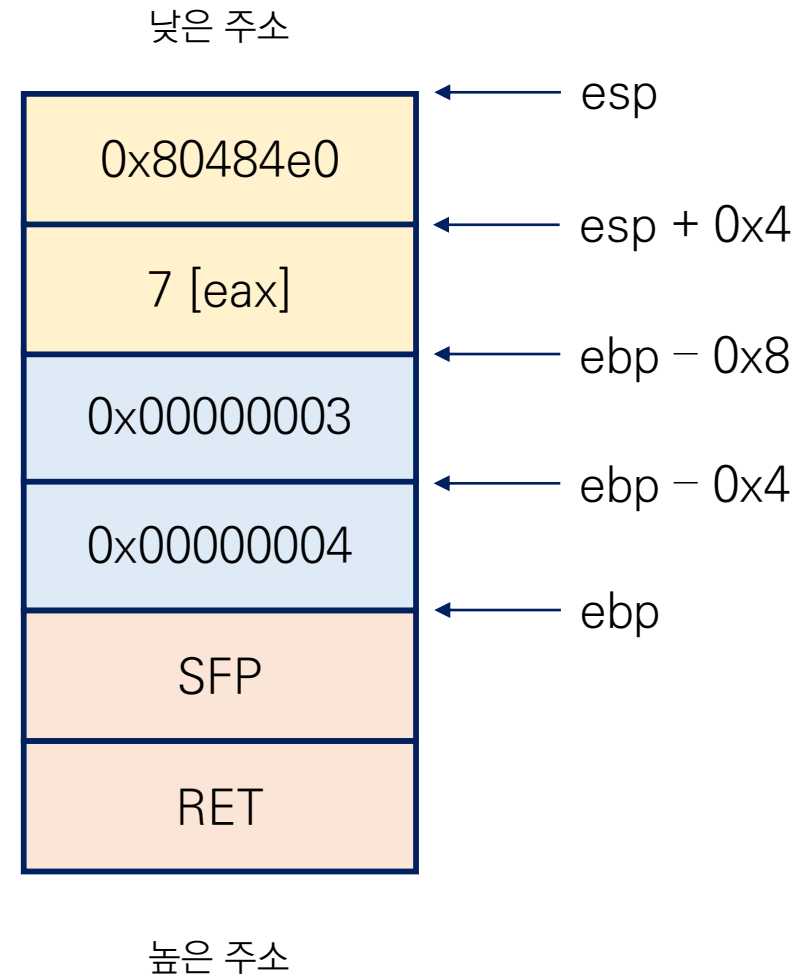
Hand-Ray란?

Dump of assembler code for function main:

```

0x0804842d <+0>:    push    ebp
0x0804842e <+1>:    mov     ebp,esp
0x08048430 <+3>:    sub     esp,0x10
0x08048433 <+6>:    mov     DWORD PTR [ebp-0x8],0x3
0x0804843a <+13>:   mov     DWORD PTR [ebp-0x4],0x4
0x08048441 <+20>:   mov     eax,DWORD PTR [ebp-0x4]
0x08048444 <+23>:   mov     edx,DWORD PTR [ebp-0x8]
0x08048447 <+26>:   add     eax,edx
0x08048449 <+28>:   mov     DWORD PTR [esp+0x4],eax
0x0804844d <+32>:   mov     DWORD PTR [esp],0x80484e0
0x08048454 <+39>:   call    0x80482e0 <printf@plt>
0x08048459 <+44>:   mov     eax,0x0
0x0804845e <+49>:   leave
0x0804845f <+50>:   ret

```



printf 함수를 호출하기 전 파라미터 저장

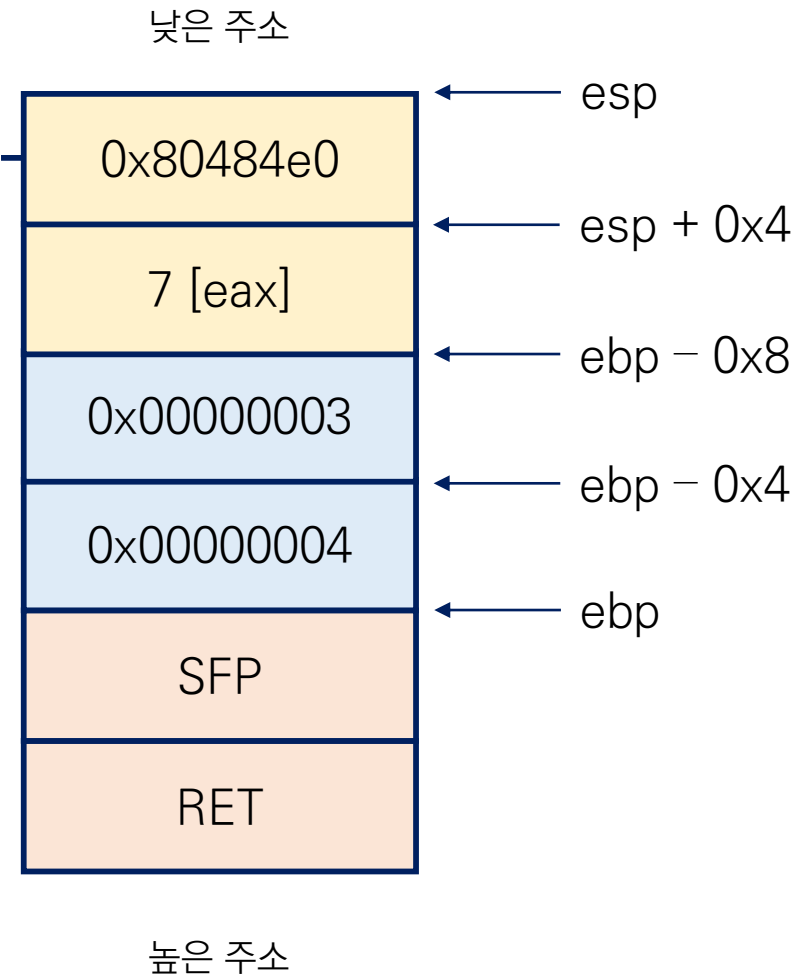
Hand-Ray

Hand-Ray란?

```
gdb-peda$ x/s 0x80484e0
0x80484e0: "%d"
```

Dump of assembler code for function main:

```
0x0804842d <+0>: push    ebp
0x0804842e <+1>: mov     ebp,esp
0x08048430 <+3>: sub     esp,0x10
0x08048433 <+6>: mov     DWORD PTR [ebp-0x8],0x3
0x0804843a <+13>: mov     DWORD PTR [ebp-0x4],0x4
0x08048441 <+20>: mov     eax,DWORD PTR [ebp-0x4]
0x08048444 <+23>: mov     edx,DWORD PTR [ebp-0x8]
0x08048447 <+26>: add     eax,edx
0x08048449 <+28>: mov     DWORD PTR [esp+0x4],eax
0x0804844d <+32>: mov     DWORD PTR [esp],0x80484e0
0x08048454 <+39>: call    0x80482e0 <printf@plt>
0x08048459 <+44>: mov     eax,0x0
0x0804845e <+49>: leave
0x0804845f <+50>: ret
```



printf 함수를 호출하기 전 파라미터 저장

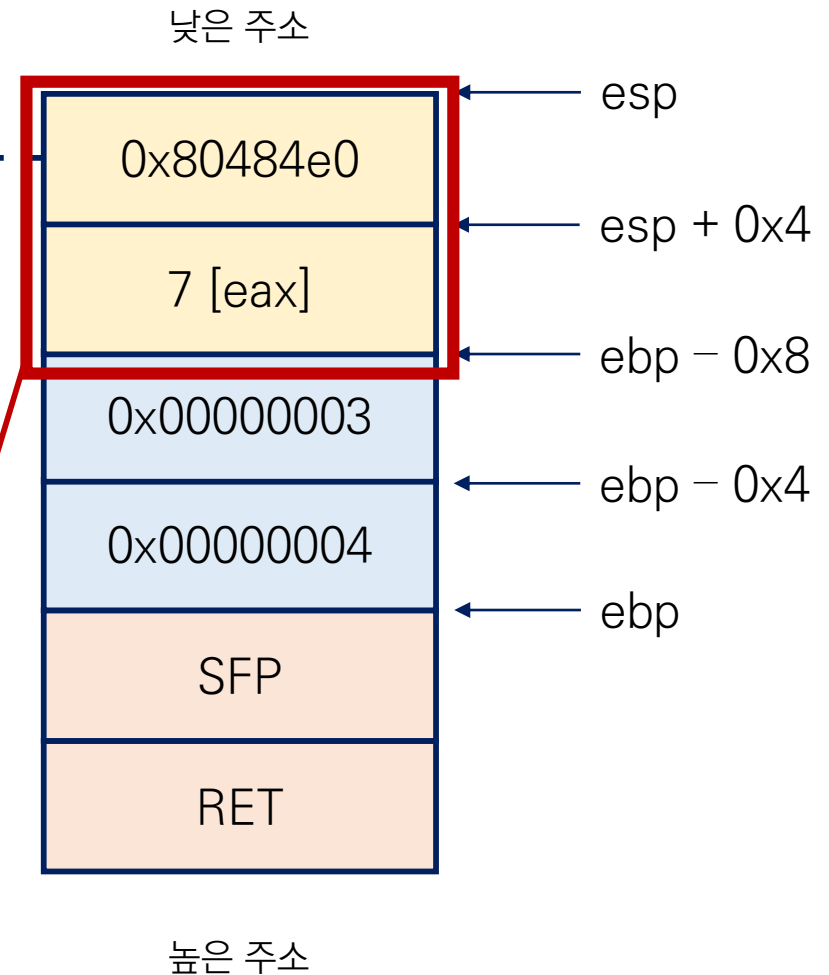
Hand-Ray

Hand-Ray란?

```
gdb-peda$ x/s 0x80484e0
0x80484e0: "%d"
```

Dump of assembler code for function main:

```
0x0804842d <+0>: push    ebp
0x0804842e <+1>: mov     ebp,esp
0x08048430 <+3>: sub     esp,0x10
0x08048433 <+6>: mov     DWORD PTR [ebp-0x8],0x3
0x0804843a <+13>: mov     DWORD PTR [ebp-0x4],0x4
0x08048441 <+20>: mov     eax,DWORD PTR [ebp-0x4]
0x08048444 <+23>: mov     edx,DWORD PTR [ebp-0x8]
0x08048447 <+26>: add     eax,edx
0x08048449 <+28>: mov     DWORD PTR [esp+0x4],eax
0x0804844d <+32>: mov     DWORD PTR [esp],0x80484e0
0x08048454 <+39>: call    0x80482e0 <printf@plt>
0x08048459 <+44>: mov     eax,0x0
0x0804845e <+49>: leave
0x0804845f <+50>: ret
```



printf("%d", 7); 호출

Hand-Ray

Hand-Ray란?

변수가 들어갈 공간을 확보 한다(프롤로그 이후 sub 명령 => 변수 선언일 경우가 유력)

두 변수에 int값을 각각 할당(스택에 저장)

연산을 하기 위해 두 값을 레지스터 eax, edx에 복사하고, add 연산을 통해 eax가 7이 된다.

printf 함수를 호출하기 전 파라미터 저장

printf("%d", 7); 호출

Hand-Ray

Hand-Ray란?

```
int v1 = 3;  
int v2 = 4;  
  
printf("%d", v1 + v2);
```

변수 선언, 제어문(if), 반복문(for), 함수 호출

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

프로로그 직후 sub명령? -> 지역 변수가 들어갈 공간을 할당

변수 선언, 제어문(if), 반복문(for), 함수 호출

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int a;
6     scanf("%d", a);
7
8     if(a == 7)
9         printf("Hello\n");
10
11     return 0;
12 }

```

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(if), 반복문(for), 함수 호출

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int a;
6     scanf("%d", a);
7
8     if(a == 7)
9         printf("Hello\n");
10
11     return 0;
12 }

```

Dump of assembler code for function main:

```

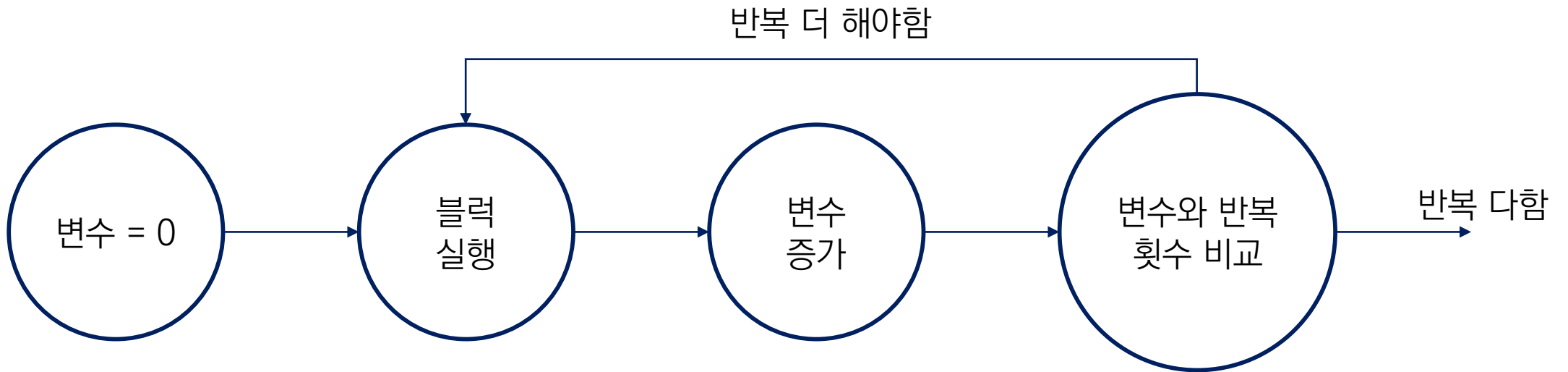
0x0804847d <+0>:    push    ebp
0x0804847e <+1>:    mov     ebp,esp
0x08048480 <+3>:    sub     esp,0xc
0x08048483 <+6>:    mov     eax,DWORD PTR [ebp-0x4]
0x08048486 <+9>:    mov     DWORD PTR [esp+0x4],eax
0x0804848a <+13>:   mov     DWORD PTR [esp],0x8048530
0x08048491 <+20>:   call    0x8048340 <_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(if), 반복문(for), 함수 호출



변수 선언, 제어문(if), 반복문(for), 함수 호출

```
1 #include <stdio.h>
2
3 int main()
4 {
5     for(int i = 0; i < 10; i++)
6         printf("Hello\n");
7
8     return 0;
9 }
```

```
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

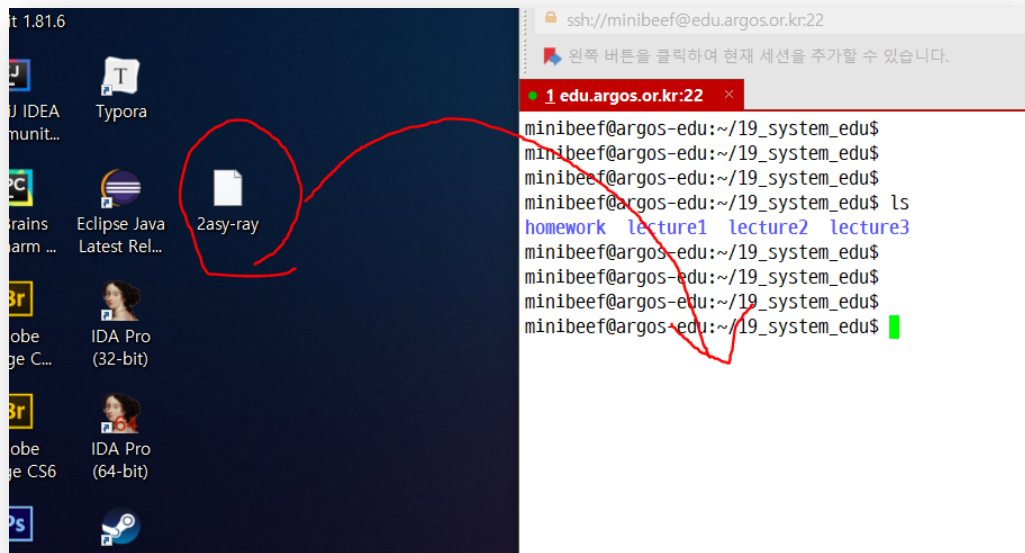
변수 선언, 제어문(if), 반복문(for), 함수 호출

```
1 #include <stdio.h>
2
3 int sum(int a, int b, int c) {
4     return a + b + c;
5 }
6 int main()
7 {
8     printf("%d", sum(1, 2, 3));
9     return 0;
10 }
```

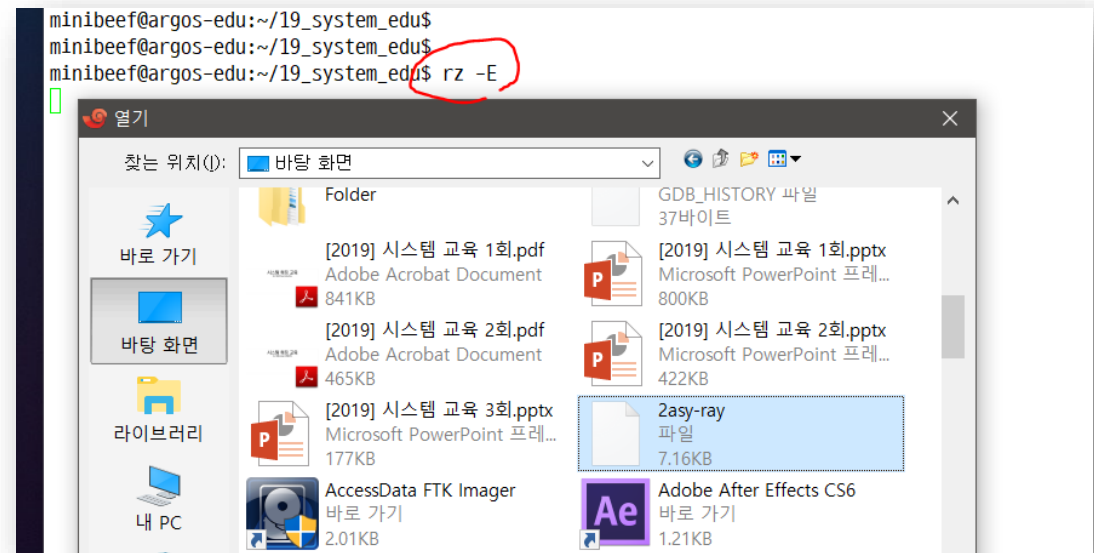
```
0x000005ab <+15>:    push    0x3
0x000005ad <+17>:    push    0x2
0x000005af <+19>:    push    0x1
0x000005b1 <+21>:    call    0x580 <sum>
```

32 비트 = 함수 호출 직전, 스택의 꼭대기에 파라미터 저장

Hand-Ray (실습) 2asy-ray



Xshell에서 Drag&Drop



rz -E

Hand-Ray

(실습) 2asy-ray

```
minibee@argos-edu:~/19_system_edu$ ls
2asy-ray homework lecture1 lecture2 lecture3
minibee@argos-edu:~/19_system_edu$ chmod +x 2asy-ray
minibee@argos-edu:~/19_system_edu$ ls
2asy-ray homework lecture1 lecture2 lecture3
minibee@argos-edu:~/19_system_edu$
```

Hand-Ray

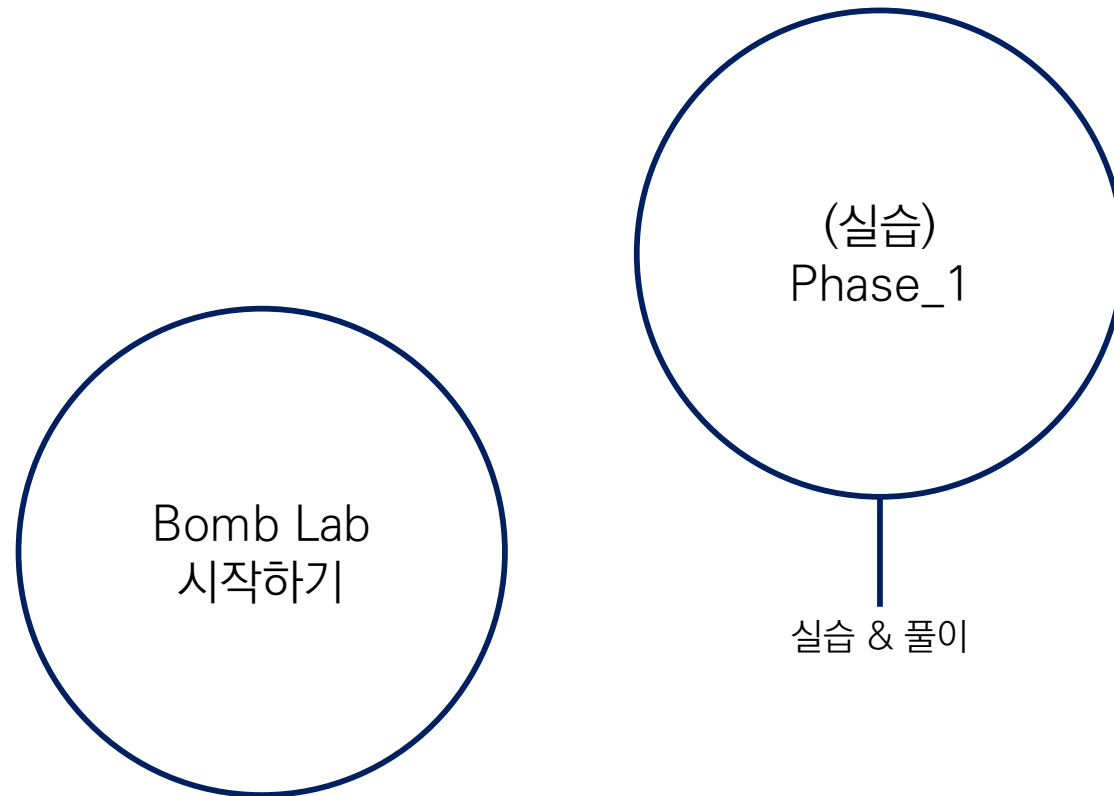
(실습) 2asy-ray

```
minibee@argos-edu:~/19_system_edu$ ./2asy-ray
guessing magic number
add
sub
add
guess plz >> XXXXXXXXXX
correct!
minibee@argos-edu:~/19_system_edu$
```

correct가 뜨도록 gdb로 분석하기 ㅎㅎ
(+메모장에 c코드로 바꿔서 쳐보기)

틀리면 터집니다..

Bomb Lab



Bomb Lab 시작하기

Bomb Lab?

리버싱을 연습하기 좋게 만들어 놓은 폭탄 게임.

1~6 단계 존재, 어셈블리 분석을 통해 각 단계에 부합하는 입력을 넣으면 클리어
단, 틀린 답을 넣으면 `explode_bomb` 함수가 실행되며 패배한다.

2학년 2학기 “시스템 프로그래밍” 과목에서 다루게 되므로 지금 잘 해놓으면 A+(?)

Bomb Lab 시작하기

```
0x0000000000400e8d phase_1  
0x0000000000400ea9 phase_2  
0x0000000000400f11 phase_3  
0x0000000000400fb8 func4  
0x0000000000400ff3 phase_4  
0x0000000000401060 phase_5  
0x00000000004010ec phase_6
```

각 단계에 브레이크포인트 (b* ~~)를 걸고 분석

(실습) phase_1

```
gdb-peda$ disas phase_1
Dump of assembler code for function phase_1:
0x0000000000400e8d <+0>:      sub    rsp,0x8
0x0000000000400e91 <+4>:      mov    esi,0x4023b0
0x0000000000400e96 <+9>:      call  0x40131d <strings_not_equal>
0x0000000000400e9b <+14>:     test   eax,eax
0x0000000000400e9d <+16>:     je     0x400ea4 <phase_1+23>
0x0000000000400e9f <+18>:     call  0x40141c <explode_bomb>
0x0000000000400ea4 <+23>:     add    rsp,0x8
0x0000000000400ea8 <+27>:     ret
End of assembler dump.
gdb-peda$
```

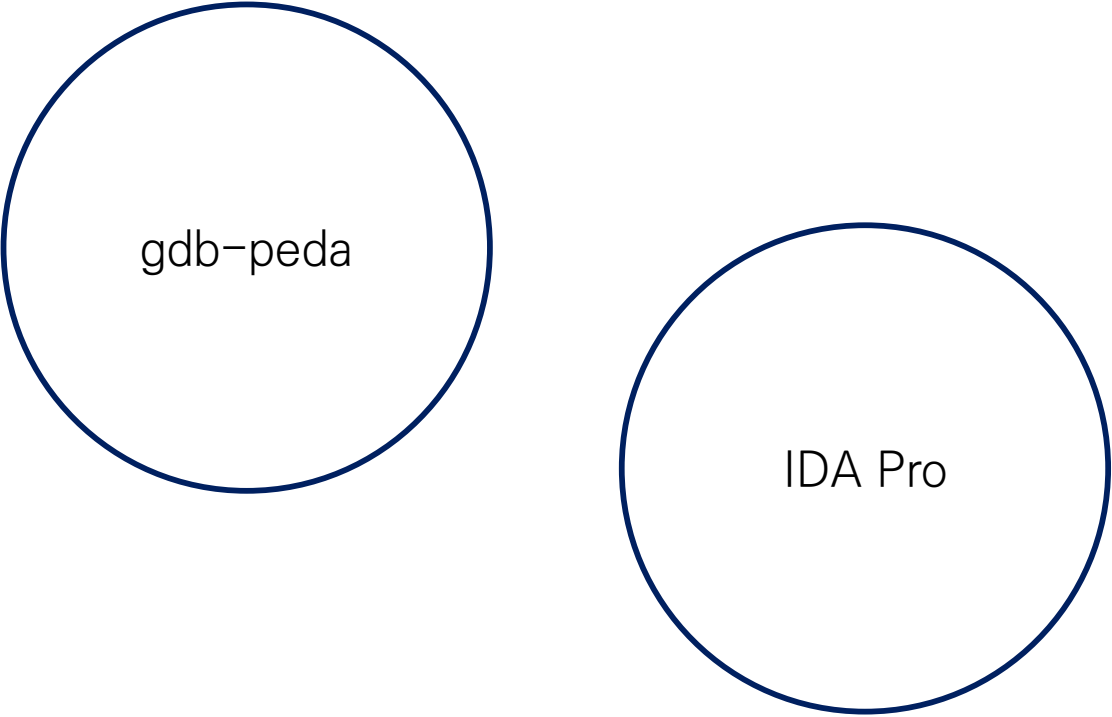
disas phase_1 하면 1 단계가 뜬다.. explode_bomb을 피해서 가보자!

힌트 1) string_not_equal..? 문자열이 다르면??

힌트 2) GDB Cheat Sheet에 x 명령어

분석을 더욱 풍요롭게

분석 툴



gdb-peda

IDA Pro

분석 툴

`gdb-peda`

```
(gdb) source /usr/share/peda/peda.py  
gdb-peda$ █
```

교육서버 gdb 상에서 `source /usr/share/peda/peda.py`

Gdb를 조금 더 스마트하게 쓸 수 있도록 고안된 플러그인

셸코드 제작, plt got, 문자열 찾기, 어셈에 하이라이팅... 무수한 기능 추가

gdb-peda

```

gdb-peda$ 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.

```

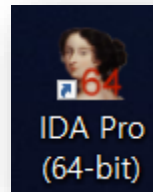
```

gdb-peda$ pd 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.

```

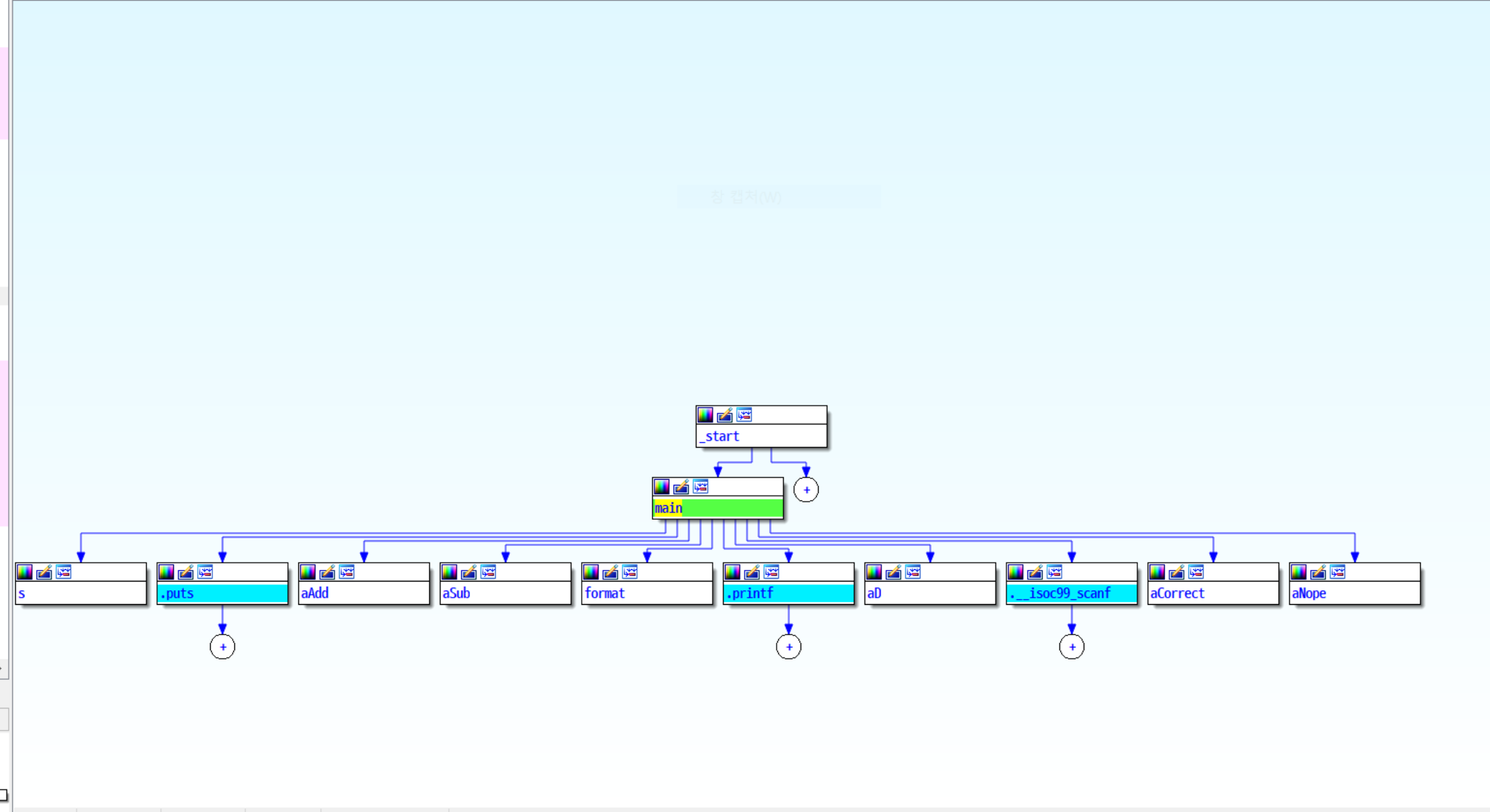
분석 툴

IDA Pro

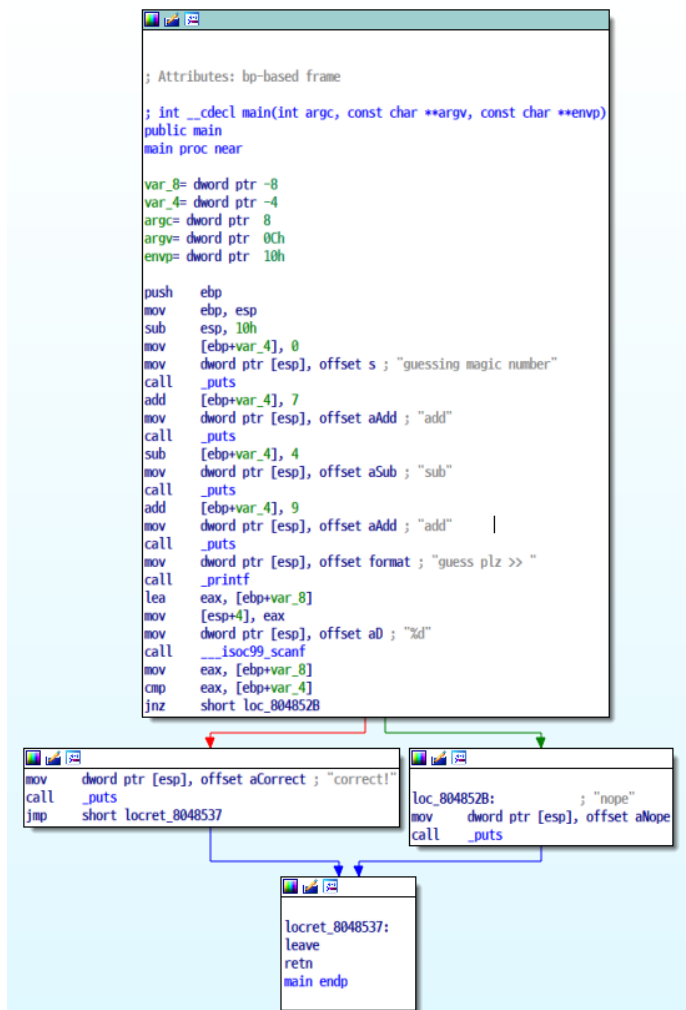


미-친듯이 강력함

Graph overview



IDA Pro



```

.text:004048AD      push    ebp
.text:004048AE      mov     ebp, esp
.text:004048B0      sub     esp, 10h
.text:004048B3      mov     [ebp+var_4], 0
.text:004048BA      mov     dword ptr [esp], offset s ; "guessing magic number"
.text:004048C1      call    _puts
.text:004048C6      add     [ebp+var_4], 7
.text:004048CA      mov     dword ptr [esp], offset aAdd ; "add"
.text:004048D1      call    _puts
.text:004048D6      sub     [ebp+var_4], 4
.text:004048DA      mov     dword ptr [esp], offset aSub ; "sub"
.text:004048E1      call    _puts
.text:004048E6      add     [ebp+var_4], 9
.text:004048EA      mov     dword ptr [esp], offset aAdd ; "add"
.text:004048F1      call    _puts
.text:004048F6      mov     dword ptr [esp], offset format ; "guess plz >> "
.text:004048FD      call    _printf
.text:004048502     lea     eax, [ebp+var_8]
.text:004048505     mov     [esp+4], eax
.text:004048509     mov     dword ptr [esp], offset aD ; "%d"
.text:004048510     call    ___isoc99_scanf
.text:004048515     mov     eax, [ebp+var_8]
.text:004048518     cmp     eax, [ebp+var_4]
.text:00404851B     jnz     short loc_804852B
.text:00404851D     mov     dword ptr [esp], offset aCorrect ; "correct!"
.text:004048524     call    _puts
.text:004048529     jmp     short locret_8048537

.text:00404852B ; -----
.text:00404852B      loc_804852B:
.text:00404852B      ; CODE XREF: main+6E1j
.text:00404852B      mov     dword ptr [esp], offset aNope ; "nope"
.text:004048532      call    _puts
.text:004048537      locret_8048537:
.text:004048537      ; CODE XREF: main+7C1j
.text:004048537      leave
.text:004048538      retn
.text:004048538      main
.text:004048538      endp
    
```

분석 툴

IDA Pro

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int result; // eax@2
4     int v4; // [sp+8h] [bp-8h]@1
5     int v5; // [sp+Ch] [bp-4h]@1
6
7     puts("guessing magic number");
8     puts("add");
9     puts("sub");
10    v5 = 12;
11    puts("add");
12    printf("guess plz >> ");
13    __isoc99_scanf("%d", &v4);
14    if ( v4 == v5 )
15        result = puts("correct!");
16    else
17        result = puts("nope");
18    return result;
19 }
```

??

IDA Pro

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int result; // eax@2
4     int v4; // [sp+8h] [bp-8h]@1
5     int v5; // [sp+Ch] [bp-4h]@1
6
7     puts("guessing magic number");
8     puts("add");
9     puts("sub");
10    v5 = 12;
11    puts("add");
12    printf("guess plz >> ");
13    __isoc99_scanf("%d", &v4);
14    if ( v4 == v5 )
15        result = puts("correct!");
16    else
17        result = puts("nope");
18    return result;
19 }

```

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int answer = 0;
6     int input;
7
8     puts("guessing magic number");
9
10    answer += 7;
11    puts("add");
12
13    answer -= 4;
14    puts("sub");
15
16    answer += 9;
17    puts("add");
18
19    printf("guess plz >> ");
20    scanf("%d", &input);
21
22    if(input == answer)
23        puts("correct!");
24    else
25        puts("nope");
26 }

```

3회 끝

ls : 파일 목록 보기

cd [디렉토리] : 디렉토리 이동

mkdir [이름] : 디렉토리 생성

rm [파일명] : 파일 삭제

rm -r [디렉토리명] : 디렉토리 삭제

mv : 파일 이동(이름 변경으로 쓸 수 있음)

vi : 소스코드 생성

gcc -o [실행파일] [소스코드] : 컴파일

GDB Cheat Sheet

(1) 시작/종료

- 시작 : gdb [프로그램명]
- 종료 : quit 혹은 q

(2) 문법 변경

set disassembly-flavor intel

(3) 분석

- 해당 함수 코드 : disas [함수명]
- 실행 : run 또는 r
- 브레이크 포인트 : b [지점]
- 브레이크 포인트 걸린 위치 코드 : disas
- 브레이크 포인트 다 지우기 : d 또는 dis
- 다음 명령어 : ni
- 진행 : c
- 강제 점프 : jump [위치] -> 함수, 행, 메모리
- info func : 쓰인 함수 보기
- info r : 레지스터 보기

(4) 정보 수집 - x 명령어

x/[범위][출력형식]

〈출력형식〉

t : 2 진수

o : 8 진수

d : 10 진수

x : 16 진수

s : 문자열

i : 어셈블리

EX1) x/100i \$eip : 100줄의 명령어를 어셈으로 보겠다

EX2) x/s \$esi : esi 위치에 있는 문자열을 보겠다.

(5) 정보 수집 - p 명령어

p/[출력형식] [계산식] : 계산 결과 확인

계산식에는 여러가지 들어갈 수 있다.(레지스터, 변수 등등)