**LINUX FORENSICS**

The Linux file system is a hierarchical structure that organizes and stores files on storage devices like hard drives, SSDs, and other media. The system follows a tree-like structure with the root directory (denoted by /) at the top, and everything else being a part of this structure, either as files or directories.

Here are the key components and concepts of the Linux file system:

**1. Root Directory (/):**

* The top-most directory in the file system hierarchy.
* All files and directories stem from this directory.

**2. Common Directories in Linux File System:**

* **/bin**: Contains essential binary executables (commands) required for basic operations (e.g., ls, cp, mv).
* **/sbin**: Contains system binaries (commands used for system maintenance, typically by the root user).
* **/etc**: Stores system configuration files. This is where most of the system-wide configuration files are located (e.g., /etc/passwd, /etc/network/interfaces).
* **/dev**: Contains device files that represent hardware devices or virtual devices (e.g., /dev/sda for the first hard disk).
* **/home**: User home directories. Each user has a subdirectory here (e.g., /home/user1).
* **/lib**: Contains shared libraries that are essential for programs in /bin and /sbin to run.
* **/opt**: Used for optional or third-party software packages.
* **/var**: Stores variable data files like logs (/var/log), spool files, and other files that change frequently.
* **/tmp**: Temporary files, typically erased on system reboot.
* **/mnt**: Temporary mount point for mounting file systems or storage devices.
* **/media**: Mount point for removable media (e.g., USB drives, CDs).
* **/usr**: Contains user-related programs, libraries, and documentation (includes /usr/bin, /usr/lib, and /usr/share).
* **/proc**: A virtual file system providing process and kernel information as files.
* **/sys**: A virtual file system providing information and interaction with the kernel and devices.
* **/root**: The root user's home directory.

**3. File Types:**

* **Regular files**: Contain data or program code (e.g., text files, images).
* **Directories**: Special files that contain a list of filenames and pointers to inodes.
* **Symbolic links**: A type of file that points to another file or directory.
* **Block devices**: Represent devices that provide buffered access (e.g., hard drives).
* **Character devices**: Represent devices that provide unbuffered access (e.g., keyboards, serial ports).
* **Sockets**: Used for inter-process communication (IPC).
* **Pipes**: Used for communication between processes.

**4. Inodes:**

* An inode is a data structure that stores metadata about a file (such as its size, ownership, permissions, and location on the disk). Every file and directory has an inode.
* The inode does not store the filename, but a pointer to the actual data blocks where the file's contents are stored.

**5. File Permissions:**

* Every file and directory has permissions for the owner, group, and others. These permissions determine who can read, write, or execute a file.
* Permissions are represented as:
  + **r** (read)
  + **w** (write)
  + **x** (execute)

File permissions are typically displayed with ls -l in the format:

**6. Mounting:**

* In Linux, file systems are mounted into the file system tree at a specific directory (mount point). For example, external drives might be mounted in /mnt or /media.
* You can mount or unmount file systems with the mount and umount commands.

**7. File System Types:**

* Linux supports various file systems, including:
  + **Ext4**: The most common file system in modern Linux distributions.
  + **XFS**: High-performance file system for large-scale systems.
  + **Btrfs**: A copy-on-write file system with advanced features like snapshots.
  + **F2FS**: Designed for NAND flash-based storage devices.
  + **NTFS**: Commonly used by Windows systems but can be read/written in Linux with additional tools.
  + **FAT32/exFAT**: Common for compatibility with Windows and embedded devices.

**8. Hard Links and Symbolic Links:**

* **Hard links**: Multiple directory entries that point to the same inode. Deleting one hard link doesn’t delete the file if others exist.
* **Symbolic (soft) links**: Files that point to another file or directory by name, similar to shortcuts in Windows. They can span file systems, unlike hard links.

**9. File System Hierarchy Standard (FHS):**

* The FHS defines the directory structure and directory contents for Unix-like systems, ensuring consistency across distributions.

**Key Commands for File Analysis:**

* **file <filename>: Identify file type.**
* **ls -l <filename>: View file permissions and metadata.**
* **stat <filename>: Get detailed metadata.**
* **cat, less, head, tail, strings: Inspect file content.**
* **find, locate, which: Locate files.**
* **sha256sum, md5sum: Check file integrity.**
* **du, df, blkid: Analyze disk usage and file system information.**

**LINUX BOOT PROCESS AND SERVICES**

The **Linux boot process** refers to the sequence of steps that the operating system follows to initialize and load into memory, starting from when the computer is powered on. This process involves a series of stages, from hardware initialization to starting user-level services. Here's a detailed breakdown of the Linux boot process and its associated services.

**1. BIOS/UEFI Stage (Boot Firmware)**

* **BIOS (Basic Input/Output System)** or **UEFI (Unified Extensible Firmware Interface)** is the first piece of software that runs when you power on a computer. Its primary job is to perform a power-on self-test (POST) to check the system’s hardware (CPU, memory, etc.).
* **UEFI** has largely replaced BIOS in modern systems because it provides more advanced features like faster booting, secure boot, and larger disk support.
* The BIOS/UEFI looks for a bootable device, typically a hard disk, SSD, CD/DVD drive, or USB drive. The boot order is set in the BIOS/UEFI settings.

**2. Bootloader Stage**

* Once BIOS/UEFI finishes POST, it hands control over to a **bootloader**, typically **GRUB** (GRand Unified Bootloader) in modern Linux distributions, though other bootloaders like LILO (Linux Loader) or Syslinux can also be used.

**GRUB**:

* + GRUB is responsible for loading the Linux kernel into memory and transferring control to it.
  + It presents a menu (if multiple operating systems are available) and allows you to choose which OS to boot. It also provides the option to pass kernel parameters (like boot options).

**Steps**:

* + GRUB loads its configuration file, usually located at /boot/grub/grub.cfg or /etc/grub.d.
  + GRUB loads the Linux kernel (usually located in /boot) and optionally an initial RAM disk (initrd or initramfs).
  + Once the kernel is loaded into memory, GRUB passes control to it.

**3. Linux Kernel Initialization**

* The Linux **kernel** is the core part of the operating system that manages hardware resources and system services. When the kernel is loaded into memory, it performs several tasks:
  + **Kernel Uncompression**: If the kernel is compressed, it gets decompressed into memory.
  + **Hardware Initialization**: The kernel detects and initializes the hardware components (CPU, memory, storage, etc.). This is where device drivers are loaded, either built into the kernel or as modules.
  + **Mounting the Root File System**: The kernel mounts the root filesystem, typically located on the hard disk or SSD. If an initrd (initial RAM disk) is used, it is also loaded here to help initialize the system's basic environment.

The root file system is crucial because it contains the system’s configuration files, binaries, libraries, and other critical components that the system needs to run.

**4. init Process**

* After the kernel is initialized, it looks for the **init** process, which is the first user-space program to run. The init process has PID (Process ID) 1 and is responsible for starting all other user-space processes.
  + In modern Linux distributions, the init process is often managed by **systemd**, a system and service manager. In older systems, it was typically managed by **SysVinit** or **Upstart**.
  + **systemd** is the most commonly used init system today and performs the following:
    - It reads the configuration files, typically found in /etc/systemd/, and determines which services (or "units") need to be started.
    - It runs the services and sets up the system environment according to the configuration.
    - **systemd** also handles process management, system logging, and dependency resolution for services.

**5. Mounting File Systems and Services**

* At this point, the root file system is mounted, but other file systems (e.g., /home, /var, /tmp) may be mounted as needed. If necessary, **network file systems** like NFS (Network File System) are also mounted.
* **systemd** begins the process of starting essential services. These services are defined as **units** in systemd, such as:
  + **network.target**: Networking services are started.
  + **multi-user.target**: The system enters a multi-user mode (for multi-user operation, with networking enabled).
  + **graphical.target**: If running a graphical environment, the system enters graphical mode, where services like the **X server** or **Wayland** are started.

**6. Starting User-Level Services**

* Once the kernel and basic system services are loaded, **user-level services** are started. These services are often **daemons**, which are background processes that handle system functions like networking, printing, and more.
* **systemd** manages the initialization of these services through **targets**, which are analogous to runlevels in traditional SysVinit. Common systemd targets include:
  + **default.target**: The default target that specifies the default system state (e.g., graphical or multi-user).
  + **graphical.target**: Starts graphical user interfaces (GUI) and related services.
  + **multi-user.target**: Starts multi-user services but does not start a graphical environment (equivalent to runlevel 3 in SysVinit).
  + **reboot.target**: Used when the system is being rebooted.

**7. User Login**

* Finally, once all essential services and user-level applications are started, the system reaches the point where a **login prompt** appears. Depending on the system setup:
  + If it’s a **graphical environment**, a display manager like **GDM**, **LightDM**, or **SDDM** starts the graphical login screen.
  + If it's a **console-based system**, a text-based login prompt is presented.
* After logging in, the user can start interacting with the system, and any additional services or applications will run based on the user’s configuration.

**Linux Boot Process Overview:**

1. **BIOS/UEFI**: Initializes hardware and locates the boot device.
2. **Bootloader (e.g., GRUB)**: Loads the Linux kernel and initial RAM disk (if used).
3. **Kernel Initialization**: The kernel initializes hardware and mounts the root file system.
4. **init Process**: The init process (typically systemd) takes over and begins the system startup.
5. **Mounting File Systems**: Other file systems (such as /home, /var) are mounted.
6. **Starting Services**: systemd starts background services based on system configuration.
7. **User Login**: The system displays a login prompt for the user.

**Common Services in the Linux Boot Process (via systemd):**

* **Network Services**:
  + networkd.service: Handles network configuration and networking.
  + NetworkManager.service: Manages network connections, especially on desktops.
* **Logging Services**:
  + systemd-journald.service: Collects and manages system logs.
  + rsyslog.service: A traditional logging service that works alongside systemd.
* **Time Services**:
  + chrony.service or ntpd.service: Synchronizes system time with external time servers.
* **Display Manager** (for graphical environments):
  + gdm.service, lightdm.service, sddm.service: Starts the graphical login manager.
* **Desktop Environment**:
  + gdm.service starts GNOME, or lightdm.service starts an environment like KDE.
* **File System Services**:
  + local-fs.target: Ensures all local file systems are mounted.
  + remote-fs.target: Manages the mounting of network file systems.
* **System Management**:
  + cron.service: Runs scheduled jobs.
  + ssh.service: Starts the SSH server for remote access.

**Key Linux System Organization and Artifacts**

1. **Kernel**: Core of the system, managing resources, device drivers, and system calls. It serves as the bridge between hardware and software and manages system resources such as CPU, memory, I/O devices, and file systems. It is responsible for process management, device drivers, file systems, networking, and security.
2. **System Libraries**: Shared libraries used by system and user programs.
3. **File System Hierarchy**:
   * /: Root directory, the starting point of the filesystem.
   * /bin/, /sbin/, /lib/: Essential system binaries and libraries.
   * /etc/: System configuration files.
   * /var/, /tmp/: Variable data and temporary files.
   * /home/: User home directories.
4. **System Services and Daemons**: Background processes that handle various system tasks.
5. **User Management**: Files like /etc/passwd and /etc/shadow to manage user information.
6. **Log Files**: Logs for system events and troubleshooting.
7. **Package Management**: Tools to manage software packages on the system.
8. **Security and Permissions**: Tools and files to manage file permissions and security, including sudoers.

A **user account** in Linux typically consists of the following:

* **Username**: A unique identifier for the user.
* **User ID (UID)**: A unique numeric identifier assigned to the user.
* **Password**: A mechanism to authenticate the user.
* **Home Directory**: A personal directory for the user to store files and settings.
* **Shell**: The command-line interpreter the user will interact with (e.g., Bash, Zsh).

**Types of User Accounts in Linux**

* **Root User**:
  + The **root** user is the superuser and has unrestricted access to all system resources and commands. The UID for the root user is always 0.
  + The root user can modify any file, install or remove software, and manage system settings.
* **Regular Users**:
  + Regular users are given limited permissions, ensuring that they can only access their own files and certain system resources. Regular users typically do not have administrative privileges unless explicitly granted (e.g., through sudo).
* **System Users**:
  + These are special users created to run system processes and services. These users typically do not have login privileges and are used by daemons (background services).
  + Common system users include nobody, daemon, and users for specific services like apache or mysql.

**User Permissions and Access Control**

In Linux, users are assigned permissions that control access to files and directories. These permissions include:

* **Read (r)**: Allows the user to view the contents of a file or list the contents of a directory.
* **Write (w)**: Allows the user to modify the contents of a file or add/remove files from a directory.
* **Execute (x)**: Allows the user to execute a file (if it's a program or script) or traverse a directory.

**LINUX FORENSICS TOOLS**

**1. Sleuth Kit (TSK) and Autopsy**

* **Sleuth Kit is an open-source suite of command-line tools used for digital forensics. It is used for investigating disk images and performing file system** analysis**. Autopsy is a graphical interface built on top of the Sleuth Kit that makes it easier to navigate and conduct investigations.**
* **Use Cases:**
  + **Analyzing disk images (e.g., .dd or .E01).**
  + **Recovering deleted files.**
  + **Inspecting file metadata and timestamps.**
* **Common Tools in Sleuth Kit:**
  + **fls: List files from an image or partition.**
  + **icat: Extract files from an image or partition.**
  + **mmls: Display the partition table of a disk image.**
* **Autopsy provides an easy-to-use interface for these tasks.**

**2. Volatility**

* **Volatility** is an open-source memory forensics framework for analyzing RAM dumps. It's used to analyze volatile data (in-memory data), such as running processes, network connections, and open files.
* **Use Cases**:
  + Analyzing memory dumps to detect malware, rootkits, and unauthorized processes.
  + Investigating network activity and processes running on a system.
  + Extracting user credentials, keys, and other sensitive information from memory.
* **Common Features**:
  + Process listing (pslist).
  + Network connections (netscan).
  + Dumping passwords or encrypted data from memory.

**3. Chkrootkit**

* **Chkrootkit** is a tool for detecting rootkits on Linux systems. It scans the system for signs of rootkits by looking for anomalies in various system files, processes, and kernel modules.
* **Use Cases**:
  + Rootkit detection in compromised Linux systems.
  + Scanning for hidden files, processes, and network activity.
* **Common Features**:
  + Checks for common rootkits like LKM, R1, and Suckit.
  + Scans system binaries and file systems for unusual activity.

**4. Lynis**

* **Lynis** is a security auditing tool that also serves as a powerful forensics tool. It checks the configuration of a Linux system for vulnerabilities and security weaknesses.
* **Use Cases**:
  + Conducting security audits on Linux systems.
  + Detecting misconfigurations, outdated software, and insecure permissions.
* **Common Features**:
  + System hardening checks.
  + Security vulnerability assessment.
  + Detailed reports with recommendations for improvements.

**5. The Log Analysis Tool (Logcheck)**

* **Logcheck** is a tool that helps with log file analysis by automatically parsing and reporting unusual or potentially harmful entries.
* **Use Cases**:
  + Detecting unauthorized access or suspicious activity by analyzing system logs.
  + Monitoring logs for failed login attempts, system events, and unusual commands.
* **Common Features**:
  + Regular expression-based filtering of logs.
  + Email reports of suspicious activity.

**6. Network Forensics Tools**

* **Wireshark**: While Wireshark is primarily a network protocol analyzer, it’s widely used for network forensics in Linux as it captures and inspects network traffic.
  + **Use Cases**:
    - Monitoring network traffic for malicious activity.
    - Investigating data exfiltration, unauthorized access, or unusual communication patterns.

**tcpdump**: A command-line packet analyzer that can capture and display packets on the network.

* **Use Cases**:
  + Capturing network packets for forensic analysis.
  + Analyzing network activity after a breach or intrusion.

**7. Forensic Analysis of Log Files**

* **Logwatch**: A tool that summarizes log files and alerts administrators to potential issues.
  + **Use Cases**:
    - Collecting and analyzing logs to detect security events like unauthorized access, failed login attempts, or changes to system files.

**GoAccess**: A real-time log analyzer that provides insights into web server logs.

* **Use Cases**:
  + Investigating web server logs for unusual or malicious activity.

**8. tct (The Coroner’s Toolkit)**

* **TCT** is a suite of forensics tools for Linux systems that allows investigators to examine file systems, analyze logs, and recover files.
* **Use Cases**:
  + Recovering deleted files.
  + Analyzing disk images and file system structures.
  + Investigating system activity and file metadata.
* **Common Tools**:
  + fls, icat, and mactime (for extracting timestamps and analyzing files).

**9. GParted (Partition Editor)**

* **GParted** is a graphical tool used for partitioning disks and recovering data.
* **Use Cases**:
  + Recovering lost or damaged partitions.
  + Modifying disk partitions in the event of a forensic investigation.

**10. Extundelete**

* **Extundelete** is a tool specifically designed for recovering deleted files from **ext3** and **ext4** file systems.
* **Use Cases**:
  + Recovering accidentally or maliciously deleted files from ext3/ext4 file systems.

**11. Rifiuti (Forensic Data Recovery)**

* **Rifiuti** is a tool designed to recover deleted files from the **ext3/ext4** file system.
* **Use Cases**:
  + Recovering deleted files or directories.
  + Forensic data recovery.

**12. Hashdeep**

* **Hashdeep** is a tool used for calculating hash values (MD5, SHA1, SHA256, etc.) of files and directories, and it can verify the integrity of files during forensic investigations.
* **Use Cases**:
  + Verifying the integrity of files during investigations.
  + Performing file hash comparisons to check for alterations.

**13. Mimikatz (on Linux with Wine or Mono)**

* **Mimikatz** is a popular tool used for extracting plaintext passwords, hashes, PINs, and Kerberos tickets from memory. While originally a Windows tool, it can be used on Linux under **Wine** or **Mono**.
* **Use Cases**:
  + Extracting credentials from compromised systems.