**4.1 Student Guide: Introduction to Linux**

**Class Overview**

This week, you will work through a series of command-line exercises in which you will explore a Linux file system, collect evidence of a security breach, audit and reconfigure a Linux machine's user and group permissions, and take steps to harden the system.

Today's class will introduce you to the Linux operating system.

**Class Objectives**

By the end of class you should be able to:

* Name three of the most important distributions of Linux.
* Navigate the Linux file structure using the command line.
* Manage processes with the top, ps, and kill commands.
* Install packages using apt install.

**Lab Environment**

You will use your local Vagrant virtual machine for today's activities.

* Username:sysadmin
* Password: cybersecurity

**Slideshow**

The slides for today can be viewed on Google Drive here: [4.1 Slides](https://docs.google.com/presentation/d/11HJiRswnw1gaX1ojBMuhCNDHMwo2yTJchJ_xrhicp9c)

**01. So, Why Linux?**

Today's lesson will introduce the fundamental concepts and details of Linux systems.

While Linux is uncommon as an operating system on personal computers, it is widespread on the internet and in enterprise networks. By some estimates, [over 70% of websites on the web run some variant of Unix](https://w3techs.com/technologies/overview/operating_system/all).

The widespread presence of Linux machines on modern networks makes them a common target for attackers. Therefore, familiarity with the operating system is crucial for cybersecurity professionals.

This knowledge is essential for the following technical roles, among others:

* **Help Desk/IT Support**: Help desk and IT support specialists often need to troubleshoot issues on Linux workstations. People in these roles often find themselves troubleshooting issues with macOS, as well. Since macOS shares most of its structure with "classic" Linux operating systems, this knowledge is valuable even for personnel who are not in charge of configuring or managing machines themselves.
* **Systems Administration**: As most complex networks contain at least one Linux server, systems administrators need to understand how to audit, configure, and secure Linux servers and workstations.
* **Penetration Testing**: Again, most complex networks contain at least one Linux server, making them frequent targets for pentesters. Since pentesters must exploit machines to gain access, deep knowledge of how the operating system works and where it stores the most important files is essential.
* **Network Forensics**: Forensics specialists spend much of their time inspecting server logs. Often, these logs are generated by or stored on a Linux server, and forensic examiners must be able to navigate the file system. In addition, they must be careful not to accidentally alter the file system while investigating, making knowledge of how and where data is stored on Linux machines a crucial element of the job.

Today's exercises will require you to investigate a malfunctioning Linux server, which users have reported is running more slowly than usual. You will be tasked with identifying evidence of suspicious activity on the system, and taking steps to contain it.

You will use the following methodology across the student activities during the day:

* **Audit files.** You’ll begin by looking for suspicious files on the machine. To do this, you will learn how files are organized on Linux, and where important system and user files are normally found. This will enable you to identify malicious files that seem out of place.
* **Audit processes.** After identifying suspicious files, you will look at which programs users are running. This will allow you to check that no one is running malicious software on the server.
* **Install security packages.** Next, you will install security applications, also known as packages. These will help secure the machine from future attacks.
* **Configure security services.** Finally, you will configure one of these packages, called Tripwire, to automatically notify you whenever someone saves suspicious files on the machine.

The skills covered will help you complete the activities.

**02. Linux History and Distributions**

A brief history of Unix and Linux:

* In 1969-1970, Dennis Ritchie, Kenneth Thompson and several others at AT&T Bell Labs began developing a small operating system called **Unix**.
* Over the next two decades, different updates and distributions of Unix were created. Most were proprietary.
* In 1984, Richard Stallman's Free Software Foundation (FSF) started the GNU project, which created a free distribution of the Unix operating system.
* In 1991, Linus Torvalds used many of the components from the GNU project to create a free, open source operating system, which is now known as **Linux**.

Note the following about operating systems in general, and Linux in particular:

* An **operating system**, or OS, is the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.
* Windows, Mac OS X, and Linux are all examples of operating systems.
* Of these operating systems, Linux is by far the most commonly used on the Internet, and most enterprise networks feature at least one Linux machine.

This is because, unlike Windows and OS X, Linux is **free, open source software (FOSS)**.

* This means that anyone can read or modify its **source code**.
* Linux is free and open source specifically because Windows and Unix, Linux's early competitors, were not. In other words, Linux was developed for researchers and students in direct response to the fact that Windows and Unix were extremely expensive and/or inaccessible in the early days of computing.
* Today, most tools in the hacker/security community are still published as open source, and a belief in free software and free information is widespread in the industry. Almost all the tools you will use in class are open source, and nearly all technical security professionals are familiar with the Linux FOSS ecosystem.
* Because Linux is FOSS, many different people have developed their own special-purpose variants of the operating system. These variants are called **distributions**, or **distros**.

The two most common distributions in use are **Ubuntu** and **Kali Linux**.

* There are many different distributions in use today. In this course, we will use Ubuntu and Kali Linux.
* Both Ubuntu and Kali Linux are distributions based on another distribution called **Debian**. Ubuntu is essentially Debian with customizations for general-purpose users, and Kali is Debian with customizations for security professionals.
* Since both distributions are based on Debian, everything you learn in this module will transfer directly to everything you learn about Kali in the latter half of the course.

One of the most important skills you will gain in this module is comfort with the **command line**, which you will use extensively on both Ubuntu and Kali.

* Knowledge of the command line is particularly important in the professional world, because most production Linux servers don't even offer a graphical interface. They can only be used from the command line!
* Command line-only machines are called **headless servers**. Headless servers are the norm because, by today's standards, the command line requires very few resources. This gives the server maximum resources to run its services and applications. A GUI based system is neither required nor desirable.
* Being able to navigate a headless Linux server is the main reason you need to learn Linux and the bash command line. Learning the bash command line environment will enable you to work with both Linux *and* Unix systems, such as Mac OS X.

**03. Activity: Distribution Research**

* [Activity File: Distribution Research](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/1/Activities/03_Distro_Research/Unsolved/README.md)

**04. Activity Review: Distribution Research**

* [Solution Guide: Distribution Research](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/1/Activities/03_Distro_Research/Solved/README.md)

**05. Linux File System Structure**

Note the following about the Linux file system structure:

* All operating systems maintain certain conventions for where and how they store different kinds of files. For example, Windows and Mac machines typically place images in a ~/Pictures folder.
* Linux also has a conventional file system structure.
* These conventions are consistent across distributions. For example, user applications are typically stored in /usr/bin, and this will always be the case across distributions.
* You therefore can apply what you learn today to other distributions they encounter, as they all have the files and folders you will learn about today.
* In addition, all popular Linux distributions use the bash shell by default. This means that you can use the command line tools you learn here on other distros, as well.
* Being able to use the same command line on many distributions is a major advantage, because GUI-based desktop environments are usually not the same across distros.
  + For example, Ubuntu and Fedora look very different, and need to be configured with different GUI tools, but the shell commands used to configure each machine are almost identical.
  + Understanding the command line allows administrators to work with any Linux machine using a bash shell, even if it has an unfamiliar desktop environment.

Once you master the Linux file system structure, you can better identify potentially malicious applications and files that are in places they do not belong. Often, this indicates that the system has been compromised.

There are several conventional directories in Linux:

* /: This is the **root** directory, which contains every other directory on the system.
* /home: This directory contains each user's private files. Users should not be able to save files anywhere else.
* /etc: This directory contains configuration files, which define how the machine should run and who is allowed to use it. For example, this directory includes information on registered users and their passwords.
* /bin and /sbin: These directories contain applications, such as web browsers, and commands, such as ls.
* /var: This directory contains files that change over time. For example, /var/log contains log files, which programs update frequently as they run.
* /tmp: This directory contains files that are only needed for a short period of time.

There are other important directories, such as /proc and /boot. However, these have very specific use cases, and are beyond the scope of today's lesson.

**Linux File Structure Walkthrough**

We will now walk through the Linux file system:

* Launch a terminal.
* We will start at the top of the directory structure, the root directory.

**/ Root Directory**

Root directory is referenced with /. The root directory contains every other directory. All files and folders are ultimately contained within the root directory.

* Run: cd /
* Run: ls

Your output should look like:

$ cd /

$ ls

bin boot cdrom dev etc home initrd.img initrd.img.old lib lb32 lib64 lost+found media mnt opt proc root run sbin snap srv swapfile sys tmp usr vagrant var vmlinuz

Each item listed here is a directory, because the main purpose of / is to store other crucial directories, not bare files.

* Because we are working within a VM we have mounting provisioners such as initrd.img, initrd.img.old, swapfile, vagrant and vmlinuz. Although we won't cover these in great detail, these files allow the Ubuntu virtual machine communicate with our host machine.

The next directory we'll explore is /home, where users store their own private data.

**/home Directory**

* Linux allows you to create multiple user accounts. This allows many people to use the machine at the same time.
* Each user is allowed to personalize their environment. For example, users can change their desktop backgrounds or install new software.
* The files that contain each user's personal preferences are called **local configurations**, or **configs**. These configurations only apply to the user who wrote them.
* Each user is given their own dedicated folder on the file system. This is where they store their own private files, folders, and local configurations.
* By default, users are only allowed to modify files and folders in their own dedicated directory. This means they cannot modify files in other users' directories, or in protected folders like /.
* These private directories are stored in /home.

The name of a user's private directory is the same as their username.

* For example, the user sally would have a directory at /home/sally, while the user jack would have a directory at /home/jack.
* Only registered, active users should have private directories in /home. Directories belonging to users who have been deleted or who no longer use the computer should be deleted.
* **Note:** root is a special user, and root's home directory is not /home/root, but rather /root.

**/etc Directory**

* The /etc contains configuration files, which define how the machine should run and who is allowed to use it.
* While normal users can only modify their own local configurations, an administrator can make changes that apply to all users of the system.
* These are called **system-wide configurations**, and apply to every user on the system. For example, an administrator might configure a machine to prevent people from going to <https://facebook.com>.
* Configurations are typically stored as simple text files. Therefore, they can easily be inspected with familiar tools like cat and less.

The files /etc/passwd and /etc/shadow contain information about all registered users on the system.

* /etc/passwd contains a list of registered users on the system. Contrary to its name, it contains no information about user passwords.
* This means that attackers who steal /etc/passwd will get a list of users to attack, but will not be able to steal their passwords.
* By contrast, /etc/shadow contains both a list of usernames and information about their passwords.
* Specifically, it stores something called a **password hash**. This is a string that is different from the user's password, but the operating system can still use it to check that they've entered the correct password.
* Because hashes aren't passwords, they cannot be used to log in. Additionally, the process of turning a hash back into a password, called **password cracking**, is extremely time-consuming.
* This means that an attacker who steals /etc/shadow cannot break into user accounts without first spending the time to crack their passwords.
* In general, an attacker cannot crack strong passwords, but cracking passwords is much easier to do if the passwords are weak.
* Since most users do not use strong passwords, an attacker who steals /etc/shadow is likely to be able to crack at least one of the passwords it contains.
* This makes /etc/shadow one of the **most important files** to protect on a Linux system.

In the next class, we’ll begin with a password cracking activity that will use the /etc/shadow file.

* Run less /etc/passwd to show the contents of /etc/passwd.
* Run sudo less /etc/shadow to show the contents of /etc/shadow .

We need sudo to read /etc/shadow because the permissions on that file are strict. sudo temporarily grants root permissions. We will cover permissions in depth on Day 2 of this unit.

* This file looks nearly identical to /etc/passwd, with one stark difference: instead of an x, the second field contains a long, complex string, such as $6$6Y/fI1nx$zQJj6AH9asTNfhxV7NoVgxByJyE.rVKK6tKXiOGNCfWBsrTGY7wtC6Cep6co9eVNkRFrpK6koXs1NU3AZQF8v/.
  + This string is a password hash.
  + This hash cannot be used to log in, but can be cracked to retrieve the user's original password.
* If a row in /etc/shadow contains a ! instead of a password hash, means that the account is locked
* If a row in /etc/shadow contains a \* instead of a password hash, that the user is not allowed to login.
  + This is usually implemented on a user reserved for the system, and not a *human* user.

**/sbin and /bin Directories**

Now we’ll look into several directories which contain program files that are run on the Linux distribution.

* Every application or command you run on a Linux system is a program whose code is stored in files.
* Files containing programs are unique because you can run them. This is in contrast to other, "normal" files, such as PDFs, which you can only read and write.
* Files you can run are called **executable** files, because they execute some task.
* All software, including malware, is stored in executable files.
* Malware files can be dangerous, because there is nothing preventing them from doing things that might harm the system.

Executables can be stored anywhere on the file system, but the two main directories that store commands are /sbin and /bin.

* bin stands for *binary*, because executable files contain machine code. This is code written in ones and zeroes, which is not readable by humans.
* The /bin directory contains programs used by "normal" users, such as ls and cp.
* The /sbin directory contains programs used by administrators, such as adduser, which can add users to a system, and passwd, which is used to change a user's password.
* Run ls /bin to display the contents of /bin.
* Run ls /sbin to display the contents of /sbin.

Output should look like:

$ ls /bin

bash cp grep ls ntfs-3g.probe ps systemd unicode\_start

bunzip2 cpio gunzip lsblk ntfscat pwd systemd-ask-password vdir

busybox dash gzexe lsmod ntfscluster rbash systemd-

...

Display the contents of /sbin by running ls /sbin:

$ ls /sbin

acpi\_available e2image ifdown lvdisplay mkfs.vfat pvresize unix\_chkpwd

agetty e2label ifquery lvextend mkhomedir\_helper pvs unix\_update

apm\_available e2undo ifup lvm mkntfs

**/var Directory**

Many programs save information about errors or suspicious behavior in **log files**. These log files often contain valuable evidence about malicious activity.

Since the list of installed programs looks good, you'll proceed to verify that these programs are saving the expected logs.

* Log files change constantly, because programs that keep them add information to them as they run. This means these files are *variable*; in other words, they change over time.
* Files that change over time are stored in /var, for variable.
* Log files are stored in their own directory, called /var/log.
* Other types of variable files are stored in /var, but they will not be discussed in today's class.
* Linux systems may contain the following logs:
  + /var/log/ufw.log: Stores firewall logs, which include attempts by users to visit unauthorized sites.
  + /var/log/auth.log: Stores failed login attempts when a user is trying to log in as the root user.
* Run ls /var/log to display the directory's contents. You should see some of the following logs:
* $ ls /var/log

alternatives.log btmp kern.log syslog apport.log dpkg.log auth.log

* + You should not have a ufw.log.
  + This listing reveals the server is *not* maintaining firewall logs, but *is* saving records about failed login attempts.
  + This means you do not know if users have visited malicious websites, but you can tell if anyone has tried and failed to log in as an administrator or different user.

**/tmp Directory**

We have one last directory to view: /tmp, which stores temporary files, and is often abused by hackers.

* Logs are meant to be *permanent*. Programs expect to add information to log files, and not to rewrite or delete them and create new ones (though administrators can do these things manually).
* Programs often need to create files that do not need to be permanent. For example, a text editor might save copies of your work every few minutes, even if you don't manually save it, to ensure it can restore your work if you accidentally close the editor before saving.
* Such files are *variable*, in that they change regularly, but also *temporary*, in that they do not need to be maintained long-term.
* Since these files are different from logs and other such files, they are typically stored in their own dedicated directory: /tmp.
* Since files saved to /tmp are intended as temporary, the operating system deletes files in /tmp whenever the machine is restarted.

Because files in /tmp are deleted between reboots, hackers often save malicious files or scripts to this directory, knowing that they'll be available for use while they're exploiting the machine, but automatically deleted by the target itself at reboot.

**Summary**

* /etc: Stores host-specific system-wide configuration files, as well as the most sensitive files on a Linux system.
* /var: Stores files that are continually updated.
* /home: Stores user home folders.
* /tmp: Where applications write temporary files that can be deleted on reboot.
* /bin and /sbin: Where the system keeps its main *binary* or program files.

**06. Activity: Linux Landmarks**

* [Activity File: Linux Landmarks](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/1/Activities/06_Linux_Landmarks/Unsolved/README.md)

**Note**: You need to run the following command to set up this activity:

* sudo bash /home/instructor/Documents/setup\_scripts/instructor/landmarks\_review.sh

**07. Activity Review: Linux Landmarks**

* [Solution Guide: Linux Landmarks](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/1/Activities/06_Linux_Landmarks/Solved/README.md)

**08. Break**

**09. Resources and Processes**

The previous activity revealed a suspicious script in /tmp.

Since scripts are executable programs, our next step is to check which programs are currently running on the machine to determine if anything malicious is taking place.

When a program runs, it must process data and potentially make changes to the file system. This is why we call a running program a **process**.

* For example, a text editing application needs to save temporary files to backup users' work, and then save their final draft to a file on disk.

When these programs process, save, and modify data, they consume a computer's **resources**.

* The two important resources we'll look at are **memory** and the **Central Processing Unit (CPU)**.
* As we will see shortly, programs compete to use a finite amount of resources available on a machine.
* This limited amount of resources is something attackers will take advantage of when attacking a machine.

Note the following about memory:

* Memory comes in two forms: **Random Access Memory (RAM)** and disk space.
* Disk space is used to save data "permanently.” Files saved to a disk persist even after a process ends.
* By contrast, RAM is used to run the program's code. It is only used while the program is running.
* The more work a process does, the more RAM it needs to use. Processes that do a lot of work often use a lot of RAM. However, there is only a finite amount of RAM available, so such "heavy" processes often slow down the computer by "hogging" available memory from other ones.

The Central Processing Unit also works to process programs on a system:

* The CPU acts as the central brain of a system, determining how much work a process has to do, and how "difficult" that work is.
* A difficult task (such as encrypting a large file) uses a lot of RAM and/or CPU, while an easier task (like executing the ls command) requires minimal amounts of these resources.

Hackers can take advantage of a system's finite resources:

* Linux systems can run multiple processes at once. Since a system only has finite memory and disk space, all of these processes must compete for resources.
* Hackers can perform denial of service (DoS) attacks by launching processes that eat up memory on a target machine. This can slow down or crash the machine, making it unavailable to users, and thus denying them service.
* Hackers can also start malicious processes that don't use a lot of memory, and are therefore not easy to spot without specifically looking for them.
  + One example is a "backdoor" process, which allows hackers to break into machines they shouldn't have access to without being detected. These don't use much memory because they essentially make a network connection to the hacker's machine and then listen for instructions.
* These threats emphasize the importance of being able to identify normal system processes from runaway or malicious processes.

Linux has several commands for managing processes:

* top allows you to see all of the running processes in real time. It updates every three seconds to show you what's happening on the system.
  + top is often used by administrators to get a general feel for how the system is running in real time. It is immediately obvious if a process is using too much memory or CPU.
* ps allows you to take a snapshot of all the running processes on the system. You can use different arguments to show different subsets of processes and use this output with other commands in the command line.
  + Snapshots are good if you want to get some process information programmatically or if you don't need to see how the program is behaving and only want some basic info.
* kill is used to stop processes, usually ones that are causing problems. kill attempts to allow a process to finish what it's doing before it shuts down. However, we will learn how to use a flag that will immediately stop any process.

Overall, we use top and ps to get a feel for the overall "system health" of the machine. These commands let us identify details such as how much memory/CPU a process is using and even what user started each process.

* A "healthy machine" is one in which a system is not being overtaxed by using too much memory, RAM, disk space, or other resources.
* When a system *is* overtaxed, it affects uptime, which in turn compromises the availability of the services that system provides. A machine only has a finite amount of resources, so when they are used up, the machine slows down and cannot process requests as easily.
* An extreme example of this would be an attacker intentionally using up a system's resources to slow the machine down and compromise its availability. This type of attack is often carried out on a web server, and stops the machine from being able to display its website. It is known as a denial of service or DoS attack.

We want to run the potentially malicious script to determine how it affects system health. This is usually done by:

* Moving the malicious script or program to a contained environment (such as a virtual machine) where it cannot escape or affect other machines.
* Executing the malicious script or program.
* Monitoring how it uses memory/CPU, which files it accesses, any network communications it makes along with any other behavior it exhibits.
* This is called **dynamic analysis** and helps us understand an attacker's intent.
* Dynamic analysis is useful for determining how we should harden the system and where else to look for consequences or evidence of compromise.

In real life you wouldn't run the script right on the machine you're auditing, but for this exercise, we will run it on the same machine.

* Note that this machine is already a VM.

In the upcoming demo, we will run the suspicious script we found in the /tmp directory during our previous audit. We will then use top and ps to understand what it does, and use kill to end the process after you understand its behavior.

**Inspecting Malicious Files Demo**

First, we’ll run top on our system to better understand its output.

* Run top

top splits its output into two panes: an upper summary pane, and a lower details pane.

* The information in the pane at the very top summarizes overall resource usage.
* The top row shows the current time of the system, how long it has been up, and how many users are using the system.
* The second row contains information on how many processes, also known as **tasks**, are currently running.
* The remaining lines in the top pane show overall CPU and memory usage. **CPU usage** refers to how much the computer has to work to finish its task. **Memory usage** refers to how much RAM and disk space are used to do that work.
* A quick glance at the upper pane reveals the general state of the system. High CPU and memory usage indicates that the computer is running inefficiently. The number of active users reveals whether more people than usual are using the machine.

While this top pane is useful for diagnosing a machine's general health, we must read the bottom pane to understand which specific processes are using CPU and/or memory.

The bottom pane breaks down usage by each individual processes running on the system.

The following columns are the most important sections for our purposes:

* **PID**: A number identifying each running process, which functions as a name for each process. The PID is extremely important, as this is how you specify which process to stop when using kill.
* **USER**: Which user started the process.
* **PR**: "Process priority." The higher the number, the more important the process. This determines how much resources are dedicated to each process and if one action gets processed before another. It is determined by the system.
* **NI**: "Nice value." This is related to the process priority. A nice value of -20 means "highest process priority." A nice value of 19 means "lowest process priority." Most processes have a nice value of zero. A process priority can be escalated in a number of ways, though it is normally managed automatically.
* **COMMAND**: Name of the command that started the process.
* **%CPU**: Percentage of CPU the process is using.
* **%MEM**: Percentage of RAM the process is using.

You can enable column highlighting and sorting by pressing the x key. By default, the %CPU column is highlighted and sorted by highest CPU usage.

You can also sort the processes by what user started it pressing the u key.

* Press the u key.
* Enter root to demonstrate sorting by processes started by the root user.

At the moment, nothing much is happening on the system, so CPU usage is relatively low. This is what you'd expect a "normal" system to look like.

You can close top by pressing q.

* Press q to return to the command line.

top generates a live view of all running processes, which includes a lot of information.

Sometimes all you need is a snapshot of which processes are running and their PIDs, so you can stop those that are causing problems.

ps is great if you want to look at only one process or a few processes and get their PID to interact with them or shut them down. Because it outputs a static list of processes it is also useful for interacting with processes programmatically.

While not as powerful as top, ps is a quick way to get information about running processes.

* Run ps

Your output should be similar to:

$ ps

PID TTY TIME CMD

1319 pts/0 00:00:00 bash

29243 pts/0 00:00:00 ps

* This lists the PID, TIME, and CMD (command).
* The bash process is the shell you're using.
* The ps process is the command you just ran.

ps can generate more detailed output with the addition of special flags.

* Type ps a

Your output should be similar to:

PID TTY STAT TIME COMMAND

625 hvc0 Ss+ 0:00 /sbin/agetty --keep-baud 115200,38400,9600 hvc0 vt220

627 tty1 Ss+ 0:00 /sbin/agetty --noclear tty1 linux

1319 pts/0 Ss 0:00 bash

3692 pts/0 T 0:00 top

17959 pts/0 R+ 0:00 ps a

27778 pts/1 Ss 0:00 bash

* ps is the process command.
* a stands for *all* processes that have a terminal assigned to them.

A terminal refers to the command line terminal that we are using.

* In the past, terminals used to be one physical screen or monitor called a teletype terminal or TTY terminal. Today, terminals are still referred to as a TTY terminal, but they are virtual. A Linux system has multiple virtual TTY terminals.

Some processes are assigned a virtual TTY terminal in case you need to interact with that process later.

* Type: ps aux
  + ps a are the same as the previous command.
  + u selects by effective user ID.
  + x lists processes that do not have a terminal assigned to them.

We do not need to start these options with - because ps is a very old program written before the - standard for options was adopted.

When used together, these three options take a snapshot of every process running within a few milliseconds of you pressing the Enter key.

* Run ps aux

Output should look like:

$ ps aux

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 0.0 0.0 204740 6276 ? Ss Jul24 0:01 /sbin/init

root 2 0.0 0.0 0 0 ? S Jul24 0:00 [kthreadd]

root 4 0.0 0.0 0 0 ? I< Jul24 0:00 [kworker/0:0H]

root 6 0.0 0.0 0 0 ? I< Jul24 0:00 [mm\_percpu\_wq]

root 7 0.0 0.0 0 0 ? S Jul24 0:00 [ksoftirqd/0]

root 8 0.0 0.0 0 0 ? I Jul24 0:12 [rcu\_sched]

root 9 0.0 0.0 0 0 ? I Jul24 0:00 [rcu\_bh]

root 10 0.0 0.0 0 0 ? S Jul24 0:00 [migration/0]

root 11 0.0 0.0 0 0 ? S Jul24 0:00 [watchdog/0]

root 12 0.0 0.0 0 0 ? S Jul24 0:00 [cpuhp/0]

root 13 0.0 0.0 0 0 ? S Jul24 0:00 [cpuhp/1]

root 14 0.0 0.0 0 0 ? S Jul24 0:00 [watchdog/1]

root 15 0.0 0.0 0 0 ? S Jul24 0:00 [migration/1]

root 16 0.0 0.0 0 0 ? S Jul24 0:00 [ksoftirqd/1]

root 18 0.0 0.0 0 0 ? I< Jul24 0:00 [kworker/1:0H]

root 19 0.0 0.0 0 0 ? S Jul24 0:00 [cpuhp/2]

...

* This output lists all processes sorted by the user that started them. It gives you a snapshot of *all* of the processes running on the machine at the time you ran the command.
* This command is great if you want to take a look at every process running from all users and potentially investigate everything from a high level.
* You can use other command line programs like cut, grep, or less to make this output more manageable.

Filter the output for any line that contains "bash".

* Run ps aux | grep bash

Your output should be similar to:

user 1319 0.0 0.0 21604 5520 pts/0 Ss Jul24 0:00 bash

user 27778 0.0 0.0 21488 5360 pts/1 Ss 17:44 0:00 bash

user 29979 0.0 0.0 12720 936 pts/0 S+ 18:56 0:00 grep bash

This output is much more manageable than sorting through *every* process manually.

Now that we understand the information report top, we’ll run our potentially malicious script to see if that information changes. This is an indication that the script is causing the system to run abnormally.

In order to both process and inspect the results with top, you'll need two terminals: one to start the script, an another to run top.

* Open two terminals.
* Locate the suspicious a9xk.sh script in your ~/research directory.
* Run cat /home/instructor/Documents/research/a9xk.sh
  + This script is using the stress command, so you will be looking for the stress process to kill it.
* Run sudo bash ~/Documents/research/a9xk.sh in one of the terminals. The script runs a sudo command, so you may need to enter a password.
* In the second terminal, open top again.
* Press x again to sort by %CPU. You should see output similar to the image below.
  + You can easily see the stress command in the command column.
  + The %CPU column reveals that this process is using a substantial amount of resources.

Use the < and > keys to sort by the CPU column. Then, sort by the Memory column.

* CPU usage is very high, but memory is relatively low.
* This is because the malicious script causes the computer to perform a lot of computations, but does not require much memory to do them.
* This demonstrates that either high CPU usage or high memory consumption can slow down a computer.
* Security professionals and systems administrators should inspect both values to accurately identify why a machine might be running slowly.

Upon identifying a suspicious process, the natural next steps include:

1. Identifying the user who started it.
2. Killing the process.

In this case, we can see that the stress command was started by root, because the script uses the sudo command.

To search for processes started by a particular user:

* Press u to get a prompt in between the two panes.
* Enter sysadmin as the username to filter for your own processes.
* Note that stress does not show up, since it was started by root.

We cannot see the process for a9xk.sh here.

top contains the PID for the stress commands and this is all the information you need to kill them. However, for the sake of demonstration, we’ll use ps aux to give us this information.

* Press q to close  top.
* Run ps aux  | less to get a snapshot of all running processes in a more navigable output.

Looking at all the processes this way is a bit cumbersome.

* Run ps a. This gives us a much more manageable output and we can see the stress processes clearly.

Note one of the stress PIDs you want to kill. Remember that the PID is the second column from the left.

Once we know what process we are looking for, we can now use the kill command to stop it. Running kill <PID> will send a signal to that process to stop what it is doing and return resources to the system before exiting.

* Run sudo kill <PID for stress>
* Run ps a again to prove that the process has been stopped.

This might take a long time if you have to kill each one individually.

Finally, note the following about programs that launch several processes:

* Some programs must launch several processes in order to do their jobs. For example, an application like Spotify might need to start a network process to stream music, as well as a GUI process to display the user interface.
* If a program has several processes going, administrators can use killall <process name> to tell the OS to kill the target process and any sub-processes it has started.
* This command is also good if you do not know the PID of the specific process to kill, but do know the name of the application you want to close.

Run sudo killall stress

* Run ps a again to show that all the stress processes are stopped.
* Finally, run top. Its output now looks like it did originally.

Quitting those processes brought CPU usage back down, allowing the machine to run faster.

While this example is simple for ease of demonstration, the kill processes is important for controlling exactly what is running on a host, which is important for both system administrators and security professionals.

**10. Activity: Process Investigation**

* [Activity File: Process Investigation](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/1/Activities/10_Process_Investigation/Unsolved/README.md)

You will need to run sudo bash /home/instructor/Documents/setup\_scripts/instructor/processes.sh </dev/null &>/dev/null & before starting this activity.

**11. Activity Review: Process Investigation**

* [Solution Guide: Process Investigation](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/1/Activities/10_Process_Investigation/Solved/README.md)

**12. Installing Packages**

In the previous exercises, you identified malicious files, then used top and ps to determine who was using them.

These suspicious files are a problem because a malicious user was able to save them to disk.

Fortunately, just like on Windows and Mac machines, we can enhance our Linux experience and download additional tools and programs to help us do our jobs more efficiently.

These additional tools are called **packages**:

* Administrators often have no choice but to install additional software to properly harden the machines they manage.
* New packages are installed with a tool called a package manager. The package manager used by Ubuntu is called *aptitude*. We will use aptitude with the command apt.
* Installing software with apt is very simple. The syntax is: sudo apt install <package name>.
* When you install a package with apt, Linux searches special databases to find information about <package name>. If it finds it, it will download and install the package.
* These databases are known as repositories. Repositories are specifically used to store and distribute packages are known as Personal Package Archives, or PPAs.
* PPAs are simply servers where Linux software is stored and maintained.

Now that we have stopped a malicious service from running and completed a basic audit, we're now going to take a look at some of the miscellaneous utilities that we can install onto our VMs. Specifically we'll be installing the following services:

* emacs is a traditional file editor.
* cowsay is a utility that takes in input, and displays a cow repeating it.
* fortune is a utility that will give you a random proverb with the intention of being interesting to user.

**Installing lynis Demo**

Open a terminal in your lab environment.

* Run: sudo apt install lynis

Since apt is an administrator's command, it must be run with sudo, which allows normal users to perform potentially dangerous tasks, such as installing new software or deleting important files. We'll cover sudo in depth in the next class.

This raises the prompt: Do you want to continue? [y/n]

* Type y for yes. This allows the program to continue.

apt will download and install the package.

* Wait for the command to complete.
* Run sudo lynis --help to show that the command was successfully installed.

Using apt to install software is practically a daily task for sysadmins, and makes installing new software straightforward.

The final activity of the day will be a take home activity to practice installing several packages.

**13. Take Home Activity: Installing Packages**

* [Activity File: Installing Packages](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/1/Activities/13_Installing_Packages/Unsolved/README.md)

**14. Take Home Activity Review: Installing Packages**

* [Solution Guide: Installing Packages](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/1/Activities/13_Installing_Packages/Solved/README.md)

© 2020 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.