Bomblab 풀이 레포트

컴퓨터공학과 / 201510621 / 김민혁

```
0x0000000000001324 <+0>:     sub    $0x8,%rsp
0x0000000000001328 <+4>:     lea    0xd1d(%rip),%rsi        # 0x204c
0x000000000000132f <+11>:    callq  0x1795 <strings_not_equal>
0x0000000000001334 <+16>:    test   %eax,%eax
0x0000000000001336 <+18>:    jne    0x133d <phase_1+25>
0x0000000000001338 <+20>:    add    $0x8,%rsp
0x000000000000133c <+24>:    retq
0x000000000000133d <+25>:    callq  0x1c6c <explode_bomb>
0x0000000000001342 <+30>:    jmp    0x1338 <phase_1+20>
```

<phase_1>

먼저 <+11>의 strings_not_equal 함수가 무엇을 인자로 받는지 확인하였다.
x/s를 하니'skawkekqrp' 라는 문자열이 나타났다. 함수 이름으로 보았을 때 문자열이 skawkekqrp와 입력값이 다를 때 0을 리턴 할 것인데 리턴값이 0이 아니여야 점프하므로 skawkekqrp와 값이 같아야 점프를 한다.
그래서 답은 skawkekqrp이다.

```
0x0000000000001344 <+0>:     push   %rbp
0x0000000000001345 <+1>:     push   %rbx
0x0000000000001346 <+2>:     sub    $0x18,%rsp
0x000000000000134a <+6>:     mov    %fs:0x28,%rax
0x0000000000001353 <+15>:    mov    %rax,0x8(%rsp)
0x0000000000001358 <+20>:    xor    %eax,%eax
0x000000000000135a <+22>:    lea    0x4(%rsp),%rcx
0x000000000000135f <+27>:    mov    %rsp,%rdx
0x0000000000001362 <+30>:    lea    0xfab(%rip),%rsi        # 0x2314
0x0000000000001369 <+37>:    callq  0xfe0 <__isoc99_sscanf@plt>
0x000000000000136e <+42>:    cmp    $0x1,%eax
0x0000000000001371 <+45>:    jle    0x137f <phase_2+59>
0x0000000000001373 <+47>:    mov    $0x5,%ebp
0x0000000000001378 <+52>:    mov    $0x1,%ebx
0x000000000000137d <+57>:    jmp    0x138f <phase_2+75>
0x000000000000137f <+59>:    callq  0x1c6c <explode_bomb>
0x0000000000001384 <+64>:    jmp    0x1373 <phase_2+47>
0x0000000000001386 <+66>:    imul   (%rsp),%ebx
0x000000000000138a <+70>:    sub    $0x1,%ebp
0x000000000000138d <+73>:    je     0x139c <phase_2+88>
0x000000000000138f <+75>:    cmp    %ebx,0x4(%rsp)
0x0000000000001393 <+79>:    jg     0x1386 <phase_2+66>
0x0000000000001395 <+81>:    callq  0x1c6c <explode_bomb>
0x000000000000139a <+86>:    jmp    0x1386 <phase_2+66>
0x000000000000139c <+88>:    cmp    %ebx,0x4(%rsp)
0x00000000000013a0 <+92>:    je     0x13a7 <phase_2+99>
0x00000000000013a2 <+94>:    callq  0x1c6c <explode_bomb>
0x00000000000013a7 <+99>:    mov    0x8(%rsp),%rax
0x00000000000013ac <+104>:   xor    %fs:0x28,%rax
0x00000000000013b5 <+113>:   jne    0x13be <phase_2+122>
0x00000000000013b7 <+115>:   add    $0x18,%rsp
0x00000000000013bb <+119>:   pop    %rbx
0x00000000000013bc <+120>:   pop    %rbp
0x00000000000013bd <+121>:   retq
0x00000000000013be <+122>:   callq  0xf10 <__stack_chk_fail@plt>
```


<phase_2>

먼저 scanf 함수가 받는 인자를 확인하니 %d %d, 즉 정수 두개를 인자로 받았다. scanf 함수의 리턴값은 입력의 개수이다. <+45>에서 리턴값<=1이면 bomb로 점프한다. 따라서 2개 이상 입력해야 한다. 여기서 (%rsp)에 첫번째 입력값, (%rsp)+4에 두번째 입력값이 들어가는 것을 확인하였다.
<+57>에서 무조건<+75>로 점프하고 <+52>에서 %ebp에 1을 저장하므로 두번째 입력값이 1보다 커야 <+66>으로 점프한다. 여기서 1*첫번째 입력값을 %edx에 넣는다.%ebp 에는 <+47>에서 5가 저장되었다. 여기서 1을 빼고 다시 %ebp에 저장하는데 점프(<+73>)를 하지 않기(루프문에서 빠져나오기) 위해서 0이 되야 하므로 5번 반복된다는 것을 알 수 있었다.(5에서 1씩 빼서 0이 되어야 하므로)
5번 반복되는 동안 %edx에 들어간 첫번째 입력값이 다섯제곱된다.루프문에서 빠져나오면 <+88>에서 두번째 입력값과 %edx가 같으면 점프 후 리턴된다.
따라서 n과 n의 오제곱이 답이므로 2,32 가 답 중의 하나이다..

```
0x00000000000013c3 <+0>:     sub    $0x28,%rsp
0x00000000000013c7 <+4>:     mov    %fs:0x28,%rax
0x00000000000013d0 <+13>:    mov    %rax,0x18(%rsp)
0x00000000000013d5 <+18>:    xor    %eax,%eax
0x00000000000013d7 <+20>:    lea    0xf(%rsp),%rcx
0x00000000000013dc <+25>:    lea    0x10(%rsp),%rdx
0x00000000000013e1 <+30>:    lea    0x14(%rsp),%r8
0x00000000000013e6 <+35>:    lea    0xc6a(%rip),%rsi        # 0x2057
0x00000000000013ed <+42>:    callq  0xfe0 <__isoc99_sscanf@plt>
0x00000000000013f2 <+47>:    cmp    $0x2,%eax
0x00000000000013f5 <+50>:    jle    0x142f <phase_3+108>
0x00000000000013f7 <+52>:    mov    0x10(%rsp),%eax
0x00000000000013fb <+56>:    cmp    $0x3,%eax
0x00000000000013fe <+59>:    jg     0x1442 <phase_3+127>
0x0000000000001400 <+61>:    mov    $0x77,%eax
0x0000000000001405 <+66>:    cmpl   $0x286,0x14(%rsp)
0x000000000000140d <+74>:    jne    0x1436 <phase_3+115>
0x000000000000140f <+76>:    cmp    %al,0xf(%rsp)
0x0000000000001413 <+80>:    je     0x141a <phase_3+87>
0x0000000000001415 <+82>:    callq  0x1c6c <explode_bomb>
0x000000000000141a <+87>:    mov    0x18(%rsp),%rax
0x000000000000141f <+92>:    xor    %fs:0x28,%rax
0x0000000000001428 <+101>:   jne    0x148e <phase_3+203>
0x000000000000142a <+103>:   add    $0x28,%rsp
0x000000000000142e <+107>:   retq
0x000000000000142f <+108>:   callq  0x1c6c <explode_bomb>
0x0000000000001434 <+113>:   jmp    0x13f7 <phase_3+52>
0x0000000000001436 <+115>:   callq  0x1c6c <explode_bomb>
0x000000000000143b <+120>:   mov    $0x77,%eax
0x0000000000001440 <+125>:   jmp    0x140f <phase_3+76>
0x0000000000001442 <+127>:   cmp    $0x4,%eax
0x0000000000001445 <+130>:   je     0x1467 <phase_3+164>
0x0000000000001447 <+132>:   cmp    $0x7,%eax
0x000000000000144a <+135>:   jg     0x1482 <phase_3+191>
0x000000000000144c <+137>:   mov    $0x72,%eax
0x0000000000001451 <+142>:   cmpl   $0x316,0x14(%rsp)
0x0000000000001459 <+150>:   je     0x140f <phase_3+76>
0x000000000000145b <+152>:   callq  0x1c6c <explode_bomb>
0x0000000000001460 <+157>:   mov    $0x72,%eax
0x0000000000001465 <+162>:   jmp    0x140f <phase_3+76>
0x0000000000001467 <+164>:   mov    $0x73,%eax
0x000000000000146c <+169>:   cmpl   $0x1b6,0x14(%rsp)
0x0000000000001474 <+177>:   je     0x140f <phase_3+76>
0x0000000000001476 <+179>:   callq  0x1c6c <explode_bomb>
0x000000000000147b <+184>:   mov    $0x73,%eax
0x0000000000001480 <+189>:   jmp    0x140f <phase_3+76>
0x0000000000001482 <+191>:   callq  0x1c6c <explode_bomb>
0x0000000000001487 <+196>:   mov    $0x66,%eax
0x000000000000148c <+201>:   jmp    0x140f <phase_3+76>
0x000000000000148e <+203>:   callq  0xf10 <__stack_chk_fail@plt>
```

<phase_3>

<+35>의 scanf 함수가 받는 인자를 확인하니 %d %c %d였다. <+50>에서 리턴값이 2보다 커야 점프를 하지 않고 첫번째 입력값인(%rsp)+16을 %eax에 저장한다.<+56>에서 이 값이 3보다 커야 <+127>로 점프하고 <+130>에서 4와 같으면 점프하는데 점프해도 폭탄이 터지지는 않지만 빠져나가지 못하는 결과가 나온다고 판단해 점프하지 않는 방향으로 갔다. <+135>에서 7보다 작아야 점프를 하지 않고 <+137>에서 %eax에 144를 저장한다.<+142>에서 0x316(790)과 세번째 입력값인(%rsp)+20이 같으면 <+76>으로 점프를 하고 %al과 (%rsp)+15인 두번째 입력값을 비교해서 같으면 <+87>로 점프해서 리턴하게 되었다. %al에는 144가 들어가 있으므로 두번째 입력값은 144,즉 'r' 이다. 따라서 답은 6 r 790이다.

```
0x00000000000014ba <+0>:     sub    $0x18,%rsp
0x00000000000014be <+4>:     mov    %fs:0x28,%rax
0x00000000000014c7 <+13>:    mov    %rax,0x8(%rsp)
0x00000000000014cc <+18>:    xor    %eax,%eax
0x00000000000014ce <+20>:    lea    0x4(%rsp),%rdx
0x00000000000014d3 <+25>:    lea    0xb83(%rip),%rsi        # 0x205d
0x00000000000014da <+32>:    callq  0xfe0 <__isoc99_sscanf@plt>
0x00000000000014df <+37>:    cmp    $0x1,%eax
0x00000000000014e2 <+40>:    jne    0x14eb <phase_4+49>
0x00000000000014e4 <+42>:    cmpl   $0x0,0x4(%rsp)
0x00000000000014e9 <+47>:    jg     0x14f0 <phase_4+54>
0x00000000000014eb <+49>:    callq  0x1c6c <explode_bomb>
0x00000000000014f0 <+54>:    mov    0x4(%rsp),%edi
0x00000000000014f4 <+58>:    callq  0x1493 <func4>
0x00000000000014f9 <+63>:    cmp    $0x961,%eax
0x00000000000014fe <+68>:    je     0x1505 <phase_4+75>
0x0000000000001500 <+70>:    callq  0x1c6c <explode_bomb>
0x0000000000001505 <+75>:    mov    0x8(%rsp),%rax
0x000000000000150a <+80>:    xor    %fs:0x28,%rax
0x0000000000001513 <+89>:    jne    0x151a <phase_4+96>
0x0000000000001515 <+91>:    add    $0x18,%rsp
0x0000000000001519 <+95>:    retq
0x000000000000151a <+96>:    callq  0xf10 <__stack_chk_fail@plt>
```

<phase_4>

<+25>에서 scanf함수가 받는 인자를 확인하였더니 %d였다. <+47>에서 리턴값(입력의 개수)이 1이 아니면 폭탄으로 점프한다.<+47>에서 입력값이 0보다 크면 <+54>로 점프한다. 입력값을 인자로 받아 func4를 실행한다.(func4는 다음페이지에) 이 함수의 리턴값이 0x961(2401)과 같으면 리턴하였다. 이 함수는 재귀함수로 입력값만큼 7을 제곱시키는 함수였다. 2401은 7의 네제곱이므로 입력값이 4여야 한다.
따라서 답은 4이다.

```
0x0000000000001493 <+0>:     mov    $0x1,%eax
0x0000000000001498 <+5>:     test   %edi,%edi
0x000000000000149a <+7>:     jg     0x149e <func4+11>
0x000000000000149c <+9>:     repz retq
0x000000000000149e <+11>:    sub    $0x8,%rsp
0x00000000000014a2 <+15>:    sub    $0x1,%edi
0x00000000000014a5 <+18>:    callq  0x1493 <func4>
0x00000000000014aa <+23>:    lea    0x0(,%rax,8),%edx
0x00000000000014b1 <+30>:    sub    %eax,%edx
0x00000000000014b3 <+32>:    mov    %edx,%eax
0x00000000000014b5 <+34>:    add    $0x8,%rsp
0x00000000000014b9 <+38>:    retq
```

<func4>

%eax에 1을 저장한다. <+5>에서 입력값끼리 test 하므로 입력값이 0이 아닌 이상 <+7>에서 <+11>로 점프한다. <+15>에서 %edi 값에서 1을빼고 다시 func4로 들어가는데 0이 되면 리턴하므로 함수 내에서 입력값만큼 func4를 불러오게된다.
불러올때마다 8*%rax를 %edx에 저장하고 %edx에서 %eax값을 빼주고 %eax에 %edx 값을 저장하는걸 반복하므로 입력값만큼 돌면서 7을 제곱하는 루프문과 성능이 같다.

```
0x000000000000151f <+0>:     push   %rbx
0x0000000000001520 <+1>:     sub    $0x10,%rsp
0x0000000000001524 <+5>:     mov    %rdi,%rbx
0x0000000000001527 <+8>:     mov    %fs:0x28,%rax
0x0000000000001530 <+17>:    mov    %rax,0x8(%rsp)
0x0000000000001535 <+22>:    xor    %eax,%eax
0x0000000000001537 <+24>:    callq  0x1778 <string_length>
0x000000000000153c <+29>:    cmp    $0x6,%eax
0x000000000000153f <+32>:    jne    0x1596 <phase_5+119>
0x0000000000001541 <+34>:    mov    $0x0,%eax
0x0000000000001546 <+39>:    lea    0xb43(%rip),%rcx        # 0x2090 <array.
3403>
0x000000000000154d <+46>:    movzbl (%rbx,%rax,1),%edx
0x0000000000001551 <+50>:    and    $0xf,%edx
0x0000000000001554 <+53>:    movzbl (%rcx,%rdx,1),%edx
0x0000000000001558 <+57>:    mov    %dl,0x1(%rsp,%rax,1)
0x000000000000155c <+61>:    add    $0x1,%rax
0x0000000000001560 <+65>:    cmp    $0x6,%rax
0x0000000000001564 <+69>:    jne    0x154d <phase_5+46>
0x0000000000001566 <+71>:    movb   $0x0,0x7(%rsp)
0x000000000000156b <+76>:    lea    0x1(%rsp),%rdi
0x0000000000001570 <+81>:    lea    0xae9(%rip),%rsi        # 0x2060
0x0000000000001577 <+88>:    callq  0x1795 <strings_not_equal>
0x000000000000157c <+93>:    test   %eax,%eax
0x000000000000157e <+95>:    jne    0x159d <phase_5+126>
0x0000000000001580 <+97>:    mov    0x8(%rsp),%rax
0x0000000000001585 <+102>:   xor    %fs:0x28,%rax
0x000000000000158e <+111>:   jne    0x15a4 <phase_5+133>
0x0000000000001590 <+113>:   add    $0x10,%rsp
0x0000000000001594 <+117>:   pop    %rbx
0x0000000000001595 <+118>:   retq
0x0000000000001596 <+119>:   callq  0x1c6c <explode_bomb>
0x000000000000159b <+124>:   jmp    0x1541 <phase_5+34>
0x000000000000159d <+126>:   callq  0x1c6c <explode_bomb>
0x00000000000015a2 <+131>:   jmp    0x1580 <phase_5+97>
0x00000000000015a4 <+133>:   callq  0xf10 <__stack_chk_fail@plt>
```

<phase_5>

<+22>에서 %eax를 초기화해준다. string_length함수의 리턴값이 6이 되어야 폭탄으로 점프를 하지 않기 때문에 6자리의 문자열을 입력해야 한다.<+39>에서 0xb43(%rip)를 %rcx로 넣는데 넣은 값을 검사해보니 <array.3403>:'I'가 나왔다.<+46>에서 %rbx값을 %edx에 저장한다 %edx 값을 0xf(1111)와 and 해서 %edx에 저장한다. 그리고 이 값을 %rcx와 더한 주소의 값을 %edx에 저장한다. %edx에서 %dl을 %rax+%rsp+1에 저장한다. 그 다음 %rax에 1을 더한다. 이것이 6이 될때까지 이 연산을 반복(<+69>에서 계속 점프)한다. 이 루프를 통과하고 나서 0을 (%rsp)+7에 저장한다. 그리고 (%rsp)+1주소 값을 %rdi에 넣는다. 여기서 내가 입력했던 문자가 다른 문자로 바뀐다.<+81>에서 인자를 받아서 strings_not_equal 함수를 부르는데 0이 안나와야 <+95>에서 점프를 안하므로 인자로 받은 값과 같아야 한다. 인자로 받은 값을 보면 'titans'가 나온다. 그런데 내가 입력한 값이 루프를 돌면서 값이 바뀌는데 연산을 통해 어떤 문자가 어떤문자로 바뀌는지 파악하기 어려워 a~z까지 입력하여서 어떻게 변하는지 확인하였다.

defabc >> eawsrv
ghijkl>>hobpnu
mnopqr>>tfgisr
그래서 얻은 답은 mpmekq 이다.

```
   0x0000000000001606 <+0>:     sub    $0x8,%rsp
   0x000000000000160a <+4>:     mov    $0xa,%edx
   0x000000000000160f <+9>:     mov    $0x0,%esi
   0x0000000000001614 <+14>:    callq  0xfc0 <strtol@plt>
   0x0000000000001619 <+19>:    mov    %eax,0x202001(%rip)        # 0x203620 <n
ode0>
   0x000000000000161f <+25>:    lea    0x201ffa(%rip),%rdi        # 0x203620 <n
ode0>
   0x0000000000001626 <+32>:    callq  0x15a9 <fun6>
   0x000000000000162b <+37>:    mov    0x8(%rax),%rax
   0x000000000000162f <+41>:    mov    0x8(%rax),%rax
   0x0000000000001633 <+45>:    mov    0x8(%rax),%rax
   0x0000000000001637 <+49>:    mov    0x8(%rax),%rax
   0x000000000000163b <+53>:    mov    0x8(%rax),%rax
   0x000000000000163f <+57>:    mov    0x8(%rax),%rax
   0x0000000000001643 <+61>:    mov    0x8(%rax),%rax
   0x0000000000001647 <+65>:    mov    0x8(%rax),%rax
   0x000000000000164b <+69>:    mov    0x201fcf(%rip),%ecx        # 0x203620 <n
ode0>
   0x0000000000001651 <+75>:    cmp    %ecx,(%rax)
   0x0000000000001653 <+77>:    je     0x165a <phase_6+84>
   0x0000000000001655 <+79>:    callq  0x1c6c <explode_bomb>
   0x000000000000165a <+84>:    add    $0x8,%rsp
   0x000000000000165e <+88>:    retq
```

<phase_6>

Strtol함수는 문자를 long타입으로 바꿔주는 함수이다.
이 리턴값을(%rip)+0x202001에 저장한다.이 주소값으로 검색하니 <node0> : 0이 나왔다. 그리고 <+25>에서 0번노드를 인자로 fun6를 콜한다.(fun6는 다음페이지)(%rax)+8을 %rax에 저장하는 것을 8번한다.콜하고 나서부터 %rax가 변하는 것을 따라가보았다.

0x555555757620 = 2 <node 0>
0x555555757630 = 108 <node 1>
0x555555757120 = -62 <node 9>
0x5555557576a0 = 44 <node8>
0x555555757660 = 27 <node4>
0x555555757670 = -19<node5>
0x555555757640 = -42<node2>
0x555555757690 = -64<node7>
0x555555757680 = - 44<node6>
0x555555757650 = 207<node3>

함수 첫 줄에서 %rdi 를 %rax로 mov 했을 때, 노드가 저장된 주소에 +0x8을 한 주소에 저장된 값을 열거하면 노드 순서가 차례대로 있었는데 뒤죽박죽 되어있는 것으로 보아 func6가 노드를 정렬시키는 함수라고 유추할 수 있었다.
그리고 0번 노드값을 %ecx에 저장한다. 0번 노드값이 입력값과 같았는데 이것이 현재 (%rax)와 같아야 하는 것을 확인하였다. 마지막으로 정렬된 값이 207이었다.
따라서 답은 207이다.

```
0x00000000000015a9 <+0>:     mov    %rdi,%rax
0x00000000000015ac <+3>:     mov    0x8(%rdi),%r8
0x00000000000015b0 <+7>:     movq   $0x0,0x8(%rdi)
0x00000000000015b8 <+15>:    test   %r8,%r8
0x00000000000015bb <+18>:    jne    0x15e8 <fun6+63>
0x00000000000015bd <+20>:    repz retq
0x00000000000015bf <+22>:    mov    %rdx,%rcx
0x00000000000015c2 <+25>:    mov    0x8(%rcx),%rdx
0x00000000000015c6 <+29>:    test   %rdx,%rdx
0x00000000000015c9 <+32>:    je     0x15cf <fun6+38>
0x00000000000015cb <+34>:    cmp    %esi,(%rdx)
0x00000000000015cd <+36>:    jg     0x15bf <fun6+22>
0x00000000000015cf <+38>:    cmp    %rdx,%rcx
0x00000000000015d2 <+41>:    je     0x15ff <fun6+86>
0x00000000000015d4 <+43>:    mov    %r8,0x8(%rcx)
0x00000000000015d8 <+47>:    mov    0x8(%r8),%rcx
0x00000000000015dc <+51>:    mov    %rdx,0x8(%r8)
0x00000000000015e0 <+55>:    mov    %rcx,%r8
0x00000000000015e3 <+58>:    test   %rcx,%rcx
0x00000000000015e6 <+61>:    je     0x1604 <fun6+91>
0x00000000000015e8 <+63>:    test   %rax,%rax
0x00000000000015eb <+66>:    je     0x15f7 <fun6+78>
0x00000000000015ed <+68>:    mov    (%r8),%esi
0x00000000000015f0 <+71>:    mov    %rax,%rcx
0x00000000000015f3 <+74>:    cmp    %esi,(%rax)
0x00000000000015f5 <+76>:    jg     0x15c2 <fun6+25>
0x00000000000015f7 <+78>:    mov    %rax,%rdx
0x00000000000015fa <+81>:    mov    %r8,%rax
0x00000000000015fd <+84>:    jmp    0x15d8 <fun6+47>
0x00000000000015ff <+86>:    mov    %r8,%rax
0x0000000000001602 <+89>:    jmp    0x15d8 <fun6+47>
0x0000000000001604 <+91>:    repz retq
```

<func_6>

0번 노드값을 %rax에 저장한다. 그리고 (%rdi)+0x08을 %r8에 저장한다. 이 값은 <node01>로 확인되었다.그리고 (%rdi)+0x08을 초기화해준다.
1번노드가 0이 아니면 <+15>에서 <+63>으로 점프한다.여기서 %rax가 0이면 78로 점프한다. 그런데 0이 아니므로 지나가면 1번노드주소값을 %esi에 저장하고 0번노드값을 %rcx에 저장한다. 그리고 0번노드 주소값이 1번노드주소값보다 크면 점프하지만 크지 않기 때문에 지나간다.
그리고 0번노드 값을 %rdx에 저장한다. 그리고 1번노드값을 %rax에 저장한다. 그리고 무조건 <+47>로 점프한다. 그리고 (%r8)+0x8을 %rcx에 저장한다. 그리고 %rdx값을 그 자리에 저장한다.<+58>에서 %rcx가 0이면 끝이난다. 하지만 0이 아니므로 다시 지나가서 그리고 <+66>도 통과하고 (%r8)을 %esi 에 저장하고 %rax를 %rcxdp 저장한다.<+76>에서 <+25>로 점프한다.
이 과정을 반복하다가 %rcx 가 0이되면 리턴한다