

# Binary Bomb(Group 3)

## Phase 03

### “Final Blow”

First phase was about the string, second was about giving the correct six numbers. Now the third phase. Phase 3 is all about giving two integer. If the two integers are correct than the phase\_3 bomb will be diffused. Else it will explode.

First of all to know if the input is two numbers we have to open the “gdb bomb” and set break point at phase three “b phase\_3” and run the answers.txt file and than input the phase\_2 numbers and after that give some random inputs. After that we have to open the disassembler and the system will show the assembly code present in phase\_3

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
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)
0x0000000000400f49 <+52>:     ja      0x400f86 <phase_3+113>
0x0000000000400f4b <+54>:     mov     (%rsp),%eax
0x0000000000400f4e <+57>:     jmpq    *0x402420(,%rax,8)
0x0000000000400f55 <+64>:     mov     $0xc6,%eax
0x0000000000400f5a <+69>:     jmp     0x400f97 <phase_3+130>
0x0000000000400f5c <+71>:     mov     $0x31e,%eax
0x0000000000400f61 <+76>:     jmp     0x400f97 <phase_3+130>
0x0000000000400f63 <+78>:     mov     $0x299,%eax
0x0000000000400f68 <+83>:     jmp     0x400f97 <phase_3+130>
0x0000000000400f6a <+85>:     mov     $0x3a,%eax
--Type <RET> for more, q to quit, c to continue without paging--
```

First we will look at line <+33> where there is callq function and it calls “scanf” so next lets look at the line above it at line <+28>. We can see that there is one hex address. So we can write “x/s the hex number” in this case “x/s 0x4025af” After giving this command we will get a clue saying that the input for phase\_3 are two integers.

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
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>
0x0000000000400f4b <+54>: mov    (%rsp),%eax
0x0000000000400f4e <+57>: jmpq   *0x402420(,%rax,8)
0x0000000000400f53 <+64>: mov    $0xc6,%eax
0x0000000000400f5a <+69>: jmp    0x400f97 <phase_3+130>
0x0000000000400f5c <+71>: mov    $0x31e,%eax
0x0000000000400f61 <+76>: jmp    0x400f97 <phase_3+130>
0x0000000000400f63 <+78>: mov    $0x299,%eax
0x0000000000400f68 <+83>: jmp    0x400f97 <phase_3+130>
0x0000000000400f6a <+85>: mov    $0x3a,%eax
--Type <RET> for more, q to quit, c to continue without paging--
0x0000000000400f6f <+90>: jmp    0x400f97 <phase_3+130>
0x0000000000400f71 <+92>: mov    $0x270,%eax
0x0000000000400f76 <+97>: jmp    0x400f97 <phase_3+130>
0x0000000000400f78 <+99>: mov    $0x10b,%eax
0x0000000000400f7d <+104>: jmp    0x400f97 <phase_3+130>
0x0000000000400f7f <+106>: mov    $0x80,%eax
0x0000000000400f84 <+111>: jmp    0x400f97 <phase_3+130>
0x0000000000400f86 <+113>: callq  0x40142a <explode_bomb>
0x0000000000400f8b <+118>: mov    $0x0,%eax
0x0000000000400f90 <+123>: jmp    0x400f97 <phase_3+130>
0x0000000000400f92 <+125>: mov    $0x3f,%eax
0x0000000000400f97 <+130>: cmp    0x4(%rsp),%eax
0x0000000000400f9b <+134>: je     0x400fa2 <phase_3+141>
0x0000000000400f9d <+136>: callq  0x40142a <explode_bomb>
0x0000000000400fa2 <+141>: mov    0x8(%rsp),%rax
0x0000000000400fa7 <+146>: xor    %fs:0x28,%rax
0x0000000000400fab <+155>: je     0x400fb7 <phase_3+162>
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 got the clue that the inputs are two integers as there is two ““%d” “%d””.

Now lets see the compare code below the line <+33>. We have to compare (1 and %eax). We can move directly to line <+38> by commanding “until\* address” After that we don’t know the value of eax so we can find it with the help of information register.

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
(gdb) disas
Dump of assembler code for function phase_3:
0x00000000400f11: <+0>: sub    $0x18,%rsp
0x00000000400f13: <+4>: mov    %fs:0x28,%rax
0x00000000400f15: <+8>: mov    %rax,0x8(%rsp)
0x00000000400f17: <+12>: xor    %eax,%eax
0x00000000400f19: <+16>: lea    0x4(%rsp),%rcx
0x00000000400f1b: <+20>: mov    %rsp,%rdx
0x00000000400f1d: <+24>: mov    $0x4025af,%esi
0x00000000400f1f: <+28>: callq  0x400bb0 <__isoc99_sscanf@plt>
=> 0x00000000400f21: <+32>: cmp    $0x1,%eax
0x00000000400f23: <+36>: jg     0x400f45 <phase_3+48>
0x00000000400f25: <+40>: callq  0x40142a <explode_bomb>
0x00000000400f27: <+44>: cmpl   $0x7,(%rsp)
0x00000000400f29: <+48>: ja     0x400f86 <phase_3+113>
0x00000000400f2b: <+52>: mov    (%rsp),%eax
0x00000000400f2d: <+56>: jmpq   *0x402420(,%rax,8)
0x00000000400f2f: <+60>: mov    $0xc6,%eax
0x00000000400f31: <+64>: jmp    0x400f97 <phase_3+130>
0x00000000400f33: <+68>: mov    $0x31e,%eax
0x00000000400f35: <+72>: jmp    0x400f97 <phase_3+130>
0x00000000400f37: <+76>: mov    $0x299,%eax
0x00000000400f39: <+80>: jmp    0x400f97 <phase_3+130>
0x00000000400f3b: <+84>: mov    $0x3a,%eax
0x00000000400f3d: <+88>: jmp    0x400f97 <phase_3+130>
0x00000000400f3f: <+92>: mov    $0x270,%eax
0x00000000400f41: <+96>: jmp    0x400f97 <phase_3+130>
0x00000000400f43: <+100>: mov    $0x10b,%eax
0x00000000400f45: <+104>: jmp    0x400f97 <phase_3+130>
0x00000000400f47: <+108>: mov    $0x80,%eax
0x00000000400f49: <+112>: jmp    0x400f97 <phase_3+130>
0x00000000400f4b: <+116>: callq  0x40142a <explode_bomb>
0x00000000400f4d: <+120>: mov    $0x0,%eax
0x00000000400f4f: <+124>: jmp    0x400f97 <phase_3+130>
0x00000000400f51: <+128>: mov    $0x3f,%eax
--Type <RET> for more, q to quit, c to continue without paging--
```

Lets trace into the assembly code so first lets go to line <+38> which compares (1 and %eax) and the line below that states a function which says jg( it is unsigned and it jump if greater than). We can trace out that if the input is less than 1 the bomb will be exploded. So the first digit should be greater than or equal to 1. If it is greater the function will move to line <+41> and call the “jg” function. After that the jg function will move to line <+48> where it compares (7 and %rsp) . The operations done here will impact the next function, if the value of rsp is greater than 7. If the value of rsp is greater than 7 than the function will move towards <+113> and the bomb will be exploded.

So we got one big clue that says the first input should be greater than or equal to 1 and less than or equal to 7.

Lets run the program and give the first input as any number from range (1-7) since the number should be greater than or equal to 1 and less than or equal to 7.

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
eip 0x400137 0x400137 ~phase_3+130
eflags 0x297 [ CF PF AF SF IF ]
cs 0x33 51
ss 0x2b 43
ds 0x0 0
es 0x0 0
fs 0x0 0
gs 0x0 0
(gdb) r answers.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kushal/Desktop/bomb/Assignment 1/bomb003/bomb answers.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 5
That's number 2. Keep going!
2 999

Breakpoint 1, 0x00000000400f10 in phase_3 ()
(gdb) disas
Dump of assembler code for function phase_3:
=> 0x00000000400f10 <+0>: sub $0x18,%rsp
0x00000000400f19 <+4>: mov %fs:0x28,%rax
0x00000000400f22 <+13>: mov %rax,0x8(%rsp)
0x00000000400f27 <+18>: xor %eax,%eax
0x00000000400f29 <+20>: lea 0x4(%rsp),%rcx
0x00000000400f2a <+25>: mov %rsp,%rdx
0x00000000400f31 <+28>: mov $0x4025af,%esi
0x00000000400f36 <+33>: callq 0x400bb0 <__isoc99_sscanf@plt>
0x00000000400f3b <+38>: cmp $0x1,%eax
0x00000000400f3c <+41>: jg 0x400f45 <phase_3+48>
0x00000000400f46 <+43>: callq 0x40142a <explode_bomb>
0x00000000400f4f <+48>: cmpl $0x7,(%rsp)
0x00000000400f54 <+52>: ja 0x400f86 <phase_3+113>
0x00000000400f56 <+54>: mov (%rsp),%eax
0x00000000400f5e <+57>: jmpq *0x402420(,%rax,8)
0x00000000400f64 <+64>: mov $0x6,%eax
```

So in this case I gave the first input as “2”. So lets find the second integer input. For that we have to go to the first compare function at line <+38>. There it will compare the given user input integer with 1. If it is greater than or equal to 1 it will move to the next function which is jg(jump if greater than or equals to) at line<+41>. After that the function will move to line <+48>. There we have to compare our input with “7”. It will check if the number is less than or equal to 7. If the number is greater than 7 it will go to next line and the boob will be exploded. Since right now our input is less than 7 so it will move to line <+54>.

After that we have several mov and jump functions. We will now go to the next compare function which is present at line <+130>. And by opening the information register we get the second integer input, since the second input is be stored in the %eax register.



```

kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x00000000400f80 <+113>: callq 0x40142a <explode_bomb>
0x00000000400f8b <+118>: mov $0x0,%eax
0x00000000400f90 <+123>: jmp 0x400f97 <phase_3+130>
0x00000000400f92 <+125>: mov $0x3f,%eax
=> 0x00000000400f97 <+130>: cmp 0x4(%rsp),%eax
0x00000000400f9b <+134>: je 0x400fa2 <phase_3+141>
--Type <RET> for more, q to quit, c to continue without paging--
0x00000000400f9d <+136>: callq 0x40142a <explode_bomb>
0x00000000400fa2 <+141>: mov 0x8(%rsp),%rax
0x00000000400fa7 <+146>: xor %fs:0x28,%rax
0x00000000400fb0 <+155>: je 0x400fb7 <phase_3+162>
0x00000000400fb2 <+157>: callq 0x400b00 <__stack_chk_fail@plt>
0x00000000400fb7 <+162>: add $0x18,%rsp
0x00000000400fbb <+166>: retq
End of assembler dump.
(gdb) i r
rax            0x31e            798
rbx            0x7fffffffde38    140737488346680
rcx            0x0
rdx            0x7fffffffdd34    140737488346420
rsi            0x0
rdi            0x7fffffffdd6e0    140737488344800
rbp            0x4021e0        0x4021e0 <_libc_csu_init>
rsp            0x7fffffffdd30    0x7fffffffdd30
r8             0xffffffff        4294967295
r9             0x0
r10            0x7ffff7f63ac0    140737353497280
r11            0x0
r12            0x400c60        4197472
r13            0x7fffffffde30    140737488346672
r14            0x0
r15            0x0
rip            0x400f97        0x400f97 <phase_3+130>
eflags        0x293            [ CF AF SF IF ]
cs             0x33            51
ss             0x2b            43
ds             0x0
es             0x0
fs             0x0
gs             0x0
(gdb)

```

Here we can see that for the first input 2 we get the second input as 798. So let's try to put 2 and 798 to see if the bomb will be defused or not.

```

kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) q
A debugging session is active.

Inferior 1 [process 4581] will be killed.

Quit anyway? (y or n) y
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/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!
2 798
Halfway there!

```

So my input was correct. But since this phase\_3 accepts the first inputs in the range of (1-7) we can give any first input in the range (1-7) so for different first input, there will be different second input. Like for 2 it was 798.

As it lets check for every first input in the range of (1-7).

**For First input 1 The second input is 198**

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x00000000040113: callq 0x40142a <explode_bomb>
0x00000000040118: mov  $0x0,%eax
0x00000000040123: jmp  0x400f97 <phase_3+130>
0x00000000040125: mov  $0x3f,%eax
=> 0x00000000040130: cmp  0x4(%rsp),%eax
0x00000000040134: je   0x400fa2 <phase_3+141>
0x00000000040136: callq 0x40142a <explode_bomb>
0x00000000040141: mov  0x8(%rsp),%rax
0x00000000040146: xor  %fs:0x28,%rax
0x00000000040155: je   0x400fb7 <phase_3+162>
--Type <RET> for more, q to quit, c to continue without paging--
0x00000000040157: callq 0x400b00 <__stack_chk_fail@plt>
0x00000000040162: add  $0x18,%rsp
0x00000000040166: retq
End of assembler dump.
(gdb) i r
rax      0xc6      198
rbx      0x7fffffffde38 140737488346680
rcx      0x0
rdx      0x7fffffffdd34 140737488346420
rsi      0x0
rdi      0x7fffffffde0 140737488344800
rbp      0x4021e0 0x4021e0 <__libc_csu_init>
rsp      0x7fffffffdd30 0x7fffffffdd30
r8       0xffffffff 4294967295
r9       0x0
r10      0x7ffff7f63ac0 140737353497280
r11      0x0
r12      0x400c60 4197472
r13      0x7fffffffde30 140737488346672
r14      0x0
r15      0x0
rip      0x400f97 0x400f97 <phase_3+130>
eflags   0x297 [ CF PF AF SF IF ]
cs       0x33 51
ss       0x2b 43
ds       0x0
es       0x0
fs       0x0
gs       0x0
(gdb)
```

**For First input 3 The second input is 655**

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x00000000040113: callq 0x40142a <explode_bomb>
0x00000000040118: mov  $0x0,%eax
0x00000000040123: jmp  0x400f97 <phase_3+130>
0x00000000040125: mov  $0x3f,%eax
=> 0x00000000040130: cmp  0x4(%rsp),%eax
0x00000000040134: je   0x400fa2 <phase_3+141>
0x00000000040136: callq 0x40142a <explode_bomb>
0x00000000040141: mov  0x8(%rsp),%rax
0x00000000040146: xor  %fs:0x28,%rax
0x00000000040155: je   0x400fb7 <phase_3+162>
--Type <RET> for more, q to quit, c to continue without paging--
0x00000000040157: callq 0x400b00 <__stack_chk_fail@plt>
0x00000000040162: add  $0x18,%rsp
0x00000000040166: retq
End of assembler dump.
(gdb) i r
rax      0x299      655
rbx      0x7fffffffde38 140737488346680
rcx      0x0
rdx      0x7fffffffdd34 140737488346420
rsi      0x0
rdi      0x7fffffffde0 140737488344800
rbp      0x4021e0 0x4021e0 <__libc_csu_init>
rsp      0x7fffffffdd30 0x7fffffffdd30
r8       0xffffffff 4294967295
r9       0x0
r10      0x7ffff7f63ac0 140737353497280
r11      0x0
r12      0x400c60 4197472
r13      0x7fffffffde30 140737488346672
r14      0x0
r15      0x0
rip      0x400f97 0x400f97 <phase_3+130>
eflags   0x297 [ CF PF AF SF IF ]
cs       0x33 51
ss       0x2b 43
ds       0x0
es       0x0
fs       0x0
gs       0x0
(gdb)
```

**For First input 4 The second input is 58**

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x0000000000400fb0 <+155>: je 0x400fb7 <phase_3+162>
--Type <RET> for more, q to quit, c to continue without paging--
0x0000000000400fb2 <+157>: callq 0x400b00 <__stack_chk_fail@plt>
0x0000000000400fb7 <+162>: add $0x18,%rsp
0x0000000000400fbb <+166>: retq
End of assembler dump.
(gdb) i r
rax 0x3a 58
rbx 0x7fffffffde38 140737488346680
rcx 0x0 0
rdx 0x7fffffffdd34 140737488346420
rsi 0x0 0
rdi 0x7fffffffdd6e0 140737488344800
rbp 0x4021e0 0x4021e0 <__libc_csu_init>
rsp 0x7fffffffdd30 0x7fffffffdd30
r8 0xffffffff 4294967295
r9 0x0 0
r10 0x7ffff7f63ac0 140737353497280
r11 0x0 0
r12 0x400c60 4197472
r13 0x7fffffffde30 140737488346672
r14 0x0 0
r15 0x0 0
rip 0x400f97 0x400f97 <phase_3+130>
eflags 0x293 [ CF AF SF IF ]
cs 0x33 51
ss 0x2b 43
ds 0x0 0
es 0x0 0
fs 0x0 0
gs 0x0 0
(gdb) □
```

**For First input 5 The second input is 624**

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x0000000000400fa7 <+146>: xor %fs:0x28,%rax
0x0000000000400fb0 <+155>: je 0x400fb7 <phase_3+162>
0x0000000000400fb2 <+157>: callq 0x400b00 <__stack_chk_fail@plt>
0x0000000000400fb7 <+162>: add $0x18,%rsp
0x0000000000400fbb <+166>: retq
End of assembler dump.
(gdb) i r
rax 0x270 624
rbx 0x7fffffffde38 140737488346680
rcx 0x0 0
rdx 0x7fffffffdd34 140737488346420
rsi 0x0 0
rdi 0x7fffffffdd6e0 140737488344800
rbp 0x4021e0 0x4021e0 <__libc_csu_init>
rsp 0x7fffffffdd30 0x7fffffffdd30
r8 0xffffffff 4294967295
r9 0x0 0
r10 0x7ffff7f63ac0 140737353497280
r11 0x0 0
r12 0x400c60 4197472
r13 0x7fffffffde30 140737488346672
r14 0x0 0
r15 0x0 0
rip 0x400f97 0x400f97 <phase_3+130>
eflags 0x293 [ CF AF SF IF ]
cs 0x33 51
ss 0x2b 43
ds 0x0 0
es 0x0 0
fs 0x0 0
gs 0x0 0
(gdb) □
```

**For First input 6 The second input is 267**

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x0000000000400fa7 <+146>: xor    %fs:0x28,%rax
0x0000000000400fb0 <+155>: je     0x400fb7 <phase_3+162>
0x0000000000400fb2 <+157>: callq 0x400b00 <__stack_chk_fail@plt>
0x0000000000400fb7 <+162>: add    $0x18,%rsp
0x0000000000400fbb <+166>: retq

End of assembler dump.
(gdb) i r
rax            0x10b            267
rbx            0x7fffffffde38    140737488346680
rcx            0x0              0
rdx            0x7fffffffdd34    140737488346420
rsi            0x0              0
rdi            0x7fffffffdd6e0    140737488344800
rbp            0x4021e0         0x4021e0 <_libc_csu_init>
rsp            0x7fffffffdd30    0x7fffffffdd30
r8             0xffffffff         4294967295
r9             0x0              0
r10            0x7ffff7f63ac0      140737353497280
r11            0x0              0
r12            0x400c60         4197472
r13            0x7fffffffde30    140737488346672
r14            0x0              0
r15            0x0              0
rip            0x400f97         0x400f97 <phase_3+130>
eflags         0x297             [ CF PF AF SF IF ]
cs             0x33             51
ss             0x2b             43
ds             0x0              0
es             0x0              0
fs             0x0              0
gs             0x0              0
(gdb) 
```

**For First input 7 The second input is 128**

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x0000000000400fa7 <+146>: xor    %fs:0x28,%rax
0x0000000000400fb0 <+155>: je     0x400fb7 <phase_3+162>
0x0000000000400fb2 <+157>: callq 0x400b00 <__stack_chk_fail@plt>
0x0000000000400fb7 <+162>: add    $0x18,%rsp
0x0000000000400fbb <+166>: retq

End of assembler dump.
(gdb) i r
rax            0x80             128
rbx            0x7fffffffde38    140737488346680
rcx            0x0              0
rdx            0x7fffffffdd34    140737488346420
rsi            0x0              0
rdi            0x7fffffffdd6e0    140737488344800
rbp            0x4021e0         0x4021e0 <_libc_csu_init>
rsp            0x7fffffffdd30    0x7fffffffdd30
r8             0xffffffff         4294967295
r9             0x0              0
r10            0x7ffff7f63ac0      140737353497280
r11            0x0              0
r12            0x400c60         4197472
r13            0x7fffffffde30    140737488346672
r14            0x0              0
r15            0x0              0
rip            0x400f97         0x400f97 <phase_3+130>
eflags         0x246             [ PF ZF IF ]
cs             0x33             51
ss             0x2b             43
ds             0x0              0
es             0x0              0
fs             0x0              0
gs             0x0              0
(gdb) 
```



**Phase 3 completed.....**  
**“Dr.Evil see I told you I will crack your bombs”**  
**“Till then I will see you after midterm in assignment 2”**

**THANK YOU**