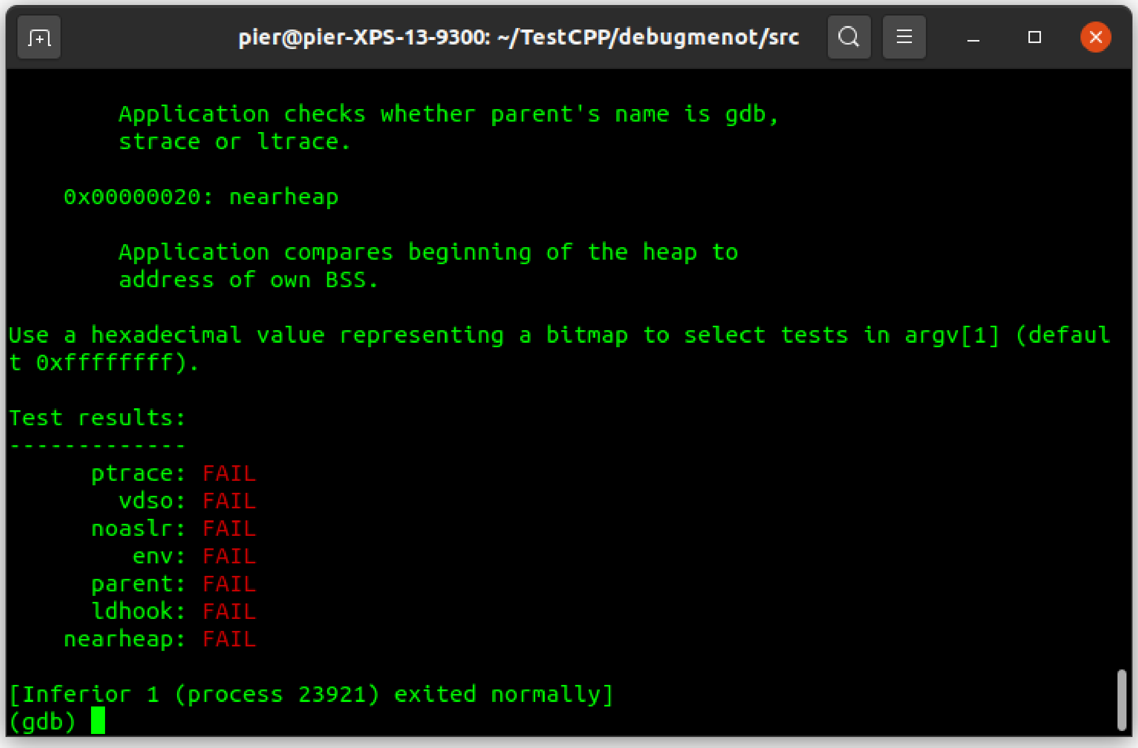
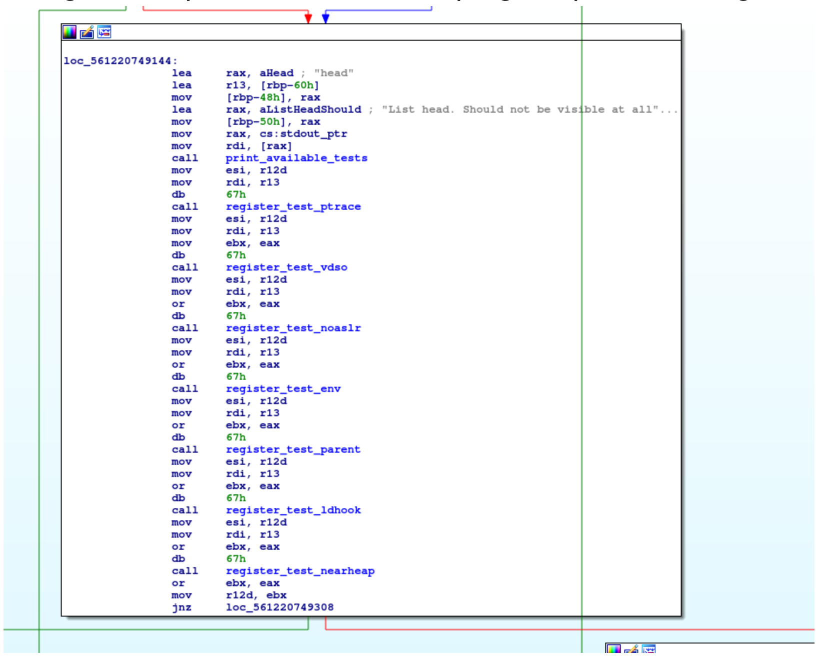
Debugmenot contains a set of tests to detect if gdb is running.
If we run the program inside of gdb, we see that all the tests fail



```
pier@pier-XPS-13-9300: ~/TestCPP/debugmenot/src

        Application checks whether parent's name is gdb,
        strace or ltrace.

    0x00000020: nearheap

        Application compares beginning of the heap to
        address of own BSS.

Use a hexadecimal value representing a bitmap to select tests in argv[1] (defaul
t 0xffffffff).

Test results:
-------------
      ptrace: FAIL
        vdso: FAIL
      noaslr: FAIL
         env: FAIL
      parent: FAIL
      ldhook: FAIL
    nearheap: FAIL

[Inferior 1 (process 23921) exited normally]
(gdb) 
```

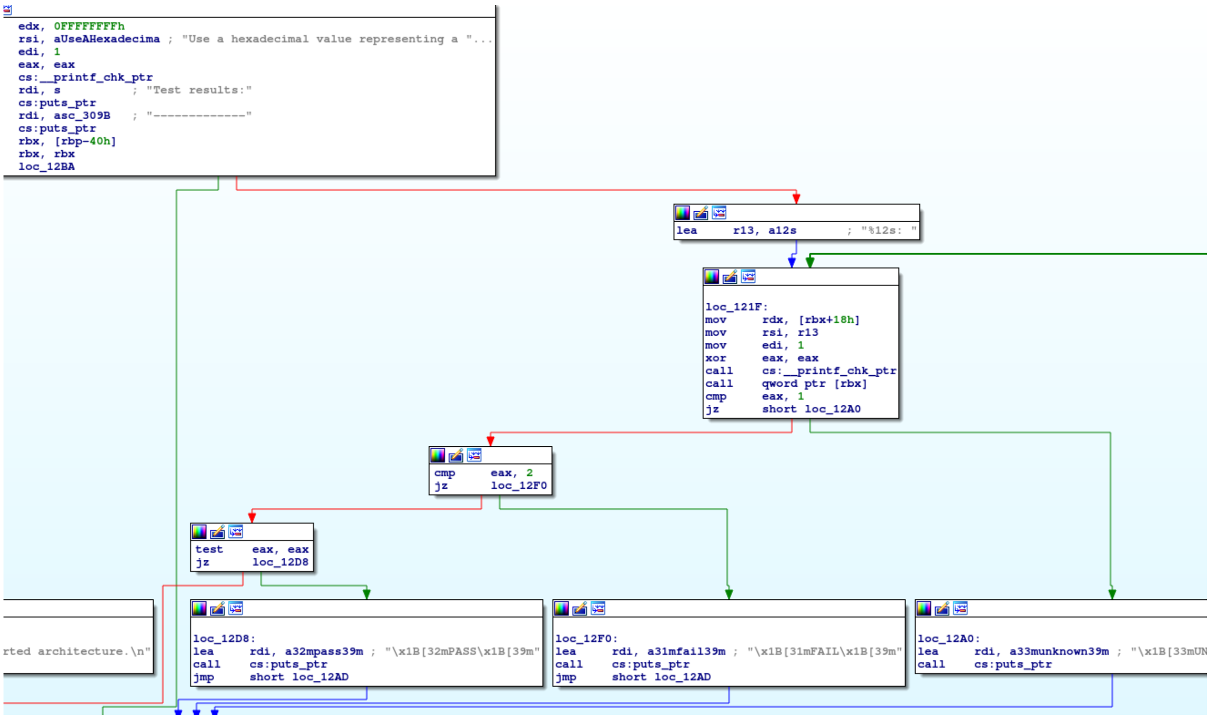
The goal is to disable all the checks. We can start disassembling it with IDA.
Using the Graph View, we see the program prints and registers the available set of tests:



```
loc_561220749144:
                lea     rax, aHead ; "head"
                lea     r13, [rbp-60h]
                mov     [rbp-48h], rax
                lea     rax, aListHeadShould ; "List head. Should not be visible at all"...
                mov     [rbp-50h], rax
                mov     rax, cs:stdout_ptr
                mov     rdi, [rax]
                call    print_available_tests
                mov     esi, r12d
                mov     rdi, r13
                db      67h
                call    register_test_ptrace
                mov     esi, r12d
                mov     rdi, r13
                mov     ebx, eax
                db      67h
                call    register_test_vdso
                mov     esi, r12d
                mov     rdi, r13
                or      ebx, eax
                db      67h
                call    register_test_noaslr
                mov     esi, r12d
                mov     rdi, r13
                or      ebx, eax
                db      67h
                call    register_test_env
                mov     esi, r12d
                mov     rdi, r13
                or      ebx, eax
                db      67h
                call    register_test_parent
                mov     esi, r12d
                mov     rdi, r13
                or      ebx, eax
                db      67h
                call    register_test_ldhook
                mov     esi, r12d
                mov     rdi, r13
                or      ebx, eax
                db      67h
                call    register_test_nearheap
                or      ebx, eax
                mov     r12d, ebx
                jnz     loc_561220749308
```

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.



```
edx, 0FFFFFFFFh
rsi, aUseAHexadecima ; "Use a hexadecimal value representing a "...
edi, 1
eax, eax
cs:__printf_chk_ptr
rdi, s          ; "Test results:"
cs:puts_ptr
rdi, asc_309B   ; "-------------"
cs:puts_ptr
rbx, [rbp-40h]
rbx, rbx
loc_12BA
```

```
lea     r13, a12s       ; "%12s: "
```

```
loc_121F:
mov     rdx, [rbx+18h]
mov     rsi, r13
mov     edi, 1
xor     eax, eax
call    cs:__printf_chk_ptr
call    qword ptr [rbx]
cmp     eax, 1
jz      short loc_12A0
```

```
cmp     eax, 2
jz      loc_12F0
```

```
test    eax, eax
jz      loc_12D8
```

```
rted architecture.\n"
```

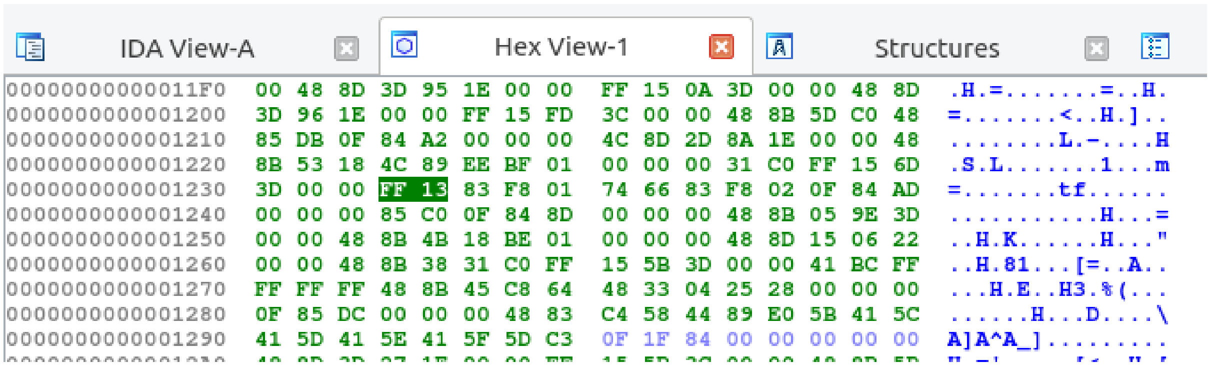
```
loc_12D8:
lea     rdi, a32mpass39m ; "\x1B[32mPASS\x1B[39m"
call    cs:puts_ptr
jmp     short loc_12AD
```

```
loc_12F0:
lea     rdi, a31mfail39m ; "\x1B[31mFAIL\x1B[39m"
call    cs:puts_ptr
jmp     short loc_12AD
```

```
loc_12A0:
lea     rdi, a33munknown39m ; "\x1B[33mUN
call    cs:puts_ptr
```

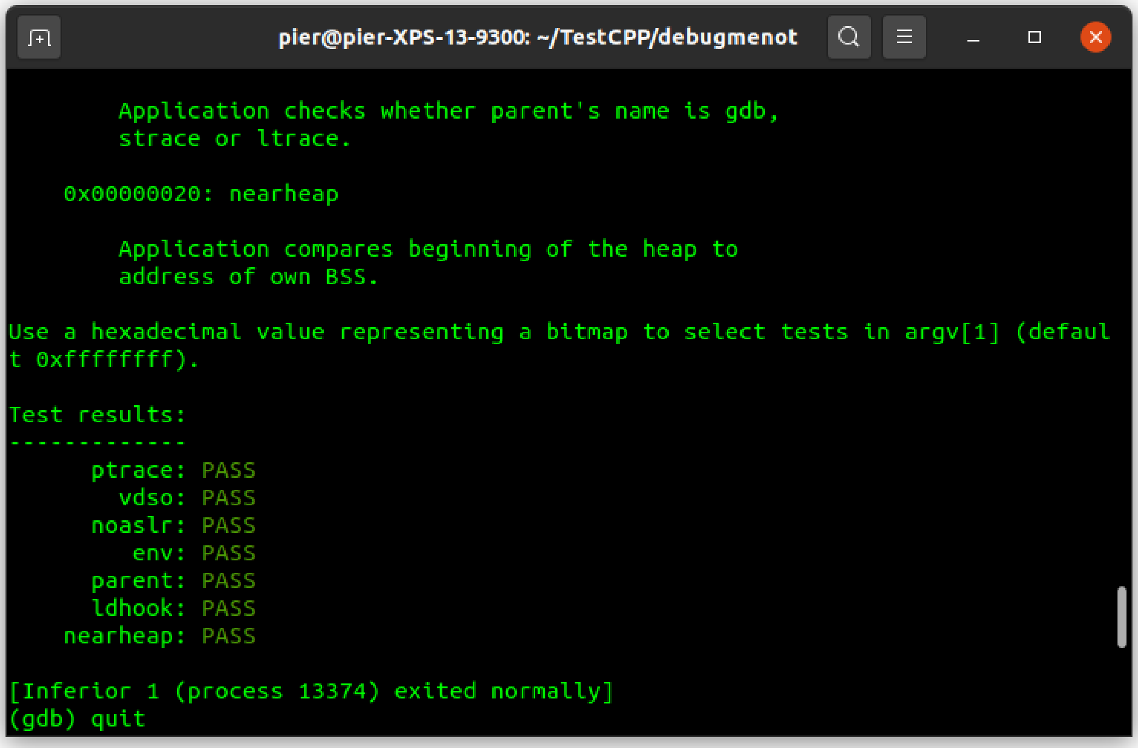
Ideally, we could go into every registered function and patch them to always return 0, but this is both time consuming and not necessary. Since we see that the tests are called in a single place and the PASS/FAIL checks are done also in the same place, it is enough to patch the program once, editing 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!!!



```
00000000000011F0  00 48 8D 3D 95 1E 00 00  FF 15 0A 3D 00 00 48 8D  .H.=.......=..H.
0000000000001200  3D 96 1E 00 00 FF 15 FD  3C 00 00 48 8B 5D C0 48  =.......<..H.]..
0000000000001210  85 DB 0F 84 A2 00 00 00  4C 8D 2D 8A 1E 00 00 48  ........L.-....H
0000000000001220  8B 53 18 4C 89 EE BF 01  00 00 00 31 C0 FF 15 6D  .S.L.......1...m
0000000000001230  3D 00 00 FF 13 83 F8 01  74 66 83 F8 02 0F 84 AD  =.......tf......
0000000000001240  00 00 00 85 C0 0F 84 8D  00 00 00 48 8B 05 9E 3D  ...........H...=
0000000000001250  00 00 48 8B 4B 18 BE 01  00 00 00 48 8D 15 06 22  ..H.K......H..."
0000000000001260  00 00 48 8B 38 31 C0 FF  15 5B 3D 00 00 41 BC FF  ..H.81...[=..A..
0000000000001270  FF FF FF 48 8B 45 C8 64  48 33 04 25 28 00 00 00  ...H.E..H3.%(...
0000000000001280  0F 85 DC 00 00 00 48 83  C4 58 44 89 E0 5B 41 5C  ......H...D....\
0000000000001290  41 5D 41 5E 41 5F 5D C3  0F 1F 84 00 00 00 00 00  A]A^A_].........
```

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



```
        Application checks whether parent's name is gdb,
        strace or ltrace.

    0x00000020: nearheap

        Application compares beginning of the heap to
        address of own BSS.

Use a hexadecimal value representing a bitmap to select tests in argv[1] (defaul
t 0xffffffff).

Test results:
-------------
      ptrace: PASS
        vdso: PASS
      noaslr: PASS
         env: PASS
      parent: PASS
      ldhook: PASS
    nearheap: PASS

[Inferior 1 (process 13374) exited normally]
(gdb) quit
```

Bye bye anti-debug tests :)