학번: 201724485 이름: 배재홍

[bomb lab - bomb.13]

<phase_1: string>

1.
```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
=> 0x00005555555552b4 <+0>:     sub    $0x8,%rsp
   0x00005555555552b8 <+4>:     lea    0x182d(%rip),%rsi        # 0x555555556aec
   0x00005555555552bf <+11>:    callq  0x5555555557b9 <strings_not_equal>
   0x00005555555552c4 <+16>:    test   %eax,%eax
   0x00005555555552c6 <+18>:    jne    0x5555555552cd <phase_1+25>
   0x00005555555552c8 <+20>:    add    $0x8,%rsp
   0x00005555555552cc <+24>:    retq
   0x00005555555552cd <+25>:    callq  0x555555555abd <explode_bomb>
   0x00005555555552d2 <+30>:    jmp    0x5555555552c8 <phase_1+20>
End of assembler dump.
(gdb) stepi
0x00005555555552b8 in phase_1 ()
(gdb) stepi
0x00005555555552bf in phase_1 ()
```

stepi를 두 번 사용하여 string_not_equal을 call합니다.

2.
```
(gdb) print /x $rsi
$4 = 0x555555556aec
(gdb) x/40c 0x555555556aec
0x555555556aec: 80 'P'  117 'u' 98 'b'  108 'l' 105 'i' 99 'c'  32 ' '  115 's'
0x555555556af4: 112 'p' 101 'e' 97 'a'  107 'k' 105 'i' 110 'n' 103 'g' 32 ' '
0x555555556afc: 105 'i' 115 's' 32 ' '  118 'v' 101 'e' 114 'r' 121 'y' 32 ' '
0x555555556b04: 101 'e' 97 'a'  115 's' 121 'y' 46 '.'  0 '\000'        0 '\000'        0
 '\000'
0x555555556b0c: 0 '\000'        0 '\000'        0 '\000'        0 '\000'        0 '\000'
0 '\000'        0 '\000'        0 '\000'
```

print /x $rsi 명령으로 %rsi에 저장된 주소값을 16진수로 받고 x/40c [주소값] 명령으로 %rsi가 가리키는 문자열이 “Public speaking is very easy.”임을 확인합니다.

<phase_2: loops>

1.
```
0x0000555555555b16 <+29>:    lea    0x12cc(%rip),%rsi        # 0x555555556de9
0x0000555555555b1d <+36>:    mov    $0x0,%eax
0x0000555555555b22 <+41>:    callq  0x555555554f90 <__isoc99_sscanf@plt>
0x0000555555555b27 <+46>:    add    $0x10,%rsp
0x0000555555555b2b <+50>:    cmp    $0x5,%eax
0x0000555555555b2e <+53>:    jle    0x555555555b35 <read_six_numbers+60>
```

disas phase_2를 해서 read_six_numbers함수를 확인하고 disas read_six_numbers를 해서 이 함수가 6개의 정수를 입력받아 스택에 저장하는 함수임을 확인합니다. +29를 실행시킨 후 x/s $rip+0x12cc를 해 본 결과 정수 6개를 입력받는 함수임을 알게됐고 입력한 수가 6개가 아닐시 +50과 +53에 의해 폭탄이 터집니다.

2.
```
0x00005555555552f2 <+30>:    cmpl   $0x0,(%rsp)
0x00005555555552f6 <+34>:    js     0x555555555302 <phase_2+46>
```

여기서 %rsp는 처음 입력한 수의 주소가 되고 처음 입력한 수(%rsp)가 음수이면 폭탄이 터지게 됩니다.

3.
```
0x00005555555552f8 <+36>:    mov    $0x1,%ebx
0x00005555555552fd <+41>:    mov    %rsp,%rbp
0x0000555555555300 <+44>:    jmp    0x555555555313 <phase_2+63>
0x0000555555555302 <+46>:    callq  0x555555555abd <explode_bomb>
0x0000555555555307 <+51>:    jmp    0x5555555552f8 <phase_2+36>
0x0000555555555309 <+53>:    add    $0x1,%rbx
0x000055555555530d <+57>:    cmp    $0x6,%rbx
0x0000555555555311 <+61>:    je     0x555555555326 <phase_2+82>
0x0000555555555313 <+63>:    mov    %ebx,%eax
0x0000555555555315 <+65>:    add    -0x4(%rbp,%rbx,4),%eax
0x0000555555555319 <+69>:    cmp    %eax,0x0(%rbp,%rbx,4)
0x000055555555531d <+73>:    je     0x555555555309 <phase_2+53>
```

%rbx에 1이 저장되고 %rbp에 처음 입력한 수의 주소가 저장됩니다. 그리고 +63으로 jump를 하고 %rax에 %rbx값이 저장됩니다. +69에 수 비교 후 je를 통해 같으면 +53으로 돌아갑니다. 여기서 +53부터 +73까지 반복문임을 알 수 있고 +53에서 %rbx에 1을 더해주고 +57, +61을 통해 %rbx가 6이되면 반복문을 탈출함을 알 수 있습니다. +63은 %rax에 %rbx값을 넣고 +65에서 %rax에 %rbp+4*%rbx-4가 가리키는 값을 더합니다. 그리고 +69에서 %rax의 값과 %rbp+4*%rbx가 가리키는 값이 같은지 비교 후 같으면 반복문을 계속 실행합니다. %rbx는 몇 번째 수인지 즉, 배열으로 비교하자면 index역할을 하는 수입니다. +63, +65에서 %rax에 index+이전 수를 저장해줍니다. 여기서 여섯 개의 수는 index+이전 수임을 알 수 있습니다. 따라서 답은 0 1 3 6 10 15임을 알 수 있습니다.

<phase_3: conditionals/swiches>

1.
```
0x000055555555535e <+28>:    lea    0x1a90(%rip),%rsi        # 0x555555556df5
0x0000555555555365 <+35>:    callq  0x555555554f90 <__isoc99_sscanf@plt>
0x000055555555536a <+40>:    cmp    $0x1,%eax
0x000055555555536d <+43>:    jle    0x555555555388 <phase_3+70>
```
+28을 실행 한 후 x/s $rip+0x1a90을 해보니 +35에서 함수를 호출하여 입력값을 두 개 받고 +40에서 return 값인 %rax와 1을 비교하고 있습니다. nexti 명령을 통해 +35에서의 %rax와 +40에서의 %rax를 확인해본 결과 +35에서는 0이 +40에서는 4가 저장돼있음을 확인했습니다. (phase_3입력으로 1 2 3 4를 넣은 상황) %rax에 입력받은 수의 개수가 저장됨을 확인했습니다. %rax와 1을 비교해 %rax가 1보다 작거나 같으면 폭탄으로 jump하게 됩니다. 즉 입력값의 수가 2개 이상이어야 합니다.

2.
```
0x000055555555536f <+45>:    cmpl   $0x7,(%rsp)
0x0000555555555373 <+49>:    ja     0x5555555553c0 <phase_3+126>
```
%rsp에는 첫 번째로 입력한 수의 주소값이 있습니다. 그 주소값이 가리키는 수가 7보다 크면 폭탄이 터지게 됩니다. 입력한 수의 첫 번째 수는 7보다 작아야합니다

3.
```
0x0000555555555375 <+51>:    mov    (%rsp),%eax
0x0000555555555378 <+54>:    lea    0x17a1(%rip),%rdx
0x000055555555537f <+61>:    movslq (%rdx,%rax,4),%rax
0x0000555555555383 <+65>:    add    %rdx,%rax
0x0000555555555386 <+68>:    jmpq   *%rax
```
+51번에서 %rax에 %rsp가 가리키는 값을 저장합니다. 여기서 %rax에 무엇이 저장됐는지 확인해본 결과 입력한 값의 첫 번째 수 1이 저장 돼있음을 확인합니다. +54에서 +65까지 연산과 대입을 진행하고 +68에서 %rax가 가리키는 주소값으로 jump를 합니다. +68에서 %rax에 저장된 값은 0x55555555538f입니다.

4.
```
0x000055555555538f <+77>:    mov    $0x25b,%eax
0x0000555555555394 <+82>:    jmp    0x5555555553d1 <phase_3+143>
0x0000555555555396 <+84>:    mov    $0x182,%eax
0x000055555555539b <+89>:    jmp    0x5555555553d1 <phase_3+143>
0x00005555555553a0 <+91>:    mov    $0x228,%eax
0x00005555555553a2 <+96>:    jmp    0x5555555553d1 <phase_3+143>
0x00005555555553a4 <+98>:    mov    $0x4d,%eax
0x00005555555553a9 <+103>:   jmp    0x5555555553d1 <phase_3+143>
0x00005555555553ab <+105>:   mov    $0x24f,%eax
0x00005555555553b0 <+110>:   jmp    0x5555555553d1 <phase_3+143>
0x00005555555553b2 <+112>:   mov    $0x1db,%eax
0x00005555555553b7 <+117>:   jmp    0x5555555553d1 <phase_3+143>
0x00005555555553b9 <+119>:   mov    $0x7e,%eax
0x00005555555553be <+124>:   jmp    0x5555555553d1 <phase_3+143>
```
0x55555555538f는 +77임을 확인하고 +77에서는 %rax에 0x25b(603)을 넣고 +143으로 jump를 합니다.

5.
```
0x00005555555553d1 <+143>:    cmp    %eax,0x4(%rsp)
0x00005555555553d5 <+147>:    je     0x5555555553dc <phase_3+154>
0x00005555555553d7 <+149>:    callq  0x555555555abd <explode bomb>
```

+143에서 %rax와 %rsp+4가 가리키는 수를 비교하여 다르면 폭탄이 터지게 됩니다. 여기서 %rsp+4는 입력한 두 번째 수의 주소값입니다. 여기서 phase_3는 첫 번째 입력한 수로 swich문을 통해 두 번째 입력한 수와 비교하는 것임을 알 수 있습니다. 따라서 답은 1 603입니다.

<phase_4: recursive calls and the stack disciplie>

1.
```
0x0000555555555451 <+28>:    lea    0x199d(%rip),%rsi        # 0x555555556df5
0x0000555555555458 <+35>:    callq  0x555555554f90 <__isoc99_sscanf@plt>
0x000055555555545d <+40>:    cmp    $0x2,%eax
0x0000555555555460 <+43>:    jne    0x555555555468 <phase_4+51>
```

위 phase에서와 같이 +28을 실행한 후 x/s $rip+0x199d를 통해 +35의 함수를 실행해 정수 두 개를 입력받아 +40과 +43에서 입력값이 두 개가 아닐시 폭탄이 터지게 됨을 확인합니다. (+35가 실행 된 후 %rax에는 입력된 인자 수가 저장됩니다.)

2.
```
0x000055555555547a <+69>:    callq  0x5555555553f6 <func4>
0x000055555555547f <+74>:    cmp    $0x1,%eax
0x0000555555555482 <+77>:    jne    0x55555555548b <phase_4+86>
```

func4를 호출하고 return 값인 %rax가 1이 아니면 폭탄이 터지게 됨을 알 수 있습니다.

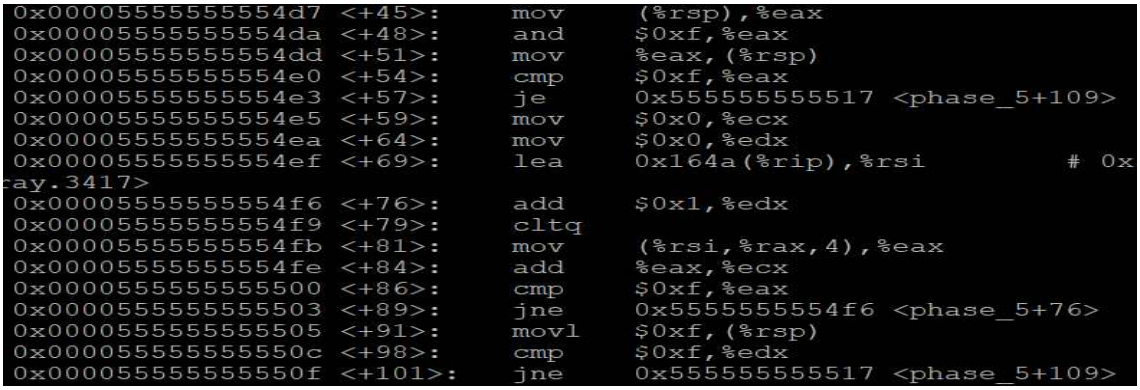
3.
```
0x0000555555555409 <+19>:    cmp    %edi,%ecx
0x000055555555540b <+21>:    jg     0x55555555541b <func4+37>
0x000055555555540d <+23>:    mov    $0x0,%eax
0x0000555555555412 <+28>:    cmp    %edi,%ecx
0x0000555555555414 <+30>:    jl     0x555555555427 <func4+49>
0x0000555555555416 <+32>:    add    $0x8,%rsp
0x000055555555541a <+36>:    retq
0x000055555555541b <+37>:    lea    -0x1(%rcx),%edx
0x000055555555541e <+40>:    callq  0x5555555553f6 <func4>
0x0000555555555423 <+45>:    add    %eax,%eax
0x0000555555555425 <+47>:    jmp    0x555555555416 <func4+32>
0x0000555555555427 <+49>:    lea    0x1(%rcx),%esi
0x000055555555542a <+52>:    callq  0x5555555553f6 <func4>
0x000055555555542f <+57>:    lea    0x1(%rax,%rax,1),%eax
0x0000555555555433 <+61>:    jmp    0x555555555416 <func4+32>
```

func4함수의 일부분인데 %rcx와 %rdi를 비교하여 같지않으면 func4를 다시 호출하는 것을 알 수 있습니다. 즉 재귀함수입니다. func4가 모두 종료됐을 때 return 값이 1이 돼야함을 알고있으므로 재귀함수가 %rcx가 %rdi보다 작을 때 한번 더 실행되고 %rcx와 %rdi가 같아 종료되게 해야함을 알 수 있습니다. 처음 func4가 실행되면 +19 전까지 연산을 진행하면 %rcx=7이 되고 첫 입력값보다 %rcx가 작다 가정하고 func4를 한번 더 실행하게 되면 %rcx=11이 됩니다. 즉 첫 번째 입력값은 11이 돼야함을 알 수 있습니다.

4. phase_4에서 %rdx에 14, %rsi에 0, %rdi에 입력된 수의 첫 번째 수가 저장돼 func4가 호출됨을 알 수 있고 cmpl $0x1, 0x4(%rsp)에서 입력된 수의 두 번째 수가 1이어야함을 알 수 있습니다. 따라서 정답은 11 1입니다.

<phase_5; pointers>

1. 위 phase들과 동일한 방법으로 정수 두 개를 입력받아야하는 것임을 알게 됐습니다.

2.


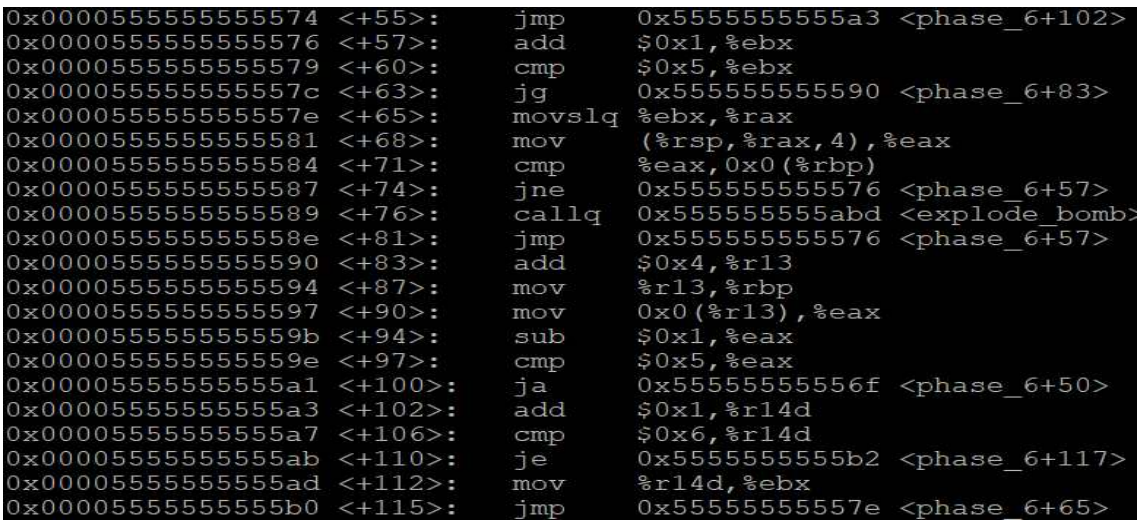
%rax에 처음 입력된 수를 넣고 0xf와 and연산을 진행합니다. 그리고 0xf와 비교해 같으면 폭탄이 터지므로 처음 입력된 수는 0xf이면 안됨을 알 수 있습니다. 다음 %rcx, %rdx에 0을 넣고 %rsi에 %rip+0x164a의 값을 넣습니다. 이 주소값을 확인해보니 배열의 시작 주소값이었습니다. +76부터 +89까지는 반복문임을 알 수 있습니다. %rdx는 index역할이 되고 탈출조건은 %rax가 0xf가 되는 것입니다. +81에서 배열을 통해 %rax 값을 변경시키고 있고 +84에서는 변경된 %rax값을 %rcx에 누적시키고 있습니다. +98, +101을 보면 index가 0xf가 돼야하므로 %rdx가 0xf가 되고 %rax가 마지막에 0xf가 되게하는 %rax를 계산해보면 5, 12, 3, 7, 11, 13, 9, 4, 8, 0, 10, 1, 2, 14, 6, 15가 되므로 처음 입력된 수는 5임을 알 수 있고 %rcx는 위 순서에서 12부터 15까지 더한 수 임을 알게됐습니다.

3.


두 번째로 입력된 수와 %rcx가 같아야함을 알 수 있습니다. 따라서 두 번째로 입력된 수는 115입니다. 따라서 phase_5의 답은 5 115임을 알 수 있습니다.

<phase_6: linked lists/pointers/structs>

1. phase_6에도 phase_2에서와 같은 read_six_numbers가 존재하므로 6개의 정수를 입력받는 것임을 알 수 있습니다.

2.


이는 이중반복문임을 알 수 있고 외부 반복문에서 하는 일은 입력한 6개의 숫자가 모두 6이하인지 검사를 진행하고 내부 반복문에서는 6개의 수가 모두 다른지 검사를 합니다.

3.
```
0x00005555555555b2 <+117>:    lea    0x18(%r12),%rcx
0x00005555555555b7 <+122>:    mov    $0x7,%edx
0x00005555555555bc <+127>:    mov    %edx,%eax
0x00005555555555be <+129>:    sub    (%r12),%eax
0x00005555555555c2 <+133>:    mov    %eax,(%r12)
Type <return> to continue, or q <return> to quit---
0x00005555555555c6 <+137>:    add    $0x4,%r12
0x00005555555555ca <+141>:    cmp    %r12,%rcx
0x00005555555555cd <+144>:    jne    0x5555555555bc <phase_6+127>
```

다음 +127에서 +144까지 반복문이 또 나옵니다. 이 반복문에서 하는 일은 7에서 입력된 수를 빼서 저장합니다. 예를 들어 첫 번째 입력값이 6이었다면 7-6이되어 1이됩니다.

4.
```
0x00005555555555d4 <+151>:    jmp    0x5555555555f0 <phase_6+179>
0x00005555555555d6 <+153>:    mov    0x8(%rdx),%rdx
0x00005555555555da <+157>:    add    $0x1,%eax
0x00005555555555dd <+160>:    cmp    %ecx,%eax
0x00005555555555df <+162>:    jne    0x5555555555d6 <phase_6+153>
0x00005555555555e1 <+164>:    mov    %rdx,0x20(%rsp,%rsi,8)
0x00005555555555e6 <+169>:    add    $0x1,%rsi
0x00005555555555ea <+173>:    cmp    $0x6,%rsi
0x00005555555555ee <+177>:    je     0x555555555606 <phase_6+201>
0x00005555555555f0 <+179>:    mov    (%rsp,%rsi,4),%ecx
0x00005555555555f3 <+182>:    mov    $0x1,%eax
0x00005555555555f8 <+187>:    lea    0x202c31(%rip),%rdx        # 0x555555758230 <node1>
0x00005555555555ff <+194>:    cmp    $0x1,%ecx
0x0000555555555602 <+197>:    jg     0x5555555555d6 <phase_6+153>
0x0000555555555604 <+199>:    jmp    0x5555555555e1 <phase_6+164>
```

다음 +153부터 +179까지 반복문이 생성됩니다. 이 반복문의 %rsp+8*%rsi+0x20에 7-입력값에 해당하는 node를 순서대로 저장합니다. 여기서 %rsi는 0부터 5까지 1씩 증가하는 값입니다. 예를 들어 입력값이 3 2 5 4 6 1 이었다면 %rsp+8*0+0x20에 node4가 저장됩니다.

5.
```
0x000055555555562f <+242>:    mov    0x48(%rsp),%rax
```

node6의 값을 찾지 못했는데 ni로 한단계씩 진행하며 %rax, %rbx, %rdx의 값을 확인해보니 +242가 실행된 후 %rax에 node6의 값이 저장됨을 확인했습니다.

6.
```
0x0000555555555645 <+264>:    jmp    0x555555555650 <phase_6+275>
0x0000555555555647 <+266>:    mov    0x8(%rbx),%rbx
0x000055555555564b <+270>:    sub    $0x1,%ebp
0x000055555555564e <+273>:    je     0x555555555661 <phase_6+292>
0x0000555555555650 <+275>:    mov    0x8(%rbx),%rax
0x0000555555555654 <+279>:    mov    (%rax),%eax
0x0000555555555656 <+281>:    cmp    %eax,(%rbx)
0x0000555555555658 <+283>:    jge    0x555555555647 <phase_6+266>
0x000055555555565a <+285>:    callq  0x555555555abd <explode_bomb>
```

+266에서 +283까지 반복문이 실행됩니다. 이 반복문에서는 %rbx에 %rsp+8*%rsi+0x20이 낮은 주소부터 저장됩니다. 즉 처음에 %rbx는 %rsp+8*0+0x20이 되고 입력값이 3 2 5 4 6 1 이었다면 node4가 저장된 주소값이 저장됩니다. %rax에는 그 다음 저장된 node값 즉 node5의 값이 저장되어 둘을 비교하여 %rbx가 작게되면 폭탄이 터집니다. 따라서 node값이 큰게 낮은 주소에 저장돼야합니다. node값은 순서대로 826 522 381 705 917 453입니다. 따라서 phase_6의 입력값은 2 6 3 5 1 4가 돼야합니다.