

## I. 소개

Bomb를 disassemble해서 해체하는 LAB이다. 총 6개의 phase가 있으며 explode\_bomb를 피해서 올바른 answer을 모두 입력하면 해체할 수 있다. 리눅스 서버에 접속 > chmod+x bomb 입력 > bomb bomb.c readme 끌어다놓기 > gdb bomb > disas phase\_x 입력해서 어셈블리어를 분석해 올바른 답을 찾아갈 수 있다.

## II. Solution 설명

## (1) Phase\_1

**Solution : You can Russia from land here in Alaska.**

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
0x0000000000001204 <+0>:    sub    $0x8,%rsp
0x0000000000001208 <+4>:    lea    0x16a1(%rip),%rsi        # 0x28b0
0x000000000000120f <+11>:   callq 0x17a5 <strings_not_equal>
0x0000000000001214 <+16>:   test   %eax,%eax
0x0000000000001216 <+18>:   jne    0x121d <phase_1+25>
0x0000000000001218 <+20>:   add    $0x8,%rsp
0x000000000000121c <+24>:   retq
0x000000000000121d <+25>:   callq 0x18b1 <explode_bomb>
0x0000000000001222 <+30>:   jmp    0x1218 <phase_1+20>
End of assembler dump.
(gdb) quit
```

그림 1. disas phase\_1

```
(gdb) disas strings_not_equal
Dump of assembler code for function strings_not_equal:
0x00000000000017a5 <+0>:    push   %r12
0x00000000000017a7 <+2>:    push   %rbp
0x00000000000017a8 <+3>:    push   %rbx
0x00000000000017a9 <+4>:    mov    %rdi,%rbx
0x00000000000017ac <+7>:    mov    %rsi,%rbp
0x00000000000017af <+10>:   callq 0x1788 <string_length>
0x00000000000017b4 <+15>:   mov    %eax,%r12d
0x00000000000017b7 <+18>:   mov    %rbp,%rdi
0x00000000000017ba <+21>:   callq 0x1788 <string_length>
0x00000000000017bf <+26>:   mov    %0x1,%edx
0x00000000000017c4 <+31>:   cmp    %eax,%r12d
0x00000000000017c7 <+34>:   je     0x17d0 <strings_not_equal+43>
0x00000000000017c9 <+36>:   mov    %edx,%eax
0x00000000000017cb <+38>:   pop    %rbx
0x00000000000017cc <+39>:   pop    %rbp
0x00000000000017cd <+40>:   pop    %r12
0x00000000000017cf <+42>:   retq
0x00000000000017d0 <+43>:   movzbl (%rbx),%eax
0x00000000000017d3 <+46>:   test   %al,%al
0x00000000000017d5 <+48>:   je     0x17fe <strings_not_equal+89>
0x00000000000017d7 <+50>:   cmp    0x0(%rbp),%al
0x00000000000017da <+53>:   jne    0x1805 <strings_not_equal+96>
0x00000000000017dc <+55>:   add    %0x1,%rbx
0x00000000000017de <+57>:   add    %0x1,%rbp
0x00000000000017e0 <+59>:   movzbl (%rbx),%eax
0x00000000000017e4 <+63>:   test   %al,%al
0x00000000000017e7 <+66>:   je     0x17f7 <strings_not_equal+82>
0x00000000000017e9 <+68>:   cmp    %al,0x0(%rbp)
0x00000000000017eb <+70>:   je     0x17dc <strings_not_equal+55>
0x00000000000017ee <+72>:   mov    %0x1,%edx
0x00000000000017f0 <+75>:   jmp    0x17c9 <strings_not_equal+36>
0x00000000000017f5 <+80>:   jmp    0x17c9 <strings_not_equal+36>
0x00000000000017f7 <+82>:   mov    %0x0,%edx
0x00000000000017fc <+87>:   jmp    0x17c9 <strings_not_equal+36>
0x00000000000017fe <+89>:   mov    %0x0,%edx
0x0000000000001803 <+94>:   jmp    0x17c9 <strings_not_equal+36>
0x0000000000001805 <+96>:   mov    %0x1,%edx
0x000000000000180a <+101>:  jmp    0x17c9 <strings_not_equal+36>
---Type <return> to continue, or q <return> to quit---
End of assembler dump.
(gdb)
```

그림 2. disas strings\_not\_equal

우선 입력했을 때 오답이어도 폭탄이 바로 터지지 않도록 b phase\_1을 통해 breakpoint를 설정해주었다. Disas phase\_1을 보면 strings\_not\_equal이라는 함수를 부르는데, 여기의 리턴값을 rax가 담게되고, test rax rax를 통해서 and연산을 통해 이 값이 1이면 1&1=1, 0이면 0&0=0해서 ZF를 세팅한다. 이를 이용해 Jne에서 rax값이 1이 아니라면 +25로 점프해서 폭탄이 터지는 것을 알 수 있다. 따라서 strings\_not\_equal 함수에서의 return값이 1이어야 함을 알 수 있다. 이 strings\_not\_equal 함수의 어셈블리 코드를 보면 rdi, rsi 두가지 매개변수를 호출받는 것을 알 수 있는데, x/s 레지스터를 실행시켜보면 rdi에는 내가 입력한 문자열, rsi에는 다른 문자열이 들어가있다. 함수 이름에서도 알 수 있듯이 두 문자열이 같으면 1을 반환하는 함수라고 생각해서 입력값을 rsi와 같이 'You can Russia from land here in Alaska.' 라고 두어 실행했더니 첫번째 phase를 해결할 수 있었다.

```
End of assembler dump.
(gdb) x/s $rdi
0x5555557586a0 <input_strings>: "string"
(gdb) x/s $rsi
0x5555555568b0: "You can Russia from land here in Alaska."
(gdb) █
```

그림 3. x/s \$rdi, \$rsi

(2) Phase\_2

Solution : 0 1 1 2 3 5

```
(gdb) disas phase_2
Dump of assembler code for function phase_2:
0x000055555555224 <+0>:    push    %rbp
0x000055555555225 <+1>:    push    %rbx
0x000055555555226 <+2>:    sub     $0x28,%rsp
0x00005555555522a <+6>:    mov     %fs:0x28,%rax
0x000055555555233 <+15>:   mov     %rax,0x18(%rsp)
0x000055555555238 <+20>:   xor     %eax,%eax
0x00005555555523a <+22>:   mov     %rsp,%rsi
0x00005555555523d <+25>:   callq   0x5555555558d7 <read_six_numbers>
0x000055555555242 <+30>:   cmpl    $0x0,(%rsp)
0x000055555555246 <+34>:   jne     0x55555555524f <phase_2+43>
0x000055555555248 <+36>:   cmpl    $0x1,0x4(%rsp)
0x00005555555524d <+41>:   je      0x555555555254 <phase_2+48>
0x00005555555524f <+43>:   callq   0x5555555558b1 <explode_bomb>
0x000055555555254 <+48>:   mov     %rsp,%rbx
0x000055555555257 <+51>:   lea     0x10(%rbx),%rbp
0x00005555555525b <+55>:   jmp     0x555555555266 <phase_2+66>
0x00005555555525d <+57>:   add     $0x4,%rbx
0x000055555555261 <+61>:   cmp     %rbp,%rbx
0x000055555555264 <+64>:   je      0x555555555277 <phase_2+83>
0x000055555555266 <+66>:   mov     0x4(%rbx),%eax
0x000055555555269 <+69>:   add     (%rbx),%eax
0x00005555555526b <+71>:   cmp     %eax,0x8(%rbx)
0x00005555555526e <+74>:   je      0x55555555525d <phase_2+57>
0x000055555555270 <+76>:   callq   0x5555555558b1 <explode_bomb>
0x000055555555275 <+81>:   jmp     0x55555555525d <phase_2+57>
0x000055555555277 <+83>:   mov     0x18(%rsp),%rax
0x00005555555527c <+88>:   xor     %fs:0x28,%rax
0x000055555555285 <+97>:   jne     0x55555555528e <phase_2+106>
0x000055555555287 <+99>:   add     $0x28,%rsp
0x00005555555528b <+103>:  pop     %rbx
0x00005555555528c <+104>:  pop     %rbp
0x00005555555528d <+105>:  retq
0x00005555555528e <+106>:  callq   0x5555555554e50 <__stack_chk_fail@plt>
End of assembler dump.
```

그림 4. disas phase\_2

```

(gdb) disas read_six_numbers
Dump of assembler code for function read_six_numbers:
0x00005555555558d7 <+0>:  sub    $0x8,%rsp
0x00005555555558db <+4>:  mov     %rsi,%rdx
0x00005555555558de <+7>:  lea     0x4(%rsi),%rcx
0x00005555555558e2 <+11>: lea     0x14(%rsi),%rax
0x00005555555558e6 <+15>:  push    %rax
0x00005555555558e7 <+16>:  lea     0x10(%rsi),%rax
0x00005555555558eb <+20>:  push    %rax
0x00005555555558ec <+21>:  lea     0xc(%rsi),%r9
0x00005555555558f0 <+25>:  lea     0x8(%rsi),%r8
0x00005555555558f4 <+29>:  lea     0x1188(%rip),%rsi    # 0x555555556a83
0x00005555555558fb <+36>:  mov     $0x0,%eax
0x0000555555555900 <+41>:  callq   0x555555554ef0 <__isoc99_sscanf@plt>
0x0000555555555905 <+46>:  add     $0x10,%rsp
0x0000555555555909 <+50>:  cmp     $0x5,%eax
0x000055555555590c <+53>:  jle     0x555555555913 <read_six_numbers+60>
0x000055555555590e <+55>:  add     $0x8,%rsp
0x0000555555555912 <+59>:  retq
0x0000555555555913 <+60>:  callq   0x5555555558b1 <explode_bomb>
End of assembler dump.
(gdb)

```

그림 5. disas read\_six\_numbers

우선 phase\_2에 break를 걸고, disas phase\_2를 해보면 read\_six\_numbers라는 함수가 있는 것을 보아 숫자 6개를 입력받는다라는 것을 알 수 있고, disas read\_six\_numbers를 보았을 때, 입력받은 값이 어떻게 표현되는지 담은 레지스터를 찾기위해 각각의 레지스터들은 x/s해보았고, x/s rsi 레지스터에서 %d %d %d %d %d %d를 담고 있음을 알게 되었다.

```

=> 0x00005555555558f4 <+29>:  lea     0x1188(%rip),%rsi    # 0x555555556a83
0x00005555555558fb <+36>:  mov     $0x0,%eax
0x0000555555555900 <+41>:  callq   0x555555554ef0 <__isoc99_sscanf@plt>
0x0000555555555905 <+46>:  add     $0x10,%rsp
0x0000555555555909 <+50>:  cmp     $0x5,%eax
0x000055555555590c <+53>:  jle     0x555555555913 <read_six_numbers+60>
0x000055555555590e <+55>:  add     $0x8,%rsp
0x0000555555555912 <+59>:  retq
0x0000555555555913 <+60>:  callq   0x5555555558b1 <explode_bomb>
End of assembler dump.
(gdb) si
0x00005555555558fb in read_six_numbers ()
(gdb) x/s $rsi
0x555555556a83: "%d %d %d %d %d %d"

```

그림 6. %d %d %d %d %d %d (입력값 확인)

다시 phase\_2로 돌아와 확인해보면 rsp와 0을 비교해서 같지 않으면 bomb -> 즉 rsp값은 0이어야 하는데, rsp값이 뭔지 확인해보면 (x/d \$rsp) 1이 들어가있다. (입력을 1 2 3 4 5 6 으로 준 상태) 따라서 아무래도 rsp에는 첫번째 입력값이 들어가있다는 것을 예상할 수 있고, 0x4(rsp) 즉, rsp+4 주소에는 integer이기에 두번째 입력값이 들어가있는 것을 알 수 있고, 이는 1이어야 bomb하지 않는다. 즉 첫번째 두번째 값은 0, 1이어야 한다. 그 뒤의 코드를 살펴보면 반복문을 8 바이트를 건너 뛰었을 때, 본래값과 4바이트를 건너뛰었을 때의 값의 add값과 같다는 사실을 알 수 있다. 즉, a1, a2, a3, a4, a5, a6가 입력이어야한다고 보았을 때 a3 = a2 + a1의 형태를 지닌다는 것을 알 수 있다. 따라서 값은 0 1 1 2 3 5임

```

(gdb) x/d $rsp
0x7fffffff4a0: 1

```

그림 7. rsp값 확인

을 확인할 수 있다.

### (3) Phase\_3

solution : 4 a 263

```
(gdb) disas phase_3
Dump of assembler code for function phase_3:
0x000055555555297 <+2>: sub    $0x20,%rsp
0x0000555555552a0 <+3>: mov    %fs:0x28,%rax
0x0000555555552a5 <+18>: xor    %eax,%eax
0x0000555555552a7 <+20>: lea    0xf(%rsp),%rcx
0x0000555555552ac <+25>: lea    0x10(%rsp),%rdx
0x0000555555552b1 <+30>: lea    0x14(%rsp),%rdi
0x0000555555552b6 <+35>: lea    0x1640(%rsp),%rsi
0x0000555555552bd <+42>: callq  0x55555554ef0 <__isoc99_sscanf@plt>
0x0000555555552c2 <+47>: cmp    $0x2,%eax
0x0000555555552c3 <+50>: jle     0x555555552e6 <phase_3+83>
0x0000555555552c7 <+52>: cmpl   $0x7,0x10(%rsp)
0x0000555555552cc <+57>: ja      0x555555553d7 <phase_3+324>
0x0000555555552d1 <+60>: mov     0x10(%rsp),%eax
0x0000555555552d6 <+67>: lea     0x1640(%rsp),%rdi
0x0000555555552dd <+74>: movslq  (%rdi,%rax,4),%rax
0x0000555555552e1 <+78>: add     %rdi,%rax
0x0000555555552e4 <+81>: jmpq    %rax
0x0000555555552e6 <+83>: callq   0x555555558b1 <explode_bomb>
0x0000555555552eb <+88>: jmp     0x555555552c7 <phase_3+52>
0x0000555555552ed <+90>: mov     $0x72,%eax
0x0000555555552f2 <+95>: cmpl   $0x160,0x14(%rsp)
0x0000555555552fa <+103>: je      0x555555553e1 <phase_3+334>
0x0000555555552fb <+108>: callq   0x555555558b1 <explode_bomb>
0x000055555555305 <+114>: mov     $0x72,%eax
0x00005555555530a <+119>: jmpq    0x555555553e1 <phase_3+334>
0x00005555555530f <+124>: mov     $0x77,%eax
0x000055555555314 <+129>: cmpl   $0x264,0x14(%rsp)
0x00005555555531c <+137>: je      0x555555553e1 <phase_3+334>
0x000055555555322 <+143>: callq   0x555555558b1 <explode_bomb>
0x000055555555327 <+148>: mov     $0x77,%eax
0x00005555555532c <+153>: jmpq    0x555555553e1 <phase_3+334>
0x000055555555331 <+158>: mov     $0x72,%eax
0x000055555555336 <+163>: cmpl   $0x264,0x14(%rsp)
0x00005555555533b <+171>: je      0x555555553e1 <phase_3+334>
0x000055555555344 <+177>: callq   0x555555558b1 <explode_bomb>
0x000055555555349 <+182>: mov     $0x72,%eax
--Type <return> to continue, or q <return> to quit--
0x00005555555534e <+187>: jmpq    0x555555553e1 <phase_3+334>
0x000055555555353 <+192>: jmpq    $0x76,%eax
0x000055555555358 <+197>: cmpl   $0x258,0x14(%rsp)
0x000055555555360 <+205>: je      0x555555553e1 <phase_3+334>
0x000055555555362 <+207>: callq   0x555555558b1 <explode_bomb>
0x000055555555367 <+212>: mov     $0x76,%eax
0x00005555555536c <+217>: jmp     0x555555553e1 <phase_3+334>
0x00005555555536e <+219>: mov     $0x61,%eax
0x000055555555373 <+224>: cmpl   $0x107,0x14(%rsp)
0x00005555555537b <+232>: je      0x555555553e1 <phase_3+334>
0x00005555555537d <+234>: callq   0x555555558b1 <explode_bomb>
0x000055555555382 <+239>: mov     $0x61,%eax
0x000055555555387 <+244>: jmp     0x555555553e1 <phase_3+334>
0x000055555555389 <+246>: mov     $0x6a,%eax
0x00005555555538e <+251>: cmpl   $0xe7,0x14(%rsp)
0x000055555555396 <+259>: je      0x555555553e1 <phase_3+334>
0x000055555555398 <+261>: callq   0x555555558b1 <explode_bomb>
0x00005555555539d <+266>: mov     $0x6a,%eax
0x0000555555553a2 <+271>: jmp     0x555555553e1 <phase_3+334>
0x0000555555553a4 <+273>: mov     $0x75,%eax
0x0000555555553a9 <+278>: cmpl   $0x57,0x14(%rsp)
0x0000555555553ae <+283>: je      0x555555553e1 <phase_3+334>
0x0000555555553b0 <+285>: callq   0x555555558b1 <explode_bomb>
0x0000555555553b5 <+290>: mov     $0x75,%eax
0x0000555555553ba <+295>: jmp     0x555555553e1 <phase_3+334>
0x0000555555553bc <+297>: mov     $0x76,%eax
0x0000555555553c1 <+302>: cmpl   $0x231,0x14(%rsp)
0x0000555555553c9 <+310>: je      0x555555553e1 <phase_3+334>
0x0000555555553cb <+312>: callq   0x555555558b1 <explode_bomb>
0x0000555555553d0 <+317>: mov     $0x76,%eax
0x0000555555553d5 <+322>: jmp     0x555555553e1 <phase_3+334>
0x0000555555553d7 <+324>: callq   0x555555558b1 <explode_bomb>
0x0000555555553dc <+329>: mov     $0x75,%eax
0x0000555555553e1 <+334>: cmp     %al,0xf(%rsp)
0x0000555555553e5 <+338>: je      0x555555553ec <phase_3+345>
0x0000555555553e7 <+340>: callq   0x555555558b1 <explode_bomb>
0x0000555555553ec <+345>: mov     0x18(%rsp),%rax
0x0000555555553f1 <+350>: xor     %fs:0x28,%rax
--Type <return> to continue, or q <return> to quit--
0x0000555555553fa <+359>: jne     0x55555555401 <phase_3+366>
0x0000555555553fc <+361>: add     $0x28,%rsp
0x000055555555400 <+365>: retq
0x000055555555401 <+366>: callq   0x555555554e0 <__stack_chk_fail@plt>
End of assembler dump.
```

그림 8. disas phase\_3

마찬가지로 phase\_3에 break를 걸어놓고 disas phase\_3을 해본다. Phase\_2와 마찬가지로 비슷한 곳 \$rip+0x1640쪽에서 입력을 받을것이라고 생각했고, 그 주소값을 x/s 해본 결과 "%d %c %d"를 입력받는다는 걸 알 수 있었다.

```
(gdb) x/s 0x555555556906
0x555555556906: "%d %c %d"
```

그림 9. 입력값 확인 (%d %c %d)

이대로 %d %c %d값을 정확하게 입력하면 rax값은 3 이상이 입력되어서 2 이하면 bomb은 피할 수 있다. 그 뒤로 7과 0x10(\$rsp)를 비교해서 above면 bomb된다. Rsp+10에는 x를 통해 확인해보면 첫번째 입력값이 들어가고있음을 알 수 있고, 이 말은 곧 첫번째값이 7보다 크면 bomb이라는 뜻이므로 7이하여야 한다는 뜻이다. 7 이하의 정수 아무값이나 입력해보면(나는 4를 입력했다) 그 뒤에 ni를 통해 코드를 계속 쫓아가보면 mov, add를 이용한 일종의 연산을 거쳐 rax레지스터의 값에 접근한다. 4를 입력했을 때에는 +219 즉, rax에는 0x61이 들어가고 0x107(=263)를 세번째값(=rsp+0x14)과 비교하여 같지 않으면 bomb, 즉 rax에 들어갈 세번째 값이 0x107, 십진수로 263이 아니면 bomb한다는 뜻이라고 추측할 수 있었다. 그 뒤 rax에 +334로 점프하는데 이때 아까 넣은 %al (rax의 하위 8비트)값과 두번째 입력값이 같아야 bomb가 아님을 알 수 있다. 즉, 0x61 (=97)을 아스키 코드로 연산한 a값이 들어가야한다는 것을 알 수 있다 -> 따라서 4 a 263가 되며, 이외에도 첫 정수가 무엇이냐에 따라 jump하는 곳이 달라져 뒤의 %c %d값도 달라질 것이다.

### (4) Phase\_4

## Solution : 5 2

```
(gdb)
Dump of assembler code for function phase_4:
0x000055555555445 <+0>:  sub    $0x18,%rsp
0x000055555555449 <+4>:  mov     %fs:0x28,%rax
0x000055555555452 <+13>: mov     %rax,0x8(%rsp)
0x000055555555457 <+18>:  xor     %eax,%eax
0x000055555555459 <+20>:  lea     0x4(%rsp),%rcx
0x00005555555545e <+25>:  mov     %rsp,%rdx
0x000055555555461 <+28>:  lea     0x1627(%rip),%rsi    # 0x555555556a8f
0x000055555555468 <+35>:  callq   0x555555554ef0 <__isoc99_sscanf@plt>
0x00005555555546d <+40>:  cmp     $0x2,%eax
0x000055555555470 <+43>:  jne     0x55555555478 <phase_4+51>
0x000055555555472 <+45>:  cmpl    $0xe,(%rsp)
0x000055555555476 <+49>:  jbe     0x5555555547d <phase_4+56>
0x000055555555478 <+51>:  callq   0x5555555558b1 <explode_bomb>
0x00005555555547d <+56>:  mov     $0xe,%edx
0x000055555555482 <+61>:  mov     $0x0,%esi
0x000055555555487 <+66>:  mov     (%rsp),%edi
0x00005555555548a <+69>:  callq   0x55555555406 <func4>
0x00005555555548f <+74>:  cmp     $0x2,%eax
0x000055555555492 <+77>:  jne     0x5555555549b <phase_4+86>
0x000055555555494 <+79>:  cmpl    $0x2,0x4(%rsp)
0x000055555555499 <+84>:  je      0x555555554a0 <phase_4+91>
0x00005555555549b <+86>:  callq   0x5555555558b1 <explode_bomb>
0x0000555555554a0 <+91>:  mov     0x8(%rsp),%rax
0x0000555555554a5 <+96>:  xor     %fs:0x28,%rax
0x0000555555554ae <+105>: jne     0x555555554b5 <phase_4+112>
0x0000555555554b0 <+107>:  add     $0x18,%rsp
0x0000555555554b4 <+111>:  retq
0x0000555555554b5 <+112>:  callq   0x555555554e50 <__stack_chk_fail@plt>
End of assembler dump.
```

그림 10. disas phase\_4

```
End of assembler dump.
(gdb) disas func4
Dump of assembler code for function func4:
0x000055555555406 <+0>:  sub     $0x8,%rsp
0x00005555555540a <+4>:  mov     %edx,%eax
0x00005555555540c <+6>:  sub     %esi,%eax
0x00005555555540e <+8>:  mov     %eax,%ecx
0x000055555555410 <+10>: shr     $0x1f,%ecx
0x000055555555413 <+13>: add     %eax,%ecx
0x000055555555415 <+15>: sar     %ecx
0x000055555555417 <+17>: add     %esi,%ecx
0x000055555555419 <+19>: cmp     %edi,%ecx
0x00005555555541b <+21>: jg      0x5555555542b <func4+37>
0x00005555555541d <+23>: mov     $0x0,%eax
0x000055555555422 <+28>: cmp     %edi,%ecx
0x000055555555424 <+30>: jl      0x55555555437 <func4+49>
0x000055555555426 <+32>: add     $0x8,%rsp
0x00005555555542a <+36>: retq
0x00005555555542b <+37>: lea     -0x1(%rcx),%edx
0x00005555555542e <+40>: callq   0x55555555406 <func4>
0x000055555555433 <+45>: add     %eax,%eax
0x000055555555435 <+47>: jmp     0x55555555426 <func4+32>
0x000055555555437 <+49>: lea     0x1(%rcx),%esi
0x00005555555543a <+52>: callq   0x55555555406 <func4>
0x00005555555543f <+57>: lea     0x1(%rax,%rax,1),%eax
0x000055555555443 <+61>: jmp     0x55555555426 <func4+32>
End of assembler dump.
```

그림 11. disas func4

우선 phase\_4에 break를 넣어주고 위와 똑같이 입력값을 확인한다. 그럼 %d %d를 확인할 수 있고 정수 두개를 입력해야한다는 것을 알 수 있다.

```
End of assembler dump.
(gdb) x/s 0x555555556a8f
0x555555556a8f: "%d %d"
(gdb) x/s %fs
```

그림 10. 입력값 (%d %d)

그 뒤로 입력 data가 올바르면 rax에 2가 들어가서 bomb으로 가지 않는다는 것을 알 수 있다. 그 뒤 %rsp값(첫 입력값)이 0xe(=14)와 비교해서 below값을 갖는다면 bomb으로 가지 않는다는 것을 알 수 있다. 그리고 edx에 14, esi에 0을 넣고 rsp(입력값)을 edi에 넣는다. 즉, 매개변수 (첫값, 0)을 func4에 넘겨주고 나온 return값 = rax값을 2와 비교해서



같지 않으면 bomb이라는 것을 알 수 있다. 사실 나 같은 경우는 첫 예시로 넣은 값이 바로 5였는데 func4를 지나고 나온 rax가 바로 2로 나와서 별 고민없이 첫번째 값을 해결한 경우이다. 그 다음 rsp+4 즉 두번째값이 2와 같으면 bomb이 아니다. 즉, 2번째 값은 2여야 한다. 즉, 답은 5 2임 알 수 있다. 나 같은 경우는 func4의 해석 없이도 바로 답을 알 수 있었는데, func4를 해석해보면 재귀함수의 형태로 계산한 값을 integer로 반환한다는 것을 알 수 있다.

## (5) Phase\_5

**Solution : 5 115**

```
(gdb)
(gdb) disas phase_5
Dump of assembler code for function phase_5:
=> 0x0000555555554ba <+0>:  sub    $0x18,%rsp
0x0000555555554be <+4>:  mov     %fs:0x28,%rax
0x0000555555554c7 <+13>:  mov     %rax,0x8(%rsp)
0x0000555555554cc <+18>:  xor     %eax,%eax
0x0000555555554ce <+20>:  lea     0x4(%rsp),%rcx
0x0000555555554d3 <+25>:  mov     %rsp,%rdx
0x0000555555554d6 <+28>:  lea     0x15b2(%rip),%rsi    # 0x555555556a8f
0x0000555555554dd <+35>:  callq   0x555555554ef0 <__isoc99_sscanf@plt>
0x0000555555554e2 <+40>:  cmp     $0x1,%eax
0x0000555555554e5 <+43>:  jle     0x555555555541 <phase_5+135>
0x0000555555554e7 <+45>:  mov     (%rsp),%eax
0x0000555555554ea <+48>:  and     $0xf,%eax
0x0000555555554ed <+51>:  mov     %eax,0x0(%rsp)
0x0000555555554f0 <+54>:  cmp     $0xf,%eax
0x0000555555554f3 <+57>:  je      0x555555555527 <phase_5+109>
0x0000555555554f5 <+59>:  mov     $0x0,%ecx
0x0000555555554fa <+64>:  mov     $0x0,%edx
0x0000555555554ff <+69>:  lea     0x143a(%rip),%rsi    # 0x555555556940 <array.3416>
0x000055555555506 <+76>:  add     $0x1,%edx
0x000055555555509 <+79>:  cltq
0x00005555555550b <+81>:  mov     (%rsi,%rax,4),%eax
0x00005555555550e <+84>:  add     %eax,%ecx
0x000055555555510 <+86>:  cmp     $0xf,%eax
0x000055555555513 <+89>:  jne     0x555555555506 <phase_5+76>
0x000055555555515 <+91>:  movl    $0xf,0x0(%rsp)
0x00005555555551c <+98>:  cmp     $0xf,%edx
0x00005555555551f <+101>:  jne     0x555555555527 <phase_5+109>
0x000055555555521 <+103>:  cmp     %ecx,0x4(%rsp)
0x000055555555525 <+107>:  je      0x55555555552c <phase_5+114>
0x000055555555527 <+109>:  callq   0x55555555558b1 <explode_bomb>
0x00005555555552c <+114>:  mov     0x8(%rsp),%rax
0x000055555555531 <+119>:  xor     %fs:0x28,%rax
0x00005555555553a <+128>:  jne     0x555555555548 <phase_5+142>
0x00005555555553c <+130>:  add     $0x18,%rsp
0x000055555555540 <+134>:  retq
0x000055555555541 <+135>:  callq   0x55555555558b1 <explode_bomb>
---Type <return> to continue, or q <return> to quit---
0x000055555555546 <+140>:  jmp     0x55555555554e7 <phase_5+45>
0x000055555555548 <+142>:  callq   0x55555555554e50 <__stack_chk_fail@plt>
End of assembler dump.
(gdb)
```

그림 11. disas phase\_5

Phase\_5에 break를 건 다음 입력값을 확인해보았다. 그랬더니 "%d %d"라는 두개의 정수 입력을 확인할 수 있었다.

```
(gdb) x/s 0x555555556a8f
0x555555556a8f: "%d %d"
```

그림 12. 입력값 (%d %d)

코드를 살펴보면 rax를 1과 비교해서 더 작으면 bomb이라는 뜻인데 마찬가지로 입력 data가 정확하지 않으면 bomb으로 간다. 그리고 rax값에 입력값 rsp를 넣고 그 값이 0xf(=15)와 비교해서 같으면 또 bomb로 간다. 그러니까 첫번째 값은 15가 아니어야 한다.

또, +69에서 rsi에 배열을 집어넣는 것을 알 수 있는데, rsi값을 도출하면 (x/16uw \$rsi) 배열이 있음을 알 수 있다. 이 값은 0부터 15까지 중복없이 랜덤으로 있는 것을 알 수 있

```
0x0000555555555506 in phase_5 ()
(gdb) x/16uw $rsi
0x5555555556940 <array.3416>: 10      2      14      7
0x5555555556950 <array.3416+16>: 8      12     15     11
0x5555555556960 <array.3416+32>: 0      4      1      13
0x5555555556970 <array.3416+48>: 3      9      6      5
(gdb)
```

그림 13. array 값

는다. 그 이후로 +89를 보면 여기가 반복문이라는 것을 알 수 있고, ecx, edx에 0을 넣어 놓고 한번 반복할때마다 edx를 1씩 증가시키면서 rax가 15가 될때까지 반복을 해주는 것이다. 즉, 배열에 있는 숫자의 인덱스로 옮겨가다가 15를 만나면 멈추는 것을 알 수 있다. 그 뒤의 코드를 보면 아까 증가시켜준 edx의 값이 0xf(=15)가 되어야 한다. 같지 않으면 bomb이 되는 것을 알 수 있다. 그렇기에 첫번째 값을 제외하고 모든 숫자를 만난 후에 15라는 값을 만나야 하므로 5를 넣어야 5->12->3->7->11->13->9->4->8->0->10->1->2->14->6->15로 가장 마지막에 만나니까 첫번째 숫자는 5가 되어야 하고, 두번째 숫자는 ecx값과 같아야 bomb로 가지 않으므로 ecx 즉 계속 eax값을 더해준 배열의 합을 의미하므로 0+1+...+15 = 115가 두번째 숫자가 되어야한다. 즉, 답은 5 115임을 알 수 있다.

(6) Phase\_6

**Solution : 3 2 4 6 5 1**

```
Dump of assembler code for function phase_6:
-> 0x000055555555554d <+0>: push %r13
0x000055555555554f <+2>: push %r12
0x0000555555555551 <+4>: push %rbp
0x0000555555555553 <+6>: push %rbx
0x0000555555555555 <+8>: sub $0x68,%rsp
0x0000555555555557 <+10>: mov %fs:0x28,%rax
0x0000555555555559 <+12>: mov %rax,0x58(%rsp)
0x000055555555555b <+14>: xor %eax,%eax
0x000055555555555d <+16>: mov %rsp,%r12
0x000055555555555f <+18>: mov %r12,%rsi
0x0000555555555561 <+20>: callq 0x55555555558d7 <read_six_numbers>
0x0000555555555563 <+22>: mov $0x0,%r18d
0x0000555555555565 <+24>: mov %r18,%r13d
0x0000555555555567 <+26>: jmp 0x555555555559f <phase_6+82>
0x0000555555555569 <+28>: callq 0x55555555558b1 <explode_bomb>
0x000055555555556b <+30>: jmp 0x55555555555ae <phase_6+97>
0x000055555555556d <+32>: add $0x1,%ebx
0x000055555555556f <+34>: cmp $0x5,%ebx
0x0000555555555571 <+36>: jg 0x555555555559b <phase_6+78>
0x0000555555555573 <+38>: movslq %ebx,%rax
0x0000555555555575 <+40>: mov (%rsp,%rax,4),%eax
0x0000555555555577 <+42>: cmp %eax,0x0(%rsp)
0x0000555555555579 <+44>: jne 0x5555555555581 <phase_6+52>
0x000055555555557b <+46>: callq 0x55555555558b1 <explode_bomb>
0x000055555555557d <+48>: jmp 0x5555555555581 <phase_6+52>
0x000055555555557f <+50>: add $0x1,%r12
0x0000555555555581 <+52>: mov %r12,%rbp
0x0000555555555583 <+54>: mov (%r12),%eax
0x0000555555555585 <+56>: sub $0x1,%eax
0x0000555555555587 <+58>: cmp $0x5,%eax
0x0000555555555589 <+60>: ja 0x555555555557a <phase_6+45>
0x000055555555558b <+62>: add $0x1,%r13d
0x000055555555558d <+64>: cmp $0x6,%r13d
0x000055555555558f <+66>: je 0x55555555555d5 <phase_6+160>
0x0000555555555591 <+68>: mov %r18,%ebx
0x0000555555555593 <+70>: jmp 0x5555555555589 <phase_6+50>
0x0000555555555595 <+72>: mov 0x8(%rdx),%rdx
0x0000555555555597 <+74>: add $0x1,%eax
0x0000555555555599 <+76>: cmp %ecx,%eax
0x000055555555559b <+78>: jne 0x55555555555bd <phase_6+112>
0x000055555555559d <+80>: cmp %ecx,%eax
0x000055555555559f <+82>: jne 0x55555555555bd <phase_6+112>
0x00005555555555a1 <+84>: cmp %ecx,%eax
0x00005555555555a3 <+86>: jne 0x55555555555bd <phase_6+112>
0x00005555555555a5 <+88>: cmp %ecx,%eax
0x00005555555555a7 <+90>: jne 0x55555555555bd <phase_6+112>
0x00005555555555a9 <+92>: cmp %ecx,%eax
0x00005555555555ab <+94>: jne 0x55555555555bd <phase_6+112>
0x00005555555555ad <+96>: cmp %ecx,%eax
0x00005555555555af <+98>: jne 0x55555555555bd <phase_6+112>
0x00005555555555b1 <+100>: cmp %ecx,%eax
0x00005555555555b3 <+102>: jne 0x55555555555bd <phase_6+112>
0x00005555555555b5 <+104>: cmp %ecx,%eax
0x00005555555555b7 <+106>: jne 0x55555555555bd <phase_6+112>
0x00005555555555b9 <+108>: cmp %ecx,%eax
0x00005555555555bb <+110>: jne 0x55555555555bd <phase_6+112>
0x00005555555555bd <+112>: mov 0x8(%rdx),%rdx
0x00005555555555bf <+114>: add $0x1,%eax
0x00005555555555c1 <+116>: cmp %ecx,%eax
0x00005555555555c3 <+118>: jne 0x55555555555bd <phase_6+112>
0x00005555555555c5 <+120>: cmp %ecx,%eax
0x00005555555555c7 <+122>: jne 0x55555555555bd <phase_6+112>
0x00005555555555c9 <+124>: cmp %ecx,%eax
0x00005555555555cb <+126>: jne 0x55555555555bd <phase_6+112>
0x00005555555555cd <+128>: cmp %ecx,%eax
0x00005555555555cf <+130>: jne 0x55555555555bd <phase_6+112>
0x00005555555555d1 <+132>: cmp %ecx,%eax
0x00005555555555d3 <+134>: jne 0x55555555555bd <phase_6+112>
0x00005555555555d5 <+136>: cmp %ecx,%eax
0x00005555555555d7 <+138>: jne 0x55555555555bd <phase_6+112>
0x00005555555555d9 <+140>: mov $0x20,%rdx
0x00005555555555db <+142>: mov %rdx,0x20(%rsp,%rsi,8)
0x00005555555555dd <+144>: add $0x1,%rsi
0x00005555555555df <+146>: cmp $0x6,%rsi
0x00005555555555e1 <+148>: je 0x55555555555f4 <phase_6+167>
0x00005555555555e3 <+150>: mov (%rsp,%rsi,4),%ecx
0x00005555555555e5 <+152>: mov $0x1,%eax
0x00005555555555e7 <+154>: lea 0x20c2a(%rsp),%rdx
0x00005555555555e9 <+156>: cmp $0x1,%ecx
0x00005555555555eb <+158>: jg 0x55555555555bd <phase_6+112>
0x00005555555555ed <+160>: jmp 0x55555555555c8 <phase_6+123>
0x00005555555555ef <+162>: mov $0x0,%esi
0x00005555555555f1 <+164>: jmp 0x55555555555d7 <phase_6+138>
0x00005555555555f3 <+166>: mov 0x20(%rsp),%rbx
0x00005555555555f5 <+168>: mov 0x28(%rsp),%rax
0x00005555555555f7 <+170>: mov %rax,0x8(%rbx)
0x00005555555555f9 <+172>: mov 0x30(%rsp),%rdx
0x00005555555555fb <+174>: mov %rdx,0x8(%rdx)
0x00005555555555fd <+176>: mov %rdx,0x8(%rdx)
0x00005555555555ff <+178>: mov $0x38,%rax
0x0000555555555601 <+180>: mov %rax,0x8(%rdx)
0x0000555555555603 <+182>: mov $0x40,%rdx
0x0000555555555605 <+184>: mov %rdx,0x8(%rdx)
0x0000555555555607 <+186>: mov $0x48,%rdx
0x0000555555555609 <+188>: mov %rdx,0x8(%rdx)
0x000055555555560b <+190>: mov $0x50,%rdx
0x000055555555560d <+192>: mov %rdx,0x8(%rdx)
0x000055555555560f <+194>: mov $0x58,%rdx
0x0000555555555611 <+196>: mov %rdx,0x8(%rdx)
0x0000555555555613 <+198>: mov $0x60,%rdx
0x0000555555555615 <+200>: mov %rdx,0x8(%rdx)
0x0000555555555617 <+202>: mov $0x68,%rdx
0x0000555555555619 <+204>: mov %rdx,0x8(%rdx)
0x000055555555561b <+206>: mov $0x70,%rdx
0x000055555555561d <+208>: mov %rdx,0x8(%rdx)
0x000055555555561f <+210>: mov $0x78,%rdx
0x0000555555555621 <+212>: mov %rdx,0x8(%rdx)
0x0000555555555623 <+214>: mov $0x80,%rdx
0x0000555555555625 <+216>: mov %rdx,0x8(%rdx)
0x0000555555555627 <+218>: mov $0x88,%rdx
0x0000555555555629 <+220>: mov %rdx,0x8(%rdx)
0x000055555555562b <+222>: jmp 0x5555555555563e <phase_6+241>
0x000055555555562d <+224>: mov 0x8(%rbx),%rbx
0x000055555555562f <+226>: sub $0x1,%rbp
0x0000555555555631 <+228>: je 0x5555555555564f <phase_6+258>
0x0000555555555633 <+230>: mov 0x8(%rbx),%rax
0x0000555555555635 <+232>: mov (%rax),%eax
0x0000555555555637 <+234>: cmp %eax,%rbx
0x0000555555555639 <+236>: jge 0x55555555555635 <phase_6+232>
0x000055555555563b <+238>: callq 0x55555555558b1 <explode_bomb>
0x000055555555563d <+240>: jmp 0x55555555555635 <phase_6+232>
0x000055555555563f <+242>: cmp %ecx,%eax
0x0000555555555641 <+244>: jne 0x55555555555bd <phase_6+112>
0x0000555555555643 <+246>: cmp %ecx,%eax
0x0000555555555645 <+248>: jne 0x55555555555bd <phase_6+112>
0x0000555555555647 <+250>: cmp %ecx,%eax
0x0000555555555649 <+252>: jne 0x55555555555bd <phase_6+112>
0x000055555555564b <+254>: cmp %ecx,%eax
0x000055555555564d <+256>: jne 0x55555555555bd <phase_6+112>
0x000055555555564f <+258>: cmp %ecx,%eax
0x0000555555555651 <+260>: jne 0x55555555555bd <phase_6+112>
0x0000555555555653 <+262>: cmp %ecx,%eax
0x0000555555555655 <+264>: jne 0x55555555555bd <phase_6+112>
0x0000555555555657 <+266>: cmp %ecx,%eax
0x0000555555555659 <+268>: jne 0x55555555555bd <phase_6+112>
0x000055555555565b <+270>: cmp %ecx,%eax
0x000055555555565d <+272>: jne 0x55555555555bd <phase_6+112>
0x000055555555565f <+274>: cmp %ecx,%eax
0x0000555555555661 <+276>: jne 0x55555555555bd <phase_6+112>
0x0000555555555663 <+278>: cmp %ecx,%eax
0x0000555555555665 <+280>: jne 0x55555555555bd <phase_6+112>
0x0000555555555667 <+282>: cmp %ecx,%eax
0x0000555555555669 <+284>: jne 0x55555555555bd <phase_6+112>
0x000055555555566b <+286>: cmp %ecx,%eax
0x000055555555566d <+288>: jne 0x55555555555bd <phase_6+112>
0x000055555555566f <+290>: cmp %ecx,%eax
0x0000555555555671 <+292>: jne 0x55555555555bd <phase_6+112>
0x0000555555555673 <+294>: cmp %ecx,%eax
0x0000555555555675 <+296>: jne 0x55555555555bd <phase_6+112>
0x0000555555555677 <+298>: cmp %ecx,%eax
0x0000555555555679 <+300>: jne 0x55555555555bd <phase_6+112>
0x000055555555567b <+302>: cmp %ecx,%eax
0x000055555555567d <+304>: jne 0x55555555555bd <phase_6+112>
0x000055555555567f <+306>: cmp %ecx,%eax
0x0000555555555681 <+308>: jne 0x55555555555bd <phase_6+112>
0x0000555555555683 <+310>: cmp %ecx,%eax
0x0000555555555685 <+312>: jne 0x55555555555bd <phase_6+112>
0x0000555555555687 <+314>: cmp %ecx,%eax
0x0000555555555689 <+316>: jne 0x55555555555bd <phase_6+112>
0x000055555555568b <+318>: cmp %ecx,%eax
0x000055555555568d <+320>: jne 0x55555555555bd <phase_6+112>
0x000055555555568f <+322>: cmp %ecx,%eax
0x0000555555555691 <+324>: jne 0x55555555555bd <phase_6+112>
0x0000555555555693 <+326>: cmp %ecx,%eax
0x0000555555555695 <+328>: jne 0x55555555555bd <phase_6+112>
0x0000555555555697 <+330>: cmp %ecx,%eax
0x0000555555555699 <+332>: jne 0x55555555555bd <phase_6+112>
0x000055555555569b <+334>: cmp %ecx,%eax
0x000055555555569d <+336>: jne 0x55555555555bd <phase_6+112>
0x000055555555569f <+338>: cmp %ecx,%eax
0x00005555555556a1 <+340>: jne 0x55555555555bd <phase_6+112>
0x00005555555556a3 <+342>: cmp %ecx,%eax
0x00005555555556a5 <+344>: jne 0x55555555555bd <phase_6+112>
0x00005555555556a7 <+346>: cmp %ecx,%eax
0x00005555555556a9 <+348>: jne 0x55555555555bd <phase_6+112>
0x00005555555556ab <+350>: cmp %ecx,%eax
0x00005555555556ad <+352>: jne 0x55555555555bd <phase_6+112>
0x00005555555556af <+354>: cmp %ecx,%eax
0x00005555555556b1 <+356>: jne 0x55555555555bd <phase_6+112>
0x00005555555556b3 <+358>: cmp %ecx,%eax
0x00005555555556b5 <+360>: jne 0x55555555555bd <phase_6+112>
0x00005555555556b7 <+362>: cmp %ecx,%eax
0x00005555555556b9 <+364>: jne 0x55555555555bd <phase_6+112>
0x00005555555556bb <+366>: cmp %ecx,%eax
0x00005555555556bd <+368>: jne 0x55555555555bd <phase_6+112>
0x00005555555556bf <+370>: cmp %ecx,%eax
0x00005555555556c1 <+372>: jne 0x55555555555bd <phase_6+112>
0x00005555555556c3 <+374>: cmp %ecx,%eax
0x00005555555556c5 <+376>: jne 0x55555555555bd <phase_6+112>
0x00005555555556c7 <+378>: cmp %ecx,%eax
0x00005555555556c9 <+380>: jne 0x55555555555bd <phase_6+112>
0x00005555555556cb <+382>: cmp %ecx,%eax
0x00005555555556cd <+384>: jne 0x55555555555bd <phase_6+112>
0x00005555555556cf <+386>: cmp %ecx,%eax
0x00005555555556d1 <+388>: jne 0x55555555555bd <phase_6+112>
0x00005555555556d3 <+390>: cmp %ecx,%eax
0x00005555555556d5 <+392>: jne 0x55555555555bd <phase_6+112>
0x00005555555556d7 <+394>: cmp %ecx,%eax
0x00005555555556d9 <+396>: jne 0x55555555555bd <phase_6+112>
0x00005555555556db <+398>: cmp %ecx,%eax
0x00005555555556dd <+400>: jne 0x55555555555bd <phase_6+112>
0x00005555555556df <+402>: cmp %ecx,%eax
0x00005555555556e1 <+404>: jne 0x55555555555bd <phase_6+112>
0x00005555555556e3 <+406>: cmp %ecx,%eax
0x00005555555556e5 <+408>: jne 0x55555555555bd <phase_6+112>
0x00005555555556e7 <+410>: cmp %ecx,%eax
0x00005555555556e9 <+412>: jne 0x55555555555bd <phase_6+112>
0x00005555555556eb <+414>: cmp %ecx,%eax
0x00005555555556ed <+416>: jne 0x55555555555bd <phase_6+112>
0x00005555555556ef <+418>: cmp %ecx,%eax
0x00005555555556f1 <+420>: jne 0x55555555555bd <phase_6+112>
0x00005555555556f3 <+422>: cmp %ecx,%eax
0x00005555555556f5 <+424>: jne 0x55555555555bd <phase_6+112>
0x00005555555556f7 <+426>: cmp %ecx,%eax
0x00005555555556f9 <+428>: jne 0x55555555555bd <phase_6+112>
0x00005555555556fb <+430>: cmp %ecx,%eax
0x00005555555556fd <+432>: jne 0x55555555555bd <phase_6+112>
0x00005555555556ff <+434>: cmp %ecx,%eax
0x0000555555555701 <+436>: jne 0x55555555555bd <phase_6+112>
0x0000555555555703 <+438>: cmp %ecx,%eax
0x0000555555555705 <+440>: jne 0x55555555555bd <phase_6+112>
0x0000555555555707 <+442>: cmp %ecx,%eax
0x0000555555555709 <+444>: jne 0x55555555555bd <phase_6+112>
0x000055555555570b <+446>: cmp %ecx,%eax
0x000055555555570d <+448>: jne 0x55555555555bd <phase_6+112>
0x000055555555570f <+450>: cmp %ecx,%eax
0x0000555555555711 <+452>: jne 0x55555555555bd <phase_6+112>
0x0000555555555713 <+454>: cmp %ecx,%eax
0x0000555555555715 <+456>: jne 0x55555555555bd <phase_6+112>
0x0000555555555717 <+458>: cmp %ecx,%eax
0x0000555555555719 <+460>: jne 0x55555555555bd <phase_6+112>
0x000055555555571b <+462>: cmp %ecx,%eax
0x000055555555571d <+464>: jne 0x55555555555bd <phase_6+112>
0x000055555555571f <+466>: cmp %ecx,%eax
0x0000555555555721 <+468>: jne 0x55555555555bd <phase_6+112>
0x0000555555555723 <+470>: cmp %ecx,%eax
0x0000555555555725 <+472>: jne 0x55555555555bd <phase_6+112>
0x0000555555555727 <+474>: cmp %ecx,%eax
0x0000555555555729 <+476>: jne 0x55555555555bd <phase_6+112>
0x000055555555572b <+478>: cmp %ecx,%eax
0x000055555555572d <+480>: jne 0x55555555555bd <phase_6+112>
0x000055555555572f <+482>: cmp %ecx,%eax
0x0000555555555731 <+484>: jne 0x55555555555bd <phase_6+112>
0x0000555555555733 <+486>: cmp %ecx,%eax
0x0000555555555735 <+488>: jne 0x55555555555bd <phase_6+112>
0x0000555555555737 <+490>: cmp %ecx,%eax
0x0000555555555739 <+492>: jne 0x55555555555bd <phase_6+112>
0x000055555555573b <+494>: cmp %ecx,%eax
0x000055555555573d <+496>: jne 0x55555555555bd <phase_6+112>
0x000055555555573f <+498>: cmp %ecx,%eax
0x0000555555555741 <+500>: jne 0x55555555555bd <phase_6+112>
0x0000555555555743 <+502>: cmp %ecx,%eax
0x0000555555555745 <+504>: jne 0x55555555555bd <phase_6+112>
0x0000555555555747 <+506>: cmp %ecx,%eax
0x0000555555555749 <+508>: jne 0x55555555555bd <phase_6+112>
0x000055555555574b <+510>: cmp %ecx,%eax
0x000055555555574d <+512>: jne 0x55555555555bd <phase_6+112>
0x000055555555574f <+514>: cmp %ecx,%eax
0x0000555555555751 <+516>: jne 0x55555555555bd <phase_6+112>
0x000
```

```

(gdb) disas read_six_numbers
Dump of assembler code for function read_six_numbers:
=> 0x0000555555558d7 <+0>:      sub    $0x8,%rsp
0x0000555555558db <+4>:      mov     %rsi,%rdx
0x0000555555558de <+7>:      lea     0x4(%rsi),%rcx
0x0000555555558e2 <+11>:     lea     0x14(%rsi),%rax
0x0000555555558e6 <+15>:     push   %rax
0x0000555555558e7 <+16>:     lea     0x10(%rsi),%rax
0x0000555555558eb <+20>:     push   %rax
0x0000555555558ec <+21>:     lea     0xc(%rsi),%r9
0x0000555555558f0 <+25>:     lea     0x8(%rsi),%r8
0x0000555555558f4 <+29>:     lea     0x1188(%rip),%rsi      # 0x555555556a83
0x0000555555558fb <+36>:     mov     $0x0,%eax
0x000055555555900 <+41>:     callq  0x555555554ef0 <__isoc99_sscanf@plt>
0x000055555555905 <+46>:     add     $0x10,%rsp
0x000055555555909 <+50>:     cmp     $0x5,%eax
0x00005555555590c <+53>:     jle     0x555555555913 <read_six_numbers+60>
0x00005555555590e <+55>:     add     $0x8,%rsp
0x000055555555912 <+59>:     retq
0x000055555555913 <+60>:     callq  0x5555555558b1 <explode_bomb>
End of assembler dump.
(gdb)

```

그림 17. disas read\_six\_numbers

```

(gdb) x/s 0x555555556a83
0x555555556a83: "%d %d %d %d %d %d"
(gdb)

```

그림 16. 입력값 (%d %d %d %d %d %d)

이후 다시 phase\_6으로 와서 살펴보면 rax에 -1을 한 후에 5와 비교를 했는데 크면 bomb으로 가기 때문에 즉, rax값이 6 이하여야 한다. 또, +146에는 node1의 값이 들어 가는데, 이 값을 바로 x/24w 주소값을 입력했더니 node1과 node5까지의 정해진 값을 볼 수 있다. 이 방법으로 node 1~5값을 알게되고 그 다음 코드로 내려갔다.

```

(gdb) x/24d 0x555555758210
0x555555758210 <node1>: 92      1      1433764384      21845
0x555555758220 <node2>: 715     2      1433764400      21845
0x555555758230 <node3>: 820     3      1433764416      21845
0x555555758240 <node4>: 461     4      1433764432      21845
0x555555758250 <node5>: 432     5      1433764112      21845
0x555555758260 <host_table>: 1431661289 21845 1431661315 21845
(gdb)

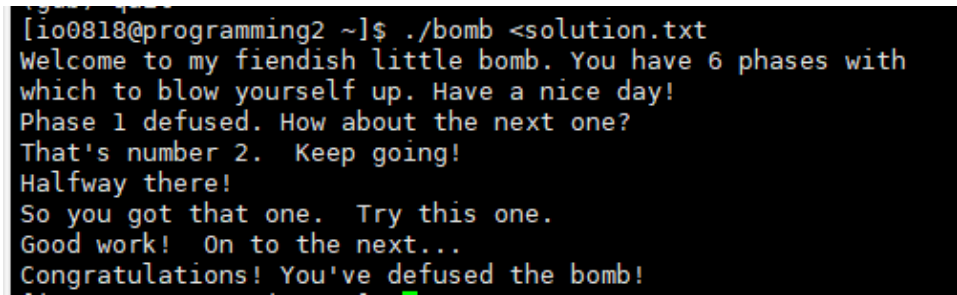
```

그림 15. node1의 x/24d

Ni를 통해서 계속 내려가다 보면 +247에서 cmp와 jge가 있는데, 해석해보면 %rax와 %rbx를 비교해서 %rbx 노드의 값이 더 작아야 bomb가 되지 않는다. 이 과정이 반복된다는 것을 알 수 있었고 코드를 계속해서 살펴보니 첫 바퀴에서는 %rbx가 처음 input값을 가지고 있고, rax는 두번째 값을 갖고 있었다. 폭탄이 터지지 않았다면 rbx는 그 다음 3세번째값을 가지고 rax는 4번째값을 가진다. 또 이 두값에서 뒷값이 더 크면 bomb -> 즉 처음수가 제일 크고 갈수록 작아져야 bomb가 터지지 않는다는 사실을 알 수 있다. 즉, 내림차순으로 정렬해야 폭탄이 터지지 않는다는 것을 알 수 있다. 3번째 바퀴에서 rbx 레지스터를 살펴보니 node6의 값이 455로 4번째로 크다는 사실을 알 수 있었다. 즉, 1~6까지를 오름차순으로 정렬했을 때 값이 3 2 4 6 5 1 임을 알 수 있다.



### III. 실행 사진 첨부



```
[io0818@programming2 ~]$ ./bomb <solution.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2. Keep going!
Halfway there!
So you got that one. Try this one.
Good work! On to the next...
Congratulations! You've defused the bomb!
```

그림 18. 폭탄 해체 완료

### IV. 결론 및 고찰

- 어셈블리어를 통해 C 코드를 유추하고 흐름을 이해할 수 있었다.
- 각종 instruction들과 operation, flag 등이 어떻게 작동하고 high level 언어로 표현이 되는지 알게 되었다.
- 함수의 핵심 파트는 이해가 어느정도 되었지만 stack을 구현하고 caller, callee resistor가 어떻게 공간을 확보하고 return해주는지에 대해 더 이해가 필요하다고 생각된다.