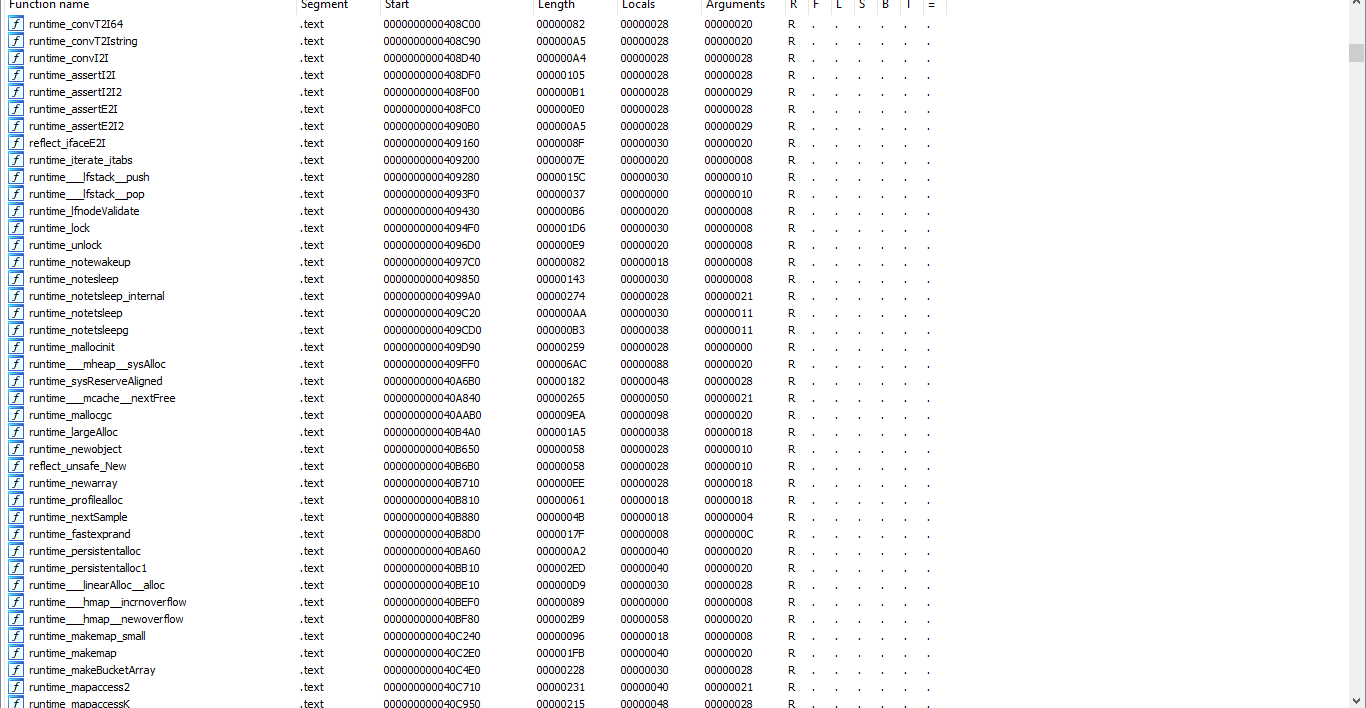
# Basics:

A lot of functions embedded:

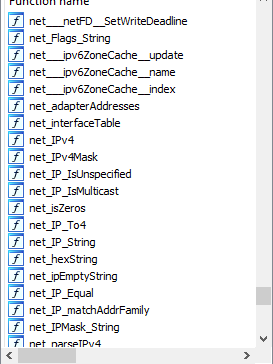


# CnC:

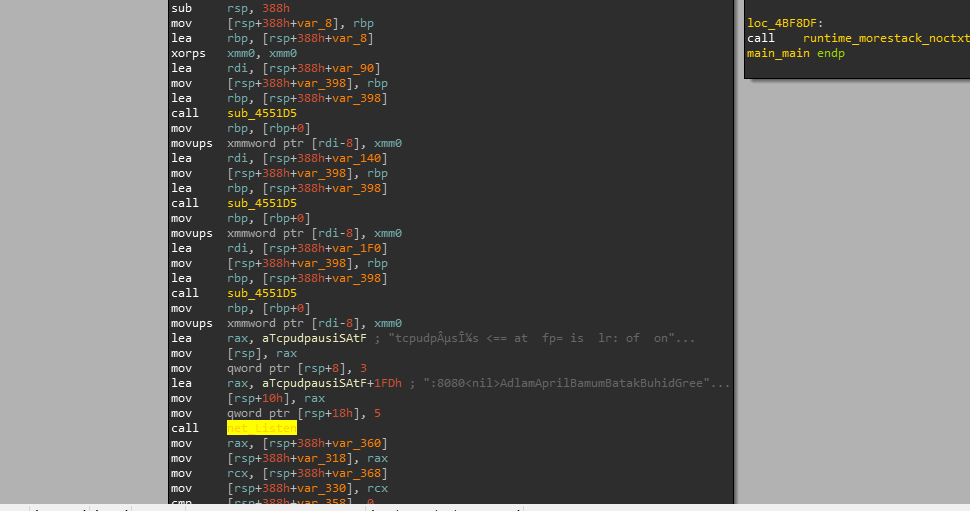
A lot of function inside the binary we did not author, but instead the compiler thought it was essential to add it.

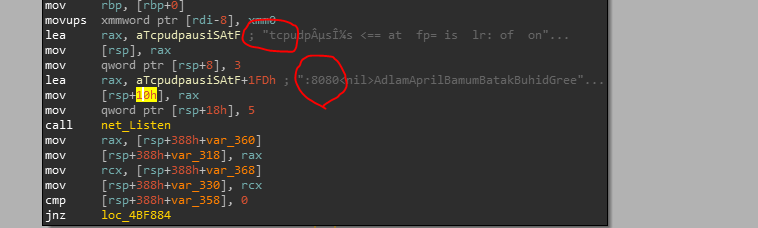
Runtime: used by go programs, like controlling goroutines,

Also a lot of package we imported are there build with the binary.

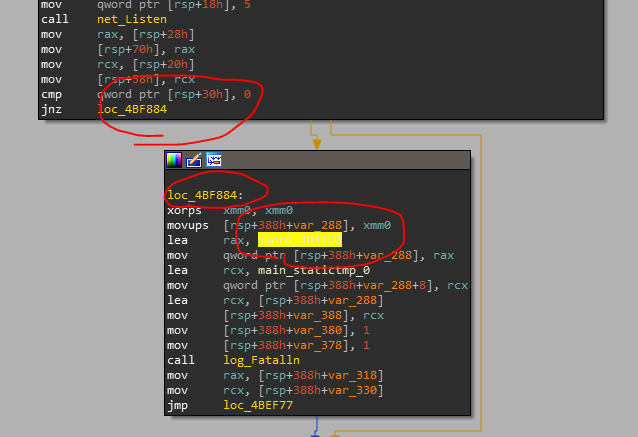


A lot of functions I did not write is also there, But lets get to the code we wrote

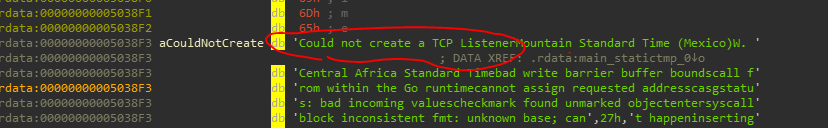




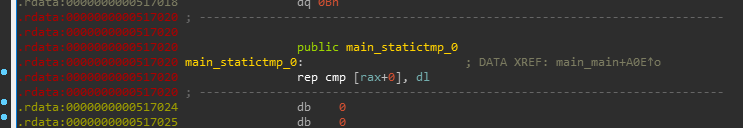
Net\_listen(“tcp”,3,”:8080”,5). Not what I originally wrote in code, it seems it is the length of the strings. IDA does not accurately decode the string. The return value does not seem to be in eax, rather on stack. The net\_listen() would return two values, ***THe listener, and the Error value***, here rsp+30h (rsp+358) is compared with 0, which is our error handling condition, we can assume that it holds the err value.



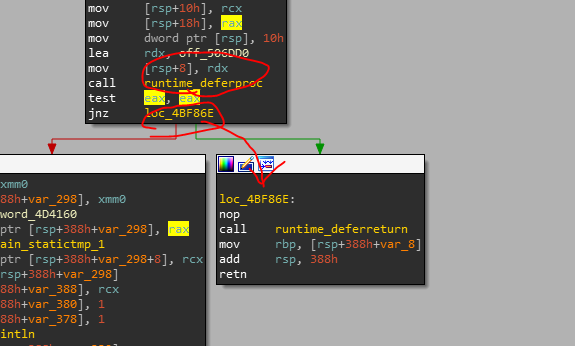
If the err is not equals to zero than jump to the highlighted location, call log\_fataln. When mainStaticint is used, which seems to be the string that compiler uses (**NONO NOT SURE**) it self than it uses a pointer to string, You gonna have to convert the pointer to offset, then jump to offset and then press A to convert to string.

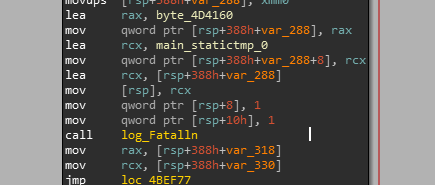


Else by default treated as code:



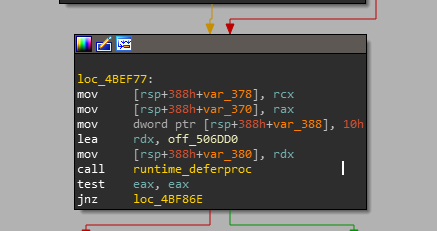
In disassembly deferring a function call can be identified with the use of runtime\_deferproc() function. And it seems when during the use of runtime\_deferproc() fcuntion, the return value seems to be passed in eax. And compiler added its own error handling in here.





Error Handling Section.

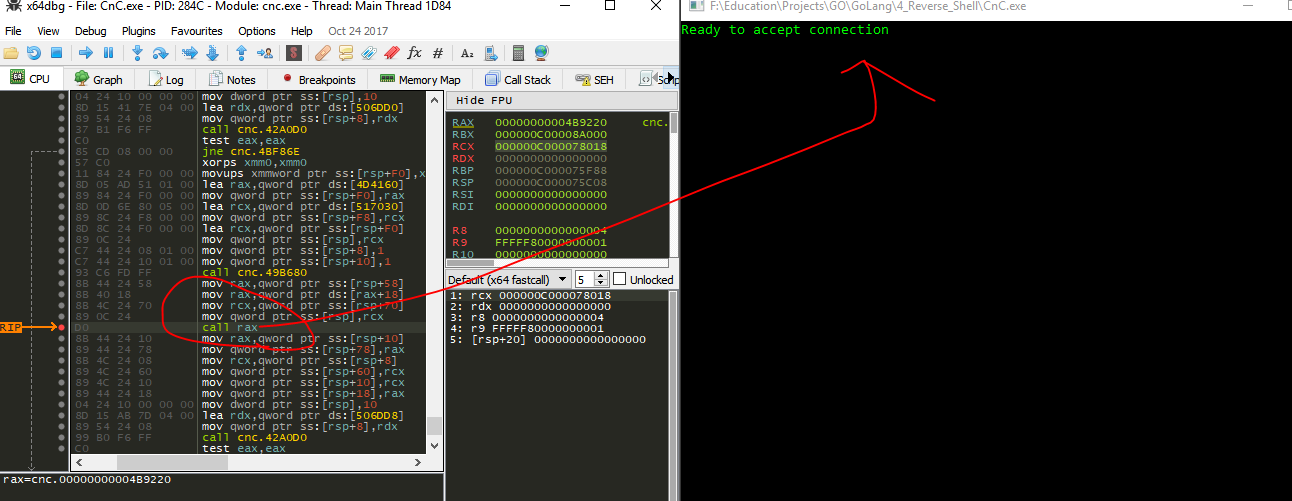
Log\_Fatalln(1,1,main\_statictmp\_0(holds pointer to: “Could not create…”))



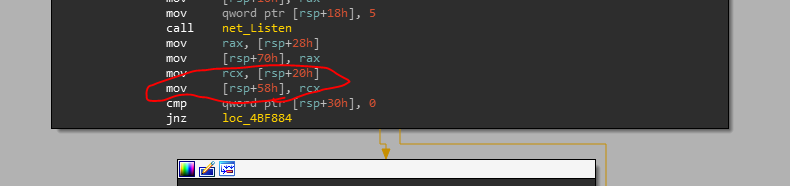
Defer

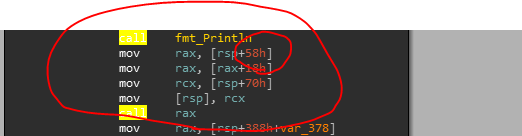
If no error, goes to print, "Ready to accept connection".

We see call to rax, What does this have?, lets check it in x64Dbg, breakpoint at that address



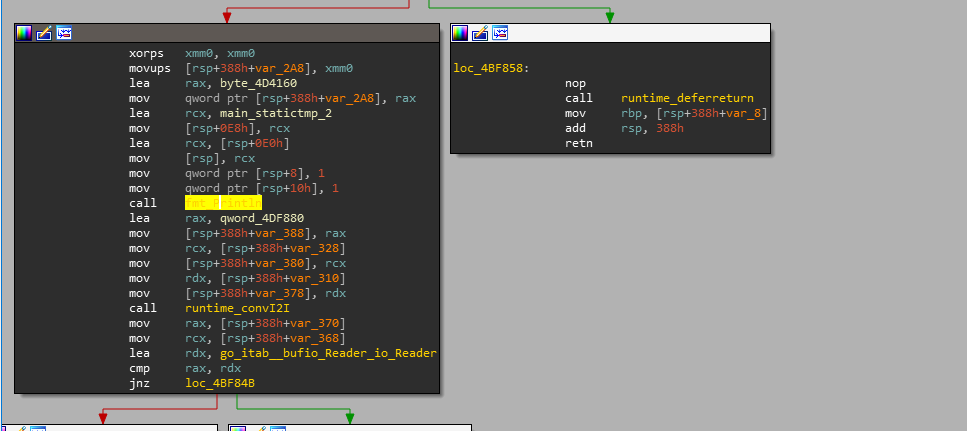
It seems to be calling the accep() function from source code. Hmmmm. I go to IDA to see what value was passed in rax, seems the value was from stack

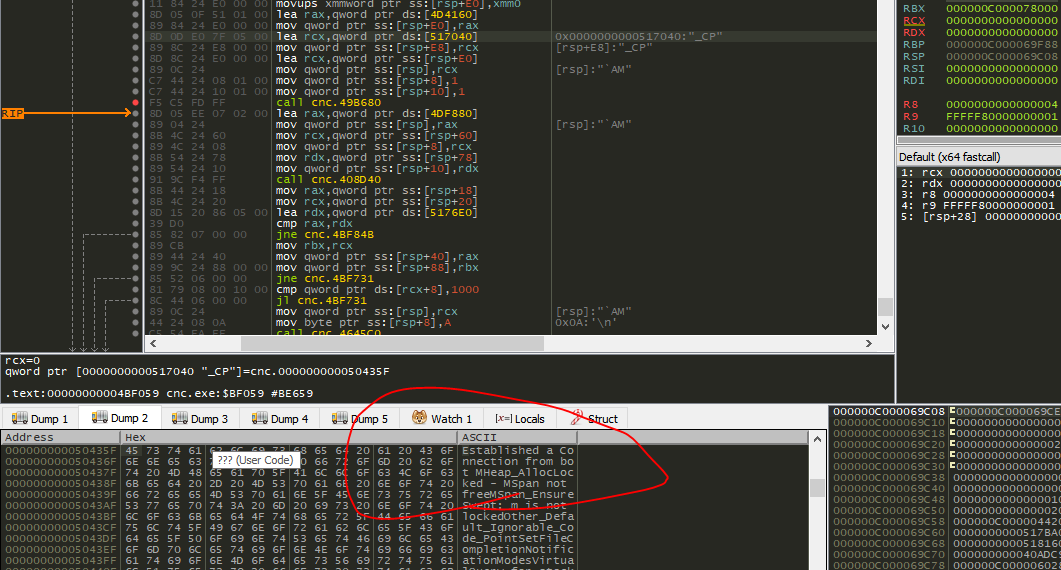




It seems the listener type is at offset rsp+0x58 and the accept() function is at rsp+0x58+0x18

When connection established, prints

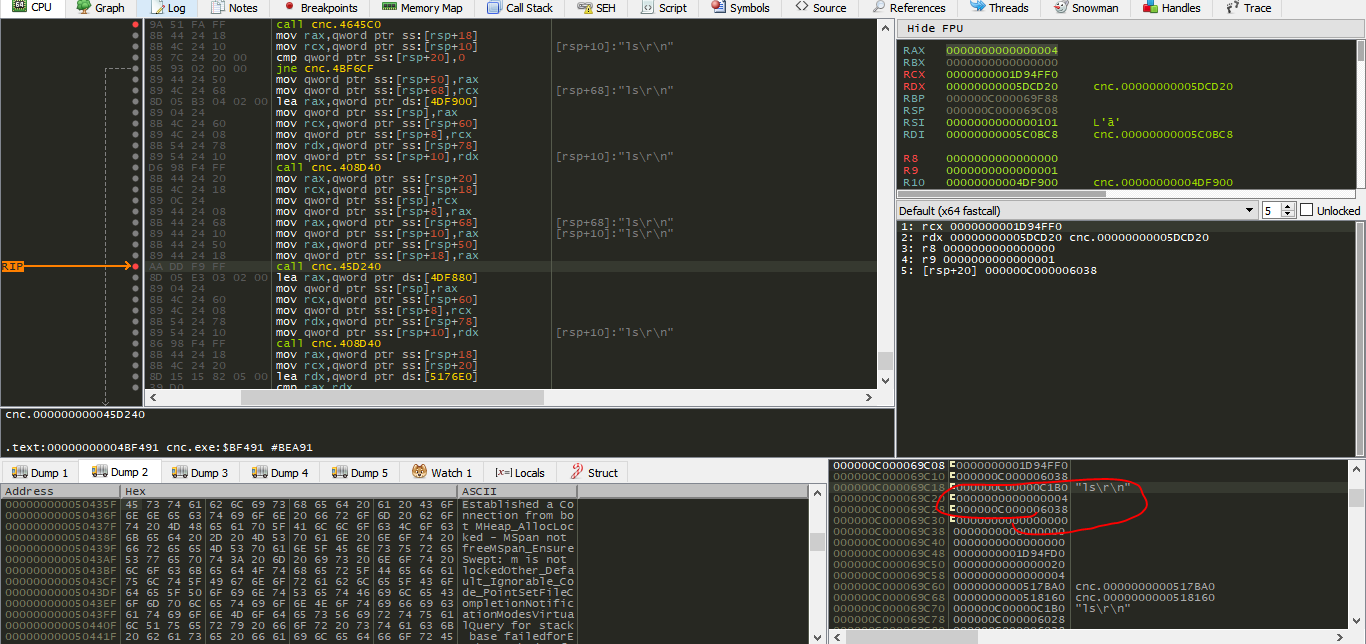




Similar stuffs with bufio reader, but before bufio.NewReader(conn).ReadString('\n') can be called, a lot of preparation has been added by the compiler. Among them was the call to runtime\_makeslice(), I guess since the user input can be of any length slice is used.

Next we have some printing and preparing for buffer reading again. And I set breakpoint to io.writestring() in x64dbg. I gave the input





Passed as the third argument. The loop continues and IDA graph can show it, the process repeats. And since I have not coded a way out of the lopp it continues till and error is received.

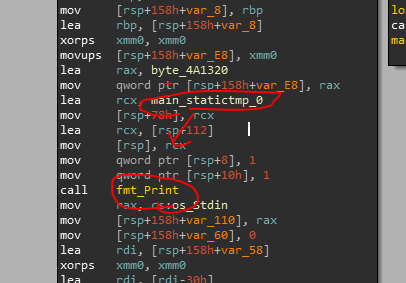
Before bufio\_reader\_Readstring() you get something like call to sub\_xxxx , rutime\_makeslice()

Crack:

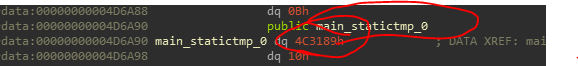
Already different then programs written in C. A lot of fucntions embedded in binary, we don’t know about. 

After a bit of digging, it seems the main function is main\_main.

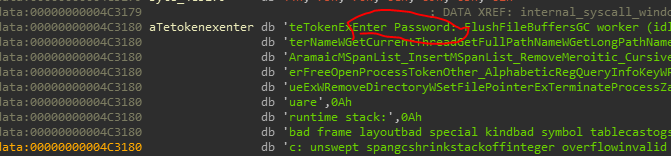
First call, fmt\_println() with, first arg, the string at top of stack, don’t know what the other two are,



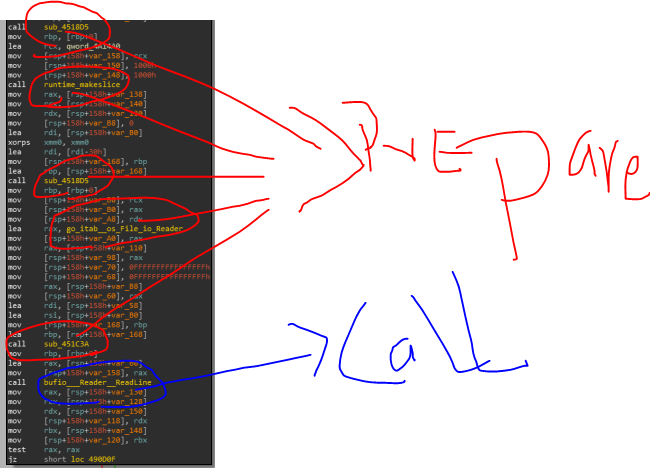
Main\_statictmp\_0 seems to be pointer,

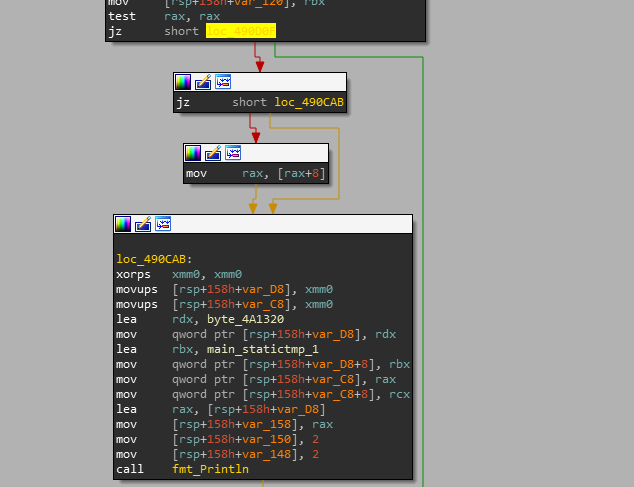


Points to string, Note that the section had to be changed to data use D.

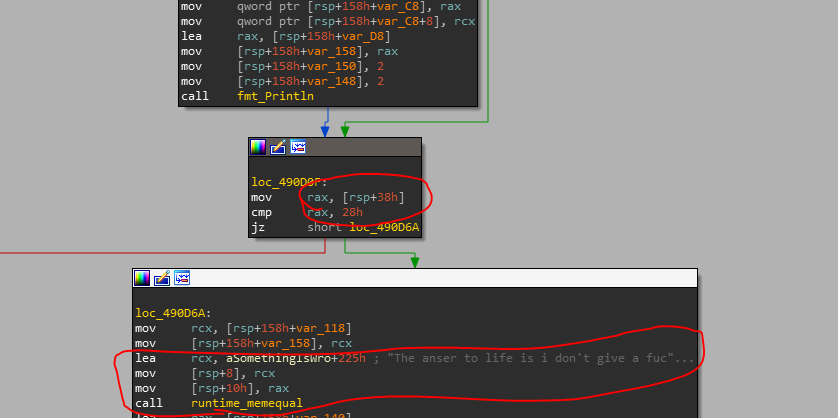


Since strings in go is not null terminated and rather Unicode string, IDA has some problems decoding them.

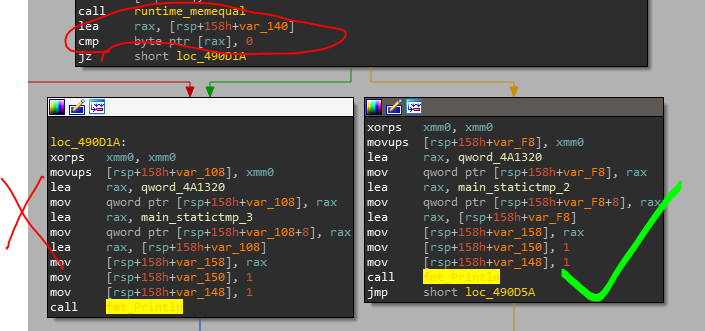
before the call to readline occurs, the programs seems to call other fucntions, slice() cause user input can be variable



This seems to be the error handling, jz (jump if zero), take the jump if value was zero, if no error was reiceved else continue man\_statictmp\_1 = pointer to string = string(Something is wrong with your computer)



Another unique thing that happens here, the return value of the readline() is not in rax, rather than in stack as we see it being used later, Compared with string length, which we did not add, perhaps the compiler added it to optimize, so that instead if length is not equal why waste time.



Here again, the returned value is again in stack, put into rax, compare with zero.

Here if zero, the value was not equal and program executes and prints you cracked. THe strings started to confuse me so I ran debugger.

