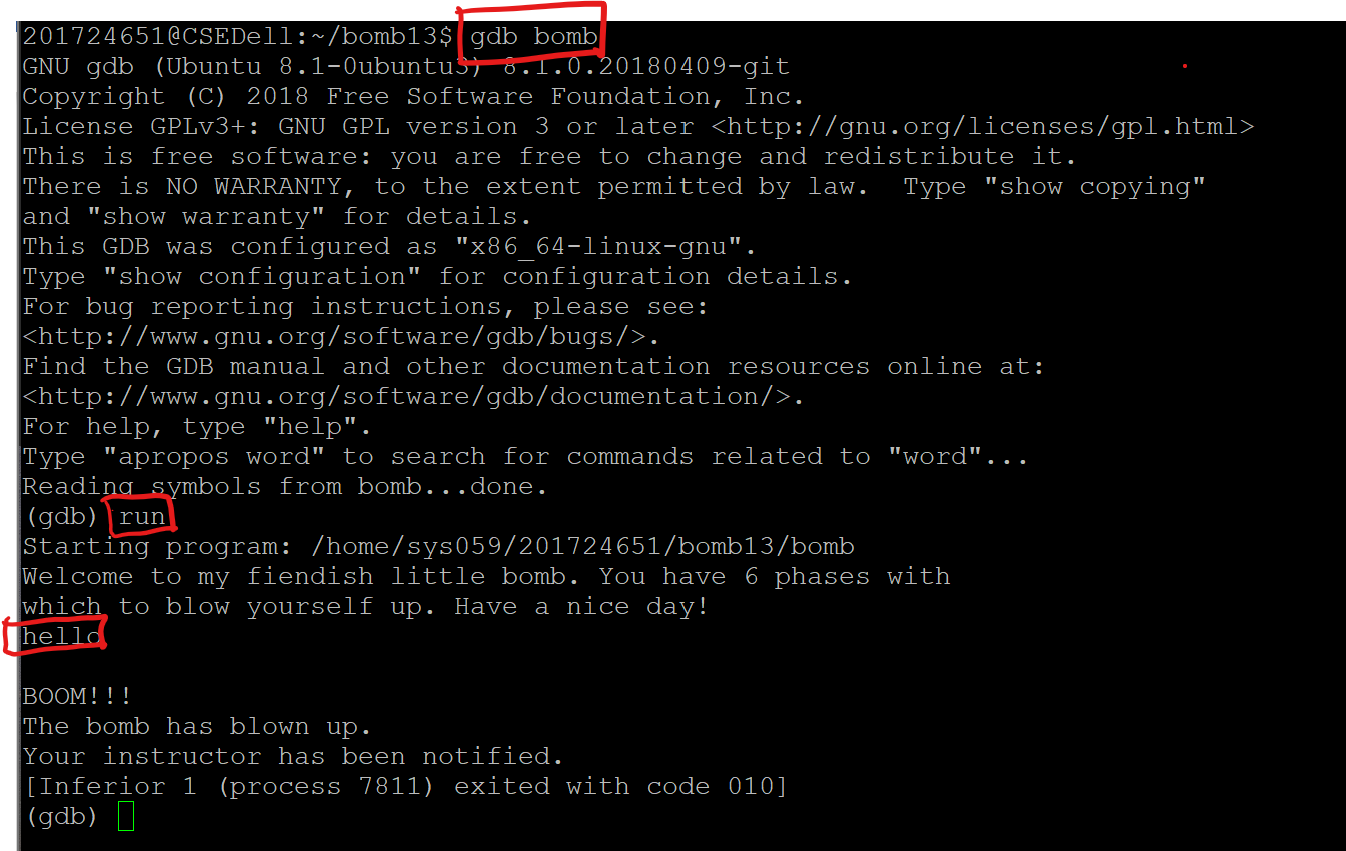


우선 가장 먼저 disas main 명령어를 통해 main문을 확인해볼 수 있다. 위에서 아래로 쭉 살펴보면 initialize\_bomb 부터 phase\_1~phase\_6까지 6단계가 있다 정도를 확인할 수 있다.

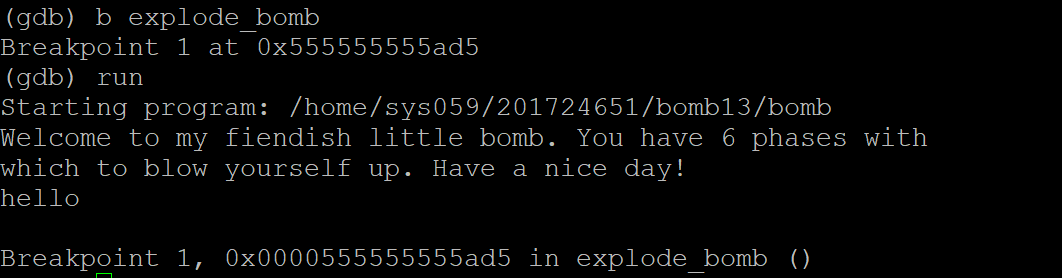
(사진은 페이지를 너무 많이 차지하므로 생략)

gdb로 들어와 run 혹은 r을 입력하여 프로그램을 실행할 수 있다.

Welcome to~~ 등의 환영한다는 메시지가 나오고 hello를 입력해봤더니 폭탄이 터져버린다.



그래서 폭탄이 바로 터져버리는 것을 방지하기 위해 b explode\_bomb이라는 명령어를 통해 break point를 생성해준다. 그러면 다시 run을 통해 프로그램을 실행하고 hello를 입력하더라도 폭탄이 터지지 않고 멈춘다. 처음에는 phase\_1에 break point를 생성해주었으나 그러면 올바른 답을 입력하더라도 다음 단계로 넘어가지 않고 phase\_1에 멈추는 것을 발견하였다.

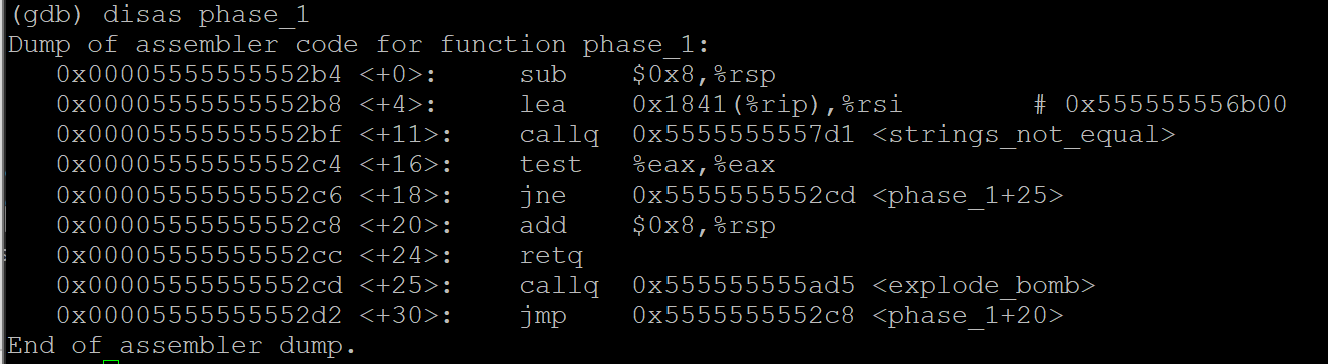


그러면 disas phase\_1 명령어로 phase\_1을 살펴보자.

+0 줄의 왼쪽을 보면 프로그램이 현재 위치를 알 수 있다. 그 아래로는 여러 명령어들을 볼 수 있는데 +11에서 함수를 부르고 +16에서 함수의 return value가 들어갈 eax 레지스터를 테스트하고 \_18에서 값을 비교하여 jne(jump if not equal), 즉 같지 않으면 +25로 점프하여 explode\_bomb을 만나 폭탄이 터지게 되어있다. +11에서 부르는 함수의 이름으로 유추해볼 수 있듯이 strings\_not\_equal -> string이 같이 않으면 1, 다르면 0을 return한다.

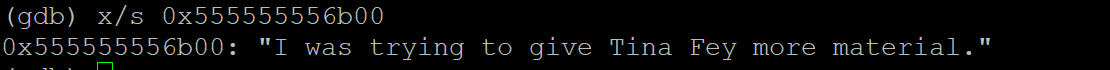
우선 한 가지 설명하고 넘어가자면 jne는 ZF(zero flag)가 0인 경우 실행되고 test는 두 연산자에 대한 비트 연산 and를 수행한다. And는 두 연산자가 0일 경우 0, 1일 경우 1을 return하므로 eax에 들어있는 값이 0인지 판단하기 위한 명령어로 볼 수 있다. 결과적으로 두 문자열이 다르면 eax 레지스터에 0이 들어가고, 0 & 0 = 0, ZF = 0 이 되어 jne 명령어로 폭탄이 터진다.

그럼 문자열을 같게 해서 eax에 1이 들어가도록 해야 한다.



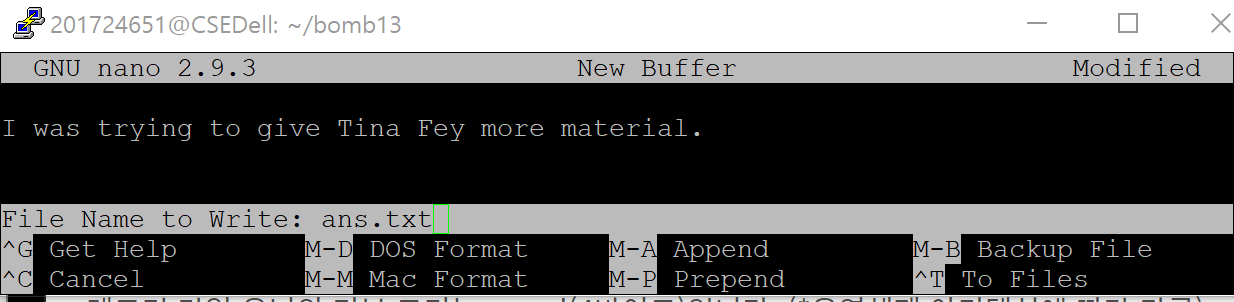
그러면 strings\_not\_equal 함수로 인해 비교될 string이 뭔지를 알아보자.

여러가지 방법이 있는데, strings\_not\_equal 함수 주소로 위치를 이동하여 x/s $rsi를 통해 알아볼 수도 있지만 +4의 명령어에서 %rsi (0x555555556b00) 위치에 string을 저장하므로 바로 해당 주소의 x/s 명령어로 해당 주소의 string을 출력해볼 수 있다.



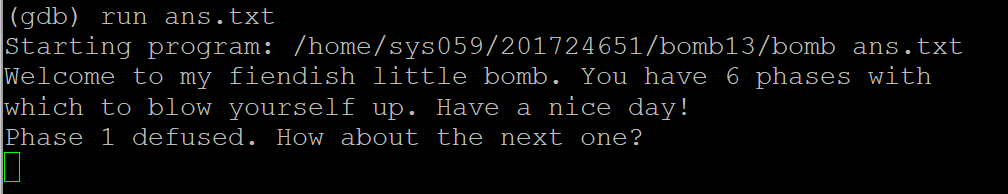
그럼 phase\_1을 해결했다. 그럼 이 답을 txt파일로 만들어두자.



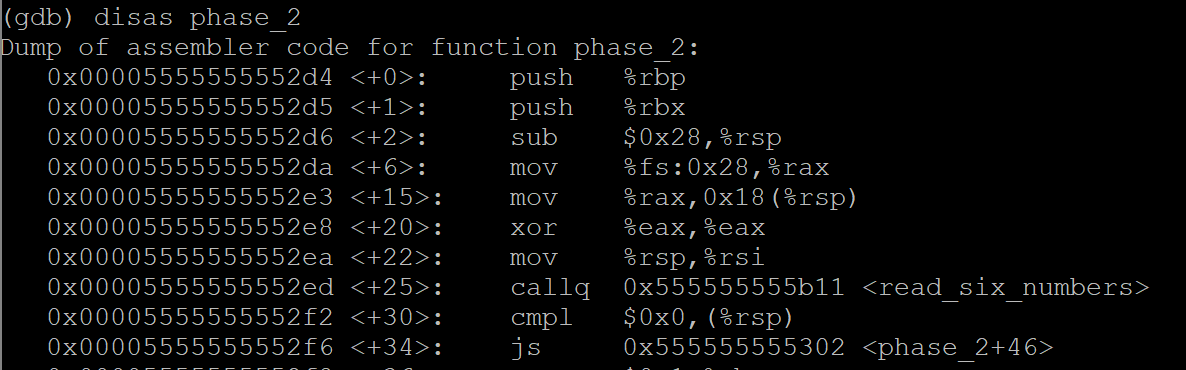




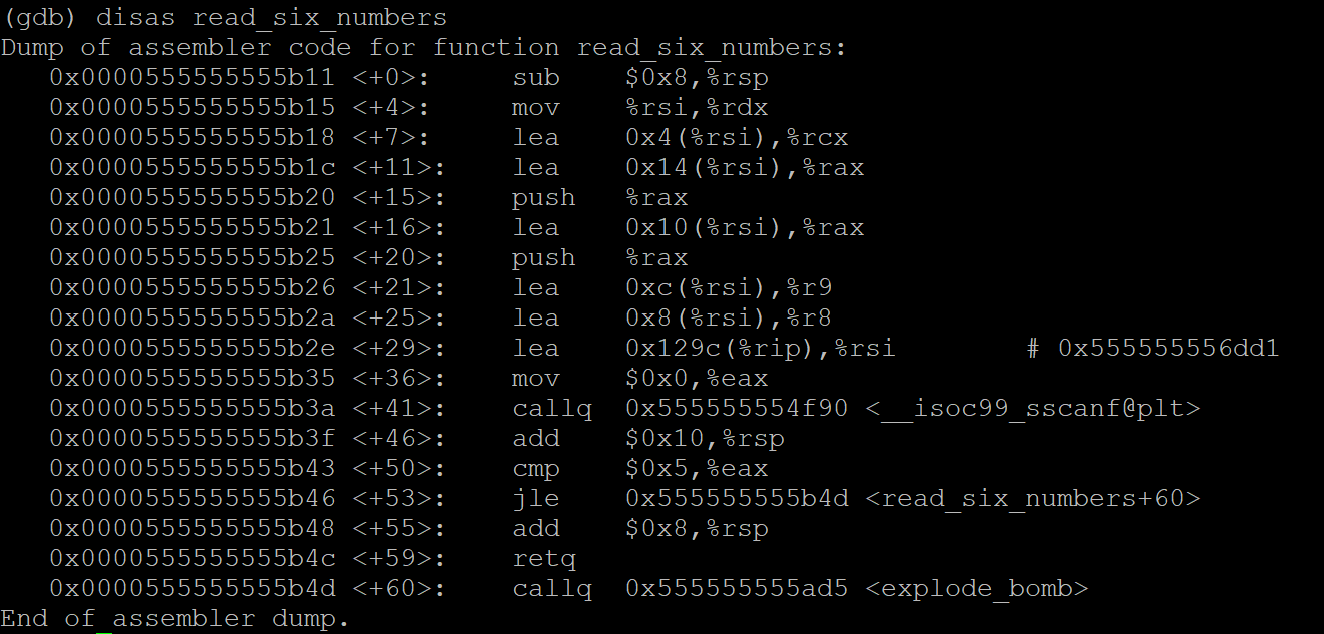
**그러면 run ans.txt 명령어로 자동으로 1단계를 클리어할 수 있다.**



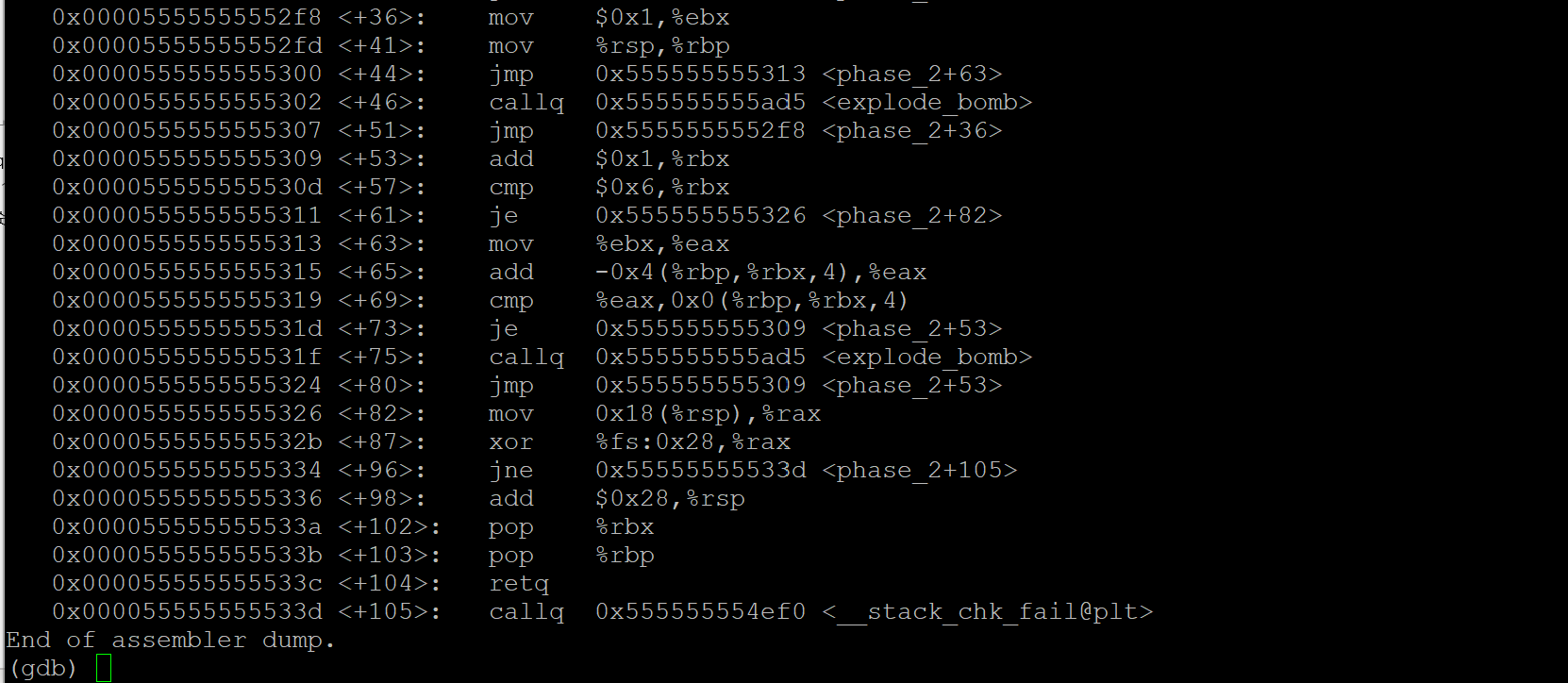
disas phase\_2 명령어로 phase\_2를 살펴보자. 대충 봐서 read\_six\_numbers라는 함수명이 있는 걸로 봐서 6개의 숫자를 입력해야 함을 짐작할 수 있다.



disas read\_six\_numbers로 함수를 살펴보면 cmp $0x5, %eax에서 5보다 큰지를 비교한다.

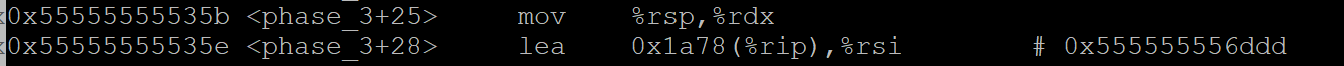
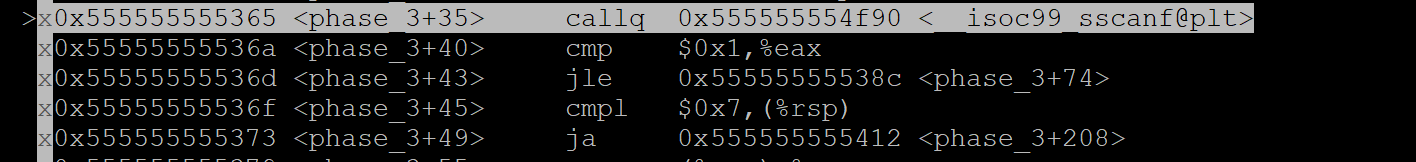


2단계에서는 1 2 3 4 5 6을 입력해봤는데 +50번째줄을 보면 cmp $0x5, $eax와 jle(jump if less or equal)를 통해 6개의 숫자를 입력했는지를 확인하는 것을 볼 수 있고, read\_six\_numbers 뒤에 rsp에 저장된 값을 봤더니 1이 저장되어 있는 걸 봐선 내가 처음에 입력한 1을 가지고 비교를 하는 듯했다. 그럼 cmpl $0x0, (%rsp)를 통해 처음 숫자가 0이어야함을 알 수 있다.

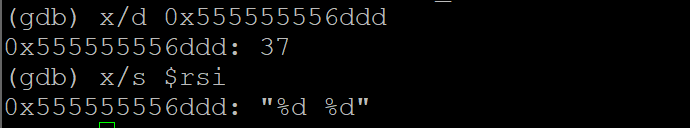


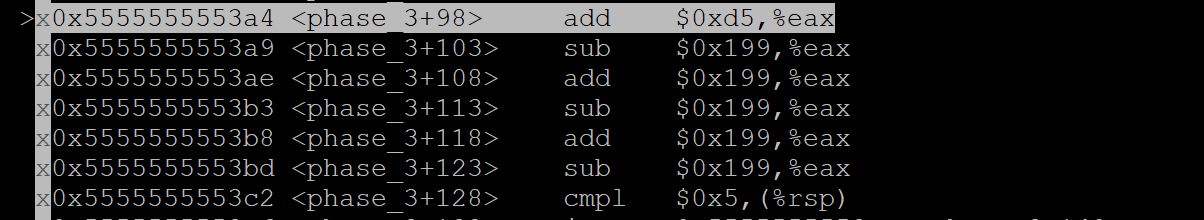
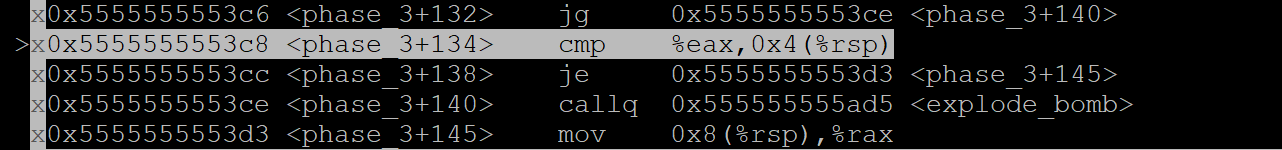
위의 사진은 disas\_phase\_2 화면의 나머지 내용들인데 이것들은 ebx를 처음에 1이었다가 1씩 증가시키는데 6이랑 비교를 한다. 그 것으로 이것은 6일 때까지 반복하는 loop문임을 알 수 있다. 그리고 +65번째 라인에서 현재의 ebx를 로 정의해주고 메모리 상의 4byte 즉 그 직전의 숫자와 더해서 해당 숫자를 결정한다. 따라서 첫번째 숫자가 0인걸 아니까 ebx가 1이니까 2번째 숫자는 2, ebx가 1씩 증가하며 현재 숫자와 더해서 다음 숫자를 결정한다. **그 결과 phase\_2의 답은 0 1 3 6 10 15 임을 알 수 있었다.**

Phase\_3을 보자. 해당 라인은 첫 번째 숫자가 2~7 사이의 숫자라는 것을 알 수 있다. 그래서 2 650 같은 숫자를 입력해보았는데 폭탄이 터졌다.

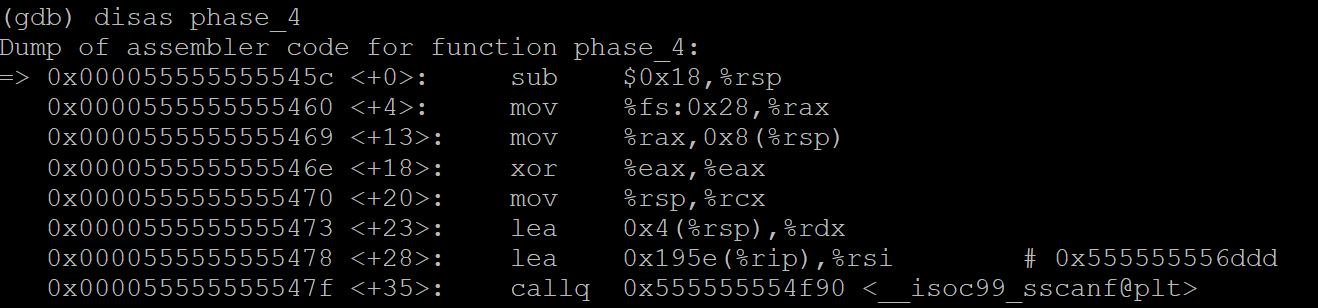
 

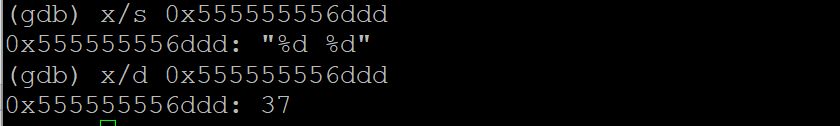
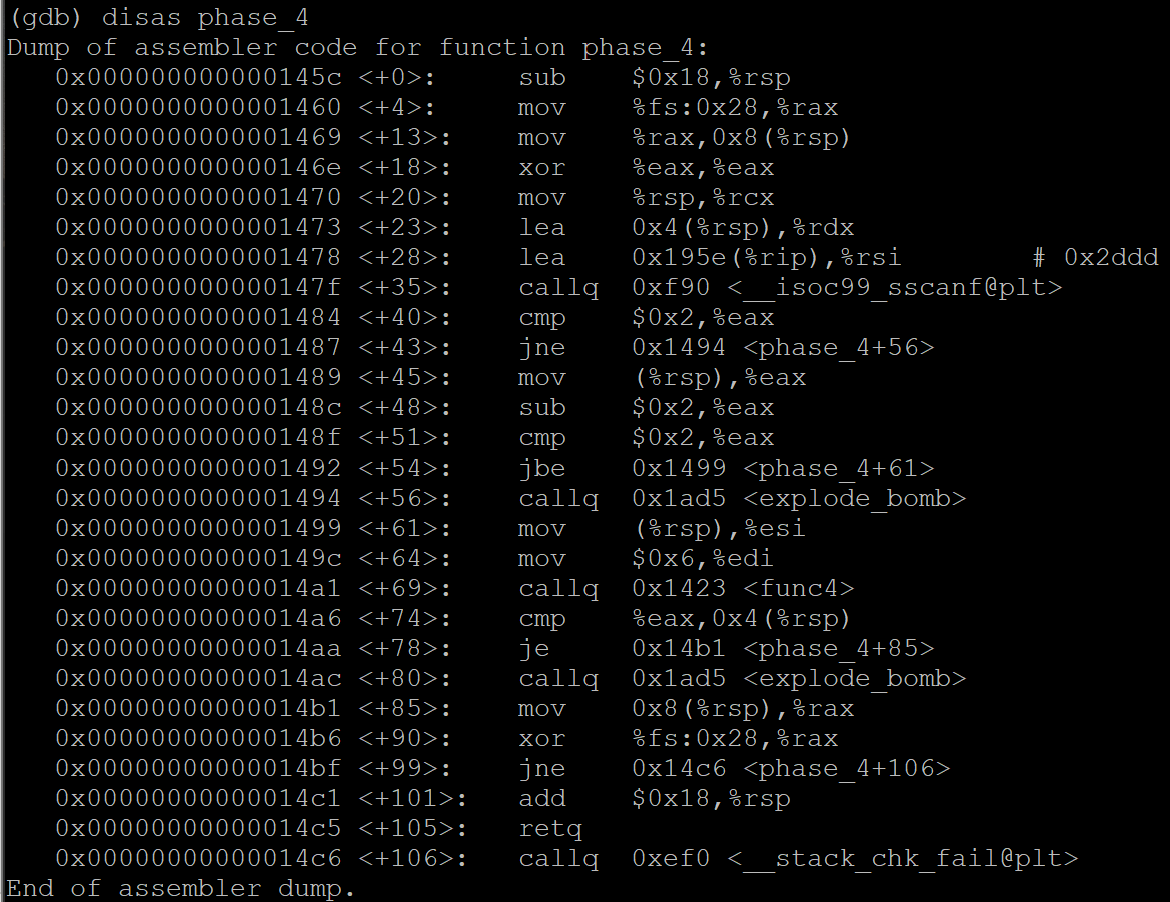
+35라인 바로 전인 +28 라인에서 아래의 명령어를 입력해보니 %d %d가 나오는걸로 보아 2개의 정수를 입력하기를 바라는 것 같았다.



각자 condition(조건)에 따라 내가 첫번째 숫자로 2~7 사이의 숫자 중 어떤 것을 고르냐에 따라 두 번째 숫자도 다르게 결정이 된다는 문제이다. 2를 골라 조건에 따라 내려가다보면 +98번 라인부터 한칸씩 내려가며 더하기 빼기를 반복하며 eax에 저장한다. 이때 0xd5 = 213, 0x199 = 409이다. 즉, 213-409+409-409+409-409이므로 -196이다. 5보다 작으므로 rsp주소보다 한 칸 뒤 즉, 내가 두번째 숫자로 고른 숫자와 eax를 비교해 같다면 폭탄이 터지지 않고 다음 단계로 넘어간다. **즉, phase\_3의 답은 2 -196이다.**



Phase\_4번을 위해서 diasa phase\_4 명령어를 실행해보면 +35번 줄의 callq로 인한 함수 호출 이전의 주소에 저장된 정보를 확인해보자. %d %d인걸로 봐서 이번에도 2개의 정수를 요구한다. 그리고 2보다 작거나 같으면 폭탄이 터지므로 첫 번째 숫자는 2보다 커야한다. 그래서 3 1을 넣어봤다. 그런데 +54번 줄에서 비교 후 jbe 명령어에 만족하지 못하고 점프를 못해 폭탄이 터지고 말았다. 내가 두번째로 입력한 1에서 sub 2를 하면 -1이라는 값이 eax에 저장이 되고 그것이 2와 비교했을 때 jbe(jump below or equal)을 통해 작거나 같을 때 점프한다. 그래서 -2를 했을 때 2와 같도록 두번째 숫자를 4로 설정해보았다.