Explanation:

Program:

Takes a string as command line input and compares it with the correct password (i.e Pass123).

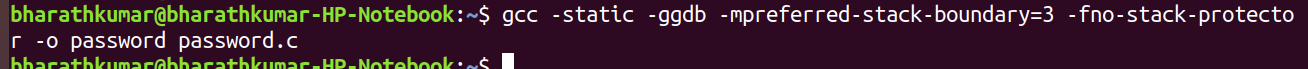
A function “compare” is called which takes 2 arguments, the user input and the correct password.

Output is printed depending on the result of the comparison.



Compile:

This program is compiled in 64 bit with stack boundary size set to 8. Therefore each entry in stack is of 8 bytes in size.

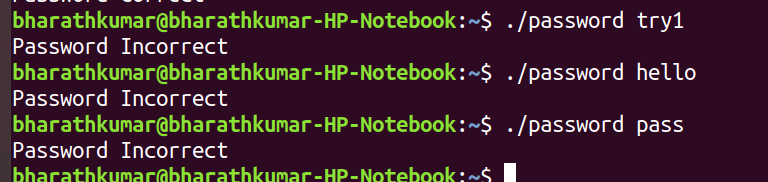


-fno-stack-protector = to remove canaries in stack

-static = to prevent dynamic linking of libraries

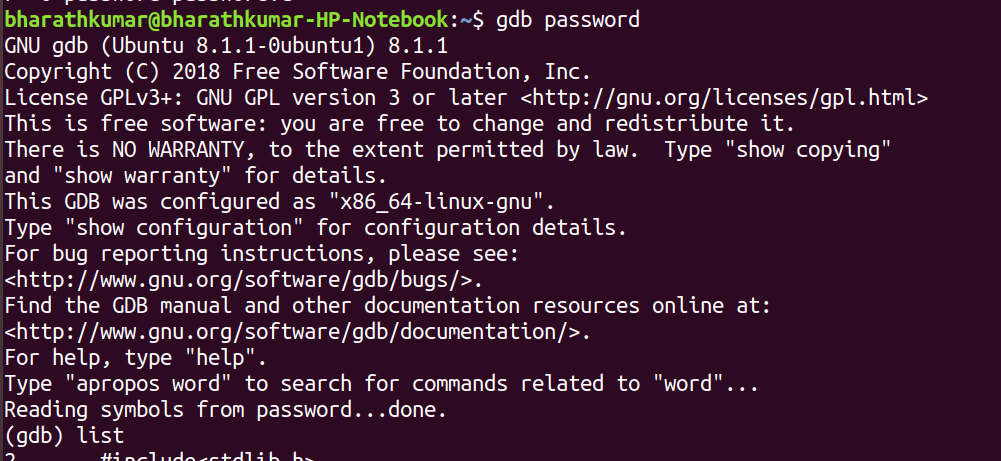
-ggdb = GDB for debugging

Test:

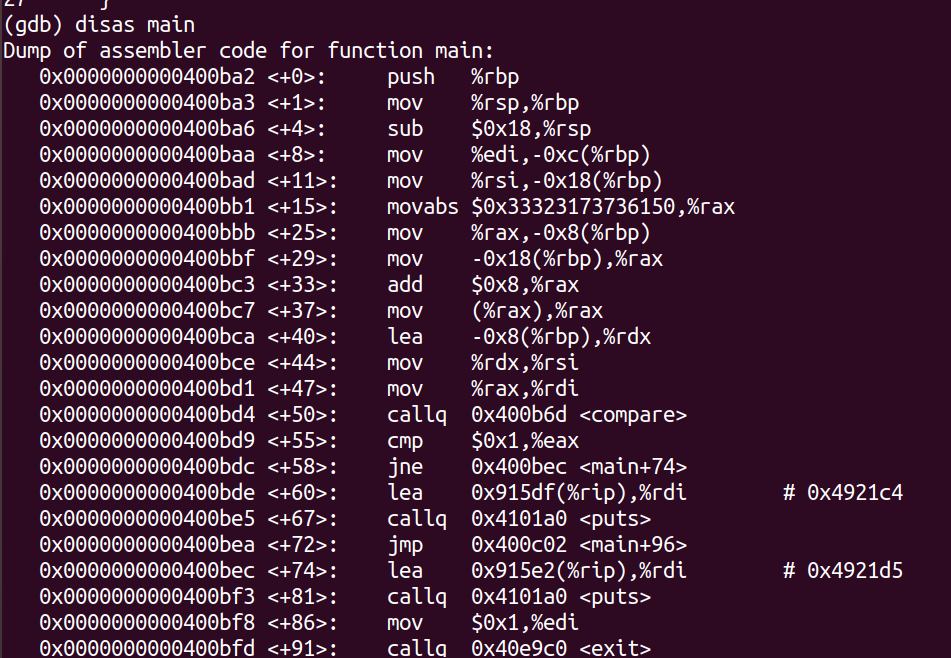


Incorrect password tries.

Running program using GDB:



1)Disassemble the main program



At \*main + 50, the call to compare function is made. Above that it is seen that the arguments to compare function are moved to %rax and %rdx.

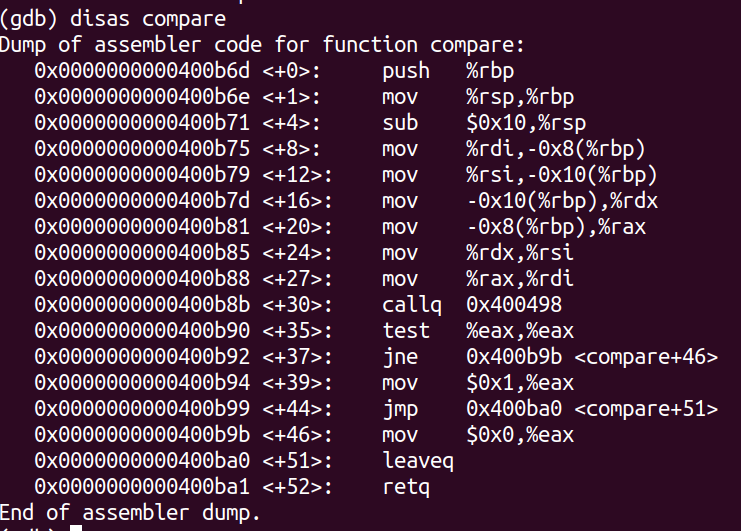
rax -> mov -0x18(%rbp) , %rax

rdx -> lea -0x8(%rbp) , %rdx

These arguments should be the user input and correct password.

To inspect the stack at that point, breakpoints are set at \*main + 50 ,\*main+55 (before and after call to compare).

Disassemble compare function



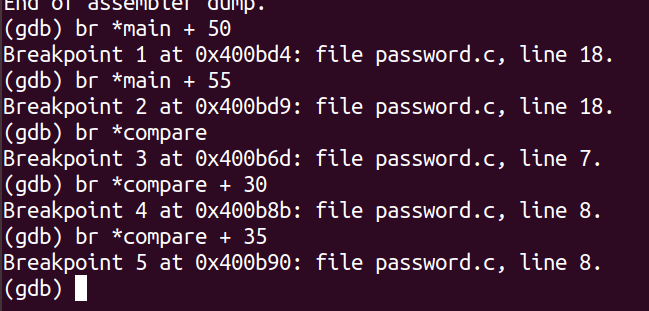
Similar to main function, a call to strcmp() is made at \*compare + 30.

In the next line a test is made with the return value in %eax and based on the result, a jump instruction is executed.

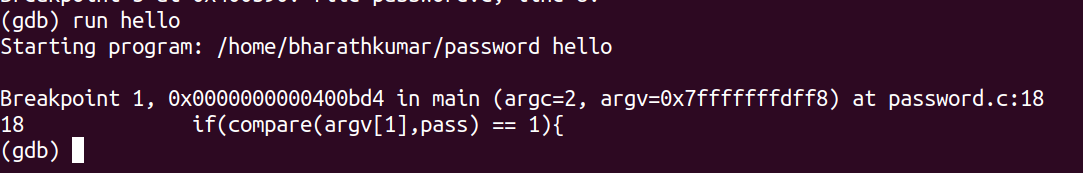
Again the stack contents here before function call should be the 2 arguments which are also stored in the registers rax, rdx.

Breakpoints are set at \*comapre, \*compare + 30 and \*compare +35.

Breakpoints

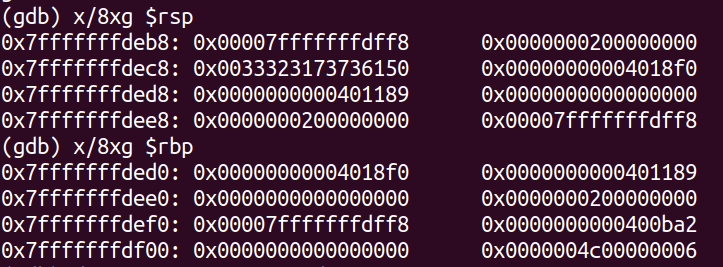


Program is run with user input. Lets say its run with “hello”.



It breaks at the first break point in main function before calling compare function.

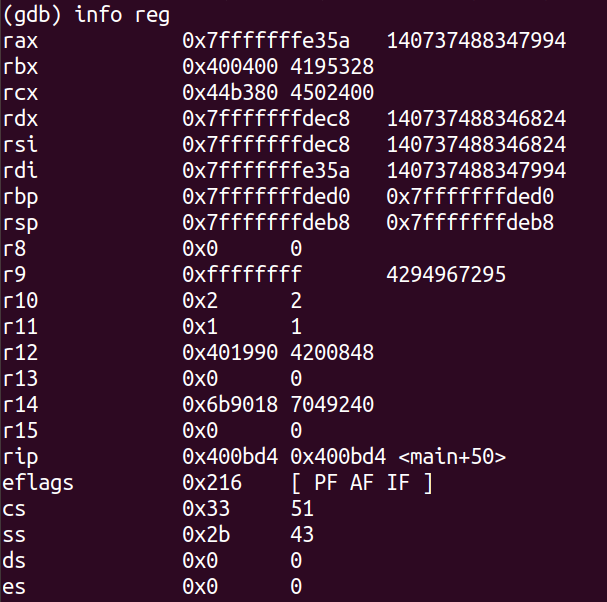
Printing rsp and rbp pointers



There 3 8-byte chunks above rbp value which might be the local variables or arguments pushed for next function call.

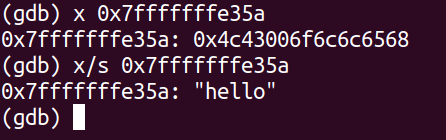
But printing them didn’t give proper values.

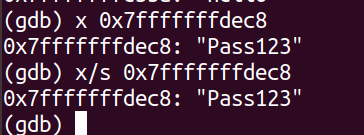
Inspecting register contents:



From disassembly rax and rdx are pushed with arguments before function call.

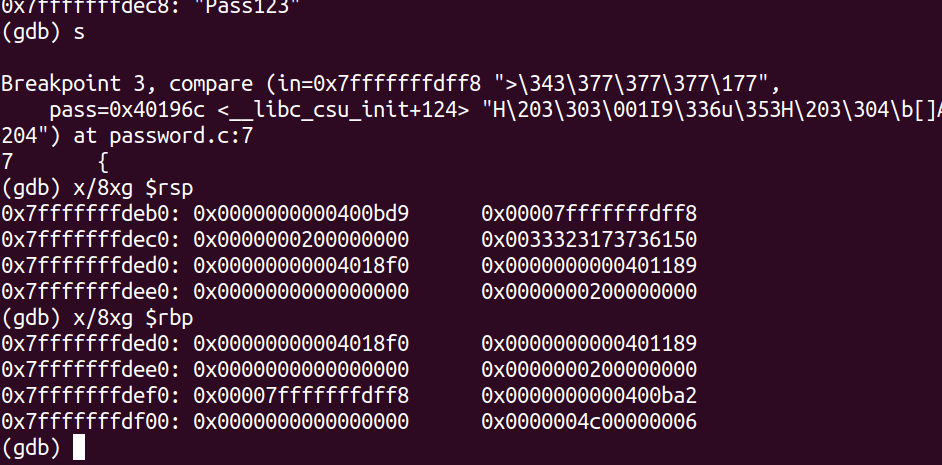
Printing their contents:





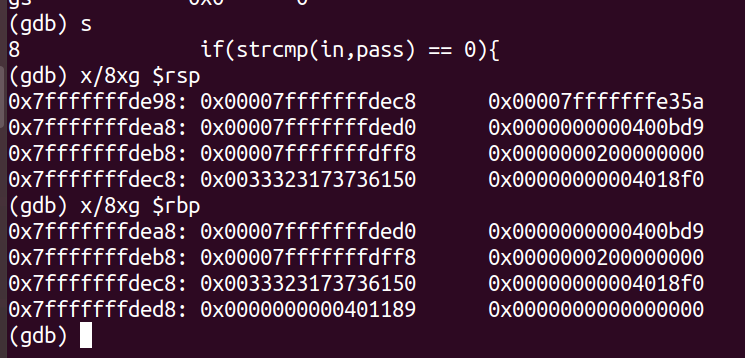
We see that rax contains the input we gave which is “hello” and rdx contains the correct password which is “Pass123”. As this is the only point where comparison is made, we can be sure that rdx stores the correct password.

2)Stepping through the program



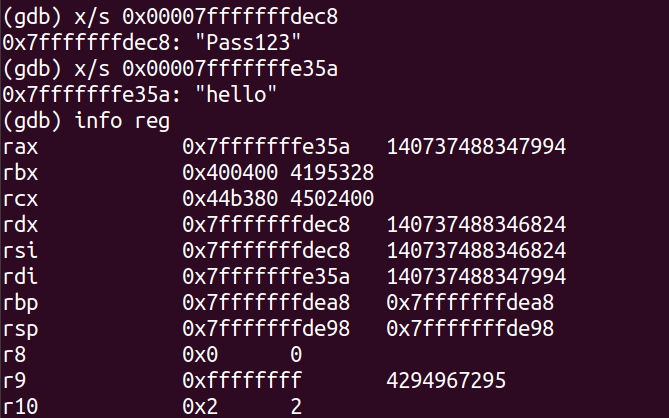
Breakpoint after entering the compare function.

Ebp value still remains the same which might mean that the function call is not complete yet and the arguments haven’t pushed yet. Stepping through the function one more line.



Now we see that there are exactly two 8 byte chunks above rbp pointer (0xded0) and this might be the two arguments which are pushed before strcmp() call.

Printing those two values:



We get the expected result and also these are the same values stored in rax and rdx registers as seen earlier.

Now the correct password is revealed and it can be used while running the program the next time.

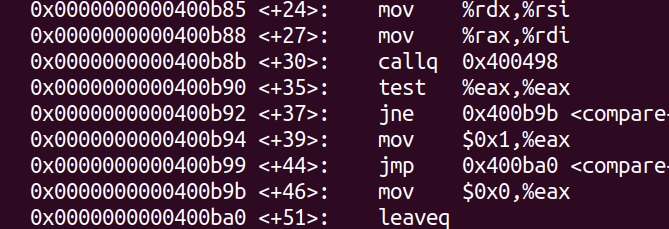
Another way is to alter the register value that points to the correct password. Changing it to the value we gave as input will let the program authenticate our current input directly. But this is just a temporary change as we are just changing stack contents and not the actual correct password.

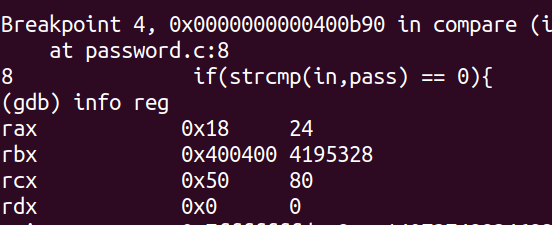
%rdx contains the correct password and we need to replace its contents with the user input string which is “hello”.

Aliter:

strcmp returns 0 if the strings match and non-zero if the strings don’t. The return value is usually stored in eax register.

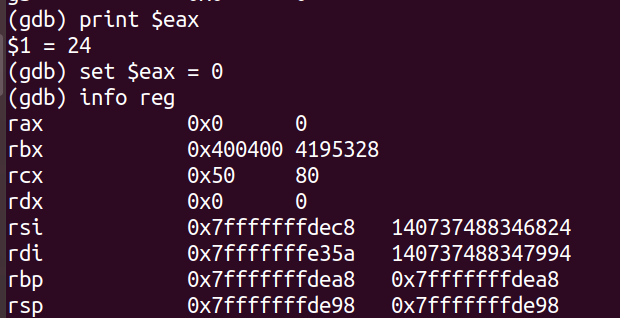
Therefore we can modify the eax register and set it to 0 , before comparison is done and this will allow the true condition to be satisfied irrespective of the passwords.





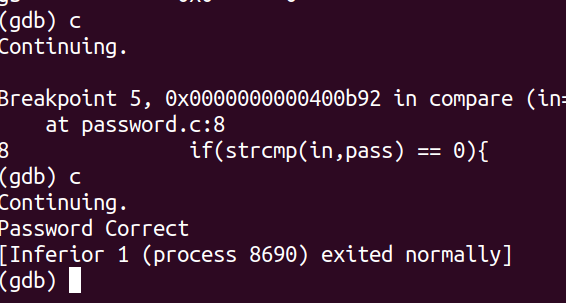
Value of rax (eax) is 24 which is non-zero.

We need to set it to 0 to enforce that the 2 strings are same.



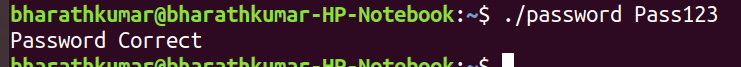
Setting eax to 0 and checking it by printing register contents.

Now the test of eax should return true and Password correct should be printed.



Continuing we see that the Password is shown to be correct but the input we gave is still “hello” which is different. Thus we bypassed the authentication step.

Another direct method now is rerun the program again with the correct password as it is revealed now.



Thus the Password is obtained by examining the given executable using GDB.