시스템 프로그래밍
– BOMB LAB

수업명 | 시스템프로그래밍

지도교수 | 박문주

작성날짜 | 2019.11.11

학번 | 201701524

이름 | 강은선

목차

1. 공통사항

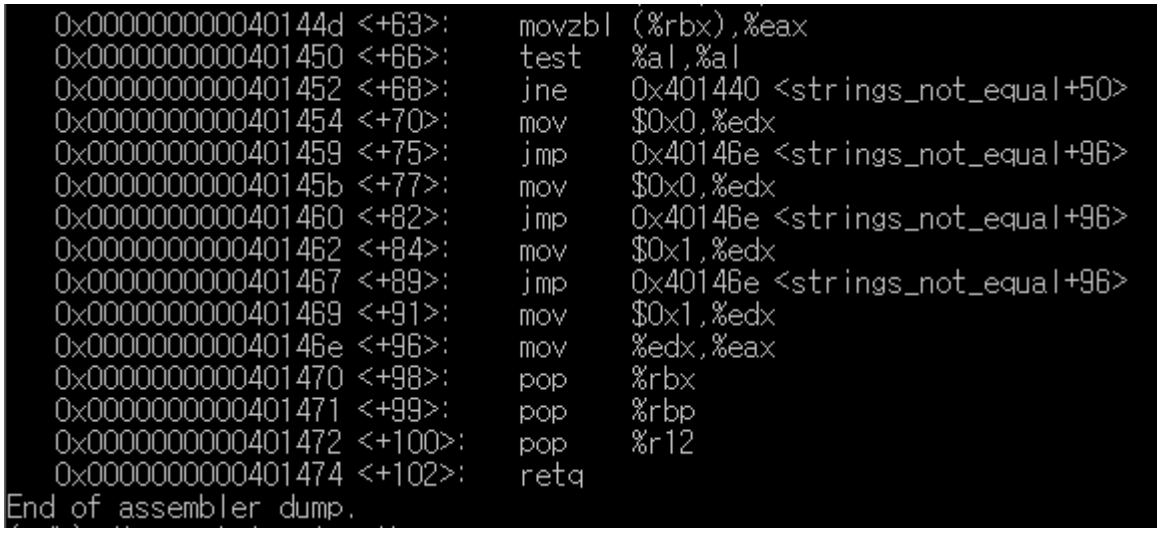
디버거는 gdb 를 사용하였으며 layout reg 를 통해 레지스터와 어셈블리어를 확인하며 진행하였다. 이때 breakpoint 를 걸어 놓고 ni 를 통해 한 줄씩 실행시켜 혹시 틀린 값을 넣었더라도 bomb 가 나지 않도록 대처했다. 또한, bomb 를 실행할 때마다 정답을 입력하는 것은 실수가 발생할 가능성이 있으므로 solution.txt 파일을 만들어 이곳에 정답을 입력한 후 해당 파일을 넘겨주며 실행시켰다.

2. phase_1

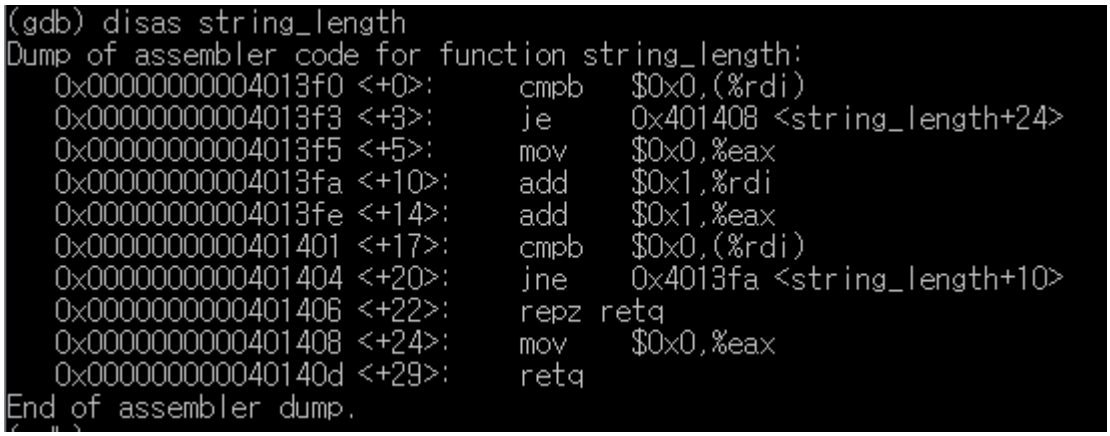
```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400f2d <+0>:     sub    $0x8,%rsp
   0x0000000000400f31 <+4>:     mov    $0x4026b0,%esi
   0x0000000000400f36 <+9>:     callq  0x40140e <strings_not_equal>
   0x0000000000400f3b <+14>:    test   %eax,%eax
   0x0000000000400f3d <+16>:    je     0x400f44 <phase_1+23>
   0x0000000000400f3f <+18>:    callq  0x4016e2 <explode_bomb>
   0x0000000000400f44 <+23>:    add    $0x8,%rsp
   0x0000000000400f48 <+27>:    retq
End of assembler dump.
```

<그림 1. phase_1 어셈블리어>

```
(gdb) disas strings_not_equal
Dump of assembler code for function strings_not_equal:
   0x000000000040140e <+0>:     push   %r12
   0x0000000000401410 <+2>:     push   %rbp
   0x0000000000401411 <+3>:     push   %rbx
   0x0000000000401412 <+4>:     mov    %rdi,%rbx
   0x0000000000401415 <+7>:     mov    %rsi,%rbp
   0x0000000000401418 <+10>:    callq  0x4013f0 <string_length>
   0x000000000040141d <+15>:    mov    %eax,%r12d
   0x0000000000401420 <+18>:    mov    %rbp,%rdi
   0x0000000000401423 <+21>:    callq  0x4013f0 <string_length>
   0x0000000000401428 <+26>:    mov    $0x1,%edx
   0x000000000040142d <+31>:    cmp    %eax,%r12d
   0x0000000000401430 <+34>:    jne    0x40146e <strings_not_equal+96>
   0x0000000000401432 <+36>:    movzbl (%rbx),%eax
   0x0000000000401435 <+39>:    test   %al,%al
   0x0000000000401437 <+41>:    je     0x40145b <strings_not_equal+77>
   0x0000000000401439 <+43>:    cmp    0x0(%rbp),%al
   0x000000000040143c <+46>:    je     0x401445 <strings_not_equal+55>
   0x000000000040143e <+48>:    jmp    0x401462 <strings_not_equal+84>
   0x0000000000401440 <+50>:    cmp    0x0(%rbp),%al
   0x0000000000401443 <+53>:    jne    0x401469 <strings_not_equal+91>
   0x0000000000401445 <+55>:    add    $0x1,%rbx
   0x0000000000401449 <+59>:    add    $0x1,%rbp
```



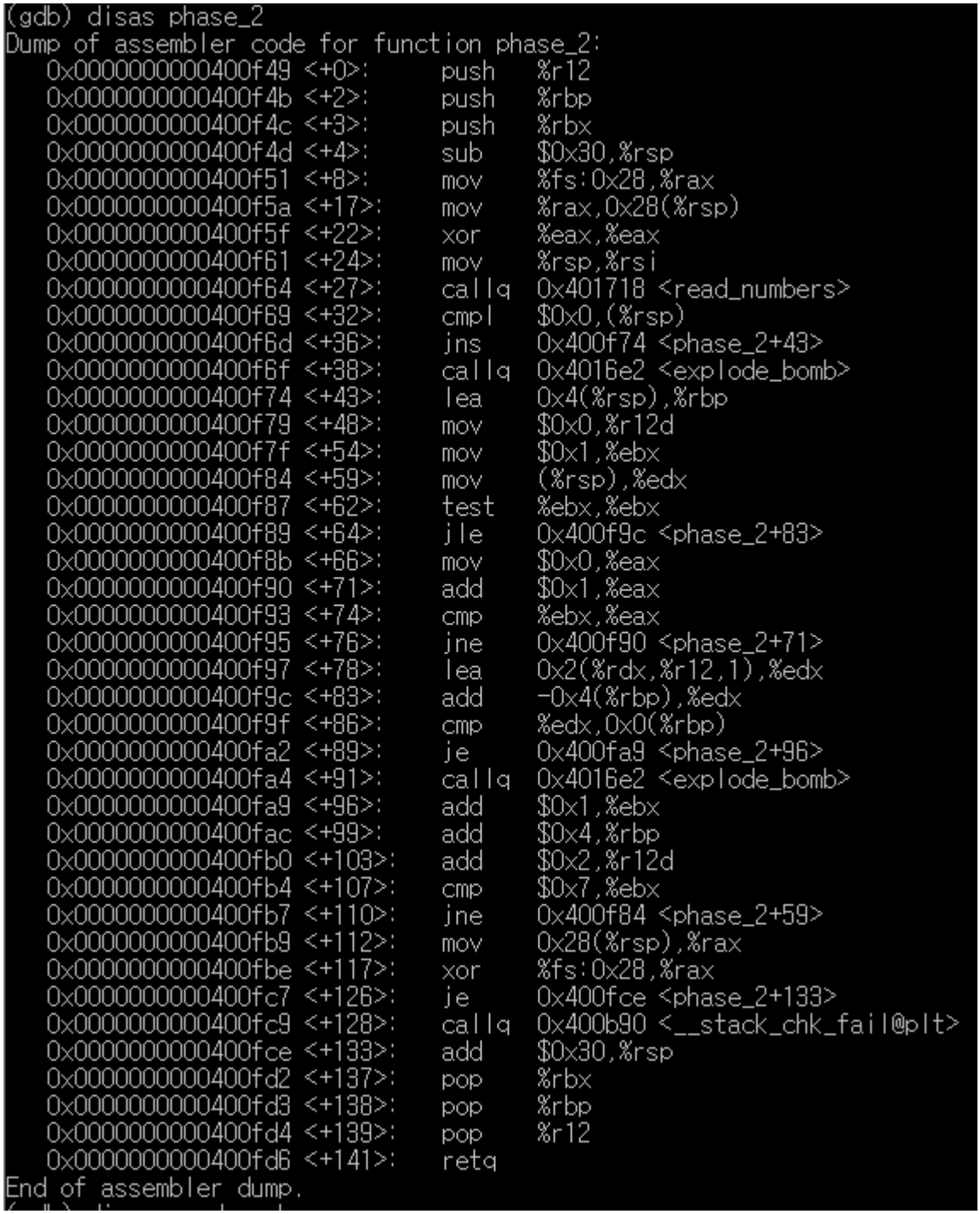
<그림 2. strings_not_equal 어셈블리어>



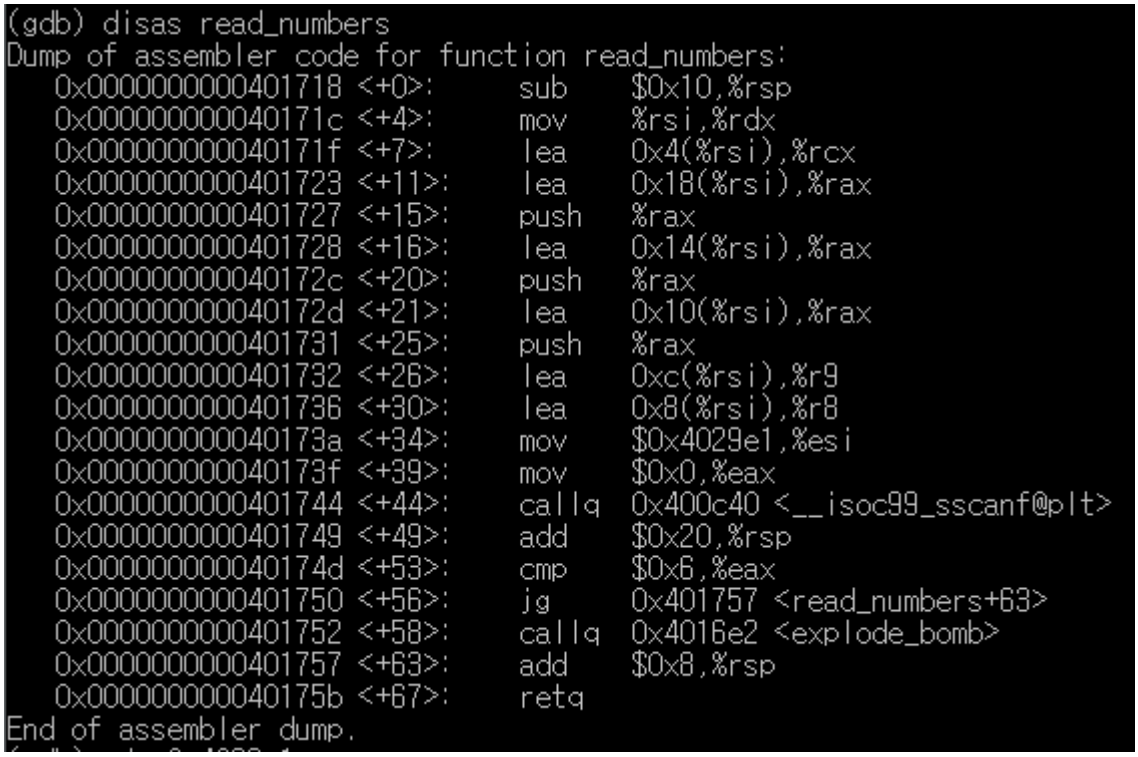
<그림 3. string_length 어셈블리어>

위의 <그림 2>에서 두개의 <string_length>가 각각 사용자가 입력한 정답과 bomb 가 터지지 않는 정답의 길이를 구하여 이 둘을 비교하는 것을 볼 수 있다. 따라서 두개의 <string_length>를 살펴보면 bomb 가 터지지 않는 정답을 구할 수 있다. 이때 검사하는 문자열의 주소를 rbp 레지스터가 담고 있으므로 해당 레지스터에 담겨있는 주소에 접근하면 "I am for medical liability at the federal level."이라는 문자열을 얻을 수 있다. 해당 문자열을 solution.txt 에 넣어 실행시키면 bomb 가 터지지 않는 것을 볼 수 있다.

3. phase_2



<그림 4. phase_2 어셈블리어>



<그림 5. read_numbers 어셈블리어>

우선 <그림 4>의 <phase_2>를 살펴보면 맨 처음 <read_numbers>를 실행시키는 것을 볼 수 있다. <그림 5>의 <read_numbers+34>에서 인자를 넘겨받아 scanf 를 실행 한 후 eax 즉, 받아온 원소의 개수가 6 개보다 크지 않으면 bomb 가 터지는 것을 알 수 있다. 이때, 넘겨받는 인자의 주소 0x4029e1 을 x/s 를 통해 살펴보면 "%d %d %d %d %d %d %d"로 bomb 가 터지지 않으려면 7 개의 정수가 필요하다는 것을 알 수 있다.

그 후 다시 <phase_2>로 돌아오면 <phase_2+32>부분에서 rsp 레지스터에 담긴 값 즉, 첫번째 정수가 0 보다 크거나 같은 경우에 bomb 가 터지지 않는다는 것을 알 수 있다.

그 후 정수의 번호를 의미하는 ebx 값이 7 이 될 때까지 <phase_2+59>부터 <phase_2+103>까지 반복하며 총 7 개의 정수를 구한다. 해당 부분을 살펴보면 정수를 구하는데 사용되는 레지스터는 r12d, edx 로 총 2 가지이다. 우선 <phase_2+48>울 통해 r12d 의 초기값은 0 임을 알 수 있다. 그 후 edx 에 첫번째 정수 값을 담는다. 그 후 <phase_2+78>으로 넘어가서(그 사이의 부분은 현재 계산하는 정수의 번호에 eax 값을 맞춰주는 부분이다.) r12 가 담고 있는 값과 rdx 가 담고 있는 값(=첫번째 값)에 2 를 더한 값을 대입한다. 그 후 현재 계산하는 정수의 전 번호 정수를 더하면 된다. 이렇게 나온 값이 사용자가 입력한 해당 번호의 정수와 일치하면 bomb 는 터지지 않는다. 즉, 각 번호의 정수는 첫번째 정수에 r12 와 이전 번호의 정수를

더한 값이다. 이때, r12 는 0 에서 시작하여 2 씩 커진다. 따라서, 첫번째 정수를 0 으로 설정했을 때 각 정수는 0 2 6 12 20 30 42 가 된다.

4. phase_3

```
Dump of assembler code for function phase_3:
   0x0000000000400fd7 <+0>:     sub    $0x18,%rsp
   0x0000000000400fdb <+4>:     mov    %fs:0x28,%rax
   0x0000000000400fe4 <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000400fe9 <+18>:    xor    %eax,%eax
   0x0000000000400feb <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000400ff0 <+25>:    mov    %rsp,%rdx
   0x0000000000400ff3 <+28>:    mov    $0x4029f0,%esi
   0x0000000000400ff8 <+33>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x0000000000400ffd <+38>:    cmp    $0x1,%eax
   0x0000000000401000 <+41>:    jg     0x401007 <phase_3+48>
   0x0000000000401002 <+43>:    callq  0x4016e2 <explode_bomb>
   0x0000000000401007 <+48>:    mov    (%rsp),%eax
   0x000000000040100a <+51>:    sub    $0x2b,%eax
   0x000000000040100d <+54>:    cmp    $0x7,%eax
   0x0000000000401010 <+57>:    ja     0x40106c <phase_3+149>
   0x0000000000401012 <+59>:    mov    %eax,%eax
   0x0000000000401014 <+61>:    jmpq   *0x402720(,%rax,8)
   0x000000000040101b <+68>:    mov    $0x133,%eax
   0x0000000000401020 <+73>:    jmp    0x401027 <phase_3+80>
   0x0000000000401022 <+75>:    mov    $0x0,%eax
   0x0000000000401027 <+80>:    sub    $0x213,%eax
   0x000000000040102c <+85>:    jmp    0x401033 <phase_3+92>
   0x000000000040102e <+87>:    mov    $0x0,%eax
   0x0000000000401033 <+92>:    add    $0x2eb,%eax
   0x0000000000401038 <+97>:    jmp    0x40103f <phase_3+104>
   0x000000000040103a <+99>:    mov    $0x0,%eax
   0x000000000040103f <+104>:   sub    $0x6f,%eax
   0x0000000000401042 <+107>:   jmp    0x401049 <phase_3+114>
   0x0000000000401044 <+109>:   mov    $0x0,%eax
   0x0000000000401049 <+114>:   add    $0x6f,%eax
   0x000000000040104c <+117>:   jmp    0x401053 <phase_3+124>
   0x000000000040104e <+119>:   mov    $0x0,%eax
   0x0000000000401053 <+124>:   sub    $0x6f,%eax
   0x0000000000401056 <+127>:   jmp    0x40105d <phase_3+134>
   0x0000000000401058 <+129>:   mov    $0x0,%eax
   0x000000000040105d <+134>:   add    $0x6f,%eax
   0x0000000000401060 <+137>:   jmp    0x401067 <phase_3+144>
   0x0000000000401062 <+139>:   mov    $0x0,%eax
   0x0000000000401067 <+144>:   sub    $0x6f,%eax
   0x000000000040106a <+147>:   jmp    0x401076 <phase_3+159>
   0x000000000040106c <+149>:   callq  0x4016e2 <explode_bomb>
   0x0000000000401071 <+154>:   mov    $0x0,%eax
   0x0000000000401076 <+159>:   cmp    0x4(%rsp),%eax
   0x000000000040107a <+163>:   je     0x401081 <phase_3+170>
   0x000000000040107c <+165>:   callq  0x4016e2 <explode_bomb>
   0x0000000000401081 <+170>:   mov    0x8(%rsp),%rax
   0x0000000000401086 <+175>:   xor    %fs:0x28,%rax
   0x000000000040108f <+184>:   je     0x401096 <phase_3+191>
   0x0000000000401091 <+186>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x0000000000401096 <+191>:   add    $0x18,%rsp
```

<그림 6. Phase_3 어셈블리어>

우선 scanf 에 넘겨주는 esi 레지스터의 주소 0x4029f0 이 담고있는 문자열 "%d %d"를 통해 정수 2 개를 입력해야 함을 알 수 있다. 또한, 2 개를 입력했을 때, eax 레지스터의 값이 1 보다 커 bomb 가 터지지 않는다.

그 후 eax 레지스터에 첫번째 원소의 값을 담고 0x2b(=43)을 뺀 값이 7 보다 커야 한다. 이때, ja 는 unsigned 이므로 eax 레지스터 값이 0 보다 커야한다. 따라서 43 에서 50 사이의 정수인 것이다. 이때, 첫번째 정수의 값을 50 이라 정하면 그 후의 <phase_3+61> 줄을 통해 0x401062 로 점프하게 되므로 eax 가 -111 이 되고 <phase_3+159>를 통해 그것이 곧 두번째 정수와 같아야 한다는 것을 알 수 있다. 따라서 두 정수는 50, -111 이다.

5. phase_4

```
(gdb) disas phase_4
Dump of assembler code for function phase_4:
   0x00000000004010d9 <+0>:     sub    $0x18,%rsp
   0x00000000004010dd <+4>:     mov    %fs:0x28,%rax
   0x00000000004010e6 <+13>:    mov    %rax,0x8(%rsp)
   0x00000000004010eb <+18>:    xor    %eax,%eax
   0x00000000004010ed <+20>:    lea    0x4(%rsp),%rcx
   0x00000000004010f2 <+25>:    mov    %rsp,%rdx
   0x00000000004010f5 <+28>:    mov    $0x4029f0,%esi
   0x00000000004010fa <+33>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x00000000004010ff <+38>:    cmp    $0x2,%eax
   0x0000000000401102 <+41>:    jne    0x40110f <phase_4+54>
   0x0000000000401104 <+43>:    mov    (%rsp),%eax
   0x0000000000401107 <+46>:    sub    $0xb,%eax
   0x000000000040110a <+49>:    cmp    $0xe,%eax
   0x000000000040110d <+52>:    jbe    0x401114 <phase_4+59>
   0x000000000040110f <+54>:    callq  0x4016e2 <explode_bomb>
   0x0000000000401114 <+59>:    mov    $0x19,%edx
   0x0000000000401119 <+64>:    mov    $0xb,%esi
   0x000000000040111e <+69>:    mov    (%rsp),%edi
   0x0000000000401121 <+72>:    callq  0x40109b <func4>
   0x0000000000401126 <+77>:    cmp    $0x6,%eax
   0x0000000000401129 <+80>:    jne    0x401132 <phase_4+89>
   0x000000000040112b <+82>:    cmpl   $0x6,0x4(%rsp)
   0x0000000000401130 <+87>:    je     0x401137 <phase_4+94>
   0x0000000000401132 <+89>:    callq  0x4016e2 <explode_bomb>
   0x0000000000401137 <+94>:    mov    0x8(%rsp),%rax
   0x000000000040113c <+99>:    xor    %fs:0x28,%rax
   0x0000000000401145 <+108>:   je     0x40114c <phase_4+115>
   0x0000000000401147 <+110>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x000000000040114c <+115>:   add    $0x18,%rsp
   0x0000000000401150 <+119>:   retq
End of assembler dump.
```

<그림 7. phase_4 어셈블리어>

<그림 7>의 <phase_4+28>을 보면 5 에서 설명한 phase_3 와 동일한 주소를 scanf 에 넘겨주는 것을 알 수 있다. 또한, <phase_4+38>에서 개수가 2 개가 아니면 bomb 로 넘어간다. 즉, 이번에도 정수 2 개가 정답인 것이다.

그 후 eax 에 첫번째 정수를 대입하고 11 을 뺀 값이 14 보다 작거나 같고 0 이상(jbe 는 unsigned 이다.)이므로 첫번째 정수가 11 과 25 사이의 정수임을 알 수 있다. 그 후 <func4>의 기능을 살펴보면 두 수의 중간 값보다 key 가 크면 rax 레지트서의 값에 곱하기 2, 작으면 곱하기 2 더하기 1 을 하는 것을 알 수 있다. 이때, 처음 시작은 11 과 25 사이의 값이므로 두 부분으로 나누어 오른쪽 왼쪽 왼쪽 오른쪽을 진행하면 17 이 나온다. 또한 <phase_4+82>를 통해 두번째 정수는 6 임을 알 수 있다. 따라서 두 정수는 17, 6 이다.

6. phase_5

```
(gdb) disas phase_5
Dump of assembler code for function phase_5:
   0x0000000000401151 <+0>:     sub    $0x18,%rsp
   0x0000000000401155 <+4>:     mov    %fs:0x28,%rax
   0x000000000040115e <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000401163 <+18>:    xor    %eax,%eax
   0x0000000000401165 <+20>:    lea    0x4(%rsp),%rcx
   0x000000000040116a <+25>:    mov    %rsp,%rdx
   0x000000000040116d <+28>:    mov    $0x4029f0,%esi
   0x0000000000401172 <+33>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x0000000000401177 <+38>:    cmp    $0x1,%eax
   0x000000000040117a <+41>:    jg     0x401181 <phase_5+48>
   0x000000000040117c <+43>:    callq  0x4016e2 <explode_bomb>
   0x0000000000401181 <+48>:    mov    (%rsp),%eax
   0x0000000000401184 <+51>:    and    $0xf,%eax
   0x0000000000401187 <+54>:    mov    %eax,(%rsp)
   0x000000000040118a <+57>:    cmp    $0xf,%eax
   0x000000000040118d <+60>:    je     0x4011be <phase_5+109>
   0x000000000040118f <+62>:    mov    $0x0,%ecx
   0x0000000000401194 <+67>:    mov    $0x0,%edx
   0x0000000000401199 <+72>:    add    $0x1,%edx
   0x000000000040119c <+75>:    cltq
   0x000000000040119e <+77>:    mov    0x402760(,%rax,4),%eax
   0x00000000004011a5 <+84>:    add    %eax,%ecx
   0x00000000004011a7 <+86>:    cmp    $0xf,%eax
   0x00000000004011aa <+89>:    jne    0x401199 <phase_5+72>
   0x00000000004011ac <+91>:    movl   $0xf,(%rsp)
   0x00000000004011b3 <+98>:    cmp    $0xf,%edx
   0x00000000004011b6 <+101>:   jne    0x4011be <phase_5+109>
   0x00000000004011b8 <+103>:   cmp    0x4(%rsp),%ecx
   0x00000000004011bc <+107>:   je     0x4011c3 <phase_5+114>
   0x00000000004011be <+109>:   callq  0x4016e2 <explode_bomb>
   0x00000000004011c3 <+114>:   mov    0x8(%rsp),%rax
   0x00000000004011c8 <+119>:   xor    %fs:0x28,%rax
   0x00000000004011d1 <+128>:   je     0x4011d8 <phase_5+135>
   0x00000000004011d3 <+130>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x00000000004011d8 <+135>:   add    $0x18,%rsp
   0x00000000004011dc <+139>:   retq
End of assembler dump.
```

<그림 8. phase_5 어셈블리어>

<phase_5> 역시 앞서 푼 <phase_3>과 <phase_4>와 같이 0x4029f0 을 인자로 가지므로 "%d %d" 즉, 정수 2 개가 정답 임을 알 수 있다.

<phase_5>에서는 사용자가 입력한 정답의 하위 4 비트만 사용하며, eax 레지스터에 저장한 그 값이 15 가 되면 bomb 로 넘어가게 된다. 그 후 일련의 과정을 통해 5, 12, 3, 7, 11, 13, 9, 4, 8, 0, 10, 1, 2, 14, 6, 15 순으로 값이 변하고 그 횟수가 edx 에 저장된다. 이때, edx 가 15 가

아니거나 eax 의 값이 15 인 경우 bomb 로 넘어가게 되므로 첫번째 정수는 15 번 시행 후 15 가 되는 5 가 되어야 한다. 또한, 모든 시행을 마친 후의 ecx 값이 두번째 정수와 같아야 하므로 두 정수는 5, 115 가 된다.

7. phase_6

```
(gdb) disas phase_6
Dump of assembler code for function phase_6:
   0x00000000004011dd <+0>:     push   %r13
   0x00000000004011df <+2>:     push   %r12
   0x00000000004011e1 <+4>:     push   %rbp
   0x00000000004011e2 <+5>:     push   %rbx
   0x00000000004011e3 <+6>:     sub    $0x68,%rsp
   0x00000000004011e7 <+10>:    mov    %fs:0x28,%rax
   0x00000000004011f0 <+19>:    mov    %rax,0x58(%rsp)
   0x00000000004011f5 <+24>:    xor    %eax,%eax
   0x00000000004011f7 <+26>:    mov    %rsp,%rsi
   0x00000000004011fa <+29>:    callq  0x401718 <read_numbers>
   0x00000000004011ff <+34>:    mov    %rsp,%r12
   0x0000000000401202 <+37>:    mov    $0x0,%r13d
   0x0000000000401208 <+43>:    mov    %r12,%rbp
   0x000000000040120b <+46>:    mov    (%r12),%eax
   0x000000000040120f <+50>:    sub    $0x1,%eax
   0x0000000000401212 <+53>:    cmp    $0x6,%eax
   0x0000000000401215 <+56>:    jbe    0x40121c <phase_6+63>
   0x0000000000401217 <+58>:    callq  0x4016e2 <explode_bomb>
   0x000000000040121c <+63>:    add    $0x1,%r13d
   0x0000000000401220 <+67>:    cmp    $0x7,%r13d
   0x0000000000401224 <+71>:    je     0x401263 <phase_6+134>
   0x0000000000401226 <+73>:    mov    %r13d,%ebx
   0x0000000000401229 <+76>:    movslq %ebx,%rax
   0x000000000040122c <+79>:    mov    (%rsp,%rax,4),%eax
   0x000000000040122f <+82>:    cmp    %eax,0x0(%rbp)
   0x0000000000401232 <+85>:    jne    0x401239 <phase_6+92>
   0x0000000000401234 <+87>:    callq  0x4016e2 <explode_bomb>
   0x0000000000401239 <+92>:    add    $0x1,%ebx
   0x000000000040123c <+95>:    cmp    $0x6,%ebx
   0x000000000040123f <+98>:    jle    0x401229 <phase_6+76>
   0x0000000000401241 <+100>:   add    $0x4,%r12
   0x0000000000401245 <+104>:   jmp    0x401208 <phase_6+43>
   0x0000000000401247 <+106>:   mov    0x8(%rdx),%rdx
   0x000000000040124b <+110>:   add    $0x1,%eax
   0x000000000040124e <+113>:   cmp    %ecx,%eax
   0x0000000000401250 <+115>:   jne    0x401247 <phase_6+106>
   0x0000000000401252 <+117>:   mov    %rdx,0x20(%rsp,%rsi,2)
   0x0000000000401257 <+122>:   add    $0x4,%rsi
   0x000000000040125b <+126>:   cmp    $0x1c,%rsi
   0x000000000040125f <+130>:   jne    0x401268 <phase_6+139>
   0x0000000000401261 <+132>:   jmp    0x40127c <phase_6+159>
   0x0000000000401263 <+134>:   mov    $0x0,%esi
   0x0000000000401268 <+139>:   mov    (%rsp,%rsi,1),%ecx
   0x000000000040126b <+142>:   mov    $0x1,%eax
   0x0000000000401270 <+147>:   mov    $0x6042f0,%edx
```

```
   0x0000000000401275 <+152>:   cmp    $0x1,%ecx
   0x0000000000401278 <+155>:   jg     0x401247 <phase_6+106>
   0x000000000040127a <+157>:   jmp    0x401252 <phase_6+117>
---Type <return> to continue, or q <return> to quit---e
   0x000000000040127c <+159>:   mov    0x20(%rsp),%rbx
   0x0000000000401281 <+164>:   lea    0x28(%rsp),%rax
   0x0000000000401286 <+169>:   lea    0x58(%rsp),%rsi
   0x000000000040128b <+174>:   mov    %rbx,%rcx
   0x000000000040128e <+177>:   mov    (%rax),%rdx
   0x0000000000401291 <+180>:   mov    %rdx,0x8(%rcx)
   0x0000000000401295 <+184>:   add    $0x8,%rax
   0x0000000000401299 <+188>:   mov    %rdx,%rcx
   0x000000000040129c <+191>:   cmp    %rax,%rsi
   0x000000000040129f <+194>:   jne    0x40128e <phase_6+177>
   0x00000000004012a1 <+196>:   movq   $0x0,0x8(%rdx)
   0x00000000004012a9 <+204>:   mov    $0x6,%ebp
   0x00000000004012ae <+209>:   mov    0x8(%rbx),%rax
   0x00000000004012b2 <+213>:   mov    (%rax),%eax
   0x00000000004012b4 <+215>:   cmp    %eax,(%rbx)
   0x00000000004012b6 <+217>:   jge    0x4012bd <phase_6+224>
   0x00000000004012b8 <+219>:   callq  0x4016e2 <explode_bomb>
   0x00000000004012bd <+224>:   mov    0x8(%rbx),%rbx
   0x00000000004012c1 <+228>:   sub    $0x1,%ebp
   0x00000000004012c4 <+231>:   jne    0x4012ae <phase_6+209>
   0x00000000004012c6 <+233>:   mov    0x58(%rsp),%rax
   0x00000000004012cb <+238>:   xor    %fs:0x28,%rax
   0x00000000004012d4 <+247>:   je     0x4012db <phase_6+254>
   0x00000000004012d6 <+249>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x00000000004012db <+254>:   add    $0x68,%rsp
   0x00000000004012df <+258>:   pop    %rbx
   0x00000000004012e0 <+259>:   pop    %rbp
   0x00000000004012e1 <+260>:   pop    %r12
   0x00000000004012e3 <+262>:   pop    %r13
   0x00000000004012e5 <+264>:   retq
End of assembler dump.
```

<그림 8. phase_6 어셈블리어>

우선 <phase_6+29>에서 <phase_2>에서 사용한 <read_numbers>와 같은 함수를 호출하므로 <phase_6>역시 7 개의 정수를 입력함을 알 수 있다. 그 후 <phase_6+43>부터 <phase_6+104>까지의 일련의 과정을 통해 7 개의 정수가 1 부터 7 사이의 중복 없는 숫자임을 알 수 있다.

그 후 <phase_6+147>의 0x6042f0 을 x/28w 를 해서 보면

```
(gdb) x/28w 0x6042f0
0x6042f0 <node1>:       266     1       6308608 0
0x604300 <node2>:       452     2       6308624 0
0x604310 <node3>:       126     3       6308640 0
0x604320 <node4>:       789     4       6308656 0
0x604330 <node5>:       400     5       6308672 0
0x604340 <node6>:       534     6       6308688 0
0x604350 <node7>:       387     7       0       0
```

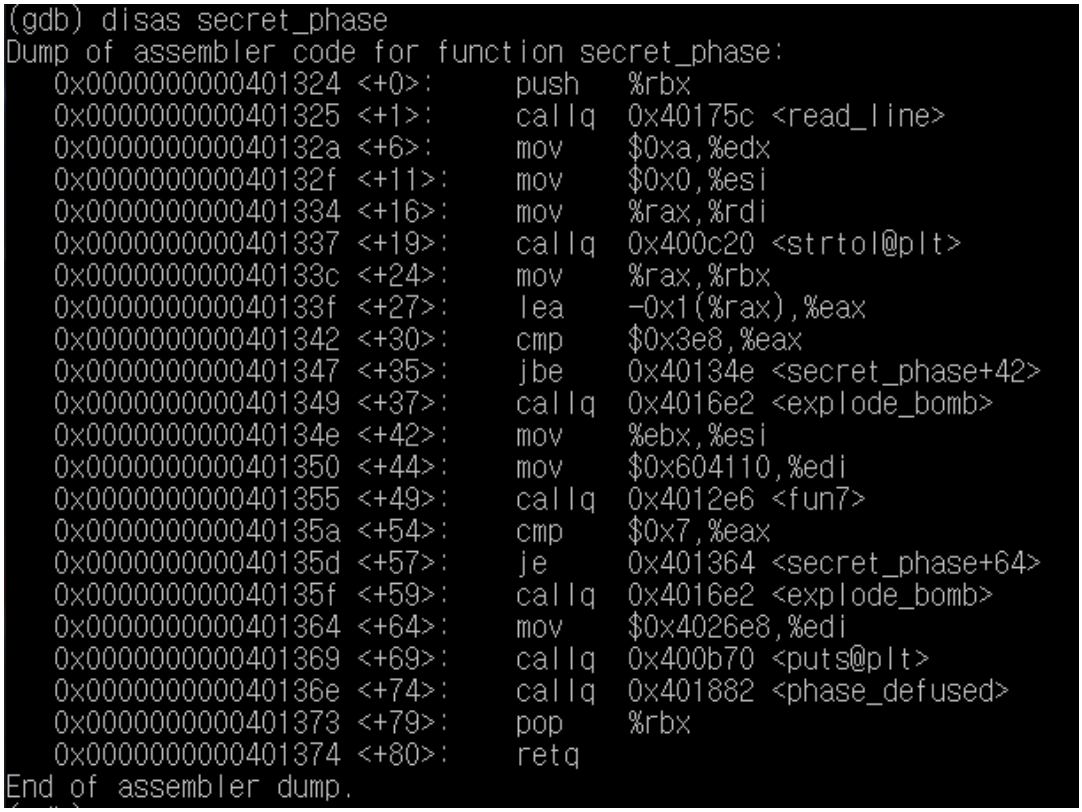
다음과 같이 7 개의 값이 담긴 노드들을 볼 수 있다. 첫번째 줄의 정수는 해당 노드의 값이고 그 다음줄이 해당 노드의 번호이므로 해당 노드의 값을 기준으로 노드를 정렬하여 그 노드의 번호를 나열하는 것임을 알 수 있다. <phase_6+217>에서 jge 즉, 앞의 것이 뒤의 것보다 더 커야 하므로 내림차순으로 정렬된다는 것을 알 수 있다. 따라서 각 노드의 값을 기준으로 내림차순으로 정렬된 4 6 2 5 7 1 3 이 정답이다.

8. secret phase

```
(gdb) disas phase_defused
Dump of assembler code for function phase_defused:
   0x0000000000401882 <+0>:     sub    $0x78,%rsp
   0x0000000000401886 <+4>:     mov    %fs:0x28,%rax
   0x000000000040188f <+13>:    mov    %rax,0x68(%rsp)
   0x0000000000401894 <+18>:    xor    %eax,%eax
   0x0000000000401896 <+20>:    mov    $0x1,%edi
   0x000000000040189b <+25>:    callq  0x4015d8 <send_msg>
   0x00000000004018a0 <+30>:    cmpl   $0x6,0x202f25(%rip)        # 0x6047cc <num_input_strings>
   0x00000000004018a7 <+37>:    jne    0x401916 <phase_defused+148>
   0x00000000004018a9 <+39>:    lea    0x10(%rsp),%r8
   0x00000000004018ae <+44>:    lea    0xc(%rsp),%rcx
   0x00000000004018b3 <+49>:    lea    0x8(%rsp),%rdx
   0x00000000004018b8 <+54>:    mov    $0x402a3a,%esi
   0x00000000004018bd <+59>:    mov    $0x6048d0,%edi
   0x00000000004018c2 <+64>:    mov    $0x0,%eax
   0x00000000004018c7 <+69>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x00000000004018cc <+74>:    cmp    $0x3,%eax
   0x00000000004018cf <+77>:    jne    0x401902 <phase_defused+128>
   0x00000000004018d1 <+79>:    mov    $0x402a43,%esi
   0x00000000004018d6 <+84>:    lea    0x10(%rsp),%rdi
   0x00000000004018db <+89>:    callq  0x40140e <strings_not_equal>
   0x00000000004018e0 <+94>:    test   %eax,%eax
   0x00000000004018e2 <+96>:    jne    0x401902 <phase_defused+128>
   0x00000000004018e4 <+98>:    mov    $0x402898,%edi
   0x00000000004018e9 <+103>:   callq  0x400b70 <puts@plt>
   0x00000000004018ee <+108>:   mov    $0x4028c0,%edi
   0x00000000004018f3 <+113>:   callq  0x400b70 <puts@plt>
   0x00000000004018f8 <+118>:   mov    $0x0,%eax
   0x00000000004018fd <+123>:   callq  0x401324 <secret_phase>
   0x0000000000401902 <+128>:   mov    $0x4028f8,%edi
   0x0000000000401907 <+133>:   callq  0x400b70 <puts@plt>
   0x000000000040190c <+138>:   mov    $0x402928,%edi
   0x0000000000401911 <+143>:   callq  0x400b70 <puts@plt>
   0x0000000000401916 <+148>:   mov    0x68(%rsp),%rax
   0x000000000040191b <+153>:   xor    %fs:0x28,%rax
   0x0000000000401924 <+162>:   je     0x40192b <phase_defused+169>
   0x0000000000401926 <+164>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x000000000040192b <+169>:   add    $0x78,%rsp
   0x000000000040192f <+173>:   retq
End of assembler dump.
```

<그림 9. phase_defused 어셈블리어>

전체 어셈블리어 코드를 쭉 보면 <phase_defused>에서 <phase_4>의 정답을 다시 검사한다는 것을 알 수 있다. 이때, 입력을 받는 인자 esi 레지스터가 담고 있는 주소 0x402a3a 를 보면 "%d %d %s"로 문자열이 추가되었음을 알 수 있다. 그 후, 비교하는 함수로 esi 레지스터가 담고 있는 0x402a34 가 넘어가는데 이 주소가 가리키는 값이 "NoOneKnowsMeBomb"이다. 즉, <phase_4>의 답 뒤에 NoOneKnowsMeBomb 를 추가하면 <secret_phase>로 넘어갈 수 있다.



<그림 10. secret_phase 어셈블리어>

<fun7>은 이진 탐색트리에서 인자로 받은 숫자를 찾는 함수인데 현재 포인터 위치가 원하는 숫자이면 0 을 리턴하고 작으면 곱하기 2, 크면 곱하기 2 더하기 1 을 리턴한다. 이때, <fun7>의 리턴값이 7 이되어야 한다. (((0*2+1)*2+1)*2+1) = 7 이므로 <fun7>에서 오른쪽으로 세번 동작해야 한다.

해당 트리를 그리기 위해 <fun7>에 인자로 넘겨준 주소를 참조하면

```
(gdb) x/60gx 0x604110
0x604110 <n1>:  0x0000000000000024      0x0000000000604130
0x604120 <n1+16>:       0x0000000000604150      0x0000000000000000
0x604130 <n21>: 0x0000000000000008      0x00000000006041b0
0x604140 <n21+16>:      0x0000000000604170      0x0000000000000000
0x604150 <n22>: 0x0000000000000032      0x0000000000604190
0x604160 <n22+16>:      0x00000000006041d0      0x0000000000000000
0x604170 <n32>: 0x0000000000000016      0x0000000000604290
0x604180 <n32+16>:      0x0000000000604250      0x0000000000000000
0x604190 <n33>: 0x000000000000002d      0x00000000006041f0
0x6041a0 <n33+16>:      0x00000000006042b0      0x0000000000000000
0x6041b0 <n31>: 0x0000000000000006      0x0000000000604210
0x6041c0 <n31+16>:      0x0000000000604270      0x0000000000000000
0x6041d0 <n34>: 0x000000000000006b      0x0000000000604230
0x6041e0 <n34+16>:      0x00000000006042d0      0x0000000000000000
0x6041f0 <n45>: 0x0000000000000028      0x0000000000000000
0x604200 <n45+16>:      0x0000000000000000      0x0000000000000000
0x604210 <n41>: 0x0000000000000001      0x0000000000000000
0x604220 <n41+16>:      0x0000000000000000      0x0000000000000000
0x604230 <n47>: 0x0000000000000063      0x0000000000000000
0x604240 <n47+16>:      0x0000000000000000      0x0000000000000000
0x604250 <n44>: 0x0000000000000023      0x0000000000000000
0x604260 <n44+16>:      0x0000000000000000      0x0000000000000000
0x604270 <n42>: 0x0000000000000007      0x0000000000000000
0x604280 <n42+16>:      0x0000000000000000      0x0000000000000000
0x604290 <n43>: 0x0000000000000014      0x0000000000000000
0x6042a0 <n43+16>:      0x0000000000000000      0x0000000000000000
0x6042b0 <n46>: 0x000000000000002f      0x0000000000000000
0x6042c0 <n46+16>:      0x0000000000000000      0x0000000000000000
0x6042d0 <n48>: 0x00000000000003e9      0x0000000000000000
0x6042e0 <n48+16>:      0x0000000000000000      0x0000000000000000
```

다음과 같은 트리를 얻을 수 있다. 이때, 맨 상위 노드에서 오른쪽으로 세번 진행하는 것이므로 <n1>에서 <n22>로, <n22>에서 <n34>로, <n34>에서 <n48>로 이동한다. 따라서 <secret_phase>의 정답은 0x3e9 즉, 1001 이다.