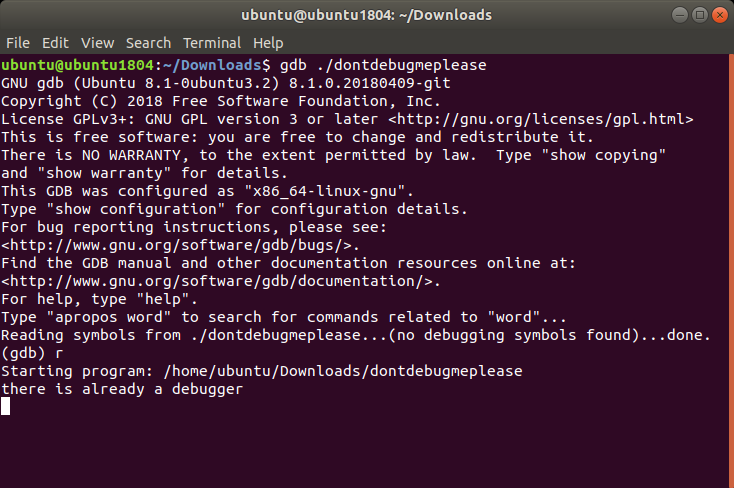
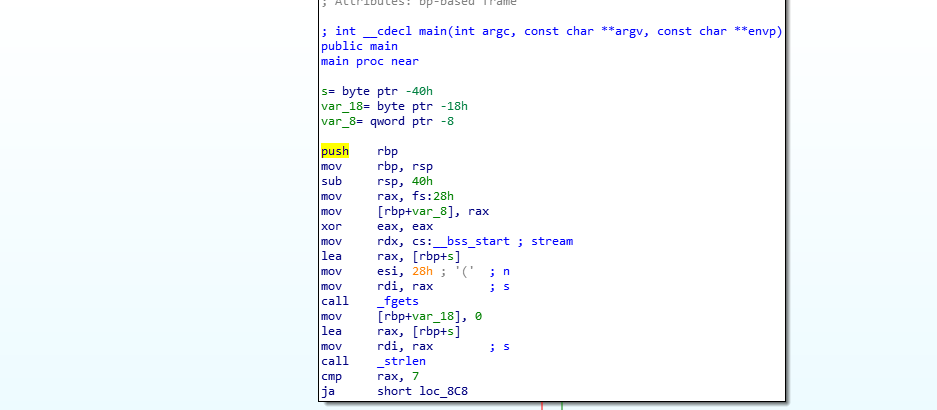
As the title say, the program detects if we are running it in a debugger:

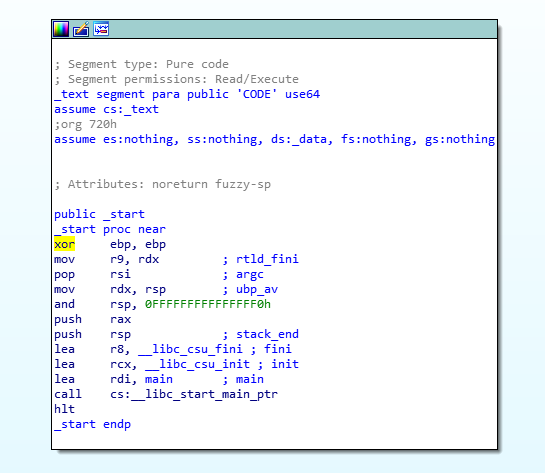


So an anti-debug technique is used. Disassembling the binary we see the main:

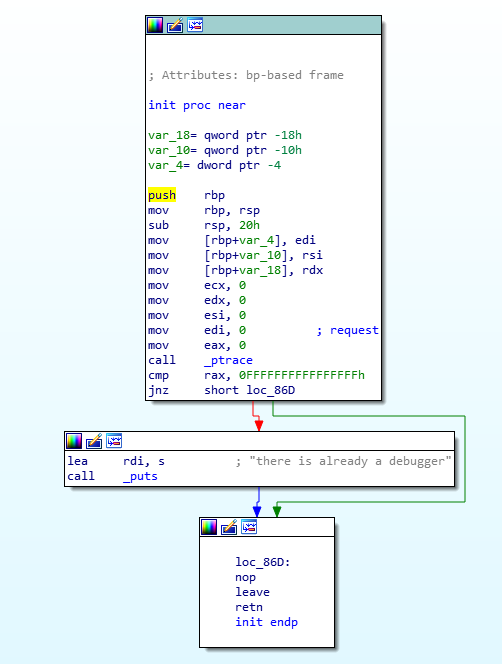


However, there are no calls to check if the debugger is running or the print “there is already a debugger”, so probably the check happens before the main starts.

We can then go to the entrypoint of the program, which is the function called \_start.



Looking at the function, we see that fini and init are defined. In particular, the init function is executed before the main, so it could contain the anti-debug check.

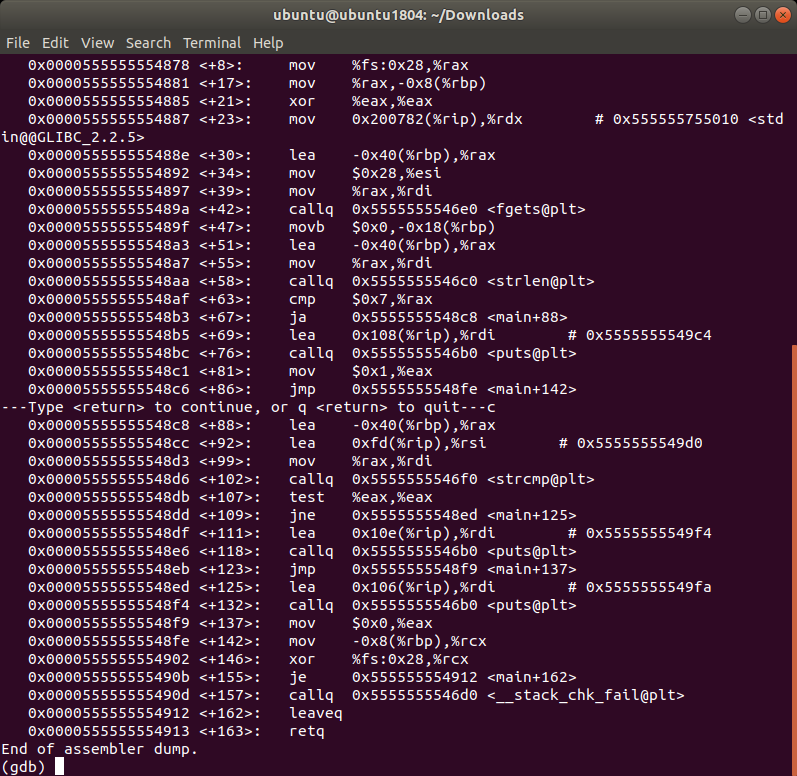


Our intuition was good! Here we can see the program calls \_ptrace to check if there is a debugging (rax == -1). We can patch the call \_ptrace instruction filling it with nops.

After the patch, we can use gdb to recover our flag.

The program waits for an input, and prints “nope” or “nope dude” if the flag is wrong, depending on the length of the input.

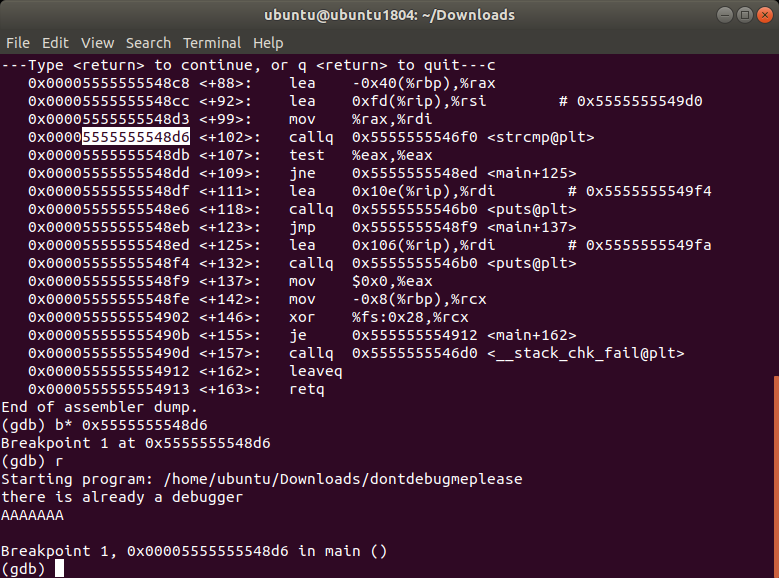
Let’s start disassembling the main



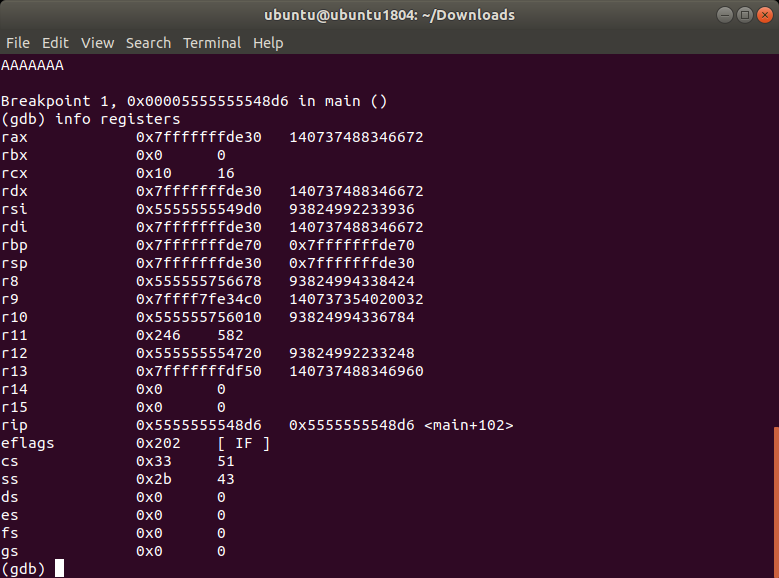
We can see that it calls fgets to get the user input, and then uses strlen to compare the input length with 7. If above, it continues the checks, otherwise it ends.

So we know that the input must be longer than 7.

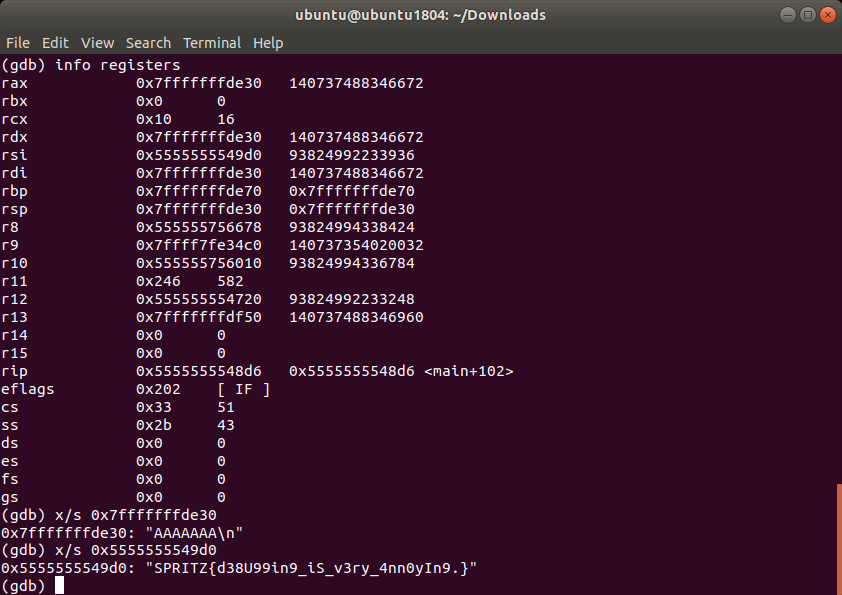
Then it uses a strcmp to check two strings, the user input again and another string, hopefully our flag. If we check the registers rdi and rsi before the strcmp call, we might be able to retrieve the flag.

So let’s put a breakpoint at the call address (0x5555555548d6), and run the program giving an input with 7 letters 

When we reach the breakpoints, we can inspect the registers using info registers



They contain memory addresses, so let’s inspect the memory, interpreting it as a string, using x/s 0xaddress



As we thought, the program compares the flag with our input.

Flag: SPRITZ{d38U99in9\_iS\_v3ry\_4nn0yIn9.}