ADMIN:DOCS

Linux Administration #7: Systemsicherheit und Härtung

Lerne die wichtigsten Aspekte der Linux-Sicherheit kennen: Von Systemhärtung über Verschlüsselung bis zur Sicherheitsüberwachung - dein Leitfaden für sichere Linux-Systeme.

Kategorie: Linux-Administration **Veröffentlicht:** 05.11.2024 **Zuletzt geprüft:** 02.11.2026

Willkommen zum siebten Teil unserer technischen Wiki-Serie über Linux-Administration!

Nachdem wir in den vorherigen Modulen die Grundlagen, die Benutzerverwaltung, das Prozess- und Ressourcenmanagement, die Netzwerkkonfiguration, das Shell-Scripting und Backup-Strategien erarbeitet haben, widmen wir uns heute einem der essenziellsten Themen im professionellen Serverbetrieb: der umfassenden Systemsicherheit und Härtung von Linux-Systemen.

Die Sicherheit eines Linux-Servers ist kein statischer Endzustand, sondern ein fortlaufender Prozess. In einer vernetzten Infrastruktur mit automatisierten Port-Scans, Credential-Stuffing und hochentwickelten Ransomware-Kampagnen reicht eine Standardinstallation nicht aus. Dieser Leitfaden führt dich Schritt für Schritt durch die Schichten der modernen Linux-Sicherheit – von PAM-Richtlinien und Post-Quantum-fähigem SSH über nftables-Firewalls bis hin zu Mandatory Access Control (AppArmor/SELinux), LUKS2-Festplattenverschlüsselung und Kernel-Auditing.

> ⚠ **Hinweis:** In diesem Artikel verwenden wir Ubuntu/Debian als primäre Beispiel-Distribution. Die grundlegenden Konzepte, Kernel-Parameter und Sicherheitsarchitekturen sind distributionsübergreifend identisch; spezifische Paketnamen, Pfade und MAC-Module

> (AppArmor vs. SELinux) werden an den entsprechenden Stellen vergleichend dargestellt.

Grundlegende Sicherheitskonzepte

Die Absicherung eines Linux-Systems basiert auf architektonischen Schutzprinzipien, die ineinandergreifen. Wer sich ausschließlich auf eine Firewall verlässt, scheitert, sobald ein Webdienst kompromittiert wird. Effektive Systemsicherheit folgt daher dem Modell von **Defense-in-Depth** (Tiefenstaffelung):

1. Die vier Grundpfeiler moderner Linux-Sicherheit

- **Defense-in-Depth (Mehrschichtige Absicherung):** Fällt eine Sicherheitsbarriere (z. B. ein fehlerhafter Dienst), verhindern nachgelagerte Barrieren (wie AppArmor/SELinux, unprivilegierte Benutzerkonten oder Kernel-Restriktionen) die vollständige Systemübernahme.
- **Prinzip der geringsten Rechte (Least Privilege):** Jeder Benutzer, Prozess und Daemon erhält exakt die minimal notwendigen Berechtigungen. Kein Netzwerkdienst darf standardmäßig mit Root-Rechten operieren.
- **Angriffsflächenminimierung (Attack Surface Reduction):** Nicht installierte Pakete enthalten keine ausnutzbaren Sicherheitslücken. Deaktiviere ungenutzte Systemd-Services, schließe offene Ports und blockiere ungenutzte Kernel-Module.
- **AAA-Architektur (Authentication, Authorization, Accounting):**
- *Authentifizierung:* Eindeutige Feststellung der Identität (Passwörter, Ed25519-SSH-Schlüssel, FIDO2/MFA).
- *Autorisierung:* Durchsetzung von Zugriffsrechten (POSIX-Rechte, ACLs, Sudo-Regeln, MAC-Policies).
- *Accounting:* Lückenlose und manipulationssichere Nachvollziehbarkeit aller administrativen Vorgänge über `auditd` und `journald`.

Bedrohungsszenarien im modernen Rechenzentrumsbetrieb

Um gezielte Härtungsmaßnahmen zu ergreifen, müssen Administratoren die typischen Angriffsvektoren verstehen:

1. Externe Angriffsvektoren

- **Automatisierte SSH-Brute-Force & Credential Stuffing:** Bots scannen IPv4- und IPv6-Bereiche permanent nach exponierten SSH-Diensten ab und testen geleakte Passwort-Listen.
- **Ransomware & Erpressungstrojaner:** Nach einem erfolgreichen Einstieg verschlüsseln Angreifer Datenbanken und Backup-Dateisysteme und exfiltrieren vertrauliche Daten.
- **Supply-Chain-Angriffe:** Kompromittierung von Drittanbieter-Repositories, Build-Pipelines oder Build-Skripten, die unbemerkt Schadcode in legitime Updates einschleusen.
- **DDoS-Attacken:** Überlastung von Bandbreite oder Socket-Tabellen durch SYN-Floods und Anwendungsüberlastung.

2. Lokale Privilege Escalation und interne Risiken

- **Fehlkonfigurierte Sudo-Berechtigungen:** Zu großzügig vergebene Rechte in `/etc/sudoers` (z. B. Ausführen von Editoren oder Interpretern mit Root-Rechten).
- **SUID-/SGID-Schwachstellen:** Veraltete oder unsichere Binärdateien mit gesetztem SUID-Bit erlauben lokalen Benutzern das Erlangen von Root-Rechten.
- **Mangelnde Rechte-Hygiene:** Weltweit beschreibbare Verzeichnisse (`777`), offene Konfigurationsdateien mit Datenbank-Passwörtern oder vergessene SSH-Private-Keys im Webroot.

Grundlegende Systemhärtung

Die Systemhärtung minimiert die Eintrittswahrscheinlichkeit und die Auswirkungen von Sicherheitsvorfällen direkt auf Betriebssystemebene.

1. Benutzer und Berechtigungen

a) Benutzerkonten überprüfen:

```
# Aktive Benutzer anzeigen
awk -F: '$3 >= 1000 && $7 != "/sbin/nologin" {print $1}' /etc/passwd

# Benutzer ohne Passwort finden
sudo awk -F: '($2 == "") {print $1}' /etc/shadow

# Überflüssige Systemkonten deaktivieren
sudo usermod -s /sbin/nologin unnötiger_benutzer
```

b) Restriktive Umask und Dateirechte:

Die Standard-Umask legt fest, welche Rechte neu erstellten Dateien und Verzeichnissen standardmäßig entzogen werden. Auf Mehrbenutzer- und Produktivservern sollte die Umask in `/etc/login.defs` und `/etc/profile` von permissiven `022` auf restriktive `027` (oder für reine Systemdienste `077`) angepasst werden:

```
# Aktuelle Umask anzeigen:
umask

# Umask 027 bedeutet:
# Neue Verzeichnisse: 750 (rwxr-x---) - Andere Benutzer haben keinerlei Zugriff
# Neue Dateien:       640 (rw-r-----) - Andere Benutzer können nicht lesen
```

2. Passwort-Policies und PAM-Härtung

Ein sicheres Authentifizierungssystem erzwingt widerstandsfähige Passwörter und blockiert Brute-Force-Angriffe zuverlässig auf PAM-Ebene (*Pluggable Authentication Modules*):

a) Passwortkomplexität mit `pam_pwquality`:

```
# PAM-Modul für Passwort-Qualität installieren:
sudo apt install libpam-pwquality

# Konfiguration anpassen (/etc/security/pwquality.conf):
sudo nano /etc/security/pwquality.conf
```

Empfohlene Konfiguration nach aktuellen Standards (BSI/NIST):

```
# Mindestens 14 Zeichen Länge:
minlen = 14
# Mindestens 3 verschiedene Zeichenklassen (Groß, Klein, Ziffern, Sonderzeichen):
minclass = 3
# Maximal 2 identische Zeichen in Folge:
maxrepeat = 2
# Mindestens 4 Zeichen Unterschied zum alten Passwort:
difok = 4
# Wörterbuch-Prüfung aktivieren:
dictcheck = 1
```

b) Passwort-Hashing: yescrypt als moderner Standard:

In modernen Linux-Distributionen (Debian 12+, Ubuntu 22.04+) ersetzt **yescrypt** das ältere SHA-512 (`$6$`). Yescrypt bietet durch Memory-Hardness massiven Schutz gegen hardwarebeschleunigte Offline-Angriffe (GPU/ASIC-Cracking):

```
# In /etc/pam.d/common-password sicherstellen:
password [success=1 default=ignore] pam_unix.so obscure yescrypt
```

c) Account Lockout mit `pam_faillock`:

Das veraltete Modul `pam_tally2` wurde aus modernen Linux-Systemen entfernt. Der aktuelle Standard zur Abwehr von Brute-Force-Attacken ist **`pam_faillock`**:

```
# Globale Einstellungen anpassen (/etc/security/faillock.conf):
sudo nano /etc/security/faillock.conf
```

```
# Nach 5 Fehlversuchen sperren:
deny = 5
# Zeitfenster für Fehlversuche (15 Minuten):
fail_interval = 900
# Sperrdauer (30 Minuten = 1800 Sekunden):
unlock_time = 1800
# Auch für Root-Konto überwachen (optional, erfordert Bedacht):
even_deny_root
```

```
# Gesperrte Konten prüfen und Status einsehen:
sudo faillock --user benutzer
```

```
# Gesperrten Benutzer manuell wieder entsperren:
sudo faillock --user benutzer --reset
```

3. Dienste-Minimierung und Port-Hygiene

Jeder aktive Netzwerkdienst vergrößert die potenzielle Angriffsfläche.

a) Aktive Daemons und Ports auditieren:

```
# Alle lauschenden TCP- und UDP-Sockets inklusive Prozessname anzeigen:
sudo ss -tulpn

# Laufende Systemd-Services auflisten:
systemctl list-units --type=service --state=running
```

b) Dienste stoppen, deaktivieren und maskieren:

Das Maskieren (`mask`) verhindert, dass ein Dienst unbeabsichtigt durch Abhängigkeiten oder Socket-Aktivierung gestartet wird:

```
# Dienst stoppen und Autostart abschalten:
sudo systemctl stop cups.service cups.socket
sudo systemctl disable cups.service cups.socket

# Dienst dauerhaft maskieren (Link auf /dev/null):
sudo systemctl mask cups.service
```

4. Kernel-Härtung über Sysctl

Der Linux-Kernel bietet zahlreiche Sicherheitsschalter gegen Spoofing, Denial-of-Service und Speichermanipulation. Lege eine gehärtete Konfiguration unter `/etc/sysctl.d/99-security.conf` an:

```
# Schutz vor Symlink- und Hardlink-Angriffen in Shared Directories (/tmp):
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
fs.protected_fifos = 2
fs.protected_regular = 2

# Kernel-Adressraum vor unprivilegierten Nutzern verbergen (ASLR-Schutz):
kernel.kptr_restrict = 2
kernel.dmesg_restrict = 1

# BPF-JIT-Compiler härten:
net.core.bpf_jit_harden = 2

# IP-Spoofing verhindern (Reverse Path Filtering):
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.all.rp_filter = 1

# SYN-Flood-Schutz aktivieren:
net.ipv4.tcp_syncookies = 1

# ICMP-Redirects ignorieren (Man-in-the-Middle-Schutz):
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
```

```
# Neue Sysctl-Einstellungen sofort ohne Neustart laden:
sudo sysctl --system
```

5. Sicheres Paketmanagement und automatische Updates

a) Deprecated `apt-key` ablösen – Moderne Schlüsselverwaltung:

Seit Debian 11 und Ubuntu 22.04 gilt `apt-key` als veraltet und unsicher, da Schlüssel systemweit für alle Repositories galten. Moderne Paketquellen verwenden dedizierte Keyrings in `/etc/apt/keyrings/`:

```
# Verzeichnis für Schlüssel anlegen:
sudo install -m 0755 -d /etc/apt/keyrings

# Signaturschlüssel sicher abrufen und de-armored speichern:
curl -fsSL https://example.com/repo.gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/example.gpg

# Repository gezielt an diesen Schlüssel binden (DEB822-Format):
echo "Types: deb
URIs: https://example.com/deb/
Suites: stable
Components: main
Signed-By: /etc/apt/keyrings/example.gpg" | sudo tee
/etc/apt/sources.list.d/example.sources
```

b) Automatische Sicherheitsupdates mit `unattended-upgrades`:

```
# Paket installieren:
sudo apt install unattended-upgrades update-notifier-common

# Konfiguration auf reine Sicherheitsupdates beschränken
(/etc/apt/apt.conf.d/50unattended-upgrades):
Unattended-Upgrade::Allowed-Origins {
    "${distro_id}:${distro_codename}-security";
```

```
};

# Automatischen Reboot bei Kernel-Patches planen (z. B. um 03:30 Uhr nachts):
Unattended-Upgrade::Automatic-Reboot "true";
Unattended-Upgrade::Automatic-Reboot-Time "03:30";
```

Netzwerksicherheit

Die Netzwerksicherheit bildet die äußere Verteidigungslinie eines Linux-Servers. Angriffe aus dem Internet treffen zuerst auf den Netzwerk-Stack des Kernels und exponierte Netzwerk-Sockets.

1. Firewall-Architektur: nftables und UFW

In modernen Linux-Distributionen ist **nftables** das Standard-Subsystem im Linux-Kernel für Paketfilterung und Network Address Translation (NAT). Es ersetzt die historische `iptables` / `ip6tables`-Architektur durch eine einheitliche, performantere Bytecode-Engine.

Auf Debian und Ubuntu dient **UFW (Uncomplicated Firewall)** als benutzerfreundliches Interface, das im Hintergrund native nftables-Regelsätze verwaltet:

```
# 1. Standard-Richtlinie: Alles Eingehende blockieren, Ausgehendes erlauben
sudo ufw default deny incoming
sudo ufw default allow outgoing

# 2. Rate-Limiting für SSH (Schutz vor simplen Brute-Force-Floods)
sudo ufw limit proto tcp from any to any port 22 comment 'SSH mit Rate-Limiting'

# 3. Gezielt Webserver-Ports freigeben
sudo ufw allow 80/tcp comment 'HTTP'
sudo ufw allow 443/tcp comment 'HTTPS'

# 4. Firewall aktivieren und Status detailliert prüfen
```

```
sudo ufw enable
sudo ufw status verbose
```

Für containerisierte Umgebungen (Docker, Podman, Kubernetes) oder komplexe Router-Szenarien empfiehlt sich die direkte Konfiguration über `/etc/nftables.conf`:

```
table inet filter {
    chain input {
        type filter hook input priority filter; policy drop;

        # Bestehende Verbindungen und Loopback erlauben:
        ct state established,related accept
        iif "lo" accept

        # Ungültige Pakete verwerfen:
        ct state invalid drop

        # ICMP/Ping zulassen (für Diagnose essenziell):
        ip protocol icmp accept
        ip6 nexthdr ipv6-icmp accept

        # SSH mit Verbindungsbegrenzung:
        tcp dport 22 ct state new limit rate 15/minute accept
    }
}
```

```
# nftables-Regeln aktivieren:
sudo systemctl enable --now nftables
sudo nft list ruleset
```

2. SSH-Härtung (OpenSSH 9.x/10.x & Post-Quantum-Sicherheit)

Der SSH-Dienst ist das primäre Einfallstor auf Linux-Servern. Moderne OpenSSH-Versionen unterstützen bereits hybride **Post-Quantum-Schlüsselaustauschverfahren** (`mlkem768x25519-sha256` bzw. `sntrup761x25519-sha512@openssh.com`), um verschlüsselten Datenverkehr gegen zukünftige Quantencomputer (*Harvest Now, Decrypt Later*) abzusichern.

> □□ **Praxis-Tipp:** Bearbeite auf modernen Distributionen nicht die Hauptdatei `/etc/ssh/sshd_config` , sondern lege eine dedizierte Drop-in-Datei `/etc/ssh/sshd_config.d/99-hardening.conf` an. Dadurch überstehen deine Einstellungen distributionsseitige Paket-Upgrades ohne Konflikte.

```
# /etc/ssh/sshd_config.d/99-hardening.conf
# 1. Authentifizierung strikt absichern
PermitRootLogin no
PasswordAuthentication no
KbdInteractiveAuthentication no
PubkeyAuthentication yes
AuthenticationMethods publickey

# 2. Begrenzung von Versuchen und Timeouts
MaxAuthTries 3
LoginGraceTime 20
ClientAliveInterval 300
ClientAliveCountMax 2

# 3. Unnötige Weiterleitungen unterbinden
X11Forwarding no
AllowAgentForwarding no
AllowTcpForwarding no

# 4. Nur moderne, sichere Krypto-Suiten und Post-Quantum KEX erzwingen
KexAlgorithms mlkem768x25519-sha256,sntrup761x25519-sha512@openssh.com,curve25519-
sha256,curve25519-sha256@libssh.org
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com
```

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

```
# SSH-Schlüsselpaar auf dem Client mit modernem Ed25519 generieren:
ssh-keygen -t ed25519 -a 100 -C "admin@firma.intern"

# Konfiguration auf Syntaxfehler prüfen vor dem Reload:
sudo sshd -t

# Dienst oder Socket neu starten (Ubuntu 24.04/26.04 nutzt standardmäßig ssh.socket):
sudo systemctl reload ssh || sudo systemctl restart ssh.socket
```

3. Intrusion Prevention mit Fail2ban (systemd-Journald-Integration)

Fail2ban überwacht Login-Fehlversuche und sperrt auffällige IP-Adressen dynamisch über die Kernel-Firewall. Da moderne Distributionen Authentifizierungsereignisse primär in `systemd-journald` speichern (und `/var/log/auth.log` teilweise entfällt), ist das `systemd`-Backend die robusteste Wahl:

```
# Fail2ban installieren:
sudo apt install fail2ban

# Lokale Jail-Konfiguration anlegen (/etc/fail2ban/jail.d/sshd.local):
sudo nano /etc/fail2ban/jail.d/sshd.local
```

```
[sshd]
```

```
enabled = true
port = 22
backend = systemd
maxretry = 3
findtime = 600
bantime = 3600
# Progressive Erhöhung bei wiederholten Sperren:
bantime.increment = true
bantime.factor = 2
```

```
# Fail2ban starten und Jail-Status überwachen:
sudo systemctl enable --now fail2ban
sudo fail2ban-client status sshd
```

4. Intrusion Detection & Integritätsprüfung (AIDE & Lynis)

- **Host-basierte Dateiintegrität (AIDE):** Erkennt, wenn Angreifer oder Rootkits Systemdateien (`/bin/login` , PAM-Module oder System-Binaries) manipulieren:

  ```
  sudo apt install aide
  # Initiale Integritätsdatenbank erzeugen und aktivieren:
  sudo aideinit
  sudo cp /var/lib/aide/aide.db.new /var/lib/aide/aide.db
  # Regelmäßige Integritätsprüfung (z. B. via systemd-Timer):
  sudo aide --check
  ```

- **Automatisiertes Sicherheits-Audit (Lynis):** Durchleuchtet das laufende Linux-System nach CIS-Benchmarks, fehlerhaften Dateirechten und offenen Konfigurationslücken:

```
sudo apt install lynis
sudo lynis audit system --quick
```

Zugriffskontrolle

Klassische POSIX-Dateiberechtigungen (Besitzer, Gruppe, Andere) reichen in komplexen Multi-Tier-Umgebungen nicht aus. Linux bietet leistungsfähige Mechanismen zur feingranularen Zugriffskontrolle.

1. Mandatory Access Control (MAC): AppArmor vs. SELinux

Das Standard-Linux-Rechtemodell ist *Discretionary Access Control* (DAC): Der Besitzer einer Datei bestimmt frei über die Rechte. Wird ein Dienst (z. B. Apache oder Nginx) kompromittiert, besitzt der Angreifer alle Rechte dieses Systemnutzers.

Mandatory Access Control (MAC) setzt darüber eine strikte, vom Kernel erzwungene Sicherheitsrichtlinie: Selbst wenn ein Prozess als Root läuft oder Eigentümer einer Datei ist, darf er nur Aktionen ausführen, die explizit in seinem Sicherheitskontext definiert sind.

AppArmor (Ubuntu, Debian, openSUSE)

Architektur & Bindung
: Pfadbasiert: Bindet Regeln an kanonische Dateipfade

Verwaltung & Lernkurve
: Sehr zugänglich; verständliche Profile in Textform

Zustandsmodi
: enforce (blockiert), complain (protokolliert nur)

Standard-Einsatz
: Schützt definierte Dienste (Nginx, MySQL, Evince)

SELinux (RHEL, Fedora, Rocky, Alma)

Architektur & Bindung

Typen- & Labelbasiert: Bindet Rechte an erweiterte Dateiattribute (xattr)

Verwaltung & Lernkurve

Hochgradig granular; erfordert tiefes Verständnis von SELinux-Typen

Zustandsmodi

enforcing (blockiert), permissive (protokolliert nur)

Standard-Einsatz

Schützt standardmäßig das gesamte Betriebssystem

a) AppArmor in der Praxis (Ubuntu/Debian):

```
# Aktuellen Status und geladene Profile anzeigen:
sudo aa-status

# Profil in den Lern-/Protokollmodus (Complain) versetzen:
sudo aa-complain /etc/apparmor.d/usr.sbin.nginx

# Profil in den scharfen Blockiermodus (Enforce) versetzen:
sudo aa-enforce /etc/apparmor.d/usr.sbin.nginx

# Neue Profile interaktiv aus Logs generieren:
sudo aa-genprof /usr/local/bin/custom_service
```

b) SELinux in der Praxis (RHEL/Fedora):

```
# SELinux-Status abfragen:
```

```
getenforce

# Sicherheitskontexte von Prozessen und Dateien anzeigen:
ls -Z /var/www/html
ps -eZ | grep nginx

# Kontext nach manueller Verschiebung reparieren:
sudo restorecon -Rv /var/www/html/

# SELinux-Booleans (Schalter) anpassen:
sudo setsebool -P httpd_can_network_connect 1
```

2. Erweiterte Dateirechte: POSIX-ACLs

Access Control Lists (ACLs) brechen die Beschränkung auf genau einen Benutzer und eine Gruppe auf. Sie erlauben es, beliebigen Nutzern oder Gruppen dedizierte Rechte zuzuweisen:

```
# Lese- und Schreibrechte für Entwicklerin 'anna' vergeben:
setfacl -m u:anna:rw- /srv/project/config.php

# Leserechte für die Gruppe 'auditors' vergeben:
setfacl -m g:auditors:r-- /srv/project/config.php

# Gesetzte ACLs und effektive Maske inspizieren:
getfacl /srv/project/config.php

# Default-ACL auf Verzeichnisebene setzen (wird an alle neuen Dateien vererbt):
setfacl -d -m g:developers:rwx /srv/project/

# Alle erweiterten ACLs rekursiv entfernen:
setfacl -R -b /srv/project/
```

3. Sudo-Härtung und Least Privilege

Gewähre Root-Rechte niemals pauschal, sondern restriktiv auf exakte Befehle beschränkt.

> ⚠ **Gefahr durch GTFOBins:** Befehle wie `vim`, `less`, `find`, `python` oder `awk` besitzen Shell-Escapes (z. B. `:!sh` in vi). Erlaubst du einem Nutzer `sudo vim /etc/hosts`, kann dieser innerhalb von Vim eine vollwertige Root-Shell öffnen. Verwende `sudoedit` statt `sudo vim`.

Best Practice: Drop-in-Dateien unter `/etc/sudoers.d/`:

Bearbeite Konfigurationsdateien niemals direkt mit einem Standardeditor, sondern stets über `visudo`:

```
# Sichere Bearbeitung einer isolierten Sudoers-Datei:
sudo visudo -f /etc/sudoers.d/developers
```

```
# /etc/sudoers.d/developers
# 1. Alias für sichere Administrationsbefehle definieren
Cmnd_Alias WEB_OPS = /usr/bin/systemctl restart nginx, /usr/bin/systemctl reload nginx,
/usr/bin/systemctl status nginx

# 2. Reines Dateibearbeiten ohne Shell-Escape über sudoedit
Cmnd_Alias EDIT_WEB = sudoedit /var/www/html/index.html

# 3. Gruppe Entwickler berechtigen (Passwort wird verlangt):
%developers ALL=(ALL) WEB_OPS, EDIT_WEB

# 4. Globale Sudo-Sicherheitsoptionen:
Defaults logfile="/var/log/sudo.log"
Defaults passwd_timeout=1
Defaults timestamp_timeout=15
Defaults !rootpw
Defaults env_reset
```

4. Sicherheits-Logging & Kernel-Auditing (`auditd`)

Während `syslog` und `journald` reguläre Systemmeldungen sammeln, protokolliert das Kernel-Audit-Subsystem (`auditd`) sicherheitsrelevante Systemaufrufe (Syscalls) manipulationssicher und unbestechlich:

```
# auditd installieren und starten:
sudo apt install auditd audispd-plugins
sudo systemctl enable --now auditd

# Regel definieren: Änderungen an /etc/passwd und /etc/shadow protokollieren:
sudo auditctl -w /etc/passwd -p wa -k identity_changes
sudo auditctl -w /etc/shadow -p wa -k identity_changes

# Ausführung des execve-Syscalls durch unprivilegierte Nutzer überwachen:
sudo auditctl -a always,exit -F arch=b64 -S execve -F auid>=1000 -k command_execution

# Dauerhafte Regeln in /etc/audit/rules.d/audit.rules hinterlegen:
# -w /etc/shadow -p wa -k identity_changes
```

```
# Audit-Logs gezielt nach Schlüssel auswerten:
sudo ausearch -k identity_changes --format text

# Zusammenfassenden Sicherheitsbericht generieren:
sudo aureport --auth
```

Verschlüsselung

Verschlüsselung schützt vertrauliche Daten sowohl im Ruhezustand (*Data at Rest*) gegen physischen Diebstahl oder nicht autorisierte Speicher-Zugriffe als auch während der

Übertragung (*Data in Transit*) über unsichere Netzwerke.

1. Block- und Festplattenverschlüsselung: LUKS2 & TPM 2.0

Der Standard für blockbasierte Festplattenverschlüsselung unter Linux ist **LUKS2** (*Linux Unified Key Setup Version 2*). Im Gegensatz zu älteren Versionen nutzt LUKS2 die speicherintensive Schlüsselableitungsfunktion **Argon2id**, die resistent gegen Angriffe mit GPUs und ASICs ist:

```
# Partition mit LUKS2 und Argon2id verschlüsseln:
sudo cryptsetup luksFormat --type luks2 --pbkdf argon2id /dev/nvme0n1p3

# Verschlüsseltes Gerät öffnen (erzeugt virtuelles Device unter /dev/mapper/):
sudo cryptsetup open /dev/nvme0n1p3 crypt_data

# Dateisystem auf dem entschlüsselten Container anlegen und einhängen:
sudo mkfs.ext4 -L secure_data /dev/mapper/crypt_data
sudo mount /dev/mapper/crypt_data /srv/secure/

# Container nach getaner Arbeit aushängen und sperren:
sudo umount /srv/secure/
sudo cryptsetup close crypt_data
```

Headless Server automatisiert entsperren mit TPM 2.0 (`systemd-cryptenroll`):

Auf modernen Servern mit TPM 2.0-Sicherheitschip kann die Entschlüsselung beim Systemstart an den hardwarebasierten Integritätsstatus (PCR-Register) gekoppelt werden, ohne dass ein Administrator manuell ein Passwort eingeben muss:

```
# LUKS2-Schlüssel sicher im TPM 2.0 Chip binden:
sudo systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs=0+2+7 /dev/nvme0n1p3
```

```
# In /etc/crypttab hinterlegen:
# crypt_data UUID=... none tpm2-device=auto
```

2. Dateisystem- und Verzeichnisverschlüsselung: fscrypt

Historische FUSE-Werkzeuge wie `encfs` gelten seit Jahren als kryptografisch kompromittiert, und `ecryptfs` wird in modernen Distributionen schrittweise abgelöst. Der moderne Standard für die native Verzeichnisverschlüsselung auf Dateisystemebene (ext4 und f2fs) ist **`fscrypt`**:

```
# fscrypt installieren und Dateisystem vorbereiten:
sudo apt install fscrypt libpam-fscrypt
sudo fscrypt setup
sudo tune2fs -O encrypt /dev/nvme0n1p2

# Verzeichnis gezielt verschlüsseln:
mkdir /srv/user_data
fscrypt encrypt /srv/user_data --source=pam_passphrase
```

3. Transportverschlüsselung (TLS 1.3 & moderne PKI)

Für alle Netzwerkdienste (Web, Mail, APIs) ist **TLS 1.3** mit *Perfect Forward Secrecy* (PFS) zwingend:

```
# Privaten Schlüssel (Ed25519 oder RSA 4096) und CSR für Zertifizierungsstelle generieren:
openssl req -new -newkey rsa:4096 -nodes -keyout /etc/ssl/private/server.key \
    -out /etc/ssl/certs/server.csr -subj "/CN=server01.firma.intern"

# Let's Encrypt / Certbot für automatisierte Zertifikate und automatische Erneuerung:
sudo apt install certbot python3-certbot-nginx
```

```
sudo certbot --nginx -d server01.firma.intern --agree-tos -m admin@firma.intern
```

4. Schlüssel- und Zertifikatsmanagement

- **GPG-Master- und Sub-Keys:** GPG-Schlüssel für administrative Signaturen mit Passphrase sichern und auf getrennten Offline-Medien archivieren.
- **Dateirechte für Secrets:**

```
# SSH-Private-Keys strikt schützen:
chmod 700 ~/.ssh
chmod 600 ~/.ssh/id_ed25519
chmod 644 ~/.ssh/id_ed25519.pub

# SSL-Zertifikatsschlüssel vor fremden Diensten abschirmen:
chmod 600 /etc/ssl/private/*.key
```

Incident Response: Reaktion auf Sicherheitsvorfälle

Wurde ein Server kompromittiert oder besteht der dringende Verdacht auf unberechtigten Zugriff, verhindert ein strukturierter Incident-Response-Prozess voreilige Fehler und die Zerstörung forensischer Beweise:

1. **1. Detektion & Verifikation** Alarme verifizieren, Angriffsvektor identifizieren und Incident-Kategorie festlegen
2. **2. Eindämmung & Isolation** Netzwerkverbindung trennen (Netzkabel/VLAN-Drop), verdächtige Prozesse anhalten

3. **3. Forensik & Beweissicherung**RAM-Dump anfertigen, Festplatten-Image (dd) sichern, System-Logs unverändert exportieren
4. **4. Bereinigung & Patching**Schwachstelle patchen, Hintertüren entfernen, alle SSH-Keys und Passwörter rotieren
5. **5. Wiederanlauf & Post-Mortem**System aus verifizierten Backups restarten, Monitoring verschärfen, Abschlussbericht erstellen

Best Practices & Sicherheits-Checkliste

Als Leitfaden für die kontinuierliche Systempflege dient die folgende Übersicht:

- **Täglich / Kontinuierlich:**
- Automatisierte Fehlversuchs-Überwachung mit Fail2ban und `pam_faillock`.
- Statusprüfungen von kritischen Diensten und offenen Ports (`ss -tulpn`).
- **Wöchentlich:**
- Sicherheitsupdates einspielen und veraltete Pakete entfernen (`unattended-upgrades`).
- AIDE-Integritätsberichte auf unerwartete Dateiänderungen prüfen.
- **Monatlich / Quartalsweise:**
- Umfassendes Sicherheitsaudit mit `lynis audit system`.
- Überprüfung der Sudoers-Berechtigungen und inaktiver Benutzerkonten.
- Disaster-Recovery- und Restore-Test der verschlüsselten Backups.

Übung: Sicherheitsüberwachung & Live-Auditing

Szenario: Du übernimmst die Administration eines Webservers. Dein Ziel ist es, fehlerhafte SSH-Anmeldeversuche über das System-Journal zu überwachen, ein Sicherheits-Audit mit Lynis durchzuführen und verdächtige offene Ports aufzudecken.

Aufgaben

1. Untersuche alle fehlgeschlagenen SSH-Anmeldeversuche der letzten Stunde über `journalctl`.
2. Identifiziere alle aktuell lauschenden Netzwerk-Sockets inklusive der zugehörigen Prozesse.
3. Führe einen schnellen Sicherheits-Audit mit Lynis durch.
4. Erstelle ein kompaktes Shell-Skript, das das Journal auf Angriffsversuche prüft.

Mögliche Lösung

1. SSH-Fehlversuche über systemd-journald analysieren

```
# Fehlgeschlagene Logins der letzten 60 Minuten herausfiltern:
sudo journalctl -u ssh -u ssh.socket --since "1 hour ago" | grep -i "Failed password"
```

2. Offene Ports und Prozesse auditieren

```
# Alle lauschenden TCP- und UDP-Sockets kompakt auflisten:
sudo ss -tulpn
```

3. Lokales Sicherheits-Audit mit Lynis starten

```
# Audit durchführen und Härtungs-Score ermitteln:
sudo lynis audit system --quick
```

4. Monitoring-Skript für fehlgeschlagene Authentifizierungen

```
#!/usr/bin/env bash
# /usr/local/sbin/check_ssh_bruteforce.sh
set -Eeuo pipefail

readonly THRESHOLD=20
readonly COUNT=$(journalctl -u ssh -u ssh.socket --since "15 minutes ago" | grep -c -i
"Failed password" || true)

if (( COUNT >= THRESHOLD )); then
    echo "[ALERT] Auffällige Brute-Force-Aktivität: ${COUNT} Fehlversuche in den letzten
15 Minuten!" | \
        mail -s "SSH Security Alert: $(hostname)" admin@firma.intern
fi
```

Befehlsreferenz (Cheatsheet)

Für den schnellen Zugriff bei der Absicherung und Systemhärtung fasst die folgende Referenz die wichtigsten Linux-Kommandos zusammen:

UFW-Firewall aktivieren Aktiviert die Uncomplicated Firewall mit Standard-Policies

$ `sudo ufw enable`

SSH-Port mit Rate-Limiting freigeben Öffnet Port 22 und blockiert wiederholte Verbindungsversuche

$ `sudo ufw limit 22/tcp`

Aktiven Firewall-Status anzeigen Listet alle konfigurierten Firewall-Regeln und Statusinformationen auf

$ `sudo ufw status verbose`

Fail2ban Jail-Status überwachen Zeigt gesperrte IP-Adressen und Fehlversuche des SSH-Jails an

$ `sudo fail2ban-client status sshd`

Gesperrten Benutzer wieder freigeben Setzt Fehlversuchszähler von pam_faillock für den Nutzer zurück

$ `sudo faillock --user benutzer --reset`

Lauschende Ports und Sockets anzeigen Listet alle aktiven Netzwerkports mit zugehörigen Daemons auf

$ `sudo ss -tulpn`

Dateiberechtigungen mit ACL erweitern Weist einem Benutzer spezifische POSIX-Zugriffsrechte zu

$ `sudo setfacl -m u:anna:rw- /pfad/datei`

Erweiterte POSIX-ACLs auslesen Zeigt Standardrechte, benutzerdefinierte ACLs und die Maske an

$ `sudo getfacl /pfad/datei`

Sudoers-Konfiguration sicher bearbeiten Validiert Syntax vor dem Speichern zur Vermeidung von Aussperrungen

$ `sudo visudo -f /etc/sudoers.d/custom`

Passwort-Alterung und Ablauf prüfen Zeigt detaillierte Ablaufzeiten und Richtlinien eines Benutzers an

$ `sudo chage -l benutzer`

Partition mit LUKS2 und Argon2id verschlüsseln Formatiert Blockgerät mit modernem Speicher-hartem Schlüsselsetup

$ `sudo cryptsetup luksFormat --type luks2 --pbkdf argon2id /dev/sdX`

LUKS-Container entschlüsseln und öffnen Erzeugt zugängliches Block-Device unter /dev/mapper/daten

$ `sudo cryptsetup open /dev/sdX daten`

LUKS2-Partition an TPM 2.0 binden Ermöglicht automatische Entschlüsselung über Sicherheitschip

$ `sudo systemd-cryptenroll --tpm2-device=auto /dev/sdX`

Syscall-Überwachung mit auditd einrichten Protokolliert alle schreibenden Zugriffe auf sensible Dateien

$ `sudo auditctl -w /etc/shadow -p wa -k identity_changes`

Audit-Logs nach Suchbegriff filtern Durchsucht manipulationssichere Kernel-Logs nach definierten Tags

$ `sudo ausearch -k identity_changes --format text`

Umfassenden Lynis-Sicherheitscheck ausführen Analysiert Systemkonfiguration, Ports und Rechte nach Best Practices

$ `sudo lynis audit system --quick`

Dateiintegrität mit AIDE prüfen Vergleicht aktuellen Dateistatus mit der signierten Prüfsummen-DB

$ `sudo aide --check`

Weiterführende Ressourcen

Die folgenden Leitfäden, Spezifikationen und internen Kursmodule vertiefen die behandelten Sicherheits- und Härtungsstrategien:

CIS Distribution Independent Linux Benchmark Offizieller Industriestandard des Center for Internet Security für Linux-Härtung

OpenSSH Security and Release Notes Offizielle Dokumentation zu KEX-Algorithmen, Post-Quantum und SSH-Sicherheit

Ubuntu Security Guide Sicherheitsleitfaden und Compliance-Dokumentation für Ubuntu Server

Debian Securing Manual Offizielles Handbuch zur Härtung und Absicherung von Debian-Systemen

Lynis Documentation & Benchmarks Vollständige Dokumentation zum Lynis Security-Auditing-Framework

Linux Administration #6: Datensicherung Das vorherige Modul: Backup-Strategien, rsync, BorgBackup und Disaster Recovery

Linux Administration #8: Virtualisierung Das nachfolgende Modul: KVM, QEMU, libvirt und LXC-Container-Management

Fazit

Die Sicherheit eines Linux-Systems ist kein einmaliger Zustand, sondern ein kontinuierlicher Prozess aus Härtung, restriktiver Rechtevergabe und proaktivem Monitoring. In diesem Modul hast du gelernt, wie du dein System durch Firewalls (UFW), Intrusion Prevention (Fail2ban), Least-Privilege-Sudo-Regeln, ACLs und moderne Verschlüsselungsverfahren zuverlässig absicherst.

Im nächsten Teil unserer Masterclass widmen wir uns der Virtualisierung und Containerisierung:

Weiter geht es mit: Linux Administration #8: Virtualisierung und VM-Management

Zur Kursübersicht: Alle Linux-Administrations-Artikel & Module