**Experiment 15 Analyze binary file in Linux**

* **File**

This will be your starting point for binary analysis. We work with files daily. Not everything is an executable type; there is a whole wide range of file types out there. Before you start, you need to understand the type of file that is being analyzed. Is it a binary file, a library file, an ASCII text file, a video file, a picture file, a PDF, a data file, etc.?

virtual-machine:~$ file /bin/ls

* ldd command comes into the picture. Running it against a dynamically linked binary shows all its dependent libraries and their paths.

virtual-machine:~$ldd /bin/ls

* ltrace command displays all the functions that are being called at run time from the library. In the below example, you can see the function names being called, along with the arguments being passed to that function. You can also see what was returned by those functions on the far right side of the output.

virtual-machine:~$ltrace ls

* **Hexdump** helps you see what exactly the file contains. You can also choose to see the ASCII representation of the data present in the file using some command-line options.

virtual-machine:~$hexdump -C /bin/ls | head

* When software is being developed, a variety of text/ASCII messages are added to it, like printing info messages, debugging info, help messages, errors, and so on. Provided all this information is present in the binary, it will be dumped to screen using strings.

virtual-machine:~$strings /bin/ls

* ELF (Executable and Linkable File Format) is the dominant file format for executable or binaries, not just on Linux but a variety of UNIX systems as well. If you have utilized tools like file command, which tells you that the file is in ELF format, the next logical step will be to use the readelf command and its various options to analyze the file further.

virtual-machine:~$readelf -h /bin/ls

* objdump utility reads the binary or executable file and dumps the assembly language instructions on the screen. Knowledge of assembly is critical to understand the output of the objdump command.

virtual-machine:~/OS$ objdump -d /bin/ls | head

* The strace utility traces system calls. System calls are how you interface with the kernel to get work done

virtual-machine:~$strace -f /bin/ls

* The nm command will provide you with the valuable information that was embedded in the binary during compilation. nm can help you identify variables and functions from the binary

jvirtual-machine:~/OS$ nm pipe | tail

* gdb is the defacto debugger. It helps you load a program, set breakpoints at specific places, analyze memory and CPU register, and do much more. It complements the other tools mentioned above and allows you to do much more runtime analysis.

virtual-machine:~/OS$ gdb -q ./pipe

Reading symbols from ./pipe...

(No debugging symbols found in ./pipe)

(gdb) break main

Breakpoint 1 at 0x1209

(gdb) info break

Num Type Disp Enb Address What

1 breakpoint keep y 0x0000000000001209 <main>

(gdb) run

Starting program: /home/joseph/OS/pipe

Breakpoint 1, 0x0000555555555209 in main ()

(gdb) bt

#0 0x0000555555555209 in main ()

(gdb) c

Continuing.

[Detaching after fork from child process 30654]

Parent Passing value to child

Child printing received value

hello

[Inferior 1 (process 30646) exited normally]

(gdb) q