LINUX VPN

IPSEC, OPENVPN UND WIREGUARD

CARSTEN STROTMANN

Version 1.0, 12.09.2025

INHALTSVERZEICHNIS

CHAPTER 1. TAG 1

1.1. EINFÜHRUNG

- TCP/IP–Sicherheit [./tcpip–security.html]

1.2. IKE AND IPSEC

1.2.1. IPSec & IKE Begriffe

- ISAKMP = Internet Security Association and Key Management Protocol
- IKE oder IKEv1 = Internet Key Exchange Version 1 (eine Implementation von ISAKMP)
- IKEv2 = Internet Key Exchange Version 2 (eine moderne Implementation von ISAKMP)
- AH = Authentication Header = ein Modus von IPsec, sichert Integrität und Authentisierung der Daten
- ESP = Encapsulating Security Payload = ein Modus von IPsec, sichert Integrität, Authentisierung und Vertraulichkeit der Daten
- Transport Mode = IPSec Modus, bei welchem der IP–Header gesichert, aber nicht geändert wird. Funktioniert nicht mit NAT
- Tunnel Mode = IPsec Modus, bei welchem das IP–Paket in ein neues IP–Paket eingepackt wird. Funktioniert mit NAT, benötigt mehr Bandbreite und kann Problemen mit MTU und Fragmentierung führen
- Security Association (SA) = Paket aus Verschlüsselungs–Algorithmen und –Parametern für eine Richtung einer IPsec Verbindung
- Security Policy Database (SPD) = Datenbank innerhalb der IPsec Implementierung, welche die Regeln für die Anwendung von IPsec auf IP–Pakete definiert (welche Verbindungen sollen durch IPsec abgesichert werden)
- Securty Association Database (SADB oder SAD) = Datenbank innerhalb der IPsec Implementierung mit Informationen zu den Verschlüsselungs–Algorithmen und –Parametern für jede Verbindung)

1.2.2. IPsec & IKE Geschichte

- Arbeit an IPsec startete in 1992, zusammen mit der Arbeit an IPv6
  - IPv6–Implementierungen sollen auch eine IPsec Implementierung beinhalten
  - IPsec für IPv4 wurde nachträglich entwickelt, als klar wurde das die IPv6 Umstellung längere Zeit benötigen wird
- Erste Implementationen für BSD–Unix wurden vom "Naval Research Labatory" (NRL) erstellt
- Dies war vor NAT (1994) und "privaten" IP–Adressen (RFC 1918, 1996)!
  - Dies war einer der Gründe warum die Einrichtung der frühen Versionen von IPsec unter IPv4 recht komplex und schwierig waren
  - Probleme mit IPsec und NAT hat zu einem schlechten Ruf von IPsec geführt

- IPsec mit IKEv1 wurde 1998 zum Internet Standard (RFC)
- Das Problem bei IPsec war der Schlüssenaustausch (IKEv1)
  - IKEv1 war nicht mit Beachtung von NAT in Netzwerken entwickelt worden und kommunizierte IP–Adressen
    - Diese IP–Adressen wurden auf dem Weg durch NAT geändert und passten nicht mehr zu den IKE–Daten = Pakete wurden verworfen
- Nach der Veröffentlichung von IPsec mit IKEv1 wurde in der IETF an einer Verbesserung von IPsec gearbeitet
  - Einfache Konfiguration
  - Bessere Fehlersuche
  - Kompatibel mit IPv4 NAT–Netzwerken
- IKEv2 für IPsec wurde 2005 als Internet–Standard veröffentlicht, IKEv1 wurde als *veraltet* markiert
- Mit RFC9395 (April 2023) wurde IKEv1 als *deprecated* markiert, soll nicht mehr implementiert und benutzt werden
  - In LibreSwan wurde IKEv1 in den neueren Versionen entfernt

1.2.3. IPSec & IKE Verbindungsaufbau

- Ein IPsec Verbindungsaufbau geschied in Schritten. In IKEv1 wurden diese Schritte Phase 1 (IKEv2 = Parent SA) und Phase 2 (IKEv2 = Child SA) genannt. Unter IKEv2 gibt es diese Phasen so nicht mehr (in den Standard–Dokumenten, RFC), die Begriffe werden jedoch weiterhin von Netzwerkadministratoren auch für IKEv2 IPsec verwendet (auch wenn das nicht mehr korrekt ist).
  - Aushandeln der Kontrollverbindung (Phase 1 bei IKEv1): die IPsec Kommunikationspartner verständigen sich auf einen verschlüsselten Kanal (Algorithmen und Parameter). Ergebnisse ist die SA (Security Association) oder auch Parent–SA.
    - Über diese (verschlüsselte und authentisierte) Kontrollverbindung werden nun die Algorithmen und Parameter für die Datenverbindung ausgehandelt (Phase 2 bei IKEv1). Dies ist die Child–SA.
    - Die Daten werden der Verbindung werden mittels der Konfiguration der Child–SA verschlüsselt und abgesichert
    - Die Parameter der Child–SA *können* sich von den Parametern der Parent–SA unterscheiden(!)
    - Die Parameter von Parent–SA und Child–SA *können* sich für jede Kommunikationsrichtung unterscheiden(!)
- Die IKE–Kommunikation benutzt die Ports 500 (default, ohne NAT) und 4500 (wenn NAT zwischen den IPsec–Kommunikationspartnern entdeckt wurde)
  - Die IKE–Kommunikation benutzt UDP als Transport–Protokoll
  - RFC 9329 (TCP Encapsulation of Internet Key Exchange and IPsec Pakets) erlaubt das "tunneln" von IKE und IPsec–Paketen über TCP. Dies ist notwendig bei Firewall–Middelboxen welche jegliche UDP–Kommunikation bklockieren (z.B. in Hotels oder öffentlichen WLAN–Netzwerken)

1.2.4. Authentication Header (AH)

- IPsec kennt zwei Arten der Absicherung von IP–Paketen: Authentication Header (AH) und Encapsulating Security Payload (ESP)
- Bei AH wird ein weiterer Header (IP–Protokoll 51) in das IP–Paket zwischen dem IP–Header und den Daten (Payload) bzw. dem Header des Transport–Protokols (UDP oder TCP) des IP–Paketes eingefügt
- Dieser Header enthält kryptografische Prüfsummen (Hashes) über Teile des IP–Headers und den Daten
- AH bietet Integrität und Autentifizierung, jedoch keine Vertraulichkeit (Verschlüsselung)
- AH kann sowohl im Transport– als auch im Tunnel–Modus von IPsec eingesetzt werden
- AH wird für Punkt–zu–Punkt Verbindungen zwischen zwei Rechnern mit öffentlichen IP–Adressen (meist IPv6) benutzt

Vorteile des AH

- AH erzeugt weniger Overhead verglichen mit ESP (weniger CPU–Last, kleinere Pakete)
- Einfachere Fehlersuche bei Netzwerkfehlern, wenn die Daten nicht vertraulich sein müssen, da die Daten im Klartext sichtbar sind

Nachteile des AH

- AH funktioniert nicht mit NAT, da die Quell– und Zieladresse des Pakets durch den AH geschützt sind
- Keine Vertraulichkeit möglich

Alternativen zu AH

- ESP kann mit einem "null" Verschlüsselungsalgorithmus benutzt werden, damit ist ESP funktionell vergleichbar mit AH (Schutz der Integrität und Authentisierung der Daten)

1.2.5. Encapsulating Security Payload (ESP)

- ESP ist eine Alternative zu AH und ist der heute meist genutzte Modus von IPsec
- Bei ESP werden die zu sichernde Daten (Payload) vollständig und unverändert in ein neues ESP–Transport–Protokoll mit einem ESP–Header eingeschlossen (IP–Protokoll 50 = ESP)
- ESP kann sowohl im Transport–Mode als auch im Tunnel–Mode eingesetzt werden
- ESP wir oft für VPNs zwischen zwei Netzwerken benutzt, bei denen jeglicher Verkehr zwischen den Netzwerken durch IPsec abgesichert sein soll
- In einigen Fällen kann ESP Vorteile beim Durchsatz von IP–Paketen bringen, wenn ESP durch Hardware–Funktionen unterstützt wird (IPsec offloading). Dies setzt Netzwerkhardware mit spezieller IPsec Offloading–Funktion vorraus

1.2.6. Transport Mode

- Sowohl AH als auch ESP können wahlweise im Transport–Modus als auch im Tunnel–Modus eingesetzt werden
- Beim Transport–Modus werden in das bestehende IP–Paket AH– oder ESP–Header eingefügt. Der IP–Header bleibt original bestehen, bei AH wird der IP–Header geschützt, bei ESP werden nur die Daten geschützt
    - Der Transport–Modus wird häufig benutzt um die Kommunikation zwischen zwei einzelne Rechner abzusichern. Bauen diese zwei Rechner eine ungesicherte Tunnelverbindung zwischen zwei Netzwerken (z.B. GRE– oder IPIP–Tunnel), so kann diese bestehende Verbindung durch IPsec–Transport–Mode abgesichert werden

1.2.7. Tunnel Mode

- Beim Tunnel–Modus wird das original IP–Paket in ein neues (IPsec–gesichertes) IP–Paket eingebettet. Die IPsec Sicherheit (AH oder ESP) sichert dabei das gesamte original IP–Paket.
    - Auf der Empfängerseite wird das Paket wieder ausgepackt und unverändert an den Netzwerk–Stack des Empfängers übergeben
    - Tunnel–Mode eignet sich gut um mehrere Netzwerke mittels IPsec–Gateway–Router miteinerander zu verbinden

1.2.8. Opportunistische Verschlüsselung mittels IPsec

- Einer der Designziele von IPsec war die vollständige Verschlüsselung von IP–Netzwerken im Internet
- Traditionelle IPsec Verbindungen werden manuell erstellt und konfiguriert
- Bei *opportunistic encryption* (OE–IPsec) wird die IPsec–Verbindung zwischen zwei Kommunikationspartnern automatisch ausgehandelt
- Beim ersten IP–Paket wird getestet, ob die Gegenstelle IPsec unterstützt
- Authentisierung geschied mittels (DNSSEC gesichertem) DNS (Prüfung von vorwärts□—□A– und AAAA–Records□—□und rückwärts (PTR–Records) DNS–Auflösung
- Im IPSECKEY–Record können pro Host IPsec–Parameter im DNS abgelegt werden (RFC 4025)
    - Der IPSECKEY–Record beinhaltet den Gateway–Typ (IPv4 oder IPv6), die Gateway–Adresse, den Schlüsselalgorithmus des öffentlichen Schlüssel des Hosts und den öffentlichen Schlüssel
    - IPSECKEY–Record sollten per DNSSEC abgesichert sein, Empfänger von IPSECKEY–Records sollten diese per DNSSEC prüfen

1.2.9. IPsec unter Linux

Linux IPSec Implementierungen

- FreeS/WAN (1996 – 2003): originale IPsec Implementation, IKEv1, IPsec als Patch fur den Linux–Kernel

- OpenSwan (2003 – 2011): Freundlicher Fork von FreeS/WAN, IKEv2, IPsec im Linux–Kernel, Beendet wegen Namensdisput
- StrongSwan (2003 – jetzt): Fork von FreeS/WAN, IKEv1 und IKEv2, Unterstützt von SecuNet
- LibreSWAN (2011 – jetzt): Fork von OpenSWAN, gleiches Entwickler–Team, neuer Name wegen Namensdisput, unterstützt von Red Hat
- DPDK – Network–Stack in Userspace: Unterstützt von Intel und der Linux Foundation
- LibreSwan und StrongSwan sind Teil von Red Hat EL (Alma Linux, Rocky Linux etc)

IKE–Daemon und Linux–Kernel

- IKEv2 wird unter Linux als User–Space–Prozess implementiert. Dies ist der `pluto` Dienst (LibreSwan) oder `pluto` und `charon` (StrongSwan)
  - Dies ist die IPsec Kontrollverbindung (control channel, control plane)
  - Die Parameter der ausgehandelten SA werden von diesen Diensten in die `SAD` und `SPD` des Linux–Kernel geschrieben
- Die Verschlüsselung und Authentisierung der IP–Pakete der Netzerkverbindung mittels IPsec geschied ausschliesslich im Linux–Kernel ohne Interaktion mit den User–Space–Programmen

1.2.10. LibreSwan installieren

```
dnf install libreswan
```

1.2.11. Hands–on: IPSec mit PSK

- Datei `/etc/ipsec.d/psk-tunnel.conf` erstellen (emacs, vim, nano)

```
conn host-host
     keyexchange=ikev2
     authby=secret
     left=<ip-vm-1>
     right=<ip-vm-2>
     auto=add
```

- IPSec Konfiguration neu laden

```
ipsec restart
```

- Status des Dienstes prüfen

```
systemctl status ipsec
```

- Logdateien des IPsec Dienstes (LibreSwan) anschauen

```
journalctl -u ipsec
```

- Logdateien im "follow"–Modus beobachten

```
journalctl -fu ipsec
```

- Benannte IPsec Verbindung starten

```
ipsec up host-host
```

- IPsec Status anzeigen

```
ipsec status
```

- Linux-Kernel IPsec Policy Datenbank anzeigen

```
ip xfrm pol
```

- Linux-Kernel IPsec Verbindungen anzeigen /ACHTUNG: dieser Befehl zeigt die IPsec Schlüssel im Klartext. Mittels der Daten dieses Befehls kann eine IPsec Kommunikation entschlüsselt werden! (mehr dazu später)

```
ip xfrm state
```

Wird die Verbindung erfolgreich aufgebaut?

1.2.12. Probleme lösen

- Firewall? Finde auf der Webseite https://libreswan.org [https://libreswan.org] die Information, wie bei Red Hat Systemen IPsec in der Firewall erlaubt wird

1.2.13. Firewall für IPsec öffnen

```
   firewall-cmd --add-service="ipsec"
   firewall-cmd --runtime-to-permanent
```

- Authentisierung? Lese die "man"-Page zu `ipsec.secrets` und erstelle einen Eintrag für die Benutzung eines Pre-Shared-Keys (PSK) in der Datei `/etc/ipsec.d/psk-tunnel.secrets`.

1.2.14. Lösung:

Datei `/etc/ipsec.d/psk-tunnel.secrets`:

```
   <ip-vm1> <ip-vm2> : PSK "mein-geheimer-schluessel"
```

1.2.15. IPsec mittels tcpdump anschauen

- IPsec hat (in der Standard-Konfiguration) keine extra Netzwerkschnittstellen, um verschlüsselten von unverschlüsselten Paketen getrennt zu betrachten.

```
# tcpdump -v -i eth0 host vpnNNN.dane.onl
```

- Die Ausgabe von `tcpdump` zeigt eingehende Pakete aus dem IPsec Tunnel unverschlüsselt an, ausgehende Pakete werden verschlüsselt gesehen (hier: ESP Pakete Protokol 50)

```
10:18:42.953708 IP (tos 0x0, ttl 64, id 38962, offset 0, flags [DF], proto ICMP (1),
length 84)
    206.189.7.68 > vpn012: ICMP echo request, id 1, seq 2090, length 64
```

```
10:18:42.953760 IP (tos 0x0, ttl 64, id 2586, offset 0, flags [none], proto ESP (50),
length 140)
    vpn012 > 206.189.7.68: ESP(spi=0xa09b1511,seq=0x82a), length 120
```

1.2.16. Hands-on: IPSec mit RSA-Schlüssel

- RSA-Schlüssel erstellen, CKAID notieren

```
# ipsec newhostkey --nssdir /var/lib/ipsec/nss
Generated RSA key pair with CKAID 3bdd3ee727c90bac3218dcaf0cdf680a0a405664 was stored
in the NSS database
The public key can be displayed using: ipsec showhostkey --left --ckaid
3bdd3ee727c90bac3218dcaf0cdf680a0a405664
```

- NSS-Schlüssel anzeigen

```
ipsec showhostkey --list
```

- RSA-Schlüssel ausgeben

```
# ipsec showhostkey --left --ckaid  3bdd3ee727c90bac3218dcaf0cdf680a0a405664
        # rsakey AwEAAc73d

leftrsasigkey=0sAwEAAc73dvkhQKUgEKG1W2bg8BrSgTXDWG47ST71j58NOs7s1gRBydtRLCrdEkO8GeTTjCS
Pmq6nW+CWCPP10zxDbsj/rHfYbuGwkBLdjEvseO96W2C0w2TKe0KhAuJCVrioGUGZ3Fdvib3gjrlryAFjSMipLV
5ykZDR9ZFi4EBMIH4N2bfuAIgnNe84iIfqFt5RnY486uTy78LU2qlSxBVHaMmlSRb50WJuCB39J7auKNdQepmmb
JZusmffYtsAoQul2ysF3sxW7nX9I3Gmmz5+PSrLefPvj9q0yv3n9DLCZPserQK+Fnxo/R7MbMYSVIOgAFyZYd/9
jCSy2+oSJkoseW5u+22KKnCYRkSbw5BD/WAqa4+KK3CNcfPSkfFAEtbD762fi+FW8BRMqHrhZln+gTIOr/IZYCO
iR2w+8Qk26ZhTJAn5nT8lTBsES9vUbvKqj5CUIOGj4emx2j6RAZAZm74zYfIcK0puweNt5wMhxGbez3yJbJRxoU
6nqko/aB5Cz5bnOfAE2pd9J5L5XhOH7PS/oNFFHILIt6FltxaAyHDFyEjlb63ZhxR7wgkFrDjfYCnMGss=
```

- RSA-Schlüssel Authentisierung (ipsec Konfiguration):

```
  [...]
  authby=rsasig
```

Aufgabe:

- Deaktiviere die PSK-Tunnel-Konfiguration (Datei von `psk-tunnel.conf` in `psk-tunnel.disabled` umbenennen)
- Erstelle eine IPsec Konfiguration mit RSA-Schlüssel-Authentisierung in der Datei `/etc/ipsec.d/rsakey-tunnel.conf`
- Erstelle neue Schlüssel auf beiden Seiten der Verbindung (VM1 und VM2)
- Trage die Schlüssel in die Konfigurationsdatei ein
- Teste die Konfiguration, überprüfe mit `tcpdump`, das die Kommunikation (z.B. ICMP-Echo aka "ping") verschlüsselt gesendet werden

Lösung:

- Datei `/etc/ipsec.d/rsakey-tunnel.conf`

```
conn rsa-rsa
     keyexchange=ikev2
     authby=rsasig
     left=<ip-von-vm1>
     right=<ip-von-vm2>
     auto=add
     # rsakey AwEAAc73d
     rightrsasigkey=0sAwEAAc7d[...]kAyHDFyEjlb63ZhxR7wgkFrDjfYCnMGss=
     # rsakey AwEAAexDT
     leftrsasigkey=0sAwEAAexDT[...]WJbPrPsquUyBOqdq9GEMGW8/cwQ95rbhc=
```

- Probleme bei der Authentisierung des Tunnels? Die "secrets" Datei des PSK-Tunnels entfernen oder deaktivieren

1.2.17. Hands-on: Transport Mode

- Teste die IPsec Konfiguration im Tunnel-Mode mit ICMP. Notiere die Grösse der ESP-Pakete
- Konfiguriere die aktuelle IPsec-Konfiguration für den Transport-Mode (Standard ist Tunnel-Mode)

```
    [...]
    type=transport
```

- Vergleiche die Grössen der Pakete bei einem ICMP-Echo Reqest/Reply ("ping") zwischen Tunnel- (Default) und Transport-Mode.

1.2.18. Tunnel automatisch starten

```
    auto=start      # Tunnel sofort mit dem IPsec IKE Daemon starten
    auto=ondemand   # Tunnel starten, sobald Pakete in den Tunnel geroutet werden
```

1.2.19. Hands-on: IPSec mit Zertifikat

- Root-CA erzeugen

```
certutil -S -k rsa -n "MyLocalCA" -s "CN=Lokale CA" -v 12 -t "CT,C,C" -x -d
sql:/var/lib/ipsec/nss
```

- Maschinen-Zertifikat erzeugen

```
certutil -S -k rsa -c "MyLocalCA" -n "vm1" -s "CN=vm1 Common Name" -v 12 -t "P,," -d
sql:/var/lib/ipsec/nss
```

- Zertifikate in der NSS-Datenbank anzeigen

```
certutil -L  -d sql:/var/lib/ipsec/nss
```

- Einzelnes Zertifikat anzeigen

```
certutil -L -n vm1  -d sql:/var/lib/ipsec/nss | less
```

- Bei Fehlern kann ein Zertifikat aus der Datenbank gelöscht werden

```
certutil -D -n <nickname>  -d sql:/var/lib/ipsec/nss
```

- Das Root-Zertifikat einer CA aus der Datenbank exportieren

```
certutil -L -n "MyLocalCA" -d sql:/var/lib/ipsec/nss -a > mylocal.crt
```

- Das Root-Zertifikat (öffentlicher Schlüsse) aus einer Datei in die NSS-Datenbank importieren

```
certutil -A -i mylocalca.crt -n "MyLocalCA" -t "CT,," -d sql:/var/lib/ipsec/nss
```

- Ein Benutzer/Maschinen-Zertifikat exportieren (nur öffentlicher Schlüssel)

```
certutil -L -n "vm1" -d sql:/var/lib/ipsec/nss -a > vm1.crt
```

- Ein Benutzer/Maschinen-Zertifikat importieren (nur öffentlicher Schlüssel)

```
certutil -A -i vm1.crt -n "vm1" -t "P,," -d sql:/var/lib/ipsec/nss
```

- Ein Benutzer/Maschinen-Zertifikat exportieren (öffentlicher UND *privater* Schlüssel)

```
pk12util -o vm2.p12 -n vm2 -d sql:/var/lib/ipsec/nss
```

- P12 (binär) in PEM (Text) umwandeln

```
openssl pkcs12 -in vm2.p12 -out vm2.pem -clcerts -nodes -legacy
```

- PEM (Text) in P12 (binär) umwandeln

```
openssl pkcs12 -export -in vm2.pem -out vm2.p12
```

- Ein Benutzer/Maschinen-Zertifikat importieren (öffentlicher und privater Schlüssel)

```
ipsec import vm2.p12
```

- Nickname eines Zertifikats in der NSS-Datenbank umbenennen

```
certutil  -n "vm2 Common Name" --rename --new-n "vm2" -d sql:/var/lib/ipsec/nss
```

- Die Vertrauens-Attribute eines Zertifikats anpassen

```
certutil -M -n vm2 -t "P,," -d sql:/var/lib/ipsec/nss
```

Aufgabe:

- Erstelle eine CA auf VM1
- Erstelle und signiere Zertifikate für "vm1" und "vm2" auf VM1
- Exportiere das Zertifikat (öffentlicher Schlüssel plus Signatur von CA) von "vm2" in eine PEM-Datei
- Exportiere das Root-CA-Zertifikat (öffentlicher Schlüssel) in eine PEM-Datei, übertrage dieses auf VM2 und importiere es dort

- Übertrage das Zertifikat (öffentlicher Schlüssel und Signatur der CA) und den privaten Schlüssel von "vm2" auf VM2 (die PEM-Dateien sind ASCII-Dateien und können über die Zwischenablage übertragen werden)
- Übertrage das Zertifikat "vm1" (öffentlicher Schlüssel plus Signatur der CA) auf VM2
- Benötigt werden:
  - VM1: Privater Schlüssel und Zertifikat von VM1, Zertifikat vom VM2, Zertifikat vom CA
  - VM2: Privater Schlüssel und Zertifikat von VM2, Zertifikat vom VM1, Zertifikat vom CA
- Beispiel-Konfiguration

```
conn cert-cert
     keyexchange=ikev2
     authby=rsa-sha2
     left=<ip-vm-1>
     leftid=%fromcert
     leftrsasigkey=%cert
     leftcert=vm1
     leftsendcert=always
     right=<ip-vm-2>
     rightid=%fromcert
     rightrsasigkey=%cert
     rightcert=vm2
     rightsendcert=always
     auto=add
```

- Ausgabe Zert-DB auf VM1:

```
certutil -L  -d sql:/var/lib/ipsec/nss

Certificate Nickname                                         Trust Attributes
                                                             SSL,S/MIME,JAR/XPI

MyLocalCA                                                    CTu,Cu,Cu
vm1                                                          Pu,u,u
vm2                                                          Pu,u,u
```

- Ausgabe Zert-DB auf VM2:

```
certutil -L  -d sql:/var/lib/ipsec/nss

Certificate Nickname                                         Trust Attributes
                                                             SSL,S/MIME,JAR/XPI

MyLocalCA                                                    CT,,
vm2                                                          Pu,u,u
vm1                                                          P,,
```

1.2.20. Hands-on: Route-based VPN using VTI

- LibreSwan unterstützt IPsec mit *Virtual Tunnel Interface* (VTI)
- VTI erzeugt eine virtuelle Netzwerkschnittstelle im Linux-System

    - Alle Netzwerkpakete, welche über diese Schnittstelle geroutet werden, sind per IPsec abgesichert
    - Die Sicherheit des VPN-Tunnel hängt am Routing und der Firewall-Konfiguration — Pakete welche nicht über das VTI geroutet werden, werden nicht IPsec gesichert
    - IPsec mit VTI vereinfacht die Konfiguration und Fehlersuche im IPsec
        - Am VTI-Interface sind die unverschlüsselten Daten zu sehen
        - Am physischen Netzwerkinterface sind die verschlüsselten IPsec-Pakete zu sehen
- Beispiel einer Konfiguration mit Zertifikaten und VTI-Schnittstelle

```
conn vti-vti
     keyexchange=ikev2
     authby=rsa-sha2
     type=tunnel
     left=vm1
     leftsubnet=10.0.0.0/8
     leftid=%fromcert
     leftrsasigkey=%cert
     leftcert=vm1
     leftsendcert=always
     right=vm2
     rightsubnet=10.0.0.0/8
     rightid=%fromcert
     rightrsasigkey=%cert
     rightcert=vm2
     rightsendcert=always
     mark=5
     vti-interface=vti01
     vti-routing=no
     auto=add
```

Aufgabe: IPsec mit VTI

- Erstelle eine neue IPsec Konfiguration mit VTI
- Starte den IPsec Tunnel
- Konfiguriere je eine IP-Adresse aus den privaten Adressräumen (RFC 1918) auf die VTI-Schnittstelle (z.B. `ip address add 10.0.0.1/8 dev vti01`)
- Teste die Verbindung über die privaten IP-Adressen mittels `ping`
- Lasse dir die Pakete auf den VTI-Schnittstellen und den physischen Schnittstellen mittels `tcpdump` anzeigen

1.2.21. Sicherheit von PSK bei IPsec mit IKEv2

- Siehe "Security Considerations" in RFC 7296 (IKEv2bis)
- TODO – Zitat
- RFC 6617 " Secure Pre-Shared Key (PSK) Authentication for the Internet Key Exchange Protocol

(IKE)"

1.2.22. Kryptographie für Linux Admins

- Kryptographie für Linux Admins [./crypto.html]

CHAPTER 2. TAG 2

2.1. DIE NEUE FIREWALL NFTABLES

- Einführung in nftables [./nftables.html]
- `nftables` Programme installieren

```
dnf install nftables
```

- Beispiel: Lesen der Firewall–Regeln aus einer Datei

```
# nft -f /etc/nftables.conf
```

2.1.1. Beispiel: Eine Regel (IPv4, Tabelle "fw", Kette "input") hinzufügen

```
# nft "add rule ip fw input drop"
```

2.1.2. Beispiel: eine interaktive *nft* Sitzung

```
        # nft -ia
        nft> list ruleset
        table inet workstation-fw {
            chain input {
                     type filter hook input priority 0;
            }
        }
```

2.1.3. Beispiel: eine interaktive *nft* Sitzung

```
        nft> add rule inet workstation-fw input tcp dport {ssh, http} accept
        nft> list ruleset
        table inet workstation-fw {
            chain input {
                     type filter hook input priority 0;
                     tcp dport { ssh, http} accept # handle 3
            }
        }
        nft> quit
```

2.1.4. Regelnsatz löschen

- Der Befehl `flush` löscht Objekte aus der *nftables* Kernel–Firewall.
- Ein `flush ruleset` löscht die gesamten Firewall–Regeln.
- Ein `flush ruleset` sollte am Anfang eines *nftables*–Regelsatzes stehen:

```
      # nft flush ruleset
```

- Über `flush chain` und `flush table` können Ketten und Tabellen gelöscht werden.

2.1.5. Regeln hinzufügen

- Der Befehl `add` fügt die Regel immer am Ende der Kette an
- mit `insert` kann eine Regel an einer beliebigen Stelle der Kette eingefügt werden.

2.1.6. Regeln löschen

- Der Befehl `delete` löscht eine Regel aus einer Kette. Zum Löschen einer Regel wird das *Handle* der Regel benötigt, das *Handle* wird über den Kommandozeilenparameter `-a` ausgegeben:

```
    # nft -a list ruleset
    ...
    ip6 saddr 2001:db8::/64 # handle 15
    ...
    # nft delete rule inet filter input handle 15
```

2.1.7. Monitor

Mittels `nft monitor` können Änderungen in der *nftables*–Firewall überwacht werden.

Der Befehl schreibt Modifikationen im Regelsatz der Firewall als `nft`–Befehlszeilen (Alternativ als XML oder JSON) auf die Konsole:

```
      # nft monitor
      add table inet filter
      add chain inet filter input { type filter hook input priority 0; }
      add rule inet filter input iif lo accept
      add rule inet filter input ct state established,related accept
      add rule inet filter input ip6 daddr ff02::1010 udp dport ntp accept
      add rule inet filter input ip6 daddr ff02::fb udp dport mdns accept
      add rule inet filter input ip daddr 224.0.0.251 udp dport mdns accept
      add rule inet filter input ip6 saddr & :: == :: udp dport dhcpv6-client accept
      add rule inet filter input udp dport ipp accept
      add rule inet filter input ip6 saddr & :: == :: icmpv6
            type { nd-neighbor-solicit, nd-router-advert, nd-neighbor-advert} accept
      add rule inet filter input ip6 saddr & :: == :: icmpv6
            type { echo-reply, echo-request, packet-too-big, destination-unreachable,
                  time-exceeded, param-problem} accept
      add rule inet filter input counter packets 0 bytes 0 log prefix "nftables drop: "
drop
```

- die aus *iptables* bekannten Konzepte von Tabellen (Tables) und Ketten (Chains) finden sich auch in *nftables* wieder
- vordefinierten Tabellen (in *iptables* 'filter', 'nat' und 'mangle') gibt es im *nftables* nicht
- Auch die in *iptables* vorhandenen Ketten ('input', 'output', 'forward') sucht man vergebens
- *nftables* kommt ohne vordefinierte Tabellen und Ketten.
- Die vordefinierten Ketten in *iptables* sind ein Performance–Problem bei grossen Datenmengen, da Netzwerkpakete auch durch unbenutze, leere Ketten geschleusst werden.
    - Unter *nftables* sind nur Tabellen und Ketten aktiv, welche der Administrator definiert hat.

- In *nftables* sind Tabellen einfache Container für beliebige Ketten.
- Eine Tabelle muss einem Protokoll zugewiesen werden (*ip* = IPv4, *arp*,/ip6/ = IPv6, *bridge*).
- Die Tabellen können beliebige Namen haben, auch wenn in vielen Beispielen im Internet die aus *iptables* bekannten Namen verwendet werden.
    - Bei den Ketten (Chains) unterscheidet man "base-chains" und "auxiliary chains".
    - Bei den "base-chains" handelt es sich um Ketten, welche in den Paketfluss des Netfilter-Framework eingeklinkt werden.
    - Diese Ketten erhalten Netzwerkpakete aus den Zugriffspunkten (Hooks) des Linux-Kernels.
    - Diese Zugriffspunkte für eine Kette vom Typ "filter" heissen "input", "output" und "forward".
    - Die Konzepte sind von *iptables* übernommen worden, der Firewall-Administrator hat nun aber mehr Freiraum, die Ketten zu organisieren.

2.1.8. Beispiel-Tabelle

- Beispiel einer Tabelle mit zwei (leeren) "filter" Ketten, für "input" und "output":

```
table inet filter01 {
        chain input01 {
                type filter hook input priority 0;
        }
        chain output01 {
                type filter hook output priority 0;
        }
}
```

2.1.9. Base- und Auxiliary-Ketten

- Mit dem Schlüsselwort *hook* wird eine Kette mit den Linux-Kernel-Internen Schnittstellen des Netfilter-Frameworks verbunden.
- Eine Kette mit Verbindung zum den Kernel-Schnittstellen ist eine "base-chain".
- Zusätzlich zu den "base-chains" kann der Administrator noch beliebig weitere "auxiliary chains" erzeugen.
- Diese Chains erhalten nur Netzwerkpakete, wenn diese aus einer "base-chain" per `goto` oder `jump` Direktive an die "auxiliary chain" verteilt werden:

2.1.10. Beispiel Base- und Auxiliary-Ketten

```
table inet filter01 {
        # auxiliary chain
        chain link-local-aux01 {
                 counter packets 0 bytes 0 log prefix "link-local: " accept
                 # weitere Regeln für link-local IPv6
                 [...]
        }
        # base chain
        chain input01 {
```

```
                type filter hook input priority 0;
                ip6 daddr fe80::/64 jump link-local-aux01
        }
        # base chain
        chain output01 {
                type filter hook output priority 0;
        }
}
```

2.1.11. Filter–Regeln

- Eine Hauptaufgabe einer Firewall ist das Filtern von Netzwerkverkehr.
- Die *nftables*–Firewall bietet umfangreiche Möglichkeiten, Netzwerkpakete in Filterregeln auszuwählen:
  - nach Quell– und Ziel–Adressen
  - UDP– und TCP–Ports
  - Meta–Informationen wie
    - Netzwerk–Interfaces
    - Protokolle
    - Hardware–MAC–Adressen
    - VLAN–Informationen
    - Status einer IP–Verbindung (Connection–Tracking)

2.1.12. Beispiel einer Filter–Regel

- In diesem Beispiel wird eine Regel der Firewall hinzugefügt. Gefiltert wird nach der IPv6–Zieladdresse `daddr ff02::fb` und dem UDP–Port `dport mdns` (Multicast–DNS):

```
# nft "add rule inet filter input ip6 daddr ff02::fb udp dport mdns accept"
```

- Eine Liste der Filter–Optionen findet sich in der *man*–Page zu `nft`.

2.1.13. Verdict Statements

- Am Ende einer Firewall–Regel wird das Urteil (Verdict) über eine Netzwerk–Verbindung getroffen, das sogennante Verdict–Statement.
- Es gibt entgültige Urteile, welche die Bearbeitung eines Netzwerkpaketes sofort beenden.
- Dazu zählen * `accept` (Paket passieren lassen) * `drop` (Paket ohne Rückmeldung an den Sender verwerfen) und * `reject` (mit Rückmeldung verwerfen).
- Bei `reject` kann optional die ICMP–Meldung spezifiziert werden, welche an den Sender des Paketes zurückgeliefert wird:

```
# nft "add rule inet filter input ip6 saddr 2001:db8::/64" \
      "reject with icmpv6 type admin-prohibited"
```

Neben den entgültigen Urteilen kann eine Regel die Bearbeitung eines Paketes auch an anderen Stelle fortsetzen:

- `continue` Bearbeitung des Paketes mit die nächsten Regel in der Kette
- `jump <chain>` verzweigt die Bearbeitung wie ein Unterprogrammaufruf in eine andere Kette. Am Ende der neuen Kette wird die Bearbeitung in der ursprünglichen Kette weitergeführt.
- `goto <chain>` wird direkt in eine andere Kette gesprungen, ohne das die Bearbeitung jemals wieder zurückkehrt.
- `queue` verzweigt die Verarbeitung an ein externes Programm im Userspace.
- `return` beendet die Bearbeitung der aktuellen Kette an dieser Stelle. Wird `return` in einer Kette der obersten Bearbeitungsebene verwendet, so wird das Paket durch die Firewall gelassen (identisch zu einem `accept`).

2.1.14. Logging und Zähler

- Über das Befehlswort *log* werden Informationen zu dem gerade bearbeiteten Netzwerk–Paket in das Kernel–Log (und damit in den meisten Systemen auch in das System–Log) geschrieben.
- Der Befehl *counter* erzeugt einen Zähler für diese Firewall–Regel.
- Anders als bei *iptables* müssen Counter für Regeln explizit erzeugt werden. Dies hat Performance–Gründe, die Zähler werden nur dann mitgeführt, wenn der Administrator diese im Regelsatz angefordert hat.
- In dem Zähler werden die Anzahl der gefilterten Pakete und die Datenmenge in Bytes gezählt.
- Der Zähler wird beim Auflisten des Regelwerkes, z.B. mit `list ruleset`, ausgegeben.
- Die Position des Befehls `log` in der Regel wichtig:
  - Steht der Befehl vor der Filter–Bedingung, so wird ein Log–Eintrag für jedes Paket geschrieben, welches von dieser Regel gesehen wird, unabhängig ob die Regel danach aktiv wird oder nicht.
  - Steht jedoch der Befehl `log` nach der Filterbedingung, so wird der Log–Eintrag nur geschrieben, wenn die Filterbedingung zutrifft.
- Gleiches gilt für die Position des Befehls `counter`.

```
# schreibe einen Log-Eintrag für jedes Paket
# welches von dieser Regel gesehen wird
log tcp dport { 22, 80, 443 } ct state new accept
# schreibe einen Log-Eintrag nur wenn die Regel zutrifft
tcp dport { 22, 80, 443 } ct state new log counter accept
```

- Dem Befehl `log` kann mit dem Parameter `prefix` eine beliebige Zeichenkette mitgegeben werden.
- Diese Zeichenkette erscheint vor jeder Log–Meldung und wird benutzt, um Log–Einträge zu finden und zu filtern.

2.1.15. IPv4 und IPv6 kombinieren

- Neben den Protokollen *ip* für IPv4 und *ip6* für IPv6 unterstützt *nftables* auch das Pseudo–

Protokoll *inet* für kombinierte IPv4– und IPv6–Filter.

- Anstatt z.B. Port 80 für einen Webserver in zwei getrennten Regeln für je IPv4 und IPv6 freizugeben, kann dies in *nftables* in nur einer Regel geschehen.
- Bei einem Server mit Dual–Stack und vielen Diensten kann dies das Regelwerk vereinfachen.
- Für das Protokoll *inet* muss eine eigene Tabelle mit Chains erstellt werden.

```
table inet filter {
    chain input {
       ...
    }
}
```

- Die Tabelle mit dem Pseudo–Protokoll *inet* existiert paralell zu Tabellen mit IPv4– und IPv6–Regeln.
- Werden Tabellen mit dedizierten IPv4 und IPv6 Regeln genutzt, so müssen die Pakete sowohl in der *inet* Tabelle als auch in der jeweiligen Tabelle für das IP–Protokoll erlaubt werden.
- Um Fehler zu vermeiden, sollte entweder kombinierte *inet* Tabellen, oder protokoll–spezifische Tabellen benutzt werden, aber nicht beides.

2.1.16. Datenstrukturen

- *nftables* bietet verschiedene Datenstrukturen, welche die Erstellung eines Regelwerkes vereinfachen:
  - Variablen
  - Intervalle
  - Sets
  - benannte und anonyme Maps
  - benannte und anonyme Dictionaries/Verdict–Maps

2.1.17. Variablen

- Variablen erlauben symbolische Namen für Werte und IP–Adressen in einem *nftables* –Regelwerk.
- Variablen sind jeweils in der Umgebung sichtbar, in der sie definiert wurden (Table, Chain).
- Variablen werden über das Schlüsselwort `define` deklariert und mit einem Wert versehen. Im Regelwerk wird dann der Variablenname mit einem vorangestellem Dollar–Zeichen "$" verwendet, vergleichbar mit Variablen der Unix–Shell.

```
define lan_if = eth0
define DMZ-Dienste = { ssh, smtp, dns, http, https }
add rule inet filter input iif $lan_if tcp dport $DMZ-Dienste accept
```

2.1.18. Intervalle

- Intervalle in *nftables* sind Wertefolgen mit einem Start– und End–Wert.

- Intervalle werden z.B. für IP–Adresse–Bereiche und TCP/UDP–Port–Bereiche verwendet.

```
nft "add rule inet filter input ip6 daddr 2001:db8::0-2001:db8::1000 drop"
nft "add rule inet filter input tcp dport 0-1023 drop"
```

2.1.19. Sets

- Sets definieren eine Sammlung von Werten eines gleichen Typs. Ein Set wird durch geschweifte Klammern eingeleitet und die Elemente des Sets werden mit Komma getrennt.
- Ein anonymes Set mit Port–Nummern (Microsoft SMB/CIFS–Protokoll):

```
nft "add rule inet filter input udp dport { 137, 138, 139, 445 } reject"
```

- Bei der Anlage eines benannten Sets muss *nftables* der Werte–Typ mitgeteilt werden.
- Eine (leider noch unvollständige) Liste der verfügbaren Werte–Typen ist in der *nft* man–Page aufgelistet:

```
nft "add set inet filter bogus { type ipv4_addr; }"
nft "add element inet filter bogus { 203.0.113.0; }"
nft "add element inet filter bogus { 203.0.113.1; }"
nft "add rule inet filter input ip saddr @bogus drop"
nft "list set inet filter bogus"
```

- Der letzte Befehl des Beispiel gibt den aktuellen Inhalt des benannten Sets *bogus* aus.
- Ein benanntes Set kann in einer Regel mit vorangestelltem "@" eingefügt werden.
- Eine Filterregel mit benanntem Set wird vom der *nftables* Virtuellen–Maschine schneller bearbeitet als eine Reihe von Einzel–Regeln.

2.1.20. Anonyme und benannte Maps und Dictionaries

- Maps und Dictionaries existieren in anonymen (statischen) und benannten Versionen.
- Die anonymen Versionen werden direkt in die Regel geschrieben und sind damit fester Bestandteil der Regel.
- Die Inhalte von anonymen Datenstrukturen können nicht zu einem späeteren Zeipunk geändert werden.
- Bei den benannten Maps und Dictionaries können die Inhalte nach dem Laden des Firewall–Regelwerkes noch angepasst werden.
- So können z.B. Programme wie *fail2ban* oder *denyhost* direkt mit der Firewall interagieren und IP–Adressen von Angreifern sperren.
- Oder eine Anti–Spam–Dienst kann das Einliefern von Spam–E–Mail über die IP–Adressen von identifizierten Spammern mittels einer Rate–Limit–Regel drosseln.

2.1.21. Maps

- Maps oder Data–Maps verbinden zwei Werte miteinander.
- Der Eingangswert wird in der Liste der Index–Werte der Map gesucht und der dort gespeicherte Wert wird an die Regel zurückgegeben.

- Eine Beispiel–Anwendung ist das Verzweigen auf verschiedene interne Server mit privaten Adressen per NAT, basierend auf der Port–Nummer des Dienstes:

```
# nft "add rule ip nat prerouting dnat" \
      "tcp dport map { 80 : 192.168.1.100, 8888 : 192.168.1.101 }"
```

2.1.22. Verdict–Maps/Dictionaries

- Verdict–Maps, auch Dictionaries genannt, verbinden einen Eingangswert des zu betrachtenen Pakets mit der Filter–Entscheidung einer Regel.
- Somit lassen sich für die verschiedenen Werte eines IP–Paketes automatisch unterschiedliche Aktionen auslösen.
- In diesem Beispiel werden die Ergebnisse einer Regel an die Quell–IPv4–Adresse des Netzwerk gebunden:

```
nft "add map inet filter aktion { type ipv4_addr : verdict; }"

nft "add element inet filter aktion" \
    "{ 192.0.2.80 : accept, 192.0.2.88 : drop, 192.0.2.99 : drop }"

nft "add rule inet filter input ip saddr vmap @aktion"
```

2.1.23. Beispiel: ein Regelwerk für eine Host–Firewall

- Das nachfolgende *nftable*–Regelwerk kann als Grundlage für eine Host–Firewall für eine Linux–Workstation benutzt werden.
- Gefiltert werden die eingehenden IPv4– und IPv6–Verbindungen.
- Gängige Dienste auf einer Linux–Workstation wie Internet Printing Protocol (IPP/Cups) und Multicast–DNS werden erlaubt:

```
flush ruleset

define any = 0::0/0

table inet filter {
        chain input {
                type filter hook input priority 0;

                # accept any localhost traffic
                iif lo accept

                # accept traffic originated from us
                ct state established,related accept

                # activate the following line to accept common local services
                #tcp dport { 22, 80, 443 } ct state new accept

                # NTP multicast
                ip6 daddr ff02::1010 udp dport 123 accept

                # mDNS (avahi)
```

```
                    ip6 daddr ff02::fb udp dport 5353 accept
                    ip  daddr 224.0.0.251 udp dport 5353 accept

                    # DHCPv6
                    ip6 saddr $any udp dport 546 accept

                    # IPP (CUPS)
                    udp dport 631 accept

                    # Accept neighbour discovery otherwise IPv6 connectivity breaks.
                    ip6 saddr $any icmpv6 type { nd-neighbor-solicit,  nd-router-
advert, nd-neighbor-advert }  accept

                    # Accept essential icmpv6
                    # nft cannot parse "param-problem" (icmpv6 type 4)
                    ip6 saddr $any icmpv6 type { echo-reply, echo-request, packet-too-
big, destination-unreachable, time-exceeded, 4 } accept

                    # count and drop any other traffic
                    counter log prefix "nftables drop: " drop
            }
    }
```

2.1.24. Beispiel: ein Regelwerk für einen Gateway-Router mit IPv4-NAT

- Nachfolgend ein *nftables*-Regelwerk für eine einfache Firewall mit DMZ-, WAN- und LAN-Schittstelle. In der DMZ befinden sich die Server mit Web und E-Mail.
- In der Postrouting-Kette werden alle Pakete aus dem LAN-Segment, welche die Firewall in Richtung Internet (WAN) verlassen, per Masquerading-NAT auf die IPv4-Adresse der WAN-Netzwerkschnittelle umgeschrieben.

```
    #!/usr/bin/nft -f
    flush ruleset
    define mgt_if = eth0
    define wan_if = eth1
    define lan_if = eth3
    define dmz_if = eth2
    define any_v6  = 0::0/0

    table inet gateway-fw {
            chain forward {
                    type filter hook forward priority 0
                    # accept established traffic
                    ct state established,related accept
                    # allow all outgoing traffic from LAN
                    iif $lan_if accept
                    # access to services in the DMZ
                    oif $dmz_if tcp dport { http, https, smtp, imap, imaps } counter
accept
                    # Accept essential icmpv6
                    # nft cannot parse "param-problem" (icmpv6 type 4)
                    ip6 saddr $any_v6 icmpv6 type { echo-reply, echo-request, packet-
too-big, destination-unreachable, time-exceeded, 4 } accept
```

```
                    # reject everything else
                    counter log reject
            }
            chain input {
                    type filter hook input priority 0
                    iif lo accept
                    iif $mgt_if tcp dport ssh accept
                    ip6 saddr $any_v6 icmpv6 type { nd-neighbor-solicit,  nd-router-
advert, nd-neighbor-advert }  accept
                    ip6 saddr $any_v6 icmpv6 type { echo-reply, echo-request, packet-
too-big, destination-unreachable, time-exceeded, 4 } accept
                    counter log reject
            }
    }
    table ip nat {
             chain prerouting {
                    type nat hook prerouting priority 0
             }
             chain postrouting {
                    type nat hook postrouting priority 0
                    oif $wan_if log masquerade
             }
    }
```

2.1.25. Fehlersuche durch Debug-Tracing

- *nftables* erlaubt es mittels "ruleset tracing" den Weg eines Pakets durch das Firewall-Regelwerk zu verfolgen
- Das *tracing* kann dabei für alle oder für ausgewählte Pakete einer Kette (Chain) aktiviert werden

```
table ip filter {
        chain input {
                type filter hook input priority filter;
        policy drop;
        # trace enabled for all packets in this chain
        meta nftrace set 1
                ct state established,related counter accept
                ct state new tcp dport 22 counter accept
        }
}
```

- *Tracing* nur für SSH-Pakete

```
table ip filter {
        chain input {
                type filter hook input priority filter;
        policy drop;
        # trace enabled for SSH packets in this chain
        tcp dport 22 meta nftrace set 1
                ct state established,related counter accept
                ct state new tcp dport 22 counter accept
        }
}
```

- *Tracing* Daten ausgeben. Zu jedem Paket wird eine eindeutige *trace id* angegeben. Alle Ausgaben mit der gleichen *trace id* wurden durch das gleiche IP–Paket erzeugt.

```
nft monitor trace
```

- *Tracing* im nftables Wiki: https://wiki.nftables.org/wiki–nftables/index.php/Ruleset_debug/tracing [https://wiki.nftables.org/wiki–nftables/index.php/Ruleset_debug/tracing]

2.1.26. Aufgabe: FirewallD

- Der Dienst `firewalld` sollte aktiv sein. `firewalld` ist ein Konfiguration–Fronted für `nftables` (und `iptables`).
- Liste die aktuellen, durch `firewalld` erstellen `nftables` Firewall–Regeln auf

```
nft list ruleset
```

- Stoppe den `firewalld` Firewall–Dienst. Alle Firewall–Regeln sollten nun gelöscht worden sein:

```
systemctl stop firewalld
nft list ruleset
```

2.1.27. Aufgabe: Einfacher Webserver

- Installiere die Pakete `lynx` (Shell–Webbrowser) und `nmap` (Netzwerk–Scan) auf VM2

```
dnf install lynx nmap
```

- Starte einen einfachen Python3 Webserver auf VM1. Dieser Server horcht auf HTTP–Verbindungen auf Port `8000`

```
python3 -m http.server
```

- Starte den Web–Browser `lynx` auf VM2 und lade die Startseite des Webserver auf VM1. Es sollten die Dateien im Verzeichnis erscheinen, in welchem der Python3–Webserver gestartet wurde

```
lynx http://<private-ip-von-VM1>:8000
```

- Scanne von VM2 aus mittels `nmap` die populären Ports auf VM1:

```
nmap -sT -vv <private-ip-von-VM1>
```

- Es sind 4 Ports erreichbar:

```
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
443/tcp  open  https
8000/tcp open  http-alt
```

- Installiere und benutze den Befehl `lsof` (List open files) auf VM1 um die Programme zu den offenen Ports zu finden

```
lsof -Poni :<Port>
```

- Beispiel

```
lsof -Poni :8000
```

- Beende den Webserver auf VM1 mit CTRL+C

2.1.28. Aufgabe: IPv4 Host-Firewall

- Erstelle eine Host-Firewall auf VM1 in der Datei `/etc/nftables.conf`, welche auf Netzwerkschnittstelle `vti01` nur folgende Zugriffe erlaubt
  - ICMP Echo-Request/Echo-Reply
  - Port 80 (HTTP), Port 443 (HTTPS) und 22 (SSH)
- Starte einen Python-Webserver als Benutzer `root` auf Port 80

```
python3 -m http.server 80
```

- Scanne per `nmap` von VM2 nach offenen Ports auf VM1, einmal mit `nftables` Firewall aktiv, einmal ohne eingeschaltete `nftables` Firewall VM1. Gibt es Unterschiede in der Ausgabe?

```
vm2# nmap -v -sT <private-IP-von-VM1>
```

2.1.29. Erweiterte Aufgabe:

- Starte den Python Webserver auf Port 8000

```
python3 -m http.server 8000
```

- Erstelle eine `redirect` oder `dnat` Destination-NAT Regel (Tabelle `nat`, Chain `prerouting`), welche Zugriffe auf Port 80 des Rechners auf Port 8000 umleitet: *Nftables* Dokumentation: `man nft`
- Teste mit `lynx` von VM2 aus, ob der Browser auf Port 80 den Webserver erreichen kann

2.1.30. Beispiel-Lösung

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

flush ruleset

table ip filter {
        chain input {
                type filter hook input priority 0;
        policy accept;

                iif lo accept

        iif vti01 icmp type { echo-reply, echo-request } accept
                iif vti01 ct state established,related accept
                iif vti01 tcp dport { ssh, http, https } ct state new accept
```

```
        iif vti01 counter log prefix "nftables drop: " drop
        }
}
```

- **Regelwerk nach Änderungen neu laden**

```
systemctl restart nftables
```

- **Aktives Regelwerk inkl. der Zähler anzeigen**

```
nft list ruleset
```

- **Log-Meldungen erscheinen im Syslog (`/var/log/messages`) und im Journal**

```
journalctl -k --grep "LEN="        # Firewall Meldungen filtern
journalctl -k --grep nftables      # eigene Firewall Meldungen mit Prefix "nftables"
journalctl -k -f --grep nftables   # Firewall Log im "follow" Modus
```

2.1.31. Beispiel-Lösung (Erweitert)

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

flush ruleset

table ip nat {
    chain prerouting {
        type nat hook prerouting priority 0;
        iif vti01 tcp dport 80 redirect to 8000
    }
    chain postrouting {
        type nat hook postrouting priority 0;
    }

}

table ip filter {
    chain input {
        type filter hook input priority 0;
        policy accept;

        iif lo accept

        iif vti01 icmp type { echo-reply, echo-request } accept
        iif vti01 ct state established,related accept
        iif vti01 tcp dport { ssh, http, https, 8000 } ct state new accept

        iif vti01 counter log prefix "nftables drop: " drop
    }
}
```

2.2. WIREGUARD

- Wireguard ist eine VPN–Lösung von Jason A. Donenfeld
- Wireguard wurde 2015 vorgestellt und ist für Linux, OpenBSD (Teil des Basis–OS), NetBSD, FreeBSD, Windows, iOS und Android verfügbar
- Wireguard ist explizit schlank gehalten und vermeidet die Komplexität anderer VPN–Systeme
  - Die Algorithmen sind fest definiert, werden nicht im Protokoll ausgehandelt und können nicht durch die Konfiguration geändert werden (Noise protocol framework, Curve25519, ChaCha20, Poly1305, BLAKE2, SipHash24, HKDF)
  - Der Programcode ist überschaubar, die Krypto–Implementierungen wurden einer formalen Verifizierung unterzogen (sind mathematisch beweisbar korrekt) und wird regelmässig einem Sicherheitsaudit unterzogen (über das OpenBSD–Projekt)
- Unter Linux ist Wireguard als Kernel–Modul implementiert
- Wireguard bietet sehr guten Durchsatz und wenig Latenz, ist hierbei oft besser als OpenVPN oder IPsec
- Wireguard benutzt ausschliesslich Kryptograhische–Schlüssel zur Authentisierung, die IP–Adressen der Kommunikationspartner können beliebig sein
  - Dies erlaubt die Benutzung durch NAT
  - Die IP–Adressen der Kommunikations–Endpunkte können wechseln, ohne das die VPN–Verbindung abbricht (Cryptrokey Routing, Roaming)
  - Wireguard ist ein Peer–to–Peer VPN, das Protokoll macht keine Unterscheidung in der Funktion der Kommunikationspartner
- Wireguard transportiert die verschlüsselten Daten via UDP, der UDP–Port ist frei wählbar und kann auf einer Seite dynamisch sein
- Wireguard ist in kurzer Zeit sehr populär geworden, ist in der Firmware vieler (Heim–)Router vorinstalliert und wird von vielen kommerziellen VPN–Anbietern angeboten

2.2.1. Wireguard AlmaLinux/Rocky–Linux Installation

- Unter Alme/Rocky–Linux können wir Wireguard direkt aus den Paketquellen installieren

```
dnf install wireguard-tools
```

2.2.2. Konfiguration Wireguard auf dem VPN–Server (Gateway)

- User–Mask für neue Dateien und Verzeichnisse setzen, ein Verzeichnis für die Wireguard–Dateien erstellen

```
umask 077
mkdir -p /etc/wireguard
cd /etc/wireguard
```

- Einen privaten Schlüssel für den Wireguard–Dienst erstellen

```
wg genkey > private.key
```

- Wireguard–Interface erstellen

```
ip link add dev wg0 type wireguard
```

- IP–Adresse auf dem neuen Wireguard–Interface `wg0` setzen

```
ip addr add 10.0.0.2/24 dev wg0
```

- Privaten Schlüssel in das `wg0` Interface laden

```
wg set wg0 private-key ./private.key
```

- Interface `wg0` aktivieren

```
ip link set wg0 up
```

- Konfiguration der Schnittstelle anzeigen

```
wg show wg0
```

- Port des Wireguard–Server in der Firewall öffnen

```
firewall-cmd --add-port <port#>/udp --permanent
firewall-cmd --reload
```

2.2.3. Konfiguration Wireguard auf dem VPN–Client

- Wireguard installieren.

```
dnf install wireguard-tools
```

- Verzeichnis für die Wireguard–Dateien erstellen

```
umask 077
mkdir /etc/wireguard
cd /etc/wireguard
```

- Einen privaten Schlüssel für den Wireguard–Dienst erstellen

```
wg genkey > private.key
```

- Wireguard–Interface erstellen

```
ip link add dev wg0 type wireguard
```

- IP–Adresse auf das neue Wireguard–Interface `wg0` setzen

```
ip addr add 10.0.0.1/24 dev wg0
```

- Privaten Schlüssel in das `wg0` Interface laden

```
wg set wg0 private-key ./private.key
```

- Interface `wg0` aktivieren

```
ip link set wg0 up
```

- Öffentlicher Schlüssel, erlaubte IP-Adressen und IP/Port des VPN-Servers eintragen

```
wg set wg0 peer <schlüssel> allowed-ips 10.0.0.0/8 endpoint <vpn-server>:<port>
```

- Konfiguration der Schnittstelle anzeigen

```
wg show wg0
```

- Konfiguration in die Konfigurationsdatei speichern

```
wg showconf wg0 > /etc/wireguard/wg0.conf
chmod 0600 /etc/wireguard/wg0.conf
```

- Konfigurationsdatei anpassen, `Address` einfügen

```
[Interface]
ListenPort = <port>
PrivateKey = <private-key>
Address = 10.0.0.1/24

[Peer]
PublicKey = <public-key-of-vpn-server>
AllowedIPs = 10.0.0.0/8
Endpoint = <ip-of-vpn-server>:<port>
```

2.2.4. Client auf dem Server erlauben

- Auf dem VPN-Server, die Wireguard-Configuration speichern

```
wg showconf wg0 > /etc/wireguard/wg0.conf
chmod 0600 /etc/wireguard/wg0.conf
```

- Konfigurationsdatei anpassen (`Address` und Abschnitt `Peer`)

```
[Interface]
ListenPort = <port>
PrivateKey = <private-key>
Address = 10.0.0.200/24

[Peer]
PublicKey = <public-key-of-client>
AllowedIPs = 10.0.0.1/32
```

2.2.5. Wireguard manuell aktivieren

- auf dem VPN-Gateway

```
vm0X:/etc/wireguard# wg-quick down wg0
[#] ip link delete dev wg0
vm0X:/etc/wireguard# wg-quick up wg0
[#] ip link add wg0 type wireguard
```

```
[#] wg setconf wg0 /dev/fd/63
[#] ip link set mtu 1420 up dev wg0
[#] ip route add 10.0.0.1/32 dev wg0
```

- auf dem VPN-Client

```
vm0Y% /etc/wireguard# wg-quick down wg0
[#] ip link delete dev wg0
vm0Y%/etc/wireguard# wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip link set mtu 1420 up dev wg0
[#] wg set wg0 fwmark 51820
[#] ip -4 route add 0.0.0.0/0 dev wg0 table 51820
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
```

2.2.6. VPN Test

- Ping vom VPN-Client Wireguard-IP zum VPN-Server

```
ping 10.0.0.200
```

2.2.7. Wireguard per systemd starten

- Die folgenden Schritte sowohl auf dem VPN-Gateway, als auch auf dem Client ausführen
- Manuell gestartetes Wireguard beenden

```
wg-quick down wg0
```

- Wireguard per Systemd aktivieren

```
systemctl enable wg-quick@wg0
systemctl start wg-quick@wg0
systemctl status wg-quick@wg0
● wg-quick@wg0.service - WireGuard via wg-quick(8) for wg0
   Loaded: loaded (/lib/systemd/system/wg-quick@.service; enabled; vendor preset:
enabled)
   Active: active (exited) since Mon 2019-05-06 16:18:02 EDT; 1s ago
     Docs: man:wg-quick(8)
       man:wg(8)
       https://www.wireguard.com/
       https://www.wireguard.com/quickstart/
       https://git.zx2c4.com/WireGuard/about/src/tools/man/wg-quick.8
       https://git.zx2c4.com/WireGuard/about/src/tools/man/wg.8
  Process: 13492 ExecStart=/usr/bin/wg-quick up wg0 (code=exited, status=0/SUCCESS)
 Main PID: 13492 (code=exited, status=0/SUCCESS)

May 06 16:18:02 router05 systemd[1]: Starting WireGuard via wg-quick(8) for wg0...
May 06 16:18:02 router05 wg-quick[13492]: [#] ip link add wg0 type wireguard
May 06 16:18:02 router05 wg-quick[13492]: [#] wg setconf wg0 /dev/fd/63
May 06 16:18:02 router05 wg-quick[13492]: [#] ip address add 10.0.0.1/24 dev wg0
May 06 16:18:02 router05 wg-quick[13492]: [#] ip link set mtu 1420 up dev wg0
```

```
May 06 16:18:02 router05 wg-quick[13492]: [#] wg set wg0 fwmark 51820
May 06 16:18:02 router05 wg-quick[13492]: [#] ip -4 route add 0.0.0.0/0 dev wg0 table
51820
May 06 16:18:02 router05 wg-quick[13492]: [#] ip -4 rule add not fwmark 51820 table
51820
May 06 16:18:02 router05 wg-quick[13492]: [#] ip -4 rule add table main
suppress_prefixlength 0
May 06 16:18:02 router05 systemd[1]: Started WireGuard via wg-quick(8) for wg0.
```

2.2.8. Wireguard Fehlersuche

- Kernel-Logging aktivieren

```
echo module wireguard +p > /sys/kernel/debug/dynamic_debug/control
```

- Wireguard Logs im Kernel verfolgen

```
journalctl -fk --grep wireguard
```

2.2.9. Links

- BoringTUN: Wireguard Implementation in Rust https://github.com/cloudflare/boringtun [https://github.com/cloudflare/boringtun]
- Route-based VPN on Linux with WireGuard https://vincent.bernat.ch/en/blog/2018-route-based-vpn-wireguard [https://vincent.bernat.ch/en/blog/2018-route-based-vpn-wireguard]
- Four Ways to View WireGuard Logs https://www.procustodibus.com/blog/2021/03/wireguard-logs/ [https://www.procustodibus.com/blog/2021/03/wireguard-logs/]

2.3. OPENVPN

2.3.1. OpenVPN Informationen

- OpenVPN ist eine populäre VPN-Lösung basierend auf OpenSSL, die erste Version von OpenVPN wurde im Jahr 2001 veröffentlicht
- Das OpenVPN Protokoll wurde für kleine VPN-Strukturen entwickelt
- Anders als Wireguard und IPsec (Peer-to-Peer VPN) ist OpenVPN ein dediziertes Client-Server VPN, d.h. das Protokoll unterscheidet aktiv zwischen einer Server-Rolle und einer Client-Rolle.
- OpenVPN benutzt SSL/TLS über UDP oder TCP, benutzt hierbei aber nicht die RFC-Standard Protokolle für TLS (over TCP) oder DTLS (TLS over UDP) sondern eine Eigenentwicklung
- Verglichen mit IPsec in 2001 (mit IKEv1 und den Probleme mit NAT) war OpenVPN damals einfacher zu konfigurieren und wurde schnell sehr populär
- OpenVPN-Implementationen existieren für die meisten Systeme (Linux, xBSD, Windows, macOS, iOS, Android ...)
- Erst seit Kernel 6.16 (April 2025) gibt es ein OpenVPN-Kernel-Modul, bei welchem die Daten-Verbindung von OpenVPN im Linux-Kernel verschlüsselt wird. Vorherige Versionen von OpenVPN, und OpenVPN auf anderen Plattformen, verschlüsseln die Daten innerhalb des Userspace-Prozesses, was bedingt die Daten von Kernel zum OpenVPN-Userland-Prozess und

wieder zurück gesendet werden. Dies erhöht die Latenz und vermindert den Durchsatz der VPN–Verbindung.

- OpenVPN Protokoll RFC–Draft: https://github.com/openvpn/openvpn–rfc [https://github.com/openvpn/openvpn–rfc] und https://openvpn.github.io/openvpn–rfc/openvpn–wire–protocol.html [https://openvpn.github.io/openvpn–rfc/openvpn–wire–protocol.html]
- Die Entwicklung von OpenVPN wird heute von der Firma OpenVPN Inc weiterentwickelt und unterstützt

2.3.2. OpenVPN mit "Pre–Shared–Keys"

- OpenVPN kann verschiedene Arten der Autentifizierung (Asymmetrische Schlüssel, x509 Zertifikate) verwenden, in unserem Beispiel benutzen wir zuerst einfache kryptografische Schlüssel (PSK oder Pre–Shared–Keys)
- In unserem Beispiel werden die IP–Adressen der OpenVPN Teilnehmer manuell verteilt. Es ist auch möglich, die IP–Adressen der VPN–Clients über das OpenVPN–Gateway zu verteilen.
- Schritt 1: OpenVPN auf dem Client und auf dem Server installieren
  - Auf dem Server (VM1)

```
server% dnf install epel-release
server% dnf install openvpn
```

- Auf dem Client (VM2)

```
client% dnf install epel-release
client% dnf install openvpn
```

- Die Schlüssel erzeugen und per **gesicherten** Kanal auf beide Endpunkte (hier zusätzlich zum Client auch auf den Server) des VPN–Tunnels bringen (z.B. per Secure Copy `scp`)

```
client% cd /etc/openvpn
client% openvpn --genkey secret static.key
client% scp static.key user@vpnNNN.dane.onl:/tmp
```

- Server–Konfiguration `/etc/openvpn/server.conf` (auf dem Server–Rechner)

```
dev tun
ifconfig 10.0.1.1 10.0.1.2
secret /etc/openvpn/static.key
cipher aes-256-cbc
```

- Auf dem Server den geheimen Schlüssel aus dem Verzeichnis `/tmp` in das OpenVPN Verzeichnis verschieben

```
server% mv /tmp/static.key /etc/openvpn/
```

- Client–Konfiguration `/etc/openvpn/client.conf` (auf dem Client–Rechner)

```
remote <oeffentliche-ip-des-servers>
dev tun
ifconfig 10.0.1.2 10.0.1.1
secret /etc/openvpn/static.key
```

```
cipher aes-256-cbc
```

- Treiber für Tunnel-Interface laden und OpenVPN start (Server)

```
server% modprobe tun
server% openvpn --config /etc/openvpn/server.conf --allow-deprecated-insecure-static
-crypto  &
```

- Auf dem Server einen Webserver (Port 8000) starten

```
server% python3 -m http.server 8000
```

- OpenVPN start (Client)

```
client% modprobe tun
client% openvpn --config /etc/openvpn/client.conf --allow-deprecated-insecure-static
-crypto &
```

- Tunnel-Interfaces anzeigen (Es sollte eine Netzwerkschnittstelle `tun0` mit der 10.0.1.x Adresse existieren)

```
client% ip a
```

- OpenVPN per ICMP `ping` und HTTP-Anfrage testen (mit `tcpdump` auf dem Client und Server die Pakete auf den physischen Netzwerkschnittellen anschauen `tcpdump -i eth0 port 80` und `tcpdump -i eth0 icmp`)

```
client% ping 10.0.1.1
client% lynx http://10.0.1.1:8000
```

2.3.3. OpenVPN mit Zertifikaten

Zertifizierungsstelle (CA) / Server

- In unserer Schulungsumgebung werden wir die CA (Certification Authority, private Zertifizierungsstellung) zum Austellen von x509-Zertifikaten auf der Server-VM installieren. In Produktionsumgebungen sollte die CA auf einem gesondert gesichertem System und getrennt von den OpenVPN-Servern eingerichtet werden.
- *Easy-RSA* Skriptesammlung zum Verwalten einer Zertifizierungs-Stelle (Certification Authority, CA) installieren

```
dnf install easy-rsa
```

- Verzeichnis für die CA-Daten erstellen

```
mkdir -p /etc/openvpn/ca
```

- Alle weiteren Befehle werden innerhalb dieses neuen Verzeichnisses ausgefuehrt

```
cd /etc/openvpn/ca
```

- Easy-RSA CA initialisieren

```
/usr/share/easy-rsa/3/easyrsa init-pki
```

- Eine CA einrichten (CA-Schluessel und Root-Zertifikat ohne Passwort erstellen)

```
/usr/share/easy-rsa/3/easyrsa build-ca nopass
```

Zertifikat fuer den OpenVPN-Server erstellen

- Ein Zertifikat fuer den Server erstellen

```
/usr/share/easy-rsa/3/easyrsa gen-req server nopass
```

- Das Server Zertifikat mit dem Root-CA-Schluessel unterschreiben (Signatur)

```
/usr/share/easy-rsa/3/easyrsa sign-req server server
```

Zertifikat fuer einen Client erstellen

- Ein Zertifikat fuer einen OpenVPN-Client erstellen

```
/usr/share/easy-rsa/3/easyrsa gen-req user1 nopass
```

- Und dieses Zertifikat mit dem Schluessel der CA unterschreiben

```
/usr/share/easy-rsa/3/easyrsa sign-req client user1
```

OpenVPN-Server Konfigurieren

- Neue Diffie-Hellman Parameter fuer den OpenVPN-Server erstellen

```
/usr/share/easy-rsa/3/easyrsa gen-dh
```

- OpenVPN-Konfiguration in `/etc/openvpn/server/server-cert.conf`

```
port 1194
proto udp
dev tun

ca /etc/openvpn/ca/pki/ca.crt
cert /etc/openvpn/ca/pki/issued/server.crt
key /etc/openvpn/ca/pki/private/server.key

dh /etc/openvpn/ca/pki/dh.pem

topology subnet

server 10.8.0.0 255.255.255.0

ifconfig-pool-persist ipp.txt

keepalive 10 120

max-clients 100
```

```
user openvpn
group openvpn

persist-tun

status openvpn-status.log

verb 3

explicit-exit-notify 1
```

- Wurde das Zertifikat des Servers auf einem gesonderten Rechner erstellt, dann das Zertifikat der CA (aber nicht der private Schluessel der CA), der private Schluessel des Servers und das Zertifikat des Servers auf den Server-Rechner übertragen
  - `/etc/openvpn/ca/pki/ca.crt`
  - `/etc/openvpn/ca/pki/issued/server.crt`
  - `/etc/openvpn/ca/pki/private/server.key`
- Port 1194 (OpenVPN) in der Firewall freischalten

```
firewall-cmd --add-service=openvpn --permanent
firewall-cmd --reload
```

- OpenVPN Server manuell starten

```
openvpn /etc/openvpn/server/server-cert.conf
```

OpenVPN Client (Alma-Linux)

- OpenVPN installieren
- Verzeichnis fuer die Zertifikats-Dateien erstellen

```
mkdir -p /etc/openvpn/pki
```

- Die folgenden Dateien von der CA-Maschine auf **sicherem** Kanal auf den OpenVPN-Client Rechner uebertragen und unter `/etc/openvpn/pki` ablegen
  - `ca.crt`
  - `client1.crt`
  - `client1.key`
- OpenVPN Client Konfiguration `/etc/openvpn/client/user1.conf`

```
client
dev tun
persist-tun
proto udp

remote server.domain.tld 1194
resolv-retry infinite
```

```
nobind

user openvpn
group openvpn

ca /etc/openvpn/pki/ca.crt
cert /etc/openvpn/pki/user1.crt
key /etc/openvpn/pki/user1.key
remote-cert-tls server

verb 3
```

- OpenVPN auf dem Client manuell starten

```
openvpn /etc/openvpn/client/user1.conf
```

2.3.4. Sicherheitsbetrachtung von OpenVPN

- Sektion 5.4.2 "Security Considerations" des OpenVPN-Protokoll-Draft beschreibt potentielle Probleme des OpenVPN-Protokolls. Das OpenVPN-Protokoll bietet weniger strukturelle Sicherheit gegen Angriffe verglichen mit (gut konfiguriertem) IPsec oder Wireguard.
- Es kann vorkommen das die IV (Initialisierungs-Vektoren) von OpenVPN-Paketen verschiedener Verbindungen kollidieren. Hat der Angreifer Sicht auf die (verschlüsselte) Kommunikation, so ist es in diesem Fall möglich durch eine XOR-Funktion der verschlüsselten Daten den Session-Key (aller) Verbindungen zu erlangen und damit die Verschlüsselung zu brechen.
- Bei der Funktion "tls-crypt" teilen sich alle Clients (und der OpenVPN-Server) den gleichen Gruppen-Schlüssel. Wird ein Client kompromitiert, so kann der Angreifer damit die Kommunikation aller Clients und des Servers angreifen. Abhilfe schafft die Benutzung von "tls-crypt-v2" (ab OpenVPN Access-Server 2.9 — kommerzielle Version von OpenVPN — aktuell in Alma-Linux 10 ist OpenVPN "Community Edition" Version 2.7)
- OpenVPN ist eine "monolithische" Software, alle Bestandteile laufen im gleichen Prozess-Kontext. Dies erhöht die Angriffsfläche, ein Fehler in einem Bereich von OpenVPN (z.B. Authentisierung) komprimitiert die gesamte Funktion.

CHAPTER 3. EXTRA THEMEN

3.1. NAMESPACES

- Dieses Kapitel zeigt, das Namespaces (und damit Container) eine Standard–Technologie des Linux–Kernel ist und keiner weiteren Software benötigt. `systemd–nspawn` und auch **Docker** sind *nur* Verwaltungsprogramme für die Namespaces und Controll–Groups des Linux–Kernels.
- Namepspaces 'virtualisieren' die Sicht auf Ressourcen des Linux–Kernels
- Programme wie Docker, Chrome, systemd–nspawn, LXC/LXD, Firejail etc. benutzen die Namespaces im Linux–Kernel
- Verfügbare Namespaces in Linux

Namespace	Konstante	Isolation
Cgroup	CLONE_NEWCGROUP	Cgroup root Verzeichnis (Ressourcen wie CPU/RAM)
IPC	CLONE_NEWIPC	System V IPC, POSIX message queues
Network	CLONE_NEWNET	Network devices, stacks, ports, Firewall etc.
Mount	CLONE_NEWNS	Mount points (Dateisysteme)
PID	CLONE_NEWPID	Process IDs
User	CLONE_NEWUSER	Benutzer und Gruppen IDs
UTS	CLONE_NEWUTS	Hostname und NIS Domain Name

3.1.1. Netzwerk Namespaces per `ip` Kommando

- Netzwerk Namespace **netns1** erzeugen

```
# ip netns add netns1
# ip netns list
```

- Befehl im Netzwerk–Namespace ausführen

```
# ip netns exec netns1 ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

- Loopback ist "down", ein `ping` geht nicht

```
# ip netns exec netns1 ping 127.0.0.1
connect: Network is unreachable
```

- Nun bringen wir das Loopback–Interface "up"

```
# ip netns exec netns1 ip link set dev lo up
# ip netns exec netns1 ping 127.0.0.1
```

- Wir erstellen nun **virtuelle** Netzwerkschnittstellen

```
# ip link add veth0 type veth peer name veth1
# ip link
```

- Netzwerkschnittstelle **veth1** wird in den Netzwerk–Namespace **netns1** verschoben

```
# ip link set veth1 netns netns1  # auch physische NICs koennen in Namespaces
verschoben werden
# ip link
```

- Wir konfigurieren ein IP–Adresse auf **veth1** im Namespace

```
# ip netns exec netns1 ifconfig veth1 10.1.1.1/24 up
# ip netns exec netns1 ip a
# ping 10.1.1.1  # ping in den Namespace geht jetzt noch nicht!
```

- Eine IP–Adresse auf dem **veth0** auf dem Linux–Host konfigurieren

```
ifconfig veth0 10.1.1.2/24 up
```

- Ping von aussen in den Namespace geht jetzt!

```
# ping 10.1.1.1
```

- Ping aus den Namespace nach **draussen** sollte auch gehen

```
ip netns exec netns1 ping 10.1.1.2
```

- der Namespace hat eine eigene Routing–Tabelle und eine eigene Netfilter–Firewall

```
ip netns exec netns1 route
ip netns exec netns1 ip route show
ip netns exec netns1 iptables -L
```

- Namespace wieder löschen

```
ip netns delete netns1
```

3.1.2. Einfache Container mit `unshare`

- Mit dem Programm `unshare` können die einzelnen Namespaces separat erstellt und eingeschaltet werden. Wird kein zu startendes Programm angegeben, so wird die Standard–Shell des Systems mit den neuen Namespaces gestartet. Beispiel: ein Namespace mit privatem Netzwerk, Mount– und Prozess–ID– Namespace:

```
    # unshare -n -p -f --mount-proc
```

- Container mit Benutzer `root`, welcher nicht `root` auf dem Host ist

```
   sudo unshare --map-root-user --user sh
```

3.1.3. Namespaces auflisten

- Befehl `lsns`:

```
[root@centos-sec-oct-2017 ~]# lsns
        NS TYPE  NPROCS   PID USER   COMMAND
4026531836 pid       81     1 root   /usr/lib/systemd/systemd --system --deserialize 22
4026531837 user      81     1 root   /usr/lib/systemd/systemd --system --deserialize 22
4026531838 uts       81     1 root   /usr/lib/systemd/systemd --system --deserialize 22
4026531839 ipc       81     1 root   /usr/lib/systemd/systemd --system --deserialize 22
4026531840 mnt       78     1 root   /usr/lib/systemd/systemd --system --deserialize 22
4026531856 mnt        1    12 root   kdevtmpfs
4026531956 net       81     1 root   /usr/lib/systemd/systemd --system --deserialize 22
4026532212 mnt        1 11299 root   /usr/lib/systemd/systemd-udevd
4026532227 mnt        1   551 chrony /usr/sbin/chronyd
```

3.1.4. Container/Namespace mit Systemd

- Jedes Linux mit Systemd bietet mächtige Container-Verwaltungs-Werkzeuge mit
- `systemd-nspawn` arbeitet neben Image-Dateien für Container auch mit installieren Linux-Root-Dateien in einem beliebigen Verzeichnis auf dem Container-Host-System
    - Damit ist es sehr einfach, ein Container-System aufzusetzen und Dateien zwischen dem Container-Host und dem Linux im Container auszutauschen
- In diesem Kapitel installieren wir als Beispiel ein Rocky-Linux (eine freie Red Hat Enterprise Linux Distribution) in einem Verzeichnis unter Debian
    - Andere Linux-Distributionen wie Ubuntu, Suse, Arch-Linux oder ältere Debian-Versionen wären auch möglich
    - Wichig ist das die Dateien im Container-Verzeichnis für die CPU-Architektur des Container-Hosts übersetzt wurden und die Programme keinen neueren Kernel benötigen (keine neuen System-Calls)
- Seit Debian 9 müssen die Systemd-Container-Tools separat installiert werden:

```
dnf install systemd-container
```

- Wir erstellen ein Verzeichnis für den neuen Container

```
mkdir -p /srv/container/rocky-linux8
```

- Initialisieren eine neue RPM-Instanz im Container-Verzeichnis (bei Debian mit **debboostrap**, bei SUSE mit **RPM** und **zypper**)

```
apt -y install rpm dnf
rpm --root /srv/container/rocky-linux8 --initdb
```

- Das Basis-Paket für Rocky-Linux laden (Versions-Nummern in den Dateinamen muessen ggf. angepasst werden)

```
wget https://ftp.plusline.net/rockylinux/8/BaseOS/x86_64/os/Packages/r/rocky-release-8.5-3.el8.noarch.rpm
wget https://ftp.plusline.net/rockylinux/8/BaseOS/x86_64/os/Packages/r/rocky-repos-8.5-3.el8.noarch.rpm
rpm --nodeps --root /srv/container/rocky-linux8 -ivh rocky-release-8.5-3.el8.noarch.rpm rocky-repos-8.5-3.el8.noarch.rpm
```

- Das Rocky-Linux Basis-System installieren

```
dnf -y --nogpg --releasever=8 --installroot=/srv/container/rocky-linux8 \
  install systemd passwd dnf procps-ng iproute tmux rocky-gpg-keys
echo 8 > /srv/container/rocky-linux8/etc/dnf/vars/releasever
```

- Eine Shell im Container starten

```
systemd-nspawn -D  /srv/container/rocky-linux8
```

- Root-Passwort setzen und neuen Benutzer anlegen

```
-bash-4.2$ passwd
-bash-4.2$ adduser nutzerXX
-bash-4.2$ passwd nutzerXX
```

- Shell im Container verlassen

```
-bash-4.2# exit
```

- Container booten

```
systemd-nspawn -bD  /srv/container/rocky-linux8
```

- Weitere Software im Container installieren (hier den Apache Webserver)

```
dnf install httpd
```

- Anwendung im Rocky Linux Container starten (prüfen das kein andere Prozess auf dem Container-Host die Ports 80 und/oder 443 belegt)

```
systemctl enable --now httpd
```

- Der Apache Webserver innerhalb des Rocky Linux Containers sollte nun vom Firefox des Debian unter http://localhost [http://localhost] erreichbar sein.
- Container stoppen (im Container eingeben)

```
conainter# poweroff
```

- Container mit privatem Netzwerk-Namespace starten:

```
systemd-nspawn -bD  /srv/container/rocky-linux8 --private-network
```

Container mit eigener (virtuellen) Netzwerkkarte (neue MAC-Adresse). `enp1s0f1` ist der Name der

physischen Netzwerkschnittstelle des Container–Hosts.

```
systemd-nspawn --network-macvlan=enp1s0f1 -M rocky01 \
  -bD /srv/container/rocky-linux8
container# ip links
container# dhclient mv-enp1s0f1
container# ip address show
container# ping 1.1.1.1
```

- Systemd–Befehle um einen Container vom Host aus zu kontrollieren

```
machinectl list
machinectl status centos
machinectl terminate centos
machinectl kill centos
```

- Per **nsenter** mit dem Container verbinden (es wird eine Prozess–ID eines Container–Prozesses benötigt!)

```
nsenter -m -u -i -n -p -t <pid-im-container> /bin/bash
```

CHAPTER 4. RESSOURCEN

- Buch: Christof Paar, Jan Pelzl — Understanding Cryptography ISBN: 3642446493
- Buch: Klaus Schmeh — Kryptografie ISBN: 3864900158
- Buch: Niels Ferguson, Bruce Schneier, Tadayoshi Kohno — Cryptography Engineering ISBN: 0470474246
- Buch: Malcolm Harkins — Managing Risk and Information Security ISBN: 1430251131
- Buch: Ross Anderson — Security Engineering (PDF: http://www.cl.cam.ac.uk/~rja14/book.html)
- Buch: Thinking Security: Stopping Next Year's Hackers ISBN-10: 0134277546/ISBN-13: 978-0134277547
- Buch: Gerhard Lienemann: TCP/IP – Grundlagen und Praxis – Protokolle, Routing, Dienste, Sicherheit. 3. Auflage (Gute Übersicht der Basis-TCP/IP-Protokolle, Kapitel zu IT-Sicherheit leider veraltet, nur Diskussion von IKEv1 im IPsec Kapitel) https://www.bookzilla.de/shop/article/50164751/gerhard_lienemann_tcp_ip_grundlagen_und_praxis.html [https://www.bookzilla.de/shop/article/50164751/gerhard_lienemann_tcp_ip_grundlagen_und_praxis.html]
- Buch: Kevin Fall, W. Stevens: TCP/IP Illustrated – The Protocols, Volume 1. 2nd edition. https://www.bookzilla.de/shop/article/14768831/kevin_fall_w_stevens_tcp_ip_illustrated.html [https://www.bookzilla.de/shop/article/14768831/kevin_fall_w_stevens_tcp_ip_illustrated.html]
- Buch: Fehlersuche bei IPsec VPN mit IKEv2 – Mathias Weidner (05.12.2020) https://leanpub.com/fehlersuchebeiikev2ipsecvpn [https://leanpub.com/fehlersuchebeiikev2ipsecvpn]
- Video: Opportunistic Encryption Using IPsec by Paul Wouters, Libreswan IPsec VPN Project https://www.youtube.com/watch?v=Me_rl6N1m7c [https://www.youtube.com/watch?v=Me_rl6N1m7c]
- Buch: NIST Special Publication 800-77 Revision 1: Guide to IPsec VPNs https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-77r1.pdf [https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-77r1.pdf]
- Buch/Webseite: BSI TR-02102 Kryptographische Verfahren: Empfehlungen und Schlüssellängen (Jan 2025) https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr02102/tr02102_node.html [https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr02102/tr02102_node.html]

4.1. Internet Standards

- RFC 7296: Internet Key Exchange Protocol Version 2 (IKEv2)
- RFC 7427: Signature Authentication in the Internet Key Exchange Version 2 (IKEv2)
- RFC 7670: Generic Raw Public-Key Support for IKEv2
- RFC 8247: Algorithm Implementation Requirements and Usage Guidance for the Internet Key Exchange Protocol Version 2 (IKEv2)
- RFC 8784: Mixing Preshared Keys in the Internet Key Exchange Protocol Version 2 (IKEv2) for Post-quantum Security
- RFC 9370: Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2)
- RFC 9395: Deprecation of the Internet Key Exchange Version 1 (IKEv1) Protocol and Obsoleted

Algorithms