ADMIN:DOCS

Arch Linux: Systemhärtung und Sicherheit – Best Practices

Tiefgehende Systemhärtung für Arch Linux: PAM-Policies, Sudo-Hardening, SUID/Capabilities, Pacman-Integrität, LUKS2 mit Argon2id, fstab-Flags, nftables-Ruleset, SSH-Krypto und Journald-FSS.

Kategorie: Arch Linux Serie **Veröffentlicht:** 15.11.2025 **Zuletzt geprüft:** 27.08.2026

Nachdem die Basisinstallation und das Paketmanagement eingerichtet sind, läuft Arch Linux als schlankes System – standardmäßig jedoch ohne nennenswerte Barrieren gegen Fehlkonfigurationen oder Angriffe von außen. Arch verfolgt das *KISS-Prinzip* (Keep It Simple, Stupid), was bedeutet: Sicherheit ist kein integrierter Automatismus, sondern liegt vollständig in der Verantwortung des Administrators.

Minimalismus allein schützt nicht vor Kompromittierung: Ein System ohne restriktive Paketfilterung, mit unbeschränkten Sudo-Privilegien oder unverschlüsselten Block-Devices bietet dieselben Angriffsflächen wie jede andere Standard-Distribution. In einer Rolling-Release-Umgebung kommen ständige Software-Änderungen hinzu, die saubere Paketverifikationen und manipulationssichere Protokollierung erfordern.

Dieser Leitfaden ist Teil unserer Arch-Linux-Serie und etabliert das **operative Sicherheitsfundament** für Workstations und Server. Wir adressieren vier reale Angriffsvektoren:

1. **Identität & Rechte:** Privilegienminimierung via `wheel`, Sudo-Hardening, Linux Capabilities und PAM-Sicherheitsmodule (`pam_faillock`, `libpwquality`).
2. **Supply Chain & Paketintegrität:** Kryptografische Signaturpflicht in Pacman, GPG-Web-of-Trust, verifizierte HTTPS-Mirrors und sichere AUR-Builds in Clean-Chroots.
3. **Data at Rest & Dateisystem:** LUKS2-Vollverschlüsselung mit Argon2id, restriktive Mount-Flags (`noexec`, `nosuid`, `nodev`, `hidepid`), und Kernel-Dateisystemschutz (`sysctl`).

Quelle: https://admindocs.de/de/arch-linux-serie/arch-linux-systemhaertung-sicherheit-best-practices

4. **Netzwerk-Exposition & Schnittstellen:** Stateful Firewalling mit `nftables` (inkl. Connection Tracking & Rate-Limiting), SSH-Kryptohärtung nach modernem Standard, Fail2Ban-Integration und kryptografisches Journal-Sealing (FSS).

> **Voraussetzungen & Scope:** Ein lauffähiges Arch-System mit Root- oder Sudo-Zugang. Dieser Artikel behandelt die grundlegende Härtung auf Betriebssystem- und Netzwerkebene. Weiterführende Mechanismen wie Mandatory Access Control (AppArmor), Kernel-Hardening (`linux-hardened`) und Container-Isolation bauen im Folgeartikel direkt auf diesem Fundament auf.

Schichtenmodell der Systemhärtung

Sicherheit folgt dem Prinzip der tiefengestaffelten Verteidigung (*Defense in Depth*). Schlägt eine Sicherheitsmaßnahme fehl, fängt die darunterliegende Schicht den Angriff ab:



1. Benutzerverwaltung, PAM & Privilegienminimierung

Das primäre Sicherheitsziel auf Betriebssystemebene lautet: Direkte Logins als `root` unterbinden, administrative Aufgaben über dedizierte Benutzerkonten delegieren und Rechte auf das absolute Minimum beschränken.

Unprivilegierte Benutzer und die `wheel`-Gruppe

Unter Arch Linux existiert standardmäßig keine Gruppe namens `sudo`. Administrative Rechte werden traditionell über die Gruppe `wheel` delegiert.

Erstelle einen neuen administrativen Benutzer, falls noch nicht geschehen:

```
# Benutzer mit Home-Verzeichnis anlegen und der wheel-Gruppe zuweisen
sudo useradd -m -G wheel -s /bin/bash adminuser

# Starkes Passwort vergeben
sudo passwd adminuser
```

Überprüfe die Gruppenzugehörigkeit:

```
id adminuser
```

Die Ausgabe bestätigt die UID und die Mitgliedschaft in der Gruppe `wheel`:

```
uid=1001(adminuser) gid=1001(adminuser) groups=1001(adminuser),998(wheel)
```

Sperre den direkten Root-Account nach erfolgreicher Einrichtung des Sudo-Zugriffs gegen Passwort-Logins:

```
# Root-Passwort sperren (Login nur noch via sudo oder SSH-Keys falls erlaubt)
sudo passwd -l root
```

Sudo-Konfiguration mit `visudo` härten

Bearbeite `/etc/sudoers` niemals mit einem regulären Texteditor, sondern ausschließlich über `visudo`. `visudo` sperrt die Datei gegen gleichzeitiges Schreiben und validiert die Syntax vor dem Speichern, um fatale Aussperrungen zu verhindern.

```
sudo visudo
```

Aktiviere administrative Rechte für Mitglieder der Gruppe `wheel` und setze restriktive Defaults:

```
# Defaults für erhöhte Sicherheit
Defaults env_reset
```

```
Defaults mail_badpass
Defaults secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
Defaults timestamp_timeout=10
Defaults passwd_tries=3
Defaults use_pty
Defaults logfile="/var/log/sudo.log"
Defaults log_input, log_output

# Mitglieder der wheel-Gruppe dürfen mit Passwort-Authentifizierung alle Befehle ausführen
%wheel ALL=(ALL:ALL) ALL
```

- `timestamp_timeout=10`: Der Sudo-Cache verfällt nach 10 Minuten Inaktivität.
- `passwd_tries=3`: Maximal drei Passwortversuche vor dem Abbruch.
- `use_pty`: Erzwingt die Ausführung von Sudo-Befehlen in einem Pseudo-Terminal (Schutz gegen Angriffe via Terminal-Injection).
- `log_input, log_output`: Protokolliert ausgeführte Befehle sowie deren I/O-Ströme auditierbar in `/var/log/sudo-io/`.

PAM-Stack härten: Brute-Force-Sperrung & Passwortqualität

Unter Arch Linux steuert PAM (Pluggable Authentication Modules) sämtliche Authentifizierungsabläufe. Wir konfigurieren zwei wesentliche Module:

1. `pam_faillock`: Sperrt Konten nach wiederholten Fehlversuchen (Schutz gegen Offline- und lokale Brute-Force-Attacken).
2. `pam_pwquality`: Erzwingt kryptografisch robuste Passwörter.

Installiere `libpwquality`:

```
sudo pacman -S libpwquality
```

Konfiguriere die globalen Qualitätsrichtlinien in `/etc/security/pwquality.conf`:

```
# Mindestlänge für Passwörter
minlen = 14

# Mindestanzahl unterschiedlicher Zeichenklassen (Groß, Klein, Ziffern, Sonderzeichen)
minclass = 3

# Maximale Wiederholung identischer aufeinanderfolgender Zeichen
maxrepeat = 2

# Maximal erlaubte Zeichenfolge aus dem Benutzernamen
maxsequence = 3

# Wörterbuch-Prüfung gegen bekannte Leaks
gecoscheck = 1
```

Teste ein Passwort interaktiv mit `pwscore`:

```
pwscore
# Eingabeaufforderung: Gib dein Testpasswort ein
```

Konfiguriere `pam_faillock` in `/etc/security/faillock.conf`:

```
# Nach 4 Fehlversuchen sperren
deny = 4

# Fehlversuchsfenster: 10 Minuten (600 Sekunden)
fail_interval = 600
```

```
# Sperrdauer: 15 Minuten (900 Sekunden)
unlock_time = 900

# Auch für lokale TTY- und SSH-Sessions erzwingen
even_deny_root = false
```

Um einen gesperrten Benutzer vor Ablauf der Sperrfrist manuell freizuschalten:

```
# Status der Fehlversuche prüfen
faillock --user adminuser

# Sperre manuell zurücksetzen
faillock --user adminuser --reset
```

Root-Wechsel einschränken mit `pam_wheel`

Standardmäßig kann jeder Benutzer mit Kenntnis des Root-Passworts `su -` ausführen. Beschränke den Zugriff auf `su` strikt auf Mitglieder der `wheel`-Gruppe in `/etc/pam.d/su`:

```
# In /etc/pam.d/su einkommentieren:
auth required pam_wheel.so use_uid
```

SUID- und SGID-Binaries auditieren

Dateien mit gesetztem SUID-Bit (`Set User ID`, Oktalwert `4000`) laufen mit den Rechten des Dateieigentümers (`root`), unabhängig davon, wer sie ausführt. Veraltete oder unnötige SUID-Binaries sind ein primäres Ziel für Privilege-Escalation-Exploits.

Ermittle alle SUID/SGID-Dateien auf dem System:

```
# SUID-Dateien auflisten
sudo find / -perm -4000 -type f -exec ls -la {} + 2>/dev/null

# SGID-Dateien auflisten
sudo find / -perm -2000 -type f -exec ls -la {} + 2>/dev/null
```

Typische legitime SUID-Programme sind `/usr/bin/sudo`, `/usr/bin/passwd`, `/usr/bin/su` und `/usr/bin/pkexec`. Wenn dein System Tools enthält, die für reguläre Benutzer nicht benötigt werden (z. B. Netzwerk-Tools wie `traceroute` oder alte Helpers), entziehe das SUID-Bit:

```
# SUID-Bit entfernen
sudo chmod u-s /usr/bin/chfn
sudo chmod u-s /usr/bin/chsh
```

Linux Capabilities statt SUID nutzen

Moderne Linux-Systeme erlauben es, Prozessen feingranulare Kernel-Rechte (*Capabilities*) zuzuweisen, anstatt ihnen über SUID pauschal volle Root-Rechte zu gewähren.

```
# Aktuell gesetzte Dateisystem-Capabilities anzeigen
sudo getcap -r /usr/bin 2>/dev/null
```

Beispiel: Soll ein Tool Netzwerk-Sockets binden dürfen, ohne Root zu sein, genügt `CAP_NET_BIND_SERVICE`:

```
# Capability statt SUID setzen
sudo setcap 'cap_net_bind_service=+ep' /pfad/zum/binary
```

2. Paketmanagement, Repositories & AUR-Hygiene

In einer Rolling-Release-Distribution wie Arch Linux ist die Integrität der Softwarequellen fundamental. Manipulierte Pakete oder kompromittierte Mirrors dürfen unter keinen Umständen unbemerkt auf das System gelangen.

Strenge GPG-Signaturprüfung in `pacman.conf`

Pacman nutzt GPG-Signaturen zur Verifikation der Paket- und Datenbankintegrität. Öffne `/etc/pacman.conf` und stelle sicher, dass Signaturen ausnahmslos erzwungen werden:

```
[options]
# Paketsignaturen müssen zwingend valide sein
SigLevel           = Required DatabaseOptional
LocalFileSigLevel  = Optional

# Für maximale Sicherheit auf allen Repositories:
[core]
SigLevel = PackageRequired DatabaseOptional

[extra]
SigLevel = PackageRequired DatabaseOptional

[multilib]
SigLevel = PackageRequired DatabaseOptional
```

Initialisiere und aktualisiere den offiziellen Arch-Schlüsselbund:

```
# Schlüsselbund initialisieren und mit Arch-Master-Keys befüllen
sudo pacman-key --init
sudo pacman-key --populate archlinux

# Schlüsselbund aktualisieren
sudo pacman -Sy archlinux-keyring
```

Sollte ein Schlüssel abgelaufen oder ungültig sein, erzwinge einen Refresh:

```
# GPG-Keys über Keyserver aktualisieren
sudo pacman-key --refresh-keys
```

> **Praxistipp – Keine Partial Upgrades:** Führe niemals `pacman -Sy <paket>` ohne `-u` durch. Ein unvollständiges Upgrade synchronisiert nur die Paketdatenbank und installiert neuere Abhängigkeiten, während der Rest des Systems auf altem Stand bleibt. Dies bricht dynamische Bibliotheken (*Shared Objects*) und führt zu Systeminstabilitäten. Verwende ausnahmslos `pacman -Syu`.

Sichere HTTPS-Mirrors mit `reflector` generieren

Paket-Downloads sollten ausschließlich über verschlüsselte HTTPS-Mirrors laufen, um Man-in-the-Middle-Angriffe auf Metadaten zu verhindern:

```
# reflector installieren
sudo pacman -S reflector
```

Generiere eine sortierte, aktuelle Mirrorliste für Deutschland über HTTPS:

```
sudo reflector \
  --country Germany \
  --latest 15 \
  --protocol https \
  --sort rate \
  --save /etc/pacman.d/mirrorlist
```

Automatisiere diesen Vorgang über den mitgelieferten systemd-Timer:

```
sudo systemctl enable --now reflector.timer
```

AUR-Sicherheit: Isolierte Builds in Clean-Chroots

Das Arch User Repository (AUR) enthält unmoderierte, von der Community bereitgestellte `PKGBUILD`-Skripte. Ein bösartiges `PKGBUILD` oder `.install`-Skript läuft standardmäßig mit den Rechten des ausführenden Benutzers.

Best Practices für den AUR-Einsatz:

1. **Kein blindes Ausführen von AUR-Helpern (`yay`, `paru`):** Überprüfe vor jedem Build den Inhalt des `PKGBUILD` und eventuell beiliegender `.install`-Dateien auf verdächtige

Netzwerkzugriffe (`curl` , `wget`) oder `chmod` -Befehle.

2. **AUR-Builds in Clean-Chroot isolieren:** Baue AUR-Pakete mit den offiziellen `devtools` in einer isolierten Chroot-Umgebung.

```
# devtools installieren
sudo pacman -S devtools

# Saubere Build-Chroot anlegen
mkdir -p ~/chroot
mkarchroot ~/chroot/root base-devel

# Paket in der isolierten Chroot bauen
cd ~/mein-aur-paket/
makechrootpkg -c -r ~/chroot
```

Dadurch kann der Build-Prozess nicht auf Dateien in deinem Benutzerverzeichnis zugreifen und hinterlässt keine Build-Reste auf dem Hostsystem.

CVE- und Sicherheits-Audits für Pakete: `arch-audit`

Mit `arch-audit` prüfst du installierte Pakete direkt gegen die offizielle Sicherheitsdatenbank der Arch Linux Security Admins:

```
# arch-audit installieren
sudo pacman -S arch-audit
```

Führe den Audit aus:

```
arch-audit
```

Beispielausgabe bei bekannten Schwachstellen:

```
Package openssh is affected by CVE-2024-6387. High severity! Update to 9.8p1-1.
```

Integriere `arch-audit` als automatisierten Pacman-Hook in `/etc/pacman.d/hooks/90-arch-audit.hook`:

```
[Trigger]
Operation = Upgrade
Operation = Install
Operation = Remove
Type = Package
Target = *

[Action]
Description = Prüfe installierte Pakete auf bekannte Sicherheitslücken (CVEs)...
When = PostTransaction
Exec = /usr/bin/arch-audit
```

Cache-Bereinigung mit `paccache`

Pacman speichert standardmäßig alle heruntergeladenen Paketversionen in `/var/cache/pacman/pkg/`. Um Speicherplatz freizugeben, ohne die Rollback-Fähigkeit für Notfälle zu verlieren:

```
# pacman-contrib installieren
sudo pacman -S pacman-contrib

# Behalte exakt die letzten 2 Versionen jedes installierten Pakets
sudo paccache -r -k 2

# Entferne alle Versionen deinstallierter Pakete
sudo paccache -ruk0
```

Automatisiere die wöchentliche Bereinigung über den systemd-Timer:

```
sudo systemctl enable --now paccache.timer
```

3. Dateisystem- & Speichersicherheit

Die Härtung auf Dateisystem-Ebene verhindert, dass Schadcode aus beschreibbaren Verzeichnissen ausgeführt wird oder physischer Diebstahl zu Klartext-Datenverlust führt.

Restriktive Mount-Optionen in `/etc/fstab`

Verzeichnisse wie `/tmp` oder `/var/tmp` sind weltbeschreibbar (`mode 1777`). Ohne Schutzmaßnahmen kann ein Angreifer Executables dort ablegen und starten.

Konfiguriere `/tmp` als isoliertes `tmpfs` und setze restriktive Flags in `/etc/fstab`:

```
# /etc/fstab Härtungskonfiguration

# Root-Dateisystem
UUID=11111111-1111-1111-1111-111111111111   /           ext4
defaults,noatime                      0 1

# EFI System Partition
UUID=2222-2222                               /boot       vfat
defaults,noexec,nosuid,nodev,umask=0077 0 2

# /tmp im RAM mit noexec, nosuid und nodev
tmpfs                                        /tmp        tmpfs
defaults,noexec,nosuid,nodev,mode=1777,size=2G 0 0

# /var/tmp mit noexec
UUID=33333333-3333-3333-3333-333333333333   /var/tmp    ext4
defaults,noexec,nosuid,nodev         0 2

# /home (nosuid und nodev)
UUID=44444444-4444-4444-4444-444444444444   /home       ext4
defaults,nosuid,nodev                0 2
```

- `noexec`: Verhindert die direkte Ausführung von Binaries auf dem Dateisystem.
- `nosuid`: Ignoriert SUID- und SGID-Bits auf dieser Partition.
- `nodev`: Verhindert die Interpretation von Block- und Character-Devices.
- `noatime`: Deaktiviert die Aktualisierung des Zugriffszeitstempels (spart I/O-Last und SSD-Verschleiß).

Wende Mount-Änderungen im laufenden Betrieb ohne Neustart an:

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

Prozess-Isolation mit `hidepid` auf `/proc`

Standardmäßig kann jeder lokale Benutzer mit `ps aux` oder `top` die Befehlszeilenargumente aller anderen Prozesse auf dem System einsehen – inklusive eventuell übergebener Tokens oder Passwörter.

Über die Mount-Option `hidepid=invisible` (bzw. `hidepid=2`) sieht jeder Benutzer ausschließlich seine eigenen Prozesse. Mitglieder einer definierten Gruppe (z. B. `wheel` oder Monitoring-Dienste) werden über `gid=` ausgenommen:

```
# In /etc/fstab ergänzen:
proc    /proc    proc
defaults,nosuid,nodev,noexec,relatime,hidepid=invisible,gid=wheel   0 0
```

Remount durchführen:

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

Festplattenverschlüsselung mit LUKS2 & Argon2id

Für mobile Endgeräte und sensitive Server ist eine Vollverschlüsselung mit LUKS2 (Linux Unified Key Setup) Pflicht. Sie schützt *Data at Rest* vor unbefugtem Zugriff bei Diebstahl des Datenträgers.

Formatiere die Partition mit moderner PBKDF `argon2id`:

```
# Partition mit LUKS2 initialisieren
sudo cryptsetup luksFormat \
  --type luks2 \
  --cipher aes-xts-plain64 \
  --key-size 512 \
  --hash sha512 \
  --pbkdf argon2id \
  --iter-time 3000 \
  /dev/nvme0n1p3
```

Öffne den verschlüsselten Container und erstelle das Dateisystem:

```
sudo cryptsetup open /dev/nvme0n1p3 cryptroot
sudo mkfs.ext4 -L root /dev/mapper/cryptroot
```

Prüfe die Header-Informationen:

```
sudo cryptsetup luksDump /dev/nvme0n1p3
```

Sichere den LUKS-Header zwingend an einem sicheren externen Ort:

```
sudo cryptsetup luksHeaderBackup /dev/nvme0n1p3 --header-backup-file /pfad/zu/luks-header-backup.img
```

Kernel-Schutz für Symlinks und Hardlinks (`sysctl`)

Böswillige symbolische Links in weltbeschreibbaren Verzeichnissen ermöglichen Time-of-Check-to-Time-of-Use-Angriffe (TOCTOU). Hinterlege hardening-relevante Kernel-Parameter in `/etc/sysctl.d/50-security.conf`:

```
# Schutz gegen Symlink- und Hardlink-Exploits in weltbeschreibbaren Verzeichnissen
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
fs.protected_fifos = 2
fs.protected_regular = 2

# Verhindert unberechtigte Core-Dumps privilegierter Prozesse
fs.suid_dumpable = 0

# Speicher-Härtung: Ptrace-Scope einschränken (nur direkte Elternprozesse)
kernel.yama.ptrace_scope = 2

# Kernel-Pointer im /proc-Dateisystem maskieren
kernel.kptr_restrict = 2

# dmesg-Zugriff für unprivilegierte Benutzer sperren
kernel.dmesg_restrict = 1

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

Wende die Parameter sofort an:

```
sudo sysctl --system
```

4. Netzwerksicherheit & Firewall mit `nftables`

Ein gehärtetes System bietet Angreifern im Netzwerk keine Angriffsfläche. Jeder offene TCP- oder UDP-Port ist ein potenzielles Einfallstor.

Offene Ports identifizieren

Überprüfe vor der Firewall-Konfiguration alle aktiven Listener auf dem System:

```
sudo ss -tulpn
```

Erläuterung der Parameter:

- `-t`: TCP-Sockets
- `-u`: UDP-Sockets
- `-l`: Nur lauschende Ports (*Listening*)
- `-p`: Zugehörigen Prozessnamen und PID anzeigen
- `-n`: Numerische Ports (keine DNS-/Service-Namensauflösung)

Beende und deaktiviere alle Dienste, die nicht zwingend benötigt werden:

```
# Ungenutzten Dienst beenden und abschalten
sudo systemctl stop avahi-daemon.service
sudo systemctl disable avahi-daemon.service
```

`nftables`-Konfiguration: Stateful & Default Drop

`nftables` ist das moderne Subsystem des Linux-Kernels zur Paketfilterung. Es ersetzt `iptables`, arbeitet ressourceneffizienter und bietet eine strukturierte, performante Regelsyntax.

Installiere `nftables`:

```
sudo pacman -S nftables
```

Erstelle ein restriktives, produktionsreifes Basis-Ruleset in `/etc/nftables.conf`:

```
#!/usr/sbin/nft -f

flush ruleset

table inet filter {
    # Definierte Sets für vertrauenswürdige Management-IPs (optional)
    set trusted_mgmt {
        type ipv4_addr
        flags interval
        elements = { 192.168.1.0/24, 10.0.0.5 }
    }
```

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

        # 1. Bereits etablierte und verwandte Verbindungen erlauben (Stateful)
        ct state established,related accept

        # 2. Ungültige Pakete sofort verwerfen
        ct state invalid drop

        # 3. Loopback-Interface vollständig erlauben
        iif "lo" accept

        # 4. TCP-Flag-Prüfung (Schutz vor Portscans / XMAS-Paketen)
        tcp flags & (fin | syn | rst | psh | ack | urg) == 0 drop
        tcp flags & (fin | syn) == fin | syn drop
        tcp flags & (syn | rst) == syn | rst drop

        # 5. ICMP / ICMPv6 (Rate-Limited gegen Flood-Angriffe)
        ip protocol icmp icmp type { echo-request, echo-reply, destination-unreachable,
time-exceeded } limit rate 5/second accept
        ip6 nexthdr icmpv6 icmpv6 type {
            echo-request, echo-reply, destination-unreachable, packet-too-big,
            time-exceeded, parameter-problem, nd-router-solicit, nd-router-advert,
            nd-neighbor-solicit, nd-neighbor-advert
        } limit rate 5/second accept

        # 6. SSH mit Rate-Limiting gegen Brute-Force (Port 22)
        tcp dport 22 ct state new limit rate 10/minute burst 5 packets accept

        # 7. Optional: Webserver (HTTP / HTTPS)
        # tcp dport { 80, 443 } ct state new accept

        # 8. Unberechtigte Pakete protokolliert verwerfen (optional für Debugging)
        limit rate 3/minute log prefix "nft-drop: " level warn
    }

    chain forward {
        type filter hook forward priority filter; policy drop;
    }

    chain output {
        type filter hook output priority filter; policy accept;
    }
}
```

Validiere die Syntax der Konfigurationsdatei vor der Aktivierung:

```
sudo nft -c -f /etc/nftables.conf
```

Regeln laden und Dienst dauerhaft aktivieren:

```
sudo nft -f /etc/nftables.conf
sudo systemctl enable --now nftables.service
```

Überprüfe das aktive Ruleset im Kernel:

```
sudo nft list ruleset
```

5. SSH-Server-Härtung

SSH ist die zentrale Administrationsschnittstelle auf Linux-Servern. Fehlkonfigurationen (z. B. erlaubte Passwort-Logins) führen auf öffentlich erreichbaren IP-Adressen innerhalb weniger Minuten zu automatisierten Brute-Force-Angriffen.

Schlüsselbasierte Authentifizierung (Ed25519)

Erstelle auf deinem Client-Rechner ein modernes Ed25519-Schlüsselpaar, falls noch nicht vorhanden:

```
ssh-keygen -t ed25519 -a 100 -C "adminuser@arch-workstation"
```

Übertrage den öffentlichen Schlüssel auf deinen Arch-Server:

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub adminuser@deine-server-ip
```

Strikte `sshd`-Konfiguration

Erstelle eine dedizierte Härtungskonfiguration unter `/etc/ssh/sshd_config.d/99-hardening.conf`:

```
# Direkten Root-Login vollständig deaktivieren
PermitRootLogin no

# Passwort-Authentifizierung abschalten (ausschließlich Schlüssel)
PasswordAuthentication no
KbdInteractiveAuthentication no
AuthenticationMethods publickey

# Strenge Authentifizierungsversuche
MaxAuthTries 3
MaxSessions 3
LoginGraceTime 30

# X11- und Forwarding-Optionen deaktivieren
X11Forwarding no
AllowAgentForwarding no
AllowTcpForwarding no
```

```
# Re-Keying nach 1GB Traffic oder 1 Stunde erzwingen
RekeyLimit 1G 1h

# Moderne Kryptografie-Suites (Mozilla Modern / BSI Standard)
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

# Nur Ed25519-Hostkeys verwenden (RSA deaktivieren sofern alle Clients aktuell)
HostKey /etc/ssh/ssh_host_ed25519_key

# Banner unterdrücken (keine OS-Informationen leaken)
PrintMotd no
DebianBanner no

# Zugriff auf bestimmte Gruppen beschränken
AllowGroups wheel
```

Validiere die SSHD-Konfiguration syntaktisch vor dem Neustart:

```
sudo sshd -t
```

> ⚠ **Aussperrschutz:** Schließe deine aktuelle SSH-Sitzung nach dem Reload nicht sofort! Öffne ein **zweites Terminalfenster** und teste den Login mit deinem Schlüssel: `ssh -i ~/.ssh/id_ed25519 adminuser@deine-server-ip` Erst wenn der Login im zweiten Terminal sauber funktioniert, ist die Konfiguration sicher aktiv.

Lade den SSH-Daemon neu:

```
sudo systemctl reload sshd.service
```

Brute-Force-Abwehr mit `fail2ban`

`fail2ban` überwacht Logdateien auf fehlgeschlagene Login-Versuche und blockiert angreifende IP-Adressen temporär über `nftables`.

```
# fail2ban installieren
sudo pacman -S fail2ban
```

Erstelle eine lokale Konfiguration `/etc/fail2ban/jail.local`:

```
[DEFAULT]
bantime   = 1h
findtime  = 10m
maxretry  = 4
banaction = nftables-multiport
backend   = systemd

[sshd]
enabled   = true
port      = ssh
mode      = aggressive
maxretry  = 3
bantime   = 24h
```

Aktiviere und starte den Dienst:

```
sudo systemctl enable --now fail2ban.service
```

Status der SSH-Jail und geblockte IPs prüfen:

```
sudo fail2ban-client status sshd
```

6. Protokollierung, Manipulationsschutz & Auditing

Ein gehärtetes System muss Anomalien und Angriffe nachvollziehbar protokollieren, damit Sicherheitsvorfälle zeitnah analysiert werden können.

`systemd-journald` persistent und manipulationssicher konfigurieren

Das systemd-Journal protokolliert Kernel- und Dienstnachrichten. Konfiguriere `/etc/systemd/journald.conf`:

```
[Journal]
# Permanente Speicherung unter /var/log/journal
Storage=persistent

# Maximale Speicherbegrenzung für Logs
SystemMaxUse=1G
SystemKeepFree=2G
```

```
# Forward Secure Sealing (FSS) aktivieren
Seal=yes

# Log-Rate-Limiting gegen Denial-of-Service
RateLimitIntervalSec=30s
RateLimitBurst=1000
```

Forward Secure Sealing (FSS) aktivieren

Forward Secure Sealing (FSS) schützt Protokolldateien kryptografisch gegen nachträgliche Manipulation. Selbst wenn ein Angreifer Root-Rechte erlangt, kann er vergangene Log-Einträge nicht unbemerkt verändern oder löschen.

Initialisiere FSS:

```
sudo journalctl --setup-keys
```

Die Ausgabe liefert einen **Verifikationsschlüssel** (`Verification Key`):

```
The sealing key is stored in: /var/log/journal/xxxxxxxxxxxxxxxxxxxx/fss
The verification key is: 3a7f-9b2c-4d1e-8f0a-...
```

> **Sicherheitshinweis:** Notiere und speichere den Verifikationsschlüssel zwingend an einem sicheren, **externen** Ort (z. B. Passwort-Manager). Auf dem Server verbleibt nur der Sealing-Key, der sich alle 15 Minuten kryptografisch weiterdreht.

Überprüfe die Integrität der Logdateien regelmäßig:

```
journalctl --verify --verify-key="3a7f-9b2c-4d1e-8f0a-..."
```

Erwartetes Ergebnis bei unversehrten Protokollen:

```
PASS: /var/log/journal/xxxxxxxxxxxxxxxxxxxx/system.journal
PASS: /var/log/journal/xxxxxxxxxxxxxxxxxxxx/user-1001.journal
```

Umfassende Sicherheits-Audits mit `lynis`

`lynis` scannt Konfigurationen, Kernel-Parameter, Dateirechte und installierte Pakete auf Schwachstellen:

```
# lynis installieren
sudo pacman -S lynis
```

Führe einen vollständigen System-Scan durch:

```
sudo lynis audit system
```

Der Scan liefert einen **Hardening Index** (z. B. `78/100`) sowie konkrete Handlungsempfehlungen:

```
# Warnungen und Vorschläge aus dem Log filtern
sudo grep -E "WARNING|SUGGESTION" /var/log/lynis.log
```

Best-Practice-Checkliste

Bereich	Maßnahme	Konfigurationsdatei / Tool
Identität	Unprivilegierter User + `wheel`	`useradd -m -G wheel`, `/etc/sudoers`
Sudo-Härtung	Timeout & I/O-Logging	`visudo` (`timestamp_timeout=10`, `use_pty`)
Passwörter	Mindestlänge 14 Zeichen & Komplexität	`/etc/security/pwquality.conf`
Brute-Force Lokal	Account-Sperre nach 4 Fehlversuchen	`/etc/security/faillock.conf`
Paketmanagement	Strikte GPG-Signaturprüfung	`/etc/pacman.conf` (`SigLevel = Required`)
Mirrors	HTTPS-Mirrors mit automatischem Update	`reflector.timer`, `/etc/pacman.d/mirrorlist`
Dateisystem	`noexec`, `nosuid` auf `/tmp` & `/var/tmp`	`/etc/fstab`
Prozesse	Versteckte fremde Prozesse	`proc` Mount mit `hidepid=invisible`
Festplatte	LUKS2 mit Argon2id PBKDF	`cryptsetup luksFormat --type luks2`
Kernel	Symlink-/Hardlink-Schutz	`/etc/sysctl.d/50-security.conf`
Firewall	Default Drop Stateful Firewall	`nftables.service`, `/etc/nftables.conf`
SSH-Server	Ed25519-Keys only, Root aus, moderne Ciphers	`/etc/ssh/sshd_config.d/99-hardening.conf`
Netzwerk-Abwehr	Automatische IP-Sperre bei SSH-Angriffen	`fail2ban.service` (`nftables-multiport`)

Bereich	Maßnahme	Konfigurationsdatei / Tool
Logging	Manipulationssicheres Journal (FSS)	`systemd-journald`, `journalctl --setup-keys`
Auditing	CVE- & System-Prüfung	`arch-audit`, `lynis audit system`

Weiterführende Ressourcen

Arch Linux: Fortgeschrittene Sicherheitsfeatures und Wartung Linux Server Härtung: FIDO2, SSH-Sicherheit und CrowdSec Pacman: Der umfassende Leitfaden Offizielles Arch Linux Wiki – Security Best Practices nftables Dokumentation (Arch Wiki)

Fazit

Sicherheit unter Linux ist kein einzelner Schalter, den man umlegt, sondern ein vielschichtiges Gesamtkonzept (*Defense in Depth*). Indem wir die Angriffsfläche auf allen Ebenen minimiert haben – von strengen PAM-Regeln und restriktiven Dateisystem-Mounts über eine kompromisslose Stateful Firewall bis hin zu kryptografisch versiegelten Logs via Forward Secure Sealing (FSS) –, steht dein System auf einem extrem robusten Fundament.

Selbst wenn ein Angreifer eine einzelne Hürde überwindet, verhindern nachgelagerte Mechanismen wie `hidepid`, restriktive Sudo-Richtlinien und strikte SSH-Konfigurationen die Ausbreitung im Netzwerk oder die unbemerkte Rechteausweitung.

> **Praxis-Tipp:** Führe nach größeren Systemänderungen oder Paket-Installationen regelmäßig einen Lynis-Scan durch (`sudo lynis audit system`). So erkennst du neu entstandene Sicherheitslücken oder zurückgesetzte Standardberechtigungen sofort.

Im sechsten und abschließenden Teil unserer Serie widmen wir uns den Profiligen der Systemhärtung: In Arch Linux: Fortgeschrittene Sicherheitsfeatures und Wartung implementieren wir Mandatory Access Control mit AppArmor, wechseln auf den gehärteten `linux-hardened`-Kernel, isolieren Container-Netzwerke via nftables, bauen ein zentrales TLS-Logging-Netzwerk auf und etablieren eine 3-2-1-Backup-Strategie.

Quelle: https://admindocs.de/de/arch-linux-serie/arch-linux-systemhaertung-sicherheit-best-practices