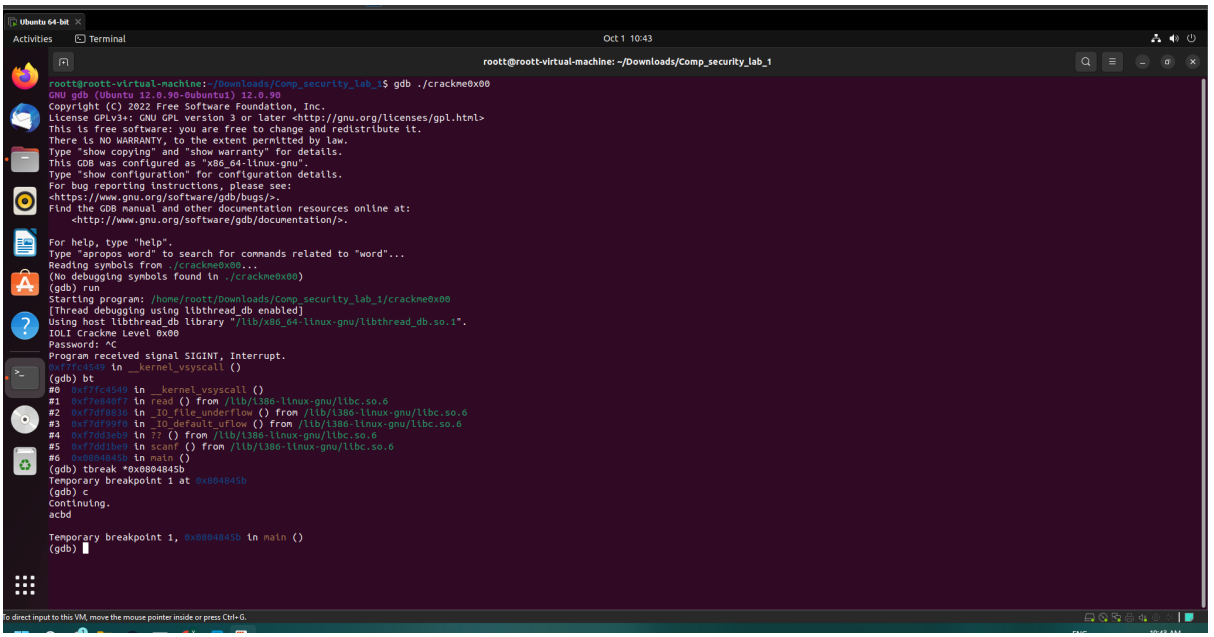
Initial Setup Steps for Ubuntu

1. Make sure the binary is executable / runnable
    a. To make a binary executable, Change permission to make the binary be executable either using GUI **(Right Click on the file → Properties → Permissions → check box for execute) or using the command line (chmod +x <path-to-file>)**
2. Make sure the correct compiler / Interpreter / Debugger is present.
    a. The binaries are in 32-bit arch and we have a 64-bit OS. So to make sure that the binaries can be executed, install the correct debugger and compiler using **(sudo apt-get install libc6:i386)**
3. Make sure the binary is run using **(./<path to binary> in terminal)**
4. Set Disassembly flavor / style
    a. We will be using att for our example but you can set any flavor using **set disassembly-flavor intel/att**

Binary 1 - Crackme0x00 - Password: 250382

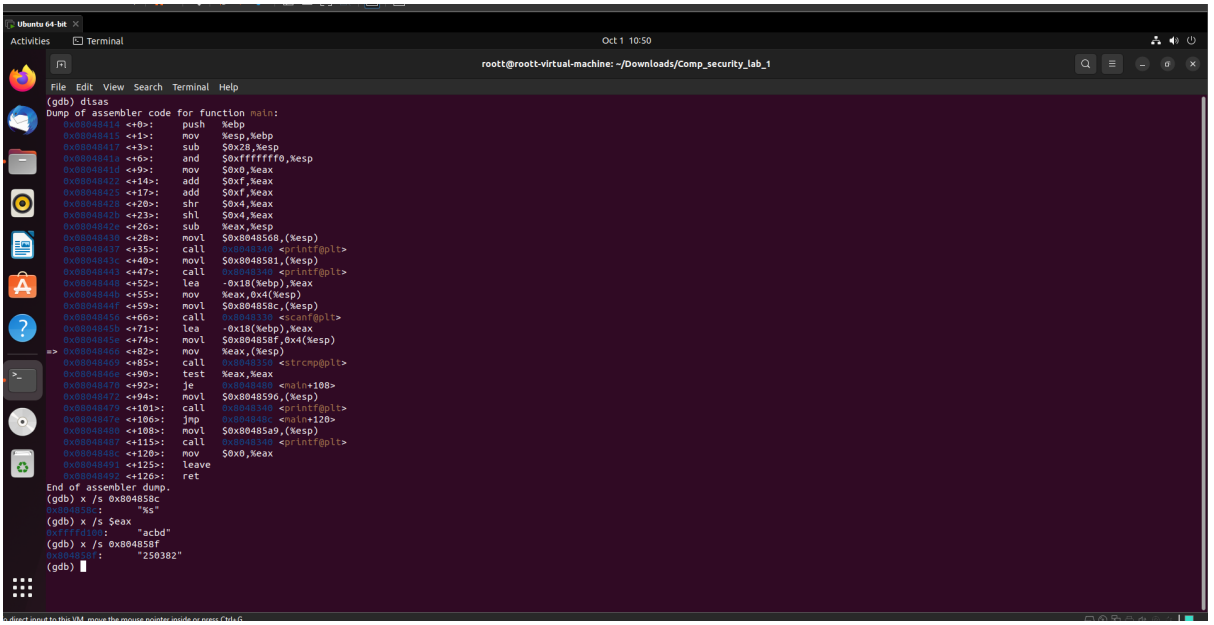
Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **acbd and press enter**
8. The screen should like like following



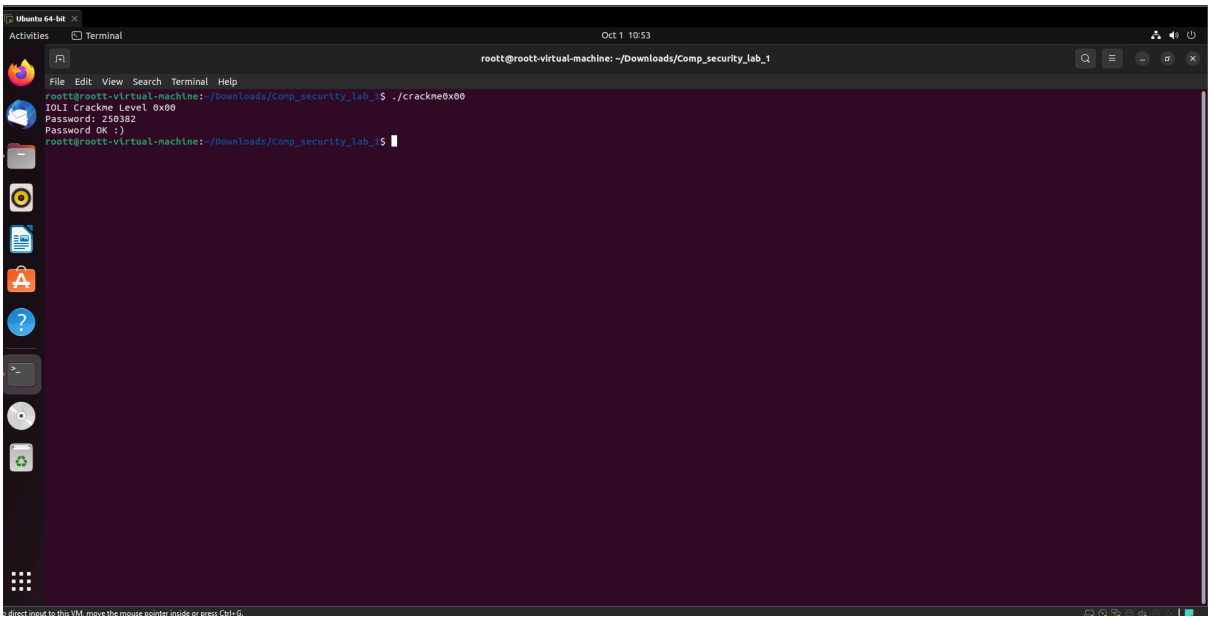
9. Disassemble the binary using **disas**

10. Move to next instruction using **nexti** such that we are just before the comparison line
11. Find the input format for **scanf**
12. Find the comparison line **strcmp**
13. Find the values being compared by looking into the individual values
14. The output should look like the following screen



15. As the value **250382** is being compared to our input value **acbd**, this means that the **password should be 250382**
16. Close the debugger using **quit** and launch the binary normally.
17. Enter the password.
18. Validate the result.

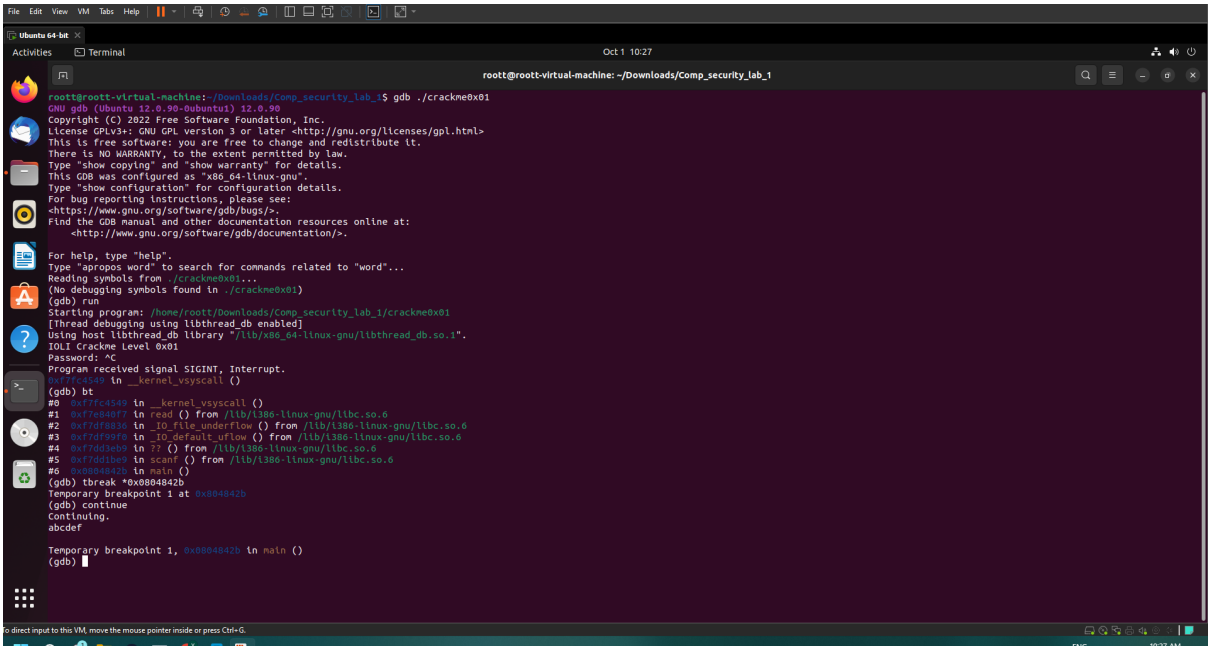
Output



Binary 2 - Crackme0x01 - Password: 5274

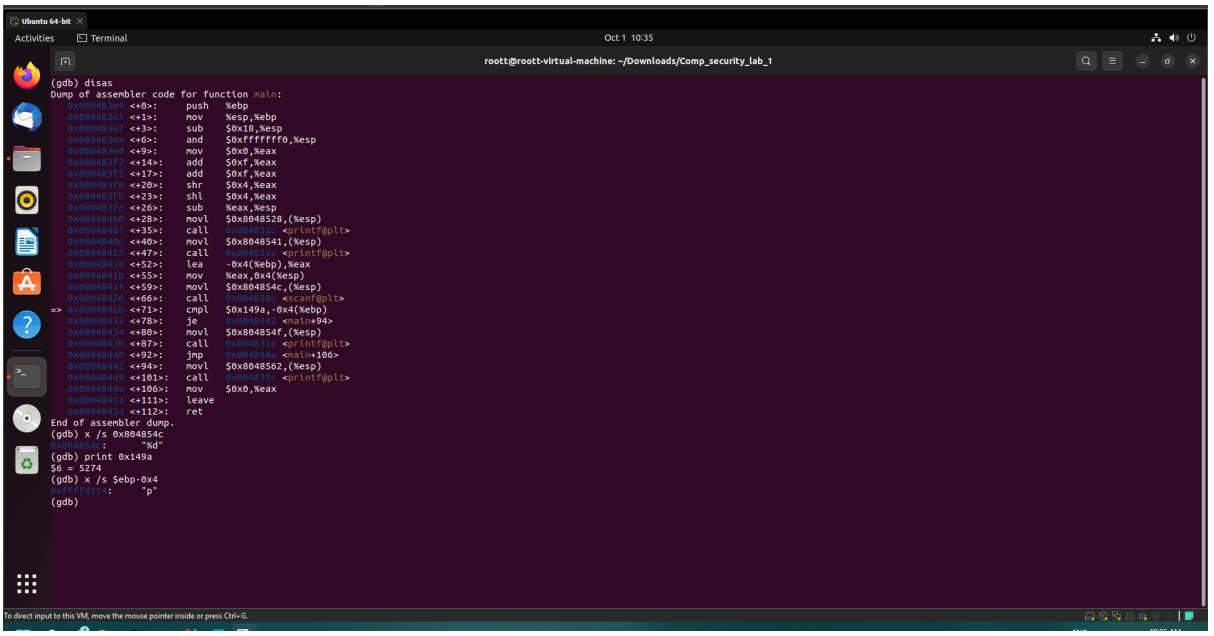
Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **abcdef and press enter**
8. The screen should like like following



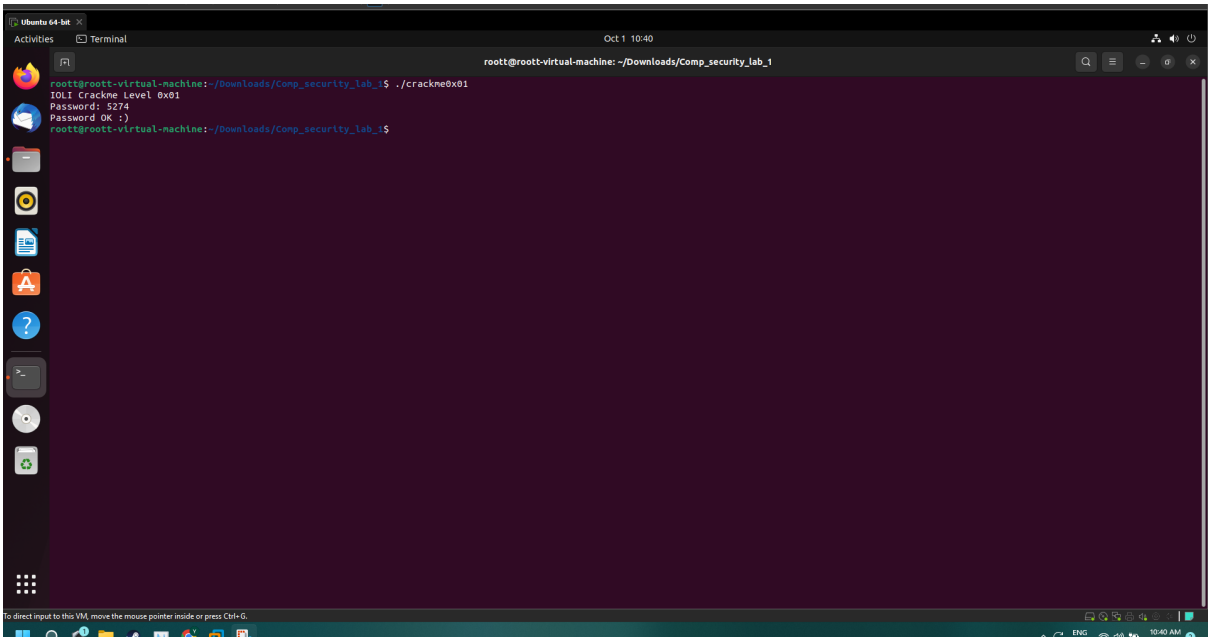
9. Disassemble the binary using **disas**
10. Find the input format for **scanf**
11. Find the comparison line **cmpl**
12. Find the values being compared by looking into the individual values

13. The output should look like the following screen



14. As the literal **value of 0x149a in hex or 5274 in decimal** is being compared to a variable, this means that the **password should be 5274**.
15. Close the debugger using **quit** and launch the binary normally.
16. Enter the password.
17. Validate the result.

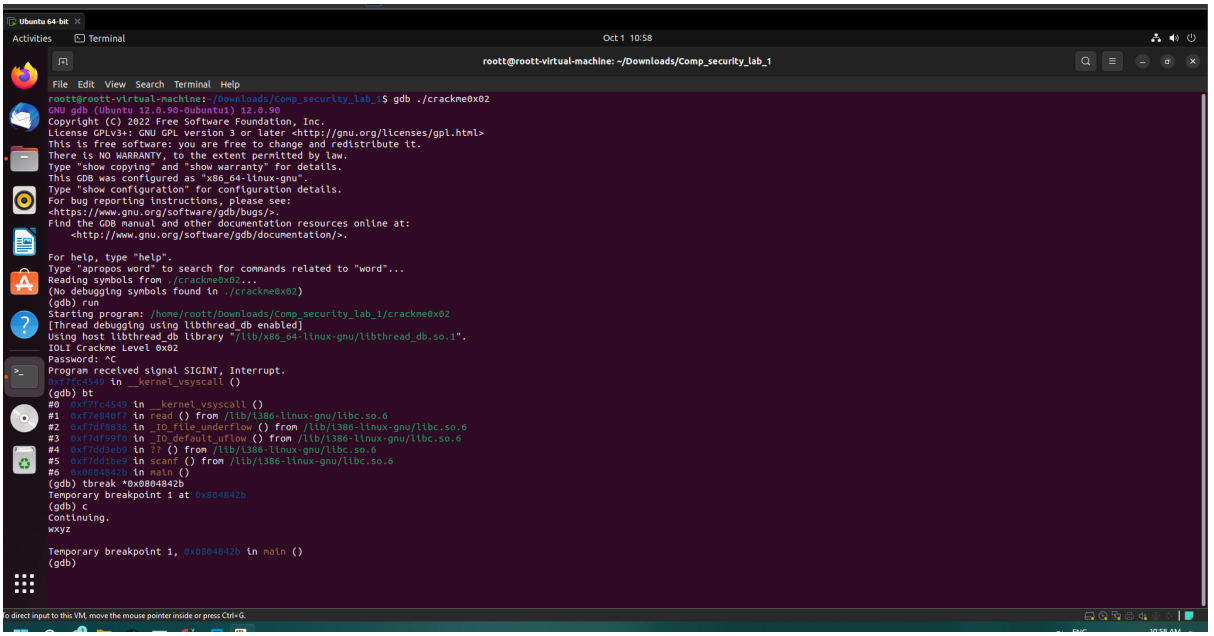
Output



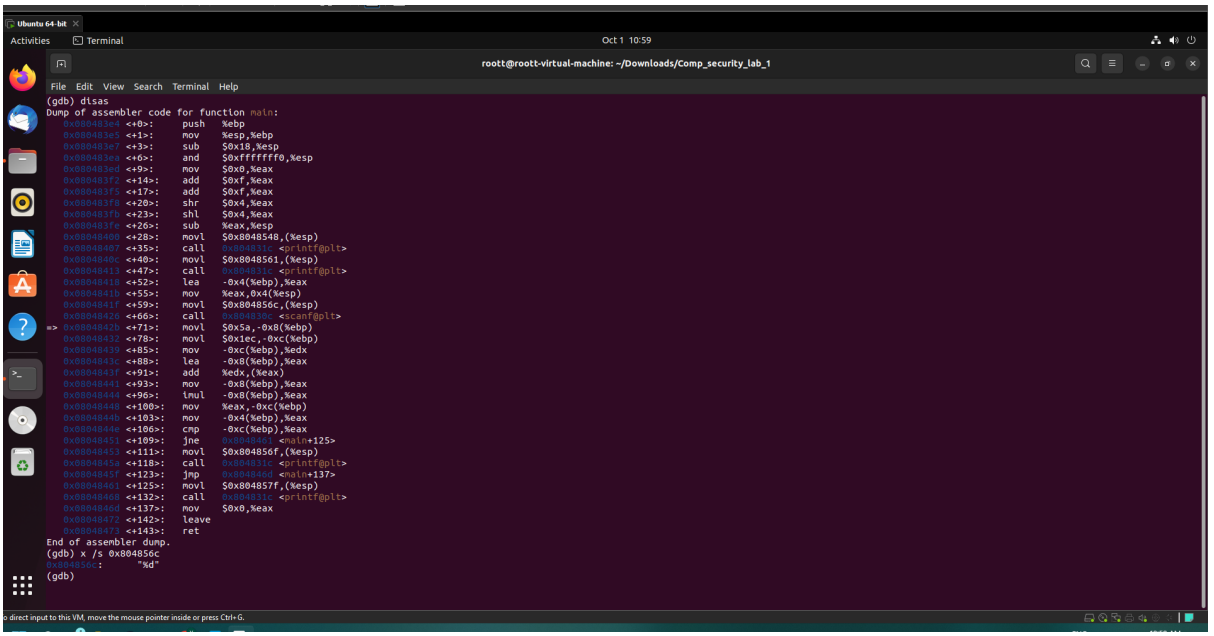
Binary 3 - Crackme0x02 - Password: 338724

Steps

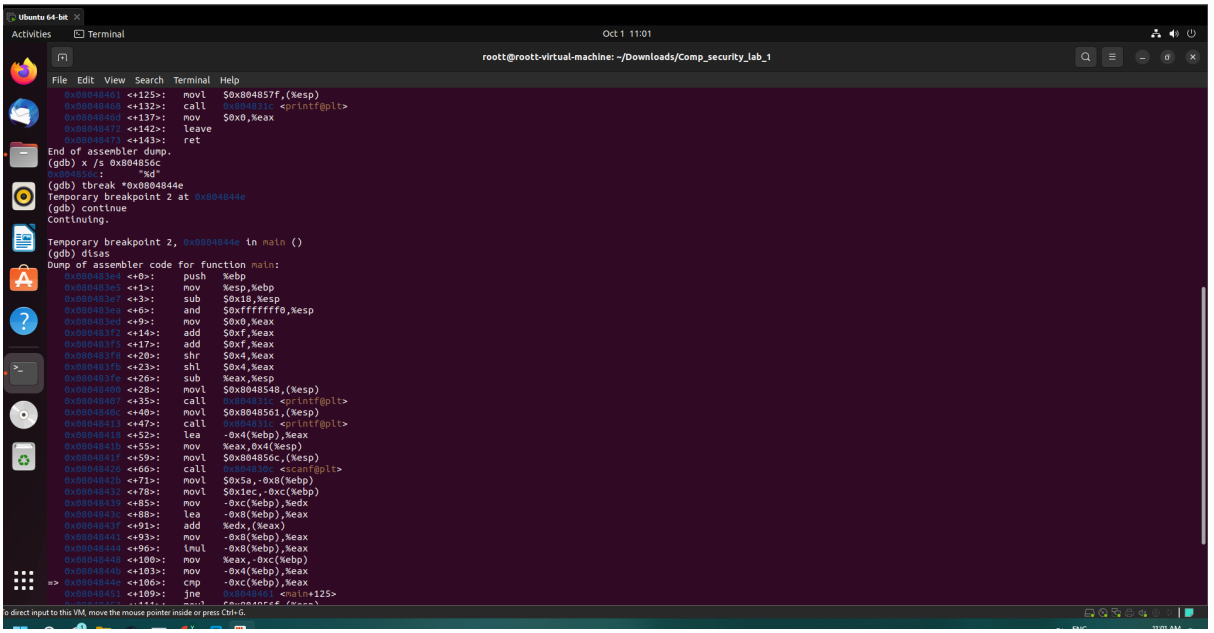
1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **998 and press enter**
8. The screen should like like following



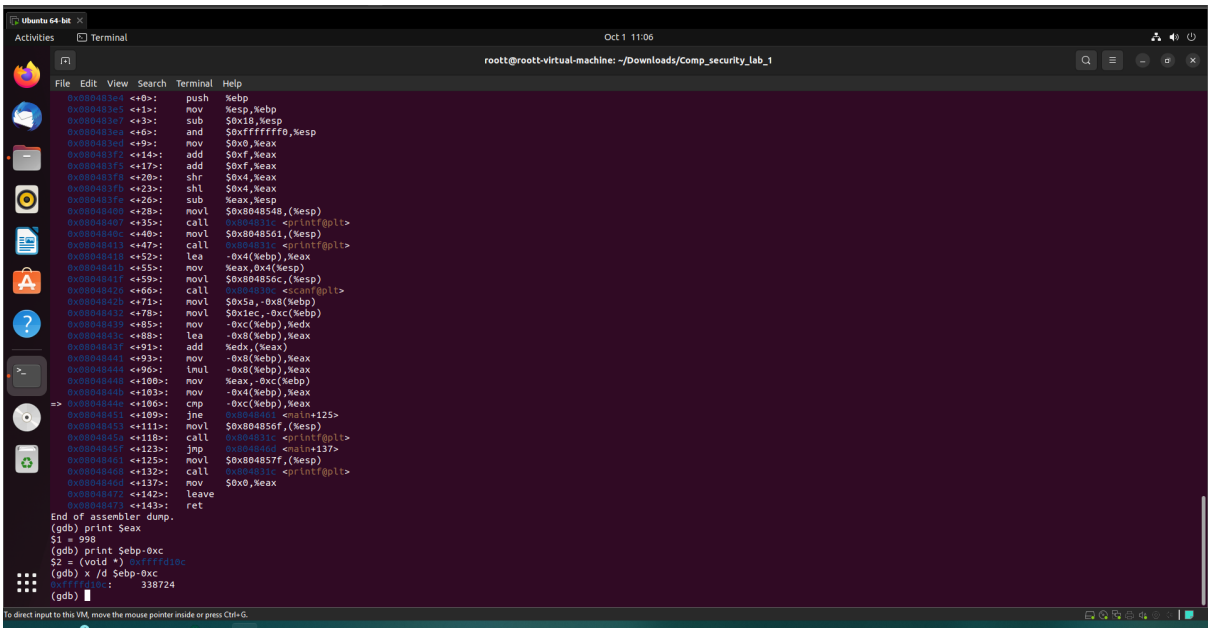
9. Disassemble the binary using **disas**
10. Find the input format for **scanf**



11. Add a breakpoint on the comparison line using **tbreak *<memory address of line of comparison>**
12. Continue code execution using **continue**
13. Disassemble the binary using **disas**

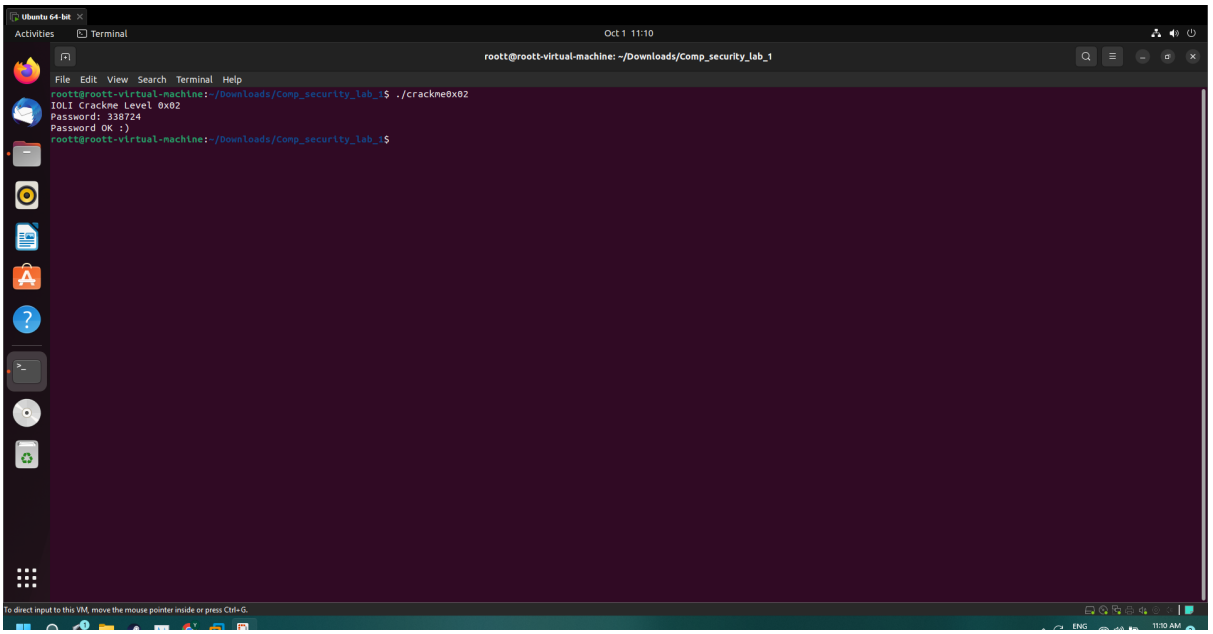


14. Find the comparison line **cmp**
15. Find the values being compared by looking into the individual values
16. The output should look like the following screen



17. As the **value of register eax = 998** is being compared to the **value of $ebp-0xc = 338724, this means that 338724** is our password.
18. Close the debugger using **quit** and launch the binary normally.
19. Enter the password.
20. Validate the result.

Output

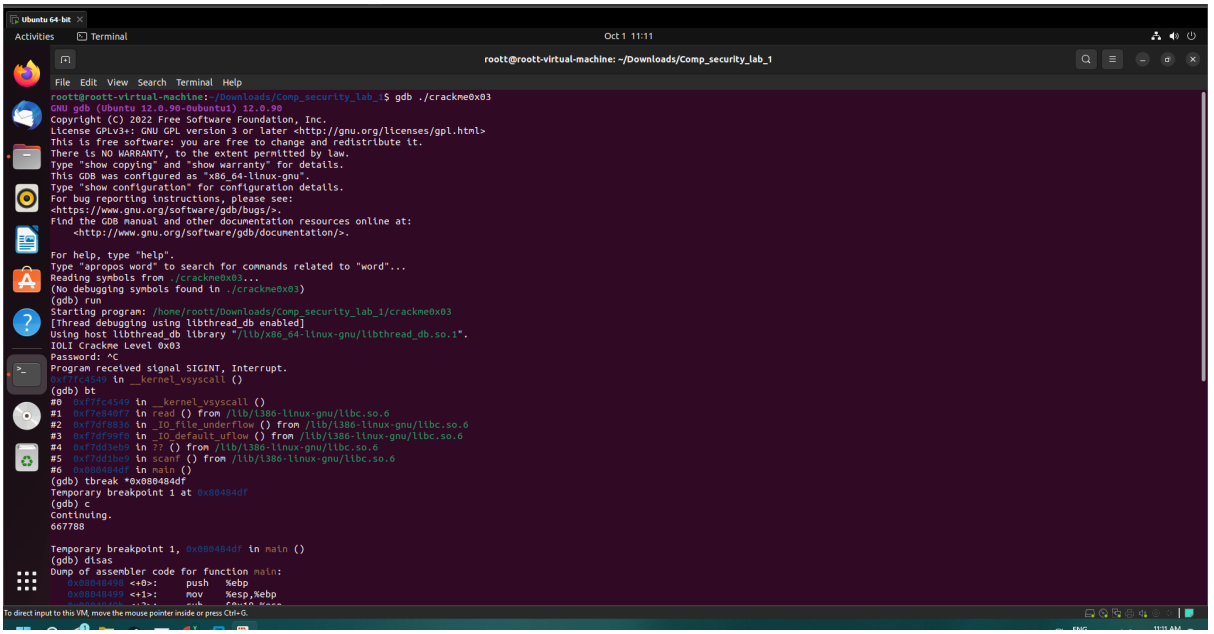


Binary 4 - Crackme0x03 - Password: 338724

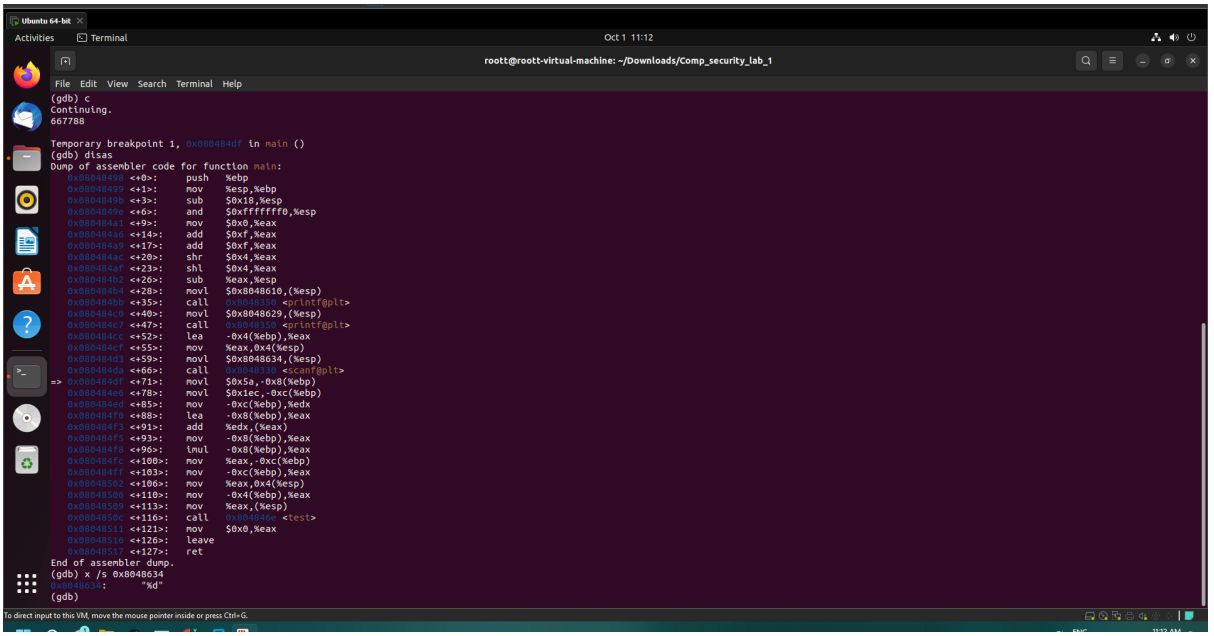
Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **667788 and press enter**

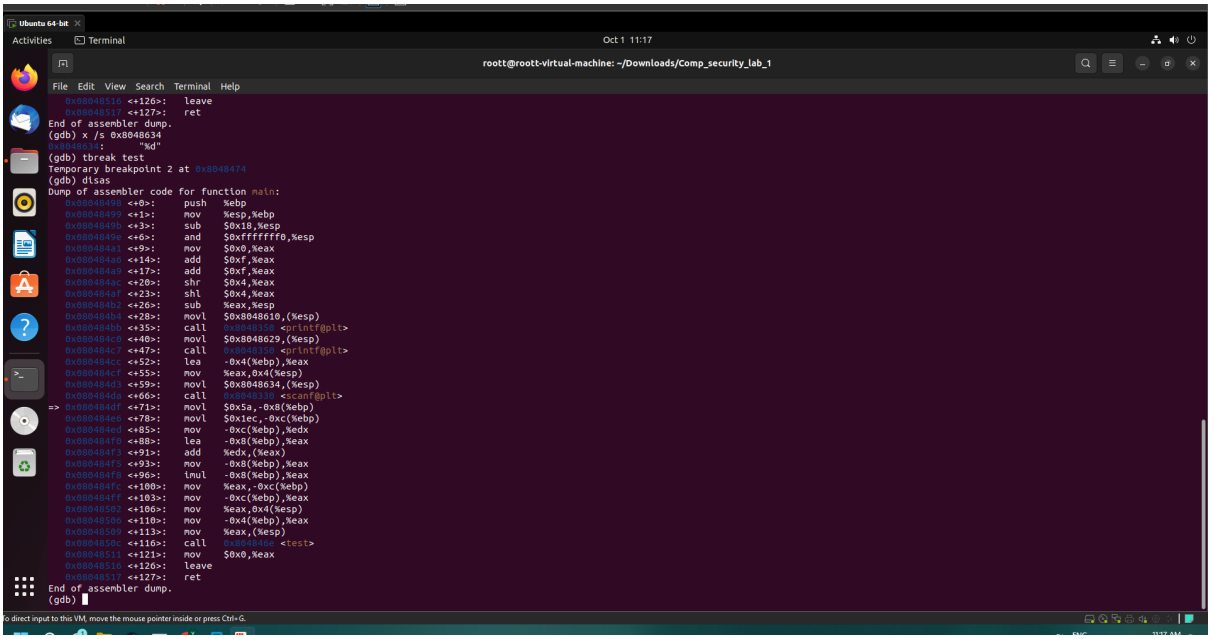
8. The screen should like like following

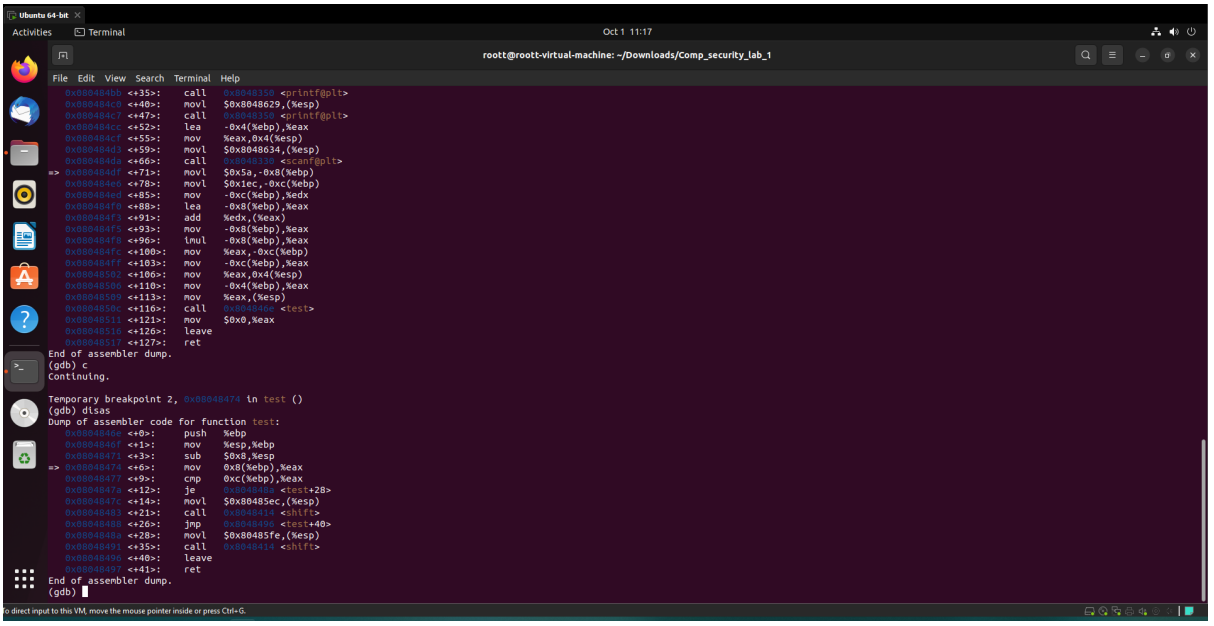


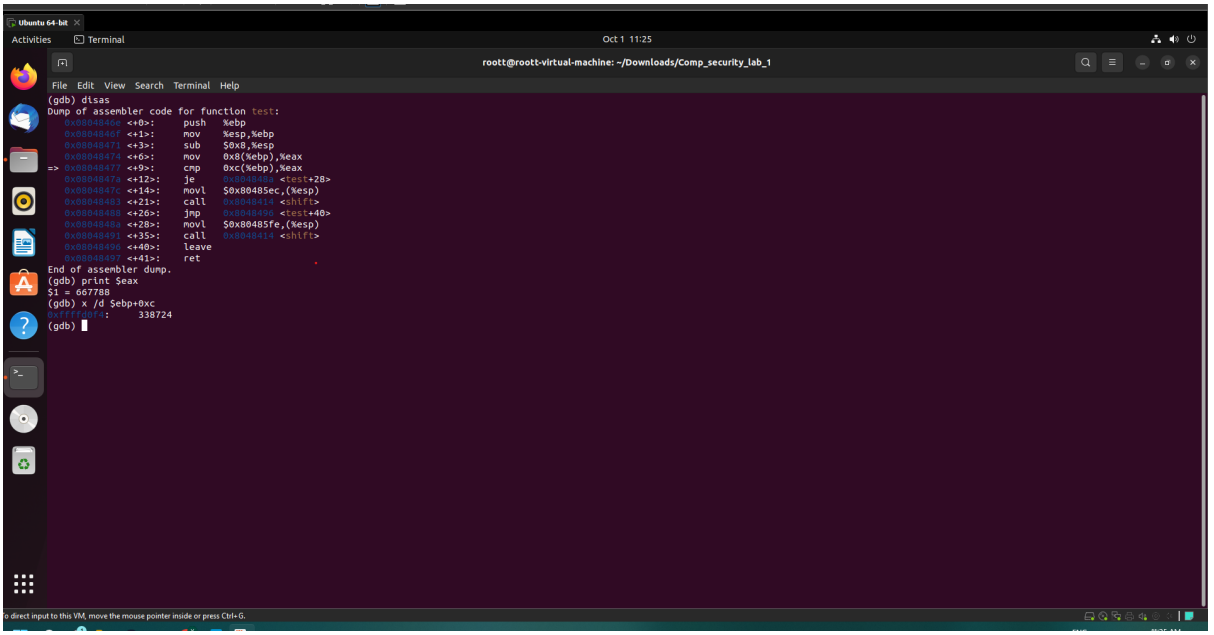
9. Disassemble the binary using **disas**
10. Find the input format for **scanf**



11. The comparison is not in this function but there is a function call made for the function **test**. We will be adding a debugger for the function test using **tbreak test** and then continue execution using **continue** and then disassemble the code again using **disas**.

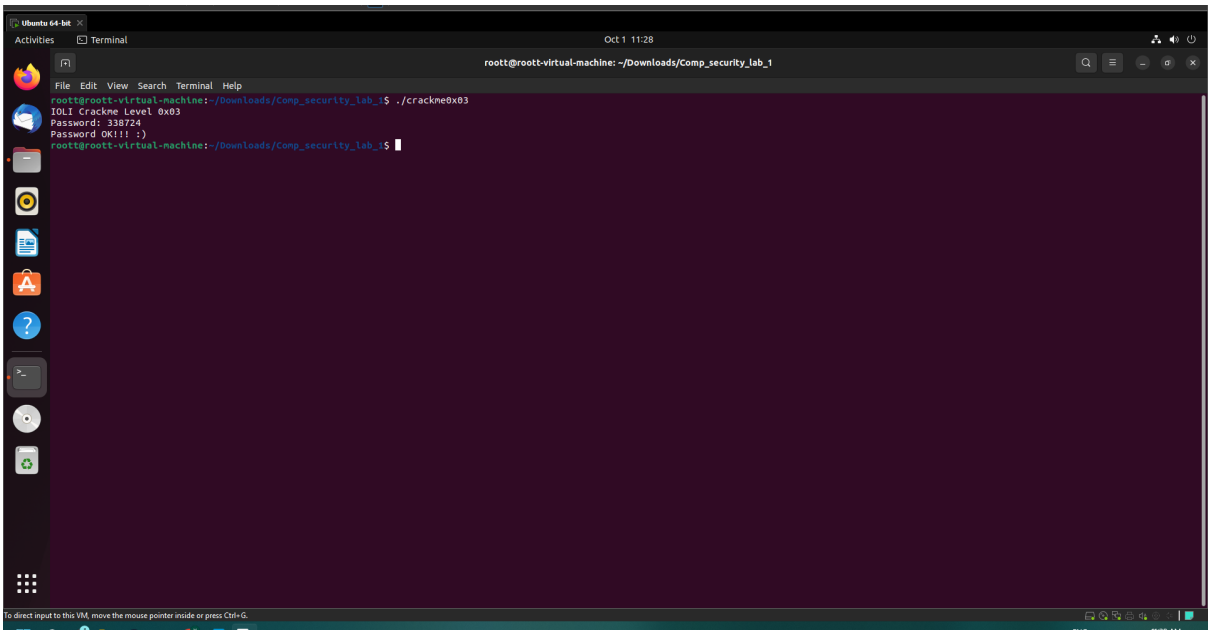

Ubuntu 64-bit
Activities
Terminal
Oct 1 11:17
root@root-virtual-machine: ~/Downloads/Comp_security_lab_1
File Edit View Search Terminal Help
0x08048512 <+126>: leave
0x08048517 <+127>: ret
End of assembler dump.
(gdb) x /s 0x8048634
0x8048634: "%d"
(gdb) tbreak test
Temporary breakpoint 2 at 0x8048474
(gdb) disas
Dump of assembler code for function main:
0x08048498 <+0>: push %ebp
0x08048499 <+1>: mov %esp,%ebp
0x0804849b <+3>: sub $0x18,%esp
0x0804849e <+6>: and $0xfffffff0,%esp
0x080484a1 <+9>: mov $0x0,%eax
0x080484a6 <+14>: add $0xf,%eax
0x080484a9 <+17>: add $0xf,%eax
0x080484ac <+20>: shr $0x4,%eax
0x080484af <+23>: shl $0x4,%eax
0x080484b2 <+26>: sub %eax,%esp
0x080484b4 <+28>: movl $0x8048610,(%esp)
0x080484bb <+35>: call 0x8048320 <printf@plt>
0x080484c0 <+40>: movl $0x8048629,(%esp)
0x080484c7 <+47>: call 0x8048320 <printf@plt>
0x080484cc <+52>: lea -0x4(%ebp),%eax
0x080484cf <+55>: mov %eax,0x4(%esp)
0x080484d3 <+59>: movl $0x8048634,(%esp)
0x080484da <+66>: call 0x8048340 <__isoc99_scanf@plt>
=> 0x080484df <+71>: movl $0x5a,-0x8(%ebp)
0x080484e6 <+78>: movl $0x1ec,-0xc(%ebp)
0x080484ed <+85>: mov -0xc(%ebp),%edx
0x080484f0 <+88>: lea -0x8(%ebp),%eax
0x080484f3 <+91>: add %edx,(%eax)
0x080484f5 <+93>: mov -0x8(%ebp),%eax
0x080484f8 <+96>: imul -0x8(%ebp),%eax
0x080484fc <+100>: mov %eax,-0xc(%ebp)
0x080484ff <+103>: mov -0xc(%ebp),%eax
0x08048502 <+106>: mov %eax,0x4(%esp)
0x08048506 <+110>: mov -0x4(%ebp),%eax
0x08048509 <+113>: mov %eax,(%esp)
0x0804850c <+116>: call 0x804846e <test>
0x08048511 <+121>: mov $0x0,%eax
0x08048516 <+126>: leave
0x08048517 <+127>: ret
End of assembler dump.
(gdb)
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.





12. As the **value of register eax = 667788** is being compared to the **value of $ebp+0xc = 338724, this means that 338724** is our password.
13. Close the debugger using **quit** and launch the binary normally.
14. Enter the password.
15. Validate the result.

Output

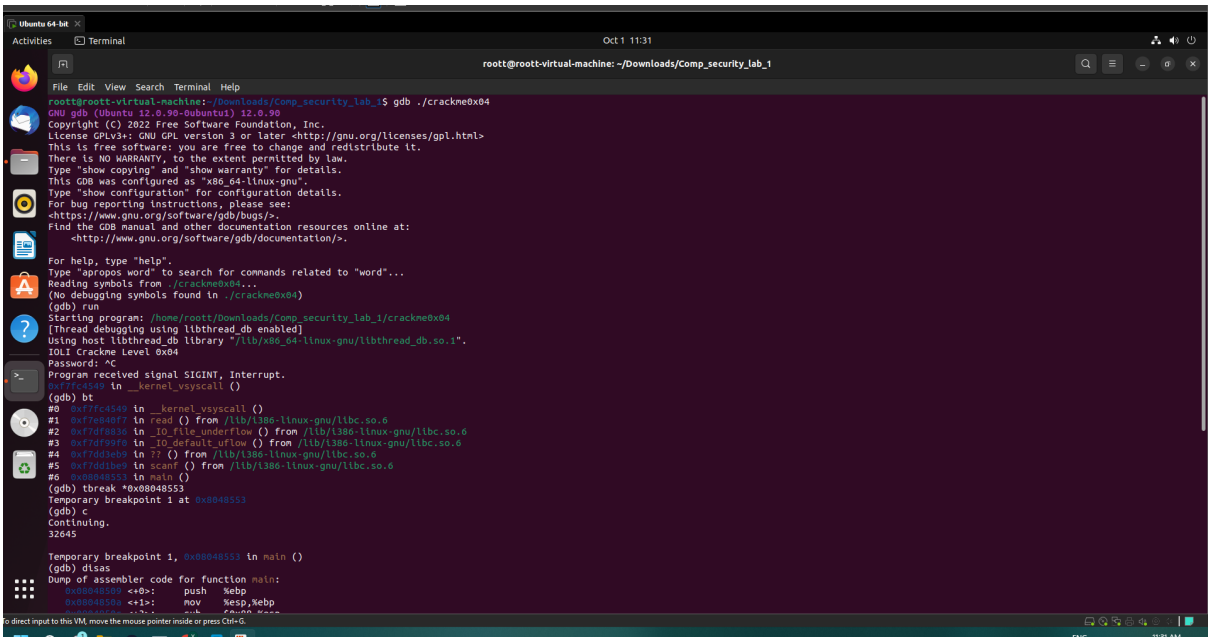


Binary 5 - Crackme0x04 - Password: Any number password with sum of digits = 15. For ex 12345 or 555 or 3336

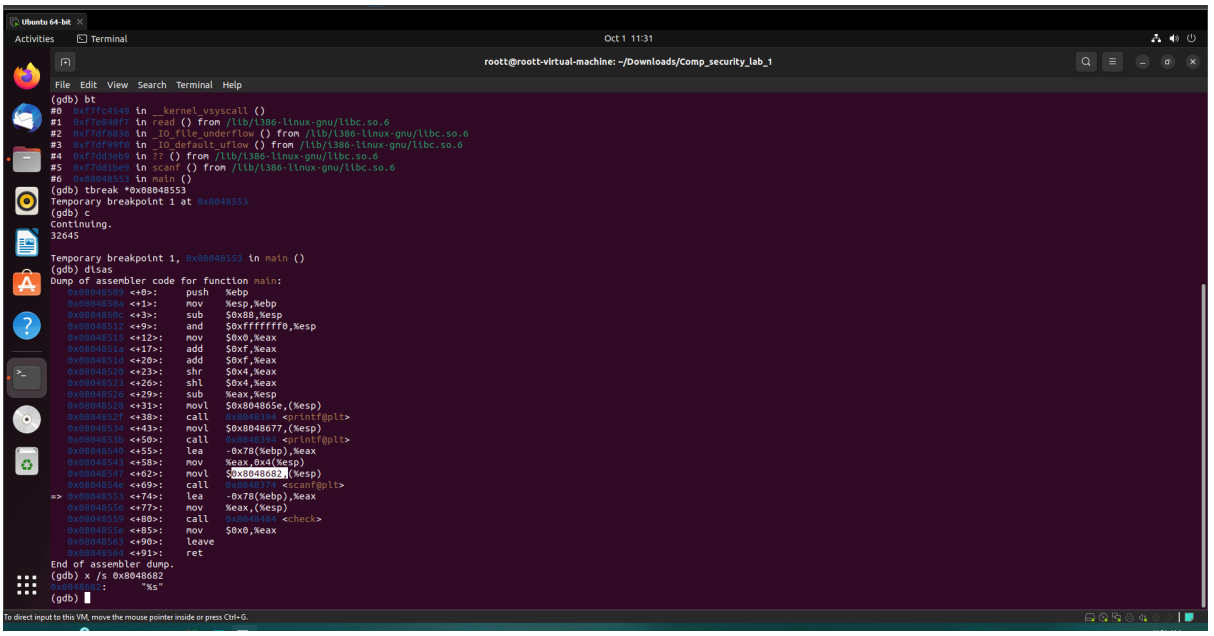
Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **32645 and press enter**

8. The screen should like like following

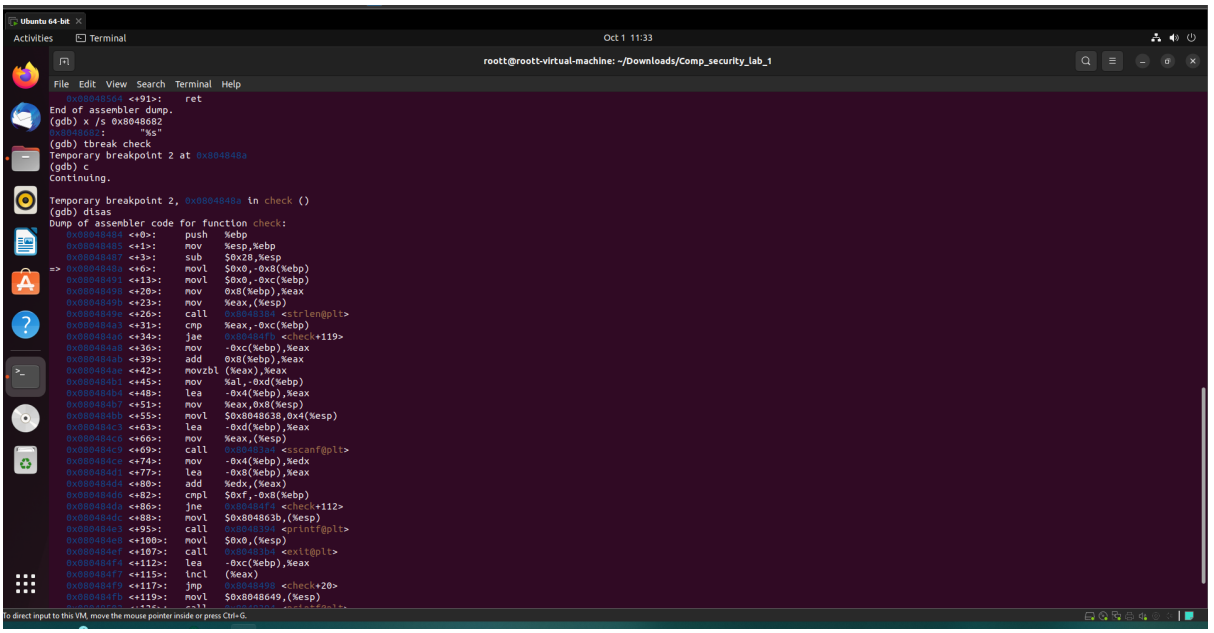


9. Disassemble the binary using **disas**
10. Find the input format for **scanf**



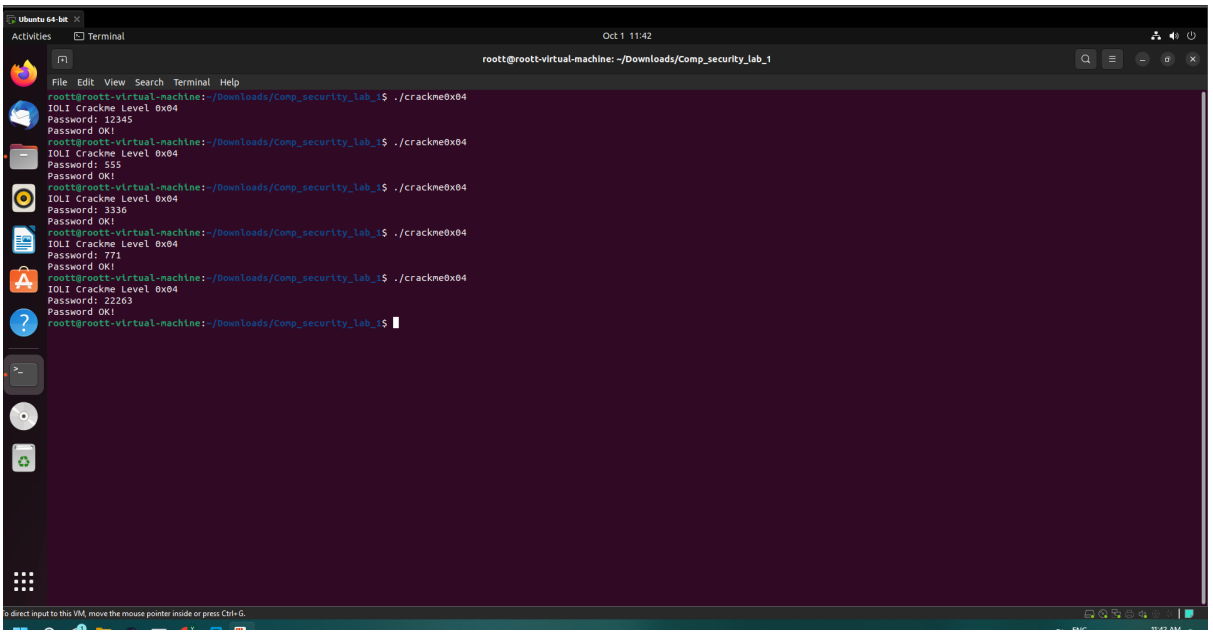
11. The comparison is not in this function but there is a function call made for the function **check**. We will be adding a debugger for the function test using **tbreak check** and then continue execution using **continue** and then disassemble the code

again using **disas**.



12. There is a comparison made on line +26 but we can ignore this as it is comparing the current string length with 0.
13. **As per the instructions, it seems like there is a loop being executed that is taking the string 1 character at a time, converting it to int using sscanf and then adding it to a variable and comparing it with 0xf hex or number 15 in decimal. This means the password is any string which has numbers that have a sum of 15.**
14. Close the debugger using **quit** and launch the binary normally.
15. Enter the password.
16. Validate the result.

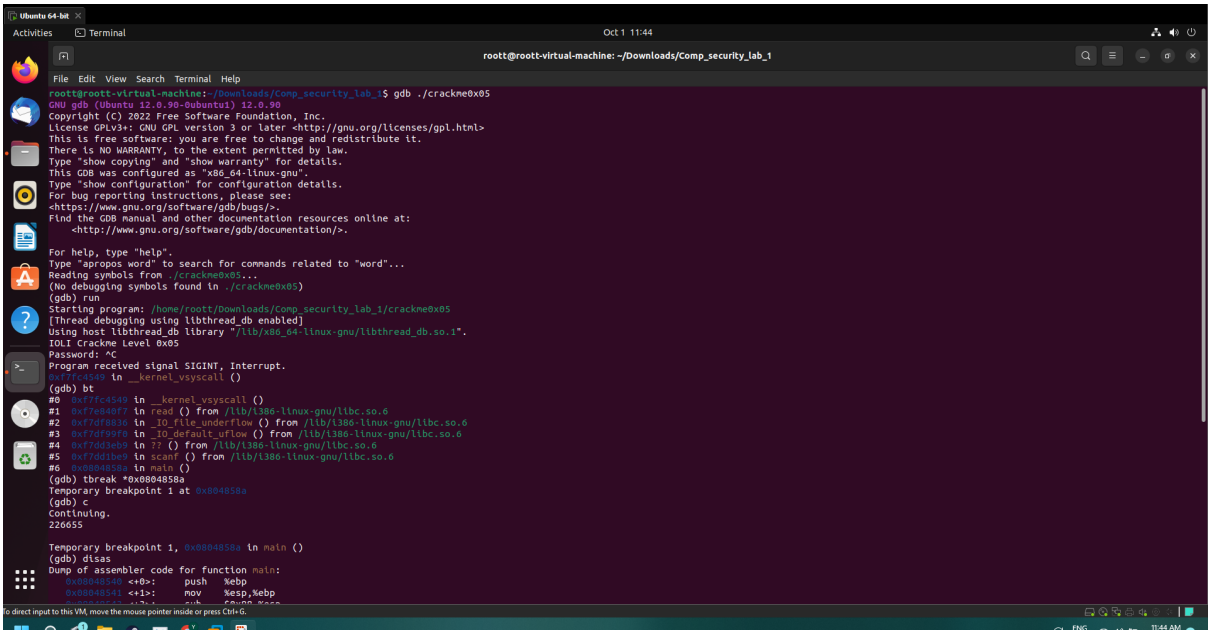
Output



Binary 6 - Crackme0x05 - Password: Any number password with sum of digits = 16 and last digit is even. For ex 88 or 772 or 33334

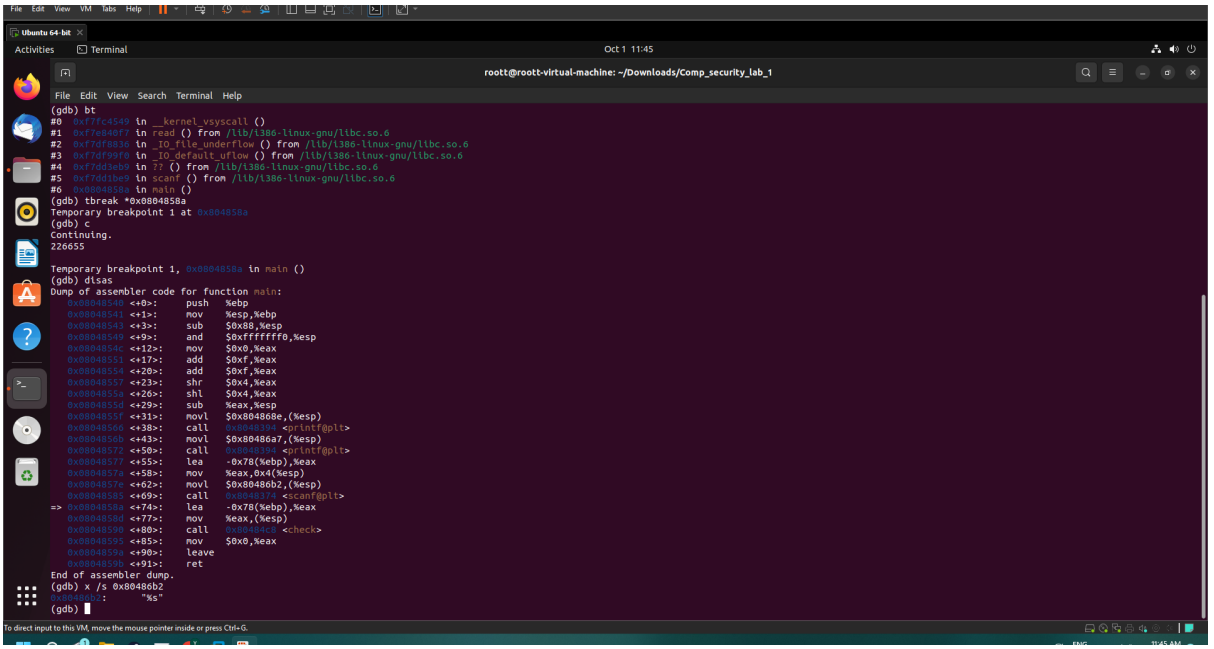
Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **226655 and press enter**
8. The screen should like like following

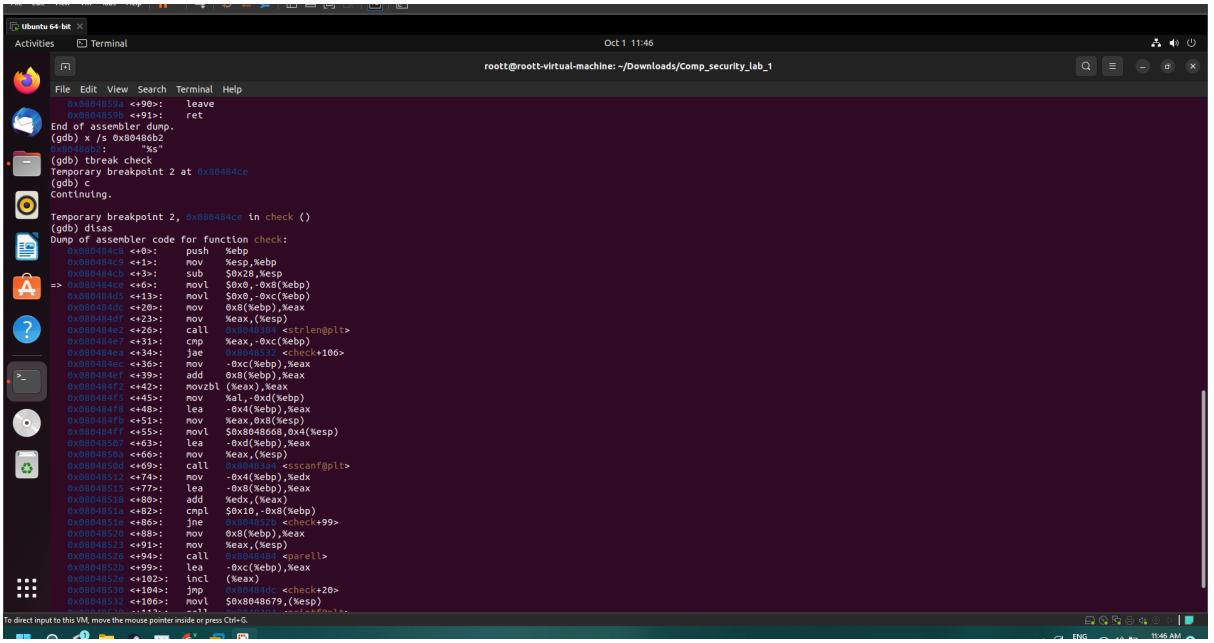


9. Disassemble the binary using **disas**

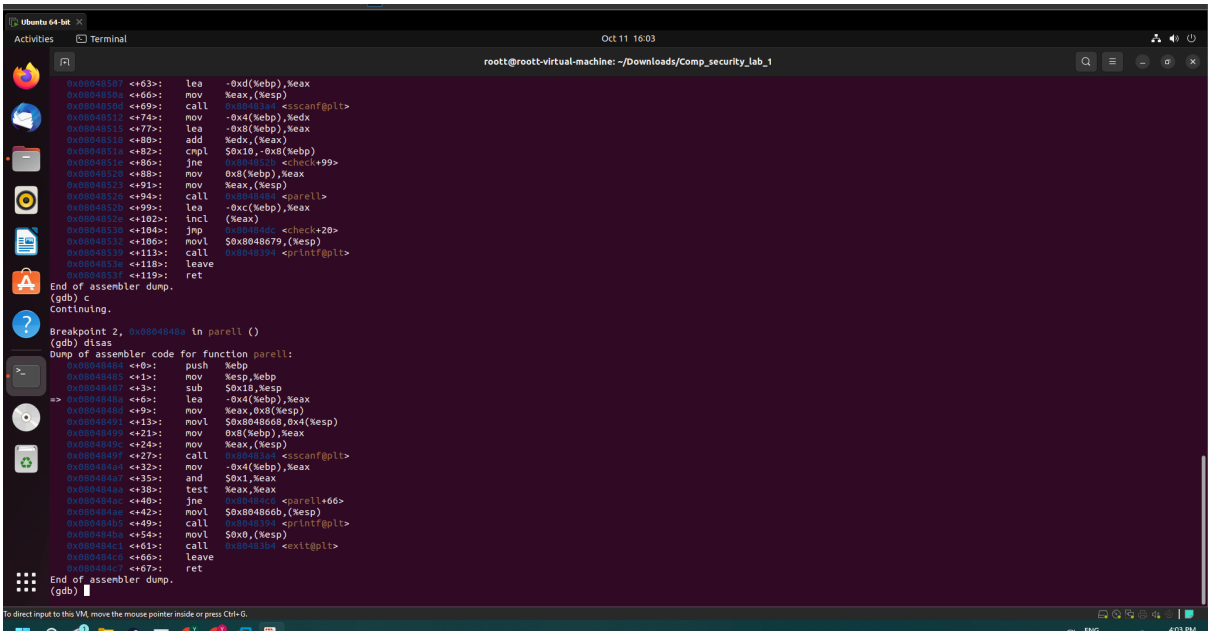
10. Find the input format for **scanf**



11. The comparison is not in this function but there is a function call made for the function **check**. We will be adding a debugger for the function test using **tbreak check** and then continue execution using **continue** and then disassemble the code again using **disas**.



12. There is a comparison made on line +26 but we can ignore this as it is comparing the current string length with 0.
13. **As per the instructions, it seems like there is a loop being executed that is taking the string 1 character at a time, converting it to int using sscanf and then adding it to a variable and comparing it with 0x10 hex or number 16 in decimal. This means the password is any string which has numbers that have a sum of 16.**
14. After this, the control of the code moves to function parell
15. Add a debugger on function parell and disasemble the code using disas

16. 

17. **On the line parell+35, there is an and operation with 1 which means that the last digit of the password must be even. So the final password should have a sum of digits as 16 and should be even so 772 works as a password but 277 does not.**
18. Close the debugger using **quit** and launch the binary normally.
19. Enter the password.
20. Validate the result.

Output