2019 시스템 프로그래밍
- Bomb Lab -

제출일자	2019.10.28
분 반	01
이 름	임준규
학 번	201602057

Phase 1	[결과 화면 캡처]

```
which to blow yourself up. Have a nice day!
There are rumors on the internets.
Phase 1 defused. How about the next one?
```

Phase 1	[진행 과정 설명]

```
0x400f2d <phase_1>       sub    $0x8,%rsp
0x400f31 <phase_1+4>     mov    $0x402710,%esi
0x400f36 <phase_1+9>     callq  0x401487 <strings_not_equal>
0x400f3b <phase_1+14>    test   %eax,%eax
0x400f3d <phase_1+16>    je     0x400f44 <phase_1+23>
0x400f3f <phase_1+18>    callq  0x40175b <explode_bomb>
0x400f44 <phase_1+23>    add    $0x8,%rsp
0x400f48 <phase_1+27>    retq
0x400f49 <phase_2>       push   %rbp
0x400f4a <phase_2+1>     push   %rbx
0x400f4b <phase_2+2>     sub    $0x28,%rsp
0x400f4f <phase_2+6>     mov    %fs:0x28,%rax
0x400f58 <phase_2+15>    mov    %rax,0x18(%rsp)
0x400f5d <phase_2+20>    xor    %eax,%eax
0x400f5f <phase_2+22>    mov    %rsp,%rsi
0x400f62 <phase_2+25>    callq  0x401791 <read_six_numbers>
0x400f67 <phase_2+30>    cmpl   $0x0,(%rsp)
0x400f6b <phase_2+34>    jne    0x400f74 <phase_2+43>
0x400f6d <phase_2+36>    cmpl   $0x1,0x4(%rsp)
0x400f72 <phase_2+41>    je     0x400f79 <phase_2+48>
0x400f74 <phase_2+43>    callq  0x40175b <explode_bomb>
0x400f79 <phase_2+48>    mov    %rsp,%rbx
0x400f7c <phase_2+51>    lea    0x10(%rsp),%rbp
0x400f81 <phase_2+56>    mov    0x4(%rbx),%eax
0x400f84 <phase_2+59>    add    (%rbx),%eax
0x400f86 <phase_2+61>    cmp    %eax,0x8(%rbx)
exec No process In:
(gdb) layout asm
(gdb) disas phase_1
(gdb) x/s 0x402710
0x402710:       "There are rumors on the internets."
```

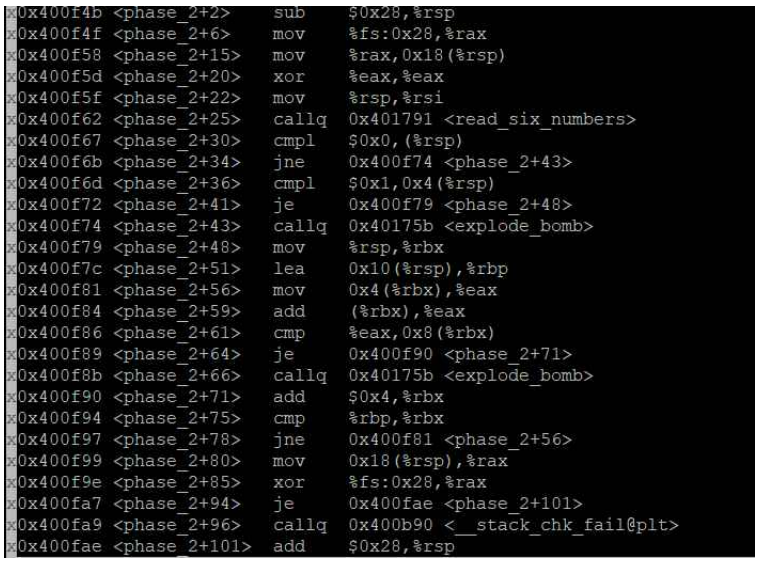
String_not_equal함수 전에 esi에 들어가는 주소값이 의심스러워 확인해보았다. 아래와 같이 확인해본 결과 There are rumors on the internets. 라는 문자열이 나왔고 그아래에 같은지 비교하고 같지 않으면 폭탄이 터지는 구조로 보아 정답은 There are rumors on the internets. 이다.

Phase 1	[정답]

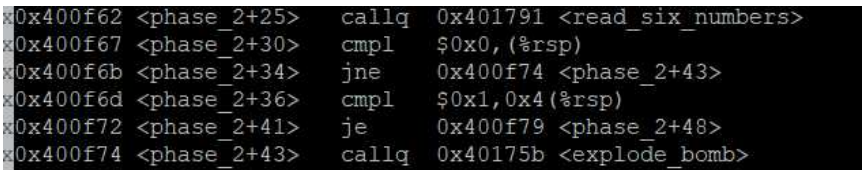
There are rumors on the internets.

Phase 2 [결과 화면 캡처]

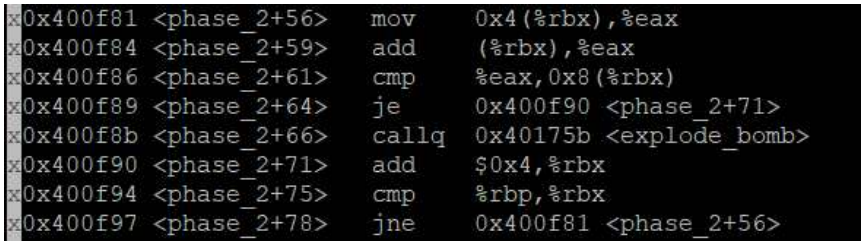
Phase 2 [진행 과정 설명]



전체 코드는 위와 같다. 처음부터 분석해보면 처음의 read_six_number함수가 보이는데, 이 함수는 6개의 수가 입력되었는 지 확인하고 아니면 폭탄을 터트려 버리는 무서운 함수이다. 함수명으로 힌트를 주었으니 시작할 때 꼭 6개의 입력을 주도록 하자.



입력값은 rsp에 주솟값 4 차이로 저장이 되는데, +30에서 첫 번째 입력이 0과 같지 않으면 폭탄으로 이동하고, +36에서 1과 같지 않으면 폭탄이 터진다.



이 다음에는 위와 같은 루프가 1개 있는데, 해석해보자.
우선 rbx(입력값 배열)의 다음 값을 eax에 저장해주고 eax의 이전 값을 더해서 저장해준다.
이후에, rbx의 다음다음값과 비교하여, 같지 않으면 폭탄이 터진다.
그리고 rbx의 주소를 4 더해서 다음 값을 가리키게 만들어준다. 그리고 이제 연산이 완료될 때 까지 루프를 돈다.
이때 우리는 이와 비슷한 것을 어디서 접한 적이 있다. 피보나치 수열의 첫 6자리인 것이다.
왜냐, 처음에 0,1을 확정적으로 받고 3번째부터는 그 전의 수와 그 전전의 수를 더한 것이 세 번째 값이 되므로 계속 확인할 필요없이 0 1 1 2 3 5를 입력하면 된다.

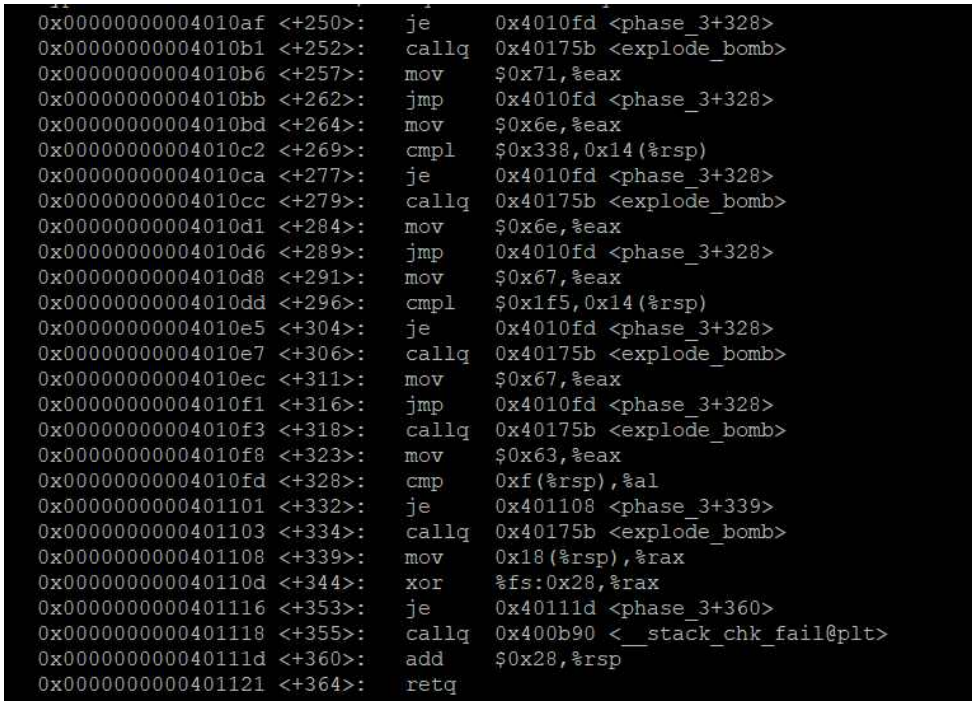
Phase 2 [정답]

0 1 1 2 3 5

Phase 3 [결과 화면 캡처]

Phase 3 [진행 과정 설명]

```
Dump of assembler code for function phase_3:
   0x0000000000400fb5 <+0>:     sub    $0x28,%rsp
   0x0000000000400fb9 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400fc2 <+13>:    mov    %rax,0x18(%rsp)
   0x0000000000400fc7 <+18>:    xor    %eax,%eax
   0x0000000000400fc9 <+20>:    lea    0x14(%rsp),%r8
   0x0000000000400fce <+25>:    lea    0xf(%rsp),%rcx
   0x0000000000400fd3 <+30>:    lea    0x10(%rsp),%rdx
   0x0000000000400fd8 <+35>:    mov    $0x40275e,%esi
   0x0000000000400fdd <+40>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x0000000000400fe2 <+45>:    cmp    $0x2,%eax
   0x0000000000400fe5 <+48>:    jg     0x400fec <phase_3+55>
   0x0000000000400fe7 <+50>:    callq  0x40175b <explode_bomb>
   0x0000000000400fec <+55>:    cmpl   $0x7,0x10(%rsp)
   0x0000000000400ff1 <+60>:    ja     0x4010f3 <phase_3+318>
   0x0000000000400ff7 <+66>:    mov    0x10(%rsp),%eax
   0x0000000000400ffb <+70>:    jmpq   *0x402780(,%rax,8)
   0x0000000000401002 <+77>:    mov    $0x65,%eax
   0x0000000000401007 <+82>:    cmpl   $0x347,0x14(%rsp)
   0x000000000040100f <+90>:    je     0x4010fd <phase_3+328>
   0x0000000000401015 <+96>:    callq  0x40175b <explode_bomb>
   0x000000000040101a <+101>:   mov    $0x65,%eax
   0x000000000040101f <+106>:   jmpq   0x4010fd <phase_3+328>
   0x0000000000401024 <+111>:   mov    $0x66,%eax
   0x0000000000401029 <+116>:   cmpl   $0x275,0x14(%rsp)
   0x0000000000401031 <+124>:   je     0x4010fd <phase_3+328>
   0x0000000000401037 <+130>:   callq  0x40175b <explode_bomb>
   0x000000000040103c <+135>:   mov    $0x66,%eax
   0x0000000000401041 <+140>:   jmpq   0x4010fd <phase_3+328>
   0x0000000000401046 <+145>:   mov    $0x68,%eax
   0x000000000040104b <+150>:   cmpl   $0x30b,0x14(%rsp)
   0x0000000000401053 <+158>:   je     0x4010fd <phase_3+328>
   0x0000000000401059 <+164>:   callq  0x40175b <explode_bomb>
   0x000000000040105e <+169>:   mov    $0x68,%eax
   0x0000000000401063 <+174>:   jmpq   0x4010fd <phase_3+328>
   0x0000000000401068 <+179>:   mov    $0x73,%eax
   0x000000000040106d <+184>:   cmpl   $0x3b8,0x14(%rsp)
   0x0000000000401075 <+192>:   je     0x4010fd <phase_3+328>
   0x000000000040107b <+198>:   callq  0x40175b <explode_bomb>
   0x0000000000401080 <+203>:   mov    $0x73,%eax
   0x0000000000401085 <+208>:   jmp    0x4010fd <phase_3+328>
   0x0000000000401087 <+210>:   mov    $0x76,%eax
   0x000000000040108c <+215>:   cmpl   $0x39f,0x14(%rsp)
   0x0000000000401094 <+223>:   je     0x4010fd <phase_3+328>
   0x0000000000401096 <+225>:   callq  0x40175b <explode_bomb>
   0x000000000040109b <+230>:   mov    $0x76,%eax
   0x00000000004010a0 <+235>:   jmp    0x4010fd <phase_3+328>
   0x00000000004010a2 <+237>:   mov    $0x71,%eax
   0x00000000004010a7 <+242>:   cmpl   $0x3c7,0x14(%rsp)
```



```
0x00000000004010af <+250>:   je     0x4010fd <phase_3+328>
0x00000000004010b1 <+252>:   callq  0x40175b <explode_bomb>
0x00000000004010b6 <+257>:   mov    $0x71,%eax
0x00000000004010bb <+262>:   jmp    0x4010fd <phase_3+328>
0x00000000004010bd <+264>:   mov    $0x6e,%eax
0x00000000004010c2 <+269>:   cmpl   $0x338,0x14(%rsp)
0x00000000004010ca <+277>:   je     0x4010fd <phase_3+328>
0x00000000004010cc <+279>:   callq  0x40175b <explode_bomb>
0x00000000004010d1 <+284>:   mov    $0x6e,%eax
0x00000000004010d6 <+289>:   jmp    0x4010fd <phase_3+328>
0x00000000004010d8 <+291>:   mov    $0x67,%eax
0x00000000004010dd <+296>:   cmpl   $0x1f5,0x14(%rsp)
0x00000000004010e5 <+304>:   je     0x4010fd <phase_3+328>
0x00000000004010e7 <+306>:   callq  0x40175b <explode_bomb>
0x00000000004010ec <+311>:   mov    $0x67,%eax
0x00000000004010f1 <+316>:   jmp    0x4010fd <phase_3+328>
0x00000000004010f3 <+318>:   callq  0x40175b <explode_bomb>
0x00000000004010f8 <+323>:   mov    $0x63,%eax
0x00000000004010fd <+328>:   cmp    0xf(%rsp),%al
0x0000000000401101 <+332>:   je     0x401108 <phase_3+339>
0x0000000000401103 <+334>:   callq  0x40175b <explode_bomb>
0x0000000000401108 <+339>:   mov    0x18(%rsp),%rax
0x000000000040110d <+344>:   xor    %fs:0x28,%rax
0x0000000000401116 <+353>:   je     0x40111d <phase_3+360>
0x0000000000401118 <+355>:   callq  0x400b90 <__stack_chk_fail@plt>
0x000000000040111d <+360>:   add    $0x28,%rsp
0x0000000000401121 <+364>:   retq
```

페이즈 3의 전체 코드이다. 3페이즈에 입장하자마자 위협하는 364줄의 압박.... 하지만 풀이해본 결과 말이 364줄이지 확인하는 코드는 얼마 안 되는 것을 알 수 있었다. 자신감을 가지고 한번 가보자.

```
0x0000000000400fd8 <+35>:    mov    $0x40275e,%esi
0x0000000000400fdd <+40>:    callq  0x400c40 <__isoc99_sscanf@plt>
0x0000000000400fe2 <+45>:    cmp    $0x2,%eax
0x0000000000400fe5 <+48>:    jg     0x400fec <phase_3+55>
0x0000000000400fe7 <+50>:    callq  0x40175b <explode_bomb>
0x0000000000400fec <+55>:    cmpl   $0x7,0x10(%rsp)
0x0000000000400ff1 <+60>:    ja     0x4010f3 <phase_3+318>
```

이제 뭔가 의심스러운 친구가 보이는 것 같은 기분이 든다. 저 +35에 있는 친구를 확인해보자. 확인해봤더니 %d %c %d가 나타났다! 우리는 3개를 입력하고 자료형은 정수형 문자형 정수형을 입력해야함을 알 수 있다.

다음 코드를 보자. eax를 2와 비교해서 2보다 작으면 폭탄이 터진다. eax는 확인해본 결과 몇 개 입력했는지 개수가 들어가있었다. 우리는 이미 앞에서 3개를 입력해야 함을 알아냈으므로 걸릴리 없는 비교문이다. 가볍게 피해주자.

다음 코드로 넘어가자. 7보다 크면 +318로 간다한다. 어 멀리가네? 좋은건가보다 하고 따라가본 결과 폭탄이었다. 그럼 저 rsp+0x10은 뭘까? 조사해 본 결과 우리의 입력값은 %rsp의 +0xf에 두 번째 입력, +0x10에 첫 번째 입력, +0x14에 세 번째 입력이 들어가 있었다. 그렇다면 이 비교문은 첫 번째 입력 값이7보다 작으면 패스임을 알려준다.

```
0x400fec <phase_3+55>   cmpl   $0x7,0x10(%rsp)
0x400ff1 <phase_3+60>   ja     0x4010f3 <phase_3+318>
0x400ff7 <phase_3+66>   mov    0x10(%rsp),%eax
0x400ffb <phase_3+70>   jmpq   *0x402780(,%rax,8)
0x401002 <phase_3+77>   mov    $0x65,%eax
0x401007 <phase_3+82>   cmpl   $0x347,0x14(%rsp)
0x40100f <phase_3+90>   je     0x4010fd <phase_3+328>
0x401015 <phase_3+96>   callq  0x40175b <explode_bomb>
0x40101a <phase_3+101>  mov    $0x65,%eax
0x40101f <phase_3+106>  jmpq   0x4010fd <phase_3+328>
0x401024 <phase_3+111>  mov    $0x66,%eax
0x401029 <phase_3+116>  cmpl   $0x275,0x14(%rsp)
0x401031 <phase_3+124>  je     0x4010fd <phase_3+328>
0x401037 <phase_3+130>  callq  0x40175b <explode_bomb>
```

7보다 작은 값을 입력해주어 가볍게 패스해주고 다음 코드를 확인하자. 70의 연산을 통해 +111으로 이동한다. 이 아래는 첫 번째에 입력가능 한 값에 따라 이동하는 위치에서의 연산하는 코드의 반복으로, if else 조건문이라 볼 수 있겠다. 이 아래는 똑같은 코드이므로 패스하고 저거 하나만 분석해보자. 0x275와 %rsp+0x14와 같아야 폭탄을 피할 수 있다. 앗! 아까 본거같은 주소다. 위에서 세 번째 입력값으로 찾아뒀던 주소다. 그러므로 세 번째 입력값은 0x275 즉 629가 되어야 함을 알 수 있다.

```
0x4010fd <phase_3+328>  cmp    0xf(%rsp),%al
0x401101 <phase_3+332>  je     0x401108 <phase_3+339>
0x401103 <phase_3+334>  callq  0x40175b <explode_bomb>
```

여기서 성공적으로 피해서 온다면 +328로 move를 하게 되는데, 여기서 또 익숙한 주소가 보인다. 두 번째 입력이 들어가 있는 주소이다! 그럼 %al의 값이 두 번째 입력이 됨을 알 수 있다.

```
(gdb) i r al
al             0x66     102
(gdb) p/c 0x66
$4 = 102 'f'
(gdb)
```

al은 102가 저장되어있었고, 두 번째 입력은 아까 위에서 문자형이라 하였으므로 아스키코드를 풀어 102에 대응되는 f를 이끌어 낸다.
결론적으로 내가 찾아낸 정답은 1 f 629이다.

허나, 첫 번째 입력이 여러개 가능하며, 그에 대응되는 코드가 만들어져 있음을 확인했으므로 정답은 하나가 아닌 여러 가지 임을 알 수 있다.

Phase 3 [정답]

1 f 629(단, 허용되는 정답은 하나가 아님)

Phase 4 [결과 화면 캡처]

```
352 4
So you got that one.  Try this one.
```

Phase 4 [진행 과정 설명]

```
x0x40115d <phase_4>       sub    $0x18,%rsp
x0x401161 <phase_4+4>     mov    %fs:0x28,%rax
x0x40116a <phase_4+13>    mov    %rax,0x8(%rsp)
x0x40116f <phase_4+18>    xor    %eax,%eax
x0x401171 <phase_4+20>    mov    %rsp,%rcx
x0x401174 <phase_4+23>    lea    0x4(%rsp),%rdx
x0x401179 <phase_4+28>    mov    $0x402a4d,%esi
x0x40117e <phase_4+33>    callq  0x400c40 <__isoc99_sscanf@plt>
x0x401183 <phase_4+38>    cmp    $0x2,%eax
x0x401186 <phase_4+41>    jne    0x401193 <phase_4+54>
x0x401188 <phase_4+43>    mov    (%rsp),%eax
x0x40118b <phase_4+46>    sub    $0x2,%eax
x0x40118e <phase_4+49>    cmp    $0x2,%eax
x0x401191 <phase_4+52>    jbe    0x401198 <phase_4+59>
x0x401193 <phase_4+54>    callq  0x40175b <explode_bomb>
x0x401198 <phase_4+59>    mov    (%rsp),%esi
x0x40119b <phase_4+62>    mov    $0x9,%edi
x0x4011a0 <phase_4+67>    callq  0x401122 <func4>
x0x4011a5 <phase_4+72>    cmp    0x4(%rsp),%eax
x0x4011a9 <phase_4+76>    je     0x4011b0 <phase_4+83>
x0x4011ab <phase_4+78>    callq  0x40175b <explode_bomb>
x0x4011b0 <phase_4+83>    mov    0x8(%rsp),%rax
x0x4011b5 <phase_4+88>    xor    %fs:0x28,%rax
x0x4011be <phase_4+97>    je     0x4011c5 <phase_4+104>
x0x4011c0 <phase_4+99>    callq  0x400b90 <__stack_chk_fail@plt>
x0x4011c5 <phase_4+104>   add    $0x18,%rsp
x0x4011c9 <phase_4+108>   retq
```

뭔가 비교적 짧아보이는 코드다. 하지만 중간에 보면 func4호출이 있다. 생각보다 손이 많이 갈 것 같다는 생각이 들었다. 차근차근 살펴보자.

```
0x401179 <phase_4+28>    mov    $0x402a4d,%esi
0x40117e <phase_4+33>    callq  0x400c40 <__isoc99_sscanf@plt>
0x401183 <phase_4+38>    cmp    $0x2,%eax
0x401186 <phase_4+41>    jne    0x401193 <phase_4+54>
0x401188 <phase_4+43>    mov    (%rsp),%eax
0x40118b <phase_4+46>    sub    $0x2,%eax
0x40118e <phase_4+49>    cmp    $0x2,%eax
0x401191 <phase_4+52>    jbe    0x401198 <phase_4+59>
0x401193 <phase_4+54>    callq  0x40175b <explode_bomb>
```

이젠 의심을 넘어 고맙다. 힌트를 주는 고마운 친구인 0x402a4d를 확인해보자.
%d %d라고 알려줬다. 우리는 두 개 입력하고 둘 다 정수형이어야 한다.

```
(gdb) x/s 0x402a4d
0x402a4d:       "%d %d"
(gdb)
```

첫 번째 비교문을 보자.
eax는 입력한 개수가 적혀있다. 위에서 찾은 대로만 해주면 걸릴 일이 없다.
다음 코드까지 넘어가서 +43을 보면 뭔가 입력한 값이 들어가 있을 것 같은 rsp에서 eax로 값을 옮기고 있다. 실행 된 이후 확인해보니 두 번째 입력 값이 들어있었다.
다음을 보면 여기서 2를 뺀 값이 2보다 작거나 같아야했다. 그러므로 두 번째 입력값은 4보다 작거나 같아야 한다.
그리고 이후에 두 번째 입력 값을 인자로 한 func4 연산에 들어간다.

```
0x401122 <func4>         test   %edi,%edi
0x401124 <func4+2>       jle    0x401151 <func4+47>
0x401126 <func4+4>       mov    %esi,%eax
0x401128 <func4+6>       cmp    $0x1,%edi
0x40112b <func4+9>       je     0x40115b <func4+57>
0x40112d <func4+11>      push   %r12
0x40112f <func4+13>      push   %rbp
0x401130 <func4+14>      push   %rbx
0x401131 <func4+15>      mov    %esi,%ebp
0x401133 <func4+17>      mov    %edi,%ebx
0x401135 <func4+19>      lea    -0x1(%rdi),%edi
0x401138 <func4+22>      callq  0x401122 <func4>
0x40113d <func4+27>      lea    0x0(%rbp,%rax,1),%r12d
0x401142 <func4+32>      lea    -0x2(%rbx),%edi
0x401145 <func4+35>      mov    %ebp,%esi
0x401147 <func4+37>      callq  0x401122 <func4>
0x40114c <func4+42>      add    %r12d,%eax
0x40114f <func4+45>      jmp    0x401157 <func4+53>
0x401151 <func4+47>      mov    $0x0,%eax
0x401156 <func4+52>      retq
0x401157 <func4+53>      pop    %rbx
0x401158 <func4+54>      pop    %rbp
0x401159 <func4+55>      pop    %r12
0x40115b <func4+57>      repz retq
```

하지만 다음 코드를 먼저 보자.

```
x0x4011a5 <phase_4+72>    cmp    0x4(%rsp),%eax
x0x4011a9 <phase_4+76>    je     0x4011b0 <phase_4+83>
x0x4011ab <phase_4+78>    callq  0x40175b <explode_bomb>
```

우리가 필요한 것은 eax의 값 뿐이고 앞의 값은 첫 번째 입력이므로 사실상 func4는 시간을 잡아먹는 함정일 뿐, +72에 breakpoint를 걸어두고 +72에 도착한다면 그때 eax의 값을 확인하는 편이 좋다. 첫 번째 입력 값이 4였을 때는 352가 나왔고 그것이 곧 첫 번째 입력 값임을 알 수 있다.

내가 찾아낸 정답은 352 4이다.

단 두 번째 입력이 한가지로 제한되지 않은 시점에서 답은 여러 개가 될 수 있고, 폭탄이 터질까 무섭지만 재미삼아 찾아본 결과 여러 개의 답이 나옴을 직접 실험해보기도 했다.

Phase 4 [정답]

352 4

(단, 여러 개의 답이 존재할 수 있음.)

Phase 5 [결과 화면 캡처]

```
5 115
Good work!  On to the next...
```

Phase 5 [진행 과정 설명]

벌써 5페이즈다. 끝이 보이는 것 같다. 코드도 뭔가 짧아 보인다.

```
x0x4011ca <phase_5>       sub    $0x18,%rsp
x0x4011ce <phase_5+4>     mov    %fs:0x28,%rax
x0x4011d7 <phase_5+13>    mov    %rax,0x8(%rsp)
x0x4011dc <phase_5+18>    xor    %eax,%eax
x0x4011de <phase_5+20>    lea    0x4(%rsp),%rcx
x0x4011e3 <phase_5+25>    mov    %rsp,%rdx
x0x4011e6 <phase_5+28>    mov    $0x402a4d,%esi
x0x4011eb <phase_5+33>    callq  0x400c40 <__isoc99_sscanf@plt>
x0x4011f0 <phase_5+38>    cmp    $0x1,%eax
x0x4011f3 <phase_5+41>    jg     0x4011fa <phase_5+48>
x0x4011f5 <phase_5+43>    callq  0x40175b <explode_bomb>
x0x4011fa <phase_5+48>    mov    (%rsp),%eax
x0x4011fd <phase_5+51>    and    $0xf,%eax
x0x401200 <phase_5+54>    mov    %eax,(%rsp)
x0x401203 <phase_5+57>    cmp    $0xf,%eax
x0x401206 <phase_5+60>    je     0x401237 <phase_5+109>
x0x401208 <phase_5+62>    mov    $0x0,%ecx
x0x40120d <phase_5+67>    mov    $0x0,%edx
x0x401212 <phase_5+72>    add    $0x1,%edx
x0x401215 <phase_5+75>    cltq
x0x401217 <phase_5+77>    mov    0x4027c0(,%rax,4),%eax
x0x40121e <phase_5+84>    add    %eax,%ecx
x0x401220 <phase_5+86>    cmp    $0xf,%eax
x0x401223 <phase_5+89>    jne    0x401212 <phase_5+72>
x0x401225 <phase_5+91>    movl   $0xf,(%rsp)
x0x40122c <phase_5+98>    cmp    $0xf,%edx
```

```
0x401225 <phase_5+91>   movl   $0xf,(%rsp)
0x40122c <phase_5+98>   cmp    $0xf,%edx
0x40122f <phase_5+101>  jne    0x401237 <phase_5+109>
0x401231 <phase_5+103>  cmp    0x4(%rsp),%ecx
0x401235 <phase_5+107>  je     0x40123c <phase_5+114>
0x401237 <phase_5+109>  callq  0x40175b <explode_bomb>
0x40123c <phase_5+114>  mov    0x8(%rsp),%rax
0x401241 <phase_5+119>  xor    %fs:0x28,%rax
0x40124a <phase_5+128>  je     0x401251 <phase_5+135>
0x40124c <phase_5+130>  callq  0x400b90 <__stack_chk_fail@plt>
0x401251 <phase_5+135>  add    $0x18,%rsp
0x401255 <phase_5+139>  retq
```

이제 비교문이 어디있는지, 힌트가 어디에 있는 지 조금씩 보면서 할 수 있을 것 같다. 처음부터 천천히 가보자.

```
0x4011e6 <phase_5+28>   mov    $0x402a4d,%esi
0x4011eb <phase_5+33>   callq  0x400c40 <__isoc99_sscanf@plt>
0x4011f0 <phase_5+38>   cmp    $0x1,%eax
0x4011f3 <phase_5+41>   jg     0x4011fa <phase_5+48>
0x4011f5 <phase_5+43>   callq  0x40175b <explode_bomb>
0x4011fa <phase_5+48>   mov    (%rsp),%eax
0x4011fd <phase_5+51>   and    $0xf,%eax
0x401200 <phase_5+54>   mov    %eax,(%rsp)
0x401203 <phase_5+57>   cmp    $0xf,%eax
0x401206 <phase_5+60>   je     0x401237 <phase_5+109>
```

언제나처럼 우선 0x402a4d를 확인해주자. %d %d가 나왔다. 두 개 입력하고 둘 다 정수형이다.

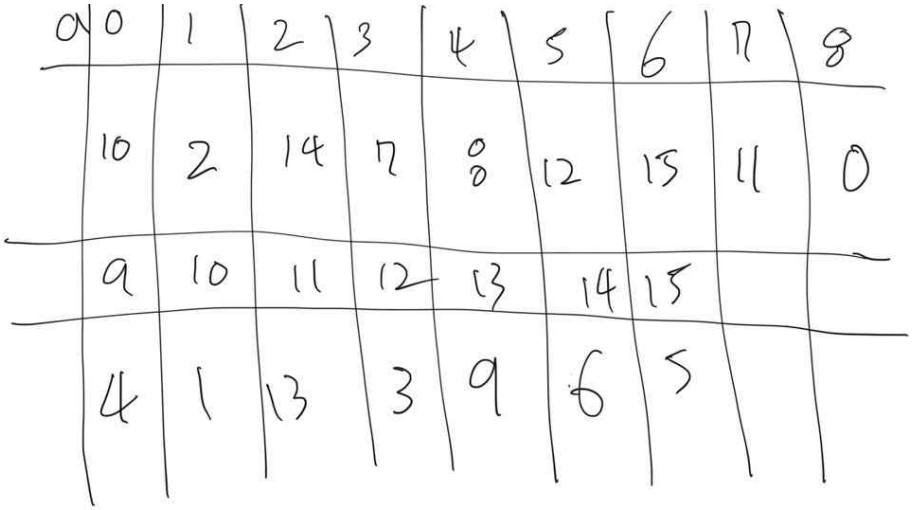
다음 코드를 보자. 입력 수가 1보다 작으면 폭탄이 터진다. 위에서 이미 확인한 내용이므로 가뿐히 피해주자.

다음 코드를 확인한다. rsp를 eax에 넣고, 이진수 1111과 and연산 후 다시 rsp에 넣어준다. 연산 후에, 이진수 1111 즉, 15이면 +109로 간다. 109? 멀리가는 것같아 좋은 줄 알았지만 역시 폭탄이었다. %rsp는 첫 번째 입력이었으므로 첫 번째 입력은 맨 아래 네비트가 다 1이면 안됨을 알 수 있었다.

```
0x401208 <phase_5+62>   mov    $0x0,%ecx
0x40120d <phase_5+67>   mov    $0x0,%edx
0x401212 <phase_5+72>   add    $0x1,%edx
0x401215 <phase_5+75>   cltq
0x401217 <phase_5+77>   mov    0x4027c0(,%rax,4),%eax
0x40121e <phase_5+84>   add    %eax,%ecx
0x401220 <phase_5+86>   cmp    $0xf,%eax
0x401223 <phase_5+89>   jne    0x401212 <phase_5+72>
0x401225 <phase_5+91>   movl   $0xf,(%rsp)
0x40122c <phase_5+98>   cmp    $0xf,%edx
0x40122f <phase_5+101>  jne    0x401237 <phase_5+109>
0x401231 <phase_5+103>  cmp    0x4(%rsp),%ecx
0x401235 <phase_5+107>  je     0x40123c <phase_5+114>
0x401237 <phase_5+109>  callq  0x40175b <explode_bomb>
```

쉬운 줄 알았던 5페이즈.... 여기서 +77을 해석하지 못해 시간 많이 잡아먹었다. 결국 알아낸 것은 사이즈 16짜리 배열이라는 것. 원소는 0-15까지 가지고 있으며 함수의 루프는 배열의 위치에 그 전 값을 넣어서 연산하여 값을 찾아주는 것이며, 이 값이 15일 경우 빠져나가고 15가 아닌 경우 그 값을 A[]에 다시 집어넣어 루프한다. 이때, edx에서 연산을 할 때마다 1씩 더해지는데, 이 값이 15여야 폭탄이 안 터지고 안전히 종료시킬 수 있다.

연산할 배열을 살펴보면 아래와 같다.



이제 위의 값이 a의 주소이고, 아래가 그 주소에 따른 값이다. 15번 연산을 하기위한 값을 찾기 위해 15가 나오는 값부터 역산을 해보자.

15-6-14-2-1-10-0-8-4-9-13-11-7-3-12-5

15번 연산을 했을때의 값이므로 16번째의 값이다. 이 값이 첫 번째 값과 같으며, 그리고 위의 함수를 살펴보면 ecx에 연산한 전체의 합이 들어가 있는데, 이 값이 두 번째 입력값과 같다.

그러므로 정답은 5 115이다.

Phase 5 [정답]

5 115

Phase 6 [결과 화면 캡처]

```
3 5 6 4 2 1
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
[Inferior 1 (process 13063) exited normally]
```

Phase 6 [진행 과정 설명]

끝이 보인다. 마지막까지 가보자.

```
0x0000000000401256 <+0>:     push   %r13
0x0000000000401258 <+2>:     push   %r12
0x000000000040125a <+4>:     push   %rbp
0x000000000040125b <+5>:     push   %rbx
0x000000000040125c <+6>:     sub    $0x68,%rsp
0x0000000000401260 <+10>:    mov    %fs:0x28,%rax
0x0000000000401269 <+19>:    mov    %rax,0x58(%rsp)
0x000000000040126e <+24>:    xor    %eax,%eax
0x0000000000401270 <+26>:    mov    %rsp,%rsi
0x0000000000401273 <+29>:    callq  0x401791 <read_six_numbers>
0x0000000000401278 <+34>:    mov    %rsp,%r12
0x000000000040127b <+37>:    mov    $0x0,%r13d
0x0000000000401281 <+43>:    mov    %r12,%rbp
0x0000000000401284 <+46>:    mov    (%r12),%eax
0x0000000000401288 <+50>:    sub    $0x1,%eax
0x000000000040128b <+53>:    cmp    $0x5,%eax
0x000000000040128e <+56>:    jbe    0x401295 <phase_6+63>
0x0000000000401290 <+58>:    callq  0x40175b <explode_bomb>
0x0000000000401295 <+63>:    add    $0x1,%r13d
0x0000000000401299 <+67>:    cmp    $0x6,%r13d
0x000000000040129d <+71>:    je     0x4012dc <phase_6+134>
0x000000000040129f <+73>:    mov    %r13d,%ebx
0x00000000004012a2 <+76>:    movslq %ebx,%rax
0x00000000004012a5 <+79>:    mov    (%rsp,%rax,4),%eax
0x00000000004012a8 <+82>:    cmp    %eax,0x0(%rbp)
0x00000000004012ab <+85>:    jne    0x4012b2 <phase_6+92>
0x00000000004012ad <+87>:    callq  0x40175b <explode_bomb>
0x00000000004012b2 <+92>:    add    $0x1,%ebx
0x00000000004012b5 <+95>:    cmp    $0x5,%ebx
0x00000000004012b8 <+98>:    jle    0x4012a2 <phase_6+76>
0x00000000004012ba <+100>:   add    $0x4,%r12
0x00000000004012be <+104>:   jmp    0x401281 <phase_6+43>
0x00000000004012c0 <+106>:   mov    0x8(%rdx),%rdx
0x00000000004012c4 <+110>:   add    $0x1,%eax
0x00000000004012c7 <+113>:   cmp    %ecx,%eax
0x00000000004012c9 <+115>:   jne    0x4012c0 <phase_6+106>
0x00000000004012cb <+117>:   mov    %rdx,0x20(%rsp,%rsi,2)
0x00000000004012d0 <+122>:   add    $0x4,%rsi
0x00000000004012d4 <+126>:   cmp    $0x18,%rsi
0x00000000004012d8 <+130>:   jne    0x4012e1 <phase_6+139>
0x00000000004012da <+132>:   jmp    0x4012f5 <phase_6+159>
0x00000000004012dc <+134>:   mov    $0x0,%esi
0x00000000004012e1 <+139>:   mov    (%rsp,%rsi,1),%ecx
0x00000000004012e4 <+142>:   mov    $0x1,%eax
0x00000000004012e9 <+147>:   mov    $0x6042f0,%edx
0x00000000004012ee <+152>:   cmp    $0x1,%ecx
0x00000000004012f1 <+155>:   jg     0x4012c0 <phase_6+106>
```

```
---Type <return> to continue, or q <return> to quit---
   0x00000000004012f3 <+157>:   jmp    0x4012cb <phase_6+117>
   0x00000000004012f5 <+159>:   mov    0x20(%rsp),%rbx
   0x00000000004012fa <+164>:   lea    0x20(%rsp),%rax
   0x00000000004012ff <+169>:   lea    0x48(%rsp),%rsi
   0x0000000000401304 <+174>:   mov    %rbx,%rcx
   0x0000000000401307 <+177>:   mov    0x8(%rax),%rdx
   0x000000000040130b <+181>:   mov    %rdx,0x8(%rcx)
   0x000000000040130f <+185>:   add    $0x8,%rax
   0x0000000000401313 <+189>:   mov    %rdx,%rcx
   0x0000000000401316 <+192>:   cmp    %rsi,%rax
   0x0000000000401319 <+195>:   jne    0x401307 <phase_6+177>
   0x000000000040131b <+197>:   movq   $0x0,0x8(%rdx)
   0x0000000000401323 <+205>:   mov    $0x5,%ebp
   0x0000000000401328 <+210>:   mov    0x8(%rbx),%rax
   0x000000000040132c <+214>:   mov    (%rax),%eax
   0x000000000040132e <+216>:   cmp    %eax,(%rbx)
   0x0000000000401330 <+218>:   jle    0x401337 <phase_6+225>
   0x0000000000401332 <+220>:   callq  0x40175b <explode_bomb>
   0x0000000000401337 <+225>:   mov    0x8(%rbx),%rbx
   0x000000000040133b <+229>:   sub    $0x1,%ebp
   0x000000000040133e <+232>:   jne    0x401328 <phase_6+210>
   0x0000000000401340 <+234>:   mov    0x58(%rsp),%rax
   0x0000000000401345 <+239>:   xor    %fs:0x28,%rax
   0x000000000040134e <+248>:   je     0x401355 <phase_6+255>
   0x0000000000401350 <+250>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x0000000000401355 <+255>:   add    $0x68,%rsp
   0x0000000000401359 <+259>:   pop    %rbx
   0x000000000040135a <+260>:   pop    %rbp
   0x000000000040135b <+261>:   pop    %r12
   0x000000000040135d <+263>:   pop    %r13
   0x000000000040135f <+265>:   retq
```

코드가 또 어마어마하게 길다. 하지만 힌트가 하나있다. 2페이즈에서 봤던 read_six_numbers.

우선 6개 정수형으로 입력해주자. 그리고 위에서부터 보자.

```
   0x401270 <phase_6+26>    mov    %rsp,%rsi
   0x401273 <phase_6+29>    callq  0x401791 <read_six_numbers>
   0x401278 <phase_6+34>    mov    %rsp,%r12
   0x40127b <phase_6+37>    mov    $0x0,%r13d
   0x401281 <phase_6+43>    mov    %r12,%rbp
   0x401284 <phase_6+46>    mov    (%r12),%eax
   0x401288 <phase_6+50>    sub    $0x1,%eax
   0x40128b <phase_6+53>    cmp    $0x5,%eax
>  0x40128e <phase_6+56>    jbe    0x401295 <phase_6+63>
   0x401290 <phase_6+58>    callq  0x40175b <explode_bomb>
   0x401295 <phase_6+63>    add    $0x1,%r13d
   0x401299 <phase_6+67>    cmp    $0x6,%r13d
   0x40129d <phase_6+71>    je     0x4012dc <phase_6+134>
   0x40129f <phase_6+73>    mov    %r13d,%ebx
   0x4012a2 <phase_6+76>    movslq %ebx,%rax
   0x4012a5 <phase_6+79>    mov    (%rsp,%rax,4),%eax
   0x4012a8 <phase_6+82>    cmp    %eax,0x0(%rbp)
   0x4012ab <phase_6+85>    jne    0x4012b2 <phase_6+92>
   0x4012ad <phase_6+87>    callq  0x40175b <explode_bomb>
   0x4012b2 <phase_6+92>    add    $0x1,%ebx
   0x4012b5 <phase_6+95>    cmp    $0x5,%ebx
   0x4012b8 <phase_6+98>    jle    0x4012a2 <phase_6+76>
   0x4012ba <phase_6+100>   add    $0x4,%r12
   0x4012be <phase_6+104>   jmp    0x401281 <phase_6+43>
   0x4012c0 <phase_6+106>   mov    0x8(%rdx),%rdx
   0x4012c4 <phase_6+110>   add    $0x1,%eax
native process 9932 In: phase_6
rax            0x7      7
```

일부러 터지기 직전의 코드로 준비했다. 아래에 보면 rax의 값이 7이다.. 이 사진은 두 개의 부분으로 나눌 수 있다.

1. +43~+67
2. +76~+98

이렇게 두 부분으로 나뉘는데, 설명해보자면 우선 1번부분은 입력 값에서 1빼주고, 그 값이 5보다 작거나 같으면 폭탄을 피한다. 이 다음에 2번부분으로 넘어가는데 2번은 루프안의 루프라고 볼 수 있다. 여기서는 그 값이 0이 아닌지, 중복이 되어있지는 않은 지를 본다. 이후에 2번부분에 아무것도 안 걸렸으면 다음 입력 값을 확인할 수 있도록 1번 부분으로 루프한다.

여기서 우리가 알아낸 1번과 2번을 간단하게 보면 입력하는 6개의 수는 6보다 작거나 같으며, 0이 아니고 중복되면 안된다.

그럼 이게 무슨 뜻이냐하면 1~6까지 하나씩 입력하면 이 부분을 안전하게 탈출할 수 있다는 뜻이다.

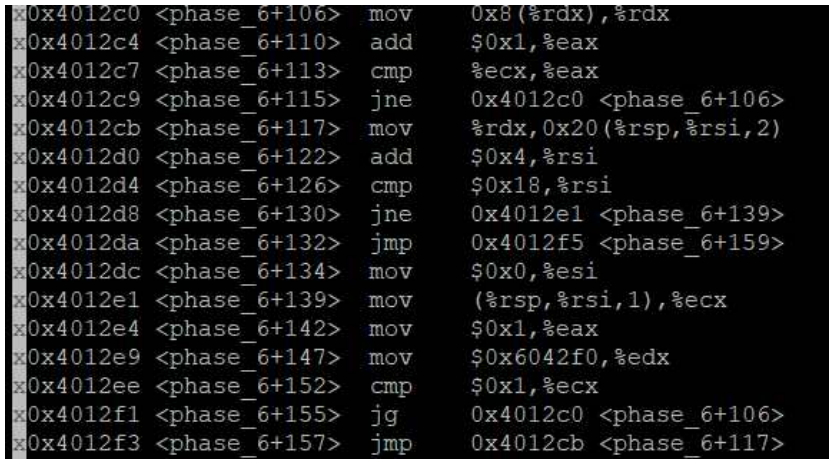
```
x0x4012c0 <phase_6+106>  mov    0x8(%rdx),%rdx
>x0x4012c4 <phase_6+110>  add    $0x1,%eax
 x0x4012c7 <phase_6+113>  cmp    %ecx,%eax
 x0x4012c9 <phase_6+115>  jne    0x4012c0 <phase_6+106>
 x0x4012cb <phase_6+117>  mov    %rdx,0x20(%rsp,%rsi,2)
 x0x4012d0 <phase_6+122>  add    $0x4,%rsi
 x0x4012d4 <phase_6+126>  cmp    $0x18,%rsi
 x0x4012d8 <phase_6+130>  jne    0x4012e1 <phase_6+139>
 x0x4012da <phase_6+132>  jmp    0x4012f5 <phase_6+159>
 x0x4012dc <phase_6+134>  mov    $0x0,%esi
 x0x4012e1 <phase_6+139>  mov    (%rsp,%rsi,1),%ecx
 x0x4012e4 <phase_6+142>  mov    $0x1,%eax
 x0x4012e9 <phase_6+147>  mov    $0x6042f0,%edx
 x0x4012ee <phase_6+152>  cmp    $0x1,%ecx
 x0x4012f1 <phase_6+155>  jg     0x4012c0 <phase_6+106>
 x0x4012f3 <phase_6+157>  jmp    0x4012cb <phase_6+117>
```

굉장히 복잡하고 어렵다.. 어디부터 손대야 할지 모르겠는데 뭔가 의심스러운 친구가 +147에 있다.

한번 출력해봤는데 이름이 Node1이다. 다 한번 보자.

```
(gdb) x/24x 0x6042f0
0x6042f0 <node1>:       0x00000319      0x00000001      0x00604300      0x00000000
0x604300 <node2>:       0x00000287      0x00000002      0x00604310      0x00000000
0x604310 <node3>:       0x00000079      0x00000003      0x00604320      0x00000000
0x604320 <node4>:       0x000001cd      0x00000004      0x00604330      0x00000000
0x604330 <node5>:       0x000000ec      0x00000005      0x00604340      0x00000000
0x604340 <node6>:       0x000001b8      0x00000006      0x00000000      0x00000000
```

아하! 이 문제는 이제 1-6까지 입력한 후에 노드의 값에 따라 1-6이 바뀌는 문제겠구나!를 알 수 있다.



이 루프문은 첫 번째 입력부터 6번째 입력까지 확인하고 각 노드를 할당하는 과정이다. 위의 노드를 잘 보면 다음 값이 저장되어있는데 이 과정을 통하여 수열이 연결리스트가 되지 않나 추정된다.

설명하자면 최초에 +134로 들어와서 루프를 시작하는데, +155에서 +106으로의 루프는 이제 노드를 할당 받아서 입력받은 수열로 가는 과정이고, 반복을 통해 입력받은 수열에 각 노드를 모두 할당 받으면 +126을 통해 +159로 이동한다.

여기서는 노드의 다음 노드의 주소를 확인하고 반복하며 다음 노드를 찾아주고, +218에서 얘가 더 작은지를 확인해준 후 +210으로 루프한다. 여기서 중요한 것을 알 수 있는데, 우리의 입력 값은 노드의 값이 작은 노드부터 큰노드 순으로 위치해야 한다는 걸 알 수 있다.

그러므로 위에서 확인한 노드의 값을 정렬하면

3 5 6 4 2 1 이 된다.

Phase 6 [정답]

3 5 6 4 2 1

Secret-Phase [결과 화면 캡처]

4 비밀값

```
352 4 DrEvil
So you got that one.  Try this one.
```

6클리어

```
3 5 6 4 2 1
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```

```
36
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 16258) exited normally]
```

시크릿 클리어

Secret-Phase [진행 과정 설명]

```
0x401914 <phase_defused+30>     cmpl   $0x6,0x202e91(%rip)        # 0x
0x40191b <phase_defused+37>     jne    0x40198a <phase_defused+148>
0x40191d <phase_defused+39>     lea    0x10(%rsp),%r8
0x401922 <phase_defused+44>     lea    0xc(%rsp),%rcx
0x401927 <phase_defused+49>     lea    0x8(%rsp),%rdx
0x40192c <phase_defused+54>     mov    $0x402a97,%esi
0x401931 <phase_defused+59>     mov    $0x6048b0,%edi
0x401936 <phase_defused+64>     mov    $0x0,%eax
0x40193b <phase_defused+69>     callq  0x400c40 <__isoc99_sscanf@plt>
0x401940 <phase_defused+74>     cmp    $0x3,%eax
0x401943 <phase_defused+77>     jne    0x401976 <phase_defused+128>
0x401945 <phase_defused+79>     mov    $0x402aa0,%esi
0x40194a <phase_defused+84>     lea    0x10(%rsp),%rdi
0x40194f <phase_defused+89>     callq  0x401487 <strings_not_equal>
0x401954 <phase_defused+94>     test   %eax,%eax
0x401956 <phase_defused+96>     jne    0x401976 <phase_defused+128>
0x401958 <phase_defused+98>     mov    $0x4028f8,%edi
0x40195d <phase_defused+103>    callq  0x400b70 <puts@plt>
0x401962 <phase_defused+108>    mov    $0x402920,%edi
0x401967 <phase_defused+113>    callq  0x400b70 <puts@plt>
0x40196c <phase_defused+118>    mov    $0x0,%eax
0x401971 <phase_defused+123>    callq  0x40139e <secret_phase>
```

다 풀었더니 아래에 fun7이 있었다. 여기저기 찾아보다가 phase_defused함수에서 찾아냈다. 처음 +30은 6까지 클리어하지 않았으면 그냥 지나가라는 뜻이고, 쭉 진행하다가 우리 자주 보던 move시리즈들과 주소 값들이 있어서 다 보았더니..

```
(gdb) x/s 0x6048b0
0x6048b0 <input_strings+240>:   "352 4 DrEvil"
(gdb) x/s 0x402a97
0x402a97:       "%d %d %s"
(gdb)
```

위와 같이 있었다. 위는 4페이즈에서 입력한 값이고, 아래는 입력받을 자료형이다. 물론 이미 클리어하고 보고서 쓰는 상황이라 정답이 들어가있지만... 분명히 4페이즈의 입력값은

정수형 두 개였는데 갑자기 바뀌었다? 더 가서 +84에서 세 번째 입력값을 꺼내오는 과정이 있고 꺼내온 값과 특정 값을 비교를 하는데 그 값이

```
(gdb) x/s 0x402aa0
0x402aa0:       "DrEvil"
```

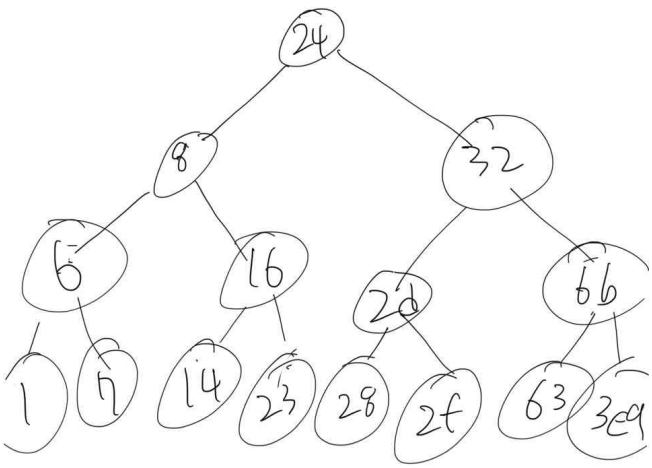
DrEvil이다. 시크릿페이즈 오는 법은 4페이즈 답에 DrEvil을 적고 6페이즈까지 클리어하는 것이었다!

찾았으니 이제 풀어보자.

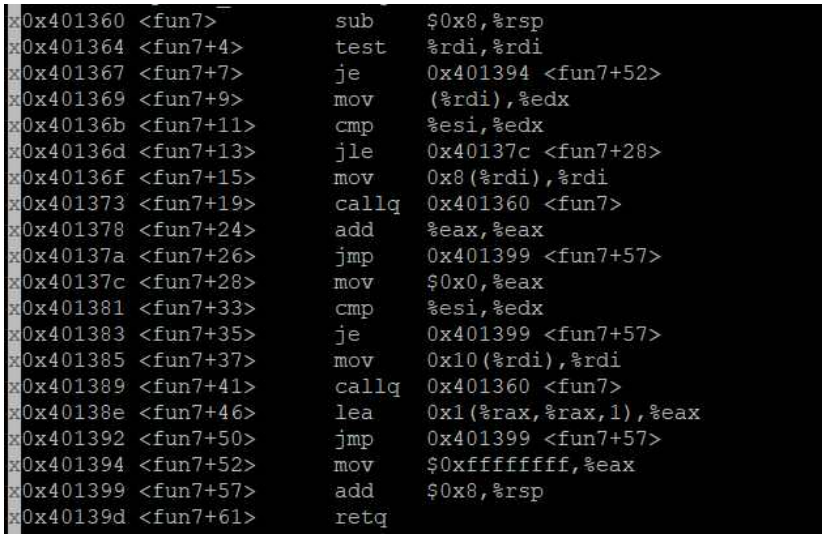
```
0x40139e <secret_phase>         push   %rbx
0x40139f <secret_phase+1>       callq  0x4017d0 <read_line>
0x4013a4 <secret_phase+6>       mov    $0xa,%edx
0x4013a9 <secret_phase+11>      mov    $0x0,%esi
0x4013ae <secret_phase+16>      mov    %rax,%rdi
0x4013b1 <secret_phase+19>      callq  0x400c20 <strtol@plt>
0x4013b6 <secret_phase+24>      mov    %rax,%rbx
0x4013b9 <secret_phase+27>      lea    -0x1(%rax),%eax
0x4013bc <secret_phase+30>      cmp    $0x3e8,%eax
0x4013c1 <secret_phase+35>      jbe    0x4013c8 <secret_phase+42>
0x4013c3 <secret_phase+37>      callq  0x40175b <explode_bomb>
0x4013c8 <secret_phase+42>      mov    %ebx,%esi
0x4013ca <secret_phase+44>      mov    $0x604110,%edi
0x4013cf <secret_phase+49>      callq  0x401360 <fun7>
0x4013d4 <secret_phase+54>      test   %eax,%eax
0x4013d6 <secret_phase+56>      je     0x4013dd <secret_phase+63>
0x4013d8 <secret_phase+58>      callq  0x40175b <explode_bomb>
0x4013dd <secret_phase+63>      mov    $0x402738,%edi
0x4013e2 <secret_phase+68>      callq  0x400b70 <puts@plt>
0x4013e7 <secret_phase+73>      callq  0x4018f6 <phase_defused>
0x4013ec <secret_phase+78>      pop    %rbx
0x4013ed <secret_phase+79>      retq
```

짧다. 근데 중간에 함수가 있다… 4페이즈가 기억난다.

우선 +44에 0x604110부터 보자 열어봤더니 이름이 n1 n21 n22 ….이렇다
이진트리의 느낌이 난다.



그림으로 그려봤더니 위와 같다. 16진수로 표현하였다. 이제 함수7로 들어가보자.



함수7이다. 보면 입력 값과 위의 노드를 인자로 쓰고 있다. 여기를 들어오기 전에 페이즈 4처럼 함수를 넘어가고 끝난 값과 이후 코드를 먼저 보고왔는데 eax가 0이어야만 했다. 그렇다면 이 함수가 +28로 빠르게 넘어가서 넣고 바로 끝나야 한다는 소리인데, +33에서 첫 값과 입력값을 비교한다. 여기서 당연히 같아야 함수를 빠져나갈 수 있으므로 정답은 당연히 0x24 즉 36이다.

Secret-Phase [정답]

36