ERNW
providing security.

ERNW WHITE PAPER 76
LINUX HARDENING GUIDE

Version:	1.0
Date:	May 19, 2026
Classification:	Public

Julian Suleder

Digital unterschrieben von Julian Suleder
Datum: 2026.05.19 12:39:10 +02'00'

ERNW
providing security.

Table of Content

1 Handling

The present document is classified as *Public*. Any distribution or disclosure of this document REQUIRES the permission of the document owner as referred in Section *Document Status and Owner*.

1.1 Document Status and Owner

As the owner of this report, the document owner has exclusive authority to decide on the dissemination of this document and responsibility for the distribution of the applicable version in each case to the places.

The possible entries for the status of the document are *Initial Draft*, *Draft*, *Effective* and *Obsolete*.

Report Information	
Title:	ERNW White Paper 76 - Linux Hardening Guide
Document Owner:	ERNW Enno Rey Netzwerke GmbH
Version:	1.0
Status:	Effective
Classification:	Public
Project Number:	-
Author(s):	Niklas Heringer
Project Lead:	Niklas Heringer, nheringer@ernw.de

Table 2: Document Status and Owner

1.2 Classification Levels

Classification Level	Audience
Public:	Everyone
Internal:	All employees and business partners
Confidential:	Only employees
Secret:	Only selected employees

Table 3: Classification Levels

1.3 Document Version History

Version	Date	Details
1.0	May 19, 2026	Initial version after quality assurance.

Table 4: Document Version History

2 Introduction

This document provides a base hardening guideline for Linux distributions in general to enhance their system security while remaining commonly usable.

2.1 Scope & Application

This hardening guide covers the recommendations for hardening a Linux computer.

Settings that might have a severe impact on the operating system's functionality and need a lot of further testing are not part of this guide or are marked as optional. Further, this guide does not claim to be complete. For example, the hardening of a system relies not only on the operating system but also on installed and used third-party software. Such software is not covered in this guide.

This guide has been developed and validated against Ubuntu 24.04 LTS as the primary reference platform, and cross-tested on Debian 12, Rocky Linux 9, Red Hat Enterprise Linux 9, openSUSE Leap 15.6, Fedora and Arch Linux using automated audit, fix and rollback cycles.

The security controls and checks are written against POSIX-compliant tooling and are broadly applicable to any modern Linux distribution.

Where package installation is required as a remediation step, Debian/Ubuntu commands are given as the primary example, with notes for RPM-based and SUSE-based systems where the syntax differs meaningfully.

Note that not all security tooling is available on every distribution. AIDE, for instance, is not available in the default repositories of RHEL 9 or Arch Linux, and checks depending on it will report as failed on those platforms *without manual installation*. Such platform-specific gaps are expected and documented where applicable.

2.2 How to Read and Use This Document

Each recommended setting in this hardening guide is *mandatory* unless marked as *optional*. The controls are described using the terms MUST or MUST NOT (mandatory) or SHOULD (optional) as defined in RFC 2119[1]. *Optional* hardening settings mean that it is recommended to apply this setting, but there may be required functionality on the system that will become unavailable once the setting is applied. Further notes are added to *Optional* settings if they significantly impact the user experience.

The code samples used in this document illustrate the implementation of the controls as well as checking for compliance with the controls. The commands are tested on the supported Linux systems and can potentially be used on other

[1] *Key words for use in RFCs to Indicate Requirement Levels:* https://datatracker.ietf.org/doc/html/rfc2119

systems; however they may differ in function and effect. Some commands may need sudo privileges for successful execution.

The following formatting/semantics are used for the presentation of commands/code:

- Expressions of style "> `command`" mean the execution of a command.
- A line without ">" after a command is equivalent to the output of the command.
- A line without the leading ">" contains a single command without output.
- A line with the leading "`#`" in a configuration file is a comment
- Code/Commands are formatted as follows:

```
> chmod 0700 /home/USER
```

- Notes are formatted as follows:

> **Note:** This is an example Note.

3 Authentication & Identity Management

This section describes essential authentication & identity management mechanisms on Linux.

3.1 Harden Administrative Accounts

In this section, we enforce two *mandatory* controls to secure high-privilege accounts: ensuring a single root account and restricting the use of the `su` command.

3.1.1 Verifying that only One Admin Account Is Present

There MUST NOT be any user other than `root` with a User ID (`UID`) of `0`, as this grants full, unrestricted administrative privileges. The `/etc/passwd` file MUST be reviewed and, if necessary, edited.

The following command can be used to verify this (i.e., that there is only one user with UID equal to `0`):

```
> awk -F: '($3 == "0") {print}' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Note: This command MUST return only the `root` user entry. If more than one line is returned, manual action is required to investigate and remedy the unauthorized administrative account(s).

3.1.2 Restrict `su` Command Access

Restricting the use of `su` forces administrators to use `sudo`, which provides better control and logging of privilege escalation.

The system MUST activate the `pam_wheel.so` module in `/etc/pam.d/su` and specify the group used for system administration. This prevents non-privileged users from attempting to switch to `root`.

To determine the correct group:

- Use `getent` to check for the `wheel` group (common on RHEL/CentOS/Fedora) or the `sudo` group (common on Debian/Ubuntu):

```
> getent group wheel
> getent group sudo
```

Then you MUST modify `/etc/pam.d/su` to include the following line, replacing `<DESIRED_GROUP>` with the correct group name:

```
[...]
```

```
auth       required   pam_wheel.so group=<DESIRED_GROUP>
[...]
```

Now only members of this designated group will be allowed to execute the `su` command.

3.2 Check that All Passwords Are Shadowed

3.2.1 Verify Password Shadowing

Enforcing password shadowing is a *mandatory* control. Password hashes MUST be stored in the restricted-access `/etc/shadow` file, and MUST NOT be visible in the world-readable `/etc/passwd` file. When shadowing is enabled, the password field (the second field) in `/etc/passwd` contains an 'x'.

The following command can be used to verify that all system passwords are correctly shadowed. This command searches for any entries in `/etc/passwd` where the password field does not contain an 'x' (or a '*' or '!' which also indicate a disabled/locked account, but not a clear text hash):

```
> awk -F: '($2 != "x" && $2 != "*" && $2 != "!") {print}' /etc/passwd
```

Note: For compliance, this command MUST return NO output. If any entries are returned, it indicates a security vulnerability where password hashes may be readable by unauthorized users.

3.2.1.1 Remediation

If accounts with non-shadowed passwords are found, the `pwconv`[2] utility is typically used to move the password information from etc/passwd to `/etc/shadow`.

3.3 Use Strong Hashing Algorithm

Ensuring that Linux Pluggable Authentication Modules (`PAM`) uses a strong hashing mechanism is a mandatory control. This measure protects user password hashes from offline brute-force and dictionary attacks. The system MUST be configured to use a modern, strong hashing algorithm. `YESCRYPT` is the recommended choice, as it is the default in modern Linux distributions and provides memory-hard hashing, making offline attacks significantly more expensive. `SHA512` is also acceptable where `YESCRYPT` is not supported. This is controlled by setting the value of the `ENCRYPT_METHOD` parameter in the file `/etc/login.defs`.

3.3.1 Verify Hashing Algorithm

The following command can be used to verify the configured hashing method for new users:

[2] *https://man7.org/linux/man-pages/man8/pwconv.8.html*

```
> grep ENCRYPT_METHOD /etc/login.defs
ENCRYPT_METHOD YESCRYPT
```

Note: If the value of this parameter is changed, users MUST change their password in order for the effect to take place. This is because this setting only applies when a password is created or changed, not to existing passwords.

3.4 Password Policy

Enforcing a strong password policy is a *mandatory* control to protect user accounts from brute-force attacks and dictionary attacks. This policy is primarily enforced through the Pluggable Authentication Module (PAM) configuration file `/etc/pam.d/common-password`.

3.4.1 Password Complexity (`pam_pwquality.so`)

The system MUST enforce strong complexity requirements using the `pam_pwquality.so` module. This ensures that new passwords meet a minimum standard for length and character diversity.

The following configuration lines demonstrate the required parameters:

Parameter	Policy Set	Description
`minlen=16`	Minimum length	The password MUST be at least `16` characters long.
`dcredit=-1`	Digit credit	The password MUST contain at least `1` digit.
`ucredit=-1`	Uppercase credit	The password MUST contain at least `1` uppercase letter.
`lcredit=-1`	Lowercase credit	The password MUST contain at least `1` lowercase letter.

3.4.2 Password History (`pam_pwhistory.so`)

Password reuse makes systems vulnerable. The `pam_pwhistory.so` module MUST be configured to prevent users from cycling through their old passwords.

- `remember=5`: Prevents reuse of the last `5` passwords.
- `enforce_for_root`: Applies the history check even to the `root` user, which is a *mandatory* hardening requirement.

3.4.3 Strong Hashing Algorithm (`pam_unix.so`)

The password hash stored in `/etc/shadow` MUST use a modern, slow, and computationally expensive algorithm to resist offline cracking. The `pam_unix.so` module is configured to use the `yescrypt` algorithm, which is the current state-of-the-art for many distributions.

The relevant line in `/etc/pam.d/common-password` MUST include the option:

```
password [success=2 default=ignore] pam_unix.so obscure use_authtok try_first_pass yescryp
↪ t
```

> **Note:** If the system is older and does not support `yescrypt`, `sha512` is the minimum acceptable hashing algorithm that MUST be used.

3.5 Account Lockout (Faillock)

Implementing an Account Lockout policy is a *mandatory* control to effectively mitigate brute-force password guessing attacks against system accounts. This is configured within the `/etc/pam.d/common-auth` file using the `pam_faillock.so` module.

The `pam_faillock.so` module MUST be configured twice: once in the pre-authentication stage (to reset the failure count upon success) and once in the authentication failure stage (to perform the lockout).

3.5.1 Account Lockout Policy Parameters

The following parameters MUST be applied to both `pam_faillock.so` lines:

Parameter	Policy Set	Effect
`deny=5`	Maximum attempts	MUST NOT allow more than `5` failed login attempts.
`unlock_time=180`	Lock duration	Locks the account for `180` seconds (`3` minutes) after the maximum failure attempts are reached.
`audit`	Logging	Logs all authentication failures to the system audit logs, improving traceability.

3.5.2 Configuration in `/etc/pam.d/common-auth`

The required configuration lines MUST be present in `/etc/pam.d/common-auth` as follows:

Pre-authentication Stage (Tracking Failures):

```
auth required pam_faillock.so preauth silent audit deny=5 unlock_time=180
```

Authentication Failure Stage (Enforcing Lockout):

```
auth [default=die] pam_faillock.so authfail audit deny=5 unlock_time=180
```

Note: The order of modules in the `/etc/pam.d/common-auth` file is critical. The `preauth` line MUST be placed near the top, and the `authfail` line MUST be placed after modules like `pam_unix.so` to ensure correct execution order.

3.6 Password & Account Aging

Enforcing *Password and Account Aging* is a *mandatory* control that minimizes the risk associated with long-lived or abandoned user accounts. This policy is defined across two main configuration files: `/etc/login.defs` (for system-wide aging) and `/etc/default/useradd` (for new user defaults).

Password Management Policy (`/etc/login.defs`)

Modern security standards from *NIST (SP 800-63B)*[3] and *ENISA*[4] no longer recommend periodic password rotation. Frequent forced changes lead to "password exhaustion," causing users to choose weaker, predictable patterns. Passwords should instead be long-lived and only changed if there is an indicator of compromise (IoC).

The parameters in `/etc/login.defs` must be adjusted to disable legacy rotation while maintaining a minimum age to prevent rapid cycling.

Parameter	Policy Set	Effect
`PASS_MAX_DAYS`	`99999`	Disables periodic expiration (aligned with NIST/ENISA).
`PASS_MIN_DAYS`	`1`	Prevents immediate changes, hindering users from cycling back to an old password instantly.
`PASS_WARN_AGE`	`7`	Provides a 7-day warning only if an expiration is manually set.

Implementation

Update `/etc/login.defs` to reflect these modern hardening standards:

```
# Set parameters to disable periodic rotation
> sed -i 's/^PASS_MAX_DAYS.*/PASS_MAX_DAYS   99999/' /etc/login.defs
```

[3] *https://www.enisa.europa.eu/sites/default/files/2025-06/ENISA_Technical_implementation_guidance_on_cybersecurity_risk_management_measures_version_1.0.pdf, page 142*
[4] *https://pages.nist.gov/800-63-4/sp800-63b.html#passwordver*

```
> sed -i 's/^PASS_MIN_DAYS.*/PASS_MIN_DAYS   1/' /etc/login.defs
> sed -i 's/^PASS_WARN_AGE.*/PASS_WARN_AGE   7/' /etc/login.defs
```

Verification

Verify the configuration:

```
> grep -iE "PASS_MAX_DAYS|PASS_MIN_DAYS|PASS_WARN_AGE" /etc/login.defs
```

Note: Existing accounts will not be updated automatically. To apply these changes to an existing user, use the `chage` command:

```
chage --maxdays 99999 --mindays 1 <username>
```

Account Inactivity Defaults (`/etc/default/useradd`)

These settings apply to new accounts created using the `useradd` utility.

File	Parameter	Policy Set	Effect
`/etc/default/useradd`	`INACTIVE=30`	Inactive period	The account is permanently disabled `30` days after its password expires. Since periodic password expiration is disabled (`PASS_MAX_DAYS 99999`), this setting only takes effect for accounts where an expiration is explicitly set, for example, temporary or service accounts configured via `chage`.
`/etc/default/useradd`	`EXPIRE=90`	Default expiration	*Optional:* This setting sets a default account expiration date (`90` days) from creation. This SHOULD only be used for temporary or service accounts, as it would cause all new user accounts to expire quickly.

3.7 Restricting Direct Root Login

Direct login as the `root` user removes the ability to audit which specific individual performed privileged actions. To ensure non-repudiation and accountability, administrators MUST log in as a standard user and escalate privileges using `sudo` or `su`.

While SSH configurations typically handle network-based root restrictions, local terminal access is controlled via PAM and the `/etc/securetty` file.

3.7.1 Configuring PAM for Secure TTYs

The system MUST check the `/etc/securetty` file during the login process. This is handled by the `pam_securetty.so` module.

The file `/etc/pam.d/login` MUST contain the following configuration line:

```
auth required pam_securetty.so
```

To verify that the module is active:

```
> grep "pam_securetty.so" /etc/pam.d/login
auth required pam_securetty.so
```

3.7.2 Restricting the Secure TTY List

The `/etc/securetty` file lists the device names of the TTYs (terminals) where the root user is allowed to log in. If this file does not exist, most systems default to allowing root login on any TTY. If the file exists but is empty, root login is disabled on all devices.

The `/etc/securetty` file MUST exist and SHOULD only contain the physical console and necessary serial terminals.

> **Note:** On virtualized cloud systems (e.g., AWS, Azure, KVM), the "physical" console is often redirected to a serial port (e.g., `ttyS0` or `hvc0`). Removing these from `/etc/securetty` will block access via the provider's emergency web console.

To safely configure this file for a standard physical server or desktop:

```
> echo -e "console\ntty1" | sudo tee /etc/securetty
```

To check the current content of the file:

```
> cat /etc/securetty
console
tty1
```

If the system requires serial console access (common for headless servers), ensure `ttyS0` (standard serial) or `hvc0` (Xen/virtualization) is included.

3.8 Harden Sudo Usage and Logging

Administrative actions MUST be logged and attributable to a specific human user. The `sudo` utility is the preferred method for privilege escalation, as it logs the actual command executed, unlike the `su` utility, which merely logs a successful user switch.

All controls outlined in this section MUST be implemented either directly in the `/etc/sudoers` file or, preferably, via a dedicated configuration file in the `/etc/sudoers.d/` directory.

3.8.1 Enforcing Pseudo-Terminal Use

The `use_pty` option forces `sudo` to allocate a pseudo-terminal for the command being executed. This prevents certain input-reading attacks and, more importantly, ensures that the output is properly recorded in the `sudo` log file, regardless of the application run.

The following MUST be added to a configuration file within `/etc/sudoers.d/`:

```
Defaults use_pty
```

3.8.2 Configure Dedicated Sudo Logging

By default, `sudo` events are logged through the system logger (Syslog), often mixed with other logs in `/var/log/auth.log` or `/var/log/secure`. To facilitate easier security monitoring and auditing, a dedicated log file SHOULD be configured.

```
Defaults logfile=/var/log/sudo.log
```

Note: The log file `/var/log/sudo.log` must be created and protected with restrictive permissions (e.g., `chmod 0600 /var/log/sudo.log`) before the setting takes effect.

3.8.3 Limiting Password Cache Timeout

When a user successfully authenticates with `sudo`, a timestamp is recorded, allowing subsequent `sudo` commands to run without requiring a password for a default duration (often 15 minutes). This duration MUST be reduced to minimize the window of opportunity if a logged-in session is left unattended.

Setting the `timestamp_timeout` to 5 minutes is a common, balanced approach for high security environments:

```
Defaults timestamp_timeout=5
```

3.8.4 Restricting Su Usage

Users who are granted administrative rights via `sudo` SHOULD NOT be allowed to use the `su` command to switch to the root account. The `su` utility is less auditable, as it does not log the individual commands executed after the switch.

This restriction is best implemented by requiring the user to be a member of a specific group (e.g., `wheel` or `adm`) via PAM. The `pam_wheel.so` module can be used in the `/etc/pam.d/su` configuration to enforce this restriction.

To verify the configuration is active for the `su` command, ensure the following line is present in `/etc/pam.d/su`:

```
auth required pam_wheel.so use_uid
```

This configuration ensures that only users who are explicitly part of the allowed group can switch user identity using `su`.

3.9 Session and Screen Lock Hardening

Unattended sessions, whether at a remote terminal or a local workstation, represent a significant security risk. To ensure a *locked-by-default* posture, both shell inactivity and graphical interface (GUI) timeouts MUST be enforced.

3.9.1 Shell Session Timeouts (CLI)

The *TMOUT* variable automatically terminates inactive shell sessions. This is critical for servers where SSH or console sessions might be left open. For high-security environments, this MUST be set to *300 seconds (5 minutes)* and made read-only to prevent user override.

Implementation

Create a global configuration file:

```
> sudo tee /etc/profile.d/90-tmout.sh <<EOF
TMOUT=300
export TMOUT
readonly TMOUT
EOF
```

3.9.2 Graphical Screen Lock (GUI)

On desktop systems, a shell timeout only closes the terminal window, leaving the rest of the desktop accessible. Therefore, the graphical session MUST be configured to lock automatically after 5 minutes of inactivity.

System-wide Enforcement (GNOME)

To prevent users from disabling the screen lock, create a *dconf* profile:

1. **Create the settings file:** `/etc/dconf/db/local.d/00-security-settings`

```
[org/gnome/desktop/session]
idle-delay=uint32 300

[org/gnome/desktop/screensaver]
lock-enabled=true
```

2. **Lock the settings (Prevent user override):** `/etc/dconf/db/local.d/locks/session`

```
/org/gnome/desktop/session/idle-delay
/org/gnome/desktop/screensaver/lock-enabled
```

Apply Changes

```
> sudo dconf update
```

3.9.3 Summary of Controls

Scope	Mechanism	Target Value	Effect
CLI	*TMOUT* Variable	*300s*	Logs user out of the shell.
GUI	*idle-delay*	*300s*	Activates screensaver/blank screen.
GUI	*lock-enabled*	*true*	Requires password to resume session.

Note: These settings do not interfere with long-running background tasks (e.g., compile jobs), only with interactive user engagement.

3.10 User Authentication Hardening

Securing how you log in to your computer is the first line of defense against physical theft or unauthorized local access. By hardening the authentication settings, you ensure that your personal data remains protected even if someone gains access to the device.

3.10.1 Strong Password Encryption

When you create a password, Linux does not store the plain text; it stores a *hash*. Older systems used weak methods that are easily cracked today. Modern systems MUST use a strong hashing algorithm. `YESCRYPT` is the recommended choice on modern distributions due to its memory-hard design, while `SHA512` remains acceptable where `YESCRYPT` is not supported.

To verify your system is using the strongest method:

```
> grep '^ENCRYPT_METHOD' /etc/login.defs
# SHOULD display YESCRYPT, while SHA512 also remains acceptable
```

3.10.2 Minimum Password Length

Short passwords are vulnerable to automated guessing. Enforcing a minimum length of *14 characters* ensures that you are using a passphrase that is computationally expensive to break.

This is managed via the *pwquality* module. You can check your current requirement here:

```
> grep 'minlen' /etc/security/pwquality.conf
```

3.10.3 Use Sudo, Not Root

On a secure personal Linux system, you should never log in directly as the *root* user. Instead, your personal user account should be granted administrative powers through *sudo*. Locking the *root* account prevents it from being targeted by malicious software or unauthorized users.

- **Verification**: You can check if the root account is locked by running `passwd -S root`. A locked account will show an *L* in the status field.
- **Safety**: Ensure your own user is in the *sudo* or *wheel* group before locking the root account to avoid losing administrative access.

4 Network Security & Services

The following sections describe hardening settings in the category Authentication. The settings include an acceptable secure password policy and hardening of the authentication process itself.

4.1 Verifying that Vulnerable and Not Required Software Is Disabled

Hardening a Linux system MUST include identifying and removing software that uses legacy protocols or exposes services to the network unnecessarily.

4.1.1 Remove Insecure Legacy Protocols

Insecure protocols transmit credentials and data in clear text. These MUST be purged from the system. Verification is done by checking the `dpkg` status.

```
> dpkg -l | grep -E '^ii\s+(rsh-client|telnetd|vsftpd)'
(Output should be empty)
```

- **R-Services (*rlogin*, *rsh*, *rcp*)**: These are fully replaced by the SSH suite.
- **Telnet Server (*telnetd*)**: Transmits all data unencrypted.
- **FTP Services (*vsftpd*, *pure-ftpd*)**: MUST be replaced with SFTP for secure file transfers.
- **Network Protocol Blacklisting**: Protocols like DCCP or SCTP should be disabled if not needed. Check via:

```
> lsmod | grep -E 'dccp|sctp'
```

4.1.2 Restrict or Disable Potentially Necessary Services

Some services provide functionality required locally. Instead of removal, these MUST be restricted to the loopback interface (`127.0.0.1`) to prevent network exposure.

- **CUPS Printing (*cups*)**: If printing is required, the daemon MUST only listen on localhost. Verify the configuration:

  ```
  > grep "^Listen" /etc/cups/cupsd.conf
  Listen localhost:631
  ```

- **Avahi Daemon (*avahi-daemon*)**: Provides Zero-configuration networking. It MUST be masked unless automatic discovery is strictly required.

  ```
  > systemctl is-active avahi-daemon
  ```

- **Bluetooth (*bluetooth*)**: On systems where Bluetooth is not used, the service MUST be masked to close the wireless attack vector.

4.1.3 Essential Hardening Targets

The following services MUST be checked. If they are not essential, they MUST be disabled.

- **TFTP Service (*tftpd*)**: Often used for network booting; MUST be removed if not in use.
- **File Sharing (*nfs*, *samba*)**: These MUST be restricted via firewall rules to specific trusted subnets if required.
- **Graphical Remote Desktop**: Remote access tools like VNC or RDP increase risk and MUST be disabled by default. Check for listening ports:

```
> ss -tulpn | grep -E '5900|3389'
```

4.2 Configuring and Hardening the Host Firewall

4.2.1 Configuring and Hardening the Host Firewall

A host-based firewall is a *mandatory* component that implements *Defense-in-Depth* by blocking unwanted traffic from the Internet or local network. The specific firewall tool differs by distribution:

- **Ubuntu / Debian / Arch**: *Uncomplicated Firewall (UFW)*, a frontend for `nftables`/`iptables`.
- **Rocky / Fedora / RHEL / OpenSUSE**: *firewalld*, a zone-based dynamic firewall daemon.

Firewall Status and Activation

The firewall MUST be enabled and running to enforce security rules.

On Ubuntu/Debian/Arch:

```
> sudo ufw status
Status: active

> sudo ufw enable        # if inactive
```

On Rocky/Fedora/RHEL/OpenSUSE:

```
> sudo firewall-cmd --state
running

> sudo systemctl enable --now firewalld   # if not running
```

4.2.2 Default Policy Hardening

A secure system follows the *Fail-Safe Default* principle: everything not explicitly permitted is denied.

- **Incoming Traffic**: MUST be set to *deny*. This prevents other devices or attackers from initiating connections to your computer.

  On Ubuntu/Debian/Arch (UFW):

  ```
  sudo ufw default deny incoming
  ```

  On Rocky/Fedora/RHEL/OpenSUSE (firewalld):

  ```
  sudo firewall-cmd --set-default-zone=drop --permanent
  sudo firewall-cmd --reload
  ```

- **Outgoing Traffic**: For personal use, this is typically set to *allow* so that browsers, updates, and applications can function without manual intervention.

  On Ubuntu/Debian/Arch (UFW):

  ```
  sudo ufw default allow outgoing
  ```

4.2.3 Reviewing and Removing Rules

Every open port is a potential entry point for attackers. If you have previously opened ports, review and remove them regularly.

On Ubuntu/Debian/Arch (UFW):

```
> sudo ufw status numbered
Status: active

#     To                         Action      From
#[ 1] 80/tcp                     ALLOW IN    Anywhere

> sudo ufw delete 1     # remove rule no longer needed
```

On Rocky/Fedora/RHEL/OpenSUSE (firewalld):

```
> sudo firewall-cmd --list-all
> sudo firewall-cmd --remove-service=http --permanent
> sudo firewall-cmd --reload
```

- **Note**: Always maintain a minimal ruleset. If you do not explicitly host a service, your ruleset should ideally be empty, relying entirely on the *default deny* policy.

4.3 SSH Client Security

Secure Shell (SSH) is a dual-purpose tool. While it is famous for managing servers, *end-users* primarily use it as a *client* to connect to other machines.

4.3.1 Eliminate the Server Attack Surface

By default, many Linux distributions install or enable the SSH server (*sshd*). For a personal workstation, this is an unnecessary open door. If you do not need to log into your computer from another device, the server component MUST be removed.

- **Verification**: Use the following command to check if the server is active. If it returns any result, the service is installed.

```
> dpkg -l | grep openssh-server
```

- **Action**: Purging the package completely removes the configuration and the listening service, ensuring no one can attempt to brute-force your machine over the network (e.g., on Debian/Ubuntu: `sudo apt-get purge -y openssh-server`).

4.3.2 Maintain the Secure Client

The SSH *client* is the part you use to initiate connections (e.g., `ssh user@remote-host`). This component is safe to keep and should be updated regularly to ensure the latest cryptographic patches are applied.

- **Safe Usage**: Even with the server removed, your client remains functional. You can still use *SFTP* through file managers like *Nautilus* or *Dolphin* to transfer files to remote storage securely.

4.4 Kernel Hardening

The Linux kernel is the core of your operating system. Hardening it involves configuring *runtime parameters* and *module restrictions* to reduce the available attack surface for both local and remote threats.

4.4.1 Runtime Self-Protection

Modern kernels include features to make exploitation significantly more difficult. These are primarily managed via *sysctl*.

- **ASLR (Address Space Layout Randomization)**: By setting `kernel.randomize_va_space=2`, the kernel randomizes where programs are loaded into memory, making it harder for attackers to predict target addresses for buffer overflows.
- **Information Leak Prevention**: Restricting *dmesg* access (`kernel.dmesg_restrict=1`) and hiding kernel pointers (`kernel.kptr_restrict=2`) prevents unprivileged users from gathering data needed to bypass security features like *KASLR*.
- **Ptrace Scope**: Setting `kernel.yama.ptrace_scope=1` ensures that a process can only be debugged by its parent, preventing malicious software from inspecting or injecting code into other running applications.
- **Disabling Core Dumps**: Setting `fs.suid_dumpable=0` prevents sensitive memory from being written to disk during application crashes, which could otherwise be mined for credentials or secrets.

4.4.2 eBPF and Sandboxing

While eBPF provides powerful tracing and networking capabilities, it also presents a significant attack surface if left open to unprivileged users.

- **eBPF Restrictions**: Setting `kernel.unprivileged_bpf_disabled=1` prevents non-root users from loading eBPF programs. Additionally, `net.core.bpf_jit_harden=2` enables JIT hardening to mitigate JIT spraying attacks.
- **User Namespace Cloning**: Setting `kernel.unprivileged_userns_clone=0` disables unprivileged user namespace creation. > **Note:** This is a high-security setting that will break rootless containers (Podman), Flatpaks, and certain browser sandboxes.
- **Filesystem Protection**: Setting `fs.protected_fifos=2` enables strict FIFO protection to prevent race conditions in named pipes, a common class of local privilege escalation.

4.4.3 Network Stack Parameters

In a Zero Trust environment, the client must protect itself even from the local network. These settings harden the stack against local MitM attacks and spoofing, which are common when connecting to untrusted networks (e.g., public Wi-Fi

or compromised internal segments). The network stack should be configured to prioritize security and integrity over legacy routing features.

- **Anti-Spoofing**: Enabling the Reverse Path Filter (`net.ipv4.conf.all.rp_filter=1`) forces the kernel to verify that a packet arrived on the interface it should have, preventing IP spoofing attacks.
- **Flooding Protection**: TCP SYN cookies (`net.ipv4.tcp_syncookies=1`) protect the system from *Denial of Service* attacks by validating connection requests without exhausting system resources.
- **Disabling Routing & Redirects**: A personal workstation should not act as a router or accept external routing updates.
  - `net.ipv4.ip_forward=0` (Disables forwarding)
  - `net.ipv4.conf.all.send_redirects=0` (Disables sending ICMP redirects)
  - `net.ipv4.conf.all.accept_redirects=0` (Disables accepting ICMP redirects)
  - `net.ipv4.conf.all.accept_source_route=0` (Disables source-routed packets)
- **ICMP Hardening**: Setting `net.ipv4.icmp_ignore_bogus_error_responses=1` ignores malformed ICMP error responses to reduce log noise and potential DoS vectors.

4.4.4 Restricting Kernel Modules

Unnecessary kernel modules can contain vulnerabilities. If a feature is not used, its code should not be allowed to execute.

- **USB Mass Storage**: For high-security environments, the `usb-storage` module can be blacklisted to prevent any USB drives from being mounted, stopping data exfiltration or malware entry.
- **Legacy Protocols**: Protocols like `DCCP` (Datagram Congestion Control Protocol) and `SCTP` (Stream Control Transmission Protocol) are rarely used on desktops and should be blacklisted to close potential network entry points.

4.4.5 Cryptography (FIPS)

For environments requiring high regulatory compliance, the OS should implement NIST FIPS-validated cryptography. This is typically audited via `/proc/sys/crypto/fips_enabled`. Enabling FIPS mode requires specific bootloader modifications and certified packages; it cannot be safely toggled via simple runtime commands as it may disable non-compliant SSH ciphers or other crypto primitives.

Implementation Procedure

To apply these settings persistently, create dedicated configuration files:

```
# General Kernel Hardening
> sudo tee /etc/sysctl.d/10-kernel-hardening.conf <<EOF
kernel.dmesg_restrict=1
kernel.randomize_va_space=2
```

```
fs.suid_dumpable=0
kernel.yama.ptrace_scope=1
kernel.kptr_restrict=2
kernel.unprivileged_bpf_disabled=1
net.core.bpf_jit_harden=2
kernel.unprivileged_userns_clone=0
fs.protected_fifos=2
EOF

# Network Hardening
>sudo tee /etc/sysctl.d/90-network-hardening.conf <<EOF
net.ipv4.tcp_syncookies=1
net.ipv4.conf.all.rp_filter=1
net.ipv4.conf.all.accept_redirects=0
net.ipv4.conf.all.accept_source_route=0
net.ipv4.icmp_ignore_bogus_error_responses=1
net.ipv4.ip_forward=0
net.ipv4.conf.all.send_redirects=0
EOF

# Module Blacklisting
> sudo tee /etc/modprobe.d/blacklist-unnecessary.conf <<EOF
install usb-storage /bin/true
blacklist dccp
blacklist sctp
EOF

# Apply sysctl changes immediately
> sudo sysctl -p /etc/sysctl.d/10-kernel-hardening.conf
> sudo sysctl -p /etc/sysctl.d/90-network-hardening.conf
```

4.5 Configure TCP Wrappers and Hosts Access

TCP Wrappers provide a host-based access control system for services compiled against the `libwrap.so` library. While most modern access control is handled by dedicated host firewalls (e.g., UFW/nftables), TCP Wrappers remain relevant as a secondary, service-specific security layer for compatible services like `sshd`, `vsftpd`, and `telnetd`.

4.5.1 Enforcement of Default Deny Policy

The security configuration MUST follow the principle of *Default Deny*. This is achieved by explicitly allowing only known good connections in `/etc/hosts.allow` and then denying everything else in `/etc/hosts.deny`.

The `/etc/hosts.deny` file MUST contain a blanket rule to deny all access to all wrapped services:

```
ALL: ALL
```

This ensures that any service linked against the library will deny a connection unless an explicit rule in `/etc/hosts.allow` overrides it.

To implement this mandatory deny rule:

```
> echo 'ALL: ALL' | sudo tee /etc/hosts.deny
```

4.5.2 Configuring Explicit Allow Rules

The `/etc/hosts.allow` file is where authorized services and hosts MUST be explicitly listed. The format is `daemon_list: client_list`.

For example, to allow SSH access for all users coming from the local network (`192.168.1.0/24`) and a specific administrative host (`10.10.10.5`):

```
sshd: 192.168.1. 10.10.10.5
```

> **Note:** You MUST define all necessary allow rules in `/etc/hosts.allow` before implementing the `ALL: ALL` deny rule in `/etc/hosts.deny`. Failure to do so will result in an immediate remote administrative lockout for the affected services. *Client systems* (workstations, endpoints) that do not require inbound SSH access MUST NOT add any `sshd` entry to `/etc/hosts.allow`. The blanket `ALL: ALL` deny rule in `/etc/hosts.deny` will then block all SSH connection attempts without any additional configuration.

4.5.3 Setting Restrictive Permissions

The integrity of the access control files is paramount. They MUST be protected against unauthorized modification. Permissions SHOULD be set to 0644, ensuring only the root user can write to the files.

To enforce the required permissions:

```
> sudo chmod 0644 /etc/hosts.allow
> sudo chmod 0644 /etc/hosts.deny
```

4.5.4 Verification of Service Linkage

To determine if a specific service is utilizing TCP Wrappers, you can check if the executable is linked against the `libwrap.so` library using the `ldd` utility.

Example check for the SSH daemon:

```
> ldd /usr/sbin/sshd | grep libwrap
```

```
libwrap.so.0 => /lib/x86_64-linux-gnu/libwrap.so.0 (0x00007f1f91b79000)
```

If the `libwrap.so` library is present in the output, the service respects the rules defined in `/etc/hosts.allow` and `/etc/hosts.deny`.

4.6 Secure Time Synchronization (Chrony)

Accurate time synchronization is mandatory for security. Reliable timestamps are critical for audit trails, log analysis, and authentication protocols such as Kerberos or TLS certificate validation. Time drift can render logs useless and break client authentication. Chrony is the preferred modern implementation over the legacy NTP daemon (ntpd).

4.6.1 Necessity of Time Synchronization

The system must maintain time synchronized with an authoritative external source. A deviation from real-world time should not exceed one minute, although best practice is to limit drift to a few seconds to ensure logs from different systems remain correlated.

Enforce Least Privilege

The time synchronization daemon must run as a dedicated, unprivileged user (typically chrony or ntp). This limits the extent of damage if the daemon were to be compromised. Modern distributions configure this separation by default via the systemd unit file.

To confirm the current user under which the daemon is running:

```
> ps -eo user,comm | grep chronyd
```

The output must show the process owned by a non-root user.

Restrict Network Access

The command and control interface for Chrony (chronyc) allows administrators to monitor and adjust the daemon. This interface must be restricted to the loopback interface (`127.0.0.1` and `::1`) to prevent external users from querying or manipulating the service.

The configuration file `/etc/chrony/chrony.conf` must contain:

```
bindcmdaddress 127.0.0.1
bindcmdaddress ::1
```

If the system must act as a time server for an internal network, the `allow` directive should be used to restrict client access to that specific subnet only.

Implement Network Time Security (NTS)

Traditional NTP is vulnerable to Machine-in-the-Middle (MitM) attacks, where attackers can supply false time to the client. This can be used to bypass security features like certificate expiration checks. Network Time Security (NTS) is the modern standard and must be used whenever possible to cryptographically authenticate the time source using TLS.

To use NTS, the configuration file requires specific servers and the nts option:

```
server nts.example.com iburst nts
```

Implementation Procedure

To apply the security configuration persistently, update the Chrony configuration and restart the service:

```
# Restrict command interface to localhost
> sudo tee -a /etc/chrony/chrony.conf <<EOF
bindcmdaddress 127.0.0.1
bindcmdaddress ::1
EOF

# Note: Ensure at least one NTS-capable server is added to /etc/chrony/chrony.conf
# Example: server nts.ntp.se iburst nts

# Apply changes
> sudo systemctl restart chronyd
```

4.7 IPv6 Attack Surface Reduction

4.7.1 IPv6 Stack Hardening

As a foundational pillar of modern networking, IPv6 is essential for system interoperability and internal container orchestration. Rather than disabling the stack, which risks breaking local service discovery and modern application dependencies, security efforts must focus on hardening the protocol parameters to mitigate discovery and redirection attacks.

Privacy Extensions (Temporary Addresses)

By default, IPv6 addresses can expose a device's hardware (MAC) address via EUI-64 identifiers, allowing for global host tracking. We enforce *Privacy Extensions* (RFC 4941) to ensure the system generates short-lived, random addresses for all outgoing traffic.

Note: Setting `net.ipv6.conf.all.use_tempaddr = 2` ensures that temporary addresses are preferred over static, hardware-derived identifiers.

Disabling Source Routing and Redirects

Paralleling IPv4 security measures[cite: 1], the IPv6 stack must be configured to reject packets attempting to dictate the return path or manipulate local routing tables via ICMPv6 redirects.

```
# Prevent MitM via routing redirects
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0

# Disable source routing (deprecated and vulnerable)
net.ipv6.conf.all.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0
```

Router Advertisements (RA) Policy

While Router Advertisements are often required for dynamic connectivity (SLAAC), they present a significant attack vector if the kernel processes all flags unconditionally. We harden the RA handling to accept the prefix while ignoring potentially malicious flags that could be used for Denial of Service (DoS).

Note: In environments with static IP assignments, set `accept_ra = 0`. For workstations: maintain `accept_ra = 1` for connectivity but strictly enforce `accept_redirects = 0` to mitigate rogue router risks.

Implementation Procedure

To apply a comprehensive hardening profile immediately:

```
# Apply IPv6 Hardening
> sudo tee /etc/sysctl.d/90-ipv6-harden.conf <<EOF
# Privacy Extensions (RFC 4941)
net.ipv6.conf.all.use_tempaddr = 2
net.ipv6.conf.default.use_tempaddr = 2

# MitM & Spoofing Protection
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
net.ipv6.conf.all.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0

# Router Advertisement Hardening
```

```
net.ipv6.conf.all.accept_ra_pinfo = 1
net.ipv6.conf.all.accept_ra_defrtr = 1
EOF

# Load the new parameters immediately
> sudo sysctl -p /etc/sysctl.d/90-ipv6-harden.conf
```

4.8 CUPS Security Hardening

4.8.1 CUPS Security Hardening: Disabling `cups-browsed`

The Common UNIX Printing System (CUPS) is a modular printing system that allows a local system to act as a print server. The cups-browsed service is responsible for discovering remote CUPS printers and advertising local shared printers on the network, typically using protocols like DNS-SD (Bonjour/mDNS).

While useful in certain network printing environments, running unnecessary network services significantly increases the attack surface of a system. The cups-browsed service, in particular, has been associated with critical remote code execution vulnerabilities (such as CVE-2024-47176). In these scenarios, attackers can potentially gain unauthorized access or control of the system by sending crafted packets to UDP port 631 and tricking the service into communicating with a malicious server.

Unless the system explicitly requires the automatic discovery or browsing of network printers, the cups-browsed service must be disabled and stopped. For most server or hardened workstation environments where printing is either not required or managed manually, this service should not be running.

Service Status Verification

The following command can be used to check the status of the service:

```
> systemctl status cups-browsed
```

The objective is for the output to show Active: inactive (dead) and for the service to be disabled. If the service is active or enabled, remediation is required.

Remediation Procedure

To stop the service immediately and prevent it from starting automatically at system boot, use the following command:

```
> sudo systemctl disable --now cups-browsed
```

This command both stops the running service and removes the symlink that enables it at startup. For systems requiring even stricter enforcement, the service can be masked to prevent any other process from triggering it:

```
> sudo systemctl mask cups-browsed
```

4.9 Disable or Remove Unnecessary File Sharing Services (Samba)

4.9.1 Disable or Remove Unnecessary File Sharing Services: Samba (SMB)

The Samba service provides *SMB/CIFS* file and print services for communication with Windows clients and other operating systems. On servers or workstations that do not explicitly need to act as a file or print server for Windows environments, the Samba service MUST be disabled or removed.

File sharing services are a significant network attack vector and have historically been subject to severe vulnerabilities. Disabling or removing this service is a fundamental step in reducing the system's attack surface.

The primary systemd units for the Samba service are typically `smbd` (the main daemon) and `nmbd` (NetBIOS name service).

Check and Disable the Samba Service

The following commands verify the status of the primary Samba service daemons:

```
> sudo systemctl status smbd
> sudo systemctl status nmbd
```

The desired output for these commands is that the services are `Active: inactive (dead)` and disabled at boot.

To ensure the services are stopped immediately and prevented from starting at boot, use the following command:

```
> sudo systemctl disable --now smbd nmbd
```

Removing the Samba Package

If Samba is not required for any function on the system, the packages SHOULD be completely removed. This is the most secure approach, as it eliminates the possibility of the service being inadvertently started or exploited.

> **Note:** The automated hardener cannot typically determine if the removal of a package will break other system dependencies. Therefore, complete removal MUST be performed manually by the administrator after assessing the impact.

Use the appropriate command for your distribution to completely remove the Samba package and its associated configuration files:

For Debian/Ubuntu-based Systems

```
> sudo apt purge samba
```

For RHEL/CentOS-based Systems

```
> sudo yum remove samba
```

For SUSE/openSUSE-based Systems

```
> sudo zypper remove samba
```

4.10 DNS Resolver Security

The Domain Name System (DNS) translates human-readable names into IP addresses. If this process is intercepted, attackers can redirect your browser to malicious websites. Protecting DNS integrity is vital for personal privacy and security.

4.10.1 Restrict Local Resolver Exposure

Many Linux desktops run a local resolver like *systemd-resolved* or *Unbound*. These services MUST be restricted to the loopback interface (`127.0.0.1`). If they listen on all interfaces, they could be exploited by other devices on your local network.

To verify your resolver is only listening locally:

```
> sudo ss -tuln | grep :53
tcp   LISTEN 0      4096      127.0.0.53%lo:53         0.0.0.0:*
```

The output should only show loopback addresses like 127.0.0.x or ::1.

4.10.2 Secure Transport and Validation

Standard DNS is unencrypted. To prevent Machine-in-the-Middle attacks, your system should use *DNS-over-TLS (DoT)* to encrypt queries or *DNSSEC* to cryptographically verify responses.

- **Systemd-resolved**: This is the default on most modern desktops. You can enable encryption by editing */etc/systemd/resolved.conf*:

  ```
  [Resolve]
  DNSOverTLS=yes
  ```

- **Validation**: Tools like *Unbound* can perform *DNSSEC* validation locally, ensuring that the data received has not been tampered with by an ISP or attackers.

4.10.3 Protect Resolver Configuration

The file */etc/resolv.conf* tells your system which DNS servers to use. It MUST be protected against unauthorized changes to prevent malware from redirecting your traffic.

Permissions MUST be set to *0644*:

```
> sudo chmod 0644 /etc/resolv.conf
```

- **Note**: On modern systems, this file is often a *symbolic link* managed by *systemd-resolved*. In this case, you are protecting the link itself and the underlying configuration it points to.

4.11 Ensure Correct Loopback and Local Host Configuration

The `/etc/hosts` file acts as a local address book for your computer. It allows the system to find itself and other local services before reaching out to the Internet. Proper configuration prevents apps from hanging while they wait for *DNS* responses that will never come.

4.11.1 Loopback Configuration

The *loopback address* is a virtual interface used for internal communication. If this is missing or misconfigured, core system components and local applications (like your web browser or printing services) may fail to start.

Both *IPv4* and *IPv6* versions MUST be defined:

```
127.0.0.1       localhost
::1             localhost
```

To verify these are present, use *grep* with a focus on the start of the line:

```
> grep -E '^\s*127\.0\.0\.1\s+localhost' /etc/hosts
127.0.0.1   localhost
```

4.11.2 2. Local Hostname Resolution

Your computer also needs to know its own name (*hostname*). On personal Linux desktops, it is standard practice to map the hostname to *127.0.1.1*. This keeps it separate from the generic *localhost* entry and avoids conflicts with some network services.

First, check your current hostname:

```
> hostname
my-linux-laptop
```

Then, ensure it appears in your *hosts* file mapped to a local address:

```
> grep "$(hostname)" /etc/hosts
127.0.1.1   my-linux-laptop
```

Note: Never map your hostname to an external *IP* address in this file if you move between networks (e.g., using a laptop on public Wi-Fi). Always use the *127.x.x.x* loopback range for your own hostname to ensure your traffic stays local.

Modifying the File: Because this is a system-critical file, you MUST use *sudo* to edit it. A simple typo can break your network connectivity entirely.

5 System Boot & Integrity Security

This section covers the security of the boot process, including BIOS/UEFI protection, bootloader hardening, and the enforcement of a cryptographic chain of trust through Secure Boot.

5.1 Secure BIOS/UEFI Settings

Security hardening MUST extend to the hardware firmware. Compromise at the *BIOS/UEFI* level allows attackers to bypass all operating system security measures.

5.1.1 BIOS/UEFI Administrator Password

An *Administrator Password* MUST be set to prevent unauthorized changes to security-critical options like *Secure Boot* or the boot order.

- **Access Control**: This password is required only to change settings, not for every boot, ensuring it does not hinder daily productivity.

5.1.2 Secure Boot Order

The system MUST NOT be allowed to boot from external media (USB, Network/PXE) by default. An unsecured boot order allows attackers with physical access to load a malicious OS and bypass your disk encryption or login screens.

- **Configuration**: Set the *local hard disk* as the first and only active boot device. Disable all other options.
- **Temporary Use**: If you need to boot from a recovery USB, you can temporarily re-enable the option using the *Administrator Password*.

5.1.3 Enable Secure Boot

Secure Boot validates that the bootloader and kernel are digitally signed by a trusted authority. This prevents rootkits from modifying the boot sequence.

Verification within the OS:

```
> sudo dmesg | grep -i "secure boot"
[    0.000000] secureboot: Secure boot enabled
```

5.1.4 Disable Unused Hardware

To reduce the physical attack surface, disable any onboard components that are not necessary for your workflow.

- **Legacy Ports**: Serial or parallel ports SHOULD be disabled.
- **Network**: If you only use Wi-Fi, consider disabling the integrated Ethernet controller.
- **Audio/Camera**: If the device is used in a high-security area, these can often be disabled at the firmware level for maximum protection.

5.2 Secure GRUB Bootloader with a Password

The *Grand Unified Bootloader (GRUB)* is the first software loaded during startup. An unsecured GRUB menu allows anyone with physical access to modify kernel boot parameters, such as booting into *single-user mode* to gain full root access without a password.

5.2.1 Password Protection Strategy

Protecting GRUB with a password ensures that kernel entries cannot be edited and the command line cannot be accessed without authentication.

- **PBKDF2 Hashing**: GRUB uses a secure *PBKDF2* hash to store passwords, ensuring that even if attackers read the configuration file, they cannot easily reverse the password.
- **Superusers**: You define a specific *superuser* (e.g., *grub_admin*) who is authorized to modify boot parameters.

5.2.2 Implementation Steps

Follow these steps to generate a secure hash and apply it to your configuration.

Generate the Hash

Execute the following utility and enter a strong, unique password:

```
> sudo grub-mkpasswd-pbkdf2
```

Copy the entire output string starting with `grub.pbkdf2.sha512...`.

Configure the Users

Add the hash and the superuser name to */etc/grub.d/40_custom* or a dedicated user file:

```
set superusers="grub_admin"
password_pbkdf2 grub_admin <YOUR_GENERATED_HASH>
```

5.2.3 3. Finalizing the Configuration

After saving your changes, the GRUB configuration MUST be regenerated to write the new settings into the active boot file.

```
> sudo update-grub
```

Note: On some distributions, the command is `sudo grub-mkconfig -o /boot/grub/grub.cfg`.

> **Note**: If you forget this password, you will be unable to access recovery modes or change boot settings without using external recovery media. Always keep a copy of this password in a secure physical location or password manager.

5.3 Kernel Command Line Hardening

Kernel Command Line parameters are applied at the earliest stage of the boot process. They are essential for enabling low-level security features and mitigating physical access attacks.

5.3.1 Memory Protection Rationale

The following parameters MUST be added to the `GRUB_CMDLINE_LINUX_DEFAULT` variable in `/etc/default/grub` to defend against advanced memory corruption and side-channel attacks.

Parameter	Purpose	Risk Mitigation
`slab_nomerge`	Prevents the kernel allocator from merging caches of different object types.	Thwarts heap exploitation techniques using Use-After-Free (UAF) bugs.
`vsyscall=none`	Disables the legacy virtual system call mechanism.	Improves Address Space Layout Randomization (ASLR) by removing fixed memory locations.
`page_poison=1`	Fills freed memory pages with a known pattern.	Detects UAF vulnerabilities by triggering a crash if poisoned pages are accessed.
`pti=on`	Page Table Isolation. Separates user-space and kernel-space page tables.	Mitigates the Meltdown (Spectre V3a) hardware vulnerability.

5.3.2 Prevent Initramfs Debug Shell Access

A critical, often overlooked attack vector allows attackers to drop into a debug shell via the Initial RAM Filesystem (initramfs). This shell can be reliably triggered if an incorrect password for the encrypted root partition is entered multiple times.

From this shell, attackers with physical access can modify the `initramfs` and inject persistent malware. This is possible because the `initramfs` itself is typically not signed, only the kernel image is, allowing modification without breaking Secure Boot signatures.

To mitigate this, the system MUST be configured to halt or reboot rather than dropping to a shell upon boot failure.

Configuration by Distribution Family

- Debian/Ubuntu: Use `panic=0` (forces reboot on panic).
- RHEL/Fedora: Use `rd.shell=0 rd.emergency=halt` (disables Dracut shell).

> **Note**: Applying this setting makes troubleshooting boot failures harder. If the system fails to boot, you will need rescue media (Live USB) instead of dropping into a maintenance shell.

Implementation

The `/etc/default/grub` file MUST be edited to include both the memory protections and the debug shell restrictions.

Example Modification (Debian/Ubuntu)

```
> cat /etc/default/grub | grep GRUB_CMDLINE_LINUX_DEFAULT
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash slab_nomerge vsyscall=none page_poison=1 pti=on p
↪ anic=0"
```

Example Modification (RHEL/Fedora)

```
> cat /etc/default/grub | grep GRUB_CMDLINE_LINUX
GRUB_CMDLINE_LINUX="... slab_nomerge vsyscall=none page_poison=1 pti=on rd.shell=0 rd.emer
↪ gency=halt"
```

After modifying the file, regenerate the GRUB configuration:

```
# Debian/Ubuntu
> sudo update-grub

# RHEL/Fedora
> sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```

Runtime Verification

After rebooting, verify that the active kernel command line contains the desired parameters.

```
> cat /proc/cmdline
BOOT_IMAGE=... slab_nomerge vsyscall=none page_poison=1 pti=on panic=0
```

5.4 UEFI Secure Boot Verification

UEFI Secure Boot is a mandatory security measure to prevent the execution of unauthorized or malicious code, such as bootkits and rootkits, during the pre-boot and boot processes. It relies on cryptographic signatures verified against keys stored in the system's firmware.

5.4.1 Verification of Secure Boot Status

The primary check is ensuring Secure Boot is actively enforced by the UEFI firmware. This state is necessary to validate the signatures of the bootloader (GRUB) and the kernel.

The `mokutil` utility is the preferred method for querying the status:

```
> mokutil --sb-state
SecureBoot is enabled
```

The output MUST explicitly state SecureBoot is enabled.

5.4.2 Verification of User Mode

Secure Boot operates in two primary modes:

1. **Setup Mode:** Allows modifications to the trusted key databases (DB, DBX, KEK). Enforcement is typically disabled or bypassed.
2. **User Mode:** Enforces the key databases, preventing any unauthorized code from running.

The system MUST be in User Mode for the protection to be active. You can check the firmware mode using `bootctl`:

```
> bootctl status | grep 'Setup Mode'
      Setup Mode: no
```

5.4.3 Kernel Module Signature Enforcement

When Secure Boot is active, the running kernel SHOULD be configured to enforce signature checks on all kernel modules loaded after boot. This ensures that attackers cannot load a malicious rootkit module into kernel memory after the system has started.

This check verifies the kernel's runtime setting:

```
> cat /proc/sys/kernel/module_sig_enforce
1
```

The value MUST be 1 (enforced). If it is 0 while Secure Boot is enabled, the kernel or kernel headers may be incorrectly configured or incompatible.

> **Note:** Secure Boot is a firmware setting. Any remediation to enable it requires physical access to the machine and configuration within the UEFI/BIOS settings. Ensure all components are signed (including third-party kernel modules like NVIDIA drivers) before enabling, or the system will fail to boot.

5.5 Verify Package Integrity

Regularly verifying the integrity of installed software is a critical detective control. This process compares current system files against the original metadata stored by the package manager to detect *corrupted*, *modified*, or *tampered* binaries.

5.5.1 Verification by Distribution

Every major package manager provides tools to audit the current state of the system against its "known-good" baseline.

Debian and Ubuntu (`debsums`)

The `debsums` utility checks the MD5 checksums of files installed via `dpkg`.

```
# Only show modified configuration files
sudo debsums -c
```

RHEL, Fedora, and openSUSE (`rpm`)

The `rpm` verify command checks size, permissions, and signatures.

```
# Verify all packages, ignoring configuration files ('c')
sudo rpm -Va | grep -v '^..c'
```

Arch Linux (`pacman`)

For Arch Linux users, the native package manager can check for missing or modified files:

```
# Check all installed packages for missing files
pacman -Qk
```

```
# Verify file integrity against package checksums (requires pacutils)
paccheck --md5sum --quiet
```

5.5.2 Interpreting Discrepancies

When a check fails, the tool provides codes indicating what changed. A *5* in RPM indicates a checksum mismatch, while an *S* indicates a change in file size.

Modifications to configuration files (often marked with a *c*) are expected if you have customized your system. > **Note**: Changes to binaries in `/bin`, `/sbin`, or `/usr/bin` that you did not explicitly update MUST be treated as a potential security breach.

5.5.3 3. Proactive Monitoring with AIDE

While package manager checks are useful, they rely on the package manager's own database, which attackers might also modify. *AIDE* (Advanced Intrusion Detection Environment) provides an independent cryptographic baseline for your most sensitive files.

- **Baseline**: After installation, run `sudo aideinit` to create the master database.
- **Routine**: Regularly run `sudo aide --check` to find any unauthorized changes to the system core.
- **Note**: Automated fixes for integrity failures are unsafe. If a binary is modified unexpectedly, treat the system as compromised and follow incident response procedures.

5.6 Protecting Single User Mode

Single User Mode (also known as Rescue Mode or Emergency Mode) is a privileged mode used for system maintenance. By default on some systems, entering this mode drops the user directly into a root shell without asking for a password. This poses a significant physical security risk: anyone with physical access to the console can reboot the machine, enter single user mode, and gain full root access without credentials.

5.6.1 Authentication Requirement

The system MUST be configured to require the root password (or another administrative credential) before granting access to Single User Mode, Rescue Mode, or Emergency Mode. This is typically achieved by forcing the use of `/sbin/sulogin`.

5.6.2 Configuration for Systemd (Modern Systems)

On modern systems using `systemd`, the behavior of rescue and emergency modes is defined by service units. You must ensure that `sulogin` is executed.

To verify this, check the `ExecStart` parameter in the relevant service files:

```
> grep /sbin/sulogin /usr/lib/systemd/system/rescue.service
ExecStart=-/usr/lib/systemd/systemd-sulogin-shell rescue
```

If `sulogin` is not present, or to enforce stricter controls, you SHOULD create override files rather than editing the system files directly.

Manual Remediation

1. **Rescue Mode:** Run `systemctl edit rescue.service` and add the following:

```
[Service]
ExecStart=
ExecStart=-/usr/lib/systemd/systemd-sulogin-shell rescue
```

2. **Emergency Mode:** Run `systemctl edit emergency.service` and add the following:

```
[Service]
ExecStart=
ExecStart=-/usr/lib/systemd/systemd-sulogin-shell emergency
```

> **Note:** The – before the path (e.g., `-/usr/lib/...`) indicates that systemd should ignore the exit code, but the use of the binary itself enforces the password prompt.

5.6.3 Operational Impact

Enabling this setting means that if the root password is lost, you cannot simply reboot into single user mode to reset it. You would need to boot from external media (Live CD/USB) or modify the kernel boot parameters (e.g., `init=/bin/bash`) to recover the system. Ensure the root password is securely stored in a password manager.

5.7 Disable Ctrl-Alt-Del Reboot Sequence

The *Ctrl-Alt-Del* key sequence is configured by default to initiate a system reboot. Unauthorized physical access combined with this sequence can be used to disrupt system availability or interrupt the boot process to attempt further exploits. The ability to reboot the system via this keyboard shortcut MUST be disabled.

5.7.1 Hardening with `systemd`

On modern systems, this key sequence is handled by the `ctrl-alt-del.target` unit file. To disable the automatic reboot, the unit file MUST be masked by creating a symbolic link to `/dev/null`.

To implement this change:

```
> sudo ln -sf /dev/null /etc/systemd/system/ctrl-alt-del.target
> sudo systemctl daemon-reload
```

The `daemon-reload` command ensures the change is registered immediately without requiring a full restart of the system.

5.7.2 Verification

You can verify the status of the target to ensure it is correctly redirected.

```
> systemctl status ctrl-alt-del.target | grep Loaded
Loaded: loaded (/dev/null; static)
```

The output MUST confirm the target is loaded from `/dev/null`. If it points to a file in `/lib/systemd/system/`, the shortcut is still active.

- **Note**: This setting only affects the physical keyboard sequence. It does not prevent an administrator from rebooting the system normally via the `reboot` command or the desktop environment's power menu.

5.8 CPU Microcode Updates

Microcode is low-level firmware that runs on the CPU itself. Because vulnerabilities like Spectre, Meltdown[5], and MDS (Microarchitectural Data Sampling)[6] exploit hardware design flaws, they often require patches to the CPU's microcode that cannot be fixed by software (kernel/OS) alone. Ensuring the latest microcode is loaded is a mandatory, non-negotiable step in modern system hardening.

5.8.1 Installation of Microcode Packages

Microcode updates are typically provided by the OS vendor in packages specific to the CPU manufacturer. The relevant package MUST be installed on the system:

- **Intel CPUs:** Install `intel-microcode` (Debian/Ubuntu) or `microcode_ctl` (RHEL/CentOS).
- **AMD CPUs:** Install `amd64-microcode` (Debian/Ubuntu) or `microcode_ctl` (RHEL/CentOS).

Example for Debian/Ubuntu:

```
> sudo apt install intel-microcode
```

5.8.2 Microcode Loading and Verification

The OS microcode is loaded during the early boot process, usually by the bootloader (GRUB) or via the initial RAM disk (`initramfs`). This overrides the older microcode embedded in the system's UEFI/BIOS.

To verify that the microcode has successfully updated, check the reported revision number in `/proc/cpuinfo`. A value greater than the BIOS default indicates a successful OS-level patch.

```
> grep 'microcode' /proc/cpuinfo
microcode   : 0x800113c
```

The output MUST show a revision number (e.g., `0x800113c`) indicating an update.

> **Note:** Installation of the microcode package requires a system reboot and sometimes a manual update of the `initramfs` (using `sudo update-initramfs -u` or equivalent) to ensure the microcode binary is included in the boot image.

5.8.3 Vulnerability Mitigation Status

Modern Linux kernels expose the mitigation status for known hardware vulnerabilities in the `/sys/devices/system/cpu/vulnerabilities/` directory.

[5] *https://meltdownattack.com/*
[6] *https://mdsattacks.com/*

The system MUST report a successful mitigation status for critical vulnerabilities like Spectre V2 (which relies heavily on microcode):

```
> cat /sys/devices/system/cpu/vulnerabilities/spectre_v2
Mitigation: eIBRS, IBPB, Retpolines
```

The presence of specific mitigation techniques (e.g., `eIBRS`, `Retpolines`) confirms that the kernel and the updated microcode are working together to protect the processor.

6 OS Hardening

The operating system layer represents the foundational attack surface of any Linux deployment. Beyond kernel and network parameter tuning, hardening at the OS level addresses physical attack vectors, process memory exposure, and privilege escalation paths that persist across reboots and user sessions. This section covers the restriction of USB mass storage devices via USBGuard, the suppression of core dump generation to prevent sensitive memory from being written to disk, and the enforcement of a clean, auditable PATH environment variable to eliminate binary hijacking opportunities.

6.1 Install and Configure USBGuard

6.1.1 Install and Configure USBGuard

USBGuard is a framework that allows you to control which USB devices are permitted to interact with your system. By default, Linux allows any plugged-in USB device to function, which creates risks such as BadUSB attacks, unauthorized data exfiltration, or the installation of malicious software.

The USBGuard service MUST be installed, configured with a restrictive policy, and actively running to mitigate these physical security risks.

> **Note:** USBGuard MUST NOT be started without a valid policy in place. Starting the service without a configured ruleset can block input devices such as keyboard and mouse, rendering the system unresponsive. Always generate and verify a baseline policy before starting the service.

Installation

The `usbguard` package MUST be installed using your distribution's package manager before proceeding.

Policy Configuration (`rules.conf`)

A secure USBGuard policy follows the principle of least privilege, using a whitelist model: only explicitly authorized devices are allowed; all others are implicitly blocked.

A secure policy consists of two primary components:

1. *Whitelist Rules*: Rules for all devices that are known and necessary (e.g., internal USB hubs, keyboard, mouse).
2. *Default Block Rule*: A final rule that ensures any unlisted or new devices are denied access.

Generating an Initial Policy

The `usbguard` utility can generate a policy based on your currently connected devices. This is the recommended starting point for creating your personal whitelist. Ensure all essential peripherals (keyboard, mouse) are connected before running this command:

```
> sudo usbguard generate-policy > /etc/usbguard/rules.conf
```

Enforcing the Block-by-Default Policy

The last line of the `/etc/usbguard/rules.conf` file MUST be the generic block rule.

```
# The final rule MUST be a block rule to deny all other devices
block
```

6.1.2 Service Activation

Once a valid policy is in place, the service MUST be started and tested before being enabled at boot.

Start for Testing only

```
> sudo systemctl start usbguard
```

Verify that all required devices (keyboard, mouse, and other essential peripherals) continue to function correctly. If a device is blocked, update the policy accordingly before proceeding.

Verification

```
> sudo systemctl is-active usbguard
```

Enable at Boot (only After Successful Validation)

```
> sudo systemctl enable usbguard
```

Applying Policy Changes

After modifying the configuration at any point, you MUST reload the policy:

```
> sudo usbguard set-policy apply
```

6.2 Disable Core Dumps

6.2.1 Disable Core Dumps

A core dump is a memory image file created by the operating system when a program terminates unexpectedly (crashes). While useful for developers and system administrators for debugging purposes, core dump files often contain sensitive data, including passwords, encryption keys, and proprietary information that was held in memory at the time of the crash. Storing these files on disk represents a significant data confidentiality risk.

To mitigate this risk, the system MUST be configured to prevent the creation of core dump files for all users and processes.

6.2.2 Configure User Limits (`/etc/security/limits.conf`)

The primary method for disabling core dumps for user processes is to set the hard limit for the core file size to zero (`0`) in the `/etc/security/limits.conf` file. This prevents unprivileged users from writing core dumps.

The following line MUST be added or verified to exist in `/etc/security/limits.conf`:

```
* hard core 0
```

- The asterisk (`*`) applies this rule to all users.
- `hard` sets an enforced maximum value.
- `core 0` sets the maximum size of a core file to zero bytes.

Verification

```
> grep -E "^\*\s+hard\s+core\s+0$" /etc/security/limits.conf
* hard core 0
```

6.2.3 Disable the `systemd-coredump` Service

Modern Linux distributions using `systemd` may use the `systemd-coredump` service to manage and store core dumps in the journal. To fully disable core dump creation, this service MUST be stopped, disabled, and masked to prevent it from being accidentally started by another service or administrator.

Remediation

```
> sudo systemctl stop systemd-coredump
> sudo systemctl disable systemd-coredump
> sudo systemctl mask systemd-coredump
```

The `mask` command ensures that the service unit file cannot be started even if another service attempts to pull it as a dependency, providing the strongest form of disabling.

Verification

```
> sudo systemctl is-enabled systemd-coredump
masked
```

6.3 Securing the Path Environment Variable

The `PATH` environment variable lists the directories searched by the shell for executable commands. If a directory in the `PATH` is writable by non-privileged users (globally writable), it creates a severe vulnerability. Attackers can place a malicious executable with the same name as a common command, such as `ls` or `cd`, in that directory. When a privileged user executes the command, the trojan may run instead of the legitimate binary, leading to privilege escalation.

The system MUST NOT include any globally writable directories in the default `PATH`.

6.3.1 1. Check for Globally Writable Paths

Standard system directories like `/bin`, `/usr/bin`, `/sbin`, and `/usr/sbin` MUST NOT be writable by anyone except the `root` user.

Run the following command to check common system directories for the world-writable bit (`/0002`):

```
> find /usr/local/sbin /usr/local/bin /usr/sbin /usr/bin /sbin /bin -type d -perm /0002 2>
↪ /dev/null
```

The output MUST be empty. Any listed directory represents an immediate security risk and requires manual intervention.

6.3.2 2. Location of PATH Configuration

The `PATH` variable is usually defined in global shell configuration files. You should review these to ensure no insecure or relative paths (like `.`) are being added:

- `/etc/profile`
- `/etc/bash.bashrc`
- Files within `/etc/profile.d/`
- **Security Note**: Regardless of where the variable is set, the primary control is ensuring the *target directories* are not writable by untrusted users.

- **Manual Action**: If a directory is found to be writable, investigate the cause before changing permissions. Use `chmod o-w <directory>` to secure it once the risk is understood.

7 File System & Permissions

A misconfigured file system is one of the most reliable footholds for local privilege escalation. Overly permissive defaults, improperly mounted partitions, and world-writable or unowned files can each be leveraged to elevate privileges or persist malicious code across system updates. This section establishes baseline controls across the entire file system: enforcing a restrictive default `umask`, separating critical partitions, hardening mount options to restrict execution and device access, locking down the `/proc` virtual file system, auditing SUID/SGID binaries, identifying world-writable and unowned files, and isolating temporary directory namespaces through polyinstantiation.

7.1 Enforce Restrictive Default Umask

The *umask* (user file-creation mode mask) is a fundamental security setting that determines the default permissions for every new file or directory you create. By enforcing a restrictive mask, you ensure a *secure-by-default* posture for all user data.

7.1.1 Defining the Hardening Standards

For a secure workstation, there are two primary tiers of protection:

- **Standard Hardening (027)**: This is the recommended baseline. It results in new files having `640` permissions (`rw-r-----`) and directories having `750` (`rwxr-x---`). This allows you full access, permits your group to read the data, and completely blocks all other users.
- **High-Security (077)**: This stricter setting is used when absolute privacy is required. It results in files having `600` (`rw-------`) and directories `700` (`rwx------`). Only the owner has any access; even group members are excluded.

7.1.2 Implementation in System Configuration

The *umask* must be defined in multiple locations to cover both account creation and active shell sessions.

Global Account Defaults

The file */etc/login.defs* controls the initial environment for new users. Ensure only one active *UMASK* entry exists to avoid configuration conflicts.

```
grep '^UMASK' /etc/login.defs
```

Shell Session Initialization

The */etc/profile* file sets the environment for interactive shells. Because the *root* user often requires a more permissive mask (*022*) to ensure system-wide readability of configuration files, conditional logic is used.

7.1.3 Recommended Conditional Logic

To implement a "really good" standard that protects users without breaking the system, add the following to your */etc/profile*:

```
if [ $UID -gt 199 ]; then
    umask 027
else
    umask 022
fi
```

- **Safety Note**: This logic ensures that unprivileged users (UID > `199`) are restricted to `027`, while system accounts and `root` maintain the standard `022` for compatibility.
- **High-Security Modification**: If your threat model requires maximum isolation, change the user value to `umask 077`. Be aware that this will prevent local file sharing between users entirely.

7.2 Partitioning and Mount Options

Partitioning and mount options provide a physical and logical defense-in-depth layer. By isolating high-growth and world-writable directories, you prevent resource exhaustion attacks and restrict the execution of unauthorized payloads.

> **Note:** Re-partitioning and modifying `cryptsetup` or `/etc/fstab` on a running system can lead to *permanent* data loss or a *non-bootable* system. Always perform a full backup before proceeding.

7.2.1 Partitioning Requirements

Isolating specific directories prevents a "Filesystem Full" condition from impacting the entire OS. The following directories SHOULD reside on dedicated partitions:

- **`/boot`**: Isolates the kernel and bootloader.
- **`/home`**: Prevents user data from consuming system space.
- **`**/var` & `/var/log**`**: Ensures application data and logs don't fill the root partition.
- **`/var/log/audit`**: Specifically protects audit logs from being overwritten or causing system crashes during high activity.
- **`/tmp`**: World-writable space susceptible to resource flooding.

Verification

Use `findmnt` to inspect current mount points:

```
> findmnt -v
```

7.2.2 Restrictive Mount Options

Apply the **Principle of Least Privilege** to your filesystems using specific flags:

- **`nodev`**: Do not interpret character or block special devices on the filesystem.
- **`nosuid`**: Disallow the set-user-identifier (SUID) bits to take effect.
- **`noexec`**: Prevent the direct execution of any binaries on the mounted filesystem.

Recommended `/etc/fstab` Configuration

Partition	Recommended Options	Purpose
`/tmp`	`defaults,nodev,nosuid,noexec`	Block malware execution in temp space.
`/dev/shm`	`defaults,nodev,nosuid,noexec`	Harden shared memory.
`/home`	`defaults,nodev,nosuid`	Prevent SUID abuse in user space.
`/boot`	`defaults,nodev,nosuid,noexec`	Protect boot files from being used as vectors.

7.2.3 Encryption (LUKS)

For portable devices, **Full Disk Encryption (FDE)** is the standard for protecting data-at-rest. LUKS (Linux Unified Key Setup) provides a standardized header and multi-slot key management system.

Setting up a LUKS Partition

1. **Initialize the LUKS container**:

```
# WARNING: This wipes all data on the target partition
> cryptsetup luksFormat /dev/sdXn
```

2. **Map the device**:

```
> cryptsetup open /dev/sdXn crypt_data
```

3. **Format and mount**:

```
> mkfs.ext4 /dev/mapper/crypt_data
> mount /dev/mapper/crypt_data /mnt
```

Automated Unlocking via `/etc/crypttab`

To ensure the OS unlocks the device during boot, add it to `/etc/crypttab`:

```
# TARGET        SOURCE DEVICE       KEY FILE    OPTIONS
crypt_data      UUID=<your-uuid>    none        luks,discard
```

Swap Encryption

Swap space can leak sensitive information (e.g., encryption keys or passwords cached in RAM) to the disk.

```
# /etc/crypttab entry for volatile (random key) swap
> swap /dev/sdXn /dev/urandom swap,cipher=aes-xts-plain64,size=256
```

Note: Using `/dev/urandom` for swap encryption breaks hibernation (Suspend-to-Disk) because the key is lost on every reboot.

7.2.4 Polyinstantiated Directories

Polyinstantiation creates private, isolated instances of `/tmp` and `/var/tmp` for each user session, mitigating symlink attacks and information leakage between users.

Configure `/etc/security/namespace.conf`:

```
/tmp     /tmp-inst/            level      root,adm
/var/tmp /var/tmp-inst/        level      root,adm
```

7.2.5 Additional Filesystem Hardening

Beyond partitioning, specific permissions must be enforced:

- **Sticky Bits**: All world-writable directories MUST have the sticky bit set to prevent users from deleting files owned by others.
- **Critical Files**: Ensure permissions for `/etc/shadow` (640) and `/etc/passwd` (644) are strictly maintained to prevent unauthorized credential access.

7.3 Enforcing Restrictive Mount Options

Applying restrictive mount options in `/etc/fstab` is a highly effective way to implement security controls that cannot be bypassed by changing file permissions. These settings implement kernel-enforced limitations on file system capabilities, significantly mitigating local attack vectors.

The following options are MANDATORY for partitions that store user-writable data, temporary files, or volatile memory:

Option	Rationale	Attack Mitigated
`nodev`	Prevents device files (special files) from being interpreted.	Creation of malicious device files (e.g., character devices) in user-writable directories.
`nosuid`	Prevents execution of SUID/SGID binaries on the partition.	Exploitation of flawed SUID programs installed in user areas, preventing local privilege escalation.
`noexec`	Prevents the direct execution of any executable file on the partition.	Execution of malware, scripts, or compiled exploits stored in temporary directories.

7.3.1 Configuration for Temporary Filesystems

Filesystems used for temporary data (e.g., `/tmp`, `/var/tmp`, `/dev/shm`) MUST be mounted with all three restrictive options: `nodev`, `nosuid`, and `noexec`.

Example **`/etc/fstab`** Entry for **`/tmp`**

```
/dev/mapper/vg0-tmp /tmp ext4 defaults,nodev,nosuid,noexec 0 2
```

To verify the current mount status for `/tmp`:

```
> findmnt -n /tmp
/tmp ext4 [rw,nosuid,nodev,noexec,relatime]
```

7.3.2 Configuration for User Filesystems

Partitions containing user home directories (e.g., `/home`) MUST be mounted with at least `nodev` and `nosuid` to prevent users from installing malicious SUID programs or device files in their personal directories. Execution of regular user binaries is generally permitted here, meaning `noexec` is usually omitted unless dictated by strict policy.

Example /etc/fstab entry for `/home`:

```
/dev/mapper/vg0-home /home ext4 defaults,nodev,nosuid 0 2
```

7.3.3 Applying Changes

After modifying `/etc/fstab`, the changes can be applied without a full system reboot by remounting the individual filesystems:

```
> sudo mount -o remount /tmp
> sudo mount -o remount /home
```

7.4 Hardening the Proc Filesystem

The `/proc` filesystem provides a window into the running kernel and processes. By default, any local user can inspect nearly all process information, including the process ID (PID), command line arguments, environment variables, and memory maps of other users, including service accounts and the root user. This information leakage is critical for attackers performing reconnaissance or planning a local privilege escalation (LPE) attack.

The system MUST restrict visibility into the `/proc` filesystem using the `hidepid` mount option.

7.4.1 Correctly Implementing Hidepid

The `hidepid` option is applied to the `proc` filesystem definition in `/etc/fstab`. Setting `hidepid` to `2` provides the highest level of security by making process directories (`/proc/<PID>`) invisible to users who are not the process owner and are not part of the specified administrative group.

The `hidepid=2` setting ensures: * Processes are invisible to unprivileged users unless they belong to the process's owning group. * The system only reveals processes necessary for the user to see (typically only their own, or system-critical processes via the designated group).

To enforce this, ensure your `/etc/fstab` entry for `/proc` includes the `hidepid=2` option:

```
proc    /proc   proc    defaults,hidepid=2      0       0
```

Note: If you need specific administrative users (e.g., members of the `adm` or `monitoring` group) to see all processes, you should use the combination `defaults,hidepid=2,gid=<group_id>`. This allows users in that `<group_id>` to view all processes, while denying access to others.

7.4.2 Applying Changes

After modifying `/etc/fstab`, the change MUST be applied by remounting the `/proc` filesystem:

```
> sudo mount -o remount /proc
```

7.4.2.1 Verification

After applying the changes, an unprivileged user attempting to list the `/proc` directory should only see a limited number of directories (mostly virtual files, and their own process directories if `gid` is not used).

7.5 Audit and Restrict SUID/SGID Executables

Files that have the Set User ID (SUID) or Set Group ID (SGID) permission bits set are a primary vector for Local Privilege Escalation (LPE) attacks. The SUID bit allows a user to run an executable with the permissions of the file owner (typically `root`). Any vulnerability in an SUID binary can be exploited by an unprivileged user to gain root access.

The system MUST undergo a regular audit to identify, justify, and minimize the number of files with these elevated permissions.

7.5.1 Auditing SUID/SGID Files

The following command MUST be used to list all files on the system with the SUID (`4000`) or SGID (`2000`) bit set, where the sum is represented by the permission code `06000`:

```
> sudo find / -xdev -type f -perm /06000 2>/dev/null
```

The key components of this command are: * **`/`**: Start search from the root directory. * **`-xdev`**: Do not cross filesystem boundaries (prevents searching mounted NFS/Samba shares or user mounts). * **`-type f`**: Only search for regular files (executables). * **`-perm /06000`**: Search for files where any of the SUID or SGID bits are set. * **`2>/dev/null`**: Suppress "Permission denied" errors for cleaner output.

7.5.2 Remediation Policy

The list generated by the audit command MUST be reviewed against a whitelist of necessary binaries. Only essential utilities that require elevated privileges to function (e.g., `passwd`, `sudo`, `mount`) SHOULD retain these bits.

Remediation

If a binary is found to have the SUID or SGID bit set unnecessarily, those bits MUST be removed using the `chmod` utility.

Example: Removing both SUID (`u-s`) and SGID (`g-s`) bits from a binary named `unnecessary_tool`:

```
> sudo chmod u-s,g-s /path/to/unnecessary_tool
```

> **Note:** You MUST ensure that the SUID bit is not removed from critical system binaries (e.g., `/usr/bin/passwd` or `/usr/bin/sudo`). If the SUID bit is removed from these files, the system will immediately lose its ability to manage user passwords or perform privilege escalation.

Verification

After remediation, verify that the permissions have been correctly removed:

```
> stat /path/to/unnecessary_tool
```

```
Access: (0755/-rwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
```

The permission string MUST NOT contain 's' where the owner or group execute bit is present.

7.6 World Writable File and Directory Audit

Files and directories writable by the *world* allow any user on the system to modify, replace, or delete content regardless of ownership. This represents a critical risk to system integrity and can facilitate *privilege escalation*.

7.6.1 Auditing for World-Writable Files

The goal is to locate files where the *other* write bit (`o+w`) is set. In a hardened environment, this check should return no results for regular files.

To audit regular files (`-type f`) across local filesystems (`-xdev`):

```
> sudo find / -xdev -type f -perm -0002 2>/dev/null
```

Remediation

Any file discovered must be inspected. Unless it is a specific socket or pipe required by an application, remove the bit immediately:

```
> sudo chmod o-w /path/to/file
```

7.6.2 System Paths vs. Home Directories

The risk profile changes depending on where the world-writable object is located.

System-Wide Paths (`/etc`, `/usr`, `/var`)

World-writable files in system paths are an emergency. If a script in `/etc/cron.daily/` or a binary in `/usr/bin/` is world-writable, any user can replace it with a malicious version that will eventually be executed with `root` privileges.

User Home Directories (`/home/user`)

Users sometimes make files world-writable to "share" data with colleagues. This is a dangerous practice. If a configuration file like `.bashrc` or an SSH key is world-writable, another user could gain control of your account.

- **Better Approach**: Use *Groups* or *Access Control Lists (ACLs)* for sharing instead of opening files to the entire system.

7.6.3 Auditing for World-Writable Directories

World-writable directories (`-type d`) are generally only permitted in designated temporary areas like `/tmp`, `/var/tmp`, and `/dev/shm`. In these areas, the *sticky bit* must be set to ensure users can only delete their own files.

Directories outside these areas MUST NOT be world-writable. Audit them using:

```
> sudo find / -xdev -type d -perm -0002 -not -path '/tmp' -not -path '/var/tmp' -not -path
↪  '/dev/shm' 2>/dev/null
```

7.6.4 Remediation and Best Practices

Permissions should always be set as restrictively as possible to follow the *principle of least privilege*.

- **Configuration Files**: These should typically be `0600` (read/write for owner only) or `0640` (read/write for owner, read-only for group).
- **System Binaries**: These must be owned by `root` and never be world-writable.
- **Directories**: Ensure the *Sticky Bit* (`+t`) is applied to any directory that must remain world-writable for functional reasons.

7.7 Audit and Remediate Unowned and Ungrouped Files

Unowned (no valid user) and ungrouped (no valid group) files exist when the numeric User ID (UID) or Group ID (GID) recorded in the file system metadata does not correspond to an entry in the system's `/etc/passwd` or `/etc/group` files. This typically occurs when a user or service account is deleted without first transferring or deleting the files they owned.

The existence of such files poses a security risk because if the orphaned UID/GID is later reassigned to a new, potentially unprivileged user, that user will automatically gain full ownership of the old files, which could lead to unauthorized data access or modification.

The number of unowned and ungrouped files MUST be zero.

7.7.1 Auditing for Unowned Files

To identify files with no valid owner, the `find` utility with the `-nouser` flag MUST be used across all local filesystems (`-xdev`).

```
> sudo find / -xdev -nouser 2>/dev/null
```

7.7.2 Auditing for Ungrouped Files

To identify files with no valid group, the `find` utility with the `-nogroup` flag MUST be used.

```
> sudo find / -xdev -nogroup 2>/dev/null
```

Remediation

Any file identified by the audit commands MUST be remediated. The remediation policy dictates one of the following actions:

1. **Deletion:** If the file is obsolete and no longer required.
2. **Reassignment:** If the file is needed, ownership MUST be reassigned to a valid system user and group (e.g., `root`, `system`, or `nobody`).

Example Reassignment: To reassign an unowned file to the `root` user and group:

```
> sudo chown root:root /path/to/unowned/file
```

Reassigning the file ensures that future user creation or system operations will not inadvertently grant ownership to an incorrect entity, thereby maintaining system integrity and accountability.

7.8 Polyinstantiation of Temporary Directories

Polyinstantiation is an advanced hardening technique that utilizes Linux namespaces and mount points to provide each user or session with a private, isolated instance of a directory. This is primarily aimed at high-risk, world-writable directories such as `/tmp` and `/var/tmp`.

The isolation prevents unauthorized information leakage and denial-of-service (DoS) attacks, as files created by one user (or a sensitive service) are invisible to other unprivileged users. This is a mandatory control for systems handling highly sensitive data.

7.8.1 Configure Namespace Definitions

The file `/etc/security/namespace.conf` defines which directories should be polyinstantiated, what type of user/group should be isolated, and the type of temporary filesystem to use (e.g., `tmpfs`).

The following lines MUST be added to `/etc/security/namespace.conf` to isolate the temporary directories based on the user ID (`user`) upon login:

```
# Directory    Type    Context     Method
/tmp          tmpfs   user        create
/var/tmp      tmpfs   user        create
```

This configuration instructs the system to create a new, private `/tmp` and `/var/tmp` directory instance every time a new user session begins.

7.8.2 Activate the PAM Module

The logic for applying polyinstantiation and manipulating the mount namespace at login is handled by the **`pam_namespace.so`** module. This module MUST be activated in the session stack of the relevant PAM configuration files (`/etc/pam.d/login` and `/etc/pam.d/sshd` are common targets).

The following entry MUST be present in the session stack of the PAM configuration files:

```
session required pam_namespace.so
```

> **Warning:** Due to the complexity of the PAM stack and the potential for disrupting critical services, enabling `pam_namespace.so` MUST be tested rigorously. Misconfiguration can lead to the inability to start remote or local sessions.

7.8.3 Runtime Verification

After implementing the configuration and logging in as two different unprivileged users (UserA and UserB), run the `mount` command in both sessions and filter for the temporary directories:

```
> mount | grep tmp
tmpfs on /tmp type tmpfs (rw,nosuid,nodev,relatime,size=...)
```

While the mount point remains `/tmp`, the underlying mount namespace and contents are isolated. You MUST verify that a file created by UserA in `/tmp` is not visible or accessible by UserB.

8 Application Security & Logging

Even a well-hardened kernel and file system can be undermined by applications operating without sufficient logging, access control, or confinement. This section addresses the security posture of core system services: ensuring `auditd` captures a complete and tamper-evident record of security-relevant events, configuring `syslog` and `journald` for reliable and integrity-protected log collection, restricting `cron` and `systemd` timer access to authorized users, hardening `systemd` unit configurations to limit service privileges, and establishing a Mandatory Access Control framework to enforce least-privilege execution boundaries across the system.

8.1 Audit Framework Installation and Setup

The Linux Auditing Framework, managed by the **`auditd`** daemon, is a fundamental security control. It MUST be installed, enabled, and running to capture security-relevant events, enforce accountability, and provide forensic data. The framework tracks system calls, file access violations, mandatory access control (MAC) failures, and privileged command execution (`sudo`, `su`).

8.1.1 Installation and Activation

The `auditd` package MUST be installed on the system. Once installed, the associated `systemd` service MUST be enabled for persistence across reboots and started immediately.

Package Installation

Use the appropriate package manager command for your distribution (e.g., `apt` for Debian/Ubuntu, `dnf` or `yum` for RHEL/CentOS):

```
> sudo apt install auditd
```

Service Enablement and Runtime Status

The service MUST be enabled and running. Use `systemctl` to manage the service state:

```
# Enable service for boot persistence
> sudo systemctl enable auditd

# Start the service immediately
> sudo systemctl start auditd
```

To verify the runtime status:

```
> systemctl is-active auditd
active
```

```
> systemctl is-enabled auditd
enabled
```

Note: If the service is not active, security events are not being logged. Continuous operation of `auditd` is mandatory for meeting compliance and security requirements.

Verification of Audit Rules

The installation of `auditd` provides the framework, but the specific events to be logged are defined in configuration files (covered in subsequent sections). A basic rule check can confirm the framework is ready to receive rules:

```
> sudo auditctl -s
enabled 1
```

The output `enabled 1` confirms that the kernel's audit subsystem is running and active.

8.2 Audit Rules for Identity and Privilege Escalation

Monitoring events related to user identity and privilege escalation is the single most important function of the Linux Auditing Framework. Every attempt to change user credentials, group memberships, or escalate privileges MUST be logged for accountability and immediate incident detection.

These rules MUST be placed in a dedicated file, such as `/etc/audit/rules.d/90-identity.rules`, and loaded via `auditctl -R`.

8.2.1 Monitoring Critical Identity Files

Any write (`w`) or attribute change (`a`) to the core user identity and group files must trigger an audit event. This detects both malicious and unintended changes to the system's authentication and authorization configuration.

The following files MUST be monitored for `wa` (Write and Attribute change):

```
-w /etc/passwd -p wa -k identity_change
-w /etc/shadow -p wa -k identity_change
-w /etc/group -p wa -k identity_change
-w /etc/gshadow -p wa -k identity_change
-w /etc/sudoers -p wa -k identity_change
-w /etc/sudoers.d/ -p wa -k identity_change
```

8.2.2 Monitoring Privilege Escalation Binaries

The execution of programs designed specifically to elevate privileges (SUID binaries) MUST be audited. This allows administrators to trace exactly who attempted to gain root access and what command was executed.

The execution of the binaries is monitored using the system call interface (`-a always,exit`) and filtering by the file path (`-F path=`):

```
-a always,exit -F path=/usr/bin/su -F perm=x -k privilege_escalation
-a always,exit -F path=/usr/bin/sudo -F perm=x -k privilege_escalation
-a always,exit -F path=/usr/bin/chage -F perm=x -k identity_change
-a always,exit -F path=/usr/bin/usermod -F perm=x -k identity_change
-a always,exit -F path=/usr/sbin/groupmod -F perm=x -k identity_change
```

8.2.3 Monitoring Session and Login Events

Login, logout, and session lifecycle events are crucial for tracking user accountability. These are logged via the system call interface.

```
-w /var/log/tallylog -p wa -k login_events
-w /var/run/faillock/ -p wa -k login_events
-w /var/log/lastlog -p wa -k login_events
```

8.2.4 Applying Rules

After creating or modifying the rules file (`90-identity.rules`), the rules MUST be loaded into the kernel:

```
> sudo auditctl -R /etc/audit/rules.d/90-identity.rules
```

This command loads the configuration without requiring a system reboot.

8.3 Audit Rules for System Integrity and File Access

System integrity relies on the ability to detect unauthorized modifications to critical files, permissions, and the running kernel state. The Linux Auditing Framework MUST monitor these low-level system calls to ensure complete visibility into system tampering.

These rules MUST be placed in a dedicated file, such as `/etc/audit/rules.d/91-system-access.rules`.

8.3.1 Monitoring File Attribute Changes

Attackers who gain a foothold on the system will attempt to hide their presence by changing permissions or ownership of files. Monitoring syscalls that modify file attributes is mandatory.

The following syscalls MUST be monitored for execution:

- `chmod`, `fchmod`, `fchmodat` (Permission changes)
- `chown`, `fchown`, `fchownat` (Ownership changes)
- `setxattr`, `fsetxattr`, `removexattr`, `fremovexattr` (Extended attribute changes)

Example Ruleset (Syscall Monitoring for Attribute Change)

```
-a always,exit -F arch=b64 -S chmod,fchmod,fchmodat -F success=0 -k perm_failure
-a always,exit -F arch=b64 -S chown,fchown,fchownat -F success=0 -k own_failure
-a always,exit -F arch=b64 -S setxattr,fsetxattr,removexattr,fremovexattr -k xattr_change
```

8.3.2 Monitoring File Deletion and Renaming

Unauthorized file deletion (Denial of Service) or malicious renaming/replacement of system binaries is a common attack step.

The following syscalls MUST be monitored:

- `unlink`, `unlinkat` (File deletion)
- `rmdir` (Directory deletion)
- `rename`, `renameat` (File/directory renaming)
- `truncate`, `ftruncate` (File truncation)

Example Ruleset (Syscall Monitoring for Deletion)

```
-a always,exit -F arch=b64 -S unlink,unlinkat,rmdir,rename,renameat -k file_delete
-a always,exit -F arch=b64 -S truncate,ftruncate -k file_truncate
```

8.3.3 Monitoring Kernel Module Management

The insertion of a malicious kernel module is the primary method for deploying a sophisticated rootkit. Monitoring all module management syscalls provides the earliest warning of such an attack.

The following syscalls MUST be monitored:

- `init_module` (Loading a new module)
- `delete_module` (Unloading a module, often done to hide a trace)

Example Ruleset (Syscall Monitoring for Kernel Activity)

```
-a always,exit -F arch=b64 -S init_module -k module_load
-a always,exit -F arch=b64 -S delete_module -k module_unload
```

8.3.4 Applying Rules

After creating or modifying the rules file (`91-system-access.rules`), the rules MUST be loaded into the kernel:

```
> sudo auditctl -R /etc/audit/rules.d/91-system-access.rules
```

This command loads the configuration without requiring a system reboot.

8.4 Enforcing Immutable Audit Configuration

After successfully configuring all necessary audit rules, the configuration MUST be protected against tampering by attackers gaining root privileges. Without protection, attackers could simply run `sudo auditctl -D` (delete all rules) and then proceed with their attack undetected.

The concept of immutable audit configuration locks the ruleset in the kernel, making it impossible to modify or clear the rules without a system reboot. This is a mandatory control to guarantee the forensic integrity of the system.

Configuration

The immutable mode is set using the `-e 2` flag, which MUST be the very last executable line in the entire audit ruleset. This prevents any subsequent rules from being loaded or the configuration from being changed.

It is recommended to place this rule in a final, dedicated configuration file (e.g., `99-finalize.rules`):

```
# Ensure this is the last line of all audit configuration files
-e 2
```

To create this file and set the rule:

```
> echo '-e 2' | sudo tee /etc/audit/rules.d/99-finalize.rules
```

8.4.1 Applying Immutable Mode

The immutable rule can only be truly enforced if it is loaded during the initial boot process.

Initial Setup

You MUST run `sudo auditctl -R` (or similar, depending on the distribution's mechanism) to load all rules, and then reboot the system.

The `auditd` service will load the ruleset, hit the `-e 2` rule, and lock the configuration.

Runtime Verification

Once the system has rebooted, the status of the audit system MUST confirm that it is running in immutable mode:

```
> sudo auditctl -s | grep enabled
enabled 2
```

The output `enabled 2` confirms the immutable status. If the status is `enabled 1`, the configuration is active but not locked, meaning attackers could still disable it.

> **Note:** Because the rules cannot be changed until the next reboot, you MUST verify that your entire ruleset is correct and does not excessively log non-security relevant data *before* setting this flag. Excessive logging can lead to a Denial of Service (DoS) by filling the disk partition.

8.5 Syslog Daemon Hardening

System logs contain critical forensic evidence, sensitive authentication attempts, and information about the system's state. Hardening the Syslog daemon (commonly `rsyslog` or `syslog-ng`) is mandatory to ensure the confidentiality and integrity of this data, and to prevent the daemon itself from becoming an attack vector.

8.5.1 Disable Network Listening (Attack Surface Reduction)

If the host is not intended to act as a centralized log collector, the Syslog daemon MUST NOT listen on any network sockets. Listening on UDP port 514 or TCP port 601 exposes the daemon to remote attacks, including Denial of Service (DoS) and potential overflows.

Verify that network receiver modules are disabled in the daemon's configuration file (e.g., `/etc/rsyslog.conf`). The following lines MUST be commented out or removed:

```
# $ModLoad imudp
# $UDPServerRun 514
# $ModLoad imtcp
# $InputTCPServerRun 601
```

After modifying the configuration file, the daemon MUST be restarted to apply the change:

```
> sudo systemctl restart rsyslog
```

8.5.2 Restrict Log File Permissions (Confidentiality)

Critical log files MUST be protected from unauthorized reading by unprivileged users. This is especially true for logs containing authentication data (`auth.log`), mail logs, and kernel messages.

Permissions MUST be set to `0640` (Read/Write for owner, Read for group, no access for others) or tighter. The log files are typically owned by `root:adm` or `syslog:adm`.

The following command MUST be run for all critical logs, particularly `auth.log`:

```
> sudo chmod 0640 /var/log/auth.log
```

To verify the permission:

```
> stat -c "%a" /var/log/auth.log
640
```

8.5.3 Secure Remote Forwarding

If the host MUST forward logs to a centralized server (SIEM), plaintext transmission over UDP/TCP is strictly prohibited. Logging MUST utilize a secure, authenticated transport protocol, such as RELP (Reliable Event Logging Protocol) or TLS/SSL encryption over TCP.

Example configuration for TLS-encrypted forwarding (rsyslog):

```
# Use reliable TCP forwarding with encryption
$ActionSendStreamDriverMode 1   # Use TLS
$ActionSendStreamDriverAuthMode anon
$ActionSendStreamDriverPermittedPeer log_collector.example.com
*.* @@log_collector.example.com:6514
```

8.6 Systemd Journal Hardening and Integrity

The `systemd-journald` daemon manages a structured, indexed log system that is the primary logging facility on modern Linux distributions. Security hardening is mandatory to ensure log integrity, retention, and restricted access.

8.6.1 Enforcing Persistent Storage (Forensic Integrity)

By default, some distributions set `Storage=auto` or `Storage=volatile`, which means logs are stored in `/run/log/journal` and cleared on reboot. For security and forensic integrity, logs MUST persist across reboots.

Set the `Storage` parameter to persistent in `/etc/systemd/journald.conf`:

```
Storage=persistent
```

This configuration ensures that logs are stored in `/var/log/journal`, which survives system reboots.

8.6.2 Enabling Compression

To prevent log files from unnecessarily consuming large amounts of disk space, especially on high-volume systems, compression SHOULD be enabled.

Set the `Compress` parameter to yes in `/etc/systemd/journald.conf`:

```
Compress=yes
```

8.6.3 Log Forwarding to Traditional Syslog

If the system utilizes a traditional syslog daemon (`rsyslog` or `syslog-ng`) for centralized log aggregation (SIEM), the journal MUST be configured to forward copies of all messages.

Set the `ForwardToSyslog` parameter to yes in `/etc/systemd/journald.conf`:

```
ForwardToSyslog=yes
```

8.6.4 Protecting Persistent Log Directory

The persistent log directory (`/var/log/journal`) contains sensitive data and MUST be protected with restrictive permissions. The directory owner (root) and group (`systemd-journal`) are the only entities that should have write access. The SetGID (2) bit MUST be set on the directory to ensure that new files created within it inherit the `systemd-journal` group ownership.

The permissions MUST be set to `2755` (rwxr-xr-x with SetGID bit):

```
> sudo chmod 2755 /var/log/journal
```

Applying Changes

After modifying `/etc/systemd/journald.conf`, the `journald` daemon MUST be restarted:

```
> sudo systemctl restart systemd-journald
```

8.7 Cron Security

8.7.1 `cron` And `at` Security

The `cron` and `at` utilities allow users to schedule repetitive or one-time tasks, respectively. Poor configuration of access controls for these utilities can lead to privilege escalation, denial-of-service conditions, or unauthorized resource usage by compromised or malicious user accounts.

To securely manage access, the system MUST use an explicit allow-list approach by configuring the `*.allow` files and ensuring that the corresponding `*.deny` files do not exist.

Implementing the Allow-List Principle

The configuration files control access to `cron` and `at` as follows (in order of precedence):

1. If the file `/etc/cron.allow` exists, only users listed in it are allowed to use `cron`.
2. If `/etc/cron.allow` does not exist, then users listed in `/etc/cron.deny` are forbidden from using `cron`.
3. If neither file exists, the default behavior is typically to allow all users (or to allow only the `root` user, depending on the distribution).

The security best practice is to enforce the most restrictive control, which is to use an explicit allow-list containing only the `root` user.

8.7.2 Configuration for `cron`

The following steps ensure that only the `root` user can create or modify `cron` jobs:

1. **Remove the deny-list:** Remove the `/etc/cron.deny` file to ensure it doesn't interfere with the allow-list setting.
2. **Create the allow-list:** Create the `/etc/cron.allow` file and add *only* the `root` user.
3. **Secure Permissions:** Set restrictive permissions on the `/etc/cron.allow` file.

The remediation command to achieve this state is:

```
> sudo rm -f /etc/cron.deny
> sudo echo "root" > /etc/cron.allow
> sudo chmod 0640 /etc/cron.allow
> sudo chown root:root /etc/cron.allow
```

You can verify the result with the following commands:

```
> sudo cat /etc/cron.allow
root
> ls -l /etc/cron.deny
# ls: cannot access '/etc/cron.deny': No such file or directory
```

8.7.3 Configuration for `at`

The same principle and steps MUST be applied to the `at` utility by configuring the `/etc/at.allow` file:

1. **Remove the deny-list:** Remove the `/etc/at.deny` file.
2. **Create the allow-list:** Create the `/etc/at.allow` file and add *only* the `root` user.
3. **Secure Permissions:** Set restrictive permissions on the `/etc/at.allow` file.

The remediation command to achieve this state is:

```
> sudo rm -f /etc/at.deny
> sudo echo "root" > /etc/at.allow
> sudo chmod 0640 /etc/at.allow
> sudo chown root:root /etc/at.allow
```

8.8 Systemd Service Sandboxing and Resource Control

`systemd` provides robust sandboxing features for system services. These features MUST be utilized to implement least privilege and attack surface reduction, minimizing potential damage if a service is compromised.

The best practice is to use configuration override directories (e.g., `/etc/systemd/system/<service>.service.d/`) to apply hardening without altering the original distribution files.

8.8.1 Protect System Directories (`ProtectSystem`)

This setting mounts core OS directories as read-only. This defends against compromised services attempting to modify system binaries or configuration.

Value	Description
`no`	No protection; allows write access to `/usr`, `/etc`, etc..
`yes`	Read-only mounts `/usr` and `/boot`.
`full`	Read-only mounts `/usr`, `/boot`, and `/etc`.
`strict`	MANDATORY: Entire file system hierarchy is read-only, except for API file systems.

For critical services, `ProtectSystem` MUST be set to `strict`:

```
[Service]
ProtectSystem=strict
```

8.8.2 Protect User Home Directories (`ProtectHome`)

This makes user home directories (`/home`, `/root`) inaccessible. A compromised web server should never have the ability to read private user files.

```
[Service]
ProtectHome=yes
```

If a service requires access to specific paths, exceptions can be made using `ReadOnlyPaths` or `ReadWritePaths`.

8.8.3 Isolate Temporary Storage (`PrivateTmp`)

`PrivateTmp` mounts a private, ephemeral instance of `/tmp` and `/var/tmp` for the service. These are destroyed when the service stops. This prevents the service from manipulating or reading temporary files used by other users or processes.

```
[Service]
PrivateTmp=true
```

8.8.4 Applying Overrides

Hardening is implemented by creating a file (e.g., `hardening.conf`) in the service's override directory.

Example for `sshd.service`:

1. Create the directory: `> sudo mkdir -p /etc/systemd/system/sshd.service.d`.
2. Create the override file: `> sudo nano /etc/systemd/system/sshd.service.d/hardening.conf`.

Example Content

```
[Service]
ProtectSystem=strict
ProtectHome=yes
PrivateTmp=true
```

Apply the changes:

```
> sudo systemctl daemon-reload
> sudo systemctl restart sshd
```

8.9 Mandatory Access Control (MAC) Implementation

Mandatory Access Control (MAC) is your system's second line of defense. While standard permissions allow you to control your own files, MAC enforces a global security policy that even the `root` user must follow. It prevents compromised applications from reaching into parts of the system they don't need.

8.9.1 SELinux (Security-Enhanced Linux)

Predominantly found on Fedora and RHEL-based systems, SELinux labels every file, process, and port with a security context. It is incredibly powerful but can be strict.

- **Enforcement**: Your system MUST be in Enforcing mode to provide actual protection. Permissive mode only logs violations without blocking them.
- **Verification**: Check your status with:

```
> getenforce
Enforcing
```

> **Note**: If you are experiencing weird "Permission Denied" errors despite having the right file permissions, check `/var/log/audit/audit.log`. It is likely SELinux doing its job.

8.9.2 AppArmor (Application Armor)

AppArmor is the standard for Ubuntu, Debian, and SUSE. It uses profiles attached to specific applications to restrict their capabilities based on their installation path.

- **Arch Linux Support**: While Arch Linux does not enable a MAC by default, it fully supports AppArmor. You can install the `apparmor` package and add `lsm=landlock,lockdown,yama,apparmor,bpf` to your kernel parameters to enable it.

> **Note**: If you use Burp Suite Professional, AppArmor's default profiles for `unprivileged_userns_clone` might break the built-in Chromium browser. You may need to create an exception or a specific profile for it.

8.9.3 Mode Selection and Testing

MAC frameworks can be disruptive if enabled blindly.

- **Monitoring**: Before going full *Enforcing*, run your system in *Permissive* (SELinux) or *Complain* (AppArmor) mode for a few days. This allows you to identify which of your personal apps might need custom rules.
- **Persistence**: Ensure your settings are saved in `/etc/selinux/config` or that the AppArmor service is enabled:

```
> sudo systemctl enable --now apparmor
```

Note: MAC is not a replacement for good file permissions, but a safety net for when an application's own security fails.