2018 시스템 프로그래밍
- Lab 05 -

제출일자	2018.10.29
분 반	01
이 름	함지희
학 번	201702087

Phase 1 [결과 화면 캡처]

```
Phase 1 defused. How about the next one?
```

Phase 1 [진행 과정 설명]

```
=> 0x0000000000400f2d <+0>:     sub    $0x8,%rsp
   0x0000000000400f31 <+4>:     mov    $0x40271c,%esi
   0x0000000000400f36 <+9>:     callq  0x401495 <strings_not_equal>
   0x0000000000400f3b <+14>:    test   %eax,%eax
   0x0000000000400f3d <+16>:    je     0x400f44 <phase_1+23>
   0x0000000000400f3f <+18>:    callq  0x401769 <explode_bomb>
   0x0000000000400f44 <+23>:    add    $0x8,%rsp
   0x0000000000400f48 <+27>:    retq
```

string not equal이 있고 정답문장과 비교를 해서 틀리면 bomb으로 가는 듯하다.

```
(gdb) x/s 0x40271c
0x40271c:       "Wow! Brazil is big."
```

esi의 값을 확인해보니 위처럼 나와서 이걸 써주었다.

Phase 1 [정답]

Wow! Brazil is big.

Phase 2 [결과 화면 캡처]

```
That's number 2.  Keep going!
```

Phase 2 [진행 과정 설명]

```
0x0000000000400f62 <+25>:    callq  0x40179f <read_six_numbers>
0x0000000000400f67 <+30>:    cmpl   $0x1,(%rsp)
0x0000000000400f6b <+34>:    je     0x400f72 <phase_2+41>
0x0000000000400f6d <+36>:    callq  0x401769 <explode_bomb>
```

첫 값은 1이다.

내가 입력한 값을 한 워드씩 이동시키면서 내 마지막 값(rbp)까지

eax+eax=eax를, rbx의 다음값(0x4를 더해주면서 주소이동)과 비교해서 다르면 bomb

Phase 2 [정답]

1 2 4 8 16 32

Phase 3 [결과 화면 캡처]

Phase 3 [진행 과정 설명]

```
(gdb) x/s 0x402730
0x402730:       "%d %c %d"
```

정수 글자 정수를 받고

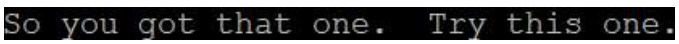
```
0x0000000000400ffa <+77>:    mov    $0x6a,%eax
0x0000000000400fff <+82>:    cmpl   $0x38e,0x14(%rsp)
0x0000000000401007 <+90>:    je     0x4010f5 <phase_3+328>
0x000000000040100d <+96>:    callq  0x401769 <explode_bomb>
```

처음 수는 7보다 작아야한다. 난 0을 넣었기 때문에 통과 되었고 비슷한 구조가 반복되는 양상이라 스위치문이라 판단했다. 0을 입력했기 때문에 첫 번째구조를 봤더니 eax에 0x6a를 넣고 rsp의 0x14주소와 0x38e를 비교하는게 나와서 중간 글자와 2번째 정수가 j와 910일거라 예상하였다.

Phase 3 [정답]

0 j 910

Phase 4 [결과 화면 캡처]



Phase 4 [진행 과정 설명]

```
(gdb) x/s 0x402a05
0x402a05:       "%d %d"
```

정수 2개를 입력받는다. 2개 입력 안 받으면 bomb

```
0x000000000040113f <+37>:    callq  0x40111a <func4>
0x0000000000401144 <+42>:    add    %r12d,%eax
0x0000000000401147 <+45>:    jmp    0x40114f <func4+53>
0x0000000000401149 <+47>:    mov    $0x0,%eax
0x000000000040114e <+52>:    retq
0x000000000040114f <+53>:    pop    %rbx
0x0000000000401150 <+54>:    pop    %rbp
```

내가 입력한 숫자인 3과 $edi = 9

를 가지고 func4로 들어간다. func4를 직접 돌리니 계속 반복을 하길래 $r12d , $eax를 더하고 비교를 하는 부분만 계속 확인했더니, 3으로 시작하는 피보나치 수열을 9번째까지 더하는 값이 최종적으로 eax에 저장된다. 그 값을 답으로 입력하였다.

Phase 4 [정답]

3 264

Phase 5 [결과 화면 캡처]

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

Phase 5 [진행 과정 설명]

```
0x00000000004011da <+24>:    callq  0x401477 <string_length>
0x00000000004011df <+29>:    cmp    $0x6,%eax
0x000000000040120e <+76>:     mov    $0x402739,%esi
0x0000000000401213 <+81>:     mov    %rsp,%rdi
0x0000000000401216 <+84>:     callq  0x401495 <strings_not_equal>
```

string not equal을 보아 그 위에 있는 esi와 rdi를 비교하여 틀리면 bomb인것같다.

```
(gdb) x/s 0x402739
0x402739:       "devils"
```

확인해보니 devils이다.

이제 rdi가 어떻게 나오는 과정을 보면

```
0x00000000004011ee <+44>:    movzbl (%rbx,%rax,1),%edx
0x00000000004011f2 <+48>:    and    $0xf,%edx
0x00000000004011f5 <+51>:    movzbl 0x402780(%rdx),%edx
0x00000000004011fc <+58>:    mov    %dl,(%rsp,%rax,1)
0x00000000004011ff <+61>:    add    $0x1,%rax
0x0000000000401203 <+65>:    cmp    $0x6,%rax
```

입력받은 6개의 글자를 순서대로 하나 꺼내어 edx에 저장하고 edx의 하위 4비트를 뽑아 다시 edx에 저장한다.

```
0x402780 <array.3600>:  "maduiersnfotvbylWow! You've defused the secret stage!"
```

하위 4비트를 주소로 더해주어 저 문장의 rdx번 인덱스의 글자를 edx에 저장한다. 그걸 다시 rsp에 넣어주고 이게 나중에 rdi가 된다.

devils를 만들 수 있도록 위의 과정을 반대로 따라가면 답이 나온다.

Phase 5 [정답]

beldog

Phase 6 [결과 화면 캡처]

```
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
```

Phase 6 [진행 과정 설명]

```
0x0000000000401276 <+55>:    sub    $0x1,%eax
0x0000000000401279 <+58>:    cmp    $0x5,%eax
0x000000000040127c <+61>:    jbe    0x401283 <phase_6+68>
0x000000000040127e <+63>:    callq  0x401769 <explode_bomb>
```

```
0x0000000000401333 <+244>:   mov    0x8(%rbx),%rax
0x0000000000401337 <+248>:   mov    (%rax),%eax
0x0000000000401339 <+250>:   cmp    %eax,(%rbx)
0x000000000040133b <+252>:   jge    0x401342 <phase_6+259>
0x000000000040133d <+254>:   callq  0x401769 <explode_bomb>
0x0000000000401342 <+259>:   mov    0x8(%rbx),%rbx
0x0000000000401346 <+263>:   sub    $0x1,%ebp
0x0000000000401349 <+266>:   jne    0x401333 <phase_6+244>
```

6보다 같거나 작은 정수 6개 받는다.

먼저 마지막 bomb부터 보면 rbx에서 하나씩 비교하면서 전 값이 다음 값보다 커야 bomb이 안 터지는 걸 볼 수 있다.

이걸 반복하면서 내림차순으로 정렬한다는 걸 알 수 있다.

```
(gdb) x/3w $rbx
0x604340 <node6>:       268     6       6308656
(gdb) x/3w 0x604330
0x604330 <node5>:       560     5       6308640
(gdb) x/3w 0x604320
0x604320 <node4>:       634     4       6308624
(gdb) x/3w 0x604310
0x604310 <node3>:       665     3       6308608
(gdb) x/3w 0x604300
0x604300 <node2>:       50      2       6308592
(gdb) x/3w 0x6042f0
0x6042f0 <node1>:       959     1       0
```

그래서 rbx를 살펴봤더니 노드들이 있었다. 노드의 값을 내림차순으로 정렬할 것이다고 예상 후 입력하니 bomb이 터지길래 확인해보니

```
0x00000000004012ae <+111>:   lea    0x18(%rsp),%rcx
0x00000000004012b3 <+116>:   mov    $0x7,%edx
0x00000000004012b8 <+121>:   mov    %edx,%eax
0x00000000004012ba <+123>:   sub    (%r12),%eax
0x00000000004012be <+127>:   mov    %eax,(%r12)
0x00000000004012c2 <+131>:   add    $0x4,%r12
0x00000000004012c6 <+135>:   cmp    %r12,%rcx
0x00000000004012c9 <+138>:   jne    0x4012b8 <phase_6+121>
```

7에서 내게 입력받은 값을 빼주는게 있어서 위에서 입력했던 답들에 적용하였다.

Phase 6 [정답]

6 4 3 2 1 5

Phase secret	[결과 화면 캡처]

```
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
```

Phase secret	[진행 과정 설명]

```
0x00000000004013e1 <+54>:    cmp    $0x6,%eax
0x00000000004013e4 <+57>:    je     0x4013eb <secret_phase+64>
0x00000000004013e6 <+59>:    callq  0x401769 <explode_bomb>
```

bomb이 터지는 곳을 확인해보니, eax가 6이어야 bomb이 터지지 않는다는 걸 알 수 있다. eax의 값은 fun7안에서 결정되는 듯하다. esi와 edi를 가지고 fun7로 들어간다.

```
(gdb) x/10xw $edi
0x604110 <n1>:  0x00000024      0x00000000      0x00604130      0x00000000
0x604120 <n1+16>:       0x00604150      0x00000000      0x00000000      0x000000
00
0x604130 <n21>: 0x00000008      0x00000000
```

fun7로 들어가기 전에 edi를 살펴보았다. 특이한 점은 n1의 3번째 칸에 n21의 주소값이 들어가 있다는 점이다.
좀 더 출력해보니 edi는 트리의 형태였다.

```
0x0000000000401376 <+9>:     mov    (%rdi),%edx
0x0000000000401378 <+11>:    cmp    %esi,%edx
```

eax를 구하기 위해 fun7로 들어가보니 내가 입력한 값과 첫 번째 노드값 (root)값을 비교하는 재귀함수였다. 내가 입력한 값이 크면 오른쪽 자식으로 가고 내가 입력한 값이 작으면 왼쪽자식으로 가서, 내가 입력한 값과 똑같은 값을 만날때까지 반복한다.
내가 입력한 값과 똑같은 노드를 만나면 eax는 0이고 이 값을 가지고 함수를 호출한 주소의 다음 주소로 돌아가게 된다.

```
0x000000000040139b <+46>:    lea    0x1(%rax,%rax,1),%eax
0x000000000040139f <+50>:    jmp    0x4013a6 <fun7+57>
```

내가 입력한 값이 더 컸을 때 호출한 fun7 다음 주소이다. rax에 2를 곱하고 1을 더해준 값을 eax에 넣는다.

```
0x0000000000401385 <+24>:    add    %eax,%eax
0x0000000000401387 <+26>:    jmp    0x4013a6 <fun7+57>
```

내가 입력한 값이 더 작을 때 호출한 fun7 의 다음 주소이다. eax의 2배를 eax에 넣는다.
내가 입력한 값이 들어있는 노드를 기준으로 root까지 찾아가면서 eax의 값을 변화시킨다. eax가 6이 나오기 위해서는 노드n44에서 eax가 0이어야한다. 고로 그 노드 값이 정답이다.

Phase secret	[정답]