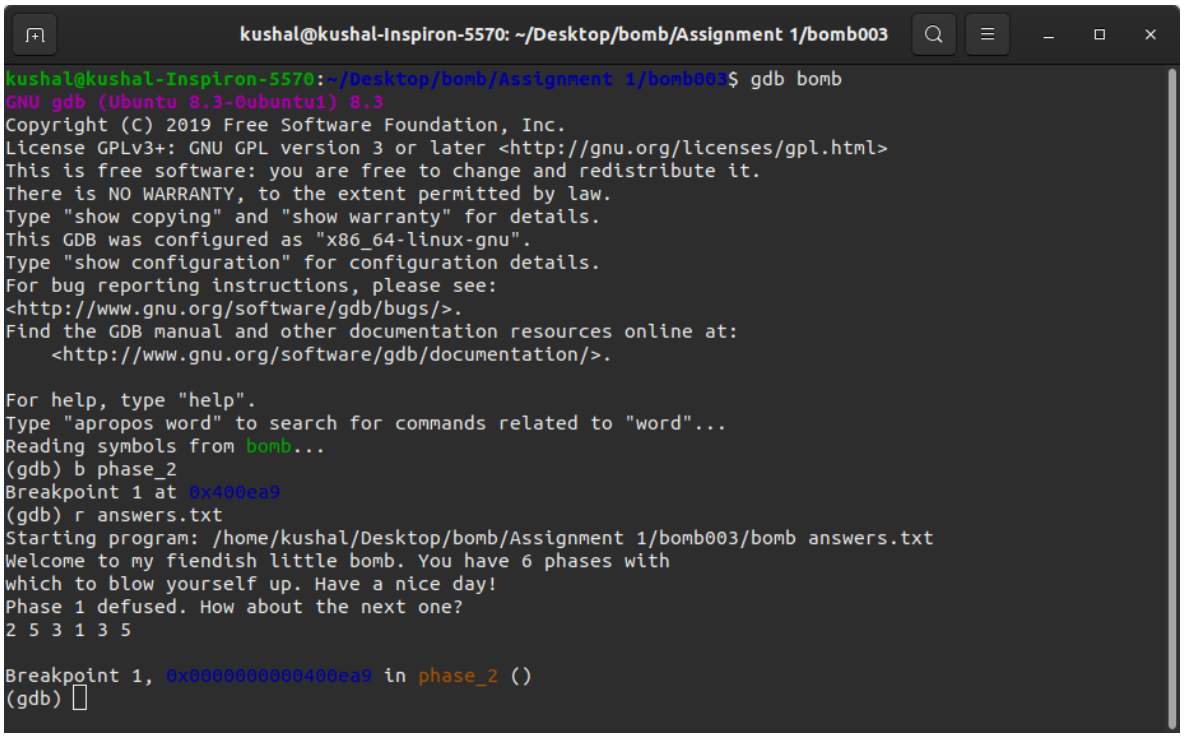
Binary Bomb(Group 3)
Phase 02
“Cracking The Code”

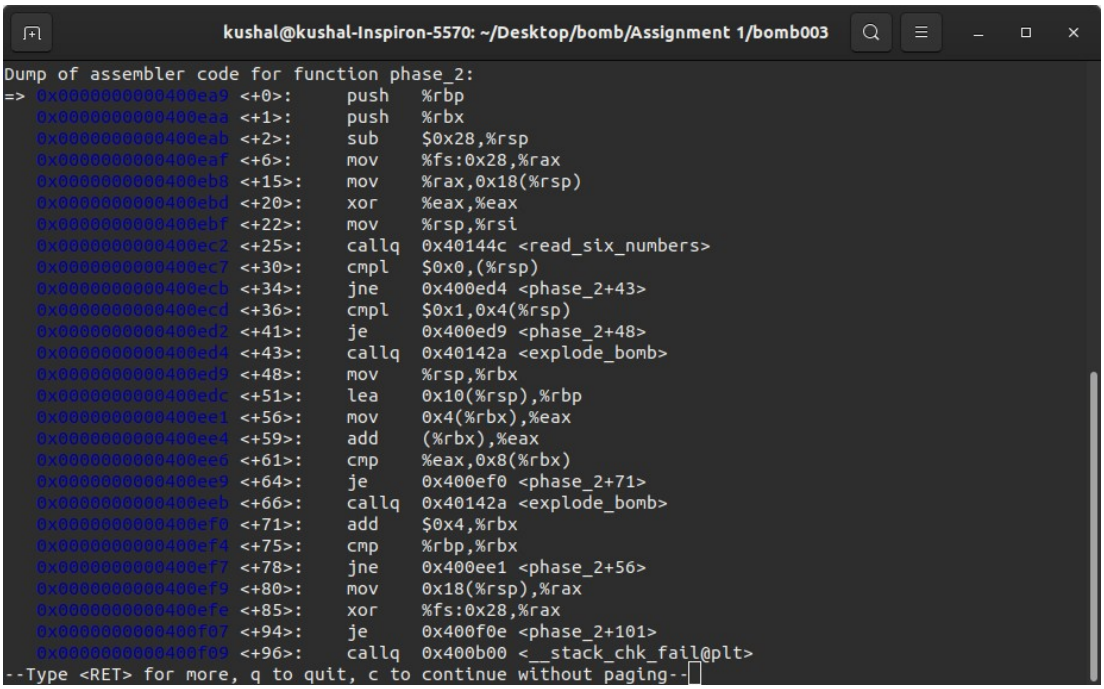
The first phase was finding the correct string and matching it with the Dr.Evil’s string. If the correct string was given than the bomb at phase 1 will be diffused and if not it will blast. Well the second phase is about giving a set of six inputs. The inputs were found to be “**0 1 1 2 3 5**” which 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.

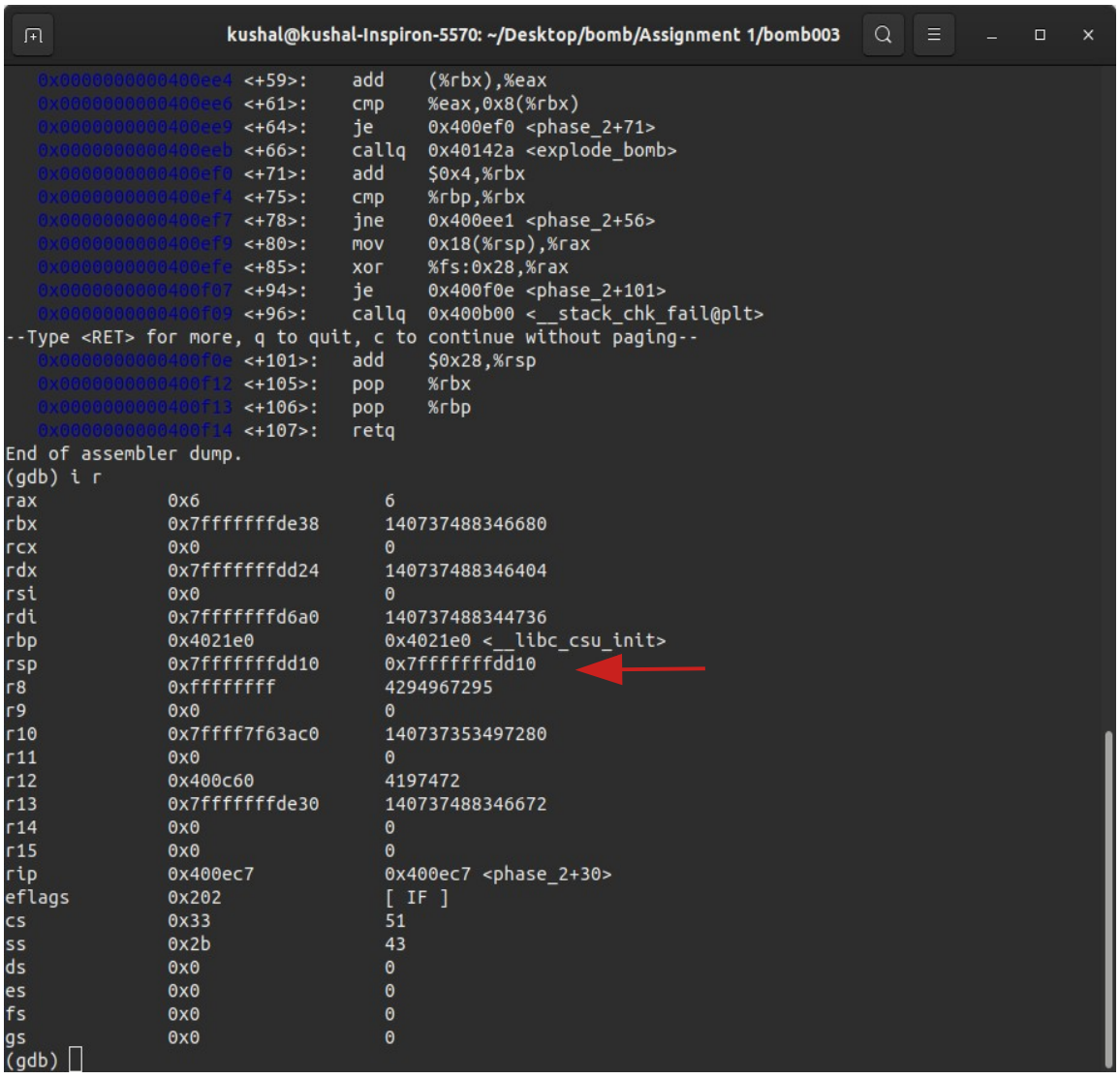
First of all it will be easier and less time consuming if we save the acquired string of phase_1 inside a txt file so that we don’t have to write the string again and again. I saved it as a answers.txt file.
After diffusing the first phase we have to again go to the gdb debugger and this time we have 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.



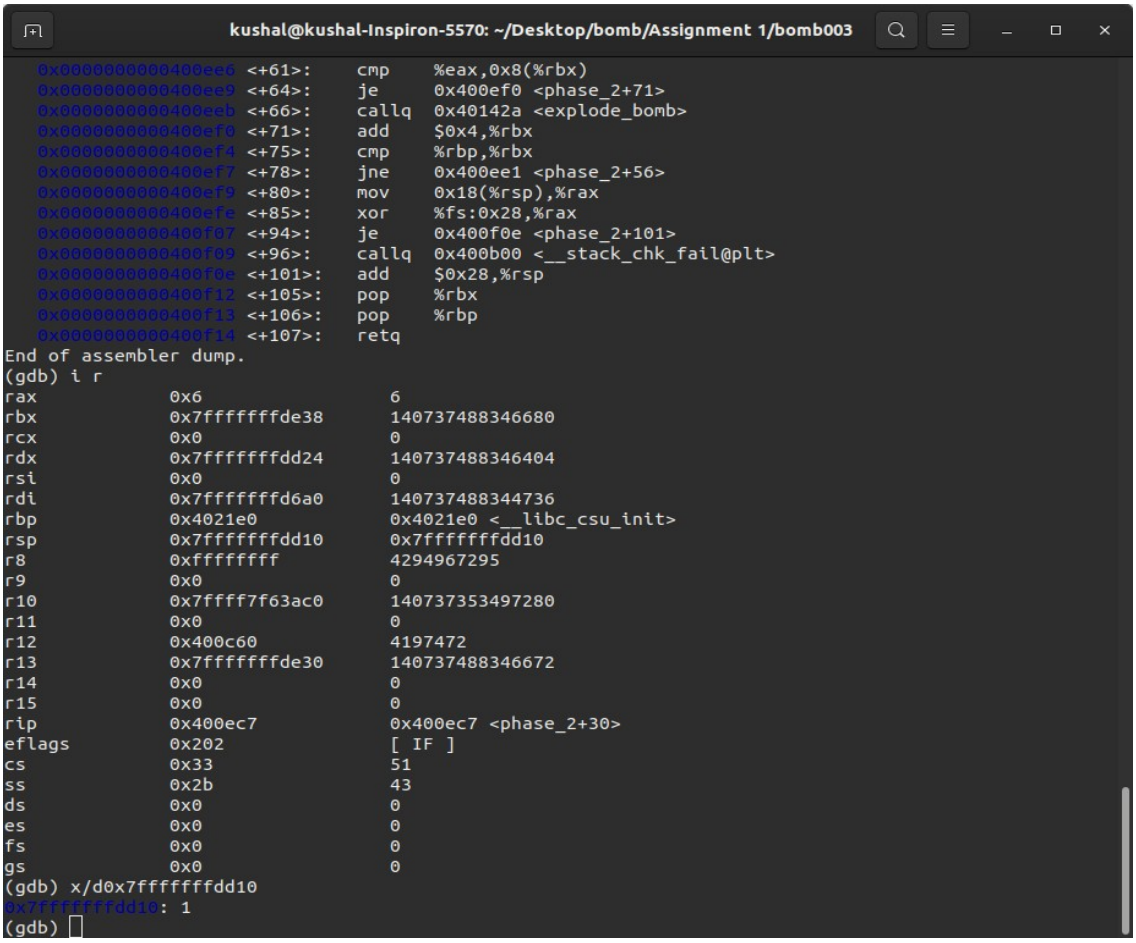
After putting some random inputs, we have to go inside disassemble for finding the real inputs



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 compre 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.

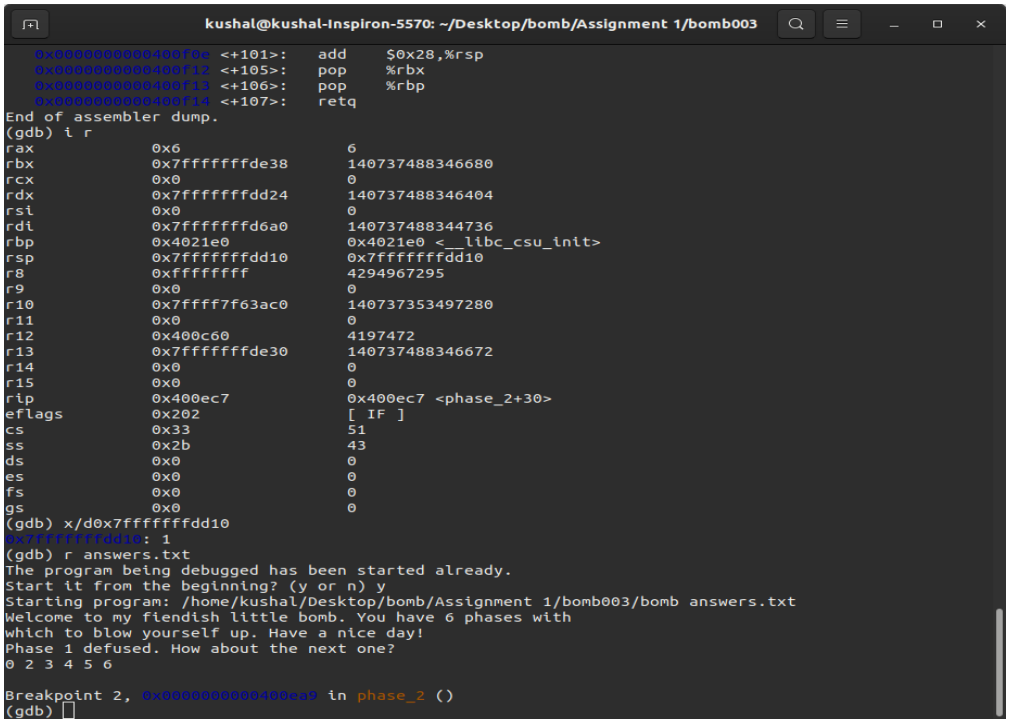


We found the value of rsp register but the value is in hexa form so we have to convert it into decimal form. In-order to convert the number into decimal we have to command “(**x/d(number)**)”.

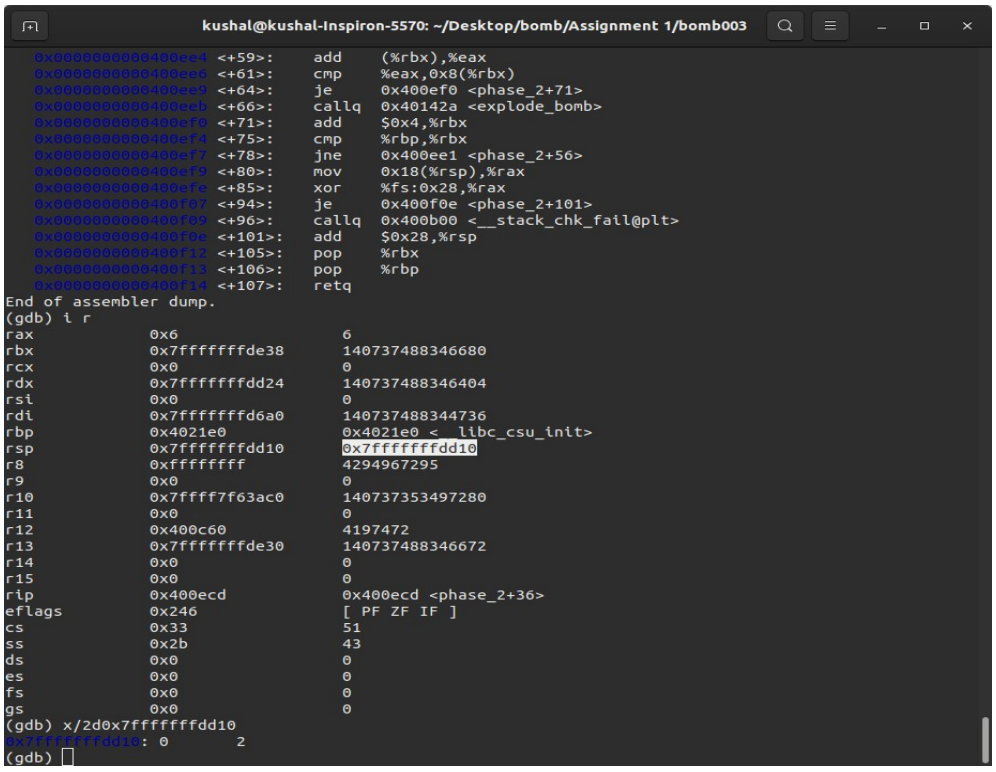


As we can see that the value we got after converting is “1”. So we have to compare (0 and 1). Since 0 and 1 are not equal so we move to next function <+34> which is jne function (jump if not equal to) after going to the next function it will jump to <+43> and the bomb will be exploded. So from here we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

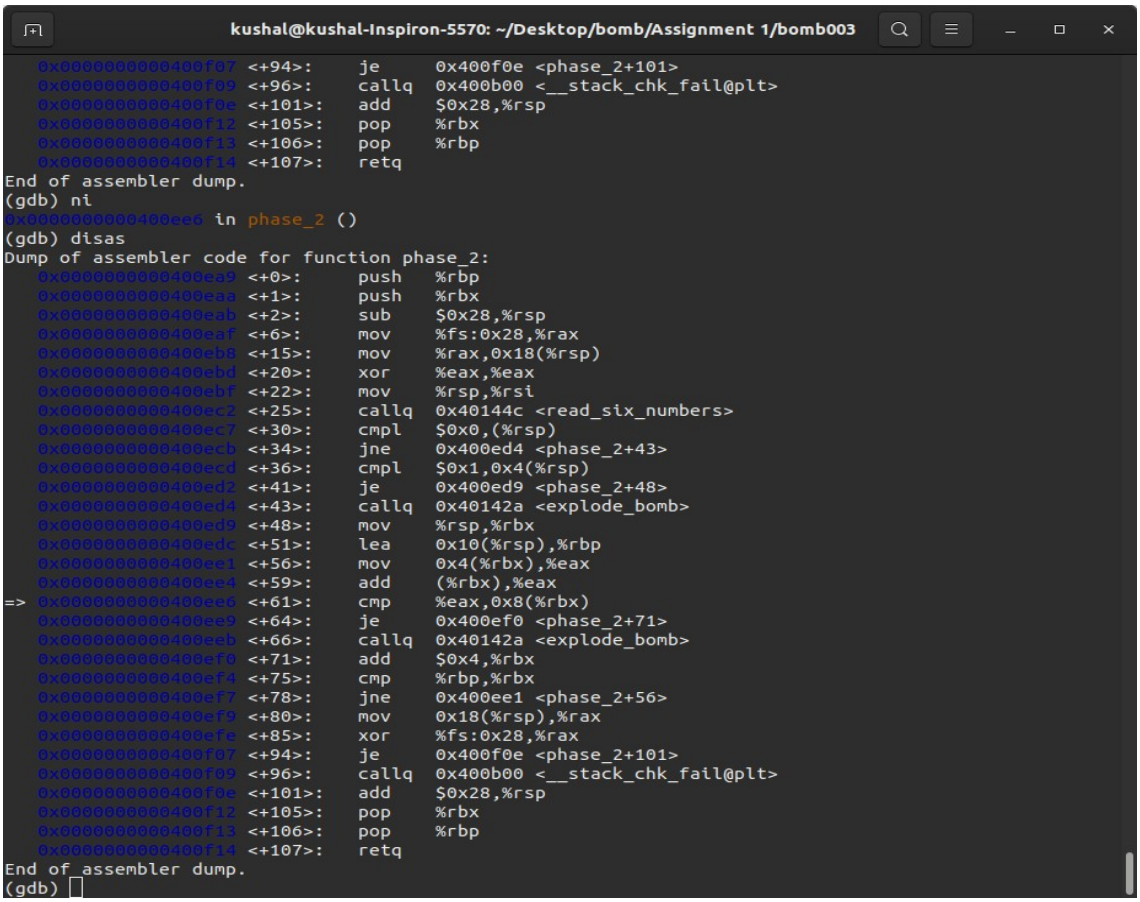


Now again we will go inside the disassemble and look for next value. Next up we will go to second compare function which compares (1 and %rsp). For that we have to find the value of rsp.

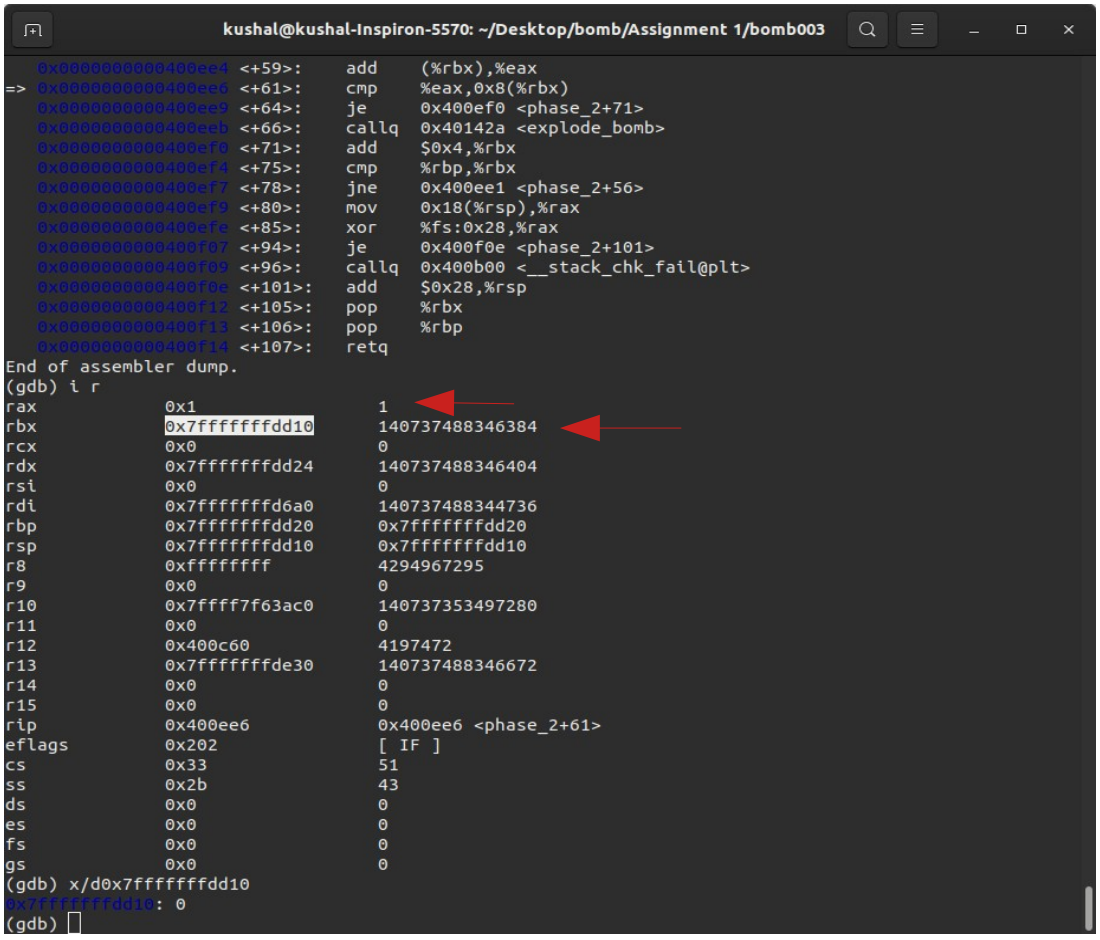


Now we have found out the value of (%rsp) which we got as “2” and we will compare the two integers which are (1 and 2). Since they are not equal the function moves to line <+43> because when we compared the two integers they were not equal. This will cause bomb to blast. So the second integer must be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

Now lets 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 dissembler 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.

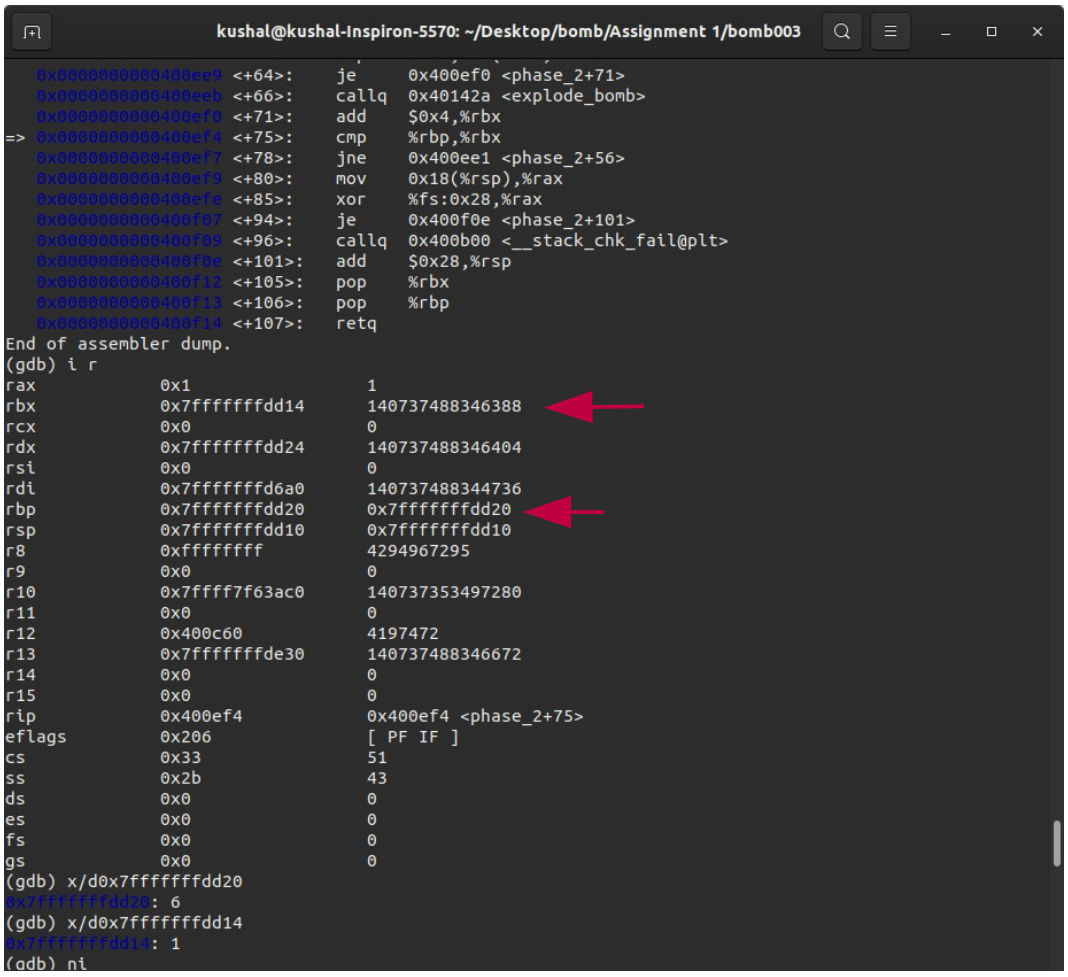


Now we have reached to line <+61> and there is a compare function which compares(%eax and %rbx). We have to find the decimal values stored in these registers. For that we will follow the same procedure by going into the information registers and getting the value of them.



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.

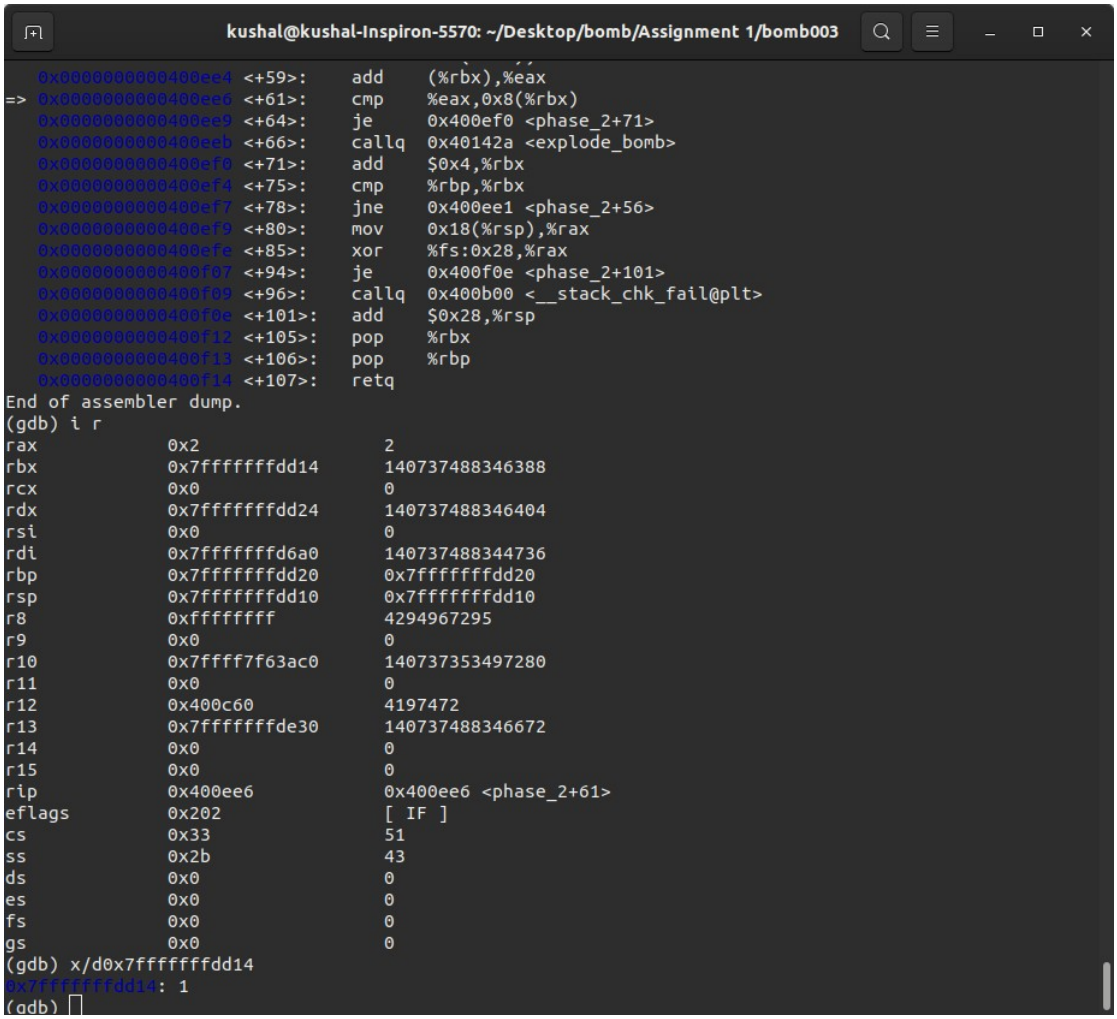
Now for the fourth digit of the input we have to run the program and give the first 3 input as 0 1 1 than we have to open the dissembler and the earlier step was in line <+64> so said je(jump if equals to) so it jumped in that function and went to line <+71> where it added two registers and compared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



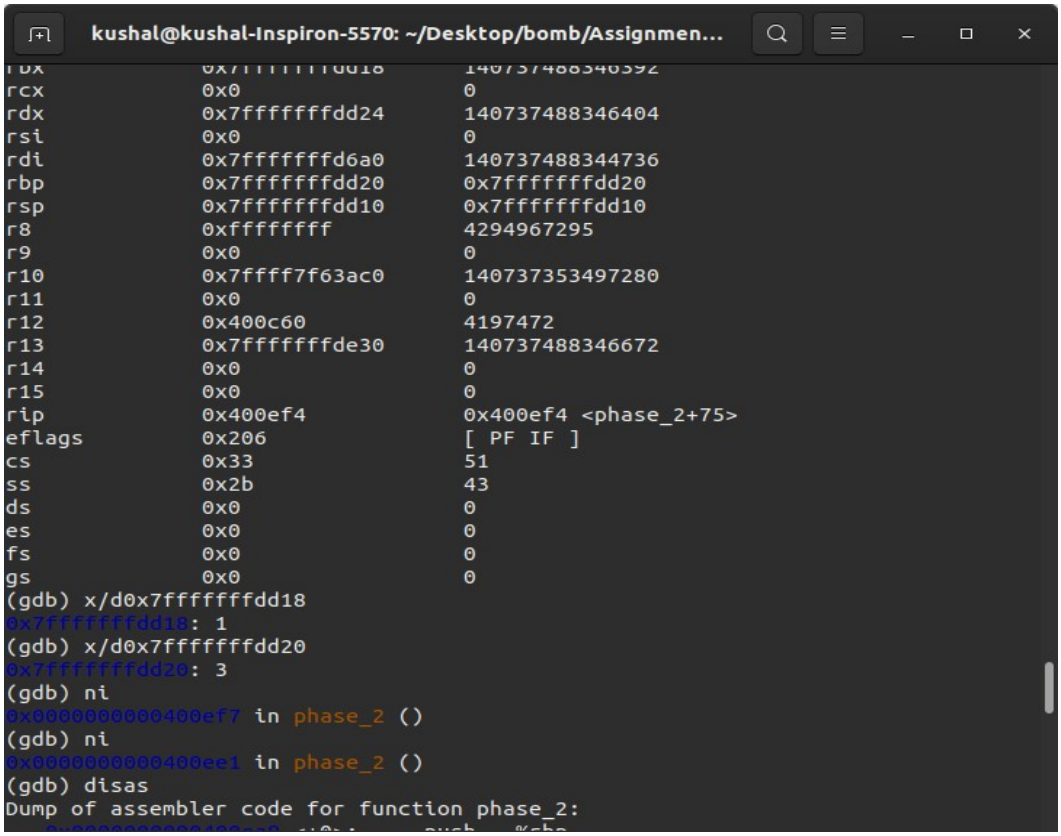
So we have compare function in line <+75> and we are comparing (%rbp and rbx) so when we opened the information register and converted the hex value to decimal we got as (6 and 1) so the two numbers are not equal so it will go to next function which says jne(jump if not equal to) so it will go in the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

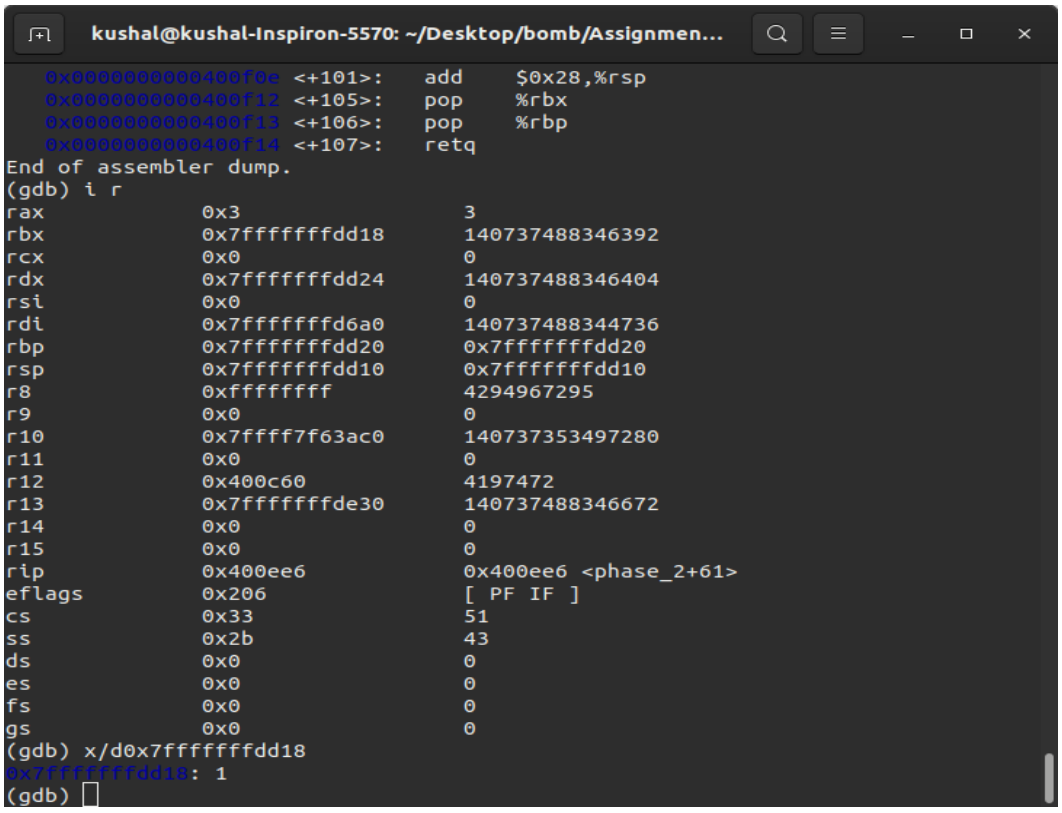
So in line <+56> we have some mov and add functions and after that in line <+61> we have the compare function which compares(eax and rbx). So after opening the information register we got the value as (2 and 1). So we know

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.



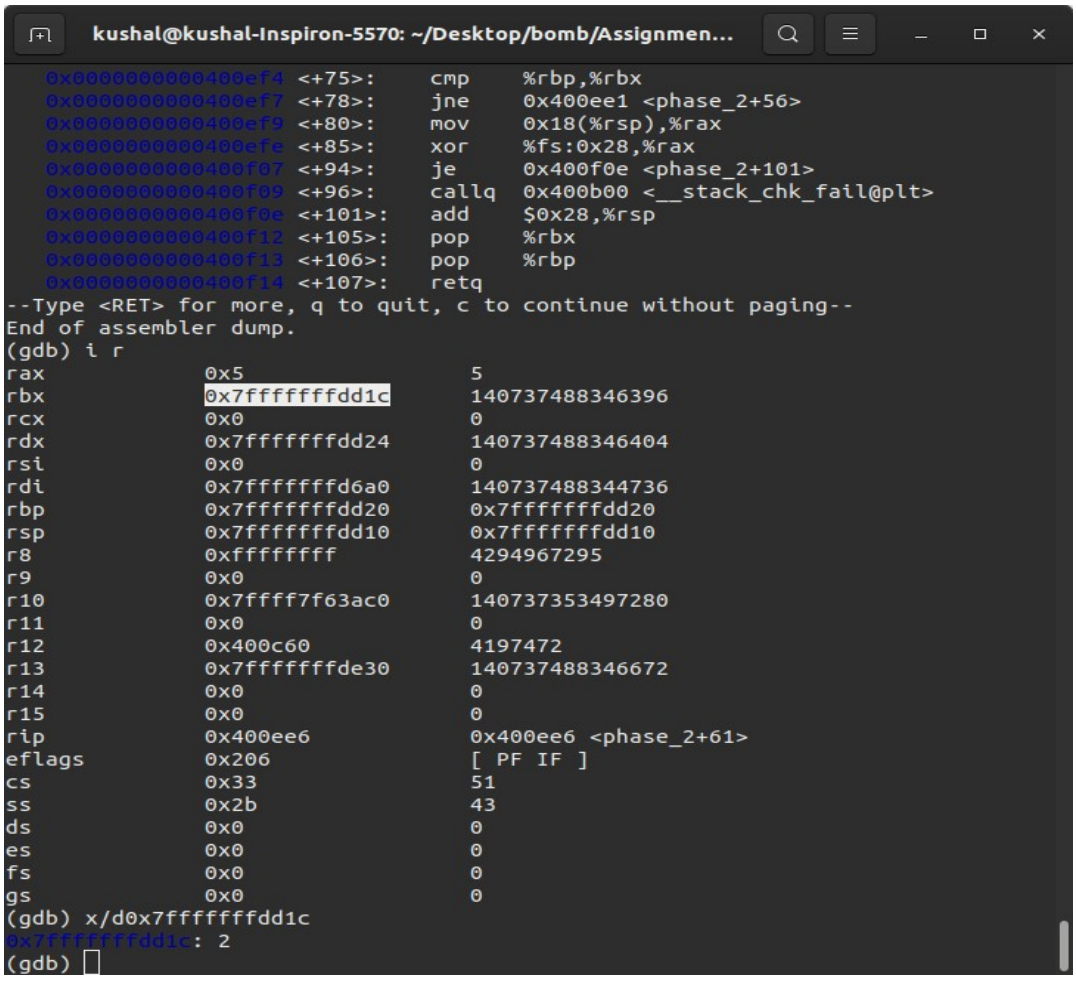
So now to find the fifth digit we have to follow the same procedure. We have reached to line<+75> now it will compare the two registers. The values were found to be (3 and 1) and since they are not equal it will jump to next line which says jne (jump if not equals to).


kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignmen...
rcx 0x0 0
rdx 0x7fffffffdd24 140737488346404
rsi 0x0 0
rdi 0x7fffffffd6a0 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:

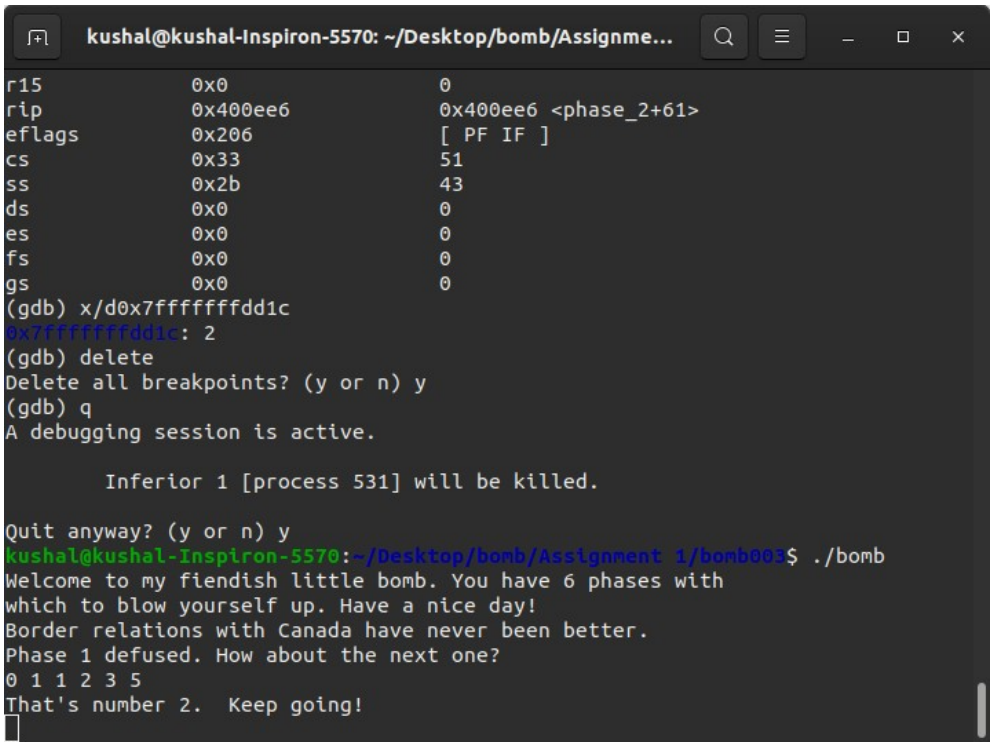

kushal@kushal-Inspiron-5570: ~/Desktop/bomb/Assignmen...
0x0000000000400f0e <+101>: add $0x28,%rsp
0x0000000000400f12 <+105>: pop %rbx
0x0000000000400f13 <+106>: pop %rbp
0x0000000000400f14 <+107>: retq
End of assembler dump.
(gdb) i r
rax 0x3 3
rbx 0x7fffffffdd18 140737488346392
rcx 0x0 0
rdx 0x7fffffffdd24 140737488346404
rsi 0x0 0
rdi 0x7fffffffd6a0 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]



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.



Phase 2 Over!!!

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