

Binary Bomb(Group 3)

Phase 02

“Cracking The Code”

The first phase was finding the correct string and matching it with the Dr.Evil's string. If the correct string was given than the bomb at phase 1 will be diffused and if not it will blast. Well the second phase is about giving a set of six inputs. The inputs were found to be “0 1 1 2 3 5” which were the first 6 Fibonacci series numbers. After putting these six digit number in the phase_2 of bomb 003, the bomb diffuses.

Here are the steps to get the six digit code.

First of all it will be easier and less time consuming if we save the acquired string of phase_1 inside a txt file so that we don't have to write the string again and again. I saved it as a answers.txt file.

After diffusing the first phase we have to again go to the gdb debugger and this time we have to set the break-point at phase_2 because now we have to stop at phase_2. After setting the break-point we can run using “**r answers.txt**” since the string of phase_1 is saved inside the answers.txt file so we don't have to type in again and again. After running the program, it will ask input for second phase. Since we don't know the input, we have to give some random inputs.

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003$ gdb bomb
GNU gdb (Ubuntu 8.3-0ubuntu1) 8.3
Copyright (C) 2019 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".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) b phase_2
Breakpoint 1 at 0x400ea9
(gdb) r answers.txt
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?
2 5 3 1 3 5

Breakpoint 1, 0x00000000400ea9 in phase_2 ()
(gdb) □
```

After putting some random inputs, we have to go inside disassemble for finding the real inputs

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
Dump of assembler code for function phase_2:
=> 0x00000000400ea9 <+0>:    push    %rbp
0x00000000400eaa <+1>:    push    %rbx
0x00000000400eab <+2>:    sub     $0x28,%rsp
0x00000000400eaf <+6>:    mov     %fs:0x28,%rax
0x00000000400eb8 <+15>:   mov     %rax,0x18(%rsp)
0x00000000400ebd <+20>:   xor     %eax,%eax
0x00000000400ebf <+22>:   mov     %rsp,%rsi
0x00000000400ec2 <+25>:   callq   0x40144c <read_six_numbers>
0x00000000400ec7 <+30>:   cmpl    $0x0,(%rsp)
0x00000000400ecb <+34>:   jne     0x400ed4 <phase_2+43>
0x00000000400ecd <+36>:   cmpl    $0x1,0x4(%rsp)
0x00000000400ed2 <+41>:   je      0x400ed9 <phase_2+48>
0x00000000400ed4 <+43>:   callq   0x40142a <explode_bomb>
0x00000000400ed9 <+48>:   mov     %rsp,%rbx
0x00000000400edc <+51>:   lea     0x10(%rsp),%rbp
0x00000000400ee1 <+56>:   mov     0x4(%rbx),%eax
0x00000000400ee4 <+59>:   add     (%rbx),%eax
0x00000000400ee6 <+61>:   cmp     %eax,0x8(%rbx)
0x00000000400ee9 <+64>:   je      0x400ef0 <phase_2+71>
0x00000000400eeb <+66>:   callq   0x40142a <explode_bomb>
0x00000000400ef0 <+71>:   add     $0x4,%rbx
0x00000000400ef4 <+75>:   cmp     %rbp,%rbx
0x00000000400ef7 <+78>:   jne     0x400ee1 <phase_2+56>
0x00000000400ef9 <+80>:   mov     0x18(%rsp),%rax
0x00000000400efe <+85>:   xor     %fs:0x28,%rax
0x00000000400f07 <+94>:   je      0x400f0e <phase_2+101>
0x00000000400f09 <+96>:   callq   0x400b00 <__stack_chk_fail@plt>
--Type <RET> for more, q to quit, c to continue without paging--
```

Here in the above image we got a small clue in line <+25> which has a callq function that says <read_six_numbers>. We got the hint that the input for phase_2 is 6 integers. Below the callq function there is cmpl function which compares 0x0 and first integer(%rsp). So lets directly jump into line <+30> and compare the two integers. For directly jumping into the particular line we have to use **(until *address present in that line)** and disas. In this case we will use **(until *0x0000000000400ec7)**, so as we can see we are in line <+30> so lets compare the two integers. Right now we don't know the value stored in the (rsp) register, in order to find the values of the register we can do “i r” command which is information register and displays the information/values of every register.

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x0000000000400ee4 <+59>: add    (%rbx),%eax
0x0000000000400ee6 <+61>: cmp    %eax,0x8(%rbx)
0x0000000000400ee9 <+64>: je     0x400ef0 <phase_2+71>
0x0000000000400eeb <+66>: callq 0x40142a <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>
--Type <RET> for more, q to quit, c to continue without paging--
0x0000000000400f0e <+101>: add    $0x28,%rsp
0x0000000000400f12 <+105>: pop    %rbx
0x0000000000400f13 <+106>: pop    %rbp
0x0000000000400f14 <+107>: retq
End of assembler dump.
(gdb) i r
rax            0x6                6
rbx            0x7fffffffde38    140737488346680
rcx            0x0
rdx            0x7fffffffdd24    140737488346404
rsi            0x0
rdi            0x7fffffffdd6a0    140737488344736
rbp            0x4021e0 <__libc_csu_init>
rsp            0x7fffffffdd10    0x7fffffffdd10
r8             0xffffffff    4294967295
r9             0x0
r10            0x7ffff7f63ac0    140737353497280
r11            0x0
r12            0x400c60        4197472
r13            0x7fffffffde30    140737488346672
r14            0x0
r15            0x0
rip            0x400ec7        0x400ec7 <phase_2+30>
eflags        0x202        [ IF ]
cs            0x33        51
ss            0x2b        43
ds            0x0
es            0x0
fs            0x0
gs            0x0
(gdb)
```

We found the value of rsp register but the value is in hexa form so we have to convert it into decimal form. In-order to convert the number into decimal we have to command “(x/d(number))”.

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x00000000400ee6 <+61>: cmp    %eax,0x8(%rbx)
0x00000000400ee9 <+64>: je     0x400ef0 <phase_2+71>
0x00000000400eeb <+66>: callq 0x40142a <explode_bomb>
0x00000000400ef0 <+71>: add    $0x4,%rbx
0x00000000400ef4 <+75>: cmp    %rbp,%rbx
0x00000000400ef7 <+78>: jne    0x400ee1 <phase_2+56>
0x00000000400ef9 <+80>: mov    0x18(%rsp),%rax
0x00000000400efe <+85>: xor    %fs:0x28,%rax
0x00000000400f07 <+94>: je     0x400f0e <phase_2+101>
0x00000000400f09 <+96>: callq 0x400b00 <__stack_chk_fail@plt>
0x00000000400f0e <+101>: add    $0x28,%rsp
0x00000000400f12 <+105>: pop    %rbx
0x00000000400f13 <+106>: pop    %rbp
0x00000000400f14 <+107>: retq

End of assembler dump.
(gdb) i r
rax                0x6                6
rbx                0x7fffffffde38       140737488346680
rcx                0x0                0
rdx                0x7fffffffdd24       140737488346404
rsi                0x0                0
rdi                0x7fffffffdd6a0      140737488344736
rbp                0x4021e0            0x4021e0 <__libc_csu_init>
rsp                0x7fffffffdd10       0x7fffffffdd10
r8                 0xffffffff        4294967295
r9                 0x0                0
r10                0x7fffff7f63ac0       140737353497280
r11                0x0                0
r12                0x400c60            4197472
r13                0x7fffffffde30       140737488346672
r14                0x0                0
r15                0x0                0
rip                0x400ec7            0x400ec7 <phase_2+30>
eflags             0x202             [ IF ]
cs                 0x33             51
ss                 0x2b             43
ds                 0x0                0
es                 0x0                0
fs                 0x0                0
gs                 0x0                0
(gdb) x/d0x7fffffffdd10
0x7fffffffdd10: 1
(gdb)
```

As we can see that the value we got after converting is “1”. So we have to compare (0 and 1). Since 0 and 1 are not equal so we move to next function <+34> which is jne function (jump if not equal to) after going to the next function it will jump to <+43> and the bomb will be exploded. So from here we know that the first input should be equal to 0, otherwise the function will move towards line <+34> and accordingly to <+43> and the bomb will be exploded. But if the first input is 0 the function will move to line <+36>.

As you can see below that I have put 0 in the first place and I don’t know the other values. So the first input is 0.

```

kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x0000000000400f0e <+101>: add    $0x28,%rsp
0x0000000000400f12 <+105>: pop    %rbx
0x0000000000400f13 <+106>: pop    %rbp
0x0000000000400f14 <+107>: retq
End of assembler dump.
(gdb) i r
rax            0x6                6
rbx            0x7fffffffde38      140737488346680
rcx            0x0                0
rdx            0x7fffffffdd24      140737488346404
rsi            0x0                0
rdi            0x7fffffffdd6a0     140737488344736
rbp            0x4021e0            0x4021e0 <_libc_csu_init>
rsp            0x7fffffffdd10      0x7fffffffdd10
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            0x400ec7            0x400ec7 <phase_2+30>
eflags         0x202              [ IF ]
cs             0x33              51
ss             0x2b              43
ds             0x0                0
es             0x0                0
fs             0x0                0
gs             0x0                0
(gdb) x/d0x7fffffffdd10
0x7fffffffdd10: 1
(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 2 3 4 5 6
Breakpoint 2, 0x0000000000400e49 in phase_2 ()
(gdb)

```

Now again we will go inside the disassemble and look for next value. Next up we will go to second compare function which compares (1 and %rsp). For that we have to find the value of rsp.

```

kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x0000000000400e44 <+59>: add    (%rbx),%eax
0x0000000000400e48 <+61>: cmp    %eax,0x0(%rbx)
0x0000000000400e49 <+64>: je     0x400ef0 <phase_2+71>
0x0000000000400e4b <+66>: callq 0x40142a <explode_bomb>
0x0000000000400e4c <+71>: add    $0x4,%rbx
0x0000000000400e4d <+75>: cmp    %rbp,%rbx
0x0000000000400e4f <+78>: jne    0x400ee1 <phase_2+56>
0x0000000000400e50 <+80>: mov    0x18(%rsp),%rax
0x0000000000400e51 <+85>: xor    %fs:0x28,%rax
0x0000000000400e52 <+94>: je     0x400f0e <phase_2+101>
0x0000000000400e53 <+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) i r
rax            0x6                6
rbx            0x7fffffffde38      140737488346680
rcx            0x0                0
rdx            0x7fffffffdd24      140737488346404
rsi            0x0                0
rdi            0x7fffffffdd6a0     140737488344736
rbp            0x4021e0            0x4021e0 <_libc_csu_init>
rsp            0x7fffffffdd10      0x7fffffffdd10
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            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/2d0x7fffffffdd10
0x7fffffffdd10: 0 2
(gdb)

```


Now we have found out the value of (%rsp) which we got as “2” and we will compare the two integers which are (1 and 2). Since they are not equal the function moves to line <+43> because when we compared the two integers they were not equal. This will cause bomb to blast. So the second integer must be 1 in-order not to explode the bomb. This is how we get the second integer which is “1”.

Now let's find the third input. For that the procedure is same as before. We have to run the program and give the first two input 0 and 1 and after that run disassembler and go to next line where the function calls. Right now it is at line <+48>. There are few operations such as mov, add and etc. But we will directly go to the compare function by commanding “ni” which means next line.

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
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) ni
0x0000000000400ee6 in phase_2 ()
(gdb) disas
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
0x0000000000400ec2 <+25>: callq 0x40144c <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 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>
0x0000000000400eeb <+66>: callq 0x40142a <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) □
```

Now we have reached to line <+61> and there is a compare function which compares(%eax and %rbx). We have to find the decimal values stored in these registers. For that we will follow the same procedure by going into the information registers and getting the value of them.

```

kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
=> 0x000000000400ee4 <+59>: add    (%rbx),%eax
0x000000000400ee6 <+61>: cmp    %eax,0x8(%rbx)
0x000000000400ee9 <+64>: je     0x400ef0 <phase_2+71>
0x000000000400eeb <+66>: callq 0x40142a <explode_bomb>
0x000000000400ef0 <+71>: add    $0x4,%rbx
0x000000000400ef4 <+75>: cmp    %rbp,%rbx
0x000000000400ef7 <+78>: jne    0x400ee1 <phase_2+56>
0x000000000400ef9 <+80>: mov    0x18(%rsp),%rax
0x000000000400efe <+85>: xor    %fs:0x28,%rax
0x000000000400f07 <+94>: je     0x400f0e <phase_2+101>
0x000000000400f09 <+96>: callq 0x400b00 <__stack_chk_fail@plt>
0x000000000400f0e <+101>: add    $0x28,%rsp
0x000000000400f12 <+105>: pop    %rbx
0x000000000400f13 <+106>: pop    %rbp
0x000000000400f14 <+107>: retq

End of assembler dump.
(gdb) i r
rax                0x1                1
rbx                0x7fffffffdd10       140737488346384
rcx                0x0                0
rdx                0x7fffffffdd24       140737488346404
rsi                0x0                0
rdi                0x7fffffffdd6a0      140737488344736
rbp                0x7fffffffdd20       0x7fffffffdd20
rsp                0x7fffffffdd10       0x7fffffffdd10
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                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/d0x7fffffffdd10
0x7fffffffdd10: 0
(gdb)

```

Now we got the values of (%eax and %rbx) which was (1 and 0) so we can see that the value is not equal so if they are not equal the function jumps to line <+66> and the bomb will be exploded. In order not to explode the bomb we have to give the third input as 1 because if the third input in 1 the comparison will be equal and it will jump to line <+64> and calls je(jump if equal to) and the bomb will not be exploded. So that's how we got the third input as 1.

Now for the fourth digit of the input we have to run the program and give the first 3 input as 0 1 1 then we have to open the disassembler and the earlier step was in line <+64> so said je(jump if equals to) so it jumped in that function and went to line <+71> where it added two registers and compared in the next step.

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
0x0000000000400ee9 <+64>: je 0x400ef0 <phase_2+71>
0x0000000000400eeb <+66>: callq 0x40142a <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) i r
rax 0x1 1
rbx 0x7fffffffdd14 140737488346388
rcx 0x0 0
rdx 0x7fffffffdd24 140737488346404
rsi 0x0 0
rdi 0x7fffffffdd6a0 140737488344736
rbp 0x7fffffffdd20 0x7fffffffdd20
rsp 0x7fffffffdd10 0x7fffffffdd10
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 0x400ef4 0x400ef4 <phase_2+75>
eflags 0x206 [ PF IF ]
cs 0x33 51
ss 0x2b 43
ds 0x0 0
es 0x0 0
fs 0x0 0
gs 0x0 0
(gdb) x/d0x7fffffffdd20
0x7fffffffdd20: 6
(gdb) x/d0x7fffffffdd14
0x7fffffffdd14: 1
(gdb) ni
```

So we have compare function in line <+75> and we are comparing (%rbp and rbx) so when we opened the information register and converted the hex value to decimal we got as (6 and 1) so the two numbers are not equal so it will go to next function which says jne(jump if not equal to) so it will go in the jne function and inside jne function it takes us to line <+56>.

So in line <+56> we have some mov and add functions and after that in line <+61> we have the compare function which compares(eax and rbx). So after opening the information register we got the value as (2 and 1). So we know

that if the two number did not match than the bomb will be exploded. So in-order not to explode the bomb the correct input is 2. And that's how we get the fourth digit as "2". So our input set of 6 digits till now are [0 1 1 2 e f] we haven't found the last two digits yet.

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignment 1/bomb003
=> 0x00000000400ee4 <+59>: add    (%rbx),%eax
0x00000000400ee5 <+61>: cmp    %eax,0x8(%rbx)
0x00000000400ee6 <+64>: je     0x400ef0 <phase_2+71>
0x00000000400eeb <+66>: callq  0x40142a <explode_bomb>
0x00000000400ef0 <+71>: add    $0x4,%rbx
0x00000000400ef4 <+75>: cmp    %rbp,%rbx
0x00000000400ef7 <+78>: jne    0x400ee1 <phase_2+56>
0x00000000400ef9 <+80>: mov    0x18(%rsp),%rax
0x00000000400efe <+85>: xor    %fs:0x28,%rax
0x00000000400f07 <+94>: je     0x400f0e <phase_2+101>
0x00000000400f09 <+96>: callq  0x400b00 <__stack_chk_fail@plt>
0x00000000400f0e <+101>: add    $0x28,%rsp
0x00000000400f12 <+105>: pop    %rbx
0x00000000400f13 <+106>: pop    %rbp
0x00000000400f14 <+107>: retq

End of assembler dump.
(gdb) i r
rax                0x2                2
rbx                0x7fffffffdd14       140737488346388
rcx                0x0                0
rdx                0x7fffffffdd24       140737488346404
rsi                0x0                0
rdi                0x7fffffffdd6a0      140737488344736
rbp                0x7fffffffdd20       0x7fffffffdd20
rsp                0x7fffffffdd10       0x7fffffffdd10
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                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/d0x7fffffffdd14
0x7fffffffdd14: 1
(gdb)
```

So now to find the fifth digit we have to follow the same procedure. We have reached to line<+75> now it will compare the two registers. The values were found to be (3 and 1) and since they are not equal it will jump to next line which says jne (jump if not equals to).

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignmen...
rax      0x7fffffff0018      140737488340392
rcx      0x0                0
rdx      0x7fffffffdd24      140737488346404
rsi      0x0                0
rdi      0x7fffffffdd6a0      140737488344736
rbp      0x7fffffffdd20      0x7fffffffdd20
rsp      0x7fffffffdd10      0x7fffffffdd10
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      0x400ef4            0x400ef4 <phase_2+75>
eflags   0x206              [ PF IF ]
cs       0x33               51
ss       0x2b               43
ds       0x0                0
es       0x0                0
fs       0x0                0
gs       0x0                0
(gdb) x/d0x7fffffffdd18
0x7fffffffdd18: 1
(gdb) x/d0x7fffffffdd20
0x7fffffffdd20: 3
(gdb) ni
0x0000000000400ef7 in phase_2 ()
(gdb) ni
0x0000000000400ee1 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
0x0000000000400ee1: push    %rbp
0x0000000000400ee2: movl    $0, %edi
0x0000000000400ee3: movl    $0, %eax
0x0000000000400ee4: movl    $0, %ebx
0x0000000000400ee5: movl    $0, %rcx
0x0000000000400ee6: movl    $0, %rdi
0x0000000000400ee7: movl    $0, %rbp
0x0000000000400ee8: movl    $0, %rsp
0x0000000000400ee9: retq
```

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignmen...
0x0000000000400f0e <+101>: add    $0x28,%rsp
0x0000000000400f12 <+105>: pop    %rbx
0x0000000000400f13 <+106>: pop    %rbp
0x0000000000400f14 <+107>: retq
End of assembler dump.
(gdb) i r
rax      0x3                3
rbx      0x7fffffffdd18      140737488346392
rcx      0x0                0
rdx      0x7fffffffdd24      140737488346404
rsi      0x0                0
rdi      0x7fffffffdd6a0      140737488344736
rbp      0x7fffffffdd20      0x7fffffffdd20
rsp      0x7fffffffdd10      0x7fffffffdd10
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      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/d0x7fffffffdd18
0x7fffffffdd18: 1
(gdb)
```

After reaching line <+78> the function will move to line <+56> where it will perform mov and add functions. After that it will compare (%eax and %rbx). The values found were (3 and 1). Since if the fifth input is not 3 than the bomb will explode. So in-order not to explode the bomb the correct fifth input is 3. So that's how we found the 5th input. Our inputs are [0 1 1 2 3 f]

Now to find the last digit we do the same thing. Finally go inside the compare function at line <+61> where we compare (eax and rbx). The value we got are (5 and 2) since they are not equal the bomb will be exploded and inorder not to explode we must input the last digit as 5. Our input finally is [0 1 1 2 3 5]

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignmen...
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

--Type <RET> for more, q to quit, c to continue without paging--
End of assembler dump.
(gdb) i r
rax                0x5                    5
rbx                0x7fffffffdd1c          140737488346396
rcx                0x0                    0
rdx                0x7fffffffdd24          140737488346404
rsi                0x0                    0
rdi                0x7fffffffdd6a0         140737488344736
rbp                0x7fffffffdd20          0x7fffffffdd20
rsp                0x7fffffffdd10          0x7fffffffdd10
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                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/d0x7fffffffdd1c
0x7fffffffdd1c: 2
(gdb)
```

Finally we found all the 6 digits inputs which are [0 1 1 2 3 5] and these are the first 6 Fibonacci numbers. So I run the ./bomb file and gave the inputs and the bomb was diffused.

```
kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignme...
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/d0x7fffffffdd1c
0x7fffffffdd1c: 2
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) q
A debugging session is active.

    Inferior 1 [process 531] 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!
```

Phase 2 Over!!!

**Haha Dr. Evil, Wait and see I am getting closer to you, I am almost
Halfway.....Be prepared Dr.....**