

# **Assignment 1**

**ITS304**

Submitted by: Kinley Gyeltshen

Enrollment no: 12190060

Class: BSc IT 'A'

### **Necessary key words to understand**

**disas** – to disassemble the assembly code.

**b** – To set breakpoints.

**r** – to run.

**i r** – To read information about registers.

**q** – To quit from the debugger.

**Jne** – Jump if not equal.

**Cmp** – compare

**x/s** – To know the number of inputs required.

## Phase 1

- ❖ To first execute the code, we have to get inside the directory in which our bomb is in.

```
user@KAY-G: ~  
user@KAY-G:~$ cd /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003
```

- ❖ Then we have to use the gdb debugger to debug our code / to understand what is actually happening in our code.

```
user@KAY-G: /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003  
user@KAY-G:~$ cd /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003  
user@KAY-G:/mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003$ gdb bomb  
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2  
Copyright (C) 2020 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law.  
Type "show copying" and "show warranty" for details.  
This GDB was configured as "x86_64-linux-gnu".  
Type "show configuration" for configuration details.  
For bug reporting instructions, please see:  
http://www.gnu.org/software/gdb/bugs/.  
Find the GDB manual and other documentation resources online at:  
http://www.gnu.org/software/gdb/documentation/.  
  
For help, type "help".
```

- ❖ After we enter inside our debugger we then set a breakpoint at phase 1, in order to prevent our bomb from exploding incase we guess the wrong answer. Then we run the program, and we are welcomed by Dr.Evil with a message. We then give our first input, and dissemble the assembly code to see the operations happening within the code.

```
(gdb) b phase_1  
Breakpoint 1 at 0x400e8d  
(gdb) disas  
No frame selected.  
(gdb) run  
Starting program: /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003/bomb  
Welcome to my fiendish little bomb. You have 6 phases with  
which to blow yourself up. Have a nice day!  
fajjdsfla  
  
Breakpoint 1, 0x0000000000400e8d in phase_1 ()  
(gdb) disas  
Dump of assembler code for function phase_1:  
=> 0x0000000000400e8d <+0>:    sub    $0x8,%rsp  
    0x0000000000400e91 <+4>:    mov    $0x4023b0,%esi  
    0x0000000000400e96 <+9>:    callq  0x40132b <strings_not_equal>  
    0x0000000000400e9b <+14>:   test   %eax,%eax  
    0x0000000000400e9d <+16>:   je    0x400ea4 <phase_1+23>  
    0x0000000000400e9f <+18>:   callq  0x40142a <explode_bomb>  
    0x0000000000400ea4 <+23>:  add    $0x8,%rsp  
    0x0000000000400ea8 <+27>:  retq  
End of assembler dump.
```

- ❖ From the <strings\_not\_equal> function we know that out input must be a string.

- ❖ We “strings bomb” to display all the strings in the code.

```
user@KAY-G: /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003
resolve H
server aH
E(ddref
E,ss
t)APRA
AWAVA
AUATL
[.]A\A]A^A_
%: Error: Couldn't open %
Usage: %s [<input_file>]
That's number 2. Keep going!
Halfway there!
Good work! On to the next...
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?
So you got that one. Try this one.
Border relations with Canada have never been better.
Wow! You've defused the secret stage!
So you think you can stop the bomb with ctrl-c, do you?
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Congratulations! You've defused the bomb!
Well...
OK. :)
Invalid phase%
BOOM!!!
The bomb has blown up.
%d %d %d %d %
Error: Premature EOF on stdin
GRADE_BOMB
Error: Input line too long
%d %d %s
```

- ❖ From the strings displayed we find a string that is completely unrelated to the Dr.evil’s speech. Hence we find the our input for the first phase.

```
user@KAY-G: /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003
resolve H
server aH
E(ddref
E,ss
t)APRA
AWAVA
AUATL
[.]A\A]A^A_
%: Error: Couldn't open %
Usage: %s [<input_file>]
That's number 2. Keep going!
Halfway there!
Good work! On to the next...
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?
So you got that one. Try this one.
Border relations with Canada have never been better. Border relations with Canada have never been better.
Wow! You've defused the secret stage!
So you think you can stop the bomb with ctrl-c, do you?
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Congratulations! You've defused the bomb!
Well...
```

## Phase 2

- ❖ After diffusing the first bomb, we then go on to the next phase, there we enter a random input.

```
user@KAY-G: /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003
Reading symbols from bomb...
(gdb) b phase_2
Breakpoint 1 at 0x400ea9
(gdb) r
Starting program: /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
0 1 1 3 4 5
```

- ❖ Then we disassemble the code using the “disas” code. From there we understand that the input is six numbers from the <read\_six\_numbers> function.

```
(gdb) disas
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
 0x0000000000400eb0 <+15>: mov %rax,0x18(%rsp)
 0x0000000000400ebd <+20>: xor %eax,%eax
 0x0000000000400ebf <+22>: mov %rsp,%rsi
 0x0000000000400ec2 <+25>: callq 0x00144c <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>: callq 0x00142a <explode_bomb>
 0x0000000000400ed9 <+48>: mov %rsp,%rbx
 0x0000000000400edc <+51>: lea 0x10(%sp),%rbp
 0x0000000000400ee1 <+56>: mov 0x4(%rbx),%eax
 0x0000000000400ea4 <+59>: add (%rbx),%eax
 0x0000000000400ea6 <+61>: cmp %eax,0x8(%bx)
 0x0000000000400ea9 <+64>: je 0x400ef0 <phase_2+71>
 0x0000000000400eb8 <+66>: callq 0x00142a <explode_bomb>
 0x0000000000400ef0 <+71>: add $0x4,%rbx
 0x0000000000400ef4 <+75>: cmp %rbp,%rbx
 0x0000000000400ef7 <+78>: jne 0x400ee1 <phase_2+56>
 0x0000000000400ef9 <+80>: mov 0x18(%sp),%rax
 0x0000000000400fe <+85>: xor %fs:0x28,%rax
 0x0000000000400f07 <+94>: je 0x400ef0 <phase_2+101>
 0x0000000000400f09 <+96>: callq 0x000b00 <_stack_chk_fail@plt>
 0x0000000000400f0e <+101>: add $0x28,%rsp
 0x0000000000400f12 <+105>: pop %rbx
 0x0000000000400f13 <+106>: pop %rbp
 0x0000000000400f14 <+107>: retq
End of assembler dump.
```

- ❖ We mostly focus on the callq functions in the code and compare functions.

```
(gdb) until *0x0000000000400ec2
0x0000000000400ec2 in phase_2 ()
```

- ❖ We move on to the compare function where it compares our first number with 0, if its not equal then the bomb explodes, if it does not then it proves that 0 is the first input required. For the second input as well 1 is compared to the second number if its correct then it proceeds if not it explodes.

```
Dump of assembler code for function phase_2:
0x0000000000400ea9 <+0>:    push    %rbp
0x0000000000400ea9 <+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
0x0000000000400ebf <+25>:   callq  0x40142a <explode_bomb>
0x0000000000400ec7 <+30>:   cmpl   $0x0,(%rsp)
0x0000000000400ecd <+36>:   cmpl   $0x1,0x4(%rsp)    2+43>
0x0000000000400ed2 <+41>:   je     0x400ed9 <phase_2+48>
0x0000000000400ed4 <+43>:   callq  0x40142a <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)
0x0000000000400e9 <+64>:   je     0x400ef0 <phase_2+71>
0x0000000000400eb <+66>:   callq  0x40142a <explode_bomb>
```

- ❖ For the third value, we again go to compare function. Where it compares if value at register %eax is equal to value at address 0x8\*%rbx. If it is equal it proceeds if not the bomb explodes. We check the values by typing “i r” and compare the values of %eax and 0x8%rbx which are both 1 so third value is 1.

```
0x0000000000400cd <+36>:   cmpl   $0x1,0x4(%rsp)
0x0000000000400ed2 <+41>:   je     0x400ed9 <phase_2+48>
0x0000000000400ed4 <+43>:   callq  0x40142a <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>
0x0000000000400eb <+66>:   callq  0x40142a <explode_bomb>
0x0000000000400ef0 <+71>:   add    $0x4,%rbx
0x0000000000400f4 <+75>:   cmp    %rbp,%rbx
0x0000000000400ef7 <+78>:   jne   0x400ee1 <phase_2+56>
0x0000000000400ef9 <+80>:   mov    0x18(%rsp),%rax
0x0000000000400fe <+85>:   xor    %fs:0x28,%rax
```

```
end of assembler dump.
(gdb) i r
rax      0x1          1
rbx      0xfffffff1b0  140737488282032
rcx      0x0          0
rdx      0xfffffff1c4  140737488282052
rsi      0x0          0
rdi      0x7fffffffdb40 140737488280384
rbp      0x7fffffff1c0  0x7fffffff1c0
rsp      0x7fffffff1b0  0x7fffffff1b0
r8       0xffffffff    4294967295
r9       0x0          0
r10      0x7fffff74eac0 140737479240384
r11      0x0          0
r12      0x400c60     4197472
r13      0x7fffffee2e0  140737488282336
r14      0x0          0
r15      0x0          0
rip      0x400ee6     0x400ee6 <phase_2+61>
eflags   0x202        [ IF ]
cs       0x33         51
ss       0x2b         43
ds       0x0          0
es       0x0          0
fs       0x0          0
gs       0x0          0
(gdb) x/d $eax
0x1:  Cannot access memory at address 0x1
(gdb) x/d $rbx+8
0x7fffffe1b8: 1
```

- ❖ We again move to the next compare function. Here it compares the value at %rbp and %rbx which are not equal so we jump to line 56.

```

0x000000000000400edc <+51>:    lea    0x10(%rsp),%rbp
0x000000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
0x000000000000400ee4 <+59>:    add    (%rbx),%eax
0x000000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
0x000000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
0x000000000000400eeb <+66>:    callq 0x40142a <explode_bomb>
0x000000000000400ef0 <+71>:    add    $0x4,%rbx
0x000000000000400ef4 <+75>:    cmp    %rbp,%rbx
=> 0x000000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
0x000000000000400ef9 <+80>:    mov    0x18(%rsp),%rax
0x000000000000400fe <+85>:    xor    %fs:0x28,%rax
0x000000000000400f07 <+94>:    je     0x400f0e <phase_2+101>
0x000000000000400f09 <+96>:    callq 0x400b00 <__stack_chk_fail@plt>
0x000000000000400f0e <+101>:   add    $0x28,%rsp
0x000000000000400f12 <+105>:  pop    %rbx
0x000000000000400f13 <+106>:  pop    %rbp
0x000000000000400f14 <+107>:  pop    %rbp
0x000000000000400f14 <+107>:  retq

r15          0x0
rip          0x400ef7          0x400ef7 <phase_2+78>
eflags        0x283           [ CF SF IF ]
cs            0x33            51
ss            0x2b            43
ds            0x0              0
es            0x0              0
fs            0x0              0
gs            0x0              0
(gdb) x/d $rbx
0x7fffffeelb4: 1
(gdb) x/d $rbp
0x7fffffeelc0: 4
(gdb) -
0x000000000000400ecd <+36>:  cmpl   $0x1,0x4(%rsp)
0x000000000000400ed2 <+41>:  je     0x400ed9 <phase_2+48>
0x000000000000400ed4 <+43>:  callq 0x40142a <explode_bomb>
0x000000000000400ed9 <+48>:  mov    %rsp,%rbx
0x000000000000400edc <+51>:  lea    0x10(%rsp),%rbp
=> 0x000000000000400ee1 <+56>:  mov    0x4(%rbx),%eax
0x000000000000400ee4 <+59>:  add    (%rbx),%eax
0x000000000000400ee6 <+61>:  cmp    %eax,0x8(%rbx)
0x000000000000400ee9 <+64>:  je     0x400ef0 <phase_2+71>
0x000000000000400eeb <+66>:  callq 0x40142a <explode_bomb>
0x000000000000400ef0 <+71>:  add    $0x4,%rbx
0x000000000000400ef4 <+75>:  cmp    %rbp,%rbx
0x000000000000400ef7 <+78>:  jne    0x400ee1 <phase_2+56>
0x000000000000400ef9 <+80>:  mov    0x18(%rsp),%rax
0x000000000000400fe <+85>:  xor    %fs:0x28,%rax
0x000000000000400f07 <+94>:  je     0x400f0e <phase_2+101>
0x000000000000400f09 <+96>:  callq 0x400b00 <__stack_chk_fail@plt>
0x000000000000400f0e <+101>: add    $0x28,%rsp
0x000000000000400f12 <+105>: pop    %rbx
0x000000000000400f13 <+106>: pop    %rbp
0x000000000000400f14 <+107>: pop    %rbp
0x000000000000400f14 <+107>: retq

```

- ❖ We move to the next compare function. It compares the value at %eax and 0x8(%rbx), if they are equal it will proceed if not it will explode.

```

0x000000000000400edc <+51>:    lea    0x10(%rsp),%rbp
0x000000000000400ee1 <+56>:    mov    0x4(%rbx),%eax
0x000000000000400ee4 <+59>:    add    (%rbx),%eax
=> 0x000000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
0x000000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
0x000000000000400eeb <+66>:    callq 0x40142a <explode_bomb>
0x000000000000400ef0 <+71>:    add    $0x4,%rbx
0x000000000000400ef4 <+75>:    cmp    %rbp,%rbx
0x000000000000400ef7 <+78>:    jne    0x400ee1 <phase_2+56>
0x000000000000400ef9 <+80>:    mov    0x18(%rsp),%rax

```

The value at 0x8(%rbx) is the value we entered that is 3 so it is being compared to %eax which has value 2, which is not equal so hence the bomb explodes. Because the bomb exploded we now know that the fourth value is 2. So currently we have 0 1 1 2 as our input, if we closely look at the inputs it follows a certain pattern that is the Fibonacci series so hence we can predict the next two values.

```
(gdb) i r
rax      0x2          2
rbx      0x7fffffff1b4  140737488282036
rcx      0x0          0
rdx      0x7fffffff1c4  140737488282052
rsi      0x0          0
rdi      0x7fffffedb4d0 140737488280384
rbp      0x7fffffff1c8  0x7fffffff1c1c0
rsp      0x7fffffff1b0  0x7fffffff1b0
r8      0xffffffff        4294967295
r9      0x0          0
r10     0x7fffff74eac0  140737479240384
r11     0x0          0
r12     0x400c60        4197472
r13     0x7fffffe2e0    140737488282336
r14     0x0          0
r15     0x0          0
rip      0x400ee6        0x400ee6 <phase_2+61>
eflags    0x202        [ IF ]
cs       0x33         51
ss       0x2b         43
ds       0x0          0
es       0x0          0
fs       0x0          0
gs       0x0          0
(gdb) x/d $rbx+8
$7fffffff1bc: 3

0x00000000000400ecd <+36>: cmpl $0x1,0x4(%rsp)
0x00000000000400ed2 <+41>: je 0x400ed9 <phase_2+48>
0x00000000000400ed4 <+43>: callq 0x40142a <explode_bomb>
0x00000000000400ed9 <+48>: mov %rsp,%rbx
0x00000000000400edc <+51>: lea 0x10(%rsp),%rbp
0x00000000000400ee1 <+56>: mov 0x4(%rbx),%eax
0x00000000000400ee4 <+59>: add (%rbx),%eax
0x00000000000400ee6 <+61>: cmp %eax,0x8(%rbx)
0x00000000000400ee9 <+64>: je 0x400ef0 <phase_2+71>
=> 0x00000000000400eeb <+66>: callq 0x40142a <explode_bomb>
0x00000000000400ef0 <+71>: add $0x4,%rbx
0x00000000000400ef4 <+75>: cmp %rbp,%rbx
0x00000000000400ef7 <+78>: jne 0x400ee1 <phase_2+56>
0x00000000000400ef8 <+79>: bne 0x400ef0,%rbx,%rbp
```

- ❖ We can predict that the inputs are 0 1 1 2 3 5. We run the program again and give the correct answer, and bomb gets difused.

```
Reading symbols from bomb...
(gdb) r
Starting program: /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
```

### Phase 3

- ❖ For phase 3 we enter a random number and enter. Then we disassemble the code and from the address above the callq function we can figure out the type of input and number of input required. So we know that we need two integers.

```
user@KAY-G: /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003
Phase 1 defused. How about the next one?
That's number 2. Keep going!
3 5 32423

Breakpoint 1, 0x0000000000400f15 in phase_3 ()
(gdb) disas
Dump of assembler code for function phase_3:
=> 0x0000000000400f15 <+0>:    sub    $0x18,%rsp
    0x0000000000400f19 <+4>:    mov    %fs:0x28,%rax
    0x0000000000400f22 <+13>:   mov    %rax,0x8(%rsp)
    0x0000000000400f27 <+18>:   xor    %eax,%eax
    0x0000000000400f29 <+20>:   lea    0x4(%rsp),%rcx
    0x0000000000400f2e <+25>:   mov    %rsp,%rdx
    0x0000000000400f31 <+28>:   mov    $0x4025af,%esi
    0x0000000000400f36 <+33>:   callq 0x400bb0 <__isoc99_sscanf@plt>
    0x0000000000400f3b <+38>:   cmp    $0x1,%eax
    0x0000000000400f3e <+41>:   jg    0x400f45 <phase_3+48>
    0x0000000000400f40 <+43>:   callq 0x40142a <explode_bomb>
    0x0000000000400f45 <+48>:   cmpl   $0x7,(%rsp)

    0x0000000000400fa2 <+141>:   mov    0x8(%rsp),%rax
    0x0000000000400fa7 <+146>:   xor    %fs:0x28,%rax
    0x0000000000400fb0 <+155>:   je    0x400fb7 <phase_3+155>
    0x0000000000400fb2 <+157>:   callq 0x400b00 <__stack_chk_fail@plt>
    0x0000000000400fb7 <+162>:   add    $0x18,%rsp
    0x0000000000400fbb <+166>:   retq

End of assembler dump.
(gdb) x/s 0x4025af
0x4025af:      "%d %d"
(gdb)
```

- ❖ We jump to the compare statement where it compares if our first input is greater than 1. If it is greater than it does not explode.

```
0x0000000000400f19 <+4>:    mov    %fs:0x28,%rax
0x0000000000400f22 <+13>:   mov    %rax,0x8(%rsp)
0x0000000000400f27 <+18>:   xor    %eax,%eax
0x0000000000400f29 <+20>:   lea    0x4(%rsp),%rcx
0x0000000000400f2e <+25>:   mov    %rsp,%rdx
0x0000000000400f31 <+28>:   mov    $0x4025af,%esi
0x0000000000400f36 <+33>:   callq 0x400bb0 <__isoc99_sscanf@plt>
=> 0x0000000000400f3b <+38>:   cmp    $0x1,%eax
    0x0000000000400f3e <+41>:   jg    0x400f45 <phase_3+48>
    0x0000000000400f40 <+43>:   callq 0x40142a <explode_bomb>
    0x0000000000400f45 <+48>:   cmpl   $0x7,(%rsp)
    0x0000000000400f49 <+52>:   ja    0x400f86 <phase_3+113>
```

- The next compare function compares our value with 7 and checks it is less than 7 , in other words our first digit must be between 1 -7.

```
0x000000000000400f3b <+38>: cmp    $0x1,%eax
0x000000000000400f3e <+41>: jg     0x400f45 <phase_3+48>
0x000000000000400f40 <+43>: callq  0x40142a <explode_bomb>
=> 0x000000000000400f45 <+48>: cmpl   $0x7,(%rsp)
0x000000000000400f49 <+52>: ja     0x400f86 <phase_3+113>
0x000000000000400f4b <+54>: mov    (%rsp),%eax
0x000000000000400f4e <+57>: jmpq   *0x402420(%rax,8)
0x000000000000400f55 <+64>: mov    $0xc0,%eax
0x000000000000400f5a <+69>: jmp    0x400f97 <phase_3+130>
0x000000000000400f5c <+71>: mov    $0x31,%eax
```

- It compares the value at address 0x4(%rsp) and %eax, and if its equal then it's the correct value if not the bomb explodes. The bomb explodes as our value is 5 and the value at %eax is 665. So we know that the second input is 665.

```
0x000000000000400f92 <+125>: mov    $0x3f,%eax
=> 0x000000000000400f97 <+130>: cmp    0x4(%rsp),%eax
0x000000000000400f9b <+134>: je    0x400fa2 <phase_3+141>
0x000000000000400f9d <+136>: callq  0x40142a <explode_bomb>
0x000000000000400fa2 <+141>: mov    0x8(%rsp),%rax
0x000000000000400fa7 <+146>: xor    %fs:0x28,%rax

(gdb) i r
rax          0x299          665
rbx          0x7fffffff2d8  140737488282328
rcx          0x0
rdx          0x7fffffff1c4  140737488282052
rsi          0x0
rdi          0x7fffffffedb70 140737488280432
rbp          0x0
rsp          0x7fffffff1c0  0x7fffffff1c0
r8           0x20           32
r9           0x0
r10          0x7fffffff74eac0 140737479240384
r11          0x0
r12          0x400c60        4197472
r13          0x7fffffff2d0  140737488282320
r14          0x0
r15          0x0
rip          0x400f9d        0x400f9d <phase_3+136>
eflags        0x202          [ IF ]
cs            0x33           51
ss            0x2b           43
ds            0x0
es            0x0
fs            0x0
gs            0x0
```

(gdb) x/d \$rsp+4  
0x7fffffff1c4: 5

- So the two inputs are 3 and 665.

```
(gdb) r
Starting program: /mnt/c/Users/user/OneDrive/Desktop/3rd/bomb003/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
3 665
Halfway there!
```