

**2018 시스템 프로그래밍**  
**- Lab 05 -**

제출일자	2018.11.06
분 반	02
이 름	진승언
학 번	201404377

## Phase 1~7 [결과 화면 캡처]

```

c201404377@2018-sp: ~/bomb16
(gdb) r
Starting program: /home/sys02/c201404377/bomb16/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The future will be better tomorrow.
Phase 1 defused. How about the next one?
1 2 4 7 11 16
That's number 2. Keep going!
1 678
Halfway there!
14 7 DrEvil
So you got that one. Try this one.
111114
Good work! On to the next...
3 5 1 6 4 2
Curses, you've found the secret phase!
But finding it and solving it are quite different...
107
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
[Inferior 1 (process 14172) exited normally]
(gdb)

```

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb19	Tue Oct 23 18:29	7	0	70	valid
2	bomb25	Tue Oct 23 19:35	7	0	70	valid
3	bomb2	Wed Oct 24 01:19	7	0	70	valid
4	bomb37	Wed Oct 24 01:41	7	0	70	valid
5	bomb48	Sun Oct 28 00:13	7	0	70	valid
6	bomb40	Tue Oct 30 05:32	7	0	70	valid
7	bomb39	Tue Oct 30 14:44	7	0	70	valid
8	bomb46	Fri Nov 2 21:48	7	0	70	valid
9	bomb20	Sun Nov 4 18:13	7	0	70	valid
10	bomb54	Mon Nov 5 17:30	7	0	70	valid
11	bomb44	Mon Nov 5 21:35	7	0	70	valid
12	bomb5	Mon Nov 5 21:37	7	0	70	valid
13	bomb16	Tue Nov 6 01:39	7	0	70	valid
14	bomb32	Tue Nov 6 01:43	7	0	70	valid
15	bomb41	Wed Oct 24 14:12	7	1	70	valid

## Phase 1 [진행 과정 설명]

```
c201404377@2018-sp: ~/bomb16
(gdb) disas phase_1
Dump of assembler code for function phase_1:
0x0000000000400f2d <+0>:      sub     $0x8,%rsp
0x0000000000400f31 <+4>:      mov     $0x402610,%esi
0x0000000000400f36 <+9>:      callq   0x40137c <strings_not_equal>
0x0000000000400f3b <+14>:     test    %eax,%eax
0x0000000000400f3d <+16>:     je      0x400f44 <phase_1+23>
0x0000000000400f3f <+18>:     callq   0x401650 <explode_bomb>
0x0000000000400f44 <+23>:     add     $0x8,%rsp
0x0000000000400f48 <+27>:     retq
End of assembler dump.
(gdb) x/s 0x402610
0x402610:      "The future will be better tomorrow."
```

먼저 이 문제뿐만 아니라 모든 문제를 exploded\_bomb에 breakpoint를 걸어 주고 풀려는 phase\_번호를 disas(disassemble) 해서 어셈블리어 코드를 확인 후에 explode\_bomb부분을 중심으로 문제를 해결하려고 했다.

+9에서 callq에서 strings\_not\_equal이라는 함수를 호출하는데 문자열을 입력 받은 후 비교를 하고 오답일 경우 explode\_bomb를 호출 한다는 것을 볼 수 있다. 그래서 비교하는 문자열을 찾아야한다고 생각해서 +4부분의 %esi에 값을 mov하는 명령어에서 값인 0x402610을 x/s 명령어를 통해 16진수를 문자열로 보여주게 하였다. 그 결과 The future will be better tomorrow. 라는 문자열을 볼 수 있었고 이게 답임을 알 수 있었다.

## Phase 1 [정답]

The future will be better tomorrow.

## Phase 2 [진행 과정 설명]

```

c201404377@2018-sp: ~/bomb16
(gdb) disas phase_2
Dump of assembler code for function phase_2:
0x0000000000400f49 <+0>:      push    %rbp
0x0000000000400f4a <+1>:      push    %rbx
0x0000000000400f4b <+2>:      sub     $0x28,%rsp
0x0000000000400f4f <+6>:      mov     %fs:0x28,%rax
0x0000000000400f58 <+15>:     mov     %rax,0x18(%rsp)
0x0000000000400f5d <+20>:     xor     %eax,%eax
0x0000000000400f5f <+22>:     mov     %rsp,%rsi
0x0000000000400f62 <+25>:     callq   0x401686 <read_six_numbers>
0x0000000000400f67 <+30>:     cmpl    $0x0, (%rsp)
0x0000000000400f6b <+34>:     jns     0x400f72 <phase_2+41>
0x0000000000400f6d <+36>:     callq   0x401650 <explode_bomb>
0x0000000000400f72 <+41>:     mov     %rsp,%rbp
---Type <return> to continue, or q <return> to quit---return
0x0000000000400f75 <+44>:     mov     $0x1,%ebx
0x0000000000400f7a <+49>:     mov     %ebx,%eax
0x0000000000400f7c <+51>:     add     0x0(%rbp),%eax
0x0000000000400f7f <+54>:     cmp     %eax,0x4(%rbp)
0x0000000000400f82 <+57>:     je      0x400f89 <phase_2+64>
0x0000000000400f84 <+59>:     callq   0x401650 <explode_bomb>
0x0000000000400f89 <+64>:     add     $0x1,%ebx
0x0000000000400f8c <+67>:     add     $0x4,%rbp
0x0000000000400f90 <+71>:     cmp     $0x6,%ebx
0x0000000000400f93 <+74>:     jne     0x400f7a <phase_2+49>
0x0000000000400f95 <+76>:     mov     0x18(%rsp),%rax
0x0000000000400f9a <+81>:     xor     %fs:0x28,%rax
0x0000000000400fa3 <+90>:     je      0x400faa <phase_2+97>
---Type <return> to continue, or q <return> to quit---return
0x0000000000400fa5 <+92>:     callq   0x400b90 <__stack_chk_fail@plt>
0x0000000000400faa <+97>:     add     $0x28,%rsp
0x0000000000400fae <+101>:    pop     %rbx
0x0000000000400faf <+102>:    pop     %rbp
0x0000000000400fb0 <+103>:    retq
End of assembler dump.
(gdb)

```

phase2는 +25에서 read\_six\_numbers라는 함수를 호출하는데 숫자 6개를 입력받는다는 것을 알 수 있다. 또한 아래코드를 보면 입력값이 6개가 아니라면 jns 명령어에 의해 폭탄이 터지는 함수에 들어가는 것을 볼 수 있다. 그리고 jns 위에 cmp는 0과 비교하고 있으므로 인풋 값이 6개보다 적다면 1을 반환하게 해서 폭탄으로 들어가게 해준다.

+64에서 ebp에 1을 더하고, rbx는 현재 배열을 가리키고있는데 주소를 0x4만큼 이동하는 것을 볼 수 있다. 0x4만큼 이동하는 것이 내가 입력한 숫자의 간격만큼 뛰어 읽어옴을 의미한다. int형 이어서 0x4만큼 더해짐을 볼 수 있다. 따라서 정리하자면 2번째 입력된 숫자를 rbp레지스터에 저장한 후 처음 입력된 수에 1을 더하고 rbp(2번째 입력값)와 비교하여 같으면 bomb을 피

할 수 있다. 이는 첫 번째 수+1이 다음 수가 됨을 의미한다. 그다음 두 번째 수와 세 번째 수를 비교할 수 있도록 rbp가 이동하고 더해지는 수도 차례대로 1씩 증가하게 된다. 그리고 더해지는 수가 6이 되면 프로그램이 종료됨을 알 수 있다.

따라서 phase\_2의 답은 앞항과 뒤 항의 차가 1씩 늘어나는 계차수열의 형태임을 알 수 있다. 처음 숫자가 무엇이든 상관없이 그 수부터 차례대로 계차가 1씩 늘어나게 된다. 처음 숫자와 상관없이 계차가 1씩 늘어남을 알 수 있다. 그러므로 정답은 1부터 입력을 시작하면 처음 입력된 수에는 1을 더하므로 2가되고 그 뒤부터는 계차수열이므로 쪽 더 해주는 것을 반복해주면 1 2 4 7 11 16이라는 결과를 얻을 수 있다.

이것을 c언어로 정리하면 다음과 같다.

```
void phase_2(char *lineptr) {
    int num[6];
    int i;
    if( read_six_numbers(lineptr, num) )
        explode(bomb);

    if (numbers[0] != 1)
        explode_bomb();


    i = 1;
    do {
        if ( i + arr[i-1] != num[i])
            explode_bomb();
        i++;
    }while (i <= 6);
}
```

<b>Phase 2</b> [정답]
---------------------

1 2 4 7 11 16



### Phase 3 [진행 과정 설명]

 c201404377@2018-sp: ~/bomb16


Breakpoint 1 at 0x401650

(gdb) disas phase\_3

Dump of assembler code for function phase\_3:

```
0x0000000000400fb1 <+0>:      sub    $0x18,%rsp
0x0000000000400fb5 <+4>:      mov    %fs:0x28,%rax
0x0000000000400fbe <+13>:     mov    %rax,0x8(%rsp)
0x0000000000400fc3 <+18>:     xor    %eax,%eax
0x0000000000400fc5 <+20>:     lea    0x4(%rsp),%rcx
0x0000000000400fca <+25>:     mov    %rsp,%rdx
0x0000000000400fcd <+28>:     mov    $0x40292d,%esi
0x0000000000400fd2 <+33>:     callq 0x400c40 <__isoc99_sscanf@plt>
0x0000000000400fd7 <+38>:     cmp    $0x1,%eax
0x0000000000400fda <+41>:     jg     0x400fe1 <phase_3+48>
0x0000000000400fdc <+43>:     callq 0x401650 <explode_bomb>
0x0000000000400fe1 <+48>:     cmpl   $0x7, (%rsp)
0x0000000000400fe5 <+52>:     ja     0x401022 <phase_3+113>
0x0000000000400fe7 <+54>:     mov    (%rsp),%eax
0x0000000000400fea <+57>:     jmpq   *0x402660(,%rax,8)
0x0000000000400ff1 <+64>:     mov    $0x2a6,%eax
0x0000000000400ff6 <+69>:     jmp    0x401033 <phase_3+130>
0x0000000000400ff8 <+71>:     mov    $0x337,%eax
0x0000000000400ffd <+76>:     jmp    0x401033 <phase_3+130>
0x0000000000400fff <+78>:     mov    $0x338,%eax
0x0000000000401004 <+83>:     jmp    0x401033 <phase_3+130>
0x0000000000401006 <+85>:     mov    $0x39c,%eax
0x000000000040100b <+90>:     jmp    0x401033 <phase_3+130>
0x000000000040100d <+92>:     mov    $0xdd,%eax
0x0000000000401012 <+97>:     jmp    0x401033 <phase_3+130>
0x0000000000401014 <+99>:     mov    $0x35b,%eax
0x0000000000401019 <+104>:    jmp    0x401033 <phase_3+130>
0x000000000040101b <+106>:    mov    $0x14a,%eax
0x0000000000401020 <+111>:    jmp    0x401033 <phase_3+130>
0x0000000000401022 <+113>:    callq 0x401650 <explode_bomb>
0x0000000000401027 <+118>:    mov    $0x0,%eax
0x000000000040102c <+123>:    jmp    0x401033 <phase_3+130>
0x000000000040102e <+125>:    mov    $0x28a,%eax
0x0000000000401033 <+130>:    cmp    0x4(%rsp),%eax
---Type <return> to continue, or q <return> to quit---return
0x0000000000401037 <+134>:    je     0x40103e <phase_3+141>
0x0000000000401039 <+136>:    callq 0x401650 <explode_bomb>
0x000000000040103e <+141>:    mov    0x8(%rsp),%rax
0x0000000000401043 <+146>:    xor    %fs:0x28,%rax
0x000000000040104c <+155>:    je     0x401053 <phase_3+162>
0x000000000040104e <+157>:    callq 0x400b90 <__stack_chk_fail@plt>
0x0000000000401053 <+162>:    add    $0x18,%rsp
0x0000000000401057 <+166>:    retq
```

End of assembler dump.

(gdb) 

(gdb) x/s 0x40292d

0x40292d: "%d %d"

먼저 +33에서 scanf가 있는 것을 보니 값을 입력받는다는 것을 알 수 있다. 그리고 x/s 명령어를 통해 값을 2개 받을 것이라고 유추할 수도 있었다. 또한 +38과 +48을 통해 switch문 입력값은 1~7이라는 것을 유추할 수 있었다. 코드를 보면 +64부터 +106까지 값을 넣고 분기시키는 것을 반복하는 것을 볼 수 있다. 이는 switch/case문을 나타낸 것이다. 그리고 이 구문이 해당 case문 입력 값과 그에 따른 결과값이 내가 입력한 두 개의 입력 값과 동일해야 폭탄을 피할수 있었다. 그래서 switch case문 부분의 값들을 x/d명령어로 10진수로 변환해보았다. case가 1일 때는 0x2a6, 2일 때는 0x377, 3일 때는 0x39c .... 7일 때는 0x14a의 값을 알 수 있다. 따라서 경우의 수 중에서 1일 때 0x2a6임을 10진수로 변환해서 1과 670을 입력값을 넣었더니 답이 맞았다.

---

Phase 3 [정답]

---

1 678

```
(gdb) disas phase_4
Dump of assembler code for function phase_4:
0x0000000000401096 <+0>:      sub     $0x18,%rsp
0x000000000040109a <+4>:      mov     %fs:0x28,%rax
0x00000000004010a3 <+13>:     mov     %rax,0x8(%rsp)
0x00000000004010a8 <+18>:     xor     %eax,%eax
0x00000000004010aa <+20>:     lea     0x4(%rsp),%rcx
0x00000000004010af <+25>:     mov     %rsp,%rdx
0x00000000004010b2 <+28>:     mov     $0x40292d,%esi
0x00000000004010b7 <+33>:     callq   0x400c40 <__isoc99_sscanf@plt>
0x00000000004010bc <+38>:     cmp     $0x2,%eax
0x00000000004010bf <+41>:     jne     0x4010c7 <phase_4+49>
0x00000000004010c1 <+43>:     cmpl    $0xe, (%rsp)
0x00000000004010c5 <+47>:     jbe     0x4010cc <phase_4+54>
0x00000000004010c7 <+49>:     callq   0x401650 <explode_bomb>
0x00000000004010cc <+54>:     mov     $0xe,%edx
0x00000000004010d1 <+59>:     mov     $0x0,%esi
0x00000000004010d6 <+64>:     mov     (%rsp),%edi
0x00000000004010d9 <+67>:     callq   0x401058 <func4>
0x00000000004010de <+72>:     cmp     $0x7,%eax
0x00000000004010e1 <+75>:     jne     0x4010ea <phase_4+84>
0x00000000004010e3 <+77>:     cmpl    $0x7,0x4(%rsp)
0x00000000004010e8 <+82>:     je      0x4010ef <phase_4+89>
0x00000000004010ea <+84>:     callq   0x401650 <explode_bomb>
---Type <return> to continue, or q <return> to quit---return
0x00000000004010ef <+89>:     mov     0x8(%rsp),%rax
0x00000000004010f4 <+94>:     xor     %fs:0x28,%rax
0x00000000004010fd <+103>:    je      0x401104 <phase_4+110>
0x00000000004010ff <+105>:    callq   0x400b90 <__stack_chk_fail@plt>
0x0000000000401104 <+110>:    add     $0x18,%rsp
0x0000000000401108 <+114>:    retq
End of assembler dump.
(gdb)
```

```
0x00000000004010a3 <+13>:     mov     %rax,0x8(%rsp)
0x00000000004010a8 <+18>:     xor     %eax,%eax
0x00000000004010aa <+20>:     lea     0x4(%rsp),%rcx
0x00000000004010af <+25>:     mov     %rsp,%rdx
0x00000000004010b2 <+28>:     mov     $0x40292d,%esi
0x00000000004010b7 <+33>:     callq   0x400c40 <__isoc99_sscanf@plt>
0x00000000004010bc <+38>:     cmp     $0x2,%eax

(gdb) x/s 0x40292d
0x40292d:      "%d %d"
```

위 사진을 보고 일단 답 입력을 int 값을 2개한다는 것을 알 수 있었다.



```

0x00000000004010c1 <+43>:    cmpl    $0xe, (%rsp)
0x00000000004010c5 <+47>:    jbe     0x4010cc <phase_4+54>
0x00000000004010c7 <+49>:    callq   0x401650 <explode_bomb>
0x00000000004010cc <+54>:    mov     $0xe, %edx

```

위 코드 부분에서 0xe는 14인데 14보다 만약 큰 수를 입력하면 폭탄을 만나게됨을 알 수 있었다. 따라서 첫 번째 값은 14보다 클 것이라는 것을 알 수 있었다.

```

0x00000000004010de <+72>:    cmp     $0x7, %eax
0x00000000004010e1 <+75>:    jne     0x4010ea <phase_4+84>
0x00000000004010e3 <+77>:    cmpl    $0x7, 0x4(%rsp)
0x00000000004010e8 <+82>:    je      0x4010ef <phase_4+89>
0x00000000004010ea <+84>:    callq   0x401650 <explode_bomb>

```

이어서 이 코드부분에서는 7과 비교해서 7과 같지 않으면 폭탄을 만나게됨을 볼 수 있었다.

따라서 답을 14 7로 했더니 정답임을 유추할 수 있었다.

```

0x0000000000401081 <+41>:    cmp     %edi, %ecx
0x0000000000401083 <+43>:    jge     0x401091 <func4+57>
0x0000000000401085 <+45>:    lea     0x1(%rcx), %esi
0x0000000000401088 <+48>:    callq   0x401058 <func4>
0x000000000040108d <+53>:    lea     0x1(%rax, %rax, 1), %eax
0x0000000000401091 <+57>:    add     $0x8, %rsp

```

또한 func4를 disassemble해봤을 때 다음과 같이 edi값과 ecx값을 비교하여 조건에 따라 재귀함수를 도는 형태를 발견했다. 모든 재귀함수를 끝낸 후 0x1(%rax,%rax,1)을 보게 되면  $0+0+1=1$ ,  $1+1+1=3$ ,  $3+3+1=7$  이 되어 func4함수의 결과가 7이 되게 됩니다. 이 때의 edi 값이 14가 되어 답을 14로 유추할 수 있게 되었습니다. func4 함수를 분석해 보면 차례대로 1씩 늘어나고 나 중에 1씩 줄어들면서 재귀함수를 돌아간 횟수를 카운트 하게 되는 것을 발견하였고 그 때마다 %eax 값에 주목하여 답을 유추해 나갔다.

Phase 4 [정답]

14 7

```

c201404377@2018-sp: ~/bomb16
Dump of assembler code for function phase_5:
0x0000000000401109 <+0>:    push    %rbx
0x000000000040110a <+1>:    mov     %rdi,%rbx
0x000000000040110d <+4>:    callq   0x40135e <string_length>
0x0000000000401112 <+9>:    cmp     $0x6,%eax
0x0000000000401115 <+12>:   je      0x40111c <phase_5+19>
0x0000000000401117 <+14>:   callq   0x401650 <explode_bomb>
0x000000000040111c <+19>:   mov     %rbx,%rax
0x000000000040111f <+22>:   lea     0x6(%rbx),%rdi
0x0000000000401123 <+26>:   mov     $0x0,%ecx
0x0000000000401128 <+31>:   movzbl  (%rax),%edx
0x000000000040112b <+34>:   and     $0xf,%edx
0x000000000040112e <+37>:   add     0x4026a0(,%rdx,4),%ecx
0x0000000000401135 <+44>:   add     $0x1,%rax
0x0000000000401139 <+48>:   cmp     %rdi,%rax
0x000000000040113c <+51>:   jne     0x401128 <phase_5+31>
0x000000000040113e <+53>:   cmp     $0x3e,%ecx
0x0000000000401141 <+56>:   je      0x401148 <phase_5+63>
0x0000000000401143 <+58>:   callq   0x401650 <explode_bomb>
0x0000000000401148 <+63>:   pop     %rbx
0x0000000000401149 <+64>:   retq
End of assembler dump.

```

phase\_5에서는 rbx레지스터에 값을 저장한 후 입력받은 값의 길이를 재는 string\_length함수를 사용하여 6개인지 확인한다. 입력받은 값의 길이가 6이라면 반복문을 이용하여 입력받은 값들에 따른 0x4026c0번지에 들어있는 값들을 6번 모두 더한 후 0x2b와 같은지 확인한다.

```

(gdb) x/16wx 0x4026a0
0x4026a0 <array.3601>: 0x00000002      0x0000000a      0x00000006      0x00000000
01
0x4026b0 <array.3601+16>: 0x0000000c      0x00000010      0x00000009      0
x00000003
0x4026c0 <array.3601+32>: 0x00000004      0x00000007      0x0000000e      0
x00000005
0x4026d0 <array.3601+48>: 0x0000000b      0x00000008      0x0000000f      0
x0000000d

```

위 코드는 x/16wx명령을 이용하여 확인해본 0x4026c0번지의 값들이다. 저것들을 10진수로 하면 2 10 6 1 12 10 9 3 4 7 14 5 11 8 15 13이다. 이것들이 차례대로 0부터 13까지 들어가있다.


```

0x000000000040113e <+53>:   cmp     $0x3e,%ecx
0x0000000000401141 <+56>:   je      0x401148 <phase_5+63>
0x0000000000401143 <+58>:   callq   0x401650 <explode_bomb>
0x0000000000401148 <+63>:   pop     %rbx

```

이 14개의 숫자를 사용해서 6자리로 0x3e 즉 62를 만들어야 한다. 그래서  $10 \times 5 + 12 = 62$ 이므로 답은 111114임을 알 수 있었다.

111114

 c201404377@2018-sp: ~/bomb16

```

x00000000d
(gdb) disas phase_6
Dump of assembler code for function phase_6:
0x000000000040114a <+0>:      push    %r13
0x000000000040114c <+2>:      push    %r12
0x000000000040114e <+4>:      push    %rbp
0x000000000040114f <+5>:      push    %rbx
0x0000000000401150 <+6>:      sub     $0x68,%rsp
0x0000000000401154 <+10>:     mov     %fs:0x28,%rax
0x000000000040115d <+19>:     mov     %rax,0x58(%rsp)
0x0000000000401162 <+24>:     xor     %eax,%eax
0x0000000000401164 <+26>:     mov     %rsp,%rsi
0x0000000000401167 <+29>:     callq  0x401686 <read_six_numbers>
0x000000000040116c <+34>:     mov     %rsp,%r12
0x000000000040116f <+37>:     mov     $0x0,%r13d
0x0000000000401175 <+43>:     mov     %r12,%rbp
0x0000000000401178 <+46>:     mov     (%r12),%eax
0x000000000040117c <+50>:     sub     $0x1,%eax
0x000000000040117f <+53>:     cmp     $0x5,%eax
0x0000000000401182 <+56>:     jbe     0x401189 <phase_6+63>
0x0000000000401184 <+58>:     callq  0x401650 <explode_bomb>
0x0000000000401189 <+63>:     add     $0x1,%r13d
0x000000000040118d <+67>:     cmp     $0x6,%r13d
0x0000000000401191 <+71>:     je      0x4011d0 <phase_6+134>
0x0000000000401193 <+73>:     mov     %r13d,%ebx
0x0000000000401196 <+76>:     movslq  %ebx,%rax
0x0000000000401199 <+79>:     mov     (%rsp,%rax,4),%eax
0x000000000040119c <+82>:     cmp     %eax,0x0(%rbp)
0x000000000040119f <+85>:     jne     0x4011a6 <phase_6+92>
0x00000000004011a1 <+87>:     callq  0x401650 <explode_bomb>
0x00000000004011a6 <+92>:     add     $0x1,%ebx
0x00000000004011a9 <+95>:     cmp     $0x5,%ebx
0x00000000004011ac <+98>:     jle     0x401196 <phase_6+76>
0x00000000004011ae <+100>:    add     $0x4,%r12
0x00000000004011b2 <+104>:    jmp     0x401175 <phase_6+43>
0x00000000004011b4 <+106>:    mov     0x8(%rdx),%rdx
0x00000000004011b8 <+110>:    add     $0x1,%eax
0x00000000004011bb <+113>:    cmp     %ecx,%eax
0x00000000004011bd <+115>:    jne     0x4011b4 <phase_6+106>
0x00000000004011bf <+117>:    mov     %rdx,0x20(%rsp,%rsi,2)
0x00000000004011c4 <+122>:    add     $0x4,%rsi
0x00000000004011c8 <+126>:    cmp     $0x18,%rsi
0x00000000004011cc <+130>:    jne     0x4011d5 <phase_6+139>

```



```

0x00000000004011ce <+132>:    jmp     0x4011e9 <phase_6+159>
---Type <return> to continue, or q <return> to quit---return
0x00000000004011d0 <+134>:    mov     $0x0,%esi
0x00000000004011d5 <+139>:    mov     (%rsp,%rsi,1),%ecx
0x00000000004011d8 <+142>:    mov     $0x1,%eax
0x00000000004011dd <+147>:    mov     $0x6042f0,%edx
0x00000000004011e2 <+152>:    cmp     $0x1,%ecx
0x00000000004011e5 <+155>:    jg      0x4011b4 <phase_6+106>
0x00000000004011e7 <+157>:    jmp     0x4011bf <phase_6+117>
0x00000000004011e9 <+159>:    mov     0x20(%rsp),%rbx
0x00000000004011ee <+164>:    lea     0x20(%rsp),%rax
0x00000000004011f3 <+169>:    lea     0x48(%rsp),%rsi
0x00000000004011f8 <+174>:    mov     %rbx,%rcx
0x00000000004011fb <+177>:    mov     0x8(%rax),%rdx
0x00000000004011ff <+181>:    mov     %rdx,0x8(%rcx)
0x0000000000401203 <+185>:    add     $0x8,%rax
0x0000000000401207 <+189>:    mov     %rdx,%rcx
0x000000000040120a <+192>:    cmp     %rsi,%rax
0x000000000040120d <+195>:    jne     0x4011fb <phase_6+177>
0x000000000040120f <+197>:    movq    $0x0,0x8(%rdx)
0x0000000000401217 <+205>:    mov     $0x5,%ebp
0x000000000040121c <+210>:    mov     0x8(%rbx),%rax
0x0000000000401220 <+214>:    mov     (%rax),%eax
0x0000000000401222 <+216>:    cmp     %eax,(%rbx)
0x0000000000401224 <+218>:    jle     0x40122b <phase_6+225>
0x0000000000401226 <+220>:    callq   0x401650 <explode_bomb>
0x000000000040122b <+225>:    mov     0x8(%rbx),%rbx
0x000000000040122f <+229>:    sub     $0x1,%ebp
0x0000000000401232 <+232>:    jne     0x40121c <phase_6+210>
0x0000000000401234 <+234>:    mov     0x58(%rsp),%rax
0x0000000000401239 <+239>:    xor     %fs:0x28,%rax
0x0000000000401242 <+248>:    je      0x401249 <phase_6+255>
0x0000000000401244 <+250>:    callq   0x400b90 <__stack_chk_fail@plt>
0x0000000000401249 <+255>:    add     $0x68,%rsp
0x000000000040124d <+259>:    pop     %rbx
0x000000000040124e <+260>:    pop     %rbp
0x000000000040124f <+261>:    pop     %r12
0x0000000000401251 <+263>:    pop     %r13
0x0000000000401253 <+265>:    retq
End of assembler dump.
(gdb) █

```

먼저 스택을 할당 해주고 read\_six\_numbers 함수를 호출하는 것을 통해 6개의 숫자를 읽는 다는 것을 알 수 있었다.

```

0x000000000040116c <+34>:    mov     %rsp,%r12
0x000000000040116f <+37>:    mov     $0x0,%r13d
0x0000000000401175 <+43>:    mov     %r12,%rbp
0x0000000000401178 <+46>:    mov     (%r12),%eax
0x000000000040117c <+50>:    sub     $0x1,%eax
0x000000000040117f <+53>:    cmp     $0x5,%eax
0x0000000000401182 <+56>:    jbe     0x401189 <phase_6+63>
0x0000000000401184 <+58>:    callq   0x401650 <explode_bomb>
0x0000000000401189 <+63>:    add     $0x1,%r13d

```

또한 위 코드를 통해 6개의 정수를 입력하되 1부터 6까지가 되어야 함을 알 수 있었다.

```

0x0000000000401193 <+73>:    mov     %r13d,%ebx
0x0000000000401196 <+76>:    movslq %ebx,%rax
0x0000000000401199 <+79>:    mov     (%rsp,%rax,4),%eax
0x000000000040119c <+82>:    cmp     %eax,0x0(%rbp)
0x000000000040119f <+85>:    jne     0x4011a6 <phase_6+92>
0x00000000004011a1 <+87>:    callq   0x401650 <explode_bomb>
0x00000000004011a6 <+92>:    add     $0x1,%ebx

```

또한 위 코드를 통해 입력된 값들이 중복 될 경우 폭탄이 터지게 되는 것을 알 수 있었다. 따라서 1부터 6까지 중복 되지 않는 값들이 입력되어야 함을 알 수 있었다.

```

0x00000000004011bf <+117>:    mov     %rdx,0x20(%rsp,%rsi,2)
0x00000000004011c4 <+122>:    add     $0x4,%rsi
0x00000000004011c8 <+126>:    cmp     $0x18,%rsi
0x00000000004011cc <+130>:    jne     0x4011d5 <phase_6+139>
0x00000000004011ce <+132>:    jmp     0x4011e9 <phase_6+159>
---Type <return> to continue, or q <return> to quit---return
0x00000000004011d0 <+134>:    mov     $0x0,%esi
0x00000000004011d5 <+139>:    mov     (%rsp,%rsi,1),%ecx
0x00000000004011d8 <+142>:    mov     $0x1,%eax
0x00000000004011dd <+147>:    mov     $0x6042f0,%edx
0x00000000004011e2 <+152>:    cmp     $0x1,%ecx
0x00000000004011e5 <+155>:    jg      0x4011b4 <phase_6+106>
0x00000000004011e7 <+157>:    jmp     0x4011bf <phase_6+117>

```

위 코드에서는 반복문의 형태가 보인다. ecx가 1보다 작거나 같을 때는 바로 0x6042f0 값을 %edx에 저장하는 것도 알 수 있다. 또한 상황에 따라 0x6042f0의 값을 넣어주고 있는 것을 확인하여 0x6042f0에 무슨 값이 들어 있는지 확인했다.

```

(gdb) x/24w 0x6042f0
0x6042f0 <node1>:    0x0000013f    0x00000001    0x00604300    0x00000000
00
0x604300 <node2>:    0x000003cc    0x00000002    0x00604310    0x00000000
00
0x604310 <node3>:    0x0000004c    0x00000003    0x00604320    0x00000000
00
0x604320 <node4>:    0x00000339    0x00000004    0x00604330    0x00000000
00
0x604330 <node5>:    0x0000006b    0x00000005    0x00604340    0x00000000
00
0x604340 <node6>:    0x000001c1    0x00000006    0x00000000    0x00000000
00

```

x/24w 명령어를 통해 확인한 결과 다음과 같았다. 노드가 6개 나오는데 답도 6개였으므로 이것이 답일 확률이 높다고 확신하였다.



```

0x00000000000040121c <+210>:  mov    0x8(%rbx),%rax
0x000000000000401220 <+214>:  mov    (%rax),%eax
0x000000000000401222 <+216>:  cmp    %eax,(%rbx)
0x000000000000401224 <+218>:  jle     0x40122b <phase_6+225>
0x000000000000401226 <+220>:  callq  0x401650 <explode_bomb>
0x00000000000040122b <+225>:  mov    0x8(%rbx),%rbx
0x00000000000040122f <+229>:  sub    $0x1,%ebp
0x000000000000401232 <+232>:  jne     0x40121c <phase_6+210>
0x000000000000401234 <+234>:  mov    0x58(%rsp),%rax

```

위 코드에서는 입력한 값의 노드 값이 뒤에 노드보다 클 경우 폭탄이 터지게 되는 것을 알았다. 이것을 보고 입력 값들이 오름차순으로 입력되는 것을 알 수 있었습니다. 따라서 오름차순대로 노드 값을 비교해보면 노드의 순서는 3 5 1 6 4 2임을 알 수 있었다.

---

**Phase 6**    [정답]

---

3 5 1 6 4 2

Phase	[진행 과정 설명]
Secret	

```

0x0000000000400e95 <+159>:  callq 0x4017eb <phase_defused>
0x0000000000400e9a <+164>:  mov    $0x4025b8,%edi
0x0000000000400e9f <+169>:  callq 0x400b70 <puts@plt>
0x0000000000400ea4 <+174>:  callq 0x4016c5 <read_line>
0x0000000000400ea9 <+179>:  mov    %rax,%rdi
---Type <return> to continue, or q <return> to quit---return
0x0000000000400eac <+182>:  callq 0x400f49 <phase_2>
0x0000000000400eb1 <+187>:  callq 0x4017eb <phase_defused>
0x0000000000400eb6 <+192>:  mov    $0x4024fd,%edi
0x0000000000400ebb <+197>:  callq 0x400b70 <puts@plt>
0x0000000000400ec0 <+202>:  callq 0x4016c5 <read_line>
0x0000000000400ec5 <+207>:  mov    %rax,%rdi
0x0000000000400ec8 <+210>:  callq 0x400fb1 <phase_3>
0x0000000000400ecd <+215>:  callq 0x4017eb <phase_defused>
0x0000000000400ed2 <+220>:  mov    $0x40251b,%edi
0x0000000000400ed7 <+225>:  callq 0x400b70 <puts@plt>
0x0000000000400edc <+230>:  callq 0x4016c5 <read_line>
0x0000000000400ee1 <+235>:  mov    %rax,%rdi
0x0000000000400ee4 <+238>:  callq 0x401096 <phase_4>
0x0000000000400ee9 <+243>:  callq 0x4017eb <phase_defused>
0x0000000000400eee <+248>:  mov    $0x4025e8,%edi
0x0000000000400ef3 <+253>:  callq 0x400b70 <puts@plt>
0x0000000000400ef8 <+258>:  callq 0x4016c5 <read_line>
0x0000000000400efd <+263>:  mov    %rax,%rdi
0x0000000000400f00 <+266>:  callq 0x401109 <phase_5>
0x0000000000400f05 <+271>:  callq 0x4017eb <phase_defused>
0x0000000000400f0a <+276>:  mov    $0x40252a,%edi
0x0000000000400f0f <+281>:  callq 0x400b70 <puts@plt>
0x0000000000400f14 <+286>:  callq 0x4016c5 <read_line>
---Type <return> to continue, or q <return> to quit---return
0x0000000000400f19 <+291>:  mov    %rax,%rdi
0x0000000000400f1c <+294>:  callq 0x40114a <phase_6>
0x0000000000400f21 <+299>:  callq 0x4017eb <phase_defused>
0x0000000000400f26 <+304>:  mov    $0x0,%eax
0x0000000000400f2b <+309>:  pop    %rbx
0x0000000000400f2c <+310>:  retq
End of assembler dump.
(gdb) █

```

위 코드는 main을 disassemble한 코드의 일부분인데 phase번호 함수들을 불러오는 중간에 phase\_defused라는 함수들이 나온다. 이 phase\_defused함수를 disassemble해보면 밑과 같다.

```

c201404377@2018-sp: ~/bomb16
---Type <return> to continue, or q <return> to quit---reutrn
0x000000000040184b <+96>:    jne     0x40186b <phase_defused+128>
0x000000000040184d <+98>:    mov     $0x4027d8,%edi
0x0000000000401852 <+103>:   callq   0x400b70 <puts@plt>
0x0000000000401857 <+108>:   mov     $0x402800,%edi
0x000000000040185c <+113>:   callq   0x400b70 <puts@plt>
0x0000000000401861 <+118>:   mov     $0x0,%eax
0x0000000000401866 <+123>:   callq   0x401292 <secret_phase>
0x000000000040186b <+128>:   mov     $0x402838,%edi
0x0000000000401870 <+133>:   callq   0x400b70 <puts@plt>
0x0000000000401875 <+138>:   mov     $0x402868,%edi
0x000000000040187a <+143>:   callq   0x400b70 <puts@plt>
0x000000000040187f <+148>:   mov     0x68(%rsp),%rax
0x0000000000401884 <+153>:   xor     %fs:0x28,%rax
0x000000000040188d <+162>:   je      0x401894 <phase_defused+169>
0x000000000040188f <+164>:   callq   0x400b90 <__stack_chk_fail@plt>
0x0000000000401894 <+169>:   add     $0x78,%rsp
0x0000000000401898 <+173>:   retq
End of assembler dump.

c201404377@2018-sp: ~/bomb16
(gdb) disas phase_defused
Dump of assembler code for function phase_defused:
0x00000000004017eb <+0>:    sub     $0x78,%rsp
0x00000000004017ef <+4>:    mov     %fs:0x28,%rax
0x00000000004017f8 <+13>:   mov     %rax,0x68(%rsp)
0x00000000004017fd <+18>:   xor     %eax,%eax
0x00000000004017ff <+20>:   mov     $0x1,%edi
0x0000000000401804 <+25>:   callq   0x401546 <send_msg>
0x0000000000401809 <+30>:   cmpl    $0x6,0x202f9c(%rip)          # 0x6047ac <nu
m_input_strings>
0x0000000000401810 <+37>:   jne     0x40187f <phase_defused+148>
0x0000000000401812 <+39>:   lea     0x10(%rsp),%r8
0x0000000000401817 <+44>:   lea     0xc(%rsp),%rcx
0x000000000040181c <+49>:   lea     0x8(%rsp),%rdx
0x0000000000401821 <+54>:   mov     $0x402977,%esi
0x0000000000401826 <+59>:   mov     $0x6048b0,%edi
0x000000000040182b <+64>:   mov     $0x0,%eax
0x0000000000401830 <+69>:   callq   0x400c40 <__isoc99_sscanf@plt>
0x0000000000401835 <+74>:   cmp     $0x3,%eax
0x0000000000401838 <+77>:   jne     0x40186b <phase_defused+128>
0x000000000040183a <+79>:   mov     $0x402980,%esi
0x000000000040183f <+84>:   lea     0x10(%rsp),%rdi
0x0000000000401844 <+89>:   callq   0x40137c <strings_not_equal>
0x0000000000401849 <+94>:   test    %eax,%eax

```

가장 먼저 +123에 보면 secret phase를 불러오는 것을 볼 수 있다. 그리고 다른 주목해야 할 부분은 phase\_defused+79에서 볼 수 있는 0x4029bf이다. 이 주소에 저장되어 있는 값을 esi에 옮겨서 비교하게 되는데 x/s명령으로 이를 확인 해 보면 다음과 같다. 그리고 이것을 phase4 답을 적을 때 같이 써주면 된다.

```

(gdb) x/s 0x402980
0x402980:      "DrEvil"

```



```

c201404377@2018-sp: ~/bomb16
(gdb) disas secret_phase
Dump of assembler code for function secret_phase:
0x0000000000401292 <+0>:      push    %rbx
0x0000000000401293 <+1>:      callq   0x4016c5 <read_line>
0x0000000000401298 <+6>:      mov     $0xa,%edx
0x000000000040129d <+11>:     mov     $0x0,%esi
0x00000000004012a2 <+16>:     mov     %rax,%rdi
0x00000000004012a5 <+19>:     callq   0x400c20 <strtol@plt>
0x00000000004012aa <+24>:     mov     %rax,%rbx
0x00000000004012ad <+27>:     lea     -0x1(%rax),%eax
0x00000000004012b0 <+30>:     cmp     $0x3e8,%eax
0x00000000004012b5 <+35>:     jbe     0x4012bc <secret_phase+42>
0x00000000004012b7 <+37>:     callq   0x401650 <explode_bomb>
0x00000000004012bc <+42>:     mov     %ebx,%esi
0x00000000004012be <+44>:     mov     $0x604110,%edi
0x00000000004012c3 <+49>:     callq   0x401254 <fun7>
0x00000000004012c8 <+54>:     cmp     $0x3,%eax
0x00000000004012cb <+57>:     je      0x4012d2 <secret_phase+64>
0x00000000004012cd <+59>:     callq   0x401650 <explode_bomb>
0x00000000004012d2 <+64>:     mov     $0x402638,%edi
0x00000000004012d7 <+69>:     callq   0x400b70 <puts@plt>
0x00000000004012dc <+74>:     callq   0x4017eb <phase_defused>
0x00000000004012e1 <+79>:     pop     %rbx
0x00000000004012e2 <+80>:     retq
---Type <return> to continue, or q <return> to quit---return
End of assembler dump.
(gdb)

```

위 코드는 secret\_phase를 disassemble한 결과이다. 보면 edi에 0x604110을 넣은 후 fun7을 부르는 것을 보인다. 그래서 0x604110값을 x/120wx 명령어로 출력해봤다. 그 출력결과를 밑 사진과 같다.

```

c201404377@2018-sp: ~/bomb16
0x00000000004012e2 <+80>:      retq
---Type <return> to continue, or q <return> to quit---reutrn
End of assembler dump.
(gdb) x/120wx 0x604110
0x604110 <n1>: 0x00000024      0x00000000      0x00604130      0x00000000
0x604120 <n1+16>: 0x00604150      0x00000000      0x00000000      0x00000000
00
0x604130 <n21>: 0x00000008      0x00000000      0x006041b0      0x00000000
0x604140 <n21+16>: 0x00604170      0x00000000      0x00000000      0x00000000
00
0x604150 <n22>: 0x00000032      0x00000000      0x00604190      0x00000000
0x604160 <n22+16>: 0x006041d0      0x00000000      0x00000000      0x00000000
00
0x604170 <n32>: 0x00000016      0x00000000      0x00604290      0x00000000
0x604180 <n32+16>: 0x00604250      0x00000000      0x00000000      0x00000000
00
0x604190 <n33>: 0x0000002d      0x00000000      0x006041f0      0x00000000
0x6041a0 <n33+16>: 0x006042b0      0x00000000      0x00000000      0x00000000
00
0x6041b0 <n31>: 0x00000006      0x00000000      0x00604210      0x00000000
0x6041c0 <n31+16>: 0x00604270      0x00000000      0x00000000      0x00000000
00
0x6041d0 <n34>: 0x0000006b      0x00000000      0x00604230      0x00000000
0x6041e0 <n34+16>: 0x006042d0      0x00000000      0x00000000      0x00000000

```

이진 탐색 트리로 값이 들어있음을 볼 수 있다. 이 값들의 10진수를 출력해 본 결과는 밑과 같다.

```
(gdb) x/d 0x604110
0x604110 <n1>: 36
(gdb) x/d 0x00604130
0x604130 <n21>: 8
(gdb) x/d 0x00000024
0x24: Cannot access memory at address 0x24
(gdb) x/d 0x00604150
0x604150 <n22>: 50
(gdb) x/d 0x006041b0
0x6041b0 <n31>: 6
(gdb) x/d 0x00604170
0x604170 <n32>: 22
(gdb) x/d 0x00604190
0x604190 <n33>: 45
(gdb) x/d 0x006041d0
0x6041d0 <n34>: 107
(gdb) x/d 0x00604290
0x604290 <n43>: 20
(gdb) x/d 0x00604250
0x604250 <n44>: 35
(gdb) x/d 0x006041f0
0x6041f0 <n45>: 40
(gdb) x/d 0x006042b0
0x6042b0 <n46>: 47
(gdb) x/d 0x00604210
0x604210 <n41>: 1
(gdb) x/d 0x00604270
0x604270 <n42>: 7
(gdb) x/d 0x00604230
0x604230 <n47>: 99
(gdb) x/d 0x006042d0
0x6042d0 <n48>: 1001
```

이것을 트리구조로 나타내면

```

          36
        /  \
       8    50
      / \  / \
     6  22 45 107
    / \ / \ / \
   1  7 20 35 40 47 99 1001

```

이런식으로 된다.

이전 코드를 보면 1000보다 작거나 같은 수가 필요해보이고 반환값은 3이어야 한다는 것을 볼 수 있다. 그래서 3번째 높이에 있고 1000하고 가장 가까운 쪽에 있는 107을 답으로 했더니 정답이되었다.

Phase	
Secret	[정답]

107