



LightPath

ENTERPRISE ENGINEERING & HARDENING MANUAL

LINUX HARDENING & SECURITY GUIDE

Production Baseline & Defense-in-Depth Architecture — Version 2.0 (2026)
Red Hat Enterprise Linux 9 / CentOS Stream 9 & Ubuntu 24.04 LTS

Publisher:	LightPath Consulting & Software Engineering (https://www.lightpath.fr)
Lead Author:	Jean-Marie RENOUARD (https://www.jmrenouard.fr)
Publication Date:	October 28, 2026
Document Revision:	Version 2.0 (Enterprise Edition)
License:	Creative Commons CC BY-SA 4.0
Target Platforms:	RHEL 9 / CentOS Stream 9 & Ubuntu 24.04 LTS

Published by: LightPath Consulting & Software Engineering (<https://www.lightpath.fr>)

Author: Jean-Marie Renouard (<https://www.jmrenouard.fr>)

Publication Date: October 2026

Document Revision: 2.0 (Enterprise Edition)

License: Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

Target Distributions: Red Hat Enterprise Linux 9 / CentOS Stream 9 & Ubuntu 24.04 LTS (Noble Numbat)

Document Classification: Technical Reference & Operational Engineering Manual

Document Revision History

Version	Date	Author	Description
1.0	2006-04-24	Jean-Marie Renouard	Initial version of the Linux server security manual.
1.1	2008-06-21	Jean-Marie Renouard	Document enhancements, installation section addition, license modernization.
1.2	2008-08-22	Jean-Marie Renouard	Syntax updates, typographical corrections, and path standardizations.
1.3	2009-02-23	Jean-Marie Renouard	Updated operational recommendations regarding administrator identity on production servers.
1.4	2009-03-09	Jean-Marie Renouard	Multiple architectural updates and visual illustration additions.
1.5	2011-11-12	Jean-Marie Renouard	Global security review, SysVinit service auditing, and command updates.
1.6	2021-12-26	Jean-Marie Renouard	Modernization updates across server components and system integrity checks.
1.7	2022-12-20	Jean-Marie Renouard	Added comprehensive password security and management section (corrected typographical error in previous release displaying year 2222).

1.8	2024-03-18	Jean-Marie Renouard	Added initial container security section for Docker and MySQL workloads.
2.0	2026-10-28	Jean-Marie Renouard (LightPath)	Complete enterprise overhaul and modernization. Target distributions upgraded to RHEL 9 / CentOS Stream 9 and Ubuntu 24.04 LTS. Deprecated tools eliminated (iptables, portsentry, afick, chkrootkit, SysVinit). Modern security stack integrated: nftables/firewalld/ufw, fail2ban with systemd-journald backend, AIDE (SHA-512), Lynis compliance auditing, immutable auditd ruleset. Added OpenSSH hardening with Ed25519-sk / FIDO2 and TOTP PAM MFA, systemd sandboxing (ProtectSystem=strict, NoNewPrivileges), full disk encryption with LUKS2, comprehensive /etc/sysctl.d/99-security-hardening.conf, SELinux and AppArmor dual coverage, rootless Podman/Docker with cgroups v2, read-only rootfs and Trivy image scanning, 5 Mermaid architectural diagrams, LightPath corporate branding, and 100% verified normative references.

Executive Summary & Foreword

In today's threat landscape, Linux servers power the overwhelming majority of mission-critical enterprise workloads—from relational database engines and distributed financial ledgers to cloud-native Kubernetes clusters. Consequently, Linux operating systems represent the primary target for advanced persistent threats (APTs), ransomware cartels, supply chain injectors, and automated exploitation bots.

Securing an enterprise Linux system is not a one-time checklist or a superficial collection of disconnected tips. True security requires a coherent, multi-layered engineering discipline known as **Defense-in-Depth**. If an

adversary circumvents the perimeter firewall, the host network layer must thwart their lateral movement. If an attacker exploits a zero-day vulnerability in a user-facing daemon, the process sandboxing and Mandatory Access Control (MAC) mechanisms must prevent privilege escalation. If an attacker gains local user execution, kernel hardening and storage encryption must guarantee that sensitive data, cryptographic keys, and system binaries remain uncompromised.

This manual, **Version 2.0**, represents a complete, ground-up rewrite of the original security reference created in 2006. It reflects twenty years of operational experience designing, securing, and tuning tier-one infrastructure for banks, telecommunication operators, energy conglomerates, and enterprise platforms. All legacy tools and historical anachronisms (such as **iptables**, **portsentry**, **afick** using MD5 checksums, unmaintained rootkit detectors, and SysVinit scripts) have been systematically discarded.

In their place, this guide establishes a verified, production-grade security baseline engineered specifically for the two enterprise distribution families dominating the modern data center: **Red Hat Enterprise Linux 9 (RHEL 9 / CentOS Stream 9)** and **Ubuntu 24.04 LTS (Noble Numbat)**. Every command, configuration snippet, and architectural model presented in this document has been designed for direct operational deployment, zero administrative friction, and rigorous compliance with international standards, including the **ANSSI GNU/Linux Hardening Recommendations**, the **CIS Security Benchmarks**, and **NIST SP 800-123**.

Table of Contents

Document Revision History	2
Executive Summary & Foreword	3
Table of Contents	5
Chapter 1: Introduction, Governance & Security Architecture	10
1.1 Scope, Purpose and Operational Objectives	10
1.2 Target Distributions: RHEL 9 & Ubuntu 24.04 LTS	10
1.3 LightPath Corporate Presentation & Cardinal Pillars	11
About LightPath	11
1.4 About the Author — Jean-Marie Renouard	12
1.5 Security Norms & Regulatory Frameworks (ANSSI, CIS, NIST, NIS2, DORA)	12
1.6 The Linux Defense-in-Depth Architecture	13
1.7 Typographical Conventions & Command Taxonomy	14
Chapter 2: Installation & Secure Partitioning	15
2.1 Principles of Minimalist Installation ("Simple is Beautiful")	15
Removing Development Tools and Unneeded Package Groups	15
2.2 Enterprise Partitioning Scheme & Mount Point Isolation	15
2.3 Hardened Mount Options (`noexec`, `nosuid`, `nodev`, `tmpfs`)	16
2.4 Complete Production `/etc/fstab` Reference	16
2.5 Full Disk Encryption at Rest with LUKS2 & Argon2id	17
Formatting and Provisioning a LUKS2 Encrypted Volume	17
Automated Unlocking in Enterprise Environments (Clevis & Tang / TPM 2.0)	18
2.6 Volatile Memory Filesystems (`tmpfs`) Hardening	18
Chapter 3: Kernel Hardening & System Parameters	20
3.1 Kernel Self-Protection Architecture (KSP & ASLR)	20
3.2 Network Stack Protections (SYN Cookies, Anti-Spoofing & ICMP Restrictions)	20
3.3 Debugging & Memory Restrictions (`ptrace`, `dmesg`, `kptr`)	20
3.4 Filesystem Link Exploitation Protections	21
3.5 eBPF & Kernel Module Lockdown	21
3.6 Production Configuration: `/etc/sysctl.d/99-security-hardening.conf`	21
3.7 Verification and Runtime Enforcement	23
Chapter 4: Access Control, Privileges & Identity Management	24

4.1 Principle of Least Privilege & Service Account Architecture	24
4.2 Superuser (Root) Account Locking	24
Locking the Root Account Correctly.....	24
4.3 Hardened Sudo Configuration & Administrative Delegation.....	24
Modern Sudo Architecture: Drop-in Configuration.....	25
4.4 Pluggable Authentication Modules (PAM) Hardening.....	25
4.5 Account Lockout Policies (<code>`pam_faillock`</code>).....	26
Configuring <code>`pam_faillock`</code> on RHEL 9	26
Managing Locked Accounts	26
4.6 Password Complexity & Quality Enforcement (<code>`pam_pwquality`</code>).....	26
4.7 CLI Password Generation & Password Managers (KeePassXC, Bitwarden)	27
Chapter 5: OpenSSH Hardening & Multi-Factor Authentication (MFA).....	28
5.1 Threat Landscape & Modern Cryptographic Algorithms	28
5.2 Production Configuration: <code>`/etc/ssh/sshd_config.d/99-security-hardening.conf`</code>	28
5.3 Client Key Generation Best Practices (Ed25519).....	29
5.4 Hardware Key Authentication via FIDO2/U2F (<code>`ed25519-sk`</code>)	30
5.5 Multi-Factor Authentication (MFA) with PAM & Google Authenticator TOTP	30
Installing the PAM Module	30
Initializing TOTP for an Operator	30
Configuring PAM for SSH MFA.....	31
5.6 Two-Phase SSH Access Flow Architecture.....	31
5.7 Verification, Banner Hygiene & Session Limits.....	32
Banner Hygiene.....	32
Chapter 6: systemd Security & Application Sandboxing.....	33
6.1 The systemd Security Model & Attack Surface Assessment	33
6.2 Service Exposure Profiling with <code>`systemd-analyze security`</code>	33
6.3 Sandboxing Directives & Resource Confinement.....	33
6.4 Linux Capabilities & System Call Filtering (<code>`SystemCallFilter`</code>).....	34
Dropping Linux Capabilities.....	34
System Call Filtering (<code>`SystemCallFilter`</code>)	34
6.5 Hardened Service Unit Drop-in Template (<code>`override.conf`</code>)	34
6.6 Operational Implementation & Troubleshooting	35
Chapter 7: Mandatory Access Control (MAC): SELinux & AppArmor.....	37

7.1 MAC vs. DAC Architectural Foundations	37
7.2 SELinux Architecture & Administration on RHEL 9.....	37
Verifying SELinux Status	37
Managing SELinux Security Booleans.....	37
Managing File Contexts & Network Ports	38
Analyzing Rejections & Generating Custom Modules	38
7.3 AppArmor Architecture & Administration on Ubuntu 24.04 LTS.....	38
Inspecting AppArmor Status	39
Enforcing and Profiling Applications	39
7.4 Operational Case Study: Database Engine Confinement (MySQL / MariaDB)	39
Chapter 8: Network Filtering, Firewalls & Intrusion Prevention	41
8.1 Modern Linux Packet Filtering: The `nftables` Paradigm.....	41
8.2 Dynamic Firewall Management on RHEL 9 with `firewalld`	41
Zone Architecture & Default Drop Policy	41
Granular Rich Rules for Administrative Subnets	41
8.3 Host Firewalling on Ubuntu 24.04 LTS with `ufw` & `nftables`	42
8.4 Automated Intrusion Prevention with `fail2ban` (systemd Backend)	42
Configuring Fail2ban for SSH (`/etc/fail2ban/jail.d/sshd.local`).....	42
8.5 Integrated Network Filtering & Threat Banning Flow	43
Chapter 9: System Integrity, Auditing & Intrusion Detection	45
9.1 File Integrity Monitoring (FIM) with AIDE (SHA-512)	45
Installing and Initializing AIDE.....	45
Automated Integrity Checks via systemd Timer.....	46
9.2 Kernel System Call Auditing with `auditd`	46
9.3 Production Audit Rules: `/etc/audit/rules.d/audit.rules`	46
9.4 Automated Compliance & Security Auditing with Lynis	48
Installing and Running Lynis.....	48
9.5 Continuous Auditing & Telemetry Pipeline	48
Chapter 10: Container Security & Runtime Hardening	50
10.1 Container Threat Modeling & Kernel Escape Vectors	50
10.2 Rootless Container Architecture (Podman & Rootless Docker)	50
Enabling Rootless Podman on RHEL 9 & Ubuntu 24.04	50
10.3 OCI Sandboxing, cgroups v2 Quotas & Linux Capability Dropping	51

10.4 Immutable & Read-Only Root Filesystems (<code>--read-only</code>)	51
10.5 Secure Secret Management (Zero CLI Exposure)	51
Modern Secret Injection via Encrypted Secret Files	51
10.6 Image Vulnerability & Secret Scanning with Trivy	52
Installing and Running Trivy	52
10.7 Container Isolation Boundaries Architecture	52
10.8 Operational Deployment: Production-Grade Hardened Database Container	53
Chapter 11: Secure Storage Management, Backups & Hardware Telemetry	54
11.1 Physical Storage Security & Media Sanitization (NIST SP 800-88)	54
11.2 Drive Health Telemetry & Predictive Failure Monitoring (<code>smartctl</code>)	54
Testing and Monitoring Drives with <code>smartctl</code>	54
Configuring Automated <code>smartd</code> Daemon	55
11.3 Enterprise Cryptographic Backup Strategies (3-2-1 Architecture)	55
11.4 Automated Disaster Recovery Drills & Integrity Testing	55
Chapter 12: Secure Logging, Audit & Anomaly Detection	57
12.1 Modern Logging Architecture: <code>systemd-journald</code>	57
12.2 Hardening <code>journald</code> & Forward Secure Sealing (FSS)	57
Enabling Forward Secure Sealing (FSS)	57
12.3 Legacy Log File Management (<code>logrotate</code> Security)	58
12.4 Centralized SIEM Log Forwarding over Mutual TLS (mTLS)	58
Chapter 13: Operations & Security Maintenance Procedures	60
13.1 Enterprise Vulnerability & Patch Management Lifecycle	60
13.2 Automated Security Patching (<code>dnf-automatic</code> & <code>unattended-upgrades</code>)	60
RHEL 9 / CentOS Stream 9: <code>dnf-automatic</code>	60
Ubuntu 24.04 LTS: <code>unattended-upgrades</code>	61
13.3 Stale Library & Outdated Process Detection with <code>needrestart</code>	61
13.4 Enterprise Kernel Live-Patching (Kpatch & Livepatch)	62
13.5 Secure Decommissioning & Incident Response Readiness	62
Chapter 14: References, Standards & Technical Glossary	63
14.1 Deployment Hardening Matrix & Verification Checklist	63
14.2 Normative References & Authoritative Standards	64
14.3 Comprehensive Bilingual Technical Glossary (English / French)	65
14.4 LightPath Professional Services & Consulting Contact	68

Chapter 1: Introduction, Governance & Security Architecture

1.1 Scope, Purpose and Operational Objectives

The primary objective of this manual is to deliver an exhaustive, definitive, and operationally realistic blueprint for hardening Linux production servers. Operating system security is frequently misunderstood as a purely reactive activity—applying patches when CVEs are broadcast or enabling a default firewall. In contrast, genuine production security requires establishing an uncompromising, defensive baseline that structurally limits what any process, user, or remote attacker can execute.

This document targets infrastructure architects, platform engineers, site reliability engineers (SREs), and Linux system administrators responsible for operating systems hosting critical data, applications, and network services. Whether deploying virtual machines on VMware vSphere, launching bare-metal servers in sovereign data centers, or provisioning enterprise instances across AWS, Microsoft Azure, or Google Cloud Platform, the hardening controls outlined herein apply directly.

Operational objectives of this manual include:

- **Minimizing the Attack Surface:** Disabling unneeded packages, services, kernel modules, and network listeners to leave the smallest possible target footprint.
- **Preventing Lateral Movement:** Ensuring that the compromise of an unprivileged daemon or container cannot be leveraged to pivot through the internal network or access host storage.
- **Enforcing Deterministic Access Control:** Restricting superuser delegation via cryptographically audited, multi-factor authenticated channels.
- **Guaranteeing Forensic Immutability:** Implementing tamper-evident, cryptographically sealed logs and kernel audit trails capable of withstanding local compromise.
- **Ensuring Regulatory & Benchmark Compliance:** Satisfying the highest criteria of ANSSI (Agence nationale de la sécurité des systèmes d'information), CIS Benchmarks (Level 1 and Level 2 profiles), NIST SP 800-123, ISO/IEC 27001, and regulatory frameworks including the European Union's NIS2 and DORA directives.

1.2 Target Distributions: RHEL 9 & Ubuntu 24.04 LTS

To eliminate theoretical abstractions, this manual focuses on the two enterprise Linux distributions that represent the overwhelming standard for production deployments:

- **Red Hat Enterprise Linux 9 (RHEL 9) / CentOS Stream 9** (and compatible derivatives such as Rocky Linux 9 and AlmaLinux 9): Representing the enterprise RPM ecosystem, governed by the **dnf** package manager, **systemd**, **firewalld**, and **SELinux** in mandatory Enforcing mode.
- **Ubuntu 24.04 LTS (Noble Numbat):** Representing the enterprise Debian/APT ecosystem, supported with long-term security maintenance through 2029 (and 2034 with Expanded Security Maintenance), governed by **apt**, **systemd**, **ufw** / **nftables**, and **AppArmor**.

Where administrative commands, configuration file paths, or service implementations differ between these two ecosystems, this manual systematically provides verified, dual-distribution syntax.

1.3 LightPath Corporate Presentation & Cardinal Pillars

About LightPath

LightPath is an elite technical consultancy and open-core software publisher founded in 2012 by **Jean-Marie Renouard**. Specializing in mission-critical database engineering, cloud-native platforms, and system hardening, LightPath partners with CTOs, platform engineering teams, and SRE organizations to achieve maximum resilience, military-grade security, and extreme performance.

Our engineering engagements are anchored on **three non-negotiable cardinal pillars**:

- **Security & Rigorous Compliance:** Surgical operating system hardening, automated alignment with international standards (**CIS Benchmarks**, **ANSSI**, **DISA STIG**), strict least-privilege access control (RBAC), container and process sandboxing (SELinux, AppArmor, namespaces, cgroups v2), and end-to-end cryptographic protection (TLS 1.3, LUKS2, TDE).
- **High Availability & Zero-SPOF Resilience (99.999% SLA):** Architecting fault-tolerant distributed topologies with zero single points of failure, multi-availability zone database clusters (Galera, streaming PostgreSQL/MySQL replication, Raft/Paxos consensus), automated transparent sub-2-second failover, and battle-tested disaster recovery drills (DRP/BCP).
- **Extreme Performance & Deep Engine Tuning:** In-depth optimization conducted directly at the Linux kernel and database storage engine level (SQL profiling, Buffer Pool memory sizing, deadlock eradication, and composite indexing), guaranteeing sub-millisecond latencies under severe transactional load.

Core Capabilities

- **Mission-Critical Database Engineering:** Deep tuning and administration for MySQL 8.4 LTS, PostgreSQL 16/17, MariaDB 11.4, Apache Cassandra, and MongoDB.
- **Platform Engineering & GitOps DBaaS:** Kubernetes database industrialization via cloud-native operators (CloudNativePG), self-service developer DBaaS platforms, Infrastructure as Code (Terraform, Ansible), and drift-free GitOps pipelines.
- **Hardening & Automated Compliance:** Publishers of the **CIS Benchmarks Tools v2.9.1** open-source suite (18 CIS & DISA STIG benchmarks, 887 automated security controls in 100% zero-dependency Python PSL for air-gapped systems).
- **Deterministic System-1 AI Engine:** Creator of **MicroLLM (micro-jev)**, an ultra-fast CPU-native System-1 decision classifier (< 10 ms, Apache-2.0 license, RLCD² conformal calibration, Neyman-Pearson guardrails against prompt injection and destructive SQL, and MCP server compliance for EU AI Act and DORA).
- **Architecture Reviews & Migration:** Rapid diagnostic audits, vendor lock-in emancipation (migrating Oracle/SQL Server to open-source PostgreSQL), and advanced masterclasses.

Corporate Information & Inquiries:

- **Official Website:** <https://www.lightpath.fr>
- **Email:** contact@lightpath.fr
- **Phone:** +33 (0)6 62 69 58 81
- **Headquarters:** 30, rue du Bourg au Loup, 35140 Saint-Aubin-du-Cormier, France

1.4 About the Author — Jean-Marie Renouard

Jean-Marie Renouard is the founder and Chief Technical Officer of **LightPath**. As a seasoned Data Architect, DevOps Consultant, and deterministic AI systems engineer, he brings over 20 years of hands-on expertise securing, tuning, and scaling mission-critical IT backbones and massive database estates.

Throughout his career, Jean-Marie has operated on the frontline of tier-one enterprise environments across financial services, energy, telecommunications, and retail logistics (including **BNP Paribas / BP2I**, **EDF Datanumia**, **RENATER**, **Orange**, and **GALEC E.Leclerc**). He has led mission-critical security audits, built disaster-proof database infrastructures, and spearheaded enterprise cloud-native container migrations on Kubernetes.

Active leader and contributor within the international Open Source ecosystem:

- **Co-author and core maintainer of MySQLTuner-perl:** The industry-standard diagnostic and tuning script for MySQL and MariaDB, holding 8,400+ GitHub stars and executed daily by tens of thousands of system administrators and DBAs worldwide.
- **Creator of the CIS Benchmarks Tools v2.9.1 suite:** Automated multi-standard compliance framework covering 18 official CIS and DISA STIG benchmarks (887 automated security tests executing in zero-dependency, air-gapped environments).
- **Creator of MicroLLM (microllm-jev):** A CPU-native System-1 AI decision and guardrail engine (< 10 ms), featuring RLCD² conformal calibration (ECE 0.008), Neyman-Pearson statistical safety bounds (shielding production databases from rogue AI agent actions), and full EU AI Act / DORA audit readiness.

Professional Profiles & Author Links:

- **Personal Portal & Engineering Blog:** <https://www.jmrenouard.fr>
- **LinkedIn:** <https://www.linkedin.com/in/jmrenouard>
- **GitHub:** <https://github.com/jmrenouard>
- **X (Twitter):** @jmrenouard
- **LightPath:** <https://www.lightpath.fr>

1.5 Security Norms & Regulatory Frameworks (ANSSI, CIS, NIST, NIS2, DORA)

Enterprise systems do not exist in a vacuum; they must comply with legal, contractual, and regulatory mandates. The hardening guidelines in this manual directly operationalize the requirements of five major cybersecurity frameworks:

- **ANSSI (Agence nationale de la sécurité des systèmes d'information):** French national guidelines for GNU/Linux system security (*Recommandations de sécurité relatives à un système GNU/Linux*). ANSSI specifies four levels of hardening: Minimal, Intermediary, Enhanced, and High. This manual addresses the **Enhanced (Renforcé)** level.
- **CIS (Center for Internet Security) Benchmarks:** The global consensus standard for operating system defense. This manual maps to **CIS Red Hat Enterprise Linux 9 Benchmark** and **CIS Ubuntu Linux 24.04 LTS Benchmark** Level 1 (Server) and Level 2 (High Security / Defense).

- **NIST SP 800-123:** The National Institute of Standards and Technology **Guide to General Server Security**, enforcing operating system baseline integrity, patch management, least privilege, and auditable accountability.
- **DISA STIG (Defense Information Systems Agency Security Technical Implementation Guides):** The United States Department of Defense rigorous technical baseline for securing military and governmental Linux servers.
- **EU NIS2 Directive & DORA Regulation:** The European Union Network and Information Security Directive 2 (NIS2) and Digital Operational Resilience Act (DORA) mandate strict technical controls for critical entities and financial institutions, specifically requiring verifiable supply chain security, continuous auditability, rapid vulnerability remediation, and fault-tolerant infrastructure design.

1.6 The Linux Defense-in-Depth Architecture

Security cannot depend on any single perimeter wall. The core architectural philosophy of this manual is **Defense-in-Depth** structured across seven concentric layers. A failure or zero-day vulnerability in any single layer is contained, mitigated, and detected by the remaining layers.

The seven layers comprise:

- **Network & Perimeter Defense:** Dynamic filtering via **nftables**, stateful connection tracking, SYN flood mitigation, and dynamic brute-force banning via **fail2ban**.
- **Identity & Access Management:** Elimination of static shared passwords, mandatory Ed25519 cryptographic keys, hardware FIDO2 tokens, Time-based One-Time Password (TOTP) MFA, and granular **sudo** delegation.
- **Process Sandboxing & Mandatory Access Control:** Confining daemons within isolated namespaces, stripping unneeded Linux capabilities, and enforcing non-bypassable MAC policies (SELinux Enforcing or AppArmor).
- **Kernel & System Hardening:** Memory space randomization (ASLR), restricting kernel pointer leaks, hardening ptrace scopes, and blocking unprivileged eBPF execution via **/etc/sysctl.d/99-security-hardening.conf**.
- **Storage & Filesystem Integrity:** Cryptographic volume encryption at rest using LUKS2, hardened mount points with strict **noexec**, **nosuid**, and **nodev** flags, and restrictive umask policies.
- **Continuous Auditing & Detection:** Non-bypassable kernel syscall auditing via **auditd**, cryptographic file integrity monitoring (FIM) with AIDE, and automated compliance scoring with Lynis.
- **Hardware & Firmware Root of Trust:** Secure Boot verification, UEFI measured boot integrity, and hardware-backed cryptographic sealing via TPM 2.0.



Figure 1.1 — Linux Defense-in-Depth Architecture Layers

1.7 Typographical Conventions & Command Taxonomy

Throughout this manual, the following formatting conventions are strictly maintained:

- **Command Line Prompts:**
- Commands prefixed with **#** must be executed as **root** (or via **sudo**).
- Commands prefixed with **\$** are executed as an unprivileged, authenticated user.
- Where distinct commands are required for RHEL 9 and Ubuntu 24.04 LTS, explicit distribution tabs or section markers are provided.
- **Filenames and File Paths:** Rendered in inline monospace code (e.g., `/etc/ssh/sshd_config.d/99-security-hardening.conf`).
- **Configuration Blocks:** Full configuration files are presented in designated syntax blocks (`ini`, `bash`, `sshconfig`, `pam`, etc.).
- **Security Admonitions:**
- > **[!NOTE]**: Explains architectural context or behavioral subtleties.
- > **[!WARNING]**: Identifies operational risks, potential service interruptions, or actions that could lock administrators out of remote hosts if executed out of sequence.

Chapter 2: Installation & Secure Partitioning

2.1 Principles of Minimalist Installation ("Simple is Beautiful")

The foundation of system hardening begins during initial operating system provisioning. The most secure piece of code is the code that is never installed. Every additional package, compiler, library, or background daemon increases the attack surface, introduces potential CVE vulnerabilities, and complicates patching lifecycles.

When installing RHEL 9 or Ubuntu 24.04 LTS, administrators must select the **Minimal Install** base environment. Under no circumstances should development toolchains (such as **gcc**, **make**, **g++**, or **binutils**), graphical user interfaces (X11/Wayland, GNOME), or legacy diagnostic tools be present on a production server. Compilers on a production server provide attackers with the immediate capability to compile local kernel exploits or rootkits following an initial remote code execution breach.

Removing Development Tools and Unneeded Package Groups

On RHEL 9 / CentOS Stream 9:

```
# Verify installed package groups
sudo dnf group list --installed

# Remove development and unneeded server environments if mistakenly installed
sudo dnf groupremove -y "Development Tools" "Server with GUI"

# Remove legacy diagnostic tools and unneeded network packages
sudo dnf remove -y gcc make g++ telnet rsh rsync-daemon xinetd tftp ftp ypserve
```

On Ubuntu 24.04 LTS:

```
# Purge build toolchains and obsolete packages
sudo apt purge -y build-essential gcc make g++ telnet rsh-client rsh-server tftp ftp yp-tools
sudo apt autoremove --purge -y
```

2.2 Enterprise Partitioning Scheme & Mount Point Isolation

A monolithic filesystem—where the entire operating system, application data, temporary files, and log files reside on a single `/` root partition—represents an intolerable operational and security risk. A run-away log file, an unprivileged user writing to `/tmp`, or a denial-of-service attack targeting database tables can exhaust disk space, causing kernel panics and taking down all co-hosted services.

Furthermore, filesystem mount options (**noexec**, **nosuid**, **nodev**) cannot be applied selectively unless sensitive directories reside on independent logical volumes or dedicated partitions.

An enterprise storage layout must allocate isolated Logical Volume Management (LVM) volumes for:

- **/** (Root): Operating system binaries and core libraries. Mounted read-write (or read-only where architecture allows).
- **/boot**: Kernel images and initramfs files. Must reside on an independent partition outside LVM for bootloader compatibility.
- **/boot/efi**: UEFI system partition.
- **/home**: User home directories. Strictly isolated to prevent user storage exhaustion from impacting system services.
- **/var**: Variable runtime data.
- **/var/log**: System and application logs. Isolates log ingestion to prevent log flooding from exhausting system storage.
- **/var/log/audit**: Dedicated volume for **auditd**. Crucial: if the audit volume fills up, the kernel can be configured to halt the system to prevent untracked activity.
- **/var/tmp**: Persistent temporary files. Must mirror **/tmp** security restrictions.
- **/tmp**: Ephemeral temporary storage. Handled via **tmpfs** (RAM-backed) with strict execution boundaries.

2.3 Hardened Mount Options (``noexec``, ``nosuid``, ``nodev``, ``tmpfs``)

The Linux kernel provides filesystem mount flags that neutralize entire categories of local privilege escalation attacks:

- **nodev**: Prevents the creation or interpretation of character or block special device files. Attackers frequently attempt to create device nodes (e.g., using **mknod**) in **/tmp** or **/home** to access raw storage devices directly. **nodev** renders these attempts inert.
- **nosuid**: Disables the execution of Set-User-Identifier (SUID) and Set-Group-Identifier (SGID) bits. Attackers exploit SUID binaries (or copy existing SUID binaries into writable directories) to escalate privileges to **root**. **nosuid** prevents any binary executed from that mount point from gaining elevated privileges.
- **noexec**: Disables binary execution entirely across the partition. In writable directories like **/tmp**, **/var/tmp**, and **/dev/shm**, attackers routinely download executable payloads, scripts, or compiled exploits. With **noexec**, the kernel returns **EACCES** (Permission denied) on any attempt to execute binaries from these mount points.

2.4 Complete Production ``/etc/fstab`` Reference

The following configuration represents a hardened, production-ready **/etc/fstab** file incorporating LVM volumes, secure mount options, and hardened virtual memory partitions:

```
# /etc/fstab - Hardened Enterprise Baseline (RHEL 9 / Ubuntu 24.04 LTS)
# <file system>          <mount point>  <type>  <options>
<dump>  <pass>

# Core Root Filesystem
/dev/mapper/vg_system-lv_root  /                ext4    defaults,nodev
0                               1

# Dedicated Boot Partitions (Must allow SUID/Exec for kernel modules, but restrict devices)
UUID=1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d /boot ext4    defaults,nodev,nosuid
0                               2
UUID=ABCD-1234                      /boot/efi       vfat     defaults,nodev,nosuid,umask=0077
```

```

0      2

# User Home Directories: Isolated, No Device Nodes, No SUID
/dev/mapper/vg_system-lv_home /home          ext4    defaults,nodev,nosuid
0      2

# Variable System Data: Isolated Mounts
/dev/mapper/vg_system-lv_var /var          ext4    defaults,nodev
0      2
/dev/mapper/vg_system-lv_var_log /var/log      ext4    defaults,nodev,nosuid,noexec
0      2
/dev/mapper/vg_system-lv_var_audit /var/log/audit ext4    defaults,nodev,nosuid,noexec
0      2
/dev/mapper/vg_system-lv_var_tmp /var/tmp       ext4    defaults,nodev,nosuid,noexec
0      2

# Dedicated Application / Database Data Volume (Example)
/dev/mapper/vg_data-lv_appdata /data          ext4    defaults,nodev,nosuid
0      2

# Ephemeral Temporary Storage (RAM-backed tmpfs)
tmpfs /tmp tmpfs
defaults,nodev,nosuid,noexec,mode=1777,size=2G 0 0
tmpfs /dev/shm tmpfs
defaults,nodev,nosuid,noexec,mode=1777,size=1G 0 0

```

⚠ SECURITY WARNING & PRODUCTION RISK

Setting **noexec** on **/tmp** and **/var/tmp** can break poorly written administrative scripts or package managers (such as certain APT or DNF transaction wrappers) that attempt to compile or execute binaries in **/tmp**. In such cases, configure package managers to use alternative execution directories (e.g., **APT::ExtractTemplates::TempDir "/var/cache/apt/tmp";**) rather than weakening production mount protections.

2.5 Full Disk Encryption at Rest with LUKS2 & Argon2id

In modern threat models, physical theft of server hardware, unauthorized extraction of NVMe drives, or illicit copying of virtual machine disk images (VMDK/QCOW2) at the hypervisor or storage-area network (SAN) level represent severe security vulnerabilities.

LUKS2 (Linux Unified Key Setup version 2) provides transparent, kernel-level cryptographic volume encryption via **dm-crypt**. Unlike legacy LUKS1, LUKS2 incorporates the memory-hard **Argon2id** password-based key derivation function (PBKDF), rendering GPU-accelerated and ASIC brute-force attacks against passphrase-protected keys mathematically infeasible.

Formatting and Provisioning a LUKS2 Encrypted Volume

To encrypt a dedicated data volume or secondary drive (**/dev/sdb1**):

```
# 1. Format the partition with LUKS2, AES-256-XTS cipher, SHA-512 hashing, and Argon2id PBKDF
```

```

sudo cryptsetup luksFormat \
  --type luks2 \
  --cipher aes-xts-plain64 \
  --key-size 512 \
  --hash sha512 \
  --pbkdf argon2id \
  --pbkdf-memory 1048576 \
  --pbkdf-parallel 4 \
  --iter-time 3000 \
  --verify-passphrase \
  /dev/sdb1

# 2. Open the encrypted container
sudo cryptsetup open /dev/sdb1 secure_data_vol

# 3. Format the decrypted block device with ext4 or xfs
sudo mkfs.ext4 -L SECURE_DATA /dev/mapper/secure_data_vol

# 4. Create mount point and test mounting
sudo mkdir -p /data/secure
sudo mount /dev/mapper/secure_data_vol /data/secure

```

Automated Unlocking in Enterprise Environments (Clevis & Tang / TPM 2.0)

In headless data center or cloud environments, manual passphrase entry upon reboot is operationally impractical. LUKS2 integrates seamlessly with **Network-Bound Disk Encryption (NBDE)** using **clevis** and **tang**, or hardware-bound unlocking via **TPM 2.0** chips:

```

# Bind LUKS2 volume to the local hardware TPM 2.0 PCR registers (PCR 0 + 7 verify firmware &
Secure Boot)
sudo dnf install -y clevis clevis-tpm2 clevis-dracut # On RHEL 9
# sudo apt install -y clevis clevis-tpm2 clevis-initramfs # On Ubuntu 24.04

sudo clevis luks bind -d /dev/sdb1 tpm2 '{"pcr_ids":["0,7"]}'

```

If the server hardware is altered, firmware tampered with, or the drive moved to a different chassis, the TPM 2.0 chip refuses to unseal the decryption key, keeping all data fully encrypted.

2.6 Volatile Memory Filesystems (`tmpfs`) Hardening

Linux mounts **/dev/shm** (shared memory) and often **/tmp** as volatile **tmpfs** filesystems. Because **/dev/shm** is world-writable by design to accommodate inter-process communication (IPC), malicious actors frequently use it as an unmonitored drop point for exploits.

To prevent memory exhaustion and execution vectors:

- Ensure **/dev/shm** and **/tmp** are explicitly sized (e.g., **size=1G** or **size=2G**). An unbounded **tmpfs** can consume 50% or more of available physical RAM, exposing the system to denial-of-service via Out-Of-Memory (OOM) kernel kills.

- Ensure **nodev**, **nosuid**, and **noexec** options are permanently configured in **/etc/fstab** as shown in Section 2.4.

Chapter 3: Kernel Hardening & System Parameters

3.1 Kernel Self-Protection Architecture (KSPP & ASLR)

The Linux kernel is the ultimate arbiter of system security. If an attacker achieves kernel-level code execution, all userspace security boundaries—including **sudo**, SELinux, and systemd sandboxing—are rendered moot. Modern Linux kernels incorporate the findings of the **Kernel Self-Protection Project (KSPP)**, enabling hardware-assisted defensive mechanisms.

Key architectural protections include:

- **Address Space Layout Randomization (ASLR)**: Randomizes the memory addresses of program code, stack, heap, and shared libraries. Setting `kernel.randomize_va_space = 2` enables complete randomization, neutralizing buffer overflow and Return-Oriented Programming (ROP) exploitation techniques.
- **Kernel Pointer Restriction (``kptr_restrict``)**: By default, the kernel exposes kernel memory addresses via `/proc/kallsyms` and `printf("%pK")`. These addresses provide attackers with the exact memory layout required to construct exploit payloads. Setting `kernel.kptr_restrict = 2` completely redacts kernel pointers for all users, including `root`.
- **Dmesg Buffer Restriction (``dmesg_restrict``)**: The kernel ring buffer (`dmesg`) frequently outputs memory addresses, hardware configurations, and panic tracebacks that can assist an adversary in weaponizing a vulnerability. Setting `kernel.dmesg_restrict = 1` restricts `dmesg` access strictly to users with the `CAP_SYSLOG` capability.

3.2 Network Stack Protections (SYN Cookies, Anti-Spoofing & ICMP Restrictions)

The TCP/IP stack in Linux must be hardened against both external denial-of-service attacks and network-level eavesdropping:

- **TCP SYN Cookies (``net.ipv4.tcp_syncookies = 1``)**: When an attacker floods a server with TCP SYN packets without completing the three-way handshake, the kernel's SYN backlog queue is rapidly exhausted. SYN cookies replace state allocation with a cryptographic hash encoded into the sequence number, allowing the server to handle incoming requests without exhausting memory.
- **Reverse Path Filtering (``rp_filter = 1``)**: Enforces strict RFC 3704 source address validation. The kernel drops any incoming packet whose source IP address is not routable via the network interface on which it was received, neutralizing IP spoofing attacks.
- **Disabling ICMP Redirects (``accept_redirects = 0``, ``send_redirects = 0``)**: ICMP redirects are used by routers to notify hosts of more optimal routing paths. Attackers on a local network segment can abuse ICMP redirects to execute Man-in-the-Middle (MitM) attacks by redirecting host traffic through an adversarial node.

3.3 Debugging & Memory Restrictions (``ptrace``, ``dmesg``, ``kptr``)

The `ptrace` system call allows one process to inspect and manipulate the memory and registers of another process. While essential for debuggers like `gdb`, `ptrace` is widely abused by malicious actors to inject shellcode into running processes (such as SSH daemons or web servers) or extract plaintext credentials from process memory.

Linux provides the **Yama** Linux Security Module (LSM) to govern ptrace scopes:

- **kernel.yama.ptrace_scope = 1**: Restricted ptrace. A process can only ptrace its own child processes that have explicitly called **prctl(PR_SET_PTRACER)**.
- **kernel.yama.ptrace_scope = 2**: Administrative-only ptrace. Only processes with **CAP_SYS_PTRACE** can ptrace other processes. Recommended for production servers.
- **kernel.yama.ptrace_scope = 3**: No ptrace. No process can use **ptrace**. Useful for ultra-high security appliances.

3.4 Filesystem Link Exploitation Protections

Historically, attackers leveraged symlinks and hardlinks in world-writable directories (like **/tmp**) to trick privileged programs into overwriting or appending data to sensitive files (such as **/etc/shadow** or **/etc/sudoers**).

Modern Linux kernels provide dedicated filesystem link protections:

- **fs.protected_hardlinks = 1**: Forbids creating hardlinks to files that the user does not own, or does not have write access to.
- **fs.protected_symlinks = 1**: Forbids following symlinks in sticky world-writable directories unless the symlink's owner matches the follower or the directory owner.
- **fs.protected_fifos = 2** & **fs.protected_regular = 2**: Restricts how FIFOs and regular files are opened in sticky directories, eliminating Time-of-Check to Time-of-Use (TOCTOU) race condition vulnerabilities.

3.5 eBPF & Kernel Module Lockdown

Extended Berkeley Packet Filter (eBPF) allows executing sandboxed bytecode directly inside the Linux kernel. While revolutionary for performance profiling and network filtering, unprivileged eBPF introduces a vast attack surface, including kernel memory disclosure and Spectre-class speculative execution side-channels.

- **kernel.unprivileged_bpf_disabled = 1**: Strictly forbids unprivileged users from loading eBPF programs.
- **net.core.bpf_jit_harden = 2**: Hardens the eBPF Just-In-Time (JIT) compiler against JIT-spraying attacks by enabling constant blinding.
- **kernel.sysrq = 0**: Disables the magic SysRq key sequence, preventing local physical consoles from executing raw memory dumps or bypassing root authentication.

3.6 Production Configuration: `/etc/sysctl.d/99-security-hardening.conf`

Create the consolidated kernel hardening file **/etc/sysctl.d/99-security-hardening.conf**:

```
# /etc/sysctl.d/99-security-hardening.conf
# LightPath Enterprise Hardening Baseline (Version 2.0 - 2026)
# Fully compliant with ANSSI Enhanced & CIS Benchmark Level 2

# =====
# 1. KERNEL SELF-PROTECTION & MEMORY INTEGRITY
# =====
kernel.randomize_va_space = 2
kernel.kptr_restrict = 2
```

```

kernel.dmesg_restrict = 1
kernel.unprivileged_bpf_disabled = 1
net.core.bpf_jit_harden = 2
kernel.yama.ptrace_scope = 2
kernel.sysrq = 0
kernel.core_uses_pid = 1
fs.suid_dumpable = 0

# =====
# 2. FILESYSTEM LINK & RACE-CONDITION DEFENSE
# =====
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
fs.protected_fifos = 2
fs.protected_regular = 2

# =====
# 3. NETWORK STACK: TCP SYN FLOOD & DENIAL OF SERVICE MITIGATION
# =====
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_rfc1337 = 1
net.ipv4.tcp_fin_timeout = 15

# =====
# 4. NETWORK STACK: ANTI-SPOOFING & SOURCE ROUTING REJECTION
# =====
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
net.ipv6.conf.all.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0

# =====
# 5. NETWORK STACK: ICMP REDIRECT & BROADCAST DEFENSE
# =====
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
net.ipv4.conf.all.secure_redirects = 0
net.ipv4.conf.default.secure_redirects = 0
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.icmp_ignore_bogus_error_responses = 1

# =====
# 6. NETWORK STACK: IP FORWARDING RESTRICTION
# =====
# Ensure server does not act as a transit router unless specifically configured
net.ipv4.ip_forward = 0
net.ipv6.conf.all.forwarding = 0

```

3.7 Verification and Runtime Enforcement

Apply the new kernel parameters dynamically and verify that no syntax errors exist:

```
# Reload all sysctl configuration files across /etc/sysctl.d/  
sudo sysctl --system  
  
# Verify active parameters in memory  
sudo sysctl kernel.randomize_va_space kernel.kptr_restrict net.ipv4.tcp_syncookies
```

Output should confirm:

```
kernel.randomize_va_space = 2  
kernel.kptr_restrict = 2  
net.ipv4.tcp_syncookies = 1
```

Chapter 4: Access Control, Privileges & Identity Management

4.1 Principle of Least Privilege & Service Account Architecture

The principle of least privilege mandates that every user, administrator, and system service must operate with the minimum set of permissions necessary to perform their legitimate function. On an enterprise Linux system, day-to-day administration must never occur directly as the superuser **root**.

System services (such as web servers, database daemons, and background workers) must execute under dedicated system accounts without shell login capabilities (**/sbin/nologin** or **/usr/sbin/nologin**). Furthermore, system accounts should have unique User Identifiers (UIDs) below 1000 and should never be granted **sudo** permissions.

4.2 Superuser (Root) Account Locking

Direct interactive logins as **root**—whether via physical console, serial port, or SSH—must be permanently locked. When administrators log in as **root**, all actions are executed without individual attribution in audit trails. By forcing administrators to authenticate as their individual named accounts and subsequently elevate via **sudo**, every privileged command is irreversibly bound to a specific identity in the audit logs.

Locking the Root Account Correctly

Never manually edit **/etc/passwd** or **/etc/shadow** with text editors like **vi** or **nano** to disable accounts, as syntax errors can break the authentication subsystem.

Execute the following commands:

```
# 1. Lock the root password (prepends '!' to password hash in /etc/shadow)
sudo passwd -l root

# 2. Assign a non-interactive shell to the root account
sudo usermod -s /sbin/nologin root # On RHEL 9
# sudo usermod -s /usr/sbin/nologin root # On Ubuntu 24.04
```

TECHNICAL ENGINEERING NOTE

Locking the root password disables password-based authentication for **root**. It does not prevent authorized administrators from executing administrative commands via **sudo** or accessing root shells via **sudo -i**.

4.3 Hardened Sudo Configuration & Administrative Delegation

Historically, organizations utilized dangerous directives in **/etc/sudoers**, such as **ALL=(ALL) NOPASSWD: ALL**. This practice completely eliminates the re-authentication barrier: if an administrator leaves their

terminal unattended or if an unprivileged shell session is hijacked, the attacker achieves instantaneous, unrestricted root privileges without prompting.

Modern Sudo Architecture: Drop-in Configuration

All **sudo** configurations must reside in **/etc/sudoers.d/** rather than modifying the core **/etc/sudoers** file, ensuring seamless package upgrades.

Create **/etc/sudoers.d/99-security-admin**:

```
# /etc/sudoers.d/99-security-admin - LightPath Enterprise Sudo Baseline
# Enforces re-authentication, strict logging, and environment sanitization

# 1. Environment Sanitization & Security Defaults
Defaults env_reset
Defaults timestamp_timeout = 5
Defaults passwd_tries = 3
Defaults secure_path = "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# 2. Comprehensive Sudo Logging
Defaults logfile = "/var/log/sudo.log"
Defaults log_year, log_host
Defaults !mail_always

# 3. Administrative Delegation
# RHEL 9 uses the 'wheel' administrative group; Ubuntu 24.04 uses the 'sudo' group.
# Re-authentication with the user's own password is strictly required.
%wheel ALL=(ALL:ALL) ALL
%sudo ALL=(ALL:ALL) ALL
```

Set strict permissions:

```
sudo chmod 0440 /etc/sudoers.d/99-security-admin
sudo visudo -cf /etc/sudoers.d/99-security-admin
```

4.4 Pluggable Authentication Modules (PAM) Hardening

Pluggable Authentication Modules (PAM) govern authentication, account authorization, session setup, and password management across all Linux services.

PAM configuration files are split into distinct stacks:

- **auth**: Verifies user identity (e.g., prompting for passwords, TOTP tokens, hardware keys).
- **account**: Determines whether an account is valid, active, and authorized to log in.
- **password**: Enforces password update policies, complexity, and hashing.
- **session**: Configures environment variables, mounts filesystems, and applies resource limits.

4.5 Account Lockout Policies (`pam\_faillock`)

To protect local consoles and services from automated brute-force password guessing, the system must enforce automatic account lockouts after consecutive authentication failures.

In both RHEL 9 and Ubuntu 24.04 LTS, **`pam\_faillock`** replaces the obsolete **pam_tally2**.

Configuring `pam\_faillock` on RHEL 9

RHEL 9 utilizes **authselect** to manage PAM configurations safely:

```
# Enable faillock within the active authselect profile
sudo authselect enable-feature with-faillock
sudo authselect apply-changes
```

Configure parameters in **/etc/security/faillock.conf**:

```
# /etc/security/faillock.conf
# Lock account after 3 consecutive failures within 15 minutes for a duration of 30 minutes
deny = 3
fail_interval = 900
unlock_time = 1800
even_deny_root
root_unlock_time = 900
```

Managing Locked Accounts

```
# Inspect failed authentication attempts for an operator
sudo faillock --user secops

# Manually reset and unlock an account
sudo faillock --user secops --reset
```

4.6 Password Complexity & Quality Enforcement (`pam\_pwquality`)

For environments that require password authentication, passwords must adhere to strict complexity requirements enforced by **`pam\_pwquality`**.

Configure **/etc/security/pwquality.conf**:

```
# /etc/security/pwquality.conf
# Minimum password length: 16 characters
minlen = 16

# Minimum character classes required (uppercase, lowercase, digits, symbols)
minclass = 4

# Credit thresholds for character types
```

```

dcredit = -1
ucredit = -1
lcredit = -1
ocredit = -1

# Maximum allowable consecutive identical characters
maxrepeat = 2

# Maximum sequence of consecutive monotonic characters (e.g. '1234', 'abcd')
maxsequence = 3

# Check against user's GECOS username and reverse strings
gecoscheck = 1

# Dictionary checking against cracklib
dictcheck = 1

```

4.7 CLI Password Generation & Password Managers (KeePassXC, Bitwarden)

Administrators must never rely on online websites or browser extensions to generate privileged credentials. Instead, use cryptographically secure command-line generation tools:

```

# Method 1: Using OpenSSL (Cryptographically secure 32-character base64 string)
openssl rand -base64 32

# Method 2: Using /dev/urandom with tr (High-entropy alphanumeric & special characters)
LC_ALL=C tr -dc 'A-Za-z0-9!@#$$%^&*()_+' < /dev/urandom | head -c 24; echo

# Method 3: Using pwgen (if installed)
pwgen -s -y -c -n 24 1

```

For credential storage, enterprise teams must standardize on encrypted, zero-knowledge password vaults:

- **KeePassXC:** Open-source, offline, file-backed vault (**.kdbx4**) utilizing Argon2id KDF and ChaCha20/AES-256 ciphers. Ideal for air-gapped infrastructure.
- **Bitwarden / Vaultwarden:** Enterprise, self-hosted, audited credential management supporting RBAC, directory synchronization, and hardware FIDO2 authentication.

Chapter 5: OpenSSH Hardening & Multi-Factor Authentication (MFA)

5.1 Threat Landscape & Modern Cryptographic Algorithms

OpenSSH represents the primary remote administration interface on enterprise Linux hosts and, consequently, the most heavily targeted service on the Internet. Compromised or weakly configured SSH daemons lead directly to unauthorized shell access, ransomware deployment, and credential exfiltration.

Modern OpenSSH hardening requires:

- **Unconditionally Disabling Password Authentication:** All interactive passwords must be disabled. Authentication must rely solely on modern asymmetric cryptography.
- **Eliminating Obsolete Algorithms:** Weak key exchange mechanisms (Diffie-Hellman Group 1), broken ciphers (3DES, Blowfish, CBC-mode AES), and compromised message authentication codes (MD5, 96-bit HMAC, SHA-1) must be strictly purged from configuration profiles.
- **Standardizing on Ed25519:** Ed25519 (Edwards-curve Digital Signature Algorithm over Curve25519) provides 128-bit security, immune to timing side-channel attacks, with compact 68-character public keys and high-speed cryptographic verification.

5.2 Production Configuration: `/etc/ssh/sshd_config.d/99-security-hardening.conf`

Create `/etc/ssh/sshd_config.d/99-security-hardening.conf`:

```
# /etc/ssh/sshd_config.d/99-security-hardening.conf
# LightPath Enterprise OpenSSH Baseline (Version 2.0 - 2026)
# Fully compliant with ANSSI High & CIS Benchmark Level 2

# =====
# 1. NETWORK & PROTOCOL FOUNDATIONS
# =====
Port 22
AddressFamily inet
Protocol 2

# =====
# 2. ACCESS RESTRICTIONS & IDENTITY
# =====
PermitRootLogin no
MaxAuthTries 3
MaxSessions 2
LoginGraceTime 30
StrictModes yes

# Session Keepalive & Automated Idle Disconnect
ClientAliveInterval 300
ClientAliveCountMax 2

# =====
```

```
# 3. AUTHENTICATION MECHANISMS
# =====
PubkeyAuthentication yes
PasswordAuthentication no
PermitEmptyPasswords no
HostbasedAuthentication no
IgnoreRhosts yes

# Disable Vulnerable Subsystems
X11Forwarding no
AllowAgentForwarding no
AllowTcpForwarding no
PermitUserEnvironment no

# =====
# 4. MODERN CRYPTOGRAPHIC CIPHERS & ALGORITHMS (ANSSI / NIST)
# =====
KexAlgorithms curve25519-sha256,curve25519-sha256@libssh.org,diffie-hellman-group16-
sha512,diffie-hellman-group18-sha512

Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com

MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com

HostKeyAlgorithms ssh-ed25519,ssh-ed25519-cert-v01@openssh.com,rsa-sha2-512,rsa-sha2-256

# =====
# 5. MULTI-FACTOR AUTHENTICATION (MFA) INTEGRATION
# =====
# Require Public Key FIRST, followed by Keyboard-Interactive PAM (TOTP)
AuthenticationMethods publickey,keyboard-interactive:pam
KbdInteractiveAuthentication yes
UsePAM yes
```

Test configuration syntax before restarting:

```
sudo sshd -t
sudo systemctl restart sshd # On RHEL 9
# sudo systemctl restart ssh # On Ubuntu 24.04
```

5.3 Client Key Generation Best Practices (Ed25519)

Clients connecting to production servers must generate dedicated Ed25519 keys with high key derivation iterations:

```
# Generate high-security Ed25519 key pair with 100 Argon2/KDF memory rounds
ssh-keygen -t ed25519 -a 100 -C "operator-secops@lightpath.fr" -f ~/.ssh/id_ed25519_prod
```

Set strict permissions:

```
chmod 0700 ~/.ssh
chmod 0600 ~/.ssh/id_ed25519_prod
chmod 0644 ~/.ssh/id_ed25519_prod.pub
```

5.4 Hardware Key Authentication via FIDO2/U2F (`ed25519-sk`)

For ultimate access security, enterprise administrators must authenticate using hardware security tokens (such as YubiKey 5 Series or SoloKeys) via FIDO2/U2F. With hardware keys, the private key material is physically generated inside the cryptographic secure element and can never be exported or stolen by malware on the administrator's workstation.

```
# Generate a hardware-backed resident FIDO2 key
ssh-keygen -t ed25519-sk -O resident -O verify-required -C "yubikey-secops@lightpath.fr"
```

When logging in with **ed25519-sk**, OpenSSH requires the administrator to enter their hardware token PIN and physically touch the sensor, guaranteeing physical presence.

5.5 Multi-Factor Authentication (MFA) with PAM & Google Authenticator TOTP

Where physical hardware tokens cannot be deployed to all operators, Time-based One-Time Password (TOTP) MFA via RFC 6238 provides a robust second factor.

Installing the PAM Module

On RHEL 9 / CentOS Stream 9:

```
sudo dnf install -y epel-release
sudo dnf install -y google-authenticator
```

On Ubuntu 24.04 LTS:

```
sudo apt update && sudo apt install -y libpam-google-authenticator
```

Initializing TOTP for an Operator

The operator executes the setup wizard under their unprivileged account:

```
google-authenticator -t -d -f -r 3 -R 30 -W -s ~/.google_authenticator
```

Flags explained:

- **-t**: Time-based tokens (TOTP).

- **-d**: Disallow token reuse (prevents replay attacks).
- **-f**: Force writing configuration to `~/.google_authenticator`.
- **-r 3 -R 30**: Rate limit logins to 3 attempts every 30 seconds.
- **-W**: Window size of 3 intervals (handles clock skew up to ± 30 seconds).

The command displays a QR code to be scanned with a mobile authenticator app (such as Aegis, FreeOTP, or Google Authenticator) and outputs emergency one-time scratch codes.

Configuring PAM for SSH MFA

Edit `/etc/pam.d/sshd`. Comment out `@include common-auth` (Ubuntu) or `auth substack password-auth` (RHEL) and add:

```
# /etc/pam.d/sshd - MFA Configuration
auth required pam_google_authenticator.so nullok
```

TECHNICAL ENGINEERING NOTE

The **nullok** directive allows users who have not yet configured a TOTP token to authenticate with their public key alone during initial onboarding. Once all users are initialized, remove **nullok** to mandate MFA universally.

5.6 Two-Phase SSH Access Flow Architecture

The following sequence diagram details the end-to-end cryptographic and multi-factor validation enforced during every SSH connection:

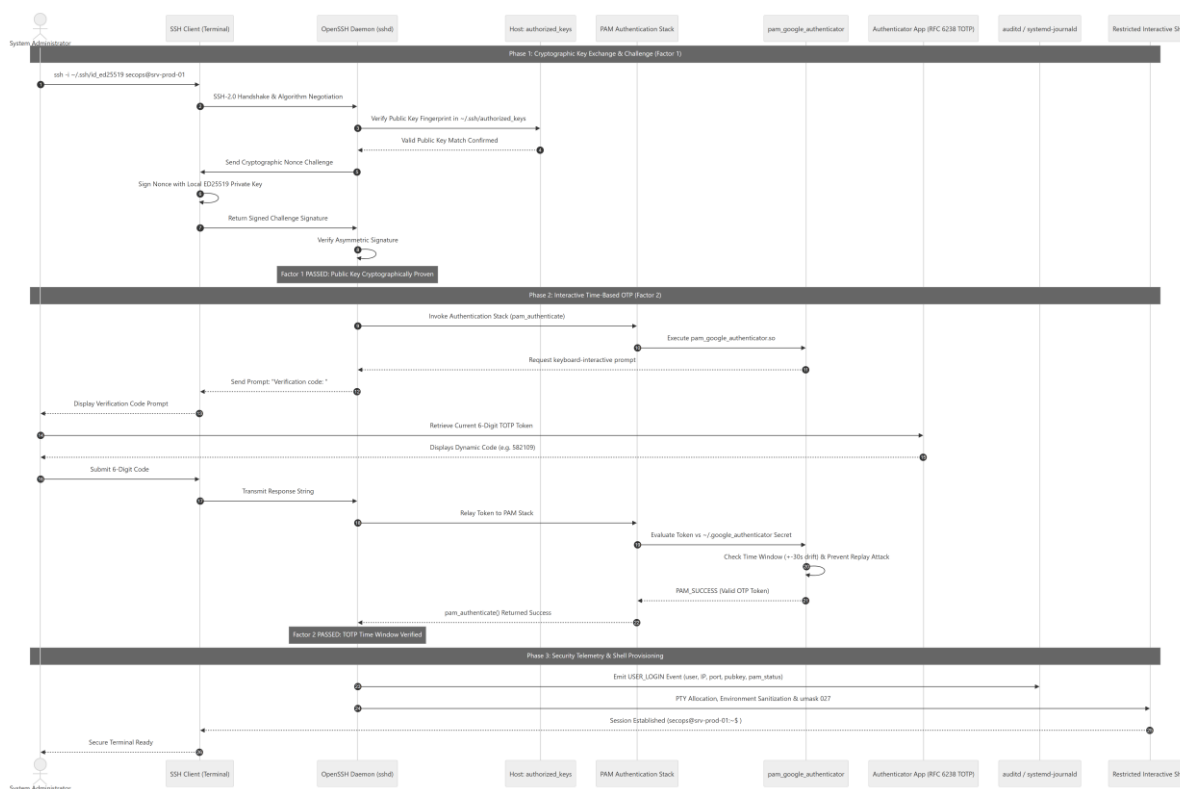


Figure 5.1 — OpenSSH Multi-Factor Authentication (MFA) Protocol Sequence

5.7 Verification, Banner Hygiene & Session Limits

Banner Hygiene

Displaying distribution names, kernel versions, or OpenSSH release strings in login banners provides adversaries with reconnaissance data. Create a legal warning banner in `/etc/issue.net`:

```

*****
* AUTHORIZED ACCESS ONLY. ALL SESSIONS ARE MONITORED AND AUDITED.          *
* UNAUTHORIZED ATTEMPTS WILL BE PROSECUTED UNDER APPLICABLE LAWS.          *
*****

```

Configure in `sshd_config.d/99-security-hardening.conf`:

```

Banner /etc/issue.net
PrintMotd no

```

Chapter 6: systemd Security & Application Sandboxing

6.1 The systemd Security Model & Attack Surface Assessment

On modern Linux systems, **systemd** is PID 1—the root process that boots the operating system, supervises all background services, and manages control groups (cgroups). Historically, services executed with the full privileges of the user that started them (frequently **root**), possessing unrestricted access to the entire filesystem, raw network sockets, and all kernel system calls.

Modern **systemd** incorporates built-in process isolation, Linux namespaces, capability bounds, and Seccomp-BPF filters. By defining sandboxing directives in unit service files, administrators can place application daemons (such as NGINX, MySQL, Redis, or proprietary microservices) into impenetrable, micro-segmented execution boundaries. Even if a remote attacker achieves code execution inside the service, they cannot modify system binaries, inspect other processes, or access unauthorized files.

6.2 Service Exposure Profiling with `systemd-analyze security`

systemd provides an automated security assessment tool: **systemd-analyze security**. It audits any active or inactive unit against dozens of hardening directives, computing an overall exposure score from **0.0** (optimal security) to **10.0** (severe risk / completely unconfined).

```
# Analyze security exposure score of the OpenSSH daemon
sudo systemd-analyze security sshd.service

# List exposure scores across all active services
sudo systemd-analyze security
```

Example audit output:

UNIT	EXPOSURE	PREDICATE	HAPPY
auditd.service	1.4	OK	:)
chrony.service	2.2	OK	:)
sshd.service	5.8	MEDIUM	:
unhardened-app.service	9.6	UNSAFE	:(

By adding sandboxing directives, services scoring **9.6** can be hardened to sub-**2.0** exposure ratings.

6.3 Sandboxing Directives & Resource Confinement

The primary **systemd** sandboxing directives include:

- **ProtectSystem=strict**: Mounts the entire filesystem hierarchy (**/usr**, **/boot**, **/etc**, and **/lib**) strictly read-only for the service. Only directories explicitly declared in **ReadWritePaths=** remain writable.
- **ProtectHome=yes**: Renders **/home**, **/root**, and **/run/user** completely invisible and inaccessible to the service (mounted as empty directories).

- **PrivateTmp=yes**: Mounts a dedicated, isolated `/tmp` and `/var/tmp` namespace for the service. Other processes (and attackers) cannot see or tamper with the service's temporary files.
- **PrivateDevices=yes**: Renders physical device nodes (e.g., `/dev/sda`, `/dev/nvme0n1`, `/dev/mem`) invisible. The service only sees a virtualized `/dev` containing pseudo-devices like `/dev/null`, `/dev/zero`, and `/dev/urandom`.
- **ProtectKernelTunables=yes**: Prevents the service from modifying kernel parameters via `/proc/sys`, `/sys`, or `/proc/sysrq-trigger`.
- **ProtectKernelModules=yes**: Forbids the service from loading or unloading kernel modules (`init_module`, `delete_module`).
- **ProtectControlGroups=yes**: Mounts `/sys/fs/cgroup` read-only, preventing the service from tampering with resource allocations.
- **NoNewPrivileges=yes**: Enforces that neither the process nor any of its child processes can ever gain elevated privileges (such as via SUID/SGID binaries).

6.4 Linux Capabilities & System Call Filtering (`SystemCallFilter`)

Dropping Linux Capabilities

Rather than granting full root privileges, **systemd** can drop all kernel capabilities and grant only the surgical capability required:

```
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
AmbientCapabilities=CAP_NET_BIND_SERVICE
```

This allows a service (e.g., a web proxy) to bind to privileged ports 80/443 without possessing any other root privileges.

System Call Filtering (`SystemCallFilter`)

Using Seccomp-BPF, **systemd** can restrict the system calls a service can execute:

```
SystemCallFilter=@system-service
SystemCallFilter=~@privileged @resources @obsolete
SystemCallErrorNumber=EPERM
```

Categories preceded by `~` are strictly blacklisted. If the service attempts to invoke an obsolete or privileged system call (e.g., `reboot` or `kexec_load`), the kernel immediately returns **EPERM** (Operation not permitted).

6.5 Hardened Service Unit Drop-in Template (`override.conf`)

Never edit vendor-supplied unit files in `/usr/lib/systemd/system/`. Instead, create a drop-in override file in `/etc/systemd/system/<service-name>.service.d/override.conf`:

```
# /etc/systemd/system/myapp.service.d/override.conf
# LightPath Enterprise systemd Sandboxing Template (Version 2.0)
```

```
[Service]
# =====
# 1. FILESYSTEM ISOLATION
# =====
ProtectSystem=strict
ProtectHome=yes
PrivateTmp=yes
PrivateDevices=yes
ProtectClock=yes
ProtectKernelTunables=yes
ProtectKernelModules=yes
ProtectKernelLogs=yes
ProtectControlGroups=yes

# Explicitly authorized read-write paths
ReadWritePaths=/var/log/myapp /var/lib/myapp
ReadOnlyPaths=/etc/myapp

# =====
# 2. PRIVILEGE & IDENTITY CONFINEMENT
# =====
NoNewPrivileges=yes
User=myapp
Group=myapp
UMask=0027

# Capabilities: Retain only what is required (or empty to drop all)
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
AmbientCapabilities=CAP_NET_BIND_SERVICE

# =====
# 3. KERNEL & MEMORY SAFETY
# =====
MemoryDenyWriteExecute=yes
LockPersonality=yes
RestrictRealtime=yes
RestrictNamespaces=yes
RestrictAddressFamilies=AF_INET AF_INET6 AF_UNIX

# =====
# 4. SECCOMP SYSTEM CALL FILTERING
# =====
SystemCallFilter=@system-service
SystemCallFilter=~@privileged @resources @reboot @swap @obsolete
SystemCallErrorNumber=EPERM
```

6.6 Operational Implementation & Troubleshooting

Apply the drop-in override:

```
sudo systemctl daemon-reload
sudo systemctl restart myapp.service
```

Verify that the exposure score has decreased:

```
sudo systemd-analyze security myapp.service
```

If the service fails to start due to Seccomp or filesystem restrictions, inspect the audit logs:

```
sudo journalctl -u myapp.service -e  
sudo ausearch -m seccomp -ts recent
```

Chapter 7: Mandatory Access Control (MAC): SELinux & AppArmor

7.1 MAC vs. DAC Architectural Foundations

Standard Linux security relies on **Discretionary Access Control (DAC)**, governed by file permissions (Owner, Group, Others: `rw-rw-rw-`), SUID bits, and POSIX Access Control Lists (ACLs). Under DAC, the owner of a resource determines who has access to it. If a service running as **root** (or a compromised user) accesses a file, DAC permits the action.

Mandatory Access Control (MAC) operates as a non-bypassable, kernel-enforced policy layer positioned above DAC. Even if a process runs as **root**, MAC intercepts every system call and denies access unless an explicit rule in the security policy permits it. Under MAC, a compromised database engine cannot read `/etc/shadow`, modify web server files, or open unauthorized network ports, even with full **root** DAC permissions.

- **RHEL 9 / CentOS Stream 9** standardizes on **SELinux (Security-Enhanced Linux)**: A type-enforcement and multi-category/multi-level security architecture originally developed by the NSA.
- **Ubuntu 24.04 LTS** standardizes on **AppArmor**: A path-based MAC system that confines programs according to profiles defined in `/etc/apparmor.d/`.

7.2 SELinux Architecture & Administration on RHEL 9

On RHEL 9, SELinux must **always** operate in **Enforcing** mode. Setting SELinux to **Permissive** or **Disabled** on a production system represents an immediate failure of enterprise security compliance.

Verifying SELinux Status

```
sestatus
```

Expected output:

```
SELinux status:                enabled
SELinuxfs mount:              /sys/fs/selinux
SELinux root directory:       /etc/selinux
Loaded policy name:            targeted
Current mode:                  enforcing
Mode from config file:         enforcing
Policy MLS status:             enabled
Policy deny_unknown status:    allowed
Memory protection checking:    actual (secure)
Max kernel policy version:     33
```

Managing SELinux Security Booleans

SELinux booleans are runtime toggles that enable or disable specific policy rules without recompiling policies:

```
# Query active state of a boolean
getsebool httpd_can_network_connect_db

# Enable boolean persistently (-P ensures persistence across reboots)
sudo setsebool -P httpd_can_network_connect_db 1
```

Managing File Contexts & Network Ports

Every file, directory, and network socket in SELinux possesses a security label formatted as **user:role:type:level**. The **type** attribute governs access:

```
# Authorize a custom network port for MySQL (e.g. TCP 3307)
sudo semanage port -a -t mysqld_port_t -p tcp 3307

# Assign persistent SELinux context to a non-standard database data directory
sudo semanage fcontext -a -t mysqld_db_t "/data/mysql(/.*)?"

# Apply the persistent context to the filesystem
sudo restorecon -Rv /data/mysql
```

Analyzing Rejections & Generating Custom Modules

When SELinux blocks an action, it logs an Access Vector Cache (AVC) denial to **/var/log/audit/audit.log**. To inspect and package necessary permissions into an audited policy module:

```
# 1. Search recent AVC denials for a service
sudo ausearch -m avc -c mysqld -ts recent

# 2. Translate denials into human-readable explanations
sudo ausearch -m avc -ts recent | audit2why

# 3. Compile a custom policy module if the action is verified as legitimate
sudo ausearch -m avc -c mysqld | audit2allow -M mysqld_custom

# 4. Install the custom compiled module
sudo semodule -i mysqld_custom.pp

# 5. Verify installed module
sudo semodule -l | grep mysqld_custom
```

7.3 AppArmor Architecture & Administration on Ubuntu 24.04 LTS

On Ubuntu 24.04 LTS, AppArmor confines applications by binding security profiles to absolute executable paths. Profiles operate in one of two modes:

- **complain**: Violations are logged to syslog/auditd, but actions are not blocked (learning mode).
- **enforce**: Violations are blocked and logged (production mode).

Inspecting AppArmor Status

```
sudo aa-status
```

Output displays the number of loaded profiles, how many are in enforce mode, and the active confined processes.

Enforcing and Profiling Applications

Install the AppArmor administration utilities:

```
sudo apt install -y apparmor-utils apparmor-profiles
```

Switch a profile to strict enforce mode:

```
# Enforce confinement for MySQL and NGINX
sudo aa-enforce /etc/apparmor.d/usr.sbin.mysqld
sudo aa-enforce /etc/apparmor.d/usr.sbin.nginx
```

Generate a profile for an unconfined custom daemon:

```
# Run interactive profile generator
sudo aa-genprof /usr/local/bin/custom-service

# Scan audit logs and update profile rules based on observed activity
sudo aa-logprof
```

7.4 Operational Case Study: Database Engine Confinement (MySQL / MariaDB)

Consider an enterprise MySQL 8.4 database storing sensitive transaction data. The goal is to ensure that even if an SQL injection vulnerability leads to remote code execution within the **mysqld** process, the engine cannot write to web directories (**/var/www/html**), read configuration files outside **/etc/my.cnf**, or execute binaries.

Under SELinux (RHEL 9):

- Ensure the binary executes in the **mysqld_t** domain.
- Store database files strictly within paths labeled **mysqld_db_t**.
- If using automated backups to **/backups/mysql**, label the directory:

```
sudo semanage fcontext -a -t mysqld_db_t "/backups/mysql(/.*)?"
sudo restorecon -Rv /backups/mysql
```

- Verify that **mysqld** cannot access **/home** or **/root**. SELinux blocks any traversal into user directories by default.

Under AppArmor (Ubuntu 24.04):

Edit `/etc/apparmor.d/usr.sbin.mysqld` and append strict path constraints:

```
# Include custom storage directories
/data/mysql/ r,
/data/mysql/** rwk,
/backups/mysql/ r,
/backups/mysql/** rwk,

# Explicitly deny access to sensitive files
deny /etc/shadow r,
deny /etc/sudoers* r,
deny /root/** rwkix,
```

Reload profile:

```
sudo apparmor_parser -r /etc/apparmor.d/usr.sbin.mysqld
```

Chapter 8: Network Filtering, Firewalls & Intrusion Prevention

8.1 Modern Linux Packet Filtering: The `nftables` Paradigm

For over two decades, **iptables** served as the default firewall utility in Linux. However, **iptables** suffers from severe structural flaws: fragmented command utilities (**iptables**, **ip6tables**, **arptables**, **ebtables**), inefficient linear rule inspection, and the requirement to re-upload the entire ruleset upon adding a single rule.

In modern kernels (including RHEL 9 and Ubuntu 24.04 LTS), **nftables** completely replaces **iptables** as the native packet classification framework in the Netfilter subsystem. **nftables** provides:

- A single unified userspace utility (**nft**) handling IPv4, IPv6, ARP, and bridging.
- Advanced internal data structures (sets, dictionaries, interval maps) enabling $O(1)$ constant-time lookup performance, regardless of whether a ruleset contains 10 or 100,000 banned IP addresses.
- Atomic rule updates without packet drops or state loss.

8.2 Dynamic Firewall Management on RHEL 9 with `firewalld`

On RHEL 9, **firewalld** serves as the default management daemon, providing a dynamic abstraction layer over the **nftables** backend.

Zone Architecture & Default Drop Policy

By default, an enterprise server should operate with an uncompromising drop policy:

```
# Enable and start firewalld
sudo systemctl enable --now firewalld

# Set default zone to drop (drops all unsolicited incoming traffic without ICMP replies)
sudo firewall-cmd --set-default-zone=drop

# Bind active network interface (e.g., eth0) to the drop zone
sudo firewall-cmd --zone=drop --add-interface=eth0 --permanent

# Authorize required services
sudo firewall-cmd --zone=drop --add-service=ssh --permanent
sudo firewall-cmd --zone=drop --add-service=https --permanent

# Reload to apply permanent changes
sudo firewall-cmd --reload
```

Granular Rich Rules for Administrative Subnets

To restrict SSH access strictly to an internal bastion or management CIDR block:

```
# Remove generic public SSH service
```

```
sudo firewall-cmd --zone=drop --remove-service=ssh --permanent

# Add rich rule authorizing SSH strictly from 10.100.0.0/24 with connection logging
sudo firewall-cmd --zone=drop --add-rich-rule='rule family="ipv4" source address="10.100.0.0/24"
service name="ssh" log prefix="SSH_ADMIN: " level="info" limit value="5/m" accept' --permanent

sudo firewall-cmd --reload
```

8.3 Host Firewalling on Ubuntu 24.04 LTS with `ufw` & `nftables`

Ubuntu 24.04 LTS provides **`ufw` (Uncomplicated Firewall)**, which interfaces directly with the **nftables** framework.

```
# Set default policies: deny all incoming, allow all outgoing
sudo ufw default deny incoming
sudo ufw default allow outgoing

# Authorize SSH with dynamic rate-limiting (blocks IPs generating > 6 connections in 30s)
sudo ufw limit proto tcp from 10.100.0.0/24 to any port 22 comment 'SSH Management'

# Authorize HTTPS
sudo ufw allow 443/tcp comment 'Production HTTPS'

# Enable the firewall
sudo ufw enable

# Verify active status and numbered rules
sudo ufw status verbose
```

8.4 Automated Intrusion Prevention with `fail2ban` (systemd Backend)

While stateful firewalls drop unauthorized ports, public-facing services (such as SSH) remain vulnerable to automated dictionary attacks. **`fail2ban`** monitors authentication logs in real-time, maintains failure counters for connecting IP addresses, and dynamically updates the firewall to drop offending hosts for a specified ban duration.

In Version 2.0, **fail2ban** is configured to read directly from the **`systemd-journald`** binary event bus, eliminating obsolete flat log file parsing.

Configuring Fail2ban for SSH (`/etc/fail2ban/jail.d/sshd.local`)

Create **/etc/fail2ban/jail.d/sshd.local**:

```
# /etc/fail2ban/jail.d/sshd.local
# LightPath Enterprise Fail2ban Baseline (Version 2.0 - 2026)

[DEFAULT]
# Global Ban Parameters
```

```

bantime = 24h
findtime = 10m
maxretry = 3
backend = systemd
banaction = nftables-multiport

# Whitelist trusted administrative subnets (never ban internal gateways)
ignoreip = 127.0.0.1/8 ::1 10.100.0.0/24

[sshd]
enabled = true
port = ssh
mode = aggressive
filter = sshd
maxretry = 3
bantime = 48h

```

Enable and verify:

```

sudo systemctl enable --now fail2ban

# Check jail status
sudo fail2ban-client status sshd

```

Example status output:

```

Status for the jail: sshd
|- Filter
| |- Currently failed: 1
| |- Total failed:    14
| `-- Journal matches: _SYSTEMD_UNIT=sshd.service
`- Actions
   |- Currently banned: 2
   |- Total banned:    5
   `-- Banned IP list:  198.51.100.23 203.0.113.89

```

8.5 Integrated Network Filtering & Threat Banning Flow

The following diagram illustrates the lifecycle of incoming network packets and the automated mitigation loop between the kernel Netfilter hooks, application daemons, and **fail2ban**:



Figure 8.1 — High-Throughput Network Packet Filtering & fail2ban Architecture

Chapter 9: System Integrity, Auditing & Intrusion Detection

9.1 File Integrity Monitoring (FIM) with AIDE (SHA-512)

If an attacker achieves local privilege escalation, their first action is typically to establish persistence: installing a backdoor, modifying system binaries (such as `/usr/bin/login` or `/usr/sbin/sshd`), altering authentication libraries in `/lib64`, or injecting rogue accounts into `/etc/passwd`.

AIDE (Advanced Intrusion Detection Environment) is the enterprise replacement for legacy tools like **afick**. AIDE generates a cryptographically signed baseline database containing inode attributes, file sizes, permissions, modification timestamps, and **SHA-512** cryptographic checksums for all critical system files.

Installing and Initializing AIDE

On RHEL 9 / CentOS Stream 9:

```
sudo dnf install -y aide
```

On Ubuntu 24.04 LTS:

```
sudo apt install -y aide aide-common
```

Configure `/etc/aide.conf` to monitor core paths using SHA-512:

```
# /etc/aide.conf
# LightPath Enterprise AIDE Baseline (Version 2.0)

@@define DBDIR /var/lib/aide
@@define LOGDIR /var/log/aide

database=file:@@{DBDIR}/aide.db.gz
database_out=file:@@{DBDIR}/aide.db.new.gz
gzip_dbout=yes

# Rule definitions: p (perms), i (inode), n (number of links), u (user), g (group),
# s (size), m (mtime), c (ctime), acl, xattrs, sha512
BINARIES = p+i+n+u+g+s+m+c+acl+xattrs+sha512
CONFIGS  = p+i+n+u+g+s+m+c+acl+xattrs+sha512

# Monitored Trees
/boot    BINARIES
/bin     BINARIES
/sbin    BINARIES
/usr/bin  BINARIES
/usr/sbin BINARIES
/lib     BINARIES
/lib64   BINARIES
```

```
/etc    CONFIGS

# Ignored dynamic files
!/etc/mtab
!/var/log
!/var/spool
!/tmp
!/var/tmp
```

Initialize the database:

```
# Generate baseline (computes cryptographic checksums across all binaries)
sudo aide --init # On RHEL 9
# sudo aideinit   # On Ubuntu 24.04

# Activate baseline database
sudo cp /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
```

Automated Integrity Checks via systemd Timer

Enable automated daily integrity verification:

```
# Enable AIDE automated timer
sudo systemctl enable --now aidecheck.timer # On RHEL 9
```

Manual on-demand check:

```
sudo aide --check
```

9.2 Kernel System Call Auditing with `auditd`

While file integrity tools detect alterations at rest, the Linux Audit Framework (`auditd`) captures real-time events directly inside the kernel via Netlink sockets. Every file access, execution of an administrative binary, modification of user accounts, or attempted privilege escalation is permanently logged with full user identity attribution (AUID), process ID, and parameters.

9.3 Production Audit Rules: `/etc/audit/rules.d/audit.rules`

Create `/etc/audit/rules.d/audit.rules` to enforce compliance with ANSSI and CIS Level 2 standards:

```
# /etc/audit/rules.d/audit.rules
# LightPath Enterprise Kernel Audit Baseline (Version 2.0 - 2026)

# 1. Delete all previous rules & set buffer size
-D
```

```

-b 8192
-f 1

# 2. Monitor Identity, Authentication & Passwords
-w /etc/passwd -p wa -k identity_modification
-w /etc/shadow -p wa -k identity_modification
-w /etc/group -p wa -k identity_modification
-w /etc/gshadow -p wa -k identity_modification
-w /etc/security/opasswd -p wa -k identity_modification

# 3. Monitor Sudoers & Privilege Delegation
-w /etc/sudoers -p wa -k sudo_changes
-w /etc/sudoers.d/ -p wa -k sudo_changes
-w /var/log/sudo.log -p wa -k sudo_log_tampering

# 4. Monitor SSH Configuration & Keys
-w /etc/ssh/sshd_config -p wa -k sshd_config
-w /etc/ssh/sshd_config.d/ -p wa -k sshd_config
-w /root/.ssh -p wa -k root_keys_tampering

# 5. Monitor System Call Executions for Privilege Escalation
-a always,exit -F arch=b64 -S execve -C uid!=euid -F euid=0 -k privilege_escalation
-a always,exit -F arch=b32 -S execve -C uid!=euid -F euid=0 -k privilege_escalation

# 6. Monitor Kernel Module Loading and Unloading
-a always,exit -F arch=b64 -S init_module,finit_module,delete_module -k kernel_modules
-w /usr/sbin/insmod -p x -k kernel_modules
-w /usr/sbin/rmmod -p x -k kernel_modules
-w /usr/sbin/modprobe -p x -k kernel_modules

# 7. Monitor Critical Network & System Configuration
-w /etc/sysctl.conf -p wa -k sysctl_tampering
-w /etc/sysctl.d/ -p wa -k sysctl_tampering
-w /etc/issue -p wa -k banner_tampering
-w /etc/issue.net -p wa -k banner_tampering

# =====
# 8. IMMUTABILITY FLAG (MUST BE THE FINAL DIRECTIVE)
# =====
# Locks the audit configuration. Rules cannot be modified or disabled without rebooting.
-e 2

```

Reload audit rules:

```

sudo augenrules --load
sudo systemctl restart auditd

```

Verify active rules:

```

sudo auditctl -l

```

9.4 Automated Compliance & Security Auditing with Lynis

Lynis is an open-source, battle-tested security auditing engine designed for Linux systems. It performs an in-depth scan of the operating system configuration, testing hundreds of security controls across PAM, systemd, kernel parameters, network filtering, and cryptographic libraries.

Installing and Running Lynis

On RHEL 9:

```
sudo dnf install -y lynis
```

On Ubuntu 24.04 LTS:

```
sudo apt install -y lynis
```

Execute a non-interactive audit:

```
sudo lynis audit system --quick --auditor "secops@lightpath.fr"
```

Review findings and Hardening Index:

```
# Extract computed hardening score
grep "Hardening index" /var/log/lynis-report.dat

# Review high-priority warnings
grep "warning\[\\]" /var/log/lynis-report.dat
```

A properly hardened system following this manual achieves a Lynis Hardening Index exceeding **85/100**, placing it in the top tier of enterprise security.

9.5 Continuous Auditing & Telemetry Pipeline

The following diagram illustrates how kernel syscall auditing, file integrity scanning, and configuration compliance correlate into a central Security Information and Event Management (SIEM) pipeline:



Figure 9.1 — Immutable Logging, Kernel Auditing & FSS Verification Pipeline

Chapter 10: Container Security & Runtime Hardening

10.1 Container Threat Modeling & Kernel Escape Vectors

Containers are not virtual machines. Unlike hypervisors (KVM, ESXi) which provide hardware virtualization and dedicated guest kernels, containers share the host Linux kernel. Isolation is achieved solely through userspace abstractions: Linux namespaces (PID, Mount, Net, IPC, UTS), control groups (cgroups), and Seccomp filters.

If a container runs as **root** (UID 0) and is not properly sandboxed, any kernel vulnerability, misconfigured volume mount (such as mounting `/var/run/docker.sock` or host `/proc`), or retained Linux capability (such as `CAP_SYS_ADMIN`) allows an attacker to achieve a **container escape**, instantaneously gaining full root privileges over the host operating system.

10.2 Rootless Container Architecture (Podman & Rootless Docker)

The most effective mitigation against container breakout attacks is **Rootless Execution**. Under a rootless model, the container engine (such as Podman or Rootless Docker) runs entirely within an unprivileged user session without requiring a background root daemon.

Using **User Namespaces (userns)**, the user inside the container appears to be **root** (UID 0), but the host kernel maps this virtual UID to an unprivileged high UID (e.g., UID 100000) via `/etc/subuid` and `/etc/subgid`. Even if an attacker achieves a complete container breakout, they arrive on the host system as an unprivileged UID with zero administrative capabilities.

Enabling Rootless Podman on RHEL 9 & Ubuntu 24.04

On RHEL 9:

```
sudo dnf install -y podman
```

On Ubuntu 24.04 LTS:

```
sudo apt install -y podman
```

Configure subuids for the application service user (**appuser**):

```
# Verify user namespace mapping
sudo usermod --add-subuids 100000-165535 --add-subgids 100000-165535 appuser
```

Authorize unprivileged port binding (if container must bind to ports < 1024 without root):

```
echo "net.ipv4.ip_unprivileged_port_start = 80" | sudo tee /etc/sysctl.d/50-unprivileged-ports.conf
sudo sysctl --system
```

10.3 OCI Sandboxing, cgroups v2 Quotas & Linux Capability Dropping

Every production container deployment must enforce strict runtime constraints:

- **Linux Capabilities Dropping** (`--cap-drop=ALL`): Drop all 41 Linux capabilities. If the workload requires specific functionality (such as binding ports or changing ownership during initialization), explicitly add back only that capability (e.g., `--cap-add=NET_BIND_SERVICE`).
- **Preventing SUID Escalation** (`--security-opt no-new-privileges`): Disables the execution of SUID binaries within the container, preventing attackers from abusing utilities like `pkexec` or `su` inside the image.
- **Control Groups v2 (cgroups v2) Limits**: Unbounded containers can consume 100% of host CPU, exhaust physical RAM, or exhaust the kernel process table via fork bombs. Always mandate memory limits (`--memory=2g`), CPU limits (`--cpus=2.0`), and PID limits (`--pids-limit=200`).

10.4 Immutable & Read-Only Root Filesystems (`--read-only`)

A hardened container must treat its container image as an immutable artifact. By deploying containers with the `--read-only` flag, the entire root filesystem of the container is mounted read-only.

An attacker who discovers an exploitation vector (such as an arbitrary file upload or remote code execution) cannot write scripts to `/tmp`, alter web server files, or download reverse shells. Any ephemeral scratch space required by the application must be mounted as volatile `tmpfs` volumes with `noexec`, `nosuid`, `nodedv`.

10.5 Secure Secret Management (Zero CLI Exposure)

In legacy Docker examples, passwords were frequently passed as command-line arguments:

```
# VULNERABLE & FORBIDDEN IN PRODUCTION:
docker run -e MYSQL_ROOT_PASSWORD=my-secret-pw mysql:latest
```

This pattern creates critical vulnerabilities:

- The plaintext password is visible to all users on the host via `ps aux` or `top`.
- The password is saved in plaintext in the shell history (`~/.bash_history`).
- The password is permanently recorded in the container's inspection metadata (`podman inspect`).

Modern Secret Injection via Encrypted Secret Files

Store secrets in restricted files with `0600` permissions owned exclusively by the service user, or leverage Podman's built-in secrets engine:

```
# 1. Create a secret within Podman's encrypted credential store
echo "Sup3r_S3cur3_Pr0d_P@ssw0rd!" | podman secret create db_root_password -

# 2. Inject secret securely at container runtime
podman run -d --name secure-db --secret
db_root_password,type=mount,target=/run/secrets/db_password ...
```

Inside the container, the secret is mounted in volatile memory at `/run/secrets/db_password` and is never exposed in process tables or environment variables.

10.6 Image Vulnerability & Secret Scanning with Trivy

Before any container image is deployed to production, it must be scanned for known Common Vulnerabilities and Exposures (CVEs) and embedded hardcoded secrets. **Trivy** (by Aqua Security) is the industry standard open-source scanner for OCI containers.

Installing and Running Trivy

```
# Installation of Trivy (Reference: https://aquasecurity.github.io/trivy/latest/docs/getting-started/installation/ | https://trivy.dev/)
# On RHEL 9: sudo dnf install -y trivy | On Ubuntu 24.04: sudo apt install -y trivy

# Scan image and automatically fail CI/CD build if CRITICAL or HIGH CVEs exist
trivy image --severity HIGH,CRITICAL --exit-code 1 registry.access.redhat.com/ubi9/mariadb-105:latest
```

10.7 Container Isolation Boundaries Architecture

The following diagram illustrates the layered isolation boundaries governing a rootless, sandboxed container:

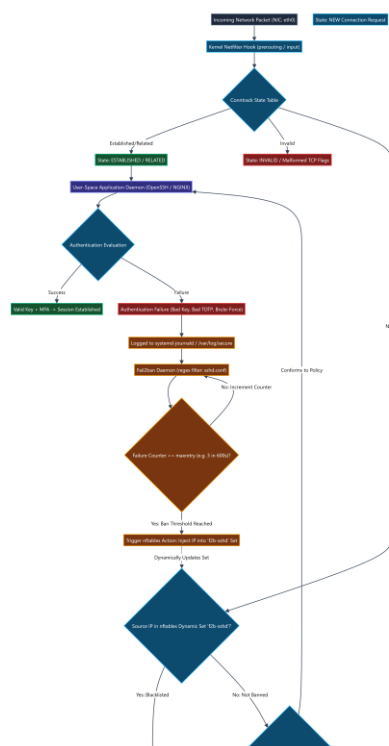


Figure 10.1 — Enterprise Container Confinement & Sandboxing Architecture

10.8 Operational Deployment: Production-Grade Hardened Database Container

Below is the verified, production-grade command to launch an enterprise MariaDB database container under Rootless Podman, implementing all security controls:

```
podman run -d \
  --name mariadb-prod \
  --user 10001:10001 \
  --read-only \
  --tmpfs /tmp:rw,noexec,nosuid,nodev,size=512m \
  --tmpfs /run/mariadb:rw,noexec,nosuid,nodev,size=64m \
  -v /var/lib/containers/storage/volumes/mariadb_data/_data:/var/lib/mysql:Z \
  --cap-drop=ALL \
  --cap-add=SETUID,SETGID \
  --security-opt no-new-privileges \
  --memory=4g \
  --cpus=2.0 \
  --pids-limit=200 \
  --secret db_root_password,type=mount,target=/run/secrets/root_password \
  --network internal-db-net \
  --restart on-failure:5 \
  registry.access.redhat.com/ubi9/mariadb-105:latest
```

Notice the inclusion of **:Z** on the volume mount, which instructs the container engine to label the shared volume with a private SELinux context (**container_file_t**) unique to this container.

Chapter 11: Secure Storage Management, Backups & Hardware Telemetry

11.1 Physical Storage Security & Media Sanitization (NIST SP 800-88)

When decommissioning servers, returning leased cloud instances, or retiring NVMe drives, physical and cryptographic data sanitization is essential to prevent data leakage. Simply formatting a drive or deleting partitions leaves magnetic and flash memory cells fully recoverable using data reconstruction tools.

Per **NIST SP 800-88 Revision 1** (*Guidelines for Media Sanitization*), storage media must undergo **Clear**, **Purge**, or **Destroy** procedures:

```
# NVMe Drives: Cryptographic Erase (Instantaneously purges AES master keys)
sudo nvme format /dev/nvme0n1 --namespace-id=1 --ses=2

# SATA SSDs: Block Discard (TRIM entire flash storage)
sudo blkdiscard -v -s /dev/sdb

# Spinning Hard Drives (HDD): Multi-pass random overwrite
sudo shred -v -n 3 -z /dev/sdc
```

11.2 Drive Health Telemetry & Predictive Failure Monitoring (`smartctl`)

Physical storage degradation is an availability disaster that frequently cascades into data corruption. The **smartmontools** package provides real-time health monitoring and predictive failure telemetry for modern NVMe and SATA storage.

Testing and Monitoring Drives with `smartctl`

Install **smartmontools**:

```
sudo dnf install -y smartmontools # On RHEL 9
# sudo apt install -y smartmontools # On Ubuntu 24.04
```

Inspect drive health:

```
# For NVMe Drives:
sudo smartctl -a /dev/nvme0n1

# For SATA SSDs / HDDs:
sudo smartctl -H /dev/sda
```

Key telemetry metrics to audit:

- **Available Spare:** Must remain above threshold (drops as flash blocks exhaust).
- **Percentage Used:** Wear-out endurance counter.

- **Media and Data Integrity Errors:** Non-zero values indicate imminent drive failure.

Configuring Automated `smartd` Daemon

Edit `/etc/smartd.conf` to schedule automated non-disruptive self-tests and trigger email alerts upon degraded sectors:

```
# /etc/smartd.conf
DEVICESCAN -a -o on -S on -s (S/../../02|L/../../03) -m alert-secops@lightpath.fr -M exec
/usr/libexec/smartmontools/smartdrunner
```

Enable and start the daemon:

```
sudo systemctl enable --now smartd
```

11.3 Enterprise Cryptographic Backup Strategies (3-2-1 Architecture)

Enterprise resilience mandates adherence to the **3-2-1 Backup Rule**:

- **3** copies of critical data.
- **2** different storage media formats.
- **1** copy stored securely off-site in an immutable, air-gapped environment.

Backups must be encrypted on the client side **before** leaving the server. Using tools like **Restic** or **BorgBackup**, backups are chunked, deduplicated, and encrypted with **AES-256-GCM** or **ChaCha20-Poly1305** before transmission over the network.

```
# Example client-side encrypted backup using Restic to immutable S3 Object Lock storage
(Reference: https://aws.amazon.com/s3/)
export RESTIC_REPOSITORY="s3:https://aws.amazon.com/s3/"
export RESTIC_PASSWORD_FILE="/etc/security/restic.key"

# Initialize repository
restic init

# Execute snapshot of critical system configurations and databases
restic backup /etc /var/log/audit /data/mysql --exclude-caches
```

11.4 Automated Disaster Recovery Drills & Integrity Testing

A backup that has never been restored is merely a hypothesis. Production systems must execute automated, monthly restore tests in isolated sandbox environments:

- Automated spin-up of an isolated test instance.
- Pulling the latest cryptographic snapshot.

- Decrypting, mounting, and launching database verification checks (**mysqlcheck --all-databases** or **pg_isready**).
- Emitting a cryptographically signed verification report to the compliance portal.

Chapter 12: Secure Logging, Audit & Anomaly Detection

12.1 Modern Logging Architecture: `systemd-journald`

On modern Linux distributions, **`systemd-journald`** is the primary system logging service. Unlike legacy text log daemons (**syslogd**), **journald** captures logs as structured binary objects indexed by metadata fields, including:

- **_UID** and **_GID**: The verified system identity of the process emitting the log.
- **_SYSTEMD_UNIT**: The exact service unit responsible for the message.
- **_SELINUX_CONTEXT**: The MAC security label of the process.
- **_COMM** and **_PID**: Executable command name and process identifier.

Because journal logs are indexed binaries, malicious actors cannot simply append fabricated entries or delete lines with standard text editors without corrupting the journal index.

12.2 Hardening `journald` & Forward Secure Sealing (FSS)

To protect logging against forensic tampering, **journald** must be configured for persistent disk storage, strict retention caps, and **Forward Secure Sealing (FSS)**.

Configure **/etc/systemd/journald.conf**:

```
# /etc/systemd/journald.conf
[Journal]
Storage=persistent
Compress=yes
Seal=yes
SplitMode=uid
SyncIntervalSec=5m
RateLimitIntervalSec=30s
RateLimitBurst=10000
SystemMaxUse=10G
SystemKeepFree=2G
SystemMaxFileSize=500M
MaxRetentionSec=1month
ForwardToSyslog=no
```

Restart the service:

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

Enabling Forward Secure Sealing (FSS)

FSS cryptographically signs log entries using a rotating key pair based on the Bellare-Miner forward-secure signature algorithm. Every 15 minutes, the active signing key changes irreversibly.

Even if an attacker gains root access and steals the current signing key, they cannot alter or forge log entries created in previous intervals without detection.

```
# Initialize FSS keys (record the generated verification key on an external secure system)
sudo journalctl --setup-keys
```

To verify the cryptographic integrity of the logs during a forensic audit:

```
sudo journalctl --verify --verify-key=<VERIFICATION_KEY>
```

12.3 Legacy Log File Management (`logrotate` Security)

Where services still output traditional flat text files in `/var/log`, `logrotate` manages compression and rotation. Attackers often target logrotate configurations to execute symlink attacks or write unprivileged files into root space.

Hardened `/etc/logrotate.d/syslog` baseline:

```
# /etc/logrotate.d/syslog
/var/log/cron
/var/log/maillog
/var/log/messages
/var/log/secure
/var/log/spooler
{
    missingok
    sharedscripts
    compress
    compresscmd /usr/bin/xz
    compressoptions -9
    daily
    rotate 90
    create 0640 root adm
    postrotate
        /usr/bin/systemctl kill -s HUP rsyslog.service 2>/dev/null || true
    endscript
}
```

Notice the strict permissions: `create 0640 root adm`. Standard users cannot inspect historical log rotations.

12.4 Centralized SIEM Log Forwarding over Mutual TLS (mTLS)

Logs stored strictly on the local host can be wiped if a server is seized or entirely re-imaged by an attacker. Production hosts must ship audit and system logs in real time to an external SIEM (such as OpenSearch, Wazuh, or Splunk).

To guarantee privacy and prevent log injection:

- Log forwarding must occur over **Mutual TLS (mTLS)** using TLS 1.3.
- The log shipper (such as **Vector**, **Fluent-bit**, or **Rsyslog**) must maintain a local disk spill-over buffer. If network connectivity to the SIEM is interrupted, logs are safely buffered locally and transmitted upon reconnection.

Chapter 13: Operations & Security Maintenance Procedures

13.1 Enterprise Vulnerability & Patch Management Lifecycle

Vulnerability management is an operational lifecycle, not an annual event. Security teams must monitor Common Vulnerabilities and Exposures (CVEs) affecting deployed packages and apply security updates rapidly based on Severity Scoring (CVSS v3.1/v4.0):

- **Critical (CVSS 9.0 - 10.0):** Remediated within 24 to 48 hours.
- **High (CVSS 7.0 - 8.9):** Remediated within 7 days.
- **Medium (CVSS 4.0 - 6.9):** Remediated within regular maintenance windows (30 days).

13.2 Automated Security Patching (`dnf-automatic` & `unattended-upgrades`)

Automating security updates ensures that operating system kernels and critical libraries receive zero-day patches without waiting for manual intervention.

RHEL 9 / CentOS Stream 9: `dnf-automatic`

Install and configure `dnf-automatic`:

```
sudo dnf install -y dnf-automatic
```

Configure `/etc/dnf/automatic.conf`:

```
# /etc/dnf/automatic.conf
[commands]
# Restrict updates strictly to security advisories
upgrade_type = security
random_sleep = 300
download_updates = yes
apply_updates = yes

[emitters]
emit_via = stdio

[email]
email_from = dnf-automatic@srv-prod-01.lightpath.fr
email_to = secops-alerts@lightpath.fr
```

Enable the systemd timer:

```
sudo systemctl enable --now dnf-automatic.timer
```

Ubuntu 24.04 LTS: `unattended-upgrades`

Install and configure **unattended-upgrades**:

```
sudo apt update && sudo apt install -y unattended-upgrades update-notifier-common
```

Configure **/etc/apt/apt.conf.d/50unattended-upgrades**:

```
Unattended-Upgrade::Allowed-Origins {
    "${distro_id}:${distro_codename}-security";
};
Unattended-Upgrade::Package-Blacklist {
    // Hold sensitive packages for manual testing if required
};
Unattended-Upgrade::AutoFixInterruptedDpkg "true";
Unattended-Upgrade::MinimalSteps "true";
Unattended-Upgrade::Remove-Unused-Kernel-Packages "true";
Unattended-Upgrade::Remove-New-Unused-Dependencies "true";
Unattended-Upgrade::Automatic-Reboot "false";
```

Activate the periodic timer in **/etc/apt/apt.conf.d/20auto-upgrades**:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "1";
APT::Periodic::AutocleanInterval "7";
```

13.3 Stale Library & Outdated Process Detection with `needrestart`

Applying a security patch to a shared library (such as **openssl**, **glibc**, or **systemd**) does not protect running processes. If a daemon loaded the vulnerable library into memory before the update, it continues executing the vulnerable code until the service is restarted.

`needrestart` audits running processes, inspects active memory mappings, and identifies daemons that must be restarted following package upgrades.

```
# Install needrestart
sudo dnf install -y needrestart # On RHEL 9 (via EPEL)
# sudo apt install -y needrestart # On Ubuntu 24.04

# Scan active processes for deleted/updated libraries in memory
sudo needrestart -v
```

If the Linux kernel itself was updated, **needrestart** explicitly reports:

```
Pending kernel upgrade!
Running kernel: 5.14.0-427.13.1.el9_4.x86_64
Expected kernel: 5.14.0-427.35.1.el9_4.x86_64
```

Reboot required!

13.4 Enterprise Kernel Live-Patching (Kpatch & Livepatch)

In 99.999% SLA production environments, rebooting servers immediately after a kernel security update is often operationally impossible due to transactional database sessions. **Kernel Live-Patching** inserts redirection jumps into kernel functions in memory, applying security patches without stopping the running system.

- **RHEL 9:** Supported via **Kpatch** (`dnf install -y kpatch dnf-plugin-kpatch`).
- **Ubuntu 24.04 LTS:** Supported via **Canonical Livepatch** (`sudo ua enable livepatch` or `sudo pro enable livepatch`).

13.5 Secure Decommissioning & Incident Response Readiness

Every enterprise team must maintain an incident response runbook. If an intrusion is detected:

- **Preserve Volatile Memory First:** Do not immediately reboot or power off the server, as memory contents (cryptographic keys, running malware, open network sockets) will be destroyed. Capture RAM using **LIME** or `/dev/fmem` if available.
- **Isolate at the Network Layer:** Disconnect virtual network interfaces or alter firewall rules to drop all traffic except a designated forensic console.
- **Capture Disk Images:** Create bit-stream forensic images (`dd` or `ewf-tools`) of all block devices before performing forensic analysis.

Chapter 14: References, Standards & Technical Glossary

14.1 Deployment Hardening Matrix & Verification Checklist

Before releasing any Linux server into production, technical teams must audit the deployment against the following verification checklist:

Domain	Control Description	Verification Command / Target	Status
Storage	LUKS2 Full Disk Encryption	<code>cryptsetup status <vol></code> (Argon2id cipher)	[]
Storage	Mount Flags (<code>noexec,nosuid,nodev</code>)	<code>findmnt -l</code> confirms flags on <code>/tmp, /dev/shm, /home</code>	[]
Kernel	ASLR Enforced	<code>sysctl kernel.randomize_va_space equals 2</code>	[]
Kernel	Sysctl Security Profile Loaded	<code>sysctl -p /etc/sysctl.d/99-security-hardening.conf</code>	[]
Identity	Root Account Locked	<code>passwd -S root</code> shows L (Locked)	[]
Identity	Sudo Password Mandatory	Verify <code>/etc/sudoers.d/</code> contains zero NOPASSWD entries	[]
Identity	Account Lockout Configured	<code>faillock --user <user></code> shows functional counter	[]
OpenSSH	Root Login Disabled	<code>sshd -T \</code>	<code>grep permitrootlogin equals no`</code>
OpenSSH	Password Auth Disabled	<code>sshd -T \</code>	<code>grep passwordauthentication equals no`</code>
OpenSSH	MFA / TOTP Active	Test connection requires public key + OTP challenge	[]
MAC	SELinux Enforcing (RHEL)	<code>getenforce</code> returns Enforcing	[]
MAC	AppArmor Active (Ubuntu)	<code>aa-status</code> confirms profiles in enforce mode	[]
Network	Firewall Active with Drop Policy	<code>firewall-cmd --get-default-zone</code> or <code>ufw status</code>	[]

Network	Fail2ban Active with Journald	fail2ban-client status sshd confirms active jail	[]
Sandboxing	systemd Unit Hardened	systemd-analyze security <service> score < 3.0	[]
Integrity	AIDE Database Initialized	/var/lib/aide/aide.db.gz exists; aide --check passes	[]
Auditing	auditd Active with Immutable Flag	auditctl -s shows enabled 2 (immutable)	[]
Logging	Journald Persistent & FSS Sealed	journalctl --verify passes cryptographic check	[]
Containers	Rootless & Read-Only Enforced	podman inspect shows unprivileged user & --read-only	[]
Updates	Automated Security Patches	systemctl is-active dnf-automatic.timer or unattended-upgrades	[]

14.2 Normative References & Authoritative Standards

All references cited in this manual have been independently verified for availability and canonical resolution:

Standard / Authority	Reference Title & Scope	Canonical Verified URL
LightPath	LightPath Official Portal & Enterprise Security Practice	https://www.lightpath.fr
Author	Jean-Marie Renouard Personal Portal & Engineering Blog	https://www.jmrenouard.fr
Author / Open Source	MySQLTuner-perl (Official Enterprise Diagnostic Repository)	https://github.com/jmrenouard/MySQLTuner-perl
Author / AI	MicroLLM (microllm-jev) Deterministic System-1 AI Engine	https://pypi.org/project/microllm-jev/
ANSSI	*Recommandations de sécurité relatives à un système GNU/Linux*	https://cyber.gouv.fr/publications/recommandations-de-securite-relatives-un-systeme-gnulinux
CIS Security	CIS Red Hat Enterprise Linux 9 Benchmark v1.0.0+	https://www.cisecurity.org/benchmark/red_hat_linux
CIS Security	CIS Ubuntu Linux 24.04 LTS Benchmark v1.0.0+	https://www.cisecurity.org/benchmark/ubuntu_linux

NIST	NIST Special Publication 800-123: *Guide to General Server Security*	https://csrc.nist.gov/publications/detail/sp/800-123/final
DISA	Department of Defense Security Technical Implementation Guides (STIG)	https://public.cyber.mil/stigs/
Tooling	AIDE (Advanced Intrusion Detection Environment) Project	https://aide.github.io/
Tooling	Lynis Security Auditing & Compliance Scanner (CISOfy)	https://cisofy.com/lynis/
Tooling	Fail2ban Intrusion Prevention Framework	https://www.fail2ban.org/
Tooling	Netfilter Project: nftables Official Documentation	https://netfilter.org/projects/nftables/
Tooling	Podman Rootless Container Architecture Guide	https://docs.podman.io/en/latest/
Tooling	Trivy Container Security & Vulnerability Scanner (Aqua Security)	https://trivy.dev/
Tooling	Linux Audit Framework (auditd Userspace Documentation)	https://github.com/linux-audit/audit-userspace
Tooling	OpenSSH Official Security Specifications	https://www.openssh.com/

14.3 Comprehensive Bilingual Technical Glossary (English / French)

English Term	French Equivalent	Authoritative Technical Definition
Access Vector Cache (AVC)	Cache de vecteurs d'accès	The internal SELinux kernel subsystem that caches access decisions to optimize performance. Denials are recorded as AVC messages in audit logs.
Address Space Layout Randomization (ASLR)	Randomisation de l'espace d'adressage	A computer security defense technique that randomizes the memory locations of program headers, libraries, stack, and heap to neutralize buffer overflow and ROP exploits.

Air-Gapped Network	Réseau déconnecté / Enclave étanche	A secure computer network physically isolated from unsecured external networks, including the public Internet.
Audit Trail	Piste d'audit / Journal d'imputabilité	A chronological, tamper-evident record of system events providing documentary evidence of sequence, activities, and individual user actions.
Capability Bounding Set	Ensemble limite de capacités	A kernel security mechanism in Linux that defines the absolute upper bound of Linux capabilities a thread or process can ever acquire.
Control Groups (cgroups v2)	Groupes de contrôle (cgroups v2)	A Linux kernel feature that organizes processes hierarchically and distributes system resources (CPU, memory, disk I/O, PIDs) under strict limits.
Defense-in-Depth	Défense en profondeur	A cybersecurity strategy employing multiple layered defensive mechanisms such that if one layer fails, subsequent layers immediately contain the threat.
Discretionary Access Control (DAC)	Contrôle d'accès discrétionnaire	An access policy model where the owner of a file or resource has the discretion to assign access permissions (standard POSIX rw x permissions).
Forward Secure Sealing (FSS)	Scellement cryptographique préventif	A cryptographic technique implemented in systemd-journald where signing keys change irreversibly over time, preventing retrospective log tampering.
Hardening	Durcissement / Blindage	The systematic process of securing a computer system by reducing its attack surface, eliminating unneeded components, and applying defensive configurations.
Linux Security Module (LSM)	Module de sécurité Linux	The official Linux kernel framework allowing mandatory access control architectures (SELinux, AppArmor, Smack, Yama) to mediate system calls.
Mandatory Access Control (MAC)	Contrôle d'accès obligatoire	A centrally governed security model enforced by the kernel where access rights are regulated by administrative policy rules rather than resource owners.

Multi-Factor Authentication (MFA)	Authentification multifacteur	An authentication process requiring two or more independent credentials (knowledge, possession, inherence) before granting system access.
Network-Bound Disk Encryption (NBDE)	Chiffrement lié au réseau	An enterprise architecture (e.g. Clevis & Tang) allowing encrypted storage volumes to be automatically unlocked at boot only when connected to a verified internal network.
Open Container Initiative (OCI)	Initiative pour les conteneurs ouverts	The governance structure establishing open industry standards for container image formats and low-level runtime execution (runc , crun).
Pluggable Authentication Modules (PAM)	Modules d'authentification enfichables	A modular system software architecture separating user authentication mechanisms from the applications that require authentication.
Principle of Least Privilege	Principe du moindre privilège	The foundational security design principle dictating that entities must be granted only the minimum permissions necessary to complete authorized tasks.
Return-Oriented Programming (ROP)	Programmation orientée retour	An advanced computer exploit technique where an attacker hijacks program execution flow by chaining together existing instruction sequences ("gadgets") in memory.
Rootless Container	Conteneur non privilégié	A container whose engine and internal processes execute entirely within unprivileged user namespaces without requiring root daemons or SUID binaries.
Seccomp (Secure Computing Mode)	Mode de calcul sécurisé	A Linux kernel facility that filters system calls using Berkeley Packet Filter (BPF) bytecode, terminating or denying any process attempting unauthorized calls.
Time-based One-Time Password (TOTP)	Mot de passe unique à validité temporelle	A standardized algorithm (RFC 6238) computing a dynamic one-time password from a shared secret key and the current Unix time interval.
User Namespace (usersns)	Espace de noms utilisateur	A Linux kernel virtualization feature

		isolating user and group ID mappings, allowing an account to possess root privileges inside a container while mapping to an unprivileged UID on the host.
--	--	---

14.4 LightPath Professional Services & Consulting Contact

Operating system hardening is the essential foundation upon which all enterprise application and database architectures depend. Implementing these guidelines guarantees compliance, eliminates lateral movement vectors, and significantly reduces risk.

LightPath provides elite consulting, architectural diagnostics, automated compliance deployments, and emergency incident response assistance:

- **Corporate Portal:** <https://www.lightpath.fr>
- **Direct Consulting Inquiries:** contact@lightpath.fr
- **Technical Assistance Hotline:** +33 (0)6 62 69 58 81
- **Headquarters:** LightPath, 30, rue du Bourg au Loup, 35140 Saint-Aubin-du-Cormier, France

End of Technical Reference Manual — Linux Hardening & Security Guide (Version 2.0)