Comp 201 – Assignment 4 – byildirim19

Phase 1:

```
(gdb) disas
Dump of assembler code for function phase_1:
=> 0x0000555555555204 <+0>:     sub    $0x8,%rsp
   0x0000555555555208 <+4>:     lea    0x17a1(%rip),%rsi        # 0x5555555569b0
   0x000055555555520f <+11>:    callq  0x5555555556bb <strings_not_equal>
   0x0000555555555214 <+16>:    test   %eax,%eax
   0x0000555555555216 <+18>:    jne    0x555555555521d <phase_1+25>
   0x0000555555555218 <+20>:    add    $0x8,%rsp
   0x000055555555521c <+24>:    retq
   0x000055555555521d <+25>:    callq  0x555555555976 <explode_bomb>
   0x0000555555555222 <+30>:    jmp    0x555555555218 <phase_1+20>
End of assembler dump.
(gdb) x/s 0x5555555569b0
0x5555555569b0: "When a problem comes along, you must zip it!"
```

There is a string comparison if we input a string we want it defuses this, if we don't it explodes the bomb. When we get the string at the corresponding address, we have found our first solution.

Phase 2:

```
   0x0000555555555224 <+0>:     push   %rbp
   0x0000555555555225 <+1>:     push   %rbx
   0x0000555555555226 <+2>:     sub    $0x28,%rsp
   0x000055555555522a <+6>:     mov    %fs:0x28,%rax
   0x0000555555555233 <+15>:    mov    %rax,0x18(%rsp)
   0x0000555555555238 <+20>:    xor    %eax,%eax
   0x000055555555523a <+22>:    mov    %rsp,%rsi
   0x000055555555523d <+25>:    callq  0x5555555559b2 <read_six_numbers>
   0x0000555555555242 <+30>:    cmpl   $0x0,(%rsp)
   0x0000555555555246 <+34>:    jne    0x55555555524f <phase_2+43>
   0x0000555555555248 <+36>:    cmpl   $0x1,0x4(%rsp)
   0x000055555555524d <+41>:    je     0x555555555254 <phase_2+48>
   0x000055555555524f <+43>:    callq  0x555555555976 <explode_bomb>
   0x0000555555555254 <+48>:    mov    %rsp,%rbx
   0x0000555555555257 <+51>:    lea    0x10(%rbx),%rbp
   0x000055555555525b <+55>:    jmp    0x555555555266 <phase_2+66>
   0x000055555555525d <+57>:    add    $0x4,%rbx
   0x0000555555555261 <+61>:    cmp    %rbp,%rbx
   0x0000555555555264 <+64>:    je     0x555555555277 <phase_2+83>
   0x0000555555555266 <+66>:    mov    0x4(%rbx),%eax
   0x0000555555555269 <+69>:    add    (%rbx),%eax
=> 0x000055555555526b <+71>:    cmp    %eax,0x8(%rbx)
---Type <return> to continue, or q <return> to quit---
   0x000055555555526e <+74>:    je     0x55555555525d <phase_2+57>
```

We call read six numbers function. When we step into this function we can see that we need 6 numbers. From line +30 we can see that our first number should be 0. From line +36 we see our second number is 1. Then we do "ni" several times until line +71. Then we check the register eax by typing "i r" to find out our next number. We continue typing "ni" to find out untill we find the all other numbers.

Phase 3:

```
=> 0x0000555555555293 <+0>:     sub    $0x18,%rsp
   0x0000555555555297 <+4>:     mov    %fs:0x28,%rax
   0x00005555555552a0 <+13>:    mov    %rax,0x8(%rsp)
   0x00005555555552a5 <+18>:    xor    %eax,%eax
   0x00005555555552a7 <+20>:    lea    0x4(%rsp),%rcx
   0x00005555555552ac <+25>:    mov    %rsp,%rdx
   0x00005555555552af <+28>:    lea    0x19df(%rip),%rsi        # 0x555555556c95
   0x00005555555552b6 <+35>:    callq  0x555555554ee0 <__isoc99_sscanf@plt>
   0x00005555555552bb <+40>:    cmp    $0x1,%eax
   0x00005555555552be <+43>:    jle    0x5555555552dd <phase_3+74>
   0x00005555555552c0 <+45>:    cmpl   $0x7,(%rsp)
   0x00005555555552c4 <+49>:    ja     0x555555555363 <phase_3+208>
   0x00005555555552ca <+55>:    mov    (%rsp),%eax
   0x00005555555552cd <+58>:    lea    0x174c(%rip),%rdx        # 0x555555556a20
   0x00005555555552d4 <+65>:    movslq (%rdx,%rax,4),%rax
   0x00005555555552d8 <+69>:    add    %rdx,%rax
   0x00005555555552db <+72>:    jmpq   *%rax
   0x00005555555552dd <+74>:    callq  0x555555555976 <explode_bomb>
   0x00005555555552e2 <+79>:    jmp    0x5555555552c0 <phase_3+45>
   0x00005555555552e4 <+81>:    mov    $0x72,%eax
---Type <return> to continue, or q <return> to quit---q
Quit
(gdb) x/8d 0x555555556a20
0x555555556a20: -5948   -5941   -5863   -5856
0x555555556a30: -5849   -5842   -5835   -5828
```

It can jump to different locations for number 2 as first input it moves to the corresponding location.

```
(gdb) i r
rax            0xffffffffffffe919       -5863
```

```
   0x00005555555552e9 <+86>:    jmp    0x5555555552f0 <phase_3+93>
   0x00005555555552eb <+88>:    mov    $0x0,%eax
   0x00005555555552f0 <+93>:    sub    $0x31d,%eax
   0x00005555555552f5 <+98>:    add    $0x2db,%eax
   0x00005555555552fa <+103>:   sub    $0x16f,%eax
   0x00005555555552ff <+108>:   add    $0x16f,%eax
   0x0000555555555304 <+113>:   sub    $0x16f,%eax
   0x0000555555555309 <+118>:   add    $0x16f,%eax
   0x000055555555530e <+123>:   sub    $0x16f,%eax
   0x0000555555555313 <+128>:   cmpl   $0x5,(%rsp)
   0x0000555555555317 <+132>:   jg     0x55555555531f <phase_3+140>
=> 0x0000555555555319 <+134>:   cmp    %eax,0x4(%rsp)
   0x000055555555531d <+138>:   je     0x555555555324 <phase_3+145>
```

When we write 2 after doing some operations we come to the line +134. We check rax by typing "i r". Hence, we found our second number as 364.

Phase 4:

```
(gdb) x/s 0x555555556c95
0x555555556c95: "%d %d"
```

First we find the input should be two integers.

```
=> 0x00005555555553a8 <+0>:     sub    $0x18,%rsp
   0x00005555555553ac <+4>:     mov    %fs:0x28,%rax
   0x00005555555553b5 <+13>:    mov    %rax,0x8(%rsp)
   0x00005555555553ba <+18>:    xor    %eax,%eax
   0x00005555555553bc <+20>:    lea    0x4(%rsp),%rcx
   0x00005555555553c1 <+25>:    mov    %rsp,%rdx
   0x00005555555553c4 <+28>:    lea    0x18ca(%rip),%rsi        # 0x555555556c95
   0x00005555555553cb <+35>:    callq  0x555555554ee0 <__isoc99_sscanf@plt>
   0x00005555555553d0 <+40>:    cmp    $0x2,%eax
   0x00005555555553d3 <+43>:    jne    0x5555555553db <phase_4+51>
   0x00005555555553d5 <+45>:    cmpl   $0xe,(%rsp)
   0x00005555555553d9 <+49>:    jbe    0x5555555553e0 <phase_4+56>
   0x00005555555553db <+51>:    callq  0x555555555976 <explode_bomb>
   0x00005555555553e0 <+56>:    mov    $0xe,%edx
   0x00005555555553e5 <+61>:    mov    $0x0,%esi
   0x00005555555553ea <+66>:    mov    (%rsp),%edi
   0x00005555555553ed <+69>:    callq  0x555555555374 <func4>
```

By looking at line +45 we can say that our first input should be less than or equal to 15(0-15 inclusive). Then we call func4.

```
=> 0x0000555555555374 <+0>:     push   %rbx
   0x0000555555555375 <+1>:     mov    %edx,%eax
   0x0000555555555377 <+3>:     sub    %esi,%eax
   0x0000555555555379 <+5>:     mov    %eax,%ebx
   0x000055555555537b <+7>:     shr    $0x1f,%ebx
   0x000055555555537e <+10>:    add    %eax,%ebx
   0x0000555555555380 <+12>:    sar    %ebx
   0x0000555555555382 <+14>:    add    %esi,%ebx
   0x0000555555555384 <+16>:    cmp    %edi,%ebx
   0x0000555555555386 <+18>:    jg     0x555555555390 <func4+28>
   0x0000555555555388 <+20>:    cmp    %edi,%ebx
   0x000055555555538a <+22>:    jl     0x55555555539c <func4+40>
   0x000055555555538c <+24>:    mov    %ebx,%eax
   0x000055555555538e <+26>:    pop    %rbx
   0x000055555555538f <+27>:    retq
   0x0000555555555390 <+28>:    lea    -0x1(%rbx),%edx
   0x0000555555555393 <+31>:    callq  0x555555555374 <func4>
   0x0000555555555398 <+36>:    add    %eax,%ebx
   0x000055555555539a <+38>:    jmp    0x55555555538c <func4+24>
   0x000055555555539c <+40>:    lea    0x1(%rbx),%esi
   0x000055555555539f <+43>:    callq  0x555555555374 <func4>
   0x00005555555553a4 <+48>:    add    %eax,%ebx
---Type <return> to continue, or q <return> to quit---
   0x00005555555553a6 <+50>:    jmp    0x55555555538c <func4+24>
```

This is the code inside func4. I translated it into python code.

```
main.py
1  def func4(edi,esi=0,edx=15):
2      eax = edx
3      eax = eax -esi
4      ebx = eax
5      ebx = ebx >> 31
6      ebx = ebx + eax
7      ebx = ebx >> 1
8      ebx = esi + ebx
9
10     if edi == ebx:
11         return 0
12     if ebx > edi:
13         eax = func4(edi,esi,ebx-1)
14         return ebx + eax
15     if ebx < edi:
16         esi = func4(edi,ebx+1,edx)
17         return ebx + eax
18
19 if __name__ == "__main__":
20     print("First input can be: ")
21     for i in range(16):
22         result = func4(i)
23         if result == 0:
24             print(str(i)+" ")

input
First input can be:
7
```

This way I found the only valid number for first input is 7. Now, we need to find our second input.

```
   0x00005555555553ed <+69>:    callq  0x555555555374 <func4>
   0x00005555555553f2 <+74>:    cmp    $0x7,%eax
   0x00005555555553f5 <+77>:    jne    0x5555555553fe <phase_4+86>
   0x00005555555553f7 <+79>:    cmpl   $0x7,0x4(%rsp)
   0x00005555555553fc <+84>:    je     0x5555555555403 <phase_4+91>
---Type <return> to continue, or q <return> to quit---
   0x00005555555553fe <+86>:    callq  0x555555555976 <explode_bomb>
   0x0000555555555403 <+91>:    mov    0x8(%rsp),%rax
   0x0000555555555408 <+96>:    xor    %fs:0x28,%rax
   0x0000555555555411 <+105>:   jne    0x555555555418 <phase_4+112>
   0x0000555555555413 <+107>:   add    $0x18,%rsp
   0x0000555555555417 <+111>:   retq
   0x0000555555555418 <+112>:   callq  0x555555554e40 <__stack_chk_fail@plt>
```

By looking at line +79 we can say that our second number is 7.

Phase 5:

```
=> 0x000055555555541d <+0>:     push   %rbx
   0x000055555555541e <+1>:     mov    %rdi,%rbx
   0x0000555555555421 <+4>:     callq  0x555555555569e <string_length>
   0x0000555555555426 <+9>:     cmp    $0x6,%eax
   0x0000555555555429 <+12>:    jne    0x555555555545c <phase_5+63>
   0x000055555555542b <+14>:    mov    %rbx,%rax
   0x000055555555542e <+17>:    lea    0x6(%rbx),%rdi
   0x0000555555555432 <+21>:    mov    $0x0,%ecx
   0x0000555555555437 <+26>:    lea    0x1602(%rip),%rsi        # 0x555555556a40
 <array.3417>
   0x000055555555543e <+33>:    movzbl (%rax),%edx
   0x0000555555555441 <+36>:    and    $0xf,%edx
   0x0000555555555444 <+39>:    add    (%rsi,%rdx,4),%ecx
   0x0000555555555447 <+42>:    add    $0x1,%rax
   0x000055555555544b <+46>:    cmp    %rdi,%rax
   0x000055555555544e <+49>:    jne    0x555555555543e <phase_5+33>
   0x0000555555555450 <+51>:    cmp    $0x31,%ecx
   0x0000555555555453 <+54>:    je     0x555555555545a <phase_5+61>
   0x0000555555555455 <+56>:    callq  0x555555555976 <explode_bomb>
   0x000055555555545a <+61>:    pop    %rbx
   0x000055555555545b <+62>:    retq
   0x000055555555545c <+63>:    callq  0x555555555976 <explode_bomb>
---Type <return> to continue, or q <return> to quit---
   0x0000555555555461 <+68>:    jmp    0x55555555542b <phase_5+14>
```

By looking at line +9 we can say that our input is a 6 character long string.

```
(gdb) i r
rax            0x555555758801   93824994347009
rbx            0x555555758800   93824994347008
rcx            0xa              10
```

By observing the change in rcx register in the loop for each character. I first input abcdef then ghijkl to find their values. Then I determined a=10, b =6, and g=3. By looking at line +51 their sum should be 0x31 (49 in decimal). Thus, I wrote aaaabg which makes the sum 49 as 4*10 + 1*6 + 1*3 = 49.

Phase 6:

```
0x0000555555555483 <+32>:    callq  0x5555555559b2 <read_six_numbers>
0x0000555555555488 <+37>:    mov    $0x0,%r13d
0x000055555555548e <+43>:    jmp    0x5555555554b5 <phase_6+82>
0x0000555555555490 <+45>:    callq  0x555555555976 <explode_bomb>
0x0000555555555495 <+50>:    jmp    0x5555555554c4 <phase_6+97>
0x0000555555555497 <+52>:    add    $0x1,%ebx
0x000055555555549a <+55>:    cmp    $0x5,%ebx
0x000055555555549d <+58>:    jg     0x5555555554b1 <phase_6+78>
0x000055555555549f <+60>:    movslq %ebx,%rax
0x00005555555554a2 <+63>:    mov    (%rsp,%rax,4),%eax
0x00005555555554a5 <+66>:    cmp    %eax,0x0(%rbp)
0x00005555555554a8 <+69>:    jne    0x555555555497 <phase_6+52>
-Type <return> to continue, or q <return> to quit---
0x00005555555554aa <+71>:    callq  0x555555555976 <explode_bomb>
0x00005555555554af <+76>:    jmp    0x5555555555497 <phase_6+52>
0x00005555555554b1 <+78>:    add    $0x4,%r12
0x00005555555554b5 <+82>:    mov    %r12,%rbp
0x00005555555554b8 <+85>:    mov    (%r12),%eax
0x00005555555554bc <+89>:    sub    $0x1,%eax
0x00005555555554bf <+92>:    cmp    $0x5,%eax
0x00005555555554c2 <+95>:    ja     0x555555555490 <phase_6+45>
0x00005555555554c4 <+97>:    add    $0x1,%r13d
0x00005555555554c8 <+101>:   cmp    $0x6,%r13d
```

By looking at line +32 we can see that our input is 6 numbers. Starting from line +63 it tells us no 2 inputed numbers are same.
1 0x00000201 = 513
2 0x0000033e = 830
3 0x00000372 = 882
4 0x000003ce = 974
5 0x00000192 = 402
6 0x00000127 = 295

```
   0x00005555555554f5 <+146>:   lea    0x202d34(%rip),%rdx        # 0x5555557582
30 <node1>
```

```
(gdb) x/3x 0x555555758230
0x555555758230 <node1>: 0x00000201      0x00000001      0x55758240
(gdb) x/3x 0x555555758240
0x555555758240 <node2>: 0x0000033e      0x00000002      0x55758250
(gdb) x/3x 0x555555758250
0x555555758250 <node3>: 0x00000372      0x00000003      0x55758260
(gdb) x/3x 0x555555758260
0x555555758260 <node4>: 0x000003ce      0x00000004      0x55758270
(gdb) x/3x 0x555555758270
0x555555758270 <node5>: 0x00000192      0x00000005      0x55758110
(gdb) x/3x 0x555555758110
0x555555758110 <node6>: 0x00000127      0x00000006      0x00000000
(gdb)
```

I input 1 2 3 4 5 6. Then check the nodes, first column gives the node value, second column gives the node number and the third column gives the pointer to next node. I sort the node values from largest to smallest and found the input as 4 3 2 1 5 6.

```
0x0000555555555554 <+241>:   mov    0x8(%rbx),%rax
0x0000555555555558 <+245>:   mov    (%rax),%eax
0x000055555555555a <+247>:   cmp    %eax,(%rbx)
0x000055555555555c <+249>:   jge    0x55555555554b <phase_6+232>
0x000055555555555e <+251>:   callq  0x555555555976 <explode_bomb>
```

This is a linked list. From line +247 we know that we're comparing the value of current node to the next and value of next should be lower than current, otherwise bomb explodes. So, we need to sort values from largest to smallest.

Secret Phase:

```
(gdb) x/s 0x555555556cdf
0x555555556cdf: "%d %d %s"
(gdb) x/s 0x5555557587b0
0x5555557587b0 <input_strings+240>:      ""
(gdb) x/s 0x555555556ce8
0x555555556ce8: "DrEvil"
```

I stepped into phase defused function. Then, I examined some addresses and find out that there is an entrypoint to the secret phase. We need two integers and 1 string. So, our entry point could be phase 4 or phase 5.

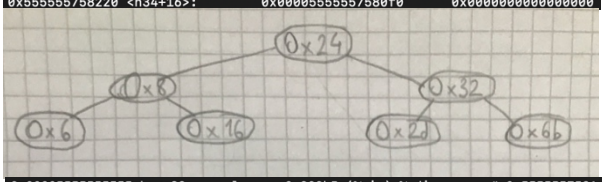
```
(gdb) watch *0x5555557587b0
Hardware watchpoint 2: *0x5555557587b0
(gdb) r psol.txt
Starting program: /Users/byildirim19/assignment4-
Welcome to my fiendish little bomb. You have 6 ph
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
Hardware watchpoint 2: *0x5555557587b0

Old value = 0
New value = 55
__memcpy_sse2 () at ../sysdeps/x86_64/memcpy.S:74
74              incq    %rsi
(gdb) x/s 0x5555557587b0
0x5555557587b0 <input_strings+240>:     "7"
```

I setted up a watchpoint to the address of input string and then I found that my entry point is phase 4 due to the 7 I wrote in phase 4. Then I changed my textfile line 4 to 7 7 DrEvil. I accessed to secret phase.

```
(gdb) x/60x 0x555555758150
0x555555758150 <n1>:    0x0000000000000024      0x0000555555758170
0x555555758160 <n1+16>: 0x0000555555758190      0x0000000000000000
0x555555758170 <n21>:   0x0000000000000008      0x00005555557581f0
0x555555758180 <n21+16>:        0x00005555557581b0      0x0000000000000000
0x555555758190 <n22>:   0x0000000000000032      0x00005555557581d0
0x5555557581a0 <n22+16>:        0x0000555555758210      0x0000000000000000
0x5555557581b0 <n32>:   0x0000000000000016      0x00005555557580b0
0x5555557581c0 <n32+16>:        0x0000555555758070      0x0000000000000000
0x5555557581d0 <n33>:   0x000000000000002d      0x0000555555758010
0x5555557581e0 <n33+16>:        0x00005555557580d0      0x0000000000000000
0x5555557581f0 <n31>:   0x0000000000000006      0x0000555555758030
0x555555758200 <n31+16>:        0x0000555555758090      0x0000000000000000
0x555555758210 <n34>:   0x000000000000006b      0x0000555555758050
0x555555758220 <n34+16>:        0x00005555557580f0      0x0000000000000000
```

I examined an address in secret phase function. I found a Binary Search Tree.



This is how the tree looks like.

```
0x00005555555555eb <+39>:    lea    0x202b5e(%rip),%rdi        # 0x5555557581
<n1>
0x00005555555555f2 <+46>:    callq  0x555555555585 <fun7>
0x00005555555555f7 <+51>:    cmp    $0x1,%eax
0x00005555555555fa <+54>:    je     0x555555555601 <secret_phase+61>
0x00005555555555fc <+56>:    callq  0x555555555976 <explode_bomb>
0x0000555555555601 <+61>:    lea    0x13d8(%rip),%rdi        # 0x5555555569e0
0x0000555555555608 <+68>:    callq  0x555555554e20 <puts@plt>
0x000055555555560d <+73>:    callq  0x555555555b37 <phase_defused>
0x0000555555555612 <+78>:    pop    %rbx
-Type <return> to continue, or q <return> to quit---
0x0000555555555613 <+79>:    retq
0x0000555555555614 <+80>:    callq  0x555555555976 <explode_bomb>
0x0000555555555619 <+85>:    jmp    0x5555555555e9 <secret_phase+37>
```

When we look at line +51 return(eax) value from fun7 function should be 1. In order to make it, we should dive into fun7 function. By examining fun7 we can say that moving right from parent to child node makes eax = eax*2+1, moving left makes eax = eax*2.

```
=> 0x0000555555555585 <+0>:     test   %rdi,%rdi
   0x0000555555555588 <+3>:     je     0x5555555555be <fun7+57>
   0x000055555555558a <+5>:     sub    $0x8,%rsp
   0x000055555555558e <+9>:     mov    (%rdi),%edx
   0x0000555555555590 <+11>:    cmp    %esi,%edx
   0x0000555555555592 <+13>:    jg     0x5555555555a2 <fun7+29>
   0x0000555555555594 <+15>:    mov    $0x0,%eax
   0x0000555555555599 <+20>:    cmp    %esi,%edx
   0x000055555555559b <+22>:    jne    0x5555555555af <fun7+42>
   0x000055555555559d <+24>:    add    $0x8,%rsp
   0x00005555555555a1 <+28>:    retq
   0x00005555555555a2 <+29>:    mov    0x8(%rdi),%rdi
   0x00005555555555a6 <+33>:    callq  0x555555555585 <fun7>
   0x00005555555555ab <+38>:    add    %eax,%eax
   0x00005555555555ad <+40>:    jmp    0x55555555559d <fun7+24>
   0x00005555555555af <+42>:    mov    0x10(%rdi),%rdi
   0x00005555555555b3 <+46>:    callq  0x555555555585 <fun7>
   0x00005555555555b8 <+51>:    lea    0x1(%rax,%rax,1),%eax
   0x00005555555555bc <+55>:    jmp    0x55555555559d <fun7+24>
   0x00005555555555be <+57>:    mov    $0xffffffff,%eax
   0x00005555555555c3 <+62>:    retq
```

We want to make eax 1. First we will need to be left of the parent node. This makes eax = 0. Then, to make eax 1, we should be in the right of the parent node. So our input should be 0x2d = 45.