BOMB LAB

PHASE_1:

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x40239c,%esi
   0x0000000000400e96 <+9>:     call   0x401311 <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    call   0x401410 <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    ret
End of assembler dump.
```

Stack is created,check memory $0x40239c(x/s 0x40239c),it has a string which is being moved into $esi register.And passes onto string_not_equal.Here it is checking user input and the string in $esi registers are same,if not explode_bomb will be called.So, the solution key is string in $esi.In my case it is “Public speaking is very easy.”

```
Reading symbols from bomb...
(gdb) run
Starting program: /home/neeraj/cp/bomb38/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
```

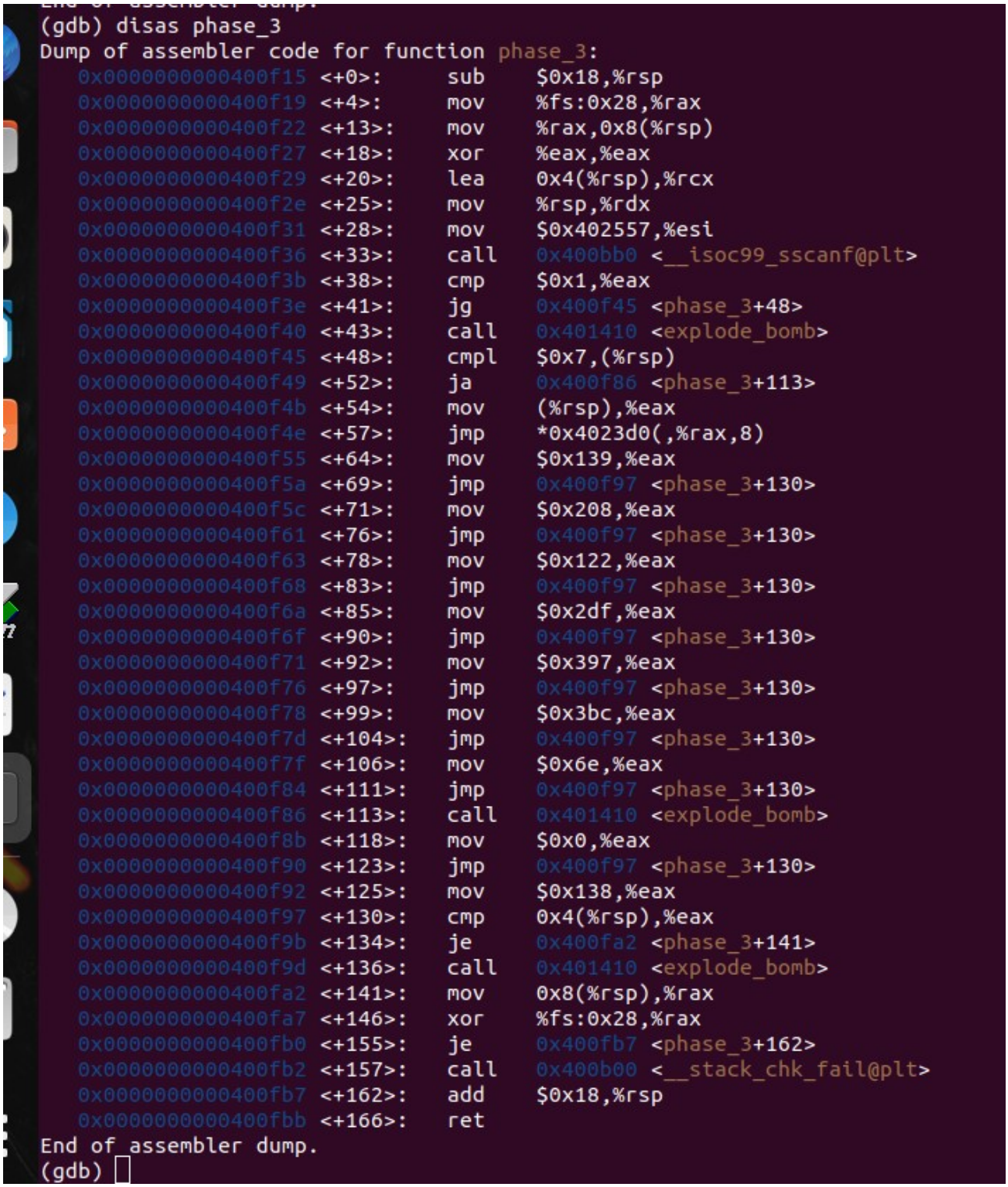
Phase_2:

```
(gdb) disas phase_2
Dump of assembler code for function phase_2:
   0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    call   0x401432 <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jne    0x400ed4 <phase_2+43>
   0x0000000000400ecd <+36>:    cmpl   $0x1,0x4(%rsp)
   0x0000000000400ed2 <+41>:    je     0x400ed9 <phase_2+48>
   0x0000000000400ed4 <+43>:    call   0x401410 <explode_bomb>
   0x0000000000400ed9 <+48>:    mov    %rsp,%rbx
   0x0000000000400edc <+51>:    lea    0x10(%rsp),%rbp
   0x0000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
   0x0000000000400ee4 <+59>:    add    (%rbx),%eax
   0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
   0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:    call   0x401410 <explode_bomb>
   0x0000000000400ef0 <+71>:    add    $0x4,%rbx
   0x0000000000400ef4 <+75>:    cmp    %rbp,%rbx
   0x0000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
   0x0000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
   0x0000000000400efe <+85>:    xor    %fs:0x28,%rax
   0x0000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
   0x0000000000400f09 <+96>:    call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   ret
End of assembler dump.
```

Firstly,stack is created and by examining read_six_numbers assembly code of this function we can conclude that solution format for phase2 is six numbers.In this phase_2 there is a loop execution it is adding previous two inputs in $eax and comparing it with $rbx+8,which is the next input.They already given first two inputs as 0,1 .So it follows fabinocci series. Therefore, the input pattern should be 0 1 1 2 3 5 .

```
(gdb) run
Starting program: /home/neeraj/cp/bomb38/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
```

Phase_3:

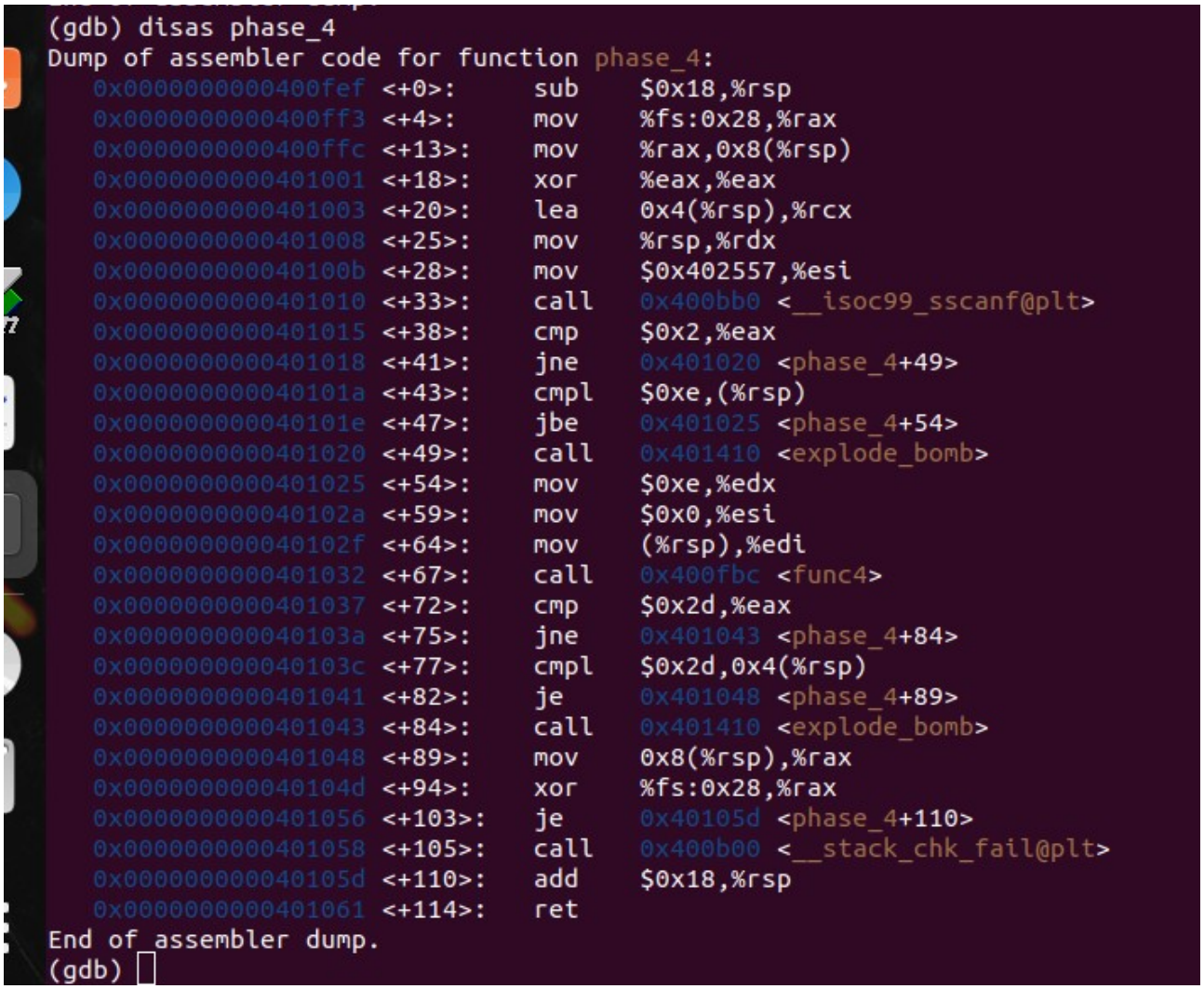


Stack is created and By Observing memory $0x400f97 which has input format for phase3 that is two integers. Its checking

whether the input 1 is greater tha equal to 1 and lessor equal to 7 ,if not it jumps to explode_bomb. So,the first input can be any number in between 1 and 7(including 1,7). • There is a switch case being executed based upon the first input it a moves a immediate value into $eax which is being compared to user input two.So, based on input 1 ,you can conclude the input2 value. In my case input1=7,then input2 should be 0x6e ==110.

```
(gdb) run
Starting program: /home/neeraj/cp/bomb38/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
7 110
Halfway there!
```

Phase_4:

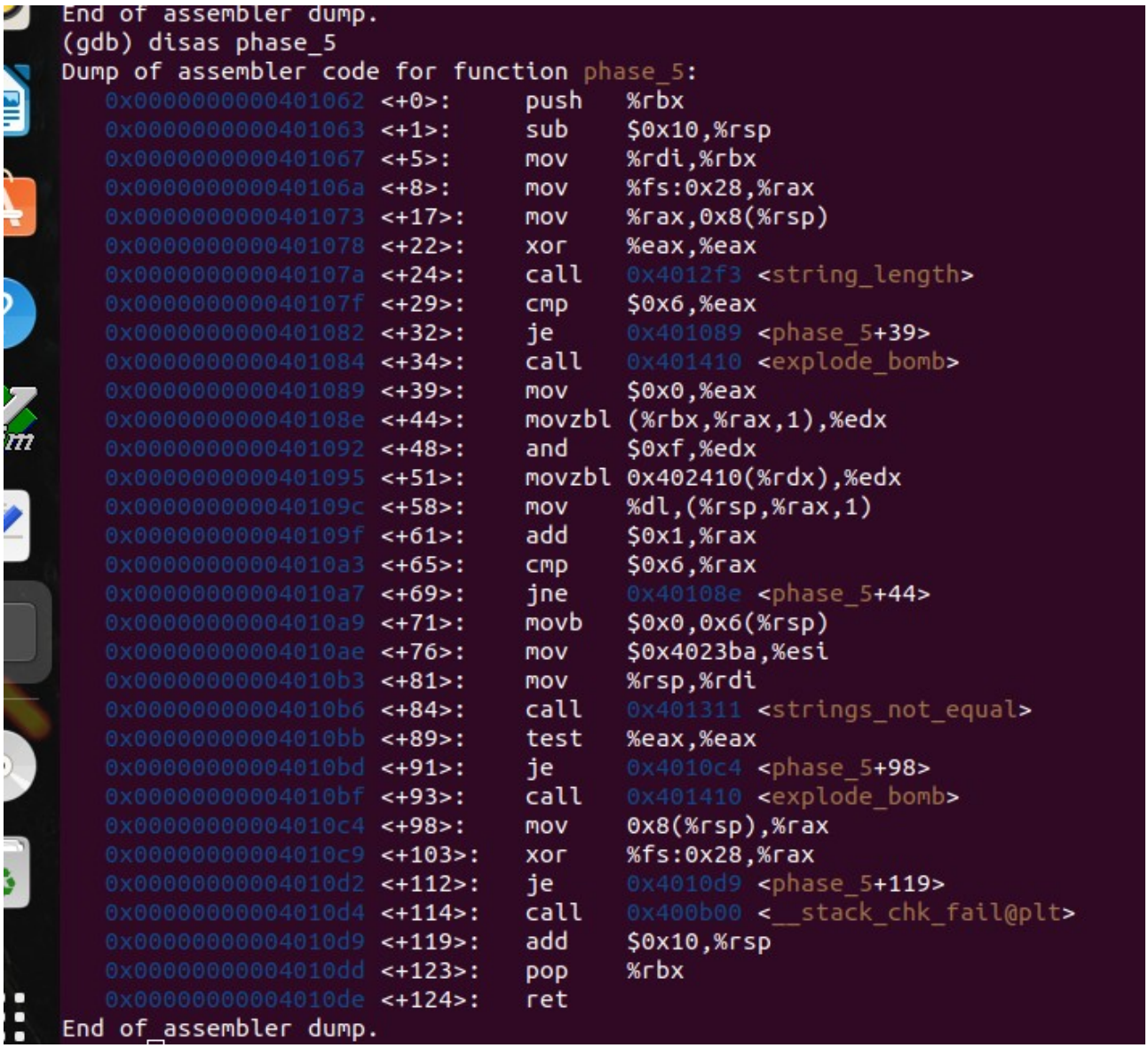


Examine phase_4 assembly code.It can be found that format input is two integers from the $0x402557. We can observe that $rsp+12 holds our first input. $rsp+8 holds our second input.And the code is checking for input1 to be between 2 ,14 from the code at phase_4+72 my second input is being compared with $0x2d(45),if not equal it jumps to explode bomb ,so my input 2 is 45.we can verify the input1 by examining the all integers from 2 to 14 or we can observe func4 assembly code

```
(gdb) disas func4
Dump of assembler code for function func4:
   0x0000000000400fbc <+0>:     push   %rbx
   0x0000000000400fbd <+1>:     mov    %edx,%eax
   0x0000000000400fbf <+3>:     sub    %esi,%eax
   0x0000000000400fc1 <+5>:     mov    %eax,%ebx
   0x0000000000400fc3 <+7>:     shr    $0x1f,%ebx
   0x0000000000400fc6 <+10>:    add    %ebx,%eax
   0x0000000000400fc8 <+12>:    sar    %eax
   0x0000000000400fca <+14>:    lea    (%rax,%rsi,1),%ebx
   0x0000000000400fcd <+17>:    cmp    %edi,%ebx
   0x0000000000400fcf <+19>:    jle    0x400fdd <func4+33>
   0x0000000000400fd1 <+21>:    lea    -0x1(%rbx),%edx
   0x0000000000400fd4 <+24>:    call   0x400fbc <func4>
   0x0000000000400fd9 <+29>:    add    %ebx,%eax
   0x0000000000400fdb <+31>:    jmp    0x400fed <func4+49>
   0x0000000000400fdd <+33>:    mov    %ebx,%eax
   0x0000000000400fdf <+35>:    cmp    %edi,%ebx
   0x0000000000400fe1 <+37>:    jge    0x400fed <func4+49>
   0x0000000000400fe3 <+39>:    lea    0x1(%rbx),%esi
   0x0000000000400fe6 <+42>:    call   0x400fbc <func4>
   0x0000000000400feb <+47>:    add    %ebx,%eax
   0x0000000000400fed <+49>:    pop    %rbx
   0x0000000000400fee <+50>:    ret
End of assembler dump.
```

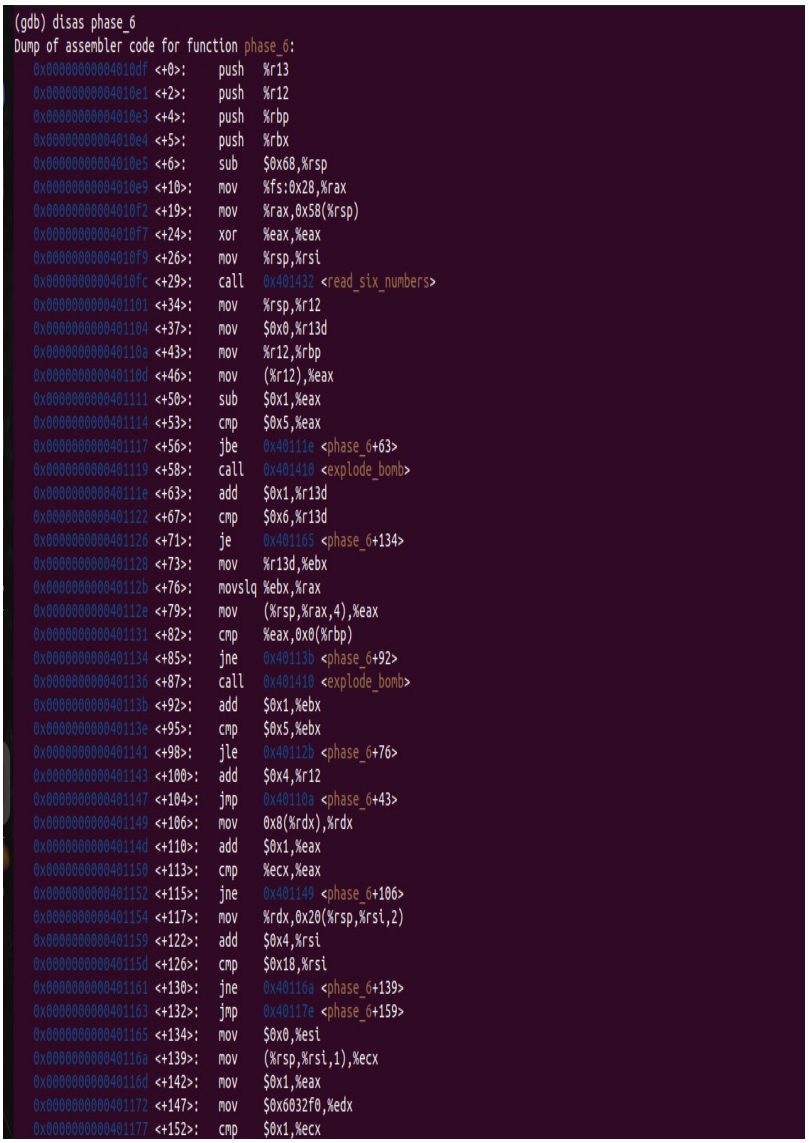
By examine func4 we can find exact value of input1,in my case it is 14.

Phase_5:



Examine phase_5 assembly code.it can seen that input should be a string of length 6. We -can also find a array containing "maduiersnfotvbylWow! You've defused the secret stage!" • There is a loop being executed ,for every iteration its taking last 4 bits of ascii value of each character in the input string. "sabres" is stored in $0x4023ba.code is mapping “sabres” char with char of array and stores index of the array. And that index is converted into ascii value (ex:’7’,7+112=119,’15’,15+96=111). Solution of the this phase is characters stored in this ascii numbers.In my case it is “wqmvuw”.

Phase_6:



```
(gdb) disas phase_6
Dump of assembler code for function phase_6:
   0x00000000004010df <+0>:     push   %r13
   0x00000000004010e1 <+2>:     push   %r12
   0x00000000004010e3 <+4>:     push   %rbp
   0x00000000004010e4 <+5>:     push   %rbx
   0x00000000004010e5 <+6>:     sub    $0x68,%rsp
   0x00000000004010e9 <+10>:    mov    %fs:0x28,%rax
   0x00000000004010f2 <+19>:    mov    %rax,0x58(%rsp)
   0x00000000004010f7 <+24>:    xor    %eax,%eax
   0x00000000004010f9 <+26>:    mov    %rsp,%rsi
   0x00000000004010fc <+29>:    call   0x401432 <read_six_numbers>
   0x0000000000401101 <+34>:    mov    %rsp,%r12
   0x0000000000401104 <+37>:    mov    $0x0,%r13d
   0x000000000040110a <+43>:    mov    %r12,%rbp
   0x000000000040110d <+46>:    mov    (%r12),%eax
   0x0000000000401111 <+50>:    sub    $0x1,%eax
   0x0000000000401114 <+53>:    cmp    $0x5,%eax
   0x0000000000401117 <+56>:    jbe    0x40111e <phase_6+63>
   0x0000000000401119 <+58>:    call   0x401410 <explode_bomb>
   0x000000000040111e <+63>:    add    $0x1,%r13d
   0x0000000000401122 <+67>:    cmp    $0x6,%r13d
   0x0000000000401126 <+71>:    je     0x401165 <phase_6+134>
   0x0000000000401128 <+73>:    mov    %r13d,%ebx
   0x000000000040112b <+76>:    movslq %ebx,%rax
   0x000000000040112e <+79>:    mov    (%rsp,%rax,4),%eax
   0x0000000000401131 <+82>:    cmp    %eax,0x0(%rbp)
   0x0000000000401134 <+85>:    jne    0x40113b <phase_6+92>
   0x0000000000401136 <+87>:    call   0x401410 <explode_bomb>
   0x000000000040113b <+92>:    add    $0x1,%ebx
   0x000000000040113e <+95>:    cmp    $0x5,%ebx
   0x0000000000401141 <+98>:    jle    0x40112b <phase_6+76>
   0x0000000000401143 <+100>:   add    $0x4,%r12
   0x0000000000401147 <+104>:   jmp    0x40110a <phase_6+43>
   0x0000000000401149 <+106>:   mov    0x8(%rdx),%rdx
   0x000000000040114d <+110>:   add    $0x1,%eax
   0x0000000000401150 <+113>:   cmp    %ecx,%eax
   0x0000000000401152 <+115>:   jne    0x401149 <phase_6+106>
   0x0000000000401154 <+117>:   mov    %rdx,0x20(%rsp,%rsi,2)
   0x0000000000401159 <+122>:   add    $0x4,%rsi
   0x000000000040115d <+126>:   cmp    $0x18,%rsi
   0x0000000000401161 <+130>:   jne    0x40116a <phase_6+139>
   0x0000000000401163 <+132>:   jmp    0x40117e <phase_6+159>
   0x0000000000401165 <+134>:   mov    $0x0,%esi
   0x000000000040116a <+139>:   mov    (%rsp,%rsi,1),%ecx
   0x000000000040116d <+142>:   mov    $0x1,%eax
   0x0000000000401172 <+147>:   mov    $0x6032f0,%edx
   0x0000000000401177 <+152>:   cmp    $0x1,%ecx
--Type <RET> for more, q to quit, c to continue without paging--c
   0x000000000040117a <+155>:   jg     0x401149 <phase_6+106>
   0x000000000040117c <+157>:   jmp    0x401154 <phase_6+117>
   0x000000000040117e <+159>:   mov    0x20(%rsp),%rbx
   0x0000000000401183 <+164>:   lea    0x20(%rsp),%rax
   0x0000000000401188 <+169>:   lea    0x48(%rsp),%rsi
   0x000000000040118d <+174>:   mov    %rbx,%rcx
   0x0000000000401190 <+177>:   mov    0x8(%rax),%rdx
   0x0000000000401194 <+181>:   mov    %rdx,0x8(%rcx)
   0x0000000000401198 <+185>:   add    $0x8,%rax
   0x000000000040119c <+189>:   mov    %rdx,%rcx
   0x000000000040119f <+192>:   cmp    %rsi,%rax
   0x00000000004011a2 <+195>:   jne    0x401190 <phase_6+177>
   0x00000000004011a4 <+197>:   movq   $0x0,0x8(%rdx)
   0x00000000004011ac <+205>:   mov    $0x5,%ebp
   0x00000000004011b1 <+210>:   mov    0x8(%rbx),%rax
   0x00000000004011b5 <+214>:   mov    (%rax),%eax
   0x00000000004011b7 <+216>:   cmp    %eax,(%rbx)
   0x00000000004011b9 <+218>:   jge    0x4011c0 <phase_6+225>
   0x00000000004011bb <+220>:   call   0x401410 <explode_bomb>
   0x00000000004011c0 <+225>:   mov    0x8(%rbx),%rbx
   0x00000000004011c4 <+229>:   sub    $0x1,%ebp
   0x00000000004011c7 <+232>:   jne    0x4011b1 <phase_6+210>
   0x00000000004011c9 <+234>:   mov    0x58(%rsp),%rax
   0x00000000004011ce <+239>:   xor    %fs:0x28,%rax
   0x00000000004011d7 <+248>:   je     0x4011de <phase_6+255>
   0x00000000004011d9 <+250>:   call   0x400b00 <__stack_chk_fail@plt>
   0x00000000004011de <+255>:   add    $0x68,%rsp
   0x00000000004011e2 <+259>:   pop    %rbx
   0x00000000004011e3 <+260>:   pop    %rbp
   0x00000000004011e4 <+261>:   pop    %r12
   0x00000000004011e6 <+263>:   pop    %r13
   0x00000000004011e8 <+265>:   ret
End of assembler dump.
```

By examining code can see read_six_numbers function .If we go through ,we can conclude that input should be 6 integers all different and 1 to 6. If we debug and go through the code and memory we can find a list of nodes which has indices and node values.

```
End of assembler dump.
(gdb) x/d 0x6032f0
0x6032f0 <node1>:       335
(gdb) x/d 0x603300
0x603300 <node2>:       80
(gdb) x/d 0x603310
0x603310 <node3>:       425
(gdb) x/d 0x603320
0x603320 <node4>:       535
(gdb) x/d 0x603330
0x603330 <node5>:       228
(gdb) x/d 0x603340
0x603340 <node6>:       935
```

Here sort the node indices based on the node values in the decreasing order. Therefore inputs 6 4 3 1 5 2.

```
1 Public speaking is very easy.
2 0 1 1 2 3 5
3 7 110
4 14 45
5 wqmvuw
6 6 4 3 1 5 2
7
```

```
(gdb) r answer.txt
Starting program: /home/neeraj/cp/bomb38/bomb answer.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
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!
So you got that one.  Try this one.
Good work!  On to the next...
Congratulations! You've defused the bomb!
[Inferior 1 (process 4959) exited normally]
(gdb)
```