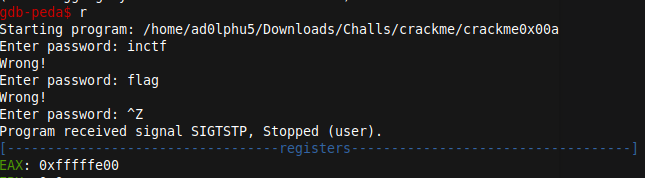
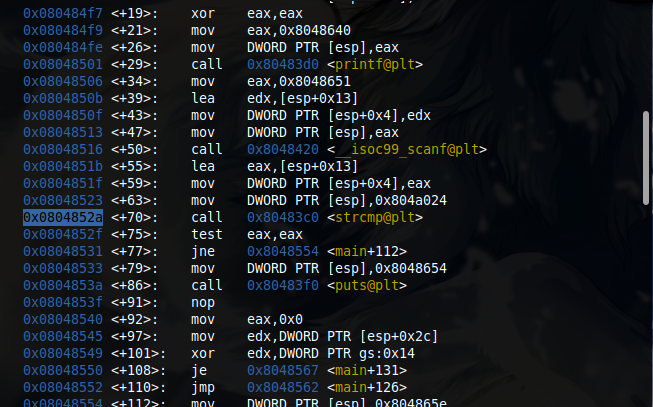
**Crackme0x00a**

I tried to run the ELF 32-bit LSB executable in gdb and it asked for a password



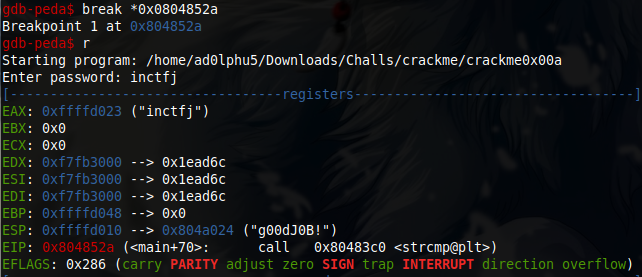
I tried checking the registers directly… but could not get the flag

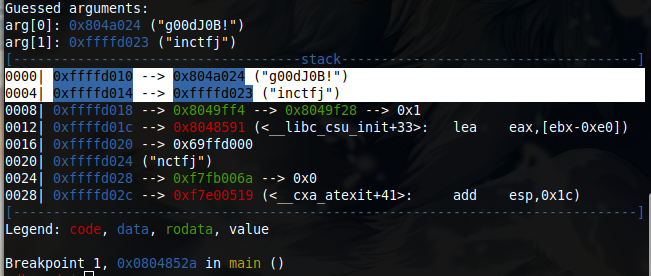
So i checked the disassembled code of main



0x0804852a <+70>: call 0x80483c0 <strcmp@plt>

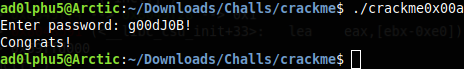
So here its comparing the string password so i gave a breakpoint at 0x0804852a .





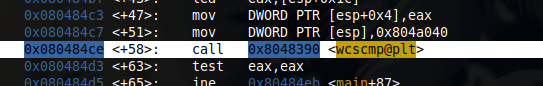
There we got the password while comparing with the guessed password i gave(inctfj)

Password ⇒ g00dJ0B!

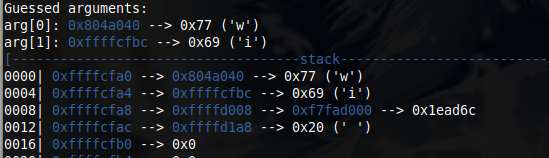


**Crackme0x00b**

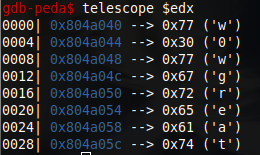
First i tried to analyse the assembly code by running in gdb .



Gave a break point at 0x080484ce in the main function and tried to run it.



There i could see the first letter of my guessed input .i.e. ‘I’ of inctfj is compared with the first letter of original password .i.e. W. gave a nexti and then telescope is used to get the stack values of edx register



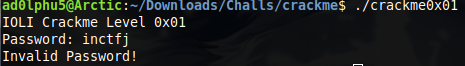
There i got the password: **w0wgreat**



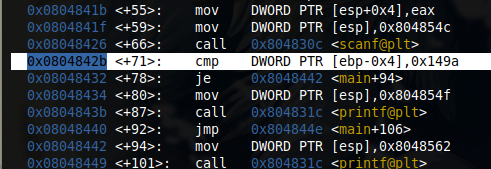
Password ⇒ **w0wgreat**

**Crackme0x01**

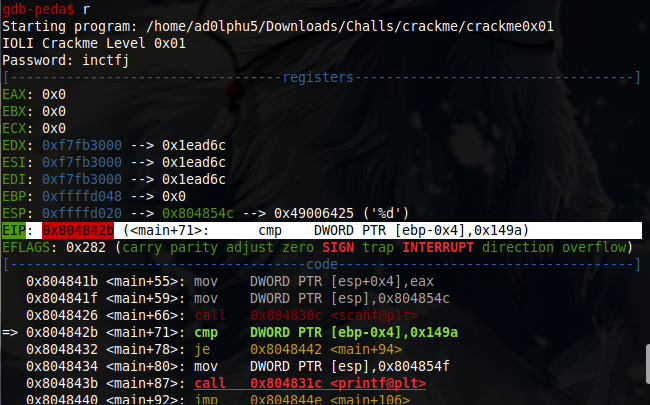
When i tied to execute the file, i asked for a password.



So i tried to run it using gdb. While disassembling the code , i saw a final cmp statement



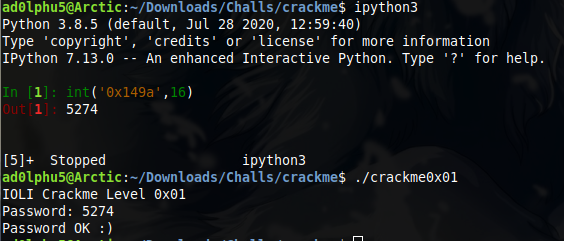
So put a break point and tried to run it.



EIP: 0x804842b (<main+71>: cmp DWORD PTR [ebp-0x4],0x149a)

PTR [ebp-0x4] is the variable which take the password and compare with the 0x149a.

So lets convert 0x149a to integer and check .



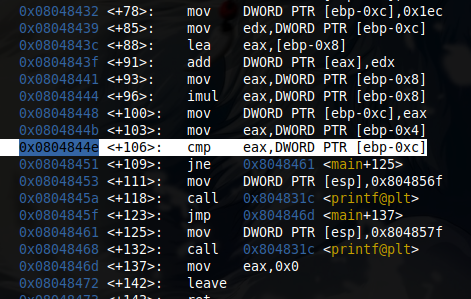
Yo… it works!!

Password ⇒ 5274

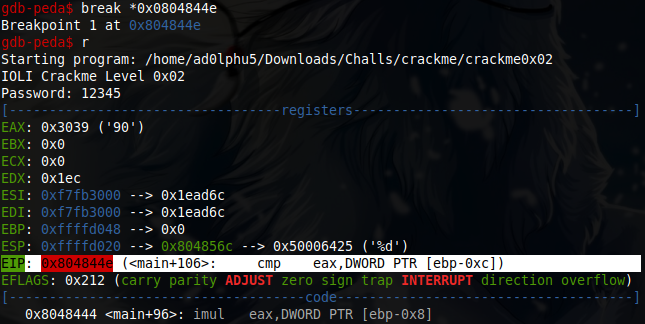
**Crackme0x02**

When i tried ro run the file, it asked for password.

So i tried to run it using gdb. Disassembled the code.



So lets give a break point there and check again

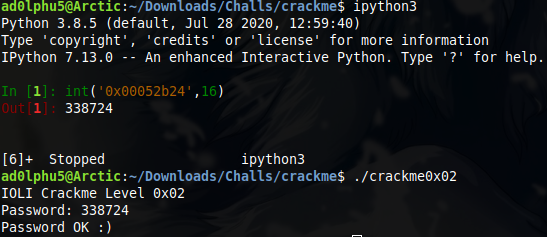


So from this we can understand that eax is compared with word which is stored in the address ebp-0xc

So, i tried to print the contents in that address epb-0xc



Then i tried to convert the hex value to int using python .

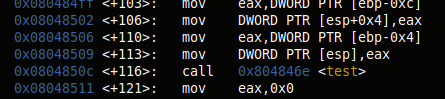


And yes… got the password

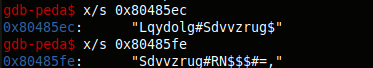
Password ⇒ 338724

**Crackme0x03**

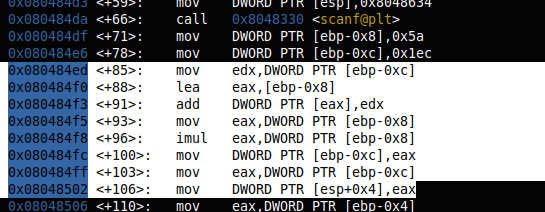
Lets run the code directly in gdb and from there we can see function test is called



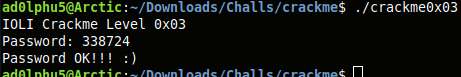
Disas test shows the shit is called with arg 0x80485fe after comparing otherwise the arg in the address 0x80485ec



After analysing the code after disas main we can say that if the pass is correct after checking with both the addresses .test > shift func check the authenticity of the password.



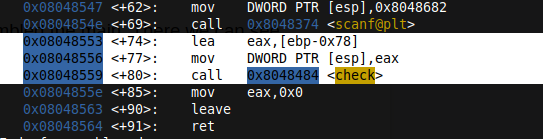
After analysing we can determine that the ans is 0x52b21⇒ 338724



password⇒ 338724

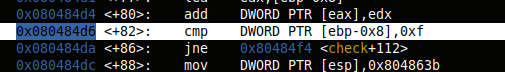
**Crackme0x04**

Used gdb to analyse the code. Disassembled the main .There we can see that input is from the user is stored in ebp-0x78

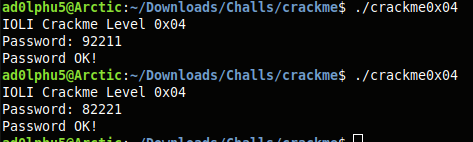


Now lets check ‘check’ function using disas check

After analysing the code for a while we can understand that the code converts the string input to int and also the sub of digits should be 15\



So lets try to give a password where sum = 15

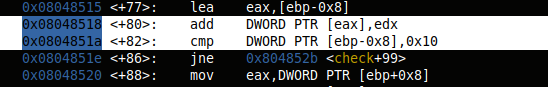


**Crackme0x05**

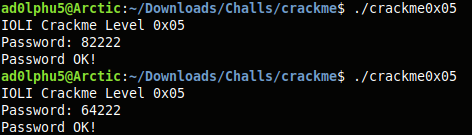
This chall is similar to previous challenge…

Followed the same procedure and in the check function we can see that it is checking each char whether its even or odd… if its even , then proceed.

It also checks the sum of digits as 16/not



So lets try with a password

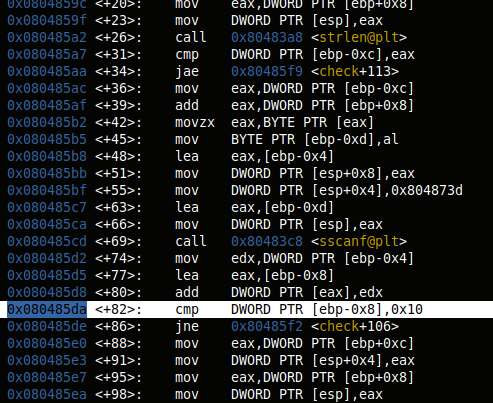


It works….

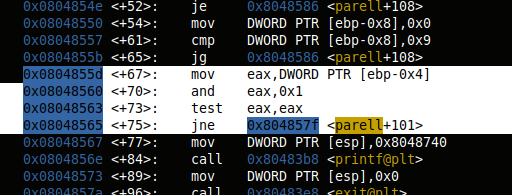
password⇒ 664222

**Crackme0x6**

When i tried to open the code using gdb. There whsn’t much difference in main function. And for disas check we can determine that the sum of the digits should be 16 and also the last digit should be even.

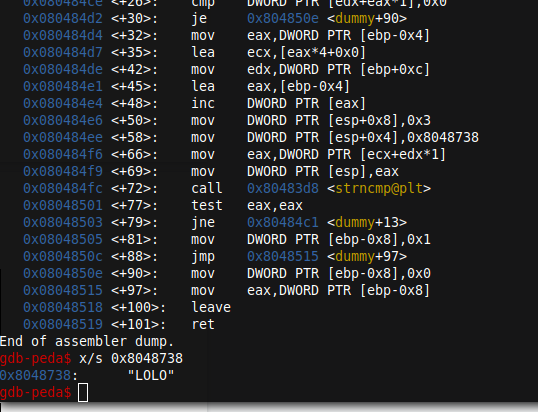


The i looked into parell function.



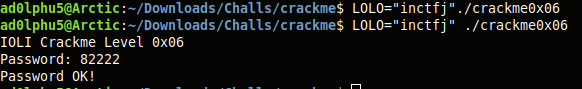
It says, input should be string of even number but it also calls another function dummy.

So looked into dummy function as well.



So here it checks for a variable “LOLO” with a string literal .

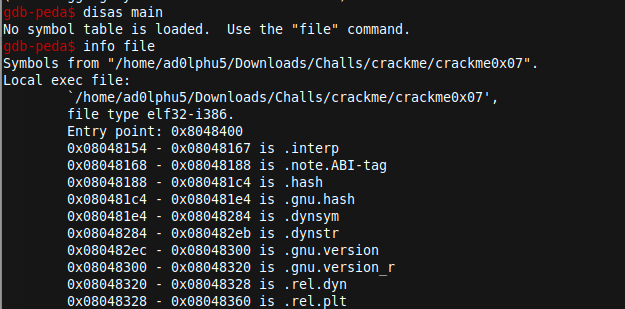
So combining all the info we got. Lets try to crack the password.



**Crackme0x07**

When i tried to run the code gdb using disas main.it failed.

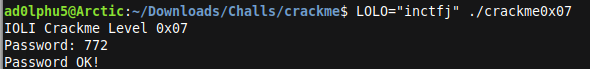
So i used file command



Disas the code using:

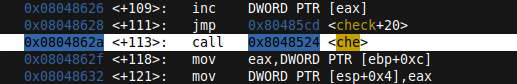
disas 0x08048400,0x08048784

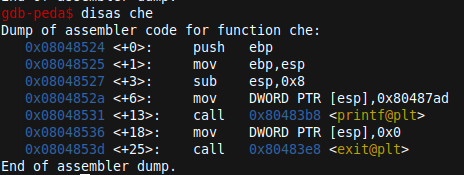
After analysing i came to know that it is similar to 0x06 and will work with same password



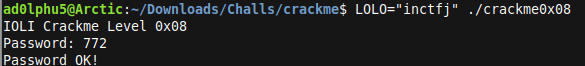
**Crackme0x08**

same as 0x06 level. Difference is that it has another funvtion che which prints password invalid and exit the prgm wichount moving on.



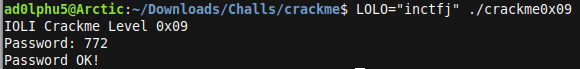


So it has to accept an env variable LOLO with a string.



**Crackme0x09**

This is similar to crackme0x07 and 0x08. I’ve to use file command and then disas the code. When i try to analyse it . It was similar to 0x08 and will work with the same password

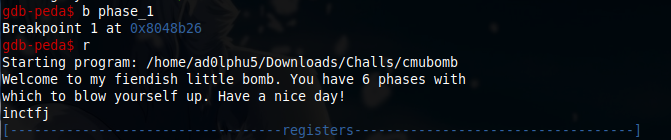


**CMUBOMB**

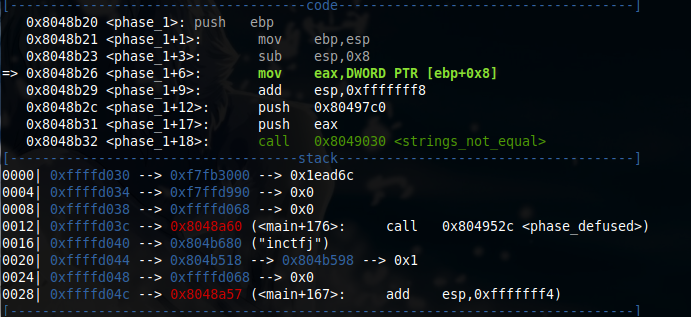
\*phase\_1

I tried analyse the assembly of cmubomb using gdb and understood that there are 6 phases to diffuse the bomb completely .

First i tried to diffuse phase\_1 . so gave a breakpoint in phase\_1 and tried to run



Gave a guessed wrong password and gone through the code and registrs.



Then I tried to look into memory address of eax and 0x80497c0.



Eax with memory address 0x804b680 has our input password .i.e. Inctfj.

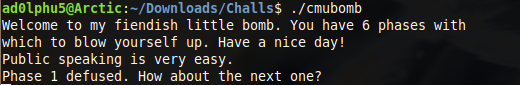
So lets look into fixed memory address 0x80497c0.



As it had hex values i used x/s to get in ascii format.

Yes…. we got a string “Public speaking is very easy.”

Then i tried with this string to diffuse the first phase



And yes,diffused the phase\_1

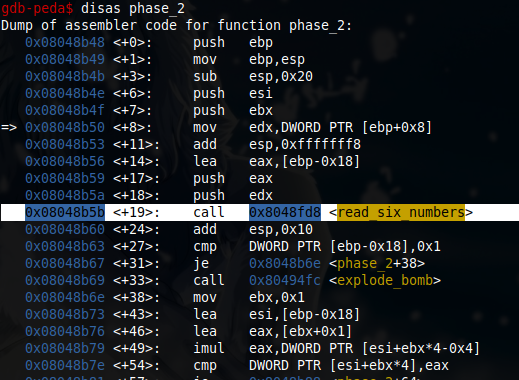
Pass ⇒ **Public speaking is very easy.**

\*phase\_2

Lets start with giving a breakpoint at phase\_2 and run it again

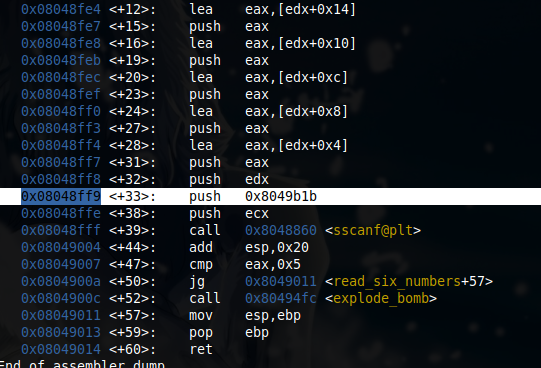


Guessed a password and boom. Lets disas phase\_2



0x08048b5b <+19>: call 0x8048fd8 <read\_six\_numbers>

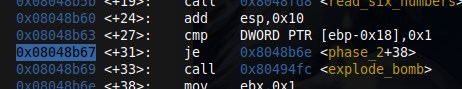
Lets break at read\_six\_numbers and check



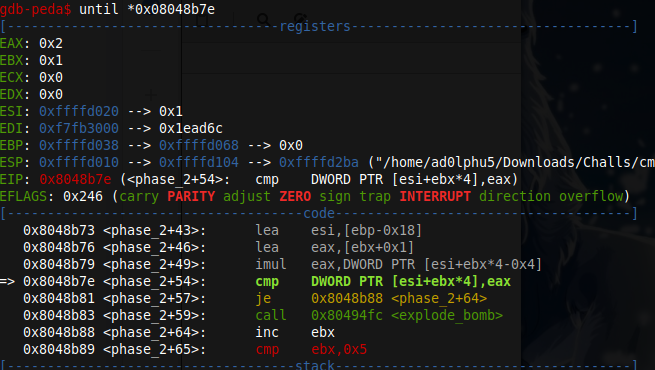
Then, i tried to check whats inside the fixed memory register 0x8049b1b



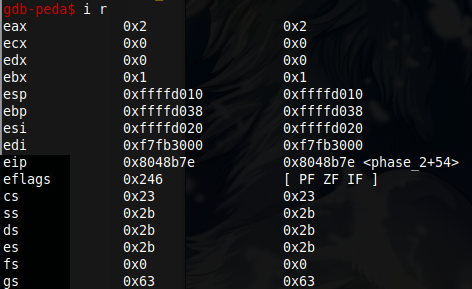
Ig the pass for phase\_2 will be 6 numbers in int format.



Lets break here and check. After that until \*0x08048b7e

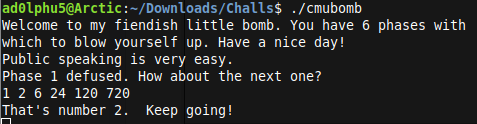


Got all the values in registers .



First number is fixed as 1.Similarly , i got the first three values… 1 2 6

Rest we can find out using the same method… i guessed the pattern as factorial . 1 2 6 24 120 720



Pass ⇒ **1 2 6 24 120 720**

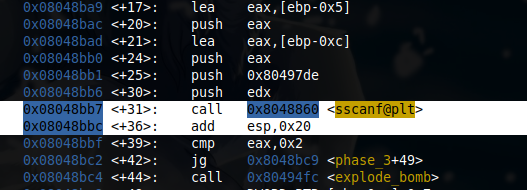
\*Phase\_3

Lets start by analysing assembly code of phase\_3. There we could see lot of compare statements . so ig i’ve to analyse each statement .

Here is also a scanf function call

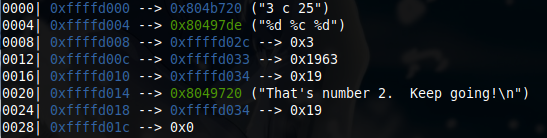
0x08048bb7 <+31>: call 0x8048860 <sscanf@plt>

Lets start with gdb



Lets see what scanf function gonna return

Put a breakpoint at 0x08048bbc and continue .



From here its clear that the input format is ‘%d %c %d’

So i again run the code with a test input as “1 a 2”

And then disas phase\_3

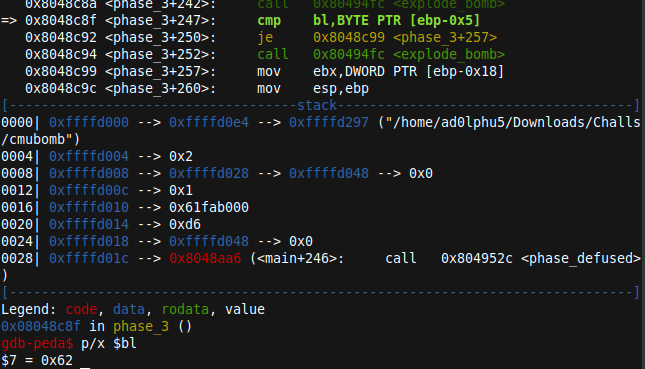


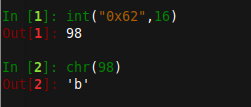
ipython3 --> int(“0xd6”,16)=214

As we assume the next digit to be 214 and again tried to run the code with input as 1 a 214. And the bomb blown up

So i used nexti till 0x8048c8f <phase\_3+247>: cmp bl,BYTE PTR [ebp-0x5]

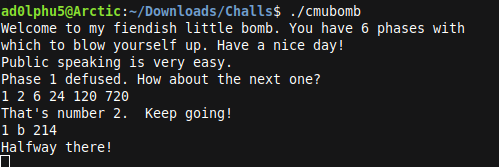
And checked the register





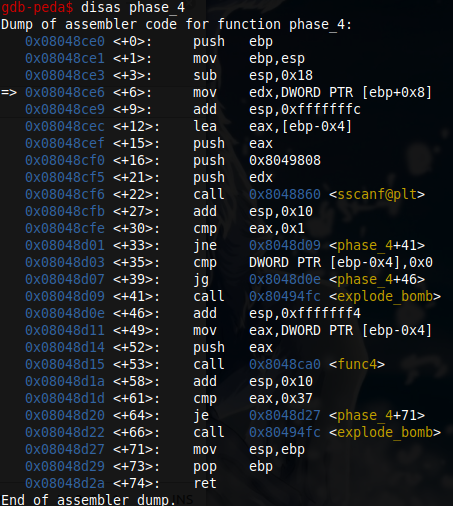
0x62==>b

Therefore out pass for phase\_3 is 1 b 214



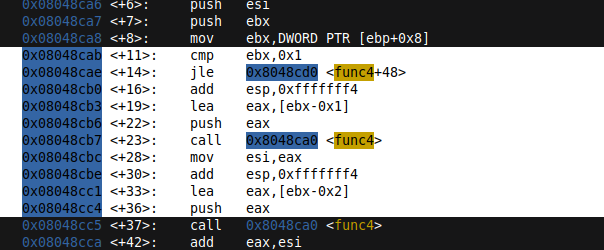
**\*phase\_4**

As before lets give a breakpoint at phase\_4 and continue analysing the code.



Here , the input is stored in eax and calling func4

So lets break and check whats inside func4



We can see that till eax =1 , function is recursively called .



Here the value is compared to 55.55 is the 10th fib number. So the possible answer would be 10-1=9



Pass ⇒ 9

Python code for verification.

def recur\_fibo(n):

if n <= 1:

return n

else:

return(recur\_fibo(n-1) + recur\_fibo(n-2))

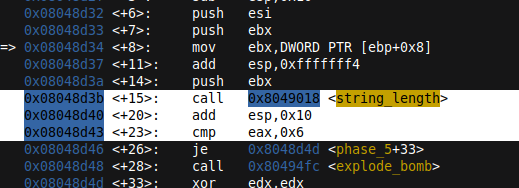
print(recur\_fibo(10))

=>55

**Phase\_5**

As usual i tried to run the code with a sample test string “inctfj” after gibing a breakpoint at phase\_5. As expected bomb exploded.

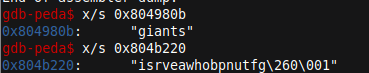
When i tried to analys the asm code.



I got the string length to be 6.

moving through a loop, itration over each char and result of str is compared to str stored in 0x804980b

There were two fixed registers. So i checked whats there inside those.



So i made a python program to depict how the password is checked.

s="isrveawhobpnutfg\260\001"

p="giants"

l=[]

for i in p:

#print(s.index(i),end=" ")

l+=[s.index(i)]

l1=[]

l2=[]

a="abcdefghijklmnopqrstuvwxyz"

y=15

for i in a:

z=ord(i) & y

l1+=[[i,z]]

for i in l:

for j in l1[:16]:

if i==j[1]:

l2+=[j]

for k in l2:

print(k[0],end='')





And password is **opekma**

**Phase\_6**

Again i tried to run the code in gdb with a test tring “inctfj” and it got exploded.

So tried to analyse the asm code

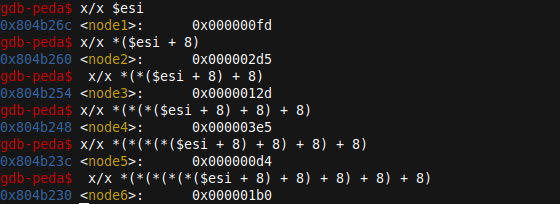
Disas phase\_6

calls a fun read 6 nums like in phase\_2 so input will be 6

0x08048e75 <+221>: cmp eax,DWORD PTR [edx]

So run the code the pointer the pointed towards 0x08048e75 using until \*0x08048e75

And checked the print the values of each node from esi register



To get the password i wrote a python code

l=["0x000000fd","0x000002d5","0x0000012d","0x000003e5","0x000000d4","0x000001b0"]

l1=[]

for i in l:

l1+=[[l.index(i)+1,int(i,16)]]

l1.sort(key=lambda x: x[1],reverse=True) #sorted the list based on converted int values

for j in l1:

print(j[0],end=" ") #prints position



**Verification**

