**CSED211 컴퓨터SW시스템개론**

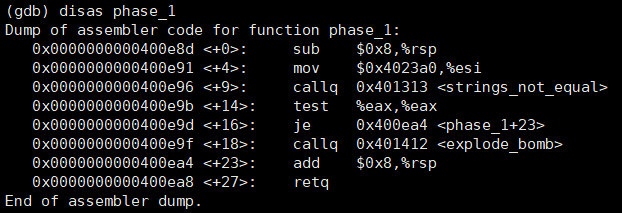
**Lab Assignment #3: Defusing a Binary Bomb**

**20180551**

**컴퓨터공학과**

**이준석**

**【Phase1】**

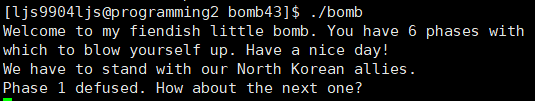


우선 phase1을 disassemble한 모습이다. Strings\_not\_equal함수에 매개변수로서 전달될 esi에 어떠한 값이 저장되는데, 무엇인지 확인해야한다.

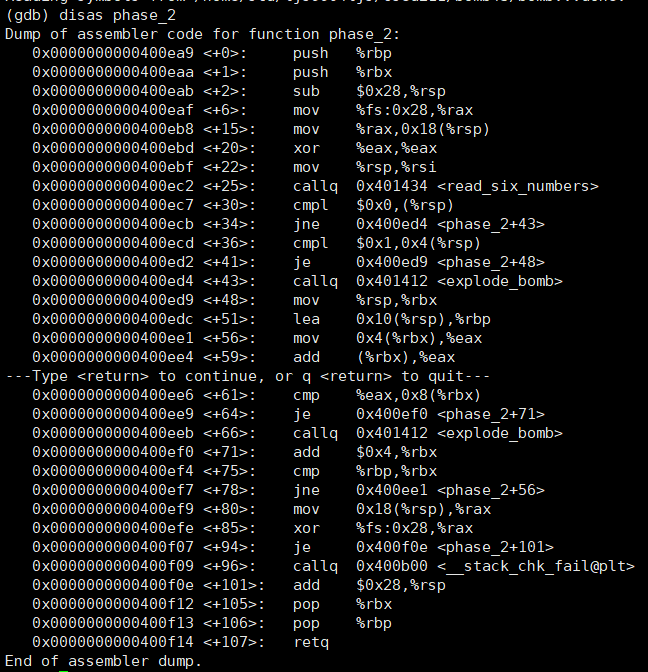
값을 확인하기 위해서 x/s 0x4023a0을 한다.



이것을 입력하면 phase1이 통과된다.



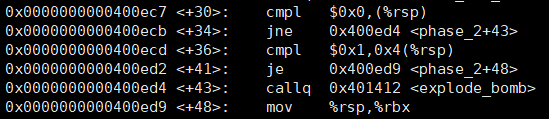
**【Phase2】**



Phase 2를 disassemble한 모습이다.



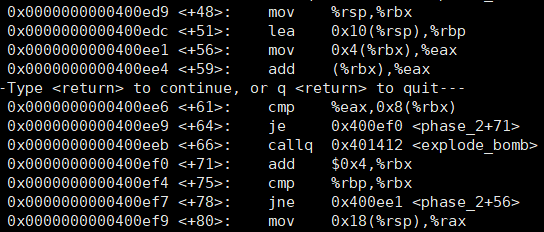
가장 먼저 6개의 숫자를 입력 받는다는 것을 알 수 있다.



<+30, +34> : 첫 입력이 0이 아니면 <+43>로 이동해서 폭탄이 폭발한다.

<+36, +41> : 두 번째 입력이 1이면 <+48>로 이동해서 <+43>을 피해서 폭발을 피할 수 있다.

즉, 첫 입력은 0이고 두 번째 입력은 1이어야 한다.



<+48> : 첫 입력을 rbx에 저장한다.

<+51> : 5번째 주소를 rbp에 저장한다.

<+56> : 두 번째 입력을 eax에 저장한다.

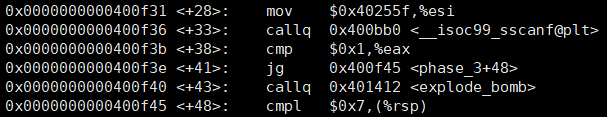
<+59> : eax값에 rbx값을 더한다.

<+61, +64> : eax값과 3번째 입력 값을 비교한다. 그리고 다르면 폭탄이 폭발한다. 같아야 한다.

즉, A(n+2) = A(n+1) + A(n)을 확인하는 것이다. 아래의 코드는 이 과정을 반복시키기 위한 코드이다. 즉, 피보나치 수열에 관한 함수가 바로 phase 2였던 것이다.

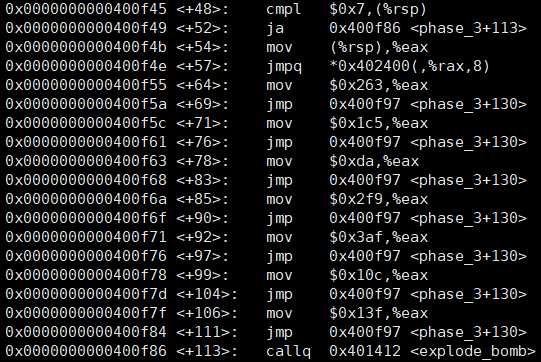
따라서 최종적인 입력은 0, 1, 1, 2, 3, 5 이 된다.

**【Phase3】**





Phase3는 숫자 2개를 입력 받는다.

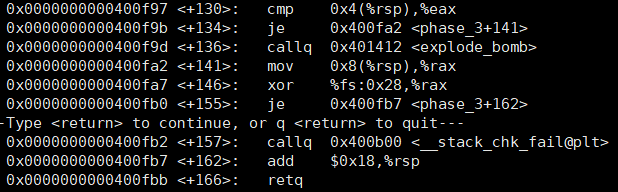


<+48, +113> : (첫 입력의 값 – 0x7)이 0이거나 CF가 1이어야 한다. 그렇지 않으면 <+113>으로 이동해서 폭탄이 폭발한다.

<+54> : ‘0x402400 + 8 곱하기 rax의 값’이 가리키는 곳으로 점프한다. 첫 입력을 7을 준다고 가정하자. 그러면 0x402400 + 56(10진수) = 0x402438이므로 그 주소를 확인해본다.



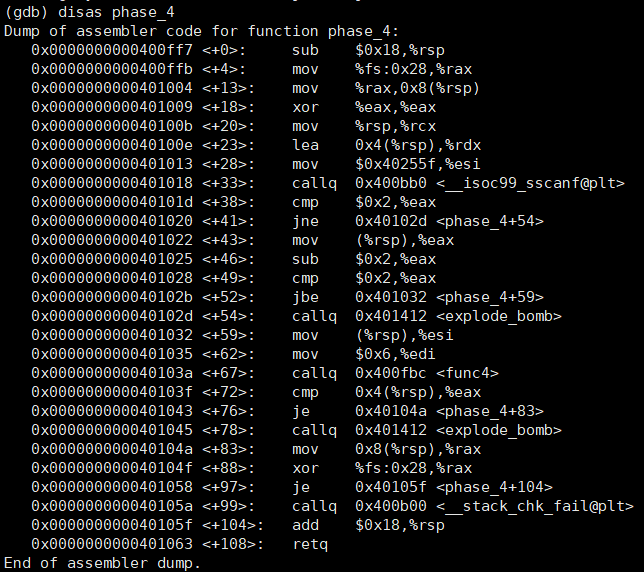
즉, <+106>으로 점프한다. 따라서 eax에는 0x13f가 저장된다. 그리고 <+130>으로 점프한다.



<+130> : 두 번째 입력 값을 eax의 값과 비교한다. 둘이 같지 않다면 폭탄이 폭발한다. 그러므로 두 값이 같아야 한다. 따라서 두 번째 입력 값은 0x13f(십진수: 319)이어야만 한다.

따라서 최종적인 입력은 [ 7 319 ] 가 된다.

**【Phase4】**





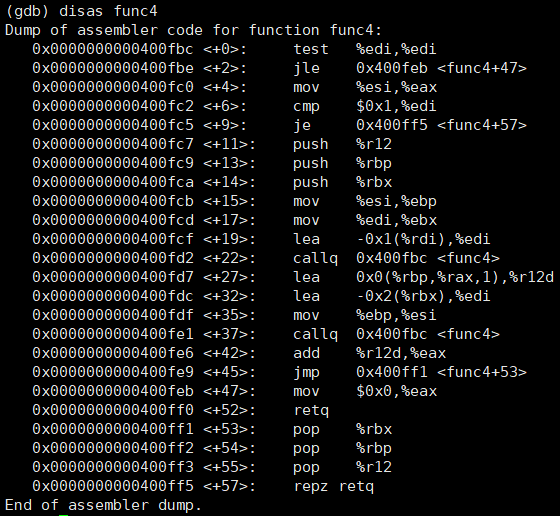
Phase 4는 숫자 2개를 입력 받고 진행된다.

<+43, +46, +49, +52> : eax = (첫 입력) – 2 하고, eax가 2 이하가 아니라면 폭탄이 폭발한다.

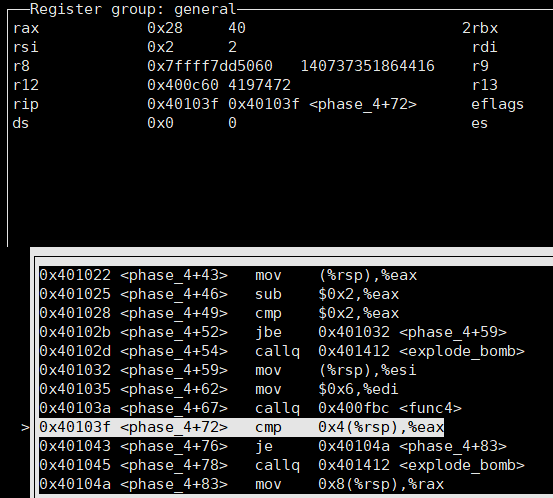
<+59> : 첫 입력을 esi에 저장한다.

<+62> : 6을 edi에 저장한다.

<+67> : func4를 호출한다.



<+22>를 보면 자기 자신을 다시 한 번 호출한다. 즉, 재귀함수이다.



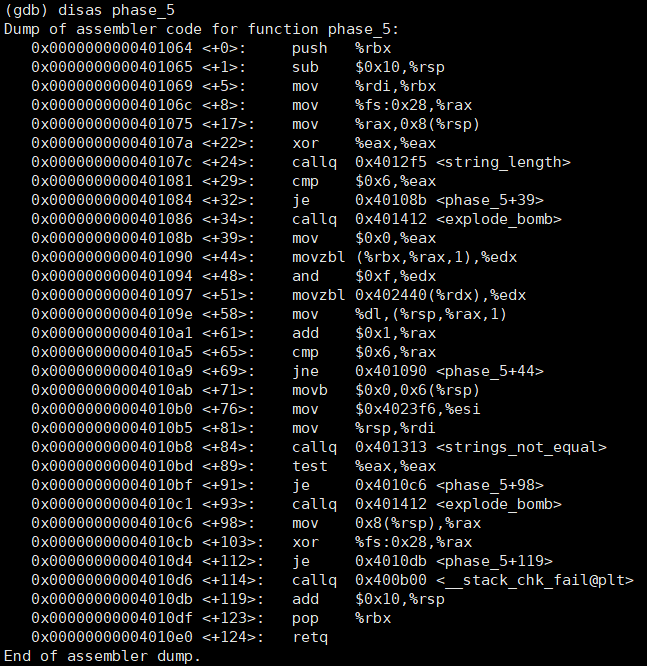
30 2를 입력값으로 주고 func4의 리턴값을 확인한 결과이다. 리턴값이 저장되는 rax에 40이 저장된 것이 확인된다. 즉, 0x4(%rsp)의 값이 40과 같아야 폭발을 피할 수 있다는 뜻이다.



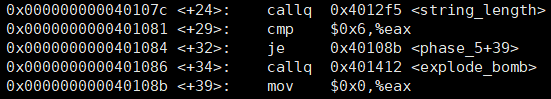
30 2를 입력했는데 2 30으로 저장된다. 즉, 0x4(%rsp)의 값은 첫 번째 입력에 대응된다. 따라서 첫 입력으로 40을 주어야 폭발을 피할 수 있다는 것을 알 수 있다.

따라서 최종적인 입력값은 40 2

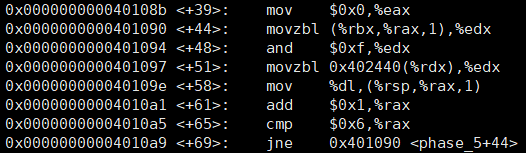
**【Phase5】**



Phase5를 disassemble한 모습이다. 인자(본인이 입력한 문자열)는 rdi에서 rbx로 전달된다.



입력한 스트링이 6글자가 아니라면 폭탄이 폭발한다는 것을 알 수 있다. 즉, 6글자를 입력값으로 주어야 한다.



<+44> : rbx + (rax \* 1)의 형식으로 주소를 이용하는 것으로 보아 1바이트의 데이터 타입인 char가 사용되는 것으로 보인다. 그리고 처음에 <+39>에서 eax가 0이었으므로 처음으로 참조되는 값은 rbx, 즉 첫 글자이다.

<+48> : 글자의 8개의 비트(1바이트이므로) 중에서 하위 4개의 비트만을 취한다. 상위 4개의 비트는 0으로 채운다. 즉, 0000XXXX 와 같은 모양을 가지게 된다.

<+51> : 8개의 비트 값 + 0x402440을 한 것을 하위 8개의 비트만을 취해서 edx에 다시 저장한다.

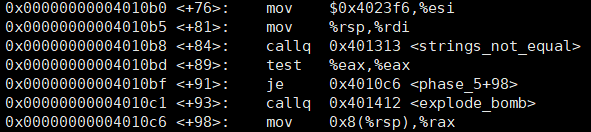
<+58> : 그 8개의 비트값을 rsp + (rax \* 1)에 저장한다.

<+61> : rax의 크기를 1만큼 증가시킨다.

<+65, +69> : rax가 6과 같지 않다면 다시 <+44>로 돌아가서 위의 과정을 반복하게 만든다.



0x402440부터 0x402448+0x8까지 문자가 저장되어 있는 일종의 표이다. 이것을 이용하여 추후의 문자열 비교를 수행하게 될 것이다.



0x4023f6에 저장되어 있는 문자열과 비교해서 같지 않으면 폭탄이 폭발한다. 따라서 그 값을 확인해보면,

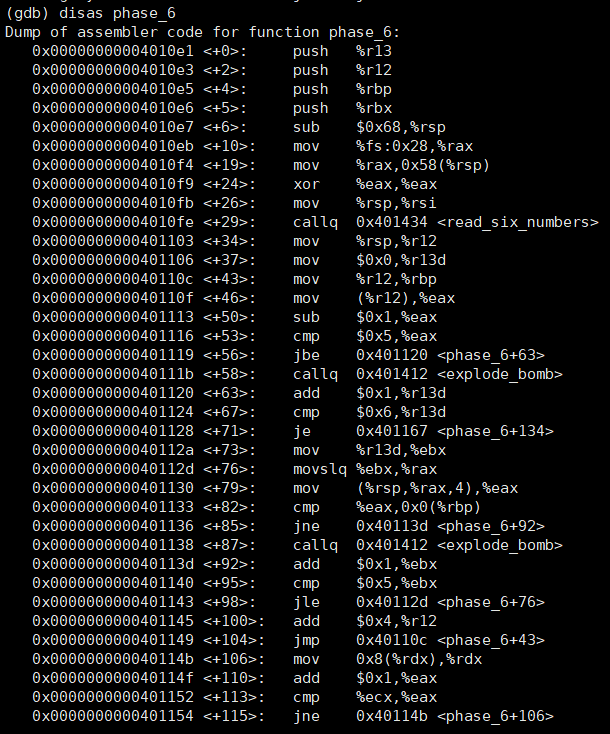


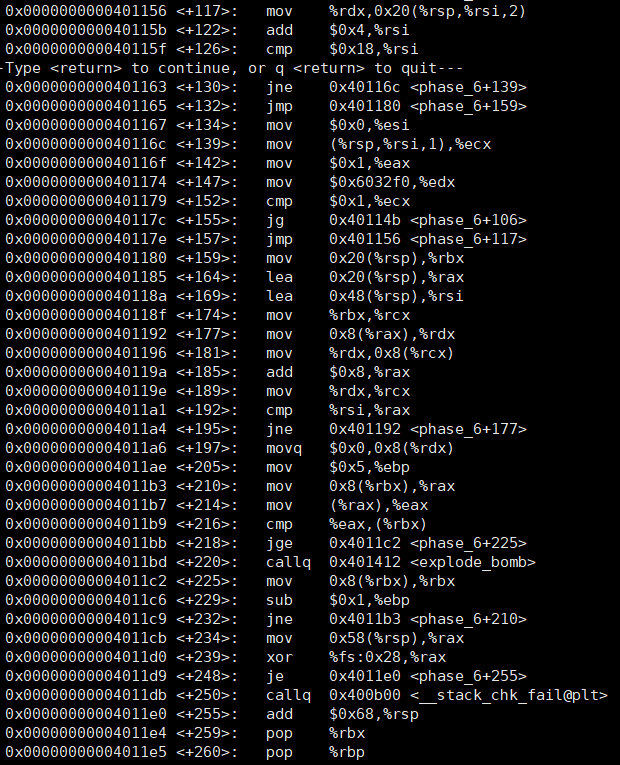
s a b r e s 라는 것을 알 수 있다.

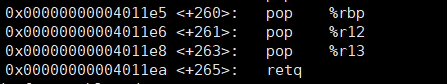
0x402440에서 s는 +7, a는 +1, b는 +13, r은 +6, e는 +5이다. 따라서 아스키 코드를 보고 하위 4비트가 0111, 0001, 1101, 0110, 0101인 문자들을 찾으면 된다. (<+48>에서 하위 4비트만을 취하므로)

따라서 최종적인 입력 문자열은 gamfeg

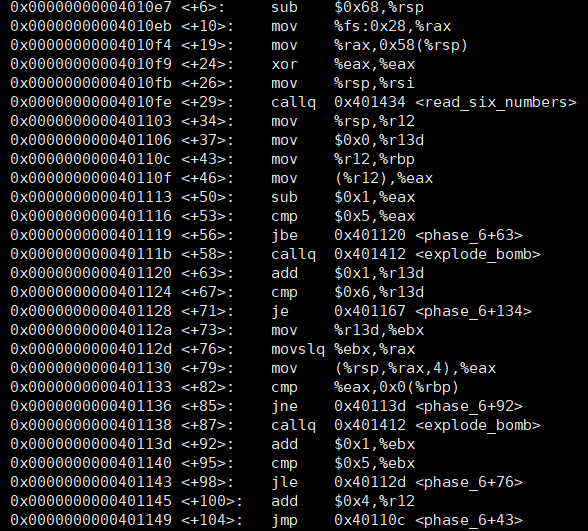
**【Phase6】**







phase 6를 disassemble한 모습이다.



숫자 6개를 입력 받는다. 그 숫자들이 모두 6이하인지 확인한다. 그리고 그 숫자들이 서로 서로와 모두 다른 지를 확인한다.

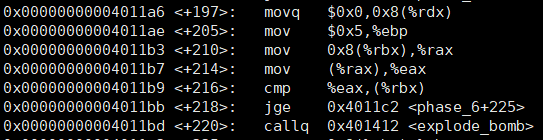
첫 입력을 기준으로 삼고 2번째, 3번째, .. 6번째 입력과 같은지를 비교한다. 그리고 2번째 입력을 기준으로 삼고 3번째, 4번째, ..., 6번째 입력과 같은지를 비교한다. 이 과정을 전부 반복하여 중복된 수와 6 초과의 수가 있는지를 확인하고 폭발 여부를 결정한다.

<+53 ~ +58> : 6 초과 검사

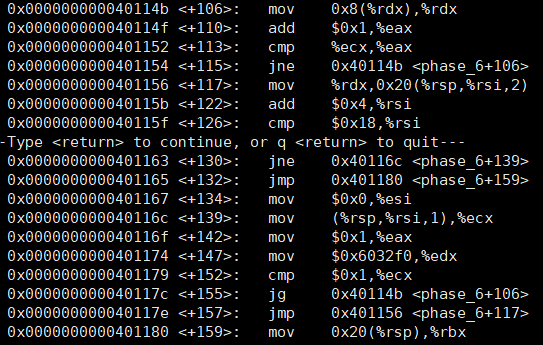
<+67 ~ +71> : 검사를 모두 마친 후 다음 단계(<+134>)로 점프

<+76 ~ +98> : 중복 검사

< 그 외 > : 반복문을 위한 조건 변경 및 점프



<+216>에서 rax에 저장되는 값은 결국 rbx+0x8의 값이다. 따라서 폭발을 피하려면, 앞서 온 값이 이후에 온 값보다 커야 한다. 즉, 어떠한 값이 내림차순이어야 한다.



rbx에 저장되는 값이 무엇인지 확인하기 위해 위의 코드로 거슬러 올라간다.

rbx에는 rsp + 0x20의 값이 저장된다. 그렇다면 rsp + 0x20에는 무엇이 저장되는가?

<+134>로 점프해서 왔으므로 그곳을 확인해 본다.

rsp + rsi의 값이 ecx에 저장되고 0x6032f0가 edx에 저장된다. eax에는 1이 저장된다.

ecx와 1을 비교하고 ecx가 0x1보다 크다면 <+106>으로 점프한다. 아니라면 <+117>로 점프한다.

<+106> : rdx에 저장되어 있는 값을 0x8만큼 증가시킨다.

<+110> : eax를 1만큼 증가시킨다.

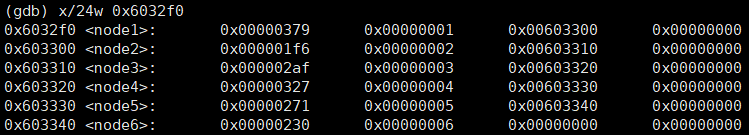
<+113, +115> : eax와 ecx의 값이 같아질 때까지 <+106, +110>을 반복한다.

같아지면 그 때의 rdx를 rsp + 0x20 + rsi \* 2에 저장한다.

<+122, +126> : <+122>에서 rsi를 4만큼씩 증가시키고 <+126>에서 0x18과 같은지 아닌지 조건 비교를 한다. 즉 rsi는 0부터 시작하므로 크게 6번만 반복문을 수행하겠다는 의미이다.

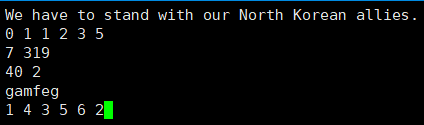
결과적으로 rsp에 저장되는 값은 rdx와 관련있고 rdx에 저장되는 값은 0x6032f0과 관련이 있다.

따라서 0x6032f0에 무엇이 있는지 확인해보아야 한다.



총 6개의 node를 갖고 있는 링크드 리스트임을 확인할 수 있다. 각 node에는 값, 자기 자신의 번호, 다음 node의 주소가 저장되어 있다. 값을 내림차순으로 node의 번호를 배치하는 것이 최종적인 입력 값이 될 것이다.

즉, 1 4 3 5 6 2



각 phase 별로 모든 답안을 모아둔 것은 위의 그림과 같다.