

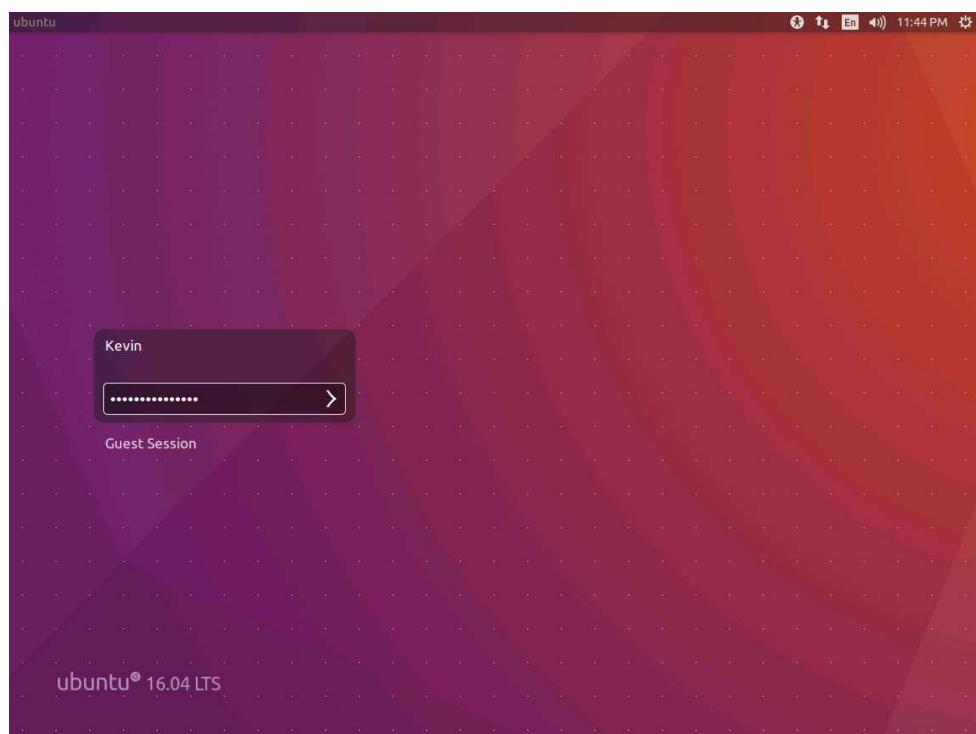
Exercise 3: Binary Analysis Methodology

Lab Objective:

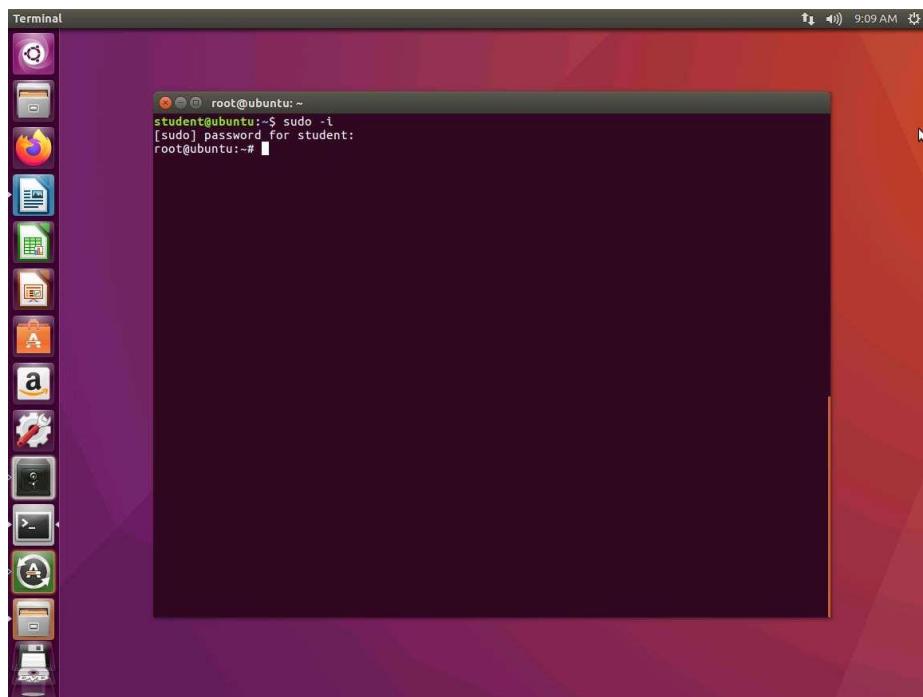
Learn binary analysis methodology.

Lab Tasks

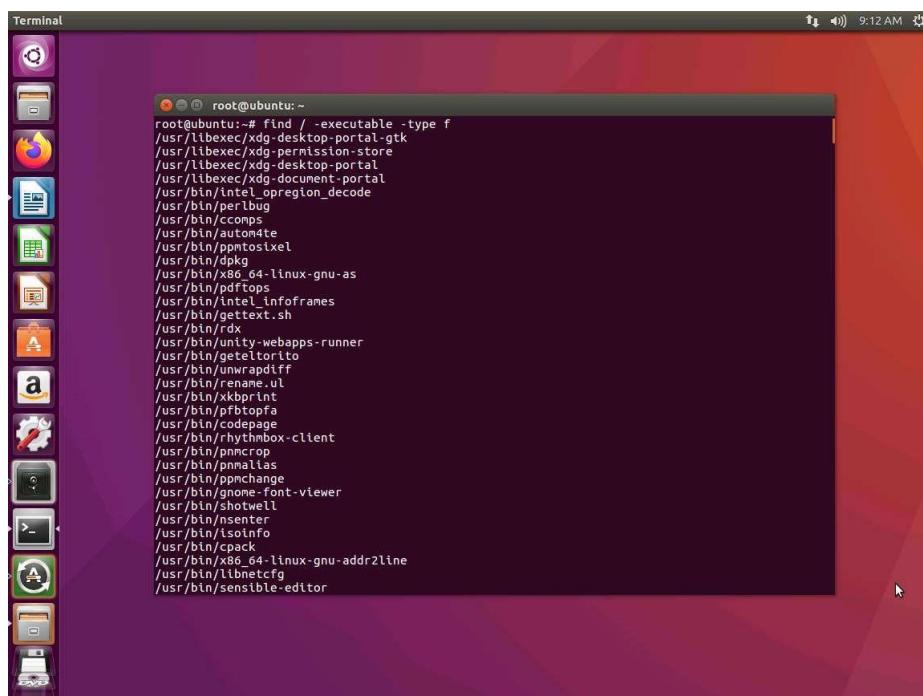
1. Login to the [Software-Test-Linux](#) machine using **studentpassword** as Password.



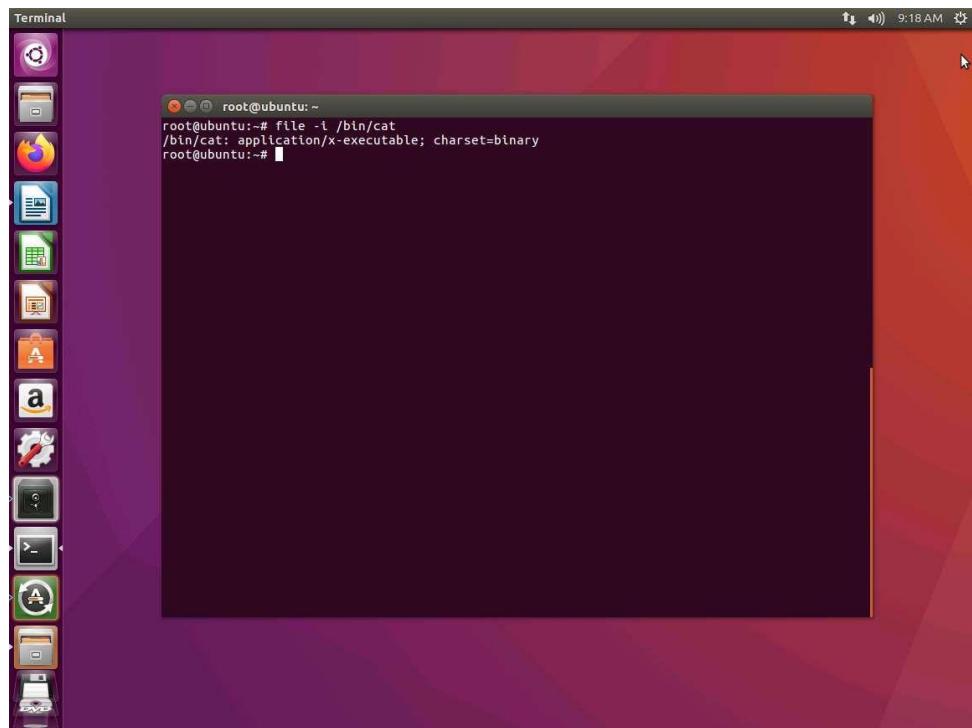
2. Launch a terminal window and enter **sudo -i**.



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.



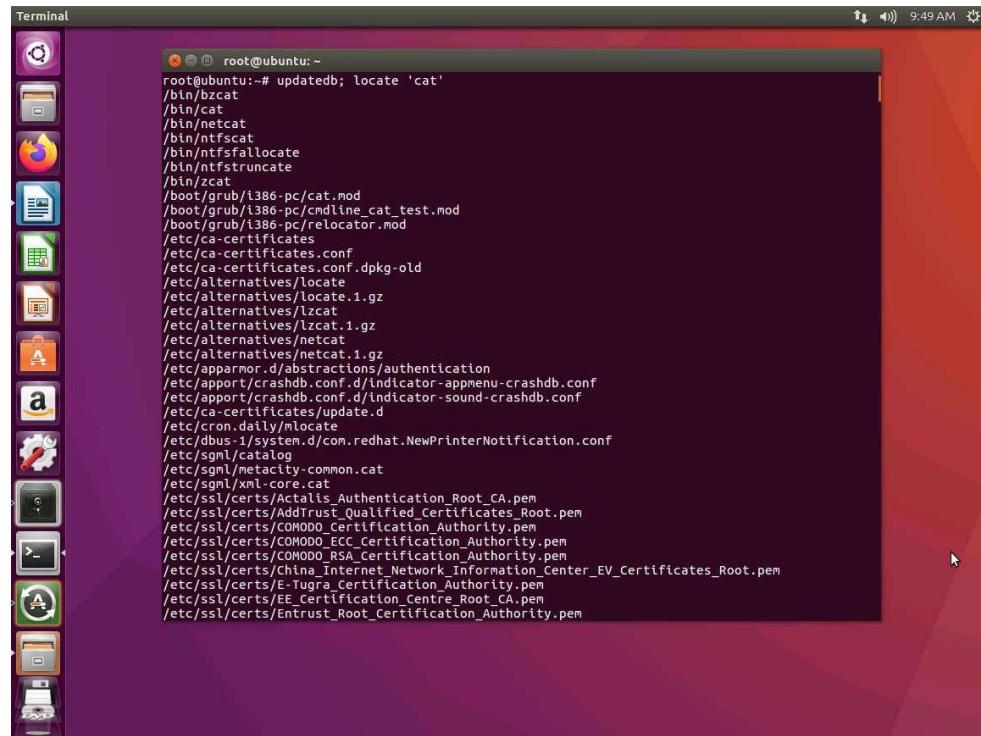
5. Next, enter **ls -alt /bin/**. The output of the command is shown in the following screenshot.

```

root@ubuntu:~# ls -alt /bin/***
-rwxr-xr-x 1 root root 192760 Mar 30 2020 /bin/hciconfig
lrwxrwxrwx 1 root root 8 Mar 28 2020 /bin/ypdomainname -> hostname
lrwxrwxrwx 1 root root 4 Mar 28 2020 /bin/sh -> dash
lrwxrwxrwx 1 root root 4 Mar 28 2020 /bin/sh.distrib -> dash
lrwxrwxrwx 1 root root 14 Mar 28 2020 /bin/pidof -> /sbin/killalls
lrwxrwxrwx 1 root root 6 Mar 28 2020 /bin/open -> openvt
lrwxrwxrwx 1 root root 8 Mar 28 2020 /bin/nisdomainname -> hostname
lrwxrwxrwx 1 root root 24 Mar 28 2020 /bin/netcat -> /etc/alternatives/netcat
lrwxrwxrwx 1 root root 20 Mar 28 2020 /bin/nc -> /etc/alternatives/nc
lrwxrwxrwx 1 root root 20 Mar 28 2020 /bin/nt -> /etc/alternatives/nt
lrwxrwxrwx 1 root root 8 Mar 28 2020 /bin/dnsdomainname -> hostname
lrwxrwxrwx 1 root root 8 Mar 28 2020 /bin/domainname -> hostname
-rwxr-xr-x 1 root root 23152 Mar 5 2020 /bin/kill
-rwxr-xr-x 1 root root 97408 Mar 5 2020 /bin/ps
-rwxr-xr-x 1 root root 64080 Feb 5 2020 /bin/systemd-hwdb
-rwxr-xr-x 1 root root 449136 Feb 5 2020 /bin/udevadm
-rwxr-xr-x 1 root root 498912 Feb 5 2020 /bin/journalctl
-rwxr-xr-x 1 root root 453856 Feb 5 2020 /bin/logindctl
-rwxr-xr-x 1 root root 678496 Feb 5 2020 /bin/networkctl
-rwxr-xr-x 1 root root 663952 Feb 5 2020 /bin/systemctl
-rwxr-xr-x 1 root root 51656 Feb 5 2020 /bin/systemd-ask-password
-rwxr-xr-x 1 root root 47544 Feb 5 2020 /bin/systemd-machine-id-setup
-rwxr-xr-x 1 root root 146008 Feb 5 2020 /bin/systemd-tmpfiles
-rwxr-xr-x 1 root root 68032 Feb 5 2020 /bin/systemd-tty-ask-password-agent
-rwxr-xr-x 1 root root 39344 Feb 5 2020 /bin/systemd-escape
-rwxr-xr-x 1 root root 281840 Feb 5 2020 /bin/systemd-inhibit
-rwxr-xr-x 1 root root 35248 Feb 5 2020 /bin/systemd-notify
lrwxrwxrwx 1 root root 28 Feb 5 2020 /bin/systemd -> /lib/systemd/systemd
-rwxr-xr-x 1 root root 60680 Jan 27 2020 /bin/dmesg
-rwxr-xr-x 1 root root 49576 Jan 27 2020 /bin/rndmmt
-rwxr-xr-x 1 root root 77280 Jan 27 2020 /bin/lsblk
-rwxr-xr-x 1 root root 39760 Jan 27 2020 /bin/more
rwsr-xr-x 1 root root 40152 Jan 27 2020 /bin/mount
-rwxr-xr-x 1 root root 14768 Jan 27 2020 /bin/mountpoint
-rwxr-xr-x 1 root root 23144 Jan 27 2020 /bin/tailf
rwsr-xr-x 1 root root 27668 Jan 27 2020 /bin/umount
-rwxr-xr-x 1 root root 31376 Jan 27 2020 /bin/wdctl
-rwxr-xr-x 1 root root 141472 Nov 5 2019 /bin/cplo
-rwxr-xr-x 1 root root 68824 Nov 5 2019 /bin/mt-gnu
-rwxr-xr-x 1 root root 36024 Aug 20 2019 /bin/fuser
-rwxr-xr-x 1 root root 103752 Jul 12 2019 /bin/bash

```

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.



```
root@ubuntu:~# updatedb; locate 'cat'
/bin/bzcat
/bin/cat
/bin/netcat
/bin/ntfsfallocate
/bin/ntfstruncate
/bin/zcat
/boot/grub/i386-pc/cat.mod
/boot/grub/i386-pc/cmdline_cat_test.mod
/boot/grub/i386-pc/relocator.mod
/etc/ca-certificates
/etc/ca-certificates.conf
/etc/ca-certificates.conf.dpkg-old
/etc/alternatives/locate
/etc/alternatives/locate.1.gz
/etc/alternatives/lzcat
/etc/alternatives/lzcat.1.gz
/etc/alternatives/netcat
/etc/alternatives/netcat.1.gz
/etc/apparmor.d/abstractions/authentication
/etc/apport/crashdb.conf.d/indicator-appmenu-crashdb.conf
/etc/apport/crashdb.conf.d/indicator-sound-crashdb.conf
/etc/ca-certificates/update.d
/etc/cron.daily/mlocate
/etc/dbus-1/system.d/com.redhat.NewPrinterNotification.conf
/etc/sgml/catalog
/etc/sgml/metacity-common.cat
/etc/sgml/xml-core.cat
/etc/ssl/certs/Actuals_Authentication_Root_CA.pem
/etc/ssl/certs/AddTrust_Qualified_Certificates_Root.pem
/etc/ssl/certs/COMODO_ECC_Certification_Authority.pem
/etc/ssl/certs/COMODO_RSA_Certification_Authority.pem
/etc/ssl/certs/China_Internet_Network_Information_Center_EV_Certificates_Root.pem
/etc/ssl/certs/E-Tugra_Certification_Authority.pem
/etc/ssl/certs/EE_Certification_Centre_Root_CA.pem
/etc/ssl/certs/Entrust_Root_Certification_Authority.pem
```

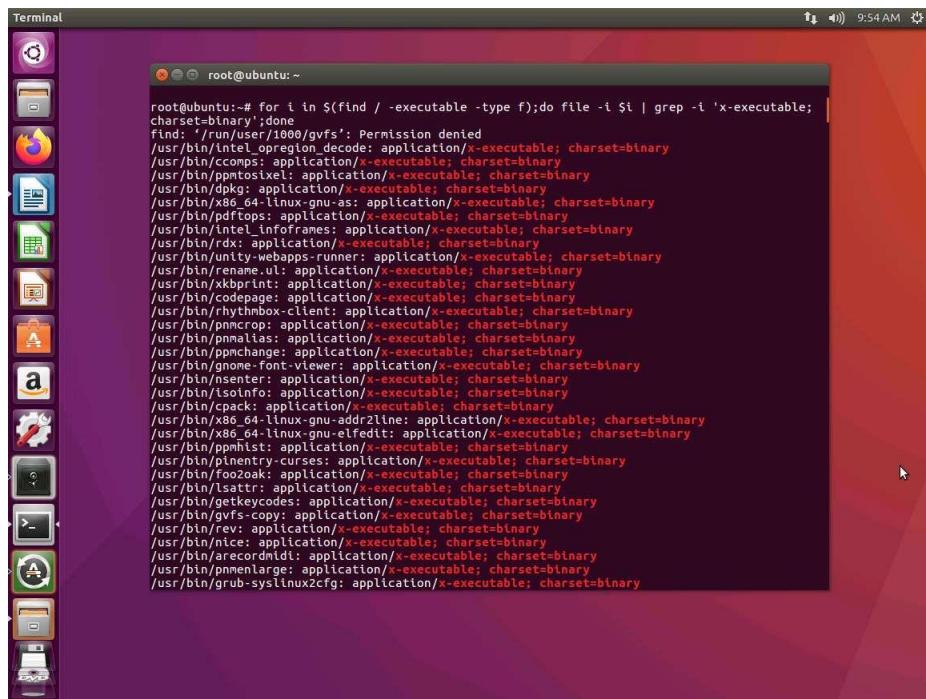
8. The command is used to update a database for the mlocate tool. We then use the locate command to hunt down any file with the word cat in the name. These tools together make it fairly easy to track down binaries written to disk once we have the name of the binary. For example, as a result of reviewing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

```

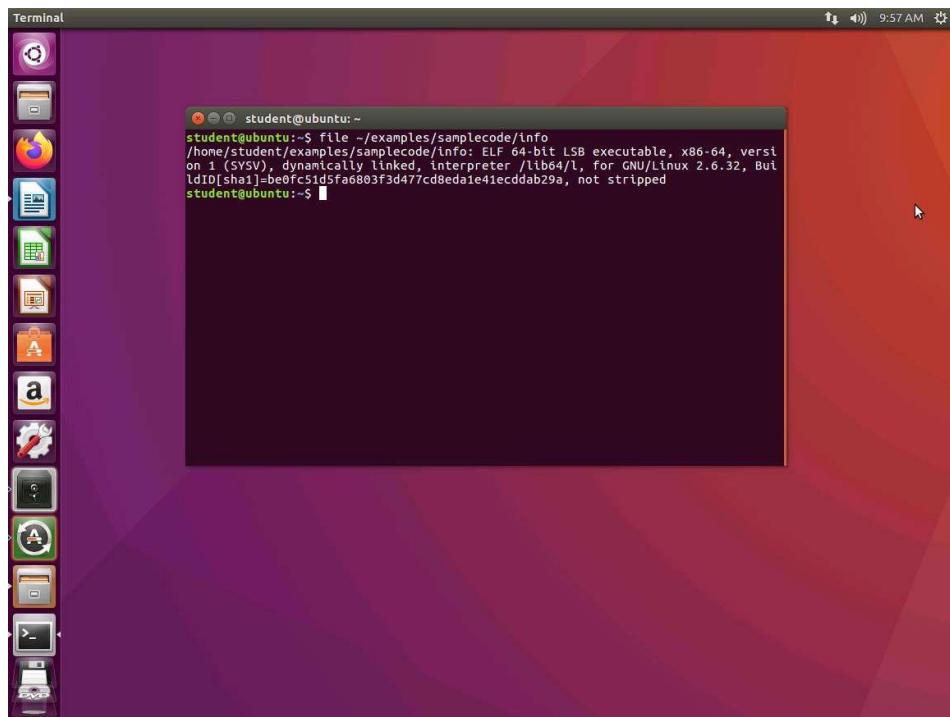
root@ubuntu:~# ps -ef
UID      PID  PPID  C STIME TTY      TIME CMD
root       1    0  0 05:55 ?    00:00:02 /sbin/init auto noprompt
root       2    0  0 05:55 ?
root       4    2  0 05:55 ?
root       6    2  0 05:55 ?
root       7    2  0 05:55 ?
root       8    2  0 05:55 ?
root       9    2  0 05:55 ?
root      10   2  0 05:55 ?
root      11   2  0 05:55 ?
root      12   2  0 05:55 ?
root      13   2  0 05:55 ?
root      14   2  0 05:55 ?
root      15   2  0 05:55 ?
root      16   2  0 05:55 ?
root      17   2  0 05:55 ?
root      18   2  0 05:55 ?
root      19   2  0 05:55 ?
root      20   2  0 05:55 ?
root      21   2  0 05:55 ?
root      22   2  0 05:55 ?
root      23   2  0 05:55 ?
root      24   2  0 05:55 ?
root      25   2  0 05:55 ?
root      26   2  0 05:55 ?
root      27   2  0 05:55 ?
root      28   2  0 05:55 ?
root      29   2  0 05:55 ?
root      30   2  0 05:55 ?
root      33   2  0 05:55 ?
root      34   2  0 05:55 ?
root      35   2  0 05:55 ?
root      77   2  0 05:55 ?
root      78   2  0 05:55 ?
root      79   2  0 05:55 ?
root      80   2  0 05:55 ?
root      81   2  0 05:55 ?
root      82   2  0 05:55 ?
root      88   2  0 05:55 ?
root      98   2  0 05:55 ?
root     115   2  0 05:55 ?
root     164   2  0 05:55 ?
root     165   2  0 05:55 ?
root     167   2  0 05:55 ?
root     169   2  0 05:55 ?

```

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.

A screenshot of a Ubuntu desktop environment. A terminal window titled 'Terminal' is open at the bottom left, showing root access. The command entered is: 'root@ubuntu:~# for i in \$(find / -executable -type f); do file -i \$i | grep -i 'x-executable; charset=binary'; done'. The output lists numerous files and their types, all of which are executable binary files. The desktop background is a red gradient, and the Unity interface is visible with its characteristic icons and dock.

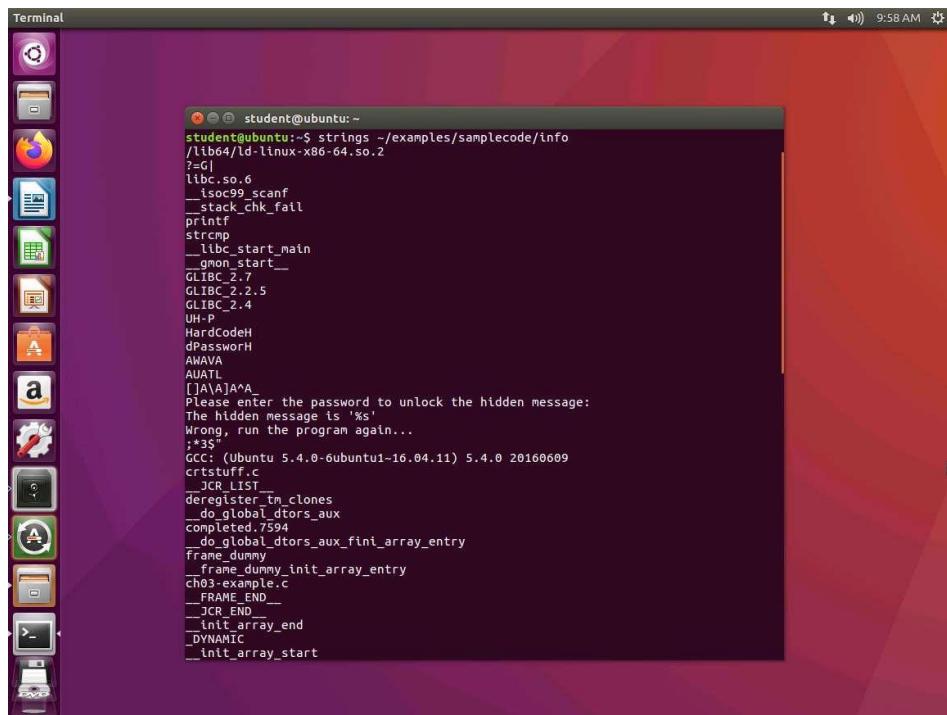
12. This command shows for every line in the output of our find command, run the file command and grep the output to only display executable binary files.
13. We are now ready for the information gathering step. Launch a new terminal window, enter **file ~examples/samplecode/info**. The output of the 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.

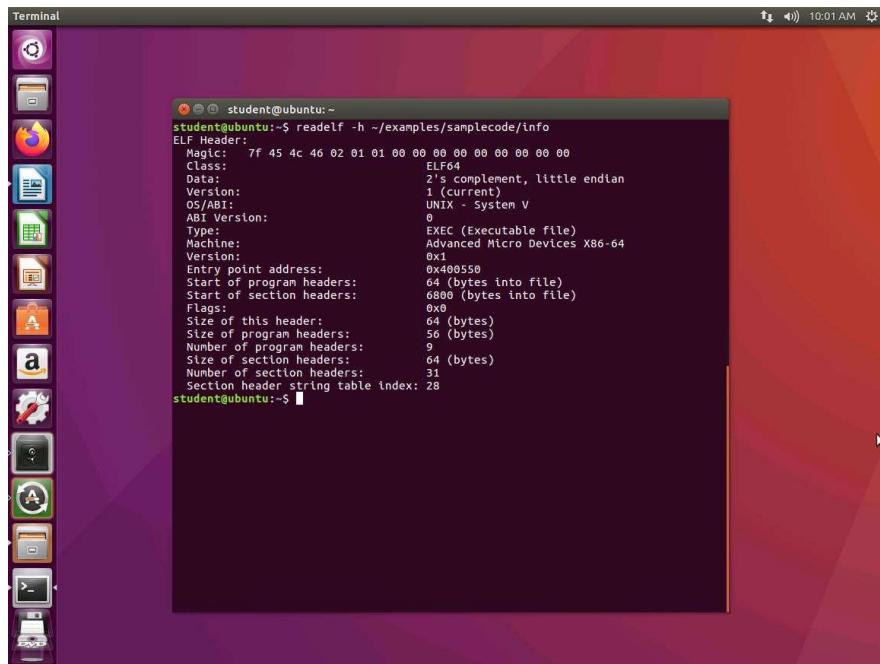
15. Next, in the terminal window, enter **strings**

~/examples/samplecode/info. The output of the command is shown in the following screenshot.

A screenshot of a Ubuntu desktop environment. A terminal window is open in the center, showing the output of the 'strings' command. The terminal window has a dark background and light-colored text. The output shows various symbols and strings from a shared library, including '_isoc99_scnaf', '_stack_chk_fail', 'printf', 'strcmp', '_libc_start_main', '_gnon_start__', 'GLIBC_2.7', 'GLIBC_2.2.5', 'GLIBC_2.4', 'UH-P', 'HardCodeH', 'dPassworH', 'AWAVA', 'AUATL', and '[JAVA]A^A'. It also includes a password prompt and a failure message: 'Please enter the password to unlock the hidden message: The hidden message is "%s" Wrong, run the program again... ;*35' followed by the compiler information 'GCC: (Ubuntu 5.4.0-6ubuntu1-16.04.11) 5.4.0 20160609 crtstuff.c' and some internal symbols like '_JCR_LIST', '_deregister_tm_clones', '_do_global_dtors_aux', '_completed_7594', '_do_global_dtors_aux_fini_array_entry', '_frame_dummy', '_frame_dummy_init_array_entry', 'ch03-example.c', '__FRAME_END__', '_JCR_END', '_init_array_end', '_DYNAMIC', and '_init_array_start'.

```
student@ubuntu:~$ strings ~/examples/samplecode/info
/lib64/ld-linux-x86-64.so.2
?_G_
 libc.so.6
 __isoc99_scnaf
 __stack_chk_fail
 printf
 strcmp
 __libc_start_main
 __gnon_start__
 GLIBC_2.7
 GLIBC_2.2.5
 GLIBC_2.4
 UH-P
 HardCodeH
 dPassworH
 AWAVA
 AUATL
 [JAVA]A^A
Please enter the password to unlock the hidden message:
The hidden message is "%s"
Wrong, run the program again...
;*35
GCC: (Ubuntu 5.4.0-6ubuntu1-16.04.11) 5.4.0 20160609
crtstuff.c
__JCR_LIST
_deregister_tm_clones
__do_global_dtors_aux
_completed_7594
__do_global_dtors_aux_fini_array_entry
_frame_dummy
__frame_dummy_init_array_entry
ch03-example.c
__FRAME_END__
__JCR_END
__init_array_end
__DYNAMIC
__init_array_start
```

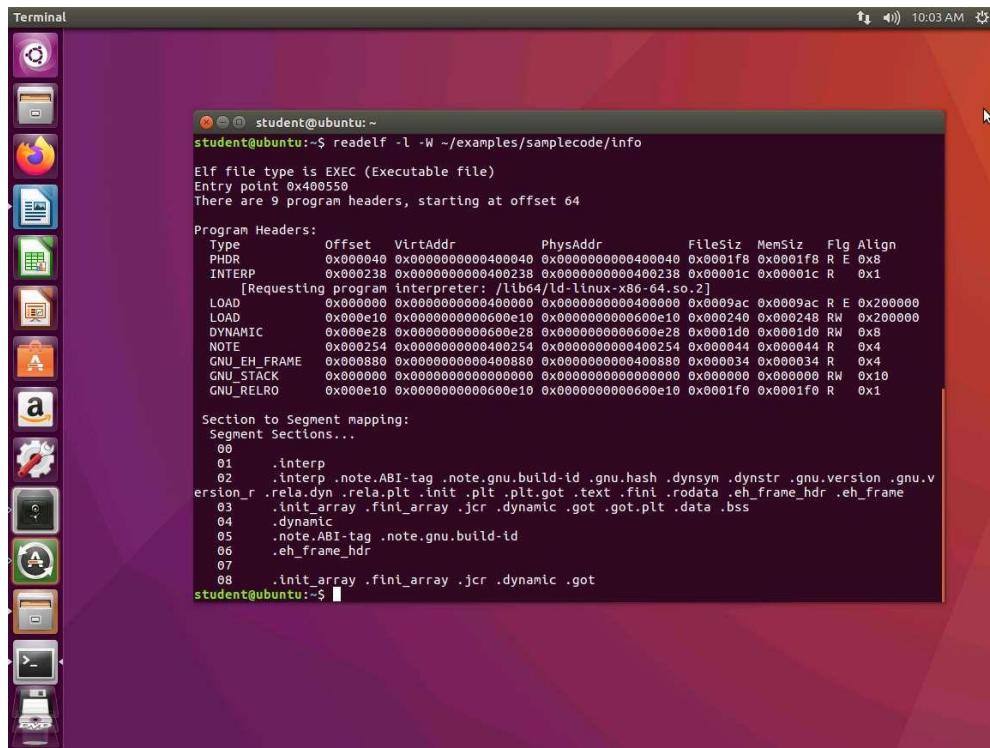
16. This output reveals what appears to be a hardcoded password, a message requesting a password, a sentence using the C-style %s format string, and what appears to be a failure message if the password is incorrect.
17. You will also notice the use of scanf, printf, and strcmp near the beginning of the 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.**



A screenshot of an Ubuntu desktop environment. A terminal window titled "Terminal" is open, displaying the output of the command `readelf -h ~/examples/samplecode/info`. The output shows the ELF Header details:

```
student@ubuntu:~$ readelf -h ~/examples/samplecode/info
ELF Header:
  Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class: ELF64
  Data: 2's complement, little endian
  Version: 1 (current)
  OS/ABI: UNIX - System V
  ABI Version: 0
  Type: EXEC (Executable file)
  Machine: Advanced Micro Devices X86-64
  Version: 0x1
  Entry point address: 0x4000550
  Start of program headers: 0x4 (bytes into file)
  Start of section headers: 0x800 (bytes into file)
  Flags: 0x8
  Size of this header: 64 (bytes)
  Size of program headers: 56 (bytes)
  Number of program headers: 9
  Size of section headers: 64 (bytes)
  Number of section headers: 31
  Section header string table index: 28
student@ubuntu:~$
```

19. The **readelf** tool is invaluable in our quest for gathering information about a binary.
20. Next, enter **readelf -I -W ~/examples/samplecode/info**. The output of the command is shown in the following screenshot.



A screenshot of a Ubuntu desktop environment. A terminal window is open in the top panel, showing the output of the command `readelf -l -W ~/examples/samplecode/info`. The terminal shows details about the ELF file's program headers, section to segment mappings, and other header information. The desktop background is a purple gradient, and various icons are visible in the dock.

```
student@ubuntu:~$ readelf -l -W ~/examples/samplecode/info
Elf file type is EXEC (Executable file)
Entry point 0x400550
There are 9 program headers, starting at offset 64
Program Headers:
Type Offset VirtAddr PhysAddr FileSiz MemSiz Flg Align
PHDR 0x000040 0x0000000000400040 0x0000000000400040 0x0001f8 0x0001f8 R E 0x8
INTERP 0x000238 0x0000000000400238 0x0000000000400238 0x00001c 0x00001c R 0x1
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
LOAD 0x000000 0x0000000000400000 0x0000000000400000 0x0009ac 0x0009ac R E 0x200000
LOAD 0x000e10 0x000000000060e10 0x000000000060e10 0x000240 0x000240 RW 0x200000
DYNAMIC 0x000e28 0x000000000060e28 0x000000000060e28 0x0001d0 0x0001d0 RW 0x8
NOTE 0x000254 0x0000000000400254 0x0000000000400254 0x000044 0x000044 R 0x4
GNU_EH_FRAME 0x000080 0x0000000000400880 0x0000000000400880 0x000034 0x000034 R 0x4
GNU_STACK 0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW 0x10
GNU_RELRO 0x0000e10 0x000000000060e10 0x000000000060e10 0x0001f0 0x0001f0 R 0x1
Section to Segment mapping:
Segment Sections...
 00
 01 .interp
 02 .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .got .text .fini .rodata .eh_frame_hdr .eh_frame
 03 .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
 04 .dynamic
 05 .note.ABI-tag .note.gnu.build-id
 06 .eh_frame_hdr
 07
 08 .init_array .fini_array .jcr .dynamic .got
```

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

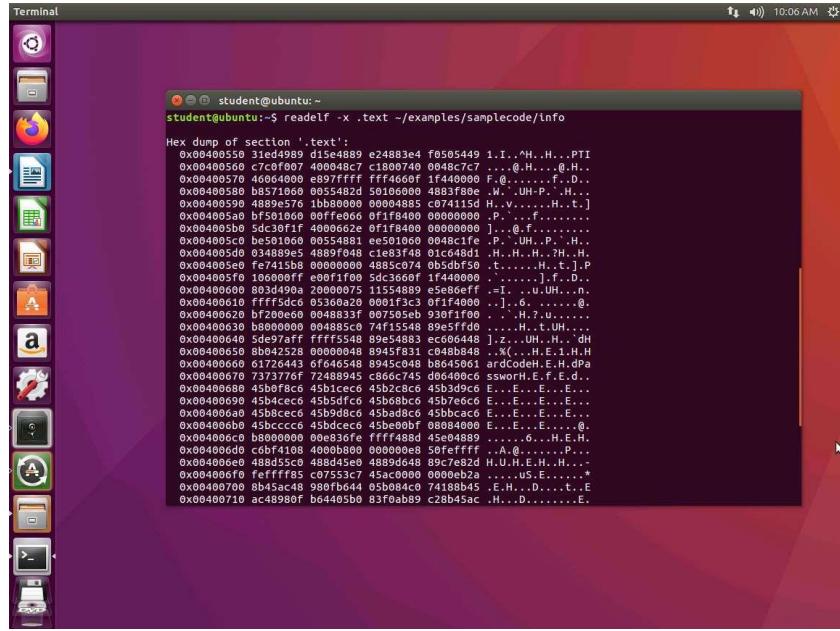
a. **readelf -S -W ~/examples/samplecode/info**

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 student@ubuntu:~$ readelf -S -W ~/examples/samplecode/elfinfo
There are 31 section headers, starting at offset 0x1a90:
Section Headers:
[Nr] Name           Type            Address          Off   Size  ES Flg Lk Inf Al
[ 0] .null         NULL            0000000000000000 000000 00000000 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 00003d 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.dyn.plt RELA            0000000000400440 000440 000018 18 A 5 24 8
[11] .init          PROGBITS        00000000004004b8 0004b8 00001a 00 AX 0 0 4
[12] .plt           PROGBITS        00000000004004e0 0004e0 000066 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        0000000000400574 000574 000009 00 AX 0 0 4
[16] .rodata        PROGBITS        0000000000400800 000800 000080 00 A 0 0 8
[17] .eh_frame_hdr PROGBITS        0000000000400888 000888 000034 00 A 0 0 4
[18] .eh_frame     PROGBITS        00000000004008b8 0008b8 0000f4 00 A 0 0 8
[19] .init_array    INIT_ARRAY     000000000060e10 000e10 000008 00 WA 0 0 8
[20] .fini_array   FINI_ARRAY    000000000060e18 000e18 000008 00 WA 0 0 8
[21] .jcr           PROGBITS        000000000060e20 000e20 000020 00 WA 0 0 8
[22] .dynamic       DYNAMIC         000000000060e28 000e28 0001d0 10 WA 6 0 8
[23] .got           PROGBITS        000000000060f8f8 000ff8 000008 08 WA 0 0 8
[24] .got.plt      PROGBITS        0000000000610000 001000 000040 08 WA 0 0 8
[25] .data          PROGBITS        0000000000651640 001040 000010 00 WA 0 0 8
```

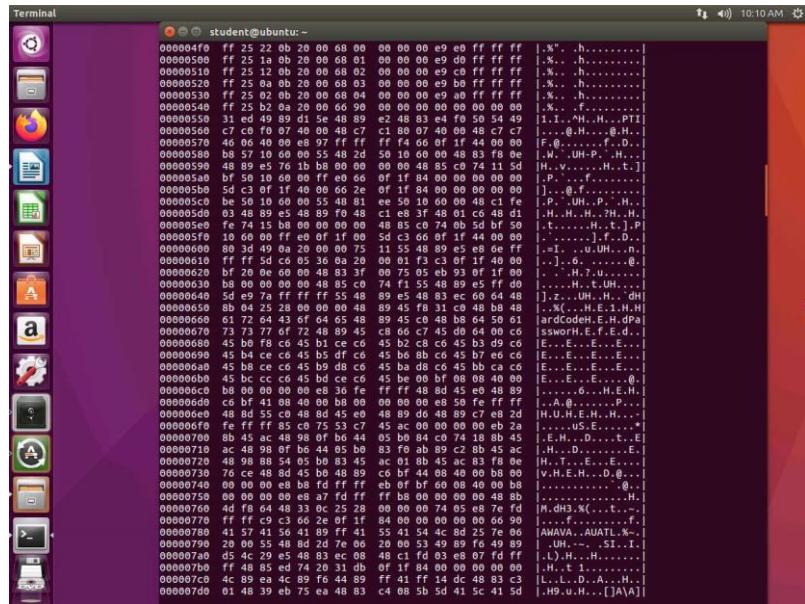


22. As the above screenshot shows, we see the hardcoded password in the dump.

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

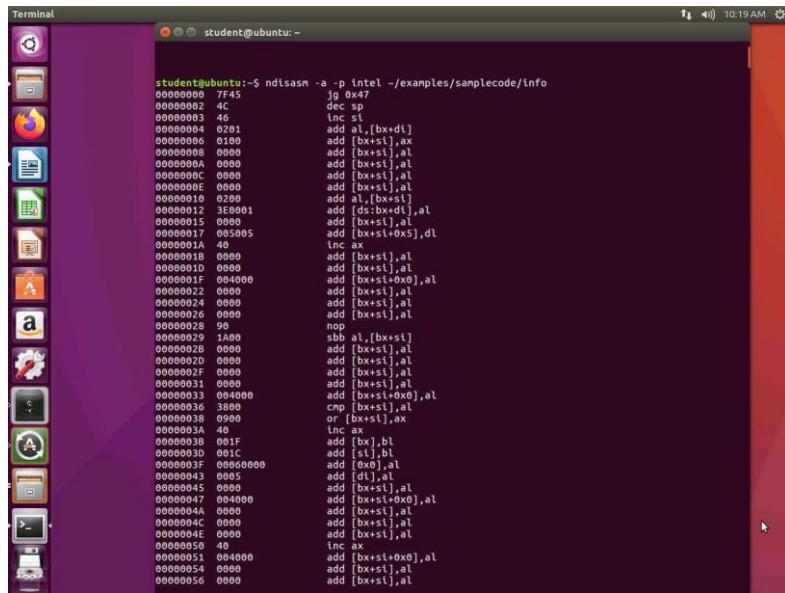
 - a. `readelf -R .text -W ~/examples/samplecode/info`
 - b. `readelf -p .strtab -W ~/examples/samplecode/info`
 - c. `objdump -f ~/examples/samplecode/info`
 - d. `objdump -j .text -s ~/examples/samplecode/info`
 - e. `objdump -x ~/examples/samplecode/info`
 - f. `hexdump -C ~/examples/samplecode/info`

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.



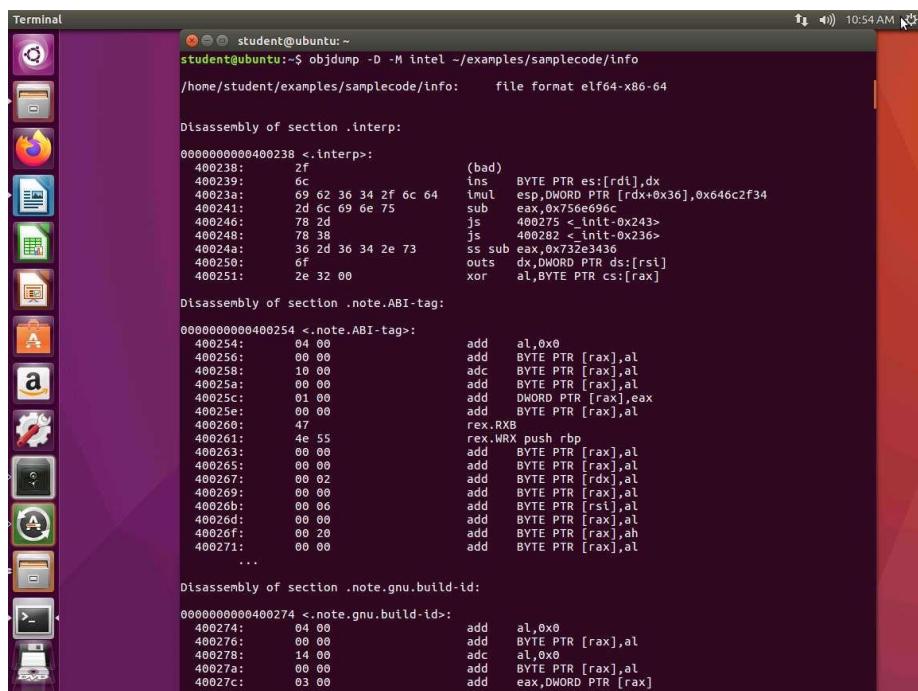
```

Terminal student@ubuntu: ~
student@ubuntu:~$ ndisasm -p intel ~/examples/samplecode/info
00000000 7f45          jg 0x47
00000000 4c             dec sp
00000003 45             neg al
00000004 0201           add al,[bx+di]
00000006 0100           add [bx+si],ax
00000008 0000           add [bx+si],al
0000000a 0000           add [bx+si],al
0000000c 0000           add [bx+si],al
0000000e 0000           add [bx+si],al
00000010 0280           add al,[bx+sl]
00000012 3E0001           add ds:[bx+dl],al
00000015 0000           add [bx+si],al
00000017 005005           add [bx+si:0x5],dl
0000001A 0000           inc al
0000001B 0000           add [bx+si],al
0000001D 0000           add [bx+si],al
0000001F 004000           add [bx+si:0x0],al
00000022 0000           add [bx+si],al
00000025 0000           add [bx+si],al
00000026 0000           add [bx+si],al
00000028 99             nop
00000029 1A00           stb al,[bx+si]
0000002B 0000           add [bx+si],al
0000002D 0000           add [bx+si],al
0000002F 0000           add [bx+si],al
00000031 0000           add [bx+si],al
00000033 004000           add [bx+si:0x0],al
00000036 3800           cm [bx+si],al
00000038 0900           or [bx+si],ax
0000003A 49             im ax
0000003B 011F           add [bx],bl
0000003D 001C           add dl,bl
0000003F 00000000           add [0x0],al
00000043 0005           add [dl],al
00000045 0000           add [bx+si],al
00000047 004000           add [bx+si:0x0],al
00000049 0000           add [bx+si],al
0000004C 0000           add [bx+si],al
0000004E 0000           add [bx+si],al
00000050 40             inc ax
00000051 004000           add [bx+si:0x0],al
00000054 0000           add [bx+si],al
00000056 0000           add [bx+si],al

```

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.



```

Terminal student@ubuntu: ~
student@ubuntu:~$ objdump -D -M intel ~/examples/samplecode/info
/home/student/examples/samplecode/info:      file format elf64-x86-64

Disassembly of section .interp:
000000000000400238 <.interp>:
400238: 2f          (bad)
400239: 6c          ins    BYTE PTR es:[rdi].dx
40023a: 69 62 36 34 2f 6c 64  imul  esp,DWORD PTR [rdx+0x36],0x646c2f34
400241: 2d 6c 69 6e 75  sub   eax,0x756e696c
400246: 78 2d          js    400275 <_init-0x243>
400248: 78 38          js    400282 <_init-0x236>
40024a: 36 2d 36 34 2e 73  ss sub eax,0x732e3436
400250: 6f          outs  dx,DWORD PTR ds:[rsi]
400251: 2e 32 00  xor   al,BYTE PTR cs:[rax]

Disassembly of section .note.ABI-tag:
000000000000400254 <.note.ABI-tag>:
400254: 04 00          add   al,0x0
400255: 00 00          add   BYTE PTR [rax],al
400256: 00 00          add   BYTE PTR [rax],al
400257: 00 00          add   BYTE PTR [rax],al
400258: 00 00          add   BYTE PTR [rax],al
400259: 01 00          add   DWORD PTR [rax],eax
40025a: 00 00          add   BYTE PTR [rax],al
40025b: 00 00          add   BYTE PTR [rax],al
40025c: 00 00          add   BYTE PTR [rax],al
40025d: 00 00          add   BYTE PTR [rax],al
40025e: 00 00          add   BYTE PTR [rax],al
40025f: 00 00          add   BYTE PTR [rax],ah
400260: 00 00          add   BYTE PTR [rax],al
...
400261: 4e 55          rex.WRX push rbp
400262: 00 00          add   al,0x0
400263: 00 00          add   BYTE PTR [rax],al
400264: 00 00          add   BYTE PTR [rax],al
400265: 00 00          add   BYTE PTR [rax],al
400266: 00 02          add   BYTE PTR [rdx],al
400267: 00 00          add   BYTE PTR [rax],al
400268: 00 00          add   BYTE PTR [rax],al
400269: 00 00          add   BYTE PTR [rax],al
40026a: 00 00          add   BYTE PTR [rsi],al
40026b: 00 00          add   BYTE PTR [rax],al
40026c: 00 00          add   BYTE PTR [rax],al
40026d: 00 00          add   BYTE PTR [rax],al
40026e: 00 20          add   BYTE PTR [rax],ah
40026f: 00 00          add   BYTE PTR [rax],al
400270: 00 00          add   BYTE PTR [rax],al
400271: 00 00          add   BYTE PTR [rax],al
...
400274: 04 00          add   al,0x0
400275: 00 00          add   BYTE PTR [rax],al
400276: 14 00          adc   al,0x0
400277: 00 00          add   BYTE PTR [rax],al
400278: 03 00          add   eax,DWORD PTR [rax]
```

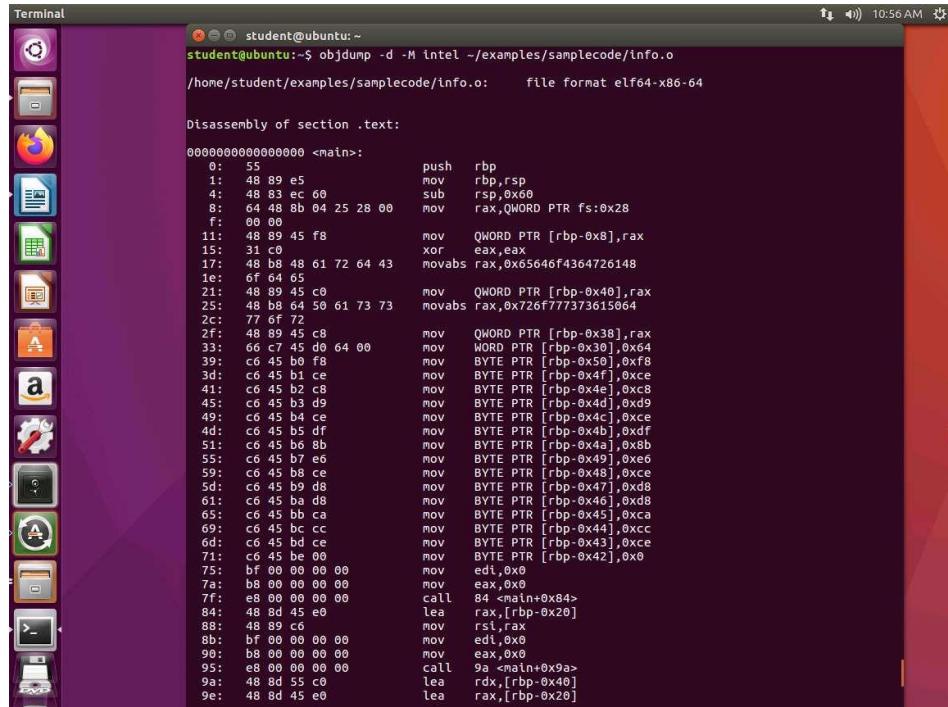
27. As a reminder, the **intel** option is to format the output in Intel and not the default of AT&T.

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.



The screenshot shows a terminal window on a Linux desktop environment. The terminal title is "Terminal" and the current user is "student@ubuntu". The command entered is "objdump -d -M intel ~/examples/samplecode/info.o". The output shows the disassembly of the ".text" section of the object file. The assembly code is presented in Intel syntax, with addresses, opcodes, and assembly mnemonics. The terminal window has a dark background with icons for various applications like Nautilus, Firefox, and LibreOffice visible on the left.

```
student@ubuntu:~$ objdump -d -M intel ~/examples/samplecode/info.o
/home/student/examples/samplecode/info.o:      file format elf64-x86-64

Disassembly of section .text:
0000000000000000 <main>:
 0: 55                      push  rbp
 1: 48 89 e5                mov    rbp,rsp
 4: 48 83 ec 60             sub    rsp,0x60
 8: 64 48 8b 04 25 28 00   mov    rax,QWORD PTR fs:0x28
 f: 00 00
11: 48 89 45 f8            mov    QWORD PTR [rbp-0x8],rax
15: 31 c0                  xor    eax,eax
17: 48 b8 48 61 72 64 43   movabs rax,0x65646f4364726148
1e: 6f 64 65
21: 48 89 45 c0            mov    QWORD PTR [rbp-0x40],rax
25: 48 b8 64 50 61 73 73   movabs rax,0x726f777373615064
2c: 77 6f 72
2f: 48 89 45 c8            mov    QWORD PTR [rbp-0x38],rax
33: 66 c7 45 d0 64 00     mov    WORD PTR [rbp-0x30],0x64
39: c6 45 b0 f8            mov    BYTE PTR [rbp-0x50],0xf8
3d: c6 45 b1 ce            mov    BYTE PTR [rbp-0x4f],0xce
41: c6 45 b2 c8            mov    BYTE PTR [rbp-0x4e],0xc8
45: c6 45 b3 d9            mov    BYTE PTR [rbp-0x4d],0xd9
49: c6 45 b4 ce            mov    BYTE PTR [rbp-0x4c],0xce
4d: c6 45 b5 df            mov    BYTE PTR [rbp-0x4b],0xdf
51: c6 45 b6 8b            mov    BYTE PTR [rbp-0x4a],0xb
55: c6 45 b7 e6            mov    BYTE PTR [rbp-0x49],0xe6
59: c6 45 b8 ce            mov    BYTE PTR [rbp-0x48],0xce
5d: c6 45 b9 d8            mov    BYTE PTR [rbp-0x47],0xd8
61: c6 45 ba d8            mov    BYTE PTR [rbp-0x46],0xd8
65: c6 45 bb ca            mov    BYTE PTR [rbp-0x45],0xca
69: c6 45 bc cc            mov    BYTE PTR [rbp-0x44],0xcc
6d: c6 45 bd ce            mov    BYTE PTR [rbp-0x43],0xce
71: c6 45 be 00            mov    BYTE PTR [rbp-0x42],0x0
75: bf 00 00 00 00          mov    edi,0x0
7a: b0 00 00 00 00          mov    eax,0x0
7f: e8 00 00 00 00          call   84 <_main+0x84>
84: 48 8d 45 e0            lea    rax,[rbp-0x20]
88: 48 89 c6                mov    rsi,rax
8b: bf 00 00 00 00          mov    edi,0x0
90: b0 00 00 00 00          mov    eax,0x0
95: e8 00 00 00 00          call   9a <_main+0x9a>
9a: 48 8d 55 c0            lea    rdx,[rbp-0x40]
9e: 48 8d 45 e0            lea    rax,[rbp-0x20]
```

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`

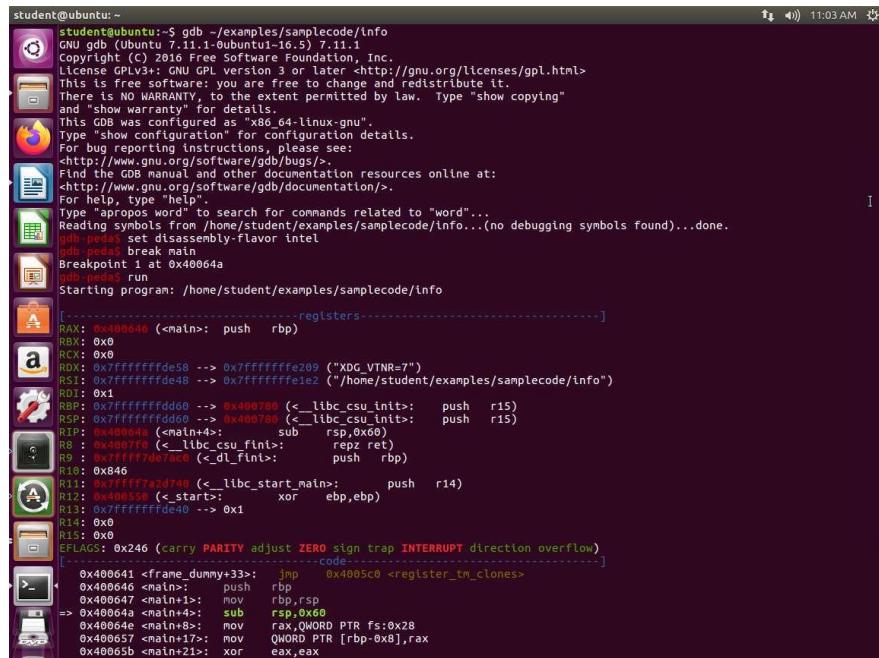
b. `set disassembly-flavor intel`

c. `break main`

d. `run`

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

following screenshot.



The screenshot shows a terminal window on an Ubuntu system. The command `gdb ~/examples/samplecode/info` is run, followed by `set disassembly-flavor intel`, `break main`, and `run`. The output shows the assembly code for the `main` function, including the instruction at address `0x40064a` which triggered a breakpoint. Registers and memory dump sections are also visible.

```
student@ubuntu:~$ gdb ~/examples/samplecode/info
GNU gdb (Ubuntu 7.11.1-0ubuntu16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from /home/student/examples/samplecode/info...(no debugging symbols found)...done.
(gdb-peda) set disassembly-flavor intel
(gdb-peda) break main
Breakpoint 1 at 0x40064a
(gdb-peda) run
Starting program: /home/student/examples/samplecode/info

[...]
RAX: 0x00000000 (<main>: push rbp)
RBX: 0x0
RCX: 0x0
RDH: 0x7fffffe209 ---> 0x7fffffe209 ("XDG_VTNR=7")
RSI: 0x7fffffe1e2 (</home/student/examples/samplecode/info>)
RDI: 0x1
RBP: 0x7fffffd6d0 ---> 0x400780 (<_libc_csu_init>: push r15)
RSP: 0x7fffffd6d0 ---> 0x400780 (<_libc_csu_init>: push r15)
RIP: 0x40064a (<main+4: sub rsp,0x60)
R8 : 0x400770 (<_libc_csu_fini>: repz ret)
R9 : 0xcffff7de7ec0 (<_dl_fini>: push rbp)
R10: 0x846
R11: 0x7ffff7a3d740 (<_libc_start_main>: push r14)
R12: 0x400550 (<_start>: xor ebp,ebp)
R13: 0x7fffffd6d0 ---> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[...]
-> 0x400641 <frame_dummy+33>: jmp 0x4005c0 <register_tm_clones>
 0x400646 <main>: push rbp
 0x400647 <main+1>: mov rbp,rsp
=> 0x400648 <main+4>: sub rsp,0x60
 0x400649 <main+8>: mov rax,QWORD PTR fs:0x28
 0x400657 <main+17>: mov QWORD PTR [rbp-0x8],rax
 0x40065b <main+21>: xor eax,eax
```

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

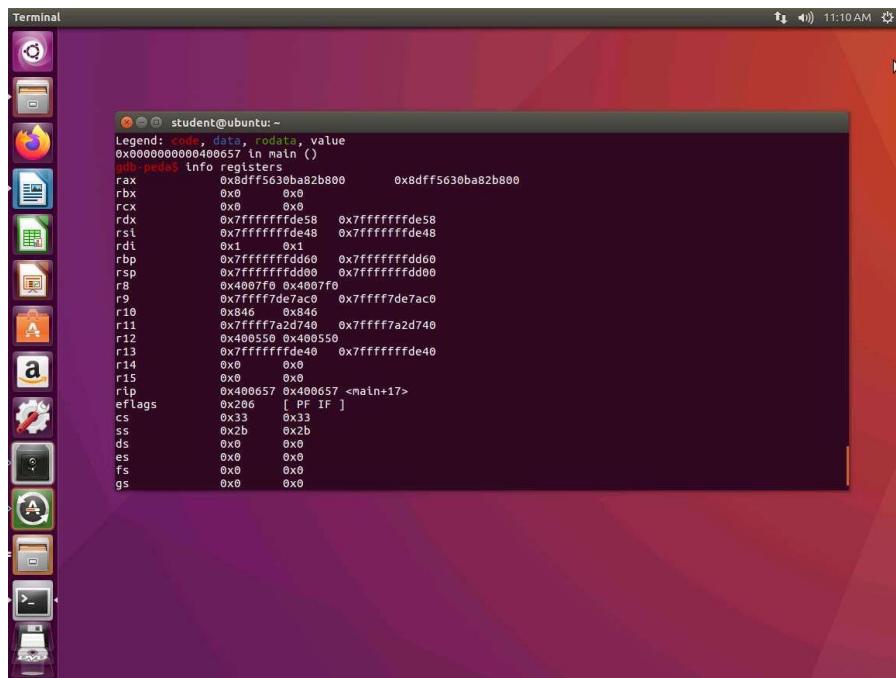
registers.

```
student@ubuntu: ~
0056 | 0x7fffffd98 --> 0x3ecb08173002d3d8
[...]
Legend: code, data, rodata, value
Breakpoint 1, 0x000000000040064a in main ()
gdb-peda$ info registers
rax            0x400646 0x400646
rbx            0x0      0x0
rcx            0x0      0x0
rdx            0x7fffffffde58 0x7fffffffde58
rsi            0x7fffffffde48 0x7fffffffde48
rdi            0x1      0x1
rbp            0x7fffffffdd60 0x7fffffffdd60
rsp            0x7fffffffdd60 0x7fffffffdd60
r8             0x4007f0 0x4007f0
r9             0x7ffff7de7ac0 0x7ffff7de7ac0
r10            0x846    0x846
r11            0x7ffff7a2d740 0x7ffff7a2d740
r12            0x400550 0x400550
r13            0x7ffff7fffe40 0x7ffff7fffe40
r14            0x0      0x0
r15            0x0      0x0
rip            0x40064a 0x40064a <main+4>
eflags          0x246   [ PF ZF IF ]
cs             0x33    0x33
ss             0x2b    0x2b
ds             0x0      0x0
es             0x0      0x0
fs             0x0      0x0
gs             0x0      0x0
gdb-peda$
```

34. Next, enter the following series of commands:

- a. nexti
- b. info registers
- c. nexti
- d. info registers

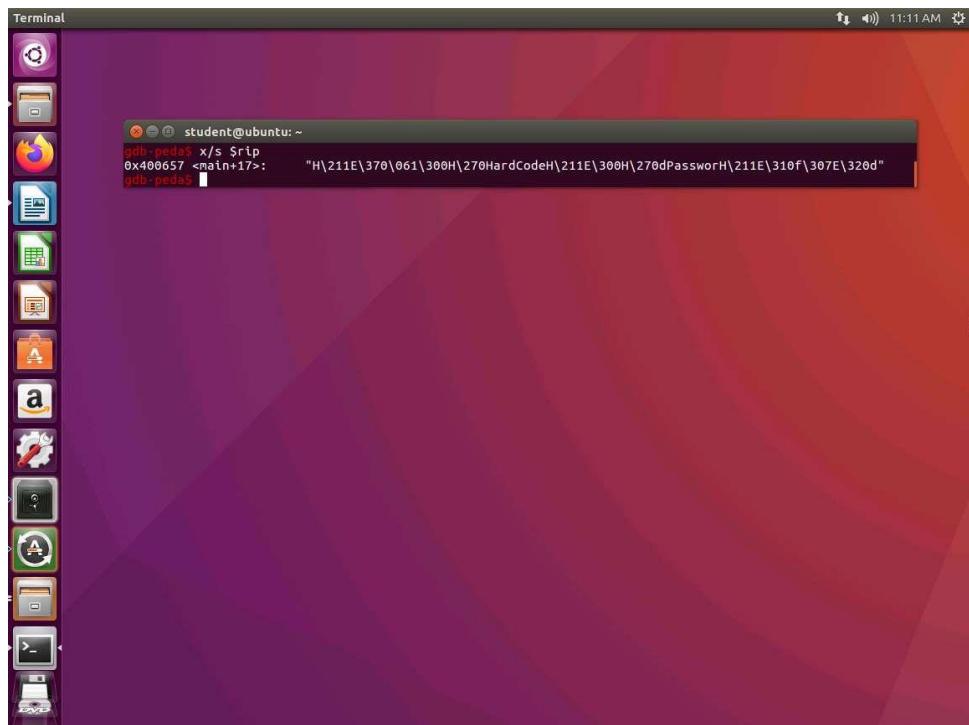
An example of the output is shown in the following screenshot.



A screenshot of a Ubuntu desktop environment. A terminal window is open, showing the output of a GDB command. The command 'info registers' is run, displaying the values of various CPU registers. The output includes:

```
Legend: code, data, rodata, value
0x000000000000400657 in main ()
gdb-peda$ info registers
rax            0x8dff5630ba82b800      0x8dff5630ba82b800
rbx            0x0            0x0
rcx            0x0            0x0
rdx            0xfffffffffdde58  0x7fffffffde58
rsi            0xfffffffffdde48  0x7fffffffde48
rdi            0x1            0x1
rbp            0x7fffffd4d68  0x7fffffd4d68
rsp            0x7fffffd4d80  0x7fffffd4d80
r8             0x4007f0  0x4007f0
r9             0x7ffff7de7ac0  0x7ffff7de7ac0
r10            0x846          0x846
r11            0x7ffff7a2d740  0x7ffff7a2d740
r12            0x400550  0x400550
r13            0x7fffffd4e40  0x7fffffd4e40
r14            0x0            0x0
r15            0x0            0x0
rip            0x400657  0x400657 <main+17>
eflags          0x266  [ PF IF ]
cs             0x31          0x31
ss             0x2b          0x2b
ds             0x6            0x6
es             0x0            0x0
fs             0x0            0x0
gs             0x0            0x0
```

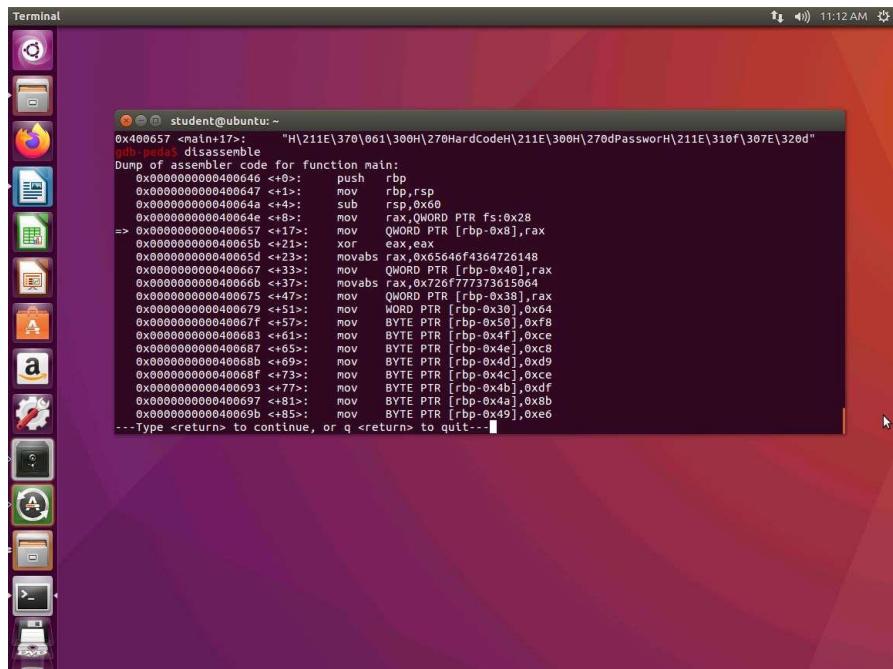
35. Enter the **x/s \$rip** command, if you are using the 64-bit system. Essentially, this final command tells GDB we want to examine the RIP register and display the output as a string.



A screenshot of a Ubuntu desktop environment. A terminal window is open, showing the output of the 'x/s \$rip' command. The command is run at address 0x400657, which is the value of the RIP register. The output shows the memory starting at the RIP address containing the string "H\211E\370\061\300H\270HardCodeH\211E\300H\270dPassworH\211E\310f\307E\320d".

```
Terminal
student@ubuntu:~$ gdb-peda x/s $rip
0x400657 <main+17>:      "H\211E\370\061\300H\270HardCodeH\211E\300H\270dPassworH\211E\310f\307E\320d"
gdb-peda$
```

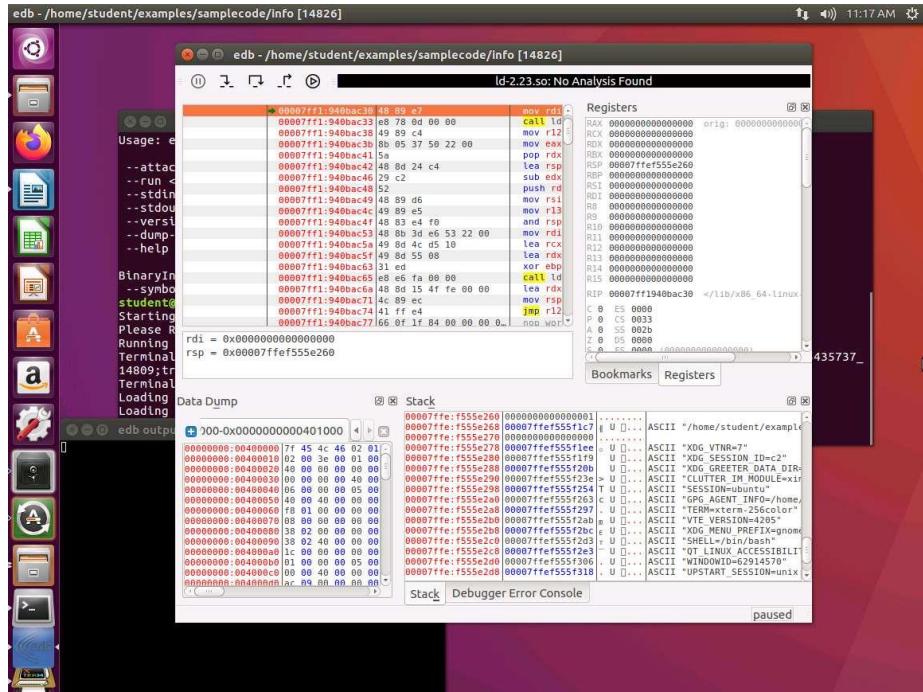
36. Next, enter disassemble. Take a few minutes to review the output of this command.



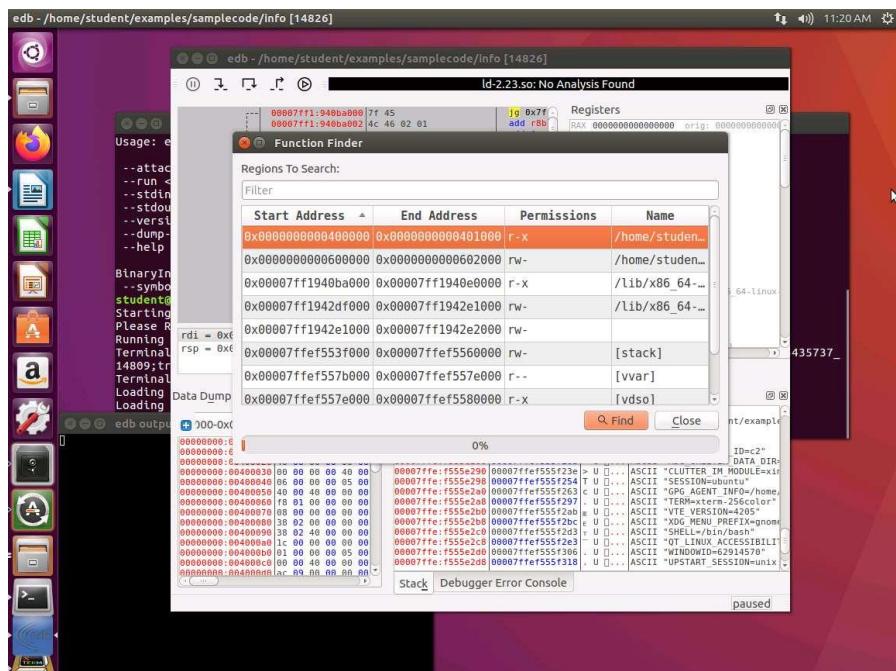
A screenshot of a Linux desktop environment with a purple gradient background. A terminal window titled "Terminal" is open, showing assembly code for a function named "main". The code is displayed in a monospaced font. The terminal window has a dark background with light-colored text. The title bar shows the path: "/211E\370\061\300H\270HardCodeH\211E\300H\270dPassworH\211E\310f\307E\320d". The status bar at the top right indicates the time is 11:12 AM. The desktop interface includes a dock with various icons like a browser, file manager, and system tools.

```
student@ubuntu:~$ ./main <1>
0x400657 <main+17>:    "H\211E\370\061\300H\270HardCodeH\211E\300H\270dPassworH\211E\310f\307E\320d"
gdb-peda$ disassemble
Dump of assembler code for function main:
0x0000000000400646 <+0>: push rbp
0x0000000000400647 <+1>: mov rbp,rsp
0x0000000000400648 <+4>: sub rsp,0x60
0x0000000000400649 <+8>: mov rax,QWORD PTR fs:0x28
=> 0x0000000000400657 <+17>: mov QWORD PTR [rbp-0x8],rax
0x000000000040065b <+21>: xor eax,eax
0x000000000040065d <+23>: movabs rax,0x65046f4364726148
0x0000000000400667 <+33>: mov QWORD PTR [rbp-0x40],rax
0x0000000000400668 <+37>: movabs rax,0x726f777373615064
0x0000000000400675 <+47>: mov QWORD PTR [rbp-0x38],rax
0x0000000000400679 <+51>: mov WORD PTR [rbp-0x30],0x64
0x000000000040067f <+57>: mov BYTE PTR [rbp-0x50],0xf8
0x0000000000400681 <+61>: mov BYTE PTR [rbp-0x4f],0xce
0x0000000000400687 <+65>: mov BYTE PTR [rbp-0x4e],0xc8
0x000000000040068b <+69>: mov BYTE PTR [rbp-0x4d],0xd9
0x000000000040068f <+73>: mov BYTE PTR [rbp-0x4c],0xce
0x0000000000400693 <+77>: mov BYTE PTR [rbp-0x4b],0xdf
0x0000000000400697 <+81>: mov BYTE PTR [rbp-0x4a],0xb
0x000000000040069b <+85>: mov BYTE PTR [rbp-0x49],0xe6
--Type <return> to continue, or q <return> to quit--
```

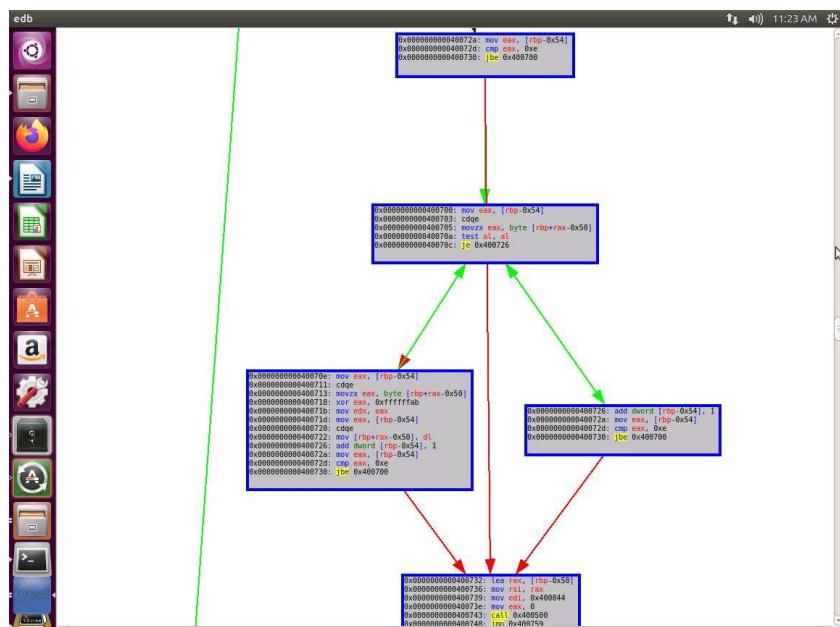
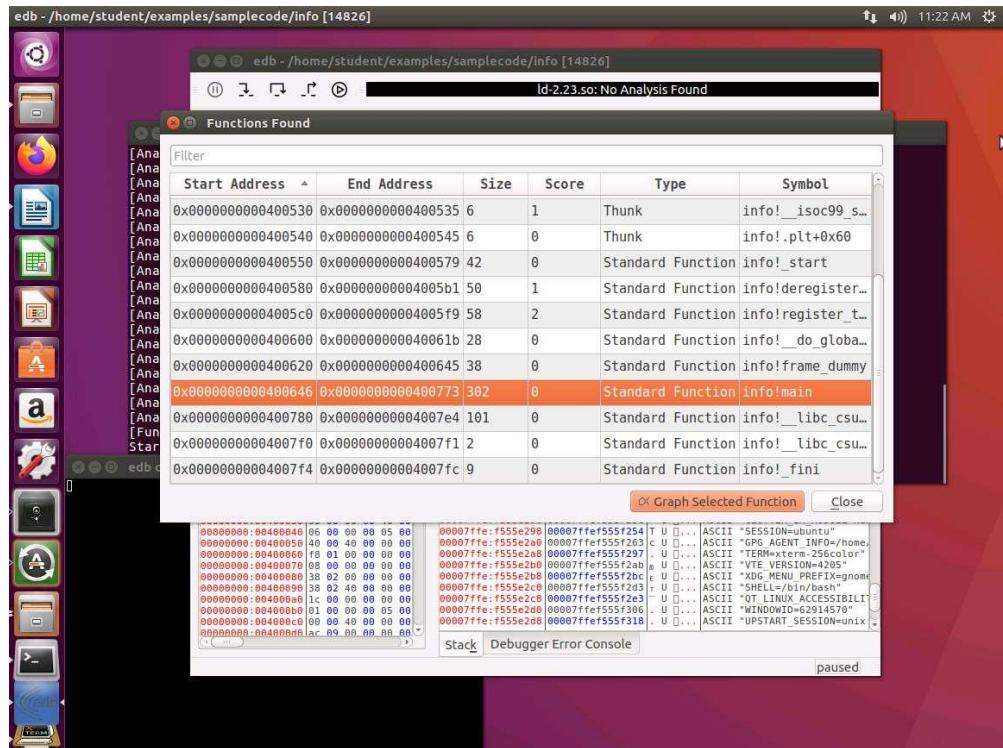
37. We are now ready to move to the next step. Enter quit. This will exit the debugger. We will try another. Enter **edb --run ~./examples/samplecode/info**. This will launch Evans Debugger as shown in the following screenshot.



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.

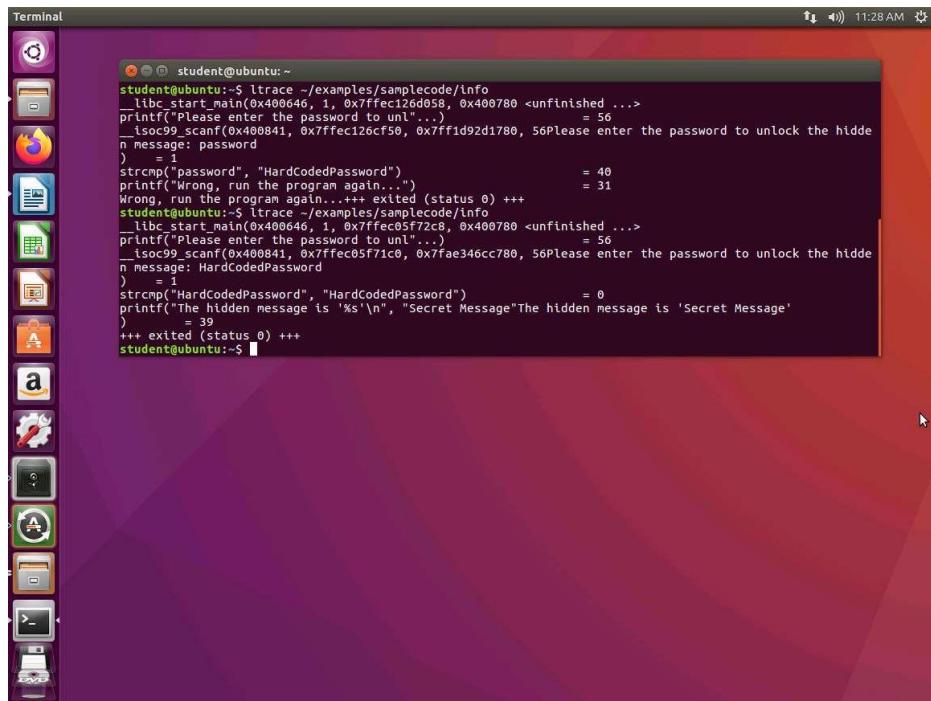


39. Next, click on the **Find** button, and look for the info main function in the symbol table. Click on it, and then select **Graph Selected Function**. Take a few minutes to review the results from the graphing of the function.



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.



A screenshot of a Ubuntu desktop environment. A terminal window titled "Terminal" is open, showing the command "ltrace -e ./examples/samplecode/info" being run. The terminal output shows a password prompt and a hardcoded password check. The desktop background is a purple gradient.

```
student@ubuntu:~$ ltrace -e ./examples/samplecode/info
__libc_start_main(0x400646, 1, 0x7ffec126d058, 0x400780 <unfinished ...>
printf("Please enter the password to unl"...)=56
_isoc99_scanf(0x400841, 0x7ffec126cf50, 0x7ff1d92d1780, 56)Please enter the password to unlock the hidde
n message: password
)=1
strcmp("password", "HardCodedPassword")=40
printf("Wrong, run the program again...")=31
Wrong, run the program again...+++ exited (status 0) +++
student@ubuntu:~$ ltrace -e ./examples/samplecode/info
__libc_start_main(0x400646, 1, 0x7ffec05f72c8, 0x400780 <unfinished ...>
printf("Please enter the password to unl"...)=56
_isoc99_scanf(0x400841, 0x7ffec05f71c0, 0x7fae346cc780, 56)Please enter the password to unlock the hidde
n message: HardCodedPassword
)=1
strcmp("HardCodedPassword", "HardCodedPassword")=0
printf("The hidden message is '%s'\n", "Secret Message")The hidden message is 'Secret Message'
)=39
+++ exited (status 0) +++
student@ubuntu:~$
```

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`

```

student@ubuntu:~$ readelf -e -W ~/examples/samplecode/info2
ELF Header:
  Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class: ELF64
  Data: Z's complement, little endian
  Version: 1 (current)
  OS/ABI: UNIX - System V
  ABI Version: 0
  Type: EXEC (Executable file)
  Machine: Advanced Micro Devices X86-64
  Version: 0x1
  Entry point address: 0x4000090
  Start of program headers: 04 (bytes into file)
  Start of section headers: 0760 (bytes into file)
  Flags: 0x0
  Size of this header: 64 (bytes)
  Size of program headers: 56 (bytes)
  Number of program headers: 9
  Size of section headers: 64 (bytes)
  Number of section headers: 31
  Section header string table index: 28

Section Headers:
[Nr] Name           Type      Address     Off   Size  ES Flg Lk Inf Al
[ 0] .NULL          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 000030 00 A 5 0 8
[ 5] .dynsym        DYNSYM 00000000004002d8 0002d8 000160 18 A 0 1 8
[ 6] .rela.dyn      RELA    0000000000400340 000340 000040 00 A 0 0 1
[ 7] .gnu.version   VERSION 0000000000400516 000516 00001c 02 A 5 0 2
[ 8] .gnu.version_r VERNEED 0000000000400538 000538 000030 00 A 6 1 8
[ 9] .rela.plt      RELA    0000000000400568 000568 000018 18 A 5 0 8
[10] .rela.plt      RELA    0000000000400580 000580 000078 18 A 5 24 8
[11] .init          PROGBITS 00000000004005f8 0005f8 00001a 00 AX 0 0 4
[12] .plt           PROGBITS 0000000000400620 000620 000060 10 AX 0 0 16

```

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

```

student@ubuntu:~$ objdump -d -M intel ~/examples/samplecode/info2
/home/student/examples/samplecode/info2:      file format elf64-x86-64

Disassembly of section .init:
00000000004005f8 <_init>:
4005f8: 48 83 ec 08      sub    rsp,0x8
4005fc: 48 8b 05 f5 09 20 00  mov    rax,QWORD PTR [rip+0x2009f5]      # 600ff8 <_DYNAMIC+0x
1e0>
400603: 48 85 c0      test   rax,rax
400606: 74 05      je    40060d <.init+0x15>
400608: e8 73 00 00 00  call   400680 <unhideMe@plt+0x10>
40060d: 48 83 c4 08  add    rsp,0x8
400611: c3      ret

Disassembly of section .plt:
0000000000400620 <hideMe@plt-0x10>:
400620: ff 35 e2 09 20 00  push   QWORD PTR [rip+0x2009e2]      # 601008 <_GLOBAL_OFFSET_
TABLE _0x8>
400626: ff 25 e4 09 20 00  jmp    QWORD PTR [rip+0x2009e4]      # 601010 <_GLOBAL_OFFSET_
TABLE _0x10>
40062c: 0f 1f 40 00  nop    DWORD PTR [rax+0x0]

0000000000400630 <hideMe@plt>:
400630: ff 25 e2 09 20 00  jmp    QWORD PTR [rip+0x2009e2]      # 601018 <_GLOBAL_OFFSET_
TABLE _0x18>
400636: 68 00 00 00 00  push   0x0
40063b: e9 e8 ff ff ff  jmp    400620 <_init+0x28>

0000000000400640 <_stack_chk_fail@plt>:
400640: ff 25 da 09 20 00  jmp    QWORD PTR [rip+0x2009da]      # 601020 <_GLOBAL_OFFSET_
TABLE _0x20>
400646: 68 01 00 00 00  push   0x1
40064b: e9 d0 ff ff ff  jmp    400620 <_init+0x28>

```

c. ltrace ~/examples/samplecode/info2

d. strace ~examples/samplecode/info2

```
Terminal 11:32 AM
student@ubuntu:~$ strace -f ./examples/samplecode/info2o
execve("./home/student/examples/samplecode/info2o", ["./home/student/examples/samplecode"...], /* 68 vars * /) = 0
brk(NULL)                                = 0x15107000
access("/etc/ld.so.nohwcap", F_OK)        = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", F_OK)         = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=51184, st_size=101554, ...}) = 0
read(3, "ELF 1.0.0 x86_64 ..., 0) = 0x7faf1e53e2000
close(3)                                   = 0
access("/etc/ld.so.nohwcap", F_OK)        = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/tls/x86_64/liblhidher.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/lib/x86_64-linux-gnu/tls/liblhidher.so", {st_mode=51184, st_size=101554, ...}) = 0
open("/lib/x86_64-linux-gnu/tls/liblhidher.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory )
stat("/lib/x86_64-linux-gnu/tls", 0x7ffc1847bc0e) = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/tls", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or direct
stat("/lib/x86_64-linux-gnu/x86_64", 0x7ffc1847bc0e) = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/liblhidher.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/lib/x86_64-linux-gnu", {st_mode=51184, st_size=16384, ...}) = 0
open("/usr/lib/x86_64-linux-gnu/x86_64/liblhidher.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file o
r directory)
stat("/usr/lib/x86_64-linux-gnu/tls/x86_64", 0x7ffc1847bc0e) = -1 ENOENT (No such file or directory)
open("/usr/lib/x86_64-linux-gnu/tls/liblhidher.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or direc
tory)
stat("/usr/lib/x86_64-linux-gnu/tls", 0x7ffc1847bc0e) = -1 ENOENT (No such file or directory)
open("/usr/lib/x86_64-linux-gnu/tls", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or direct
stat("/usr/lib/x86_64-linux-gnu/x86_64", 0x7ffc1847bc0e) = -1 ENOENT (No such file or directory)
open("/usr/lib/x86_64-linux-gnu/liblhidher.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory
)
stat("/usr/lib/x86_64-linux-gnu", {st_mode=51184, st_size=9632, ...}) = 0
open("/lib/tls/x86_64/liblhidher.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/lib/tls/x86_64", 0xffffc1847bc0e) = -1 ENOENT (No such file or directory)
open("/lib/tls/liblhidher.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/lib/tls", 0x7ffc1847bc0e) = -1 ENOENT (No such file or directory)
stat("/lib/x86_64", 0x7ffc1847bc0e) = -1 ENOENT (No such file or directory)
open("/lib/liblhidher.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
stat("/lib", {st_mode=51184, st_size=4096, ...}) = 0
```

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