포너블 교육

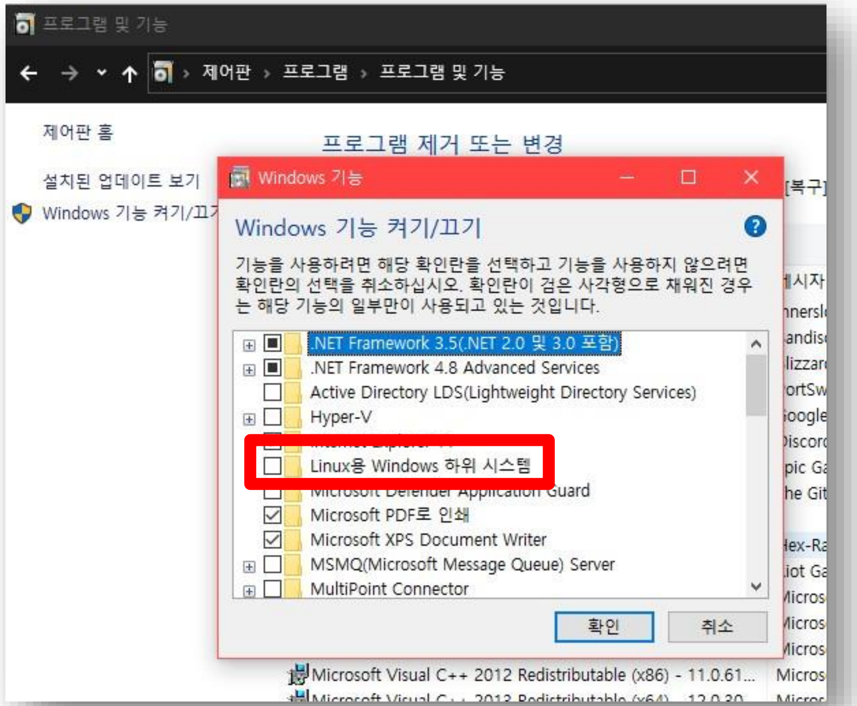
리버스엔지니어링 기초

index

금일 교육 구성

환경 구축

WSL 설치

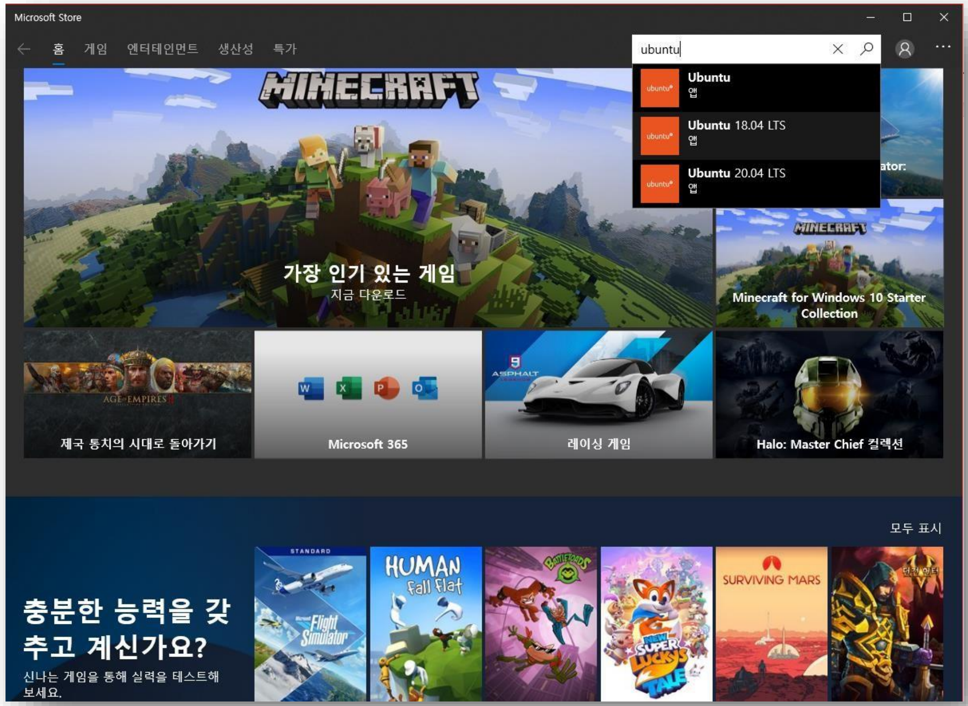


Windows 기능 켜기/끄기

기능을 사용하려면 해당 확인란을 선택하고 기능을 사용하지 않으려면 확인란의 선택을 취소하십시오. 확인란이 검은 사각형으로 채워진 경우는 해당 기능의 일부만이 사용되고 있는 것입니다.

- .NET Framework 3.5(.NET 2.0 및 3.0 포함)
- .NET Framework 4.8 Advanced Services
- Active Directory LDS(Lightweight Directory Services)
- Hyper-V
- Linux용 Windows 하위 시스템
- Microsoft Defender Application Guard
- [✓] Microsoft PDF로 인쇄
- [✓] Microsoft XPS Document Writer
- MSMQ(Microsoft Message Queue) Server
- MultiPoint Connector

Microsoft Visual C++ 2012 Redistributable (x86) - 11.0.61...



Microsoft Store
홈
게임
엔터테인먼트
생산성
특가
ubuntu
Ubuntu
Ubuntu 18.04 LTS
Ubuntu 20.04 LTS
가장 인기 있는 게임
지금 다운로드
Minecraft for Windows 10 Starter Collection
제국 통치의 시대로 돌아가기
Microsoft 365
레이싱 게임
Halo: Master Chief 컬렉션
모두 표시
충분한 능력을 갖추고 계신가요?
신나는 게임을 통해 실력을 테스트해 보세요.

제어판 -> 프로그램 및 기능 -> 좌측 탭의 Windows 기능 켜기/끄기 -> Linux용 Windows 하위 시스템 체크(다시시작 필요)

Microsoft Store에서 Ubuntu 18.04 LTS 설치

환경 구축

WSL 설치

아래 명령 차례대로 입력

```
sudo apt update
```

```
sudo apt install gdb gcc build-essential git
```

바이너리가 무엇인가?

바이너리(binary)

0과 1로 이루어진 이진 상태를 일컫는 말이다.
컴퓨터의 실행 파일을 의미하기도 한다.

* 이 ⑨의에서는 ELF(리눅스 실행파일)에 한정하여 바이너리라고 칭한다.

왜 리버싱을 하는가?

리버싱(역공학, Reverse Engineering)
완성된 제품(SW/OS/HW)을 분석하여 설계 단계로 돌리는 기술
(설계 -> 개발 -> 제품화의 역과정)

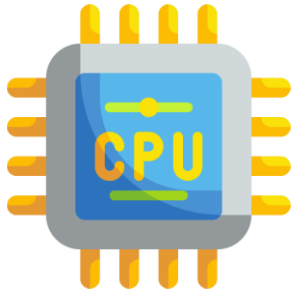
학생의 관점
- **바이너리(실행파일)와 기본적인 컴퓨터 시스템에 대한 이해도가 올라간다.**

개발자의 관점
- **프로그램에 존재하는 잠재적인 버그를 잡을 수 있다(코드를 한줄씩 실행하면서)**

해커(or 보안 전문가)의 관점
- **컴퓨터 시스템에 대한 이해도가 있는 상태에서 프로그램에 존재하는 보안상 약점을 발견할 수 있다.**

기초적인 컴퓨터 시스템

CPU 레지스터



1 + 2 = ? → 연산을 하기 위해서는 피연산자를 담을 공간이 필요 → CPU 레지스터

CPU 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 인자 값)

기초적인 컴퓨터 시스템

CPU 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

함수 호출 인자

함수가 호출될 때 필요한 인자들을 넘겨주는 역할

rdi rsi rcx rdx …

기초적인 컴퓨터 시스템

CPU 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

rsp

rip

스택 포인터

스택 메모리의 가장 위쪽을 가르킴. 스택은 함수가 사용할 지역 변수들을 저장하기 위해 준비해놓은 공간임.

CPU 레지스터

rax rbx rcx rdx rsi rdi

rbp r8 r9 r10 r11 r12 r13 r14 r15

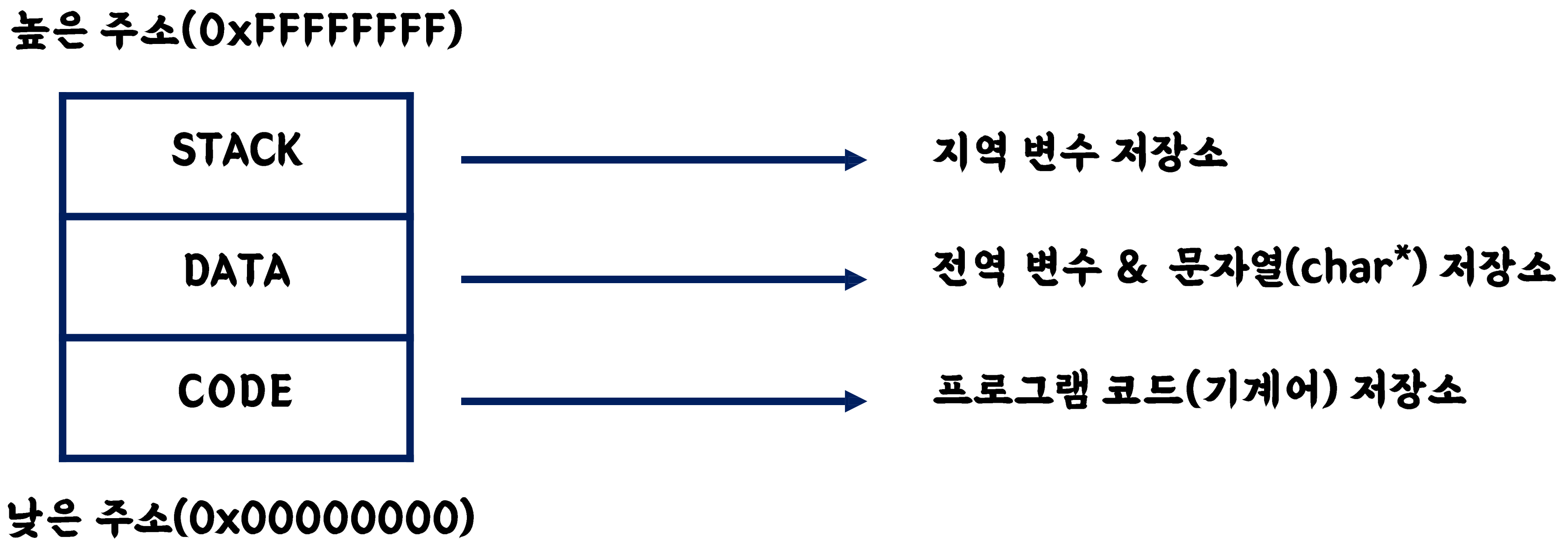
rsp

rip

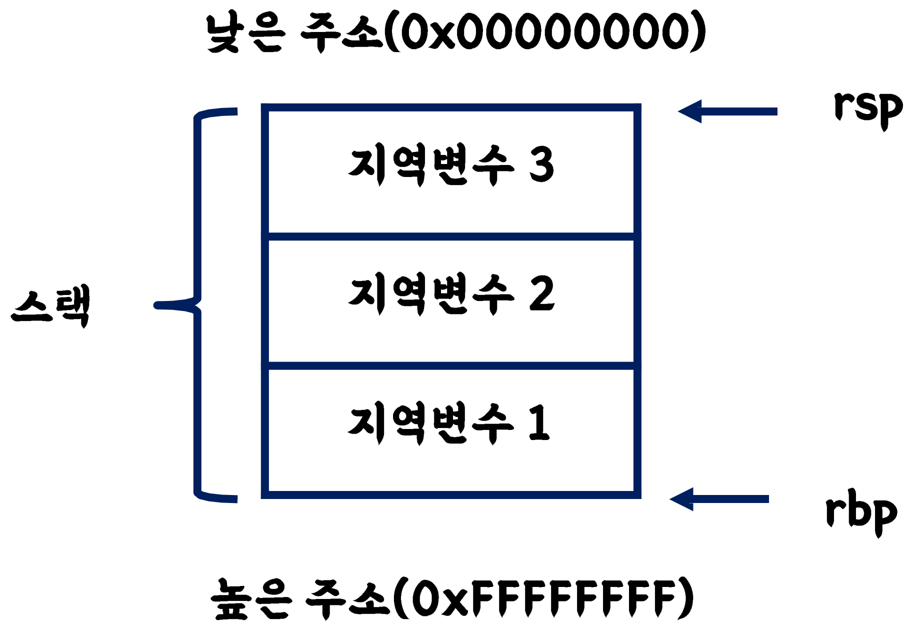
프로그램 카운터

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

메모리


높은 주소(0xFFFFFFFF)
STACK
DATA
CODE
낮은 주소(0x00000000)
지역 변수 저장소
전역 변수 & 문자열(char*) 저장소
프로그램 코드(기계어) 저장소

메모리


낮은 주소(0x00000000)
rsp
지역변수 3
지역변수 2
지역변수 1
스택
rbp
높은 주소(0xFFFFFFFF)

어셈블리 형태

$$\text{[연산자] } \overbrace{\text{[피연산자1] [피연산자2]}}^{\text{상수, 레지스터, 주소 등..}}$$

*예시 : add rax, rbx

어셈블리 기초

대표적인 연산자

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.

어셈블리 기초

대표적인 연산자

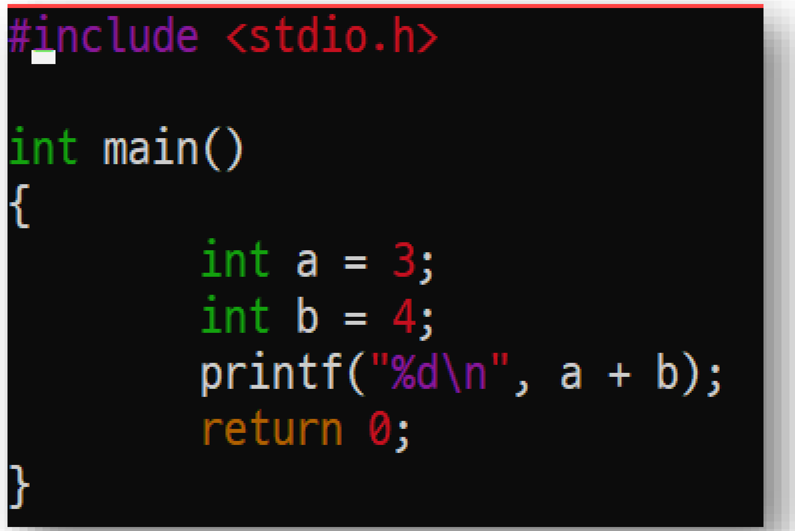
add a, b	a와 b를 더해서 a에 결과를 넣는다 (a += b)
sub a, b	a와 b를 뺀 결과를 a에 넣는다 (a -= b)
imul a, b	a와 b를 곱한 결과를 a에 너허는다 (a *=b)
xor a, b	a와 b를 xor 한 결과를 a 에 넣는다 (a^=b)

어셈블리 기초

대표적인 연산자

jmp	해당 코드로 점프
je	cmp 결과가 같으면 점프
jne	cmp 결과가 다르면 점프
call	함수 호출

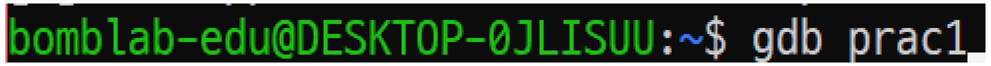
실습 1 - 계산기



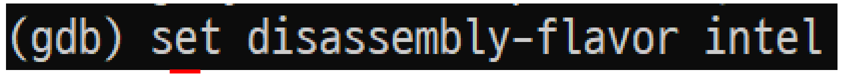
```c
#include <stdio.h>

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

코드 작성 후 컴파일

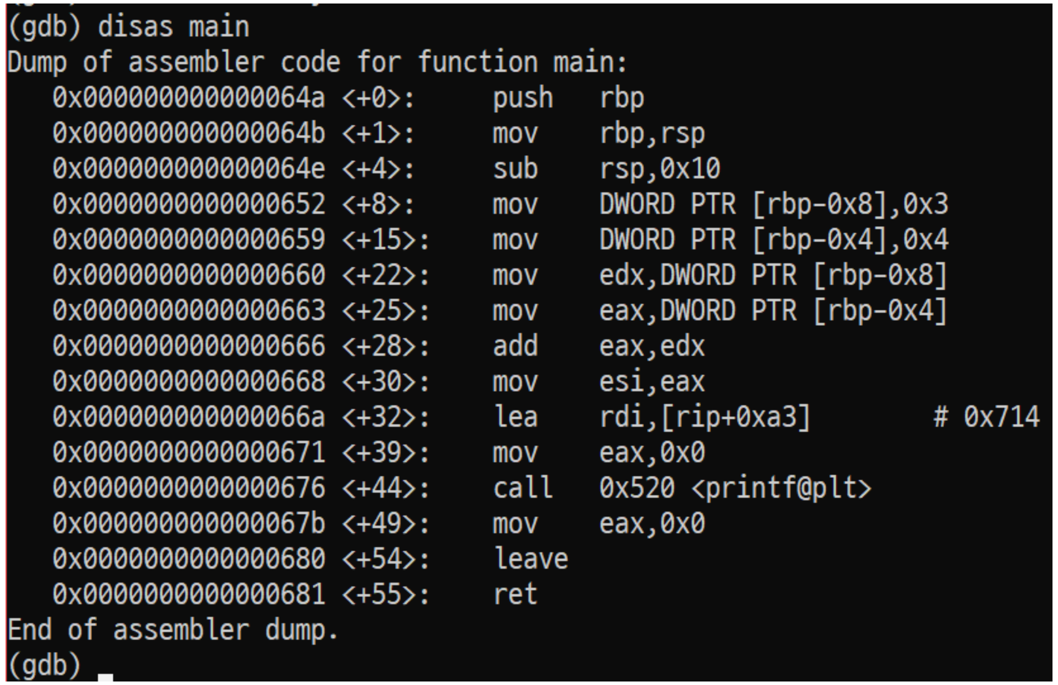
실습 1 - 계산기



```
bomblab-edu@DESKTOP-0JLISUU:~$ gdb prac1
```

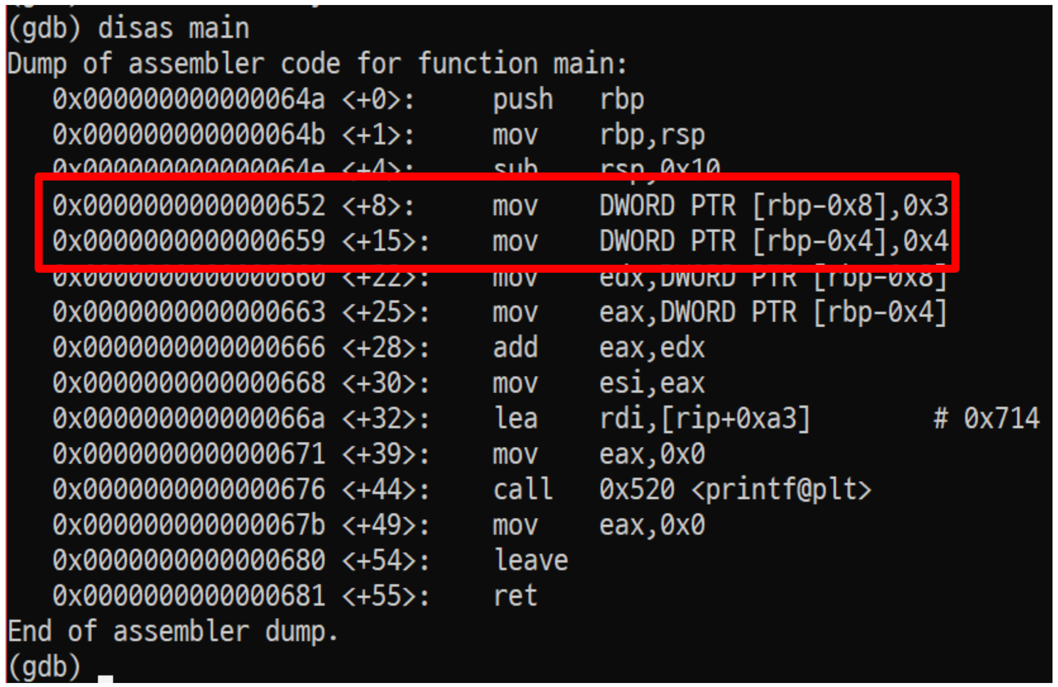


```
(gdb) set disassembly-flavor intel
```

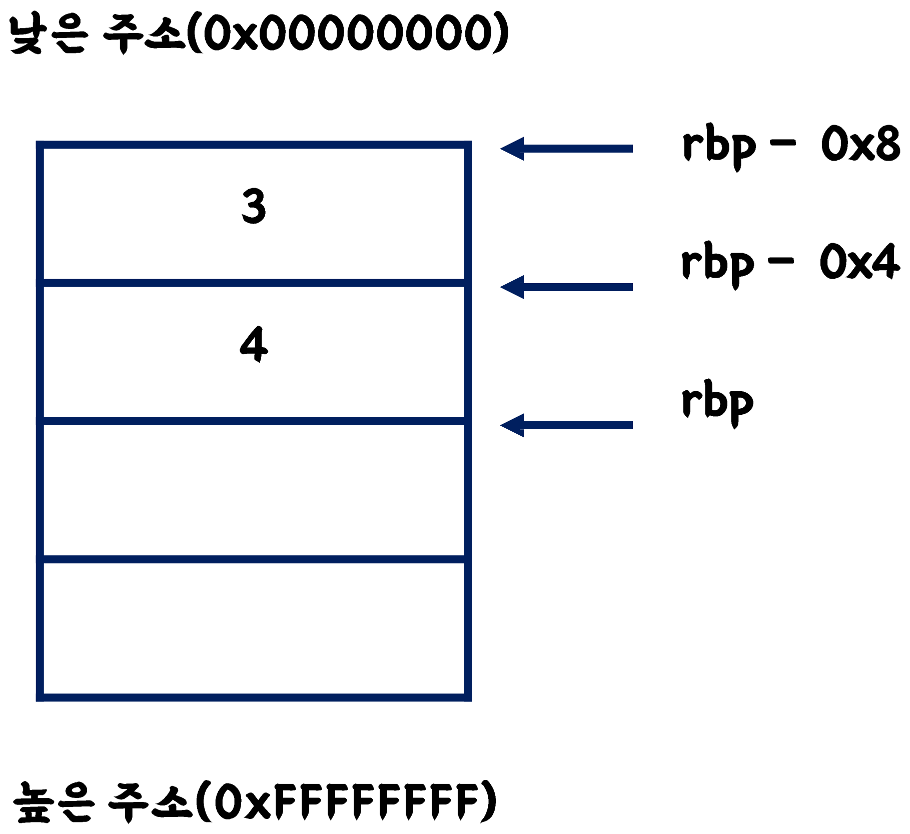


```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    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.
(gdb)
```

실습 1 - 계산기

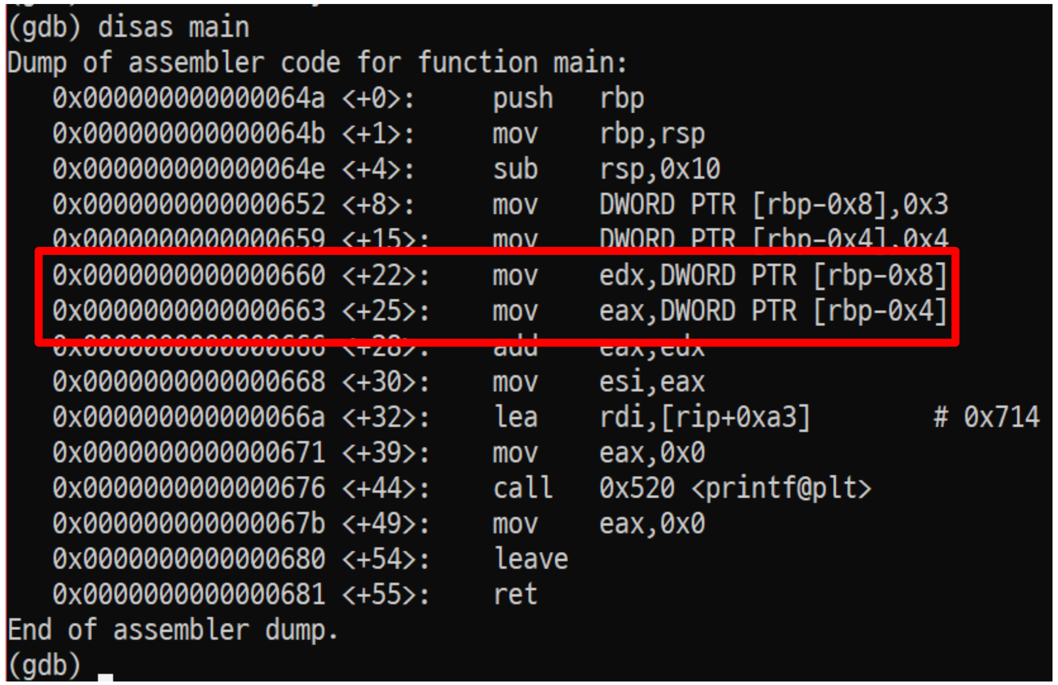


```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    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.
(gdb) _
```

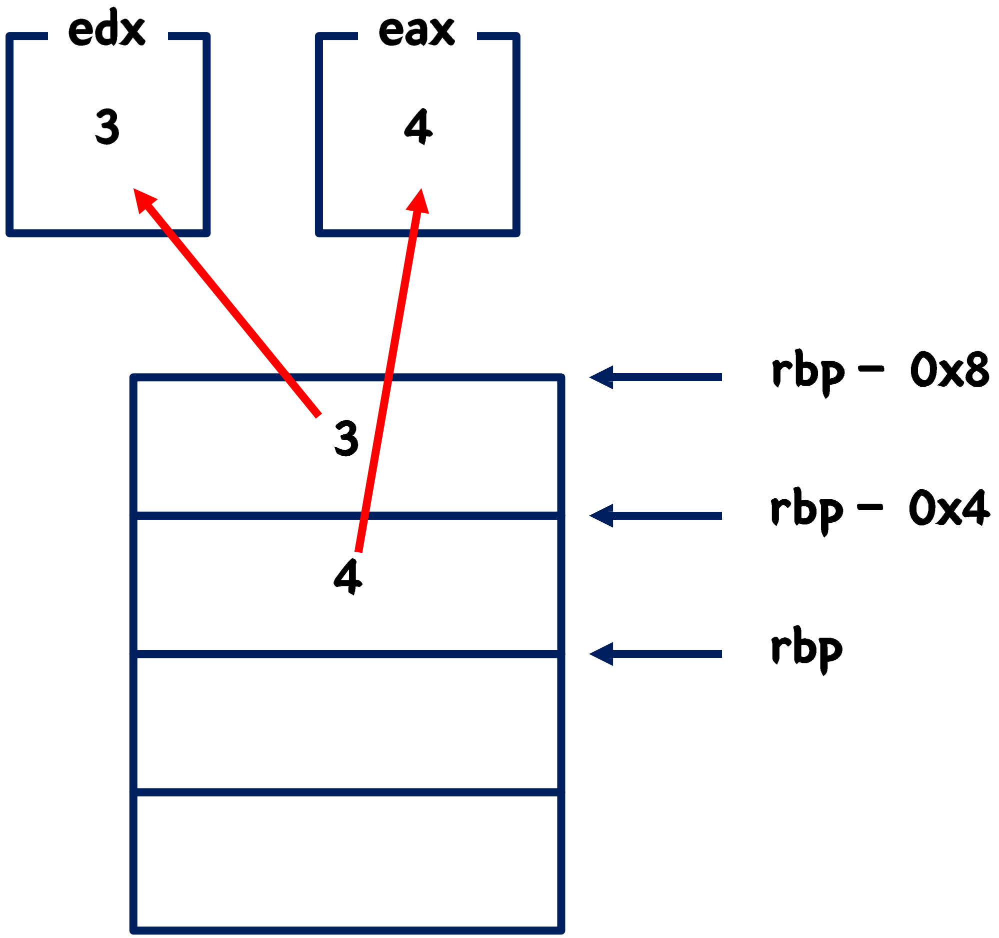


어셈블리 기초

실습 1 - 계산기

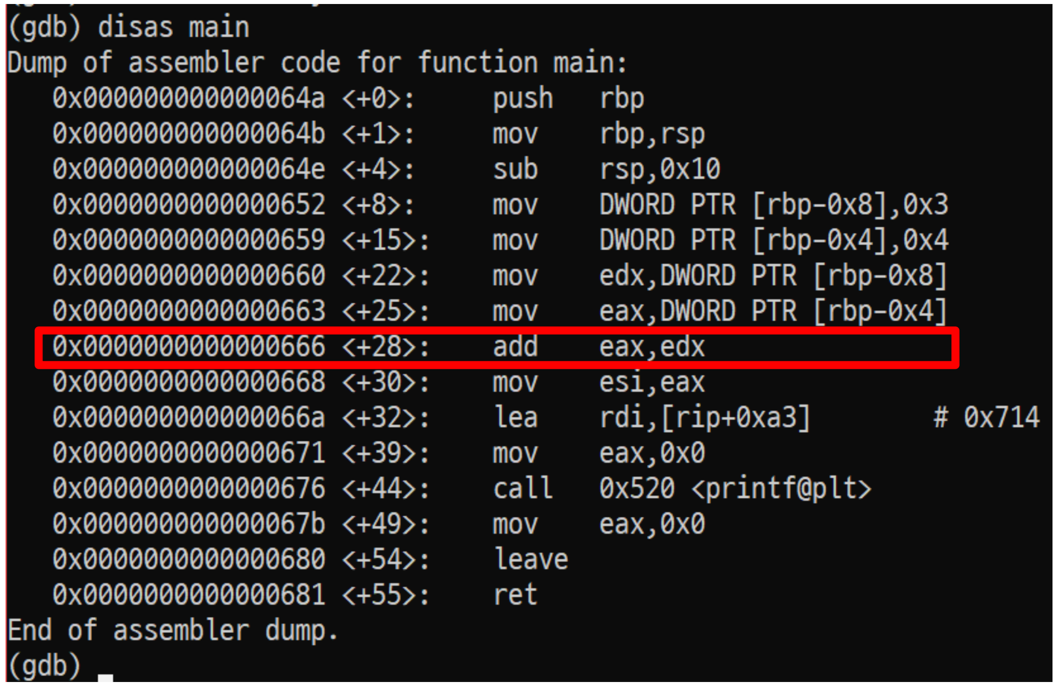


```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    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.
(gdb) _
```

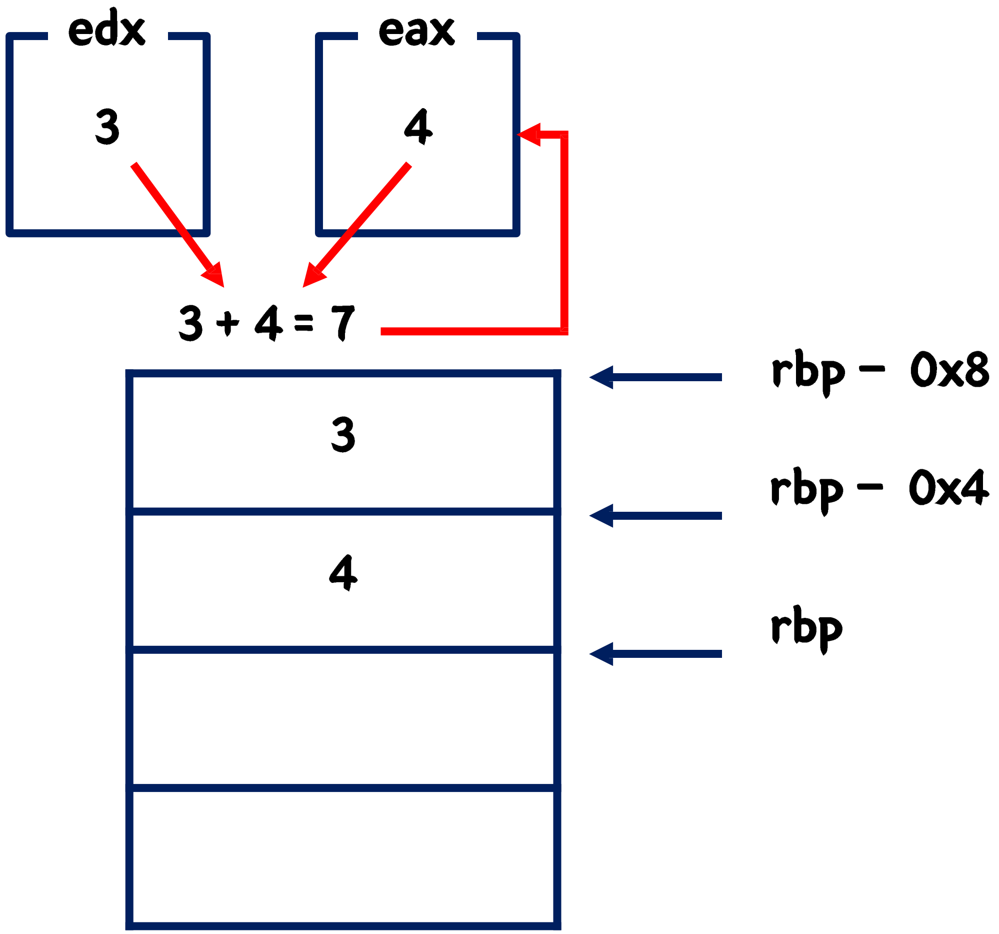


어셈블리 기초

실습 1 - 계산기

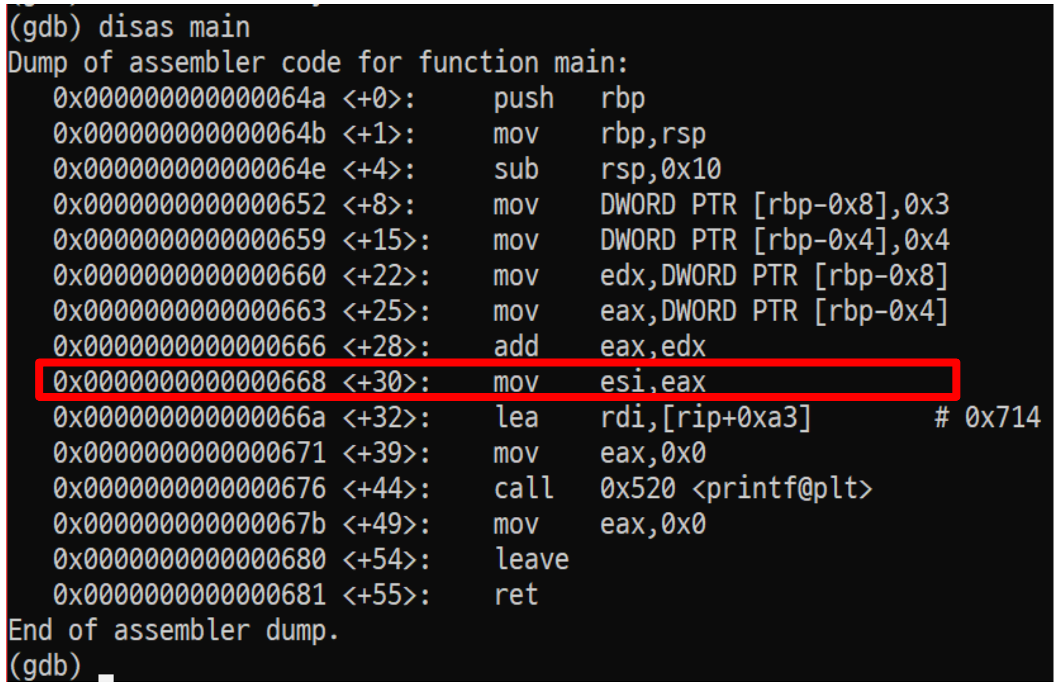


```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    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.
(gdb)
```

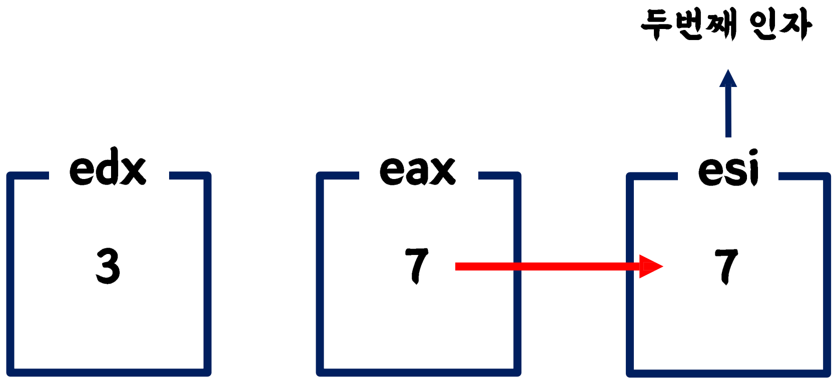


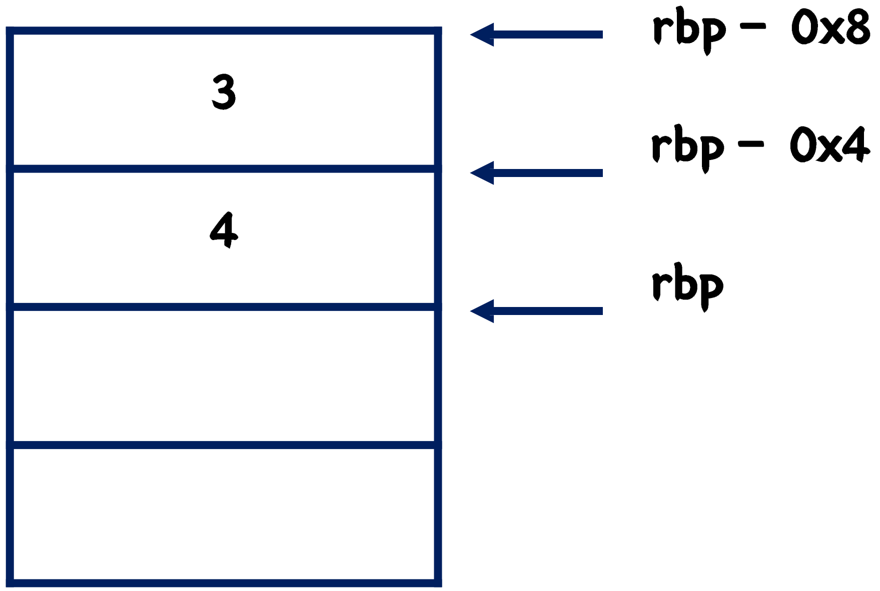
어셈블리 기초

실습 1 - 계산기



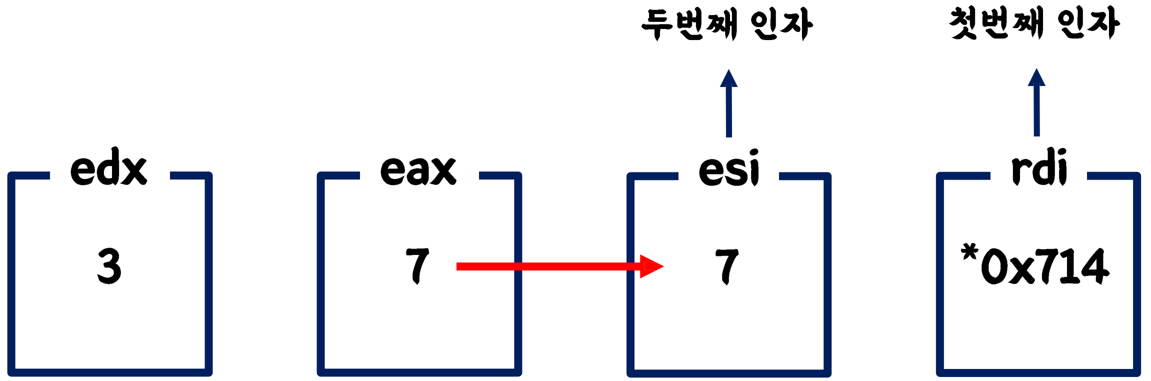
```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    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.
(gdb)
```

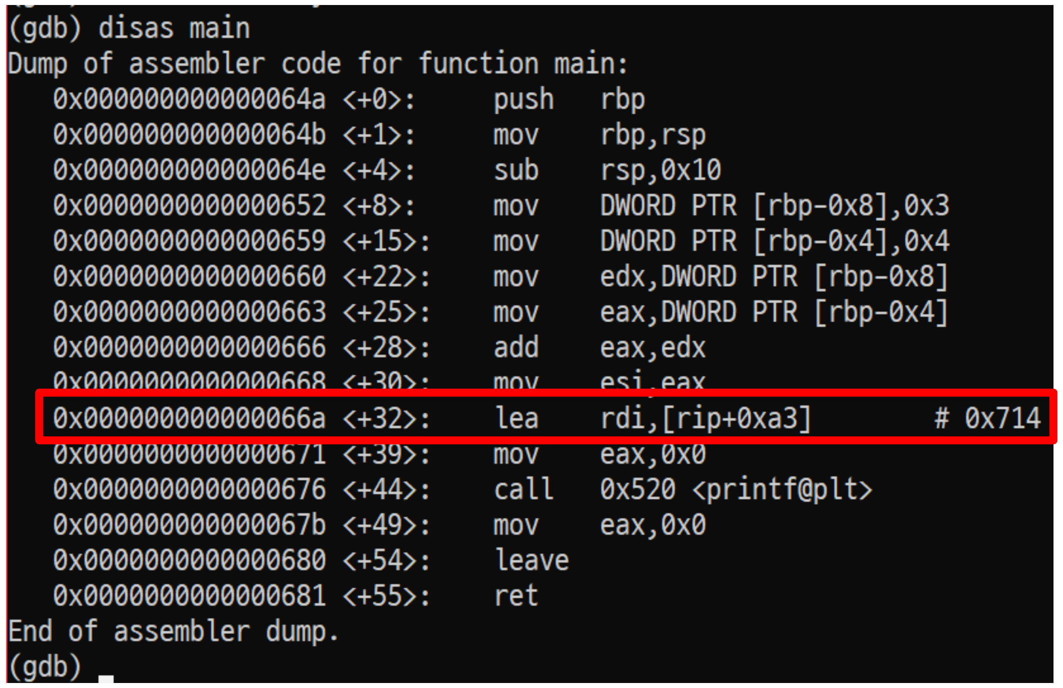




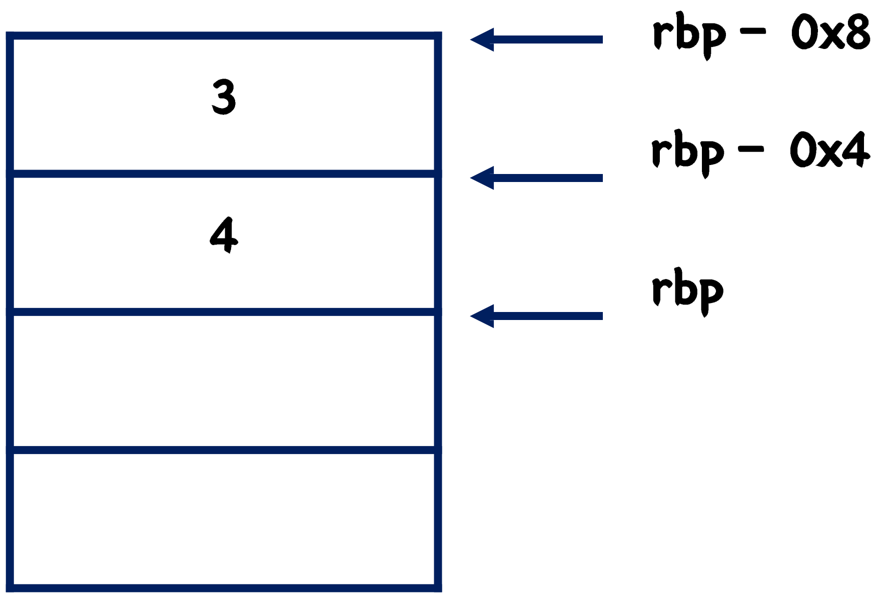
어셈블리 기초

실습 1 - 계산기



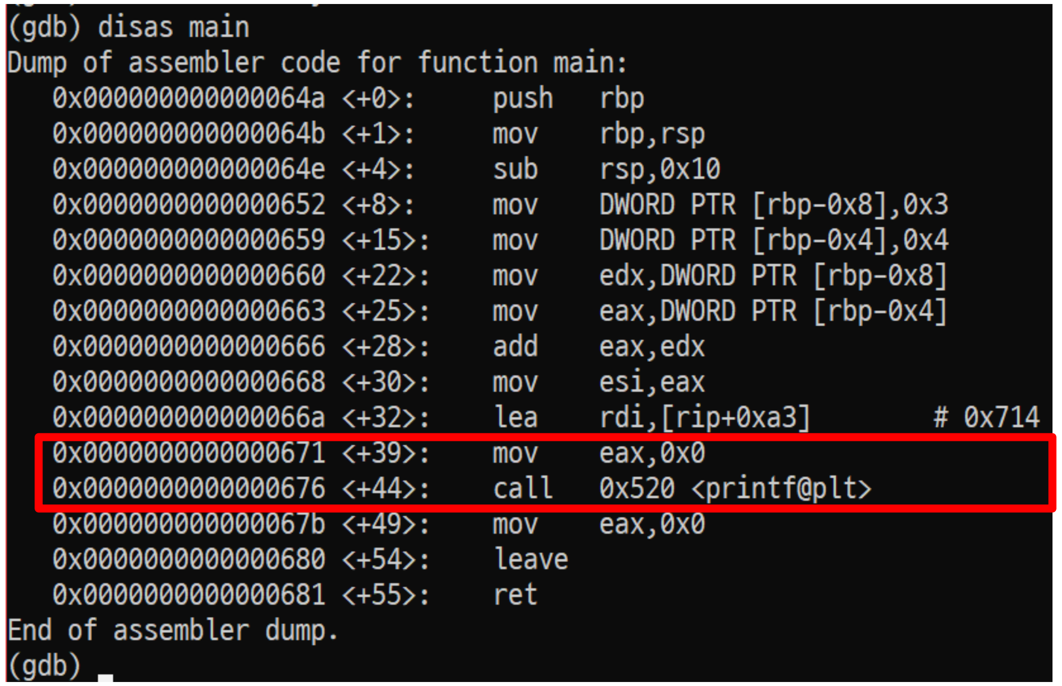


```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    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.
(gdb)
```

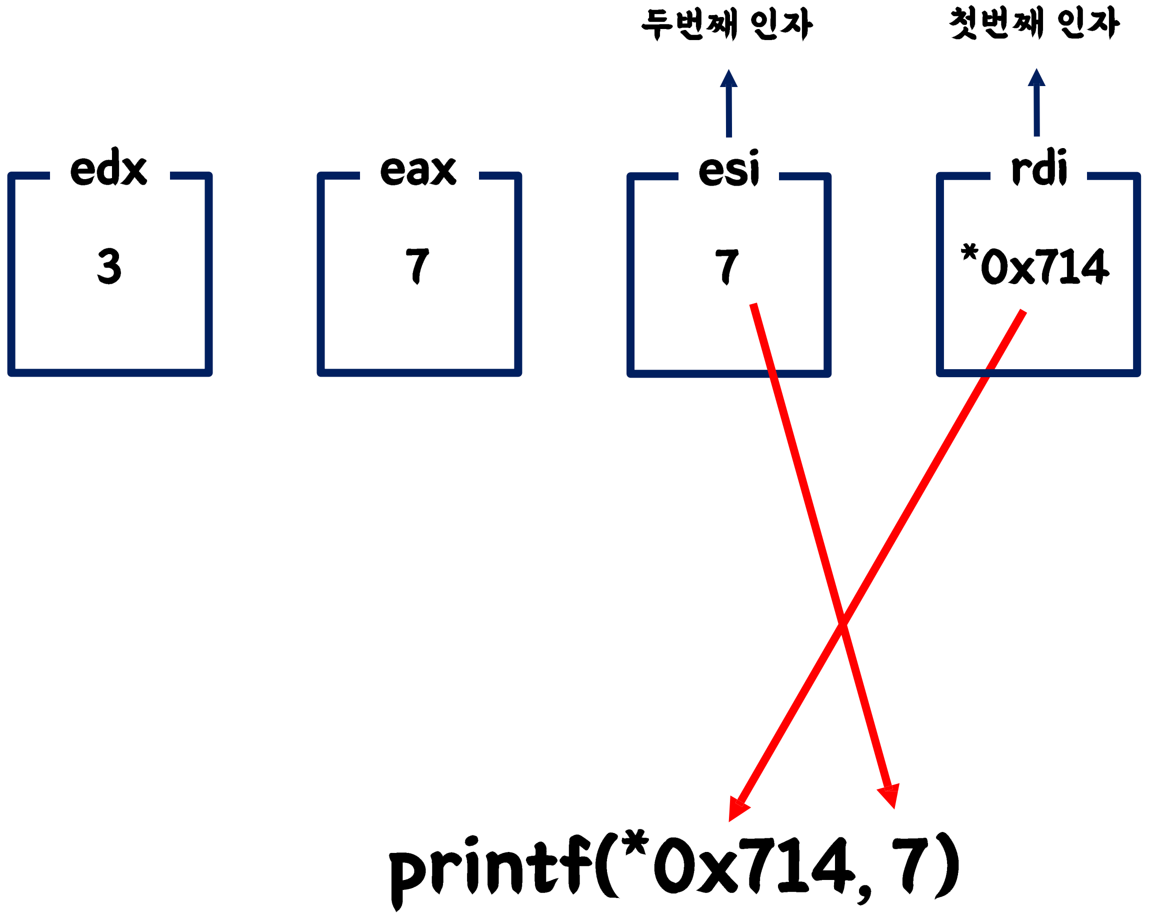


어셈블리 기초

실습 1 - 계산기

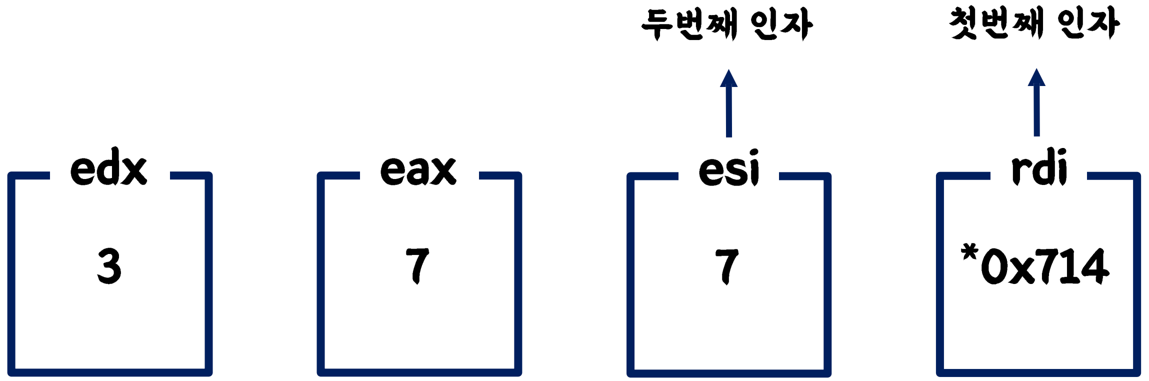


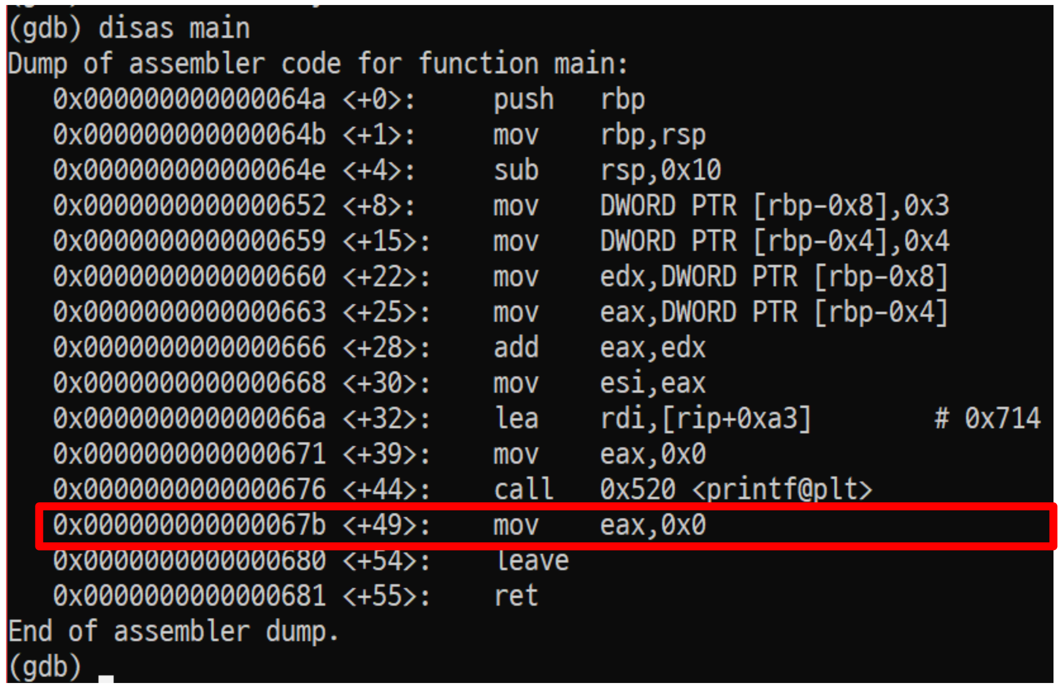
```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    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.
(gdb)
```



어셈블리 기초

실습 1 - 계산기



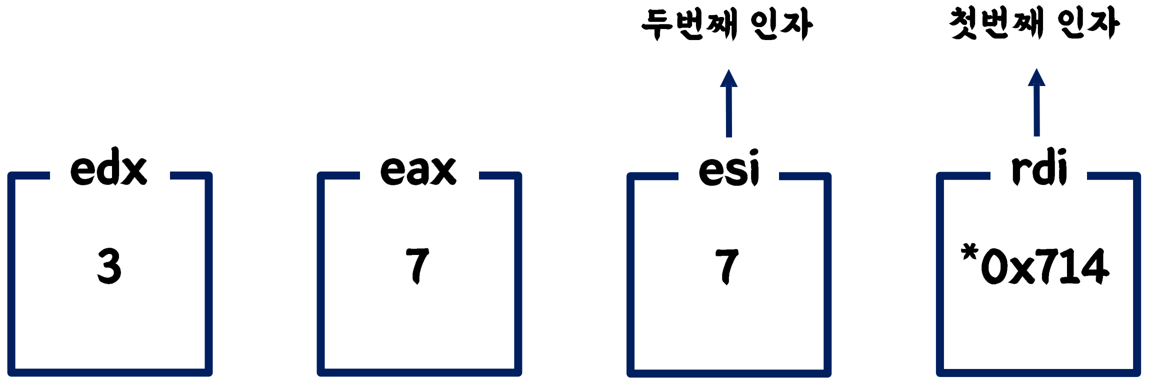


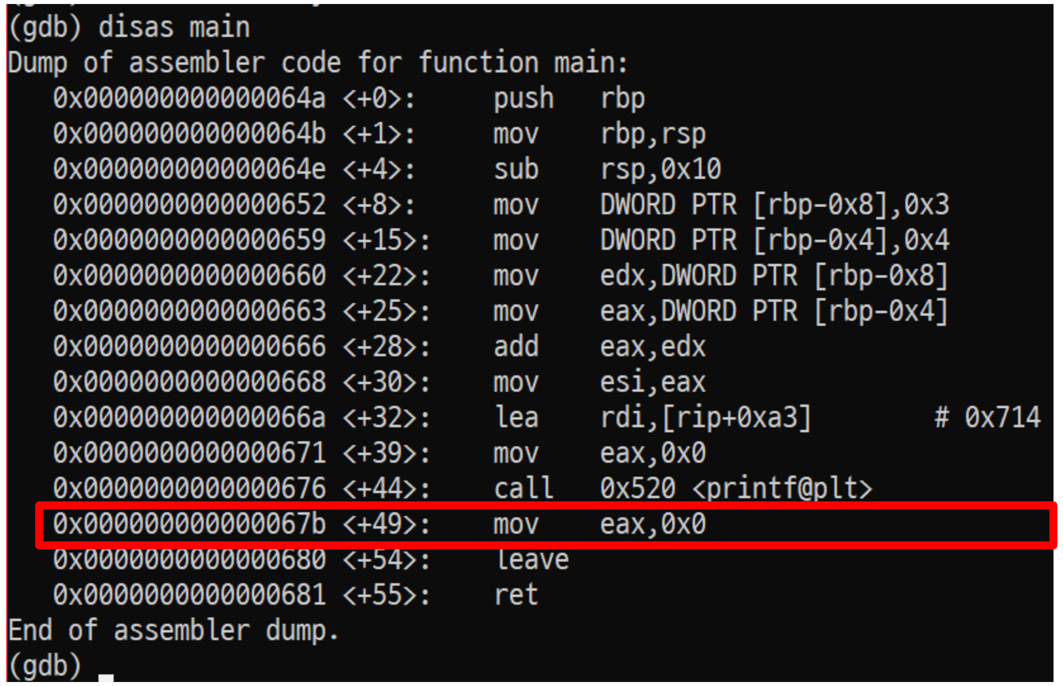
```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    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.
(gdb)
```

printf(*0x714, 7)
return 0;

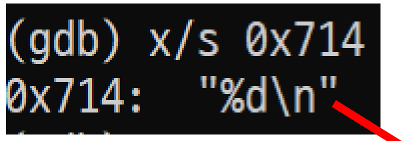
어셈블리 기초

실습 1 - 계산기





```
(gdb) disas main
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.
(gdb)
```

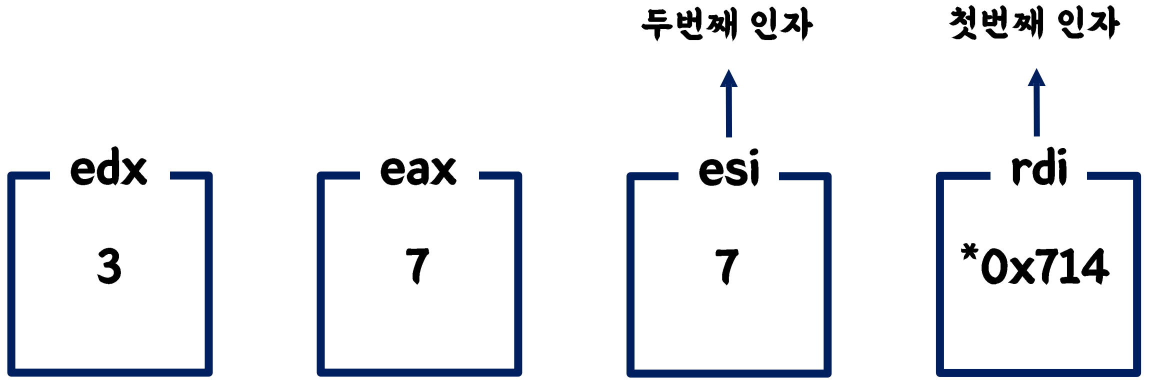


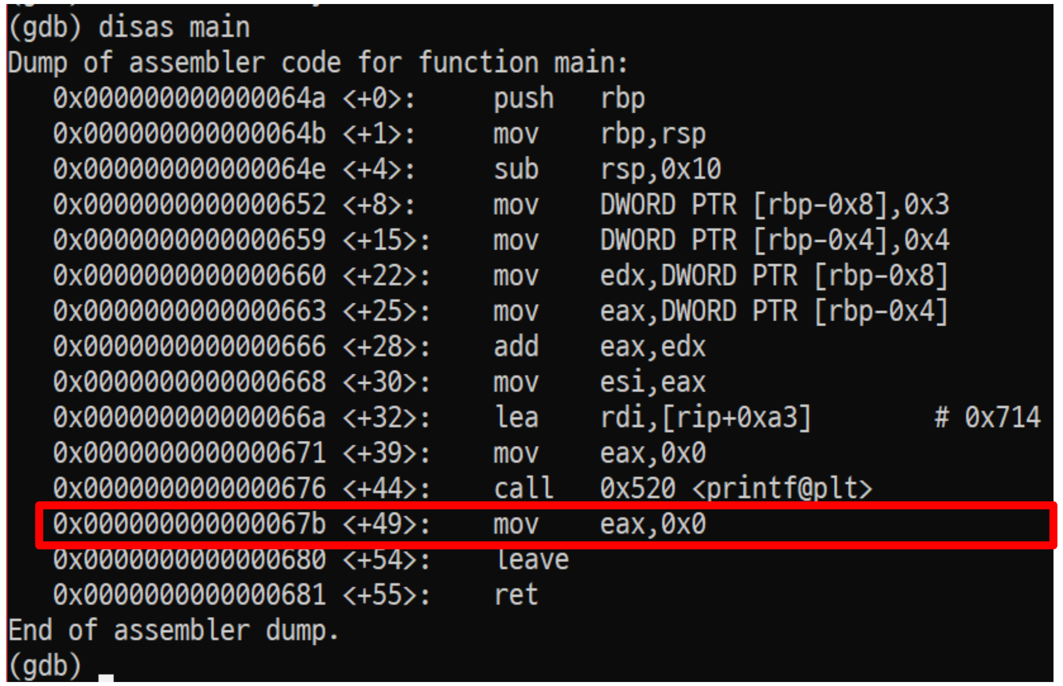
```
(gdb) x/s 0x714
0x714:   "%d\n"
```

printf("%d\n", 7)
return 0;

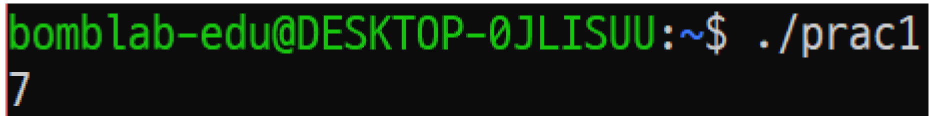
어셈블리 기초

실습 1 - 계산기





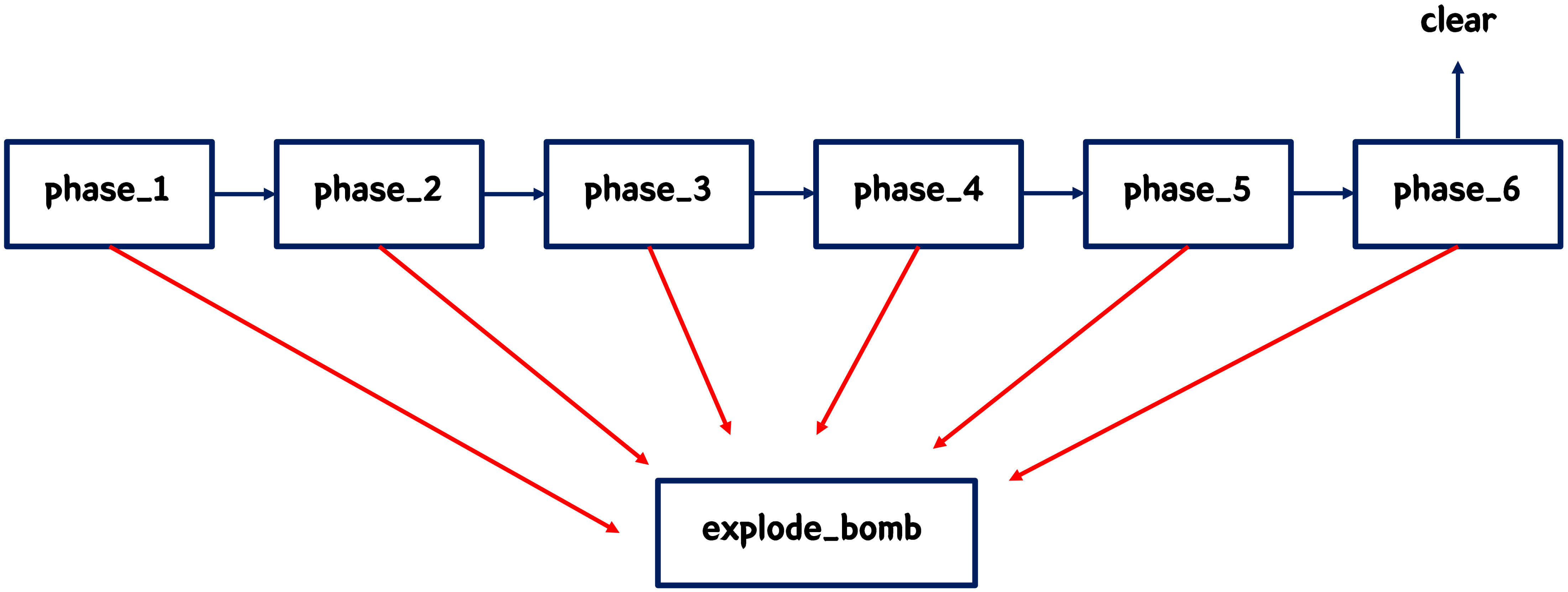
```
(gdb) disas main
Dump of assembler code for function main:
   0x000000000000064a <+0>:     push   rbp
   0x000000000000064b <+1>:     mov    rbp,rsp
   0x000000000000064e <+4>:     sub    rsp,0x10
   0x0000000000000652 <+8>:     mov    DWORD PTR [rbp-0x8],0x3
   0x0000000000000659 <+15>:    mov    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.
(gdb)
```



```
bomblab-edu@DESKTOP-0JLISUU:~$ ./prac1
7
```

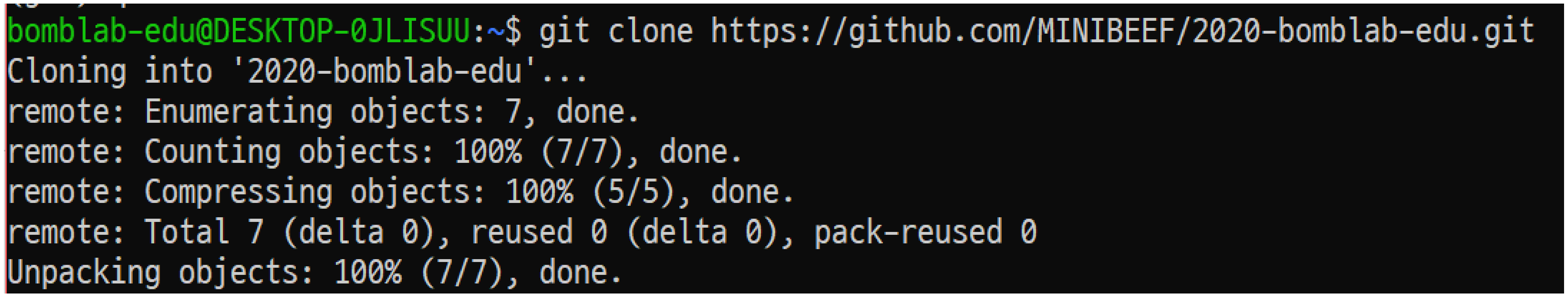
밤랩 소개 및 1단계 해법

BOMBLAB



밤랩 소개 및 1단계 해법

BOMBLAB



```
bomblab-edu@DESKTOP-0JLISUU:~$ git clone https://github.com/MINIBEEF/2020-bomblab-edu.git
Cloning into '2020-bomblab-edu'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.
```

```
cd 2020-bomblab-edu ; chmod +x bomb
```

밤랩 소개 및 1단계 해법

BOMBLAB

b* 위치 : 브레이크 포인트

ni : 다음 스텝

disas : 현재 위치

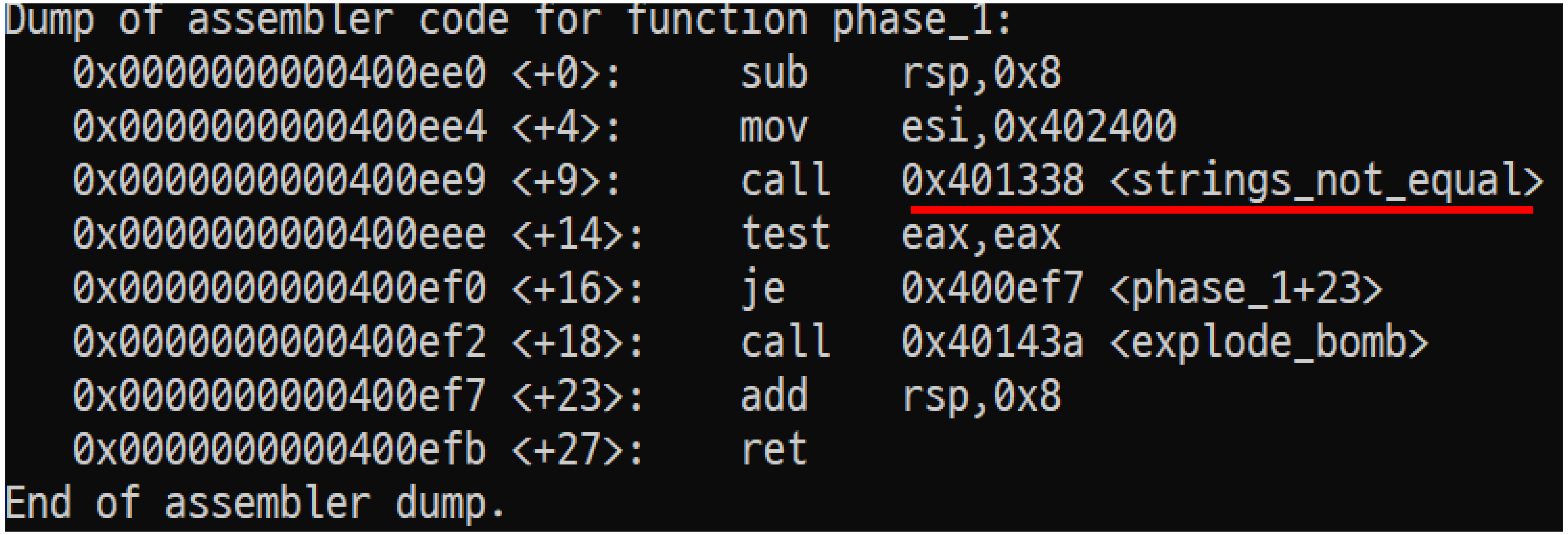
disas 함수이름 : 함수 코드 보기

i r : 현재 레지스터 상태

r : 시작

밤랩 소개 및 1단계 해법

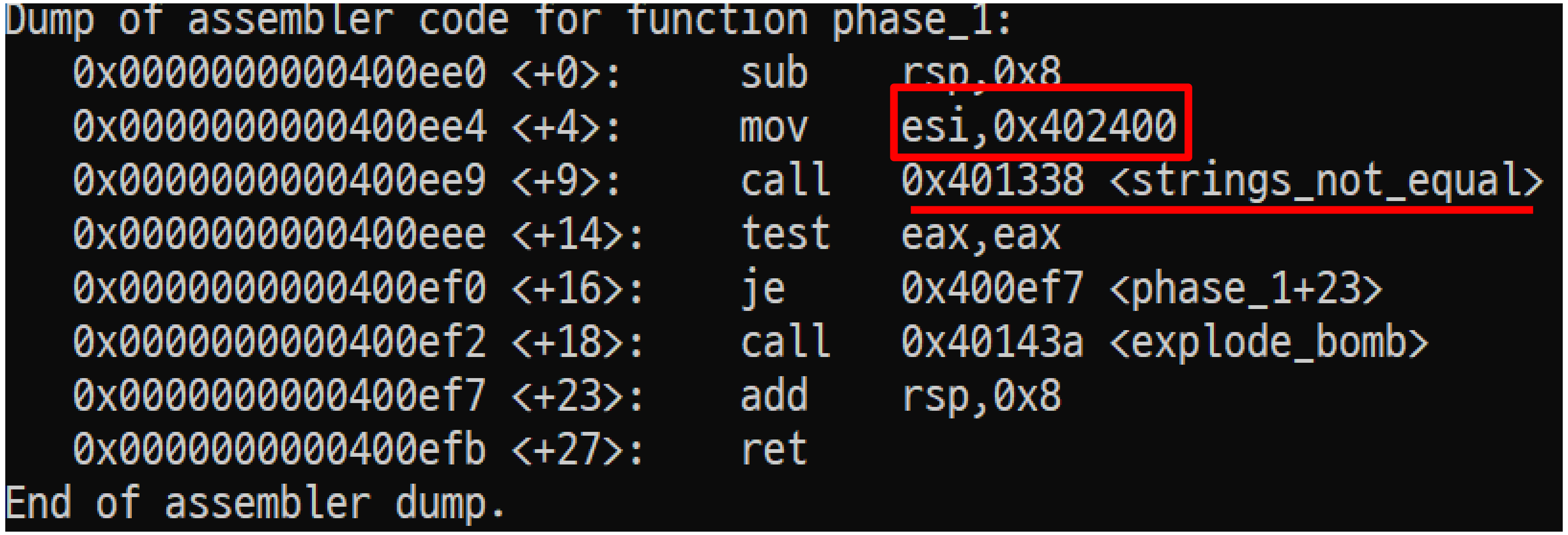
BOMBLAB



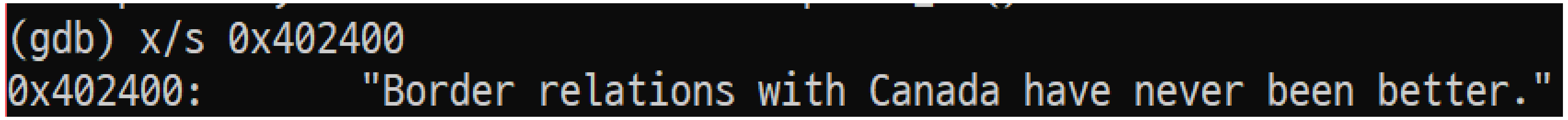
```
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    rsp,0x8
   0x0000000000400ee4 <+4>:     mov    esi,0x402400
   0x0000000000400ee9 <+9>:     call   0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   eax,eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    call   0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    rsp,0x8
   0x0000000000400efb <+27>:    ret
End of assembler dump.
```

인자로 들어온 두 문자열이 같음을 확인 -> 같으면 통과 -> 아니면 폭탄

BOMBLAB



```
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    rsp,0x8
   0x0000000000400ee4 <+4>:     mov    esi,0x402400
   0x0000000000400ee9 <+9>:     call   0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   eax,eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    call   0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    rsp,0x8
   0x0000000000400efb <+27>:    ret
End of assembler dump.
```

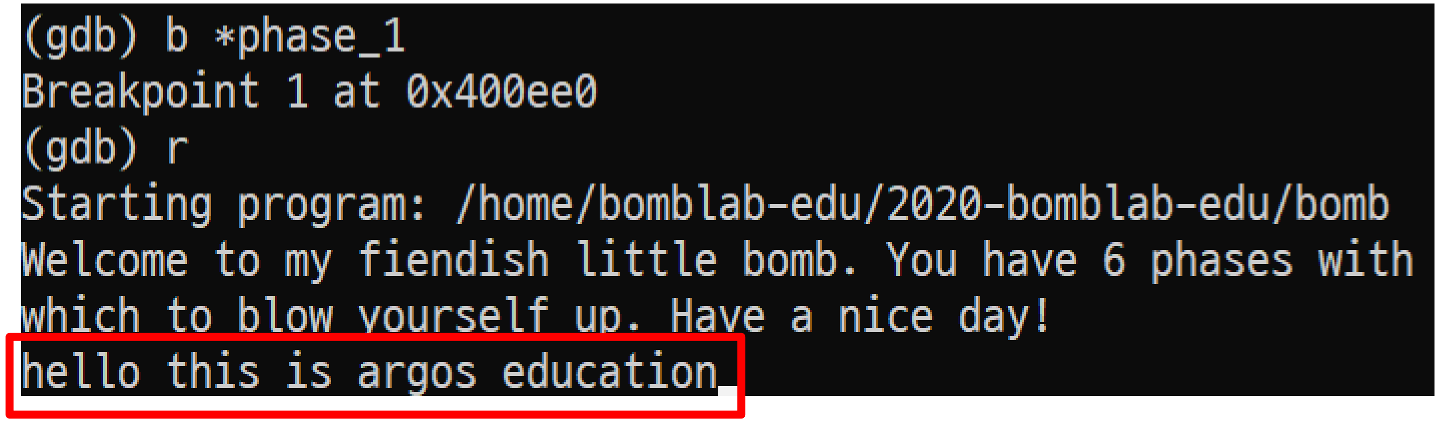


```
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
```

두번째 인자 획득, 첫번째 인자는?

밤랩 소개 및 1단계 해법

BOMBLAB



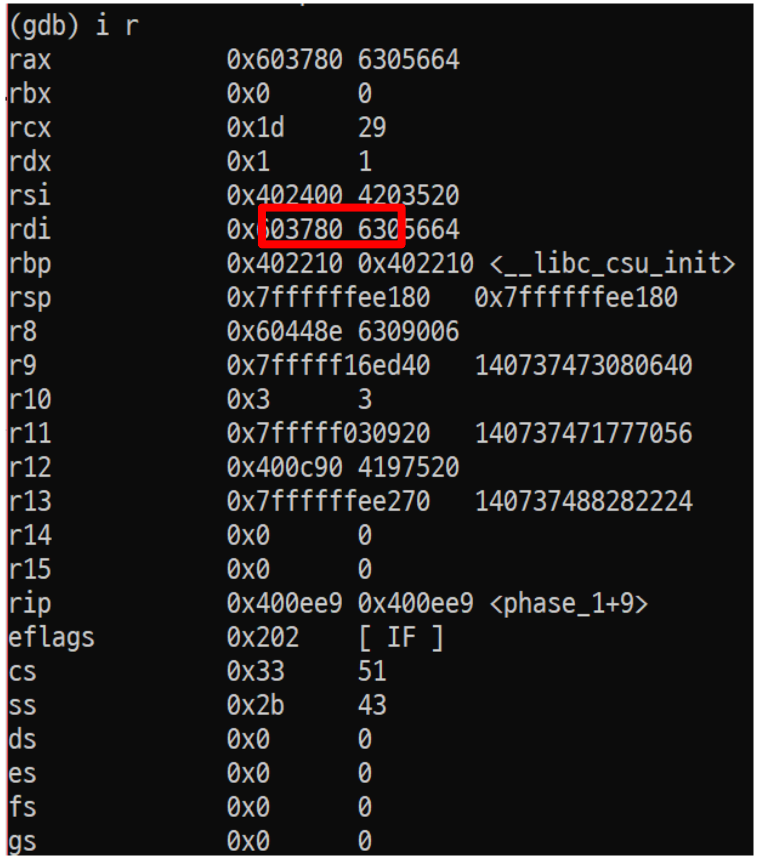
```
(gdb) b *phase_1
Breakpoint 1 at 0x400ee0
(gdb) r
Starting program: /home/bomblab-edu/2020-bomblab-edu/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
hello this is argos education
```

브레이크 포인트 설치 후(b *phase_1) -> 실행(r)

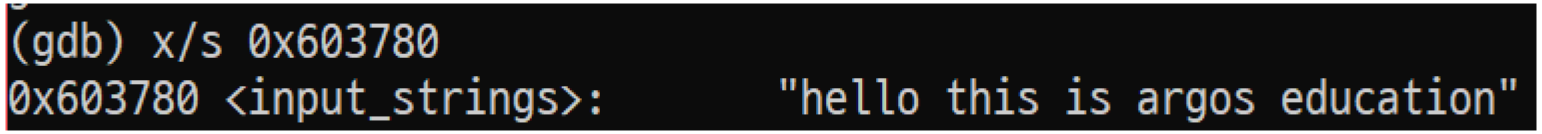
-> 우리가 알아볼 수 있는 문자열 입력

밤랩 소개 및 1단계 해법

BOMBLAB



```
(gdb) i r
rax            0x603780 6305664
rbx            0x0      0
rcx            0x1d     29
rdx            0x1      1
rsi            0x402400 4203520
rdi            0x603780 6305664
rbp            0x402210 0x402210 <__libc_csu_init>
rsp            0x7fffffffe180   0x7fffffffe180
r8             0x60448e 6309006
r9             0x7ffff16ed40    140737473080640
r10            0x3      3
r11            0x7ffff030920    140737471777056
r12            0x400c90 4197520
r13            0x7fffffffe270   140737488282224
r14            0x0      0
r15            0x0      0
rip            0x400ee9 0x400ee9 <phase_1+9>
eflags         0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
```



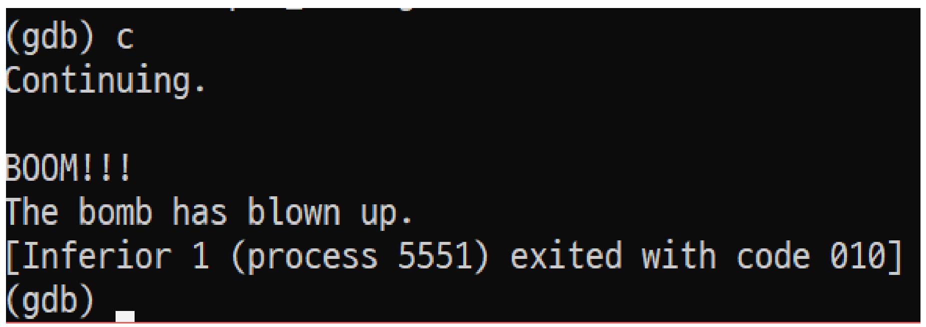
```
(gdb) x/s 0x603780
0x603780 <input_strings>:       "hello this is argos education"
```

첫번째 인자는 우리가 입력한 값...

즉, 우리가 입력한 값이 두번째 문자열과 같아야함

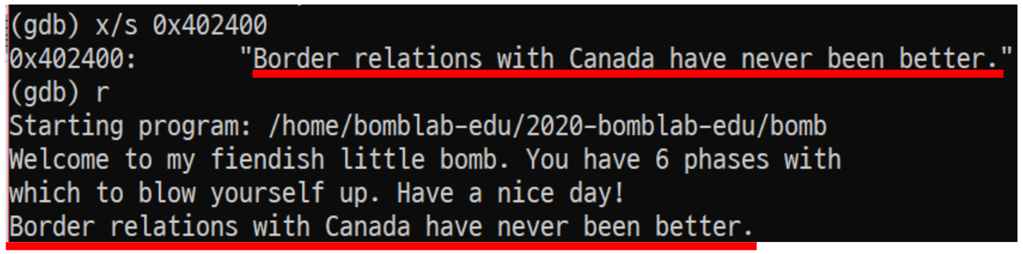
밤랩 소개 및 1단계 해법

BOMBLAB

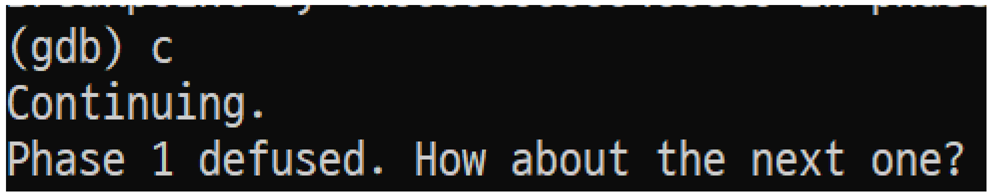


```
(gdb) c
Continuing.

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 5551) exited with code 010]
(gdb)
```

역시나 다른 문자열을 넣으면 폭탄 터짐



```
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
(gdb) r
Starting program: /home/bomblab-edu/2020-bomblab-edu/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
```

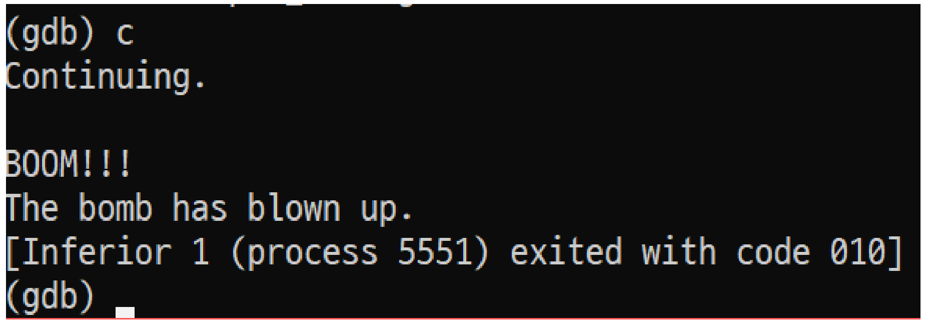


```
(gdb) c
Continuing.
Phase 1 defused. How about the next one?
```

1단계 해결

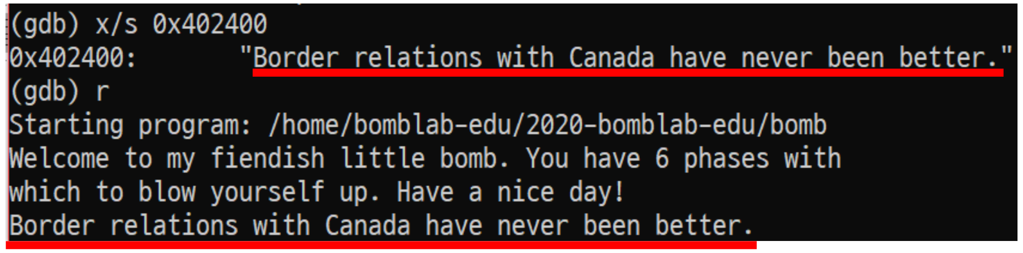
밤랩 소개 및 1단계 해법

BOMBLAB

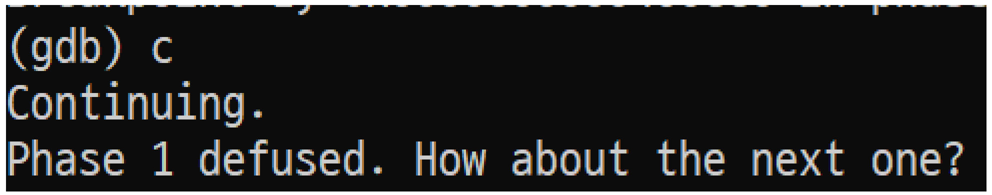


```
(gdb) c
Continuing.

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 5551) exited with code 010]
(gdb)
```

역시나 다른 문자열을 넣으면 폭탄 터짐



```
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
(gdb) r
Starting program: /home/bomblab-edu/2020-bomblab-edu/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
```

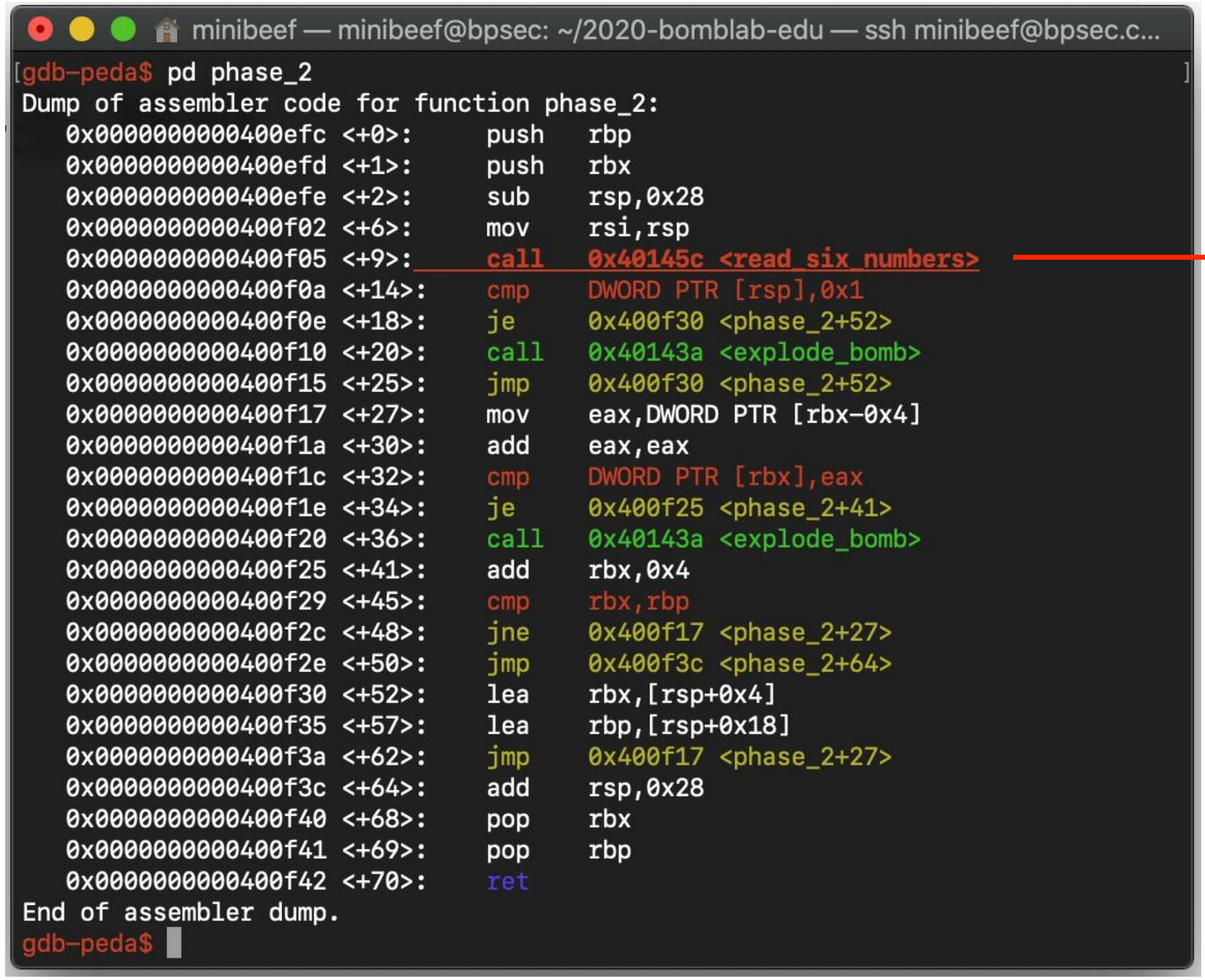


```
(gdb) c
Continuing.
Phase 1 defused. How about the next one?
```

1단계 해결

bomb lab phase 2

phase 2 구조 분석


minibeef — minibeef@bpsec: ~/2020-bomblab-edu — ssh minibeef@bpsec.c...

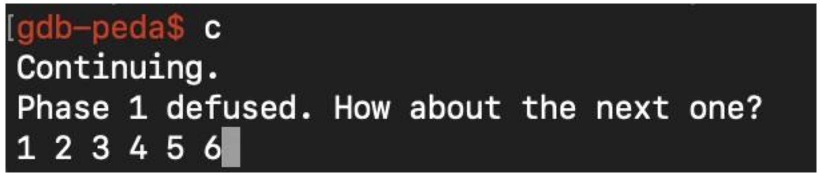
```
gdb-peda$ pd phase_2
Dump of assembler code for function phase_2:
   0x0000000000400efc <+0>:     push   rbp
   0x0000000000400efd <+1>:     push   rbx
   0x0000000000400efe <+2>:     sub    rsp,0x28
   0x0000000000400f02 <+6>:     mov    rsi,rsp
   0x0000000000400f05 <+9>:     call   0x40145c <read_six_numbers>
   0x0000000000400f0a <+14>:    cmp    DWORD PTR [rsp],0x1
   0x0000000000400f0e <+18>:    je     0x400f30 <phase_2+52>
   0x0000000000400f10 <+20>:    call   0x40143a <explode_bomb>
   0x0000000000400f15 <+25>:    jmp    0x400f30 <phase_2+52>
   0x0000000000400f17 <+27>:    mov    eax,DWORD PTR [rbx-0x4]
   0x0000000000400f1a <+30>:    add    eax,eax
   0x0000000000400f1c <+32>:    cmp    DWORD PTR [rbx],eax
   0x0000000000400f1e <+34>:    je     0x400f25 <phase_2+41>
   0x0000000000400f20 <+36>:    call   0x40143a <explode_bomb>
   0x0000000000400f25 <+41>:    add    rbx,0x4
   0x0000000000400f29 <+45>:    cmp    rbx,rbp
   0x0000000000400f2c <+48>:    jne    0x400f17 <phase_2+27>
   0x0000000000400f2e <+50>:    jmp    0x400f3c <phase_2+64>
   0x0000000000400f30 <+52>:    lea    rbx,[rsp+0x4]
   0x0000000000400f35 <+57>:    lea    rbp,[rsp+0x18]
   0x0000000000400f3a <+62>:    jmp    0x400f17 <phase_2+27>
   0x0000000000400f3c <+64>:    add    rsp,0x28
   0x0000000000400f40 <+68>:    pop    rbx
   0x0000000000400f41 <+69>:    pop    rbp
   0x0000000000400f42 <+70>:    ret
End of assembler dump.
gdb-peda$
```

숫자 6개를 입력 받는다

bomb lab phase 2

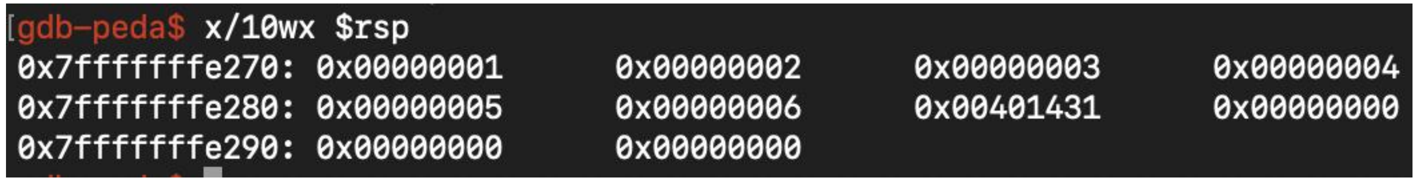
phase 2 구조 분석

1 2 3 4 5 6 입력



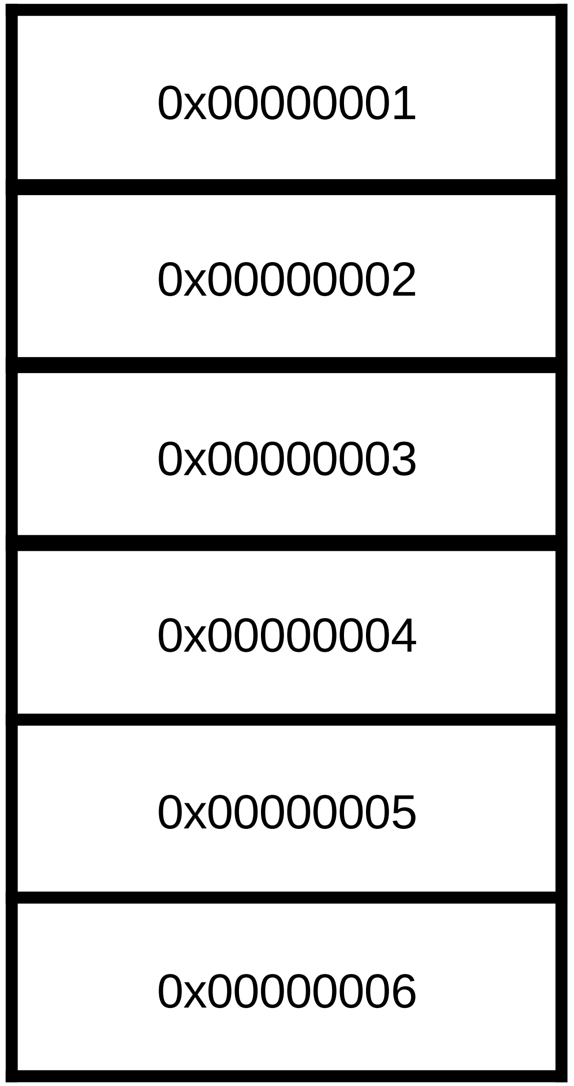
```
gdb-peda$ c
Continuing.
Phase 1 defused. How about the next one?
1 2 3 4 5 6
```

read_six_numbers() 이후 스택의 상태



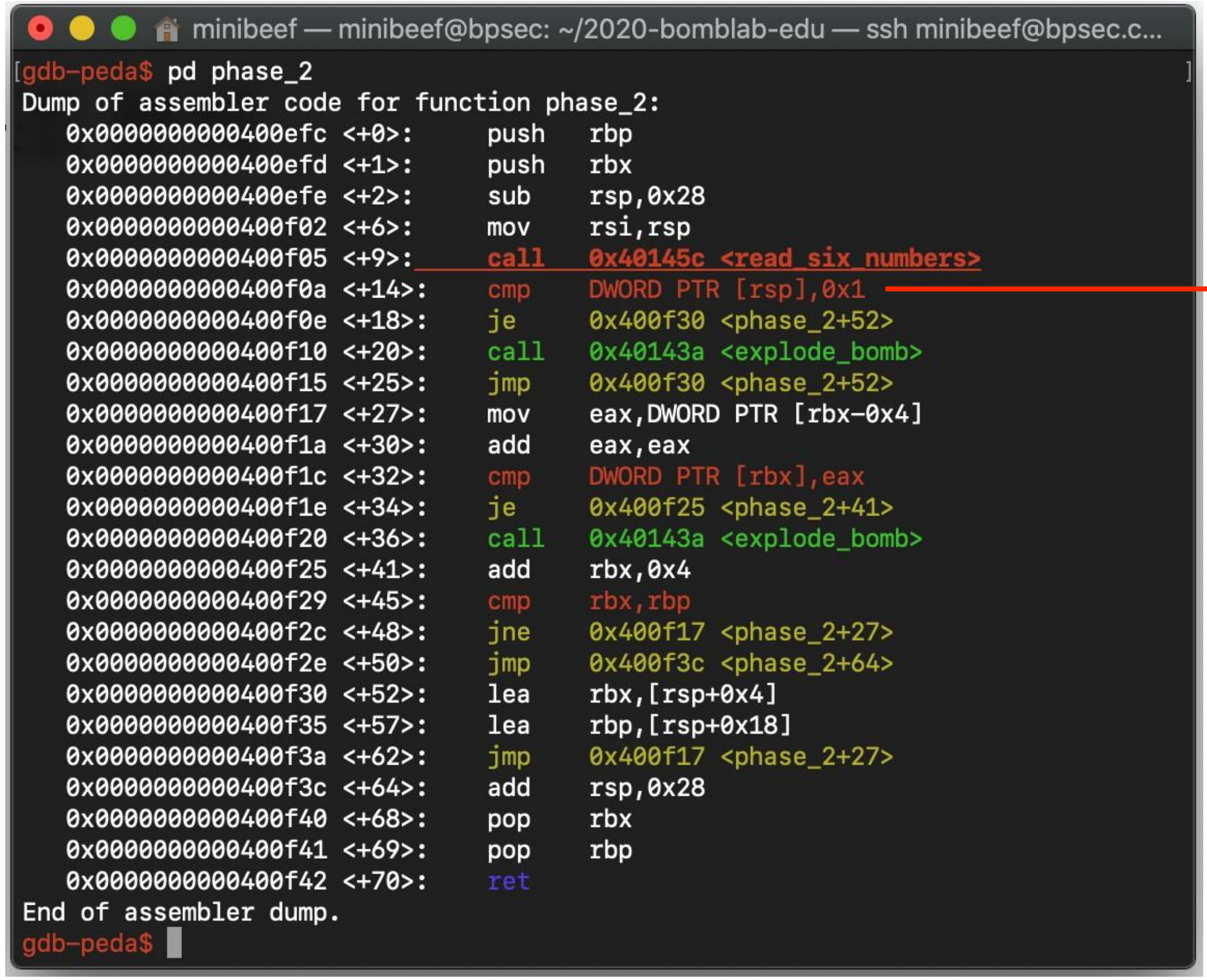
```
gdb-peda$ x/10wx $rsp
0x7fffffffe270: 0x00000001      0x00000002      0x00000003      0x00000004
0x7fffffffe280: 0x00000005      0x00000006      0x00401431      0x00000000
0x7fffffffe290: 0x00000000      0x00000000
```

Stack Pointer



bomb lab phase 2

phase 2 구조 분석



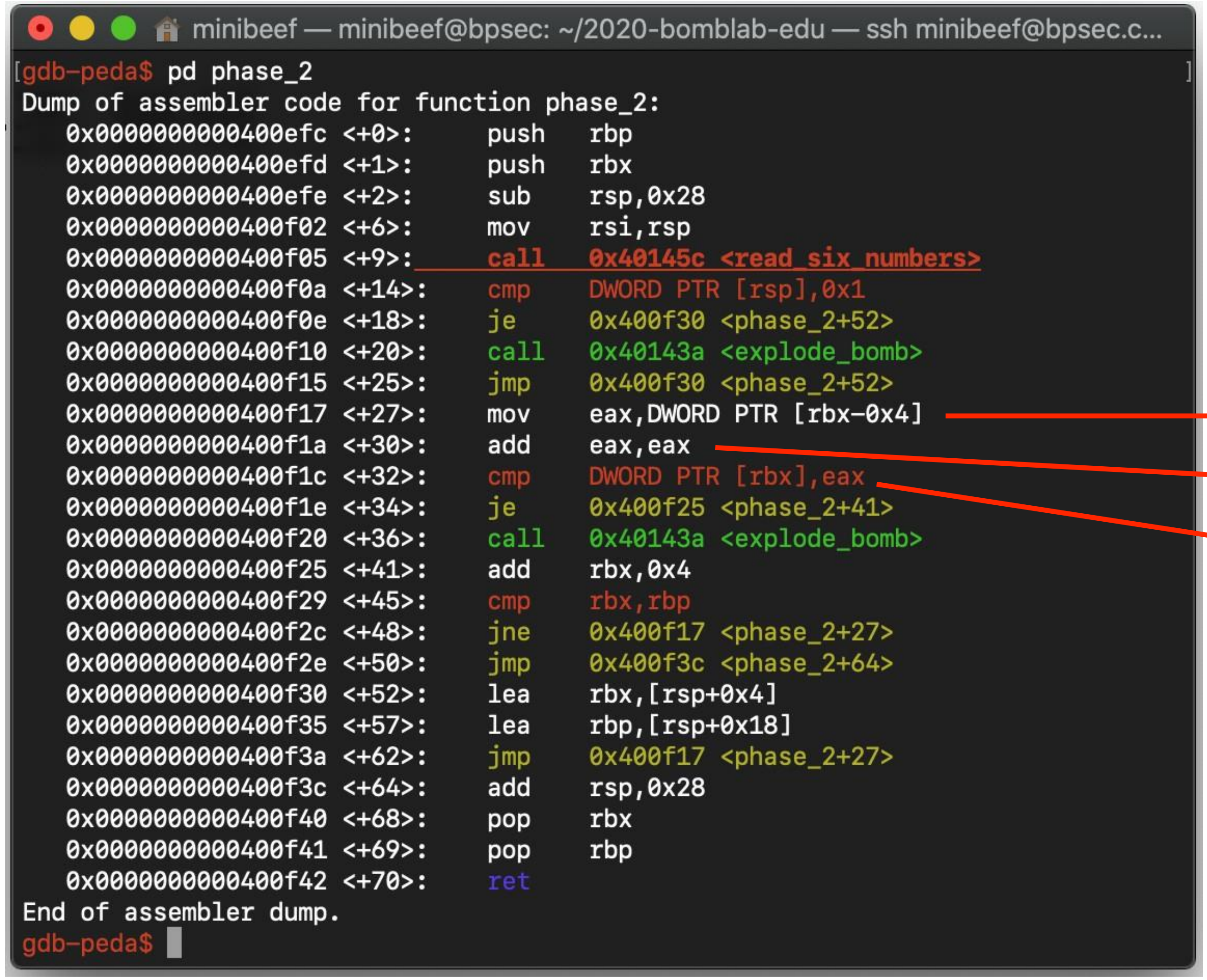
minibeef — minibeef@bpsec: ~/2020-bomblab-edu — ssh minibeef@bpsec.c...

```
gdb-peda$ pd phase_2
Dump of assembler code for function phase_2:
   0x0000000000400efc <+0>:     push   rbp
   0x0000000000400efd <+1>:     push   rbx
   0x0000000000400efe <+2>:     sub    rsp,0x28
   0x0000000000400f02 <+6>:     mov    rsi,rsp
   0x0000000000400f05 <+9>:     call   0x40145c <read_six_numbers>
   0x0000000000400f0a <+14>:    cmp    DWORD PTR [rsp],0x1
   0x0000000000400f0e <+18>:    je     0x400f30 <phase_2+52>
   0x0000000000400f10 <+20>:    call   0x40143a <explode_bomb>
   0x0000000000400f15 <+25>:    jmp    0x400f30 <phase_2+52>
   0x0000000000400f17 <+27>:    mov    eax,DWORD PTR [rbx-0x4]
   0x0000000000400f1a <+30>:    add    eax,eax
   0x0000000000400f1c <+32>:    cmp    DWORD PTR [rbx],eax
   0x0000000000400f1e <+34>:    je     0x400f25 <phase_2+41>
   0x0000000000400f20 <+36>:    call   0x40143a <explode_bomb>
   0x0000000000400f25 <+41>:    add    rbx,0x4
   0x0000000000400f29 <+45>:    cmp    rbx,rbp
   0x0000000000400f2c <+48>:    jne    0x400f17 <phase_2+27>
   0x0000000000400f2e <+50>:    jmp    0x400f3c <phase_2+64>
   0x0000000000400f30 <+52>:    lea    rbx,[rsp+0x4]
   0x0000000000400f35 <+57>:    lea    rbp,[rsp+0x18]
   0x0000000000400f3a <+62>:    jmp    0x400f17 <phase_2+27>
   0x0000000000400f3c <+64>:    add    rsp,0x28
   0x0000000000400f40 <+68>:    pop    rbx
   0x0000000000400f41 <+69>:    pop    rbp
   0x0000000000400f42 <+70>:    ret
End of assembler dump.
gdb-peda$
```

첫 번째 숫자는 1 이어야 함

bomb lab phase 2

phase 2 구조 분석



```
minibeef — minibeef@bpsec: ~/2020-bomblab-edu — ssh minibeef@bpsec.c...
gdb-peda$ pd phase_2
Dump of assembler code for function phase_2:
   0x0000000000400efc <+0>:     push   rbp
   0x0000000000400efd <+1>:     push   rbx
   0x0000000000400efe <+2>:     sub    rsp,0x28
   0x0000000000400f02 <+6>:     mov    rsi,rsp
   0x0000000000400f05 <+9>:     call   0x40145c <read_six_numbers>
   0x0000000000400f0a <+14>:    cmp    DWORD PTR [rsp],0x1
   0x0000000000400f0e <+18>:    je     0x400f30 <phase_2+52>
   0x0000000000400f10 <+20>:    call   0x40143a <explode_bomb>
   0x0000000000400f15 <+25>:    jmp    0x400f30 <phase_2+52>
   0x0000000000400f17 <+27>:    mov    eax,DWORD PTR [rbx-0x4]
   0x0000000000400f1a <+30>:    add    eax,eax
   0x0000000000400f1c <+32>:    cmp    DWORD PTR [rbx],eax
   0x0000000000400f1e <+34>:    je     0x400f25 <phase_2+41>
   0x0000000000400f20 <+36>:    call   0x40143a <explode_bomb>
   0x0000000000400f25 <+41>:    add    rbx,0x4
   0x0000000000400f29 <+45>:    cmp    rbx,rbp
   0x0000000000400f2c <+48>:    jne    0x400f17 <phase_2+27>
   0x0000000000400f2e <+50>:    jmp    0x400f3c <phase_2+64>
   0x0000000000400f30 <+52>:    lea    rbx,[rsp+0x4]
   0x0000000000400f35 <+57>:    lea    rbp,[rsp+0x18]
   0x0000000000400f3a <+62>:    jmp    0x400f17 <phase_2+27>
   0x0000000000400f3c <+64>:    add    rsp,0x28
   0x0000000000400f40 <+68>:    pop    rbx
   0x0000000000400f41 <+69>:    pop    rbp
   0x0000000000400f42 <+70>:    ret
End of assembler dump.
gdb-peda$
```

현재 가르키고 있는 숫자(1)을

2배한 값이

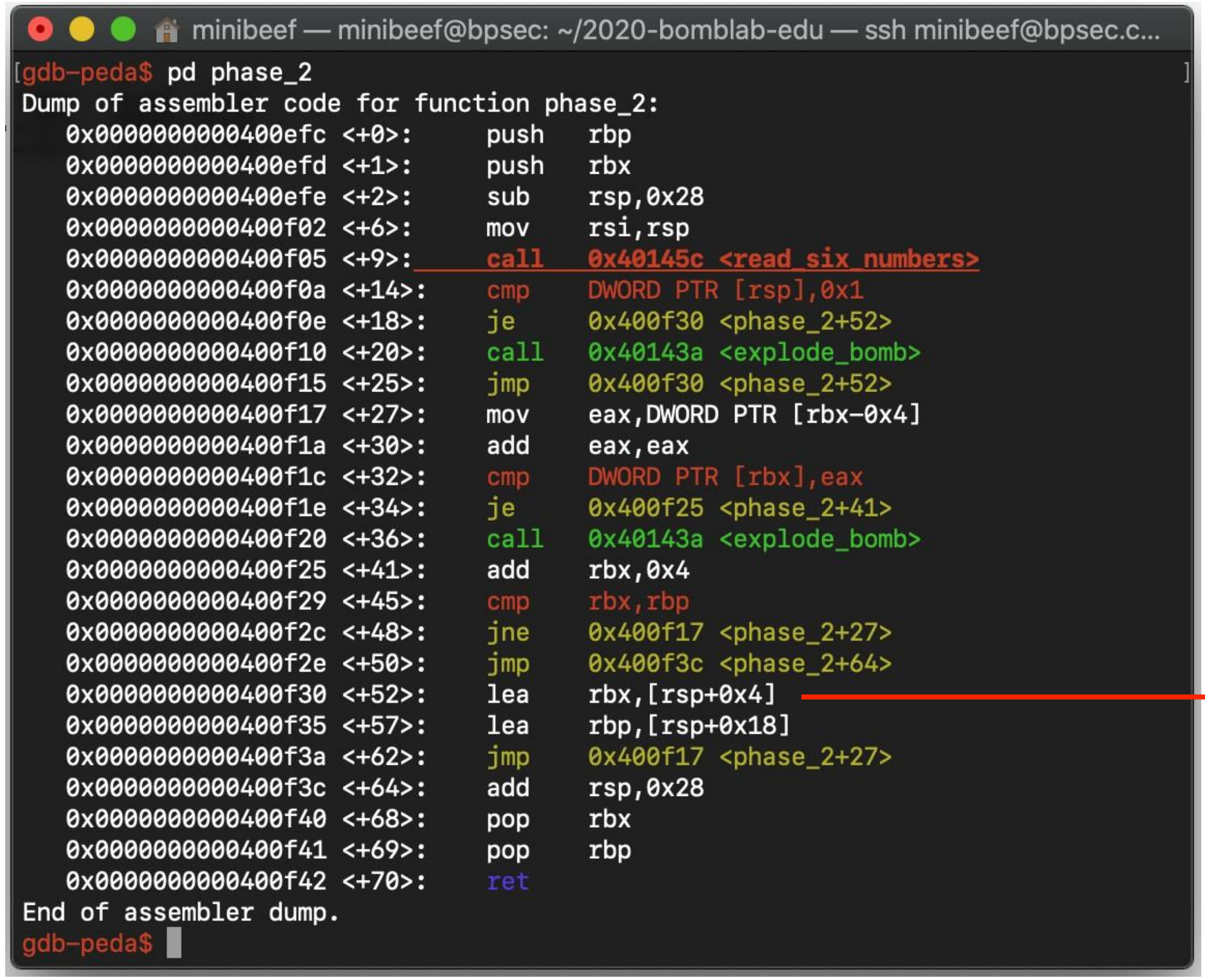
다음 숫자여야함

즉, 첫 숫자가 1 이었으므로

1 2 4 8 16 32

bomb lab phase 2

phase 2 구조 분석

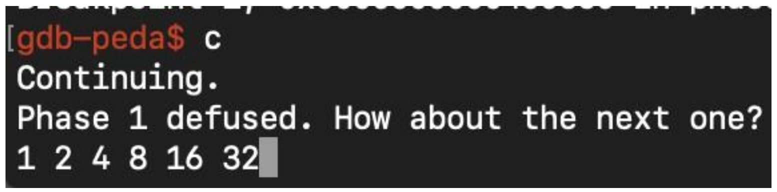

minibeef — minibeef@bpsec: ~/2020-bomblab-edu — ssh minibeef@bpsec.c...

```
gdb-peda$ pd phase_2
Dump of assembler code for function phase_2:
   0x0000000000400efc <+0>:     push   rbp
   0x0000000000400efd <+1>:     push   rbx
   0x0000000000400efe <+2>:     sub    rsp,0x28
   0x0000000000400f02 <+6>:     mov    rsi,rsp
   0x0000000000400f05 <+9>:     call   0x40145c <read_six_numbers>
   0x0000000000400f0a <+14>:    cmp    DWORD PTR [rsp],0x1
   0x0000000000400f0e <+18>:    je     0x400f30 <phase_2+52>
   0x0000000000400f10 <+20>:    call   0x40143a <explode_bomb>
   0x0000000000400f15 <+25>:    jmp    0x400f30 <phase_2+52>
   0x0000000000400f17 <+27>:    mov    eax,DWORD PTR [rbx-0x4]
   0x0000000000400f1a <+30>:    add    eax,eax
   0x0000000000400f1c <+32>:    cmp    DWORD PTR [rbx],eax
   0x0000000000400f1e <+34>:    je     0x400f25 <phase_2+41>
   0x0000000000400f20 <+36>:    call   0x40143a <explode_bomb>
   0x0000000000400f25 <+41>:    add    rbx,0x4
   0x0000000000400f29 <+45>:    cmp    rbx,rbp
   0x0000000000400f2c <+48>:    jne    0x400f17 <phase_2+27>
   0x0000000000400f2e <+50>:    jmp    0x400f3c <phase_2+64>
   0x0000000000400f30 <+52>:    lea    rbx,[rsp+0x4]
   0x0000000000400f35 <+57>:    lea    rbp,[rsp+0x18]
   0x0000000000400f3a <+62>:    jmp    0x400f17 <phase_2+27>
   0x0000000000400f3c <+64>:    add    rsp,0x28
   0x0000000000400f40 <+68>:    pop    rbx
   0x0000000000400f41 <+69>:    pop    rbp
   0x0000000000400f42 <+70>:    ret
End of assembler dump.
gdb-peda$
```

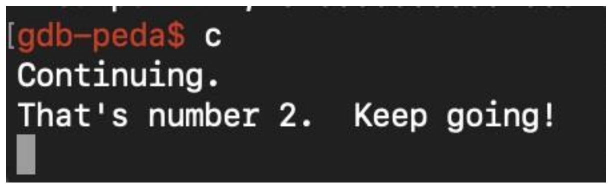
추가로… 해당 구문을 통해 rbx가 가르키는 값을 한 칸(4 바이트) 이동

bomb lab phase 2

phase 2 구조 분석



```
[gdb-peda$ c
Continuing.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
```



```
[gdb-peda$ c
Continuing.
That's number 2.  Keep going!
```

IDA

Hex-Ray?

Assembly를 C로 변환하여 보여줄 수 있는 기술

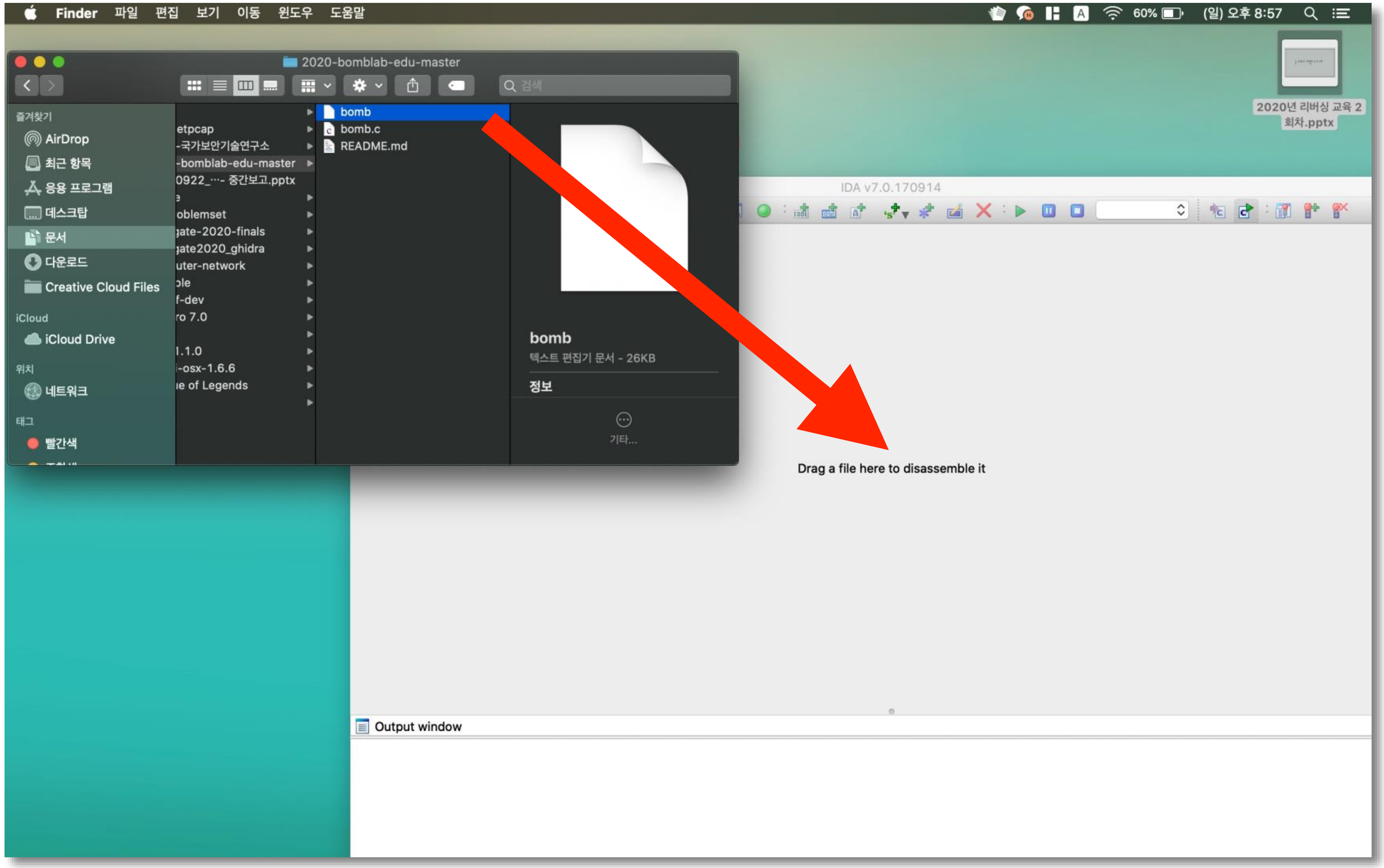
IDA Hex-Ray

IDA

ida64 오픈

IDA Hex-Ray

IDA

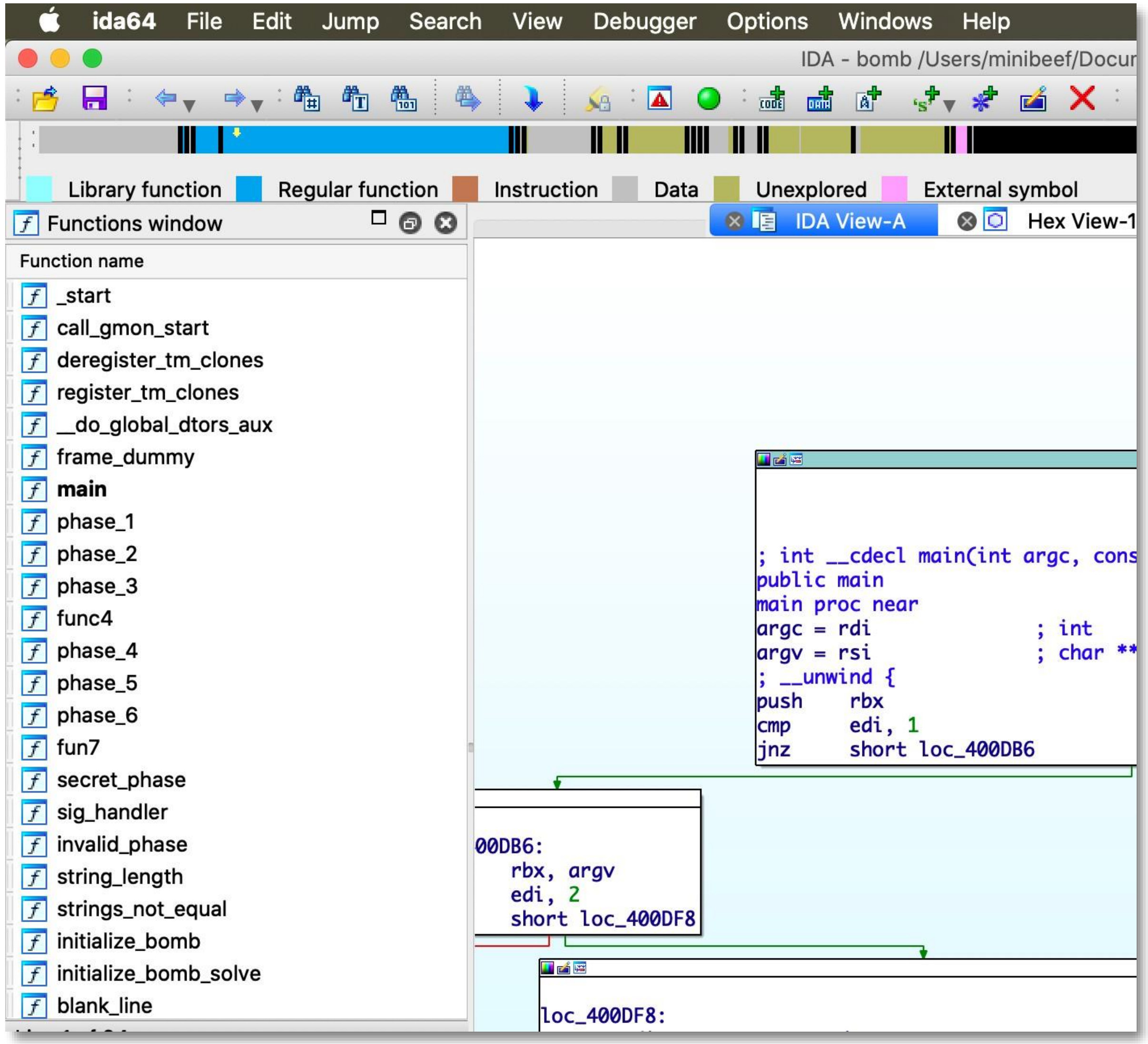


Drag a file here to disassemble it

IDA Hex-Ray

IDA

좌측 함수 리스트에서 함수 선택



```
_start
call_gmon_start
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
main
phase_1
phase_2
phase_3
func4
phase_4
phase_5
phase_6
fun7
secret_phase
sig_handler
invalid_phase
string_length
strings_not_equal
initialize_bomb
initialize_bomb_solve
blank_line
```

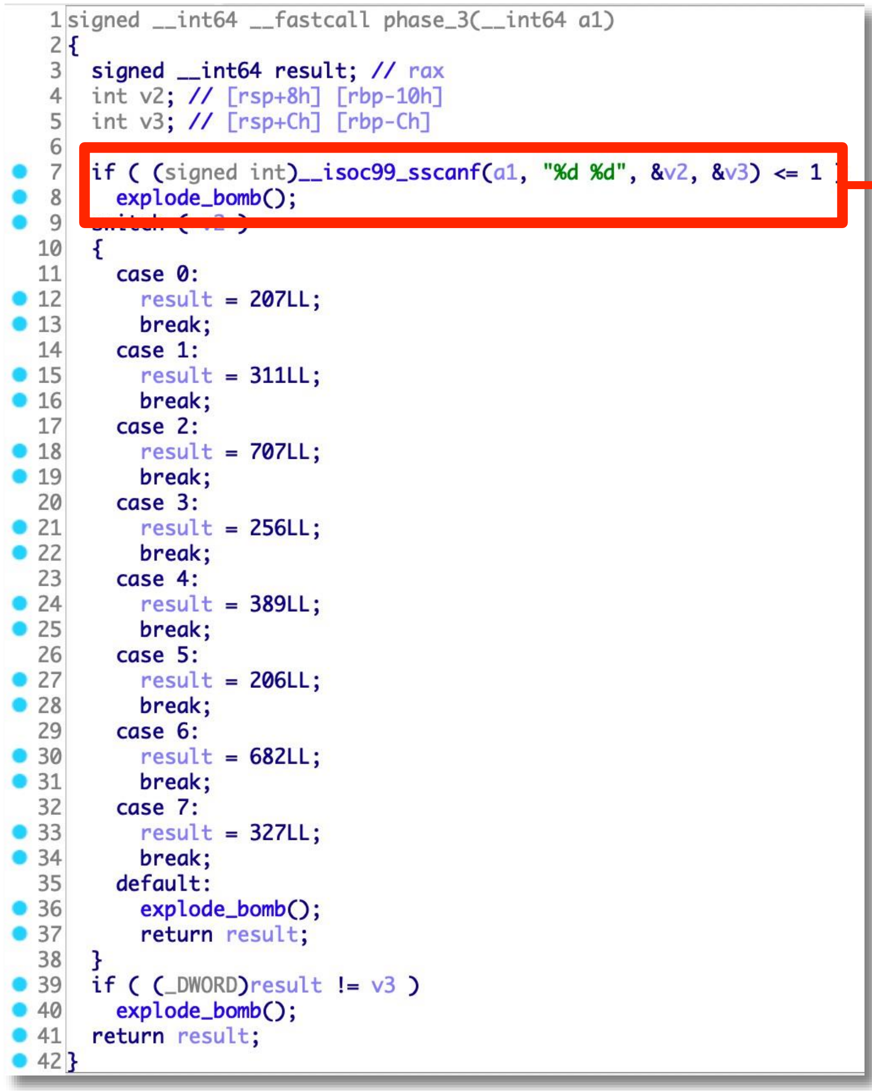
```
; int __cdecl main(int argc, cons
public main
main proc near
argc = rdi                    ; int
argv = rsi                    ; char **
; __unwind {
push    rbx
cmp     edi, 1
jnz     short loc_400DB6
```

```
00DB6:
    rbx, argv
    edi, 2
    short loc_400DF8
```

```
loc_400DF8:
```

IDA Hex-Ray

phase 3 with IDA

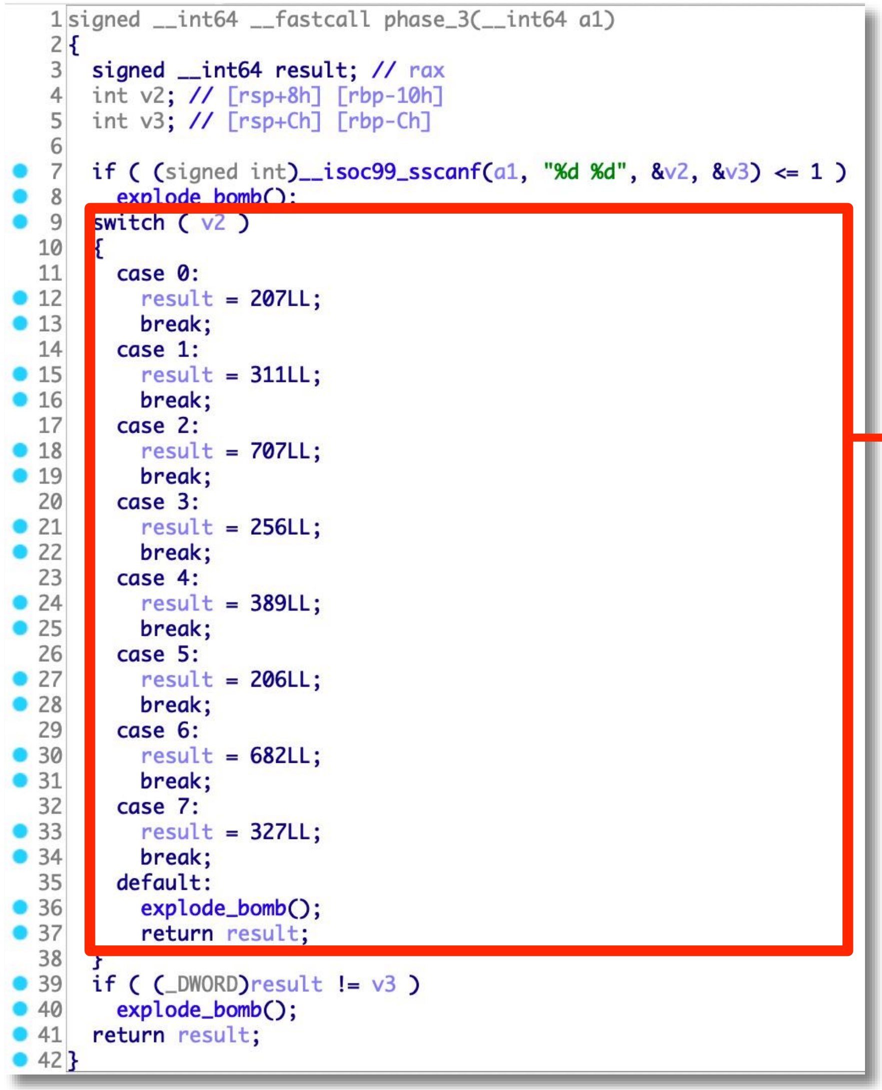


```
signed __int64 __fastcall phase_3(__int64 a1)
{
  signed __int64 result; // rax
  int v2; // [rsp+8h] [rbp-10h]
  int v3; // [rsp+Ch] [rbp-Ch]

  if ( (signed int)__isoc99_sscanf(a1, "%d %d", &v2, &v3) <= 1 )
    explode_bomb();
  switch ( v2 )
  {
    case 0:
      result = 207LL;
      break;
    case 1:
      result = 311LL;
      break;
    case 2:
      result = 707LL;
      break;
    case 3:
      result = 256LL;
      break;
    case 4:
      result = 389LL;
      break;
    case 5:
      result = 206LL;
      break;
    case 6:
      result = 682LL;
      break;
    case 7:
      result = 327LL;
      break;
    default:
      explode_bomb();
      return result;
  }
  if ( (_DWORD)result != v3 )
    explode_bomb();
  return result;
}
```

숫자 두개 입력 -> v2, v3

IDA Hex-Ray

phase 3 with IDA

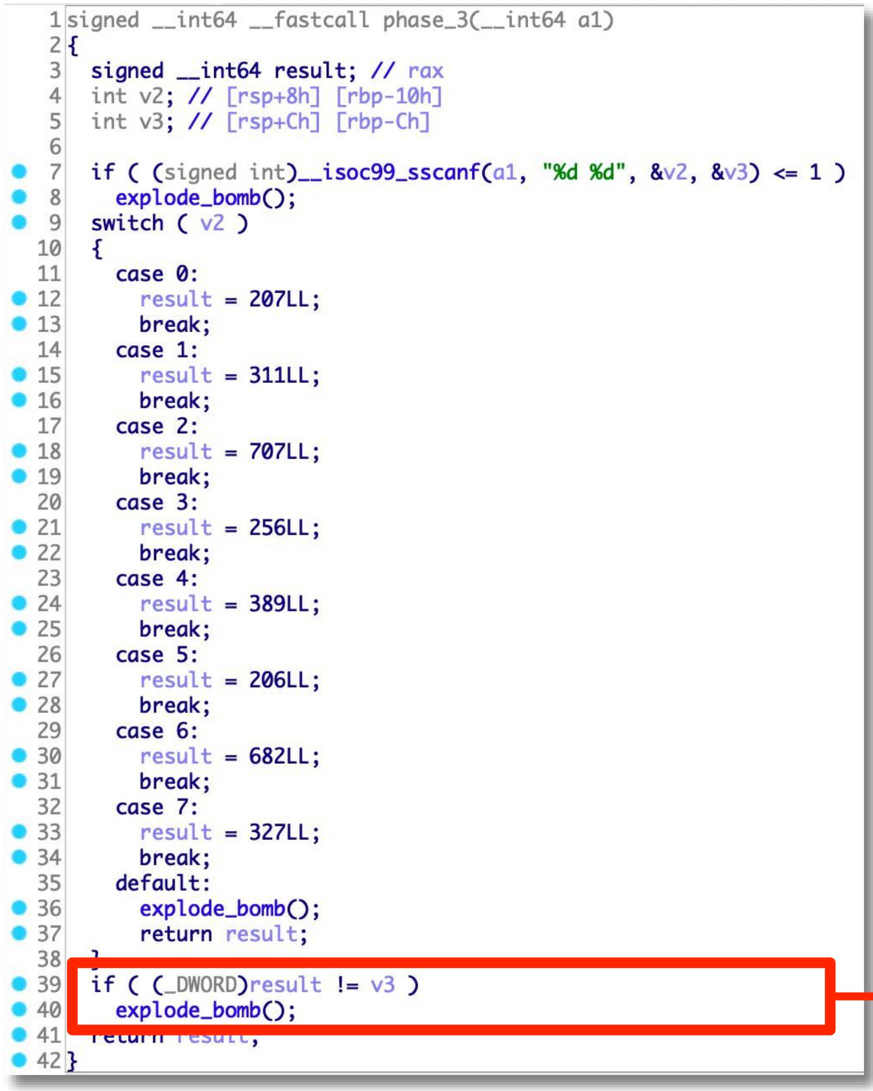


```
signed __int64 __fastcall phase_3(__int64 a1)
{
  signed __int64 result; // rax
  int v2; // [rsp+8h] [rbp-10h]
  int v3; // [rsp+Ch] [rbp-Ch]

  if ( (signed int)__isoc99_sscanf(a1, "%d %d", &v2, &v3) <= 1 )
    explode_bomb();
  switch ( v2 )
  {
    case 0:
      result = 207LL;
      break;
    case 1:
      result = 311LL;
      break;
    case 2:
      result = 707LL;
      break;
    case 3:
      result = 256LL;
      break;
    case 4:
      result = 389LL;
      break;
    case 5:
      result = 206LL;
      break;
    case 6:
      result = 682LL;
      break;
    case 7:
      result = 327LL;
      break;
    default:
      explode_bomb();
      return result;
  }
  if ( (_DWORD)result != v3 )
    explode_bomb();
  return result;
}
```

1) v2는 0~7 사이 숫자여야 함 -> 아니면 폭발
2) v2 값에 따른 result 변수가 존재

IDA Hex-Ray

phase 3 with IDA

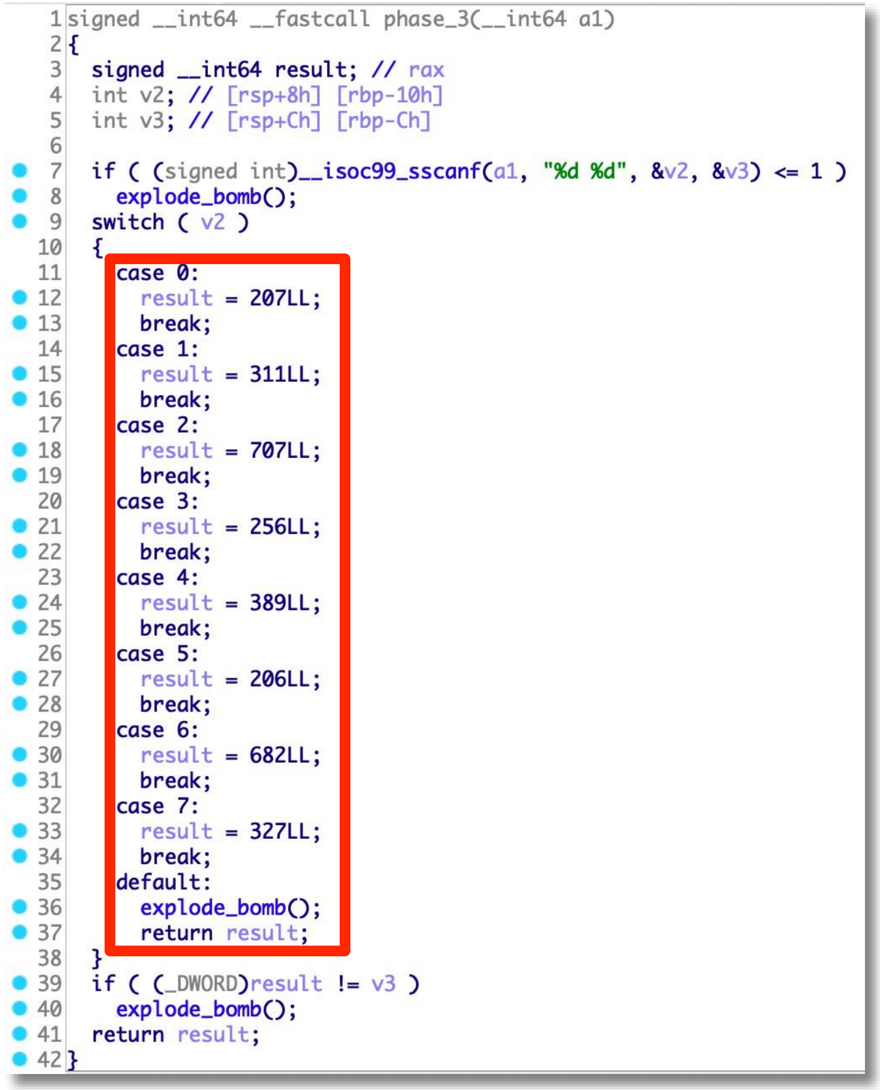


```
signed __int64 __fastcall phase_3(__int64 a1)
{
  signed __int64 result; // rax
  int v2; // [rsp+8h] [rbp-10h]
  int v3; // [rsp+Ch] [rbp-Ch]

  if ( (signed int)__isoc99_sscanf(a1, "%d %d", &v2, &v3) <= 1 )
    explode_bomb();
  switch ( v2 )
  {
    case 0:
      result = 207LL;
      break;
    case 1:
      result = 311LL;
      break;
    case 2:
      result = 707LL;
      break;
    case 3:
      result = 256LL;
      break;
    case 4:
      result = 389LL;
      break;
    case 5:
      result = 206LL;
      break;
    case 6:
      result = 682LL;
      break;
    case 7:
      result = 327LL;
      break;
    default:
      explode_bomb();
      return result;
  }
  if ( (_DWORD)result != v3 )
    explode_bomb();
  return result;
}
```

두번 째 입력한 숫자가 result와 같아야함

IDA Hex-Ray

phase 3 with IDA



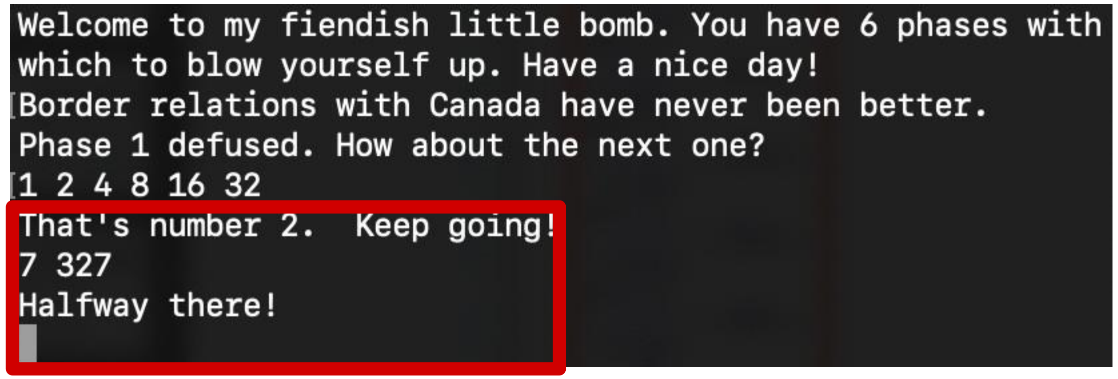
```
signed __int64 __fastcall phase_3(__int64 a1)
{
  signed __int64 result; // rax
  int v2; // [rsp+8h] [rbp-10h]
  int v3; // [rsp+Ch] [rbp-Ch]

  if ( (signed int)__isoc99_sscanf(a1, "%d %d", &v2, &v3) <= 1 )
    explode_bomb();
  switch ( v2 )
  {
    case 0:
      result = 207LL;
      break;
    case 1:
      result = 311LL;
      break;
    case 2:
      result = 707LL;
      break;
    case 3:
      result = 256LL;
      break;
    case 4:
      result = 389LL;
      break;
    case 5:
      result = 206LL;
      break;
    case 6:
      result = 682LL;
      break;
    case 7:
      result = 327LL;
      break;
    default:
      explode_bomb();
      return result;
  }
  if ( (_DWORD)result != v3 )
    explode_bomb();
  return result;
}
```

즉 다음과 같은 숫자 쌍 중에 하나를 입력해야 함

0 207
1 311
2 707
3 256
4 389
5 206
6 682
7 327

IDA Hex-Ray

phase 3 with IDA



```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
7 327
Halfway there!
```

임의로 숫자를 골라서 입력 했을 때 -> 정답

끝