Infrastructure Security Using Linux

Computer science / Cybersecurity

System Management and Access Control

Booting

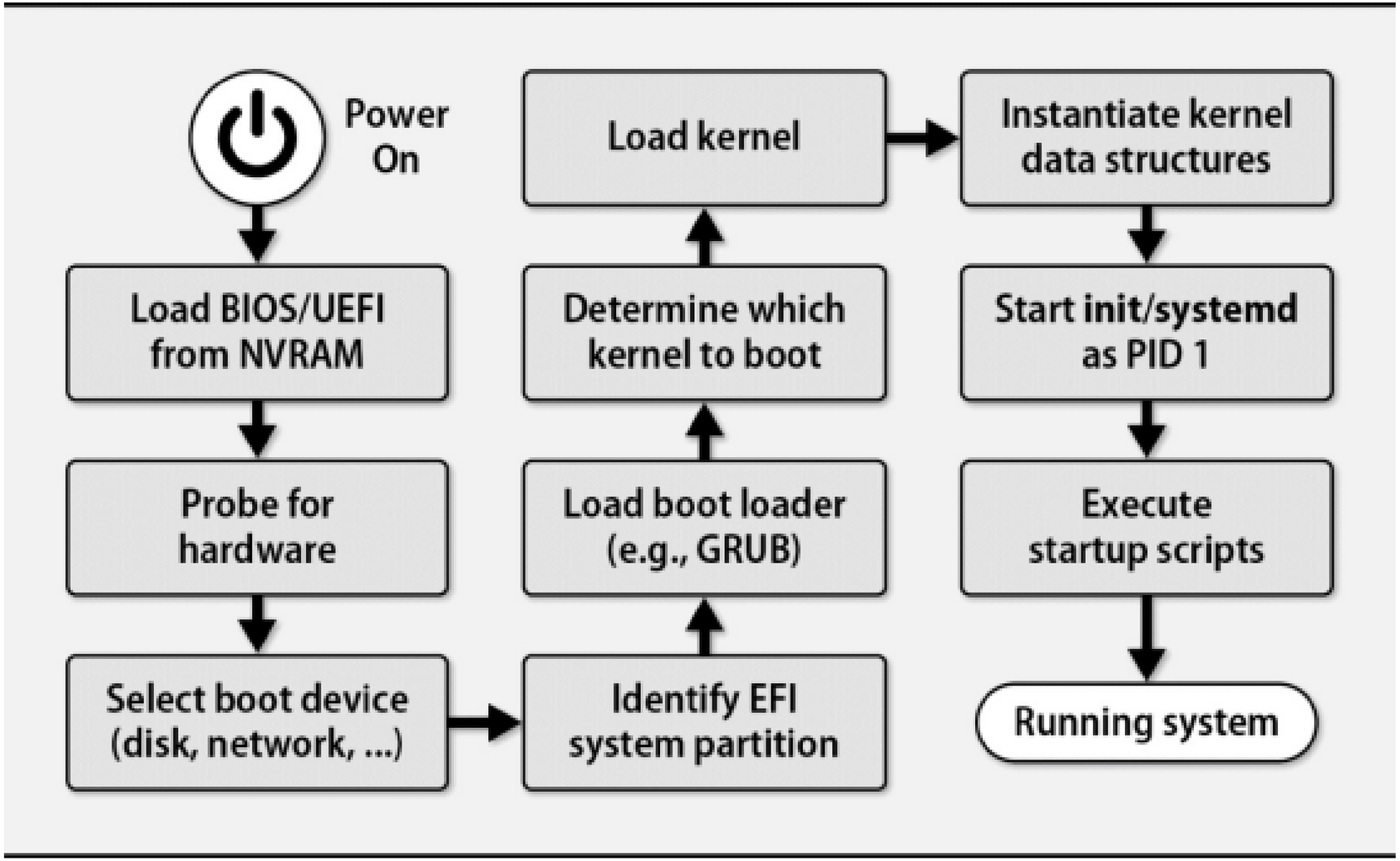
- **Booting**
  - Booting = Starting up a computer (from "bootstrapping" → "pulling itself up by its own bootstraps").
- **Boot Process Steps:**
  - **Find, load, and run** bootstrapping code.
  - **Find, load, and run** the OS kernel.
  - **Run startup scripts** and system daemons.
  - **Manage system state** and ensure process hygiene and managing system state transitions.

Boot process overview

- **Modern Linux Boot Process**
- Most Linux distros use **systemd** (replaces UNIX init).
- **systemd** improvements:
  - Dependency management
  - Concurrent startup processes
  - Comprehensive logging
- **Bootstrapping:**
  - Kernel loaded into memory and executed.

Boot process overview

Exhibit A: Linux & UNIX boot process



Boot process overview

- **Pre-Boot Procedures**
  - **Filesystem checks** and **mounting** are required before full boot.
  - **System daemons** are started.
  - Managed by:
    - shell scripts (sometimes called **init scripts** for traditional init)
    - **Unit files** (for systemd)
  - **Init** runs scripts sequentially; **systemd** parses unit files.

BIOS vs UEFI

- The Basic Input/Output System (**BIOS**) and Unified Extensible Firmware Interface (**UEFI**) are two different firmware interfaces that are used to start up a computer.

- **BIOS :**
  - Uses MBR partitioning scheme.
  - 16-bit processor mode.
  - Limited to text-based interface.
  - Simple and less extensible.
  - Lacks advanced security features.

- **UEFI :**
  - Supports both MBR and GPT partitioning schemes.
  - 32-bit and 64-bit processor modes.
  - Graphical interface with mouse support.
  - Modular and extensible design.
  - Includes advanced security features like Secure Boot.

UEFI

- **UEFI** = Modern, feature-rich firmware (better compatibility, security, and extensibility), **compared** to the older BIOS.
- **efibootmgr** command:
  - Modifies UEFI boot manager and it’s configrations.
  - Adjusts boot order on a running system.

UEFI

```
$ efibootmgr -v

BootCurrent: 0004
BootOrder: 0004,0000,0001,0002,0003
Boot0000* Windows Boot Manager        HD(1,GPT,3e0e3e3e-3e3e-3e3e-3e3e-
3e3e3e3e3e3e,0x800,0x82000)/File(\EFI\MICROSOFT\BOOT\BOOTMGFW.EFI)WINDOWS.....x...B.C.D.O.B.J.E.C.T.=.{
.9.d.e.a.8.6.2.c.-.5.c.d.d.-.4.e.7.0.-.a.c.c.1.-.f.3.2.b.3.4.4.d.
Boot0001* UEFI: Built-in EFI Shell          VenMedia(5023b95c-db26-429b-a648-bd47664c8012)..BO
Boot0002* UEFI: SanDisk Cruzer Glide 1.26, Partition 1
                PciRoot(0x0)/Pci(0x14,0x0)/USB(1,0)/HD(1,MBR,0x0,0x800,0x2000)..BO
Boot0003* UEFI: SanDisk Cruzer Glide 1.26, Partition 2
                PciRoot(0x0)/Pci(0x14,0x0)/USB(1,0)/HD(2,MBR,0x0,0x800,0x2000)..BO
Boot0004* ubuntu   HD(1,GPT,3e0e3e3e-3e3e-3e3e-3e3e-
3e3e3e3e3e3e,0x800,0x82000)/File(\EFI\UBUNTU\SHIMX64.EFI)

$ efibootmgr -o 0000,0004 # Change the boot order to boot Windows first
```

- On systems (typically those with **systemd**) that allow write access by default, **rm -rf /** can be enough to permanently destroy the system at the firmware level; in addition to removing files, **rm** also removes variables and other UEFI information accessible through **/sys**.

GRUB

- **GRUB (Grand Unified Bootloader)**
- Popular boot loader for most Linux distros.
- Features:
  - Boots multiple OS.
  - Loads Linux kernel modules.
  - Provides CLI for troubleshooting.

Table of some grub commands:

Command	Description
boot	Boots the system from the specified kernel image.
help	Displays a list of available commands.
linux	Loads the specified Linux kernel.
reboot	Reboots the system.
search	Searches devices by file, filesystem label, or UUID.
usb	Tests the USB driver.

System management daemons

- **AKA “Spontaneous” Processes**
  - Created by the kernel **after initialization**.
  - Kernel starts them **autonomously** (unlike normal processes, which are created by existing processes).

System management daemons

- **Characteristics of Spontaneous Processes**
  - Part of kernel implementation (not linked to filesystem programs).
  - **Non-configurable** and **require no admin attention**.
  - Identified by:
    - **Low PIDs** in **ps** listings.
    - **Square brackets** (e.g., [kthreadd]).

System management daemons

- **Exception:** init (PID 1) – manages system processes.
- **systemd** (modern replacement for init):
  - More powerful and flexible.
  - Standard in most Linux distros.

System management daemons

- **System Modes (Defined by init)**
  - **Single-user mode:**
    - Minimal state (only root filesystem and essential daemons).
    - Used for **maintenance and recovery**.
  - **Multi-user mode:**
    - Fully operational (all filesystems and daemons).
    - **Normal operation**.
  - **Server mode:**
    - Like multi-user mode but **without a GUI**.

systemd **Structure**

- **Not a single daemon** – collection of programs, daemons, libraries, and kernel components.
- **Units and Unit Files:**
  - **Unit** = Managed entity (e.g., service, socket, device, mount point, swap, timer).
  - **Unit file** defines unit behavior:
    - Location of executable.
    - Start/stop instructions.
    - Dependencies on other units.

Example of **rsync** service unit file

```
[Unit]
Description=fast remote file copy program daemon
ConditionPathExists=/etc/rsyncd.conf


[Service]
ExecStart=/usr/bin/rsync --daemon --no-detach


[Install]
WantedBy=multi-user.target
```

systemd

Unit File Locations

- **Stock unit files:**
  - /usr/lib/systemd/system or /lib/systemd/system (installed by packages).
  - **Do not modify** these files.
- **Custom unit files:**
  - /etc/systemd/system (for local customizations).
- **Transient units:**
  - /run/systemd/system (temporary).
- **Unit Types:**
- **Service units:** .service
- **Socket units:** .socket
- Others follow similar naming patterns.

Systemctl **(systemd management tool)**

- **Primary functions:**
  - Start, stop, enable, disable, check status of units.
  - List units and dependencies.
  - Show unit logs.
- **Command Structure:**
  - First argument = **subcommand** (sets general task).
  - Subsequent arguments = **specific to subcommand**.
  - Bundled for consistency and clarity (like Git).

Examples

```
# show all loaded and active services, sockets, targets, mounts, and devices

$ systemctl list-units


# show all loaded and active services

$ systemctl list-units --type=service


# show all installed unit files

$ systemctl list-unit-files --type=socket
```

systemd

Subcommand	Function
list-unit-files [*pattern*]	Shows installed units; optionally matching *pattern*
enable *unit*	Enables *unit* to activate at boot
disable *unit*	Prevents *unit* from activating at boot
isolate *target*	Changes operating mode to *target*
start *unit*	Activates *unit* immediately
stop *unit*	Deactivates *unit* immediately
restart *unit*	Restarts (or starts, if not running) *unit* immediately
status *unit*	Shows *unit*'s status and recent log entries
kill *pattern*	Sends a signal to units matching *pattern*
reboot	Reboots the computer
daemon-reload	Reloads unit files and **systemd** configuration

systemd

- The unit file statuses are:
  - **bad** : The unit file is bad or broken.
  - **disabled** : The unit file is installed but not configured to start autonomously.
  - **enabled** : The unit file is installed and configured to start autonomously.
  - **masked** : Banished from the **systemd** world from a logical perspective.
  - **static** : Depended upon by another unit; has no install requirements.
  - **indirect** : Disabled, but has peers in Also clauses that may be enabled.
  - **linked** : A symbolic link to another unit file.
- Unit files can declare their relationships to other units in a variety of ways. The most common is to specify a **WantedBy** or **RequiredBy** directive in the **[Install]** section of the unit file. These directives specify the target units that the unit should be started or stopped with.

Rebooting and shutting down

- **Shutdown Commands**
  - **halt** – Shuts down system:
    - Logs shutdown.
    - Kills nonessential processes.
    - Flushes filesystem caches.
    - **halt -p** → Powers off system.
- **reboot** – Same as **halt**, but **restarts** the system.
- **shutdown** – Wrapper over halt and reboot:
  - Allows **scheduled shutdowns**.
  - Sends warnings to logged-in users (legacy feature).
  - **Largely obsolete** for modern systems.

Access Control and Rootly Powers

Access Control and Rootly Powers

Standard UNIX Access Control

- **UNIX Access Control Model**
  - **User-based access control** – Access depends on user identity or group membership.
  - **Objects (e.g., files, processes):**
    - Have owners with broad control (not necessarily unrestricted).
    - You own the objects you create.
  - **Root user:**
    - Can act as the owner of any object.
    - Only root can perform sensitive admin tasks.

Filesystem access control

File Ownership and Groups

- Every file has an **owner** and a **group** ("group owner").
- Tracked as **numbers** (not names):
  - **UID** → Mapped to usernames in /etc/passwd.
  - **GID** → Mapped to group names in /etc/group.

The **root** account

- **Root (it might be considered as Superuser!!!) Account**
- Most powerful account → **Bypasses all access control checks**.
- UID of root = **0** (hard-coded in the kernel, not configurable).
- Any process with **effective UID = 0** = Superuser.
- Superuser can perform **any valid operation**:
  - "Valid" → Must comply with basic file permissions (e.g., cannot execute files without execute bit).

The **root** account

- Some examples of restricted operations include:
  - Creating device files in /dev
  - Setting the system clock
  - Raising resource usage limits and process priorities
  - Setting the system's hostname
  - Configuring network interfaces
  - Shutting down the system

The **root** account

Superuser vs Root

- **Superuser** = Any process with **effective UID = 0** (not necessarily the root account).
- **Setuid root program** → Runs with effective UID of root but isn't the root account.

Example:

- **login** program runs as root → Prompts for username and password.
- If credentials are valid:
  - Process **changes UID and GID** to user's values.
  - Once downgraded to a normal user, it **cannot regain root privileges**.

The **setuid** and **setgid** execution

setuid and setgid Bits

- **setuid** → Executable runs with the **effective UID** of the file's owner.
- **setgid** → Executable runs with the **effective GID** of the file's group owner.

Security Risks:

- **setuid root programs** = High risk (can grant root access if exploited).
- Mitigation:
  - Ignore setuid on file systems mounted with the **noexec** option.

Management of the root account

su (Substitute User Identity)

- **Switch to another user** (including root) by entering the target user's password.
- **Without username** → Defaults to root.

Security and Usage Notes:

- **Commands run via su are not logged** (hard to trace actions).
- **Log entry** records who became root and when.
- Use **full path** (/bin/su) to avoid trojans.
- Requires membership in **wheel group** on most systems.
- **sudo** → Preferred for security; use su for emergencies only.

Management of the root account

sudo (Limited su)

- Allows running commands as another user (including root).
- **Logs all commands** run with sudo (tracks who did what).

How sudo Works:

- Command is checked against **/etc/sudoers**:
  - Lists allowed users and commands.
- If allowed:
  - Prompts for user's password → Executes command.

Configuration:

- **Edit sudoers** with visudo (checks syntax before saving).
- visudo uses $EDITOR or defaults to **vi**.

sudo without password

- If you want to allow a user to run **sudo** without entering a password, you can use the **NOPASSWD** tag. For example, to allow the user **bob** to run **sudo** without entering a password, you can use the following line in the **sudoers** file:

```
bob ALL = NOPASSWD: ALL  # Don't try this at home!!
```

- The most common cases are when performing remote configuration management with tools like **ansible** or **puppet**.
- A better alternative to **NOPASSWD** in the context of remote configuration management is to use **ssh** keys to authenticate to the remote system. This way, the user doesn't need to enter a password to run **sudo**.

- SSH key forwading can be used to authenticate to the remote system using **ssh** keys and then run **sudo** without entering a password.
- The sudo program is maintained by Todd C. Miller and is available from www.sudo.ws.
- **Disabling the root account**
  - Some administrators disable the root account entirely. This is done by setting the root account's password to an **impossible value**, such as a **long string of random characters**. This makes it impossible to log in as root, even if you know the root password.

PAM: Pluggable Authentication Modules

- **PAM (Pluggable Authentication Modules)**
  - Configurable authentication framework for UNIX systems.
  - Used by **login**, **su**, **sudo**, and other programs.
- **Examples:**
  - Require **2FA** for specific users.
  - Combine **password** and **fingerprint** for authentication.

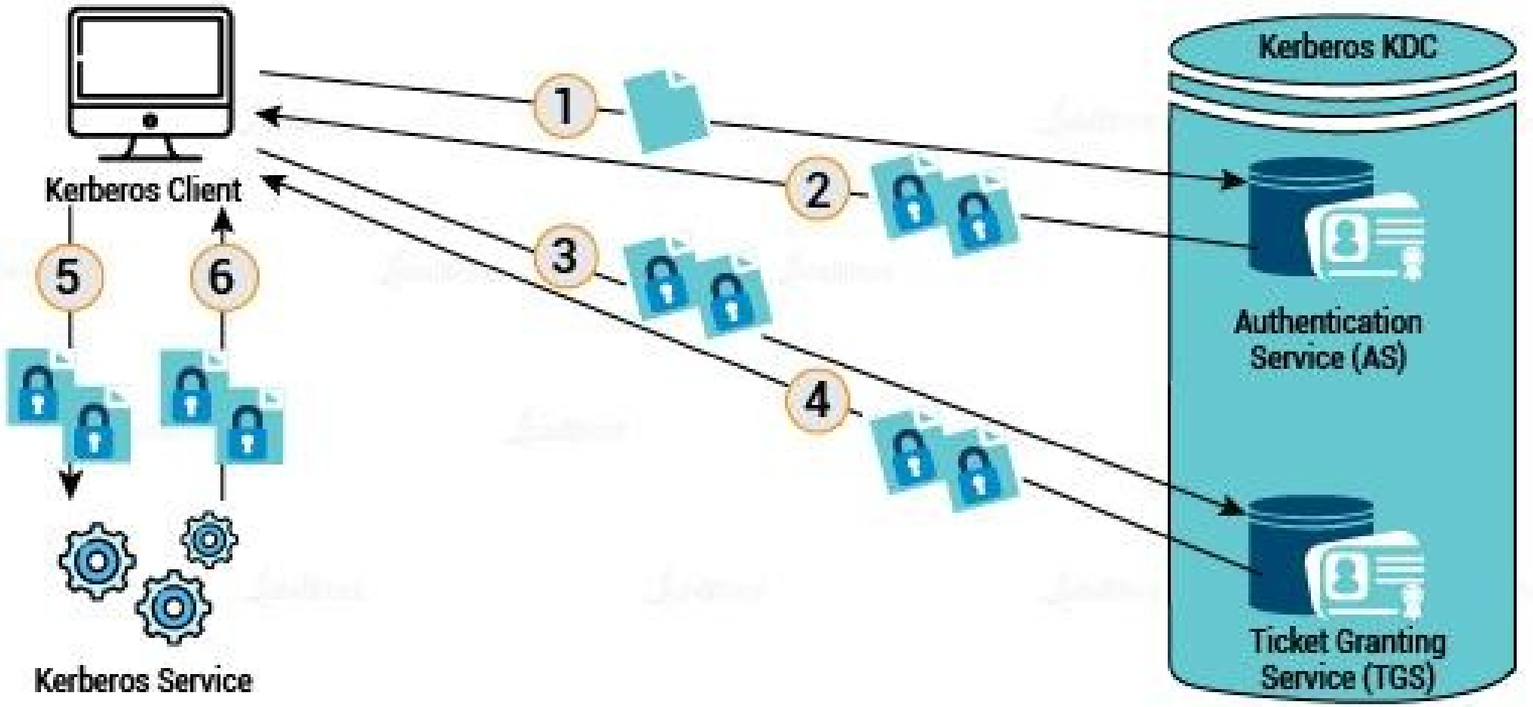
Kerberos: network cryptographic authentication

- **Kerberos**
  - **Network authentication system –** No password transmission over the network.
  - Common in **large organizations** (e.g., academia, research).
- **PAM vs Kerberos:**
  - **PAM** = Authentication framework.
  - **Kerberos** = Specific authentication method.
  - PAM can be configured to **use Kerberos** for authentication.

Kerberos: network cryptographic authentication

- **Kerberos Authentication Process**
  - Uses a **trusted third party** (Key Distribution Center – KDC) for network-wide authentication.
  - Authenticate to **Kerberos service** → Receive cryptographic credentials for access to other services.
- **Key Distribution Center (KDC):**
  - **Authentication Server (AS):** Authenticates users and issues tickets for the TGS.
  - **Ticket Granting Server (TGS):** Issues tickets for network services.

Kerberos: network cryptographic authentication



Capabilities

- **Capabilities**
  - Provide **fine-grained control** over process privileges (vs full superuser powers).
  - Allow granting specific powers **without full root access**.
- **Example:**
  - Grant a process the capability to **bind to a low-numbered port** (<1024) **without file access**.
  - Useful for network services needing privileged ports.

Linux namespaces

- **Linux Namespaces**
  - Create **isolated environments** within a Linux system.
  - Each namespace has its own:
    - **Process tree**
    - **Network interfaces**
    - **Filesystem**
- **Example:**
  - A container with its own **network namespace** gets separate network interfaces and IP addresses.
  - Allows multiple containers to run independently on the same system.

Modern access control

- **SELinux (Security-Enhanced Linux)**
  - Developed by **NSA** (2001) – Implements **Mandatory Access Control (MAC)**.
  - More **fine-grained** than traditional UNIX access control.
- **Linux Security Modules (LSM):**
  - Kernel-level interface for integrating access control systems.
  - Allows loading **custom security modules** into the kernel (including SELinux).

Mandatory Access Control (MAC)

- **MAC:**
  - Central authority controls access decisions.
  - Example: A process can read but not write to a file, as defined by the authority.

Mandatory Access Control (MAC) vs Discretionary Access Control (DAC)

- **DAC:**
  - File owner controls access permissions.
  - Example: Owner decides who can read and write to a file.

Role-Based Access Control (RBAC)

- **Role-Based Access Control (RBAC)**
  - Access based on **roles** rather than individual users.
  - More **fine-grained** than UNIX groups.
- **Example:**
  - **Administrator role** → Perform admin tasks.
  - **User role** → Perform normal tasks.
  - A user can hold **multiple roles** simultaneously.

SELinux: Security-Enhanced Linux

- **SELinux (Security-Enhanced Linux)**
  - One of the oldest **Linux MAC implementations**.
  - Developed by **NSA**; now maintained by the **open source community**.
- **Supports:**
  - **Role-Based Access Control (RBAC)**
  - **Type Enforcement (TE)**
  - **Multi-Level Security (MLS)**
- **Complexity:**
  - Requires **careful configuration** for proper operation.

AppArmor

- **AppArmor** is a MAC system that is similar to SELinux. It was developed by Novell, and is now maintained by the open source community. AppArmor is designed to be easier to use than SELinux, and is often used on desktop systems.

- WSO2 Identity Server provides the **#1 open source**, market-leading full lifecycle platform for building, integrating, securing, of devices and data, visualization of sensor data and event management.