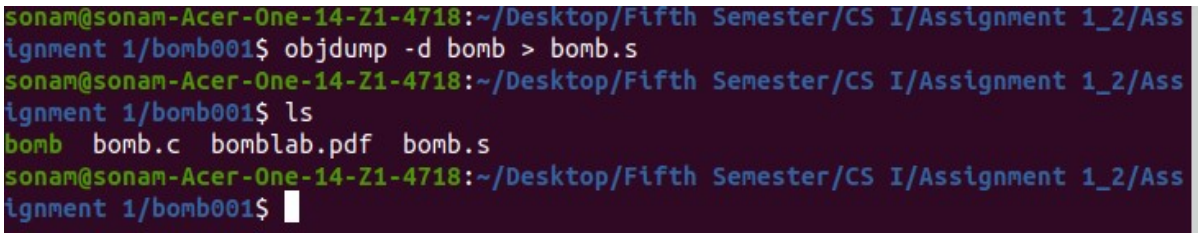
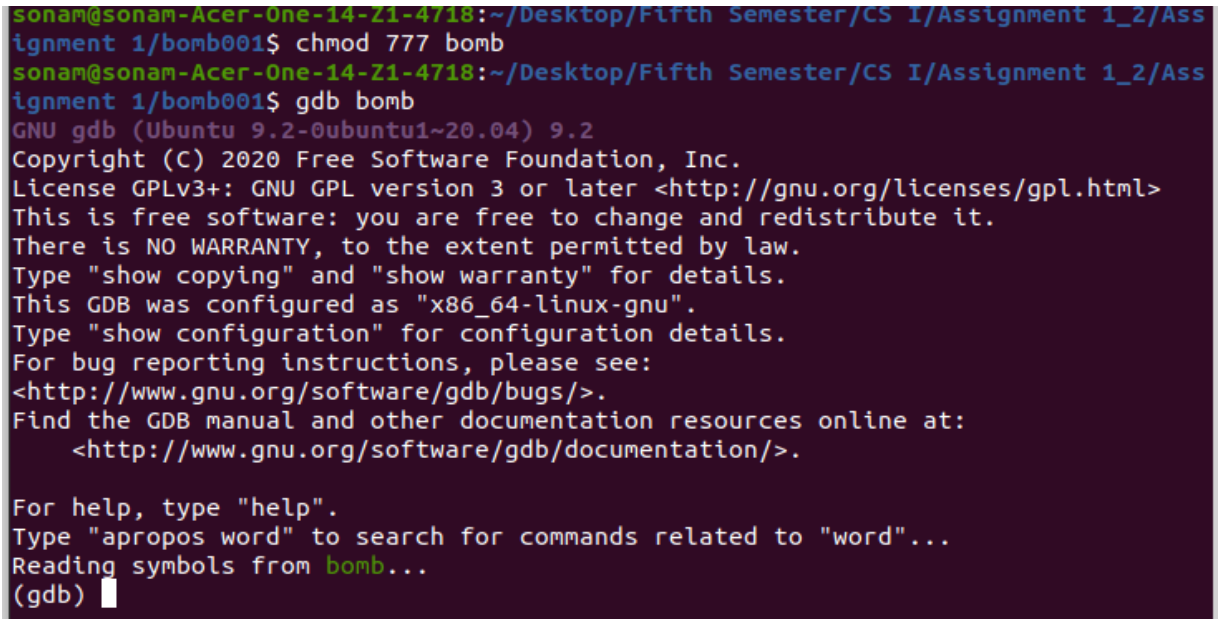
Bomb001
phase_1

The first phase_1 of bomb001 is about giving a string to Dr. Evil. If the user input string store in %eax matches the string of Dr Evil stored in 0x4023d0 than the phase 1 of the bomb will be deffused, prompting us to next phase, else it will get blow up. As we don't know the string set by Dr Evil, It is our work to find the correct string to deffused the bomb. The string which I got from phase_1 bomb001 is "The moon unit will be divided into two divisions."

First, open the terminal from bomb001 folder and run the command *"objdump -d bomb > bomb.s"* The command will convert the bomb file from machine langauge into assembly langauge in bomb.s file to look more at the assembly code for the next phase. And it also dissamble the code and major function, reads the assembler code and displays information about one or more object files.



chmod 777 bomb will set the permission of bomb file to read, write and execute the code by all the users.
The *gdb bomb* command will debug the assembly file (bomb.s) and helps to run a program up to certain point and stop to print out the values of certain variables. After that we have to set the break point at phase 1 in order to stop at phase_1 for debugging purpose. After setting the break-point we can run the program using "run" command.



After that set the break point "*break phase_1"* in the phase_1 of bomb001 to stop at phase_1 for debugging purpose. It will ensure that the bomb doesn't blow up when we run the program. Also set *break explode_bomb* function in order to pause the execution after entering the phase_1 function. After that disassemble the phase_1 using *"disas phase_1"* to go inside the assembly code of the

phase_1 and then run the program using *r.*

```
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) b phase_1
Breakpoint 1 at 0x400e8d
(gdb) b explode_bomb
Breakpoint 2 at 0x40143d
(gdb) disas
No frame selected.
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x4023d0,%esi
   0x0000000000400e96 <+9>:     callq  0x40133e <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    callq  0x40143d <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    retq
End of assembler dump.
(gdb) r
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb
Welcome to my fiendish little bomb. You have 6 phases with
```

After running the program give any rand input to test the string. In my case I have given *test string* and the answer does not match matched the input given by Dr. Evil, yet the bomb didn't explode due to setting of break point in the phase_1. Then execute the next instruction until you reached test %ea %eax.

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
test string

Breakpoint 1, 0x0000000000400e8d in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
=> 0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x4023d0,%esi
   0x0000000000400e96 <+9>:     callq  0x40133e <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    callq  0x40143d <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    retq
End of assembler dump.
(gdb) ni
0x0000000000400e91 in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
=> 0x0000000000400e91 <+4>:     mov    $0x4023d0,%esi
   0x0000000000400e96 <+9>:     callq  0x40133e <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    callq  0x40143d <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    retq
End of assembler dump.
(gdb) x/s 0x4023d0
```

After reaching the instruction line 4, inspect what is being moved from address 0x4023d0. As We know it has to be a string so we use '/s' using the command *x/s 0x4023d0.* Running this command, we get the string input set by Dr. Evil in the phase_1 of bmb001. *The string is "The moon will be divide into two divisions".*

```
(gdb) ni
0x0000000000400e91 in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
=> 0x0000000000400e91 <+4>:     mov    $0x4023d0,%esi
   0x0000000000400e96 <+9>:     callq  0x40133e <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    callq  0x40143d <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    retq
End of assembler dump.
(gdb) x/s 0x4023d0
0x4023d0:       "The moon unit will be divided into two divisions."
(gdb) ni
0x0000000000400e96 in phase_1 ()
(gdb) ni
0x0000000000400e9b in phase_1 ()
```

Once we reached the text instruction %eax%eax , we can run the *i r* command to see the information of the register. Here the eax value is 1 which will call the explode_bomb, since the string does not matched with the given string.

```
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x4023d0,%esi
   0x0000000000400e96 <+9>:     callq  0x40133e <strings_not_equal>
=> 0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    callq  0x40143d <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    retq
End of assembler dump.
(gdb) i r
rax            0x1                 1
rbx            0x4021f0            4202992
rcx            0xb                 11
rdx            0x1                 1
rsi            0x4023d0            4203472
rdi            0x402401            4203521
rbp            0x0                 0x0
rsp            0x7fffffffde00      0x7fffffffde00
r8             0x6037a0            6305696
r9             0x7c                124
r10            0xfffffffffffff6ed  -2323
r11            0x7ffff7df9400      140737352012800
r12            0x400c60            4197472
r13            0x7fffffffdf00      140737488346880
r14            0x0                 0
r15            0x0                 0
rip            0x400e9b            0x400e9b <phase_1+14>
eflags         0x283               [ CF SF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

Then run the program again by giving the input string we got from the address 0x4023d0. Check the information register to see the value of eax. If the value is zero “0” then it will execute line 23 and then will return. Thus, the phase_1 of bom001 will get defused and we will be prompted to the next phase.

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The moon unit will be divided into two divisions.

Breakpoint 1, 0x0000000000400e8d in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
=> 0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x4023d0,%esi
   0x0000000000400e96 <+9>:     callq  0x40133e <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    callq  0x40143d <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    retq
End of assembler dump.
```

Since the eax value is zero, line 23 will get executed and will return the value.

```
(gdb) u* 0x0000000000400e9b
0x0000000000400e9b in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x4023d0,%esi
   0x0000000000400e96 <+9>:     callq  0x40133e <strings_not_equal>
=> 0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    callq  0x40143d <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    retq
End of assembler dump.
(gdb) i r
rax            0x0                 0
rbx            0x4021f0            4202992
rcx            0x31                49
rdx            0x0                 0
rsi            0x4023d0            4203472
rdi            0x402401            4203521
rbp            0x0                 0x0
rsp            0x7fffffffde00      0x7fffffffde00
r8             0x6037a0            6305696
r9             0x7c                124
r10            0xfffffffffffff6ed  -2323
r11            0x7ffff7df9400      140737352012800
r12            0x400c60            4197472
r13            0x7fffffffdf00      140737488346880
r14            0x0                 0
r15            0x0                 0
rip            0x400e9b            0x400e9b <phase_1+14>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
```

Used the command *info break* to see about the information about the break point we set earlier and after that remove break point in order to defused the phase_1, else the break point will prevent the phase from defusing.

```
End of assembler dump.
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000400e8d <phase_1>
        breakpoint already hit 1 time
2       breakpoint     keep y   0x000000000040143d <explode_bomb>
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The moon unit will be divided into two divisions.
Phase 1 defused. How about the next one?
```

Phase_2

Set the break point in the phase_2 and explode_bomb just like in the phase_1 and run the file name.txt eg:“answer.txt “ of the phase_1 we saved in. And run the program giving input string as we don’t know the input format of the phase_2 answer. Eg: morning

```
Reading symbols from bomb...
(gdb) b phase_2
Breakpoint 1 at 0x400ea9
(gdb) b explode_bomb
Breakpoint 2 at 0x40143d
(gdb) r answer.txt
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb answer.txt
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?
morning

Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(gdb)
```

Go to disas of the phase_2 .

```
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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>:    callq  0x40143d <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>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
```

Then go to the line 25 using until * address to see the format of the phase_2 answer.

```
(gdb) until * 0x0000000000400ec2
0x0000000000400ec2 in phase_2 ()
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
=> 0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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>:    callq  0x40143d <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>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
```

Enter si command (step into) to check the input format. And x/s 0x4025c3 to see how many integer are there in the phase_2. So from this we get a hint that the phase_2 contains six integer as an answer to defused the bomb.

```
(gdb) si
0x000000000040145f in read_six_numbers ()
(gdb) disas
Dump of assembler code for function read_six_numbers:
=> 0x000000000040145f <+0>:     sub    $0x8,%rsp
   0x0000000000401463 <+4>:     mov    %rsi,%rdx
   0x0000000000401466 <+7>:     lea    0x4(%rsi),%rcx
   0x000000000040146a <+11>:    lea    0x14(%rsi),%rax
   0x000000000040146e <+15>:    push   %rax
   0x000000000040146f <+16>:    lea    0x10(%rsi),%rax
   0x0000000000401473 <+20>:    push   %rax
   0x0000000000401474 <+21>:    lea    0xc(%rsi),%r9
   0x0000000000401478 <+25>:    lea    0x8(%rsi),%r8
   0x000000000040147c <+29>:    mov    $0x4025c3,%esi
   0x0000000000401481 <+34>:    mov    $0x0,%eax
   0x0000000000401486 <+39>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x000000000040148b <+44>:    add    $0x10,%rsp
   0x000000000040148f <+48>:    cmp    $0x5,%eax
   0x0000000000401492 <+51>:    jg     0x401499 <read_six_numbers+58>
   0x0000000000401494 <+53>:    callq  0x40143d <explode_bomb>
   0x0000000000401499 <+58>:    add    $0x8,%rsp
   0x000000000040149d <+62>:    retq
End of assembler dump.
(gdb) x/s 0x4025c3
0x4025c3:       "%d %d %d %d %d %d"
```

After knowing the answer format, give any six random integer number as a input. Then to to disas assembly language of the file.

```
(gdb) r answer.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb answer.txt
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?
0 1 2 3 4 5

Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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>:    callq  0x40143d <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>
```

Then go *util * 0x0000000000400ec7* directly to save our time rather than proceeding with next instruction (ni). Then, go to the assembly code of that particular address. We have to compared the eax value zero with the rsp value. If the rsp value is not equal to than 0 then, it will execute the line 43, else bomb will get exploded.

```
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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>:    callq  0x40143d <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>:    callq  0x400b00 <__stack_chk_fail@plt
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
```

Here, in info register the value of rsp is 0 which is equal to the value of 0, so It will execute line 36and we can say that the first integer we gave is correct.

```
End of assembler dump.
(gdb) x/d $rsp
0x7fffffffddc0: 0
(gdb)
```

Execute the compared function. Compared the value of 1 with the rsp value plus 4. If both the value is equal, it will execute line 48 or else bomb will get exploded.

```
(gdb) u* 0x0000000000400ecd
0x0000000000400ecd in phase_2 ()
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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)
```

The value value of rsp plus 4 is 1. since the value are equal it will execute line 48. Here, the second input integer value is correct.

```
(gdb) i r
rax            0x6                 6
rbx            0x7fffffffdef8      140737488346872
rcx            0x0                 0
rdx            0x7fffffffddd4      140737488346580
rsi            0x0                 0
rdi            0x7fffffffd750      140737488344912
rbp            0x0                 0x0
rsp            0x7fffffffddc0      0x7fffffffddc0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f60ac0      140737353484992
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffdef0      140737488346864
r14            0x0                 0
r15            0x0                 0
rip            0x400ecd            0x400ecd <phase_2+36>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) x/d 0x7fffffffddc4
0x7fffffffddc4: 1
```

Go diretly util* address of the line 61 where there is a compared function. If the value of eax and rbs plus 8 is equal, it will execute line 71, else bomb will get exploded.

```
(gdb) u*  0x0000000000400ee6
0x0000000000400ee6 in phase_2 ()
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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>:    callq  0x40143d <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>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
```

Since the value of eax is 1 and the value of rbx plus 8 is is 2 which is not equal, the bomb will get exploded. Thus, we know that the third input string should be 1. So we will run the program again by giving the third integer as 1.

```
(gdb) i r
rax            0x1                 1
rbx            0x7fffffffddc0      140737488346560
rcx            0x0                 0
rdx            0x7fffffffddd4      140737488346580
rsi            0x0                 0
rdi            0x7fffffffd750      140737488344912
rbp            0x7fffffffddd0      0x7fffffffddd0
rsp            0x7fffffffddc0      0x7fffffffddc0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f60ac0      140737353484992
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffdef0      140737488346864
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  0x7fffffffddc8
0x7fffffffddc8: 2
```

After getting the correct answer, execute the line 75 using util* address of line 75. In this, if the value of rbp and rbx is not eqal it will excute line 56 and if it is equal It will execute next instructions. As the value of rbp and rbx is is not it will execute line 56.

```
=> 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>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) x/d $rbp
0x7fffffffddd0: 4
(gdb) x/d $rbx
0x7fffffffddc4: 1
```

Again, here the value of eax is 2 and 8 plus rbx is 3 which not equal. Thus, the bomb will get exploded. So we know that the fourth value should be 2.

Run the program again by entering the fourth value 2. Then the line 71 will executed .So the phase_2 is all about looping again and again unless we get the same value set by the Dr.Evil.

```
Rhythmbox
mbler dump.
(
rax            0x2                 2
rbx            0x7fffffffddc4      140737488346564
rcx            0x0                 0
rdx            0x7fffffffddd4      140737488346580
rsi            0x0                 0
rdi            0x7fffffffd750      140737488344912
rbp            0x7fffffffddd0      0x7fffffffddd0
rsp            0x7fffffffddc0      0x7fffffffddc0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f60ac0      140737353484992
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffdef0      140737488346864
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  0x7fffffffddcc
0x7fffffffddcc: 3
```

Execute the line 74 where there is compared function. Go to u* address of line 75 and If rbp and rbx value is not equal it will again execute line 56 or if the value is equal it will execute the next instruction.

```
(gdb) u*    0x0000000000400ef4
0x0000000000400ef4 in phase_2 ()
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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>:    callq  0x40143d <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>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
```

As the value of rbp is 4 and that of rbx is 1 in information register which is not equal, so it will execute line 56.

```
(gdb) i r
rax            0x1                 1
rbx            0x7fffffffddc4      140737488346564
rcx            0x0                 0
rdx            0x7fffffffddd4      140737488346580
rsi            0x0                 0
rdi            0x7fffffffd750      140737488344912
rbp            0x7fffffffddd0      0x7fffffffddd0
rsp            0x7fffffffddc0      0x7fffffffddc0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f60ac0      140737353484992
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffdef0      140737488346864
r14            0x0                 0
r15            0x0                 0
rip            0x400ef4            0x400ef4 <phase_2+75>
eflags         0x202               [ IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) x/d $rbp
0x7fffffffddd0: 4
(gdb) x/d $rbx
0x7fffffffddc4: 1
```

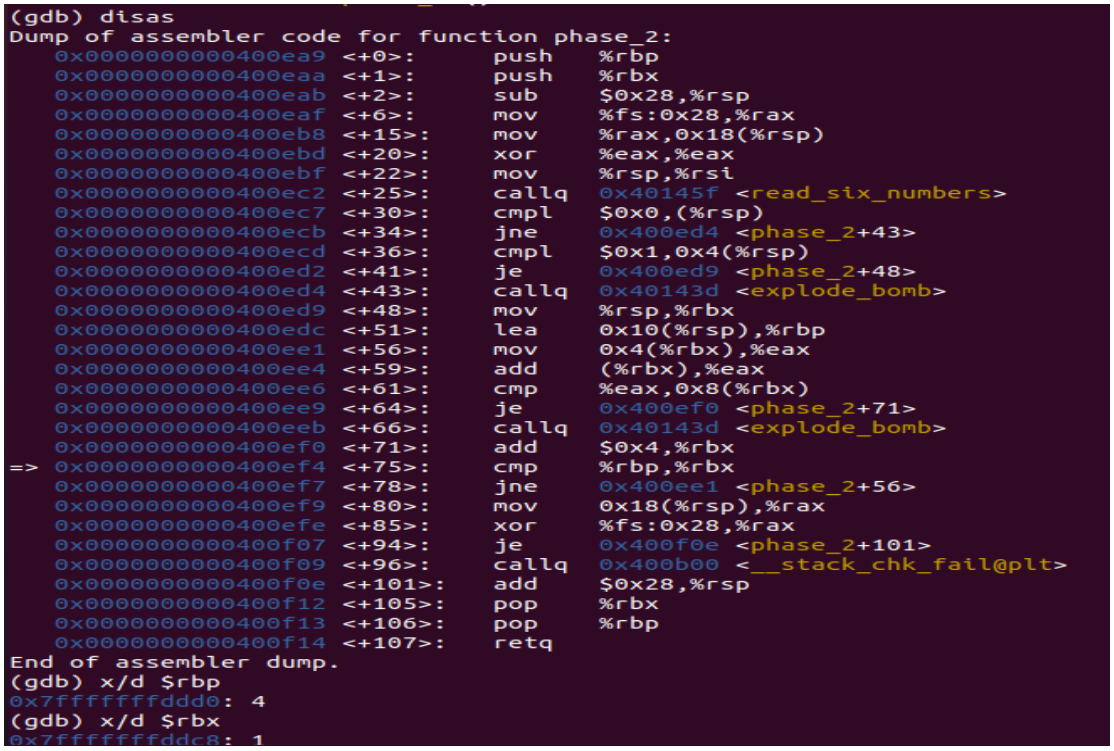
Now go to u* address of line 61. If the value of eax and 8 plus rbx is equal, it will execute line 71 or else the bomb will get exploded.

```
(gdb) u*   0x0000000000400ee6
0x0000000000400ee6 in phase_2 ()
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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>:    callq  0x40143d <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>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
```

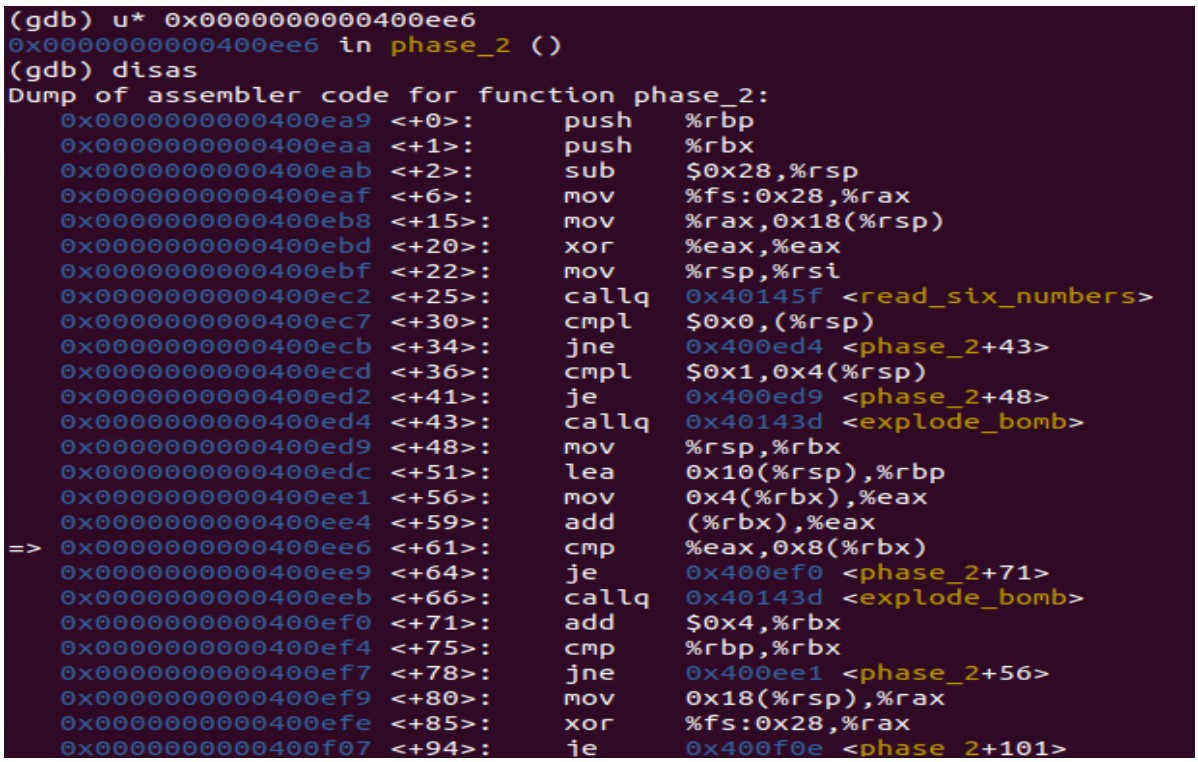
The value of eax is 2 and the 8 plus rbx is 2 which is equal, so it will execute the line 71.

```
(gdb) i r
rax            0x2                 2
rbx            0x7fffffffddc4      140737488346564
rcx            0x0                 0
rdx            0x7fffffffddd4      140737488346580
rsi            0x0                 0
rdi            0x7fffffffd750      140737488344912
rbp            0x7fffffffddd0      0x7fffffffddd0
rsp            0x7fffffffddc0      0x7fffffffddc0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f60ac0      140737353484992
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffdef0      140737488346864
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  0x7fffffffddcc
0x7fffffffddcc: 2
```

Go to u* address of line 75. If the value of rbp and rbx is equal it will proceed to execute next line but it the value is not equal it will loop again to execute line 56.



Go to u* address of line 61 where there is compared function. If the eax and eax plus 8 value is equal it will excute line 71 else bomb will get exploded.



Since the value of eax is 3 and that of eax plus 8 in information register is 4 which is not equal leading to bomb exploitation. Thus, we know that the fourth value should be 3.

```
End of assembler dump.
(gdb) i r
rax            0x3                 3
rbx            0x7fffffffddc8      140737488346568
rcx            0x0                 0
rdx            0x7fffffffddd4      140737488346580
rsi            0x0                 0
rdi            0x7fffffffd750      140737488344912
rbp            0x7fffffffddd0      0x7fffffffddd0
rsp            0x7fffffffddc0      0x7fffffffddc0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f60ac0      140737353484992
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffdef0      140737488346864
r14            0x0                 0
r15            0x0                 0
rip            0x400ee6            0x400ee6 <phase_2+61>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

```
(gdb) x/d  0x7fffffffddd0
0x7fffffffddd0: 4
(gdb)
```

Run the program again giving the 3 as fourth input value. Then go to u* 0x0000000000400ef4 the address of line 75. if the value of rbp and rbx is not equal, it will loop to execute line 56 or else it will proceed with next instruction if the value is equal.

```
(gdb) u*  0x0000000000400ef4
0x0000000000400ef4 in phase_2 ()
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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>:    callq  0x40143d <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>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
```

As the value of rbp is 3 where that of rbx is 1 which is not equal, so it will execute the line 56 again.

So go u* address of line 61. If the eax and 8 plus rbx value is equal it will execute line 71 or the bomb will get exploded.

```
(gdb) x/d $rbp
0x7fffffffddd0: 3
(gdb) x/d $rbx
0x7fffffffddc4: 1
(gdb) u*  0x0000000000400ee6
0x0000000000400ee6 in phase_2 ()
(gdb)
0x0000000000400ee6 in phase_2 ()
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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>:    callq  0x40143d <explode_bomb>
   0x0000000000400ef0 <+71>:    add    $0x4,%rbx
```

Go to u* address of line 75 where there is a compared function. If the value of rbp and rbx is equal it will proceed with next instruction but it is not equal it will loop again to execute line 56. The value of rbp is 3 where as bx is 2 which is not equal, so it will execute line 56.

```
   0x0000000000400ee6 <+61>:    cmp    %eax,0x8(%rbx)
   0x0000000000400ee9 <+64>:    je     0x400ef0 <phase_2+71>
   0x0000000000400eeb <+66>:    callq  0x40143d <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>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0e <+101>:   add    $0x28,%rsp
   0x0000000000400f12 <+105>:   pop    %rbx
   0x0000000000400f13 <+106>:   pop    %rbp
   0x0000000000400f14 <+107>:   retq
End of assembler dump.
(gdb) x/d $rbp
0x7fffffffddd0: 3
(gdb) x/d $rbx
0x7fffffffddcc: 2
```

Ubuntu Software

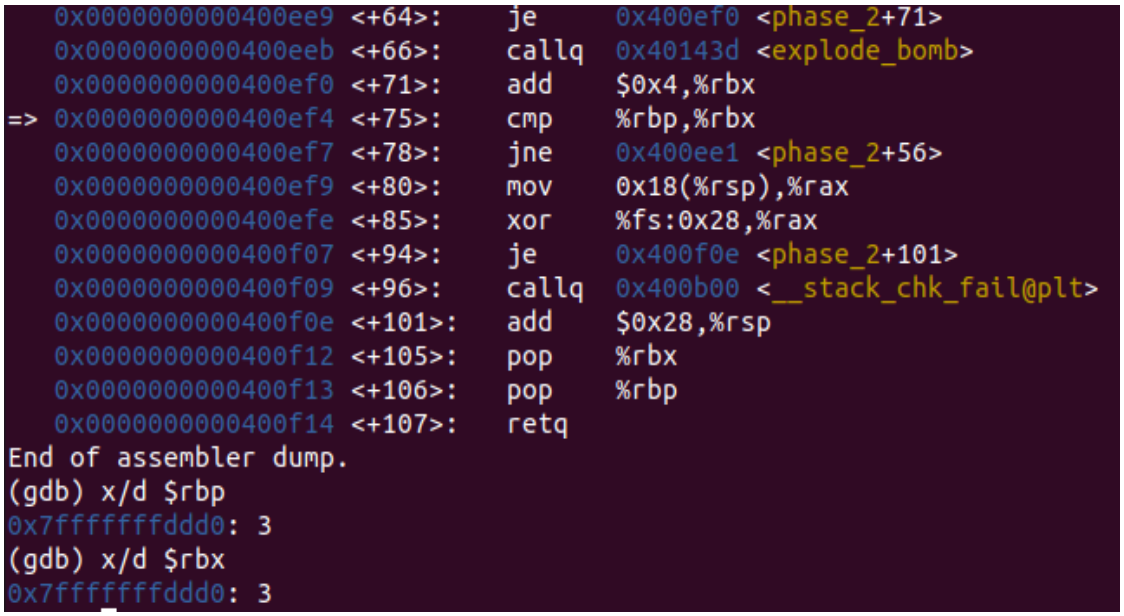
u* address of line 61 and then go the disas of the line 60. Line 71 will be executed if eax value and rbx plus 8 value is equal. Otherwise, bomb will explode.

```
(gdb) u*  0x0000000000400ee6
0x0000000000400ee6 in phase_2 ()
(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
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    callq  0x40145f <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  0x40143d <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>:    callq  0x40143d <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
```

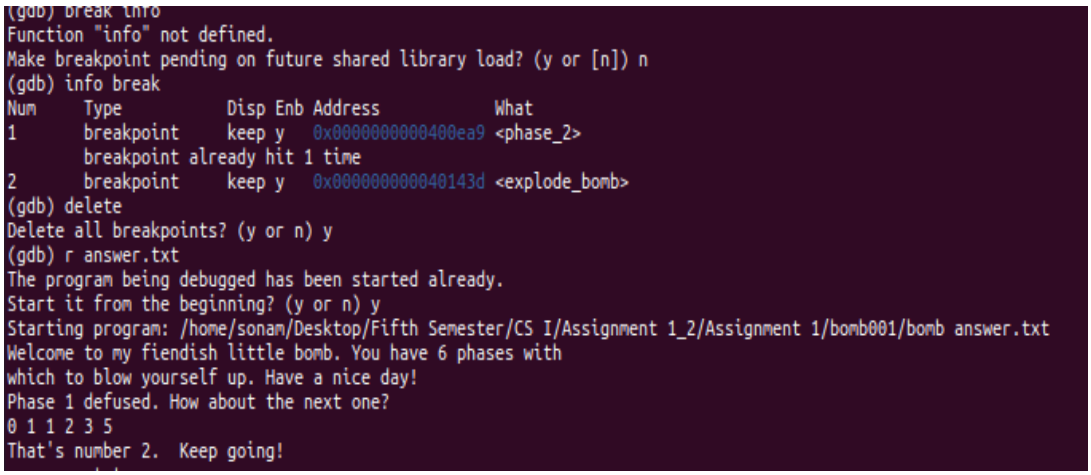
The eax value is 5 as well as rbx plus 8, line 71 will be executed.

```
(gdb) i r
rax            0x5                 5
rbx            0x7fffffffddcc      140737488346572
rcx            0x0                 0
rdx            0x7fffffffddd4      140737488346580
rsi            0x0                 0
rdi            0x7fffffffd750      140737488344912
rbp            0x7fffffffddd0      0x7fffffffddd0
rsp            0x7fffffffddc0      0x7fffffffddc0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f60ac0      140737353484992
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffdef0      140737488346864
r14            0x0                 0
r15            0x0                 0
rip            0x400ee6            0x400ee6 <phase_2+61>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) x/d 0x7fffffffddd4
0x7fffffffddd4: 5
```

The line 75 which have compared function will be executed. The value of rbp and rbx is equal, so sixth value we gave is right.



Finally the phase_2 will get defused when we enter the Fibonacci series we got from the above.



Phase_3

Set the break point at phase_3 as well as explode_bomb. Then go the disassemble file of the phase_3.

```
(gdb) b phase_3
Breakpoint 3 at 0x400f15
(gdb) b explode_bomb
Breakpoint 4 at 0x40143d
(gdb) disas phase_3
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    $0x4025cf,%esi
   0x0000000000400f36 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f3b <+38>:    cmp    $0x1,%eax
   0x0000000000400f3e <+41>:    jg     0x400f45 <phase_3+48>
   0x0000000000400f40 <+43>:    callq  0x40143d <explode_bomb>
   0x0000000000400f45 <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400f49 <+52>:    ja     0x400fa6 <phase_3+145>
   0x0000000000400f4b <+54>:    mov    (%rsp),%eax
   0x0000000000400f4e <+57>:    jmpq   *0x402440(,%rax,8)
   0x0000000000400f55 <+64>:    mov    $0x134,%eax
   0x0000000000400f5a <+69>:    jmp    0x400f61 <phase_3+76>
   0x0000000000400f5c <+71>:    mov    $0x0,%eax
   0x0000000000400f61 <+76>:    sub    $0x85,%eax
   0x0000000000400f66 <+81>:    jmp    0x400f6d <phase_3+88>
   0x0000000000400f68 <+83>:    mov    $0x0,%eax
   0x0000000000400f6d <+88>:    add    $0x201,%eax
   0x0000000000400f72 <+93>:    jmp    0x400f79 <phase_3+100>
   0x0000000000400f74 <+95>:    mov    $0x0,%eax
   0x0000000000400f79 <+100>:   sub    $0x68,%eax
   0x0000000000400f7c <+103>:   jmp    0x400f83 <phase_3+110>
   0x0000000000400f7e <+105>:   mov    $0x0,%eax
   0x0000000000400f83 <+110>:   add    $0x68,%eax
   0x0000000000400f86 <+113>:   jmp    0x400f8d <phase_3+120>
   0x0000000000400f88 <+115>:   mov    $0x0,%eax
   0x0000000000400f8d <+120>:   sub    $0x68,%eax
   0x0000000000400f90 <+123>:   jmp    0x400f97 <phase_3+130>
   0x0000000000400f92 <+125>:   mov    $0x0,%eax
   0x0000000000400f97 <+130>:   add    $0x68,%eax
   0x0000000000400f9a <+133>:   jmp    0x400fa1 <phase_3+140>
   0x0000000000400f9c <+135>:   mov    $0x0,%eax
   0x0000000000400fa1 <+140>:   sub    $0x68,%eax
   0x0000000000400fa4 <+143>:   jmp    0x400fb0 <phase_3+155>
   0x0000000000400fa6 <+145>:   callq  0x40143d <explode_bomb>
```

Run the program with phase_1 answer saved. Give the random string as the input a we are not aware of the input format of the phase_3.

```
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb answer.txt
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?
0 1 1 2 3 5
That's number 2.  Keep going!
heyy

Breakpoint 3, 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    $0x4025cf,%esi
   0x0000000000400f36 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f3b <+38>:    cmp    $0x1,%eax
   0x0000000000400f3e <+41>:    jg     0x400f45 <phase_3+48>
   0x0000000000400f40 <+43>:    callq  0x40143d <explode_bomb>
   0x0000000000400f45 <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400f49 <+52>:    ja     0x400fa6 <phase_3+145>
   0x0000000000400f4b <+54>:    mov    (%rsp),%eax
   0x0000000000400f4e <+57>:    jmpq   *0x402440(,%rax,8)
   0x0000000000400f55 <+64>:    mov    $0x134,%eax
   0x0000000000400f5a <+69>:    jmp    0x400f61 <phase_3+76>
   0x0000000000400f5c <+71>:    mov    $0x0,%eax
   0x0000000000400f61 <+76>:    sub    $0x85,%eax
   0x0000000000400f66 <+81>:    jmp    0x400f6d <phase_3+88>
   0x0000000000400f68 <+83>:    mov    $0x0,%eax
   0x0000000000400f6d <+88>:    add    $0x201,%eax
   0x0000000000400f72 <+93>:    jmp    0x400f79 <phase_3+100>
```

Then run x/s 0x4025cf to see the answer input format contained by the phase_3. From this command we came to know that the answer format of phase_3 contains two integer value.

```
(gdb) x/s 0x4025cf
0x4025cf:       "%d %d"
(gdb)
0x4025d5:       "Error: Premature EOF on stdin"
(gdb) r answer.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb answer.txt
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?
0 1 1 2 3 5
That's number 2.  Keep going!
2 9
```

After getting hint about the answer format of the phase_3, we can run the program and give any two integer value.

```
(gdb) r answer.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb answer.txt
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?
0 1 1 2 3 5
That's number 2.  Keep going!
2 9
```

Then go directly the function having compared value by u* address of the line 38. If the value is eax is greater than 1, then line 145 will get executed, otherwise line 48 will get executed.

```
(gdb) u*  0x0000000000400f3b
0x0000000000400f3b 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    $0x4025cf,%esi
   0x0000000000400f36 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
=> 0x0000000000400f3b <+38>:    cmp    $0x1,%eax
   0x0000000000400f3e <+41>:    jg     0x400f45 <phase_3+48>
   0x0000000000400f40 <+43>:    callq  0x40143d <explode_bomb>
   0x0000000000400f45 <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400f49 <+52>:    ja     0x400fa6 <phase_3+145>
   0x0000000000400f4b <+54>:    mov    (%rsp),%eax
   0x0000000000400f4e <+57>:    jmpq   *0x402440(,%rax,8)
   0x0000000000400f55 <+64>:    mov    $0x134,%eax
   0x0000000000400f5a <+69>:    jmp    0x400f61 <phase_3+76>
   0x0000000000400f5c <+71>:    mov    $0x0,%eax
   0x0000000000400f61 <+76>:    sub    $0x85,%eax
   0x0000000000400f66 <+81>:    jmp    0x400f6d <phase_3+88>
   0x0000000000400f68 <+83>:    mov    $0x0,%eax
   0x0000000000400f6d <+88>:    add    $0x201,%eax
   0x0000000000400f72 <+93>:    jmp    0x400f79 <phase_3+100>
   0x0000000000400f74 <+95>:    mov    $0x0,%eax
   0x0000000000400f79 <+100>:   sub    $0x68,%eax
   0x0000000000400f7c <+103>:   jmp    0x400f83 <phase_3+110>
   0x0000000000400f7e <+105>:   mov    $0x0,%eax
   0x0000000000400f83 <+110>:   add    $0x68,%eax
```

The value of eax is 2 which is greater than 1, so line 48 will get executed.

```
End of assembler dump.
(gdb) i r
rax            0x2                 2
rbx            0x7fffffffdef8      140737488346872
rcx            0x0                 0
rdx            0x7fffffffdde4      140737488346596
rsi            0x0                 0
rdi            0x7fffffffd790      140737488344976
rbp            0x0                 0x0
rsp            0x7fffffffdde0      0x7fffffffdde0
r8             0xffffffff          4294967295
r9             0x0                 0
r10            0x7ffff7f60ac0      140737353484992
r11            0x0                 0
r12            0x400c60            4197472
r13            0x7fffffffdef0      140737488346864
r14            0x0                 0
r15            0x0                 0
rip            0x400f3b            0x400f3b <phase_3+38>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) u*    0x0000000000400f45
0x0000000000400f45 in phase_3 ()
```

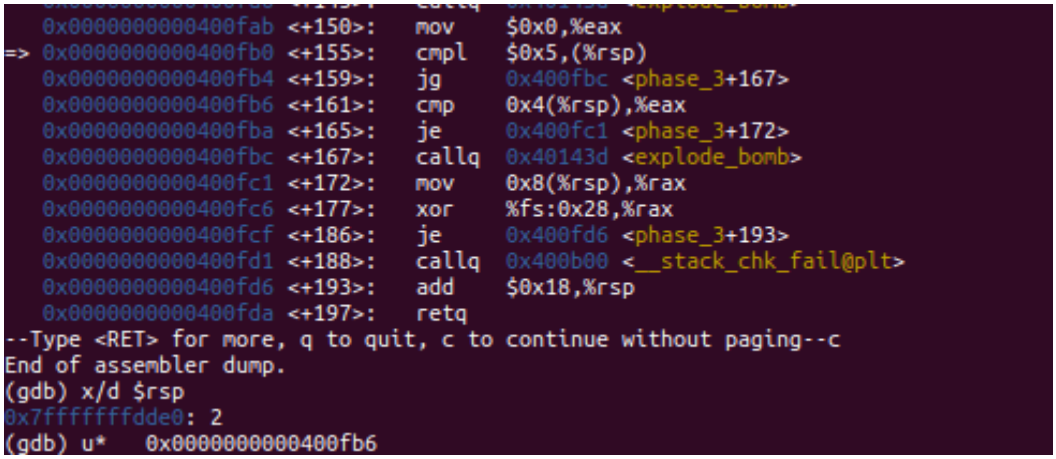
Go to line 48 by unil* 0x0000000000400f45 . The value 7 will be compared with the value of rsp. Incase rsp value is greater than 7 then, line 14 5 will get executed.

```
(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    $0x4025cf,%esi
   0x0000000000400f36 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f3b <+38>:    cmp    $0x1,%eax
   0x0000000000400f3e <+41>:    jg     0x400f45 <phase_3+48>
   0x0000000000400f40 <+43>:    callq  0x40143d <explode_bomb>
=> 0x0000000000400f45 <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400f49 <+52>:    ja     0x400fa6 <phase_3+145>
   0x0000000000400f4b <+54>:    mov    (%rsp),%eax
   0x0000000000400f4e <+57>:    jmpq   *0x402440(,%rax,8)
   0x0000000000400f55 <+64>:    mov    $0x134,%eax
   0x0000000000400f5a <+69>:    jmp    0x400f61 <phase_3+76>
   0x0000000000400f5c <+71>:    mov    $0x0,%eax
   0x0000000000400f61 <+76>:    sub    $0x85,%eax
   0x0000000000400f66 <+81>:    jmp    0x400f6d <phase_3+88>
   0x0000000000400f68 <+83>:    mov    $0x0,%eax
   0x0000000000400f6d <+88>:    add    $0x201,%eax
```

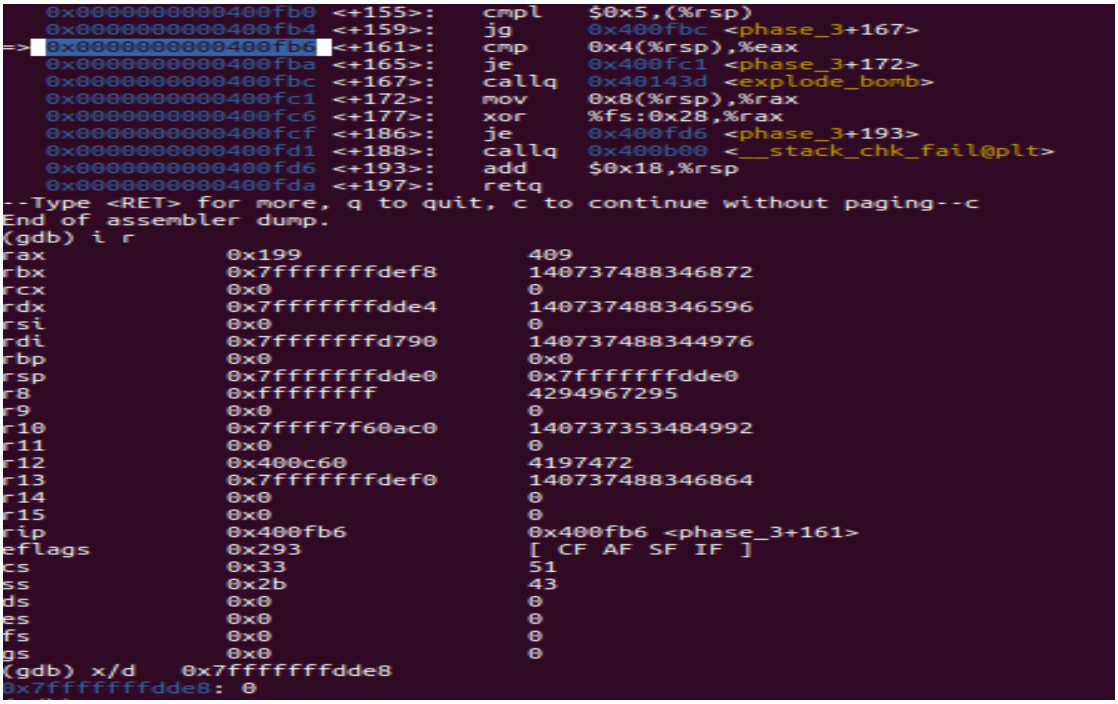
As the rsp value is 2 which Is lesser than 7, it will execute next instruction or the line 155. Go to u* address of line 155..

```
(gdb) x/d $rsp
0x7fffffffdde0: 2
(gdb) u*    0x0000000000400fb0
0x0000000000400fb0 in phase_3 ()
```

The compared function will compare the 5 with that of value for rsp. If rsp is greater than the 7, line 167 will executed. Since rsp value is 2 which is less than 7, so line 16 1will get executed.
Go to u* 0x0000000000400fb6 .



The compared function will again compare the 4 plus rsp value and eax value. If the value is equal, line 172 will get executed, otherwise, bomb will get exploded. As the value of 4 plus rsp is 0 where as eax value is 409 which is not equal. So in order to have equal value, we can substitute value 409 as eax value to balanced the value of 4 plus rsp.



After getting the second integer value, we can run the program and before defusing phase_3, delete the break point, otherwise bomb will not get defused. Thus phase_3 of bomb001 is also defused.

```
(gdb) r answer.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb answer.txt
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?
0 1 1 2 3 5
That's number 2.  Keep going!

2 409

Breakpoint 3, 0x0000000000400f15 in phase_3 ()
(gdb) info break
Num     Type           Disp Enb Address            What
3       breakpoint     keep y   0x0000000000400f15 <phase_3>
        breakpoint already hit 1 time
4       breakpoint     keep y   0x000000000040143d <explode_bomb>
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) r answer.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/sonam/Desktop/Fifth Semester/CS I/Assignment 1_2/Assignment 1/bomb001/bomb answer.txt
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?
0 1 1 2 3 5
That's number 2.  Keep going!
2 409
Halfway there!
```