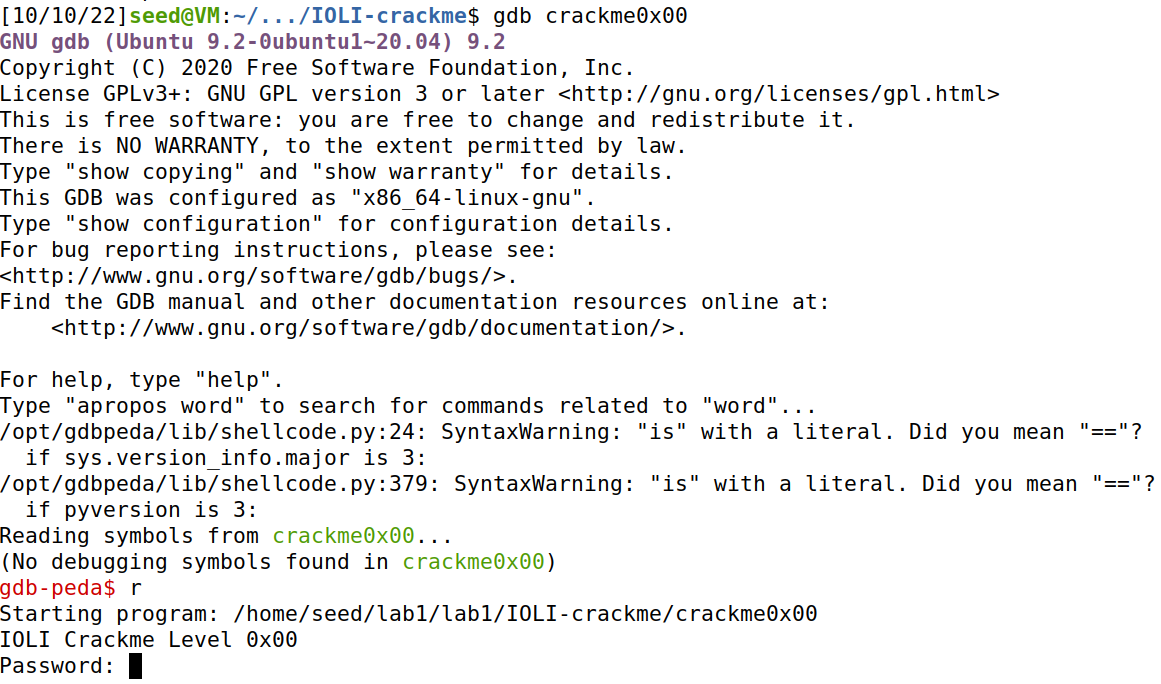
Jourdon Freeman

862006435

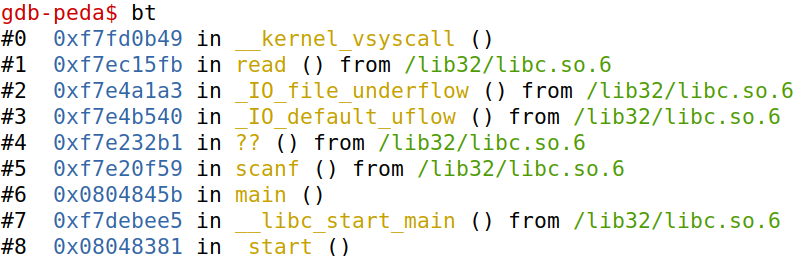
CS 255 Lab 1: Reverse Engineering

# Introduction

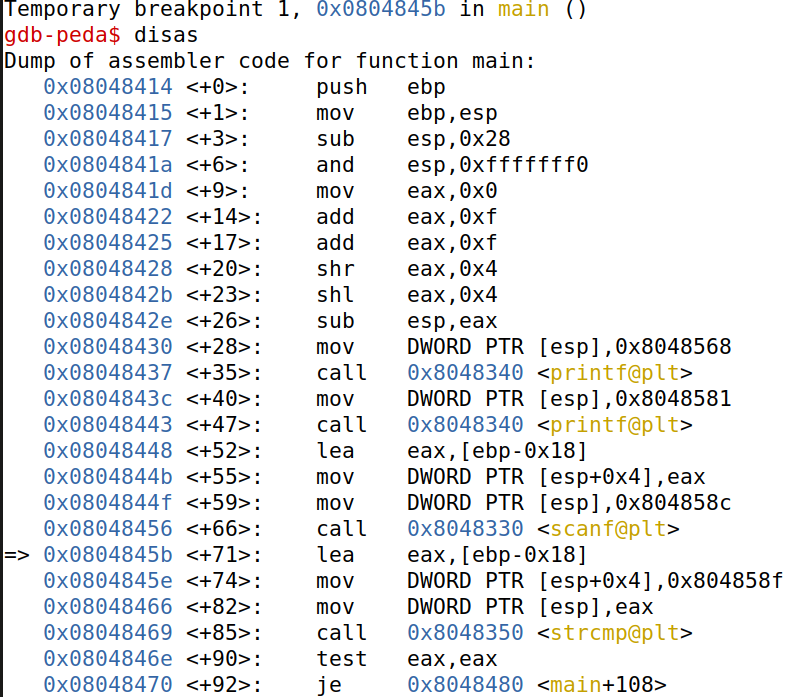
This lab is about using gdb tools to reverse engineer the criteria for generating a correct password for 7 different binaries. The provided VM for VirtualBox was used as the environment. Each program is run through gdb (crackme0x00 will be used as an example).



Once the program asks for user input, press ctrl+c to pause and use the bt command to print what’s on the stack.

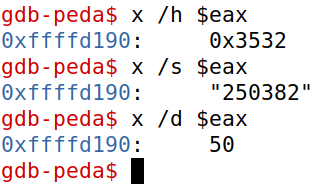


Using tbreak \*0x0804845b will add a breakpoint in main. Using the c command will resume the program. After entering the password, the breakpoint will activate. Using the disas command will print out the assembler code for the current function.



The arrow indicates where the program currently is (in this case, where the breakpoint was placed). Using the tbreak command on a line that has the call instruction will pause the program at the start of the function being called. Once there, the disas command can be used to print that functions assembler code as well. The n command can be used to move to the next instruction.

The x command will print the data at the memory location specified. Depending on the flag used, the command will print out the data in decimal, string, or hex format.



If the user input is correct, the program will output a variation of “Password OK :)” and “Invalid Password!” if incorrect. Using these commands, the criteria for the passwords of the given binaries can be determined.

# crackme0x00

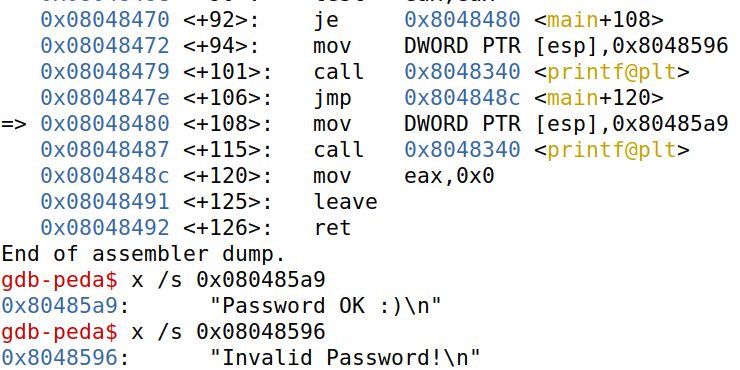
The first binary only has a main function. After the user’s data is inputted, both it and the string at address 0x0804858f are compared using the strcmp function.

Ox804844f 
ox8048456 
Ox804845b 
Ox804845e 
ox8048466 
ox8048469 
Ox804846e 
ox8048470 
s-m-ain+59>: 
in+66> : 
in+82> : 
in+85> : 
<ma in+90> : 
in+92> : 
mov 
call 
lea 
mov 
mov 
call 
test 
DWORD PTR [espl , ox804858c 
Ox8048330 
eax, [ebp-0x181 
DWORD PTR 
DWORD PTR 
ox8048350 
eax , eax 
ox8048480 
[esp+Ox4] , Ox804858f 
[espl , eax 
<main+108> 

The x commands shows the string’s value to be 250382



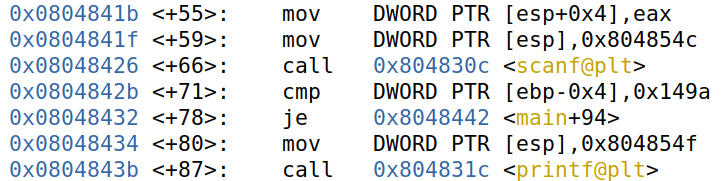
The test instruction checks if the user input and the string are the same. If so, the stack pointer will jump to line 108 in main and print “Password OK :)”. If not, it’ll print out “Invalid Password!”



**Answer:** The user input must be equal to the string “250382”

# crackme0x01

Similar to the first binary, this one takes the user input and compares it to a value. In this case, the user input is compared to hex value 0x149a or 5274 in decimal

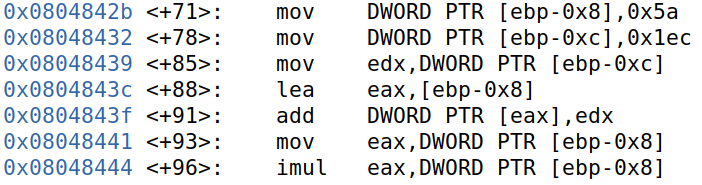


If the input is equal to 5274, the password will be accepted and rejected if otherwise.

**Answer:** The user input must equal hex value 0x149a which is 5274 in decimal

# crackme0x02

In this binary, hex values 0x5a and 0x1ec, which are 90 and 492 respectively in decimal, are added together then squared resulting in the value 0x52b24 which is 338724 in decimal.

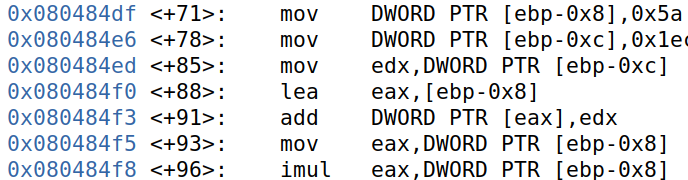


If the user input is equal to this result, the password will be accepted.

exo 
ox08048451 
ox08048453 
Ox0804845a 
Ox0804845f 
ox08048461 
ox08048468 
Ox0804846d 
ox08048472 
ox08048473 
<+123»: 
125> : 
cmp 
j ne 
mov 
call 
j mp 
mov 
call 
mov 
leave 
ret 
eax,D 
ox8048461 
DWORD PTR 
ox804831c 
Ox804846d 
DWORD PTR 
ox804831c 
eax , OXO 
[ebp-0xc] 
+125> 
<rna 1 n 
[espl , ox804856f 
137 > 
<rna 1 n 
[espl , Ox804857f 
End of assembler dump. 
gdb-peda$ x /s Ox0804856f 
Ox8€4856f : 
"Password OK 
gdb-peda$ x /s Ox0804857f 
Ox804857f : 
"Invalid Password! 

# crackme0x03

Similar to the previous binary, 0x5a and 0x1ec are added together then squared resulting in 0x52b24.



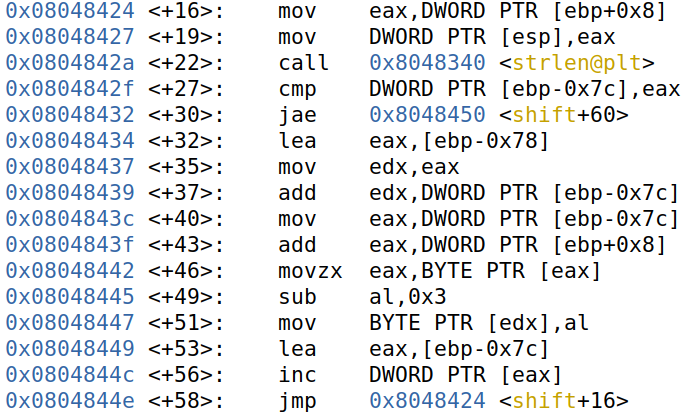
This time however, the user input(arg[0]) and 0x52b24(arg[1]) are used as the parameters of the test functions which is called on line 116 in main.

ox804850c 
ox8048511 
ox8048516 
ox8048517 
ox8048518 : 
<main+116>: 
<main+127>: 
nop 
call 
mov 
leave 
ret 
Ox804846e 
eax, OXO 
Guessed arguments: 
arg Ox852ae 
arg Ox52b24 

Within the test function, these values are compared and the shift function is called. The argument of the shift function will change depending on whether the user input matches 0x52b24. If it does, the string passed into shift will be “Sdvvzrug#RN$$$#=”. If it doesn’t it’ll be “Lqydolg#Sdvvzrug$”.

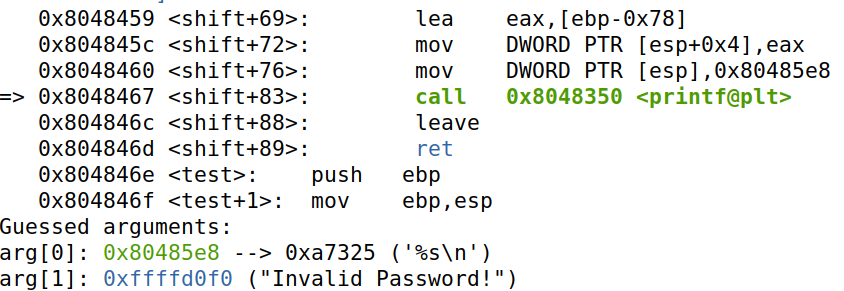
ox08048477 
Ox0804847a 
ox0804847c 
ox08048483 
ox08048488 
Ox0804848a 
ox€8048491 
ox08048496 
ox68048497 
End of assembler dump. 
cmp 
je 
mov 
call 
j mp 
mov 
call 
leave 
ret 
eax, DWORD 
Ox804848a 
DWORD PTR 
ox8048414 
ox8048496 
DWORD PTR 
PTR [ebp+Oxc] 
<test 
[espl , Ox80485ec 
<shift> 
<test 
[espl , Ox80485fe 
gdb-pedaS x [s Ox080485ec 
Ox80485ec : 
" LqydoIg#Sdvvzrug$" 
gdb-peda$ x /s Ox080485fe 
Ox80485fe: 
"Sdvvz , " 

The shift function will iterate through the string passed in and shift it’s ASCII/Unix value by subtracting 3 from it (e.g. ‘S’, which is 0x53, will become ‘P’ which is 0x50).

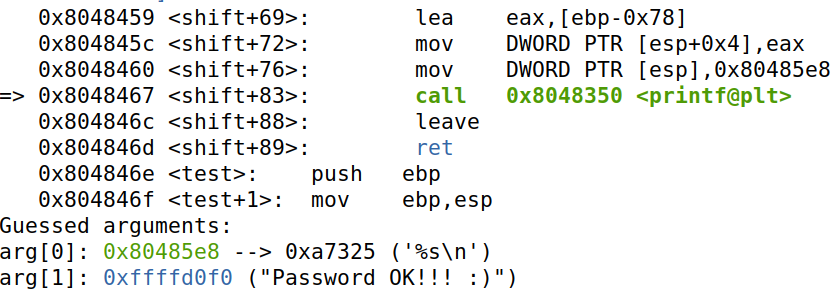


The shifted string will be the output.

When user input does not equal 0x52b24:



When user input does equal 0x52b24:



**Answer:** The user input must equal 338724 which is 0x52b24 in hex

# crackme0x04

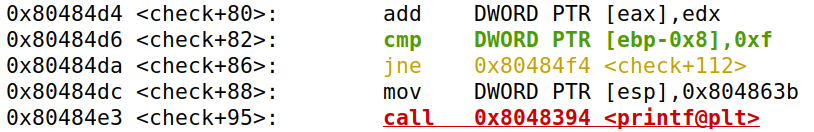
For this binary, the user input is passed into the check function

Ox804854e 
ox8048553 
ox8048556 
ox8048559 
Ox804855e 
eax, OXO 
ox8048563 
ox8048564 
ox8048565 : 
in+69> : 
call 
mov 
call 
mov 
<ma in+90> : 
leave 
in+91> : 
ret 
nop 
Ox8648374 <scanf@plt> 
eax, [ebp-Ox78] 
DWORD PTR [esp] ,eax 
ox8048484 
Guessed arguments: 
arg [O]: 
Oxffffd130 
ox3738 

Inside the check function, a loop will iterate through each character in the input string. During the iteration, the sscanf function is called to check if the current character is a number.

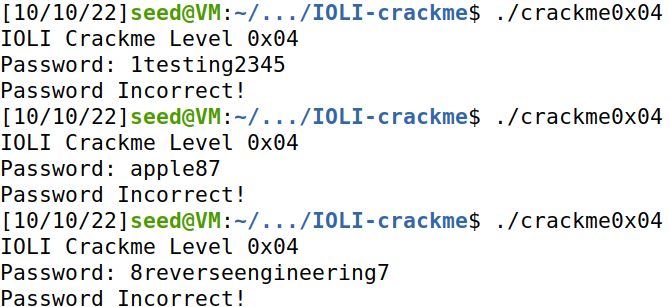
Ox80484bb 
ox80484c3 
ox80484c6 
ox80484c9 
Ox80484ce 
Ox80484d1 
Ox80484d4 
Ox80484d6 
<check+55>: 
hec k+63> : 
hec : 
hec k+69> : 
heck+74>: 
<check+77>: 
hec k+80> : 
heck+82> : 
mov 
mov 
catt 
mov 
add 
cmp 
Guessed arguments: 
arg 
Oxffffd0fb 
arg[l]: ox8048638 
. Oxffffd104 
ox38 
Oxf7ffd990 
DWORD PTR [esp+0x4] , 
eax, [ebp-Oxd] 
DWORD PTR [espl ,eax 
Ox80483a4 
edx,DWORD PTR [ebp-Ox4] 
eax, [ebp-Ox8] 
DWORD PTR [eax] ,edx 
DWORD PTR [ebp-Ox8] , oxf 
Oxo 

If the character is a number, it is added to the sum variable (ebp-0x8). The sum variable is then compared to 0xf which is 15 in decimal. If the sum doesn’t equal 15, the loop will continue until it runs out of characters.

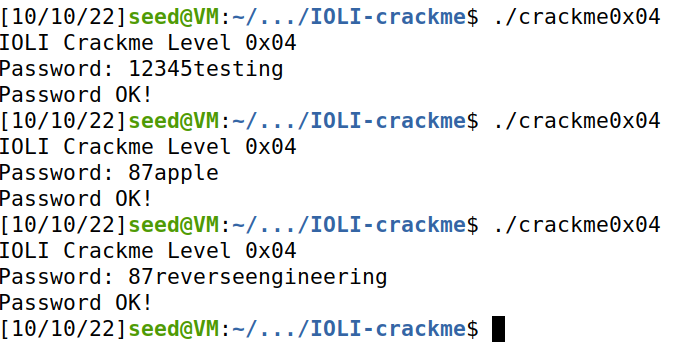


While the order of the numbers doesn’t matter (e.g. 78 vs 87 or 12345 vs 53124), any non-number character, like an alphabetical character, placed before or in the middle of the summation will render the string invalid.

For example, these string will not pass:



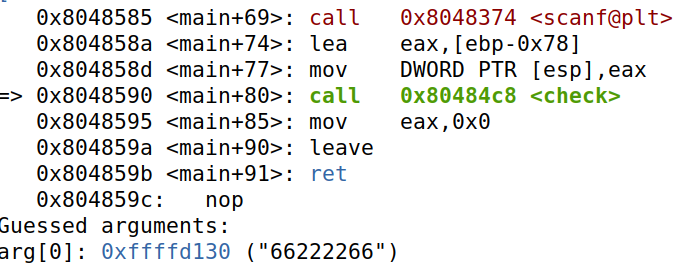
If these non-numbered characters are placed at the end, the string will pass:

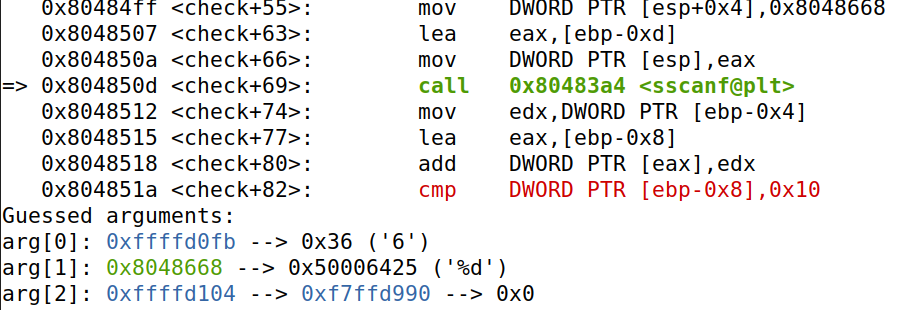


**Answer:** the beginning of the user input must be a set of numbers from 0 to 9 that add up to exactly 15. Any non-numbered characters placed before or in the middle of the summation will result in an invalid string.

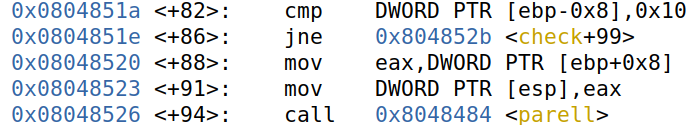
# crackme0x05

Similar to the previous binary, the user input is passed into the check function which iterates through the input and uses the sscanf function to check if the current character is a number.

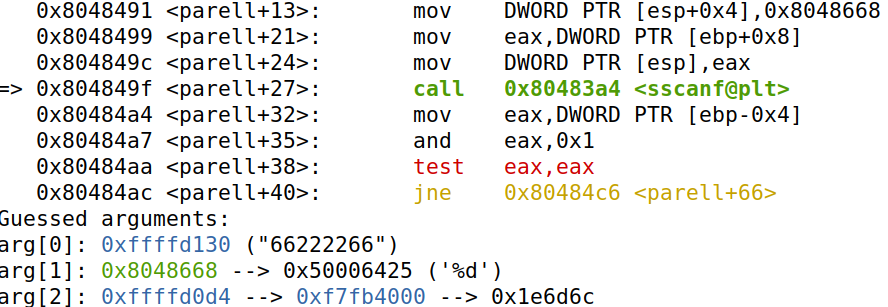




If the current character is a number, it will be added to the sum variable (ebp-0x8) which is then compared to x10 or 16. If the sum of the characters checked so far equal 16, the loop will break and the parell function will be called.



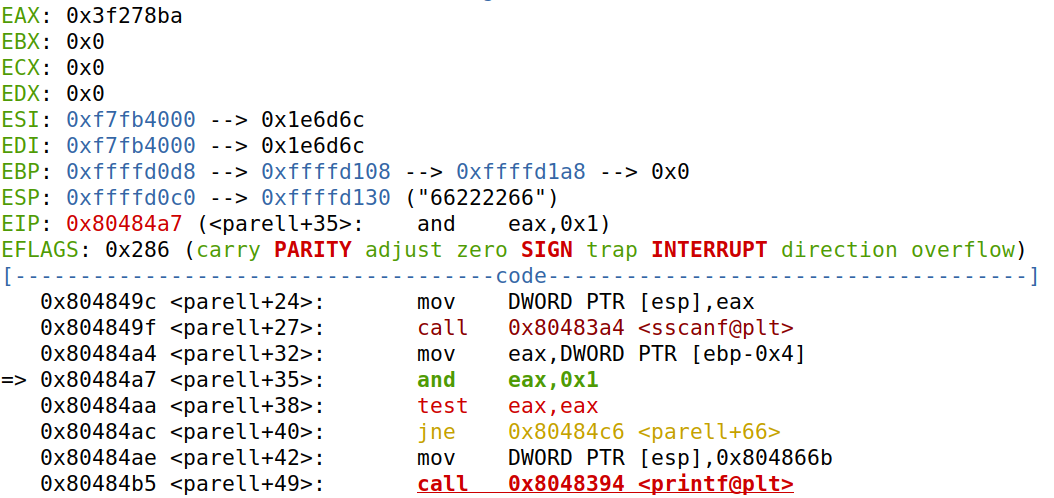
Within the parell function, sscanf is called again to turn the input into a decimal.

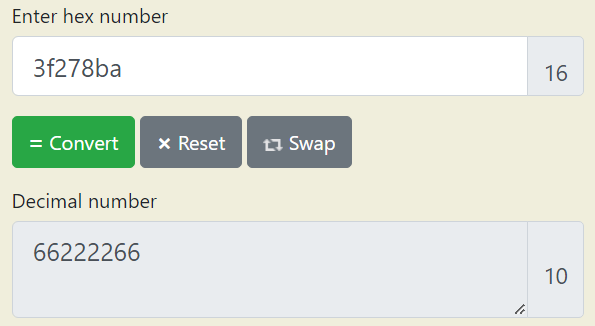


Before sscanf (check eax):

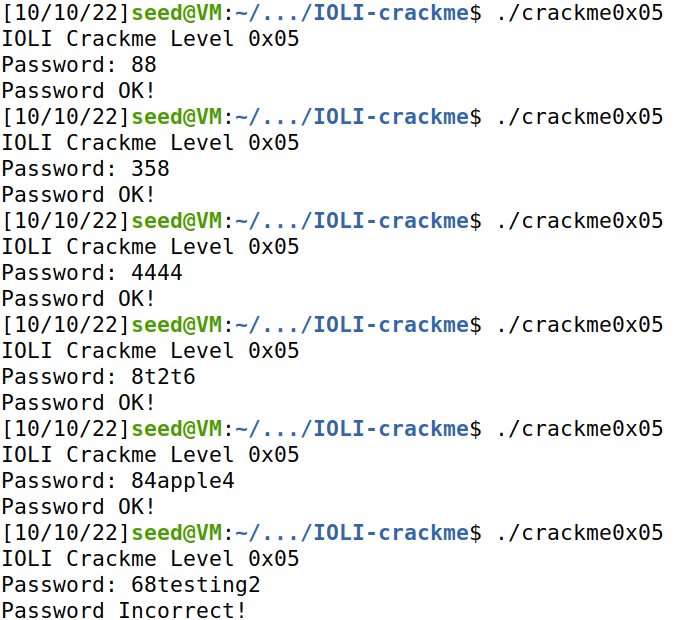


After sscanf (check eax):





The input is then anded with 1 and compared to 0 using the test instruction. This ensures the final character is even (e.g. 853 and 8611 will fail but 538 and 8116 will pass). Non-numbered characters can be added to the end of the summation. If put in the middle, the numbers in front of them must be multiples of 2. If the test is passed, the password will be accepted.



**Answer:** The user input must be numbers that add up to 16. The final character that adds up to 16 must be an even number. Non-number characters cannot be added to the front of the input string but can always be appended. If the non-numbered characters are in the middle of the summation, the number’s in front of it must be multiples of 2.

# Results

