Google - CyberSecurity - 4th Chapter

Compare operating systems

You previously explored why operating systems are an important part of how a computer works. In this reading, you'll compare some popular operating systems used today. You'll also focus on the risks of using legacy operating systems.

Common operating systems

The following operating systems are useful to know in the security industry: Windows, macOS®, Linux, ChromeOS, Android, and iOS.

Windows and macOS

Windows and macOS are both common operating systems. The Windows operating system was introduced in 1985, and macOS was introduced in 1984. Both operating systems are used in personal and enterprise computers.

Windows is a closed-source operating system, which means the source code is not shared freely with the public. macOS is partially open source. It has some open-source components, such as macOS's kernel. macOS also has some closed-source components.

Linux

The first version of Linux was released in 1991, and other major releases followed in the early 1990s. Linux is a completely open-source operating system, which means that anyone can access Linux and its source code. The open-source nature of Linux allows developers in the Linux community to collaborate.

Linux is particularly important to the security industry. There are some distributions that are specifically designed for security. Later in this course, you'll learn about Linux and its importance to the security industry.

ChromeOS

ChromeOS launched in 2011. It's partially open source and is derived from Chromium OS, which is completely open source. ChromeOS is frequently used in the education field.

Android and iOS

Android and iOS are both mobile operating systems. Unlike the other operating systems mentioned, mobile operating systems are typically used in mobile devices, such as phones, tablets, and watches. Android was introduced for public use in 2008, and iOS was introduced in 2007. Android is open source, and iOS is partially open source.

Operating systems and vulnerabilities

Security issues are inevitable with all operating systems. An important part of protecting an operating system is keeping the system and all of its components up to date.

Legacy operating systems

A **legacy operating system** is an operating system that is outdated but still being used. Some organizations continue to use legacy operating systems because software they rely on is not compatible with newer

operating systems. This can be more common in industries that use a lot of equipment that requires embedded software—software that's placed inside components of the equipment.

Legacy operating systems can be vulnerable to security issues because they're no longer supported or updated. This means that legacy operating systems might be vulnerable to new threats.

Other vulnerabilities

Even when operating systems are kept up to date, they can still become vulnerable to attack. Below are several resources that include information on operating systems and their vulnerabilities.

- Microsoft Security Response Center (MSRC): A list of known vulnerabilities affecting Microsoft products and services
- Apple Security Updates: A list of security updates and information for Apple® operating systems, including macOS and iOS, and other products
- Common Vulnerabilities and Exposures (CVE) Report for Ubuntu: A list of known vulnerabilities affecting Ubuntu, which is a specific distribution of Linux
- Google Cloud Security Bulletin: A list of known vulnerabilities affecting Google Cloud products and services

Keeping an operating system up to date is one key way to help the system stay secure. Because it can be difficult to keep all systems updated at all times, it's important for security analysts to be knowledgeable about legacy operating systems and the risks they can create.

Requests to the operating system

Operating systems are a critical component of a computer. They make connections between applications and hardware to allow users to perform tasks. In this reading, you'll explore this complex process further and consider it using a new analogy and a new example.

Booting the computer

When you boot, or turn on, your computer, either a BIOS or UEFI microchip is activated. The **Basic Input/Output System (BIOS)** is a microchip that contains loading instructions for the computer and is prevalent in older systems. The **Unified Extensible Firmware Interface (UEFI)** is a microchip that contains loading instructions for the computer and replaces BIOS on more modern systems.

The BIOS and UEFI chips both perform the same function for booting the computer. BIOS was the standard chip until 2007, when UEFI chips increased in use. Now, most new computers include a UEFI chip. UEFI provides enhanced security features.

The BIOS or UEFI microchips contain a variety of loading instructions for the computer to follow. For example, one of the loading instructions is to verify the health of the computer's hardware.

The last instruction from the BIOS or UEFI activates the bootloader. The **bootloader** is a software program that boots the operating system. Once the operating system has finished booting, your computer is ready for use.

Completing a task

As previously discussed, operating systems help us use computers more efficiently. Once a computer has gone through the booting process, completing a task on a computer is a four-part process.

User

The first part of the process is the user. The user initiates the process by having something they want to accomplish on the computer. Right now, you're a user! You've initiated the process of accessing this reading.

Application

The application is the software program that users interact with to complete a task. For example, if you want to calculate something, you would use the calculator application. If you want to write a report, you would use a word processing application. This is the second part of the process.

Operating system

The operating system receives the user's request from the application. It's the operating system's job to interpret the request and direct its flow. In order to complete the task, the operating system sends it on to applicable components of the hardware.

Hardware

The hardware is where all the processing is done to complete the tasks initiated by the user. For example, when a user wants to calculate a number, the CPU figures out the answer. As another example, when a user wants to save a file, another component of the hardware, the hard drive, handles this task.

After the work is done by the hardware, it sends the output back through the operating system to the application so that it can display the results to the user.

The OS at work behind the scenes

Consider once again how a computer is similar to a car. There are processes that someone won't directly observe when operating a car, but they do feel it move forward when they press the gas pedal. It's the same with a computer. Important work happens inside a computer that you don't experience directly. This work involves the operating system.

You can explore this through another analogy. The process of using an operating system is also similar to ordering at a restaurant. At a restaurant you place an order and get your food, but you don't see what's happening in the kitchen when the cooks prepare the food.

Ordering food is similar to using an application on a computer. When you order your food, you make a specific request like "a small soup, very hot." When you use an application, you also make specific requests like "print three double-sided copies of this document."

You can compare the food you receive to what happens when the hardware sends output. You receive the food that you ordered. You receive the document that you wanted to print.

Finally, the kitchen is like the OS. You don't know what happens in the kitchen, but it's critical in interpreting the request and ensuring you receive what you ordered. Similarly, though the work of the OS is not directly transparent to you, it's critical in completing your tasks.

An example: Downloading a file from an internet browser

Previously, you explored how operating systems, applications, and hardware work together by examining a task involving a calculation. You can expand this understanding by exploring how the OS completes another task, downloading a file from an internet browser:

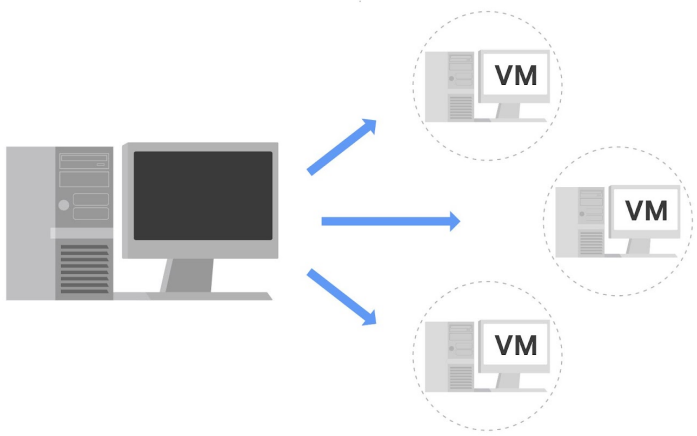
- First, the user decides they want to download a file that they found online, so they click on a download button near the file in the internet browser application.
- Then, the internet browser communicates this action to the OS.
- The OS sends the request to download the file to the appropriate hardware for processing.
- The hardware begins downloading the file, and the OS sends this information to the internet browser application. The internet browser then informs the user when the file has been downloaded.

Virtualization technology

You've explored a lot about operating systems. One more aspect to consider is that operating systems can run on virtual machines. In this reading, you'll learn about virtual machines and the general concept of virtualization. You'll explore how virtual machines work and the benefits of using them.

What is a virtual machine?

A **virtual machine (VM)** is a virtual version of a physical computer. Virtual machines are one example of virtualization. Virtualization is the process of using software to create virtual representations of various physical machines. The term "virtual" refers to machines that don't exist physically, but operate like they do because their software simulates physical hardware. Virtual systems don't use dedicated physical hardware. Instead, they use software-defined versions of the physical hardware. This means that a single virtual machine has a virtual CPU, virtual storage, and other virtual hardware. Virtual systems are just code.



You can run multiple virtual machines using the physical hardware of a single computer. This involves dividing the resources of the host computer to be shared across all physical and virtual components. For example, **Random Access Memory (RAM)** is a hardware component used for short-term memory. If a computer has 16GB of RAM, it can host three virtual machines so that the physical computer and virtual machines each have

4GB of RAM. Also, each of these virtual machines would have their own operating system and function similarly to a typical computer.

Benefits of virtual machines

Security professionals commonly use virtualization and virtual machines. Virtualization can increase security for many tasks and can also increase efficiency.

Security

One benefit is that virtualization can provide an isolated environment, or a sandbox, on the physical host machine. When a computer has multiple virtual machines, these virtual machines are “guests” of the computer. Specifically, they are isolated from the host computer and other guest virtual machines. This provides a layer of security, because virtual machines can be kept separate from the other systems. For example, if an individual virtual machine becomes infected with malware, it can be dealt with more securely because it’s isolated from the other machines. A security professional could also intentionally place malware on a virtual machine to examine it in a more secure environment.

Note: Although using virtual machines is useful when investigating potentially infected machines or running malware in a constrained environment, there are still some risks. For example, a malicious program can escape virtualization and access the host machine. This is why you should never completely trust virtualized systems.

Efficiency

Using virtual machines can also be an efficient and convenient way to perform security tasks. You can open multiple virtual machines at once and switch easily between them. This allows you to streamline security tasks, such as testing and exploring various applications.

You can compare the efficiency of a virtual machine to a city bus. A single city bus has a lot of room and is an efficient way to transport many people simultaneously. If city buses didn’t exist, then everyone on the bus would have to drive their own cars. This uses more gas, cars, and other resources than riding the city bus.

Similar to how many people can ride one bus, many virtual machines can be hosted on the same physical machine. That way, separate physical machines aren't needed to perform certain tasks.

Managing virtual machines

Virtual machines can be managed with a software called a hypervisor. Hypervisors help users manage multiple virtual machines and connect the virtual and physical hardware. Hypervisors also help with allocating the shared resources of the physical host machine to one or more virtual machines.

One hypervisor that is useful for you to be familiar with is the Kernel-based Virtual Machine (KVM). KVM is an open-source hypervisor that is supported by most major Linux distributions. It is built into the Linux kernel, which means it can be used to create virtual machines on any machine running a Linux operating system without the need for additional software.

Other forms of virtualization

In addition to virtual machines, there are other forms of virtualization. Some of these virtualization technologies do not use operating systems. For example, multiple virtual servers can be created from a single physical server. Virtual networks can also be created to more efficiently use the hardware of a physical network.

An user communicates with the operating system via an interface. A **user interface** is a program that allows a user to control the functions of the operating system. Two user interfaces that we'll discuss are the graphical user interface, or GUI, and the command-line interface, or CLI. Let's cover these interfaces in more detail.

GUI - A GUI is a user interface that uses icons on the screen to manage different tasks on the computer. Most operating systems can be used with a graphical user interface. If you've used a personal computer or a cell phone, you have experienced operating a GUI. Most GUIs include these components: a start menu with program groups, a task bar for launching programs, and a desktop with icons and shortcuts. All these components help you communicate with the OS to execute tasks. In addition to clicking on icons, when you use a GUI, you can also search for files or applications from the start menu. You just have to remember the icon or name of the program to activate an application.

CLI - In comparison, the command-line interface, or CLI, is a text-based user interface that uses commands to interact with the computer. These commands communicate with the operating system and execute tasks like opening programs. The command-line interface is a much different structure than the graphical user interface. When you use the CLI, you'll immediately notice a difference. There are no icons or graphics on the screen. The command-line interface looks similar to lines of code using certain text languages. A CLI is more flexible and more powerful than a GUI. Think about using a CLI like creating whatever meal you'd like from ingredients bought at a grocery store. This gives you a lot of control and customization about what you're going to eat.

In comparison, using a GUI is more like ordering food from a restaurant. You can only order what's on the menu. If you want both a noodle dish and pizza, but the first restaurant you go to only has pizza, you'll have to go to another restaurant to order the noodles. With a graphical user interface, you must do one task at a time. But the command-line interface allows for customization, which lets you complete multiple tasks simultaneously. For example, imagine you have a folder with hundreds of files of different file types, and you need to move only the JPEG files to a new folder. Think about how slow and tedious this would be as you use a GUI to find each JPEG file in this folder and move it into the new one. On the other hand, the CLI would allow you to streamline this process and move them all at once.

The command line in use

Previously, you explored graphical user interfaces (GUI) and command-line user interfaces (CLI). In this reading, you'll compare these two interfaces and learn more about how they're used in cybersecurity.

CLI vs. GUI

A **graphical user interface (GUI)** is a user interface that uses icons on the screen to manage different tasks on the computer. A **command-line interface (CLI)** is a text-based user interface that uses commands to interact with the computer.

Display

One notable difference between these two interfaces is how they appear on the screen. A GUI has graphics and icons, such as the icons on your desktop or taskbar for launching programs. In contrast, a CLI only has text. It looks similar to lines of code.

Function

These two interfaces also differ in how they function. A GUI is an interface that only allows you to make one request at a time. However, a CLI allows you to make multiple requests at a time.

Advantages of a CLI in cybersecurity

The choice between using a GUI or CLI is partly based on personal preference, but security analysts should be able to use both interfaces. Using a CLI can provide certain advantages.

Efficiency

Some prefer the CLI because it can be used more quickly when you know how to manage this interface. For a new user, a GUI might be more efficient because they're easier for beginners to navigate.

Because a CLI can accept multiple requests at one time, it's more powerful when you need to perform multiple tasks efficiently. For example, if you had to create multiple new files in your system, you could quickly perform this task in a CLI. If you were using a GUI, this could take much longer, because you have to repeat the same steps for each new file.

History file

For security analysts, using the Linux CLI is helpful because it records a history file of all the commands and actions in the CLI. If you were using a GUI, your actions are not necessarily saved in a history file.

For example, you might be in a situation where you're responding to an incident using a playbook. The playbook's instructions require you to run a series of different commands. If you used a CLI, you'd be able to go back to the history and ensure all of the commands were correctly used. This could be helpful if there were issues using the playbook and you had to review the steps you performed in the command line.

Additionally, if you suspect an attacker has compromised your system, you might be able to trace their actions using the history file.

Linux architecture explained

Understanding the Linux architecture is important for a security analyst. When you understand how a system is organized, it makes it easier to understand how it functions. In this reading, you'll learn more about the individual components in the Linux architecture. A request to complete a task starts with the user and then flows through applications, the shell, the Filesystem Hierarchy Standard, the kernel, and the hardware.

User

The **user** is the person interacting with a computer. They initiate and manage computer tasks. Linux is a multi-user system, which means that multiple users can use the same resources at the same time.

Applications

An **application** is a program that performs a specific task. There are many different applications on your computer. Some applications typically come pre-installed on your computer, such as calculators or calendars. Other applications might have to be installed, such as some web browsers or email clients. In Linux, you'll often use a package manager to install applications. A **package manager** is a tool that helps users install, manage, and remove packages or applications. A **package** is a piece of software that can be combined with other packages to form an application.

Shell

The **shell** is the command-line interpreter. Everything entered into the shell is text based. The shell allows users to give commands to the kernel and receive responses from it. You can think of the shell as a translator between you and your computer. The shell translates the commands you enter so that the computer can perform the tasks you want.

Filesystem Hierarchy Standard (FHS)

The **Filesystem Hierarchy Standard (FHS)** is the component of the Linux OS that organizes data. It specifies the location where data is stored in the operating system.

A **directory** is a file that organizes where other files are stored. Directories are sometimes called “folders,” and they can contain files or other directories. The FHS defines how directories, directory contents, and other storage is organized so the operating system knows where to find specific data.

Kernel

The **kernel** is the component of the Linux OS that manages processes and memory. It communicates with the applications to route commands. The Linux kernel is unique to the Linux OS and is critical for allocating resources in the system. The kernel controls all major functions of the hardware, which can help get tasks expedited more efficiently.

Hardware

The **hardware** is the physical components of a computer. You might be familiar with some hardware components, such as hard drives or CPUs. Hardware is categorized as either peripheral or internal.

Peripheral devices

Peripheral devices are hardware components that are attached and controlled by the computer system. They are not core components needed to run the computer system. Peripheral devices can be added or removed freely. Examples of peripheral devices include monitors, printers, the keyboard, and the mouse.

Internal hardware

Internal hardware are the components required to run the computer. Internal hardware includes a main circuit board and all components attached to it. This main circuit board is also called the motherboard. Internal hardware includes the following:

- The **Central Processing Unit (CPU)** is a computer’s main processor, which is used to perform general computing tasks on a computer. The CPU executes the instructions provided by programs, which enables these programs to run.
- **Random Access Memory (RAM)** is a hardware component used for short-term memory. It’s where data is stored temporarily as you perform tasks on your computer. For example, if you’re writing a report on your computer, the data needed for this is stored in RAM. After you’ve finished writing the report and closed down that program, this data is deleted from RAM. Information in RAM cannot be accessed once the computer has been turned off. The CPU takes the data from RAM to run programs.
- The **hard drive** is a hardware component used for long-term memory. It’s where programs and files are stored for the computer to access later. Information on the hard drive can be accessed even after a computer has been turned off and on again. A computer can have multiple hard drives.

More Linux distributions

Previously, you were introduced to the different distributions of Linux. This included KALI LINUX ™. (KALI LINUX ™ is a trademark of OffSec.) In addition to KALI LINUX ™, there are multiple other Linux distributions that security analysts should be familiar with. In this reading, you'll learn about additional Linux distributions.

KALI LINUX ™

KALI LINUX ™ is an open-source distribution of Linux that is widely used in the security industry. This is because KALI LINUX ™, which is Debian-based, is pre-installed with many useful tools for penetration testing and digital forensics. A **penetration test** is a simulated attack that helps identify vulnerabilities in systems, networks, websites, applications, and processes. **Digital forensics** is the practice of collecting and analyzing data to determine what has happened after an attack. These are key activities in the security industry.

However, KALI LINUX ™ is not the only Linux distribution that is used in cybersecurity.

Ubuntu

Ubuntu is an open-source, user-friendly distribution that is widely used in security and other industries. It has both a command-line interface (CLI) and a graphical user interface (GUI). Ubuntu is also Debian-derived and includes common applications by default. Users can also download many more applications from a package manager, including security-focused tools. Because of its wide use, Ubuntu has an especially large number of community resources to support users.

Ubuntu is also widely used for cloud computing. As organizations migrate to cloud servers, cybersecurity work may more regularly involve Ubuntu derivatives.

Parrot

Parrot is an open-source distribution that is commonly used for security. Similar to KALI LINUX ™, Parrot comes with pre-installed tools related to penetration testing and digital forensics. Like both KALI LINUX ™ and Ubuntu, it is based on Debian.

Parrot is also considered to be a user-friendly Linux distribution. This is because it has a GUI that many find easy to navigate. This is in addition to Parrot's CLI.

Red Hat® Enterprise Linux®

Red Hat Enterprise Linux is a subscription-based distribution of Linux built for enterprise use. Red Hat is not free, which is a major difference from the previously mentioned distributions. Because it's built and supported for enterprise use, Red Hat also offers a dedicated support team for customers to call about issues.

CentOS

CentOS is an open-source distribution that is closely related to Red Hat. It uses source code published by Red Hat to provide a similar platform. However, CentOS does not offer the same enterprise support that Red Hat provides and is supported through the community.

Package managers for installing applications

Previously, you learned about Linux distributions and that different distributions derive from different sources, such as Debian or Red Hat Enterprise Linux distribution. You were also introduced to package managers, and

learned that Linux applications are commonly distributed through package managers. In this reading, you'll apply this knowledge to learn more about package managers.

Introduction to package managers

A **package** is a piece of software that can be combined with other packages to form an application. Some packages may be large enough to form applications on their own.

Packages contain the files necessary for an application to be installed. These files include dependencies, which are supplemental files used to run an application.

Package managers can help resolve any issues with dependencies and perform other management tasks. A **package manager** is a tool that helps users install, manage, and remove packages or applications. Linux uses multiple package managers.

Note: It's important to use the most recent version of a package when possible. The most recent version has the most up-to-date bug fixes and security patches. These help keep your system more secure.

Types of package managers

Many commonly used Linux distributions are derived from the same parent distribution. For example, KALI LINUX ™, Ubuntu, and Parrot all come from Debian. CentOS comes from Red Hat.

This knowledge is useful when installing applications because certain package managers work with certain distributions. For example, the Red Hat Package Manager (RPM) can be used for Linux distributions derived from Red Hat, and package managers such as dpkg can be used for Linux distributions derived from Debian.

Different package managers typically use different file extensions. For example, Red Hat Package Manager (RPM) has files which use the **.rpm** file extension, such as **Package-Version-Release_Architecture.rpm**. Package managers for Debian-derived Linux distributions, such as dpkg, have files which use the **.deb** file extension, such as **Package_Version-Release_Architecture.deb**.

Package management tools

In addition to package managers like RPM and dpkg, there are also package management tools that allow you to easily work with packages through the shell. Package management tools are sometimes utilized instead of package managers because they allow users to more easily perform basic tasks, such as installing a new package. Two notable tools are the Advanced Package Tool (APT) and Yellowdog Updater Modified (YUM).

Advanced Package Tool (APT)

APT is a tool used with Debian-derived distributions. It is run from the command-line interface to manage, search, and install packages.

Yellowdog Updater Modified (YUM)

YUM is a tool used with Red Hat-derived distributions. It is run from the command-line interface to manage, search, and install packages. YUM works with **.rpm** files.

Different types of shells

Knowing how to work with Linux shells is an important skill for cybersecurity professionals. Shells can be used for many common tasks. Previously, you were introduced to shells and their functions. This reading will review shells and introduce you to different types, including the one that you'll use in this course.

Communicate through a shell

As you explored previously, the **shell** is the command-line interpreter. You can think of a shell as a translator between you and the computer system. Shells allow you to give commands to the computer and receive responses from it. When you enter a command into a shell, the shell executes many internal processes to interpret your command, send it to the kernel, and return your results.

Types of shells

The many different types of Linux shells include the following:

- Bourne-Again Shell (bash)
- C Shell (csh)
- Korn Shell (ksh)
- Enhanced C shell (tcsh)
- Z Shell (zsh)

All Linux shells use common Linux commands, but they can differ in other features. For example, ksh and bash use the dollar sign (**$**) to indicate where users type in their commands. Other shells, such as zsh, use the percent sign (**%**) for this purpose.

Bash

Bash is the default shell in most Linux distributions. It's considered a user-friendly shell. You can use bash for basic Linux commands as well as larger projects.

Bash is also the most popular shell in the cybersecurity profession. You'll use bash throughout this course as you learn and practice Linux commands.

Shells are a fundamental part of the Linux operating system. Shells allow you to give commands to the computer and receive responses from it. They can be thought of as a translator between you and your computer system. There are many different types of shells, but the bash shell is the most commonly used shell in the cybersecurity profession. You'll learn how to enter Linux commands through the bash shell later in this course.

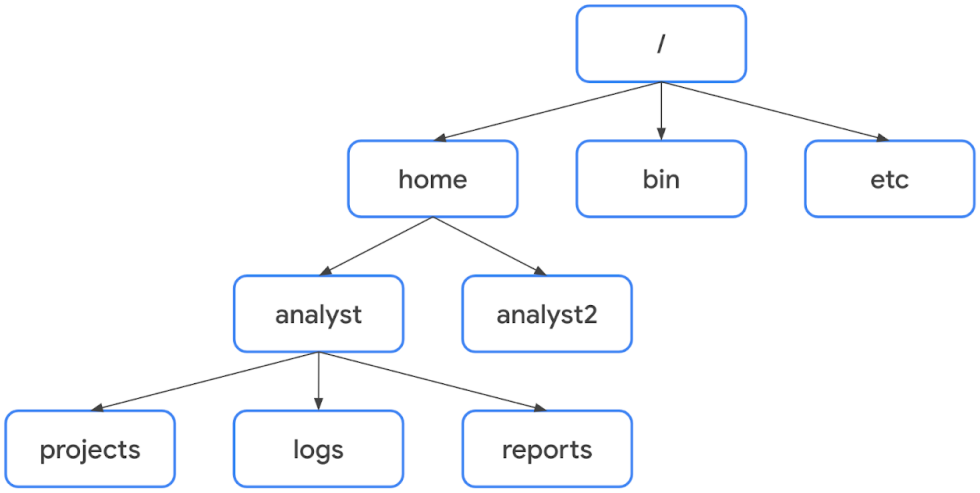
Navigate Linux and read file content

In this reading, you'll review how to navigate the file system using Linux commands in Bash. You'll further explore the organization of the Linux Filesystem Hierarchy Standard, review several common Linux commands for navigation and reading file content, and learn a couple of new commands.

Filesystem Hierarchy Standard (FHS)

Previously, you learned that the **Filesystem Hierarchy Standard (FHS)** is the component of Linux that organizes data. The FHS is important because it defines how directories, directory contents, and other storage is organized in the operating system.

This diagram illustrates the hierarchy of relationships under the FHS:



Under the FHS, a file's location can be described by a file path. A **file path** is the location of a file or directory. In the file path, the different levels of the hierarchy are separated by a forward slash (**/**).

Root directory

The **root directory** is the highest-level directory in Linux, and it's always represented with a forward slash (**/**). All subdirectories branch off the root directory. Subdirectories can continue branching out to as many levels as necessary.

Standard FHS directories

Directly below the root directory, you'll find standard FHS directories. In the diagram, **home**, **bin**, and **etc** are standard FHS directories. Here are a few examples of what standard directories contain:

- **/home**: Each user in the system gets their own home directory.
- **/bin**: This directory stands for "binary" and contains binary files and other executables. Executables are files that contain a series of commands a computer needs to follow to run programs and perform other functions.
- **/etc**: This directory stores the system's configuration files.
- **/tmp**: This directory stores many temporary files. The **/tmp** directory is commonly used by attackers because anyone in the system can modify data in these files.
- **/mnt**: This directory stands for "mount" and stores media, such as USB drives and hard drives.

Pro Tip: You can use the **man hier** command to learn more about the FHS and its standard directories.

User-specific subdirectories

Under **home** are subdirectories for specific users. In the diagram, these users are **analyst** and **analyst2**. Each user has their own personal subdirectories, such as **projects**, **logs**, or **reports**.

Note: When the path leads to a subdirectory below the user's home directory, the user's home directory can be represented as the tilde (**~**). For example, **/home/analyst/logs** can also be represented as **~/logs**.

You can navigate to specific subdirectories using their absolute or relative file paths. The **absolute file path** is the full file path, which starts from the root. For example, **/home/analyst/projects** is an absolute file path. The **relative file path** is the file path that starts from a user's current directory.

Note: Relative file paths can use a dot (**.**) to represent the current directory, or two dots (**..**) to represent the parent of the current directory. An example of a relative file path could be **../projects**.

Key commands for navigating the file system

The following Linux commands can be used to navigate the file system: **pwd**, **ls**, and **cd**.

pwd

The **pwd** command prints the working directory to the screen. Or in other words, it returns the directory that you're currently in.

The output gives you the absolute path to this directory. For example, if you're in your **home** directory and your username is **analyst**, entering **pwd** returns **/home/analyst**.

Pro Tip: To learn what your username is, use the **whoami** command. The **whoami** command returns the username of the current user. For example, if your username is **analyst**, entering **whoami** returns **analyst**.

ls

The **ls** command displays the names of the files and directories in the current working directory. For example, in the video, **ls** returned directories such as **logs**, and a file called **updates.txt**.

Note: If you want to return the contents of a directory that's not your current working directory, you can add an argument after **ls** with the absolute or relative file path to the desired directory. For example, if you're in the **/home/analyst** directory but want to list the contents of its **projects** subdirectory, you can enter **ls /home/analyst/projects** or just **ls projects**.

cd

The **cd** command navigates between directories. When you need to change directories, you should use this command.

To navigate to a subdirectory of the current directory, you can add an argument after **cd** with the subdirectory name. For example, if you're in the **/home/analyst** directory and want to navigate to its **projects** subdirectory, you can enter **cd projects**.

You can also navigate to any specific directory by entering the absolute file path. For example, if you're in **/home/analyst/projects**, entering **cd /home/analyst/logs** changes your current directory to **/home/analyst/logs**.

Pro Tip: You can use the relative file path and enter **cd ..** to go up one level in the file structure. For example, if the current directory is **/home/analyst/projects**, entering **cd ..** would change your working directory to **/home/analyst**.

Common commands for reading file content

The following Linux commands are useful for reading file content: **cat**, **head**, **tail**, and **less**.

cat

The **cat** command displays the content of a file. For example, entering **cat updates.txt** returns everything in the **updates.txt** file.

head

The **head** command displays just the beginning of a file, by default 10 lines. The **head** command can be useful when you want to know the basic contents of a file but don't need the full contents. Entering **head updates.txt** returns only the first 10 lines of the **updates.txt** file.

Pro Tip: If you want to change the number of lines returned by **head**, you can specify the number of lines by including **-n**. For example, if you only want to display the first five lines of the **updates.txt** file, enter **head -n 5 updates.txt**.

tail

The **tail** command does the opposite of **head**. This command can be used to display just the end of a file, by default 10 lines. Entering **tail updates.txt** returns only the last 10 lines of the **updates.txt** file.

Pro Tip: You can use **tail** to read the most recent information in a log file.

less

The **less** command returns the content of a file one page at a time. For example, entering **less updates.txt** changes the terminal window to display the contents of **updates.txt** one page at a time. This allows you to easily move forward and backward through the content.

Once you've accessed your content with the **less** command, you can use several keyboard controls to move through the file:

- **Space bar**: Move forward one page
- **b**: Move back one page
- **Down arrow**: Move forward one line
- **Up arrow**: Move back one line
- **q**: Quit and return to the previous terminal window

It's important for security analysts to be able to navigate Linux and the file system of the FHS. Some key commands for navigating the file system include **pwd**, **ls**, and **cd**. Reading file content is also an important skill in the security profession. This can be done with commands such as **cat**, **head**, **tail**, and **less**.

Filter content in Linux

In this reading, you'll continue exploring Linux commands, which can help you filter for the information you need. You'll learn a new Linux command, **find**, which can help you search files and directories for specific information.

Filtering for information

You previously explored how filtering for information is an important skill for security analysts. **Filtering** is selecting data that match a certain condition. For example, if you had a virus in your system that only affected the **.txt** files, you could use filtering to find these files quickly. Filtering allows you to search based on specific criteria, such as file extension or a string of text.

grep

The **grep** command searches a specified file and returns all lines in the file containing a specified string. The **grep** command commonly takes two arguments: a specific string to search for and a specific file to search through.

For example, entering **grep OS updates.txt** returns all lines containing **OS** in the **updates.txt** file. In this example, **OS** is the specific string to search for, and **updates.txt** is the specific file to search through.

Piping

The pipe command is accessed using the pipe character (**|**). **Piping** sends the standard output of one command as standard input to another command for further processing. As a reminder, **standard output** is information returned by the OS through the shell, and **standard input** is information received by the OS via the command line.

The pipe character (**|**) is located in various places on a keyboard. On many keyboards, it's located on the same key as the backslash character (****). On some keyboards, the **|** can look different and have a small space through the middle of the line. If you can't find the **|**, search online for its location on your particular keyboard.

When used with **grep**, the pipe can help you find directories and files containing a specific word in their names. For example, **ls /home/analyst/reports | grep users** returns the file and directory names in the **reports** directory that contain **users**. Before the pipe, **ls** indicates to list the names of the files and directories in **reports**. Then, it sends this output to the command after the pipe. In this case, **grep users** returns all of the file or directory names containing **users** from the input it received.

Note: Piping is a general form of redirection in Linux and can be used for multiple tasks other than filtering. You can think of piping as a general tool that you can use whenever you want the output of one command to become the input of another command.

find

The **find** command searches for directories and files that meet specified criteria. There's a wide range of criteria that can be specified with **find**. For example, you can search for files and directories that

- Contain a specific string in the name,
- Are a certain file size, or
- Were last modified within a certain time frame.

When using **find**, the first argument after **find** indicates where to start searching. For example, entering **find /home/analyst/projects** searches for everything starting at the **projects** directory.

After this first argument, you need to indicate your criteria for the search. If you don't include a specific search criteria with your second argument, your search will likely return a lot of directories and files.

Specifying criteria involves options. **Options** modify the behavior of a command and commonly begin with a hyphen (**-**).

name and -iname

One key criteria analysts might use with **find** is to find file or directory names that contain a specific string. The specific string you're searching for must be entered in quotes after the **-name** or **-iname** options. The difference between these two options is that **-name** is case-sensitive, and **-iname** is not.

For example, you might want to find all files in the **projects** directory that contain the word "log" in the file name. To do this, you'd enter **find /home/analyst/projects -name "*log*"**. You could also enter **find /home/analyst/projects -iname "*log*"**.

In these examples, the output would be all files in the **projects** directory that contain **log** surrounded by zero or more characters. The **"*log*"** portion of the command is the search criteria that indicates to search for the

string “log”. When **-name** is the option, files with names that include **Log** or **LOG**, for example, wouldn’t be returned because this option is case-sensitive. However, they would be returned when **-iname** is the option.

Note: An asterisk (*****) is used as a wildcard to represent zero or more unknown characters.

mtime

Security analysts might also use **find** to find files or directories last modified within a certain time frame. The **-mtime** option can be used for this search. For example, entering **find /home/analyst/projects -mtime -3** returns all files and directories in the **projects** directory that have been modified within the past three days.

The **-mtime** option search is based on days, so entering **-mtime +1** indicates all files or directories last modified more than one day ago, and entering **-mtime -1** indicates all files or directories last modified less than one day ago.

Note: The option **-mmin** can be used instead of **-mtime** if you want to base the search on minutes rather than days.

Filtering for information using Linux commands is an important skill for security analysts so that they can customize data to fit their needs. Three key Linux commands for this are **grep**, piping (**|**), and **find**. These commands can be used to navigate and filter for information in the file system.

Manage directories and files

Previously, you explored how to manage the file system using Linux commands. The following commands were introduced: **mkdir**, **rmdir**, **touch**, **rm**, **mv**, and **cp**. In this reading, you’ll review these commands, the nano text editor, and learn another way to write to files.

Creating and modifying directories

mkdir

The **mkdir** command creates a new directory. Like all of the commands presented in this reading, you can either provide the new directory as the absolute file path, which starts from the root, or as a relative file path, which starts from your current directory.

For example, if you want to create a new directory called **network** in your **/home/analyst/logs** directory, you can enter **mkdir /home/analyst/logs/network** to create this new directory. If you’re already in the **/home/analyst/logs** directory, you can also create this new directory by entering **mkdir network**.

Pro Tip: You can use the **ls** command to confirm the new directory was added.

rmdir

The **rmdir** command removes, or deletes, a directory. For example, entering **rmdir /home/analyst/logs/network** would remove this empty directory from the file system.

Note: The **rmdir** command cannot delete directories with files or subdirectories inside. For example, entering **rmdir /home/analyst** returns an error message.

Creating and modifying files

touch and rm

The **touch** command creates a new file. This file won't have any content inside. If your current directory is **/home/analyst/reports**, entering **touch permissions.txt** creates a new file in the **reports** subdirectory called **permissions.txt**.

The **rm** command removes, or deletes, a file. This command should be used carefully because it's not easy to recover files deleted with **rm**. To remove the permissions file you just created, enter **rm permissions.txt**.

Pro Tip: You can verify that **permissions.txt** was successfully created or removed by entering **ls**.

mv and cp

You can also use **mv** and **cp** when working with files. The **mv** command moves a file or directory to a new location, and the **cp** command copies a file or directory into a new location. The first argument after **mv** or **cp** is the file or directory you want to move or copy, and the second argument is the location you want to move or copy it to.

To move **permissions.txt** into the **logs** subdirectory, enter **mv permissions.txt /home/analyst/logs**. Moving a file removes the file from its original location. However, copying a file doesn't remove it from its original location. To copy **permissions.txt** into the **logs** subdirectory while also keeping it in its original location, enter **cp permissions.txt /home/analyst/logs**.

Note: The **mv** command can also be used to rename files. To rename a file, pass the new name in as the second argument instead of the new location. For example, entering **mv permissions.txt perm.txt** renames the **permissions.txt** file to **perm.txt**.

nano text editor

nano is a command-line file editor that is available by default in many Linux distributions. Many beginners find it easy to use, and it's widely used in the security profession. You can perform multiple basic tasks in nano, such as creating new files and modifying file contents.

To open an existing file in nano from the directory that contains it, enter **nano** followed by the file name. For example, entering **nano permissions.txt** from the **/home/analyst/reports** directory opens a new nano editing window with the **permissions.txt** file open for editing. You can also provide the absolute file path to the file if you're not in the directory that contains it.

You can also create a new file in nano by entering **nano** followed by a new file name. For example, entering **nano authorized_users.txt** from the **/home/analyst/reports** directory creates the **authorized_users.txt** file within that directory and opens it in a new nano editing window.

Since there isn't an auto-saving feature in nano, it's important to save your work before exiting. To save a file in nano, use the keyboard shortcut **Ctrl + O**. You'll be prompted to confirm the file name before saving. To exit out of nano, use the keyboard shortcut **Ctrl + X**.

Note: Vim and Emacs are also popular command-line text editors.

Standard output redirection

There's an additional way you can write to files. Previously, you learned about standard input and standard output. **Standard input** is information received by the OS via the command line, and **standard output** is information returned by the OS through the shell.

You've also learned about piping. **Piping** sends the standard output of one command as standard input to another command for further processing. It uses the pipe character (|).

In addition to the pipe (**|**), you can also use the right angle bracket (**>**) and double right angle bracket (**>>**) operators to redirect standard output.

When used with **echo**, the **>** and **>>** operators can be used to send the output of **echo** to a specified file rather than the screen. The difference between the two is that **>** overwrites your existing file, and **>>** adds your content to the end of the existing file instead of overwriting it. The **>** operator should be used carefully, because it's not easy to recover overwritten files.

When you're inside the directory containing the **permissions.txt** file, entering **echo "last updated date" >> permissions.txt** adds the string "last updated date" to the file contents. Entering **echo "time" > permissions.txt** after this command overwrites the entire file contents of **permissions.txt** with the string "time".

Note: Both the **>** and **>>** operators will create a new file if one doesn't already exist with your specified name.

Knowing how to manage the file system in Linux is an important skill for security analysts. Useful commands for this include: mkdir, rmdir, touch, rm, mv, and cp. When security analysts need to write to files, they can use the nano text editor, or the > and >> operators.

Permission commands

Previously, you explored file permissions and the commands that you can use to display and change them. In this reading, you'll review these concepts and also focus on an example of how these commands work together when putting the principle of least privilege into practice.

Reading permissions

In Linux, permissions are represented with a 10-character string. Permissions include:

- **read**: for files, this is the ability to read the file contents; for directories, this is the ability to read all contents in the directory including both files and subdirectories
- **write**: for files, this is the ability to make modifications on the file contents; for directories, this is the ability to create new files in the directory
- **execute**: for files, this is the ability to execute the file if it's a program; for directories, this is the ability to enter the directory and access its files

These permissions are given to these types of owners:

- **user**: the owner of the file
- **group**: a larger group that the owner is a part of
- **other**: all other users on the system

Each character in the 10-character string conveys different information about these permissions. The following table describes the purpose of each character:

Character	Example	Meaning
1st	**d**rwxrwxrwx	file type • **d** for directory

		• **-** for a regular file
2nd	d**r**wxrwxrwx	read permissions for the user • **r** if the user has read permissions • **-** if the user lacks read permissions
3rd	dr**w**xrwxrwx	write permissions for the user • **w** if the user has write permissions • **-** if the user lacks write permissions
4th	drw**x**rwxrwx	execute permissions for the user • **x** if the user has execute permissions • **-** if the user lacks execute permissions
5th	drwx**r**wxrwx	read permissions for the group • **r** if the group has read permissions • **-** if the group lacks read permissions
6th	drwxr**w**xrwx	write permissions for the group • **w** if the group has write permissions • **-** if the group lacks write permissions
7th	drwxrw**x**rwx	execute permissions for the group • **x** if the group has execute permissions • **-** if the group lacks execute permissions
8th	drwxrwx**r**wx	read permissions for other • **r** if the other owner type has read permissions • **-** if the other owner type lacks read permissions
9th	drwxrwxr**w**x	write permissions for other • **w** if the other owner type has write permissions • **-** if the other owner type lacks write permissions
10th	drwxrwxrw**x**	execute permissions for other • **x** if the other owner type has execute permissions • **-** if the other owner type lacks execute permissions

Exploring existing permissions

You can use the **ls** command to investigate who has permissions on files and directories. Previously, you learned that **ls** displays the names of files in directories in the current working directory.

There are additional options you can add to the **ls** command to make your command more specific. Some of these options provide details about permissions. Here are a few important **ls** options for security analysts:

- **ls -a**: Displays hidden files. Hidden files start with a period (**.**) at the beginning.
- **ls -l**: Displays permissions to files and directories. Also displays other additional information, including owner name, group, file size, and the time of last modification.
- **ls -la**: Displays permissions to files and directories, including hidden files. This is a combination of the other two options.

Changing permissions

The **principle of least privilege** is the concept of granting only the minimal access and authorization required to complete a task or function. In other words, users should not have privileges that are beyond what is necessary. Not following the principle of least privilege can create security risks.

The **chmod** command can help you manage this authorization. The **chmod** command changes permissions on files and directories.

Using chmod

The **chmod** command requires two arguments. The first argument indicates how to change permissions, and the second argument indicates the file or directory that you want to change permissions for. For example, the following command would add all permissions to **login_sessions.txt**:

chmod u+rwx,g+rwx,o+rwx login_sessions.txt

If you wanted to take all the permissions away, you could use

chmod u-rwx,g-rwx,o-rwx login_sessions.txt

Another way to assign these permissions is to use the equals sign (**=**) in this first argument. Using **=** with **chmod** sets, or assigns, the permissions exactly as specified. For example, the following command would set read permissions for **login_sessions.txt** for user, group, and other:

chmod u=r,g=r,o=r login_sessions.txt

This command overwrites existing permissions. For instance, if the user previously had write permissions, these write permissions are removed after you specify only read permissions with =.

The following table reviews how each character is used within the first argument of **chmod**:

Character	Description
u	indicates changes will be made to user permissions
g	indicates changes will be made to group permissions
o	indicates changes will be made to other permissions
+	adds permissions to the user, group, or other
-	removes permissions from the user, group, or other
=	assigns permissions for the user, group, or other

Note: When there are permission changes to more than one owner type, commas are needed to separate changes for each owner type. You should not add spaces after those commas.

The principle of least privilege in action

As a security analyst, you may encounter a situation like this one: There's a file called **bonuses.txt** within a compensation directory. The owner of this file is a member of the Human Resources department with a username of **hrrep1**. It has been decided that **hrrep1** needs access to this file. But, since this file contains confidential information, no one else in the **hr** group needs access.

You run **ls -l** to check the permissions of files in the compensation directory and discover that the permissions for **bonuses.txt** are **-rw-rw----**. The group owner type has read and write permissions that do not align with the principle of least privilege.

To remedy the situation, you input **chmod g-rw bonuses.txt**. Now, only the user who needs to access this file to carry out their job responsibilities can access this file.

Managing directory and file permissions may be a part of your work as a security analyst. Using **ls** with the **-l** and **-la** options allows you to investigate directory and file permissions. Using **chmod** allows you to change user permissions and ensure they are aligned with the principle of least privilege.

Activity - 1

Responsible use of sudo

Previously, you explored authorization, authentication, and Linux commands with **sudo**, **useradd**, and **userdel**. The **sudo** command is important for security analysts because it allows users to have elevated permissions without risking the system by running commands as the root user. You'll continue exploring authorization, authentication, and Linux commands in this reading and learn two more commands that can be used with **sudo**: **usermod** and **chown**.

Responsible use of sudo

To manage authorization and authentication, you need to be a **root user,** or a user with elevated privileges to modify the system. The root user can also be called the "super user." You become a root user by logging in as the root user. However, running commands as the root user is not recommended in Linux because it can create security risks if malicious actors compromise that account. It's also easy to make irreversible mistakes, and the system can't track who ran a command. For these reasons, rather than logging in as the root user, it's recommended you use **sudo** in Linux when you need elevated privileges.

The **sudo** command temporarily grants elevated permissions to specific users. The name of this command comes from "super user do." Users must be given access in a configuration file to use **sudo**. This file is called the "sudoers file." Although using **sudo** is preferable to logging in as the root user, it's important to be aware that users with the elevated permissions to use **sudo** might be more at risk in the event of an attack.

You can compare this to a hotel with a master key. The master key can be used to access any room in the hotel. There are some workers at the hotel who need this key to perform their work. For example, to clean all the rooms, the janitor would scan their ID badge and then use this master key. However, if someone outside the hotel's network gained access to the janitor's ID badge and master key, they could access any room in the hotel. In this example, the janitor with the master key represents a user using **sudo** for elevated privileges. Because of the dangers of **sudo**, only users who really need to use it should have these permissions.

Additionally, even if you need access to **sudo**, you should be careful about using it with only the commands you need and nothing more. Running commands with **sudo** allows users to bypass the typical security controls that are in place to prevent elevated access to an attacker.

Note: Be aware of **sudo** if copying commands from an online source. It's important you don't use **sudo** accidentally.

Authentication and authorization with sudo

You can use **sudo** with many authentication and authorization management tasks. As a reminder, **authentication** is the process of verifying who someone is, and **authorization** is the concept of granting access to specific resources in a system. Some of the key commands used for these tasks include the following:

useradd

The **useradd** command adds a user to the system. To add a user with the username of **fgarcia** with **sudo**, enter **sudo useradd fgarcia**. There are additional options you can use with **useradd**:

- **g**: Sets the user's default group, also called their primary group
- **G**: Adds the user to additional groups, also called supplemental or secondary groups

To use the **-g** option, the primary group must be specified after **-g**. For example, entering **sudo useradd -g security fgarcia** adds **fgarcia** as a new user and assigns their primary group to be **security**.

To use the **-G** option, the supplemental group must be passed into the command after **-G**. You can add more than one supplemental group at a time with the **-G** option. Entering **sudo useradd -G finance,admin fgarcia** adds **fgarcia** as a new user and adds them to the existing **finance** and **admin** groups.

usermod

The **usermod** command modifies existing user accounts. The same **-g** and **-G** options from the **useradd** command can be used with **usermod** if a user already exists.

To change the primary group of an existing user, you need the **-g** option. For example, entering **sudo usermod -g executive fgarcia** would change **fgarcia**'s primary group to the **executive** group.

To add a supplemental group for an existing user, you need the **-G** option. You also need a **-a** option, which appends the user to an existing group and is only used with the **-G** option. For example, entering **sudo usermod -a -G marketing fgarcia** would add the existing **fgarcia** user to the supplemental **marketing** group.

Note: When changing the supplemental group of an existing user, if you don't include the **-a** option, **-G** will replace any existing supplemental groups with the groups specified after **usermod**. Using **-a** with **-G** ensures that the new groups are added but existing groups are not replaced.

There are other options you can use with **usermod** to specify how you want to modify the user, including:

- **d**: Changes the user's home directory.
- **l**: Changes the user's login name.
- **L**: Locks the account so the user can't log in.

The option always goes after the **usermod** command. For example, to change **fgarcia**'s home directory to **/home/garcia_f**, enter **sudo usermod -d /home/garcia_f fgarcia**. The option **-d** directly follows the command **usermod** before the other two needed arguments.

userdel

The **userdel** command deletes a user from the system. For example, entering **sudo userdel fgarcia** deletes **fgarcia** as a user. Be careful before you delete a user using this command.

The **userdel** command doesn't delete the files in the user's home directory unless you use the **-r** option. Entering **sudo userdel -r fgarcia** would delete **fgarcia** as a user and delete all files in their home directory. Before deleting any user files, you should ensure you have backups in case you need them later.

Note: Instead of deleting the user, you could consider deactivating their account with **usermod -L**. This prevents the user from logging in while still giving you access to their account and associated permissions. For example, if a user left an organization, this option would allow you to identify which files they have ownership over, so you could move this ownership to other users.

Note: *When you create a new user in Linux, a group with the same name as the user is automatically created and the user is the only member of that group. After removing users, it is good practice to clean up any such empty groups that may remain behind.*

Therfore use example : (sudo groupdel researcher9)

chown

The **chown** command changes ownership of a file or directory. You can use **chown** to change user or group ownership. To change the user owner of the **access.txt** file to **fgarcia**, enter **sudo chown fgarcia access.txt**. To change the group owner of **access.txt** to **security**, enter **sudo chown :security access.txt**. You must enter a colon (**:**) before **security** to designate it as a group name.

Similar to **useradd**, **usermod**, and **userdel**, there are additional options that can be used with **chown**.

Authentication is the process of a user verifying their identity, and authorization is the process of determining what they have access to. You can use the **sudo** command to temporarily run commands with elevated privileges to complete authentication and authorization management tasks. Specifically, **useradd**, **userdel, usermod**, and **chown** can be used to manage users and file ownership.

Linux resources

Previously, you were introduced to the Linux community and some resources that exist to help Linux users. Linux has many options available to give users the information they need. This reading will review these resources. When you're aware of the resources available to you, you can continue to learn Linux independently. You can also discover even more ways that Linux can support your work as a security analyst.

Linux community

Linux has a large online community, and this is a huge resource for Linux users of all levels. You can likely find the answers to your questions with a simple online search. Troubleshooting issues by searching and reading online is an effective way to discover how others approached your issue. It's also a great way for beginners to learn more about Linux.

The UNIX and Linux Stack Exchange is a trusted resource for troubleshooting Linux issues. The Unix and Linux Stack Exchange is a question and answer website where community members can ask and answer questions about Linux. Community members vote on answers, so the higher quality answers are displayed at the top. Many of the questions are related to specific topics from advanced users, and the topics might help you troubleshoot issues as you continue using Linux.

Integrated Linux support

Linux also has several commands that you can use for support.

man

The **man** command displays information on other commands and how they work. It's short for "manual." To search for information on a command, enter the command after **man**. For example, entering **man chown** returns detailed information about **chown**, including the various options you can use with it. The output of the **man** command is also called a "man page."

apropos

The **apropos** command searches the man page descriptions for a specified string. Man pages can be lengthy and difficult to search through if you're looking for a specific keyword. To use **apropos**, enter the keyword after **apropos**.

You can also include the **-a** option to search for multiple words. For example, entering **apropos -a graph editor** outputs man pages that contain both the words "graph" and "editor" in their descriptions.

whatis

The **whatis** command displays a description of a command on a single line. For example, entering **whatis nano** outputs the description of **nano**. This command is useful when you don't need a detailed description, just a general idea of the command. This might be as a reminder. Or, it might be after you discover a new command through a colleague or online resource and want to know more.

Key takeaways

There are many resources available for troubleshooting issues or getting support for Linux. Linux has a large global community of users who ask and answer questions on online resources, such as the Unix and Linux Stack Exchange. You can also use integrated support commands in Linux, such as **man**, **apropos**, and **whatis**.

SQL filtering versus Linux filtering

Previously, you explored the Linux commands that allow you to filter for specific information contained within files or directories. And, more recently, you examined how SQL helps you efficiently filter for the information you need. In this reading, you'll explore differences between the two tools as they relate to filtering. You'll also learn that one way to access SQL is through the Linux command line.

Accessing SQL

There are many interfaces for accessing SQL and many different versions of SQL. One way to access SQL is through the Linux command line.

To access SQL from Linux, you need to type in a command for the version of SQL that you want to use. For example, if you want to access SQLite, you can enter the command **sqlite3** in the command line.

After this, any commands typed in the command line will be directed to SQL instead of Linux commands.

Differences between Linux and SQL filtering

Although both Linux and SQL allow you to filter through data, there are some differences that affect which one you should choose.

Purpose

Linux filters data in the context of files and directories on a computer system. It's used for tasks like searching for specific files, manipulating file permissions, or managing processes.

SQL is used to filter data within a database management system. It's used for querying and manipulating data stored in tables and retrieving specific information based on defined criteria.

Syntax

Linux uses various commands and command-line options specific to each filtering tool. Syntax varies depending on the tool and purpose. Some examples of Linux commands are find, sed, cut, e grep

SQL uses the Structured Query Language (SQL), a standardized language with specific keywords and clauses for filtering data across different SQL databases. Some examples of SQL keywords and clauses are WHERE, SELECT, JOIN

Structure

SQL offers a lot more structure than Linux, which is more free-form and not as tidy.

For example, if you wanted to access a log of employee log-in attempts, SQL would have each record separated into columns. Linux would print the data as a line of text without this organization. As a result, selecting a specific column to analyze would be easier and more efficient in SQL.

In terms of structure, SQL provides results that are more easily readable and that can be adjusted more quickly than when using Linux.

Joining tables

Some security-related decisions require information from different tables. SQL allows the analyst to join multiple tables together when returning data. Linux doesn't have that same functionality; it doesn't allow data to be connected to other information on your computer. This is more restrictive for an analyst going through security logs.

Best uses

As a security analyst, it's important to understand when you can use which tool. Although SQL has a more organized structure and allows you to join tables, this doesn't mean that there aren't situations that would require you to filter data in Linux.

A lot of data used in cybersecurity will be stored in a database format that works with SQL. However, other logs might be in a format that is not compatible with SQL. For instance, if the data is stored in a text file, you cannot search through it with SQL. In those cases, it is useful to know how to filter in Linux.

The WHERE clause and basic operators

Previously, you focused on how to refine your SQL queries by using the **WHERE** clause to filter results. In this reading, you'll further explore how to use the **WHERE** clause, the **LIKE** operator and the percentage sign (**%**) wildcard. You'll also be introduced to the underscore (**_**), another wildcard that can help you filter queries.

How filtering helps

As a security analyst, you'll often be responsible for working with very large and complicated security logs. To find the information you need, you'll often need to use SQL to filter the logs.

In a cybersecurity context, you might use filters to find the login attempts of a specific user or all login attempts made at the time of a security issue. As another example, you might filter to find the devices that are running a specific version of an application.

Activity - 2 (SQL)

Continuous learning in SQL

You've explored a lot about SQL, including applying filters to SQL queries and joining multiple tables together in a query. There's still more that you can do with SQL. This reading will explore an example of something new you can add to your SQL toolbox: aggregate functions. You'll then focus on how you can continue learning about this and other SQL topics on your own.

Aggregate functions

In SQL, **aggregate functions** are functions that perform a calculation over multiple data points and return the result of the calculation. The actual data is not returned.

There are various aggregate functions that perform different calculations:

- **COUNT** returns a single number that represents the number of rows returned from your query.
- **AVG** returns a single number that represents the average of the numerical data in a column.
- **SUM** returns a single number that represents the sum of the numerical data in a column.

Continuing to learn SQL

SQL is a widely used querying language, with many more keywords and applications. You can continue to learn more about aggregate functions and other aspects of using SQL on your own.

Most importantly, approach new tasks with curiosity and a willingness to find new ways to apply SQL to your work as a security analyst. Identify the data results that you need and try to use SQL to obtain these results.

Fortunately, SQL is one of the most important tools for working with databases and analyzing data, so you'll find a lot of support in trying to learn SQL online. First, try searching for the concepts you've already learned and practiced to find resources that have accurate easy-to-follow explanations. When you identify these resources, you can use them to extend your knowledge.

Continuing your practical experience with SQL is also important. You can also search for new databases that allow you to perform SQL queries using what you've learned.

Note -

Aggregate functions like **COUNT**, **SUM**, and **AVG** allow you to work with SQL in new ways. There are many other additional aspects of SQL that could be useful to you as an analyst. By continuing to explore SQL on your own, you can expand the ways you can apply SQL in a cybersecurity context.