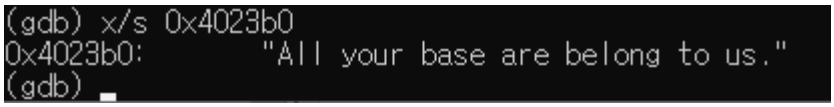
SW Lab 3 - BombLab
컴퓨터공학과
20180038 박형규

<Phase 1>
gdb bomb를 통해 gdb에 진입하고 break phase_1을 이용하여 phase_1 함수에 진입하면 break가 되게 하였다. run을 하여 함수를 실행시키고 중단된 곳부터 시스템의 진행 과정을 살펴보니 phase_1+4 에서 $0x4023b0을 %esi로 'strings_not equal' 이라는 함수에 매개변수로 넘겨주고 있었다. 이 매개변수를 살펴보기 위하여 x/s 0x4023b0 라는 명령을 통하여 살펴보니 아래 그림과 같은 string이 나왔다.



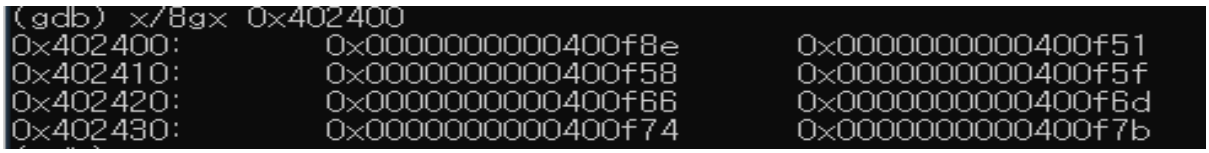
이 string과 입력 받은 string을 비교하는 것이 strings_not_equal 함수에서 실행되는 것이므로 이 string이 답이다. 따라서 phase 1의 정답은 "All your base are belong to us." 이다.

<Phase 2>
break phase_2를 이용하여 phase_2 함수에 진입하면 break가 되게 하였다. run을 하여 break 된 곳부터 시스템의 진행 과정을 살펴 보았다. phase_2+25 에서 read_six_numbers 라는 함수를 호출하는 부분을 볼 수 있는데 이 함수 안을 살펴보면 sscanf로 숫자를 받고 입력한 숫자가 6개 미만이면 폭탄을 터뜨리는 모습을 볼 수 있었다. 먼저 첫 번째 입력받은 숫자를 phase_2+30에서 0과 비교하여 phase_2+34에서 0보다 크다면 41번째 줄로 이동하는 모습을 볼 수 있다. 따라서 첫번째 입력 값은 0보다 큰 수이다. 41번째 줄로 이동하면 stack pointer의 register(%rsp)를 %rbp에 할당한다. 그리고 %ebx에 1을 저장하고 %eax에 %ebx를 할당한다. 그리고 %eax에 %rbp가 가지고 있는 값을 할당한다. 그리고 &eax와 %rbp가 가르키고 있는 다음 칸의 값을 비교하여 이것이 같다면 phase_2+64로 이동한다. 여기로 이동한 후에는 %ebx에 1을 더해주고 %rbp에 4를 더해준다. 이는 %ebx는 진행되고 있는 순서 값을 저장하는 것을 의미하고 %rbp가 가르키고 있는 포인터를 다음 칸으로 옮겨주는 작업을 의미한다. 그 후에 %ebx르 6과 비교하여 같지 않다면 다시 phase_2+49로 이동한다. 이는 이력받는 숫자의 개수가 6개임을 의미하고 이와 같은 과정을 6번 반복하는 것을 의미한다. 이 과정을 반복하게 되면 첫 번째 입력한 수부터 차례대로 1, 2, 3, 4, 5가 더해진다. 예를 들어 첫 번째 입력한 수가 1이면 6개의 값은 1, 2, 4, 7, 11, 16이 된다. 따라서 첫 번째 입력값에 따라 정답은 여러가지가 있고, 그 중 하나는 1, 2, 4, 7, 11, 16이다.

<Phase 3>
break phase_3를 이용하여 phase_3 함수에 진입하면 중단이 되도록 설정하였다. phase_3+18에서 %eax 안의 값을 0으로 설정해 주었고, phase_3+20에서 %rcx에 %rsp의 두 번째 칸의 값을 저장했고, phase_3+25에서 %rdx에 %rsp의 첫 번째 칸의 값을 저장했다.

phase_3+33에서는 서식문자열을 sscanf 함수에 인자로 전달하는 부분이다. 따라서 이 부분이 입력값이므로 x/s 0x40258f 명령어를 통하여 확인해 보았는데 아래 그림과 같이 나왔다.

따라서 입력값으로는 두 개의 정수가 입력되는 형태 일 것이라고 생각하였다. 그리고 이 두 개의 정수가 앞에서 말했던 것과 같이 %rcx와 %rdx에 저장된다. phase_3+48을 보면 7과 rsp를 비교하여 rsp의 첫번째 값이 7보다 크면 폭탄이 터지게 된다. 따라서 첫 번째 입력값으로 입력될 수 있는 값은 0부터 7까지의 정수값이다. phase_3+57에서는 0x402400+(8*rax) 만큼 더해준 주소에 저장된 주소로 점프를 하는 모습을 볼 수 있다. 이는 0x402400에 점프할 주소의 값이 담겨 있다는 것이다. 따라서 x/8gx 0x402400 의 명령어로 확인해본 결과 아래와 같은 결과를 확인 할 수 있었다.



여기 있는 8개의 주소가 각각 첫 번째 입력값인 0부터 7에 따른 점프 할 주소값들이다. 이는 각각 125, 64, 71, 78, 85, 92, 99, 106번째 줄에 해당된다. 해당되는 줄에 가보면 eax값에 특정한 값을 부여하는데 이는 각각 십진법으로 147, 425, 682, 580, 211, 277, 888, 56에 해당된다. 그 후에 130번째 줄로 이동하여 eax와 두 번째 입력값을 비교하여 같으면 폭탄이 터지지 않는 구조이다. 따라서 정답은 (0, 147), (1, 425), (2, 682), (3, 580), (4, 211), (5, 277), (6, 888), (7, 56)의 8가지 쌍이다.

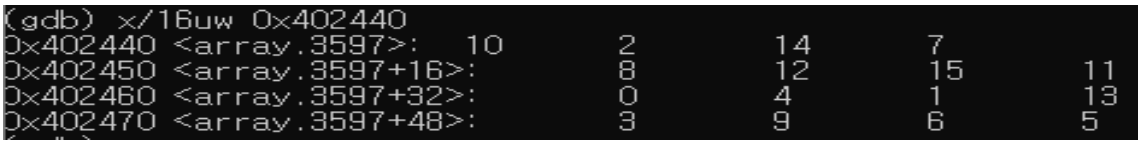
<Phase 4>

phase_4+33에서 sscanf를 이용하여 입력값을 입력받는다. 입력값의 매개변수로써 28번째 줄에서 0x40258f를 넘기고 있으므로 x/s 0x40258f 명령어를 이용하여 입력값을 확인하여보니 “%d %d” 형식이었다. 따라서 입력값이 두 개의 정수인 것을 알 수 있었다. 46번째 줄에서 %eax = %eax – 2를 하고, 49번째 줄에서 2와 %eax를 비교한다. 그 뒤 52번째 줄에서 jbe를 통하여 작거나 같을 경우에 59번째 줄로 점프하도록 한다. 이를 조합하면 두 번째 입력값이 4이하여야 한다는 것을 알 수 있다. 또한 jbe를 사용하였으므로 이는 unsigned를 기준으로 비교한다. 따라서 %eax가 음수가 된다면 이것이 16진수로 표현된 그대로 unsigned 처리되므로 MSB가 1인 16진수가 되고 이는 항상 jbe의 조건을 만족하지 못하게 된다. 이에 따라 두 번째 입력값은 2이상이어야 한다는 조건이 생긴다. 따라서 두 번째 입력값은 2, 3, 4의 세 가지 정수이다. 72번째 줄에서는 func4에서 리턴한 값과 첫 번째 입력값을 비교하는데, 이 값들이 같을 경우에는 폭탄이 터지지 않게 된다. 따라서 두 번째 입력값이 각각 2, 3, 4일 때 break를 이용하여 func4에서 리턴해오는 값이 무엇인지 확인하였는데 이는 각각 40, 60, 80 이었다. 따라서 정답은 (40, 2), (60, 3), (80, 4)의 세 쌍이다.

<Phase 5>

입력값의 형태를 살펴보기 위하여 28번째 줄에 있는 매개변수의 주소값을 x/s 0x40258f 명령어로 확인하였더니 “%d %d” 형태를 입력받는 것을 알 수 있었다. 따라서 입력값은 두 개의 정수이다. 그 뒤에 57번째 줄에서 0xf와 %eax를 cmp하여 equal이면 폭탄이 터지는 모습

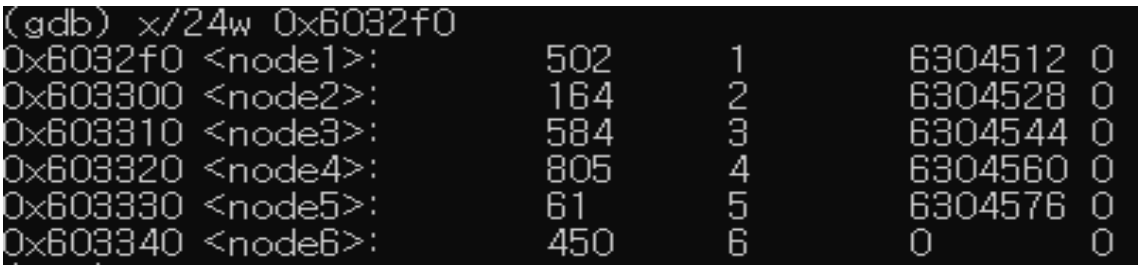
을 확인할 수 있었다. 따라서 첫 번째 입력값은 15가 되면 안된다. 그리고 89번째 줄에서는 %eax가 0xf와 같을 때까지 72번째 줄로 되돌아가 그 사이의 명령들을 반복하는 모습을 보인다. 따라서 이는 반복문의 형태를 가지고 있음을 알 수 있었다. 그 사이의 명령들을 보면 %edx에 1을 더해주고 %eax = 0x402440 + 4(%rax)의 명령을 수행하는 모습을 볼 수 있다. 0x402440에 무엇이 들어있는지를 보기 위하여 x/16uw 0x402440 명령어로 확인하였더니 아래 그림과 같은 배열이 들어있었다.



따라서 %eax = 0x402440 + 4(%rax) 왼쪽 명령어의 의미는 이전에 %rax에 저장되있던 수에 해당하는 배열 위치를 찾아가서 이를 %eax에 저장하라는 의미이다. 또한 86번째 줄에서 0xf와 %eax를 비교하여 같을 때까지 반복문을 반복하고, 98번째 줄에서 %edx와 0xf를 비교하여 같을 때만 폭탄이 터지지 않으니 반복문이 15번 반복되고, 마지막에 %eax에 저장되어 있어야 하는 수가 15여야 한다는 사실을 알 수 있다. 따라서 마지막부터 거꾸로 진행 과정을 살펴보면 15->6->14->2->1->10->0->8->4->9->13->11->7->3->12->5 이다. 따라서 첫 번째 입력값은 5이고, 103번째 줄에서 %ecx와 두 번째 입력값을 비교하여 같으면 폭탄이 터지지 않는 것을 보니 첫 번째 입력값을 제외한 0부터 15까지의 숫자에서 5를 제외하고 모두 더한 값인 115가 두 번째 입력값이 되어야 한다. 따라서 정답은 5 115이다.

<Phase 6>

29번째 줄에 보면 <read_six_numbers>를 호출하여 사용하는 것을 볼 수 있다. 따라서 phase6의 입력값은 6개의 일련된 정수라는 것을 알 수 있다. 그리고 34번째 줄에서 입력값을 %r12에 저장한 뒤 46번째 줄부터 53번째 줄까지를 살펴 보면 입력값이 6이하여야 한다는 것을 알 수 있다. 그리고 %r13d가 6이 될 때 까지 이 과정을 반복하게 되는데 반복하는 과정 속에서 73번째 줄부터 98번째 줄이 또 반복된다. 이 반복문에서는 처음부터 끝까지 반복하며 숫자가 중복되는지 확인하는 역할을 한다. 이 두 개의 반복문이 종료된 후에는 134번째 줄로 넘어오게 된다. 여기서 147번째 줄에 보면 %edx에 $0x6032f0이라는 것을 부여하고 있는데 이를 명령어 x/24w 0x6032f0으로 찾아봤더니 아래 그림과 같은 결과가 등장했다.



이제 139번째 줄부터 155번째 줄까지 진행이 된 후, 106번째 줄로 넘어가서 130번째 줄까지 수행되는 작업이 입력값이 1이 아닌 경우인 5번 동안 반복되고, 입력값이 1인 경우에는 117번째 줄로 넘어가게 된다. 이 과정에서는 위에 그림에서 첫 번째 열에 있는 값을 스택에 순서대로 저장하는 작업을 수행하게 된다. 그 후에는 159번째 줄로 넘어와서 마지막 검사 작업을 수행하게 되는데, 216번째 줄에서 %eax와 (%rbx)를 비교하여 jle일 경우 통과한다. 이는 순서대로 node에 존재하는 값이 작은 것부터 순서대로 들어와야 함을 의미하고 따라서 정답은

5 2 6 1 3 4 이다.