2020 시스템 프로그래밍
- Bomb Lab -

제출일자	2020.10.25
분 반	00
이 름	신세정
학 번	201600253

Phase 1 [결과 화면 캡처]

```
(gdb) r
Starting program: /home/sys00/a201600253/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am the mayor. I can do anything I want.

Breakpoint 1, 0x00005555555552c4 in phase_1 ()
(gdb) c
Continuing.

Breakpoint 2, 0x00005555555557ec in strings_not_equal ()
(gdb) c
Continuing.
Phase 1 defused. How about the next one?
```

Phase 1 [진행 과정 설명]

폭탄을 안전하게 해제하기 위해서 가장 먼저 phase_1함수에 breakpoint를 걸어준다(b phase_1).

그 다음 phase_1에 있는 strings_not_equal함수에도 breakpoint를 걸어준다(b stirngs_not_equal).

이 strings_not_equal함수에서 <+4>와 <+7>을 보면 $rdi = Input값, $rsi = 정답 String 임을 알 수 있다.

x/s $rsi를 하면 정답값을 알 수 있고, 그 답을 복사해 함수를 실행하면 폭탄 해제가 마무리 된다.

Phase 1 [정답]

I am the mayor. I can do anything I want.

Phase 2 [결과 화면 캡처]

Phase 2 [진행 과정 설명]

<+30>을 보면 0x0과 $rsp가 같지 않으면 폭탄이 터진다. 그러므로 첫 번째 값은 0x0이 된다.

<+36>에서 0x1과 그 다음 수(int형은 4바이트) 0x4(%rsp)가 다르면 폭탄이 터지기 때문에 0x1이 두 번째 값이 된다.

<+48>에서 첫 번째 값을 rbx에 저장하고 <+66>에서 두 번째 값을 eax에 젖아 한다.

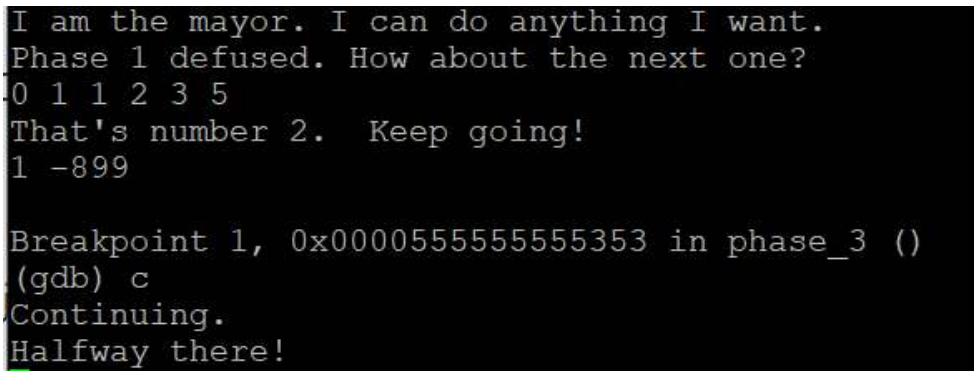
<+61>에서 첫 번째 값(rbx)와 두 번째 값 (eax)을 더한 후 이 값을 세 번째 값인 0x8(%rsp)과 비교해 다르면 폭탄이 터지기 때문에, 세 번째 값은 첫 번째 값과 두 번째 값을 더한 것임을 알 수 있다. 이를 여섯 번 반복한다.

즉 피보나치 수열이라고 할 수 있다.

Phase 2 [정답]

0 1 1 2 3 5

Phase 3 [결과 화면 캡처]



Phase 3 [진행 과정 설명]

<+28> 주소를 x/s address 해보면 2개의 정수를 입력해야함을 알 수 있다. 먼저 explode_bomb에 break를 걸고 아무 정수 2개를 입력했다(1 10 입력함). 입력이 끝나고 <+40>에 1과 eax를 비교하는 문장이 있어서 x/d를 통해 <+35> 진입 전과 <+40>진입 전의 eax 주소값을 확인했다. <+40>에서 eax에 저장된 값과 1을 비교해 작거나 같으면 폭탄이 터지고, <+35> 실행전엔 eax가 0이었다가 현재 2인 것을 확인할 수 있다. 2는 1보다 크니까 폭탄이 터지지 않는다.(jle instruction 때문) <+45>에서 rsp에 있는 값과 7을 비교한다. 바로 다음 <+49>에서 rsp값이 7보다 크면 <+74>에서 폭탄이 터지므로 첫 번째 정수는 7이하의 정수여야함을 알 수 있었다. 그러므로 <+72>에서 이동할 주소가 중요하고 rax 주소값을 확인해보니 <+88>로 점프하는 것을 확인할 수 있었다. 계산을 수행하면 계산 결과가 -899가 나오는 것을 알 수 있다. 연산을 끝내고 <+128>에서 rsp를 5와 비교했을 때 여전히 우리가 입력한 1과 비교가 되므로 jg(rsp가 5보다 크면 폭탄)는 실행되지 않는다. 그리고 eax와 0x4(%rsp)를 비교하는데 이는 두 번째 입력값을 의미한다. 둘이 같으면 폭탄이 터지지 않으므로 두 번째 입력값은 -719가 된다.

Phase 3 [정답]

1 -899

Phase 4 [결과 화면 캡처]

```
Halfway there!
4 19

Breakpoint 2, 0x0000555555555468 in phase_4 ()
=> 0x0000555555555468 <phase_4+0>:      48 83 ec 18     sub    $0x18,%rsp
(gdb)
(gdb) c
Continuing.

Breakpoint 3, 0x00005555555554b2 in phase_4 ()
=> 0x00005555555554b2 <phase_4+74>:     83 f8 13        cmp    $0x13,%eax
(gdb) c
Continuing.
So you got that one.  Try this one.
```

Phase 4 [진행 과정 설명]

<+28> rsi 주소의 x/s address를 확인하면 %d %d가 나오고, <+40>에서 eax 2와 비교했을 때 다르면 폭탄이 터지므로 입력 값은 정수 두 개이다.

<+45> rsp가 0xe보다 <+49> 작거나 같으면 <+56>으로 점프, 아니면 폭탄이 터지므로 첫 번째 입력값은 0xe이하인 unsigned int이다.

<+56>부터 func4까지 함수로 써보면

```
int func4(a, x, y){
        int num = y-x;
        num += (num < 0)? 1 : 2;
        num /= 2;
        if( num > a) {
                Return (num + func4(a, x, num-1));
        } else if( num < a) {
                Return (num + func(a, x, num+1));
        } else {
                Return num;
        }
}
```

이 함수의 리턴값이 19가 나와야한다. (<+74> 참고)

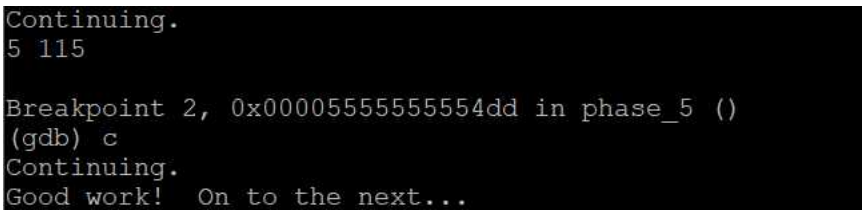
그리고 rsp의 다음 값이 0x4(%rsp)이므로 이 값이 0x13이어야만 폭탄이 터지지 않으므로 두 번째 입력값은 0x13인 19가 된다.

Func4의 함수를 거꾸로 풀면 답이 4가 나온다. 이 값은 첫 번째 입력값이 된다.

Phase 4 [정답]

4 19

Phase 5 [결과 화면 캡처]



Phase 5 [진행 과정 설명]

<+28>를 x/s address해보니 %d %d가 나왔다. 그러므로 입력해야 하는 값은 정수 두 개임을 알 수 있다.

<+45>~<+57>에서 첫 번째 인자를 입력받고 이것과 0xf를 비교해서 다르면 진행하고 같으면 폭탄이 터진다.

인자1과 인자2에 각각 1과 2를 넣고 통과해서 <+59>부터 <+86>까지 진행하면

<+59>,<+64> : ecx에 0x0을 넣고 edx에 0x0을 넣는다.

<+76> : edx에 0x1을 더한다.

<+81> : eax에 rsi,rax,4를 넣는다

<+84> : ecx에 eax를 넣는다.

<+86> : eax와 0xf를 비교해서 다르면 +<76>으로 되돌아가고 , 같으면 <+91>으로 계속 진행한다.

여기서 eax에 넣는 것이 계속 다른게 나와서 순서대로 적어보자면 1 2 e 6 f 가 나오는데 f가 나와서 <+91>로 진행된다.

<+91>rsp에 0xf를 넣고 <+98>, <+101>edx와 0xf이 다르면 폭탄이 터진다.

아까 edx를 분석한 부분에서 eax에 넣는 것이 바뀔 때 마다 1씩 추가가 되었는데 여기서 0xf일 때 통과되는 것을 보면, 지금은 f가 4번째 나와서 4인데 eax가 0xf일 때 통과되고, edx도 0xf일 때 통과되는걸 보면 eax에 넣는 것이 마지막으로 0xf가 되야하는 것처럼 보인다. 그래서 첫 번째 인자(인자1)에 1 2 e 6 f를 제외하고 3을 넣어봤다. 이번엔 3으로 시작되어 3 7 b d 9 4 8 0 a 1 2 e 6 f 총 14개로 끝난다. 여기서 1부터 f까지 나오지 않은 수는 5와 c이기 때문에 인자1로 5를 인자2로 2를 넣어봤다. 이때도 5를 시작으로 5 c 3 7 b d 9 4 8 0 a 1 2 e 6 f로 rdx가 마지막 0xf일 때 eax에도 0xf가 들어있다. 그러므로 5를 인자1로 해야 0xf가 마지막에 오는 순서인 것으로 보인다.

<+86>~<+101> : eax는 0xf이기 때문에 통과, edx도 0xf이기 때문에 통과

<+103> : 인자2와 ecx를 비교해서 같으면 폭탄을 통과한다.

rcx의 값을 확인해보면 0x73으로 10진수로 변환하면 16*7 + 1*3 = 115 이다.

그러므로 인자1은 5, 인자2는 115를 입력하면 된다.

Phase 5 [정답]

5 115

Phase 6 [결과 화면 캡처]

```
Continuing.
1 5 6 3 2 4

Breakpoint 2, 0x0000555555555570 in phase_6 ()
(gdb) c
Continuing.
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
```

Phase 6 [진행 과정 설명]

<+187>에서 0x202bee(%rip)의 주소 부분이 가리키는 값이 복사된다는 것을 알 수 있다. 이 주소에 어떤 값이 저장되어 있는지 살펴보면

```
(gdb) x/30x 0x555555758220
0x555555758220 <node1>: 0x000001e5      0x00000001      0x55758230      0x000
05555
0x555555758230 <node2>: 0x00000363      0x00000002      0x55758240      0x000
05555
0x555555758240 <node3>: 0x0000004f      0x00000003      0x55758250      0x000
05555
0x555555758250 <node4>: 0x0000018d      0x00000004      0x55758260      0x000
05555
0x555555758260 <node5>: 0x000000ec      0x00000005      0x55758110      0x000
05555
0x555555758270: 0x00000000      0x00000000      0x00000000      0x00000000
0x555555758280 <host_table>:    0x55556ea7      0x00005555      0x55556ec1  0
x00005555
0x555555758290 <host_table+16>: 0x55556edb      0x00005555
```

위의 값을 표로 정리하면

노드	주소 위치	10진수
Node1	0x555555758220	485
Node2	0x555555758230	867
Node3	0x555555758240	79
Node4	0x555555758250	397
Node5	0x555555758260	236
Node6	0x555555758110	964

즉 우리는 해당하는 주소 부분에 6개의 노드들이 저장되어 있는 것을 알 수 있다.

<+94>에서 eax -= 0x1을 한 뒤 0x5보다 크면 폭탄이 터지므로 eax 인자들은 0x6이하임을 알 수 있다. 그리고 <+48>에서 r14d는 인자수를 세는 수인데 이것이 0x6이 아니면 폭탄이 터진다. 즉 6개의 십진수를 입력해야한다.

<+68>부터 <+71>을 보면 i번째 입력한 수와 i+1번째 입력한 수가 같아도 폭탄이 터진다.

그러면 <+179>부터 <+306>을 보면 입력된 값들을 연속으로 비교해 앞에 입력된 값이 뒤에 입력한 값보다 커야함을 알 수 있다. 내림차순 정렬이라는 뜻이다(6 2

1 4 5 3). 그리고 <+122>부터 <+133>을 확인하면 7에서 정렬된 노드의 번호를 뺀다.

즉, 우리가 입력할 값은 7-내림차순 정렬 노드가 된다.(1 5 6 3 2 4)

Phase 6 [정답]

1 5 6 3 2 4