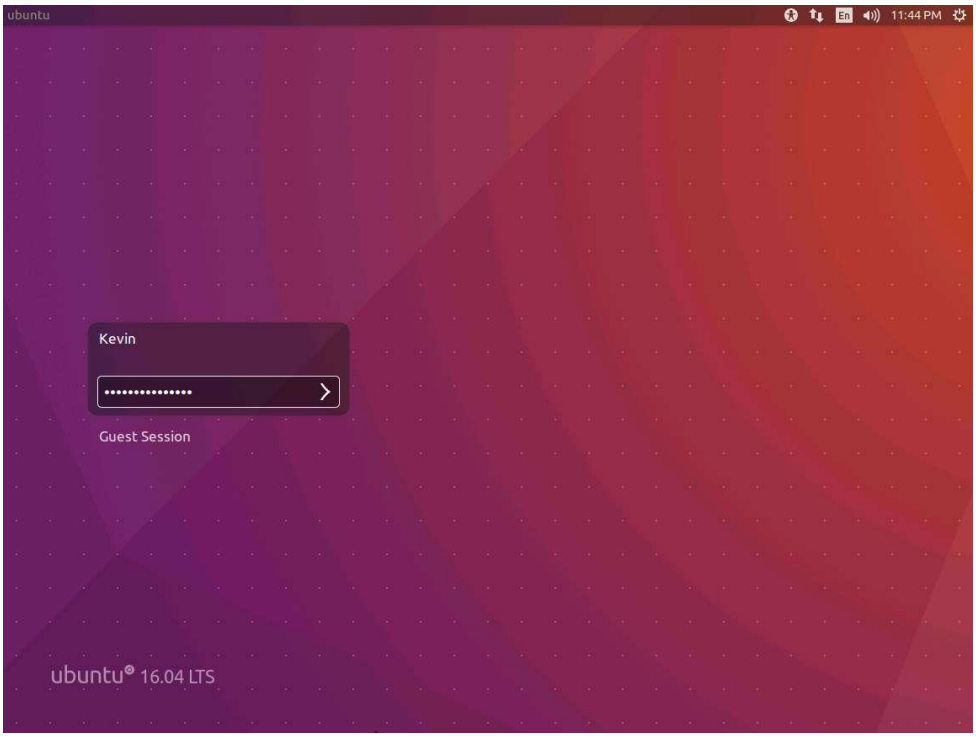
Exercise 3: Binary Analysis Methodology

Lab Objective:

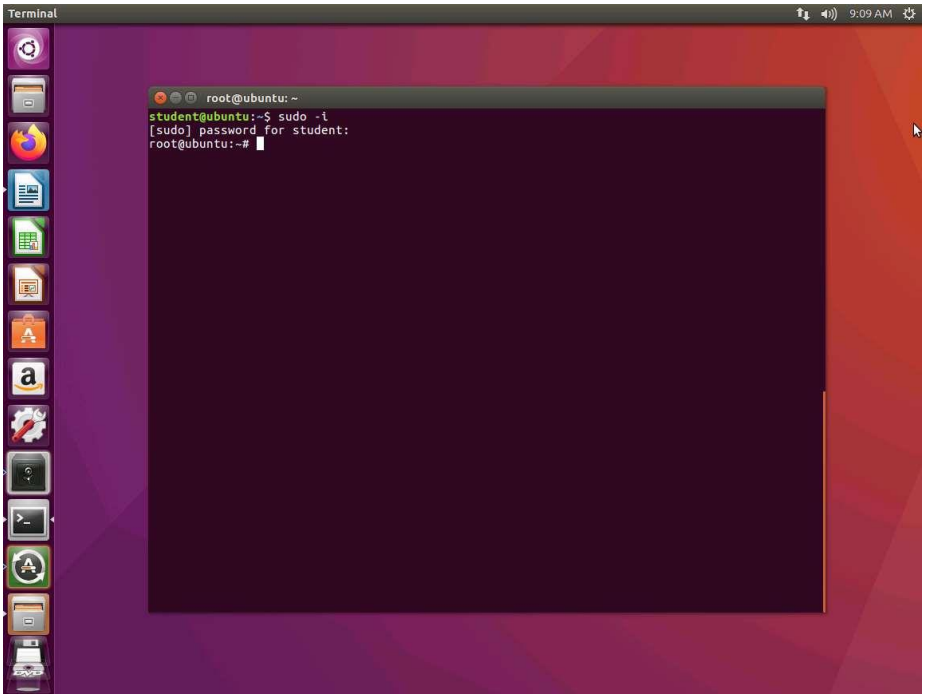
Learn binary analysis methodology.

Lab Tasks

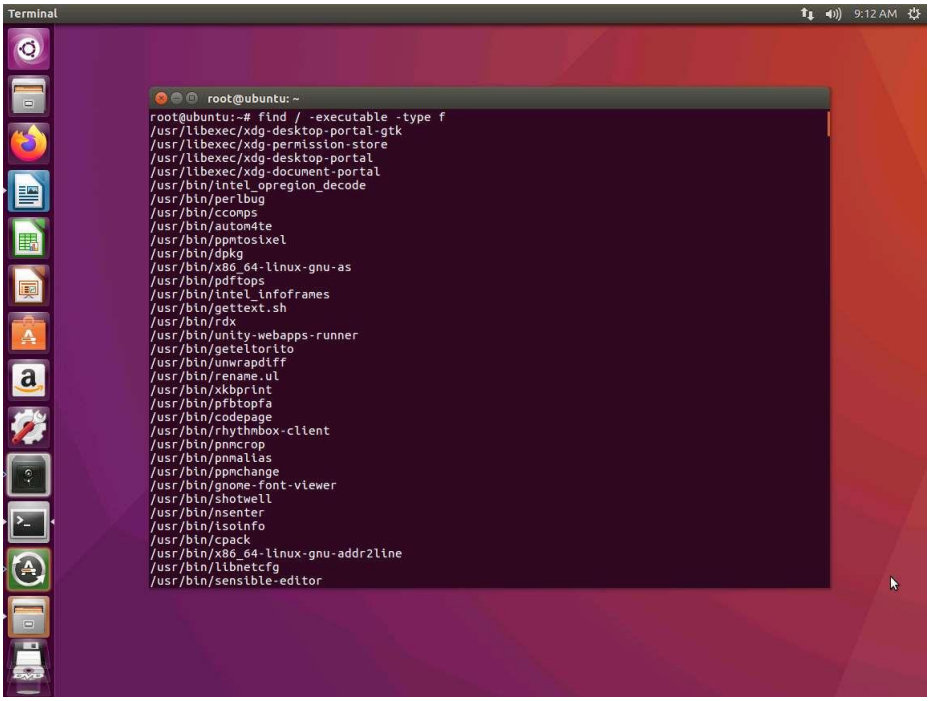
1. ☐ Login to the Software-Test-Linux machine using **studentpassword** as Password.



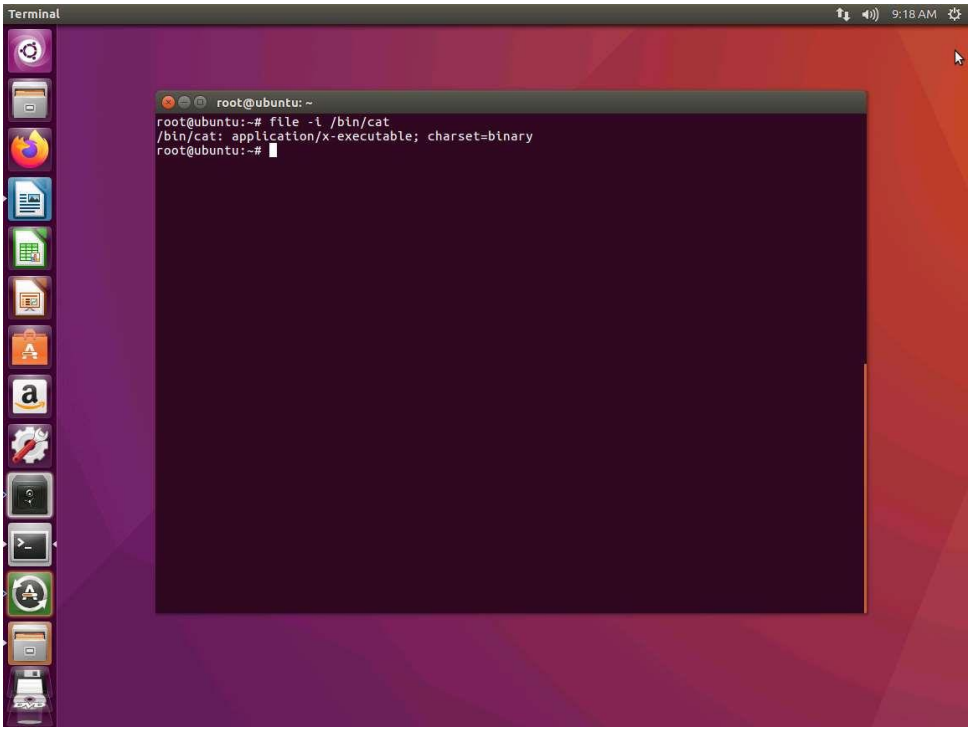
2. ☐ Launch a terminal window and enter **sudo -i**.



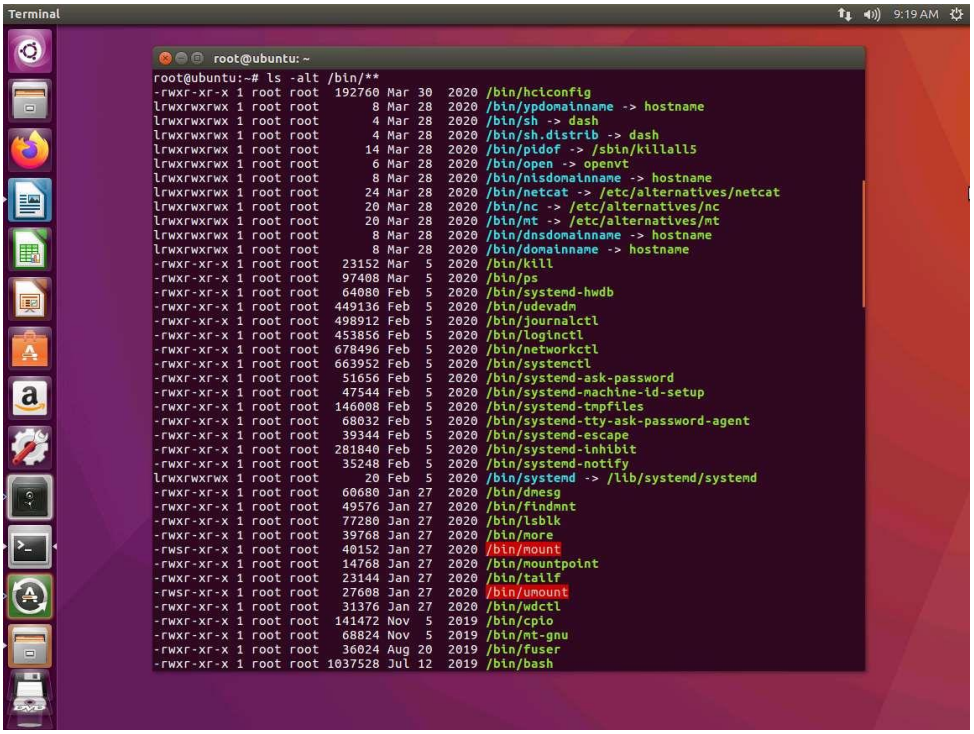
3. The first step is discovery. In the terminal window, enter **find / -executable -type f**. The output of the command will display a long list of programs, but we wanted to cover the step, since we are doing binary analysis and need to find executable files.



4. Next, in the terminal, enter **file -i /bin/cat**. The output of the command is shown in the following screenshot.

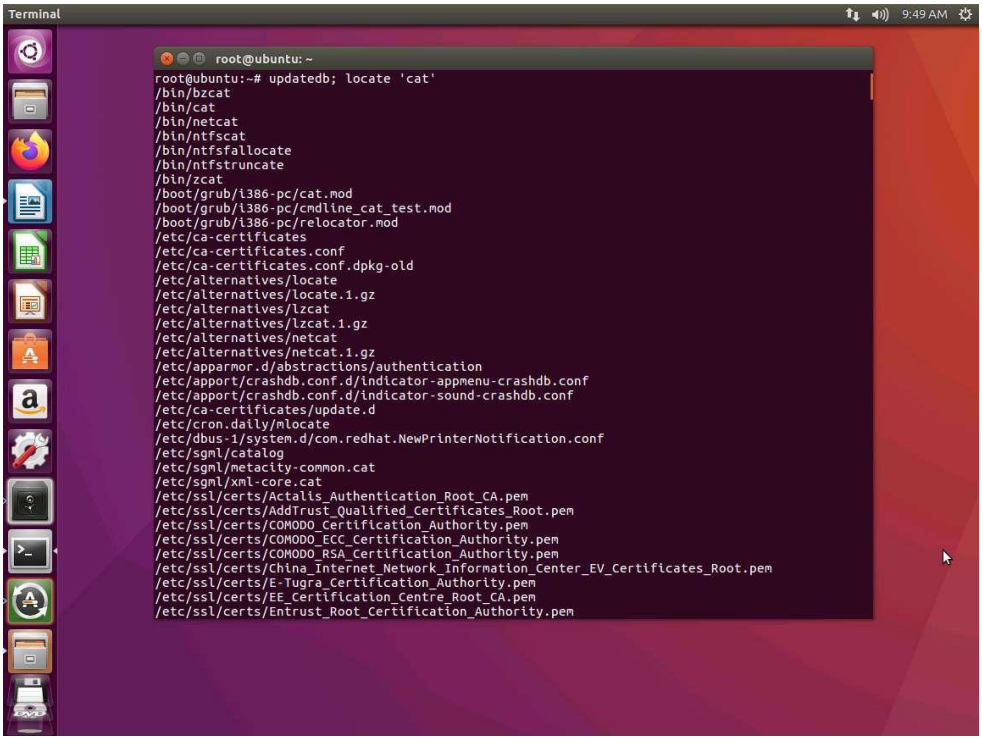


5. Next, enter **ls -alt /bin/**. The output of the command is shown in the following screenshot.



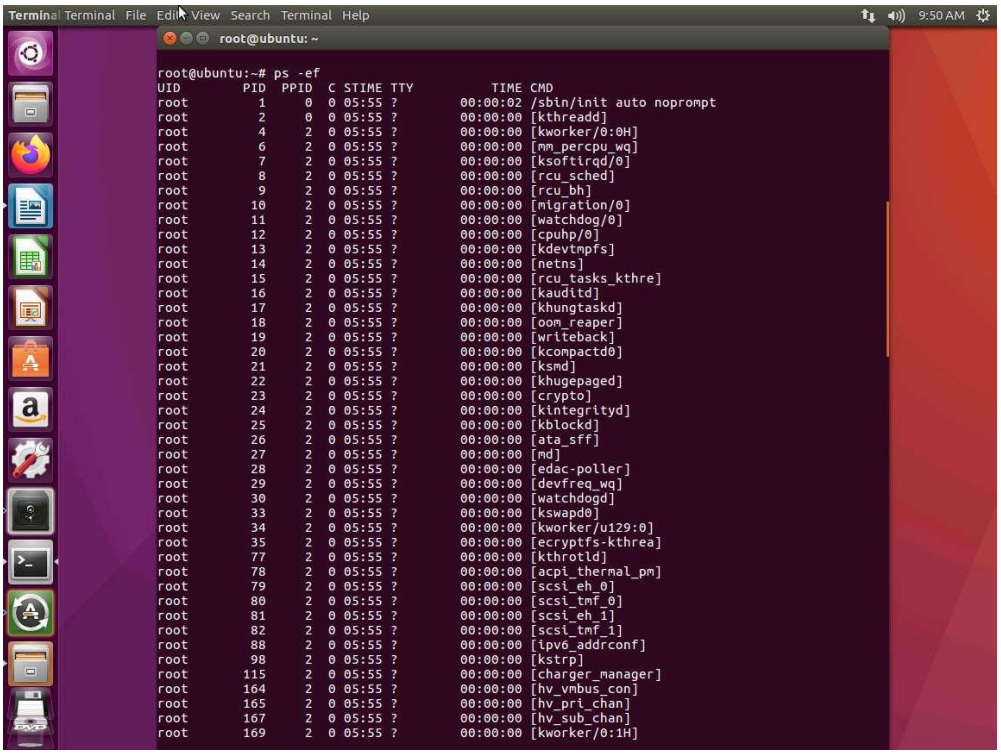
6. The **-i** argument is used to look at the /**bin/cat** binary and to show the results as strings for the mime type and mime encoding of the binary itself. Note the application/x-executable; charset=binary portion of the output.

7. Next, in the terminal window, enter **updatedb; locate 'cat'**. The output of this command is shown in the following screenshot.



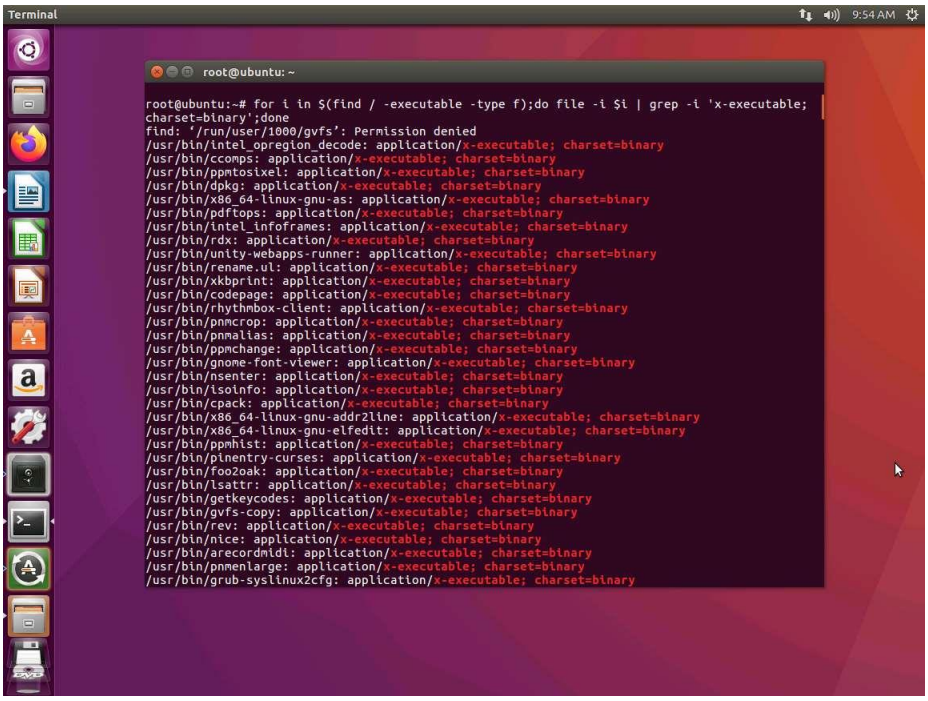
8. The command is used to update a database for the mlocate tool. We then use the locate command to hunt down any file with the word cat in the name. These tools together make it fairly easy to track down binaries written to disk once we have the name of the binary. For example, as a result of reviewing the running processes on the host.

9. In the terminal window, enter **ps -ef**. An example of the output of the command is shown in the following screenshot.



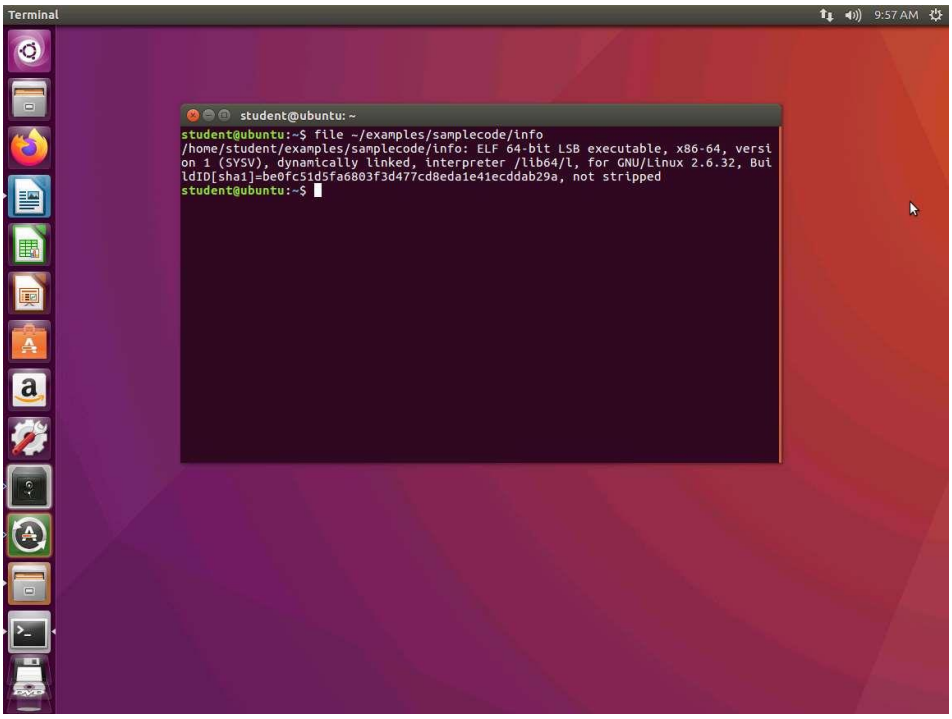
10. This command displays all running processes by all users of the system, using the full format for displaying the output.

11. In the terminal window, enter **for i in $(find / -executable -type f);do file -i $i | grep -i 'x-executable; charset=binary';done**. An example of the output is shown in the following screenshot.



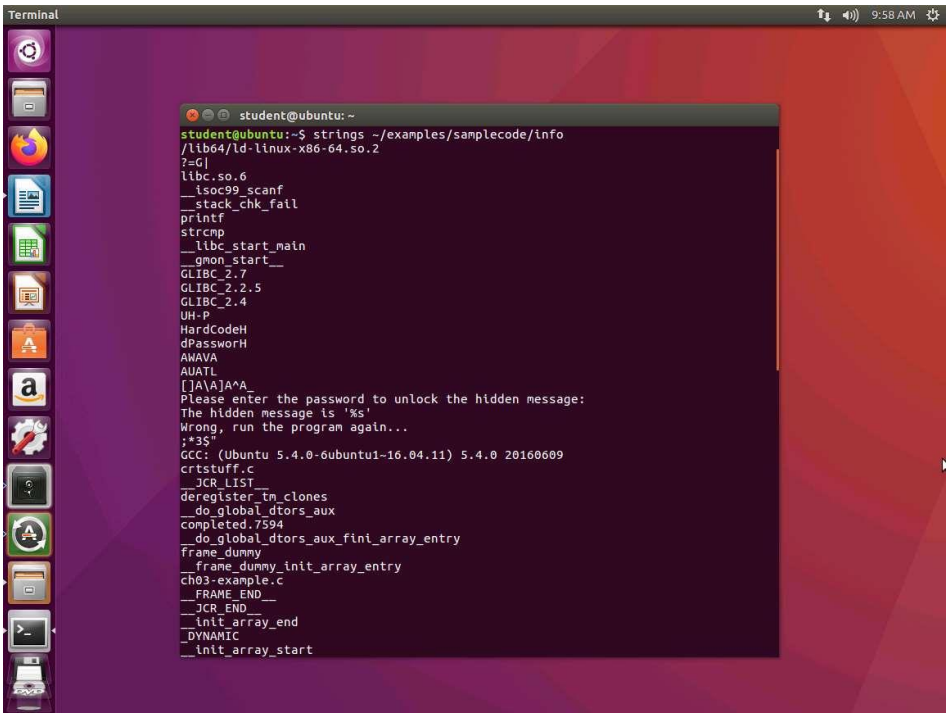
12. This command shows for every line in the output of our find command, run the file command and grep the output to only display executable binary files.

13. We are now ready for the information gathering step. Launch a new terminal window, enter **file ~/examples/samplecode/info**. The output of the command is shown in the following screenshot.



14. ☐ We can see that this is a 64-bit executable and linking format (ELF) formatted executable binary that contains its symbol table (because it is not stripped in our output). The file command is a great way to start because it gives us quick yet detailed information about the format of the binary and other pertinent information such as the architecture, whether we are dealing with an executable or a relocatable object file, the binary hash, and whether or not the binary has been stripped of its symbol table or not. The output of the file command is largely dependent upon the options used when the binary was compiled.

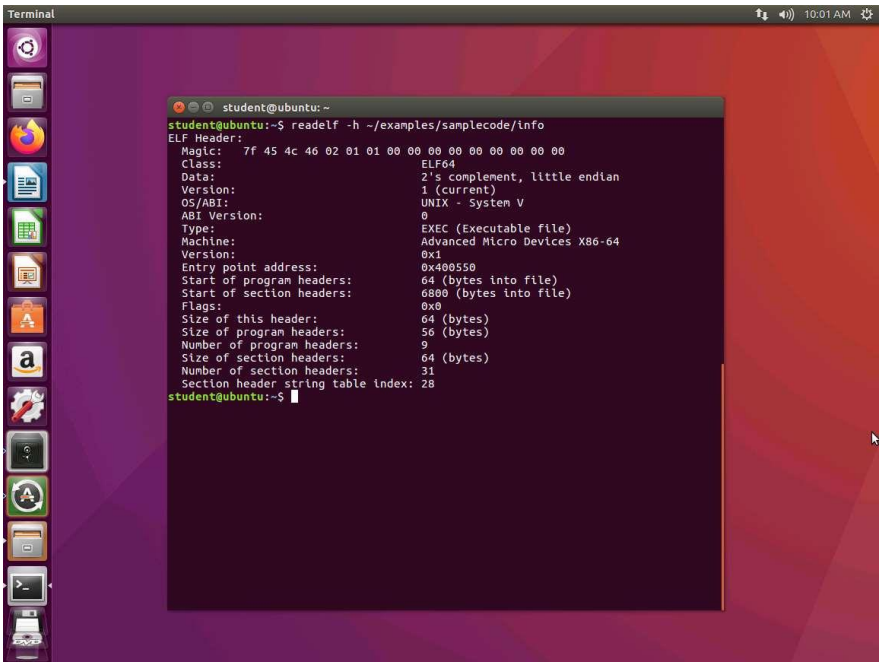
15. ☐ Next, in the terminal window, enter **strings ~/examples/samplecode/info**. The output of the command is shown in the following screenshot.



16. ☐ This output reveals what appears to be a hardcoded password, a message requesting a password, a sentence using the C-style %s format string, and what appears to be a failure message if the password is incorrect.

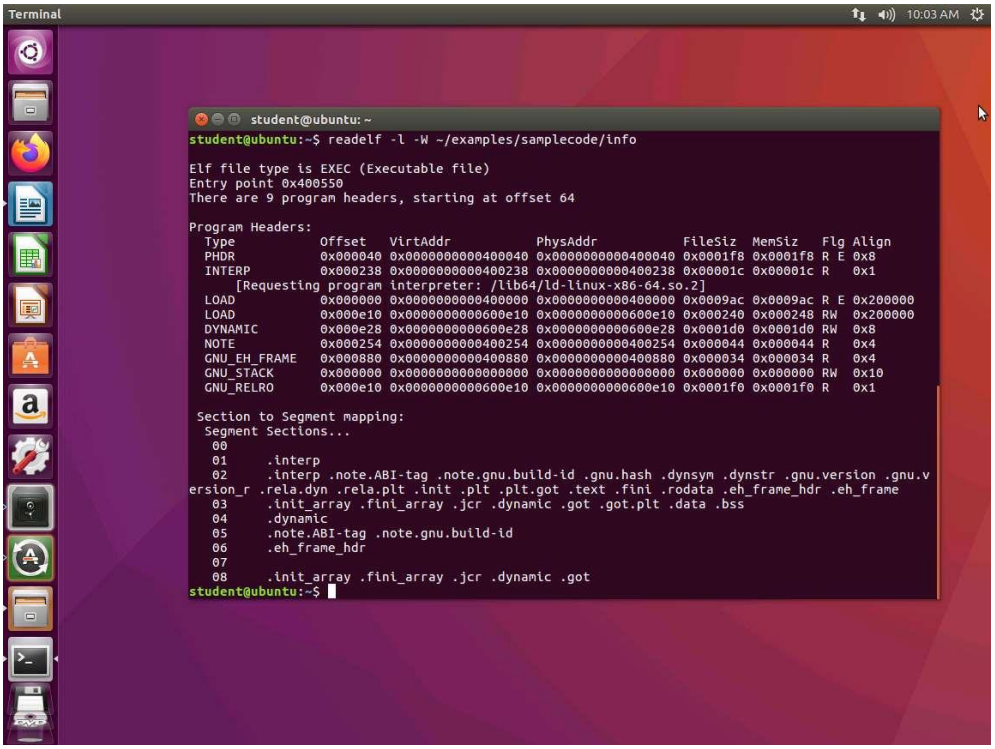
17. ☐ You will also notice the use of scanf, printf, and strcmp near the beginning of the output. For those familiar with C programming, printf is used to display output to stdout, scanf is used to take input from stdin, and strcmp is used to compare two strings. Therefore, what this tells us is that this program uses these functions in some way. Based on the hardcoded password and the sentence requesting the password, you could make some assumptions about the program; however, it is much safer to be certain than to assume anything from the output.

18. ☐ Next, we will return to our readelf command. Enter **readelf -h ~/examples/samplecode/info**.



19. ☐ The readelf tool is invaluable in our quest for gathering information about a binary.

20. ☐ Next, enter **readelf -l -W ~/examples/samplecode/info**. The output of the command is shown in the following screenshot.



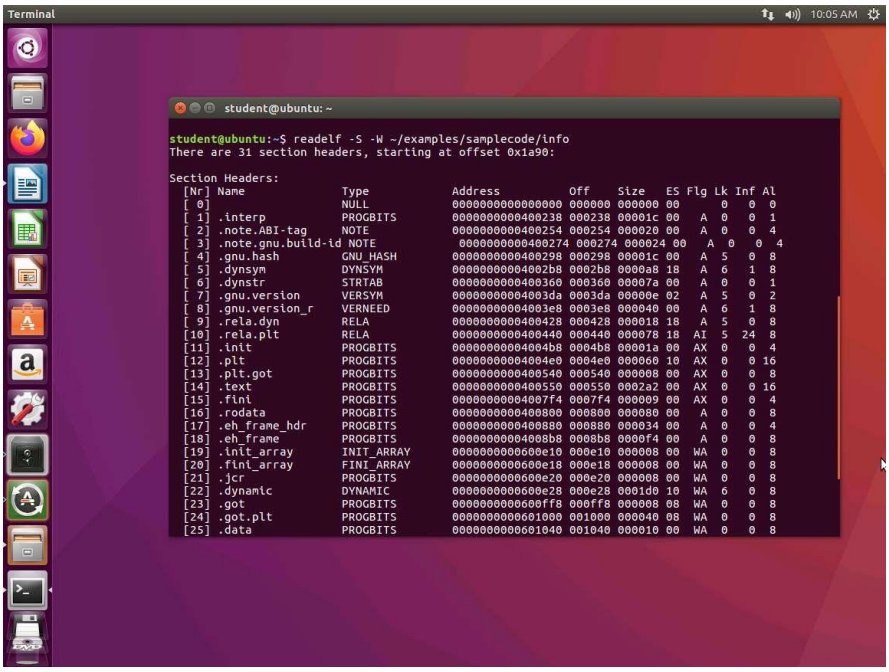
21. ☐ This provides us with additional information as well. Next, enter the following commands and review the output of each one.

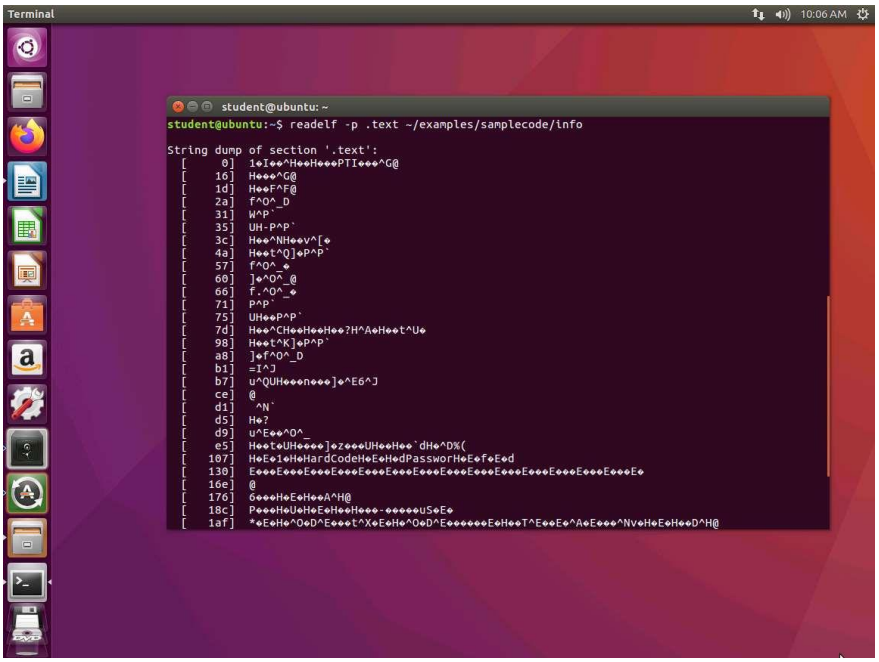
    a. **readelf -S -W ~/examples/samplecode/info**

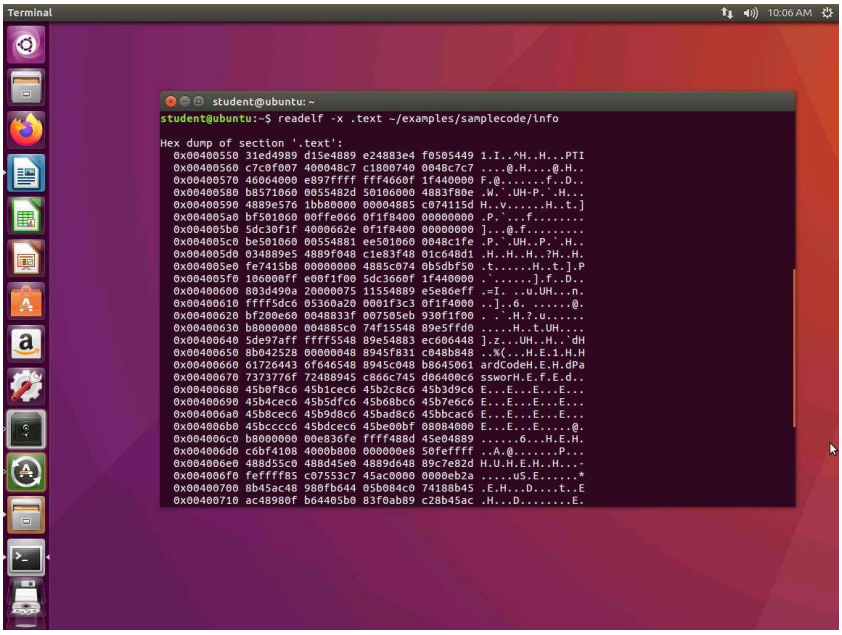
    b. **readelf -p .text ~/examples/samplecode/info**

    c. **readelf -x .text -W ~/examples/samplecode/info**

    An example of the output of these commands is shown in the following screenshot.


Terminal
10:05 AM
student@ubuntu: ~
student@ubuntu:~$ readelf -S -W ~/examples/samplecode/info
There are 31 section headers, starting at offset 0x1a90:
Section Headers:
[Nr] Name Type Address Off Size ES Flg Lk Inf Al
[0] NULL 0000000000000000 000000 000000 00 0 0 0
[1] .interp PROGBITS 0000000000400238 000238 00001c 00 A 0 0 1
[2] .note.ABI-tag NOTE 0000000000400254 000254 000020 00 A 0 0 4
[3] .note.gnu.build-id NOTE 0000000000400274 000274 000024 00 A 0 0 4
[4] .gnu.hash GNU_HASH 0000000000400298 000298 00001c 00 A 5 0 8
[5] .dynsym DYNSYM 00000000004002b8 0002b8 0000a8 18 A 6 1 8
[6] .dynstr STRTAB 0000000000400360 000360 00007a 00 A 0 0 1
[7] .gnu.version VERSYM 00000000004003da 0003da 00000e 02 A 5 0 2
[8] .gnu.version_r VERNEED 00000000004003e8 0003e8 000040 00 A 6 1 8
[9] .rela.dyn RELA 0000000000400428 000428 000018 18 A 5 0 8
[10] .rela.plt RELA 0000000000400440 000440 000078 18 AI 5 24 8
[11] .init PROGBITS 00000000004004b8 0004b8 00001a 00 AX 0 0 4
[12] .plt PROGBITS 00000000004004e0 0004e0 000060 10 AX 0 0 16
[13] .plt.got PROGBITS 0000000000400540 000540 000008 00 AX 0 0 8
[14] .text PROGBITS 0000000000400550 000550 0002a2 00 AX 0 0 16
[15] .fini PROGBITS 00000000004007f4 0007f4 000009 00 AX 0 0 4
[16] .rodata PROGBITS 0000000000400800 000800 000080 00 A 0 0 8
[17] .eh_frame_hdr PROGBITS 0000000000400880 000880 000034 00 A 0 0 4
[18] .eh_frame PROGBITS 00000000004008b8 0008b8 0000f4 00 A 0 0 8
[19] .init_array INIT_ARRAY 0000000000600e10 000e10 000008 00 WA 0 0 8
[20] .fini_array FINI_ARRAY 0000000000600e18 000e18 000008 00 WA 0 0 8
[21] .jcr PROGBITS 0000000000600e20 000e20 000008 00 WA 0 0 8
[22] .dynamic DYNAMIC 0000000000600e28 000e28 0001d0 10 WA 6 0 8
[23] .got PROGBITS 0000000000600ff8 000ff8 000008 08 WA 0 0 8
[24] .got.plt PROGBITS 0000000000601000 001000 000040 08 WA 0 0 8
[25] .data PROGBITS 0000000000601040 001040 000010 00 WA 0 0 8


Terminal
10:06 AM
student@ubuntu: ~
student@ubuntu:~$ readelf -p .text ~/examples/samplecode/info
String dump of section '.text':
[0] 1eIe◆^He◆He◆◆PTIe◆◆^G@
[16] He◆◆^G@
[1d] He◆F^F@
[2a] f^O^_D
[31] W^P`
[35] UH-P^P`
[3c] He◆^NHe◆v^[◆
[4a] He◆t^Q]◆P^P`
[57] f^O^_◆
[60]]◆^O^_@
[66] f.^O^_◆
[71] P^P`
[75] UHe◆P^P`
[7d] He◆^CHe◆He◆He◆?H^A◆He◆t^U◆
[98] He◆t^K]◆P^P`
[a8]]◆f^O^_D
[b1] =I^J
[b7] u^QUHe◆◆n◆◆◆]◆^E6^J
[ce] @
[d1] ^N`
[d5] He?
[d9] u^E◆◆^O^_
[e5] He◆t◆UHe◆◆◆]◆z◆◆◆UHe◆He◆`dHe◆^D%(
[107] HeEe1◆HeHardCodeHeEeHedPasswordHeEefeEed
[130] E◆◆◆E◆◆◆E◆◆◆E◆◆◆E◆◆◆E◆◆◆E◆◆◆E◆◆◆E◆◆◆E◆◆◆E◆◆◆E◆◆◆E◆◆◆E◆
[16e] @
[176] 6◆◆◆HeEeHe◆A^H@
[18c] P◆◆◆HeUeHeEeHe◆He◆◆-◆◆◆◆◆uSeE◆
[1af] *◆EeHe^O◆D^E◆◆◆t^X◆EeHe^O◆D^E◆◆◆◆◆◆EeHe◆T^E◆◆E◆^A◆E◆◆◆^NveHeEeHe◆D^H@



22. As the above screenshot shows, we see the hardcoded password in the dump.

23. In the terminal, enter the next series of commands and review the output of each one.

    a. readelf -R .text -W ~/examples/samplecode/info

    b. readelf -p .strtab -W ~/examples/samplecode/info

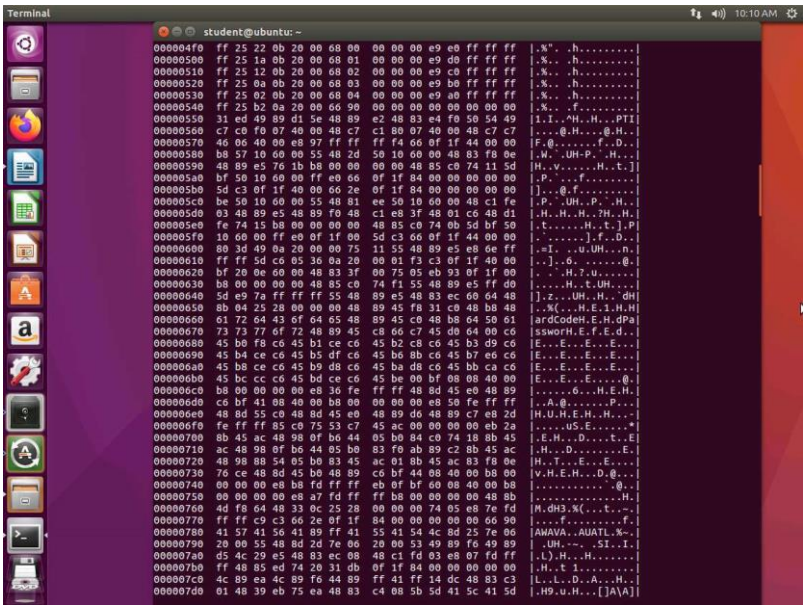
    c. objdump -f ~/examples/samplecode/info

    d. objdump -j .text -s ~/examples/samplecode/info

    e. objdump -x ~/examples/samplecode/info

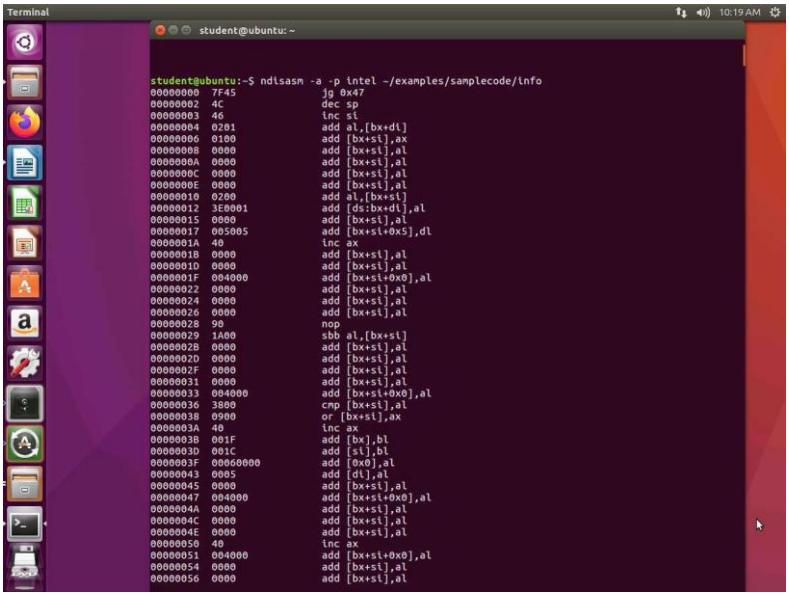
    f. hexdump -C ~/examples/samplecode/info

    An example of the output of these commands is shown in the following screenshot.

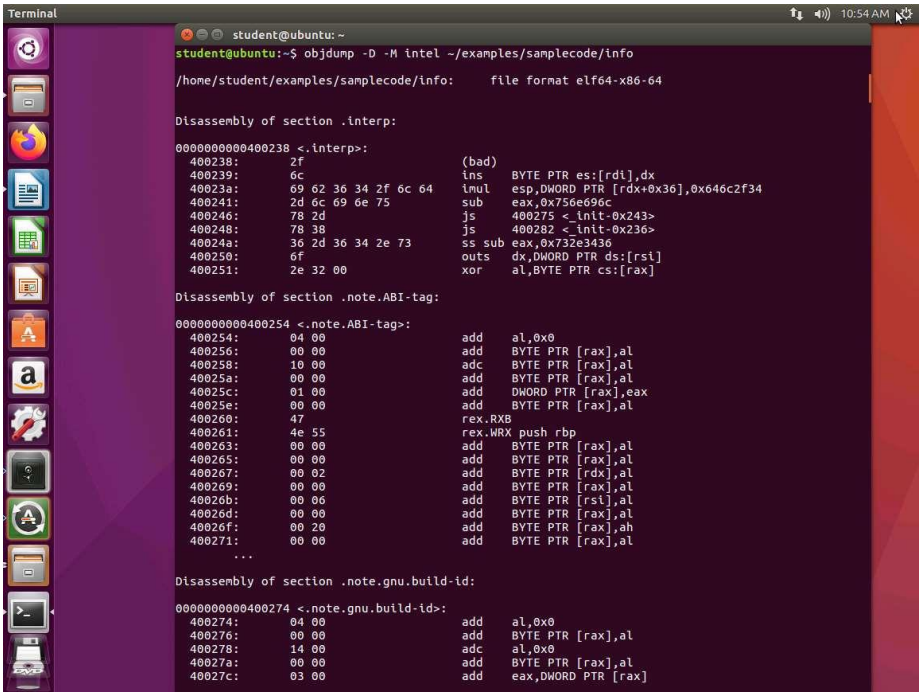


24. ☐ You are encouraged to examine which output you prefer and use that tool while also understanding any shortcomings it may have. An alternative tool to **hexdump** is **xxd**. A more powerful tool, because it allows us to modify the binary, is **hexedit**. You can install any of these using the APT package manager on the Ubuntu virtual machines by typing the following in a terminal session: **sudo apt install hexedit -y**. Note that an Internet connection is mandatory.

25. ☐ The next tool we will use is the **Netwide Disassembler**. In the terminal window, enter **ndisasm -a -p intel ~/examples/samplecode/info**. An example of the output of the command is shown in the following screenshot.



26. ☐ The next command to enter is **objdump -D -M intel ~/examples/samplecode/info**. An example of the output of the command is shown in the following screenshot.



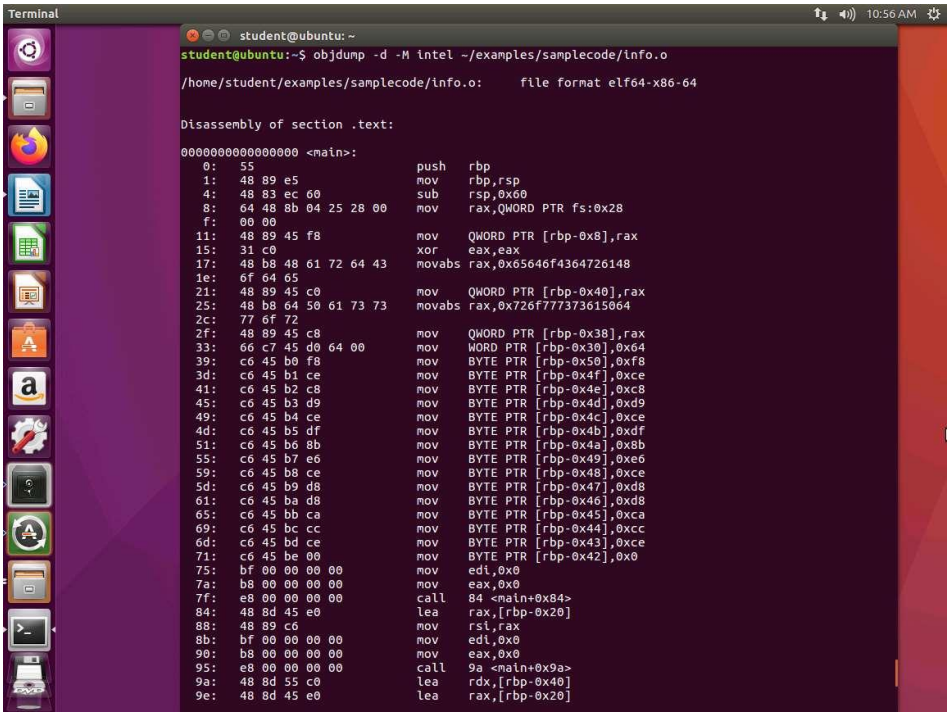
27. ☐ As a reminder, the **intel** option is to format the output in Intel and not the default of AT&T.

28. ☐ Enter the next series of commands and review the output of each:

    a. objdump -d -M intel ~/examples/samplecode/info

    b. objdump -d -M intel ~/examples/samplecode/info.o

29. ☐ An example of the dump of the object code file is shown in the following screenshot.



30. ☐ Note the difference in output between disassembling the assembled object file only. This file has not been linked yet and has only undergone the preprocessor and assembly processes.

31. ☐ At this stage, we want to look at dynamic analysis. Before we do that, we can work through the debugger to get more practice. Enter the following commands and examine the output from each one.

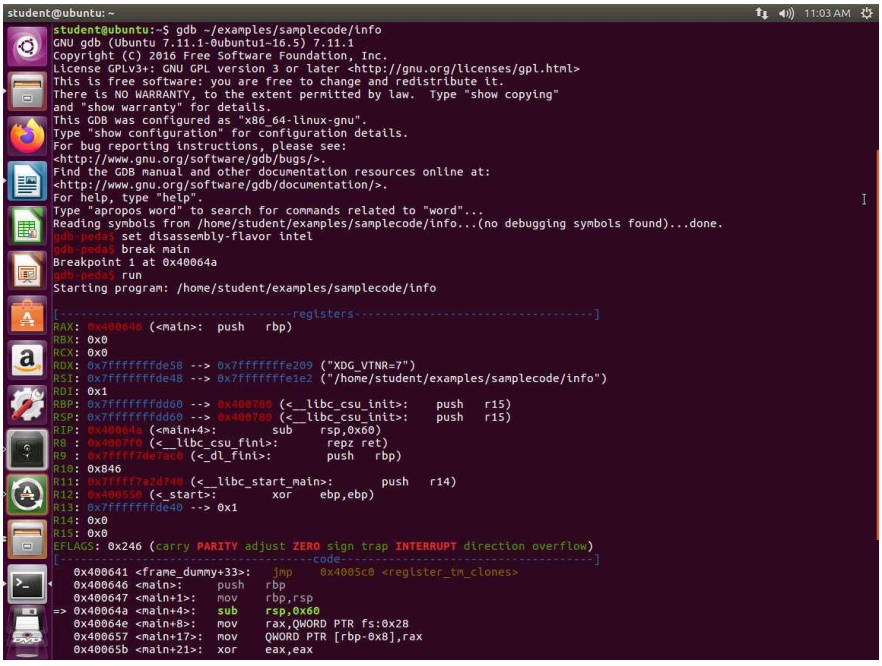
a. gdb ~/examples/samplecode/info

b. set disassembly-flavor intel

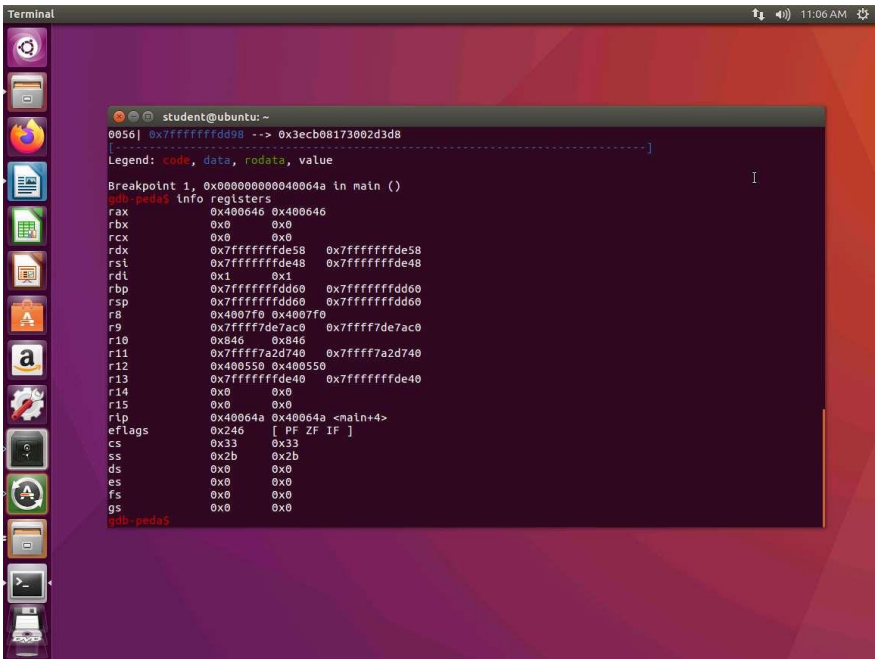
c. break main

d. run

32. ☐ An example of when the program hits the breakpoint is shown in the following screenshot.



33. ☐ Now, we are ready to view the contents of the registers. Enter **info registers**.



34. Next, enter the following series of commands:

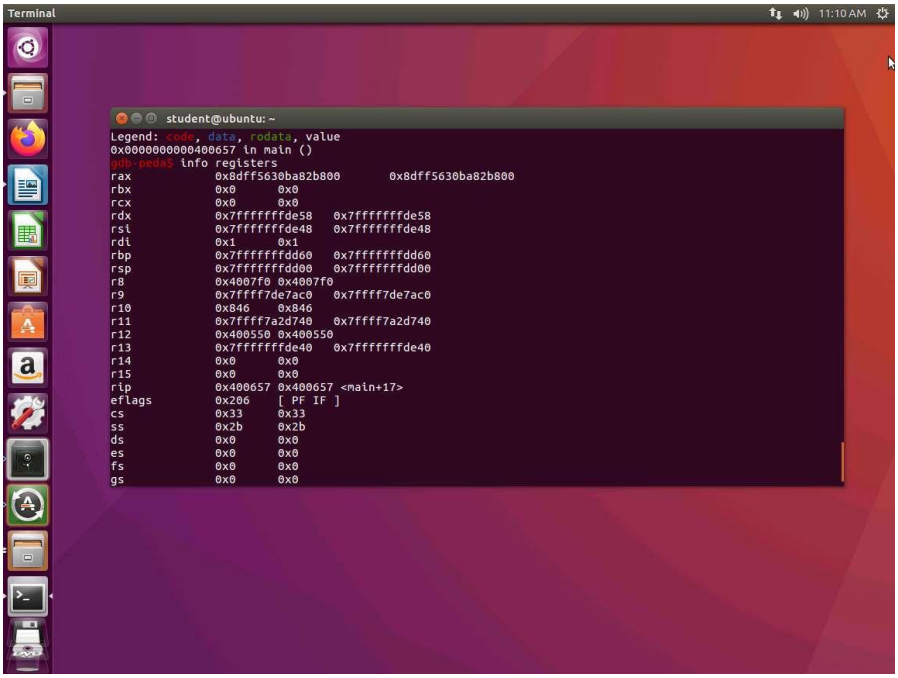
a. nexti

b. info registers

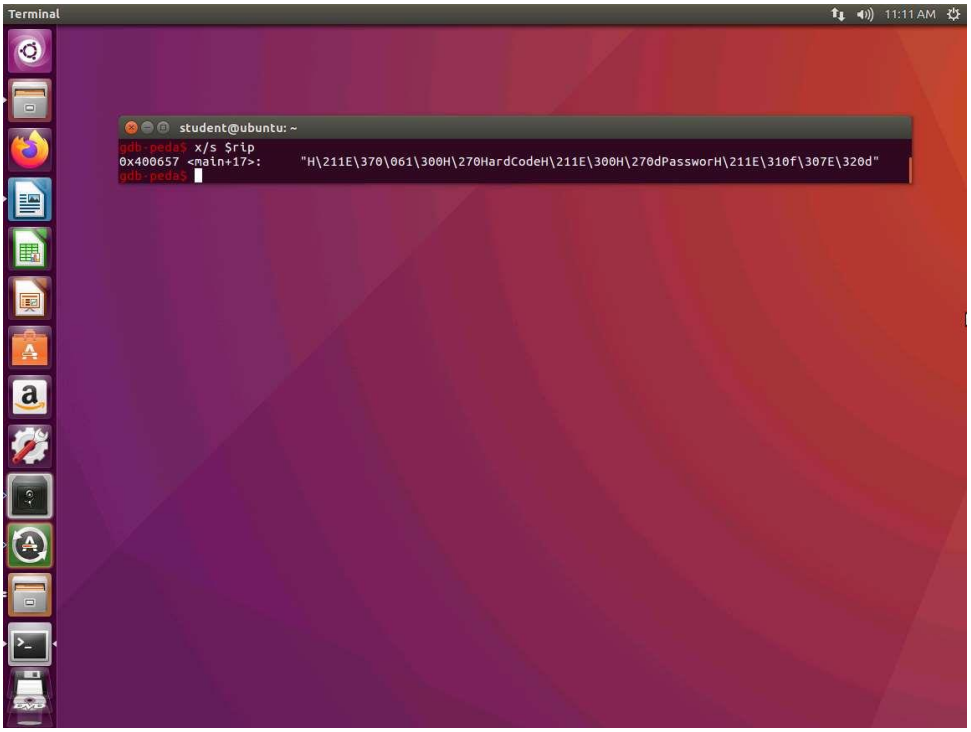
c. nexti

d. info registers

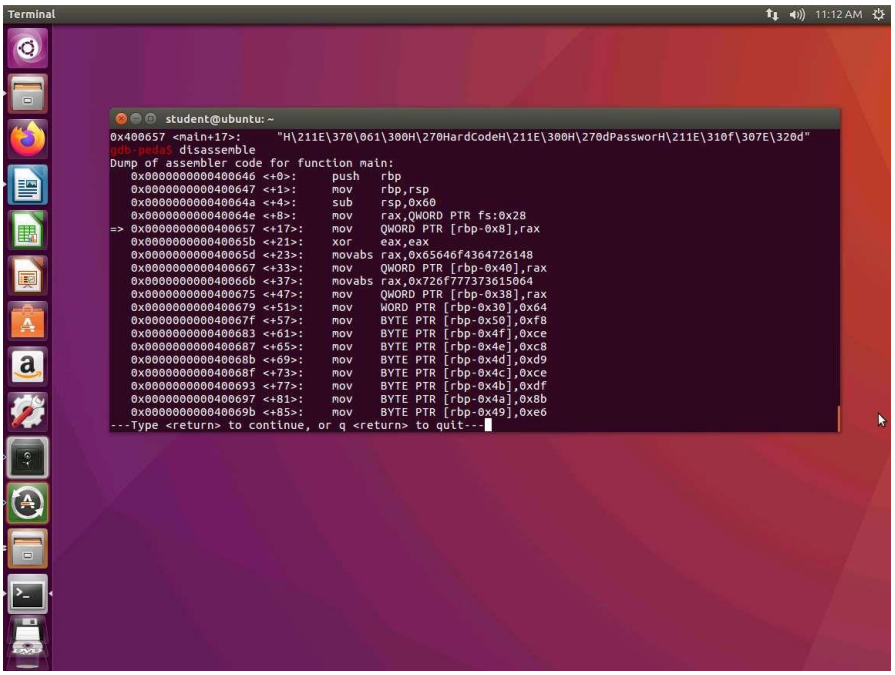
An example of the output is shown in the following screenshot.



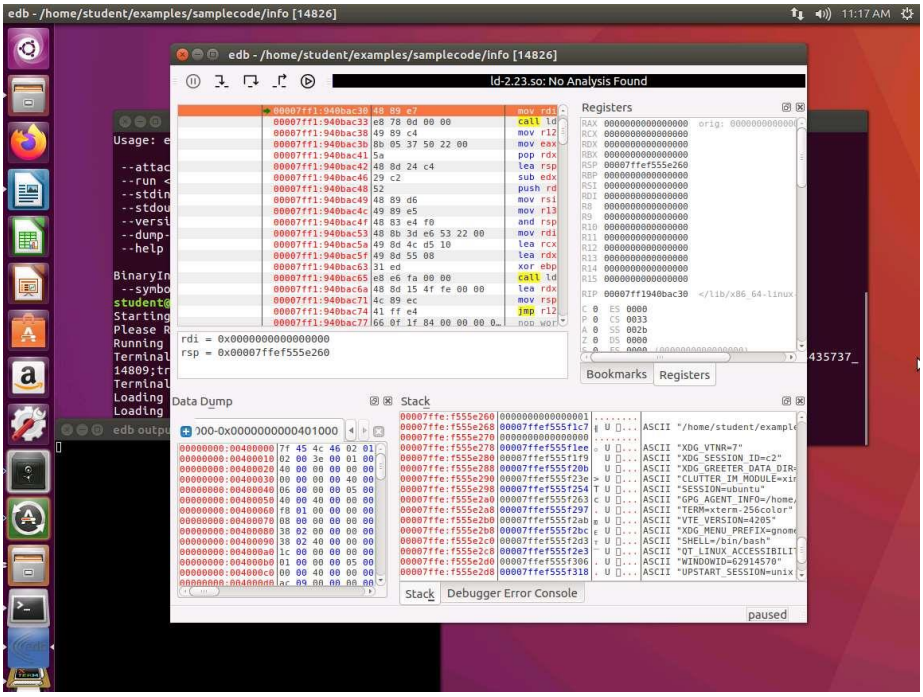
35. Enter the **x/s $rip** command, if you are using the 64-bit system. Essentially, this final command tells GDB we want to examine the RIP register and display the output as a string.



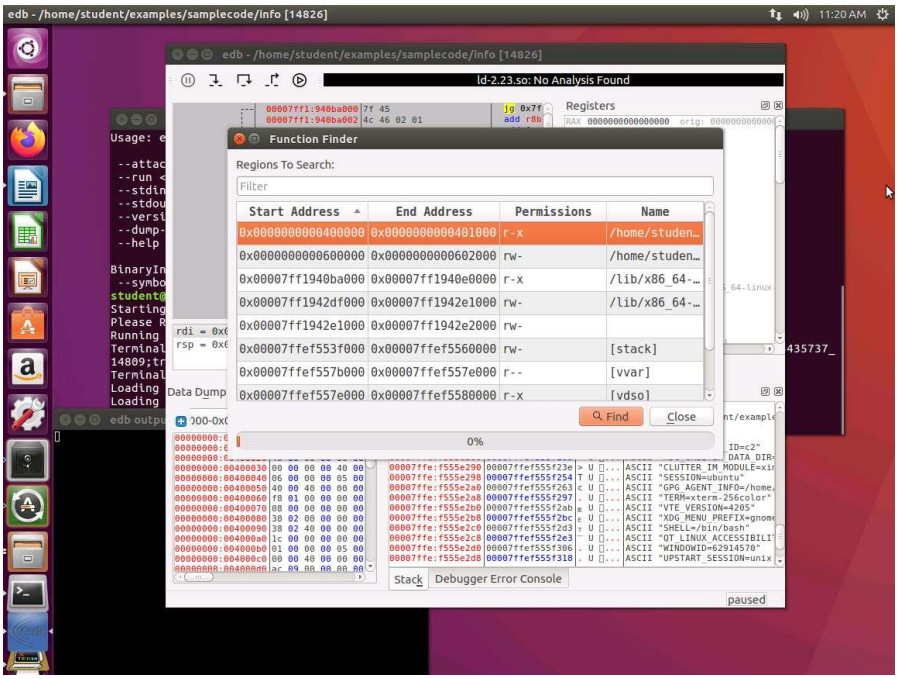
36. ☐ Next, enter disassemble. Take a few minutes to review the output of this command.



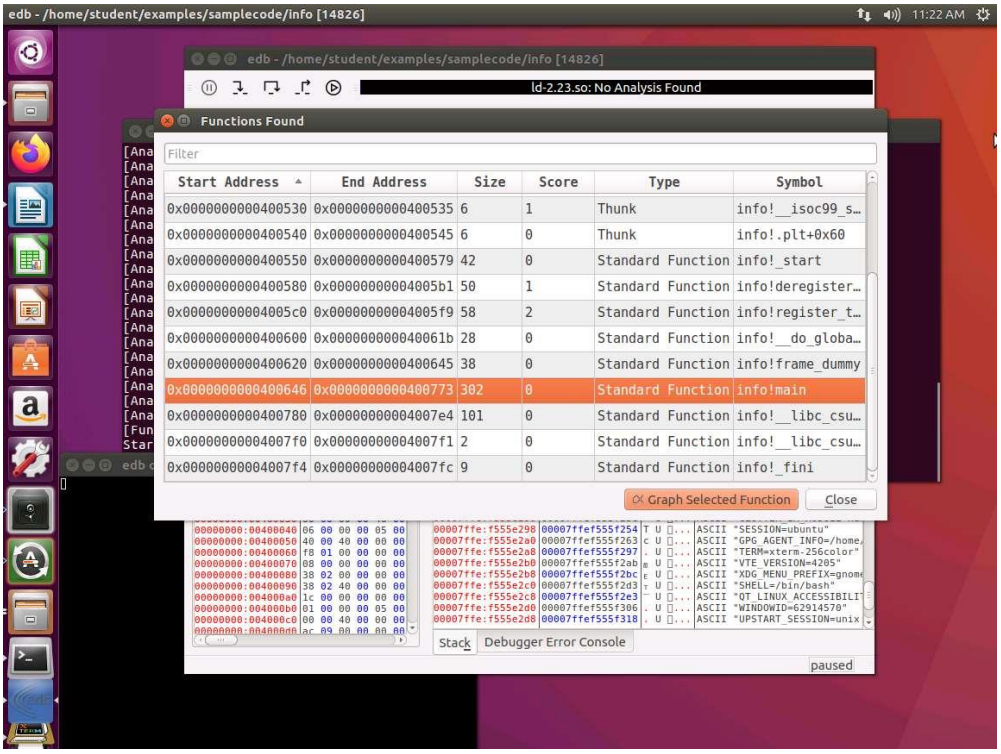
37. ☐ We are now ready to move to the next step. Enter quit. This will exit the debugger. We will try another. Enter **edb --run ~/examples/samplecode/info**. This will launch Evans Debugger as shown in the following screenshot.

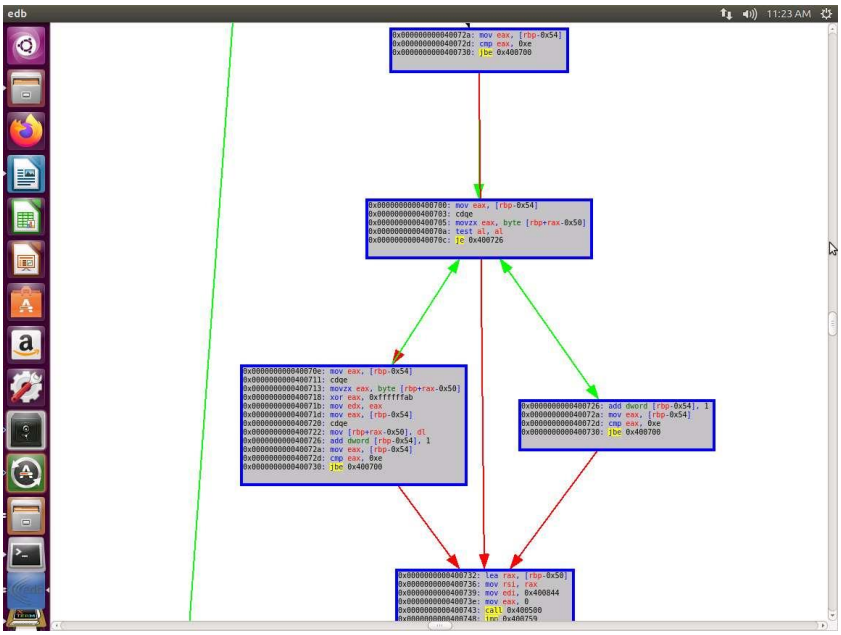


38. Next, click on **CTL+SHIFT+f** to open the function finder. The window will open with the first instruction highlighted as shown in the following screenshot.

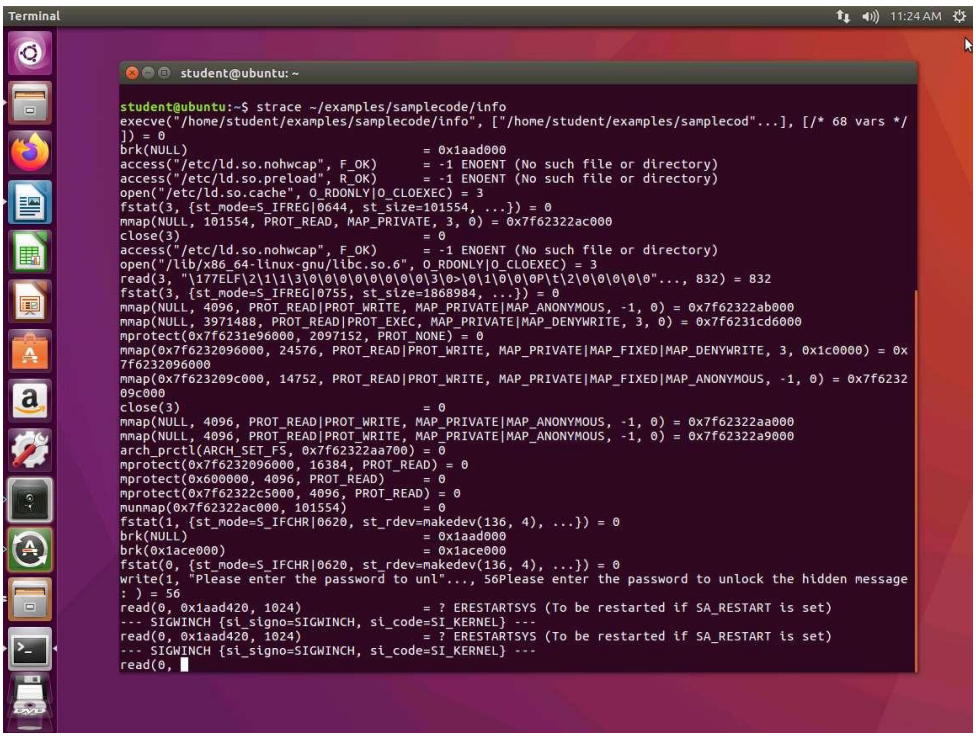


39. ☐ Next, click on the **Find** button, and look for the info main function in the symbol table. Click on it, and then select **Graph Selected Function**. Take a few minutes to review the results from the graphing of the function.



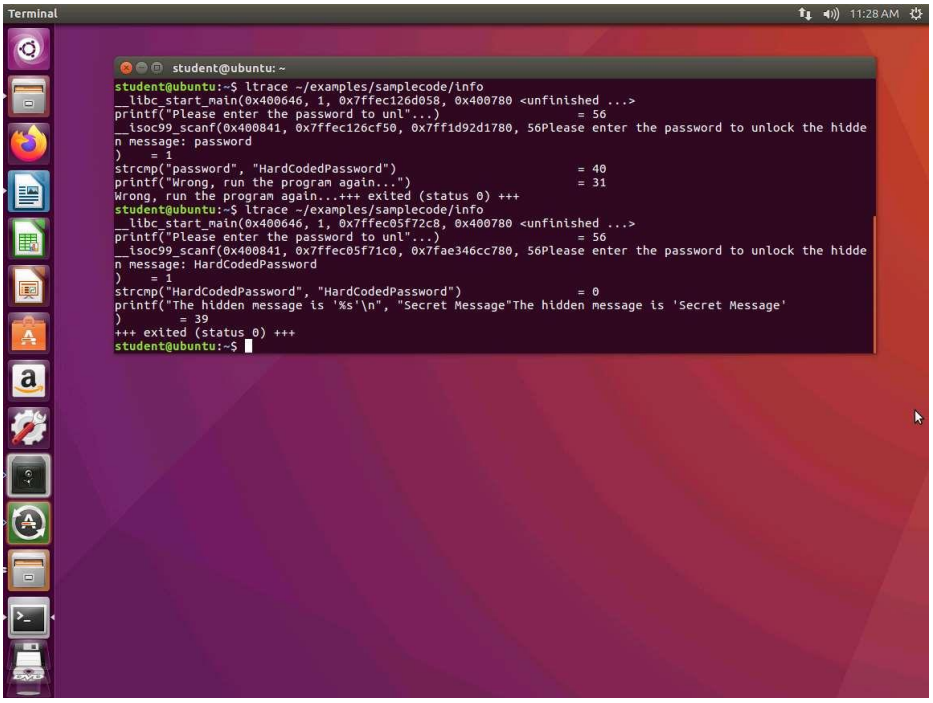


40. As you see, there is a lot of capability and power with the tool edb. Close edb and enter **strace ~/examples/samplecode/info**. An example of the output of the command is shown in the following screenshot.



41. Take a few minutes and observe the results. Enter a variety of strings and observe the results again.

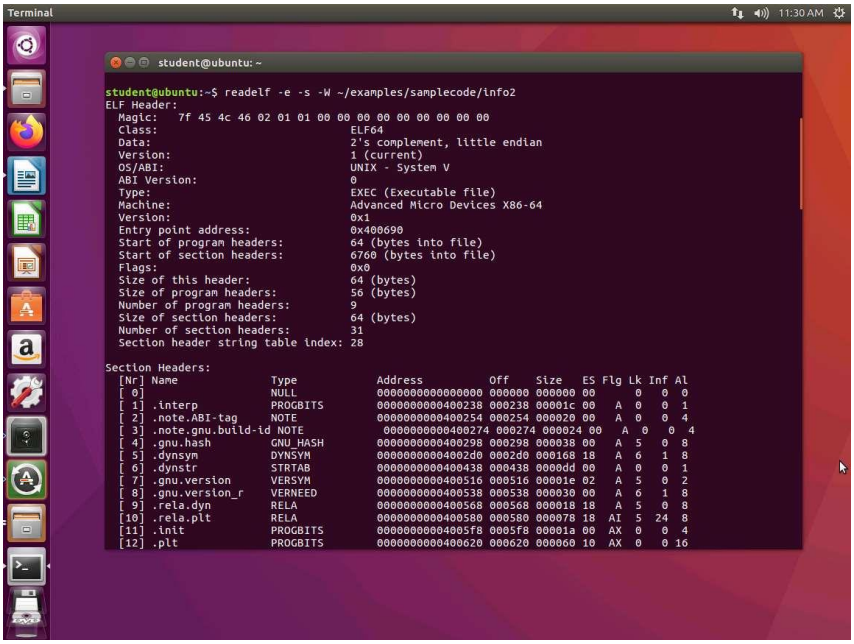
42. Next, enter **ltrace ~/examples/samplecode/info**. Again, enter different strings and observe the output. You will see the string compare function. This is because **ltrace** shows the libraries that are in the executable. An example of this is shown in the following screenshot.



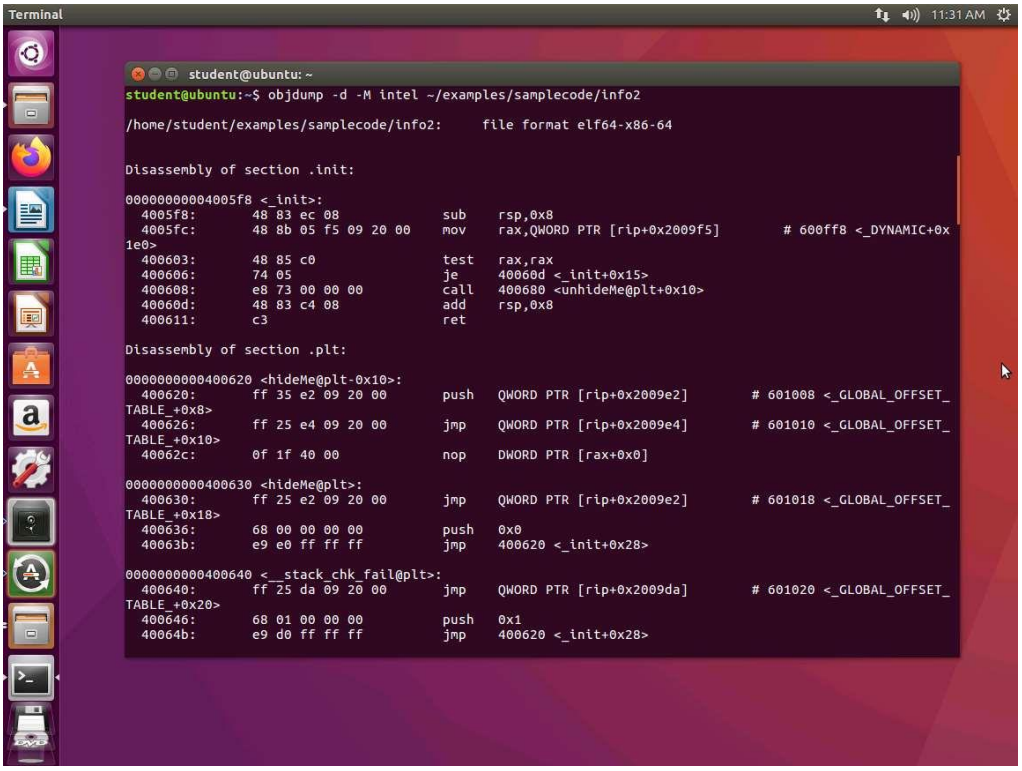
43. ☐ It is a good idea to use automation. An example of an automation script is in the samplecode folder, and it is called **automation.sh**.

44. ☐ We will analyze one more binary before we complete the lab. Enter the following commands and analyze the output:

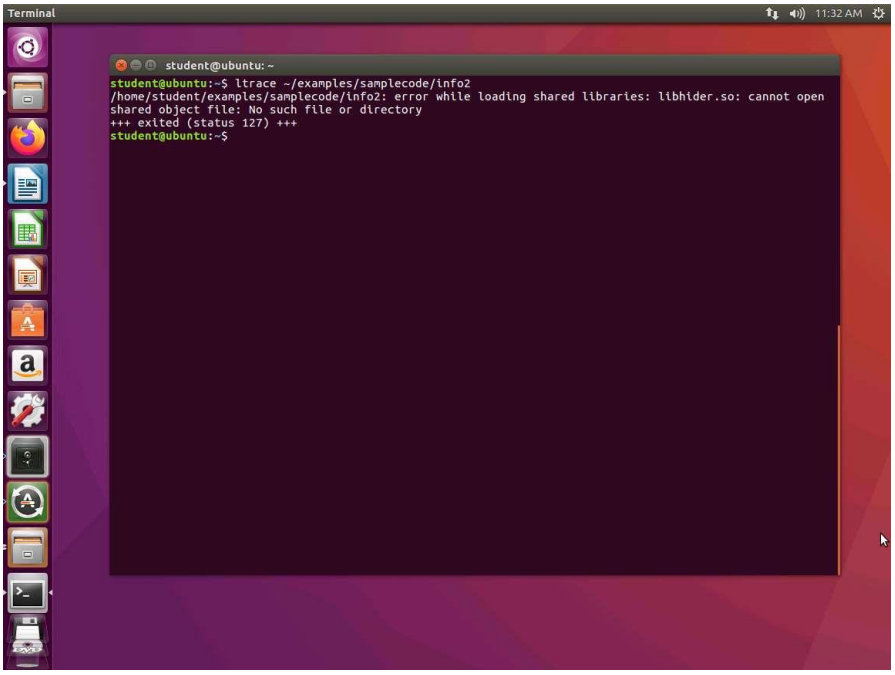
    a. readelf -e -s -W ~/examples/samplecode/info2



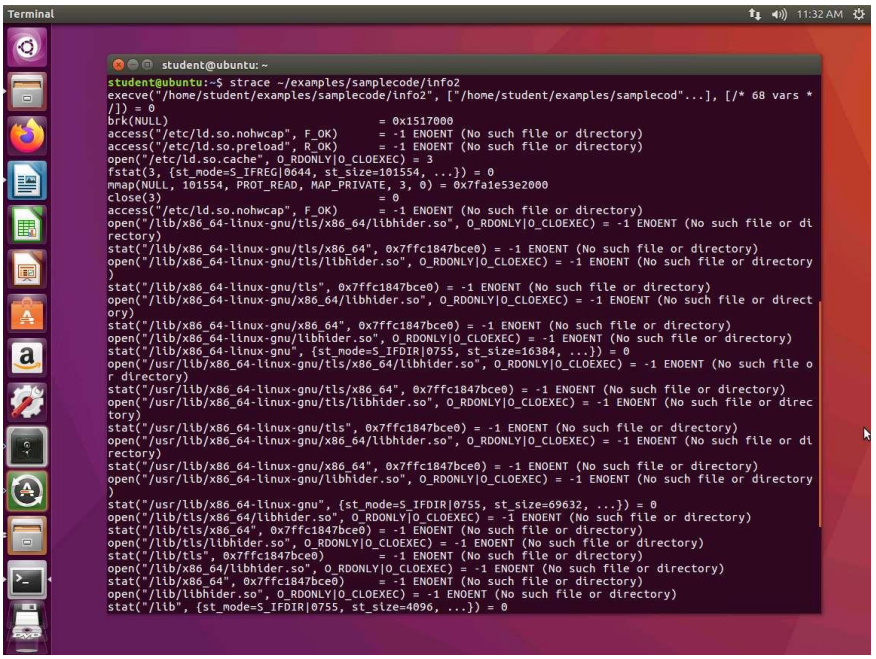
b. objdump -d -M intel ~/examples/samplecode/info2



c. ltrace ~/examples/samplecode/info2



d. strace ~/examples/samplecode/info2



45. ☐ The lab objectives have been achieved. Close all windows and clean up as required.