2022\_Spring System-Programming

**Assignment #2**

20191128 Jian Park

**[ Phase\_1 ]**

**﹅ Disassemble Code:**

0000000000400e63 <phase\_1>:

400e63: 48 83 ec 08 sub $0x8,%rsp

400e67: be 70 22 40 00 mov $0x402270,%esi

400e6c: e8 21 04 00 00 callq 401292 <strings\_not\_equal>

400e71: 85 c0 test %eax,%eax

400e73: 75 05 jne 400e7a <phase\_1+0x17>

400e75: 48 83 c4 08 add $0x8,%rsp

400e79: c3 retq

400e7a: e8 10 05 00 00 callq 40138f <explode\_bomb>

400e7f: eb f4 jmp 400e75 <phase\_1+0x12>

I explored in backward approach from the address 400e7a that called the bomb-destructing function, expode\_bomb. 400e7a is called by 400e73. And 400e73 is the instruction ‘jne’ to jump to the corresponding address if the condition code ZE is 0.

At 400e71, change the condition code by ‘test’ instruction the return value of strings\_not\_equal, $eax (ZF = (Src1&src2)==0). So, it means if %eax is false, it will be jump.

At 400e67, the value stored in the address 0x402270 of memory is stored in %esi by instruction ‘mov’.

To sum it up, phase\_1 is a function that if input is not same with the value stored at the address 0x402270, it will explode bomb. The value stored in 0x402270 could be confirmed using the command 'x/s 0x402270', and the result was 'The future will be better tomorrow'.

텍스트이(가) 표시된 사진

자동 생성된 설명

**➜** So, the input to dismantle this bomb is **'The future will be better tomorrow.'**

**[ Phase\_2 ]**

**﹅ Disassemble Code:**

0000000000400e81 <phase\_2>:

400e81: 53 push %rbx

400e82: 48 83 ec 20 sub $0x20,%rsp

400e86: 48 89 e6 mov %rsp,%rsi

400e89: e8 23 05 00 00 callq 4013b1 <read\_six\_numbers>

400e8e: 83 3c 24 00 cmpl $0x0,(%rsp)

400e92: 78 07 js 400e9b <phase\_2+0x1a>

400e94: bb 01 00 00 00 mov $0x1,%ebx

400e99: eb 11 jmp 400eac <phase\_2+0x2b>

400e9b: e8 ef 04 00 00 callq 40138f <explode\_bomb>

400ea0: eb f2 jmp 400e94 <phase\_2+0x13>

400ea2: 48 83 c3 01 add $0x1,%rbx

400ea6: 48 83 fb 06 cmp $0x6,%rbx

400eaa: 74 12 je 400ebe <phase\_2+0x3d>

400eac: 89 d8 mov %ebx,%eax

400eae: 03 44 9c fc add -0x4(%rsp,%rbx,4),%eax

400eb2: 39 04 9c cmp %eax,(%rsp,%rbx,4)

400eb5: 74 eb je 400ea2 <phase\_2+0x21>

400eb7: e8 d3 04 00 00 callq 40138f <explode\_bomb>

400ebc: eb e4 jmp 400ea2 <phase\_2+0x21>

400ebe: 48 83 c4 20 add $0x20,%rsp

400ec2: 5b pop %rbx

400ec3: c3 retq

00000000004013b1 <read\_six\_numbers>:

4013b1: 48 83 ec 08 sub $0x8,%rsp

4013b5: 48 89 f2 mov %rsi,%rdx

4013b8: 48 8d 4e 04 lea 0x4(%rsi),%rcx

4013bc: 48 8d 46 14 lea 0x14(%rsi),%rax

4013c0: 50 push %rax

4013c1: 48 8d 46 10 lea 0x10(%rsi),%rax

4013c5: 50 push %rax

4013c6: 4c 8d 4e 0c lea 0xc(%rsi),%r9

4013ca: 4c 8d 46 08 lea 0x8(%rsi),%r8

4013ce: be 43 24 40 00 mov $0x402443,%esi

4013d3: b8 00 00 00 00 mov $0x0,%eax

4013d8: e8 e3 f7 ff ff callq 400bc0 <\_\_isoc99\_sscanf@plt>

4013dd: 48 83 c4 10 add $0x10,%rsp

4013e1: 83 f8 05 cmp $0x5,%eax

4013e4: 7e 05 jle 4013eb <read\_six\_numbers+0x3a>

4013e6: 48 83 c4 08 add $0x8,%rsp

4013ea: c3 retq

4013eb: e8 9f ff ff ff callq 40138f <explode\_bomb>

Set breakpoint at phase\_2 and disassembled. In phase\_2, there were exist two instructions to call expode\_bomb, and there was also exist instructions to call read\_six\_number. So, I guess it’s get 6 numbers as input.

텍스트이(가) 표시된 사진

자동 생성된 설명

To confirm this, I used the command ‘x/s 0x402443’ to check the string stored in 0x402443 and found that '%d %d %d %d %d'. So, my guess is correct that it would be the function that receives six integers as input.

텍스트이(가) 표시된 사진

자동 생성된 설명

Set breakpoint at 0x400e8 where the address that sorted command immediately after the read\_six\_number call. And I entered ‘6 6 6 6 6’, and I tried to execute the instruction one by one using ‘stepi’ command. Then, I found second bomb(400ebc) was occurred.

The results of the analysis of the two bombs in phase\_2 are as follows

**1. at 400e9b**

By 400e92 it moves to a 400e9b that calls expode\_bomb.

The ‘js’ is an instruction that if condition code SF is 1 it would be jump to the corresponding address. In other words, if it is negative, it jumps.

The ‘cmpl’ at 400e8e is an instruction that changes the condition (SF = (src1 – src2) < 0)

**➜ If ($0x0 – (%rsp)) < 0,** it will be jump

**2. at 400eb7**

If you do not jump from 400eb5, 400eb7 will be executed, which calls the bomb. The ‘je’ at 400eb5 is an instruction that if condition code ZF is 1 it would be jump to the corresponding address. The ‘cmp’ at 400eb2 is an instruction that changes the condition code (ZF = (src1==src2))

**➜ If %eax == (%rsp, %rbx, 4),** it wil be jump.

**So, %eax must be same with (%rsp, %rbx, 4).**

Let input array is a[6]. Then it means input array a[] must be satisfy that

**: a[i] + i == a[i+1] (when i: 0 ~ 4)**

텍스트이(가) 표시된 사진

자동 생성된 설명

➜ So the input to dismantle this bomb is **‘1 2 4 7 11 16’**

**[ Phase\_3 ]**

**﹅ Disassemble Code:**

0000000000400ec4 <phase\_3>:

400ec4: 48 83 ec 18 sub $0x18,%rsp

400ec8: 48 8d 4c 24 08 lea 0x8(%rsp),%rcx

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

400ed2: be 4f 24 40 00 mov $0x40244f,%esi

400ed7: b8 00 00 00 00 mov $0x0,%eax

400edc: e8 df fc ff ff callq 400bc0 <\_\_isoc99\_sscanf@plt>

400ee1: 83 f8 01 cmp $0x1,%eax

400ee4: 7e 12 jle 400ef8 <phase\_3+0x34>

400ee6: 83 7c 24 0c 07 cmpl $0x7,0xc(%rsp)

400eeb: 77 43 ja 400f30 <phase\_3+0x6c>

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

400ef1: ff 24 c5 c0 22 40 00 jmpq \*0x4022c0(,%rax,8)

400ef8: e8 92 04 00 00 callq 40138f <explode\_bomb>

400efd: eb e7 jmp 400ee6 <phase\_3+0x22>

400eff: b8 cc 00 00 00 mov $0xcc,%eax

400f04: eb 3b jmp 400f41 <phase\_3+0x7d>

400f06: b8 ac 02 00 00 mov $0x2ac,%eax

400f0b: eb 34 jmp 400f41 <phase\_3+0x7d>

400f0d: b8 9a 02 00 00 mov $0x29a,%eax

400f12: eb 2d jmp 400f41 <phase\_3+0x7d>

400f14: b8 e7 00 00 00 mov $0xe7,%eax

400f19: eb 26 jmp 400f41 <phase\_3+0x7d>

400f1b: b8 c6 00 00 00 mov $0xc6,%eax

400f20: eb 1f jmp 400f41 <phase\_3+0x7d>

400f22: b8 b2 00 00 00 mov $0xb2,%eax

400f27: eb 18 jmp 400f41 <phase\_3+0x7d>

400f29: b8 7d 00 00 00 mov $0x7d,%eax

400f2e: eb 11 jmp 400f41 <phase\_3+0x7d>

400f30: e8 5a 04 00 00 callq 40138f <explode\_bomb>

400f35: b8 00 00 00 00 mov $0x0,%eax

400f3a: eb 05 jmp 400f41 <phase\_3+0x7d>

400f3c: b8 26 02 00 00 mov $0x226,%eax

400f41: 39 44 24 08 cmp %eax,0x8(%rsp)

400f45: 74 05 je 400f4c <phase\_3+0x88>

400f47: e8 43 04 00 00 callq 40138f <explode\_bomb>

400f4c: 48 83 c4 18 add $0x18,%rsp

400f50: c3 retq

텍스트이(가) 표시된 사진

자동 생성된 설명

Set breakpoint at phase\_3. Likewise phase\_2, I used the command ‘x/s 0x40244f’ to check the string stored in 0x40244f and found that '%d %d'. In fact, we can infer by instructions of 400ec8 & 400ecd without using the above command. The first argument is stored at 0xc(%rsp), and the second argument is at 0x8(%rsp).

The ‘cmp’ and ‘jle’ instructions at 400ee1 and 400ee4, it means if %eax is less than or equal to $0x1, jump to the corresponding address that calls expode\_bomb. **So, if number of input is less than or equal to 1, it explodes!**

The ‘cmpl’ and ‘ja’ instructions at 400ee6 and 400eeb, it means if 0xc(%rsp) (first arguments) is greater than 0x7, jump to the to the corresponding address(400f30) that calls expode\_bomb.

**So, first arguments should be less than or equal to 7.**

텍스트이(가) 표시된 사진

자동 생성된 설명

The ‘mov’ instruction at 400eed, the value of first arguments is stored at %eax. And it jump to \*0x4022c0(,%rax,8). But I don’t know what is stored at this address. So, And I entered ‘2 3’, and I tried to execute the instruction one by one using ‘stepi’ command. Before this. I found it jump to 0x400f06.

The ‘mov’ instruction at 400f06, 0x2ac = 684 is stored at %eax. Before executing 400f0b, it jump to 400f41. The ‘cmp’ and ‘je’ instructions at 400f41 and 400f45, it means if 0x8(%rsp) (second arguments) is equal to %eax (==684), jump to the corresponding address (400f4c) that avoid calls expode\_bomb. **So, it must be same.**

텍스트이(가) 표시된 사진

자동 생성된 설명

➜ So the input to dismantle this bomb is **‘2 684’**

**[ Phase\_4 ]**

**﹅ Disassemble Code:**

0000000000400f85 <phase\_4>:

400f85: 48 83 ec 18 sub $0x18,%rsp

400f89: 48 8d 4c 24 08 lea 0x8(%rsp),%rcx

400f8e: 48 8d 54 24 0c lea 0xc(%rsp),%rdx

400f93: be 4f 24 40 00 mov $0x40244f,%esi

400f98: b8 00 00 00 00 mov $0x0,%eax

400f9d: e8 1e fc ff ff callq 400bc0 <\_\_isoc99\_sscanf@plt>

400fa2: 83 f8 02 cmp $0x2,%eax

400fa5: 75 07 jne 400fae <phase\_4+0x29>

400fa7: 83 7c 24 0c 0e cmpl $0xe,0xc(%rsp)

400fac: 76 05 jbe 400fb3 <phase\_4+0x2e>

400fae: e8 dc 03 00 00 callq 40138f <explode\_bomb>

400fb3: ba 0e 00 00 00 mov $0xe,%edx

400fb8: be 00 00 00 00 mov $0x0,%esi

400fbd: 8b 7c 24 0c mov 0xc(%rsp),%edi

400fc1: e8 8b ff ff ff callq 400f51 <func4>

400fc6: 83 f8 1b cmp $0x1b,%eax

400fc9: 75 07 jne 400fd2 <phase\_4+0x4d>

400fcb: 83 7c 24 08 1b cmpl $0x1b,0x8(%rsp)

400fd0: 74 05 je 400fd7 <phase\_4+0x52>

400fd2: e8 b8 03 00 00 callq 40138f <explode\_bomb>

400fd7: 48 83 c4 18 add $0x18,%rsp

400fdb: c3 retq

텍스트이(가) 표시된 사진

자동 생성된 설명

Set breakpoint at phase\_4. I used the command ‘x/s 0x40244f’ to check the string stored in 0x40244f and found that '%d %d'. The first argument is stored at 0xc(%rsp), and the second argument is at 0x8(%rsp).

The ‘cmp’ and ‘jne’ instructions at 400fa2 and 400fa5, it means if %eax is not equal to $0x2, jump to the corresponding address that calls expode\_bomb. **So, if number of inputs is not equal to 2, it explodes!**

The ‘cmpl’ and ‘jbe’ instructions at 400fa7 and 400fac, it means if 0xc(%rsp) (first arguments) is less than or equal to $0xe(==14), jump to the corresponding address (400fb3) that avoid calls expode\_bomb. **So, if first argument is greater than 14, it explodes!**

And it stored 14 at %edx, stored 0 at %esi, stored first argument’s value at %edi.

Then call function ‘func4’, and compare return value with 0x1b. The ‘cmp’ and ‘jne’ instructions at 400fc6 and 400fc9, it means if %eax (return value of ‘func4’) is not equal to $0x1b (==27), jump to the corresponding address that calls expode\_bomb. **So, ‘func4’must return 27.**

The ‘cmpl’ and ‘je’ instructions at 400fcb and 400fd0, it means if 0x8(%rsp) (second arguments) is equal to $0x1b (==27), jump to the corresponding address (400fd7) that avoid calls expode\_bomb. **So, if second argument is not equal to 27, it explodes!**

0000000000400f51 <func4>:

400f51: 53 push %rbx

400f52: 89 d0 mov %edx,%eax // %eax = %edx

400f54: 29 f0 sub %esi,%eax // %eax = %eax - %esi

400f56: 89 c3 mov %eax,%ebx // %ebx = %eax

400f58: c1 eb 1f shr $0x1f,%ebx // %ebx = %ebx >> 31 (only sign)

400f5b: 01 c3 add %eax,%ebx // %ebx = %ebx + %eax

400f5d: d1 fb sar %ebx // %ebx = %ebx >> 1 (%ebx/2)

400f5f: 01 f3 add %esi,%ebx // %ebx = %ebx + %esi

400f61: 39 fb cmp %edi,%ebx

400f63: 7f 08 jg 400f6d <func4+0x1c> // if %ebx > %edi, jump

400f65: 39 fb cmp %edi,%ebx

400f67: 7c 10 jl 400f79 <func4+0x28> // else if %ebx < %edi , jump

400f69: 89 d8 mov %ebx,%eax // else (%ebx == %edi), return

400f6b: 5b pop %rbx

400f6c: c3 retq // return %eax (== %ebx)

400f6d: 8d 53 ff lea -0x1(%rbx),%edx // %edx = (%rdx) - 1

400f70: e8 dc ff ff ff callq 400f51 <func4> // recursive

400f75: 01 c3 add %eax,%ebx

400f77: eb f0 jmp 400f69 <func4+0x18> // recursive

400f79: 8d 73 01 lea 0x1(%rbx),%esi // %esi = (%rdx) + 1

400f7c: e8 d0 ff ff ff callq 400f51 <func4>

400f81: 01 c3 add %eax,%ebx

400f83: eb e4 jmp 400f69 <func4+0x18> // %ebx = %ebx + %eax

This is disassembled code of ‘func4’. And I added comments to some of the lines.

Interpret each part,

**✘ [ ~ 400f5f ] :**

If %edx – %esi <0, ebx = (edx + esi)/2 + 0.5.

Else ebx = (edx + esi)/2

**✘ [ 400f61 ~ ] :**

Ⅰ. If %ebx > %edi

%edx = %rbx – 1, and call ‘func4’.

Return (the return value of the recursive call) + %ebx

Ⅱ. else if %ebx < %edi

%edx = %rbx + 1, and call ‘func4’.

Return (the return value of the recursive call) + %ebx

Ⅲ else ( case of %ebx == %edi )

return %ebx

To sum it up, ‘func4’ is a function that returns the sum of the visited values while binary searching to find value of %edi in the range [%edx, %esi].

To this value to be 27 (target value),

[14,0] 7 (20) ➜ [14,8] 11 (9) ➜ [10,8] 9 (0) [range] mid (target value - mid)

텍스트이(가) 표시된 사진

자동 생성된 설명

➜ So the input to dismantle this bomb is **‘9 27’**

**[ Phase\_5 ]**

**﹅ Disassemble Code:**

0000000000400fdc <phase\_5>:

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

400fe0: 48 8d 4c 24 08 lea 0x8(%rsp),%rcx

400fe5: 48 8d 54 24 0c lea 0xc(%rsp),%rdx

400fea: be 4f 24 40 00 mov $0x40244f,%esi

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

400ff4: e8 c7 fb ff ff callq 400bc0 <\_\_isoc99\_sscanf@plt>

400ff9: 83 f8 01 cmp $0x1,%eax

400ffc: 7e 4a jle 401048 <phase\_5+0x6c>

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

401002: 83 e0 0f and $0xf,%eax

401005: 89 44 24 0c mov %eax,0xc(%rsp)

401009: 83 f8 0f cmp $0xf,%eax

40100c: 74 30 je 40103e <phase\_5+0x62>

40100e: b9 00 00 00 00 mov $0x0,%ecx

401013: ba 00 00 00 00 mov $0x0,%edx

401018: 83 c2 01 add $0x1,%edx

40101b: 48 98 cltq

40101d: 8b 04 85 00 23 40 00 mov 0x402300(,%rax,4),%eax

401024: 01 c1 add %eax,%ecx

401026: 83 f8 0f cmp $0xf,%eax

401029: 75 ed jne 401018 <phase\_5+0x3c>

40102b: c7 44 24 0c 0f 00 00 movl $0xf,0xc(%rsp)

401032: 00

401033: 83 fa 0f cmp $0xf,%edx

401036: 75 06 jne 40103e <phase\_5+0x62>

401038: 39 4c 24 08 cmp %ecx,0x8(%rsp)

40103c: 74 05 je 401043 <phase\_5+0x67>

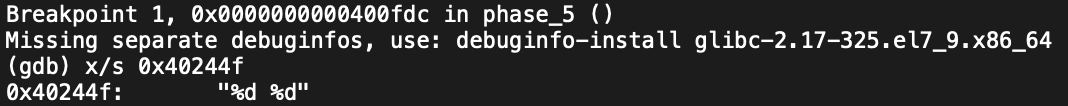
40103e: e8 4c 03 00 00 callq 40138f <explode\_bomb>

401043: 48 83 c4 18 add $0x18,%rsp

401047: c3 retq

401048: e8 42 03 00 00 callq 40138f <explode\_bomb>

40104d: eb af jmp 400ffe <phase\_5+0x22>



Set breakpoint at phase\_5. I used the command ‘x/s 0x40244f’ to check the string stored in 0x40244f and found that '%d %d'. The first argument is stored at 0xc(%rsp), and the second argument is at 0x8(%rsp). The ‘cmp’ and ‘jle’ instructions at 400ff9 and 400ffc, it means if %eax is less than or equal to $0x1, jump to the corresponding address that calls expode\_bomb. **So, if number of inputs is less than or equal to 1, it explodes!**

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

401002: 83 e0 0f and $0xf,%eax

401005: 89 44 24 0c mov %eax,0xc(%rsp)

401009: 83 f8 0f cmp $0xf,%eax

40100c: 74 30 je 40103e <phase\_5+0x62>

Store first argument at %eax, and perform ‘and’ operation with 0xf(= 15 = 11112). Then only the least 4 bits are left, and it is the same as the remainder when divided by 16. And restore it at 0xc(%rsp) that address of first argument. Compare %eax with 0xf(= 15) if the values are same jump to address that calls expode\_bomb.

**So, if the remainder when first argument divided by 16 is equal to 15, it explodes!**

40100e: b9 00 00 00 00 mov $0x0,%ecx

401013: ba 00 00 00 00 mov $0x0,%edx

401018: 83 c2 01 add $0x1,%edx

40101b: 48 98 cltq

40101d: 8b 04 85 00 23 40 00 mov 0x402300(,%rax,4),%eax

401024: 01 c1 add %eax,%ecx

401026: 83 f8 0f cmp $0xf,%eax

401029: 75 ed jne 401018 <phase\_5+0x3c>

Stored 0x0 at %eax and %edx. And below 401018 loop statement. %edx is number of iterations.

do while(%eax != 15){

%edx = %edx + 1

sign expands %eax and stores it in %rax. (instruction ‘cltq’)

%eax = 0x402300(,%rax,4) = 4 \* (%rax) + 0x402300

%ecx = %ecx + %eax

}

40102b: c7 44 24 0c 0f 00 00 movl $0xf,0xc(%rsp)

401032: 00

401033: 83 fa 0f cmp $0xf,%edx

401036: 75 06 jne 40103e <phase\_5+0x62>

401038: 39 4c 24 08 cmp %ecx,0x8(%rsp)

40103c: 74 05 je 401043 <phase\_5+0x67>

40103e: e8 4c 03 00 00 callq 40138f <explode\_bomb>

401043: 48 83 c4 18 add $0x18,%rsp

401047: c3 retq

If %eax is equal to 15, so escapes the loop statement, store 15 at 0xc (%rsp) that address first arguments stored.

Compared to %edx with 15, if %edx is not equal to 15, jump to address that calls expode\_bomb.

If you avoid a bomb because %edx is eqaual to 15, compare the second arguments with %ecx and if the two values are equal jump to the corresponding address (401043) that avoid calls expode\_bomb. If else, it exploded!

So, %edx is equal to 15 and second argument is equal to %ecx.

%rsp = %rsp + 24

Done.

What value will be stored at 0x402300 in 0x402300(,%rax,4)?



Can confirm using the command 'x/g 0x402300', and the result was array. So, it is move the index as %rax in the array and store value of a[%rax] at %eax. I can guess that the length of the array is 16 because the remainder divided by 16 should be less than 15.

텍스트이(가) 표시된 사진

자동 생성된 설명

It can confirm using the command 'x/17w 0x402300', (Just in case, I confirmed 17 values.)

To sum it up, starting at a[first argument] and the value of the array positioned when moved 15 times (15 iterations) is must be 15. And second argument is sum of the value that stored in past array position.

Then we just decide starting index, and it can find by explore backward.

Modify given array that add index to easy find index of array.

0x402300 <array.3236>: 10 [0] 2 [1] 14 [2] 7 [3]

0x402310 <array.3236+16>: 8 [4] 12 [5] 15 [6] 11 [7]

0x402320 <array.3236+32>: 0 [8] 4 [9] 1 [10] 13 [11]

0x402330 <array.3236+48>: 3 [12] 9 [13] 6 [14] 5 [15]

15 [6] ➜ 6 [14] ➜ 14 [2] ➜ 2 [1] ➜ 1 [10] ➜ 10 [0] ➜ 0 [8] ➜ 8 [4] ➜ 4 [9] ➜ 9[13] ➜ 13[11] ➜ 11[7] ➜ 7[3] ➜ 3[12] ➜ 12[5] value [index]

We start at 5, and sum of value is 115.

텍스트이(가) 표시된 사진

자동 생성된 설명

➜ So the input to dismantle this bomb is **‘5 115’**

**[ Phase\_6 ]**

**﹅ Disassemble Code:**

000000000040104f <phase\_6>:

40104f: 41 56 push %r14

401051: 41 55 push %r13

401053: 41 54 push %r12

401055: 55 push %rbp

401056: 53 push %rbx

401057: 48 83 ec 50 sub $0x50,%rsp

40105b: 48 8d 74 24 30 lea 0x30(%rsp),%rsi

401060: e8 4c 03 00 00 callq 4013b1 <read\_six\_numbers>

401065: 4c 8d 64 24 30 lea 0x30(%rsp),%r12

40106a: 4d 89 e5 mov %r12,%r13

40106d: 41 be 00 00 00 00 mov $0x0,%r14d

401073: eb 26 jmp 40109b <phase\_6+0x4c>

401075: e8 15 03 00 00 callq 40138f <explode\_bomb>

40107a: eb 2e jmp 4010aa <phase\_6+0x5b>

40107c: 83 c3 01 add $0x1,%ebx

40107f: 83 fb 05 cmp $0x5,%ebx

401082: 7f 13 jg 401097 <phase\_6+0x48>

401084: 48 63 c3 movslq %ebx,%rax

401087: 8b 44 84 30 mov 0x30(%rsp,%rax,4),%eax

40108b: 39 45 00 cmp %eax,0x0(%rbp)

40108e: 75 ec jne 40107c <phase\_6+0x2d>

401090: e8 fa 02 00 00 callq 40138f <explode\_bomb>

401095: eb e5 jmp 40107c <phase\_6+0x2d>

401097: 49 83 c5 04 add $0x4,%r13

40109b: 4c 89 ed mov %r13,%rbp

40109e: 41 8b 45 00 mov 0x0(%r13),%eax

4010a2: 83 e8 01 sub $0x1,%eax

4010a5: 83 f8 05 cmp $0x5,%eax

4010a8: 77 cb ja 401075 <phase\_6+0x26>

4010aa: 41 83 c6 01 add $0x1,%r14d

4010ae: 41 83 fe 06 cmp $0x6,%r14d

4010b2: 74 05 je 4010b9 <phase\_6+0x6a>

4010b4: 44 89 f3 mov %r14d,%ebx

4010b7: eb cb jmp 401084 <phase\_6+0x35>

4010b9: 49 8d 4c 24 18 lea 0x18(%r12),%rcx

4010be: ba 07 00 00 00 mov $0x7,%edx

4010c3: 89 d0 mov %edx,%eax

4010c5: 41 2b 04 24 sub (%r12),%eax

4010c9: 41 89 04 24 mov %eax,(%r12)

4010cd: 49 83 c4 04 add $0x4,%r12

4010d1: 4c 39 e1 cmp %r12,%rcx

4010d4: 75 ed jne 4010c3 <phase\_6+0x74>

4010d6: be 00 00 00 00 mov $0x0,%esi

4010db: eb 19 jmp 4010f6 <phase\_6+0xa7>

4010dd: 48 8b 52 08 mov 0x8(%rdx),%rdx

4010e1: 83 c0 01 add $0x1,%eax

4010e4: 39 c8 cmp %ecx,%eax

4010e6: 75 f5 jne 4010dd <phase\_6+0x8e>

4010e8: 48 89 14 f4 mov %rdx,(%rsp,%rsi,8)

4010ec: 48 83 c6 01 add $0x1,%rsi

4010f0: 48 83 fe 06 cmp $0x6,%rsi

4010f4: 74 15 je 40110b <phase\_6+0xbc>

4010f6: 8b 4c b4 30 mov 0x30(%rsp,%rsi,4),%ecx

4010fa: b8 01 00 00 00 mov $0x1,%eax

4010ff: ba f0 32 60 00 mov $0x6032f0,%edx

401104: 83 f9 01 cmp $0x1,%ecx

401107: 7f d4 jg 4010dd <phase\_6+0x8e>

401109: eb dd jmp 4010e8 <phase\_6+0x99>

40110b: 48 8b 1c 24 mov (%rsp),%rbx

40110f: 48 8b 44 24 08 mov 0x8(%rsp),%rax

401114: 48 89 43 08 mov %rax,0x8(%rbx)

401118: 48 8b 54 24 10 mov 0x10(%rsp),%rdx

40111d: 48 89 50 08 mov %rdx,0x8(%rax)

401121: 48 8b 44 24 18 mov 0x18(%rsp),%rax

401126: 48 89 42 08 mov %rax,0x8(%rdx)

40112a: 48 8b 54 24 20 mov 0x20(%rsp),%rdx

40112f: 48 89 50 08 mov %rdx,0x8(%rax)

401133: 48 8b 44 24 28 mov 0x28(%rsp),%rax

401138: 48 89 42 08 mov %rax,0x8(%rdx)

40113c: 48 c7 40 08 00 00 00 movq $0x0,0x8(%rax)

401143: 00

401144: bd 05 00 00 00 mov $0x5,%ebp

401149: eb 09 jmp 401154 <phase\_6+0x105>

40114b: 48 8b 5b 08 mov 0x8(%rbx),%rbx

40114f: 83 ed 01 sub $0x1,%ebp

401152: 74 11 je 401165 <phase\_6+0x116>

401154: 48 8b 43 08 mov 0x8(%rbx),%rax

401158: 8b 00 mov (%rax),%eax

40115a: 39 03 cmp %eax,(%rbx)

40115c: 7d ed jge 40114b <phase\_6+0xfc>

40115e: e8 2c 02 00 00 callq 40138f <explode\_bomb>

401163: eb e6 jmp 40114b <phase\_6+0xfc>

401165: 48 83 c4 50 add $0x50,%rsp

401169: 5b pop %rbx

40116a: 5d pop %rbp

40116b: 41 5c pop %r12

40116d: 41 5d pop %r13

40116f: 41 5e pop %r14

401171: c3 retq

It has ‘read\_six\_number’ function. So, it’s get 6 interger numbers as input.

401065: 4c 8d 64 24 30 lea 0x30(%rsp),%r12

40106a: 4d 89 e5 mov %r12,%r13

40106d: 41 be 00 00 00 00 mov $0x0,%r14d

401073: eb 26 jmp 40109b <phase\_6+0x4c>

After input, store 0x30(%rsp) at %r12 and %r13, store 0x0 at %r14d. Then jump to 40109b.

40109b: 4c 89 ed mov %r13,%rbp

40109e: 41 8b 45 00 mov 0x0(%r13),%eax

4010a2: 83 e8 01 sub $0x1,%eax

4010a5: 83 f8 05 cmp $0x5,%eax

4010a8: 77 cb ja 401075 <phase\_6+0x26>

Store starting address of argument at %rbp, value of first argument at %eax. Compared with (%eax) - 1 and 5, if (%eax) - 1 is greater than 5, then jump to address that calls expode\_bomb. There for the first arguments is less than or equal 6.

I interpreted each part separately.

There is double loop statement.

**> 4010aa < : Check outer loop condition**

4010aa: 41 83 c6 01 add $0x1,%r14d

4010ae: 41 83 fe 06 cmp $0x6,%r14d

4010b2: 74 05 je 4010b9 <phase\_6+0x6a>

4010b4: 44 89 f3 mov %r14d,%ebx

4010b7: eb cb jmp 401084 <phase\_6+0x35>

%r14d = %r14d + 1

Compared this with 6, if they are same jump to 4010b9

else (not same) store %r14dat %ebx and jump to 401084

%r14d is number of iterations outer loop.

**> 401084 < : Body of outer loop**

401084: 48 63 c3 movslq %ebx,%rax

401087: 8b 44 84 30 mov 0x30(%rsp,%rax,4),%eax

40108b: 39 45 00 cmp %eax,0x0(%rbp)

40108e: 75 ec jne 40107c <phase\_6+0x2d>

401090: e8 fa 02 00 00 callq 40138f <explode\_bomb>

Store value of 0x30(%rsp, %rax, 4) that value moved %rax (%r14d at 4010aa) from %rsp at %eax. So, it is (%rax)th value in array

Compare this with first arguments, if they are not same jump to 40107c, else (same) calls expode\_bomb.

**> 40107c < : Check inner loop condition**

40107c: 83 c3 01 add $0x1,%ebx

40107f: 83 fb 05 cmp $0x5,%ebx

401082: 7f 13 jg 401097 <phase\_6+0x48>

%ebx = %ebx + 1

Compare this with 5, if greater than 5, jump to 401097. Else repeat 401084.

%ebx is number of iterations outer loop.

**> 401097 < : Body of inner loop**

401097: 49 83 c5 04 add $0x4,%r13

40109b: 4c 89 ed mov %r13,%rbp

40109e: 41 8b 45 00 mov 0x0(%r13),%eax

4010a2: 83 e8 01 sub $0x1,%eax

4010a5: 83 f8 05 cmp $0x5,%eax

4010a8: 77 cb ja 401075 <phase\_6+0x26>

4010aa: 41 83 c6 01 add $0x1,%r14d

4010ae: 41 83 fe 06 cmp $0x6,%r14d

4010b2: 74 05 je 4010b9 <phase\_6+0x6a>

4010b4: 44 89 f3 mov %r14d,%ebx

4010b7: eb cb jmp 401084 <phase\_6+0x35>

Add 4 at %rbp to move address 4, so move 1 index in array.

Check if it is in inner loop range. If out of range jump to 401075 that calls expode\_bomb.

Else %r14d = %r14d + 1, and compare this with 6. If they are same jump to 4010b9 (escape outer loop). else %ebx = %r14d and jump to 401084 (body of outer loop)

To sum it up, this is double loop statement that checks if there are equal values in the array.

**> 4010b9 <**

4010b9: 49 8d 4c 24 18 lea 0x18(%r12),%rcx

4010be: ba 07 00 00 00 mov $0x7,%edx

4010c3 …

%rcx = (%r12) + 0x18 (== 32 == 8 \* 4)

%edx = 7

And execute 4010c3.

**> 4010c3 < : Store value of (7 - origin input value) at origin address**

4010c3: 89 d0 mov %edx,%eax

4010c5: 41 2b 04 24 sub (%r12),%eax

4010c9: 41 89 04 24 mov %eax,(%r12)

4010cd: 49 83 c4 04 add $0x4,%r12

4010d1: 4c 39 e1 cmp %r12,%rcx

4010d4: 75 ed jne 4010c3 <phase\_6+0x74>

4010d6: be 00 00 00 00 mov $0x0,%esi

4010db: eb 19 jmp 4010f6 <phase\_6+0xa7>

%eax = %edx = 7 (before 401069)

%eax = %eax – ($r12)

(%r12) = eax

%r12 = %r12 + 0x4, so move 1 index in array.

Compare %r12 with %rcx, if they are not same jump to 4010c3 (repeat), else set %esi to 0 and jump to 4010f6.

**> 4010f6 < : Move in input value array**

4010f6: 8b 4c b4 30 mov 0x30(%rsp,%rsi,4),%ecx

4010fa: b8 01 00 00 00 mov $0x1,%eax

4010ff: ba f0 32 60 00 mov $0x6032f0,%edx

401104: 83 f9 01 cmp $0x1,%ecx

401107: 7f d4 jg 4010dd <phase\_6+0x8e>

401109: eb dd jmp 4010e8 <phase\_6+0x99>

Store value of 0x30(%rsp, %rsi, 4) that value moved %rsi (that set to 0 at 4010c3) from %rsp at %ecx. So, it is (%rsi)th value in array

%eax = 1

%edx = $0x6032f0

If ecx > 1, jump to 4010dd. Else jump to 4010e8

What value will be stored at $0x6032f0?

I can guess that there would be array that has some six values because looping 6 times.

테이블이(가) 표시된 사진

자동 생성된 설명

It can confirm using the command 'x/32w 0x6032f0'.

It was an array containing 6 nodes with the size of each node of 16 bytes and each value is 4 bytes.

**> 4010dd < : move address**

4010dd: 48 8b 52 08 mov 0x8(%rdx),%rdx

4010e1: 83 c0 01 add $0x1,%eax

4010e4: 39 c8 cmp %ecx,%eax

4010e6: 75 f5 jne 4010dd <phase\_6+0x8e>

401038 …

%rdx = (%rdx) + 0x8

%eax = %eax + 1

If %eax is not equal to %ecx, jump to 4010dd (repeat). Else (same) execute 4010e8.

**> 4010e8 < : store value of node and check loop condition**

4010e8: 48 89 14 f4 mov %rdx,(%rsp,%rsi,8)

4010ec: 48 83 c6 01 add $0x1,%rsi

4010f0: 48 83 fe 06 cmp $0x6,%rsi

4010f4: 74 15 je 40110b <phase\_6+0xbc>

4010f6 …

(%rsp, %rsi, 8) = rdx

%rsi = %rsi + 1

If %rsi is 6 (done to loop), jump to 40110b. Else execute 4010f6

**> 40110b < : copy array that stored at %rsp**

40110b: 48 8b 1c 24 mov (%rsp),%rbx

40110f: 48 8b 44 24 08 mov 0x8(%rsp),%rax

401114: 48 89 43 08 mov %rax,0x8(%rbx)

401118: 48 8b 54 24 10 mov 0x10(%rsp),%rdx

40111d: 48 89 50 08 mov %rdx,0x8(%rax)

401121: 48 8b 44 24 18 mov 0x18(%rsp),%rax

401126: 48 89 42 08 mov %rax,0x8(%rdx)

40112a: 48 8b 54 24 20 mov 0x20(%rsp),%rdx

40112f: 48 89 50 08 mov %rdx,0x8(%rax)

401133: 48 8b 44 24 28 mov 0x28(%rsp),%rax

401138: 48 89 42 08 mov %rax,0x8(%rdx)

40113c: 48 c7 40 08 00 00 00 movq $0x0,0x8(%rax)

401143: 00

401144: bd 05 00 00 00 mov $0x5,%ebp

401149: eb 09 jmp 401154 <phase\_6+0x105>

**%rbx[0] = %rsp[0]**

**%**rax = **%**rsp[1]

**%rbx[1] = %rsp[1]**

**%**rdx = **%**rsp[2]

**%**rax[1] = **%**rsp[2]

**%**rax = **%**rsp[3]

**%**rdx[1] = **%**rsp[3]

**%rdx = %rsp[4]**

**%**rax[1] = **%**rsp[4]

**%rax = %rsp[5]**

**%rdx[1] = %rsp[5]**

**%rax[1] = 0**

%ebp = 5

**➜ %rbx [rsp[0], %%rsp[1]] %rdx [%rsp[4], %rsp[5]] %rax [%rsp[5], 0]**

**> 401154 < : comparison of values attached to each other**

401154: 48 8b 43 08 mov 0x8(%rbx),%rax

401158: 8b 00 mov (%rax),%eax

40115a: 39 03 cmp %eax,(%rbx)

40115c: 7d ed jge 40114b <phase\_6+0xfc>

40115e: e8 2c 02 00 00 callq 40138f <explode\_bomb>

401163: eb e6 jmp 40114b <phase\_6+0xfc>

If (%rbx) >= eax = (%rbx) + 8, jump to 40114b.

Else calls expode\_bomb.

So, the preceding value must be greater than or equal to the latter value.

**> 40114b < : check it iterations all done**

40114b: 48 8b 5b 08 mov 0x8(%rbx),%rbx

40114f: 83 ed 01 sub $0x1,%ebp

401152: 74 11 je 401165 <phase\_6+0x116>

401154 …

%rbx = %rbx + 8

If %ebp is 1, jump to 401165. Else execute 401154

**> 401165 < : Done!**

401165: 48 83 c4 50 add $0x50,%rsp

401169: 5b pop %rbx

40116a: 5d pop %rbp

40116b: 41 5c pop %r12

40116d: 41 5d pop %r13

40116f: 41 5e pop %r14

401171: c3 retq

Done!

To sum it up, it is a program that checks whether the value in the node is in descending order when the value of the input array is used as an index.

node name value index pointer to

0x6032f0 <node1>: 518 1 6304512 0

0x603300 <node2>: 654 2 6304528 0

0x603310 <node3>: 767 3 6304544 0

0x603320 <node4>: 626 4 6304560 0

0x603330 <node5>: 606 5 6304576 0

0x603340 <node6>: 136 6 0 0

Sort 518 [1] 654[2] 767[3] 626[4] 606[5] 136[6] with descending order,

result is 767[3] 654[2] 626[4] 606[5] 518[1] 136[6]. value [index]

So the answer should be **‘3 2 4 5 1 6’**

But I enter this, it exploded. Because I forget 4010c3 that store value of (7 - origin input value) at origin address.

텍스트이(가) 표시된 사진

자동 생성된 설명

➜ So the input to dismantle this bomb is **‘4 5 3 2 6 1’**

**! issue**

I have some issue about screenshot.txt. I have done the assignment 6 times, so this file made by combine 6 terminal files. I did the assignment using VScode, but the length of terminal out was too long, so some results (about phase\_1, phase\_6) disappeared before I saved it.

I'm trying to reproduce previous procedure. I've added commands as soon as I can remember. This is indicated with # before the command in ‘screenshot32.txt’.