CSED211: Lab 3

손량(20220323)

Last compiled on: Saturday 30th September, 2023, 09:46

1 개요

실행 파일만 주어졌을 때, 어셈블리 명령어를 확인해 보면서 실행 파일의 동작을 분석해 본다.

2 코드 분석

2.1 `phase_1` – Phase 1

`objdump`를 사용해 얻은 디스어셈블리는 다음과 같다.

```
0000000000400ef0 <phase_1>:
  400ef0: 48 83 ec 08                   subq    $8, %rsp
  400ef4: be b0 24 40 00                movl    $4203696, %esi          #
      imm = 0x4024B0
  400ef9: e8 00 04 00 00                callq   0x4012fe <strings_not_equal>
  400efe: 85 c0                         testl   %eax, %eax
  400f00: 74 05                         je      0x400f07 <phase_1+0x17>
  400f02: e8 5d 06 00 00                callq   0x401564 <explode_bomb>
  400f07: 48 83 c4 08                   addq    $8, %rsp
  400f0b: c3                            retq
```

`esi` 레지스터에 `0x4024B0`이라는 주소를 넣고 `strings_not_equal` 함수를 호출한다. 이 함수의 디스어셈블리는 다음과 같다.

```
00000000004012e1 <string_length>:
  4012e1: 80 3f 00                      cmpb    $0, (%rdi)
  4012e4: 74 12                         je      0x4012f8 <string_length+0x17>
  4012e6: 48 89 fa                      movq    %rdi, %rdx
  4012e9: 48 83 c2 01                   addq    $1, %rdx
  4012ed: 89 d0                         movl    %edx, %eax
  4012ef: 29 f8                         subl    %edi, %eax
  4012f1: 80 3a 00                      cmpb    $0, (%rdx)
  4012f4: 75 f3                         jne     0x4012e9 <string_length+0x8>
  4012f6: f3 c3                         rep             retq
  4012f8: b8 00 00 00 00                movl    $0, %eax
  4012fd: c3                            retq

00000000004012fe <strings_not_equal>:
```

```
4012fe: 41 54                    pushq   %r12
401300: 55                       pushq   %rbp
401301: 53                       pushq   %rbx
401302: 48 89 fb                 movq    %rdi, %rbx
401305: 48 89 f5                 movq    %rsi, %rbp
401308: e8 d4 ff ff ff           callq   0x4012e1 <string_length>
40130d: 41 89 c4                 movl    %eax, %r12d
401310: 48 89 ef                 movq    %rbp, %rdi
401313: e8 c9 ff ff ff           callq   0x4012e1 <string_length>
401318: ba 01 00 00 00           movl    $1, %edx
40131d: 41 39 c4                 cmpl    %eax, %r12d
401320: 75 3e                    jne     0x401360
   <strings_not_equal+0x62>
401322: 0f b6 03                 movzbl  (%rbx), %eax
401325: 84 c0                    testb   %al, %al
401327: 74 24                    je      0x40134d
   <strings_not_equal+0x4f>
401329: 3a 45 00                 cmpb    (%rbp), %al
40132c: 74 09                    je      0x401337
   <strings_not_equal+0x39>
40132e: 66 90                    nop
401330: eb 22                    jmp     0x401354
   <strings_not_equal+0x56>
401332: 3a 45 00                 cmpb    (%rbp), %al
401335: 75 24                    jne     0x40135b
   <strings_not_equal+0x5d>
401337: 48 83 c3 01              addq    $1, %rbx
40133b: 48 83 c5 01              addq    $1, %rbp
40133f: 0f b6 03                 movzbl  (%rbx), %eax
401342: 84 c0                    testb   %al, %al
401344: 75 ec                    jne     0x401332
   <strings_not_equal+0x34>
401346: ba 00 00 00 00           movl    $0, %edx
40134b: eb 13                    jmp     0x401360
   <strings_not_equal+0x62>
40134d: ba 00 00 00 00           movl    $0, %edx
401352: eb 0c                    jmp     0x401360
   <strings_not_equal+0x62>
401354: ba 01 00 00 00           movl    $1, %edx
401359: eb 05                    jmp     0x401360
   <strings_not_equal+0x62>
40135b: ba 01 00 00 00           movl    $1, %edx
401360: 89 d0                    movl    %edx, %eax
401362: 5b                       popq    %rbx
401363: 5d                       popq    %rbp
401364: 41 5c                    popq    %r12
401366: c3                       retq
```

`string_length` 함수는 첫 번째 인자, 즉 `rdi`로 문자열의 주소를 받고, `rdx`에 주소를 넣고 1 증가시킨 다음 참조하는 것을 반복하여 C의 null terminator `'\0'`을 읽을 때까지 `eax`에 `edx`

- edi 값을 넣는다. 따라서 이 함수는 이름이 말하는 것처럼 문자열의 길이를 구함을 알 수 있다. strings_not_equal 함수의 경우 함수의 시작에서 r12, rbp, rbx 레지스터를 push 했다가 리턴 직전에 pop 하는 것을 볼 수 있다. 이는 함수에서 사용하는 레지스터의 값을 보존하기 위해서로 보인다. 우선 string_length 함수를 사용해 얻은 첫 번째 문자열 길이를 r12d 레지스터에 받고, 두 번째 문자열의 길이를 eax 레지스터로 받은 다음 두 문자열의 길이가 다른 경우 edx에 1을 넣고 점프한다. 점프한 곳에서는 edx에 있는 값을 eax에 넣고 리턴하기 때문에, 이는 문자열의 길이가 다르면 1을 리턴한다고 볼 수 있다. 이후 코드에서는 한 글자씩 두 문자열의 내용을 비교하여, null terminator를 만날 때까지 같다면 0을, 그렇지 않다면 1을 edx에 넣고 이를 eax에 넣어 반환한다. 즉, phase_1에 해당하는 코드는 0x4024B0 위치에 있는 문자열과 rdi에 있는 문자열을 비교하여 같다면 리턴한다. 0x4024B0 위치의 문자열을 확인해 본 결과, When I get angry, Mr. Bigglesworth gets upset. 임을 알 수 있었다.

2.2 phase_2 – Phase 2

디스어셈블 결과는 다음과 같다.

```
0000000000400f0c <phase_2>:
  400f0c: 55                          pushq   %rbp
  400f0d: 53                          pushq   %rbx
  400f0e: 48 83 ec 28                 subq    $40, %rsp
  400f12: 48 89 e6                    movq    %rsp, %rsi
  400f15: e8 80 06 00 00              callq   0x40159a <read_six_numbers>
  400f1a: 83 3c 24 00                 cmpl    $0, (%rsp)
  400f1e: 75 07                       jne     0x400f27 <phase_2+0x1b>
  400f20: 83 7c 24 04 01              cmpl    $1, 4(%rsp)
  400f25: 74 21                       je      0x400f48 <phase_2+0x3c>
  400f27: e8 38 06 00 00              callq   0x401564 <explode_bomb>
  400f2c: eb 1a                       jmp     0x400f48 <phase_2+0x3c>
  400f2e: 8b 43 f8                    movl    -8(%rbx), %eax
  400f31: 03 43 fc                    addl    -4(%rbx), %eax
  400f34: 39 03                       cmpl    %eax, (%rbx)
  400f36: 74 05                       je      0x400f3d <phase_2+0x31>
  400f38: e8 27 06 00 00              callq   0x401564 <explode_bomb>
  400f3d: 48 83 c3 04                 addq    $4, %rbx
  400f41: 48 39 eb                    cmpq    %rbp, %rbx
  400f44: 75 e8                       jne     0x400f2e <phase_2+0x22>
  400f46: eb 0c                       jmp     0x400f54 <phase_2+0x48>
  400f48: 48 8d 5c 24 08              leaq    8(%rsp), %rbx
  400f4d: 48 8d 6c 24 18              leaq    24(%rsp), %rbp
  400f52: eb da                       jmp     0x400f2e <phase_2+0x22>
  400f54: 48 83 c4 28                 addq    $40, %rsp
  400f58: 5b                          popq    %rbx
  400f59: 5d                          popq    %rbp
  400f5a: c3                          retq
```

우선 0x400f15 주소에서 read_six_numbers 함수를 호출하는 것을 볼 수 있다. 이 함수의 분석 결과, 이 함수는 문자열을 rdi, 즉 첫 번째 인자로 받고, 6개의 숫자를 저장할 배열을 rdi, 즉 두 번째 인자로 받아 저장한다. 여기에서는 여섯 개의 숫자는 rsp, rsp + 4, rsp + 8, rsp + 12, rsp + 16, rsp + 20 위치에 저장한다. 이후 이 함수는 배열의 0번 인덱스에

저장된 숫자, 즉 rsp 위치에 있는 숫자가 0인지, 그리고 1번 인덱스에 저장된 숫자, 즉 rsp + 4 위치에 있는 숫자가 1인지 확인하여 그렇지 않다면 폭탄을 터뜨린다. 이후에는 어셈블리 명령어로 이루어진 반복문이 실행되는데, rbx 레지스터에 배열의 2번 인덱스의 주솟값, 즉 rsp + 8의 주소를 넣고, rbp 레지스터에는 배열의 5번 인덱스 보다 4바이트 뒤의 주솟값, 즉 rsp + 24의 주솟값을 넣은 다음, rbx와 rbp의 값이 다른 동안 반복문을 수행해 -8(%rbx), -4(%rbx) 값을 더한 값이 (%rbx) 값과 같은지를 확인한다. 즉, 입력한 6개의 숫자들이 피보나치 숫자의 첫 6항과 같으면 된다. 따라서 phase 2를 해결하는 입력은 0 1 1 2 3 5 임을 알 수 있었다.

2.3 phase_3 – Phase 3

디스어셈블 결과는 다음과 같다.

```
0000000000400f5b <phase_3>:
  400f5b: 48 83 ec 18                 subq   $24, %rsp
  400f5f: 48 8d 4c 24 08              leaq   8(%rsp), %rcx
  400f64: 48 8d 54 24 0c              leaq   12(%rsp), %rdx
  400f69: be ad 27 40 00              movl   $4204461, %esi          #
     imm = 0x4027AD
  400f6e: b8 00 00 00 00              movl   $0, %eax
  400f73: e8 b8 fc ff ff              callq  0x400c30
     <__isoc99_sscanf@plt>
  400f78: 83 f8 01                    cmpl   $1, %eax
  400f7b: 7f 05                       jg     0x400f82 <phase_3+0x27>
  400f7d: e8 e2 05 00 00              callq  0x401564 <explode_bomb>
  400f82: 83 7c 24 0c 07              cmpl   $7, 12(%rsp)
  400f87: 77 3c                       ja     0x400fc5 <phase_3+0x6a>
  400f89: 8b 44 24 0c                 movl   12(%rsp), %eax
  400f8d: ff 24 c5 10 25 40 00        jmpq   *4203792(,%rax,8)
  400f94: b8 90 01 00 00              movl   $400, %eax              #
     imm = 0x190
  400f99: eb 3b                       jmp    0x400fd6 <phase_3+0x7b>
  400f9b: b8 6b 02 00 00              movl   $619, %eax              #
     imm = 0x26B
  400fa0: eb 34                       jmp    0x400fd6 <phase_3+0x7b>
  400fa2: b8 da 00 00 00              movl   $218, %eax
  400fa7: eb 2d                       jmp    0x400fd6 <phase_3+0x7b>
  400fa9: b8 d3 02 00 00              movl   $723, %eax              #
     imm = 0x2D3
  400fae: eb 26                       jmp    0x400fd6 <phase_3+0x7b>
  400fb0: b8 d6 00 00 00              movl   $214, %eax
  400fb5: eb 1f                       jmp    0x400fd6 <phase_3+0x7b>
  400fb7: b8 73 00 00 00              movl   $115, %eax
  400fbc: eb 18                       jmp    0x400fd6 <phase_3+0x7b>
  400fbe: b8 b7 01 00 00              movl   $439, %eax              #
     imm = 0x1B7
  400fc3: eb 11                       jmp    0x400fd6 <phase_3+0x7b>
  400fc5: e8 9a 05 00 00              callq  0x401564 <explode_bomb>
  400fca: b8 00 00 00 00              movl   $0, %eax
  400fcf: eb 05                       jmp    0x400fd6 <phase_3+0x7b>
```

```
  400fd1: b8 36 00 00 00          movl    $54, %eax
  400fd6: 3b 44 24 08             cmpl    8(%rsp), %eax
  400fda: 74 05                   je      0x400fe1 <phase_3+0x86>
  400fdc: e8 83 05 00 00          callq   0x401564 <explode_bomb>
  400fe1: 48 83 c4 18             addq    $24, %rsp
  400fe5: c3                      retq
```

우선 `0x400f73` 에서 `sscanf` 함수를 호출하는 것을 볼 수 있다. `sscanf` 함수는 두 번째 인자로 format string을 받는데, `esi` 레지스터에 넣은 `0x4027AD` 주소를 확인해 본 결과 `%d %d`임을 알 수 있었다. 즉, `sscanf` 함수를 통해 두 개의 정수를 입력받는다. 두 개의 정수는 `12(%rsp)`와 `8(%rsp)`에 저장된다. `sscanf`에서 두 개의 숫자를 입력받았는지 확인한 다음, `0x400f8d` 주소에서 jump table을 참조하여 indirect jump를 수행함을 알 수 있다. Jump table을 복구한 결과는 다음과 같다.

인덱스	주소
0	0x400f94
1	0x400fd1
2	0x400f9b
3	0x400fa2
4	0x400fa9
5	0x400fb0
6	0x400fb7
7	**0x400fbe**

어떤 주소로 점프하는 분기를 타든 `eax`의 값과 `8(%rsp)`의 값이 같으면 되므로 `0 400`을 입력하여 해결하였다.

2.4 `phase_4` – Phase 4

디스어셈블 결과는 다음과 같다.

```
0000000000400fe6 <func4>:
  400fe6: 41 54                   pushq   %r12
  400fe8: 55                      pushq   %rbp
  400fe9: 53                      pushq   %rbx
  400fea: 89 fb                   movl    %edi, %ebx
  400fec: 85 ff                   testl   %edi, %edi
  400fee: 7e 24                   jle     0x401014 <func4+0x2e>
  400ff0: 89 f5                   movl    %esi, %ebp
  400ff2: 89 f0                   movl    %esi, %eax
  400ff4: 83 ff 01                cmpl    $1, %edi
  400ff7: 74 20                   je      0x401019 <func4+0x33>
  400ff9: 8d 7f ff                leal    -1(%rdi), %edi
  400ffc: e8 e5 ff ff ff          callq   0x400fe6 <func4>
  401001: 44 8d 24 28             leal    (%rax,%rbp), %r12d
  401005: 8d 7b fe                leal    -2(%rbx), %edi
  401008: 89 ee                   movl    %ebp, %esi
  40100a: e8 d7 ff ff ff          callq   0x400fe6 <func4>
  40100f: 44 01 e0                addl    %r12d, %eax
  401012: eb 05                   jmp     0x401019 <func4+0x33>
```

```
  401014: b8 00 00 00 00               movl    $0, %eax
  401019: 5b                           popq    %rbx
  40101a: 5d                           popq    %rbp
  40101b: 41 5c                        popq    %r12
  40101d: c3                           retq

000000000040101e <phase_4>:
  40101e: 48 83 ec 18                  subq    $24, %rsp
  401022: 48 8d 4c 24 0c               leaq    12(%rsp), %rcx
  401027: 48 8d 54 24 08               leaq    8(%rsp), %rdx
  40102c: be ad 27 40 00               movl    $4204461, %esi          #
     imm = 0x4027AD
  401031: b8 00 00 00 00               movl    $0, %eax
  401036: e8 f5 fb ff ff               callq   0x400c30
     <__isoc99_sscanf@plt>
  40103b: 83 f8 02                     cmpl    $2, %eax
  40103e: 75 0c                        jne     0x40104c <phase_4+0x2e>
  401040: 8b 44 24 0c                  movl    12(%rsp), %eax
  401044: 83 e8 02                     subl    $2, %eax
  401047: 83 f8 02                     cmpl    $2, %eax
  40104a: 76 05                        jbe     0x401051 <phase_4+0x33>
  40104c: e8 13 05 00 00               callq   0x401564 <explode_bomb>
  401051: 8b 74 24 0c                  movl    12(%rsp), %esi
  401055: bf 07 00 00 00               movl    $7, %edi
  40105a: e8 87 ff ff ff               callq   0x400fe6 <func4>
  40105f: 3b 44 24 08                  cmpl    8(%rsp), %eax
  401063: 74 05                        je      0x40106a <phase_4+0x4c>
  401065: e8 fa 04 00 00               callq   0x401564 <explode_bomb>
  40106a: 48 83 c4 18                  addq    $24, %rsp
  40106e: c3                           retq
```

우선 `phase_4` 함수의 내용을 보면, 앞선 `phase_3`와 같이 두 수를 입력받는 것을 볼 수 있다. `0x401040`에서 `0xc(%rsp)`에 입력받은 수를 `eax`에 저장한 뒤, `eax`에서 2를 뺀 뒤 그 값이 2보다 작거나 같은지 검사한다. 만약 그렇지 않다면 폭탄을 터뜨리고, 조건이 만족되었다면 7, `0xc(%rsp)`를 각각 `func4` 함수의 첫 번째, 두 번째 인자에 넣고 호출한다. 최종적으로, `func4` 함수의 리턴 값이 `0x8(%rsp)`와 같아야 phase 4가 해결된다. `func4` 함수를 분석해 보자. 우선 `0x400fec`에서는 같은 값을 비교하기 때문에, 이후에 나오는 `jle` 명령어에서 점프가 일어나기 위해서는 `edi`의 MSB가 1이거나 `edi`에 저장된 값이 0이어야 한다. `jle`에 의해 점프가 일어난다면 함수는 0을 반환한다. 이후 나오는 코드를 따라가 보면, `func4`는 다음과 같이 정의된 재귀함수임을 알 수 있다.

$$f(x,y) = \begin{cases} 0 & (x \le 0) \\ y & (x = 1) \\ f(x-1,y) + y + f(x-2,y) & (x > 1) \end{cases}$$

$f(7,4) = 132$이므로, 이를 이용해 `132 4`를 입력하여 해결하였다. 물론 이 입력 외에도 함수에서 계산한 값과 맞는다면, 다른 입력으로도 phase 4를 해결할 수 있을 것이다.

2.5 `phase_5` – Phase 5

디스어셈블리는 다음과 같다.

```
000000000040106f <phase_5>:
  40106f: 53                            pushq   %rbx
  401070: 48 83 ec 10                   subq    $16, %rsp
  401074: 48 89 fb                      movq    %rdi, %rbx
  401077: e8 65 02 00 00                callq   0x4012e1 <string_length>
  40107c: 83 f8 06                      cmpl    $6, %eax
  40107f: 74 41                         je      0x4010c2 <phase_5+0x53>
  401081: e8 de 04 00 00                callq   0x401564 <explode_bomb>
  401086: eb 3a                         jmp     0x4010c2 <phase_5+0x53>
  401088: 0f b6 14 03                   movzbl  (%rbx,%rax), %edx
  40108c: 83 e2 0f                      andl    $15, %edx
  40108f: 0f b6 92 50 25 40 00          movzbl  4203856(%rdx), %edx
  401096: 88 14 04                      movb    %dl, (%rsp,%rax)
  401099: 48 83 c0 01                   addq    $1, %rax
  40109d: 48 83 f8 06                   cmpq    $6, %rax
  4010a1: 75 e5                         jne     0x401088 <phase_5+0x19>
  4010a3: c6 44 24 06 00                movb    $0, 6(%rsp)
  4010a8: be 06 25 40 00                movl    $4203782, %esi          #
     imm = 0x402506
  4010ad: 48 89 e7                      movq    %rsp, %rdi
  4010b0: e8 49 02 00 00                callq   0x4012fe <strings_not_equal>
  4010b5: 85 c0                         testl   %eax, %eax
  4010b7: 74 10                         je      0x4010c9 <phase_5+0x5a>
  4010b9: e8 a6 04 00 00                callq   0x401564 <explode_bomb>
  4010be: 66 90                         nop
  4010c0: eb 07                         jmp     0x4010c9 <phase_5+0x5a>
  4010c2: b8 00 00 00 00                movl    $0, %eax
  4010c7: eb bf                         jmp     0x401088 <phase_5+0x19>
  4010c9: 48 83 c4 10                   addq    $16, %rsp
  4010cd: 5b                            popq    %rbx
  4010ce: c3                            retq
```

우선 입력받은 문자열의 길이를 `string_length`에서 확인하여, 길이가 6이 아닌 경우 폭탄을 터뜨리는 것을 알 수 있다. 이후 `eax`에 0을 넣고 반복문을 수행하는데, 입력된 문자열의 0번째부터 5번째까지 각 문자의 ascii code의 마지막 4비트를 잘라내고, `0x402550` 주소에 있는 배열을 잘라낸 4비트로 인덱싱하여 얻은 값들을 `rsp`부터 `rsp + 5`까지 각각 써 넣음을 알 수 있다. 이후 `rsp + 6`에는 0을 써서 null terminated string으로 만드는 모습을 볼 수 있다. 이후 `rsp` 주소에 앞서 써 놓은 문자열과 `0x402506` 주소의 문자열과 비교하여 다른 경우 폭탄을 터뜨린다. `0x402506` 주소의 문자열은 `devils`이고, `0x402550` 위치에 있는 배열의 내용을 문자열으로 나타내면 `maduiersnfotvbyl`이다. Phase 5를 해결하기 위해서는 입력 문자열의 하위 4비트는 `[2, 5, 12, 4, 15, 7]`이어야 한다. 어떤 문자열을 넣든 하위 4비트가 이와 같다면 상관 없을 것이다. 이런 문자열의 예로 `be|dog` 등이 있다.

2.6 `phase_6` – Phase 6

디스어셈블리는 다음과 같다.

```
00000000004010cf <phase_6>:
  4010cf: 41 56                         pushq   %r14
  4010d1: 41 55                         pushq   %r13
```

```
4010d3: 41 54                   pushq   %r12
4010d5: 55                      pushq   %rbp
4010d6: 53                      pushq   %rbx
4010d7: 48 83 ec 50             subq    $80, %rsp
4010db: 4c 8d 6c 24 30          leaq    48(%rsp), %r13
4010e0: 4c 89 ee                movq    %r13, %rsi
4010e3: e8 b2 04 00 00          callq   0x40159a <read_six_numbers>
4010e8: 4d 89 ee                movq    %r13, %r14
4010eb: 41 bc 00 00 00 00       movl    $0, %r12d
4010f1: 4c 89 ed                movq    %r13, %rbp
4010f4: 41 8b 45 00             movl    (%r13), %eax
4010f8: 83 e8 01                subl    $1, %eax
4010fb: 83 f8 05                cmpl    $5, %eax
4010fe: 76 05                   jbe     0x401105 <phase_6+0x36>
401100: e8 5f 04 00 00          callq   0x401564 <explode_bomb>
401105: 41 83 c4 01             addl    $1, %r12d
401109: 41 83 fc 06             cmpl    $6, %r12d
40110d: 74 22                   je      0x401131 <phase_6+0x62>
40110f: 44 89 e3                movl    %r12d, %ebx
401112: 48 63 c3                movslq  %ebx, %rax
401115: 8b 44 84 30             movl    48(%rsp,%rax,4), %eax
401119: 39 45 00                cmpl    %eax, (%rbp)
40111c: 75 05                   jne     0x401123 <phase_6+0x54>
40111e: e8 41 04 00 00          callq   0x401564 <explode_bomb>
401123: 83 c3 01                addl    $1, %ebx
401126: 83 fb 05                cmpl    $5, %ebx
401129: 7e e7                   jle     0x401112 <phase_6+0x43>
40112b: 49 83 c5 04             addq    $4, %r13
40112f: eb c0                   jmp     0x4010f1 <phase_6+0x22>
401131: 48 8d 74 24 48          leaq    72(%rsp), %rsi
401136: 4c 89 f0                movq    %r14, %rax
401139: b9 07 00 00 00          movl    $7, %ecx
40113e: 89 ca                   movl    %ecx, %edx
401140: 2b 10                   subl    (%rax), %edx
401142: 89 10                   movl    %edx, (%rax)
401144: 48 83 c0 04             addq    $4, %rax
401148: 48 39 f0                cmpq    %rsi, %rax
40114b: 75 f1                   jne     0x40113e <phase_6+0x6f>
40114d: be 00 00 00 00          movl    $0, %esi
401152: eb 20                   jmp     0x401174 <phase_6+0xa5>
401154: 48 8b 52 08             movq    8(%rdx), %rdx
401158: 83 c0 01                addl    $1, %eax
40115b: 39 c8                   cmpl    %ecx, %eax
40115d: 75 f5                   jne     0x401154 <phase_6+0x85>
40115f: eb 05                   jmp     0x401166 <phase_6+0x97>
401161: ba f0 42 60 00          movl    $6308592, %edx          #
   imm = 0x6042F0
401166: 48 89 14 74             movq    %rdx, (%rsp,%rsi,2)
40116a: 48 83 c6 04             addq    $4, %rsi
```

```
40116e: 48 83 fe 18                  cmpq    $24, %rsi
401172: 74 15                        je      0x401189 <phase_6+0xba>
401174: 8b 4c 34 30                  movl    48(%rsp,%rsi), %ecx
401178: 83 f9 01                     cmpl    $1, %ecx
40117b: 7e e4                        jle     0x401161 <phase_6+0x92>
40117d: b8 01 00 00 00               movl    $1, %eax
401182: ba f0 42 60 00               movl    $6308592, %edx          #
    imm = 0x6042F0
401187: eb cb                        jmp     0x401154 <phase_6+0x85>
401189: 48 8b 1c 24                  movq    (%rsp), %rbx
40118d: 48 8d 44 24 08               leaq    8(%rsp), %rax
401192: 48 8d 74 24 30               leaq    48(%rsp), %rsi
401197: 48 89 d9                     movq    %rbx, %rcx
40119a: 48 8b 10                     movq    (%rax), %rdx
40119d: 48 89 51 08                  movq    %rdx, 8(%rcx)
4011a1: 48 83 c0 08                  addq    $8, %rax
4011a5: 48 39 f0                     cmpq    %rsi, %rax
4011a8: 74 05                        je      0x4011af <phase_6+0xe0>
4011aa: 48 89 d1                     movq    %rdx, %rcx
4011ad: eb eb                        jmp     0x40119a <phase_6+0xcb>
4011af: 48 c7 42 08 00 00 00 00      movq    $0, 8(%rdx)
4011b7: bd 05 00 00 00               movl    $5, %ebp
4011bc: 48 8b 43 08                  movq    8(%rbx), %rax
4011c0: 8b 00                        movl    (%rax), %eax
4011c2: 39 03                        cmpl    %eax, (%rbx)
4011c4: 7d 05                        jge     0x4011cb <phase_6+0xfc>
4011c6: e8 99 03 00 00               callq   0x401564 <explode_bomb>
4011cb: 48 8b 5b 08                  movq    8(%rbx), %rbx
4011cf: 83 ed 01                     subl    $1, %ebp
4011d2: 75 e8                        jne     0x4011bc <phase_6+0xed>
4011d4: 48 83 c4 50                  addq    $80, %rsp
4011d8: 5b                           popq    %rbx
4011d9: 5d                           popq    %rbp
4011da: 41 5c                        popq    %r12
4011dc: 41 5d                        popq    %r13
4011de: 41 5e                        popq    %r14
4011e0: c3                           retq
```

우선 `read_six_numbers` 함수를 사용하여 6개의 숫자를 `0x30(%rsp)`부터 시작하는 배열에 입력받는 것을 알 수 있다. 이후 `0x4010fb`부터 `0x40112f`의 반복문에서 배열의 0번 인덱스의 원소에서 1을 뺀 값이 5 이하임을 검사하고, 1번부터 5번 인덱스의 원소가 모두 0번 인덱스의 원소와 다른지 검사한 뒤, 1번 인덱스에서 1을 뺀 값이 5 이하, 2번에서 5번 인덱스 원소와 모두 다른지 검서... 를 반복한다. 즉, 이 부분의 반복문은 입력받은 숫자들이 1 이상 6 이하의 서로 다른 숫자들인지 확인하는 것이다. 이후 `0x40113e`부터 `0x40114b`까지의 반복문은 각 숫자들을 7에서 뺀다. 이후 반복문은 `0x6042f0` 주소부터 6개 있는 연결 리스트의 node들을 7에서 뺀 뒤의 배열에 적힌 숫자대로 노드를 재배열한다. 각 node들의 첫 quad word의 뒤쪽 부분에는 숫자들이 적혀 있는데, 이 숫자들이 재배열 후 역순정렬되어 있지 않다면 폭탄이 터진다. 따라서 재배열 후 역순정렬이 되도록 하는 입력을 찾았고, `3 1 6 4 5 2`라는 입력을 찾을 수 있었다.

이로써 lab 3를 성공적인 폭탄 해체와 함께 마칠 수 있었다.

2.7 `secret_phase` – Secret Phase

사실 secret phase가 하나 더 있었다. 별로 의심스럽지 않은 `phase_defused` 함수의 내부를 보면 다음과 같다.

```
0000000000401702 <phase_defused>:
  401702: 48 83 ec 68                 subq    $104, %rsp
  401706: bf 01 00 00 00              movl    $1, %edi
  40170b: e8 90 fd ff ff              callq   0x4014a0 <send_msg>
  401710: 83 3d 85 30 20 00 06        cmpl    $6, 2109573(%rip)       #
     0x60479c <num_input_strings>
  401717: 75 6d                       jne     0x401786 <phase_defused+0x84>
  401719: 4c 8d 44 24 10              leaq    16(%rsp), %r8
  40171e: 48 8d 4c 24 08              leaq    8(%rsp), %rcx
  401723: 48 8d 54 24 0c              leaq    12(%rsp), %rdx
  401728: be f7 27 40 00              movl    $4204535, %esi          #
     imm = 0x4027F7
  40172d: bf b0 48 60 00              movl    $6310064, %edi          #
     imm = 0x6048B0
  401732: b8 00 00 00 00              movl    $0, %eax
  401737: e8 f4 f4 ff ff              callq   0x400c30
     <__isoc99_sscanf@plt>
  40173c: 83 f8 03                    cmpl    $3, %eax
  40173f: 75 31                       jne     0x401772 <phase_defused+0x70>
  401741: be 00 28 40 00              movl    $4204544, %esi          #
     imm = 0x402800
  401746: 48 8d 7c 24 10              leaq    16(%rsp), %rdi
  40174b: e8 ae fb ff ff              callq   0x4012fe <strings_not_equal>
  401750: 85 c0                       testl   %eax, %eax
  401752: 75 1e                       jne     0x401772 <phase_defused+0x70>
  401754: bf 58 26 40 00              movl    $4204120, %edi          #
     imm = 0x402658
  401759: e8 e2 f3 ff ff              callq   0x400b40 <puts@plt>
  40175e: bf 80 26 40 00              movl    $4204160, %edi          #
     imm = 0x402680
  401763: e8 d8 f3 ff ff              callq   0x400b40 <puts@plt>
  401768: b8 00 00 00 00              movl    $0, %eax
  40176d: e8 ad fa ff ff              callq   0x40121f <secret_phase>
  401772: bf b8 26 40 00              movl    $4204216, %edi          #
     imm = 0x4026B8
  401777: e8 c4 f3 ff ff              callq   0x400b40 <puts@plt>
  40177c: bf e8 26 40 00              movl    $4204264, %edi          #
     imm = 0x4026E8
  401781: e8 ba f3 ff ff              callq   0x400b40 <puts@plt>
  401786: 48 83 c4 68                 addq    $104, %rsp
  40178a: c3                          retq
  40178b: 0f 1f 44 00 00              nopl    (%rax,%rax)
```

`0x401710`에서 현재 phase의 값이 6인지 확인하고 `0x401737`에서 `sscanf` 결과 3개의 문자열을 파싱하였으며, 세 번째 문자열이 `DrEvil`일 때 `secret_phase`가 실행된다. 이때 파싱 대상이 되는 문자열은 phase 4에서 입력받는 문자열이다. 따라서 phase 4의 입력에서 `132`

4 DrEvil을 입력하면 secret_phase를 볼 수 있다. 디스어셈블 해보면 다음과 같다.

```
00000000004011e1 <fun7>:
  4011e1: 48 83 ec 08                   subq    $8, %rsp
  4011e5: 48 85 ff                      testq   %rdi, %rdi
  4011e8: 74 2b                         je      0x401215 <fun7+0x34>
  4011ea: 8b 17                         movl    (%rdi), %edx
  4011ec: 39 f2                         cmpl    %esi, %edx
  4011ee: 7e 0d                         jle     0x4011fd <fun7+0x1c>
  4011f0: 48 8b 7f 08                   movq    8(%rdi), %rdi
  4011f4: e8 e8 ff ff ff                callq   0x4011e1 <fun7>
  4011f9: 01 c0                         addl    %eax, %eax
  4011fb: eb 1d                         jmp     0x40121a <fun7+0x39>
  4011fd: b8 00 00 00 00                movl    $0, %eax
  401202: 39 f2                         cmpl    %esi, %edx
  401204: 74 14                         je      0x40121a <fun7+0x39>
  401206: 48 8b 7f 10                   movq    16(%rdi), %rdi
  40120a: e8 d2 ff ff ff                callq   0x4011e1 <fun7>
  40120f: 8d 44 00 01                   leal    1(%rax,%rax), %eax
  401213: eb 05                         jmp     0x40121a <fun7+0x39>
  401215: b8 ff ff ff ff                movl    $4294967295, %eax       #
     imm = 0xFFFFFFFF
  40121a: 48 83 c4 08                   addq    $8, %rsp
  40121e: c3                            retq

000000000040121f <secret_phase>:
  40121f: 53                            pushq   %rbx
  401220: e8 b7 03 00 00                callq   0x4015dc <read_line>
  401225: ba 0a 00 00 00                movl    $10, %edx
  40122a: be 00 00 00 00                movl    $0, %esi
  40122f: 48 89 c7                      movq    %rax, %rdi
  401232: e8 c9 f9 ff ff                callq   0x400c00 <strtol@plt>
  401237: 48 89 c3                      movq    %rax, %rbx
  40123a: 8d 40 ff                      leal    -1(%rax), %eax
  40123d: 3d e8 03 00 00                cmpl    $1000, %eax             #
     imm = 0x3E8
  401242: 76 05                         jbe     0x401249 <secret_phase+0x2a>
  401244: e8 1b 03 00 00                callq   0x401564 <explode_bomb>
  401249: 89 de                         movl    %ebx, %esi
  40124b: bf 10 41 60 00                movl    $6308112, %edi          #
     imm = 0x604110
  401250: e8 8c ff ff ff                callq   0x4011e1 <fun7>
  401255: 83 f8 02                      cmpl    $2, %eax
  401258: 74 05                         je      0x40125f <secret_phase+0x40>
  40125a: e8 05 03 00 00                callq   0x401564 <explode_bomb>
  40125f: bf e0 24 40 00                movl    $4203744, %edi          #
     imm = 0x4024E0
  401264: e8 d7 f8 ff ff                callq   0x400b40 <puts@plt>
  401269: e8 94 04 00 00                callq   0x401702 <phase_defused>
  40126e: 5b                            popq    %rbx
```

```
  40126f: c3                                    retq
```

우선 입력받은 문자열을 `strtol` 함수를 통해 정수로 변환하고, 1을 뺀 숫자가 1000 초과일 때 폭탄을 터뜨린다. 1000 이하라면 `fun7` 함수를 실행한다. `fun7` 함수에서 리턴한 값이 2여야 phase가 해결된다. `fun7`의 코드를 파이썬과 비슷한 언어로 번역하면 다음과 같다.

```
def fun7(rdi, rsi):
    if rdi == 0:
        eax = 0xffffffff
        return eax
    edx = *rdi
    if edx <= esi:
        eax = 0
        if edx == esi:
            return eax
        else:
            rdi = *(rdi + 0x10)
            eax = fun7(rdi, rsi)
            eax = eax + eax + 1
            return eax
    else:
        rdi = *(rdi + 8)
        eax = fun7(rdi, rsi)
        eax += eax
        return eax
```

`fun7`의 코드를 하나하나 분석해 문제를 해결할 수도 있겠지만, 코드의 내용이 간단하고, 참조하는 메모리 위치도 예측 가능할 뿐더러 앞선 검사에서 입력 범위를 1부터 1001까지로 제한한다. 따라서 `fun7`의 코드를 파이썬으로 구현하고, 1부터 1001까지의 숫자들들 모두 시도하여 2가 나오는 입력을 찾아보았다. 이때 입력으로 20을 넣으면 2가 리턴됨을 확인하였고, 이를 입력해 secret phase까지 해결할 수 있었다.