**Computer Security Lab 2**

Well here we are in the 2nd Lab of Computer Security 1. I kind of just hit the ground running in the last Lab and we got into some commands. In this Lab we are going to get into some nitty gritty details about Operating Systems, Kernels and other stuff. So basically how Linux functions. This Lab is going to be heavy in the theory towards the mid-point of this Lab there will most likely be some hands on Ubuntu but to really get the full grasp of how Linux works this is a necessary topic.

**The kernel and the shell**

The Linux Operating system functions off two major components the Linux kernel and the shell. The Linux kernel is the core of the Linux operating system it is a computer program which interacts with the actual hardware in machine language(machine language is just machine code which is just a set of instructions written in binary/hexadecimal that a computer responds to). The Linux kernel is written in the C-programming language. The kernel’s responsibilities are to: manage computer memory, manage file systems, control access to the computer, handle errors to perform IP services, calculate the resources of the computer for all the users (resources meaning peripherals like: keyboards, disk drivers, mouse, printers and network adapters). Effectively a kernel is nothing more than a resource manager something that manages the following resources listed above.

**Shells**

The shell is an interface **(an interface is the place where two systems meet AKA one system being the Shell and another being the kernel)** to the kernel. We (users) input commands though the shell and the kernel receives the tasks form the shell and performs them. The shell always does a certain 4 step regiment of jobs: (1) It always displays a prompt, (2) reads a command, (3) process the given command and (4) then executes the command. There are two main types of shells used today: C-shell/csh (C shell) and Bash-shell (bash). Throughout this course we are going to use Bash.

**Linux Kernel**

We understand that the kernel is a piece of code written by someone that sits in the operating system and manages the resources. But what is the Linux Kernel and what is it made up of.

At a high level view when we look at Linux we think of it just as an operating system. But it is much more than that. The fundamental architecture for how the Linux Operating System works is shown below:



**Figure 2.1:** A high level view of the Linux Kernel.

At the top of this is the user or Application space. This is where the user’s applications are executed. Below the user space is the GNU C library (glibc) this provides the system call interface or (a location that connects to the kernel) and provides a mechanism to transition between the user space application and the kernel. At the Kernel space lies the formation of the Linux kernel. We will get into more of this later on throughout this lab and in other labs.

**Subsystems of the Linux Kernel**

In the Linux Kernel there are some major components which are always being used but we don’t realize that they are because to us it might be the click of a button or an input in of a command. In this section of the Lab we are going to deal with an architectural perspective of the Linux Kernel which is shown below:

System Call Interface (SCI)

Memory Management

Network Stack

Virtual File System(VFS)

Process Management

Device Drivers (like the Floppy disk driver, Ethernet cord drivers, and Integrated Digital Electronic hard disk drivers)

 **Figure 2.2:** A high level view of the Subsystems working within the Linux Kernel.

This on a high level is how the Linux operating system functions we will get into more nitty gritty details about this in a second. I just want to mention that most operating systems run somewhat with the same structure in mind just the code and other features are different.

Without further ado let’s break down this Operating system figure above. As you might have guessed in this section we are exploring about operating systems and how Ubuntu uses them.

**System Call interface**

The System Call Interface (SCI) is the first layer we are going to talk about. In this layer we are able to perform functions calls **(a function is just a set of code to perform a certain task. So a function call is the calling of that function to perform said task)** from the user space **(the user space is everything else outside the kernel)** into the kernel.

 For instance if a user requires access to RAM or another type of hardware resource.

1. The program must ask the kernel to provide access to that resource this is called a system call.
2. When a program makes a system call the mode is switched from user mode to kernel mode this is called a **context switch**.
3. Then the kernel provides the resource which the program requested. After that another **context switch** occurs which changes the kernel mode back to user mode.

Let’s clear something up the user space and the kernel space are two different things. **The user space** is a set of locations where normal user processes run (everything other than the kernel). The role of the **kernel** as we know is to manage applications running in this space. The **kernel space** which is the location where the code of the kernel is it has full access to hardware and system resources. (I know there are a lot of concepts in this section just take it slow and re-read anything if you need to)

System calls are made by the user level programs while performing the following situations:

* Creating, opening, deleting or closing files in the file system
* Creating and managing new processes
* Creating a connection in the network to receive and send packets
* Requesting access to a hardware device like a printer.

We will not be getting into the programming of the system calls (if you would like to try them for yourself you need to learn a little C programming). The only thing we are doing is learn the concepts.

**Process Management**

Before we get into all about process management let’s define a few words which you might be unsure of. The first one is going to be “**Process**”. We know what a process is outside of the realm of operating systems. But when we are talking about them in an operating systems sense we are referring to an action in the software or a command which executes a certain action.

An example of this is if we run a command in the terminal. That command now becomes a **process** inside the terminal which we can view. It is important to note that we can only view the process of this command for the duration of the command.

Another term to define is a **thread**. Threads are an execution within a process. Typically it is an execution of a set of code within the process to lower the time the program takes to run.

The Linux Process Management system goes through everything we stated above to create a process. Whenever a process is created by you or Linux the process is shown by the “**ps or top commands” (we will see these later in lab 5).**

**Memory Management**

 Another important resource that is managed by the kernel is the **memory**. Due to efficiency reasons the virtual memory is managed in what we call **pages** or (4KB sizes). Linux includes the means to manage available memory as well as the hardware mechanisms for both the physical and virtual mappings.

In memory management there are **blocks** of data. (Think of a block as a chunk of memory) Each block of data is 4KB or 4096 bytes. When data is written to a file that is contained in a file system the operating system needs to allocate blocks of storage to contain the data that will be written to the file.

If the above statement doesn’t make think of a memory management system as a filing cabinet. Every filing cabinet has about 4 drawers. Each draw can represent one block of data. Once all 4 drawers are filled then we have 4 blocks of data filled. So that means there is 16kb of data in the filing cabinet or 16,384 bytes.

In short the **memory management** handles and manages primary memory (or memory accessed directly by the CPU) or secondary memory (memory that is going to be kept on a long term basis) and moves processes back and forth between the main memory and the disk during execution. The memory management system determines also how memory is allocated and received. For now this is all I’m going to say about this.

**Virtual File System**

The virtual file system is another aspect of the Linux Kernel it provides a common interface for file systems. The virtual file system provides a switching layer between the System call interface and the file systems supported by the kernel. Am image which illustrates this concept is shown below:



Let’s break down the figure above we first have the virtual file system which is an interface for the kernel and the file system. The purpose of the Virtual File System is to allow applications access different file systems.

At the file system functions we have functions which are defining how the file system works which we are trying to access. For instance we have the NTFS (for Microsoft windows this is their file system), FAT 32 file systems (for old Linux systems) and the extended file system which is the up to date standard for Linux.

 Below the file system functions is the buffer cache which provides a set of functions to the file system layer. In this buffer cache layer we optimize access to the physical devices by keeping data around for a limited time.

Below the buffer cache are the device drivers which implement the interface for the physical devices which we are using.

**Network Stack**

The network/protocol stack by design is an implementation of computer networking protocols. We know that the Internet Protocol (IP) is the principal communications protocol that governs the way data is sent over the internet. The protocols that are typically known are Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).

**TCP**

TCP also known as the Transmission control Protocol. It is a very common protocol on the internet. TCP is also a connection oriented stream over an IP network. It guarantees that all sent packets will reach the destination perhaps not in the correct order but marked with a number so that device can re order them. In addition the TCP network protocol sends acknowledgement packets back to the sender. This addition Acknowledgement causes delays and is less efficient for TCP. There are some other minor details but the big one is that TCP sends an Acknowledgement back to the host saying that it has been received the data. Also in reference to TCP there is a three way handshake protocol which is used when setting up two hosts. The following diagram below shows the three way handshake:

Host A sends a TCP “SYN”chronize Packet to Host B. Host B receives A’s SYN

Host A

Host B

 \

Host A sends an Acknowledgment. Then Host B receives the ACK and the TCP Socket connection is established.

Host B send’s a SYNchronize-“ACK”nowledgement. Host A receives the SYN-ACK

**Figure 2.4:** The demonstration of the 3 way handshake protocol is illustrated above

This is the way the TCP 3-way handshake works One host sends a “SYN” and Host B sends an acknowledgement of the Synchronize ”SYN-ACK” and lastly Host A sends an Acknowledgement back to Host B. So “SYN” “SYN-ACK” “SYN” is the three way handshake for TCP (just remember that it is very important)

Some examples of programs which use the TCP protocol are listed below:

* HTTPS
* HTTP
* SSH

There are a lot more examples but anything that involves transmitting data is typically utilizing TCP.

**UDP**

UDP or User Datagram Protocol is a connection-less protocol. Communication is “datagram” oriented (just means that it is a packet of data being sent connectionless). The integrity is not guaranteed on the datagram since it is connectionless there are a lot of ways for you to lose some data or not even receive the proper data. Datagrams reach the destination and can arrive out of order or won’t arrive at all but you might not know because UDP doesn’t send an acknowledgement bit back. The fact that UDP doesn’t utilize an ACK bit makes it more efficient then TCP.

Response

Response

Request

Host B

Host A

**Figure 2.5:** An example of how UDP works

As you can see from our little example Host B requested a file or data and Host A sent it via a connectionless protocol. But the real question is did all the data send? We won’t know because this isn’t real life networks.

If you are still confused about the network protocols think about them like below:

The **TCP** protocol is similar to mailing a letter with a return receipt at the post office. Except the post master will organize the letters in order of mailing and only deliver them in order. Meanwhile UDP is just like mailing a letter at a post office. There is no way to know if the piece of mail is getting to the destination.

Some examples of UDP are listed below:

* Streaming media if you lose some bits/ frames that is okay.
* DNS (usually)
* Gaming with games that require optional updates.

**Device Drivers**

A lot of source code in the Linux Kernel exist within device drivers. This makes the hardware usable. The Linux system provides a drivers subdirectory that is further divided by the various devices that are supported. (For instance device drivers for Bluetooth, printers, keyboard).

**Out of the Theory and into the good stuff**

This section of the Lab contains some information on more Linux commands and how we visualize operating systems and services within Ubuntu. These Labs are “Hit the ground running type of Labs” where we just dive right into it.

**All it takes is a touch**

There are numerous commands which can create files (besides using nano or another editor). The most basic method is something called the touch command. This will create an empty file using the name and location specified.

For this example we are going to navigate to the home directory and go through an example:



**Figure 2.6:** Illustrating how we can create a file using the touch command

The first line in the snippet above is using the command “man touch” this will display see what is the capability of the touch command. I highly encourage you to read through this to see the different touch flags.

Then in the second command we make sure we are in the home directory “the cd ~” is a short hand way for this also

In the third command we create a file within our present working directory called file1 we simply just use “touch” and pass it a parameter of our file name called “file1”. Hence the command “touch file1”.

In the fourth command we view the contents of the directory to verify that the file “file1” is in there using the ls command.

If we happen to use the “touch” command on an existing file the command updates the data. Then our file system stores the time when the file was last accessed/modified.

We can also create multiple files with the touch command and create files in specific paths using the command check out the image below:



**Figure 2.7:** This figure illustrates how we can create multiple files within same/different directories.

Notice how we passed the touch command and then two parameters both creating files named file2 and file3 in the Desktop. Then I went to the Desktop directory and then viewed the contents using the ls command.

**Making directories**

Directories! What a great way to organize files in a command line interface. Directories as we know are ways of storing data and other files just without button clicking so through using the cd command we are able to access the directory. But how do we make other directories you might be familiar how in windows and other Desktop environments we right click and make a new folder. Well in Ubuntu and most Linux environments we use a command called “mkdir”. The mkdir command is similar to the touch command but instead of it making an empty file it makes an empty directory. For instance check out the shell output below:



**Figure 2.8:** In the above figure we illustrate how we are able to make directories.

In the first command we tried to access the test directory when it didn’t exist and we got an error thrown “bash: cd: test: No such file or directory” this means the directory doesn’t exist

In the second command we make the test directory “mkdir test”. The mkdir makes a directory (check out man mkdir for more help) and then the next parameter is the name of the directory which we are making.

The third command we call the ls command which lists the current files within the directory. We can see through the result of this command that we have a test directory.

We can also make a directory within the current directory by just passing one more path through the current made directory. Check out the example below:



**Figure 2.9:** This figure illustrates how an error ca be caused when a given directory does not exist.

We know from the prior example we have the test directory already made.

The first command specifies again that we want to make another directory within the test directory and name it example2. This command only works because we have the test directory already made. “mkdir test/example2/” is saying make another directory within the test directory which we know exists and name it example 2.

The theory behind the second command shows us that if we want to make multiple directories when they don’t exist yet Ubuntu won’t just let us make the directories we need to somehow specify to it that we want to make these directories. This means if we run the command “mkdir test/example1/example” despite the test directory being made we can’t make the example1 directory because that needs the “example” directory. Hence if the example directory isn’t made then we can’t make the example1 directory. We can only make 1 directory at a time unless a flag is specified.

In order for the mkdir command to make any directory despite its existence we need to use the –p flag. This allows you to create nested directories in one step regardless if they exist or not check out the commands below: 

**Figure 2.10:** The figure above shows how we can make directories even when they don’t exist. This can be done with –p flag.

The first command is “mkdir –p hello/world/through/directories” which means we are going to make a directory but there is no error that will be thrown if the directories we are calling don’t exist they will just be made. So we make the current directories hello/world/through/directories

The second command we are viewing the contents of our current directory to make sure hello is in there

The third command we “cd hello” or enter in the hello directory.

 The fourth command we view the ls command and see the world directory

The fifth command we “cd world” and are now into the world directory

The sixth command we use the ls command and see the “through” directory

The seventh command we go into the through directory through the “cd” command

The eighth command we view the contents of the current directory which we see the “directories” directory.

Then the ninth command we go and “cd directories” and cd into it.

Directories are fun to use and are incredibly powerful within the realm of computing and Linux.

**Moving files and renaming files/directories**

Moving files from one location to another or one directory to another is also important. For instance if we create a file named “SammyTheShark” in our home directory and we want to move it to our new directory called DigitalOcean how are we going to do that? Enter in the mv command which allows us to move files and rename directories/file. Check out the command snippet to see how we can utilize it:



**Figure 2.11:** In the figure above we are illustrating how we can move a file from one directory to another.

First we create our file using either touch or nano it’s up to you for our example I used the touch command. Then I named the file “SammyTheShark”

The second command we used the “ls” and viewed the current directory to make that SammyTheShark is in there.

In the third command we made a directory and called it DigitalOcean

Then in the fourth command we utilized the mv command. In the most basic form the mv command is called then we call the file name (in our case SammyTheShark) then the directory “DigitalOcean” we want it to go to. In case you are still confused check out a breakdown below:

|  |  |  |
| --- | --- | --- |
| mv | SammyTheShark | DigitalOcean |

mv: the mv command moves the file

SammyTheShark: the file created by touch which is being moved to another directory.

DigitalOcean: The directory which the file SammyTheShark is being moved to.

If you are still confused type in “man mv” and read more about its usage.

Then in the fifth command we can view the contents again by using the ls command

Then in the sixth command we can go into the DigitalOcean directory by way of the “cd” command.

Then we can use the “ls” command and view the contents of it which is SammyTheShark.

But let’s say now that this file is inside the DigitalOcean directory that we want to move back into the home directory. In order to do this we can use a similar set up for the command except that we can use a “.” at the end of a command. Check out the commands below to see how we can verify that this worked:



**Figure 2.12:**  In this figure we are showing how to move a moved file from one directory back to the home directory.

In the first command we utilize is the “cd” command because in order to do this we need to make sure we are in the home directory.

In the second command we use is the “pwd” command and this shows us that we are in the home directory.

In the third command we use the mv command again but specify the path to the file which we want to bring back to the home directory. In this case the path is DigitalOcean/SammyTheShark. Then after we do this we put the “.” symbol there which tells Ubuntu that we want this file back into the home directory.

In the fourth command we check the directory and see the contents and behold! SammyTheShark is back in there!!

In the fifth command we want to access the directory Digital Ocean to make sure that there is nothing in there. The command I entered is to show you that you will get an error unless you exactly write the name of the directory/file which you are trying to access.

In the sixth and seventh command we enter the DigitalOcean directory and then view the contents in it and see nothing there. Hence we know the file was moved!

The last thing we can discuss with mv is renaming directories. In this example we are going to rename the DigitalOcean directory to OctoCat. Check out the image below:



**Figure 2.13:** In the figure above we renamed the directory “DigitalOcean” to “OctoCat”

The first command the “cd” command is used to get back to the home directory which we need to be in order to use this command.

The second command is the mv command where it takes in parameters such as [the current directory name] [directory we want the name to change to]. The current directory is called DigitalOcean and we want to change it to OctoCat. If you are still a little confused check out the little detail snippet below:

mv DigitalOcean OctoCat

 The mv command moves/renames files/directories

The DigitalOcean directory is the name of the directory we want changed

The OctoCat name is the name of the directory we want DigitalOcean to be changed to.

Then we can view the contents of the folder using the “ls command” which shows the change in directory name.

Then we went into the directory and viewed the contents and saw it had nothing in it which it should have!

\*Note it is absolutely imperative to realize that the Linux system will not prevent you from destructive actions. If you are renaming a file and choose a name that already exists the previous file will be overwritten by the file you are moving. If this occurs there is no way to recover the previous file even if it was an accident.

\*Also note Sammy the Shark is the Digital Ocean Mascot (<https://www.digitalocean.com/>)

\*Also note OctoCat is Github’s mascot. (<https://github.com/> )

**Copying Files and Directories**

We previously discussed that we could use the “mv” command to rename or move files and directories but we could not copy the files/directories. This is where the “cp” command can make a new copy of an existing item.

 For the example below we are going to make a new file named computersecurity.txt and put some contents in it and then we will copy it to another file called something.txt:

In the command below we called the nano text editor and specified the text file which we are making called computersecurity

**Figure 2.14:** We are creating a text file called computersecurity.txt

We then populate the text file with some contents in my case I wrote a little blurb about what we are doing. Once completed I hit: (1) ctrl+x (remember the “^” is ctrl) then (2) Y to save the modified buffer and lastly (3) the enter button to confirm writing to the file.



**Figure 2.15:** The contents of the text editor

Then in the screenshot below we use the cp command which we know copies files. The first parameter if it’s not a flag is usually the file which we want copied in this case it is “computersecurity.txt” and then the next parameter is the name of the file which we want to be copied in this case “something.txt”.



**Figure 2.16:** The illustration of the cp command copying the contents from one file to another.

If you are still confused check out the more in depth explanation below:

cp computersecurity.txt something.txt

The cp command is used to copy files and directories

The file computersecrity.txt is the file that has contents in it which we want copied.

The file something.txt is the file that we are making that will get all the contents of the file computersecurity.txt.

If you are still confused check out man cp for more clarification on the cp command.

Something to note similarly to the mv command it is possible to overwrite a file if you aren’t careful about the filename. If I already have a predefined file name config.txt and then I say copy the contents of the file computersecurity.txt to config.txt the text file will update and you will lose anything that was initially in the config.txt file.

In order to copy directories and its contents we need to use a recursive flag along with the cp command. The recursive flag is the “-r flag” This flag needs to be added to a cp command regardless if the directory is empty or not. Check out the commands below to see how we did this:



**Figure 2.17:**  Copying directories from on directory to another.

The first command I wrote is the “ls” command the ls command which I use to make sure the directory I want copied is there in this case it is (we are going to transfer the OctoCat directory to a MobyTheWhale directory)

The second command I move the file something.txt into the OctoCat directory. (remember the OctoCat directory will get copied to the directory MobyTheWhale)

The third command we do is the ls command to verify that something.txt was moved out of the home directory

The fourth and fifth command I am entering or “cding” into the OctoCat directory to view the contents and see that something.txt is in there.

The sixth command I “cd” into the home directory by just typing the cd keyword.

The seventh command the “cp –r OctoCat MobyTheWhale” performs a recursive copy on the OctoCat directory and copies its contents into a new directory called MobyTheWhale (note Ubuntu will automatically this directory for you).

The eight command I use the ls command and make sure the OctoCat directory and the MobyTheWhale directory are there.

The ninth and tenth command I enter or “cd” into the directory “MobyTheWhale” then view the contents with ls and see the file was copied.

The last thing I want to cover with copying files and directories is how to copy one file into a directory. In order to do this we use the cp command call the file we want copied and then directory. In order to see how I did this use the image below:



**Figure 2.18:** Copying one file from a directory into another directory.

In the first command I use the cd command again to get out of my current directory and into the home directory.

The second command I use the ls command to view the contents of the home directory. (For this example we are going to move the farmingdale.txt file into the MobyTheWhale directory)

The third command we use is the cp command. We call the cp command along with the file we want copied and then directory we want that file copied to.

The fourth command we view the contents of the directory to make sure it is in there still.

The fifth and sixth command we go into the MobyTheWhale directory and view the contents and see that farmingdale.txt is in there. Hence we know that the copying worked,

**Removing Files & Directories**

One last thing for this lab let’s discuss how to remove files and how to remove directories, Deleting files/directories which aren’t being used or are irrelevant are key to managing your workspaces. There is no need for you to have 10 directories which are unused so we can remove them or if we don’t need a file we can just remove that. Enough talk let’s get into it.

Removing files and directories are an incredibly delicate job because once we remove the file there is no undo button for the command. So if by accident you destroy an important file I’m sorry.

In order to remove a file we use the “rm” command we call the rm command followed by the file. To see an example we are going to create a file named trash.txt then remove it from the directory:



**Figure 2.19:** In the figure above we are illustrating how we can remove a file to the trash.

In the first command we use the nano text editor and create a file named trash.txt

In the second command we use the ls command and view the contents of the directory to see that the trash.txt file is removed.

In the third command we call the rm command with the file name so we call “rm” then the file we want to destroy trash.txt

In the fourth command we view the contents again and see that the trash.txt file is gone hence the removal was a success.

 The next thing we are going to look at is how to remove empty directories. This can be easily achieved using the “rmdir” command. As long as there is no contents in the directory it will be easy to remove. For the example below we are going to create a directory named “pineapples” and even a directory named apples within the orange directory.

In the first command below we are going to create a pineapple directory with nothing in it.

In the second command below we view the contents of the home directory and we are able to see the pineapple directory.

In the third command we remove the pineapple directory.

In the fourth command we view the contents of the home directory to make sure it is removed.



**Figure 2.20:** In the figure above we are illustrating how to remove a directory.

In the command snippet below we are going to go over multi-level directories which are empty and some errors you might get:



**Figure 2.21:** Going through multi leveled directories and how we can remove them.

In the first command we made a directory and named it orange

In the second command we entered orange directory and then in the third command we made an apple directory (so now we have an apple directory within our orange directory)

In the fourth, fifth and sixth command we view the contents of the orange directory, then we go into the apple directory and make sure everything is good the we cd back out of it to our home directory.

In the seventh command we remove the apple directory **(notice how we specified the path to it because YOU CAN’T BE INSIDE A DIRECTORY WHEN YOU DELETE IT AND WE CAN’T DELETE THE ORANGE DIRECTORY AS YOU WILL SEE)**

In the eighth, ninth, tenth and eleventh command we view the contents of the home directory make sure orange is in there still then we enter the orange directory, next we make a directory called twizzlers. Then lastly we view the contents and see twizzlers is in there.

The twelfth and thirteenth command shows errors what happens when we try to make a directory that exists in our present working directory. The thirteenth command shows what happens when you try to remove a non-empty directory.

If we want to remove a non-empty directory like this orange directory we need to pass a flag through it. As you might have guessed we need to use the “-r flag”. Check out the continuation of the snippet above to see how I got rid of it. 

**Figure 2.22:** Removing a non-empty directory is illustrated above.

The first line we use rm then pass it the –r flag and call the orange directory and Ubuntu will destroy everything in there because it’s a recursive call.

The second command we view the contents and see that the orange directory is not there and the third command shows if we try to access the directory it doesn’t exist because well it’s removed.