2016 시스템 프로그래밍
- Lab 07 -

제출일자	2016.11.08
분 반	02
이 름	신종욱
학 번	2013024223

2.목차

3.개요

Bomb는 여러 단계로 이루어진 프로그램이다. 각 단계마다 화면에 문자열을 입력하게 되어 있다. 암호를 입력하게 되면, 해당 폭탄은 해체되고 다음 단계로 넘어가게 된다.
하지만 오답을 입력하게 되면, 해체에 실패하여 "BOOM!!!" 이라는 문장을 출력한 뒤 종료될 것이다.

- 사용할 명령어

(1) GDB

- GNU의 debugger로서, 실행 단계단계 어떻게 진행되고 어디에서 멈출지 어떻게 작동되는지 확인할수있다
    ex)1.disasseble
        2.x/x
        3.b explode_bomb

(2) Objdump

- 라이브러리, 컴파일된 오브젝트 모듈, 공유 오브젝트 파일, 독립 실행파일등의 바이너리 파일들의 정보를 보여주는 프로그램이다. objdump는 ELF 파일을 어셈블리어로 보여주는 디스어셈블러로 사용될 수 있다

2-Flow Chart

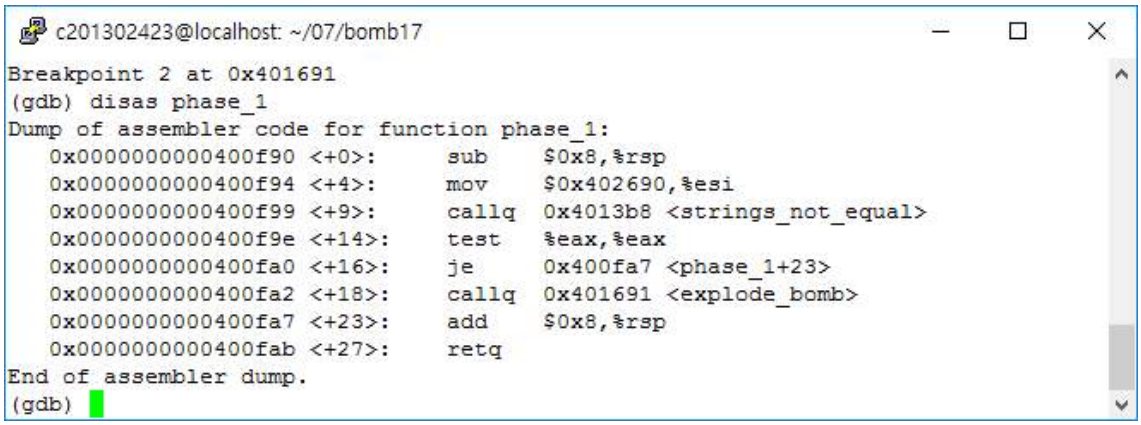
3-정답

4. 각 단계에 대한 해결방법

4-1 phase_1

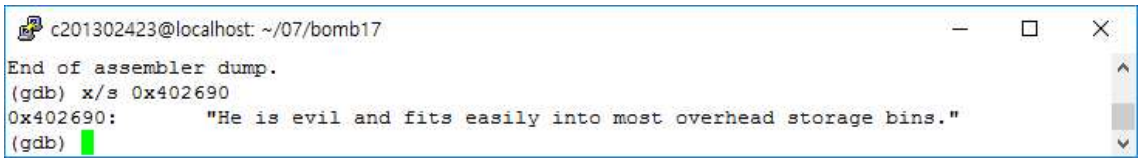
1-해결방법

일단 phase_1을 disassemble 한다

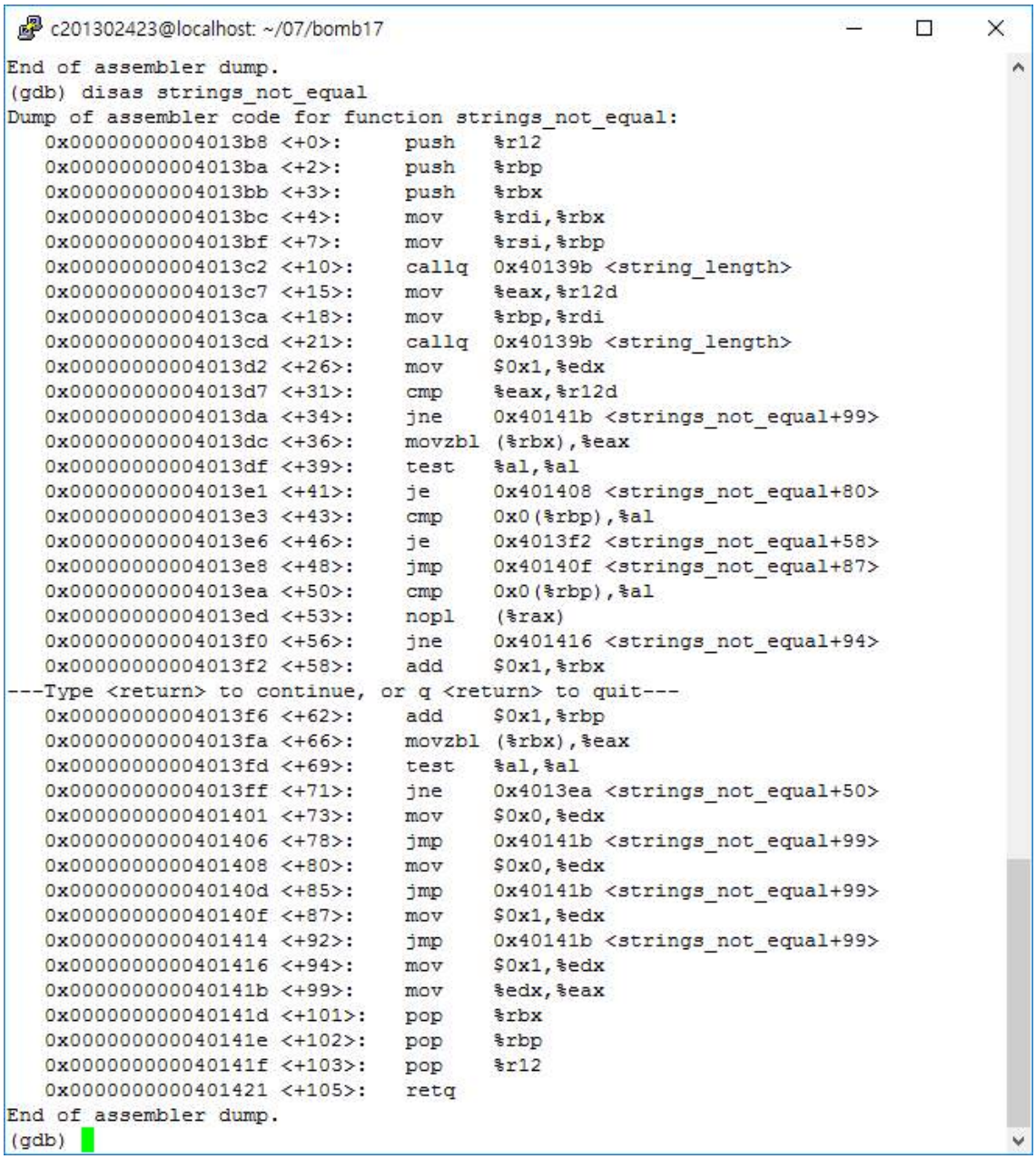


je명령문에서 점프하지못하면 폭발이 일어난다

$esi에 0x402690을 옮기는데 확인해보면.



이런 문자열이 나온다.



strings_not equal 함수 :

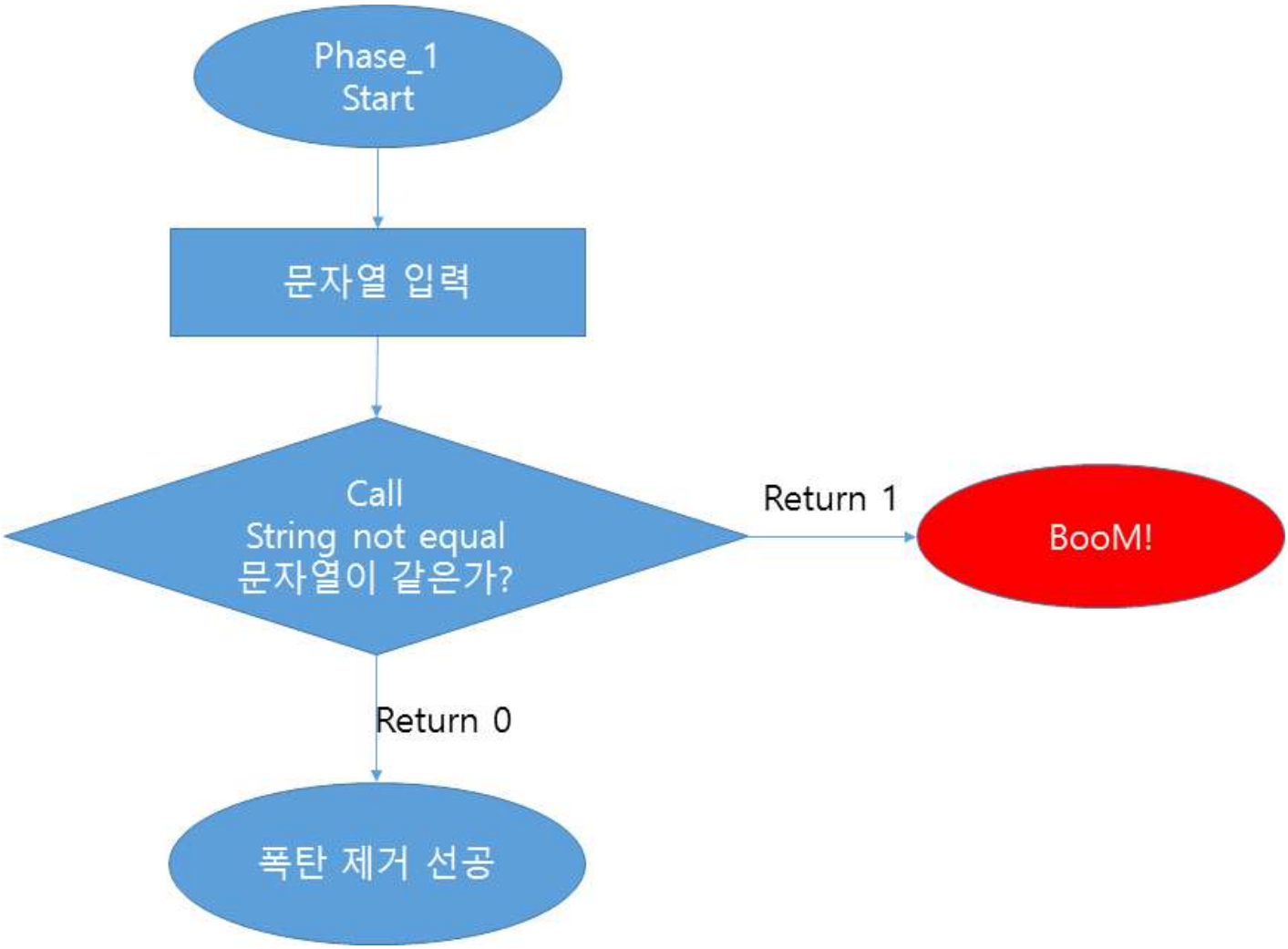
일단 길이가 같은지 판단하고

은 아마 입력한 스트링과 0x402690의 문자열이 같은지 판단후 맞다면 eax에 0을주고 아니면 1을 주게된다

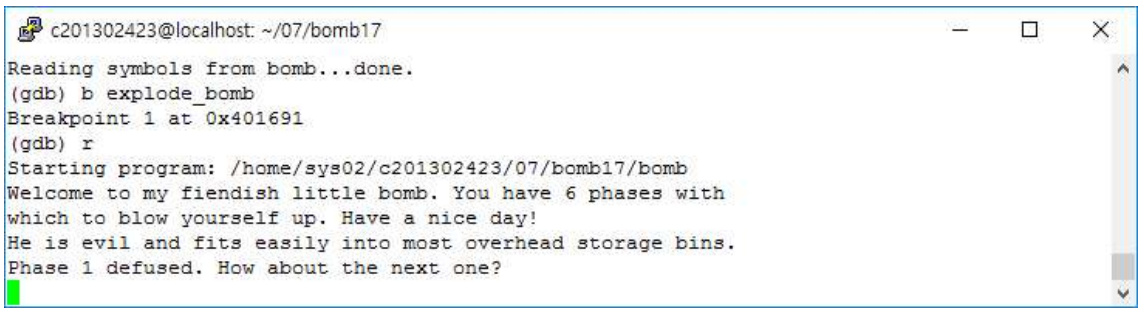
%eax값을 0,1를 리턴하는데 0일경우 test의 값이 0이라 je로가 점프하지만

1일경우에는 test값이 1이라 폭발한다

2-FlowChart



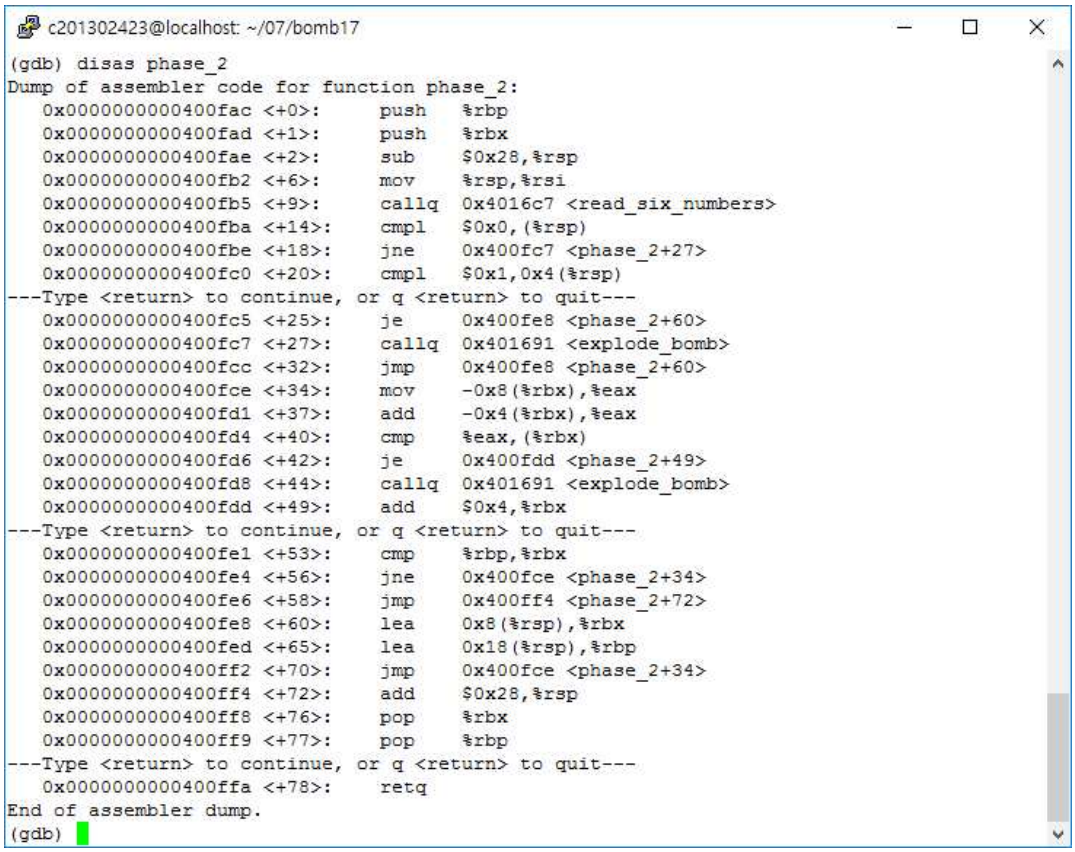
3-정답



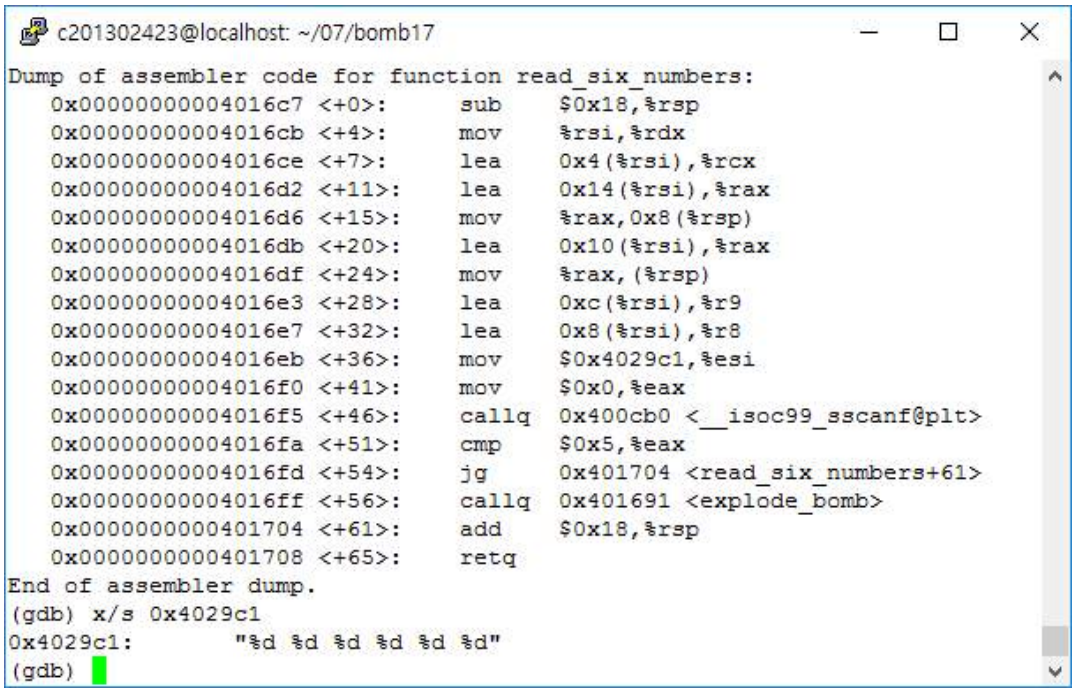
4-2 phase_2

1-해결방법

일단 phase_2을 disassemble 한다



일단 read six number를 disassemble 해보자

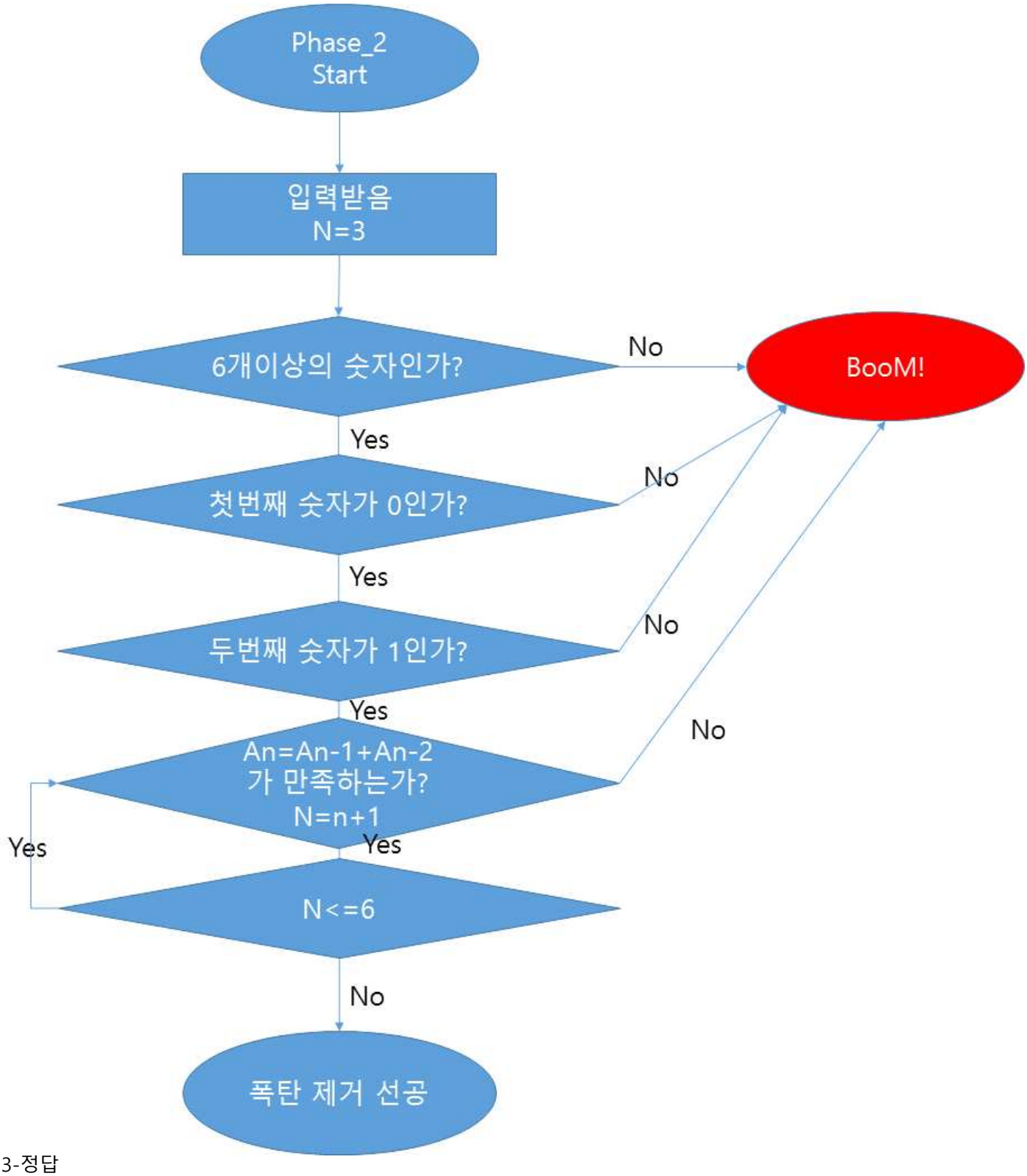


중간에 esi에 0x4029c1의 값을 옮기는데 확인해보면 %d가 6개 정수 6개를 스캔받는 함수이다 만약 6개 미만이라면 폭발한다.

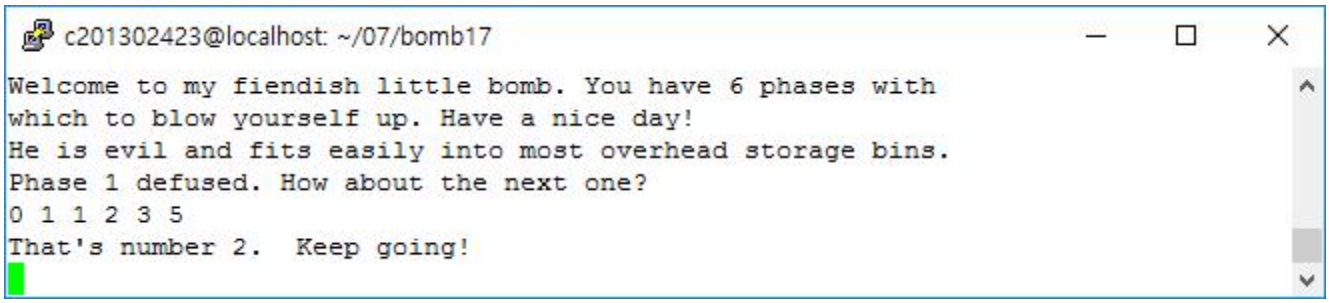
```
   0x0000000000400fba <+14>:    cmpl   $0x0,(%rsp)
   0x0000000000400fbe <+18>:    jne    0x400fc7 <phase_2+27>
   0x0000000000400fc0 <+20>:    cmpl   $0x1,0x4(%rsp)
---Type <return> to continue, or q <return> to quit---
   0x0000000000400fc5 <+25>:    je     0x400fe8 <phase_2+60>
   0x0000000000400fc7 <+27>:    callq  0x401691 <explode_bomb>
   0x0000000000400fcc <+32>:    jmp    0x400fe8 <phase_2+60>
   0x0000000000400fce <+34>:    mov    -0x8(%rbx),%eax
   0x0000000000400fd1 <+37>:    add    -0x4(%rbx),%eax
   0x0000000000400fd4 <+40>:    cmp    %eax,(%rbx)
   0x0000000000400fd6 <+42>:    je     0x400fdd <phase_2+49>
   0x0000000000400fd8 <+44>:    callq  0x401691 <explode_bomb>
   0x0000000000400fdd <+49>:    add    $0x4,%rbx
---Type <return> to continue, or q <return> to quit---
   0x0000000000400fe1 <+53>:    cmp    %rbp,%rbx
   0x0000000000400fe4 <+56>:    jne    0x400fce <phase_2+34>
   0x0000000000400fe6 <+58>:    jmp    0x400ff4 <phase_2+72>
   0x0000000000400fe8 <+60>:    lea    0x8(%rsp),%rbx
   0x0000000000400fed <+65>:    lea    0x18(%rsp),%rbp
   0x0000000000400ff2 <+70>:    jmp    0x400fce <phase_2+34>
   0x0000000000400ff4 <+72>:    add    $0x28,%rsp
   0x0000000000400ff8 <+76>:    pop    %rbx
   0x0000000000400ff9 <+77>:    pop    %rbp
```

6개를 읽은후 rsp 즉 첫 번째 숫자가 0인지 확인하고 4(rsp) 즉 두 번째 숫자가 1인지 확인한다
둘다 통과후 3번째 숫자 주소와,6번째 숫자 다음주소를 rbx rbp에 넣은후 다시 위로 올라간다
-0x8(%rbx)는 현재가르키고 있는 rbx의 전전숫자 예를들어 만약 rbx가 세 번째 숫자라면 첫 번째 숫자이다
그값은 eax에넣고 전숫자와 더한다 그렇게 구한 숫자를 현재 rbx와 같은지 비교후 틀리면 폭발한다
이말은 $A_n=A_{n-1}+A_{n-2}$이다 즉 피보나치 수열이다.
같으면 rbx의 값을 한 칸 앞으로 옮기고 rbx값과 rbp의 값을 비교후 같지않으면 반복한다 즉 rbx가 6번째 숫자 다음 주소를 가르킬때까지 실행한다
1,2번째 숫자가 0,1이기에 피보나치수열을 만들면
0 1 1 2 3 5 가 정답이된다.

2-FlowChart



3-정답


c201302423@localhost: ~/07/bomb17
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
He is evil and fits easily into most overhead storage bins.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!

4-3 phase_3

1-해결방법

```
c201302423@localhost: ~/07/bomb17
(gdb) disas phase_3
Dump of assembler code for function phase_3:
   0x0000000000400ffb <+0>:     sub    $0x18,%rsp
   0x0000000000400fff <+4>:     lea    0xc(%rsp),%rcx
   0x0000000000401004 <+9>:     lea    0x8(%rsp),%rdx
   0x0000000000401009 <+14>:    mov    $0x4029cd,%esi
   0x000000000040100e <+19>:    mov    $0x0,%eax
   0x0000000000401013 <+24>:    callq  0x400cb0 <__isoc99_sscanf@plt>
   0x0000000000401018 <+29>:    cmp    $0x1,%eax
   0x000000000040101b <+32>:    jg     0x401022 <phase_3+39>
   0x000000000040101d <+34>:    callq  0x401691 <explode_bomb>
   0x0000000000401022 <+39>:    cmpl   $0x7,0x8(%rsp)
   0x0000000000401027 <+44>:    ja     0x40108f <phase_3+148>
   0x0000000000401029 <+46>:    mov    0x8(%rsp),%eax
   0x000000000040102d <+50>:    jmpq   *0x402700(,%rax,8)
   0x0000000000401034 <+57>:    mov    $0x0,%eax
   0x0000000000401039 <+62>:    jmp    0x401040 <phase_3+69>
   0x000000000040103b <+64>:    mov    $0x2d2,%eax
   0x0000000000401040 <+69>:    sub    $0x112,%eax
   0x0000000000401045 <+74>:    jmp    0x40104c <phase_3+81>
   0x0000000000401047 <+76>:    mov    $0x0,%eax
   0x000000000040104c <+81>:    add    $0x1c9,%eax
   0x0000000000401051 <+86>:    jmp    0x401058 <phase_3+93>
   0x0000000000401053 <+88>:    mov    $0x0,%eax
---Type <return> to continue, or q <return> to quit---
   0x0000000000401058 <+93>:    sub    $0x37b,%eax
   0x000000000040105d <+98>:    jmp    0x401064 <phase_3+105>
   0x000000000040105f <+100>:   mov    $0x0,%eax
   0x0000000000401064 <+105>:   add    $0x37b,%eax
   0x0000000000401069 <+110>:   jmp    0x401070 <phase_3+117>
   0x000000000040106b <+112>:   mov    $0x0,%eax
   0x0000000000401070 <+117>:   sub    $0x37b,%eax
   0x0000000000401075 <+122>:   jmp    0x40107c <phase_3+129>
   0x0000000000401077 <+124>:   mov    $0x0,%eax
   0x000000000040107c <+129>:   add    $0x37b,%eax
   0x0000000000401081 <+134>:   jmp    0x401088 <phase_3+141>
   0x0000000000401083 <+136>:   mov    $0x0,%eax
   0x0000000000401088 <+141>:   sub    $0x37b,%eax
   0x000000000040108d <+146>:   jmp    0x401099 <phase_3+158>
   0x000000000040108f <+148>:   callq  0x401691 <explode_bomb>
   0x0000000000401094 <+153>:   mov    $0x0,%eax
   0x0000000000401099 <+158>:   cmpl   $0x5,0x8(%rsp)
   0x000000000040109e <+163>:   jg     0x4010a6 <phase_3+171>
   0x00000000004010a0 <+165>:   cmp    0xc(%rsp),%eax
   0x00000000004010a4 <+169>:   je     0x4010ab <phase_3+176>
   0x00000000004010a6 <+171>:   callq  0x401691 <explode_bomb>
   0x00000000004010ab <+176>:   add    $0x18,%rsp
   0x00000000004010af <+180>:   retq
---Type <return> to continue, or q <return> to quit---
End of assembler dump.
(gdb)
```

scaf에 들어가는 형태가 어떤지 확인하기위해 esi로 들어가는 0x4029cd를 확인했다

```
c201302423@localhost: ~/07/bomb17
End of assembler dump.
(gdb) x/s 0x4029cd
0x4029cd:       "%d %d"
(gdb)
```

읽은게 2개미만이면 폭발해버린다 2개이상이라면 다음으로 넘어간다
첫 번째 숫자가 7보다 큰지 확인한다 만약 크다면 폭발한다

```
   0x000000000040102d <+50>:    jmpq   *0x402700(,%rax,8)
   0x0000000000401034 <+57>:    mov    $0x0,%eax
   0x0000000000401039 <+62>:    jmp    0x401040 <phase_3+69>
   0x000000000040103b <+64>:    mov    $0x2d2,%eax
   0x0000000000401040 <+69>:    sub    $0x112,%eax
   0x0000000000401045 <+74>:    jmp    0x40104c <phase_3+81>
   0x0000000000401047 <+76>:    mov    $0x0,%eax
   0x000000000040104c <+81>:    add    $0x1c9,%eax
   0x0000000000401051 <+86>:    jmp    0x401058 <phase_3+93>
   0x0000000000401053 <+88>:    mov    $0x0,%eax
---Type <return> to continue, or q <return> to quit---
   0x0000000000401058 <+93>:    sub    $0x37b,%eax
   0x000000000040105d <+98>:    jmp    0x401064 <phase_3+105>
   0x000000000040105f <+100>:   mov    $0x0,%eax
   0x0000000000401064 <+105>:   add    $0x37b,%eax
   0x0000000000401069 <+110>:   jmp    0x401070 <phase_3+117>
   0x000000000040106b <+112>:   mov    $0x0,%eax
   0x0000000000401070 <+117>:   sub    $0x37b,%eax
   0x0000000000401075 <+122>:   jmp    0x40107c <phase_3+129>
   0x0000000000401077 <+124>:   mov    $0x0,%eax
   0x000000000040107c <+129>:   add    $0x37b,%eax
   0x0000000000401081 <+134>:   jmp    0x401088 <phase_3+141>
   0x0000000000401083 <+136>:   mov    $0x0,%eax
   0x0000000000401088 <+141>:   sub    $0x37b,%eax
   0x000000000040108d <+146>:   jmp    0x401099 <phase_3+158>
   0x000000000040108f <+148>:   callq  0x401691 <explode_bomb>
   0x0000000000401094 <+153>:   mov    $0x0,%eax
   0x0000000000401099 <+158>:   cmpl   $0x5,0x8(%rsp)
   0x000000000040109e <+163>:   jg     0x4010a6 <phase_3+171>
   0x00000000004010a0 <+165>:   cmp    0xc(%rsp),%eax
   0x00000000004010a4 <+169>:   je     0x4010ab <phase_3+176>
   0x00000000004010a6 <+171>:   callq  0x401691 <explode_bomb>
   0x00000000004010ab <+176>:   add    $0x18,%rsp
   0x00000000004010af <+180>:   retq
---Type <return> to continue, or q <return> to quit---
End of assembler dump.
(gdb)
```

+457-891+891-891+891-891

그다음 분기분은 점프테이블로 가는것인데 점프테이블 확인해보면

```
c201302423@localhost: ~/07/bomb17
x00
(gdb) x/64x 0x402700
0x402700:       0x3b    0x10    0x40    0x00    0x00    0x00    0x00    0x00
0x402708:       0x34    0x10    0x40    0x00    0x00    0x00    0x00    0x00
0x402710:       0x47    0x10    0x40    0x00    0x00    0x00    0x00    0x00
0x402718:       0x53    0x10    0x40    0x00    0x00    0x00    0x00    0x00
0x402720:       0x5f    0x10    0x40    0x00    0x00    0x00    0x00    0x00
0x402728:       0x6b    0x10    0x40    0x00    0x00    0x00    0x00    0x00
0x402730:       0x77    0x10    0x40    0x00    0x00    0x00    0x00    0x00
0x402738:       0x83    0x10    0x40    0x00    0x00    0x00    0x00    0x00
(gdb)
```

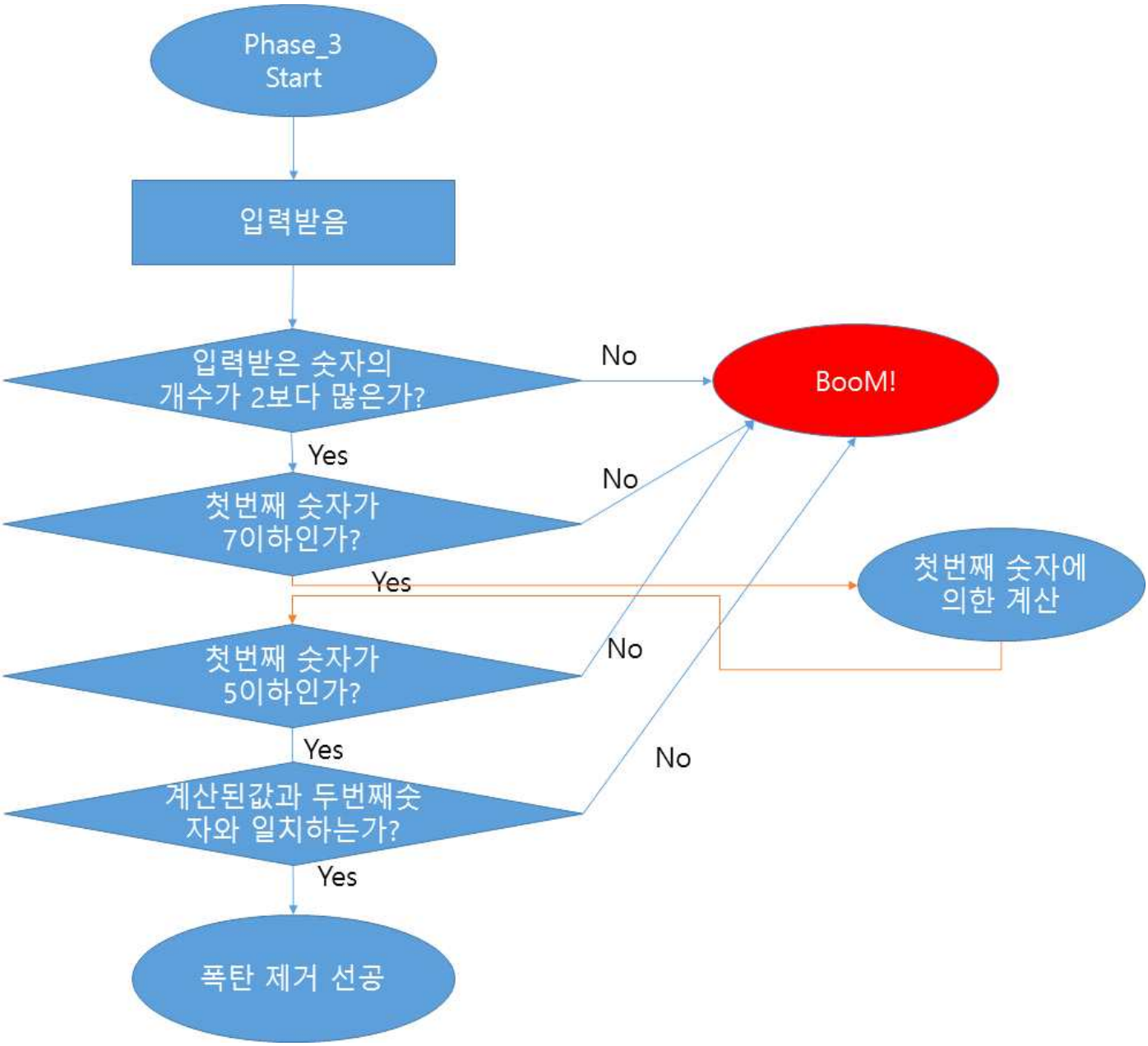
첫번째숫자	도착 주소
0	0x000000000040103b<+64>
1	0x0000000000401034<+57>
2	0x0000000000401047<+76>
3	0x0000000000401053<+88>
4	0x000000000040105f<+100>
5	0x000000000040106b<+112>
6	0x0000000000401077<+124>
7	0x0000000000401083<+136>

나는 첫 번째 숫자를 0으로 선택하였다
계산순서를 적어보면 일단 eax에 722를 저장한후 연산을 따라가보면
722-274+457-891+891-891+891-891=14가 최종적으로 eax에 남게된다.

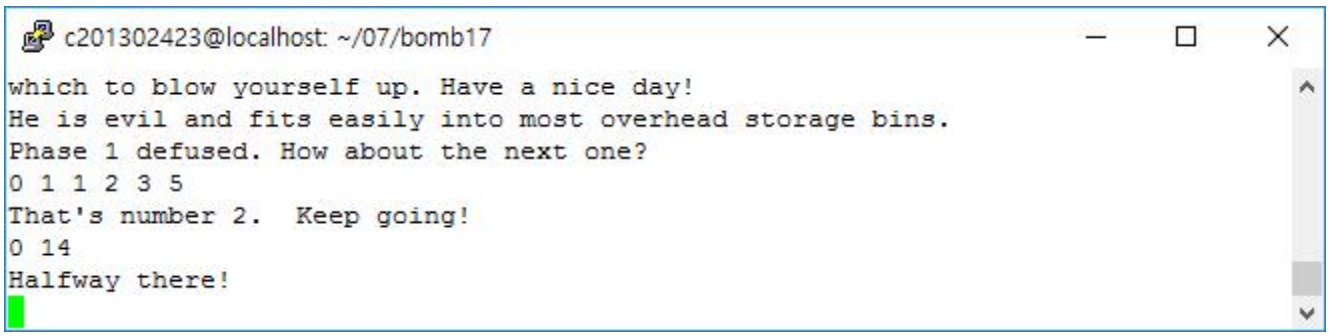
```
   0x0000000000401099 <+158>:   cmpl   $0x5,0x8(%rsp)
   0x000000000040109e <+163>:   jg     0x4010a6 <phase_3+171>
   0x00000000004010a0 <+165>:   cmp    0xc(%rsp),%eax
   0x00000000004010a4 <+169>:   je     0x4010ab <phase_3+176>
   0x00000000004010a6 <+171>:   callq  0x401691 <explode_bomb>
   0x00000000004010ab <+176>:   add    $0x18,%rsp
   0x00000000004010af <+180>:   retq
---Type <return> to continue, or q <return> to quit---
End of assembler dump.
(gdb)
```

그다음 분기문은 첫 번째 숫자가 다시 5이하인지 확인하고 두 번째입력된 숫자와 eax가 같은지 확인한다
같다면 폭탄을 해체하게된다. 즉 답은 첫 번째숫자는 0~5이고 두 번째 숫자는 그에맞은 점프 테이블후 계산되어 나오는 숫자이다.
0,14가 내가 구한 정답이다

2-FlowChart

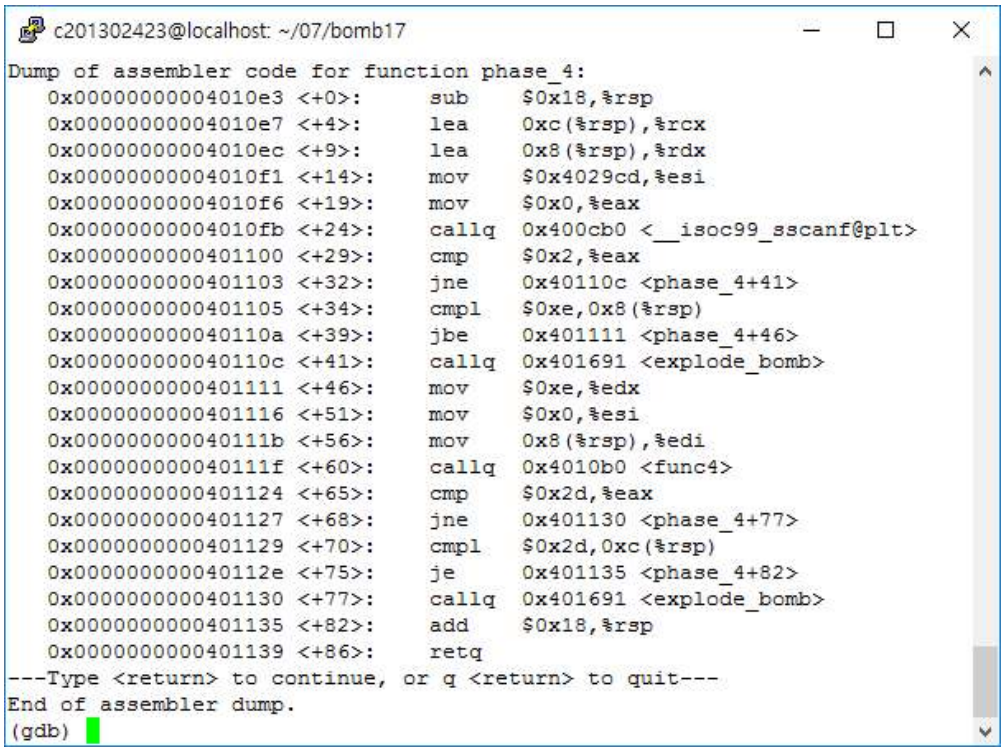


3-정답



4-4 phase_4

1-해결방법



rsp에서 8만큼 떨어진주소를 rdx 12만큼 떨어진 주소는 rcx에 저장된다.

이번에도 일단 esi에 들어가는 것을 확인했다.

두가지 정수가 들어가고

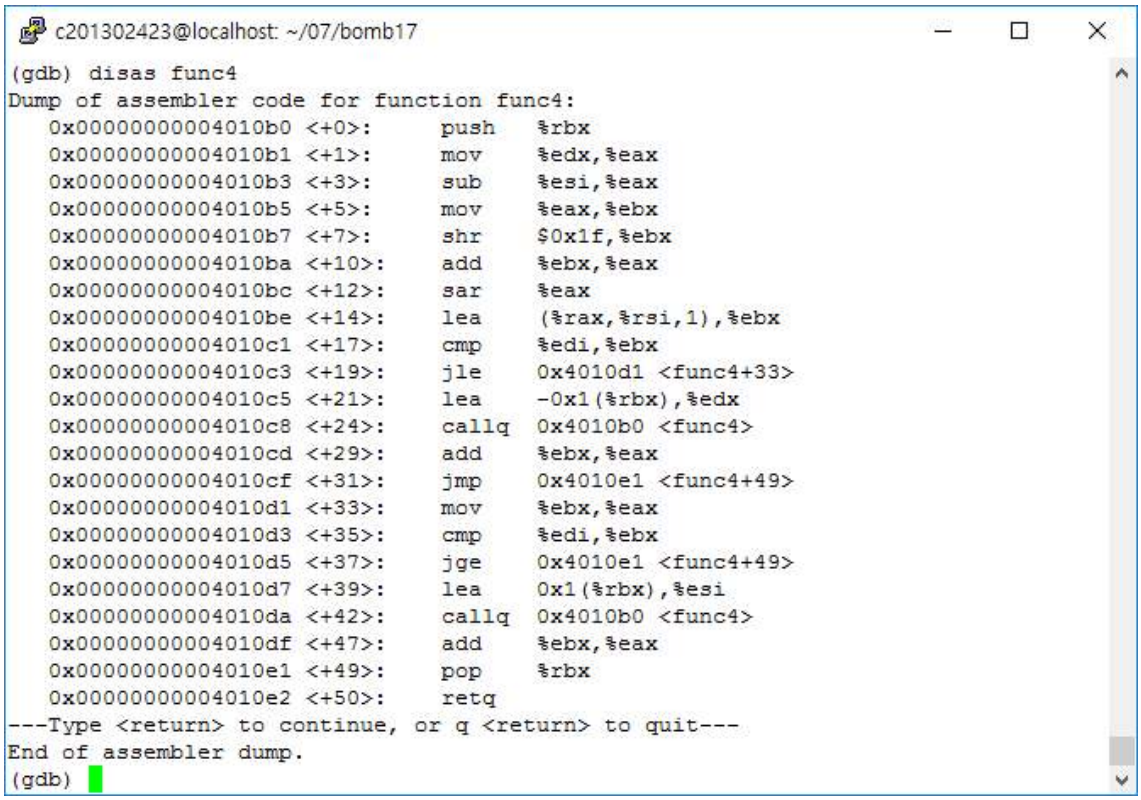
첫 번째 숫자가 14이하인지 확인후에

edx,esi에 14,0으로 초기화하고 edi에 첫 번째 숫자를 넣는다.

그후 func4가 실행되고 return 되는값이 45면 넘어간다 그리고 두 번째 입력받은 숫자가 45인지 확인한후 맞으면 폭탄이 해체된다.

일단 답은 ? 45이다 ?에들어가 14이하의 적절한 숫자를 찾아야한다

이제 func4를 파헤쳐보자.



일단 초기값은 edx는 14 esi는 0 edi는 14이하 숫자이다.

여기의 수식을 c언어로 한번 표현해보았다

```
int func4(int edx, int esi, int edi){
        int eax = edx;
        eax = eax - esi;
        int ebx = eax;
        ebx = ebx >> 31;//ebx가 양수나 0이면 0이고 음수이면 -1이 ebx에 저장된다
        eax = eax + ebx;
        eax = eax >> 1;//eax를 나누기 2하는 명령이다
        ebx = eax + esi;
        if (ebx - edi <= 0) {
                eax = ebx;
        if(ebx - edi >= 0){
                return eax;
        }
        else {
                esi=ebx+1;
                eax = func4(edx,esi,edi);
                eax=eax+ebx;
                return eax;
        }
} else {
        edx = ebx - 1;
        eax = func4(edx, esi, edi);
        eax = eax + ebx;
        return eax;

}}
```

edx가 0~14이니 이고 esi는 0 edi는 14이니 항상 처음에는 if문 전까지의 값들은
ebx=eax=eax/2 라고 할 수 있고 첫 if문으로가면 ebx는 아무리커도 7이다 그럼으로 만족한다
eax=ebx 두 번째 if문으로가면 처음에는 항상 틀림으로 else로간다 esi의 값을바꾸고 한번더 func4을 실행한다
그리고 나온뒤 그전의 ebx과 리턴값을 더해 리턴한다.
그밑에 else문은 edx값을 바꾸고 func4을 실행후 리턴값에 ebx를 더한값을 리턴한다

즉 이 함수는 ebx와 edi가 같아지는 순간이 있어야만 끝나는 재귀함수라고 할 수 있다.

이 함수를 보면 edi값을 변하지않는다 즉 계속 14이다 ebx가 14가 되도록 하면 프로그램이 끝날 것이다
그리고 ebx<=edi이면 edx값도 변하지않는다.
첫 번째 else이 일어날 때 esi는 edx/2+1이 되어 다시 실행된다
그럼 다음 func4일 때 ebx= eax+esi를 할 때 esi가 이제 0이아닌 양수이기 때문에 여기에서 ebx가 14가
되도록해야한다

한 개씩 숫자를 대입하여 확인한 결과 첫 번째 숫자가 14일 때 제대로 값이 구해지는데
첫 if문에 올 때 값들을 표로 정리하였다(eax는 변화를 봐야해서 if전에 변화하는 것을 적었다

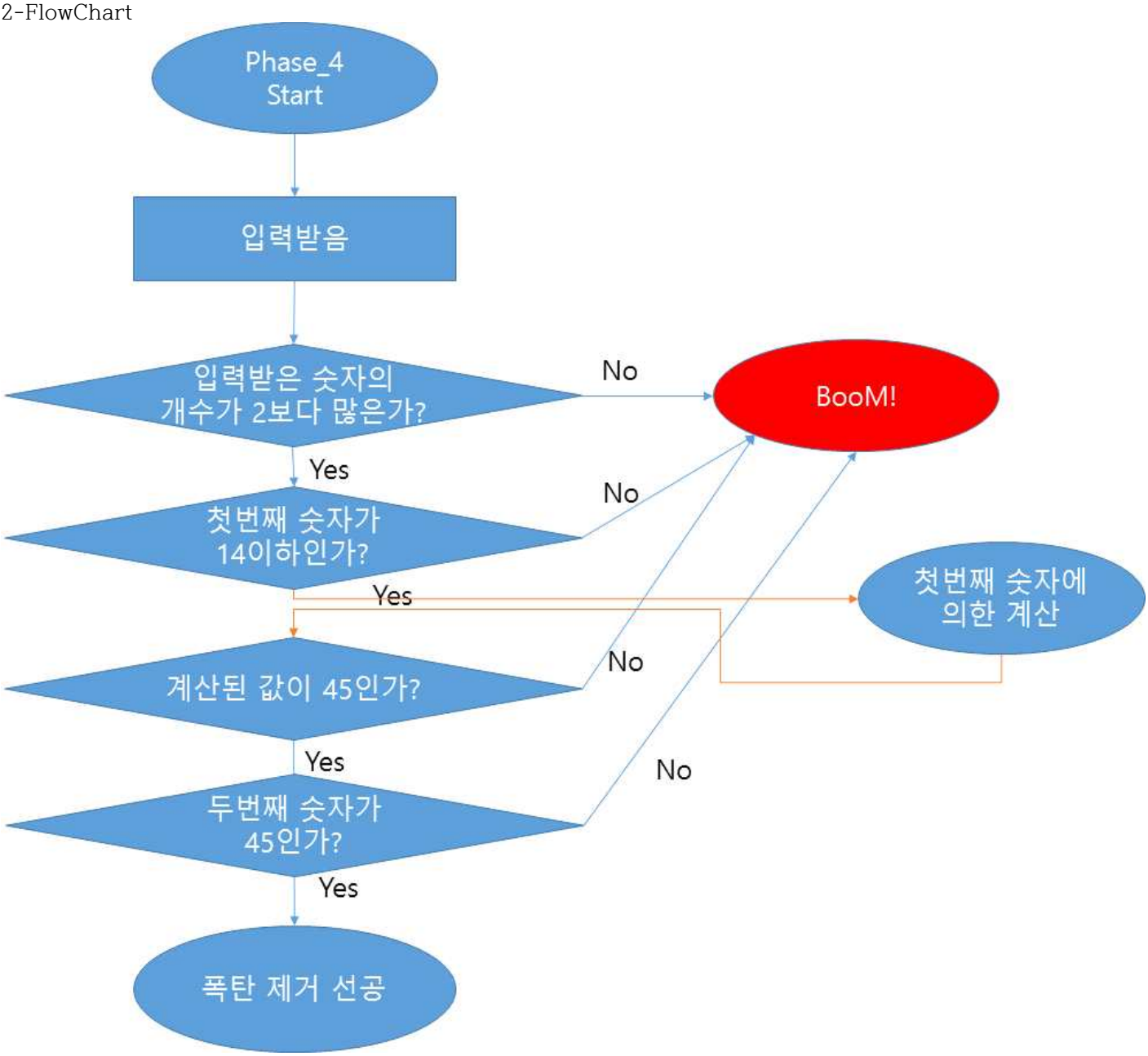
	edx(첫번째숫자)	esi	edi	ebx	eax
1	14	0	14	7(7(eax)+0(esi))	14>7
2	14	8(7(ebx)+1)	14	6->11(3eax)+8(esi))	14->6->3
3	14	12(11(ebx)+1)	14	2->13(1eax)+12(esi))	14->2->1
4	14	14(13(ebx)+1)	14	14(0eax)+14(esi))	0

역으로 불러오면서 0+14+13+11+7=45가 되어서 func4는 최종적으로 종료하면서 45를 리턴된다.

```
   0x0000000000401124 <+65>:    cmp    $0x2d,%eax
   0x0000000000401127 <+68>:    jne    0x401130 <phase_4+77>
   0x0000000000401129 <+70>:    cmpl   $0x2d,0xc(%rsp)
   0x000000000040112e <+75>:    je     0x401135 <phase_4+82>
   0x0000000000401130 <+77>:    callq  0x401691 <explode_bomb>
   0x0000000000401135 <+82>:    add    $0x18,%rsp
   0x0000000000401139 <+86>:    retq
---Type <return> to continue, or q <return> to quit---
End of assembler dump.
(gdb)
```

나온후 0x2d 즉 45와 비교후 같다면 0xc(%rsp) 두 번째 들어온 숫자도 45와 같은지 확인후 같다면 폭탄이 해체된다.

2-FlowChart



3-정답

```
c201302423@localhost: ~/07/bomb17
which to blow yourself up. Have a nice day!
He is evil and fits easily into most overhead storage bins.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
0 14
Halfway there!
14 45
So you got that one.  Try this one.
```

4-5 phase_5

1-해결방법

```
c201302423@localhost: ~/07/bomb17
End of assembler dump.
(gdb) disas phase_5
Dump of assembler code for function phase_5:
   0x000000000040113a <+0>:     push   %rbx
   0x000000000040113b <+1>:     mov    %rdi,%rbx
   0x000000000040113e <+4>:     callq  0x40139b <string_length>
   0x0000000000401143 <+9>:     cmp    $0x6,%eax
   0x0000000000401146 <+12>:    je     0x40114d <phase_5+19>
   0x0000000000401148 <+14>:    callq  0x401691 <explode_bomb>
   0x000000000040114d <+19>:    mov    $0x0,%eax
   0x0000000000401152 <+24>:    mov    $0x0,%edx
   0x0000000000401157 <+29>:    movzbl (%rbx,%rax,1),%ecx
   0x000000000040115b <+33>:    and    $0xf,%ecx
   0x000000000040115e <+36>:    add    0x402740(,%rcx,4),%edx
   0x0000000000401165 <+43>:    add    $0x1,%rax
   0x0000000000401169 <+47>:    cmp    $0x6,%rax
   0x000000000040116d <+51>:    jne    0x401157 <phase_5+29>
   0x000000000040116f <+53>:    cmp    $0x41,%edx
   0x0000000000401172 <+56>:    je     0x401179 <phase_5+63>
   0x0000000000401174 <+58>:    callq  0x401691 <explode_bomb>
   0x0000000000401179 <+63>:    pop    %rbx
   0x000000000040117a <+64>:    retq
End of assembler dump.
(gdb)
```

String이 들어오는데 길이를 판단하여 만약 길이가 6이라면 폭발하지않고 6이 아니라면 폭발한다.

그다음 분기문은 eax와 edx에 0을 저장하고

ecx=rax+rbx를하고 ecx의 4비트만 살리도록 and 연산을한다.

그후 4*rcx+0x402740주소의 값을 edx에 더한다

```
c201302423@localhost: ~/07/bomb17
(gdb) x/16xw 0x402740
0x402740 <array.3495>:  0x00000002      0x0000000a      0x00000006      0x00000001
0x402750 <array.3495+16>:       0x0000000c      0x00000010      0x00000009      0x00000003
0x402760 <array.3495+32>:       0x00000004      0x00000007      0x0000000e      0x00000005
0x402770 <array.3495+48>:       0x0000000b      0x00000008      0x0000000f      0x0000000d
(gdb)
```

0x402740을 확인해보니 이렇게 나와있다

이말은 rcx값이 만약 0이라면 2를 edx에 더하고 1이라면 a인 10을 더한다는 말이다.

더한후에 rax값을 1증가시키고 rax가 6이될때까지 한다 즉 6번 반복한후에 edx값이 65인지 확인하고 65면 폭탄이 해체된다. 이제 6번 더해서 65가 되어야하기 때문에 나는 임의로

10+11+12+13+14+5로 하기로했다

그러기 위해선 a,b,c,d,e,5를 선택해야한다. 이것들의 배열의 위치값으로 변경하면

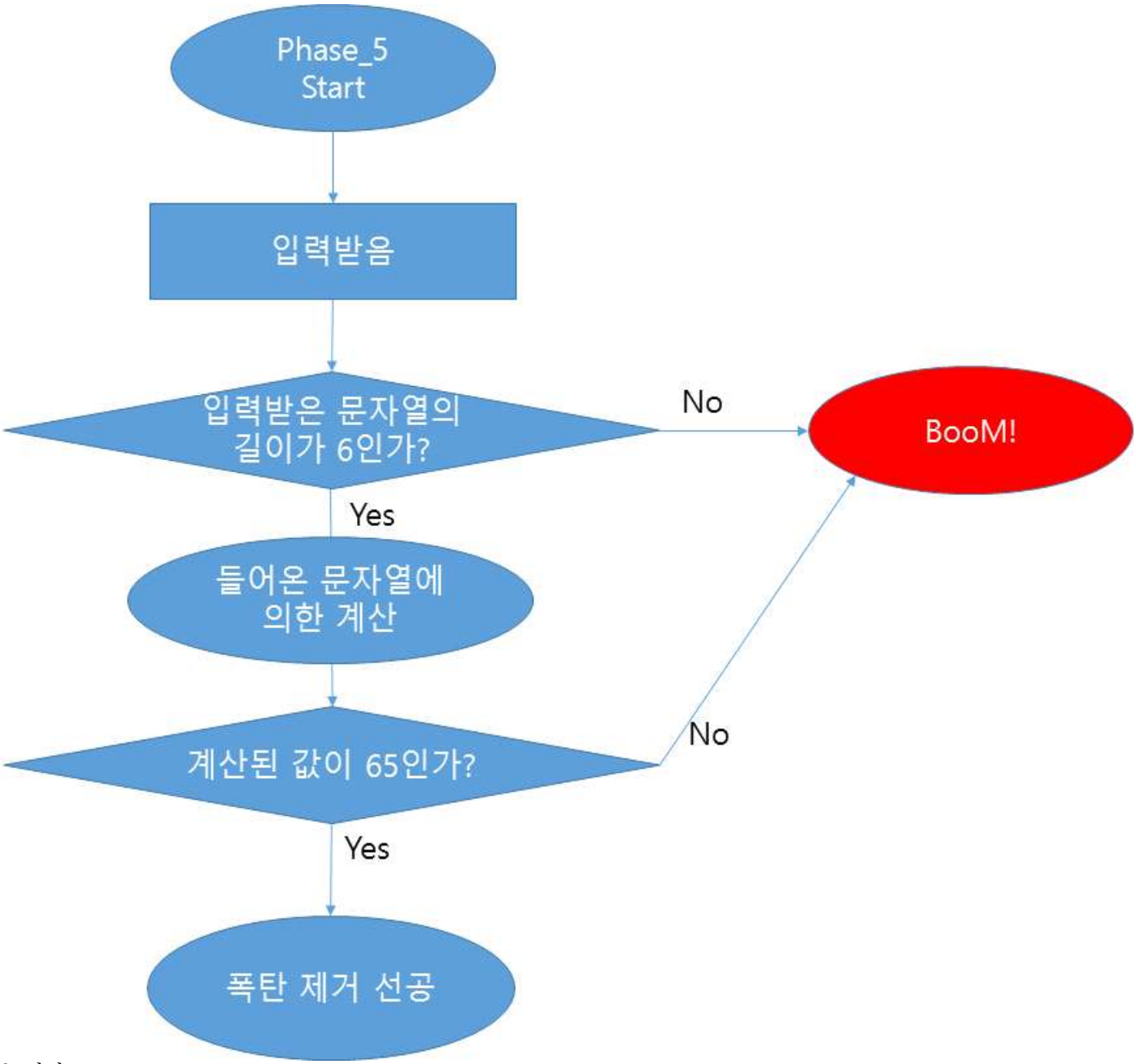
1,4,10,11,12,15번째 배열을 선택하면 된다.

그럼 이제 아스키코드로 하위4비트가 1,4,10,11,12,15를 가르키는 문자열을 찾으면된다

0110000	0	1010000	P
0110001	1	1010001	Q
0110010	2	1010010	R
0110011	3	1010011	S
0110100	4	1010100	T
0110101	5	1010101	U
0110110	6	1010110	V
0110111	7	1010111	W
0111000	8	1011000	X
0111001	9	1011001	Y
0111010	:	1011010	Z
0111011	;	1011011	[
0111100	<	1011100	₩
0111101	=	1011101	]
0111110	>	1011110	^
0111111	?	1011111	_

(아스키 코드값중 일부이다)
15는 ?
12는 <
11는 ;
10는 :
4는 T
1은 Q로 선택하면된다.
그러므로 정답은 ?<::TQ이다.
물론 그 외에도 다양한 답들이 존재할수있다

2-FlowChart



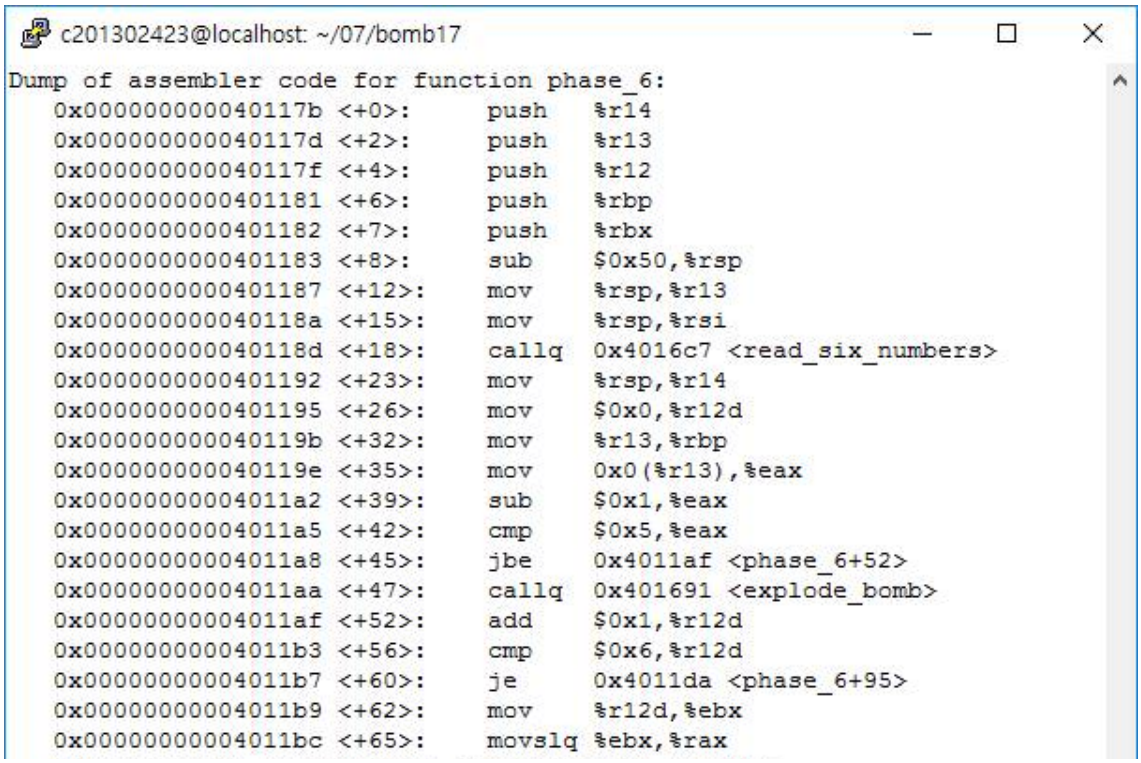
3-정답

```
c201302423@localhost: ~/07/bomb17
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
0 14
Halfway there!
14 45 DrEvil
So you got that one.  Try this one.
?<;:TQ
Good work!  On to the next...
```

4-6 phase_6

1-해결방법

코드가 길어 나눠서 설명하겠다.

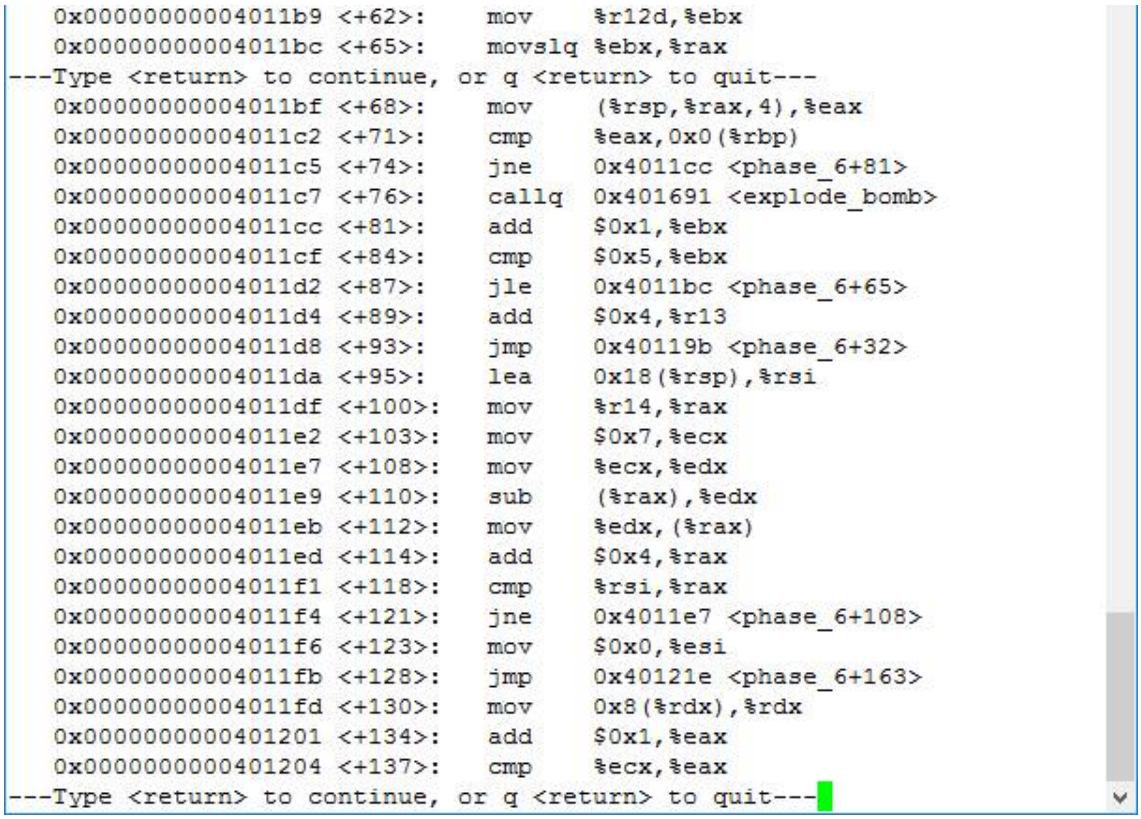


일단 옮긴 rsp를 r13과 rsi에 저장한다 그후

read six numbers에서 6개의 숫자를 구한걸 r14에 저장후 r12d에는 0을 rbp에는 r13즉 아까 저장한 rsp를 저장한다.

그 r13을 다시 eax에 저장하고 eax에서 1을 뺀후 5보다 크다면 폭탄이 터집니다.

r12d에 1을 더한후 같다면 점프하지만 최초 실행은 r12d가 0에서 1로가기 때문에 아마 6번 반복되는 구조일 것이다.



ebx에 r12를 저장하고

ebx를 rax로 옮기면서 비트를 확장한다

eax에 %rsp+4*%rax의 값을 저장한다 현재 rsp는 첫 번째 숫자이고 rax는 1임으로 eax에 두 번째 숫자가 저장된다.

두 번째 숫자가 0아니라면 ebx값을 1 증가시키고 5와 비교한다. 작거나 같으면 다시 위로 올라가서 루프를 돈다.

이 루프문은

그다음 r13를 한칸전진시킨다 원래 rsp였기 때문에 이제 두 번째 숫자를 가리키게 된다.

그다음 다시 루프를시작하는데 r12가 6이 될 때까지 돈다 즉 총 6번을 수행한후에 +95에서 시작하게된다.

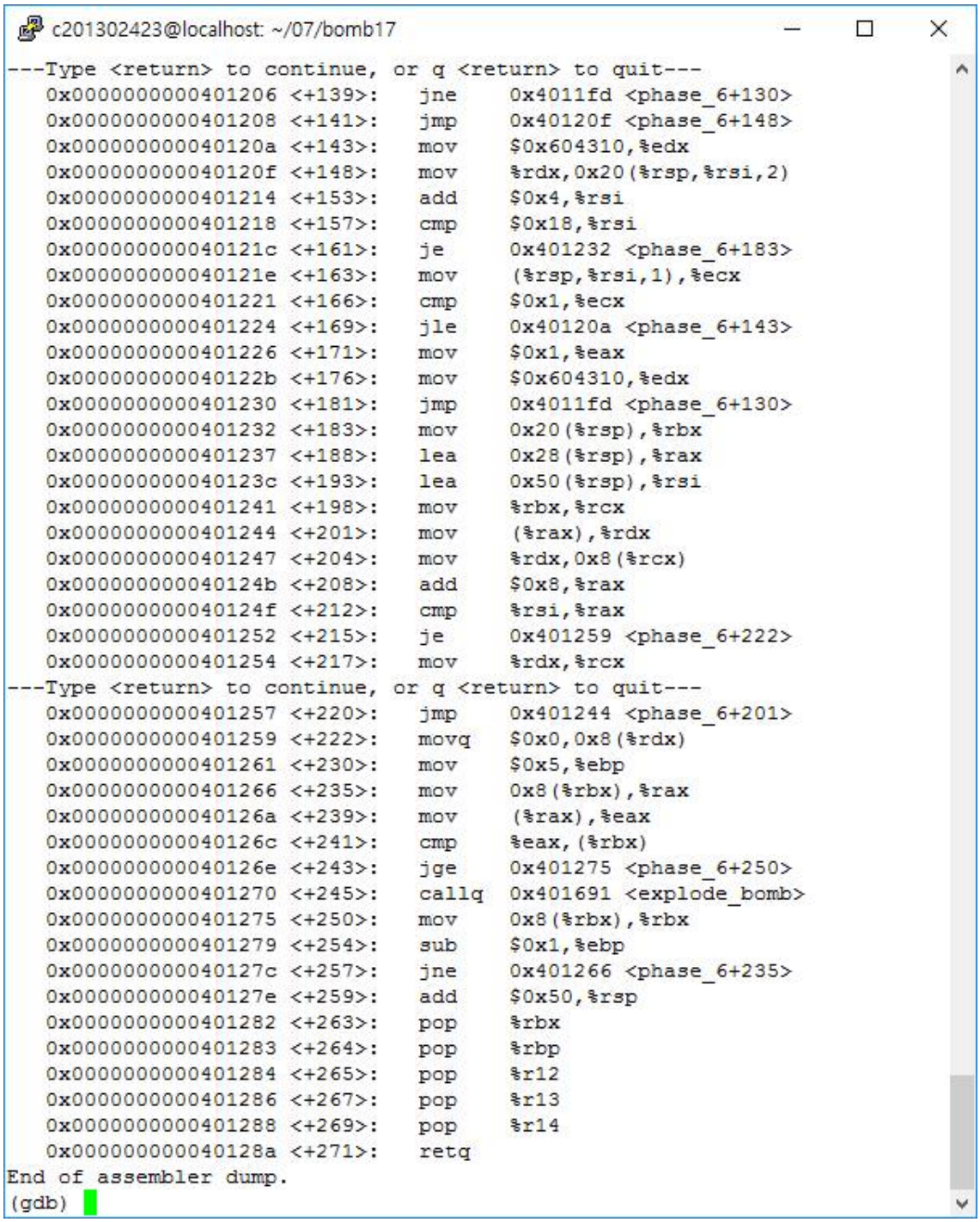
rsi는 rsp+24 6번째 숫자 다음 위치의 주소를 저장한다

rax는 r14의값 즉 최초의 rsp값을 저장하고 ecx는 7 edx는 ecx의값을 받는다

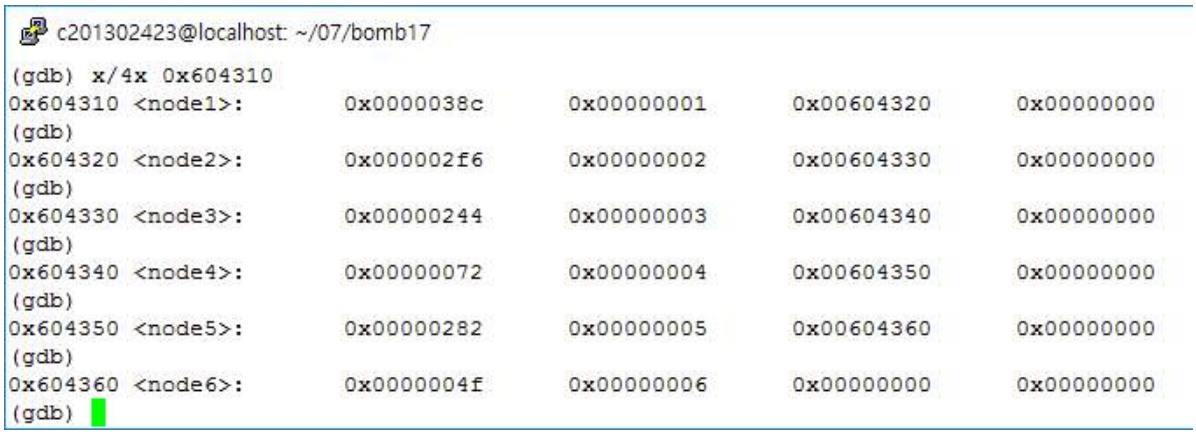
edx는 %rax값만큼 뺀걸 저장한다 rax를 한칸씩 전진하면서 반복하게되는데 7에서 최초의 입력만큼의 값들은 빼서 값들을 바꾸는 과정이다

예를들어 1 2 3 4 5 6 이 들어오면 6 5 4 3 2 1로 바꾼다는 말이다.

그다음 163으로 점프한다.



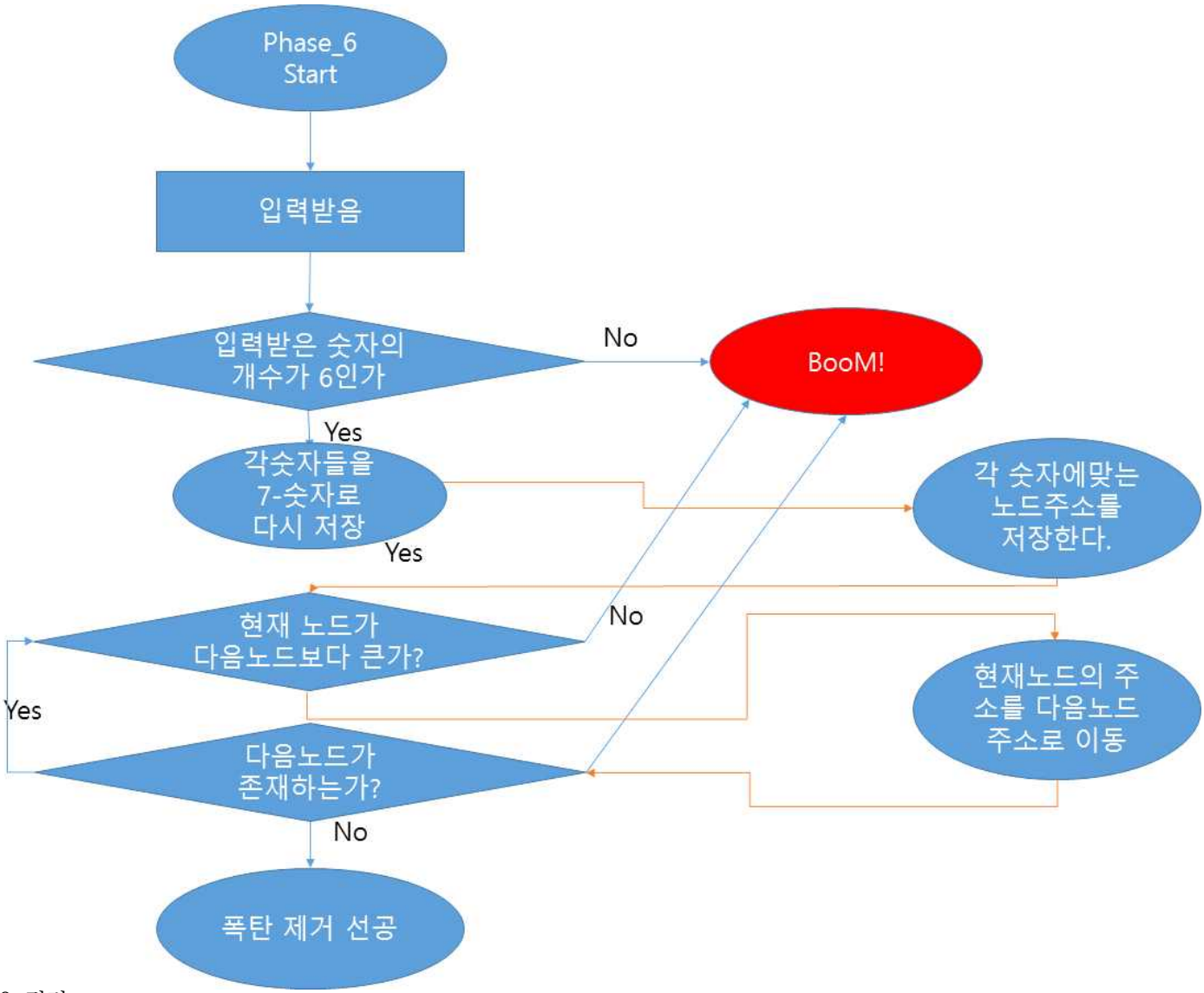
먼저 이함수가 어떻게 해야 끝나는지 판단하기위해 밑에서부터 확인해보면
ebp에 5를 저장후 rbx의 값에서 rbx+8주소의 값에서 을 뺐을 때 크거나 같으면 rbx를 한칸씩옮기고 ebp를 1씩 빼는 과정이다 5번을 확인한후에 프로그램이 종료된다. 아마도 rbx의 값들을 가지고 대소 비교를하는데
첫숫자가 제일크고 점차 줄어드는 내림차순 일 것이다.
일단 +143에서 edx에 들어가는 0x604310을 조사해보면 노드가 나오는데
0x604310을 분석해보면



아마도 노드를 가지고 폭탄을 해체할거 같다.
다시 +163에 돌아와 설명을하면 ecx = rsp+rsi인데 ecx가 1보다 작거나 같으면 +143으로 간다
+143에서 주소를 edx에 넣은뒤 그 rdx를 rsp+2*rsi+0x20에 저장한다 rsi를 4씩 더하면서 24가될 때 까지 함으로 6번 반복하면된다 rsp+0x20+0,8,16,24..에는 rdx의 값들이 저장되어있다.
그다음 +183으로간다
일단 rsp+0x20만큼을 rbx에 +0x28만큼을 rax에 저장한다
rdx와 rdx의 그다음 값을 가르키는 뜻이다 rbx값을 rcx에, rax값을 rdx에 저장한다
그리고 rcx+8에 rdx를 저장한다 rax에 8을 더하고 이 rax가 rsi와 같은지 비교한다 같지않다면 같을 때 까지 반복하게된다. 최초의 rax가 rsp+40이였고 rsi가 rsp+80임으로 5번 실행하게 될 것이다.
이과정은 받은 노드의 값들을 rsp+40부터 받은 순서대로 저장하는 것이다
이제 222로가서 분석해보면
rdx+8에 0을 저장하고 rbx+8의값 즉 현재 가르키는 다음 노드의 값을 rax에 저장한다
rbx-rax 현재노드-다음노드가 양수이면 폭탄이 터지지않고 rbx를 한칸 전진시켜 다음 노드를 가르키게된다
5번할 때 까지 이 조건이 모두 만족하면 폭탄을 해제하게된다.

노드를 내림차순으로 정리하면된다
node 1 -> 2-> 5-> 3-> 4-> 6 순이다
하지만 중간에 처음에 들어온값에서 7-x를 했기 때문에
입력해야할값은 6 5 2 4 3 1이다

2-FlowChart



3-정답

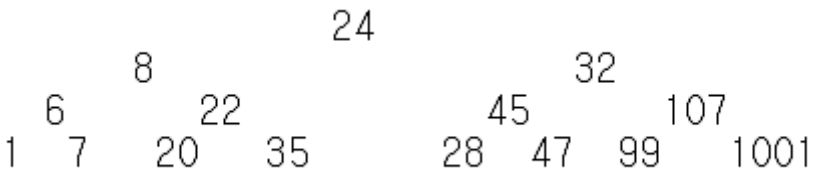
```
c201302423@localhost: ~/07/bomb17
(gdb) r
Starting program: /home/sys02/c201302423/07/bomb17/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
He is evil and fits easily into most overhead storage bins.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
0 14
Halfway there!
14 45 DrEvil
So you got that one.  Try this one.
?<;:TQ
Good work!  On to the next...
6 5 2 4 3 1
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```

4-7 secret_phase

1-해결방법

```
c201302423@localhost: ~/07/bomb17
Breakpoint 1 at 0x401691
(gdb) disas secret_phase
Dump of assembler code for function secret_phase:
   0x00000000004012c9 <+0>:     push   %rbx
   0x00000000004012ca <+1>:     callq  0x401709 <read_line>
   0x00000000004012cf <+6>:     mov    $0xa,%edx
   0x00000000004012d4 <+11>:    mov    $0x0,%esi
   0x00000000004012d9 <+16>:    mov    %rax,%rdi
   0x00000000004012dc <+19>:    callq  0x400c90 <strtol@plt>
   0x00000000004012e1 <+24>:    mov    %rax,%rbx
   0x00000000004012e4 <+27>:    lea    -0x1(%rax),%eax
---Type <return> to continue, or q <return> to quit---
   0x00000000004012e7 <+30>:    cmp    $0x3e8,%eax
   0x00000000004012ec <+35>:    jbe    0x4012f3 <secret_phase+42>
   0x00000000004012ee <+37>:    callq  0x401691 <explode_bomb>
   0x00000000004012f3 <+42>:    mov    %ebx,%esi
   0x00000000004012f5 <+44>:    mov    $0x604130,%edi
   0x00000000004012fa <+49>:    callq  0x40128b <fun7>
   0x00000000004012ff <+54>:    cmp    $0x3,%eax
   0x0000000000401302 <+57>:    je     0x401309 <secret_phase+64>
   0x0000000000401304 <+59>:    callq  0x401691 <explode_bomb>
---Type <return> to continue, or q <return> to quit---
   0x0000000000401309 <+64>:    mov    $0x4026d0,%edi
   0x000000000040130e <+69>:    callq  0x400bc0 <puts@plt>
   0x0000000000401313 <+74>:    callq  0x40182f <phase_defused>
   0x0000000000401318 <+79>:    pop    %rbx
   0x0000000000401319 <+80>:    retq
End of assembler dump.
(gdb)
```

일단 0x604130을 분석결과 완전이진트리가 나왔다



24- 8 -32-22-45-6-107-28-1-99-35-7-20-47-1001

이런식으로 구현되어있다

```
c201302423@localhost: ~/07/bomb17
Dump of assembler code for function fun7:
   0x000000000040128b <+0>:     sub    $0x8,%rsp
   0x000000000040128f <+4>:     test   %rdi,%rdi
   0x0000000000401292 <+7>:     je     0x4012bf <fun7+52>
   0x0000000000401294 <+9>:     mov    (%rdi),%edx
   0x0000000000401296 <+11>:    cmp    %esi,%edx
   0x0000000000401298 <+13>:    jle    0x4012a7 <fun7+28>
   0x000000000040129a <+15>:    mov    0x8(%rdi),%rdi
   0x000000000040129e <+19>:    callq  0x40128b <fun7>
---Type <return> to continue, or q <return> to quit---
   0x00000000004012a3 <+24>:    add    %eax,%eax
   0x00000000004012a5 <+26>:    jmp    0x4012c4 <fun7+57>
   0x00000000004012a7 <+28>:    mov    $0x0,%eax
   0x00000000004012ac <+33>:    cmp    %esi,%edx
   0x00000000004012ae <+35>:    je     0x4012c4 <fun7+57>
   0x00000000004012b0 <+37>:    mov    0x10(%rdi),%rdi
   0x00000000004012b4 <+41>:    callq  0x40128b <fun7>
   0x00000000004012b9 <+46>:    lea    0x1(%rax,%rax,1),%eax
   0x00000000004012bd <+50>:    jmp    0x4012c4 <fun7+57>
---Type <return> to continue, or q <return> to quit---
   0x00000000004012bf <+52>:    mov    $0xffffffff,%eax
   0x00000000004012c4 <+57>:    add    $0x8,%rsp
   0x00000000004012c8 <+61>:    retq
End of assembler dump.
(gdb) Quit
(gdb)
```

fun7을 확인해보자

일단 rdi값이 0이면 eax는 -1로 리턴되고 종료된다. 아니라면 rdi값을 edx에 넣고 esi와 edx를 비교후 esi가 더클경우에는 28로 점프 아닐경우에는 rdi를 한칸 전진시킨후 다수 fun7을 수행후 리턴값을 *2한후 종료된다.

+28에가면 edx와 esi가 같으면 종료하고 그게아니라면 rdi를 16칸전진 시킨후 fun7을 실행한다

그후 2*rax+1값을 eax에 저장한후 끝낸다.

```
   0x00000000004012ff <+54>:    cmp    $0x3,%eax
   0x0000000000401302 <+57>:    je     0x401309 <secret_phase+64>
   0x0000000000401304 <+59>:    callq  0x401691 <explode_bomb>
---Type <return> to continue, or q <return> to quit---
   0x0000000000401309 <+64>:    mov    $0x4026d0,%edi
   0x000000000040130e <+69>:    callq  0x400bc0 <puts@plt>
   0x0000000000401313 <+74>:    callq  0x40182f <phase_defused>
   0x0000000000401318 <+79>:    pop    %rbx
   0x0000000000401319 <+80>:    retq
End of assembler dump.
(gdb)
```

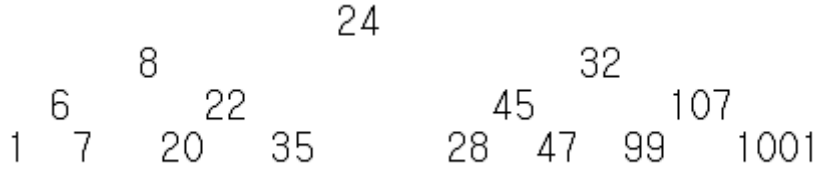
fun7의 리턴값이 3이라면 제거에 성공하고 0x4026d0은 분석결과 제거 성공 메시지이다.

3이 나오게할려면 2*rax+1두번 실행되면된다.

이 func7를 c로바꿔서 표현하면

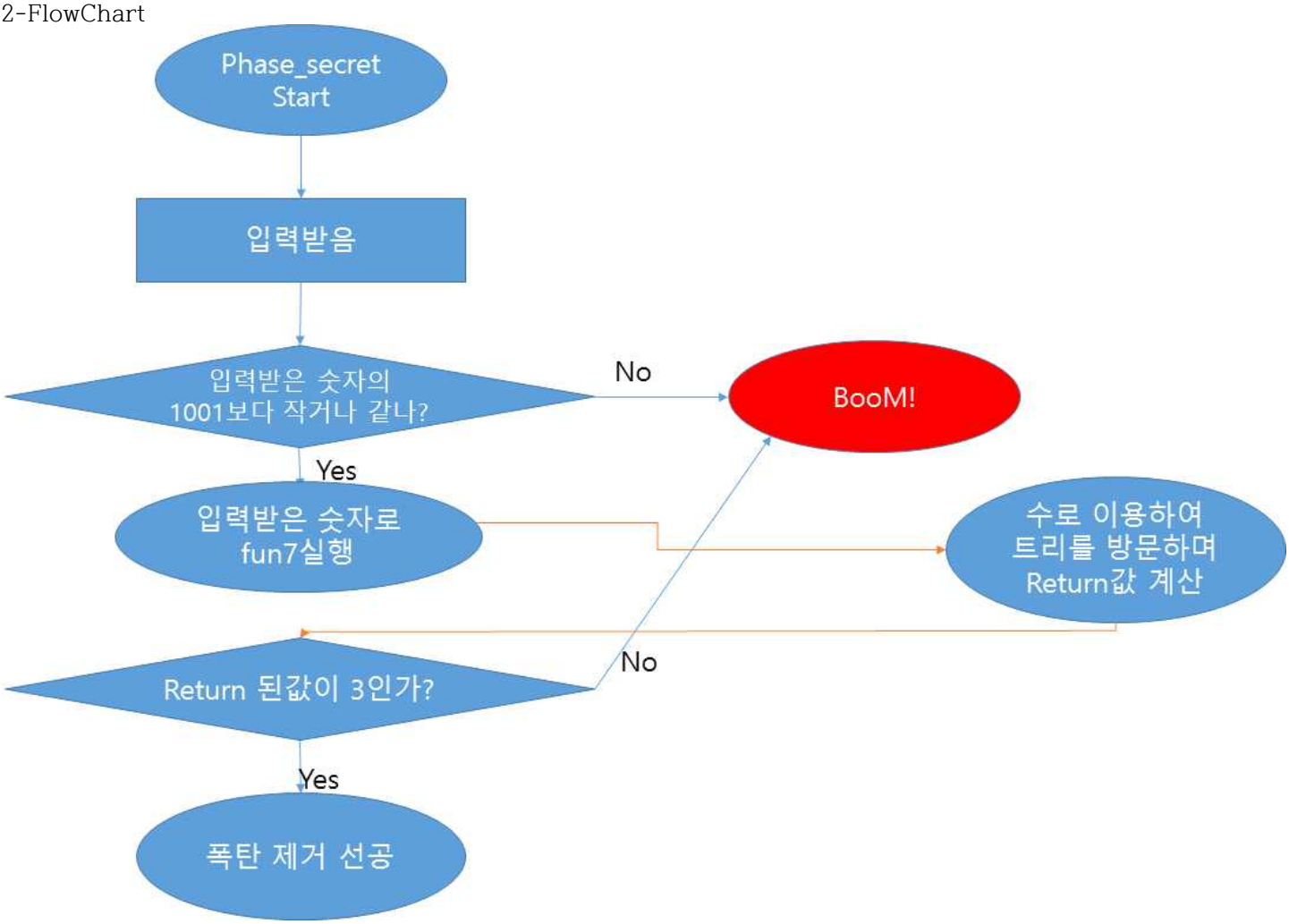
```
int fun7(int *x, int y)
{
        if (x == NULL)
                return -1;
        int ret = 0;
        if (*x - y < 0)
        {
                ret = fun7(*(x + 0x10), y);
                ret = ret * 2 + 1;
        }
        else if (*x == y)
        {
                return 0;
        }
        else
        {
                ret = fun7(*(x + 0x8), b);
                ret *= 2;
        }
}
```

이다 즉 현재 포인터값보다 클 경우 오른쪽 트리로 가고 2*x+1을 리턴
작을경우에는 왼쪽으로 가고 2*x를 리턴
같을 경우 0을 리턴한다
두 번 오른쪽으로 간다음 멈춰야 하기 때문에



107이 정답이다

2-FlowChart



3-정답

```
c201302423@localhost: ~/07/bomb17
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
He is evil and fits easily into most overhead storage bins.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
0 14
Halfway there!
14 45 DrEvil
So you got that one.  Try this one.
?<;:TQ
Good work!  On to the next...
6 5 2 4 3 1
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.
```

5.고찰 및 느낀점

답만 찾는데만 생각하면 생각보다 빨리 답을 찾을수 있지만 하나하나 분석하면서 흐름도를 쫓아가고 명령어 하나하나 생각하면서 하니 보고서를 작성하는데 오래걸렸다
물론 아직도 완벽하게 이해하지는 못했다.
그래도 이번 과제를 하면서 좀더 어셈블리어의 구조나 본질에 대해서 배운점이 많은 것 같다.
그리고 문제를 풀어가는 과정이 흥미로웠다.