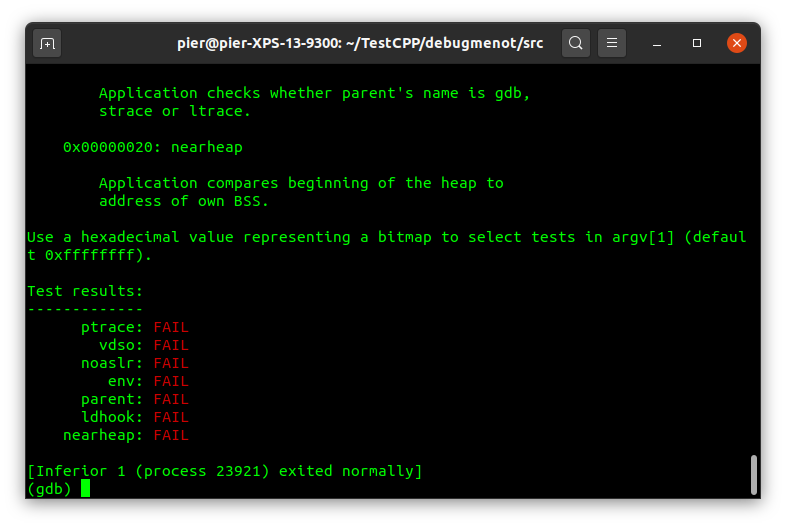
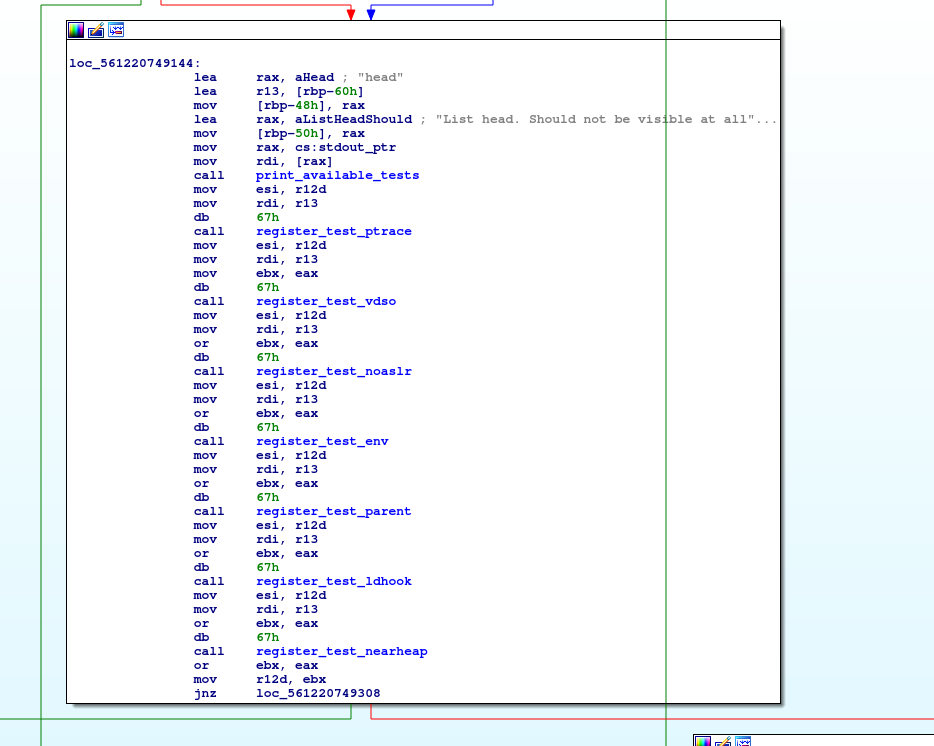
Debugmenot contains a set of tests to detect if gdb is running.

If we run the program inside of gdb, we see that all the tests fail

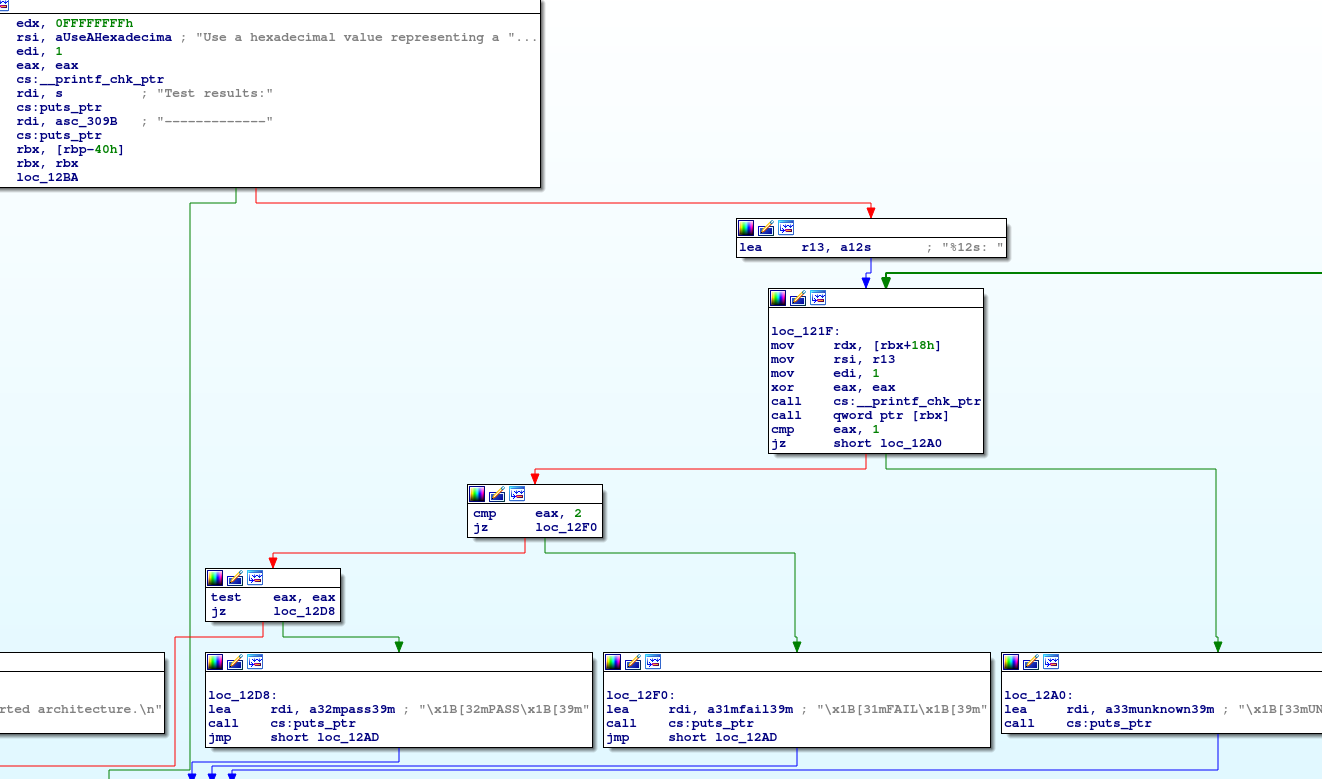


The goal is to disable all the checks. We can start disassembling it with IDA.

Using the Graph View, we see the program prints and registers the available set of tests:



One might think to remove the calls from here, but then we won’t see the test results at all. Instead, we want to see PASS for every test.

Moving forward, we can see the program prints “Test results” and then there is a cycle that tests every function and prints UNKNOWN if the result is 0, FAIL if the result is 2, PASS if the result is 1, and signals a bug for different results. The tests call are done calling the pointer in [rbx], and the result is stored into eax. 

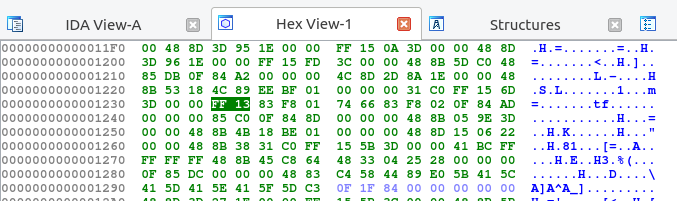
Ideally, we could go into every registered function and patch them to always return 0, but this is both time consuming and not necessary. Since we see that the tests are called in a single place and the PASS/FAIL checks are done also in the same place, it is enough to patch the program once, editing the return value of the test calling function. In other words, after the call ptr [rbx], eax must be 0. In this way we would pass every test. Unfortunately we don’t have space to set eax to 0 before the cmp eax, 1. Also, we cannot just remove the call ptr[rbx], since the previous call \_printf\_chk\_ptr sets eax to a value usually different to 0.

Instead, we can replace the call ptr [rbx] with an instruction that sets eax to 0.

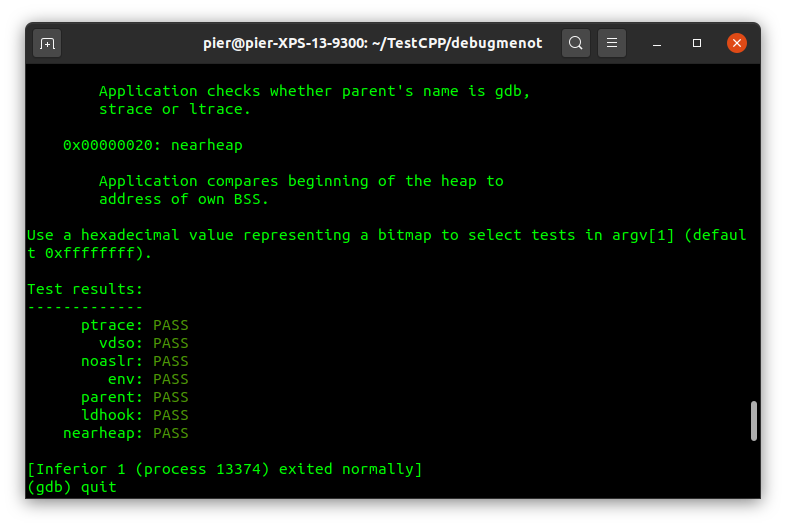
Looking at the hex view, the call instruction uses two bytes (FF 13). How can we set eax to 0 with two bytes? An easy way is to copy the above instruction xor eax,eax which sets eax to 0 with just two bytes (31 C0).

Let’s then find the call instruction into the binary and patch it! We can use an Hex Editor like bless to do so. On text view select the call instruction, and switch on HEX view to locate the instruction. Here, we can just copy some bytes around the instruction, for instance 3D 00 00 FF 13 83 F8 01, and we search for them on the hex editor. When we find them, and we are sure they are the only ones matching, we can replace FF 13 with 31 C0 (the xor eax,eax instruction), and save.

Be sure that you replace them and not add them, otherwise the program could crash!!!



We can run the program in gdb again, and test the results!



Bye bye anti-debug tests :)