**PHASE 1:**

0x0000000000400e2d <+0>: sub $0x8,%rsp

0x0000000000400e31 <+4>: mov  **$0x4023f0**,%esi

0x0000000000400e36 <+9>: callq 0x4012df <**strings\_not\_equal**>

0x0000000000400e3b <+14>: test %eax,%eax

0x0000000000400e3d <+16>: je 0x400e44 <phase\_1+23>

0x0000000000400e3f <+18>: callq 0x4014b6 <explode\_bomb>

0x0000000000400e44 <+23>: add $0x8,%rsp

0x0000000000400e48 <+27>: retq

LEA stands for Load Effective Address. It takes the result from address generation, and puts that result directly in a register, without accessing memory.

MOV, on the other hand, will move data to or from memory at the computed offset.

Mov 0x40(rax,4),rax ->take value at that address and put it into rax = value at 0x48=6

Lea 0x40(rax,4),rax ->Just doing the math hence 2\*4 +0x40 = 0x48

**PHASE 2:(loop)**

=> 0x0000000000400e49 <+0>: push %rbp // save callee registers

0x0000000000400e4a <+1>: push %rbx

0x0000000000400e4b <+2>: sub $0x28,%rsp

0x0000000000400e4f <+6>: mov %rsp,%rsi // rsi=1

0x0000000000400e52 <+9>: callq 0x4014ec <*read\_six\_numbers*>

Dump of assembler code for function read\_six\_numbers:

=> 0x00000000004014ec <+0>: sub $0x8,%rsp

0x00000000004014f0 <+4>: mov %rsi,%rdx

0x00000000004014f3 <+7>: lea **0x4**(%rsi),%rcx

0x00000000004014f7 <+11>: lea  **0x14**(%rsi),%rax

0x00000000004014fb <+15>: push %rax

0x00000000004014fc <+16>: lea **0x10**(%rsi),%rax

0x0000000000401500 <+20>: push %rax

0x0000000000401501 <+21>: lea **0xc**(%rsi),%r9

0x0000000000401505 <+25>: lea **0x8**(%rsi),%r8

0x0000000000401509 <+29>: mov $0x402697,%esi

0x000000000040150e <+34>: mov $0x0,%eax

0x0000000000401513 <+39>: callq 0x400b50 <\_\_isoc99\_sscanf@plt>

0x0000000000401518 <+44>: add $0x10,%rsp

0x000000000040151c <+48>: cmp $0x5,%eax

0x000000000040151f <+51>: jg 0x401526 <read\_six\_numbers+58>

0x0000000000401521 <+53>: callq 0x4014b6 <explode\_bomb>

0x0000000000401526 <+58>: add $0x8,%rsp

0x000000000040152a <+62>: retq

0x0000000000400e57 <+14>:  **cmpl $0x0,(%rsp) //0**

0x0000000000400e5b <+18>: jne 0x400e64 <phase\_2+27>

0x0000000000400e5d <+20>:  **cmpl $0x1,0x4(%rsp) //1**

0x0000000000400e62 <+25>: je 0x400e69 <phase\_2+32>

0x0000000000400e64 <+27>: callq 0x4014b6 <explode\_bomb>

0x0000000000400e69 <+32>: mov %rsp,%rbx //rax=2

0x0000000000400e6c <+35>:  **lea 0x10(%rsp),%rbp**

0x0000000000400e71 <+40>: mov 0x4(%rbx),%eax

0x0000000000400e74 <+43>: add **(%rbx),%eax //eax=1 this line is where we add our current number to the previous number**

0x0000000000400e76 <+45>: cmp %eax,0x8(%rbx) //rbx=0 rax=1

0x0000000000400e79 <+48>: je 0x400e80 <phase\_2+55>

0x0000000000400e7b <+50>: callq 0x4014b6 <explode\_bomb>

0x0000000000400e80 <+55>: add $0x4,%rbx //address of rbx changed 4 because 4 is how much space an int takes

0x0000000000400e84 <+59>: cmp %rbp,%rbx

0x0000000000400e87 <+62>: jne 0x400e71 <phase\_2+40>

0x0000000000400e89 <+64>: add $0x28,%rsp

0x0000000000400e8d <+68>: pop %rbx

0x0000000000400e8e <+69>: pop %rbp

0x0000000000400e8f <+70>: retq

//put space so it reads as 6 different things instead of one number

//x/d $ebp-0x1c

0 1 1 2 3 5

Vi bomb.c will print the c code

display/i $pc = will show where you are every time you do ni

Strings bomb (before entering gdb)

**PHASE 3(switch case)**

0x0000000000400e90 <+0>: sub $0x18,%rsp

0x0000000000400e94 <+4>: lea 0x8(%rsp),%r8(last value inputed)

0x0000000000400e99 <+9>: lea 0x7(%rsp),%rcx

0x0000000000400e9e <+14>: lea 0xc(%rsp),%rdx //1st value inputed

0x0000000000400ea3 <+19>: mov $0x402446,%esi check what’s in the address it is "%d %c %d" hence d=decimal c=characters

0x0000000000400ea8 <+24>: mov $0x0,%eax

0x0000000000400ead <+29>: callq 0x400b50 <\_\_isoc99\_sscanf@plt>

Dump of assembler code for function \_\_isoc99\_sscanf@plt:

=> 0x0000000000400b50 <+0>: jmpq \*0x20353a(%rip) # 0x604090

0x0000000000400b56 <+6>: pushq $0xf

0x0000000000400b5b <+11>: jmpq 0x400a50

0x0000000000400eb2 <+34>: cmp $0x2,%eax //eax=3

0x0000000000400eb5 <+37>: jg 0x400ebc <phase\_3+44>

0x0000000000400eb7 <+39>: callq 0x4014b6 <explode\_bomb>

0x0000000000400ebc <+44>: **cmpl $0x7,0xc(%rsp) //range 0-7 //my number is 1**

0x0000000000400ec1 <+49>: ja 0x400fc3 <phase\_3+307>//jump if above

0x0000000000400ec7 <+55>: mov 0xc(%rsp),%eax

0x0000000000400ecb <+59>: jmpq **\*0x402460**(,%rax,8) //+8 and get address jump table

x/8xw \_\_\_\_address will show the jump table

0x0000000000400ed2 <+66>: mov $0x68,%eax

0x0000000000400ed7 <+71>: cmpl $0x1f1,0x8(%rsp)

0x0000000000400edf <+79>: je 0x400fcd <phase\_3+317>

0x0000000000400ee5 <+85>: callq 0x4014b6 <explode\_bomb>

0x0000000000400eea <+90>: mov $0x68,%eax

0x0000000000400eef <+95>: jmpq 0x400fcd <phase\_3+317>

0x0000000000400ef4 <+100>: **mov $0x79,%eax //convert 0x79 into char**

0x0000000000400ef9 <+105>: **cmpl $0x1ae,0x8(%rsp) //3rd thing,converted into decimal**

0x0000000000400f01 <+113>: je 0x400fcd <phase\_3+317>

0x0000000000400f07 <+119>: callq 0x4014b6 <explode\_bomb>

0x0000000000400f0c <+124>: mov $0x79,%eax

0x0000000000400f11 <+129>: jmpq 0x400fcd <phase\_3+317>

0x0000000000400f16 <+134>: mov $0x67,%eax

0x0000000000400f1b <+139>: cmpl $0x1ac,0x8(%rsp)

0x0000000000400f23 <+147>: je 0x400fcd <phase\_3+317>

0x0000000000400f29 <+153>: callq 0x4014b6 <explode\_bomb>

0x0000000000400f2e <+158>: mov $0x67,%eax

0x0000000000400f33 <+163>: jmpq 0x400fcd <phase\_3+317>

0x0000000000400f38 <+168>: mov $0x68,%eax

0x0000000000400f3d <+173>: cmpl $0x144,0x8(%rsp)

0x0000000000400f45 <+181>: je 0x400fcd <phase\_3+317>

0x0000000000400f4b <+187>: callq 0x4014b6 <explode\_bomb>

0x0000000000400f50 <+192>: mov $0x68,%eax

0x0000000000400f55 <+197>: jmp 0x400fcd <phase\_3+317>

0x0000000000400f57 <+199>: mov $0x61,%eax

0x0000000000400f5c <+204>: cmpl $0x238,0x8(%rsp)

0x0000000000400f64 <+212>: je 0x400fcd <phase\_3+317>

0x0000000000400f66 <+214>: callq 0x4014b6 <explode\_bomb>

0x0000000000400f6b <+219>: mov $0x61,%eax

0x0000000000400f70 <+224>: jmp 0x400fcd <phase\_3+317>

0x0000000000400f72 <+226>: mov $0x76,%eax

0x0000000000400f77 <+231>: cmpl $0xac,0x8(%rsp)

0x0000000000400f7f <+239>: je 0x400fcd <phase\_3+317>

0x0000000000400f81 <+241>: callq 0x4014b6 <explode\_bomb>

0x0000000000400f86 <+246>: mov $0x76,%eax

0x0000000000400f8b <+251>: jmp 0x400fcd <phase\_3+317>

0x0000000000400f8d <+253>: mov $0x65,%eax

0x0000000000400f92 <+258>: cmpl $0x348,0x8(%rsp)

0x0000000000400f9a <+266>: je 0x400fcd <phase\_3+317>

0x0000000000400f9c <+268>: callq 0x4014b6 <explode\_bomb>

0x0000000000400fa1 <+273>: mov $0x65,%eax

0x0000000000400fa6 <+278>: jmp 0x400fcd <phase\_3+317>

0x0000000000400fa8 <+280>: mov $0x6c,%eax

0x0000000000400fad <+285>: cmpl $0x32b,0x8(%rsp)

0x0000000000400fb5 <+293>: je 0x400fcd <phase\_3+317>

0x0000000000400fb7 <+295>: callq 0x4014b6 <explode\_bomb>

0x0000000000400fbc <+300>: mov $0x6c,%eax

0x0000000000400fc1 <+305>: jmp 0x400fcd <phase\_3+317>

0x0000000000400fc3 <+307>: callq 0x4014b6 <explode\_bomb>

0x0000000000400fc8 <+312>: mov $0x72,%eax

0x0000000000400fcd <+317>:  **cmp 0x7(%rsp),%al//jumped here**

0x0000000000400fd1 <+321>: je 0x400fd8 <phase\_3+328>

0x0000000000400fd3 <+323>: callq 0x4014b6 <explode\_bomb>

0x0000000000400fd8 <+328>: add $0x18,%rsp

0x0000000000400fdc <+332>: retq

400e90: 48 83 ec 18 sub $0x18,%rsp

400e94: 4c 8d 44 24 08 lea 0x8(%rsp),%r8

400e99: 48 8d 4c 24 07 lea 0x7(%rsp),%rcx

400e9e: 48 8d 54 24 0c lea 0xc(%rsp),%rdx

400ea3: be 46 24 40 00 mov $0x402446,%esi

400ea8: b8 00 00 00 00 mov $0x0,%eax

400ead: e8 9e fc ff ff callq 400b50 <\_\_isoc99\_sscanf@plt>

400eb2: 83 f8 02 cmp $0x2,%eax

400eb5: 7f 05 jg 400ebc <phase\_3+0x2c>

400eb7: e8 fa 05 00 00 callq 4014b6 <explode\_bomb>

400ebc: 83 7c 24 0c 07 cmpl $0x7,0xc(%rsp)

400ec1: 0f 87 fc 00 00 00 ja 400fc3 <phase\_3+0x133>

400ec7: 8b 44 24 0c mov 0xc(%rsp),%eax

400ecb: ff 24 c5 60 24 40 00 jmpq \*0x402460(,%rax,8)

400ed2: b8 68 00 00 00 mov $0x68,%eax

400ed7: 81 7c 24 08 f1 01 00 cmpl $0x1f1,0x8(%rsp)

400ede: 00

400edf: 0f 84 e8 00 00 00 je 400fcd <phase\_3+0x13d>

400ee5: e8 cc 05 00 00 callq 4014b6 <explode\_bomb>

400eea: b8 68 00 00 00 mov $0x68,%eax

400eef: e9 d9 00 00 00 jmpq 400fcd <phase\_3+0x13d>

400ef4: b8 79 00 00 00 mov $0x79,%eax

400ef9: 81 7c 24 08 ae 01 00 cmpl $0x1ae,0x8(%rsp)

400f00: 00

400f01: 0f 84 c6 00 00 00 je 400fcd <phase\_3+0x13d>

400f07: e8 aa 05 00 00 callq 4014b6 <explode\_bomb>

400f0c: b8 79 00 00 00 mov $0x79,%eax

400f11: e9 b7 00 00 00 jmpq 400fcd <phase\_3+0x13d>

400f16: b8 67 00 00 00 mov $0x67,%eax

400f1b: 81 7c 24 08 ac 01 00 cmpl $0x1ac,0x8(%rsp)

400f22: 00

400f23: 0f 84 a4 00 00 00 je 400fcd <phase\_3+0x13d>

400f29: e8 88 05 00 00 callq 4014b6 <explode\_bomb>

400f2e: b8 67 00 00 00 mov $0x67,%eax

400f33: e9 95 00 00 00 jmpq 400fcd <phase\_3+0x13d>

400f38: b8 68 00 00 00 mov $0x68,%eax

400f3d: 81 7c 24 08 44 01 00 cmpl $0x144,0x8(%rsp)

400f44: 00

400f45: 0f 84 82 00 00 00 je 400fcd <phase\_3+0x13d>

400f4b: e8 66 05 00 00 callq 4014b6 <explode\_bomb>

400f50: b8 68 00 00 00 mov $0x68,%eax

400f55: eb 76 jmp 400fcd <phase\_3+0x13d>

400f57: b8 61 00 00 00 mov $0x61,%eax

400f5c: 81 7c 24 08 38 02 00 cmpl $0x238,0x8(%rsp)

400f63: 00

400f64: 74 67 je 400fcd <phase\_3+0x13d>

400f66: e8 4b 05 00 00 callq 4014b6 <explode\_bomb>

400f6b: b8 61 00 00 00 mov $0x61,%eax

400f70: eb 5b jmp 400fcd <phase\_3+0x13d>

400f72: b8 76 00 00 00 mov $0x76,%eax

400f77: 81 7c 24 08 ac 00 00 cmpl $0xac,0x8(%rsp)

400f7e: 00

400f7f: 74 4c je 400fcd <phase\_3+0x13d>

400f81: e8 30 05 00 00 callq 4014b6 <explode\_bomb>

400f86: b8 76 00 00 00 mov $0x76,%eax

400f8b: eb 40 jmp 400fcd <phase\_3+0x13d>

400f8d: b8 65 00 00 00 mov $0x65,%eax

400f92: 81 7c 24 08 48 03 00 cmpl $0x348,0x8(%rsp)

400f99: 00

400f9a: 74 31 je 400fcd <phase\_3+0x13d>

400f9c: e8 15 05 00 00 callq 4014b6 <explode\_bomb>

400fa1: b8 65 00 00 00 mov $0x65,%eax

400fa6: eb 25 jmp 400fcd <phase\_3+0x13d>

400fa8: b8 6c 00 00 00 mov $0x6c,%eax

400fad: 81 7c 24 08 2b 03 00 cmpl $0x32b,0x8(%rsp)

400fb4: 00

400fb5: 74 16 je 400fcd <phase\_3+0x13d>

400fb7: e8 fa 04 00 00 callq 4014b6 <explode\_bomb>

400fbc: b8 6c 00 00 00 mov $0x6c,%eax

400fc1: eb 0a jmp 400fcd <phase\_3+0x13d>

400fc3: e8 ee 04 00 00 callq 4014b6 <explode\_bomb>

400fc8: b8 72 00 00 00 mov $0x72,%eax

400fcd: 3a 44 24 07 cmp 0x7(%rsp),%al

400fd1: 74 05 je 400fd8 <phase\_3+0x148>

400fd3: e8 de 04 00 00 callq 4014b6 <explode\_bomb>

400fd8: 48 83 c4 18 add $0x18,%rsp

400fdc: c3 retq

**PHASE 4:(Recursion)**

Dump of assembler code for function phase\_4:

0x0000000000401008 <+0>: sub $0x18,%rsp

0x000000000040100c <+4>: lea 0x8(%rsp),%rcx

0x0000000000401011 <+9>: lea 0xc(%rsp),%rdx

=> 0x0000000000401016 <+14>: mov $0x4026a3,%esi //"%d %d"

0x000000000040101b <+19>: mov $0x0,%eax

0x0000000000401020 <+24>: callq 0x400b50 <\_\_isoc99\_sscanf@plt> //will read my two input, //sscanf puts 2 into eax

0x0000000000401025 <+29>: cmp $0x2,%eax // \*0x0000000000401025(breakpoint at address) //you need 2 input

0x0000000000401028 <+32>: jne 0x401031 <phase\_4+41>

0x000000000040102a <+34>: cmpl  **$0xe(#14)**,0xc(%rsp)

0x000000000040102f <+39>: jbe 0x401036 <phase\_4+46> //jump below equal

0x0000000000401031 <+41>: callq 0x4014b6 <explode\_bomb>

0x0000000000401036 <+46>: *mov* *$0xe,%edx //14=edx*

*0x000000000040103b <+51>: mov $0x0,%esi //0=esi*

*0x0000000000401040 <+56>: mov 0xc(%rsp),%edi //first input*

0x0000000000401044 <+60>: callq 0x400fdd <func4> //edi in func 4 is my first input

0000000000400fdd <func4>: //edx,esi,edi are 3 argument

400fdd: 53 push %rbx

400fde: 89 d3 mov %edx,%ebx //edx=0,14

400fe0: 29 f3 sub %esi,%ebx //ebx=14

400fe2: d1 eb shr %ebx

//shift right by 1 since no operand saying how much to shift //shift by 1 is like dividing by 2 hence ebx is now 7

400fe4: 01 f3 add %esi,%ebx //0+7, ebx=7

400fe6: 39 fb cmp %edi,%ebx //first input(edi) vs 7(ebx)

400fe8: 76 0c jbe 400ff6 <func4+0x19>

**400fea: 8d 53 ff lea -0x1(%rbx),%edx //(rbx=ebx, 7-1=6)**

**400fed: e8 eb ff ff ff callq 400fdd <func4> //edx=6 ebx/rbx=7 edi=10 esi=0 edx=**

400ff2: 01 d8 add %ebx,%eax //after calling func4, ebx=7 eax=37.

400ff4: eb 10 jmp 401006 <func4+0x29>

400ff6: 89 d8 mov %ebx,%eax //eax=ebx=7

rax 0x2 2

rbx 0x7 7

rcx 0x0 0

rdx 0xe 14

rsi 0x0 0

rdi 0xa 10

400ff8: 39 fb cmp %edi,%ebx

400ffa: 73 0a jae 401006 <func4+0x29>

**400ffc: 8d 73 01 lea 0x1(%rbx),%esi //esi=8** (rbx=7, then we did 7+1=8 and put that into esi)

**400fff: e8 d9 ff ff ff callq 400fdd <func4>**

401004: 01 d8 add %ebx,%eax

401006: 5b pop %rbx

401007: c3 retq

0x0000000000401049 <+65>:  **cmp $0x25,**%eax //compare $eax to 37

0x000000000040104c <+68>: jne 0x401055 <phase\_4+77>

0x000000000040104e <+70>: **cmpl $0x25**,0x8(%rsp)

0x0000000000401053 <+75>: je 0x40105a <phase\_4+82>

0x0000000000401055 <+77>: callq 0x4014b6 <explode\_bomb>

0x000000000040105a <+82>: add $0x18,%rsp

0x000000000040105e <+86>: retq

$-> literal (int or string depends on context)

**PHASE 5**:(Table/array)

Dump of assembler code for function phase\_5:

=> 0x000000000040105f <+0>: sub $0x18,%rsp

0x0000000000401063 <+4>: lea 0x8(%rsp),%rcx //rcx=5

0x0000000000401068 <+9>: lea 0xc(%rsp),%rdx

0x000000000040106d <+14>: mov $0x4026a3,%esi //x/s 0x4026a3 ="%d %d" (hence input needs to be two number) //esi=4

0x0000000000401072 <+19>: mov $0x0,%eax

0x0000000000401077 <+24>: callq 0x400b50 <\_\_isoc99\_sscanf@plt>

0x000000000040107c <+29>: cmp $0x1,%eax //eax=2(number of input given) compare it to 1

0x000000000040107f <+32>: jg 0x401086 <phase\_5+39> //Making sure I have 2 input

0x0000000000401081 <+34>: callq 0x4014b6 <explode\_bomb>

0x0000000000401086 <+39>: mov  **0xc(%rsp)**,%eax //bold= contains my first input

0x000000000040108a <+43>: and $0xf,%eax AND it with 15 //eax=11(first input)(making sure answer is below 15)

0x000000000040108d <+46>: mov %eax,0xc(%rsp)

0x0000000000401091 <+50>: cmp $0xf,%eax

0x0000000000401094 <+53>: je 0x4010c6 <phase\_5+103> //if 0xf=eax go to explode bomb my first number can’t be 15

0x0000000000401096 <+55>: mov $0x0,%ecx

0x000000000040109b <+60>: mov $0x0,%edx //edx=counter

0x00000000004010a0 <+65>: add $0x1,%edx

0x00000000004010a3 <+68>: cltq //convert doubleword(16) to quadword

0x00000000004010a5 <+70>: mov 0x4024a0(,%rax,4),%eax //eax=13 //goes through an array

(gdb) x/a 0x4024a0

0x4024a0 <array.3601>: 0x2000x/xxx0000a

(gdb) x/16dw 0x4024a0

0x4024a0 <array.3601>: 10 2 14 7

0x4024b0 <array.3601+16>: 8 12 15 11

0x4024c0 <array.3601+32>: 0 4 1 13

0x4024d0 <array.3601+48>: 3 9  **6( lead to 15)**  **5**

0x00000000004010ac <+77>: add %eax,%ecx //ecx=will accumulate all the index value i hit

0x00000000004010ae <+79>: cmp $0xf,%eax //15 cmp to 15(end of loop)

0x00000000004010b1 <+82>: jne 0x4010a0 <phase\_5+65>

0x00000000004010b3 <+84>: movl $0xf,0xc(%rsp)

0x00000000004010bb <+92>:  **cmp $0xf,%edx** //comparing 15 to counter //The loop needs to loop 15 times

0x00000000004010be <+95>: jne 0x4010c6 <phase\_5+103> //so it needs to be equal to not go to explode bomb

**0x00000000004010c0 <+97>: cmp 0x8(%rsp),%ecx //ecx=115 //sum up every element in the array except 5**

0x00000000004010c4 <+101>: je 0x4010cb <phase\_5+108>

0x00000000004010c6 <+103>: callq 0x4014b6 <explode\_bomb>

0x00000000004010cb <+108>: add $0x18,%rsp

0x00000000004010cf <+112>: retq

~~11 41~~

x/30s array loc

Page 246 do exercise

**PHASE 6: 2 4 1 3 5 6 (LInked list)**

Dump of assembler code for function phase\_6:

=> 0x00000000004010d0 <+0>: push %r13

0x00000000004010d2 <+2>: push %r12

0x00000000004010d4 <+4>: push %rbp

0x00000000004010d5 <+5>: push %rbx

0x00000000004010d6 <+6>: sub $0x58,%rsp

0x00000000004010da <+10>: lea 0x30(%rsp),%rsi //rsi=5

0x00000000004010df <+15>: callq 0x4014ec <read\_six\_numbers> //input gonna be 6 number

0x00000000004010e4 <+20>: lea 0x30(%rsp),%r12 //rax=6(got 6 from function above) every important value is 0

0x00000000004010e9 <+25>: mov $0x0,%r13d //r12=2

0x00000000004010ef <+31>: mov %r12,%rbp //rax=6, rbp=2

0x00000000004010f2 <+34>: mov (%r12),%eax //eax=2

0x00000000004010f6 <+38>: sub $0x1,%eax//rax=1

0x00000000004010f9 <+41>: cmp $0x5,%eax //kind of a range that it needs to be less than 5

0x00000000004010fc <+44>: jbe 0x401103 <phase\_6+51> //jump if number is less than 5

0x00000000004010fe <+46>: callq 0x4014b6 <explode\_bomb>

0x0000000000401103 <+51>: add $0x1,%r13d

0x0000000000401107 <+55>: cmp $0x6,%r13d //r13=1 //6 vs 1

0x000000000040110b <+59>: je 0x40114a <phase\_6+122>

0x000000000040110d <+61>: mov %r13d,%ebx

0x0000000000401110 <+64>: movslq %ebx,%rax //rbx=1

0x0000000000401113 <+67>: mov 0x30(%rsp,%rax,4),%eax //eax=4

0x0000000000401117 <+71>: cmp %eax,0x0(%rbp) //eax=4 rbp=2

0x000000000040111a <+74>: jne 0x401121 <phase\_6+81>

0x000000000040111c <+76>: callq 0x4014b6 <explode\_bomb>

0x0000000000401121 <+81>: add $0x1,%ebx //ebx=2

0x0000000000401124 <+84>: cmp $0x5,%ebx //loop happens 6 times

0x0000000000401127 <+87>: jle 0x401110 <phase\_6+64> //rax & rbx=6

0x0000000000401129 <+89>: add $0x4,%r12 //r12=4

0x000000000040112d <+93>: jmp 0x4010ef <phase\_6+31> //r13 is changing by 1 gets out of the loop when it reaches 6

0x000000000040112f <+95>: mov 0x8(%rdx),%rdx (At that point rax and rbx are 6)

(gdb) x/32dw $edx //**x** /[*Length*][*Format*] [*Address expression*]

0x6042e0 <node1>: 544 1 6308592 0

0x6042f0 <node2>: 787 2 6308608 0

0x604300 <node3>: 503 3 6308624 0

0x604310 <node4>: 571 4 6308640 0

0x604320 <node5>: 264 5 6308656 0

0x604330 <node6>: 67 6 0 0

0x0000000000401133 <+99>: add $0x1,%eax

0x0000000000401136 <+102>: cmp %ecx,%eax

0x0000000000401138 <+104>: jne 0x40112f <phase\_6+95> //Loop

0x000000000040113a <+106>: mov %rdx,(%rsp,%rsi,2)

0x000000000040113e <+110>: add $0x4,%rsi// add 4 to your 4th input)

0x0000000000401142 <+114>: cmp $0x18,%rsi

0x0000000000401146 <+118>: jne 0x40114f <phase\_6+127>

0x0000000000401148 <+120>: jmp 0x401164 <phase\_6+148>

0x000000000040114a <+122>: mov $0x0,%esi

0x000000000040114f <+127>: **mov 0x30(%rsp,%rsi,1),%ecx**

0x0000000000401153 <+131>: mov $0x1,%eax //eax=1 //ecx=2

**0x0000000000401158 <+136>: mov $0x6042e0,%edx //hold second node**

**gdb) x/3x $edx**

**0x6042f0 <node2>: 0x00000313 0x00000002**

**x/d $0x6042f0(**

0x000000000040115d <+141>: cmp $0x1,%ecx

0x0000000000401160 <+144>: jg 0x40112f <phase\_6+95>

0x0000000000401162 <+146>: jmp 0x40113a <phase\_6+106>

0x0000000000401164 <+148>: mov (%rsp),%rbx //value into register

0x0000000000401168 <+152>: mov %rsp,%rax //pointer/address into the temp

0x000000000040116b <+155>: lea 0x28(%rsp),%rsi

0x0000000000401170 <+160>: mov %rbx,%rcx

0x0000000000401173 <+163>: mov 0x8(%rax),%rdx

0x0000000000401177 <+167>:  *mov %rdx,0x8(%rcx)*

*0x000000000040117b <+171>: add $0x8,%rax*

0x000000000040117f <+175>: mov %rdx,%rcx

(gdb) x/3x $rcx

0x604330 <node6>: 0x00000043 0x00000006 0x00000000

0x0000000000401182 <+178>: cmp %rsi,%rax

0x0000000000401185 <+181>: jne 0x401173 <phase\_6+163>

0x0000000000401187 <+183>: movq $0x0,0x8(%rdx)

0x000000000040118f <+191>: mov $0x5,%ebp

0x0000000000401194 <+196>: mov 0x8(%rbx),%rax

gdb) x/3x $rbx

0x6042e0 <node1>: 0x00000220 0x00000001 0x006042f0

(IN GDB TUTORIAL:

We can also use `x` to examine the memory that makes up a LinkedList structure:

```

(gdb) x/4x root

0x6024a0: 0x00000003 0x00000000 0x00602480 0x00000000

)

**0x0000000000401198 <+200>: mov (%rax),%eax(dereference memory)| mov %rax,%eax (operand size mismatch)**

**0x000000000040119a <+202>: cmp %eax,(%rbx) //compared 2nd and 1st node**

**0x000000000040119c <+204>: jge 0x4011a3 <phase\_6+211>**

0x000000000040119e <+206>: callq 0x4014b6 <explode\_bomb>

0x00000000004011a3 <+211>: mov **0x8(%rbx)**,%rbx //+8 for pointer

0x00000000004011a7 <+215>: sub $0x1,%ebp

0x00000000004011aa <+218>: jne 0x401194 <phase\_6+196>

0x00000000004011ac <+220>: add $0x58,%rsp

0x00000000004011b0 <+224>: pop %rbx

0x00000000004011b1 <+225>: pop %rbp

0x00000000004011b2 <+226>: pop %r12

0x00000000004011b4 <+228>: pop %r13

0x00000000004011b6 <+230>: retq

(gdb) x/3x ($rbx)

0x6042e0 <node1>: 0x00000220 0x00000001 0x006042f0 =544

(gdb) x/3x \*($rbx+8)

0x6042f0 <node2>: 0x00000313 0x00000002 0x00604300 =787 (Other way to figure out 787 is by doing p \*$rbx

(gdb) x/3x \*(\*($rbx+8)+8)

0x604300 <node3>: 0x000001f7 0x00000003 0x00604310 =503

(gdb) x/3x \*(\*(\*($rbx+8)+8)+8)

0x604310 <node4>: 0x0000023b 0x00000004 0x00604320 =571

(gdb) x/3x \*(\*(\*(\*($rbx+8)+8)+8)+8)

0x604320 <node5>: 0x00000108 0x00000005 0x00604330 =264

(gdb) x/3x \*(\*(\*(\*(\*($rbx+8)+8)+8)+8)+8)

0x604330 <node6>: 0x00000043 0x00000006 0x00000000 =67

Greatest to smallest

Dump of assembler code for function phase\_defused:

=> 0x0000000000401651 <+0>: sub $0x68,%rsp

0x0000000000401655 <+4>: mov $0x1,%edi

0x000000000040165a <+9>: callq 0x4013de <send\_msg>

0x000000000040165f <+14>: cmpl $0x6,0x203526(%rip) # 0x604b8c <num\_input\_strings>

0x0000000000401666 <+21>: jne 0x4016d5 <phase\_defused+132>

0x0000000000401668 <+23>: lea 0x10(%rsp),%r8

0x000000000040166d <+28>: lea 0x8(%rsp),%rcx

0x0000000000401672 <+33>: lea 0xc(%rsp),%rdx

0x0000000000401677 <+38>: mov $0x4026ed,%esi

(gdb) x/s $esi

0x4026ed: "%d %d %s" //ooo need to add a string to phase 4

0x000000000040167c <+43>: mov $0x604c90,%edi

(gdb) x/s $edi

0x604c90 <input\_strings+240>: "10 37" //WOW that’s phase 4

0x0000000000401681 <+48>: mov $0x0,%eax

0x0000000000401686 <+53>: callq 0x400b50 <\_\_isoc99\_sscanf@plt>

0x000000000040168b <+58>: cmp $0x3,%eax //eax=2

0x000000000040168e <+61>: jne 0x4016c1 <phase\_defused+112>

0x0000000000401690 <+63>: mov $0x4026f6,%esi

(gdb) x/s $esi

0x4026f6: "DrEvil"

0x0000000000401695 <+68>: lea 0x10(%rsp),%rdi

0x000000000040169a <+73>: callq 0x4012df <strings\_not\_equal>

0x000000000040169f <+78>: test %eax,%eax

0x00000000004016a1 <+80>: jne 0x4016c1 <phase\_defused+112> since they were equal i continued and were able to get into the secret phase

0x00000000004016a3 <+82>: mov $0x402568,%edi

0x00000000004016a8 <+87>: callq 0x400a90 <puts@plt>

0x00000000004016ad <+92>: mov $0x402590,%edi

0x00000000004016b2 <+97>: callq 0x400a90 <puts@plt>

0x00000000004016b7 <+102>: mov $0x0,%eax

0x00000000004016bc <+107>: callq 0x4011f5 <secret\_phase>

0x00000000004016c1 <+112>: mov $0x4025c8,%edi

0x00000000004016c6 <+117>: callq 0x400a90 <puts@plt>

0x00000000004016cb <+122>: mov $0x4025f8,%edi

0x00000000004016d0 <+127>: callq 0x400a90 <puts@plt>

0x00000000004016d5 <+132>: add $0x68,%rsp

0x00000000004016d9 <+136>: retq

Dump of assembler code for function secret\_phase:

=> 0x00000000004011f5 <+0>: push %rbx

0x00000000004011f6 <+1>: callq 0x40152b <read\_line> //oooo this requires a number input

0x00000000004011fb <+6>: mov $0xa,%edx

0x0000000000401200 <+11>: mov $0x0,%esi

0x0000000000401205 <+16>: mov %rax,%rdi

0x0000000000401208 <+19>: callq 0x400b30 <strtol@plt>

0x000000000040120d <+24>: mov %rax,%rbx

0x0000000000401210 <+27>: lea **-0x1(%rax)**,%eax // My input into eax

0x0000000000401213 <+30>: cmp $0x3e8,%eax //0x3e8=1000 decimal

0x0000000000401218 <+35>: jbe 0x40121f <secret\_phase+42> //hence my input need to be below 1000

0x000000000040121a <+37>: callq 0x4014b6 <explode\_bomb>

0x000000000040121f <+42>: mov %ebx,%esi //here rax= my input - 1 and rbx=my input

0x0000000000401221 <+44>: mov $0x604100,%edi

(gdb) x/w $edi

0x604100 <n1>: 36 //FIRST Node

0x0000000000401226 <+49>: callq 0x4011b7 <fun7>

0x000000000040122b <+54>: cmp $0x3,%eax //the answer from fun7 needs to be 3, if it’s 3 then that means your input is correct.

0x000000000040122e <+57>: je 0x401235 <secret\_phase+64>

0x0000000000401230 <+59>: callq 0x4014b6 <explode\_bomb>

0x0000000000401235 <+64>: mov $0x402420,%edi

0x000000000040123a <+69>: callq 0x400a90 <puts@plt>

0x000000000040123f <+74>: callq 0x401651 <phase\_defused>

0x0000000000401244 <+79>: pop %rbx

0x0000000000401245 <+80>: retq

End of assembler dump.

Dump of assembler code for function fun7:

=> 0x00000000004011b7 <+0>: sub $0x8,%rsp

0x00000000004011bb <+4>: test %rdi,%rdi

0x00000000004011be <+7>: je 0x4011eb <fun7+52>

0x00000000004011c0 <+9>: mov (%rdi),%edx

0x00000000004011c2 <+11>: cmp %esi,%edx //edx=36 //esi=my input

0x00000000004011c4 <+13>: jle 0x4011d3 <fun7+28>

0x00000000004011c6 <+15>: mov 0x8(%rdi),%rdi

0x00000000004011ca <+19>: callq 0x4011b7 <fun7>

0x00000000004011cf <+24>: add %eax,%eax

0x00000000004011d1 <+26>: jmp 0x4011f0 <fun7+57>

0x00000000004011d3 <+28>: mov $0x0,%eax

0x00000000004011d8 <+33>: cmp %esi,%edx //your input shouldn’t be 36

0x00000000004011da <+35>: je 0x4011f0 <fun7+57>

0x00000000004011dc <+37>: mov 0x10(%rdi),%rdi

0x00000000004011e0 <+41>:  **callq 0x4011b7 <fun7>**

0x00000000004011e5 <+46>: lea 0x1(%rax,%rax,1),%eax

0x00000000004011e9 <+50>: jmp 0x4011f0 <fun7+57>

0x00000000004011eb <+52>: mov $0xffffffff,%eax

0x00000000004011f0 <+57>: add $0x8,%rsp

0x00000000004011f4 <+61>: retq

End of assembler dump.

gdb) x/32 0x604100

**0x604100 <n1>: 0x00000024** 0x00000000 0x00604120 0x00000000

0x604110 <n1+16>: 0x00604140 0x00000000 0x00000000 0x00000000

**0x604120 <n21>: 0x00000008**  0x00000000 0x006041a0 0x00000000

0x604130 <n21+16>: 0x00604160 0x00000000 0x00000000 0x00000000

**0x604140 <n22>: 0x00000032**  0x00000000 0x00604180 0x00000000

0x604150 <n22+16>: 0x006041c0 0x00000000 0x00000000 0x00000000

**0x604160 <n32>: 0x00000016**  0x00000000 0x00604280 0x00000000

0x604170 <n32+16>: 0x00604240 0x00000000 0x00000000 0x00000000

**(gdb)**

**0x604180 <n33>: 0x0000002d**

(gdb)

0x604184 <n33+4>: 0x00000000

(gdb)

0x604188 <n33+8>: 0x006041e0

(gdb)

0x60418c <n33+12>: 0x00000000

(gdb)

0x604190 <n33+16>: 0x006042a0

(gdb)

0x604194 <n33+20>: 0x00000000

**gdb)**

**0x6041a0 <n31>: 0x00000006**

(gdb)

0x6041a4 <n31+4>: 0x00000000

(gdb)

0x6041a8 <n31+8>: 0x00604200

(gdb)

0x6041ac <n31+12>: 0x00000000

(gdb)

0x6041b0 <n31+16>: 0x00604260

(gdb)

0x6041b4 <n31+20>: 0x00000000

**(gdb)**

**0x6041c0 <n34>: 0x0000006b**

(gdb)

0x6041c4 <n34+4>: 0x00000000

(gdb)

0x6041c8 <n34+8>: 0x00604220

(gdb)

0x6041cc <n34+12>: 0x00000000

(gdb)

0x6041d0 <n34+16>: 0x006042c0

(gdb)

0x6041d4 <n34+20>: 0x00000000

**(gdb)**

**0x6041e0 <n45>: 0x00000028**

(gdb)

0x6041e4 <n45+4>: 0x00000000

(gdb)

0x6041e8 <n45+8>: 0x00000000

(gdb)

0x6041ec <n45+12>: 0x00000000

(gdb)

0x6041f0 <n45+16>: 0x00000000

(gdb)

0x6041f4 <n45+20>: 0x00000000

**(gdb)**

**0x604200 <n41>: 0x00000001**

(gdb)

0x604204 <n41+4>: 0x00000000

(gdb)

0x604208 <n41+8>: 0x00000000

(gdb)

0x60420c <n41+12>: 0x00000000

(gdb)

0x604210 <n41+16>: 0x00000000

(gdb)

0x604214 <n41+20>: 0x00000000

**(gdb)**

**0x604220 <n47>: 0x00000063**

(gdb)

0x604224 <n47+4>: 0x00000000

(gdb)

0x604228 <n47+8>: 0x00000000

(gdb)

0x60422c <n47+12>: 0x00000000

(gdb)

0x604230 <n47+16>: 0x00000000

(gdb)

0x604234 <n47+20>: 0x00000000

**(gdb)**

**0x604240 <n44>: 0x00000023**

(gdb)

0x604244 <n44+4>: 0x00000000

(gdb)

0x604248 <n44+8>: 0x00000000

(gdb)

0x60424c <n44+12>: 0x00000000

(gdb)

0x604250 <n44+16>: 0x00000000

(gdb)

0x604254 <n44+20>: 0x00000000

**(gdb)**

**0x604260 <n42>: 0x00000007**

**(gdb)**

**0x604280 <n43>: 0x00000014**

**(gdb)**

**0x6042a0 <n46>: 0x0000002f**

**(gdb)**

**0x6042c0 <n48>: 0x000003e9**