ADMIN:DOCS

Arch Linux: Fortgeschrittene Sicherheitsfeatures und Wartung

AppArmor, Kernel-Härtung mit linux-hardened, GRUB-Passwörter, Container-Isolation mit nftables, Multi-Host-Logging und BTRFS-Snapshots – fortgeschrittene Sicherheitsmaßnahmen für Arch Linux im Praxiseinsatz.

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

Du hast im letzten Artikel dieser Serie bereits die Grundlagen der Systemhärtung umgesetzt: Benutzerverwaltung mit minimierten Root-Rechten, Pacman-Signaturen für Paketintegrität, nftables-Grundregeln, SSH auf Schlüssel-Basis und Fail2Ban gegen Brute-Force. Das blockiert bereits einen Großteil der üblichen Angriffsvektoren. Aber: Die Maßnahmen arbeiten auf derselben Schicht wie ein potenzieller Angreifer – auf Kernel- und Userspace-Ebene, ohne dass ein zusätzlicher Sicherheitslayer dazwischen steht.

Genau hier setzt dieser Artikel an. Er baut direkt auf dem auf, was du im vorherigen Beitrag umgesetzt hast, und führt dein Arch-System von „solide gehärtet" in Richtung „langfristig wartbar, isoliert und widerstandsfähig". Die Themen greifen den Ausblick des letzten Artikels auf: Mandatory Access Control mit AppArmor, Kernel-Härtung mit `linux-hardened` und restriktiven Sysctl-Parametern, GRUB-Passwörter gegen physische Angriffe, erweiterte nftables-Regeln für Container-Isolation, zentrale Log-Auswertung für Multi-Host-Setups und Backup-Strategien mit BTRFS-Snapshots. Dazu kommen regelmäßige Audits mit `checksec` und `lynis`, damit du den Sicherheitszustand deines Systems jederzeit messbar und reproduzierbar dokumentieren kannst.

> **Voraussetzungen:** Du solltest die Grundlagen der Systemhärtung, nftables-Regeln und Pacman aus den vorherigen Artikeln der Serie kennen. Die Befehle in diesem Artikel sind getestet auf Arch Linux mit aktuellem Standard- sowie linux-hardened-Kernel.

Quelle: https://admindocs.de/de/arch-linux-serie/arch-linux-fortgeschrittene-sicherheitsfeatures-und-wartung

Mandatory Access Control mit AppArmor

Warum AppArmor auf Arch Linux

Linux kennt zwei prinzipielle Ansätze für Mandatory Access Control: AppArmor arbeitet pfadbasiert, SELinux labelbasiert. Der Unterschied ist nicht nur akademisch. Bei AppArmor entscheidet der Dateipfad, ob ein Prozess auf eine Ressource zugreifen darf. Bei SELinux ist das ein Security-Label, das an Dateien, Prozessen und Ports hängt und unabhängig vom Pfad existiert.

Auf Arch Linux ist AppArmor der pragmatische Weg. Es liegt in den offiziellen Repositories vor, funktioniert mit allen unterstützten Kerneln und erfordert keine umfangreichen Paket-Anpassungen. Der AppArmor-Kernel-Code ist seit Linux 2.6.36 im Mainstream enthalten und wird aktiv gepflegt. Die Userspace-Tools sind stabil und gut dokumentiert.

SELinux hingegen ist auf Arch nur als AUR-Paket verfügbar. Um es vollständig zu betreiben, müsstest du mehrere Dutzend Pakete – `coreutils`, `systemd`, `openssh`, `sudo`, `util-linux`, `iproute2`, `findutils`, `logrotate`, `pam`, `shadow`, `psmisc` – mit SELinux-Flags neu bauen. Dazu kommt eine deutlich steilere Lernkurve: SELinux arbeitet mit Security-Labels auf Dateien, Prozessen und Ports, die über Extended Attributes (`xattr`) gespeichert werden. Die Reference Policy für Arch ist nicht offiziell unterstützt.

> ⚠ **Klare Entscheidung:** AppArmor ist auf Arch der Standard. SELinux ist machbar, aber nur für Administratoren, die ein Unsupported-Setup pflegen und bei jedem Paket-Update prüfen wollen, ob Rebuilds nötig sind.

Wie AppArmor im Kernel arbeitet

AppArmor ist ein Linux Security Module (LSM). Das bedeutet: Der Kernel ruft AppArmor-Hooks auf, bevor ein Prozess auf eine Ressource zugreift. AppArmor prüft dann anhand des geladenen Profils, ob der Zugriff erlaubt ist.

Der Ablauf im Detail:



Wichtig: AppArmor prüft den **Dateipfad**, nicht die Datei-Inode. Das bedeutet, eine harte Verknüpfung (Hardlink) auf dieselbe Datei wird als separater Pfad behandelt. Die Prüfung erfolgt zum Zeitpunkt des `exec()`-Aufrufs. Ein bereits laufender Prozess wird nicht mehr nachträglich eingeschränkt, es sei denn, das Profil wird explizit neu geladen.

Installation und Aktivierung

```
# AppArmor installieren
sudo pacman -S apparmor
```

AppArmor muss als Linux Security Module im Kernel aktiviert werden. Dafür setzt du den `lsm=`-Kernelparameter in der GRUB-Konfiguration:

```
sudo nano /etc/default/grub
```

```
# In GRUB_CMDLINE_LINUX_DEFAULT hinzufügen:
GRUB_CMDLINE_LINUX_DEFAULT="loglevel=3 quiet
lsm=landlock,lockdown,yama,integrity,apparmor,bpf"
```

Die Reihenfolge im `lsm=`-Parameter ist entscheidend. `apparmor` muss das erste „major" Modul sein. Die anderen Module sind kleine, spezialisierte LSMs:

LSM	Funktion
`landlock`	Unprivilegierte Sandbox (Kernel 5.13+)
`lockdown`	Kernel-Integrität schützen (Secure Boot)
`yama`	Ptrace-Einschränkung
`integrity`	Datei-Integritätsprüfung (IMA)

Quelle: https://admindocs.de/de/arch-linux-serie/arch-linux-fortgeschrittene-sicherheitsfeatures-und-wartung

LSM	Funktion
`apparmor`	Mandatory Access Control (Hauptmodul)
`bpf`	BPF-Programmzugriff kontrollieren

```
# GRUB neu generieren
sudo grub-mkconfig -o /boot/grub/grub.cfg

# AppArmor beim Start laden und aktivieren
sudo systemctl enable --now apparmor.service
```

Verifikation:

```
# 1. Prüfen, ob AppArmor als aktives LSM geladen ist
cat /sys/kernel/security/lsm
# Erwartete Ausgabe: landlock,lockdown,yama,integrity,apparmor,bpf

# 2. Prüfen, ob das Kernelmodul geladen ist
lsmod | grep apparmor

# 3. Status des systemd-Dienstes prüfen
systemctl status apparmor.service
```

Fehlt `apparmor` in der Ausgabe von `cat /sys/kernel/security/lsm`, wurde der Kernelparameter nicht korrekt übernommen. Prüfe `/etc/default/grub` und führe `sudo grub-mkconfig -o /boot/grub/grub.cfg` erneut aus.

Profile verstehen

AppArmor schränkt Prozesse über Profile ein – Textdateien in `/etc/apparmor.d/`, die festlegen, welche Dateien, Netzwerk-Ressourcen, System-Calls und Capabilities ein Prozess nutzen darf. Jedes Profil hat einen von drei Modi:

Modus	Kernel-Verhalten	Anwendungsverhalten
`enforce`	Regelverstöße werden geblockt, Kernel generiert AVC-Audit-Log	Anwendung erhält `EACCES` oder `EPERM`
`complain`	Regelverstöße werden nur geloggt, nicht geblockt	Anwendung verhält sich, als gäbe es kein Profil
`unconfined`	Keine Einschränkungen durch AppArmor	Prozess hat vollen Zugriff (nur DAC-Regeln)

Geladene Profile anzeigen:

```
sudo aa-status
```

Beispielhafte Statusausgabe:

```
apparmor module is loaded.
38 profiles are loaded.
38 profiles are in enforce mode.
   /usr/bin/cat
   /usr/bin/gpg
   /usr/sbin/cron
   ...
0 profiles are in complain mode.
```

Auf einem frischen Arch-System mit installiertem `apparmor` sind typischerweise bereits Standardprofile aktiv, die Tools wie `ping` oder `dhclient` einschränken.

Profil-Syntax im Detail

Ein AppArmor-Profil besteht aus mehreren Abschnitten:

Dateizugriffs-Regeln:

Die Buchstaben nach dem Pfad definieren die erlaubten Operationen:

Zeichen	Berechtigung	Entspricht
`r`	Lesen	`open(O_RDONLY)`, `read()`
`w`	Schreiben	`open(O_WRONLY|O_RDWR)`, `write()`, `truncate()`
`a`	Anhängen	`open(O_APPEND)`
`l`	Hardlink erstellen	`link()`
`k`	Lock (Dateisperre)	`flock()`, `fcntl(F_SETLK)`
`m`	mmap ausführen	`mmap(PROT_EXEC)`
`ix`	Inheritiertes Exec	Prozess wird mit dem gleichen Profil ausgeführt
`Px`	Discretes Exec	Prozess wird mit seinem eigenen Profil ausgeführt
`Cx`	Child Exec	Neuer Prozess bekommt ein Kind-Profil
`ux`	Unconfined Exec	Prozess verliert alle AppArmor-Einschränkungen

Beispiel – Dateiregeln mit Pfad-Globbing:

```
# Nur lesbar
/etc/nginx/nginx.conf r,
```

```
# Lesen und Schreiben in allen Unterordnern
/var/log/nginx/** rw,

# Ausführbar (mit inherited Profil)
/usr/bin/nginx ix,

# mmap mit Exec-Berechtigung (für Shared Libraries)
/usr/lib/libssl.so* mr,
```

Netzwerk-Regeln:

```
# IPv4 TCP (Stream)
network inet stream,

# IPv6 TCP
network inet6 stream,

# IPv4 UDP (Datagram)
network inet dgram,

# IPv6 UDP
network inet6 dgram,

# Bestimmte Ports einschränken
network inet stream, peer=(label=nginx),
```

Capability-Regeln:

Linux Capabilities sind die feingranulare Alternative zu vollen Root-Rechten. AppArmor kann Capabilities gezielt erlauben oder verbieten:

```
# Capabilities erlauben
capability net_bind_service,    # Port < 1024 binden
capability setuid,              # UID wechseln
capability setgid,              # GID wechseln

# Capabilities explizit verbieten (deny-Regel)
deny capability sys_admin,     # Keine Admin-Capabilities
deny capability sys_rawio,     # Keinen direkten Hardware-Zugriff
```

D-Bus-Regeln:

Für Dienste, die über D-Bus kommunizieren (z. B. NetworkManager, Avahi):

```
# D-Bus System-Bus Kommunikation
dbus (send) bus=system path=/org/freedesktop/NetworkManager,
dbus (receive) bus=system path=/org/freedesktop/NetworkManager,

# D-Bus Signale begrenzen
dbus (send) bus=system path=/org/freedesktop/NetworkManager
    interface=org.freedesktop.NetworkManager
    member=StateChanged,
```

Signal- und Ptrace-Regeln:

```
# Signals an andere Prozesse senden
signal (send) set=(term, kill) peer=/usr/sbin/nginx,

# Ptrace (Debugging) einschränken
```

```
    ptrace (read, trace) peer=/usr/bin/gdb,
```

Abstractions – wiederverwendbare Regel-Sets:

Abstractions sind vordefinierte Sammlungen von Regeln, die typische Zugriffsmuster abdecken und Redundanzen vermeiden:

```
# Häufig genutzte Abstractions einbinden
#include <abstractions/base>           # Grundlegende Systemzugriffe
#include <abstractions/nameservice>    # DNS-Auflösung, /etc/passwd
#include <abstractions/ssl_certs>      # Zertifikate lesen
#include <abstractions/apache2-common> # Typische Webserver-Regeln
#include <abstractions/nvidia>         # Nvidia-Treiber-Zugriff
```

Eigene Abstractions können unter `/etc/apparmor.d/abstractions/` hinterlegt werden:

```
# /etc/apparmor.d/abstractions/my-webapp
#include <abstractions/base>
#include <abstractions/nameservice>

/var/webapp/** r,
/var/webapp/uploads/ rw,
/run/webapp/*.sock rw,
```

Tunables und Variablen:

```
# Variablen-Definition (Standard in /etc/apparmor.d/tunables/home)
@{HOME}=@{HOMEDIRS}/* /root/

# In Profilen nutzbar:
owner @{HOME}/.config/** r,
```

`@{HOMEDIRS}` ist eine vordefinierte Variable, die `/home` enthält. `owner`-Regeln greifen nur, wenn der Prozess die Datei mit der passenden UID besitzt.

Ein Profil mit `aa-genprof` erstellen

`aa-genprof` beobachtet über das Linux Audit-Subsystem, welche Ressourcen ein Prozess tatsächlich nutzt, und generiert interaktiv passende Profilregeln.

```
# AppArmor-Utilities installieren
sudo pacman -S apparmor-utils

# Profil für Nginx interaktiv generieren
sudo aa-genprof nginx
```

Der Generator startet, und du wirst aufgefordert, `nginx` in einem zweiten Terminal zu starten und typische Anfragen zu senden (statische Dateien laden, Reverse-Proxy aufrufen, Logs schreiben). AppArmor beobachtet jede `open()`, `connect()`, `mmap()`- und `capability`-Anfrage und schlägt die passende Profilregel vor.

> □□ **aa-genprof ist der sicherste Weg** zum ersten Profil: Es gibt kein Raten über notwendige Zugriffe – du siehst exakt, was die Anwendung zur Laufzeit benötigt, und

> bestätigst jede Regel einzeln.

Ein manuelles Profil erstellen

Für maximale Kontrolle schreibst du das Profil selbst. Hier ein produktionsnahes Profil für Nginx in `/etc/apparmor.d/usr.bin.nginx`:

```
#include <tunables/global>

/usr/bin/nginx {
  #include <abstractions/base>
  #include <abstractions/nameservice>
  #include <abstractions/ssl_certs>

  # ── Netzwerk ──────────────────────────────
  network inet stream,
  network inet6 stream,

  # ── Binary ────────────────────────────────
  /usr/bin/nginx mr,

  # ── Konfiguration ─────────────────────────
  /etc/nginx/** r,
  /etc/nginx/mime.types r,

  # ── Web-Root ──────────────────────────────
  /var/www/** r,
  /var/www/html/** rw,

  # ── Logs & PID ────────────────────────────
  /var/log/nginx/** w,
  /run/nginx.pid w,

  # ── Temporäre Dateien ─────────────────────
  /var/lib/nginx/** rw,
  /tmp/** w,
```

```
  # ── Unix-Sockets ──────────────────────────
  unix stream,

  # ── Capabilities ──────────────────────────
  capability net_bind_service,
  capability setuid,
  capability setgid,

  # ── Verbotene Zugriffe ────────────────────
  deny /etc/shadow r,
  deny /etc/gshadow r,
  deny capability sys_admin,
}
```

Profil laden und aktivieren:

```
sudo apparmor_parser -r /etc/apparmor.d/usr.bin.nginx
```

Verifikation:

```
sudo aa-status | grep nginx
# Ausgabe: /usr/bin/nginx
```

Complain-Modus zum Testen

Bevor ein Profil in Produktion geht, wird es im Complain-Modus getestet.

Verstöße werden protokolliert, aber nicht blockiert:

```
# Profil in Complain-Modus versetzen
sudo aa-complain /usr/bin/nginx
```

Führe typische Anfragen gegen nginx aus und prüfe die Denials im Journal:

```
sudo journalctl -k -g "apparmor"
```

Erst wenn keine unerwarteten Verstöße mehr geloggt werden, wechselst du in den produktiven Enforce-Modus:

```
# Profil scharfschalten
sudo aa-enforce /usr/bin/nginx
```

> ⚠ **Complain-Modus ist kein Schutz:** Er dient ausschließlich dem Profil-Profiling. In produktiven Umgebungen müssen Profile im Enforce-Modus laufen.

AVC-Audit-Messages verstehen

Wenn AppArmor einen Zugriff blockiert, generiert der Kernel eine AVC-Meldung (*Access Vector Cache*):

```
audit: type=1400 audit=1724764800.123:456
  apparmor="DENIED"
  operation="open"
  profile="/usr/bin/nginx"
  name="/etc/nginx/conf.d/secrets.conf"
  pid=1234 comm="nginx"
  requested_mask="r"
  denied_mask="r"
  fsuid=33
  ouid=33
```

Feld	Bedeutung
`apparmor="DENIED"`	Zugriff wurde geblockt (oder `ALLOWED` bei Complain)
`operation="open"`	Versuchter System-Call
`profile="/usr/bin/nginx"`	Aktives AppArmor-Profil
`name="/etc/..."`	Pfad der angeforderten Ressource
`pid=1234`	Prozess-ID
`comm="nginx"`	Prozessname
`requested_mask="r"`	Gewünschte Berechtigung (`r` = Read)
`denied_mask="r"`	Verweigerte Berechtigung
`fsuid=33`	Dateisystem-UID des Prozesses
`ouid=33`	Eigentümer-UID der Datei

Audit-Meldungen gezielt mit `ausearch` abfragen:

```
# Alle AVC-Denials der letzten Stunde filtern
sudo ausearch -m AVC,USER_AVC -ts recent | grep "profile=/usr/bin/nginx"
```

Systematische Fehlersuche

Symptom	Ursache	Lösung
Dienst startet nicht	Profil blockiert Dateizugriff	`aa-complain` setzen, `journalctl -k -g apparmor` prüfen, Pfadregel ergänzen
Datei nicht lesbar	`r`-Regel fehlt	Pfad-Globbing im Profil anpassen
Socket nicht erreichbar	`unix stream` oder `network` fehlt	Socket-Berechtigung im Profil ergänzen
Performance-Probleme	Hohes Audit-Logging durch Denials	Unnötige `deny`-Regeln bereinigen, Profil optimieren

AppArmor bei systemd-Diensten

systemd kann AppArmor-Profile direkt an Service-Units koppeln:

```
sudo systemctl edit nginx.service
```

```
[Service]
AppArmorProfile=/etc/apparmor.d/usr.bin.nginx
```

```
sudo systemctl daemon-reload
sudo systemctl restart nginx
```

> **Profil-Validierung vor dem Laden:** Prüfe ein Profil vorab syntaktisch mit `sudo apparmor_parser -Q /etc/apparmor.d/usr.bin.nginx` (der Flag `-Q` parst die Datei, ohne sie in den Kernel zu laden).

Kernel-Härtung mit linux-hardened

Was linux-hardened anders macht

Der Standard-Kernel (`linux`) enthält bereits grundlegende Schutzmechanismen wie NX-Bit, ASLR und Stack-Canaries. Der offizielle Kernel `linux-hardened` wendet jedoch einen kontinuierlich gepflegten Sicherheits-Patchsatz an, der drei entscheidende Ebenen ergänzt:

1. **Strengere Compiler-Defaults:** Vollständiges Position Independent Executable (PIE), `FORTIFY_SOURCE` und Full RELRO.
2. **Restriktive Kernel-Parameter:** Standardmäßige Schließung von Informationslecks und Debugging-Schnittstellen.
3. **Exploit-Mitigationen:** Kernel-Patches, die Exploit-Techniken wie Ret2usr, unkontrollierte Kernel-Memory-Zugriffe und das Nachladen von Modulen nach dem Boot unterbinden.

Feature	`linux` (Standard)	`linux-hardened`	Sicherheitseffekt
ASLR (Mmap-Rnd-Bits)	28 Bit	32 Bit	16-fache Entropie bei Speicher-Mappings

Feature	`linux` (Standard)	`linux-hardened`	Sicherheitseffekt
`kernel.kptr_restrict`	0 (offen)	2 (versteckt)	Kernel-Symboladressen in `/proc/kallsyms` unlesbar
`kernel.dmesg_restrict`	0 (offen)	1 (eingeschränkt)	Boot- und Kernelmeldungen nur für Root sichtbar
`kernel.modules_disabled`	nie gesetzt	nach Boot setzbar	Modul-Nachladen nach Systemstart vollständig blockierbar
BPF-Härtung	Standard	`bpf_jit_harden=2`	JIT-Konstanten-Randomisierung gegen Spray-Angriffe
`kernel.perf_event_paranoid`	1	3	`perf_event_open` für unprivilegierte Benutzer gesperrt

> **Parallelbetrieb:** Du kannst `linux` und `linux-hardened` problemlos parallel installiert halten. Im GRUB-Menü wählst du beim Booten den gewünschten Kernel aus.

Installation

```
# linux-hardened und Headers für DKMS installieren
sudo pacman -S linux-hardened linux-hardened-headers
```

`linux-hardened-headers` werden für DKMS-Module (Nvidia, VirtualBox, WireGuard-Kernelmodule) zwingend benötigt.

Aktualisiere die Bootloader-Konfiguration und generiere die Initramfs-Abbilder:

```
# GRUB neu generieren
sudo grub-mkconfig -o /boot/grub/grub.cfg

# Initramfs für alle installierten Kernel neu bauen
sudo mkinitcpio -P
```

> ⚠ **DKMS-Module prüfen:** Nach der Installation des hardened-Kernels müssen alle DKMS-Module neu gebaut werden. Prüfe mit `dkms status`, ob alle Treiber erfolgreich für `linux-hardened` kompiliert wurden.

Verifikation nach dem Neustart:

```
# Kernel-Version prüfen
uname -r
# Ausgabe: 6.12.9-hardened1-1-hardened

# Aktive LSMs prüfen
cat /sys/kernel/security/lsm
# Ausgabe: landlock,lockdown,yama,integrity,apparmor,bpf

# Hardening-Build-Flags im Kernel überprüfen
zgrep HARDENED /proc/config.gz
# CONFIG_HARDENED_USERCOPY=y
# CONFIG_HARDENED_USERCOPY_FALLBACK=n
# CONFIG_INIT_STACK_ALL_ZERO=y
```

ASLR effektiv erhöhen

Address Space Layout Randomization (ASLR) randomisiert die Speicheradressen von Stack, Heap, Shared Libraries und mmap-Bereichen, um ROP-Exploits (*Return-Oriented Programming*)

abzuwehren.

Auf x86_64-Systemen sind standardmäßig 28 Bits für mmap-Mappings reserviert ($2^{28} \approx 268$ Millionen Adressen). `linux-hardened` erlaubt eine Erhöhung auf 32 Bits ($2^{32} \approx 4{,}3$ Milliarden Adressen):



Konfiguration in `/etc/sysctl.d/51-aslr.conf`:

```
# Maximaler Zufall für 64-Bit-Userspace-Prozesse
vm.mmap_rnd_bits = 32

# Kompatibilität für 32-Bit-Prozesse auf x86_64
vm.mmap_rnd_compat_bits = 16
```

Wende die Einstellungen an:

```
sudo sysctl --system

# Verifikation
cat /proc/sys/vm/mmap_rnd_bits
# Ausgabe: 32
```

Fortgeschrittene Sysctl-Härtung

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

```
# ── Kernel-Symboladressen verbergen ──────────────
# Auch Root kann /proc/kallsyms nicht mehr im Klartext lesen
kernel.kptr_restrict = 2

# ── dmesg-Zugriff beschränken ────────────────────
# Verhindert Informationslecks über Hardware & Treiber für reguläre Benutzer
kernel.dmesg_restrict = 1

# ── Ptrace-Scope einschränken ────────────────────
# Nur Prozesse mit CAP_SYS_PTRACE dürfen fremde Prozesse debuggen
kernel.yama.ptrace_scope = 2

# ── Magic SysRq deaktivieren ─────────────────────
# Verhindert Memory-Dumps und Notfall-Befehle über physische Tastatureingaben
kernel.sysrq = 0

# ── Perf-Events einschränken ─────────────────────
# Verhindert Seitenkanal-Angriffe über das Performance-Monitoring
kernel.perf_event_paranoid = 3

# ── Kexec deaktivieren ──────────────────────────
```

```
# Verhindert das Laden unauthorisierter Kernel im laufenden Betrieb
kernel.kexec_load_disabled = 1

# ── Unprivileged BPF sperren ──────────────────────
kernel.unprivileged_bpf_disabled = 1

# ── BPF JIT-Härtung ───────────────────────────────
# Konstanten-Randomisierung gegen JIT-Spray-Attacken
net.core.bpf_jit_harden = 2
```

Wende die Konfiguration an und prüfe die Parameter:

```
sudo sysctl --system

sudo sysctl \
  kernel.kptr_restrict \
  kernel.dmesg_restrict \
  kernel.yama.ptrace_scope \
  kernel.perf_event_paranoid \
  kernel.kexec_load_disabled \
  kernel.unprivileged_bpf_disabled \
  net.core.bpf_jit_harden
```

Binary- und Kernel-Audits mit `checksec`

`checksec` analysiert ausführbare Binaries und den Kernel auf aktive Schutzmechanismen:

```
# checksec installieren
sudo pacman -S checksec
```

Laufenden Kernel überprüfen:

```
checksec kernel
```

Ausgabe bei gehärtetem Kernel:

```
RELRO           STACK CANARY      NX            PIE             RPATH/RUNPATH
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH

Fortify Source   FORTIFY Functions  Format String  Pointer Protection
Yes              Yes                Yes            Yes
```

Bedeutung der Sicherheitsmerkmale:

Feature	Guter Wert	Schutzmechanismus
RELRO	`Full RELRO`	Global Offset Table (GOT) ist schreibgeschützt (verhindert GOT-Overwrite)
Stack Canary	`Canary found`	Schutz gegen Stack-Buffer-Overflows
NX	`NX enabled`	Stack und Heap sind nicht ausführbar (*No-Execute*)
PIE	`PIE enabled`	Binary ist vollständig adressrandomisiert
Fortify Source	`Yes`	Automatische Puffergrenzenprüfung bei String-Funktionen
RPATH	`No RPATH`	Verhindert Dynamic-Linker-Hijacking über unsichere Suchpfade

Alle laufenden Prozesse scannen:

```
checksec --no-banner --no-warnings procAll
```

> **checksec vs. lynis:** `checksec` prüft binäre und speicherbezogene Härtungsfeatures (RELRO, NX, PIE, ASLR). `lynis` auditiert Konfigurationen, PAM, Dienste und Berechtigungen. Beide Werkzeuge ergänzen sich ideal.

Boot-Sicherheit: GRUB-Passwort & LUKS2 fortgeschritten

Angriffsszenario: Physischer Zugriff

Ein Server mit lokal zugänglicher Hardware unterliegt anderen Bedrohungsvektoren als ein rein netzwerkbasierter Host. Ohne Bootloader- und Header-Härtung kann ein Angreifer mit wenigen Handgriffen das System kompromittieren:

1. **Kernel-Parameter manipulieren:** Über den interaktiven GRUB-Bootloader `init=/bin/bash` oder `rd.break` übergeben, um die Passwortabfrage des Logins zu umgehen.
2. **Direkter Datendiebstahl:** Festplatte entnehmen und an einem Zweitsystem mounten (bei unverschlüsselten Partitionen sofort lesbar).
3. **Evil-Maid-Angriff:** Unverschlüsselte Boot-Dateien manipulieren, um beim nächsten regulären Start das LUKS-Passwort abzufangen.

```
┌──────────────────────────────────────────────────────────┐
│ ANGRIFFSSZENARIO: PHYSISCHER ZUGRIFF                     │
├──────────────────────────────────────────────────────────┤
│ 1. Physischer Zugriff auf den Server                     │
│    ↓                                                     │
│ 2. GRUB-Menü erscheint (kein Passwort, Parameter editierbar)│
│    ↓                                                     │
│ 3. Angreifer übergibt `init=/bin/bash` als Kernel-Parameter │
│    ↓                                                     │
│ 4. Kernel bootet und startet Root-Shell statt Login-Maske   │
│    ↓                                                     │
│ 5. Root-Shell aktiv: Unverschlüsselte Daten sofort lesbar   │
│    ↓                                                     │
│ 6. Bei LUKS: Manipulation der Boot-Dateien / Evil-Maid      │
│                                                          │
│ FAZIT: GRUB-Passwort & LUKS-Header-Backup schützen den Boot.│
└──────────────────────────────────────────────────────────┘
```

GRUB-Passwort einrichten

Schritt 1: PBKDF2-Passwort-Hash generieren

```
# Hash für das GRUB-Superuser-Passwort erzeugen
sudo grub-mkpasswd-pbkdf2
```

Kopiere den generierten Hash (`grub.pbkdf2.sha512.10000...`).

Schritt 2: GRUB-Konfiguration in `/etc/grub.d/40_custom` anpassen

```
#!/bin/sh
exec tail -n +3 $0

# ── GRUB Superuser und Hash ──────────────────────
set superusers="root"
password_pbkdf2 root grub.pbkdf2.sha512.10000.DEIN_GENERIERTER_HASH...
```

Schritt 3: Kernel-Parameter gegen Shell-Zugriff im Initramfs sperren

Öffne `/etc/default/grub` und ergänze:

```
# Notfall-Shells im Initramfs deaktivieren
GRUB_CMDLINE_LINUX_DEFAULT="loglevel=3 quiet rd.shell=0"

# Recovery-Einträge aus dem Boot-Menü entfernen
GRUB_DISABLE_RECOVERY=true
```

GRUB-Konfiguration neu generieren:

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
```

Schutzziel	GRUB-Passwort & `rd.shell=0`	Wirkung
Boot-Parameter ändern	Geschützt	Passwort vor Druck auf `e` erforderlich
Single-User Mode	Geschützt	Recovery-Eintrag entfernt

Schutzziel	GRUB-Passwort & `rd.shell=0`	Wirkung
Shell im Initramfs	✅ Geschützt	`rd.shell=0` unterbindet Fallback-Shell
Plattenentnahme	❌ Kein Schutz	Hierfür ist LUKS2-Vollverschlüsselung zuständig

LUKS2 Header-Management & Offline-Backups

Der LUKS-Header enthält alle kryptografischen Metadaten und Keyslots. Wird dieser Datenblock beschädigt (z. B. durch Dateisystemfehler oder Speicherdefekte), sind alle Daten unwiederbringlich verloren – selbst bei bekanntem Passwort.



Header-Backup erstellen:

```
# Backup-Verzeichnis anlegen
sudo mkdir -p /root/luks-backups

# LUKS2-Header sichern
sudo cryptsetup luksHeaderBackup /dev/nvme0n1p2 \
    --header-backup-file /root/luks-backups/nvme0n1p2-header.img

# Auf externen Offline-Datenträger transferieren
sudo mount /dev/sdb1 /mnt/usb
sudo cp /root/luks-backups/nvme0n1p2-header.img /mnt/usb/
sudo umount /mnt/usb
```

> ⚠ **Offline-Pflicht:** Lagere den Header-Backup-Stick physisch getrennt vom Server. Ein Backup auf derselben Partition ist bei einer Beschädigung des Datenträgers nutzlos.

Header im Notfall wiederherstellen:

```
# Aus einem Live-System heraus:
sudo cryptsetup luksHeaderRestore /dev/nvme0n1p2 \
    --header-backup-file /mnt/usb/nvme0n1p2-header.img
```

LUKS2-Keyslots verwalten

```
# Header und aktive Keyslots inspizieren
sudo cryptsetup luksDump /dev/nvme0n1p2
```

Beispielhafte Ausgabe:

```
LUKS header information
Version:        2
Epoch:          5
Metadata area:  [16384, 8176640]
Keyslots:
  0: luks2
    Key:        512 bits
    Priority:   regular
    PBKDF:      argon2id
    Time cost:  4
    Memory:     1048576
    Threads:    4
```

Zusätzliche Notfall-Passphrase hinzufügen:

```
sudo cryptsetup luksAddKey /dev/nvme0n1p2
```

Kompromittierte Passphrase entfernen:

```
# Keyslot 1 gezielt löschen
sudo cryptsetup luksKillSlot /dev/nvme0n1p2 1
```

Schlüsselfeature für automatisches Mounten sekundärer Laufwerke:

```
# Keyfile generieren und strikt berechtigen
sudo dd if=/dev/urandom of=/root/data-drive.keyfile bs=512 count=8
sudo chmod 400 /root/data-drive.keyfile
sudo chown root:root /root/data-drive.keyfile

# Keyfile dem sekundären Volume zuweisen
sudo cryptsetup luksAddKey /dev/sdb1 /root/data-drive.keyfile

# In /etc/crypttab hinterlegen
echo "data-volume  /dev/sdb1  /root/data-drive.keyfile  luks" | sudo tee -a /etc/crypttab
```

Automatische Entsperrung mit TPM2 (`systemd-cryptenroll`)

`systemd-cryptenroll` erlaubt es, LUKS2-Partitionen an das hardwarebasierte Trusted Platform Module (TPM 2.0) zu binden. Das Volume wird beim Booten nur freigegeben, wenn die Platform Configuration Registers (PCRs) exakt den erwarteten Systemzustand nachweisen.

```
┌──────────────────────────────────────────────────────────┐
│ TPM2-BINDUNG: AUSWIRKUNG VON SYSTEM-ÄNDERUNGEN           │
├──────────────────────────────────────────────────────────┤
│ BIOS/UEFI-UPDATE:                                        │
│ • PCR 0 ändert sich → TPM2 blockiert automatischen Unlock │
│ • Lösung: Manuelle Passphrase als Fallback eingeben      │
│                                                          │
│ SECURE BOOT DEAKTIVIERT:                                 │
│ • PCR 7 ändert sich → TPM2 verweigert Entsperrung        │
│ • Schutz gegen Boot unautorisierter Live-Systeme         │
│                                                          │
│ DIEBSTAHL DES LAUFWERKS:                                 │
│ • TPM2 ist fest auf dem Mainboard verlötet               │
│ • Auf fremdem Mainboard ohne Passphrase nicht lesbar     │
│                                                          │
│ WICHTIG: Passphrase im Tresor bleibt Pflicht-Fallback!   │
```



PCR-Bedeutung für die Bindung:

PCR	Überwachter Zustand	Schutzfunktion
PCR 0	Core UEFI-Firmware-Code	Schützt gegen BIOS-Manipulationen
PCR 1	UEFI-Konfiguration	Verhindert Ändern von Boot-Optionen
PCR 7	Secure-Boot-Status & Signaturen	Schützt gegen Boot unzertifizierter Kernel

TPM2-Bindung aktivieren:

```
# TPM2-Gerät auf dem System identifizieren
sudo systemd-cryptenroll --tpm2-device=list

# Volume an PCR 0, 1 und 7 binden
sudo systemd-cryptenroll /dev/nvme0n1p2 \
    --tpm2-device=auto \
    --tpm2-pcrs=0+1+7
```

> ⚠ **Passphrase als Pflicht-Fallback:** Das Binden an das TPM2 ersetzt niemals die Notfall-Passphrase. Nach einem Firmware-Update ändert sich PCR 0 und das System fordert beim nächsten Booten zwingend die manuelle Passphrase an.

> □□ **Regelmäßiger Recovery-Drill:** Teste das Emergency-Recovery einmal pro Quartal: Boote das System von einem Arch-Live-USB-Stick und simuliere die Wiederherstellung des

> LUKS-Headers und den Zugriff über die Notfall-Passphrase.

Container-Netzwerk-Isolation mit nftables

Warum nftables für Container?

Docker verwaltet standardmäßig iptables-Regeln autonom. Bei jedem `docker run` erstellt Docker Regeln in `nat`, `filter` und `mangle`, die Port-Forwarding und NAT für den Container konfigurieren. Das Problem: Diese Regeln umgehen Host-Firewalls in der `PREROUTING`-Phase und können manuelle Isolationsregeln überschreiben.

`nftables` löst dieses Problem deterministisch. Mit `--iptables=false` übergibst du die Netzwerk-Kontrolle vollständig an nftables. Docker erstellt keine autonomen Regeln mehr – du kontrollierst jeden einzelnen Paketfluss direkt im Kernel.

```
# ═══════════════════════════════════════════════
# Vergleich: Docker iptables vs. nftables
# ═══════════════════════════════════════════════

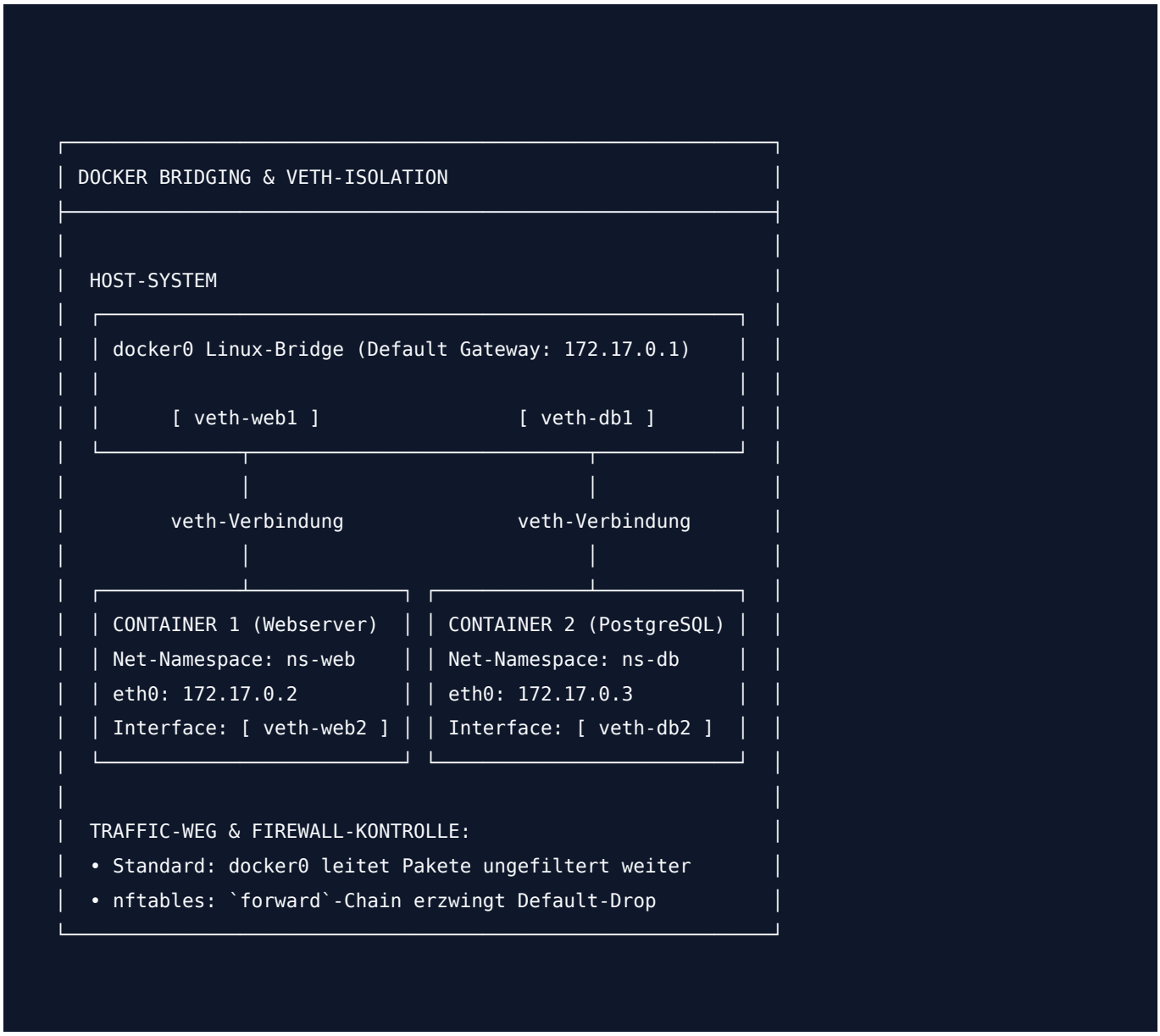
# Docker iptables (Standard):
# - Docker schreibt autonome Regeln in nat, filter, mangle
# - Manuelle Regeln können bei Container-Neustart überschrieben werden
# - Port-Forwarding via -p 8080:80 erzeugt unkontrollierte DNAT-Regeln
# - Keine granulare Kontrolle über Container-to-Container Traffic

# nftables (mit --iptables=false):
# - Docker erstellt KEINE autonomen Regeln
# - Volle Kontrolle über jeden Paketfluss im Host-Kernel
# - Regeln überleben Container-Neustarts dauerhaft
# - Granulare Isolation zwischen Micro-Services
# - Connection Tracking (conntrack) für zustandsbehaftete Filterung
```

Docker-Netzwerk-Architektur & Isolation

Docker erstellt standardmäßig das Bridge-Netzwerk `docker0`. Jeder Container erhält:

1. Einen isolierten **Network Namespace** mit eigenem IP-Stack.
2. Ein **veth-Paar** (*Virtual Ethernet*), das den Container mit `docker0` verbindet.
3. Einen Eintrag in der Bridge, die standardmäßig sämtlichen Datenverkehr zwischen allen Containern ungefiltert weiterleitet.



Status der Bridge inspizieren:

```
# Bridge-Interface prüfen
ip link show docker0

# Aktive Nachbarn auf der Bridge
ip neigh show dev docker0
```

Funktionsunterschied bei deaktiviertem Docker-iptables:

Funktion	Mit Docker-iptables	Mit nftables (`--iptables=false`)
Port-Forwarding	Automatisch via DNAT	Explizit über nftables DNAT
NAT / Internet-Zugang	Automatisch via MASQUERADE	Explizit in postrouting MASQUERADE
Container-to-Container	Standardmäßig offen	Default Drop, nur Whitelist
DNS-Auflösung	Docker-Resolver (127.0.0.11)	Explizite Port-53-Regel

> ⚠ **Docker Daemon-Anpassung:** Das Setzen von `"iptables": false` erfolgt in `/etc/docker/daemon.json`. Nach dem Daemon-Neustart müssen alle Container neu gestartet werden.

nftables-Regelwerk für Container-Isolation

Vollständiges, gehärtetes Regelwerk in `/etc/nftables.conf`:

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

flush ruleset
```

```
define CONTAINER_NET = 172.17.0.0/16
define HOST_IP = 192.168.1.100

table inet container_filter {

    # Dynamische IP-Sets für Container-Rechte
    set internet_allowed {
        type ipv4_addr
        flags interval, timeout
    }

    set host_allowed {
        type ipv4_addr
        flags interval, timeout
    }

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

        iif lo accept
        ct state established,related accept
        ct state invalid drop

        # SSH nur aus dem physischen LAN erlauben, niemals aus Containern
        tcp dport 22 ip saddr != $CONTAINER_NET accept

        ip protocol icmp accept
        ip6 nexthdr icmpv6 accept

        # Autorisierte Container-to-Host Zugriffe (z. B. DNS oder Monitoring)
        ip saddr @host_allowed tcp dport { 80, 443, 8080 } accept
        ip saddr @host_allowed udp dport 53 accept

        log prefix "[CONTAINER-INPUT-DROP] " counter drop
    }

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

        ct state established,related accept
        ct state invalid drop
```

```
        # Container-to-Internet: Nur Container im Set internet_allowed
        ip saddr @internet_allowed oif "eth0" accept

        # Explizite Freigabe für Micro-Services (z. B. Web zu DB)
        ip saddr 172.17.0.2 ip daddr 172.17.0.3 tcp dport 5432 accept

        # DNS-Anfragen der Container weiterleiten
        ip saddr $CONTAINER_NET udp dport 53 accept

        log prefix "[CONTAINER-FORWARD-DROP] " counter drop
    }

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

table ip nat {
    chain prerouting {
        type nat hook prerouting priority dstnat;

        # Port-Forwarding: Host Port 8443 auf Container 172.17.0.2:443
        tcp dport 8443 dnat to 172.17.0.2:443
    }

    chain postrouting {
        type nat hook postrouting priority srcnat;

        # Masquerading für autorisierten ausgehenden WAN-Traffic
        ip saddr $CONTAINER_NET oif "eth0" masquerade
    }
}
```

Automatisierte Befüllung der IP-Sets:

```
sudo nano /usr/local/bin/update-container-ips.sh
```

```
#!/bin/bash
set -euo pipefail

INTERNET_ALLOWED=("172.17.0.2" "172.17.0.3")
HOST_ALLOWED=("172.17.0.4" "172.17.0.5")

nft flush set inet container_filter internet_allowed
nft flush set inet container_filter host_allowed

for ip in "${INTERNET_ALLOWED[@]}"; do
    nft add element inet container_filter internet_allowed { "$ip" }
done

for ip in "${HOST_ALLOWED[@]}"; do
    nft add element inet container_filter host_allowed { "$ip" }
done
```

```
sudo chmod 750 /usr/local/bin/update-container-ips.sh
```

Praxisbeispiel 1: Web-Applikation & PostgreSQL-Isolation

```
┌──────────────────────────────────────────────────────────┐
│ TRAFFIC-ISOLATION: WEBSERVER & POSTGRESQL                │
├──────────────────────────────────────────────────────────┤
│ WEBSERVER (172.17.0.2):                                  │
│ • Zu postgres:5432  -> [ERLAUBT]  (SQL-Queries an DB)    │
│ • Zu Internet:443   -> [ERLAUBT]  (Paket-Updates)        │
│ • Zu Host:22        -> [GESPERRT] (SSH-Zugriff blockiert)  │
```

```
   │                                                              │
   │ POSTGRESQL (172.17.0.3):                                     │
   │ • Zu Webserver      -> [GESPERRT] (Kein Initiieren nötig)    │
   │ • Zu Internet       -> [GESPERRT] (Vollständig isoliert)     │
   │ • Zu Host           -> [GESPERRT] (Kein Host-Zugriff)        │
   │                                                              │
   │ SICHERHEIT: Web-Compromise erreicht niemals Host oder WAN!   │
   └──────────────────────────────────────────────────────────────┘
```

Container mit statischen IP-Adressen starten:

```
docker run -d --name web --ip 172.17.0.2 nginx
docker run -d --name db --ip 172.17.0.3 postgres:16
```

Webserver im `internet_allowed`-Set registrieren:

```
sudo nft add element inet container_filter internet_allowed { 172.17.0.2 }
```

Praxisbeispiel 2: Isolierter Build-Container ohne WAN-Zugriff

Für CI/CD-Pipelines sollen Build-Container oft nur auf die lokale Container-Registry zugreifen können:

```
# Build-Container ohne Internetzugang starten
```

```
docker run -d --name builder --ip 172.17.0.10 registry.local/build-env

# Nur Traffic zur lokalen Registry auf 192.168.1.50:5000 erlauben
sudo nft add rule inet container_filter forward \
    ip saddr 172.17.0.10 ip daddr 192.168.1.50 tcp dport 5000 accept

# DNS erlauben
sudo nft add rule inet container_filter forward \
    ip saddr 172.17.0.10 udp dport 53 accept
```

Praxisbeispiel 3: Monitoring mit Prometheus & Node-Exporter

```
docker run -d --name prometheus --ip 172.17.0.20 prom/prometheus
docker run -d --name node-exporter --ip 172.17.0.21 prom/node-exporter

# Nur Prometheus darf Node-Exporter auf Port 9100 scrapen
sudo nft add rule inet container_filter forward \
    ip saddr 172.17.0.20 ip daddr 172.17.0.21 tcp dport 9100 accept
```

Podman-Netzwerk und Rootless-Isolation

Podman benötigt im Gegensatz zu Docker keinen Root-Daemon. Für Rootless-Container setzt Podman standardmäßig auf `pasta` (ab Podman 4.x) oder `slirp4netns`:

Merkmal	**Docker (`--iptables=false`)**	**Podman (Rootless)**
Architektur	Root-Daemon & Bridge `docker0`	Daemonless, User-Namespace
Firewall-Ebene	Kernel-nftables des Hosts	User-Space Netzwerkstack (`pasta`)
Sicherheitsvorteil	Zentrales Audit im Host-Kernel	Kein Root-Recht im gesamten Stack

```
# Eigenes isoliertes Netzwerk in Podman erstellen
podman network create --subnet 10.89.0.0/24 podman-net

# Container im Subnetz ausführen
podman run -d --network podman-net --name web nginx
podman run -d --network podman-net --name db postgres:16
```

Logging, Debugging & Troubleshooting

Audit-Logs für verworfene Container-Pakete in `/etc/rsyslog.d/10-container.conf` erfassen:

```
:msg, contains, "[CONTAINER-" /var/log/container-firewall.log
& stop
```

Logrotate in `/etc/logrotate.d/container-firewall` einrichten:

```
/var/log/container-firewall.log {
    daily
    rotate 7
    compress
    missingok
    notifempty
    create 0640 root root
}
```

> **Debugging mit tcpdump:** Bei Verbindungsproblemen zwischen Containern capturest du den Traffic direkt auf dem Interface: `sudo tcpdump -i docker0 -n`.

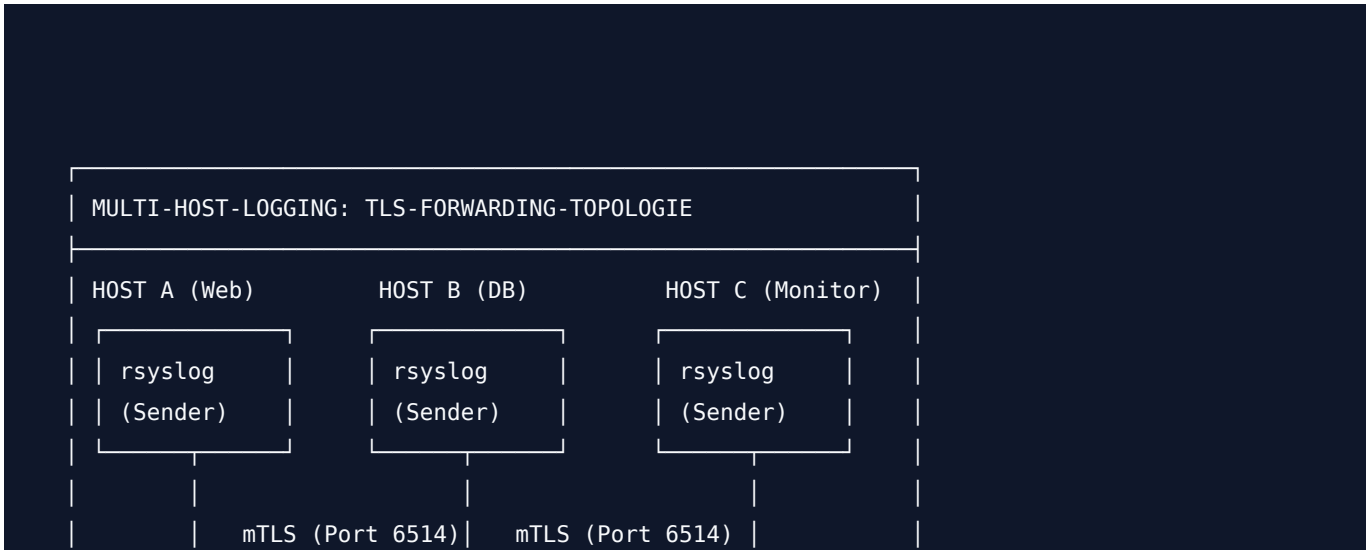
Multi-Host-Logging: Zentrale Logs mit TLS-Forwarding

Warum zentrale Logs?

Ein Angreifer, der einen Server kompromittiert, löscht als Erstes die lokalen Log-Dateien. `~/.bash_history`, `/var/log/auth.log`, `/var/log/syslog` – alles wird überschrieben oder gelöscht. Ohne zentrale Logs hast du danach keine forensische Grundlage mehr.

Zentrale Logs lösen drei Probleme:

1. **Anti-Forensics verhindern:** Logs auf dem Log-Server sind vom Angreifer nicht erreichbar
2. **Korrelation:** Warnungen über mehrere Hosts hinweg erkennen (z.B. gleichzeitige SSH-Logins auf Web- und DB-Server)
3. **Compliance:** DSGVO und IT-Sicherheitsgesetz verlangen auditierbare Zugriffsprotokolle mit Aufbewahrungsfristen





Zertifikate für TLS-Logging

TLS-Forwarding schützt Logs vor Mitlesen im Netzwerk. Jeder Host braucht ein Client-Zertifikat, der Log-Server ein Server-Zertifikat. Wir erstellen eine eigene CA (Certificate Authority) dafür.

Schritt 1: CA erstellen

```
# Verzeichnis für Zertifikate erstellen
sudo mkdir -p /etc/rsyslog/ca
cd /etc/rsyslog/ca

# CA privaten Schlüssel erzeugen (4096 Bit, RSA)
sudo openssl genrsa -out ca.key 4096

# CA-Zertifikat erzeugen (gültig für 10 Jahre)
sudo openssl req -x509 -new -nodes -key ca.key \
    -sha256 -days 3650 \
    -out ca.crt \
    -subj "/C=DE/ST=Bayern/L=Muenchen/O=Homelab/CN=Log-CA"

# Verifikation
sudo openssl x509 -in ca.crt -text -noout | grep -A2 "Issuer"
# Issuer: C = DE, ST = Bayern, L = Muenchen, O = Homelab, CN = Log-CA
```

Schritt 2: Server-Zertifikat für Log-Receiver

```
cd /etc/rsyslog/ca

# Server-Key erzeugen
sudo openssl genrsa -out server.key 4096

# CSR (Certificate Signing Request) erzeugen
sudo openssl req -new -key server.key \
    -out server.csr \
    -subj "/C=DE/ST=Bayern/L=Muenchen/O=Homelab/CN=logserver.local"

# Server-Zertifikat signieren (gültig für 5 Jahre)
sudo openssl x509 -req -in server.csr \
    -CA ca.crt -CAkey ca.key -CAcreateserial \
    -out server.crt -days 1825 -sha256

# SAN (Subject Alternative Name) hinzufügen
# ACHTUNG: Ohne SAN verweigert rsyslog die Verbindung!
sudo openssl x509 -in server.crt -out server-san.crt \
    -extfile <(printf "subjectAltName=DNS:logserver.local,DNS:logserver,IP:192.168.1.12")
sudo mv server-san.crt server.crt
```

Schritt 3: Client-Zertifikate für jeden Host

```
# Für jeden Host ein eigenes Zertifikat erzeugen
for HOST in webserver dbserver monitor; do
    sudo openssl genrsa -out ${HOST}.key 4096
    sudo openssl req -new -key ${HOST}.key \
        -out ${HOST}.csr \
        -subj "/C=DE/ST=Bayern/L=Muenchen/O=Homelab/CN=${HOST}.local"
    sudo openssl x509 -req -in ${HOST}.csr \
        -CA ca.crt -CAkey ca.key -CAcreateserial \
        -out ${HOST}.crt -days 1825 -sha256
done
```

Schritt 4: Zertifikate auf Hosts verteilen

```
# Auf jedem Host:
sudo mkdir -p /etc/rsyslog/certs

# CA-Zertifikat (auf alle Hosts)
sudo cp ca.crt /etc/rsyslog/certs/

# Server-Zertifikat (nur auf Log-Server)
sudo cp server.key server.crt /etc/rsyslog/certs/

# Client-Zertifikat (auf den jeweiligen Host)
sudo cp webserver.key webserver.crt /etc/rsyslog/certs/
```

Schritt 5: Verifikation

```
# Prüfen, ob Zertifikat von CA signiert ist
sudo openssl verify -CAfile ca.crt server.crt
# server.crt: OK

sudo openssl verify -CAfile ca.crt webserver.crt
# webserver.crt: OK

# Zertifikat-Details anzeigen
sudo openssl x509 -in server.crt -text -noout | grep -A2 "Subject:"
# Subject: C = DE, ST = Bayern, L = Muenchen, O = Homelab, CN = logserver.local
```

> ⚠ **SAN ist Pflicht:** Ohne `subjectAltName` im Server-Zertifikat verweigert rsyslog die TLS-Verbindung mit `SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed`. Der SAN muss den Hostnamen UND die IP-Adresse des Log-Servers enthalten.

rsyslog Sender konfigurieren

Auf jedem Host, der Logs an den Log-Server sendet:

```
sudo nano /etc/rsyslog.d/50-forward.conf
```

```
# ═══════════════════════════════════════════════
# rsyslog TLS-Forwarding (Sender)
# ═══════════════════════════════════════════════

# Globale Module laden
module(load="imuxsock")    # Lokale Logs
module(load="imklog")      # Kernel-Logs
module(load="imfile")      # Datei-Monitoring
module(load="omfwd")       # Forwarding

# ─────────────────────────────────────────────
# TLS-Transport-Definition
# ─────────────────────────────────────────────

# TLS-Zertifikate
$DefaultNetstreamDriverCAFile /etc/rsyslog/certs/ca.crt
$DefaultNetstreamDriverCertFile /etc/rsyslog/certs/webserver.crt
$DefaultNetstreamDriverKeyFile /etc/rsyslog/certs/webserver.key

# TLS-Transport aktivieren
$ActionSendStreamDriver gtls
$ActionSendStreamDriverMode 1
$ActionSendStreamDriverAuthMode x509/certvalid

# ─────────────────────────────────────────────
# Forwarding-Regeln
# ─────────────────────────────────────────────
```

```
# Alle lokalen Logs an Log-Server senden
*.* @@(omfwd)logserver.local:6514;RSYSLOG_SyslogProtocol23Format

# ─────────────────────────────────────────────────
# Queueing für Zuverlässigkeit
# ─────────────────────────────────────────────────

# Queue-Typ: Disk-assisted (overflow auf Festplatte)
$ActionQueueType LinkedList
$ActionQueueFileName fwd_queue
$ActionQueueMaxDiskSpace 256m
$ActionQueueSaveOnShutdown on
$ActionQueueTimeoutEnqueue 100
$ActionResumeRetryCount -1
$ActionResumeInterval 30

# ─────────────────────────────────────────────────
# Rate-Limiting gegen Log-Flooding
# ─────────────────────────────────────────────────

# Maximal 1000 Nachrichten pro Sekunde
$NetstatsRateLimit 1000
```

```
# Verifikation: Syntax prüfen
sudo rsyslogd -N1
# rsyslogd: version ..., config validation run took 0.001 seconds

# rsyslog neu starten
sudo systemctl restart rsyslog

# Verbindungsstatus prüfen
sudo systemctl status rsyslog
# active (running)
```

Was passiert bei Netzwerkausfall:

```
┌──────────────────────────────────────────────────────────┐
│ QUEUEING-VERHALTEN BEI NETZWERKAUSFALL                   │
├──────────────────────────────────────────────────────────┤
│ 1. NETZWERKAUSFALL:                                      │
│ • rsyslog puffert Logs in /var/spool/rsyslog/fwd_queue   │
│ • Maximal 256 MB Disk-assisted Spooling aktiv            │
│                                                          │
│ 2. WIEDERVERBINDUNG:                                     │
│ • TCP-Verbindung zu Port 6514 wird automatisch reaktiviert  │
│ • Lokale Queue wird chronologisch an Log-Server übertragen  │
│                                                          │
│ 3. REBOOT-SICHERHEIT:                                    │
│ • Unversendete Logs überstehen einen Server-Neustart     │
│ • Kein Datenverlust bei Verbindungsabbrüchen             │
└──────────────────────────────────────────────────────────┘
```

rsyslog Receiver konfigurieren

Auf dem Log-Server:

```
sudo nano /etc/rsyslog.d/00-receiver.conf
```

```
# ═══════════════════════════════════════════════
# rsyslog TLS-Receiver (Log-Server)
# ═══════════════════════════════════════════════

# Modul für TCP-Eingang laden
module(load="imtcp"
```

```
        StreamDriver="gtls"
        StreamDriver.AuthMode="x509/certvalid"
        StreamDriver.Mode="1")

# TLS-Zertifikate
$DefaultNetstreamDriverCAFile /etc/rsyslog/certs/ca.crt
$DefaultNetstreamDriverCertFile /etc/rsyslog/certs/server.crt
$DefaultNetstreamDriverKeyFile /etc/rsyslog/certs/server.key

# Port öffnen
input(type="imtcp" port="6514")

# ─────────────────────────────────────────────────
# Log-Speicherung nach Host/Tag
# ─────────────────────────────────────────────────

# Template: Speicherpfad mit Hostname
template(name="HostLog" type="string"
    string="/logs/%programname%.log")

# Template: Pfad mit Hostname + Datum
template(name="HostLogRotated" type="string"
    string="/logs/%HOSTNAME%/%$year%-%$month%-%$day%.log")

# Empfangene Logs nach Host speichern
if $hostname != 'logserver' then {
    # Logs des jeweiligen Hosts speichern
    action(type="omfile"
           dynaFile="HostLogRotated"
           FileCreateMode="0640"
           DirCreateMode="0750")

    # Auth-Logs separat speichern (für Audit)
    if $programname == "sshd" or $programname == "sudo" then {
        action(type="omfile"
               file="/logs/auth/audit.log"
               FileCreateMode="0640")
    }
}
```

Log-Verzeichnisstruktur erstellen:

```
# Verzeichnisse erstellen
sudo mkdir -p /logs/{auth,web,db,monitor}
sudo mkdir -p /logs/{webserver,dbserver,monitor}

# Berechtigungen setzen
sudo chown -R root:adm /logs
sudo chmod -R 750 /logs

# Log-Rotation konfigurieren
sudo nano /etc/logrotate.d/remote-logs
```

```
/logs/*/*.log {
    daily
    rotate 90
    compress
    delaycompress
    missingok
    notifempty
    create 0640 root adm
    dateext
    dateformat -%Y-%m-%d
}
```

journald Remote-Forwarding

Falls du `journald` statt `rsyslog` als primären Logger nutzt (oder zusätzlich):

```
# Auf dem Sender (Host A/B/C):
```

```
sudo nano /etc/systemd/journald.conf
```

```
[Journal]
# Remote-Forwarding an rsyslog
ForwardToSyslog=yes
ForwardToSyslogSocket=yes
SyslogSocket=192.168.1.12:6514

# Lokale Logs beibehalten
Storage=persistent
SystemMaxUse=1G
SystemMaxFileSize=100M
MaxRetentionSec=30day
```

```
# Auf dem Receiver (Log-Server):
sudo nano /etc/rsyslog.d/10-journald.conf
```

```
# journald-Logs annehmen (falls zusätzlich zu rsyslog)
module(load="imudp")
input(type="imudp" port="514")
```

```
# journald neu starten
sudo systemctl restart systemd-journald
```

> **rsyslog vs. journald für Multi-Host:** rsyslog ist für Multi-Host-Logging besser geeignet. Es unterstützt TLS nativ, hat Queueing und kann Logs nach Host/Tag/Programm aufteilen. journald ist primär für lokale Logs gedacht. Empfehlung: rsyslog als Sender, journald nur lokal.

Log-Integrität

Logs müssen vor Manipulation geschützt werden. Zwei Methoden: Signierung und append-only Mount.

Method 1: Log-Signierung mit rsyslog

```
# Auf dem Log-Server:
sudo nano /etc/rsyslog.d/20-signing.conf
```

```
# ═══════════════════════════════════════════════
# Log-Signierung
# ═══════════════════════════════════════════════

# Signierungs-Schlüssel erzeugen (einmalig)
# openssl genrsa -out /etc/rsyslog/signing.key 4096
```

```
module(load="omfile")

# Logs nach dem Schreiben signieren
template(name="SignedLog" type="string"
    string="/logs/%HOSTNAME%/%programname%.log")

action(type="omfile"
       dynaFile="SignedLog"
       FileCreateMode="0640"
       DirCreateMode="0750"
       cmd="/usr/bin/rsyslog-log-signer -s /etc/rsyslog/signing.key")
```

Method 2: Append-only Mount (einfacher und effektiver)

```
# Log-Partition mounten (append-only)
# ACHTUNG: Danach können Logs nur hinzugefügt, nicht gelöscht werden!
sudo mount -o remount,append-only /logs

# Verifikation
mount | grep /logs
# /dev/sdb1 on /logs type ext4 (rw,appendonly)
```

```
┌─────────────────────────────────────────────────────────┐
│ LOG-INTEGRITÄT: VERGLEICH DER SICHERUNGSMETHODEN        │
├─────────────────────────────────────────────────────────┤
│ APPEND-ONLY FILESYSTEM (mount -o append-only / chattr +a):  │
│ • Verhindert: Löschen und Überschreiben bestehender Zeilen  │
│ • Erlaubt: Nur kontinuierliches Anhängen neuer Log-Daten    │
│ • Praxis: Geringer Overhead, ideal für Linux-Logserver      │
│                                                         │
```

```
  │ KRYPTOGRAFISCHE SIGNIERUNG (rsyslog-log-signer / RFC 5424): │
  │ • Verhindert: Fälschung und Manipulation durch Angreifer    │
  │ • Schutz: Integritätsprüfung jedes einzelnen Log-Blocks     │
  │                                                             │
  │ EMPFEHLUNG: mTLS-Forwarding kombiniert mit Append-Only!     │
  └─────────────────────────────────────────────────────────────┘
```

Praxis: 3-Host-Setup

Host-Übersicht:

Host	IP	Rolle	rsyslog-Rolle
webserver	192.168.1.10	nginx, Docker	Sender
dbserver	192.168.1.11	PostgreSQL	Sender
logserver	192.168.1.12	Log-Aggregation	Receiver

Vollständige Konfiguration für jeden Host:

```
# ──────────────────────────────────────────────────
# Log-Server (192.168.1.12)
# ──────────────────────────────────────────────────

# 1. Zertifikate installieren
sudo cp ca.crt server.key server.crt /etc/rsyslog/certs/

# 2. Receiver-Konfiguration aktivieren
# /etc/rsyslog.d/00-receiver.conf

# 3. Firewall öffnen
sudo nft add rule inet filter input \
    tcp dport 6514 accept comment "rsyslog TLS"
```

```
# 4. Log-Verzeichnisse erstellen
sudo mkdir -p /logs/{auth,web,db,monitor}
sudo chown -R root:adm /logs

# 5. rsyslog starten
sudo systemctl enable --now rsyslog

# 6. Verifikation
sudo ss -tlnp | grep 6514
# LISTEN  0  128  0.0.0.0:6514  0.0.0.0:*  users:(("rsyslogd",pid=1234,fd=6))
```

```
# ──────────────────────────────────────────────────
# Web-Server (192.168.1.10)
# ──────────────────────────────────────────────────

# 1. Zertifikate installieren
sudo cp ca.crt webserver.key webserver.crt /etc/rsyslog/certs/

# 2. Sender-Konfiguration hinterlegen
# /etc/rsyslog.d/50-forward.conf

# 3. rsyslog starten
sudo systemctl enable --now rsyslog

# 4. Verifikation
sudo rsyslogd -N1
# config validation run took 0.001 seconds
```

```
# ──────────────────────────────────────────────────
# DB-Server (192.168.1.11)
# ──────────────────────────────────────────────────
```

```
# 1. Zertifikate installieren
sudo cp ca.crt dbserver.key dbserver.crt /etc/rsyslog/certs/

# 2. Sender-Konfiguration hinterlegen
# /etc/rsyslog.d/50-forward.conf

# 3. PostgreSQL-Logs zusätzlich erfassen
# /etc/rsyslog.d/51-postgres.conf:
# $AddUnixListenSocket /var/run/postgresql/.s.PGSQL.5432
# :programname, isequal, "postgres" /var/log/postgres.log
# & stop

# 4. rsyslog starten
sudo systemctl enable --now rsyslog
```

End-to-End-Verifikation:

```
# 1. Auf Web-Server: Test-Log generieren
logger -t test "Multi-Host-Logging Test from webserver"

# 2. Auf Log-Server: Prüfen
sudo tail -f /logs/webserver/test.log
# Aug 27 12:34:56 webserver test: Multi-Host-Logging Test from webserver

# 3. Auth-Logs prüfen
sudo tail -f /logs/auth/audit.log
# Aug 27 12:34:56 webserver sshd[1234]: Accepted publickey for user

# 4. TLS-Verbindung prüfen
openssl s_client -connect logserver:6514 \
    -cert /etc/rsyslog/certs/webserver.crt \
    -key /etc/rsyslog/certs/webserver.key \
    -CAfile /etc/rsyslog/certs/ca.crt
# Verify return code: 0 (ok)
```

Aufbewahrung & Compliance

DSGVO-Anforderungen:

- **Zugriffsprotokolle**: 10 Tage (Mindestaufbewahrung)
- **Vollständige Logs**: 6 Monate (bei Datenschutzverletzung)
- **Löschkonzept**: Automatische Löschung nach Ablauf

IT-Sicherheitsgesetz (IT-SiG):

- **Technische Log-Daten**: 12 Monate
- **Audit-Logs**: 12 Monate
- **Bei kritischen Infrastrukturen (KRITIS)**: 24 Monate

Log-Rotation mit automatischer Löschung:

```
# /etc/logrotate.d/remote-logs
/logs/*/*.log {
    daily
    rotate 90
    compress
    delaycompress
    missingok
    notifempty
    create 0640 root adm
    dateext
    dateformat -%Y-%m-%d
    postrotate
        /usr/bin/systemctl kill -s HUP rsyslog.service > /dev/null 2>&1 || true
    endscript
}
```

```
# /etc/cron.d/log-cleanup (Täglich um 03:00 Uhr: Logs älter als 365 Tage löschen)
0 3 * * * root find /logs -name "*.log.*" -mtime +365 -delete
```

> **Audit vor dem Löschen:** Vor der automatischen Löschung sollte ein Admin die betroffenen Logs gesichtet haben.

Verifikation

```
# 1. Auf ALLEN Hosts: rsyslog-Syntax prüfen
sudo rsyslogd -N1
# Erwartung: "config validation run took 0.001 seconds"

# 2. Log-Server: Port 6514 geöffnet?
sudo ss -tlnp | grep 6514
# LISTEN  0  128  0.0.0.0:6514  0.0.0.0:*  users:(("rsyslogd",...))

# 3. Sender → Receiver: TLS-Verbindung
sudo openssl s_client -connect logserver:6514 \
    -cert /etc/rsyslog/certs/webserver.crt \
    -key /etc/rsyslog/certs/webserver.key \
    -CAfile /etc/rsyslog/certs/ca.crt \
    </dev/null 2>/dev/null | grep "Verify return code"
# Verify return code: 0 (ok)

# 4. Log-Eingang auf Log-Server
logger -t verification "Test from $(hostname)"
sleep 1
sudo tail -5 /logs/$(hostname)/$(date +%Y-%m-%d).log
# Aug 27 12:34:56 webserver verification: Test from webserver

# 5. Queue-Status prüfen (bei Netzwerkproblemen)
```

```
sudo find /var/spool/rsyslog/ -name "fwd_queue*" -ls

# 6. Log-Integrität prüfen
mount | grep /logs
# /dev/sdb1 on /logs type ext4 (rw,appendonly)
```

> ⚠ **Firewall nicht vergessen:** Der Log-Server muss `Port 6514` (TCP) für jeden Host freigeben: `nft add rule inet filter input tcp dport 6514 accept`. Ohne diese Regel können keine Logs empfangen werden.

Backup-Sicherheit: Verschlüsselung, Integrität und 3-2-1-Regel

Warum Backup-Sicherheit?

Ein Angreifer, der Root-Rechte auf einem Server erlangt, hat zwei Prioritäten: Erstens die Spuren verwischen (Logs löschen), zweitens die Daten als Geiseln nehmen. Das bedeutet: Backups chiffrieren, lokale Snapshots löschen und eine Ransomware-Forderung stellen. Ohne gesicherte, verschlüsselte Offsite-Backups stehst du vor der Wahl: Daten verloren oder Lösegeld zahlen.

DSGVO Art. 34 verlangt bei Datenschutzverletzungen eine Meldung an die Aufsichtsbehörde. Wenn du nachweisen kannst, dass Backups existieren und wiederherstellbar sind, reduziert sich der Schaden erheblich. Wenn nicht, droht ein Bußgeld.

```
┌──────────────────────────────────────────────────────────┐
│ ANGRIFFS-PLAYBOOK: RANSOMWARE AUF LINUX-SERVERN          │
├──────────────────────────────────────────────────────────┤
```

```
┌──────────────────────────────────────────────────────┐
│ 1. ROOT-RECHTE ERLANGEN:                             │
│ • SSH-Compromise, Kernel-Exploit oder Container-Escape │
│                                                      │
│ 2. BACKUPS & SNAPSHOTS IDENTIFIZIEREN:               │
│ • Lokale BTRFS-Snapshots: `btrfs subvolume list /`   │
│ • Lokale Archive unter /backup/ und /var/backup/ scannen │
│                                                      │
│ 3. LOKALE BACKUPS VERNICHTEN & DATEIEN CHIFFRIEREN:  │
│ • Snapshots löschen: `btrfs subvolume delete /snapshots/*` │
│ • Produktivdaten verschlüsseln & Ransom-Note hinterlegen │
│                                                      │
│ SCHUTZ: Unantastbare, unveränderliche Offsite-Backups! │
└──────────────────────────────────────────────────────┘
```

LUKS Header Backup (Kreuzverweis)

Die Sicherung des LUKS-Headers wurde bereits im Abschnitt LUKS2 Header-Management & Offline-Backups ausführlich behandelt.

Kurzfassung:

- **Header-Backup auf USB-Stick** (offline lagern, nicht auf demselben Server)
- **Regelmäßiger Test:** `sudo cryptsetup luksHeaderRestore` vom USB-Stick
- **Warum kritisch:** Ohne Header sind alle Daten unzugänglich, selbst mit bekanntem Passwort

> □□ **LUKS-Header + Offsite-Backup = volle Absicherung.** Der LUKS-Header schützt den Boot-Vorgang, Offsite-Backups schützen die Daten bei Ransomware. Beides zusammen ergibt ein vollständiges Sicherheitskonzept.

BTRFS Snapshot-Sicherheit

BTRFS-Snapshots sind ein mächtiges Werkzeug für schnelle Rollbacks, aber sie sind **kein Backup**. Ein Snapshot ist ein konsistenter Zustand zu einem bestimmten Zeitpunkt – aber er bietet keinen Schutz vor Löschung durch einen Angreifer oder vor Hardware-Defekt.

Was passiert bei Compromise:

```
# Angreifer findet Snapshots
sudo btrfs subvolume list /
# ID 257 gen 5 path @
# ID 258 gen 100 path @home
# ID 300 gen 200 path @snapshots/2026-08-01
# ID 301 gen 210 path @snapshots/2026-08-15
# ID 302 gen 220 path @snapshots/2026-08-27

# Angreifer löscht ALLE Snapshots
sudo btrfs subvolume delete /@snapshots/2026-08-01
sudo btrfs subvolume delete /@snapshots/2026-08-15
sudo btrfs subvolume delete /@snapshots/2026-08-27
# → Alle Snapshots weg. Kein Rollback möglich.
```

Schutz: Read-only Snapshots

```
# Read-only Snapshot erstellen (kann nicht gelöscht werden, nur über *)
sudo btrfs subvolume snapshot -r / /@snapshots/2026-08-27-readonly

# ACHTUNG: read-only Snapshots können NICHT gelöscht werden
# * Außer: Angreifer mountet das Volume und macht es read-write
# → Daher: Snapshots auf EXTERNES Volume sichern!
```

Snapshots auf externes Volume sichern:

```
# ──────────────────────────────────────────────────
```

```
# Snapper + Push-to-NAS (automatisiert)
# ──────────────────────────────────────────────────────

# NAS einbinden (z.B. via NFS)
sudo mount -t nfs 192.168.1.200:/backup /mnt/nas

# Snapper-Konfiguration anpassen
sudo nano /etc/snapper/configs/root
```

```
# Snapper-Konfiguration
SUBVOLUME="/"
FSTYPE="btrfs"

# Snapshot-Verzeichnis auf NAS
TIMELINE_MIN_AGE="1800"
TIMELINE_LIMIT_HOURLY="5"
TIMELINE_LIMIT_DAILY="7"
TIMELINE_LIMIT_WEEKLY="4"
TIMELINE_LIMIT_MONTHLY="6"

# Post-Snapshot-Skript: Snapshot auf NAS pushen
RUN_FREQUENCY=1
```

```
# Post-Snapshot-Skript erstellen
sudo nano /etc/snapper/post-snapshot.d/10-push-to-nas.sh
```

```
#!/bin/bash
# Snapshot auf NAS synchronisieren (nach jedem neuen Snapshot)

SNAPSHOT_DIR="/@snapshots"
NAS_TARGET="/mnt/nas/snapshots/$(hostname)"

# NAS-Zielverzeichnis erstellen
mkdir -p "$NAS_TARGET"

# Neuesten Snapshot auf NAS pushen
NEWEST=$(ls -td ${SNAPSHOT_DIR}/*/  | head -1)
rsync -a --delete "${NEWEST}" "${NAS_TARGET}/"

# Verifikation
echo "Snapshot pushed: $(basename $NEWEST) → $NAS_TARGET"
```

```
# Ausführbar machen
sudo chmod +x /etc/snapper/post-snapshot.d/10-push-to-nas.sh
```

BTRFS Scrub (Integritätsprüfung):

```
# Regelmäßige Integritätsprüfung aller Snapshots
sudo btrfs scrub start /

# Status prüfen
sudo btrfs scrub status /
# UUID:             xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
# Scrub started:    Mon Aug 27 12:00:00 2026
```

```
# Status:           finished
# Duration:         0:05:23
# Total to scrub:   128.00GB
# Rate:             400.00MB/s
# Error summary:    no errors found

# Cron: Monatlich automatisch scruben
echo "0 3 1 * * root btrfs scrub start /" | \
    sudo tee /etc/cron.d/btrfs-scrub
```

> ⚠ **Snapshots sind kein Backup.** Ein Snapshot schützt vor versehentlichem Löschen von Dateien und ermöglicht schnelle Rollbacks. Aber er schützt NICHT vor: Hardware-Defekt, Ransomware (die Snapshots löscht), Diebstahl des Servers. Für diese Szenarien brauchst du Offsite-Backups.

Cross-Ref: Snapper-Grundlagen

Die Installation und Konfiguration von Snapper mit BTRFS wurde im letzten Artikel ausführlich behandelt. Dort findest du das Setup für `snapper`, `snap-pac` und `grub-btrfs`.

Backup-Verschlüsselung

Es gibt mehrere Werkzeuge für verschlüsselte Backups. Jedes hat eigene Stärken:

Vergleich der Methoden:

Feature	Restic	BorgBackup	rsync + GPG	LUKS-Container
Verschlüsselung	AES-256	AES-256	GPG (variiert)	AES-256 (LUKS)
Deduplizierung	□	□	□	□
Incremental	□	□	□ (rsync)	□
Anonymisierung	□	□	□	□
Komplexität	Mittel	Mittel	Niedrig	Niedrig

Feature	Restic	BorgBackup	rsync + GPG	LUKS-Container
Performance	Gut	Sehr gut	Gut	Sehr gut
Multi-Backend	✅ (S3, B2, SFTP)	✅ (SSH)	✅ (SSH)	✅ (lokales Volume)

Methode 1: Restic (empfohlen für Homelab)

```
# ──────────────────────────────────────────────────────
# Restic: Verschlüsseltes Backup mit Deduplizierung
# ──────────────────────────────────────────────────────

# Installieren
sudo pacman -S restic

# Repository erstellen (verschlüsselt)
# Passwort wird interaktiv abgefragt
restic init --repo /mnt/nas/backups/$(hostname)

# Oder: Passwort-Datei verwenden
echo "MeinSicheresBackupPasswort123!" | \
    restic init --repo /mnt/nas/backups/$(hostname) --password-file /dev/stdin

# Backup erstellen
restic backup /home /etc /var/log \
    --repo /mnt/nas/backups/$(hostname) \
    --password-file /mnt/usb/backup-passwd.txt \
    --verbose

# Inkrementelles Backup (nur geänderte Daten)
restic backup /home /etc /var/log \
    --repo /mnt/nas/backups/$(hostname) \
    --password-file /mnt/usb/backup-passwd.txt

# Snapshots anzeigen
restic snapshots --repo /mnt/nas/backups/$(hostname)

# Backup wiederherstellen
restic restore latest \
```

```
    --repo /mnt/nas/backups/$(hostname) \
    --target /restore/
```

Restic-Cron für automatische Backups:

```
sudo nano /etc/cron.d/restic-backup
```

```
# Täglich um 02:00 Uhr: Backup erstellen
0 2 * * * root restic backup /home /etc /var/log \
    --repo /mnt/nas/backups/$(hostname) \
    --password-file /mnt/usb/backup-passwd.txt \
    --verbose >> /var/log/restic-backup.log 2>&1

# Wöchentlich (Sonntag 04:00): Alte Snapshots löschen
# (behält: 7 täglich, 4 wöchentlich, 6 monatlich)
0 4 * * 0 root restic forget --repo /mnt/nas/backups/$(hostname) \
    --password-file /mnt/usb/backup-passwd.txt \
    --keep-daily 7 --keep-weekly 4 --keep-monthly 6 \
    --prune --verbose >> /var/log/restic-forget.log 2>&1
```

Methode 2: BorgBackup (Pull-Modus)

```
# ──────────────────────────────────────────────────────
# BorgBackup: Deduplizierung + Verschlüsselung
# ──────────────────────────────────────────────────────
```

```
# Auf dem Backup-Server (NAS):
sudo pacman -S borg

# Borg-Repository erstellen
borg init --encryption=repokey /mnt/nas/backups/$(hostname)

# Auf dem Quell-Server (Pull-Modus):
# Der Backup-Server ruft die Daten ab (kein SSH-Key auf dem Quellserver nötig!)

# Borg-Service auf dem Backup-Server konfigurieren
sudo nano /etc/borg.d/pull-webserver.conf
```

```
# Borg Pull-Konfiguration
SOURCE_HOST="webserver.local"
SOURCE_USER="backup"
SOURCE_PATH="/home /etc"
REMOTE_BORG_REPO="/mnt/nas/backups/webserver"

# SSH-Verbindung zum Quell-Server
borg create --stats --progress \
    ssh://${SOURCE_USER}@${SOURCE_HOST}${REMOTE_BORG_REPO}::'{hostname}-{now}' \
    ${SOURCE_PATH}
```

Methode 3: rsync + GPG (einfach und flexibel)

```
# ──────────────────────────────────────────────────────
# rsync + GPG: Verschlüsselter Transfer
# ──────────────────────────────────────────────────────

# GPG-Schlüssel erzeugen (einmalig)
gpg --gen-key
# Fingerabdruck notieren!
```

```
## Security Audits & Auditd

### Warum Security Audits?

Ein Security Audit ist keine Einmal-Aktion, sondern ein wiederkehrender Prozess. Die Frage
ist nicht, ob ein System kompromittiert wird, sondern wann. Regelmäßige Audits finden
Schwachstellen, bevor ein Angreifer sie ausnutzt.

**Wann wird ein Audit durchgeführt?**

| Anlass | Häufigkeit | Schwerpunkt |
|--------|------------|-------------|
| Routine | Halbjährlich | Komplett-Check |
| Nach Änderungen | Bei Bedarf | Betroffene Komponente |
| Incident Response | Nach Vorfall | Schadensbegrenzung |
| Compliance | Jährlich | DSGVO, IT-SiG, BSI-IT-Grundschutz |
| Vor Deploy | Bei Bedarf | Konfiguration, Abhängigkeiten |

### Audit-Checkliste

Die folgende Checkliste dient als Arbeitsvorlage für jedes Audit. Jeder Punkt wird mit
Status (OK/fehlend/kritisch) und Begründung dokumentiert.

**System:**

| # | Prüfpunkt | Befehl | Status |
|---|-----------|--------|--------|
| 1.1 | Kernel-Version (hardened?) | `uname -r` | |
| 1.2 | Installierte Pakete aktuell? | `checkupdates` | |
| 1.3 | Benutzer ohne Passwort? | `awk -F: '($2 == "") {print $1}' /etc/shadow` | |
| 1.4 | Root ohne Passwort? | `passwd -S root` | |
| 1.5 | Unbenutzte Benutzer deaktiviert? | `lastlog \| grep -v "Never logged in"` | |
| 1.6 | SUID-Binaries reduziert? | `find / -perm -4000 -type f 2>/dev/null` | |

**Netzwerk:**

| # | Prüfpunkt | Befehl | Status |
|---|-----------|--------|--------|
| 2.1 | Offene Ports nur notwendige? | `ss -tlnp` | |
| 2.2 | Firewall aktiv? | `nft list ruleset` | |
| 2.3 | SSH: Root-Login deaktiviert? | `grep PermitRootLogin /etc/ssh/sshd_config` | |
| 2.4 | SSH: Passwort-Auth deaktiviert? | `grep PasswordAuthentication
/etc/ssh/sshd_config` | |
```

```
| 2.5 | TLS auf allen Services? | `openssl s_client -connect ...` | |

**Verschlüsselung:**

| # | Prüfpunkt | Befehl | Status |
|---|-----------|--------|--------|
| 3.1 | LUKS aktiv? | `lsblk -f` | |
| 3.2 | LUKS-Header-Backup vorhanden? | Physisch geprüft | |
| 3.3 | SSH-Key-Verschlüsselung? | `ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key` | |
| 3.4 | TLS-Zertifikate gültig? | `openssl x509 -in ... -checkend 0` | |

**Logging:**

| # | Prüfpunkt | Befehl | Status |
|---|-----------|--------|--------|
| 4.1 | auditd aktiv? | `systemctl status auditd` | |
| 4.2 | rsyslog aktiv? | `systemctl status rsyslog` | |
| 4.3 | Zentrale Logs vorhanden? | `ls /logs/` | |
| 4.4 | Log-Rotation aktiv? | `cat /etc/logrotate.d/remote-logs` | |

**Backups:**

| # | Prüfpunkt | Befehl | Status |
|---|-----------|--------|--------|
| 5.1 | Backups vorhanden? | `restic snapshots --repo ...` | |
| 5.2 | Integrität geprüft? | `restic check --read-data ...` | |
| 5.3 | Restore-Test durchgeführt? | Dokumentation vorhanden | |
| 5.4 | Offsite-Backup vorhanden? | Physisch geprüft | |

### auditd: Das Linux Audit-Framework

`auditd` ist das Kernel-Level-Auditing für Linux. Es protokolliert System-Calls,
Dateizugriffe und Benutzeraktionen auf Kernel-Ebene – das heißt, selbst ein root-
Kompromittierter kann die Audit-Logs nicht löschen, ohne den Audit-Daemon zu stoppen (was
im Normalfall nicht möglich ist).

**Installation und Aktivierung:**
```

Installieren

sudo pacman -S audit

Aktivieren und starten

sudo systemctl enable --now auditd

Verifikation

sudo systemctl status auditd

active (running)

```
**Audit-Regeln schreiben:**
```

sudo nano /etc/audit/rules.d/10-security.rules

```
```

Audit-Regeln für Security-Monitoring

1. Dateizugriff überwachen

/etc/shadow: Jeder Zugriff wird geloggt

(Benutzer-Hashes werden ausgelesen)

-w /etc/shadow -p rwa -k shadow_access

/etc/sudoers: Änderungen werden geloggt

-w /etc/sudoers -p wa -k sudoers_mod -w /etc/sudoers.d/ -p wa -k sudoers_mod

/etc/ssh/sshd_config: Änderungen werden geloggt

-w /etc/ssh/sshd_config -p wa -k sshd_config

/etc/passwd, /etc/group: Änderungen werden geloggt

-w /etc/passwd -p wa -k user_mod -w /etc/group -p wa -k group_mod -w /etc/gshadow -p wa -k group_mod

---

2. System-Calls überwachen

---

execve: Welche Programme werden ausgeführt?

(Nur für verdächtige Programme, nicht für alles!)

-a always,exit -F arch=b64 -S execve -k program_exec

connect: Welche Netzwerkverbindungen werden aufgebaut?

-a always,exit -F arch=b64 -S connect -k network_connect

mount: Dateisysteme werden gemountet

-a always,exit -F arch=b64 -S mount -k mount_op

umount: Dateisysteme werden unmounted

-a always,exit -F arch=b64 -S umount2 -k umount_op

---

3. Benutzeraktionen überwachen

Anmeldeversuche (erfolgreich und fehlgeschlagen)

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

SSH-Logins

-w /var/log/auth.log -p wa -k auth_log

sudo-Benutzung

-w /var/log/sudo.log -p wa -k sudo_usage

---

4. Kernel-Module überwachen

Module laden/entladen

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

5. Performance: Nur relevante Events loggen

Buffer-Größe für Audit-Events erhöhen

-b 8192

Verlust von Audit-Events verhindern

-f 1

Nur bestimmte Benutzer überwachen (nicht root!)

(root wird ohnehin von auditd überwacht)

```
**Audit-Regeln laden:**
```

Regeln laden

sudo auditctl -R /etc/audit/rules.d/10-security.rules

Geladene Regeln anzeigen

sudo auditctl -l

-w /etc/shadow -p rwa -k shadow_access

-w /etc/sudoers -p wa -k sudoers_mod

-a always,exit -F arch=b64 -S execve -k program_exec

...

```
**Audit-Logs auswerten:**
```

ausearch: Spezifische Events suchen

Alle Zugriffe auf /etc/shadow

sudo ausearch -k shadow_access --interpret

Alle execve-Events (Programmaufrufe)

sudo ausearch -k program_exec --interpret

Alle Netzwerkverbindungen

sudo ausearch -k network_connect --interpret

Events der letzten Stunde

sudo ausearch -ts recent -k auth_log

Events eines bestimmten Benutzers

sudo ausearch -ua 1000 --interpret

aureport: Zusammenfassungen erstellen



Gesamtübersicht

sudo aureport

Auth-Bericht (Anmeldeversuche)

sudo aureport --auth

Failed-Login-Bericht

sudo aureport --failed

Login-Bericht nach Benutzer

sudo aureport --login --summary

File-Access-Bericht

sudo aureport -f

Error-Bericht

sudo aureport -e

```
**Beispiel: Verdächtige Aktivität erkennen**
```

Alle Zugriffe auf /etc/shadow in den letzten 24 Stunden

sudo ausearch -k shadow_access -ts today --interpret

Erwartete Ausgabe:

type=PROCTITLE msg=audit(2026-08-27 12:34:56.789:1234) : proctitle=cat /etc/shadow

type=SYSCALL msg=audit(2026-08-27 12:34:56.789:1234) : arch=c000003e syscall=257

success=yes ...

uid=0 root (erwartet: nur root sollte das lesen)

exe=/usr/bin/cat

Oder: Zugriff durch unbekannten Benutzer (KRITISCH!)

type=SYSCALL msg=audit(...) : uid=1001 webserver

exe=/usr/bin/cat → WARNUNG: webserver liest /etc/shadow!

```
<blockquote class="infobox infobox--warn">
⚠ **Audit-Performance:** Zu viele Audit-Regeln können die Performance beeinträchtigen. Jedes execve-Event erzeugt einen Audit-Eintrag. Bei einem aktiven Webserver mit vielen HTTP-Requests kann das zu erheblichem Overhead führen. Beschränke execve-Auditing auf verdächtige Programme, nicht auf alles.
</blockquote>
```

```
**Audit-Regeln optimieren:**
```

Performance-optimierte Audit-Regeln

NUR execve für spezifische Programme loggen

(nicht für alles, sondern nur für verdächtige)

-a always,exit -F arch=b64 -S execve -F path=/usr/bin/sudo -k sudo_exec -a always,exit -F arch=b64 -S execve -F path=/usr/bin/su -k su_exec -a always,exit -F arch=b64 -S execve -F path=/usr/bin/passwd -k passwd_exec

Nur connect für bestimmte UID loggen

-a always,exit -F arch=b64 -S connect -F uid=0 -k root_network

excluded Programs (Performance)

Diese Programme werden NICHT geloggt

-a never,exit -F arch=b64 -S execve -F exe=/usr/bin/bash -a never,exit -F arch=b64 -S execve -F exe=/usr/bin/zsh

```
### LinPEAS: Automatisierte Schwachstellensuche

LinPEAS (Linux Privilege Escalation Awesome Scripts) ist ein automatisiertes Tool, das ein System auf bekannte Schwachstellen und Fehlkonfigurationen prüft, die zu einer Privilege-Eskalation führen können.

**Installation und Ausführung:**
```

LinPEAS herunterladen

curl -L https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh -o /tmp/linpeas.sh chmod +x /tmp/linpeas.sh

Als root ausführen (mit farbiger Ausgabe)

sudo /tmp/linpeas.sh

Als root ausführen (ohne Farben, für Log-Datei)

sudo /tmp/linpeas.sh -a 2>&1 | tee /var/log/linpeas-$(date +%Y%m%d).log

```
**Wichtige Befunde und deren Bedeutung:**
```

LinPEAS: Wichtige Befunde

1. SUID-Binaries (rot = kritisch)

→ Binaries mit SUID-Bit können root-Rechte erben

→ Weakness: Wenn ein SUID-Binary eine Schwachstelle hat,

kann ein Angreifer damit root werden

→ Aktion: Unnötige SUID-Bits entfernen

2. Writable /etc/passwd (rot = kritisch)

→ Jeder kann /etc/passwd beschreiben

→ Weakness: Angreifer kann eigenen root-Benutzer eintragen

→ Aktion: Berechtigungen korrigieren (0644)

3._world_writable files (gelb = mittel)

→ Dateien, die jeder beschreiben kann

→ Weakness: Angreifer kann Konfigurationsdateien ändern

→ Aktion: Berechtigungen korrigieren

4. Cleartext Passwords in Config-Files (rot = kritisch)

→ Passwörter in Klartext in Konfigurationsdateien

→ Weakness: Bei Compromise sofort lesbar

→ Aktion: Passwörter in Secrets-Backends verschieben

5. Insecure SSH Config (gelb = mittel)

→ PermitRootLogin yes, PasswordAuthentication yes

→ Weakness: Brute-Force oder root-Zugang über SSH

→ Aktion: SSH-Konfiguration härten



```
<blockquote class="infobox infobox--info">
□□**LinPEAS vs. manuelles Audit:** LinPEAS findet automatisch die häufigsten Schwachstellen. Aber es ersetzt kein manuelles Audit. Es prüft nicht auf: unbekannte Zero-Days, unternehmensspezifische Anforderungen, oder komplexe Angriffsvektoren. LinPEAS ist der ERSTE Schritt, nicht der letzte.
</blockquote>

### Audit-Kombination: Alle Tools zusammen

Ein vollständiges Security-Audit kombiniert mehrere Werkzeuge in Phasen:
```

```
┌──────────────────────────────────────────────────────────┐
│              Security-Audit: 3-Phasen-Modell             │
├──────────────────────────────────────────────────────────┤
│                                                          │
│  Phase 1: Automatisiert (Dauer: ~30 Min.)                │
│  ──────────────────────────────────────                  │
│  - lynis audit system                                    │
│  - checksec procAll                                      │
│  - LinPEAS (Privilege Escalation Scanner)                │
│  - auditd: ausearch -ts today                            │
│  - Output: Rohdaten, Befunde & Schwachstellen            │
│                                                          │
│  Phase 2: Manuell (Dauer: 2-4 Std.)                      │
│  ──────────────────────────────────                      │
│  - Kernel-Härtung & sysctl prüfen                        │
│  - Offene Ports & TLS-Zertifikate auditieren             │
│  - Benutzer-Rechte & SUID-Binaries prüfen                │
│  - Backup-Integrität & Restores verifizieren             │
│  - auditd-Regelsatz & Filter validieren                  │
│                                                          │
│  Phase 3: Bericht (Dauer: 1-2 Std.)                      │
│  ──────────────────────────────────                      │
│  - Befunde nach Kritikalität bewerten (Kritisch/Hoch/...)│
│  - Konkreten Maßnahmen- & Härtungsplan ableiten          │
│  - Revisionssicheres Archiv anlegen (3 Jahre)            │
```

```
    │  - Nächsten Audit-Termin terminieren (+6 Monate)           │
    │                                                            │
    └────────────────────────────────────────────────────────────┘
```

```
**Phase 1: Automatisiert**
```

Lynis-Ausgabe für Bericht speichern

sudo lynis audit system --no-colors 2>&1 | tee /tmp/lynis-report.txt

checksec für alle Prozesse

checksec --no-banner --no-warnings procAll 2>&1 | tee /tmp/checksec-report.txt

LinPEAS für Bericht speichern

sudo /tmp/linpeas.sh -a 2>&1 | tee /tmp/linpeas-report.txt

Audit-Events der letzten 24 Stunden

sudo ausearch -ts today --interpret 2>&1 | tee /tmp/audit-report.txt

```
**Phase 2: Manuell (Auswahl)**
```

Kernel-Version prüfen

uname -r

→ 7.1.9-hardened1-1-hardened ✓

Unbenutzte Benutzer prüfen

lastlog | grep "Never logged in"

→ Keine unbenutzten Benutzer ✓

Offene Ports prüfen

ss -tlnp

→ Nur SSH (22), nginx (80, 443) ✓

TLS-Zertifikate prüfen

openssl x509 -in /etc/nginx/certs/site.crt -checkend 2592000

→ "Certificate will expire in 365 days" ✓

```
### Praxis: Halbjähriger Security-Audit

**Schritt 1: Vorbereitung**
```

Audit-Datum und Scope festlegen

AUDIT_DATE=$(date +%Y-%m-%d) AUDIT_DIR="/root/audits/${AUDIT_DATE}" mkdir -p "$AUDIT_DIR"

Alle Reports in Audit-Verzeichnis speichern

echo "Security Audit gestartet: $AUDIT_DATE" > "$AUDIT_DIR/README.md" echo "Scope: Komplettes System" >> "$AUDIT_DIR/README.md"

```
**Schritt 2: Automatisierte Tools**
```

Lynis

sudo lynis audit system --no-colors 2>&1 | tee "$AUDIT_DIR/lynis.txt"

checksec

checksec --no-banner --no-warnings procAll 2>&1 | tee "$AUDIT_DIR/checksec.txt"

LinPEAS

sudo /tmp/linpeas.sh -a 2>&1 | tee "$AUDIT_DIR/linpeas.txt"

```
**Schritt 3: Manuelle Prüfungen**
```

Kernel

uname -r > "$AUDIT_DIR/kernel.txt"

Benutzer

lastlog > "$AUDIT_DIR/users.txt" awk -F: '($2 == "") {print $1}' /etc/shadow >> "$AUDIT_DIR/users.txt"

Netzwerk

ss -tlnp > "$AUDIT_DIR/ports.txt" nft list ruleset > "$AUDIT_DIR/firewall.txt"

Verschlüsselung

lsblk -f > "$AUDIT_DIR/encryption.txt"

Backups

restic snapshots --repo /mnt/nas/backups/$(hostname) > "$AUDIT_DIR/backups.txt" 2>&1

Audit

sudo auditctl -l > "$AUDIT_DIR/audit-rules.txt" sudo aureport > "$AUDIT_DIR/audit-summary.txt"

```
**Schritt 4: Bericht erstellen**
```

cat > "$AUDIT_DIR/bericht.md" << 'EOF'

Security Audit Bericht

Datum

$(date +%Y-%m-%d)

Zusammenfassung

Kategorie	Status	Bemerkung
Kernel	OK	linux-hardened aktiv
Benutzer	OK	Keine unbenutzten Accounts
Netzwerk	OK	Nur notwendige Ports offen
Verschlüsselung	OK	LUKS aktiv, Header-Backup vorhanden
Logging	OK	auditd + rsyslog aktiv
Backups	OK	Restic aktiv, Integrität geprüft

Befunde

Kritisch

- Keine

Hoch

- Keine

Mittel

- LinPEAS: 2 SUID-Binaries als unnötig identifiziert

Niedrig

- Kernel-Updates verfügbar (7.1.9 → 7.1.10)

Empfehlungen

1. [ ] Unnötige SUID-Bits entfernen

2. [ ] Kernel-Update durchführen
3. [ ] Nächster Audit: $(date -d "+6 months" +%Y-%m-%d)

EOF

```
```

Audit-Bereich archivieren

tar czf "/root/audits/${AUDIT_DATE}.tar.gz" "$AUDIT_DIR" echo "Audit abgeschlossen: $AUDIT_DIR"

```
<blockquote class="infobox infobox--info">
□□**Audit-Archivierung:** Speichere jeden Audit-Bericht mindestens 3 Jahre. Bei einem Sicherheitsvorfall oder einer Compliance-Prüfung kannst du nachweisen, dass regelmäßige Audits durchgeführt wurden.
</blockquote>

### Verifikation
```

Audit-Status prüfen

1. auditd aktiv?

sudo systemctl status auditd

active (running)

2. Audit-Regeln geladen?

sudo auditctl -l

-w /etc/shadow -p rwa -k shadow_access

...

3. Audit-Events vorhanden?

sudo ausearch -ts today --interpret | wc -l

> 0

4. Lynis-Bericht vorhanden?

ls -la /root/audits/

drwxr-x--- 2 root root 4096 Aug 27 12:34 2026-08-27

5. Nächster Audit-Termin?

echo "Nächster Audit: $(date -d '+6 months' +%Y-%m-%d)"

Nächster Audit: 2027-02-27

```
## Offizielle Dokumentation

| Thema | Dokumentation |
|-------|---------------|
| Arch Linux Wiki | [wiki.archlinux.org](https://wiki.archlinux.org/) |
| AppArmor | [apparmor.net](https://apparmor.net/) |
| linux-hardened | [github.com/anthraxx/linux-hardened](https://github.com/anthraxx/linux-
hardened) |
| nftables Wiki | [wiki.nftables.org](https://wiki.nftables.org/) |
| auditd | [github.com/linux-audit/audit-userspace](https://github.com/linux-audit/audit-
userspace) |
| Restic | [restic.net](https://restic.net/) |
| BorgBackup | [borgbackup.readthedocs.io](https://borgbackup.readthedocs.io/) |
| checksec | [github.com/slimm609/checksec.sh](https://github.com/slimm609/checksec.sh) |
| LinPEAS | [github.com/carlospolop/PEASS-ng](https://github.com/carlospolop/PEASS-ng) |
```

```
| systemd-cryptenroll | [freedesktop.org/software/systemd/man/systemd-
cryptenroll.html](https://www.freedesktop.org/software/systemd/man/systemd-
cryptenroll.html) |

## Weiterführende Artikel in dieser Serie

| Artikel | Thema |
|---------|-------|
| [Systemhärtung und Sicherheit](/de/arch-linux-serie/arch-linux-systemhaertung-
sicherheit-best-practices){.badge-link-text} | Grundlagen: Benutzerverwaltung, nftables,
SSH, Fail2Ban |
| [Best Practices und Tipps](/de/arch-linux-serie/arch-linux-best-practices-tipps){.badge-
link-text} | BTRFS, Snapper, snap-pac, grub-btrfs |
| [Paketmanagement mit Pacman](/de/arch-linux-serie/arch-linux-der-umfassende-leitfaden-
vom-paketmanager-pacman){.badge-link-text} | Paketquellen, Signaturen, Bereinigung |
| [Installation und Grundkonfiguration](/de/arch-linux-serie/archlinux-installation-und-
grundkonfiguration){.badge-link-text} | UEFI, Partitionierung, BTRFS, Verschlüsselung |
| [Grafische Benutzeroberfläche](/de/arch-linux-serie/arch-linux-installation-der-
grafischen-benutzeroberflaeche){.badge-link-text} | Wayland, PipeWire, SDDM, KDE Plasma 6
|

## Fazit: Das gehärtete Arch-Linux-Gesamtsystem

Mit dem Abschluss dieses sechsten Teils hast du den gesamten Weg von einem leeren
Speichermedium bis zu einer hochgradig abgesicherten, enterprise-tauglichen Linux-Umgebung
gemeistert. Arch Linux hat dabei bewiesen, dass es weit mehr als eine reine Bastel-
Distribution ist: Durch die absolute Transparenz und minimale Basis lässt sich ein
Sicherheitsniveau realisieren, das selbst vielen vorkonfigurierten Server-Distributionen
überlegen ist.

**Wir haben das System Schicht für Schicht gehärtet:**

1. **Fundament:** Sauberes Paketmanagement mit `pacman`, verschlüsselte BTRFS-Subvolumes
und sicheres Booten via GRUB und TPM2.
2. **Betrieb:** Wartungsarme Automatisierung mit Systemd-Timern und ausfallsichere
Rollbacks durch `snapper` und `grub-btrfs`.
3. **Defense in Depth:** Mandatory Access Control mit `AppArmor`, Kernel-Härtung durch
`linux-hardened` und ASLR-Optimierung.
4. **Isolation:** Container-Netzwerk-Segmentierung mit `nftables` ohne gefährliche Docker-
Defaults.
5. **Observability & Resilienz:** Manipulationssicheres zentrales TLS-Logging,
revisionssichere `auditd`-Überwachung und eine unveränderliche 3-2-1-Backup-Strategie.

<blockquote class="infobox infobox--info">
□□**Abschließender Praxis-Tipp:** Sicherheit ist kein statischer Zustand, sondern ein
```

kontinuierlicher Prozess. Halte deine Audit-Routinen halbjährlich ein, prüfe Audit-Logs regelmäßig auf Anomalien und teste deine Backup-Restores mindestens einmal im Quartal unter realen Bedingungen.
</blockquote>

Du besitzt nun ein tiefgreifendes Verständnis für die Architektur moderner Linux-Systeme und die Werkzeuge, um Produktivsysteme nach höchsten Sicherheitsstandards zu konzipieren, zu betreiben und zu verteidigen.

Quelle: https://admindocs.de/de/arch-linux-serie/arch-linux-fortgeschrittene-sicherheitsfeatures-und-wartung