ADMIN:DOCS

Linux Server Härtung: SSH mit FIDO2/YubiKey & CrowdSec

Linux-Server-Härtung mit SSH FIDO2/YubiKey für passwortlose Authentifizierung und CrowdSec als kollaboratives Intrusion-Prevention-System auf Ubuntu 24.04/26.04.

Kategorie: Server environments **Veröffentlicht:** 26.08.2026

SSH-Brute-Force-Attacken laufen auf jedem öffentlichen Server im Dauerbetrieb. Der Großteil der Linux-Server setzt standardmäßig auf Passwörter oder statische SSH-Keys – beides hat bekannte Schwächen. Passwörter sind durch Credential-Stuffing und GPU-Cluster in Minuten geknackt, statische Keys bleiben unverändert, bis jemand sie manuell rotiert.

Dieser Artikel zeigt, wie du auf **Ubuntu 24.04 LTS** und **Ubuntu 26.04 LTS** deinen Server mit zwei konkreten Maßnahmen absicherst: **SSH mit FIDO2 und YubiKey** für hardwarebasierte, passwortlose Authentifizierung und **CrowdSec** als kollaboratives Intrusion-Prevention-System. Beides zusammen schließt die häufigsten Angriffsvektoren, ohne dass du dich um Passwort-Rotation oder Key-Management kümmern musst.

Die Konfiguration funktioniert auf einem frischen Server genauso wie auf einem bestehenden Setup – wir machen keine Annahmen über den aktuellen Zustand.

> **Voraussetzungen:** Grundkenntnisse in `SSH` und `systemd`. Ein Ubuntu 24.04 oder 26.04 Server mit Root- oder sudo-Zugang. Für FIDO2 benötigst du einen YubiKey (oder ein anderes FIDO2-Gerät). Für CrowdSec reicht ein TCP-Port für SSH. Der Artikel ist für Linux-Administratoren, DevOps Engineers und Homelab-Betreiber, die bestehende Server absichern oder neue nach aktuellem Standard aufsetzen möchten.

> ⚠ **Hinweis:** Dieser Artikel behandelt Serverhärtung auf Betriebssystem-Ebene. Anwendungsspezifische Härtung (Webserver, Datenbanken) bleibt außen vor, da jede Anwendung eigene Anforderungen hat.

SSH mit FIDO2 & YubiKey

Warum FIDO2 für SSH?

Passwörter sind das schwächste Glied in der SSH-Kette. Jeder Server mit öffentlicher IP wird permanent von Brute-Force-Scannern bearbeitet – `fail2ban`-Logs zeigen das täglich. Statische SSH-Keys (Ed25519, RSA) sind besser, aber sie rotieren nicht. Wer einen kompromittierten Private Key hat, bleibt unbemerkt aktuell.

FIDO2 löst beide Probleme:

Der Private Key wird auf einem physischen Gerät gespeichert (YubiKey) und verlässt dieses nie. Ohne den YubiKey am lokalen Rechner ist kein Login möglich – egal wie gut Passwort oder Key sind. Zusätzlich erzwingt FIDO2 eine Benutzer-Verifikation (PIN oder Biometrie) bei jedem Login.

Was sich ändert:

Eigenschaft	Passwort	Statischer SSH-Key	FIDO2 + YubiKey
Physische Komponente	Nein	Nein	Ja
Brute-Force-resistent	Bedingt	Ja	Ja
Phishing-resistent	Nein	Bedingt	Ja
Rotation	Manuell	Manuell	Automatisch (Key auf Gerät)
Multi-Factor	Optional	Nein	Standard

Die Kombination aus physischem Token und PIN macht SSH-Logins nahezu unangreifbar. Ein Angreifer bräuchte sowohl den YubiKey als auch den PIN – und beides an derselben physischen Station.

□□ **FIDO2 vs. U2F:** FIDO2 ist der Nachfolger von U2F und unterstützt WebAuthn (Browser) und CTAP2 (native Apps). Für SSH ist CTAP2 relevant – der YubiKey wird direkt vom SSH-Client angesprochen, ohne Browser-Umweg.

FIDO2 & WebAuthn: Wie es funktioniert

FIDO2 arbeitet Challenge-Response-basiert. Der Server sendet eine zufällige Challenge, der YubiKey signiert sie mit einem auf dem Gerät gespeicherten privaten Schlüssel, und der Server verifiziert die Signatur. Der private Schlüssel verlässt nie den YubiKey.



```
└──────────────────────────────────────────────────┘
```

Was passiert im Detail:

1. Der SSH-Client erkennt, dass ein FIDO2-Key konfiguriert ist
2. Er fordert den YubiKey zur Signierung der Challenge auf
3. Der Nutzer gibt seinen PIN ein oder berührt den Sensor
4. Der YubiKey signiert die Challenge mit dem ed25519-sk Schlüsselpaar
5. Der Server verifiziert die Signatur gegen den gespeicherten öffentlichen Schlüssel
6. Der Login wird gewährt

Der entscheidende Unterschied zu statischen Keys: Die Challenge ist bei jedem Login-Versuch anders. Selbst wenn jemand die Signatur abfängt, kann er sie nicht wiederverwenden (Replay-Schutz).

YubiKey Hardware-Setup

Für SSH mit FIDO2 eignen sich alle YubiKeys der 5er-Serie mit USB-C oder USB-A. Wichtig ist, dass der Key **FIDO2** und nicht nur U2F unterstützt – das erkennst du an der Produktbezeichnung "YubiKey 5" (nicht "YubiKey Security Key", der nur U2F kann).

YubiKey initialisieren:

```
# YubiKey Manager installieren (Debian/Ubuntu)
sudo apt install yubikey-manager -y

# FIDO2-PIN setzen (Standard-PIN ist leer)
ykman fido access change-pin
```

Der PIN schützt den YubiKey vor unbefugter Nutzung. Wähl einen PIN mit mindestens 6 Zeichen. Ohne PIN kann jeder mit physischem Zugriff zum YubiKey die FIDO2-Keys auslesen und nutzen.

Verifikation:

```
# Prüfe ob der YubiKey erkannt wird
ykman fido info
```

Erwartete Ausgabe:

```
Applications: FIDO2
    slots: 2
     pin: true
     pin retries: 3
```

> ⚠ **Backup-YubiKey & Break-Glass-Notfallanker:** Erstelle immer mindestens einen zweiten YubiKey als physisches Backup an einem sicheren Ort. Wenn du deinen einzigen YubiKey verlierst, bist du ausgesperrt – es gibt keinen Reset-Mechanismus für FIDO2-Keys. Für Root- und Cloud-Server empfiehlt sich zudem die Absicherung der Out-of-Band-Webkonsole des Hosters (z. B. Hetzner Robot, Proxmox NoVNC oder IPMI) mit separatem 2FA als Notfallanker.

SSH-Schlüssel mit FIDO2 erzeugen

OpenSSH 8.2+ unterstützt FIDO2-Schlüssel nativ. Der Befehl `ssh-keygen` erzeugt mit der Option `-t ed25519-sk` einen Schlüssel, der auf dem YubiKey gespeichert wird.

Schlüssel erzeugen:

```
# FIDO2-SSH-Schlüssel erzeugen
ssh-keygen -t ed25519-sk -C "admin@server"
```

Während der Erzeugung wirst du aufgefordert, den YubiKey zu berühren (Touch) und deinen FIDO2-PIN einzugeben. Der private Schlüssel wird auf dem YubiKey gespeichert, nicht auf der Festplatte.

Was passiert bei der Erzeugung:

1. `ssh-keygen` erzeugt ein ed25519-sk Schlüsselpaar
2. Der private Schlüssel wird im Secure Element des YubiKeys abgelegt
3. Der öffentliche Schlüssel wird in `~/.ssh/id_ed25519-sk.pub` gespeichert
4. Für jeden Login-Versuch wird der YubiKey benötigt

Optionen:

```
# Ohne Touch (nur PIN)
ssh-keygen -t ed25519-sk -O no-touch-required

# Mit benutzerdefiniertem Dateinamen
ssh-keygen -t ed25519-sk -f ~/.ssh/id_yubikey -C "admin@server"
```

> □□ **Resident vs. Non-Resident Keys:** Mit `-O resident` wird der private Schlüssel vollständig auf dem YubiKey gespeichert und kann von dort exportiert werden. Ohne diese Option wird nur ein Verweis gespeichert. Für SSH empfiehlt sich Non-Resident (Standard), da der Schlüssel dann nicht exportierbar ist.

Öffentlichen Schlüssel auf den Server kopieren:

```
# Öffentlichen Schlüssel auf den Server übertragen
ssh-copy-id -i ~/.ssh/id_ed25519-sk.pub user@server
```

Falls `ssh-copy-id` mit FIDO2 Probleme macht (manchmal der Fall bei älteren OpenSSH-Versionen), kopiere den Schlüssel manuell:

```
# Manuelle Installation
cat ~/.ssh/id_ed25519-sk.pub | ssh user@server "mkdir -p ~/.ssh && cat >>
~/.ssh/authorized_keys"
```

Überprüfung auf dem Server:

```
# Prüfe den installierten Schlüssel
cat ~/.ssh/authorized_keys | grep "sk@"
```

Die Zeile sollte mit `ssh-ed25519-cert-v01@openssh.com` beginnen oder `sk@` enthalten – das bestätigt, dass es sich um einen FIDO2-Schlüssel handelt.

SSH-Server für FIDO2 konfigurieren

Der SSH-Server muss wissen, dass er FIDO2-Schlüssel akzeptiert. In der Standardkonfiguration ist das bereits der Fall, aber wir wollen explizit sein und zusätzliche Sicherheitseinstellungen setzen.

Server-Konfiguration:

```
sudo nano /etc/ssh/sshd_config
```

Relevante Einstellungen:

```
# FIDO2-Schlüssel erlauben (Standard: yes)
PubkeyAuthentication yes

# Nur SSH-Keys erlauben (Passwörter deaktivieren)
PasswordAuthentication no

# Root-Login verbieten
PermitRootLogin no

# Maximal 3 Authentifizierungsversuche
MaxAuthTries 3

# Grace Time auf 30 Sekunden begrenzen
LoginGraceTime 30

# Information Disclosure minimieren
DebianBanner no
Banner none

# Keepalive für Long-Running Sessions
ClientAliveInterval 300
ClientAliveCountMax 2
```

Optional: Nur FIDO2-Keys erlauben (keine statischen Keys mehr):

Wenn du ausschließlich FIDO2-Key verwenden möchtest, kannst du die `PubkeyAuthOptions` einschränken:

```
# Nur verifizierte FIDO2-Keys akzeptieren
PubkeyAuthOptions verify-required
```

> ⚠ **Vorsicht mit `verify-required`:** Diese Option akzeptiert nur FIDO2-Keys mit Benutzer-Verifikation (PIN oder Biometrie). Statische Ed25519- und RSA-Keys werden abgelehnt. Teste die Änderung zuerst in einer zusätzlichen Session, bevor du die bestehende schließt.

SSH-Server neu starten:

```
sudo systemctl restart sshd
```

> ⚠ **Nie die aktive Session schließen, bevor du getestet hast!** Öffne ein zweites Terminal und teste den Login mit dem neuen FIDO2-Key. Erst wenn der Login funktioniert, schließt du die alte Session.

Testen:

```
# Login mit FIDO2-Key
ssh -i ~/.ssh/id_ed25519-sk user@server

# Prüfe, welche Methode verwendet wurde
ssh -v user@server 2>&1 | grep "Offering"
```

Die Verbose-Ausgabe sollte `Offering public key: .../id_ed25519-sk` zeigen und `Accepted publickey` als Ergebnis.

Zertifikatsbasierte Authentifizierung

Für größere Umgebungen lohnt sich eine SSH Certificate Authority (CA). Statt einzelne Keys auf jedem Server zu verteilen, signiert eine CA öffentliche Schlüssel und der Server vertraut nur der CA.

CA auf dem signierenden Host erstellen:

```
# CA-Schlüsselpaar erzeugen
sudo mkdir -p /etc/ssh/ca
sudo ssh-keygen -t ed25519 -f /etc/ssh/ca/server-ca -C "SSH CA"
```

Öffentlichen CA-Schlüssel auf allen Servern verteilen:

```
# Auf jedem Server: CA Public Key als TrustedKey eintragen
echo "cert-authority $(cat /etc/ssh/ca/server-ca.pub)" >>
/etc/ssh/authorized_keys.d/trusted-ca
```

Benutzer-Key mit CA signieren:

```
# FIDO2-Key signieren (erzeugt ein Zertifikat)
sudo ssh-keygen -s /etc/ssh/ca/server-ca \
    -I "admin-fido2" \
```

```
    -n admin \
    -V +52w \
    ~/.ssh/id_ed25519-sk.pub
```

Optionen:

Option	Bedeutung
`-s`	CA-Private Key zum Signieren
`-I`	Zertifikat-ID (für Logs)
`-n`	Erlaubte Usernames (kommagetrennt)
`-V`	Gültigkeitsdauer (`+52w` = 52 Wochen)

Das erzeugte Zertifikat (`id_ed25519-sk-cert.pub`) wird vom SSH-Client automatisch erkannt und beim Login mitgeschickt.

> **Vorteil der CA:** Du musst auf jedem Server nur den öffentlichen CA-Schlüssel hinterlegen. Individuelle Benutzer-Keys werden zentral verwaltet und können über die CA widerrufen werden, ohne dass du auf jedem Server die `authorized_keys` editierst.

SSH-Config & ssh-agent

Für den täglichen Gebrauch lohnt sich eine saubere SSH-Konfiguration auf dem Client.

`~/.ssh/config`:

```
# Server-Definition mit FIDO2-Key
Host server
    HostName 192.168.1.100
```

```
    User admin
    IdentityFile ~/.ssh/id_ed25519-sk
    IdentitiesOnly yes

# Jump-Host für interne Server
Host internal-server
    HostName 10.0.0.50
    User admin
    ProxyJump server
    IdentityFile ~/.ssh/id_ed25519-sk
    IdentitiesOnly yes
```

`IdentitiesOnly yes` zwingt den SSH-Client, nur den angegebenen Key zu verwenden. Das verhindert, dass versehentlich ein anderer Key aus dem Agent angeboten wird.

ssh-agent mit FIDO2:

Der `ssh-agent` kann FIDO2-Keys unterstützen, aber da der private Schlüssel auf dem YubiKey bleibt, wird nur der Verweis im Agent gespeichert. Für FIDO2 macht der Agent wenig Sinn – der YubiKey muss sowieso physisch angeschlossen sein.

```
# ssh-agent starten (optional für FIDO2)
eval $(ssh-agent -s)

# Key hinzufügen (Touch wird bei jedem Login angefordert)
ssh-add ~/.ssh/id_ed25519-sk
```

> □□ **Empfehlung:** Für FIDO2-Keys den ssh-agent weglassen. Der Agent bringt keinen Vorteil, da der YubiKey bei jedem Login geberührt werden muss. Stattdessen die `IdentityFile` in `~/.ssh/config` nutzen.

Troubleshooting: Häufige SSH-FIDO2-Probleme

Problem: "Key rejected by server"

```
# Prüfe ob FIDO2 im SSH-Server unterstützt wird
ssh -V  # OpenSSH_8.9 oder höher

# Prüfe die Server-Konfiguration
sudo sshd -T | grep pubkeyauth
```

> ⚠ **Ursache: OpenSSH-Version zu alt (unter 8.2). FIDO2-Support kam mit OpenSSH 8.2. Ubuntu 24.04 hat OpenSSH 9.6, Ubuntu 26.04 hat OpenSSH 9.7 – beide unterstützen FIDO2.

Problem: "Sign_and_send_pubkey: signing failed: agent refused operation"

```
# ssh-agent läuft und versucht den Key zu signieren
# Lösung: Key explizit angeben oder Agent deaktivieren
ssh -i ~/.ssh/id_ed25519-sk user@server
```

Ursache: Der ssh-agent versucht, den Key mit einer alten Methode zu signieren. `IdentitiesOnly yes` in der SSH-Config löst das Problem.

Problem: "Touch your authenticator" erscheint nicht

```
# Prüfe ob der YubiKey erkannt wird
lsusb | grep Yubico

# Prüfe ob das FIDO2-Modul geladen ist
ls /dev/hidraw*
```

Ursache: YubiKey nicht erkannt oder Treiber fehlen. Unter Ubuntu sind alle nötigen Treiber standardmäßig vorhanden.

Problem: "PIN auth required" aber PIN wird nicht abgefragt

```
# FIDO2-PIN manuell setzen oder zurücksetzen
ykman fido access change-pin
```

> ⚠ **Ursache:** Der YubiKey wurde ohne PIN initialisiert oder der PIN ist veraltet.

Verifikation auf dem Server:

```
# Logs prüfen
journalctl -u sshd -f

# Erfolgreiche FIDO2-Logins erkennen
journalctl -u sshd | grep "Accepted publickey"
```

Die Logs zeigen, welcher Key verwendet wurde und ob es ein FIDO2-Key (`sk@`) war.

CrowdSec: Kollaborativer Intrusion-Prevention

Fail2Ban vs. CrowdSec

Fail2Ban liest lokalen Log-Dateien und bannt IPs über iptables/nftables-Regeln. Jeder Server agiert isoliert – ein Angreifer, der von 1.000 IPs scannt, muss auf jedem Server einzeln gebannt werden.

CrowdSec geht anders vor: Der Agent analysiert Logs und Entscheidungen (Decisions) werden zentral in einer Community-Datenbank geteilt. Wenn ein Angreifer auf einem Community-Server gebannt wird, profitieren alle anderen davon. Zusätzlich trennt CrowdSec die Analyse (Agent) von der Durchsetzung (Bouncer), was die Architektur flexibler macht.

Was sich ändert:

Eigenschaft	Fail2Ban	CrowdSec
Log-Analyse	Lokal	Lokal + Community
Durchsetzung	iptables direkt	Bouncer (extern)
Performance	Guter bei wenigen Jails	Skalierbar für viele Server
Community	Keine	Geteilte Decisions
Dashboard	Kein	Web-Dashboard + Metriken
Architektur	Monolith	Agent + Bouncer getrennt

> **Kernidee:** CrowdSec sammelt Signaturen von Angriffsmustern (Scenarios) und vergleicht sie mit deinen Logs. Treffer führen zu Decisions (Bans), die von Bouncers durchgesetzt werden. Die Community-Intelligence ergänzt lokale Erkennung.

Installation & Ersteinrichtung

CrowdSec Agent installieren:

```
# Repository hinzufügen (Debian/Ubuntu)
curl -s https://packagecloud.io/install/repositories/crowdsec/crowdsec/script.deb.sh |
sudo bash

# CrowdSec installieren
sudo apt install crowdsec -y
```

Firewall-Bouncer installieren:

```
# Bouncer für nftables/iptables
sudo apt install crowdsec-firewall-bouncer-nftables -y
```

> ⚠ **Reihenfolge beachten:** Zuerst den Agent installieren, dann den Bouncer. Der Bouncer verbindet sich mit dem Agenten über die lokale API. Starte den Bouncer nicht, bevor der Agent läuft.

Status prüfen:

```
sudo systemctl status crowdsec
sudo systemctl status crowdsec-firewall-bouncer
```

Erwartete Ausgabe für den Agenten:

```
● crowdsec.service - CrowdSec SIEM
     Loaded: loaded (/lib/systemd/system/crowdsec.service; enabled)
     Active: active (running) since Mon 2026-08-26 12:00:00 UTC
```

Erste Decisions prüfen:

```
sudo cscli decisions list
```

Im Anfangszustand ist die Liste leer. Sobald der Agent Logs analysiert und Angriffsmuster erkennt, erscheinen hier Decisions.

Szenarien & Entscheidungen

CrowdSec arbeitet mit Scenarios – das sind Signaturen, die bestimmte Angriffsmuster beschreiben. Der Agent vergleicht Log-Einträge mit diesen Scenarios und erstellt Decisions bei Treffern.

Standardmäßig aktivierte Scenarios:

```
sudo cscli scenarios list
```

Name	Service	Type

`ssh-bf`	ssh	leak
`ssh-slow-bf`	ssh	leak
`http-bad-user-agent`	http	leak
`http-dos-bf-path`	http	leak

Für SSH relevante Scenarios:

Scenario	Erkennung	Aktion
`ssh-bf`	5 fehlgeschlagene Logins in 10 Min	Ban für 1h
`ssh-slow-bf`	Langsamer Brute-Force über Stunden	Ban für 24h
`ssh-crawl`	Verhaltensanalyse von Login-Versuchen	Ban für 4h

Eigene Scenarios erstellen:

Falls die Standard-Scenarios nicht passen, kannst du eigene definieren:

```
sudo nano /etc/crowdsec/scenarios/custom-ssh-aggressive.yaml
```

```
type: leak
name: ssh-aggressive-scan
description: "Aggressive SSH-Scans mit vielen fehlgeschlagenen Logins"
labels:
  service: ssh
  remediation: true
filter:
  LABELS.service == "ssh" && MSG.ContainerName == "sshd"
```

```
groupby: LABELS.source_ip
capacity: 5
leakspeed: 1m
blackhole: 1m
certainty: 75
for: 10m
```

Verifikation:

```
# Scenarios neu laden
sudo cscli scenarios list | grep custom

# Testszenario simulieren
sudo cscli scenarios install --local /etc/crowdsec/scenarios/custom-ssh-aggressive.yaml
```

Bouncers (Firewall-Integration)

Bouncers setzen die Decisions des Agents durch. Es gibt verschiedene Bouncer-Typen für unterschiedliche Anwendungsfälle.

nftables-Bouncer (Standard):

```
# Prüfe ob der Bouncer aktiv ist
sudo cscli bouncers list
```

```
Name       | IP Address | Valid Until | Type
-----------+------------+-------------+------
firewall   | 127.0.0.1  | 2027-01-01  | nftables
```

Bouncer-Typen:

Bouncer	Einsatz	Vorteil
`nftables`	Server-Firewall	Schnell, kernel-nah
`nginx`	Webserver-Inline	Kein extra Port
`caddy`	Caddy-Integration	Einfache Konfiguration
`crowdsec-openapi`	Externe APIs	Flexibel

Firewall-Regeln prüfen:

```
# CrowdSec-Chain in nftables prüfen
sudo nft list chain inet crowdsec-crowdsec-block
```

```
table inet crowdsec-crowdsec {
    chain crowdsec-block {
        type filter hook input priority filter; policy accept;
        ip saddr @crowdsec-blacklist-jump drop
        ip6 saddr @crowdsec-blacklist6-jump drop
```

```
    }
}
```

Whitelisting

Nicht jede IP soll gebannt werden. Lokale Server, Monitoring-Systeme oder vertrauenswürdige Netzwerke brauchen Whitelisting.

Whitelist-Datei:

```
sudo nano /etc/crowdsec/parsers/s02-enrich/whitelist.yaml
```

```
filter: "Meta.IP != ''"
whitelist:
  reason: "Lokale Adresse"
  expression: "Ip('127.0.0.1') || Ip('::1')"
  expression: "Ip('192.168.0.0/16')"
  expression: "Ip('10.0.0.0/8')"
```

Alternativ: Whitelist über cscli:

```
# Einzelne IP whitelishten
sudo cscli decisions add --ip 192.168.1.100 --duration 0 --type whitelist --reason
"Monitoring"
```

```
# IP-Range whitelisten
sudo cscli decisions add --range 192.168.1.0/24 --duration 0 --type whitelist --reason
"Lokales Netz"
```

> ⚠ **Whitelist vor Bans prüfen:** Prüfe immer, ob eine IP möglicherweise aus einem lokalen Monitoring-System oder einem vertrauenswürdigen Netzwerk kommt, bevor du Banns analysierst. Sonst bannt CrowdSec versehentlich eigene Systeme.

Dashboards & Alerting

CrowdSec bietet ein Web-Dashboard und Metriken für Monitoring-Systeme.

Web-Dashboard:

```
# Dashboard-Zugangsdaten anzeigen
sudo cscli console status
```

Das Dashboard ist auf `http://localhost:6060` erreichbar (Standard-Port). Für externe Zugriffe die Firewall anpassen.

Metriken für Prometheus:

```
# Metriken-Endpunkt prüfen
curl -s http://localhost:6060/metrics | head -20
```

```
# Wichtige Metriken
crowdsec_decisions_total{source_ip="...",scenario="..."}
crowdsec_alerts_total{scenario="..."}
crowdsec_bouncers_total{type="..."}
```

Alerting über Webhook:

```
sudo nano /etc/crowdsec/notifications/email.yaml
```

```
type: email
name: email_default
log_level: info
 # Email-Konfiguration
smtp_host: "smtp.example.com"
smtp_port: 587
smtp_username: "alert@example.com"
smtp_password: "DEIN_PASSWORT"
smtp_from: "crowdsec@example.com"
smtp_to:
  - "admin@example.com"
smtp_subject: "CrowdSec Alert: {scenario}"
format: |
  CrowdSec hat einen Angriff erkannt.
  Scenario: {scenario}
  Source IP: {source_ip}
  Entscheidung: {decision}
  Zeit: {created_at}
```

Verifikation:

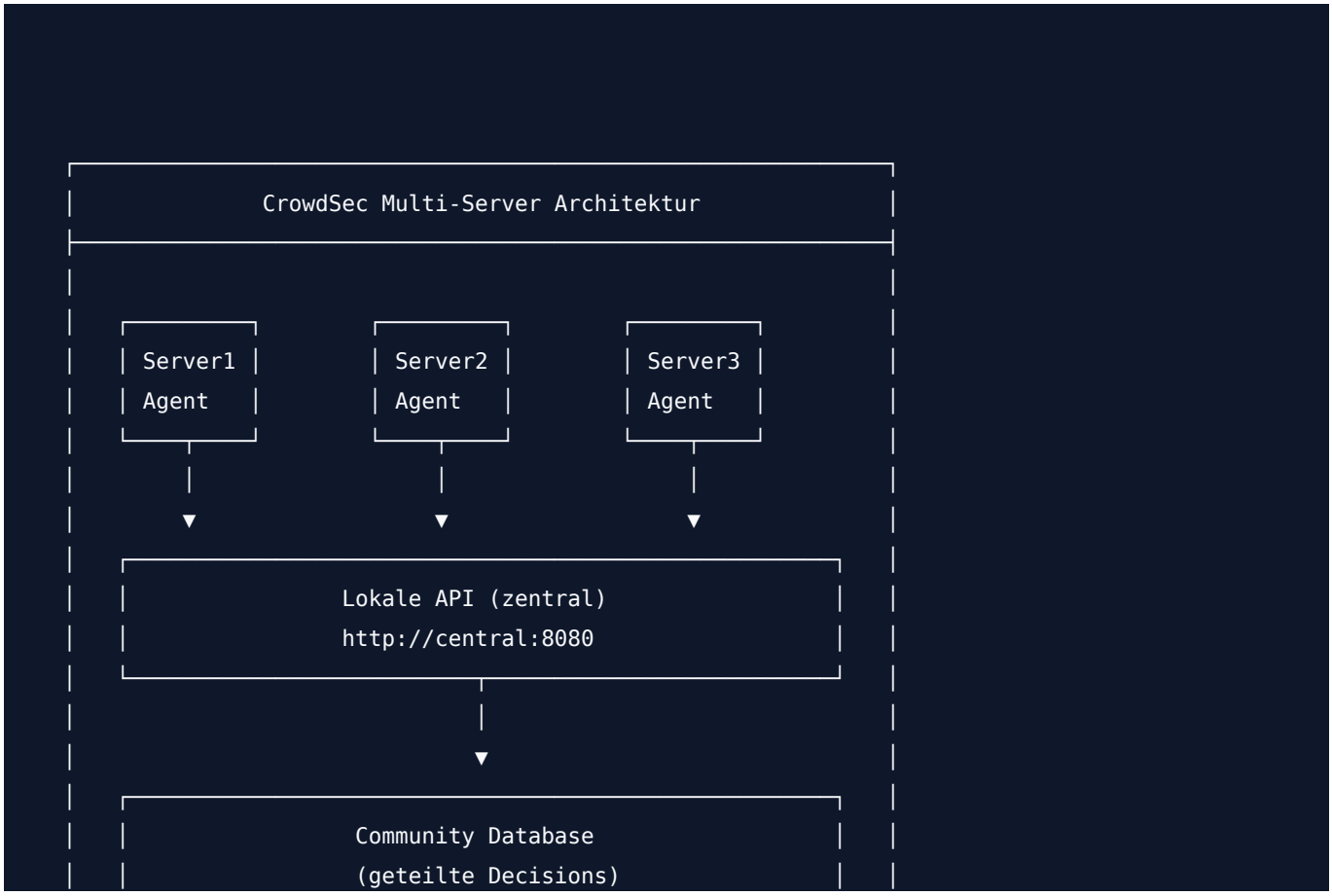
```
# Notifications prüfen
sudo cscli notifications list

# Test-Benachrichtigung senden
sudo cscli alert inspect --id 1
```

Performance & Skalierung

Für größere Umgebungen (mehr als 10 Server) lohnt sich eine zentrale CrowdSec-Instanz.

Architektur für mehrere Server:





Lokalen Agent auf zentrale API zeigen:

```
sudo nano /etc/crowdsec/config.yaml
```

```
api:
  client:
    insecure_skip_verify: false
  server:
    listen_uri: http://127.0.0.1:8080
```

> **Performance:** Ein einzelner CrowdSec-Agent kann problemlos 10.000+ Log-Einträge pro Sekunde verarbeiten. Für die meisten Setups reicht ein zentraler Agent auf einem dedizierten Server.

Troubleshooting

Problem: "crowdsec service not running"

```
# Logs prüfen
sudo journalctl -u crowdsec -f

# Häufige Ursache: Port 8080 belegt
sudo ss -tulnp | grep 8080
```

Problem: "Bouncer nicht verbunden"

```
# Bouncer-API-Token prüfen
sudo cscli bouncers list

# Token neu generieren
sudo cscli bouncers add my-bouncer --auto
```

Problem: Eigene IPs werden gebannt

```
# Whitelist prüfen
sudo cscli decisions list --type whitelist

# Lokale IP zur Whitelist hinzufügen
sudo cscli decisions add --ip 192.168.1.100 --duration 0 --type whitelist --reason
"Eigener Server"
```

Verifikation:

```
# Aktive Decisions prüfen
sudo cscli decisions list

# Logs des Agenten prüfen
sudo journalctl -u crowdsec | grep "decision"
```

System-Härtung

Kernel-Parameter (sysctl)

Der Linux-Kernel bietet über `sysctl` Dutzende von Parametern, die Angriffsfläche minimieren. Die Standardeinstellungen sind auf Kompatibilität optimiert, nicht auf Sicherheit.

Datei erstellen:

```
sudo nano /etc/sysctl.d/99-hardening.conf
```

Netzwerk-Sicherheit:

```
# IP-Spoofing verhindern
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1

# ICMP-Redirects deaktivieren
net.ipv4.conf.all.accept_redirects = 0
```

```
net.ipv4.conf.default.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0

# Source-Routing deaktivieren
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
net.ipv6.conf.all.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0

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

# Log Martian-Pakete
net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.default.log_martians = 1
```

Speicher-Schutz:

```
# Address Space Layout Randomization
kernel.randomize_va_space = 2

# Dmesg einschränken (nur für Root)
kernel.dmesg_restrict = 1

# Kernel-Pointer verbergen
kernel.kptr_restrict = 2

# BPF einschränken
kernel.unprivileged_bpf_disabled = 1

# User namespaces einschränken (für Container)
kernel.unprivileged_userns_clone = 0
```

Kernel-Laden verhindern:

```
# Module nicht mehr laden können (nach Boot)
kernel.modules_disabled = 1
```

> ⚠ `kernel.modules_disabled = 1` verhindert das Laden neuer Kernel-Module nach dem Boot. Das kann etwa NVIDIA-Treiber oder VirtualBox-Module brechen. Nur aktivieren, wenn du sicher bist, dass alle nötigen Module beim Boot geladen werden.

Parameter laden:

```
sudo sysctl -p /etc/sysctl.d/99-hardening.conf
```

Verifikation:

```
# Prüfe ob Parameter gesetzt sind
sysctl net.ipv4.conf.all.rp_filter
sysctl kernel.randomize_va_space
```

Erwartete Ausgabe:

```
net.ipv4.conf.all.rp_filter = 1
kernel.randomize_va_space = 2
```

Filesystem-Hardening

Standardmäßig haben viele Dateien und Verzeichnisse zu weite Berechtigungen. Besonders `/tmp`, `/var/tmp` und `/dev/shm` sind Angriffsflächen.

Mount-Optionen für `/tmp`:

```
# /tmp ist oft bereits ein tmpfs, prüfe mit:
mount | grep tmp
```

Falls `/tmp` auf der Hauptfestplatte liegt, empfiehlt sich eine dedizierte Partition mit sicheren Mount-Optionen:

```
# In /etc/fstab eintragen (bei separater /tmp-Partition)
# UUID=... /tmp ext4 defaults,noexec,nosuid,nodev 0 2
```

Verzeichnisrechte strikt setzen:

```
# Kritische Verzeichnisse absichern
sudo chmod 700 /root
sudo chmod 600 /etc/shadow
sudo chmod 600 /etc/gshadow
sudo chmod 644 /etc/passwd
sudo chmod 644 /etc/group

# SSH-Verzeichnis
sudo chmod 700 /root/.ssh
sudo chmod 600 /root/.ssh/authorized_keys
```

SUID/SGID-Bits reduzieren:

```
# SUID-Bits auf Standard-Tools prüfen
find / -perm -4000 -type f 2>/dev/null
```

Unnötige SUID-Bits entfernen (Beispiele):

```
# Nur wenn die Programme nicht als SUID benötigt werden
sudo chmod u-s /usr/bin/newgrp
sudo chmod u-s /usr/bin/chsh
sudo chmod u-s /usr/bin/chfn
```

> **Tipp:** `find / -perm -4000 -type f` zeigt alle Programme mit SUID-Bit. Nicht blind entfernen – manche Programme brauchen das Bit für ihre Funktion (z. B. `sudo`, `passwd`).

Service-Minimierung

Jeder laufende Dienst ist eine potenzielle Angriffsfläche. Auf einem frischen Ubuntu-Server laufen Dutzende Services, die du möglicherweise nicht brauchst.

Aktive Services prüfen:

```
# Alle laufenden Services anzeigen
systemctl list-units --type=service --state=running

# Häufig unnötige Services auf einem Server
systemctl list-units --type=service --state=running | grep -E "cups|bluetooth|avahi|modem"
```

Unnötige Services deaktivieren:

```
# Drucker-Service (auf einem Server nicht benötigt)
sudo systemctl stop cups
sudo systemctl disable cups
sudo systemctl mask cups

# Bluetooth (auf einem Server nicht benötigt)
sudo systemctl stop bluetooth
sudo systemctl disable bluetooth
sudo systemctl mask bluetooth

# mDNS/Avahi (Service Discovery – auf einem Server unnötig)
sudo systemctl stop avahi-daemon
sudo systemctl disable avahi-daemon
sudo systemctl mask avahi-daemon
```

`mask` vs. `disable`:

Aktion	Effekt
`disable`	Service startet nicht automatisch, kann aber manuell gestartet werden
`mask`	Service kann überhaupt nicht gestartet werden (Symlink auf `/dev/null`)

> ⚠ **Nur masken, was du wirklich nicht brauchst.** Falls du später doch einen Service brauchst, musst du erst `systemctl unmask` ausführen.

Systemd-Security-Score prüfen:

```
# Security Score für einen Service anzeigen
systemd-analyze security sshd.service
```

```
→ Overall exposure level for sshd.service: 9.6 OK
```

Je niedriger der Wert, desto besser. `sshd` hat normalerweise einen guten Score, wenn `PasswordAuthentication no` und `PermitRootLogin no` gesetzt sind.

Audit-Logging

`auditd` protokolliert sicherheitsrelevante Ereignisse und speichert sie in einer manipulationsresistenten Log-Datei. Das ist die Grundlage für Forensik und Compliance.

Installation:

```
sudo apt install auditd -y
sudo systemctl enable auditd
```

Regeln für SSH und sudo:

```
# SSH-Konfiguration überwachen
sudo auditctl -w /etc/ssh/sshd_config -p wa -k sshd_config

# Authorized-Keys überwachen
sudo auditctl -w /root/.ssh/authorized_keys -p wa -k ssh_keys

# sudo-Konfiguration überwachen
sudo auditctl -w /etc/sudoers -p wa -k sudoers
sudo auditctl -w /etc/sudoers.d/ -p wa -k sudoers

# Crontab überwachen
sudo auditctl -w /etc/crontab -p wa -k cron
sudo auditctl -w /var/spool/cron/ -p wa -k cron
```

Regeln dauerhaft speichern:

```
sudo nano /etc/audit/rules.d/hardening.rules
```

```
# SSH
-w /etc/ssh/sshd_config -p wa -k sshd_config
-w /root/.ssh/authorized_keys -p wa -k ssh_keys

# Sudo
-w /etc/sudoers -p wa -k sudoers
-w /etc/sudoers.d/ -p wa -k sudoers

# Cron
-w /etc/crontab -p wa -k cron
-w /var/spool/cron/ -p wa -k cron

# Benutzeränderungen
-w /etc/passwd -p wa -k user_change
-w /etc/shadow -p wa -k user_change
-w /etc/group -p wa -k group_change
```

Verifikation:

```
# Regeln anzeigen
sudo auditctl -l

# Audits für einen Key suchen
sudo ausearch -k sshd_config --interpret

# Audit-Report erstellen
sudo aureport --auth
```

File-Integrity-Monitoring (AIDE)

AIDE (Advanced Intrusion Detection Environment) vergleicht Dateien mit einem bekannten Zustand und meldet Änderungen.

Installation:

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

Initial-Scan:

```
# Datenbank initialisieren
sudo aideinit
```

> ⚠ **Initial-Scan kann dauern.** Auf einem großen Server kann der erste Scan 10-30 Minuten dauern. Führe ihn nicht während Produktivbetrieb aus.

Datenbank aktivieren:

```
# Neue Datenbank verschieben
sudo cp /var/lib/aide/aide.db.new /var/lib/aide/aide.db
```

Cron-basierte Überprüfung:

```
sudo nano /etc/cron.d/aide-check
```

```
# Täglicher Check um 03:00
0 3 * * * root /usr/bin/aide --check | mail -s "AIDE Report" admin@example.com
```

Verifikation:

```
# Manuellen Check durchführen
sudo aide --check

# Erwartete Ausgabe bei sauberem System:
# AIDE found NO differences between database and filesystem.
```

Automatische Sicherheits-Patches (Unattended-Upgrades)

Eine rein manuelle Patch-Routine lässt Zero-Day-Lücken und bekannte CVEs tagelang ungeschützt. Automatische Sicherheits-Patches schließen kritische Schwachstellen zeitnah, während `needrestart` veraltete Prozesse nach Bibliotheks-Updates neu startet.

Installation & Aktivierung:

```
# Automatische Updates und Neustart-Prüfer installieren
sudo apt install unattended-upgrades needrestart -y

# Konfiguration aktivieren
sudo dpkg-reconfigure -plow unattended-upgrades
```

Konfiguration anpassen (`/etc/apt/apt.conf.d/50unattended-upgrades`):

```
// Nur Sicherheitsupdates automatisch installieren
Unattended-Upgrade::Allowed-Origins {
    "${distro_id}:${distro_codename}-security";
};

// Automatisch verwaiste Abhängigkeiten entfernen
Unattended-Upgrade::Remove-Unused-Dependencies "true";

// Automatischer Neustart bei Kernel-Updates um 04:00 Uhr
Unattended-Upgrade::Automatic-Reboot "true";
Unattended-Upgrade::Automatic-Reboot-Time "04:00";
```

Verifikation:

```
# Testlauf ohne Änderungen durchführen (Dry-Run)
sudo unattended-upgrade --dry-run --debug
```

Mandatory Access Control (AppArmor)

AppArmor schränkt Anwendungen (wie Nginx, Unbound oder PHP-FPM) auf die absolut notwendigen Dateisystem- und Netzwerk-Zugriffe ein. Selbst wenn ein Dienst eine Remote-Code-Execution-Schwachstelle hat, verhindert das AppArmor-Profil das Ausbrechen ins System.

Status und aktive Profile prüfen:

```
sudo aa-status
```

Erwartete Ausgabe:

```
apparmor module is loaded.
XX profiles are in enforce mode.
0 processes are unconfined but have a profile defined.
```

Profile in den Enforce-Modus schalten:

```
# Profile aktivieren und strikt erzwingen
sudo apt install apparmor-utils -y
sudo aa-enforce /etc/apparmor.d/*
```

> **Tipp:** Neue oder eigene Profile kannst du mit `aa-genprof <programm>` im interaktiven Lernmodus aufzeichnen und anschließend per `aa-enforce` scharf schalten.

Netzwerk-Härtung

nftables: Die moderne Firewall

`nftables` hat `iptables` als Standard-Firewall-Manager abgelöst. Unter Ubuntu 24.04 und 26.04 ist `nftables` der Kernel-Standard. Wer noch `iptables` nutzt, arbeitet mit einer Kompatibilitätsschicht – das funktioniert, aber `nftables` ist effizienter und bietet klarere Syntax.

Grundstruktur:

```
# Aktuelle Tabellen anzeigen
sudo nft list ruleset
```

```
table inet filter {
    chain input {
        type filter hook input priority filter; policy drop;
        # Regeln hier
    }
    chain forward {
        type filter hook forward priority filter; policy drop;
    }
    chain output {
        type filter hook output priority filter; policy accept;
    }
}
```

Firewall-Grundregeln erstellen:

```
sudo nano /etc/nftables.conf
```

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

flush ruleset

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

        # Erlaube etablierte Verbindungen
        ct state established,related accept

        # Erlaube Loopback
        iif lo accept

        # Erlaube ICMP (Ping)
        ip protocol icmp accept
        ip6 nexthdr icmpv6 accept

        # Erlaube SSH (Port 22)
        tcp dport 22 accept

        # Erlaube HTTP/HTTPS (falls Webserver)
        tcp dport { 80, 443 } accept

        # Logging für abgelehnten Traffic
        log prefix "nft-drop: " counter drop
    }

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

```
    }

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

Service starten:

```
sudo systemctl enable nftables
sudo systemctl start nftables
```

> ⚠ **Reihenfolge beachten:** Bei Remote-Servern immer zuerst SSH (Port 22) erlauben, bevor du die Standard-Policy auf `drop` setzt. Sonst sperrst du dich selbst aus.

Rate-Limiting für SSH:

```
# SSH-Rate-Limit: Maximal 5 Verbindungen pro Minute
sudo nft add rule inet filter input tcp dport 22 meter ssh-rate { limit rate 5/minute
burst 5 packets } accept
```

Verifikation:

```
# Aktive Regeln prüfen
sudo nft list ruleset

# Verbindungen prüfen
sudo ss -tulnp | grep -E "22|80|443"

# Drops in Logs prüfen
sudo journalctl -k | grep "nft-drop"
```

CrowdSec + nftables Integration

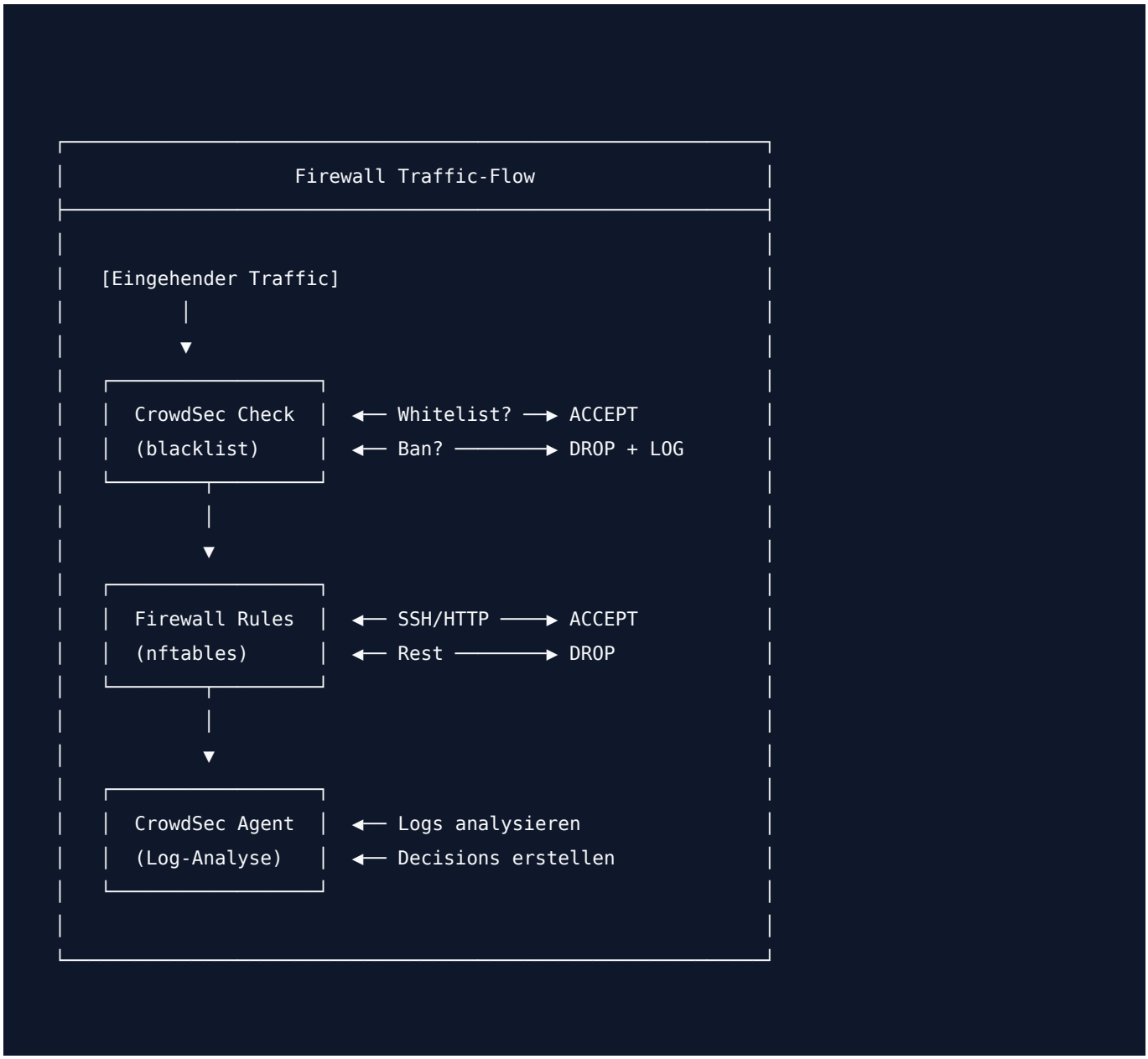
CrowdSec arbeitet mit `nftables` über den Firewall-Bouncer zusammen. Der Agent analysiert Logs und der Bouncer setzt Banns über `nftables`-Regeln durch.

Bouncer-Regeln prüfen:

```
# CrowdSec-spezifische Chains anzeigen
sudo nft list chain inet crowdsec-crowdsec-block
```

```
table inet crowdsec-crowdsec {
    chain crowdsec-block {
        type filter hook input priority filter; policy accept;
        ip saddr @crowdsec-blacklist-jump drop
        ip6 saddr @crowdsec-blacklist6-jump drop
    }
}
```

Traffic-Flow:



TLS-Konfiguration

Für Webserver ist TLS-1.3 der aktuelle Standard. TLS 1.2 ist noch akzeptabel, aber ältere Versionen (1.0, 1.1) sollten deaktiviert werden.

Let's Encrypt mit Certbot:

```
# Certbot installieren
sudo apt install certbot -y

# Zertifikat anfordern (Nginx)
sudo certbot --nginx -d example.com

# Zertifikat anfordern (Apache)
sudo certbot --apache -d example.com
```

TLS-Konfiguration für Nginx:

```
sudo nano /etc/nginx/snippets/ssl-params.conf
```

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers on;
ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384;
ssl_session_timeout 1d;
ssl_session_cache shared:SSL:10m;
ssl_session_tickets off;

# OCSP Stapling
ssl_stapling on;
ssl_stapling_verify on;

# Security Headers
add_header Strict-Transport-Security "max-age=63072000; includeSubDomains" always;
add_header X-Content-Type-Options nosniff always;
```

```
    add_header X-Frame-Options DENY always;
```

Zertifikat-Auto-Renewal:

```
# Cron-Job für automatische Erneuerung
sudo crontab -e
```

```
# Täglich um 02:30, Zertifikat bei Bedarf erneuern
30 2 * * * certbot renew --quiet --post-hook "systemctl reload nginx"
```

Verifikation:

```
# TLS-Konfiguration testen
openssl s_client -connect example.com:443 -tls1_3

# Zertifikats-Details prüfen
echo | openssl s_client -connect example.com:443 2>/dev/null | openssl x509 -noout -dates
```

DNS-over-HTTPS

Standardmäßig sendet dein Server DNS-Anfragen unverschlüsselt über UDP Port 53. Jeder auf dem Weg kann sehen, welche Domains du auflöst. DNS-over-HTTPS (DoH) verschlüsselt diese Anfragen.

Unbound als lokaler Resolver:

```
sudo apt install unbound -y
```

Konfiguration:

```
sudo nano /etc/unbound/unbound.conf.d/hardening.conf
```

```
server:
    interface: 127.0.0.1
    access-control: 127.0.0.0/8 allow

    # DNSSEC aktivieren
    auto-trust-anchor-file: /var/lib/unbound/root.key

    # Datenschutz
    hide-identity: yes
    hide-version: yes

    # Performance
    msg-cache-slabs: 8
    rrset-cache-slabs: 8
    infra-cache-slabs: 8
    key-cache-slabs: 8
```

Mit Cloudflare DoH aufwärtsauflösen:

```
forward-zone:
    name: "."
    forward-addr: 1.1.1.1@853#cloudflare-dns.com
    forward-addr: 9.9.9.9@853#dns.quad9.net
    forward-tls-upstream: yes
```

Service starten:

```
sudo systemctl enable unbound
sudo systemctl start unbound
```

Verifikation:

```
# DNS-Auflösung testen
dig example.com @127.0.0.1

# DNSSEC-Test
dig dnssec-failed.org @127.0.0.1
# Erwartung: SERVFAIL (DNSSEC-Breaker korrekt erkannt)

# DNS-over-HTTPS testen
curl -v https://cloudflare-dns.com/dns-query?name=example.com 2>&1 | grep "HTTP/2"
```

> **Empfehlung:** Für Server mit vielen Clients (z. B. Homelab) ist Unbound als lokaler Resolver die beste Wahl. Clients konfigurierst du mit `127.0.0.1` als DNS-Server, Unbound resolved aufwärts über DoH/DoT.

Automatisierung

Ansible-Playbook für Server-Härtung

Die bisherigen Abschnitte haben gezeigt, welche Konfigurationen nötig sind. Einzelne Befehle auf jedem Server auszuführen ist arbeitsintensiv und fehleranfällig. Ansible automatisiert das – du beschreibst den gewünschten Zustand, und Ansible setzt ihn auf allen Servern um.

Projektstruktur:

```
mkdir -p ~/ansible-hardening/{roles,inventory,group_vars}
```

Inventory:

```
nano ~/ansible-hardening/inventory/hosts.ini
```

```
[production]
```

```
server1 ansible_host=192.168.1.100
server2 ansible_host=192.168.1.101

[homelab]
server3 ansible_host=192.168.10.50

[all:vars]
ansible_user=admin
ansible_python_interpreter=/usr/bin/python3
```

Playbook:

```
nano ~/ansible-hardening/hardening.yml
```

```
---
- name: Linux Server Härtung
  hosts: all
  become: yes
  vars:
    ssh_port: 22
    allowed_users:
      - admin
    enable_crowdsec: true
    enable_aide: true

  tasks:
    # System-Updates
    - name: System aktualisieren
      apt:
        update_cache: yes
        upgrade: dist
```

```
        # SSH-Härtung
        - name: SSH-Konfiguration härten
          lineinfile:
            path: /etc/ssh/sshd_config
            regexp: "{{ item.regexp }}"
            line: "{{ item.line }}"
          loop:
            - { regexp: '^#?PermitRootLogin', line: 'PermitRootLogin no' }
            - { regexp: '^#?PasswordAuthentication', line: 'PasswordAuthentication no' }
            - { regexp: '^#?MaxAuthTries', line: 'MaxAuthTries 3' }
            - { regexp: '^#?LoginGraceTime', line: 'LoginGraceTime 30' }
          notify: sshd neu starten

        # Sysctl-Härtung
        - name: Sysctl-Parameter setzen
          sysctl:
            name: "{{ item.name }}"
            value: "{{ item.value }}"
            state: present
            reload: yes
          loop:
            - { name: 'net.ipv4.conf.all.rp_filter', value: '1' }
            - { name: 'net.ipv4.conf.all.accept_redirects', value: '0' }
            - { name: 'kernel.randomize_va_space', value: '2' }
            - { name: 'kernel.dmesg_restrict', value: '1' }

        # Unnötige Services
        - name: Unnötige Services deaktivieren
          systemd:
            name: "{{ item }}"
            state: stopped
            enabled: no
            masked: yes
          loop:
            - cups
            - bluetooth
            - avahi-daemon

        # Audit-Logging
        - name: Auditd installieren
          apt:
            name: auditd
            state: present
```

```
    - name: Audit-Regeln deployen
      copy:
        src: files/audit-rules.rules
        dest: /etc/audit/rules.d/hardening.rules
      notify: auditd neu starten

  handlers:
    - name: sshd neu starten
      systemd:
        name: sshd
        state: restarted

    - name: auditd neu starten
      systemd:
        name: auditd
        state: restarted
```

Playbook ausführen:

```
cd ~/ansible-hardening
ansible-playbook -i inventory/hosts.ini hardening.yml
```

> □□ **Idempotenz:** Ansible ist idempotent – du kannst das Playbook mehrmals ausführen, ohne dass sich am System unnötig etwas ändert. Nur abweichende Zustände werden korrigiert.

Verifikation:

```
# Prüfe ob alle Server erreichbar sind
ansible all -i inventory/hosts.ini -m ping

# SSH-Konfiguration prüfen
ansible all -i inventory/hosts.ini -m shell -a "sshd -T | grep permitrootlogin"
```

Erwartete Ausgabe:

```
server1 | CHANGED | rc=0 >>
permitrootlogin no

server2 | CHANGED | rc=0 >>
permitrootlogin no
```

CIS-Scoring mit OpenSCAP

CIS (Center for Internet Security) Benchmarks definieren Sicherheitsstandards für verschiedene Systeme. OpenSCAP prüft dein System gegen diese Benchmarks und gibt einen Score aus.

Installation:

```
sudo apt install libopenscap8 scap-security-guide -y
```

Ubuntu 24.04/26.04 Benchmark ausführen:

```
# Verfügbare Profiles anzeigen
oscap info /usr/share/xml/scap/ssg/content/ssg-ubuntu2404-ds.xml | grep "Profile"

# CIS-Level-1-Scan
sudo oscap xccdf eval \
    --profile cis_level1_server \
    --results cis-results.xml \
    --report cis-report.html \
    /usr/share/xml/scap/ssg/content/ssg-ubuntu2404-ds.xml
```

Report im Browser öffnen:

```
# Report auf den lokalen Rechner kopieren und öffnen
scp admin@server:/tmp/cis-report.html ~/Desktop/
```

Score interpretieren:

Score	Bedeutung
100%	Alle CIS-Empfehlungen erfüllt
80-99%	Guter Stand, kleinere Anpassungen nötig
60-79%	Grundkonfiguration, einige Härtungen fehlen
< 60%	Kritische Lücken, dringender Handlungsbedarf

Automatisierte Remediation (Vorsicht):

```
# Nur wenn du die Auswirkungen verstehst
sudo oscap xccdf generate fix \
    --profile cis_level1_server \
    --fix-type ansible \
    --output cis-remediation.yml \
    /usr/share/xml/scap/ssg/content/ssg-ubuntu2404-ds.xml
```

> ⚠ **Remediation nur gezielt:** Automatische Remediation kann funktionierende Konfigurationen überschreiben. Prüfe jede Änderung vor dem Ausführen. Im Zweifel: Manuell und mit Verstand.

Verifikation:

```
# Score prüfen
oscap xccdf generate report cis-results.xml > /dev/null
grep -E "score|pass|fail" cis-results.xml | tail -10
```

Härtungs-Checkliste

Schritt	Maßnahme	Wirkung
1	SSH: `PasswordAuthentication no`, `PermitRootLogin no`	Brute-Force blockiert
2	FIDO2/YubiKey einrichten	Passwortlose, hardwarebasierte Auth
3	CrowdSec Agent + Bouncer installieren	Kollaborativer Intrusion-Schutz
4	Sysctl-Härtung anwenden	Kernel-Parameter absichern

Schritt	Maßnahme	Wirkung
5	Unnötige Services maskieren	Angriffsfläche verkleinern
6	auditd aktivieren	Sicherheitsereignisse protokollieren
7	AIDE initialisieren	File-Integrity-Überwachung
8	nftables Firewall aufsetzen	Netzwerk-Traffic filtern
9	TLS 1.3 + Security Headers	Webserver absichern
10	Unbound + DoH konfigurieren	DNS-Anfragen verschlüsseln

Jeder Schritt baut auf dem vorherigen auf. Die Reihenfolge ist bewusst so gewählt: Zuerst die kritischsten Schwachstellen (SSH, CrowdSec), dann System-, dann Netzwerk-Schicht.

Weiterführende Ressourcen

Offizielle Dokumentationen

OpenSSH FIDO2 CrowdSec Dokumentation nftables Wiki CIS Benchmarks OpenSCAP Dokumentation

Tools & Pakete

YubiKey Manager CrowdSec Hub Certbot Unbound

Verwandte Artikel auf admindocs.de

Thema	Artikel
VPN-Grundlagen	WireGuard VPN-Server einrichten
Container	Ubuntu 26.04 Docker
Automatisierung	Ansible Grundlagen

Thema	Artikel
Netzwerk	Netzwerkprobleme analysieren
Grundlagen	LPIC-1 Serie
Server	Ubuntu-Upgrade: 22.04 auf 24.04 LTS

Fazit

Serverhärtung ist kein einmaliges Projekt, sondern ein fortlaufender Prozess. Die in diesem Artikel beschriebenen Maßnahmen – SSH mit FIDO2, CrowdSec, Kernel-Härtung, nftables und TLS – bilden das Fundament, auf dem jedes produktive Linux-System aufbauen sollte.

Wichtig ist der Startpunkt: Nicht alles auf einmal, sondern schrittweise. Die Härtungs-Checkliste oben gibt die Reihenfolge vor. Mit jedem Schritt wird die Angriffsfläche kleiner, und die Kombination aus mehreren Maßnahmen schafft ein robustes Sicherheitsprofil, das einzelnen Schwachstellen nicht das gleiche Schadenspotenzial lässt.

Für Teams lohnt sich die Automatisierung mit Ansible – was einmal konfiguriert ist, kann auf beliebig viele Server übertragen werden. CIS-Benchmarks mit OpenSCAP geben den Score vor und zeigen, wo noch Luft nach oben ist.

Ein gehärteter Server ist kein Server, der gegen alles abgesichert ist. Es ist ein Server, bei dem du weißt, was abgesichert ist und was nicht – und warum.