산업경영공학과
20150545 이원학

main

+78까지 실행이 되면 "Welcome to my fiendish little bomb. You have 6 phases with"와 " which to blow yourself up. Have a nice day!"를 출력한 뒤, text파일을 read_line 함수를 통해 읽는다. read_line의 결과값인 %rax는 %rdi로 옮겨져 phase_n 함수의 input으로 사용이 된다. phase_n 함수의 호출이 완료되면 phase_defused 함수가 호출이 된다. main+246에 있는 phase_6까지 모두 통과하여 phase_defused가 호출이 되면 bomb해체가 완료 된다.

phase_1

solution: "When I get angry, Mr. Bigglesworth gets upset."

phase_1은 input이 특정 문자열과 같은지 확인하여 같으면 다음 phase로, 다르면 explode_bomb을 실행한다.

+4에서 %rip+0x1681 주소에 있는 값을 %rsi로 옮긴다. 그리고 strings_not_equal 함수가 호출이 되는데, strings_not_eqaul 함수는 input인 %rdi와 %rsi가 같으면 1, 다르면 0을 반환하는 함수다. +16의 test %eax %eax는 %eax가 0이면 ZF flag가 1로 설정이 된다. ZF flag가 1이면 +18의 조건분기 명령어 jne를 통과하여 함수가 종료된다. 따라서 %rsi의 값이 phase_1이 solution이 된다. x/s $rsi를 통해 %rsi의 값이 When I get angry, Mr. Bigglesworth gets upset. 인 것을 확인할 수 있다.

strings_not_equal

+4, +7에서 %rdi와 %rsi를 각각 %rbx, %rbp에 옮긴다. 처음 string_length 함수를 호출하여 첫번째 입력 문자열의 길이를 %r12d에 저장한다. 그리고 두번째 입력 문자열이 %rbp를 %rdi로 옮기고 다시 string_length를 호출한다.

+31에서 두 반환값을 비교하여 같으면 +43으로, 다르면 1을 반환한다. (+26에서 1을 %edx에 옮긴다)

+43은 두번째 입력 문자열의 길이가 0이 아닌지 확인한 후, +50에서 첫번째 문자를 비교한다. 같으면 +55와 +59에서 다음 문자열로 이동하여 비교한다. 다음 문자열이 없을 경우 edx에 0을 저장하고 +36으로 이동하여 0을 반환한다. 다음 문자열을 비교하여 같으면 +55로 다시 이동하여 문자열이 없을 때 까지 반복한다. jne 분기를 타는 경우에 edx가 1이 되어 1을 반환한다.

string_length

%rdi를 0과 비교하여 같으면 0을 반환한다.

0이 아니면 %rdi를 %rdx로 옮긴다. 그 후 %rdx에 1을 더하며 다음 문자열로 이동한다. %eax에 %edx를 저장한 뒤, %edi를 뺀 값을 %eax에 저장한다. 따라서 %eax는 다음 문자열로 이동한 횟수가 저장이 된다. (%rdx)가 0이 아니면 +5로 돌아가 반복하고, 0이면 반환한다.

phase_2

solution: 1 2 4 8 16 32
+25 : read_six_numbers를 호출한다.
+30 : %rsp가 가리키는 값과 1을 비교하는데 x/6w %rsp로 확인해보면 rsp가 input인 것을 알 수 있다. %rsp가 1이 아니면 explode_bomb를 호출한다.
+36 : %rbx에 %rsp를 복사한다.
+39 : %rbp에 %rbx+0x14를 넣는데, %rbx가 6개의 숫자를 가지고 있으므로 %rbx+0x14는 마지막 숫자의 주소를 의미한다.

+61 : %eax에 %rbx 값을 넣고
+63 : %eax를 2배 한다.
+65 : %eax와 %rbx+0x4가 가리키는 값, 즉 두번째(다음) 숫자를 비교한다.
+68 : 비교하여 같으면 +52로 이동하고, 다르면 explode_bomb이 실행된다.

+52 : %rbx에 0x4를 더한다. 즉 다음 숫자로 이동을 한다.
+56 : 마지막을 가리키는 %rbp와 비교하여 같으면 +77로 이동하여 함수를 종료한다. 다르면 +61로 이동한다.

phase_2는 입력 받은 숫자들을 처음부터 순회하면서 (i번째 숫자*2)와 (i+1번째 숫자)가 같은지 확인하고 다르면 explode_bomb, 마지막 숫자까지 진행이 된다면 pass하는 과정이다. 첫번째 값은 1이 되어야 하므로 나머지 5개의 숫자는 1, 2, 4, 8, 16, 32가 된다.

read_six_numbers
%rsi, %rsi+4, %rsi+8, %rsi+12, %rsi+16, %rsi+20를 각각의 레지스터에 저장하는 것을 보아 %rsi가 size 6의 array 형태임을 알 수 있다. +50에서 sscanf의 결과값이 5보다 작거나 같으면 explode_bomb을 실행한다. 또한 +29에서 0x119c(%rip)를 x/s로 확인해보면 %d %d %d %d %d %d가 나오는 것을 보아 6개의 정수를 입력 받는 것을 알 수 있다.

+30에서 rsp와 1을 비교 다르면 bomb
+36 rsp를 rbx에 복사 (rsp는 6개의 숫자를 가지는 stack)
+39 %rbp = %rbx+0x14 -> rbx가 6개의 숫자를 가지는 stack이므로 rbx+20은 마지막 숫자를 가리킴
+43 +61로 jump
+61 eax에 rbx의 값(첫번째 값)이 가리키는 값 넣음
+63 eax = 1 +1 = 2
+65 eax(2) 와 rbx+4(두번째 값) 비교 -> 두번째 값은 첫번째 값 2배여야 함
+52로 jump

+52 rbx주소에 4 더함 -> 두번째 값

+56 두번째랑 마지막 값 비교(주소를 비교함)
+59 rbx가 6번째(마지막)이면 +77로 점프
아니면 다시 +52로 jump
rbp값과 같아질 때 까지 4를 더함 (5번째와 6번째를 비교할 때 까지)

phase 3

solution: 3 a 798
+35: 0x162f(%rip)가 가리키는 값은 x/s를 통해 확인하면 %d %c %d이다. 2개의 정수와 1개의 문자를 입력 받는다.
+47 input의 개수가 2이하이면 explode_bomb을 호출한다.
+52 rsp+0x10를 x/d로 확인하면 첫 번째 input인 것을 알 수 있다. 7과 rsp+0x10(첫번째 input)을 비교하여 7보다 더 크면 explode_bomb를 호출한다. 즉, 첫번째 input은 7보다 작아야 한다.
+63 %eax에 첫번째 input을 저장한다.
+74에서 %rax(첫번째 input)을 이용하여 %rax를 계산하고 그 주소로 jump한다.
3을 입력할 경우 +192로 jump한다
+192: $eax에 0x62저장
+197: 0x14(%rsp)는 3번째 input을 가리킨다. 3번째 input과 0x31e(=798)와 비교하여 다르면 explode_bomb을 호출한다. 같으면 +341로 이동한다. 3번째 input은 798이 되어야 한다.
+341: %al 값을 확인하면 98이 나온다. %al을 2번째 input과 비교하여 다르면 explode_bomb을 호출한다. 2번째 input은 문자이기 때문에 ascii코드 값으로 98을 가지는 값, 즉 a가 solution이 된다.

phase 4

solution: 264 3 DrEvil // DrEvil은 secret_phase를 위한 값
+28: x/s $rip+0x160c 결과가 %d %d이다. 2개의 정수를 입력 받는 것을 알 수 있다.
+40: input의 개수가 2개가 아니면 explode_bomb를 호출한다.
+45 %rsp가 가리키는 값을 확인하면 두번째 input인 것을 확인할 수 있다. %eax에 두번째 input을 저장한다.
+48 %eax에 2를 뺀다.
+51 %eax를 2와 비교해서 %eax가 2보다 작거나 같으면 +61로 이동한다. 즉 두번째 input은 4이하의 정수가 되어야 한다.

+61 %esi에 두번째 input을 저장
+64 %edi에 9를 저장한다.
+69 func4를 호출한다.
+74 func4의 반환값을 $rsp+0x4, 즉 첫번째 input과 비교하여 같으면 함수를 종료한다.

func4

+0 %eax에 0을 저장하여 기본 반환값을 0을 한다.

+5 %edi값이 0이면 0을 함수를 종료하고 0을 반환한다.

+11 %edi가 1이면 함수를 종료하고 3을 반환한다.

+22 2번째 input인 %esi를 %r12d에 저장한다.

+25 %ebx에 $edi를 저장한다.

+27 %edi에 1을 뺀다.

+30 func4를 다시 호출한다. 기존 func4의 파라미터인 $edi보다 1 작은 값을 파라미터로 입력 받는다.

+35 +30에서 호출한 func4의 반환값인 %rax에 %r12(두번째 input)을 더한 값을 %ebp에 저장한다. %ebp = func4 + 두번째 input과 같은 형태가 된다.

+39 %edi에 %rbx(%edi와 같은 값)에 2를 뺀 값을 저장한다.

+45 다시 func4를 호출한다. 기존 func4의 파라미터인 $edi보다 2 작은 값을 파라미터로 입력 받는다.

+50 %ebp에서 +45에서 호출한 func4의 반환값을 더한다. +35에서 %ebp는 +30에서 호출된 func4의 반환값 +3이기 때문에 func4(n)의 최종 반환값은 func4(n-1) + 3 + func4(n-2)가 된다.

func4(n)

phase_4의 solution은 첫번째 input로 func4(9)의 반환값을 받는다. func4(n)은 아래와 같은 점화식을 가진다.

f(n) = f(n-1) + f(n-2) + d

f(0) = 0, f(1) = d

phase_4의 solution의 두번째 input은 위의 식에서 d가 된다. 따라서 두번째 input으로 3, 첫번째 input으로 264를 입력한다.

phase 5

solution: aaaadf

+1 %rbx에 %rdi를 입력한다. %rbx에 사용자 input이 들어가게 된다.

+4 string_length를 호출하여 +9에서 반환값이 6이 아니면 explode_bomb을 호출한다.

+14 %rbx를 %rax에 저장한다.

+17 : %rdi에 %rbx에 0x6을 더한 값을 저장한다. 따라서 %rdi는 input 문자열 바로 전 주소를 가리킨다.

+21 %ecx에 0 저장

+26 : rsi에 2를 저장

+33 %rax를 zero byte extended 시킨다.

+36 0xf와 and operation결과를 %edx에 저장. %edx가 마지막 4bit만 남는다.

+39 : %rsi의 주소에 $rdx *4를 더해서 %ecx에 저장한다. %rsi를 x/32d를 이용해 확인하면 2 10 6 1 12 16 9 3 4와 같은 array형태의 값을 가지고 있다. %ecx에는 %rdx의 값을 index로 하는 값이 더해진다. %ecx += %rsi[%rdx]와 같은 형태이다.

+42 %rax에 1을 더해 다음 문자를 가리키게 한다.
+46 %rdi와 %rax비교하여 다르면 +33부터 반복한다. %rdi가 마지막 문자의 다음을 가리키고 있기 때문에 %rdi와 %rax가 같다는 것은 문자열을 전체 순회했다는 의미이다.
+51 문자열 순회가 끝난 뒤, %ecx 값을 0x3d(=61)과 비교하여 같으면 함수를 종료한다.

입력만 문자열의 각 문자가 가지는 값의 마지막 4bit를 구한다. 문자는 ascii코드이기 때문에 ascii 코드를 2진법으로 바꾸어 마지막 4bit를 확인하면 된다. 구한 4bit를 k라고 했을 때, %rsi+4k가 가리키는 값을 모든 문자에 대하여 더한다. 더한 값이 0x3d(=61)과 같은 문자열이 solution이 된다. x/32d $rsi를 하면 2, 10, 6, 1, 12, 16, 9, 3, 4 순으로 값을 확인할 수 있다. 61 = 10 * 4 + 12 + 9이므로 마지막 4bit가 1, 4, 6이 되는 문자를 구한다. a의 ascii 코드는 01100001, d의 ascii 코드는 01100100, f의 ascii 코드는 01100110이다. 따라서 aaaadf가 solution이 된다.

phase_6

solution: 1 4 6 2 3 5

+28 %r13에 %rsp을 저장한다.
+31 %rsi에 $r13을 저장한다.
+34 read_six_numbers 함수를 호출한다.
+39 %r12에 %r13을 저장한다.
+42 %r14d에 0을 저장한다.
+46 +87로 이동한다.

+83 %r13에 0x4를 더한다. %r13은 다음 숫자를 가리키게 된다.
+87 %rbp에 %r13을 저장한다. x/6dw로 %r13을 확인하면 6개의 input인 것을 확인할 수 있다.
+90 %eax에 %r13+0가 가리키는 값을 저장한다.
+94 %eax에 1을 뺀다.
+97 %eax와 5를 비교하여 5보다 더 크면 explode_bomb을 호출한다.
+102 %r14d에 1을 더한다.
+106 %r14d가 6이면 +117로 이동한다.
+112 %r14d를 %ebx에 저장한다
+65 %rax에 %ebx를 sign extension 하여 저장한다.
+68 %eax에 %rsp+4*%rax가 가리키는 값을 저장한다. %eax는 다음 숫자를 가지게 된다.
+71 %eax를 %rbp와 비교한다. %eax는 %rbp 다음 숫자를 가지고 있으므로, i번째와 i+1번째 숫자를 비교하는 것과 같다.
+74 %eax와 %rbp가 같으면 explode_bomb를 호출한다. 다르면 +57로 이동한다.
+57 %ebx에 1을 더한다.
+60 %ebx가 5보다 크면 +83으로 이동한다. 그렇지 않으면 +65로 가서 반복한다.

입력 받은 숫자를 차례대로 a0, a1, a2, a3, a4, a5라고 한다. 처음에 %eax와 %rbp는 a0를 가리킨다. %r14d는 +42에서 0에서 1이 된다. %eax는 %rsp+4*%rax가 가리키는 값으로 변경이 되는데, %rax는 %r14d와 같다. 따라서 %eax는 a1을 가리키게 된다. +74에서 %eax와 %rbp를 비교하는데 같으면 explode_bomb이 호출되기 때문에 a0와 a1는 같은 값을 가지면 안된다.
+57에서 %ebx에 1을 더해주는데 %ebx는 %r14d와 같은 값을 가진다. 따라서 %ebx는 2가 된다. %ebx는 아직 5보다 작기 때문에 +65로 이동한다.
%rax에 %ebx(=2)가 저장된다. %eax는 %rsp+4*2가 되기 때문에 a2를 가리키게 된다. 다시 %rbp와 %eax를 비교하여 a0와 a3가 다른지 확인한다. 이 과정이 %ebx가 6이 될 때 가지 반복되고 결과적으로 a0와 a1, a2 .. a5가 다른지 확인하게 된다. %ebx가 6이 되면 +83으로 이동한다. +83에서 $r13에 0x4가 더해진다. 그 후 +87의 과정이 반복되는데 이는 a1이 나머지 a2, a3, a4, a5와 다른지 확인하는 것과 같다. 따라서 위 과정은 6개의 input 숫자들이 서로 모두 다르면서 동시에 6 이하인지를 확인하는 과정이다. %r14d가 6번 호출 되어 위 과정이 끝나면 +117로 이동하게 된다.

+117 %rcx는 %r12+0x18이 가리키는 값이 들어간다. %r12는 +39에서 input의 시작 주소를 저장하고 있다. 따라서 %rcx는 6번째 숫자 다음 주소를 가리키게 된다.
+122 %edx = 7
+127 %eax에 %edx를 저장한다
+129 %eax에 %r12를 뺀 값을 저장한다
+133 %r12에 %eax를 저장한다
+137 %r12에 4를 더해 다음 숫자를 가리키게 한다.
+141 %r12와 %rcx를 비교, 즉 %r12가 6번째 숫자 다음을 가리키는지 확인하고 아니면 +127로 가서 과정을 반복한다. 같다면 %esi를 0으로 하고 +179로 이동한다.

위 과정은 6개의 input을 순서대로 돌아가며 7에서 해당 숫자를 뺀 값으로 바꾸어 준다. 예를 들어 1 2 3 4 5 6의 순서대로 있었다면 위 과정을 통해 6 5 4 3 2 1 로 바뀌게 된다.

+179 %ecx에 %rsp+%rsi*4가 가리키는 값을 저장한다. %rsi의 초기 값은 0이다
+182 %eax에 1을 저장한다.
+187 $rdx에 0x202c59(%rip)를 저장한다. x/d $rdx를 통해 값이 769인 것을 확인한다.
+194 %ecx가 1보다 크면 +153으로 이동한다.

+153 %rdx에 8을 더한다.
+157 %eax에 1을 더한다.
+160 %ecx와 %eax를 비교하여 같지 않으면 +153으로 이동해 반복한다. 같으면 +164로 이동한다.

+164 %rdx를 %rsp+%rsi*8 +0x20에 저장한다. %rsp+0x20은 6개의 input 바로 다음 주소이다. 따라서 953을 가리키는 주소는 6번째 input 다음으로 부터 %rsi만큼 떨어진 곳에 위치하게 된다.

+169 %rsi에 1을 더한다.
+173 %rsi와 6을 비교하여 같으면 +201로 이동한다. 그렇지 않으면 +179로 이동하며 반복한다

처음에 %eax는 1이고, $rdx는 953을 가리킨다. %ecx는 첫번째 input을 저장하고 있는데 %ecx가 1보다 크면 +153으로 이동하여 %eax와 %ecx가 같을 때까지 %rdx에 8을 더한다. %eax와 %ecx가 같으면 %rdx를 6번째 input 다음으로 부터 %rsi만큼 떨어진 곳에 저장한다. (input -1)만큼 %rdx에 8을 더하고 해당 input 숫자의 순서대로 %rdx가 저장이 된다. input이 4, 2, 1과 같은 순서로 들어오면 %rdx+24은 $rsp+0x20에 %rdx+16은 %rsp+0x28에, %rdx+24는 %rsp+0x30에 저장이 된다.

x/6d %rdx를 확인해보면 rdx가 769, 377, 841, 509, 659, 933 순으로 있다는 것을 확인할 수 있다. 이 경우 input의 순서가 3, 1, 2...와 같은 순서라면 6번째 input 뒤에 841, 769, 377 순으로 주소가 저장이 된다.

+201 에서 +251의 과정을 거치면 아래와 같이 된다. +259에서 %ebp에 5를 저장한다.

0
rax
rdx
rbx
7-a5
7-a4
7-a3
7-a2
7-a1
7-a0

+275 %rax에 %rbx+8이 가리키는 값을 저장한다.
+279 %eax에 %rax가 가리키는 값을 저장한다.
+281 %eax와 %rbx를 비교하여 %rbx가 %eax보다 작으면 explode_bomb를 호출한다.
+266 %rbx에 8을 더한다.
+270 %ebp에 1을 빼고 %ebp가 0이 되면 함수를 종료한다.

위 과정은 아래서부터 내림차순으로 정렬되어 있는지 확인하는 과정이다. 따라서 아래와 같은 순으로 저장이 되어야 한다.

0

377
509
659
769
841
933
b5= 7-a5
b4 = 7-a4
b3 = 7-a3
b2 = 7-a2
b1 = 7-a1
b0 = 7-a0

rdx가 769, 377, 841, 509, 659, 933 있으므로 6, 3, 1, 4, 3, 2의 순이 되어야 한다. 하지만 이 순서는 처음 7 - input을 기준으로 했기 때문에 최종 input은 1, 4, 6, 2, 3, 5가 되어야 한다.

secret_phase

solution: 7
phase_defused
+20 : 현재 스테이지가 6이면 +50으로 이동한다.
+65 %rsi에 x/s %rip + 0x1049를 저장한다.
+72 %rdi에 x/s %rip + 0x202d19를 저장한다.
%rsi는 "%d %d %s"를 가리키고 있으며 %rdi는 phase_4의 solution을 가리키고 있다.
+84 파라미터가 3개면 +103으로 이동한다

%rsp+10과 %rip+0x1027를 비교하여 같으면 +143까지 진행하여 secret_phase를 진행한다.
%rsp+10는 solution 4의 3번째 input을 가리키고 있으며 %rsp+0x1027는 "DrEvil"이라는 문자열을 가리키고 있다. 따라서 solution 4에 DrEvil을 추가한다.

secret_phase
+6 %edx에 10을 저장한다.
+11 %esi에 0을 저장한다.
+16 read_line의 결과를 %rdi에 저장한다.
+19 strtol 함수를 호출 하여 문자열을 정수로 분리한다
+24 strtol 함수의 반환값을 %rbx에 저장한다.
+27 rax에서 1을 빼 eax에 저장한다. %rax는 첫번째 정수를 가리킨다.
+30 eax가 0x3e8(1000)보다 크면 explode_bomb를 호출한다.
+37 esi에 ebx(첫번째 input)을 저장한다.
+39 rdi에 0x202a88(%rip)의 값을 저장한다.

+46 fun7을 호출한다.
+51 fun7의 반환값이 4가 아니면 explode_bomb을 호출한다. 4이면 함수를 종료한다.

func7
+0 %rdi가 0이면 0xffffffff을 반환한다.
+9 %rdi를 %edx에 복사
+11 %esi와 %edx 비교한다. %esi는 input이다.
+13 %edx가 더 크면 +29로 이동한다
+15 %eax에 0을 저장한다.
+20 %esi와 %edx를 비교하여 다르면 +42로 이동한다.
+28 반환한다. 0을 반환하게 된다.

+29 %rdi에 8을 더한다.
+33 fun7을 호출한다.
+38 +33의 반환값의 2배를 %eax에 저장하고 반환한다.

+42 %rdi에 16을 더한다.
+46 fun7을 호출한다
+51 +46의 반환값의 2배에 1을 더하여 %eax에 저장하고 반환한다.

pseudo code로 표현하면 아래와 같이 된다.

```
func(%rsi){
   if(%esi < %edx){
      %rdi = %rdi + 8
      return 2*fun7(%rsi)
   }
   else if(%esi > %edx){
      %rdi = %rdi + 16
      return 2*fun7(%rsi) +1
   }
   else{
      return 0;
   }
}
```

%rdi의 값을 확인하면 아래와 같다.

```
0x555555758030 <n41>:    0x0000000000000001      0x0000000000000000
0x555555758040 <n41+16>:         0x0000000000000000      0x0000000000000000
0x555555758050 <n47>:    0x0000000000000063      0x0000000000000000
0x555555758060 <n47+16>:         0x0000000000000000      0x0000000000000000
0x555555758070 <n44>:    0x0000000000000023      0x0000000000000000
0x555555758080 <n44+16>:         0x0000000000000000      0x0000000000000000
0x555555758090 <n42>:    0x0000000000000007      0x0000000000000000
0x5555557580a0 <n42+16>:         0x0000000000000000      0x0000000000000000
0x5555557580b0 <n43>:    0x0000000000000014      0x0000000000000000
0x5555557580c0 <n43+16>:         0x0000000000000000      0x0000000000000000
0x5555557580d0 <n46>:    0x000000000000002f      0x0000000000000000
0x5555557580e0 <n46+16>:         0x0000000000000000      0x0000000000000000
0x5555557580f0 <n48>:    0x00000000000003e9      0x0000000000000000
0x555555758100 <n48+16>:         0x0000000000000000      0x0000000000000000
0x555555758110 <node6>: 0x00000006000003a5      0x0000555555758230
0x555555758120 <bomb_id>:        0x0000000000000221      0x0000000000000000
0x555555758130 <n1>:     0x0000000000000024      0x0000555555758150
0x555555758140 <n1+16>: 0x0000555555758170      0x0000000000000000
0x555555758150 <n21>:    0x0000000000000008      0x00005555557581d0
0x555555758160 <n21+16>:         0x0000555555758190      0x0000000000000000
0x555555758170 <n22>:    0x0000000000000032      0x00005555557581b0
0x555555758180 <n22+16>:         0x00005555557581f0      0x0000000000000000
0x555555758190 <n32>:    0x0000000000000016      0x00005555557580b0
0x5555557581a0 <n32+16>:         0x0000555555758070      0x0000000000000000
0x5555557581b0 <n33>:    0x000000000000002d      0x0000555555758010
0x5555557581c0 <n33+16>:         0x00005555557580d0      0x0000000000000000
0x5555557581d0 <n31>:    0x0000000000000006      0x0000555555758030
0x5555557581e0 <n31+16>:         0x0000555555758090      0x0000000000000000
0x5555557581f0 <n34>:    0x000000000000006b      0x0000555555758050
0x555555758200 <n34+16>:         0x00005555557580f0      0x0000000000000000
```

처음 %rdi는 24의 값을 가진다. input이 24보다 작을 경우 현재 주소에서 8만큼 더한 주소의 값인 0x0000555555758150을 가지게 된다. 24보다 클 경우 0x0000555555758170을 가지게 된다.
해당 주소를 따라가면 0x0000555555758150에는 8이 있고 0x0000555555758170에는 32가 있다. 0x0000555555758150와 0x0000555555758170에 각각 +8, +16을 하면 또 다른 주소를 확인할 수 있다. 이는 아래와 같이 표현된다.

8 - 24 - 32

6 - 8 - 16

2d - 32 - 6b

1 - 6 - 7

14 - 16 - 23

28 - 2d - 2f

63 - 6b - 3e9

secret_phase에서 fun7의 반환값이 4가 되어야 한다. 4는 (((0*2+1)*2)*2)로 표현할 수 있다. 따라서 2번 %rdi보다 작고, 1번 %rdi보다 크고 1번 %rdi와 같아야 한다. 따라서 7이 solution이 된다.