**시스템소프트웨어(CB2400108-059) - HW2:Bomblab**

정보컴퓨터공학부 202255513 김대욱

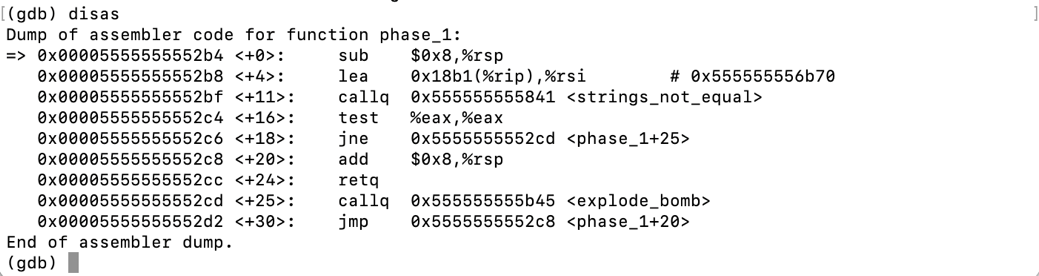
폭탄이 터지는 것을 막기 위해 *gdb bomb* 명령어를 입력한 후, 아래와 같이 breakpoint를 걸어주어야 한다. 문제 해결을 위한 일련의 과정을 거쳐 정답을 알아낸 후, *continue* 명령어를 입력하여 breakpoint를 넘어가면 된다.



**[ phase\_1 ]**

phase\_1 에 breakpoint가 걸렸을 때, disas 명령어를 사용하여, assembly 코드를 확인할 수 있다. # 0x555555556b70에 phase\_1의 정답이 들어있다.

정답 비교 함수는 <strings\_not\_equal> 이다.



Table

Description automatically generated

Text, letter

Description automatically generated

**\* phase\_1 정답 : You can Russia from land here in Alaska.**

**[ phase\_2 ]**

phase\_2의 어셈블리 코드를 확인해보자.

A picture containing text, font, receipt, white

Description automatically generated

위 코드에서 <read\_six\_numbers>를 통해 6개의 숫자를 입력해야 함을 알 수 있다. <+25>에서 6개의 숫자를 입력받아 stack에 저장하고, <+30>에서 첫 번째 입력 값이 1인지 확인한다. %rsp = $0x1 이 아니라면, <+34>에서 <+45>로 이동하여, <explode\_bomb> 함수를 호출하게 되어 폭탄이 터진다.



<+36>에서 %rbx에 %rsp값($0x1)을 넣어두고, <+39>에서 %rbx + 0x14 인 값을 %rbp에 넣는다. 이후, 코드에서 <+61>으로 이동하게 된다.

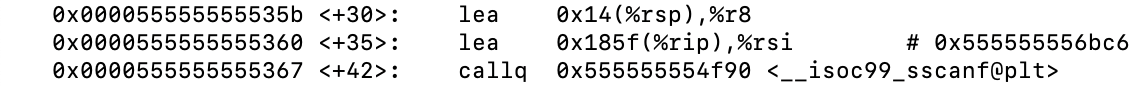
A picture containing text, font, screenshot, receipt

Description automatically generated

<+61>에서 (%rbx)의 값을 %eax에 넣은 후, <+63>의 코드를 보면, %eax = %eax + %eax이다. 해당 코드 부분에서 값을 2배하게 된다. int 자료형의 값이므로, %eax의 값과, (%rbx) + 0x4 위치의 값을 비교하여 값이 같다면, <+52>로 이동하게 되는 반복문의 형태를 띄게 된다.

**\* phase\_2 정답 : 1 2 4 8 16 32**

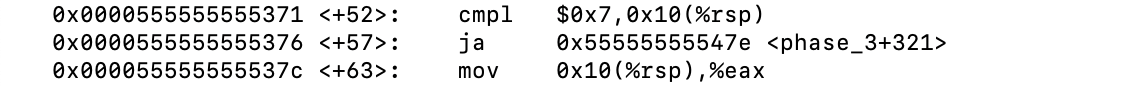
**[ phase\_3 ]**



<+42>에서 scanf 함수가 있는 것으로 보아, 입력값을 받아 처리하는 것을 알 수 있다. 입력 값의 형태를 알기 위해서는, %rsi로 전달되는 0x555555556bc6의 내용을 통해 예측할 수 있다. x/s 명령어를 통해 확인해보면 아래와 같다.



두 개의 숫자와 하나의 문자를 입력값으로 필요하다는 것을 알게 되었다.



<+52>에서 0x7보다 값이 큰 값이 들어온다면, <+321> explode\_bomb으로 이동하여 폭탄이 터지게 된다. 따라서 첫 번째 값은 0~7 사이의 값이 들어와야한다.

A picture containing text, font, screenshot, algebra

Description automatically generated

두 번째 입력값은 %eax로 0x63 값이 저장된다. 아스키코드로, 소문자 'c' 이고, 세 번째 입력값은 0x297로 숫자 '663'이다.

**\* phase\_3 정답 : 0 c 663**

**[ phase\_4 ]**



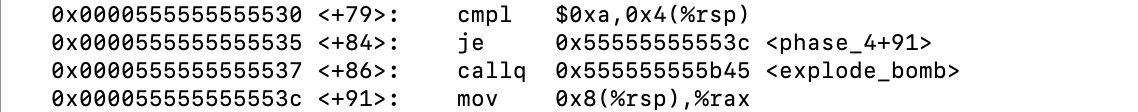
입력 값의 형태를 알기 위해서는, %rsi로 전달되는 0x555555556e8d의 내용을 통해 예측할 수 있다. x/s 명령어를 통해 확인해보면 아래와 같다.

****

첫 번째 입력값이 0xe보다 작거나 같을 경우 <+56>으로 이동하여 폭탄이 터지지 않고 이동하게 된다.



<+79>에서 두 번째 입력값이 0xa가 아닐 경우, 폭탄이 터지므로 두 번째 입력값은 0xa 즉 10이 된다.



func4 에서 나온 결과값이 0xa와 같아야 하므로, %edx = 14, %esi = 0, %edi = 첫 번째 입력값 임을 이용하여, func4의 결과값이 10일때의 첫 번째 입력값을 구하면 된다.

A black text on a white background

Description automatically generated with low confidence

func4 내부 함수에서 <+24>에서 %eax가 10이 될때, 해당 phase를 해결할 수 있으므로, 적절한 값을 찾으면 %edi는 3이 되는 것을 알 수 있다.

A screenshot of a computer code

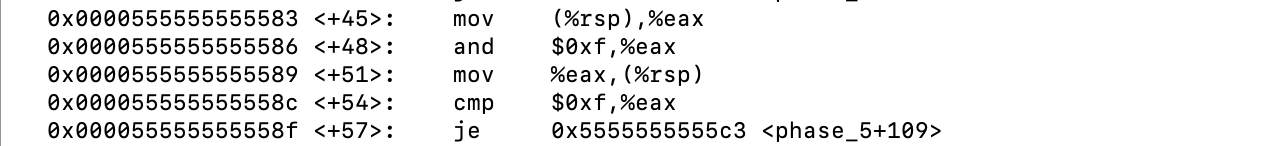
Description automatically generated with low confidence

**\* phase\_4 정답 : 3 10**

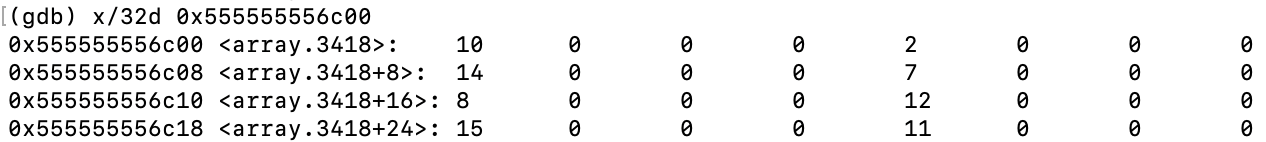
**[ phase\_5 ]**

앞서 풀었던 문제들과 마찬가지로, 입력값의 형태를 파악하면 정수 숫자 2개를 받고 있음을 알 수 있다.





위 코드에서, 첫 번째 입력 값이 0xf인 경우, <+109>로 이동하여 폭탄이 터지게 되므로 첫 번째 입력값의 마지막 bit는 0xf가 아니다. 그 뒤, %ecx와 %edx를 0으로 초기화한 후, lea 명령어를 이용하여 계산한 주소를 %rsi , 즉 array의 시작 주소로 넘겼음을 알 수 있다.



아래 코드는 반복문을 실행하는 코드로, array[%rax] 값이 %eax에 저장되며, %eax의 값이 15가 되어야 함을 알 수 있다.

A picture containing text, receipt, font, screenshot

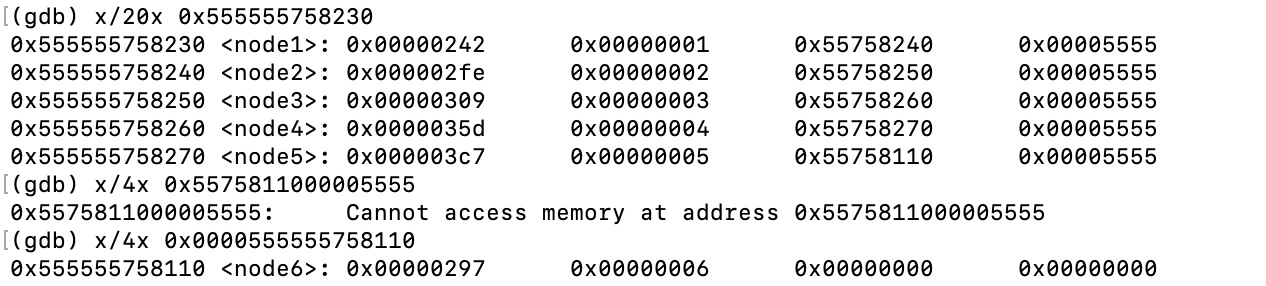
Description automatically generated

Array[6]에 15값이 저장되어 있으므로, %rax는 6이라고 가정을 한 뒤, 첫 번째 값에 집어넣으면 다음 명령어에서 %edx의 값이 15가 맞는지 확인하는 과정을 거치게 된다. %edx의 값은 처음 0으로 초기화된 이후 <+76>으로 돌아가는 과정이 있으므로, %edx와 %rax 값 모두 15를 만족시키는 첫 번째 입력값은 1, 2, 3, 4 ... 로 바꾸어가며 실행하면, 입력값이 5임을 알 수 있다. %rbx와 %rcx를 통해 두 번째 숫자가 저장되어 있는 %rsp+4의 값이 115라는 것을 확인할 수 있으므로 정답은 아래와 같다.

**\* phase\_5 정답 : 5 115**

**[ phase\_6 ]**

<read\_six\_numbers>



node5의 오른쪽 두개의 주소값을 조합하여 검색을 해보면, node6 주소값 또한 확인할 수 있다. 6개의 node 주소 순서를 확인하면, 0x00000242(1), 0x00000297(6), 0x000002fe(2), 0x00000309(3), 0x0000035d(4), 0x000003c7(5)가 답이 된다.

**\* phase\_6 정답 : 1 6 2 3 4 5**