Lab report #3

CSED 20170302 Kim Dae Hui

이번 랩은 bomb\_lab이다. assembly언어로 되어 있는 환경에서 bomb를 피해서 함수가 원하는 기능을 하도록 input을 넣으면 되는 것이다. 본 랩은 수업 pdf 4장,5장을 참조하여 assembly어의 기본을 익히고 나서 하면 좋을 것 같다.

|  |
| --- |
| 0x000000000040115a <phase\_1+0>: sub $0x8,%rsp  0x000000000040115e <phase\_1+4>: mov $0x402368,%esi  0x0000000000401163 <phase\_1+9>: callq 0x40119f <strings\_not\_equal>  0x0000000000401168 <phase\_1+14>: test %eax,%eax  0x000000000040116a <phase\_1+16>: je 0x401171 <phase\_1+23>  0x000000000040116c <phase\_1+18>: callq 0x40127b <explode\_bomb>  0x0000000000401171 <phase\_1+23>: add $0x8,%rsp  0x0000000000401175 <phase\_1+27>: retq  End of assembler dump.  (gdb) x/s 0x402368  0x402368 <\_\_dso\_handle+384>: "The future will be better tomorrow." |

1번 문제는 가장 먼저 풀 수 있는 phase로 비교적 간단하다. 처음에 rsp 값을 설정한 후에 esi에 스트링 값이 들어간다. 그리고 test부분에서 해당 함수에 들어가있던 스트링과 esi에 들어간 스트링의 값이 같으면 <phase\_1+23>부분으로 넘어가 explode\_bomb하지 않게 된다. 따라서 esi에 들어가게된 0x402368의 스트링이 뭔지 확인하여 답을알아냈다.

|  |
| --- |
| 0x0000000000400fa8 <phase\_2+0>: push %rbp  0x0000000000400fa9 <phase\_2+1>: push %rbx  0x0000000000400faa <phase\_2+2>: sub $0x28,%rsp  0x0000000000400fae <phase\_2+6>: mov %rsp,%rsi  0x0000000000400fb1 <phase\_2+9>: callq 0x40129d <read\_six\_numbers>  0x0000000000400fb6 <phase\_2+14>: cmpl $0x0,(%rsp)//0  0x0000000000400fba <phase\_2+18>: jne 0x400fc3 <phase\_2+27>  0x0000000000400fbc <phase\_2+20>: cmpl $0x1,0x4(%rsp)//1  0x0000000000400fc1 <phase\_2+25>: je 0x400fc8 <phase\_2+32>  0x0000000000400fc3 <phase\_2+27>: callq 0x40127b <explode\_bomb>  0x0000000000400fc8 <phase\_2+32>: lea 0x8(%rsp),%rbp //1  0x0000000000400fcd <phase\_2+37>: lea 0x18(%rsp),%rbp  0x0000000000400fd2 <phase\_2+42>: mov -0x8(%rbx),%eax  0x0000000000400fd5 <phase\_2+45>: add -0x4(%rbx),%eax  0x0000000000400fd8 <phase\_2+48>: cmp %eax,(%rbx)  0x0000000000400fda <phase\_2+50>: je 0x400fe1 <phase\_2+57>  0x0000000000400fdc <phase\_2+52>: callq 0x40127b <explode\_bomb>  0x0000000000400fe1 <phase\_2+57>: add $0x4,%rbx  0x0000000000400fe5 <phase\_2+61>: cmp %rbp,%rbx  0x0000000000400fe8 <phase\_2+64>: jne 0x400fd2 <phase\_2+42>  0x0000000000400fea <phase\_2+66>: add $0x28,%rsp  0x0000000000400fee <phase\_2+70>: pop %rbx  0x0000000000400fef <phase\_2+71>: pop %rbp  0x0000000000400ff0 <phase\_2+72>: retq |

2번문제는 함수의 이름으로부터 힌트를 얻었다. 6가지 숫자를 읽는다는 뜻이니 숫자가 6개 들어갈 것 같았다. 먼저 <phase2+14>에서 첫번째 숫자가0임을 알아낸 후 <phase\_2+20>에서 보면 그 다음 숫자는 1임을 알 수 있다. 그리고 <phase\_2+32>에서 1과 같은 값이 와야하므로 세번째 숫자도 1이된다. 그 다음 반복문 <phase\_2+64>까지를 살펴보면 앞의 숫자값을 더한 값과 그 다음값이 같아져야함을 알 수 있다. 따라서 여섯가지 숫자는 0 1 1 2 3 5로 추정할 수 있었다.

|  |
| --- |
| 0x00000000004010c1 <phase\_3+0>: sub $0x18,%rsp  0x00000000004010c5 <phase\_3+4>: lea 0x10(%rsp),%rcx  0x00000000004010ca <phase\_3+9>: lea 0x14(%rsp),%rdx  0x00000000004010cf <phase\_3+14>: mov $0x40245c,%esi  0x00000000004010d4 <phase\_3+19>: mov $0x0,%eax  0x00000000004010d9 <phase\_3+24>: callq 0x400af0 <sscanf@plt>  0x00000000004010de <phase\_3+29>: cmp $0x1,%eax  0x00000000004010e1 <phase\_3+32>: jg 0x4010e8 <phase\_3+39>  0x00000000004010e3 <phase\_3+34>: callq 0x40127b <explode\_bomb>  0x00000000004010e8 <phase\_3+39>: cmpl $0x7,0x14(%rsp)  0x00000000004010ed <phase\_3+44>: nopl (%rax)  0x00000000004010f0 <phase\_3+47>: ja 0x401139 <phase\_3+120>  0x00000000004010f2 <phase\_3+49>: mov 0x14(%rsp),%eax  0x00000000004010f6 <phase\_3+53>: jmpq \*0x4023a0(,%rax,8)  0x00000000004010fd <phase\_3+60>: mov $0x272,%eax  0x0000000000401102 <phase\_3+65>: jmp 0x40114a <phase\_3+137>  0x0000000000401104 <phase\_3+67>: mov $0x2e2,%eax  0x0000000000401109 <phase\_3+72>: jmp 0x40114a <phase\_3+137>  0x000000000040110b <phase\_3+74>: mov $0x231,%eax  0x0000000000401110 <phase\_3+79>: jmp 0x40114a <phase\_3+137>  0x0000000000401112 <phase\_3+81>: mov $0x132,%eax  0x0000000000401117 <phase\_3+86>: jmp 0x40114a <phase\_3+137>  0x0000000000401119 <phase\_3+88>: mov $0x3d3,%eax  0x000000000040111e <phase\_3+93>: xchg %ax,%ax  0x0000000000401120 <phase\_3+95>: jmp 0x40114a <phase\_3+137>  0x0000000000401122 <phase\_3+97>: mov $0x3cb,%eax  0x0000000000401127 <phase\_3+102>: jmp 0x40114a <phase\_3+137>  0x0000000000401129 <phase\_3+104>: mov $0x330,%eax  0x000000000040112e <phase\_3+109>: xchg %ax,%ax  0x0000000000401130 <phase\_3+111>: jmp 0x40114a <phase\_3+137>  0x0000000000401132 <phase\_3+113>: mov $0x36c,%eax  0x0000000000401137 <phase\_3+118>: jmp 0x40114a <phase\_3+137>  0x0000000000401139 <phase\_3+120>: nopl 0x0(%rax)  0x0000000000401140 <phase\_3+127>: callq 0x40127b <explode\_bomb>  0x0000000000401145 <phase\_3+132>: mov $0x0,%eax  0x000000000040114a <phase\_3+137>: cmp 0x10(%rsp),%eax  0x000000000040114e <phase\_3+141>: je 0x401155 <phase\_3+148>  0x0000000000401150 <phase\_3+143>: callq 0x40127b <explode\_bomb>  0x0000000000401155 <phase\_3+148>: add $0x18,%rsp  0x0000000000401159 <phase\_3+152>: retq |

3번 문제에서는 scanf함수가 사용되기 때문에 scanf함수가 받아들이는 인자들이 어디에 저장되는지, 그리고 어떤 인자들이 담기는지를 봐야한다. 따라서 I r 명령어를 통해서 scanf함수 전후를 살피는 것이 중요했다. 먼저 esi에 들어가는 문자열을 확인하기 위해 x/s $0x40245c로 알아보니 “%d %d”로 두가지 숫자가 들어간다는 점을 깨달았다.

그리고 나서 <phase\_3+32>를 확인하여 첫번째 들어가는 인자의 값이 1보다 커야한다는 점을 알아낸다. 그리고<phase\_3+39>를 통해 그 값이 7보다는 작아야함도 알아낸다. 그리고 eax에는 scanf에 들어온 인자의 갯수가 저장되는데, <phase3+127>의 bomb를 피하기 위해서 2보다 큰 값이 들어가도록 해줘야하기 때문에 결과적으로 3~6까지 수가 가능해진다.

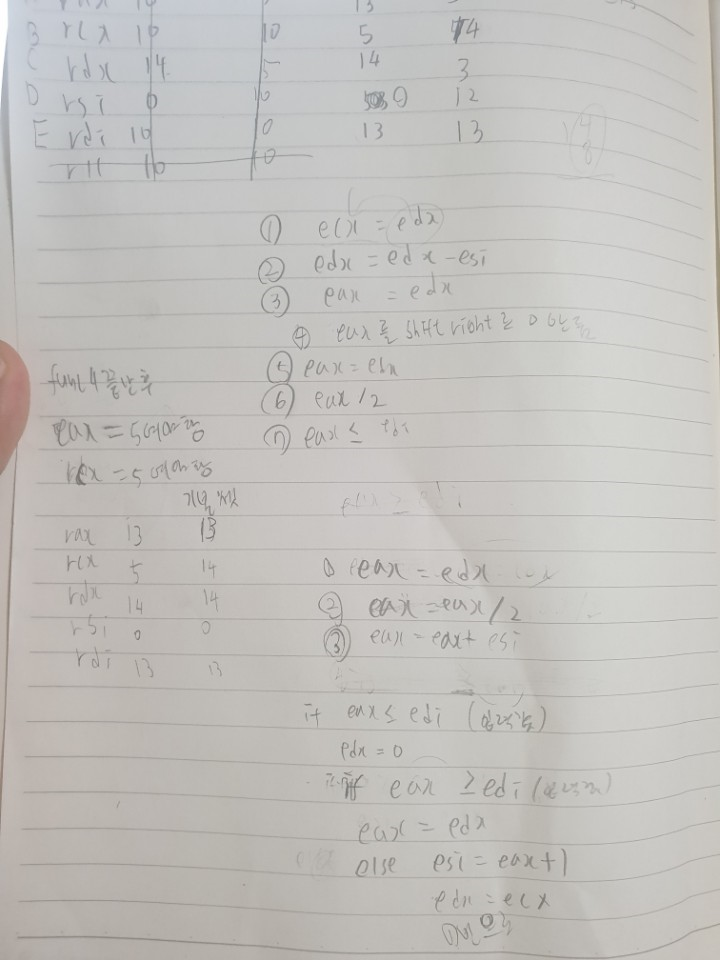
그리고 나서 뒤에 필요없는 점프문은 사용하지말고 <phase\_3+97>로 돌아가도록 값을 넣어줘야 bomb를 피할 수 있는데, 따라서 그 값과 같아지도록 0x3c b 인 971이 두번째 수임을 알게 되고 동시에 첫번째 값이 3임을 알게된다. 따라서 답은 3 971로 추정할 수 있게 된다.

|  |
| --- |
| 0x0000000000401062 <phase\_4+0>: sub $0x18,%rsp  0x0000000000401066 <phase\_4+4>: lea 0x10(%rsp),%rcx  0x000000000040106b <phase\_4+9>: lea 0x14(%rsp),%rdx  0x0000000000401070 <phase\_4+14>: mov $0x40245c,%esi  0x0000000000401075 <phase\_4+19>: mov $0x0,%eax  0x000000000040107a <phase\_4+24>: callq 0x400af0 <sscanf@plt>  0x000000000040107f <phase\_4+29>: cmp $0x2,%eax  0x0000000000401082 <phase\_4+32>: jne 0x401092 <phase\_4+48>  0x0000000000401084 <phase\_4+34>: mov 0x14(%rsp),%eax  0x0000000000401088 <phase\_4+38>: test %eax,%eax  0x000000000040108a <phase\_4+40>: js 0x401092 <phase\_4+48>  0x000000000040108c <phase\_4+42>: cmp $0xe,%eax  0x000000000040108f <phase\_4+45>: nop  0x0000000000401090 <phase\_4+46>: jle 0x401097 <phase\_4+53>  0x0000000000401092 <phase\_4+48>: callq 0x40127b <explode\_bomb>  0x0000000000401097 <phase\_4+53>: mov 0x14(%rsp),%edi  0x000000000040109b <phase\_4+57>: mov $0xe,%edx  0x00000000004010a0 <phase\_4+62>: mov $0x0,%esi  0x00000000004010a5 <phase\_4+67>: callq 0x400dd0 <func4>  0x00000000004010aa <phase\_4+72>: cmp $0x5,%eax  0x00000000004010ad <phase\_4+75>: jne 0x4010b6 <phase\_4+84>  0x00000000004010af <phase\_4+77>: cmpl $0x5,0x10(%rsp)  0x00000000004010b4 <phase\_4+82>: je 0x4010bb <phase\_4+89>  0x00000000004010b6 <phase\_4+84>: callq 0x40127b <explode\_bomb>  0x00000000004010bb <phase\_4+89>: add $0x18,%rsp  0x00000000004010bf <phase\_4+93>: nop  0x00000000004010c0 <phase\_4+94>: retq  0x0000000000400dd0 <func4+0>: mov %edx,%ecx  0x0000000000400dd2 <func4+2>: sub %esi,%edx  0x0000000000400dd4 <func4+4>: mov %edx,%eax  0x0000000000400dd6 <func4+6>: shr $0x1f,%eax  0x0000000000400dd9 <func4+9>: add %edx,%eax  0x0000000000400ddb <func4+11>: sar %eax  0x0000000000400ddd <func4+13>: add %esi,%eax  0x0000000000400ddf <func4+15>: cmp %edi,%eax  0x0000000000400de1 <func4+17>: jle 0x400df0 <func4+32>  0x0000000000400de3 <func4+19>: lea -0x1(%rax),%edx  0x0000000000400de6 <func4+22>: callq 0x400dd0 <func4>  0x0000000000400deb <func4+27>: lea (%rax,%rax,1),%edx  0x0000000000400dee <func4+30>: jmp 0x400e07 <func4+55>  0x0000000000400df0 <func4+32>: mov $0x0,%edx  0x0000000000400df5 <func4+37>: cmp %edi,%eax  0x0000000000400df7 <func4+39>: jge 0x400e07 <func4+55>  0x0000000000400df9 <func4+41>: lea 0x1(%rax),%esi  0x0000000000400dfc <func4+44>: mov %ecx,%edx  0x0000000000400dfe <func4+46>: callq 0x400dd0 <func4>  0x0000000000400e03 <func4+51>: lea 0x1(%rax,%rax,1),%edx  0x0000000000400e07 <func4+55>: mov %edx,%eax  0x0000000000400e09 <func4+57>: retq |

4번에서도 3번의 앞부분과 같이 몇개의 인자가 들어가는지 같은 방법으로 알아냈고 따라서 적당한 값을 집어넣고 돌리기 시작했다. 이문제는 크게 두 가지 영역으로 나뉜다. 첫번째 부분은 <func4>가 call되기 전phase\_4부분들이고 두번째는 func4의 부분이다.

결론부터 말하자면 우선 내가 입력해야할 값의 두번째는 매우 찾기 쉬웠다. Func4의 return값과 <phase\_4+77>의 cmpl문에서 같아야 bomb하지 않기 때문에 0x5의 값을 가진decimal인 5를 입력하면 되는 것을 처음에 func4를 읽지 않고도 쉽게 알아낼 수 있었다 이것은 그냥 코드를 따라가면 되므로 설명은 생략하겠다.

하지만 문제는 첫번째 입력값이다. 우선 첫번째 입력값의 범위는 <phase\_4+42>에서 0xe보다 작거나 같음으로 제한된다. 그 후 func4를 보면 먼저 초기 셋팅이 있고 그 후에 eax값과 edx값을 통해 phase\_4에 있던 값들을 변경시키는 재귀함수 형태임을 알 수 있다. 먼저 rax는 rsi와 edx의 평균값을 가지는 셋팅을 보인다. 앞에서 들어올 때 0xe의 값을 rdx가 가지고 있으므로 14로 치고, 만약에 내가 첫번째 입력한 수가 rax보다 작거나 같으면 한번더 함수를 실행하지 않고 그게 아니라면 rsi에 rax+1을 넣고 함수를 한 번 더 실행하게 된다. 따라서 초기의 eax는 14와 0의 평균인 7의 값을 가지고, (rax+1)의 값은8이고 14와의 평균은 11이 된다. 내 입력값을 여기서 10으로 해주면 함수가 1바퀴만 돌고 끝나기 때문에 내가 입력한 5의 값을 return받을 수 있게 되기 떄문에 답은 10과 5가 된다.



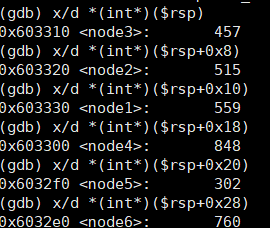
|  |
| --- |
| 0x0000000000400ff1 <phase\_5+0>: sub $0x18,%rsp  0x0000000000400ff5 <phase\_5+4>: lea 0x10(%rsp),%rcx  0x0000000000400ffa <phase\_5+9>: lea 0x14(%rsp),%rdx  0x0000000000400fff <phase\_5+14>: mov $0x40245c,%esi  0x0000000000401004 <phase\_5+19>: mov $0x0,%eax  0x0000000000401009 <phase\_5+24>: callq 0x400af0 <sscanf@plt>  0x000000000040100e <phase\_5+29>: cmp $0x1,%eax  0x0000000000401011 <phase\_5+32>: jg 0x401018 <phase\_5+39>  0x0000000000401013 <phase\_5+34>: callq 0x40127b <explode\_bomb>  0x0000000000401018 <phase\_5+39>: mov 0x14(%rsp),%eax  0x000000000040101c <phase\_5+43>: and $0xf,%eax  0x000000000040101f <phase\_5+46>: mov %eax,0x14(%rsp)  0x0000000000401023 <phase\_5+50>: cmp $0xf,%eax  0x0000000000401026 <phase\_5+53>: je 0x401058 <phase\_5+103>  0x0000000000401028 <phase\_5+55>: mov $0x0,%edx  0x000000000040102d <phase\_5+60>: mov $0x0,%ecx  0x0000000000401032 <phase\_5+65>: add $0x1,%edx  0x0000000000401035 <phase\_5+68>: cltq  0x0000000000401037 <phase\_5+70>: mov 0x4023e0(,%rax,4),%eax  0x000000000040103e <phase\_5+77>: add %eax,%ecx  0x0000000000401040 <phase\_5+79>: cmp $0xf,%eax  0x0000000000401043 <phase\_5+82>: jne 0x401032 <phase\_5+65>  0x0000000000401045 <phase\_5+84>: movl $0xf,0x14(%rsp)  0x000000000040104d <phase\_5+92>: cmp $0xf,%edx  0x0000000000401050 <phase\_5+95>: jne 0x401058 <phase\_5+103>  0x0000000000401052 <phase\_5+97>: cmp %ecx,0x10(%rsp)  0x0000000000401056 <phase\_5+101>: je 0x40105d <phase\_5+108>  0x0000000000401058 <phase\_5+103>: callq 0x40127b <explode\_bomb>  0x000000000040105d <phase\_5+108>: add $0x18,%rsp  0x0000000000401061 <phase\_5+112>: retq |

5번 문제 또한 앞의 문제와 같은 방식으로 두 숫자가 scanf의 인자로 들어온다는 사실을 깨달았다. 그리고 eax는 <phase\_5+43>에서 16으로 나눈 나머지임을 알 수 있다.그리고 <phase\_5+65>부터 <phase\_5+82>까지 부분에서 eax에 들어가는 값이 15가 될때까지 ecx=ecx+eax를 해주면서 ecx에 더해주는 숫자를 계속 쌓아가면서 15가 될때까지 숫자를 더하고 15로 나눔을 반복한다. <phast\_5+95>에서 explode\_bomb로 가지 않게 하기 위한인자는 5가 됨을 계산을 통해 알게 되었고 5를 넣어 15가 나올때까지 더해지는 수는 총115이므로 두 입력값은 5 115가 된다.

|  |
| --- |
| 0x0000000000400e93 <phase\_6+0>: push %r15  0x0000000000400e95 <phase\_6+2>: push %r14  0x0000000000400e97 <phase\_6+4>: push %r13  0x0000000000400e99 <phase\_6+6>: push %r12  0x0000000000400e9b <phase\_6+8>: push %rbp  0x0000000000400e9c <phase\_6+9>: push %rbx  0x0000000000400e9d <phase\_6+10>: sub $0x58,%rsp  0x0000000000400ea1 <phase\_6+14>: lea 0x30(%rsp),%rsi  0x0000000000400ea6 <phase\_6+19>: callq 0x40129d <read\_six\_numbers>  0x0000000000400eab <phase\_6+24>: mov $0x0,%r13d  0x0000000000400eb1 <phase\_6+30>: lea 0x30(%rsp),%r14  0x0000000000400eb6 <phase\_6+35>: mov %r14,%r15  0x0000000000400eb9 <phase\_6+38>: mov %r13,%r12  0x0000000000400ebc <phase\_6+41>: mov (%r14,%r13,4),%eax  0x0000000000400ec0 <phase\_6+45>: sub $0x1,%eax  0x0000000000400ec3 <phase\_6+48>: cmp $0x5,%eax  0x0000000000400ec6 <phase\_6+51>: jbe 0x400ecd <phase\_6+58>  0x0000000000400ec8 <phase\_6+53>: callq 0x40127b <explode\_bomb>  0x0000000000400ecd <phase\_6+58>: lea 0x1(%r13),%ebx  0x0000000000400ed1 <phase\_6+62>: cmp $0x5,%r13  0x0000000000400ed5 <phase\_6+66>: je 0x400f85 <phase\_6+242>  0x0000000000400edb <phase\_6+72>: mov %r15,%rbp  0x0000000000400ede <phase\_6+75>: movslq %ebx,%rdx  0x0000000000400ee1 <phase\_6+78>: mov 0x0(%rbp,%r12,4),%eax  0x0000000000400ee6 <phase\_6+83>: cmp 0x30(%rsp,%rdx,4),%eax  0x0000000000400eea <phase\_6+87>: jne 0x400ef1 <phase\_6+94>  0x0000000000400eec <phase\_6+89>: callq 0x40127b <explode\_bomb>  0x0000000000400ef1 <phase\_6+94>: add $0x1,%ebx  0x0000000000400ef4 <phase\_6+97>: cmp $0x5,%ebx  0x0000000000400ef7 <phase\_6+100>: jle 0x400ede <phase\_6+75>  0x0000000000400ef9 <phase\_6+102>: add $0x1,%r13  0x0000000000400efd <phase\_6+106>: jmp 0x400eb9 <phase\_6+38>  0x0000000000400eff <phase\_6+108>: mov 0x8(%rsi),%rsi  0x0000000000400f03 <phase\_6+112>: add $0x1,%edx  0x0000000000400f06 <phase\_6+115>: movslq %ecx,%rax  0x0000000000400f09 <phase\_6+118>: cmp 0x30(%rsp,%rax,4),%edx  0x0000000000400f0d <phase\_6+122>: jl 0x400eff <phase\_6+108>  0x0000000000400f0f <phase\_6+124>: mov %rsi,(%rsp,%rax,8)  0x0000000000400f13 <phase\_6+128>: add $0x1,%ecx  0x0000000000400f16 <phase\_6+131>: cmp $0x5,%ecx  0x0000000000400f19 <phase\_6+134>: jg 0x400f27 <phase\_6+148>  0x0000000000400f1b <phase\_6+136>: mov $0x603330,%esi  0x0000000000400f20 <phase\_6+141>: mov $0x1,%edx  0x0000000000400f25 <phase\_6+146>: jmp 0x400f06 <phase\_6+115>  0x0000000000400f27 <phase\_6+148>: mov (%rsp),%rcx  0x0000000000400f2b <phase\_6+152>: mov 0x8(%rsp),%rax  0x0000000000400f30 <phase\_6+157>: mov %rax,0x8(%rcx)  0x0000000000400f34 <phase\_6+161>: mov 0x10(%rsp),%rdx  0x0000000000400f39 <phase\_6+166>: mov %rdx,0x8(%rax)  0x0000000000400f3d <phase\_6+170>: mov 0x18(%rsp),%rax  0x0000000000400f42 <phase\_6+175>: mov %rax,0x8(%rdx)  0x0000000000400f46 <phase\_6+179>: mov 0x20(%rsp),%rdx  0x0000000000400f4b <phase\_6+184>: mov %rdx,0x8(%rax)  0x0000000000400f4f <phase\_6+188>: mov 0x28(%rsp),%rax  0x0000000000400f54 <phase\_6+193>: mov %rax,0x8(%rdx)  0x0000000000400f58 <phase\_6+197>: movq $0x0,0x8(%rax)  0x0000000000400f60 <phase\_6+205>: mov %rcx,%rbx  0x0000000000400f63 <phase\_6+208>: mov $0x0,%ebp  0x0000000000400f68 <phase\_6+213>: mov 0x8(%rbx),%rdx  0x0000000000400f6c <phase\_6+217>: mov (%rbx),%eax  0x0000000000400f6e <phase\_6+219>: cmp (%rdx),%eax  0x0000000000400f70 <phase\_6+221>: jle 0x400f77 <phase\_6+228>  0x0000000000400f72 <phase\_6+223>: callq 0x40127b <explode\_bomb>  0x0000000000400f77 <phase\_6+228>: mov 0x8(%rbx),%rbx  0x0000000000400f7b <phase\_6+232>: add $0x1,%ebp  0x0000000000400f7e <phase\_6+235>: cmp $0x5,%ebp  0x0000000000400f81 <phase\_6+238>: je 0x400f99 <phase\_6+262>  0x0000000000400f83 <phase\_6+240>: jmp 0x400f68 <phase\_6+213>  0x0000000000400f85 <phase\_6+242>: mov $0x603330,%esi  0x0000000000400f8a <phase\_6+247>: mov $0x1,%edx  0x0000000000400f8f <phase\_6+252>: mov $0x0,%ecx  0x0000000000400f94 <phase\_6+257>: jmpq 0x400f06 <phase\_6+115>  0x0000000000400f99 <phase\_6+262>: add $0x58,%rsp  0x0000000000400f9d <phase\_6+266>: pop %rbx  0x0000000000400f9e <phase\_6+267>: pop %rbp  0x0000000000400f9f <phase\_6+268>: pop %r12  0x0000000000400fa1 <phase\_6+270>: pop %r13  0x0000000000400fa3 <phase\_6+272>: pop %r14  0x0000000000400fa5 <phase\_6+274>: pop %r15  0x0000000000400fa7 <phase\_6+276>: retq |

6번 문제에서 push갯수를 통해 숫자가 6개 들어간다는 점을 유추할 수 있었다. 그리고 phase의 앞부분에서 6가지의 숫자가 다 다른 숫자가 들어가도록 반복문을 통해 검사한다. 검사가 끝난 후에 숫자들이 1이상이 되어야하고 7을 뺏을 때 0보다 커야하므로 숫자들은 1부터 6까지의 자연수가 되야함을 유추할 수 있다.

그리고 뒷부분을 보면 포인터값들이 작은 순으로 배열되지 않으면 explode\_bomb가 됨을 알 수 있다. 따라서, 처음에 push된 것들의 포인터값들을 알아내야 한다.



이 같은 명령어를 통해서 입력한 값들의 포인터 값들을 알아낼 수 있다. 그리고 나서 +221에 따라 포인터 값이 작은 것에서부터 큰 순으로 배열해보면 답은 5 3 2 1 6 4가 됨을 유추해볼 수 있다.

|  |
| --- |
| 0x0000000000400e3e <secret\_phase+0>: push %rbx  0x0000000000400e3f <secret\_phase+1>: callq 0x40135f <read\_line>  0x0000000000400e44 <secret\_phase+6>: mov %rax,%rdi  0x0000000000400e47 <secret\_phase+9>: mov $0x0,%ecx  0x0000000000400e4c <secret\_phase+14>: mov $0xa,%edx  0x0000000000400e51 <secret\_phase+19>: mov $0x0,%esi  0x0000000000400e56 <secret\_phase+24>: callq 0x400ae0 <\_\_strtol\_internal@plt>  0x0000000000400e5b <secret\_phase+29>: mov %eax,%ebx  0x0000000000400e5d <secret\_phase+31>: lea -0x1(%rax),%eax  0x0000000000400e60 <secret\_phase+34>: cmp $0x3e8,%eax  0x0000000000400e65 <secret\_phase+39>: jbe 0x400e6c <secret\_phase+46>  0x0000000000400e67 <secret\_phase+41>: callq 0x40127b <explode\_bomb>  0x0000000000400e6c <secret\_phase+46>: mov %ebx,%esi  0x0000000000400e6e <secret\_phase+48>: mov $0x603500,%edi  0x0000000000400e73 <secret\_phase+53>: callq 0x400e0a <fun7>  0x0000000000400e78 <secret\_phase+58>: cmp $0x1,%eax  0x0000000000400e7b <secret\_phase+61>: je 0x400e82 <secret\_phase+68>  0x0000000000400e7d <secret\_phase+63>: callq 0x40127b <explode\_bomb>  0x0000000000400e82 <secret\_phase+68>: mov $0x402340,%edi  0x0000000000400e87 <secret\_phase+73>: callq 0x400a50 <puts@plt>  0x0000000000400e8c <secret\_phase+78>: callq 0x401203 <phase\_defused>  0x0000000000400e91 <secret\_phase+83>: pop %rbx  0x0000000000400e92 <secret\_phase+84>: retq |

시크릿페이즈는 발견하였으나 풀지 못했기 때문에 보고서에 적지 않았다.