2019-2학기

컴퓨터 SW시스템 개론

Lab #3

20180085 무은재학부 송수민

Source Code & Explanation

Phase 1

```
Breakpoint 1, 0x0000000000400e8d in phase_1 ()
1: $eax = 6305696
(gdb) disas
Dump of assembler code for function phase_1:
=> 0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x402390,%esi
   0x0000000000400e96 <+9>:     callq  0x401300 <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    callq  0x4013ff <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    retq
End of assembler dump.
(gdb) x/s 0x402390
0x402390:       "When I get angry, Mr. Bigglesworth gets upset."
(gdb) i r
rax            0x6037a0 6305696
rbx            0x0      0
rcx            0x2e     46
rdx            0x6037a0 6305696
rsi            0x0      0
rdi            0x6037a0 6305696
rbp            0x4021b0 0x4021b0 <__libc_csu_init>
rsp            0x7fffffffe488   0x7fffffffe488
r8             0x60469f 6309535
r9             0x7ffff7fe2540   140737354016064
r10            0x3      3
r11            0x7ffff7a14890   140737347930256
r12            0x400c60 4197472
r13            0x7fffffffe570   140737488348528
r14            0x0      0
r15            0x0      0
rip            0x400e8d 0x400e8d <phase_1>
eflags         0x206    [ PF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
```

```
(gdb) x/s $eax
0x6037a0 <input_strings>:       "When I get angry, Mr. Bigglesworth gets upset."
(gdb)
```

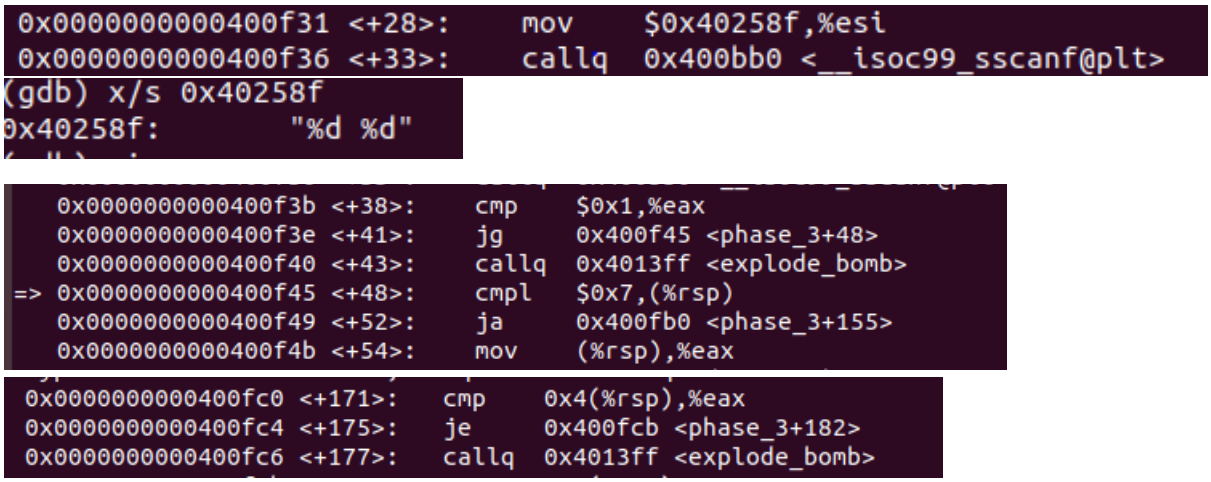
Phase 1은 strings_not_equal이라는 함수명을 보면 문자열이 같은지를 비교하는 문제이다. Strings_not_eqaul을 호출하기 전에, 어떠한 값을 esi레지스터로 넘기는데, 이를 strings_not_equal로 인자로 넘겨 eax에 있는 입력값과 비교하는 구조이다. 따라서, 문제가 요구하는 문자열은 0x402390에 담겨있으며, x/s command를 이용하여 이를 밝혀 낼 수 있다. Phase 1의 정답은 "When I get angry, Mr. Bigglesworth gets upset."이다.

Phase 2

```
0x0000000000400eaa in phase_2 ()
1: $eax = 6305776
(gdb) disas
Dump of assembler code for function phase_2:
   0x0000000000400ea9 <+0>:     push   %rbp
=> 0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x401421 <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:    callq  0x4013ff <explode_bomb>
   0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
   0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
   0x0000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
   0x0000000000400ee4 <+59>:    add    (%rbx),%eax
   0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
   0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:    callq  0x4013ff <explode_bomb>
   0x0000000000400ef0 <+71>:    add    $0x4,%rbx
   0x0000000000400ef4 <+75>:    cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
   0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
   0x0000000000400f09 <+96>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
```

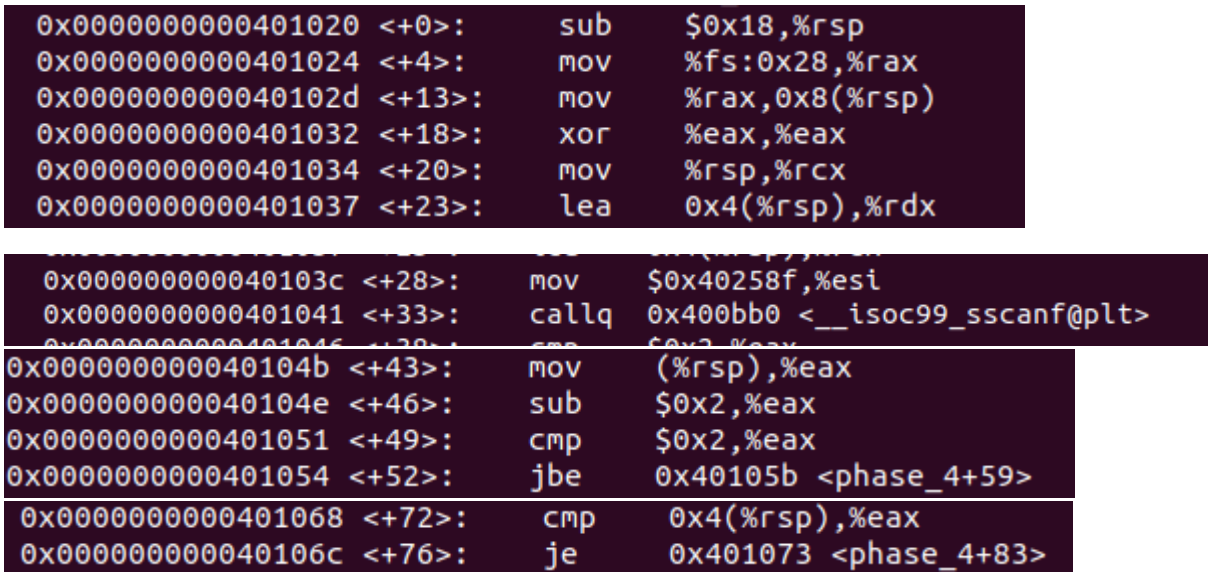
Phase 2는 read_six_numbers라는 함수명을 보면 여섯개의 숫자를 입력해야 한다는 것을 알 수 있다. <+30>을 보면 0과 rsp가 가리키는 값을 비교하는데 다음 점프문을 보면 같지 않으면 폭탄이 터진다는 것을 알 수 있다. Rsp가 가리키는 것은 첫번째 입력값일 것이고, 첫번째 입력 값은 0이어야 한다는 것을 알 수있다. <+36>을 보면 rsp+0x4와 1을 비교하는데 같아야만 폭탄이 터지지 않는다는 것을 알 수 있다. Rsp+0x4는 두번째 입력값이므로 두번째 입력값은 1이어야 한다는 것을 알 수 있다. 그 다음 부터는 일종의 반복문이 계속 적용되는데, 기본적인 메커니즘은 다음과 같다. 현재 가리키는 스택포인터를 다음 입력 값으로 옮겨가면서 그 후의 입력값과 더한 것을 비교한다. 결국 Phase 2에서 요구하는 수는 6번째 까지의 Base case가 0과 1인 피보나치 수열이다. 따라서, 정답은 "0 1 1 2 3 5" 이다.

Phase 3



Phase 3에서는 <+28>에서 어떠한 값을 함수가 호출되기 전에 옮기는 과정이 있다. 이를 조사해 보면, %d %d라는 값이 나온다. 이는 정수 2개를 입력하라는 것이다. 3번째 사진을 보면, $rsp값과 7을 비교하는 cmpl부분이 있는데, 7보다 큰 값을 입력하면 폭탄이 터진다. 다음 두번째 수는, %eax와 두번째 입력값을 비교하는 구문이 <+171>에 있다. 이는 범위가 정해진 것이 아니라, 정확히 숫자가 일치해야 폭탄이 터지지 않는다는 것을 <+175>의 je를 통해 알 수 있다. 이를 위해, display $eax를 입력하여 eax값의 변화를 살펴본 결과, -250이라는 값이 저 시점에 나왔으며, 이를 통해 두번째 입력값은 -250이라는 것을 알 수 있다. 따라서, Phase 3의 정답은 "2, -250"으로 하였다.

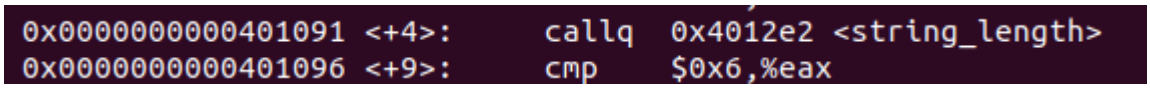
Phase 4

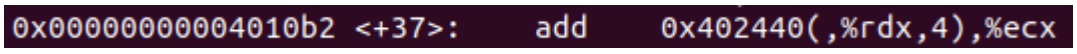


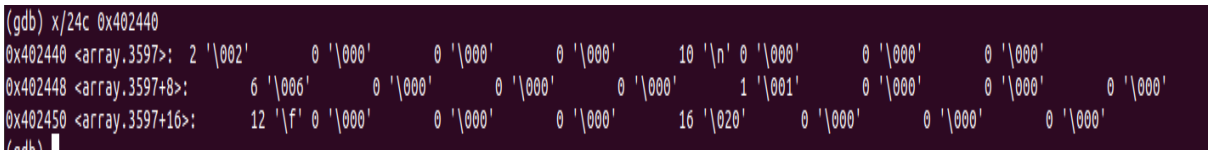
Phase 4는 Phase 3과 답을 도출해내는 방식이 거의 비슷하다. scanf함수를 호출 하기전에 어떠한 값을 esi에 옮기는데, 이 값은 Phase3에서 2번째 사진과 같다. 이를 통해, Phase 4 또한 정수 2개를 입력해야 한다는 것을 알 수 있다. 첫번째 수는 rsp가 가리키고 있는 값인데, 이에 2를 빼고 2

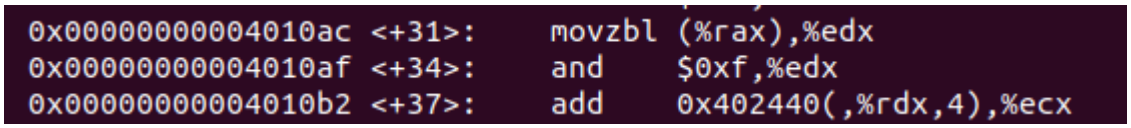
와 비교를 한다. Jbe를 보아 일단 같으면 폭탄이 터지지 않으므로 x-2 = 2이다. 따라서, 4를 넣어도 폭탄은 터지지 않을 것이다. 두번째 값은 eax와 비교하는데, eax는 fun4라는 재귀함수 과정을 거쳐 변화한다. 결국 eax와 두번째 값을 비교하는 것이므로 display $eax를 하여 저 시점의 eax값을 알아낸다. Eax는 132이다 하지만, 첫번째 사진에서 두 수를 가리키는 것이 바뀌므로, 정답은 132 4 라는 것을 알아낼 수 있다.

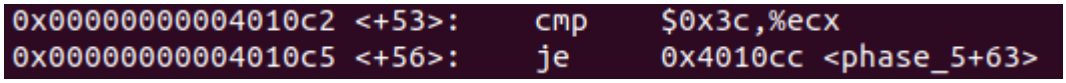
Phase 5











Phase 5에서는 string_length라는 함수명을 보아 문자열의 길이를 선검사한다. String_length의 반환 값은 문자열의 길이이고, 이것은 eax에 저장되고 이를 6과 비교하는 것을 보아 문자열의 길이는 6이다. 2번째 사진을 보면 어떤 메모리에서 레지스터로 값을 더한다. 4번째 사진을 보아 edx와 2진수로 1111의 and연산을 가지고 다음을 처리하는 것을 보아, 0x402440에 들어있는 메모리의 값을 살펴볼 필요가 있다. 6글자인데, 4칸 단위로 넣으므로 24칸을 조사한다. x/24c를 통해 조사해본 결과 3번째 사진과 같은 결과가 나왔다. 결국, 이는 and연산을 취한 값이 0이면 2를 더하고, 1이면 10을 더하는 형식인 것이다. 5번째 사진을 보았을 때, add된 값이 저장된 ecx가 60과 비교되는 것을 보아 ecx에 저장되어야하는 값은 60이다. 10x6 = 60이므로 1인 값이 필요한데, 우리가 입력하는 값은 문자열 형식이다. 문자의 아스키코드와 비교하는 것이므로, 소문자 a의 끝 4자리가 0001이므로 a를 6번 입력하면 합은 60이 된다. 정답은 "aaaaaa"로 하였다.

Phase 6

```
0x00000000004010e8 <+26>:    mov    %rsp,%rsi
0x00000000004010eb <+29>:    callq  0x401421 <read_six_numbers>
```

```
(gdb) x/24w 0x6032f0
0x6032f0 <node1>:       224     1       6304512 0
0x603300 <node2>:       693     2       6304528 0
0x603310 <node3>:       853     3       6304544 0
0x603320 <node4>:       181     4       6304560 0
0x603330 <node5>:       744     5       6304576 0
0x603340 <node6>:       642     6       0       0
```

```
 0x00000000004011a8 <+218>:   jle    0x4011af <phase_6+225>
 0x00000000004011aa <+220>:   callq  0x4013ff <explode_bomb>
```

Phase 6에서는 read_six_numbers라는 함수명을 보아 숫자 6개를 넣어야 한다는 것을 알 수 있다. 그리고 이중 포문을 통해, 입력된 수가 1사이에서 6인가를 판별한다. 이 사이의 숫자가 아니면 폭탄이 터지는 구조이다. 이 과정이 지나가면 입력된 숫자가 연결리스트 node에 치환되어서 저장되는데 그것의 시작점이 0x6032f0이다. x/24w 커맨드를 사용하여 6개의 노드를 검사한다. 그 결과 2번째 사진처럼 결과가 나오는데, 입력된 수가 정렬되어 있고 왼편에 치환된 수가 존재한다. 3번째 사진에서 <+218>을 기점으로 반복문이 도는데 계속 주소를 옮겨가면서 비교하는 동시에, jle인 것을 보아 작거나 같으면 폭탄이 터지므로, 오름차순이라는 것을 알 수 있다. 따라서 181-224-642-693-744-853 순으로 입력되어야 하고, 이는 치환된 값이므로 입력해야 하는 값은 4 1 6 2 5 3 이라는 것을 알 수 있다. 이로써 모든 폭탄이 해제되었다.