Security Lab – Linux Firewall mit nftables

VMware

Dieses Lab müssen Sie mit dem **Ubuntu-Image** durchführen.

1 Einleitung

In diesem Praktikum werden Sie eine Firewall unter Linux mit nftables konfigurieren.

Im ersten Teil werden Sie zuerst die Grundlagen von nftables und nmap kennenlernen. Einiges davon haben Sie bereits in der Vorlesung erfahren; in diesem Sinne dient dieser Teil auch als Repetition und Vertiefung. Lesen Sie diesen Teil durch, um sich die Grundlagen für den praktischen zweiten Teil anzueignen.

Im zweiten Teil werden Sie User Mode Linux (UML) verwenden, um eine virtuelle, praxisnahe Umgebung zu simulieren, in der Sie nftables und nmap anwenden können.

2 Grundlagen

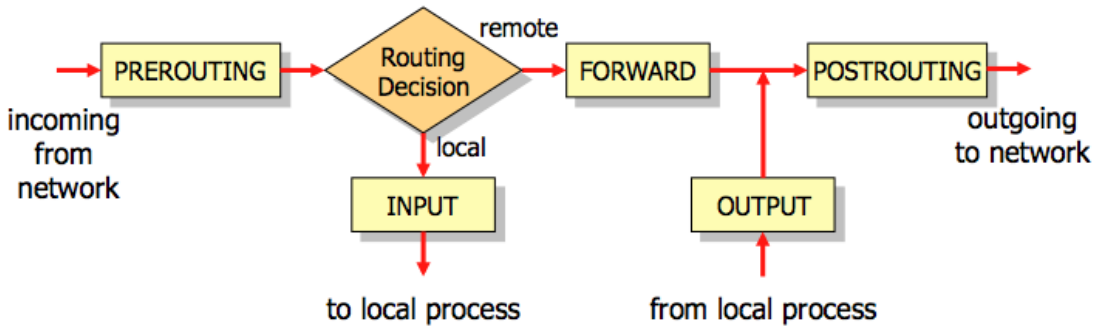
2.1 nftables

nftables ist der Name des Frameworks im Linux-Kern, das zur Paketklassifikation angewendet wird. Es ersetzt das seit langem in Linux vorhandene Framework iptables. Obwohl nftables ein sehr allgemeines Framework ist, wird es meistens dazu verwendet werden, Firewalls zu bauen. Wir werden daher in diesem Praktikum hauptsächlich die Teile von nftables anschauen, die für diese Aufgabe geeignet sind. Mit anderen Worten: Dies ist keine vollständige Einführung in nftables.

nftables verwendet die seit langem im Linux-Kern etablierten *hooks*, das bezeichnet die Möglichkeit, bei bestimmten Stellen in der Paketverarbeitung eigenen Code einzuhängen. Diese sind:

- **prerouting**: Das Paket kommt gerade frisch von der Netzwerkkarte und es ist noch keine Entscheidung getroffen worden, wohin dieses Paket letztlich gesendet werden soll
- **input/output**: Das Paket soll zu einem Prozess auf diesem Rechner gelangen (input) oder wurde von einem Prozess auf diesem Rechner erzeugt (output).
- **forward**: Das Paket ist weder für diesen Rechner bestimmt, noch geht es von diesem Rechner aus. Stattdessen kommt es auf einem Netzwerkinterface herein und geht auf einem anderen wieder heraus.
- **postrouting**: Das Paket wird diesen Rechner gleich auf einem Netztwerkinterface verlassen.

Die folgende Abbildung illustriert die verschiedenen Hooks und wie sie auf welche Pakete angewendet werden. Die pre- und postrouting-Hooks werden wir vorerst ignorieren.



In nftables ist die Paketklassifikation aufgeteilt auf *tables* (Tabellen), *chains* (Ketten) und *rules* (Regeln). Fangen wir zunächst mit den Regeln an.

Eine Regel (*rule*) hat zwei Teile. Der erste Teil einer Regel sagt, auf welche Pakete diese Regel anwendbar ist (Klassifikation, *classification*) und der zweite Teil sagt, was mit einem Paket geschehen soll, auf das der erste Teil anwendbar ist (Aktion, *action*). Es gibt hauptsächlich drei Aktionen:

- **accept**: Das Paket wird akzeptiert und weiterverarbeitet
- **drop**: Die Verarbeitung des Pakets wird gestoppt und das Paket wird verworfen, ohne dem Absender eine Fehlermeldung zuzustellen
- **reject**: Die Verarbeitung des Pakets wird gestoppt und das Paket wird verworfen, dem Absender wird aber eine Fehlermeldung zugestellt

Die herrschende Empfehlung ist, Pakete kommentarlos zu entfernen (`drop`), aber es gibt auch durchaus abweichende Meinungen, die sagen, dass man damit auch die Arbeit von legitimen Systemverwaltern erschwert, der dann bei bestimmten Operationen (ping, nmap, …) immer erst auf das Ablaufen eines Timeouts warten muss, bevor er sieht, dass eine Aktion fehlgeschlagen ist. Diese Personen empfehlen den Einsatz von `reject` statt `drop`. Die Existenz einer Firewall lässt sich durch `drop` auch nicht verheimlichen.

Möchte man beispielsweise alle Pakete verwerfen, die als IPv4-Zieladresse 8.8.8.8 haben, dann könnte eine Regel so aussehen:

```
ip daddr 8.8.8.8 drop
```

Eine Zusatzfunktion in nftables ist, dass man sowohl die Pakete als auch die Anzahl der mit ihnen übertragenen Bytes zählen kann:

```
ip daddr 8.8.8.8 counter drop
```

Dabei wird die Regel von links nach rechts ausgewertet. Würde man also die Regel so formulieren:

```
counter ip daddr 8.8.8.8 drop
```

Dann würde *jedes* Paket gezählt, das in diese Regel eintritt, unabhängig davon, ob die Kriterien auf es zutreffen oder nicht. Dem Sender kann man noch eine ICMP-Fehlermeldung zustellen, etwa mit:

```
ip daddr 8.8.8.8 reject
```

Diese Regel führt dann im Anwendungsfall zu einer ICMP oder ICMPv6-Meldung vom Typ «port unreachable». Will man eine andere ICMP-Meldung schicken, kann man das konfigurieren, z.B.:

```
ip daddr 8.8.8.8 reject with icmp type host-unreachable
```

Pakete sind in Ketten (*chains*) organisiert, die jeweils mit einem bestimmten Hook assoziiert sind. Innerhalb einer Chain werden die Regeln von der ersten bis zur letzten Regel abgearbeitet, solange bis eine gefunden wird, auf welche die Classification zutrifft. In diesem Fall wird die Aktion der Regel ausgewertet. Die Bearbeitung der Chain wird dann abgebrochen.

Für den Fall, dass keine Regel der Chain auf ein gegebenes Paket zutrifft, hat eine Chain noch eine *policy*. Das ist die Aktion, die zutrifft, falls keine Regel explizit angewendet werden kann. Die beiden möglichen Policies sind `accept` und `drop`.

Chains haben ausserdem noch einen Typ (*type*). Wir nehmen hier zunächst den Typ `filter`, der dazu dient, Pakete zu filtern. Später werden wir noch den Typ `nat` kennenlernen, der Network Address Translation unterstützt.

Zuletzt besitzt jede Chain noch eine Priorität (*priority*). Werden Pakete akzeptiert (accept) und existiert für denselben Hook eine weitere Chain mit einer späteren (höheren) Priorität, wird das Paket durch diese später priorisierte Chain geschickt, also *erneut* ausgewertet. Pakete werden also solange ausgewertet, wie sie akzeptiert werden und es später priorisierte Chains gibt.

```
chain ssh {
```

```
  type filter hook input priority 0; policy drop;
  # ssh packet accepted
  tcp dport ssh count accept
}

# this chain is evaluated last due to priority
chain myinput {
  type filter hook input priority 1; policy drop;
  # the same ssh packet is dropped here by means of default policy
}
```

Hier haben wir beispielsweise zwei Chains, `ssh` und `myinput`, die gemäss ihren Prioritäten so angeordnet sind, dass `ssh` vor `myinput` bearbeitet wird. Ein Paket, das an den `ssh`-Port gesendet wird, landet zunächst in der `ssh`-Chain, wird dort gezählt und akzeptiert. Da es jetzt noch die `myinput`-Chain im selben Hook aber mit späterer Priorität gibt, wird diese auch noch durchlaufen und dort wird das Paket nun abgelehnt. Hätte die `myinput`-Chain beispielsweise die Priorität -1 gehabt, wäre das Paket zwar auch abgelehnt worden, aber ohne es zu zählen, denn die `ssh`-Chain wäre gar nicht durchlaufen worden.

Die vorletzte Hierarchieebene sind Tabellen (*tables*), in denen die Chains organisiert sind. Aus unserer Sicht dienen Tables hauptsächlich dazu, Chains und Rules zusammenzubringen, die für einen bestimmten Typ von Paket geeignet sind. Diese Pakettypen heissen im Sprachgebrauch *address families*. Es gibt für nftables sechs verschiedene Families, von denen wir aber nur drei betrachten:

- **ip**: Nur IPv4-Pakete haben diese Family
- **ip6**: Nur IPv6-Pakete haben diese Family
- **inet**: Sowohl IPv4 als auch IPv6-Pakete haben diese Family

Hier ist eine Table namens `myfilter`, die die oben angeführten Chains `ssh` und `myinput` enthält und die Pakete sowohl für IPv4 als auch für IPv6 akzeptiert:

```
table inet myfilter {
  chain ssh {
    type filter hook input priority 0; policy drop;
    tcp dport ssh count accept
  }
  chain myinput {
    type filter hook input priority 1; policy drop;
  }
}
```

Am Schluss gibt es noch den Regelsatz (*ruleset*), der alle Tables zusammenfasst.

Zusammenfassend:

- **Ruleset**: Enthält alle Tables
- **Tables**: Enthalten Chains und sind für eine bestimmte Address Family zuständig
- **Chains**: Enthalten Rules, sind einem bestimmten Hook zugeordnet und haben eine Priorität und eine Policy.
- **Rules**: Enthalten eine Klassifikation und eine Aktion. Die Klassifikation sagt, auf welche Pakete die Regel zutrifft und die Aktion sagt, was mit dem Paket innerhalb dieser Chain geschehen soll.

2.2 `nft` Command-Line Tool

`nft` ist das Command-Line Tool, um die Firewall-Regeln zu konfigurieren. Im Folgenden geben wir eine Übersicht über die für dieses Praktikum relevanten Optionen des Tools. Prinzipiell wird `nft` wie folgt verwendet:

```
nft [options] operation family table [chain [rule]]
```

Dabei bezeichnet *operation* was gemacht werden soll, also etwas hinzufügen (`add`), löschen (`delete`), auflsten (`list`) oder Ähnliches, *family* die Address Family, *table* die Table und *chain* die Chain.

Es gibt eigentlich nur eine sinnvolle Operationen auf dem Ruleset und das ist `list`, wobei die Option `-a` angibt, dass Objekte mit ihrem sogenannten *handle* ausgegeben werden sollen. Das Handle ist eine Zahl, die das Objekt eindeutig identifiziert. Das ist später nützlich, weil man dann beispielsweise sagen kann "lösche Regel 4". Die Zahl 4 wäre dann das Handle der Regel, die man löschen will.

```
nft [-a] list ruleset
```

Je weiter man die Hierarchiestufen in Richtung Rules durchschreitet, desto mehr muss man von den höheren Hierarchiestufen angeben: Bei Operationen auf Tables die Address Family, bei Chains die Address Family und Table und bei Operationen auf Rules die Address Family, die Table und die Chain.

Tables kann man hinzufügen, auflisten, leeren und löschen:

- **add**: Fügt eine Table zum Ruleset hinzu.
- **delete**: Löscht eine Table mitsamt ihren Chains und Regeln aus dem Ruleset.
- **list**: Listet die Chains und Rules einer Table auf. Bei Verwendung der Option `-n` wird nicht versucht, IP-Adressen in Namen aufzulösen. Bei Verwendung der Option `-nn` wird nicht versucht, Dienstnummern in Namen aufzulösen. Beispielsweise bleibt Googles öffentlicher DNS-Server `google-public-dns-a.google.com` einfach 8.8.8.8 und `ssh` bleibt einfach 22. Das möchte man manchmal haben, weil die Auflösung von Nummern in Namen manchmal selbst Netzwerkverkehr verursacht.
- **flush**: Löscht alle Regeln in allen Chains in dieser Table, aber nicht die Chains selbst.

```
nft add table inet myinput  # Fügt Table myinput hinzu
nft list table inet myinput # Listet Chains/Rules von myinput auf
```

Chains kann man erzeugen, löschen, auflisten und leeren:

- **add**: Fügt eine Chain zur genannten Table hinzu.
- **delete**: Löscht eine Chain mitsamt ihren Regeln aus der Table.
- **list**: Listet die Regeln einer Chain. Bei Verwendung der Option `-n` wird nicht versucht, IP-Adressen in Namen aufzulösen. Bei Verwendung der Option `-nn` wird nicht versucht, Dienstnummern in Namen aufzulösen.
- **flush**: Löscht alle Regeln aus einer Chain, aber nicht die Chain selbst.

```
nft add chain inet myinput ssh \           # Fügt Chain ssh
  { type filter hook input priority 0 \; \ # zur Table myinput
    policy drop \; }          # mit Priorität 0 und Policy drop
```

Vorsicht bei der Eingabe dieser Regeln über die Kommandozeile, da muss man das Semikolon noch durch einen Backslash escapen, wie hier angegeben. Wird nftables über Konfigurationsdateien konfiguiert (siehe unten), ist das nicht nur nicht nötig, sondern sogar falsch..

Regeln kann man hinzufügen, ersetzen oder löschen

- **add**: Fügt die Regel am Schluss der Chain/Table an
- **insert**: Fügt die Regel am Anfang der Chain/Table an
- **replace:** Ersetzt die entsprechende Regel
- **delete**: Löscht die entsprechende Regel

```
# Fügt Regel der Table myfilter, Chain ssh hinzu, nach der
# alle Pakete, die (a) TCP-Pakete sind und (b) als Zielport ssh
# haben (1) gezählt und (2) akzeptiert werden
nft add rule inet myfilter ssh tcp dport ssh counter accept
# Löscht die Regel mit dem Handle 4 aus Table myfilter, Chain ssh
nft delete rule inet myfilter ssh handle 4
# Fügt Regel nach Regel mit Handle 8 ein
nft add rule inet myfilter ssh position 8 tcp dport ssh accept
# Ersetzt Regel mit Handle 8
nft replace rule inet myfilter ssh handle 8 tcp dport ssh counter
```

Wir werden nftables auch im Scripting-Modus verwenden. Dabei schreibt man eine ausführbare Datei ähnlich einem Shellskript, gibt aber `/usr/sbin/nft -f` als Interpreter an. Dort kann man unter anderem Variablen definieren. Wir nutzen das, um Schreibfehler bei der wiederholten Verwendung von Netzwerkinterfaces und Netzwerkbezeichnungen zu vermeiden. Im vorbereiteten firewall-Skript finden sich bereits Variablen für die verschiedenen Interfaces, Host- und Netzwerkadressen. Wir empfehlen, diese ausgiebig zu nutzen.

Besonders im IPv6-Bereich ist das Debuggen von Regeln oft knifflig. Dazu kann man in einer Chain

```
nft add rule inet myfilter ssh meta nftrace set 1
```

angeben, wodurch Tracing eingeschaltet wird. Den Trace selber kann man dann auf der Kommandozeile mit

```
nft monitor trace
```

anschauen. Im Firewall-Skript, das Sie entwickeln sollen, ist das schon vorgesehen, aber auskommentiert. Bei Bedarf kommentieren Sie das wieder ein.

Für weitere Parameter verweisen wir auf die man-Page von `nft`. Diese ist im nicht auf den UML-Instanzen verfügbar, lässt sich aber per «man nft» einfach googlen.

2.3 `nmap` Command-Line Tool

`nmap` wurde entwickelt, um Hosts hinsichtlich offenen Ports zu scannen. `nmap` unterstützt eine Vielzahl verschiedener Scanning-Techniken. Ebenso bietet nmap eine grosse Zahl von zusätzlichen Möglichkeiten wie z.B. das Erkennen von Betriebssystemen mittels TCP/IP-Fingerprinting.

Ein Scanvorgang startet man mit folgendem Befehl:

```
nmap options {host|net}
```

Einige Optionen sind im Folgenden angegeben:

`-6` Verwendet ausschliesslich IPv6.

`-4` Verwendet ausschliesslich IPv4.

`-v` Verbose-Modus - Ermöglicht eine erweiterte Ausgabe von Informationen.

`-sT` TCP connect() scan - Die Basisform des TCP-Scanning. Benötigt keine speziellen User-Privilegien.

`-Pn` Per Default wird nmap immer mit einem Ping (ICMP echo-request/reply) testen, ob ein Host "up and running" ist, bevor er gescannt wird. Diese Option unterdrückt den Ping und scannt den Host direkt. Die Option ist dann nötig, wenn der Zielhost oder eine Firewall davor die Pings blockiert, worauf der Host von nmap nicht gescannt würde.

`-p port(bereich)` Spezifiziert, welche Ports gescannt werden sollen. Entweder einzelne Ports (z.B. `-p 80`) oder ein Portbereich (z.B. `-p 20-200`).

Für weitere Optionen verweisen wir auf die man-Page von nmap.

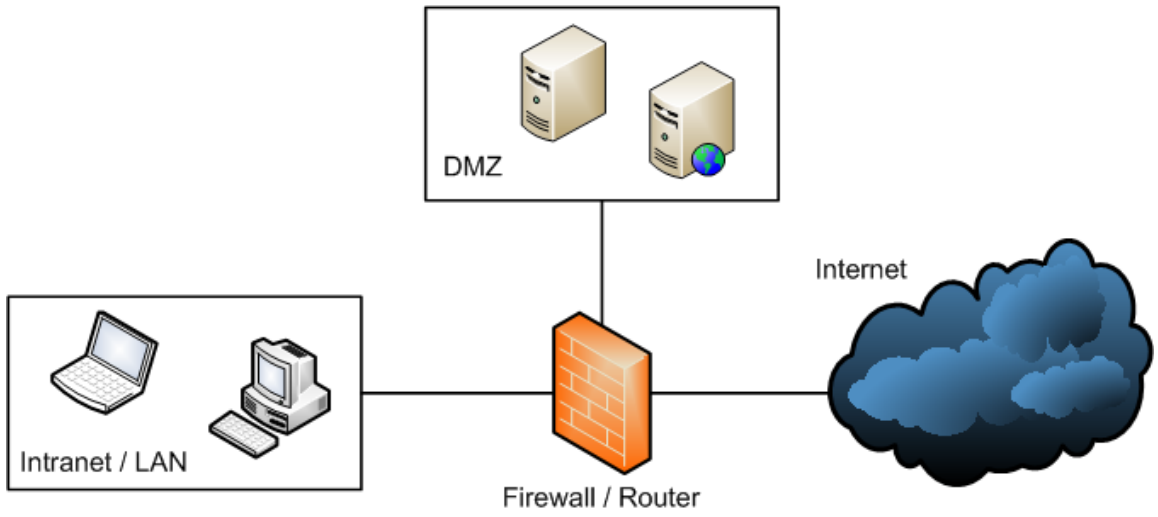
2.4 `nmap` Beispiele

`nmap -v -Pn -sT ziel.host.com`
Scannt alle well-known TCP-Ports des Hosts ziel.host.com.

`nmap -v -Pn -sT -p1-30 192.168.55.66`
Scannt die Ports 1-30 des Hosts mit der IP-Adresse 192.168.55.66.

3 `nftables` und `nmap` anwenden

Im zweiten Teil des Praktikums werden wir das in der nachfolgenden Abbildung aufgezeichnete Netzwerk verwenden.



Wie in der Abbildung ersichtlich, möchten wir mit dieser Netzarchitektur eine vereinfachte Unternehmensumgebung mit folgenden Komponenten realisieren:

- Ein interner Bereich, welcher das unternehmensinterne LAN darstellt.
- Ein externer Bereich, welcher das öffentliche Internet darstellt.
- Eine DMZ, welche gewisse Dienste nach innen und aussen anbietet.
- Eine Firewall, welche die Zugriffe der verschiedenen Netze steuert und überwacht.

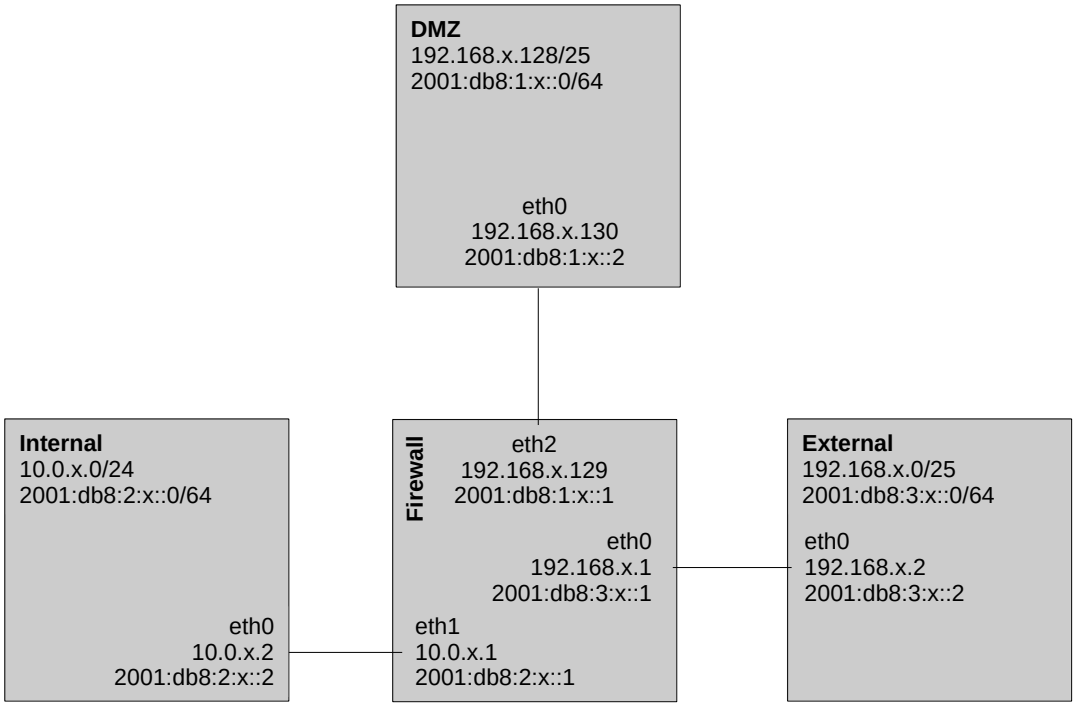
In der Realität würde man dieses Netzwerk z.B. mit drei physischen Netzen und einer Firewall mit drei Netzwerkinterfaces implementieren. Weil wir im Praktikumsraum eine solche Umgebung nur schlecht aufbauen können, werden wir das gesamte Netzwerk auf einem Host realisieren und verwenden dazu User Mode Linux.

3.1 User Mode Linux (UML)

User Mode Linux (mit UML abgekürzt und nicht zu verwechseln mit der Unified Modeling Language) ist eine Modifikation des Linux-Kernels, die es erlaubt, mehrere komplette Linux-Kernel als virtuelle Maschinen (bezeichnet als Gastsysteme) innerhalb eines laufenden Linux-Systems (bezeichnet als Hostsystem) zu betreiben.

Vieles wird durch die Verwendung von UML möglich. Zum Beispiel können Netzwerkdienste in einer UML-Instanz komplett isoliert vom Hostsystem ablaufen. UML kann auch eingesetzt werden, um neue Software einschliesslich des Linux-Kernels zu debuggen oder zu testen, ohne das Hostsystem zu beeinflussen. Für weitere Informationen über UML verweisen wir auf *http://user-mode-linux.sourceforge.net*.

In unserem Fall verwenden wir UML um die gesamte oben dargestellte Netzwerkumgebung auf einem Hostsystem zu betreiben. Als Hostsystem verwenden wir das Ubuntu-System basierend auf dem Ubuntu-Image, das ja bereits selbst als virtuelles System auf Ihrem Laptop läuft. Um es möglich einfach zu halten simulieren wir die drei Netze und insgesamt vier Hosts: die Firewall und zusätzlich je einen Host in jedem der drei Netze. Einer der Hosts steht dabei für einen Host im Intranet, einer für einen Server in der DMZ und einer für einen Host im Internet. Dies genügt, um sämtliche Kommunikation zwischen den verschiedenen Netzen zu ermöglichen und um die Firewall zu konfigurieren und auszutesten. Für die drei Netze verwenden wir dabei private IP-Adressen. Die komplette Umgebung ist in der nachfolgenden Abbildung dargestellt:



Das x in den IP-Adressen steht für Ihre Gruppennummer; es werden also alle Gruppen eigene private Adressbereiche verwenden. Bei den IPv4-Adressen ist x dezimal, bei den IPv6-Adressen jedoch hexadezimal. Die Firewall hat also auf eth2 bei Gruppe 12 die IPv4-Adresse 192.168.12.129 und die IPv6-Adresse 2001:db8:1:c::1. Dies wäre nicht nötig, weil die Netze der einzelnen Gruppen sowieso strikt voneinander separiert sind; es wird jedoch für die Praktikumsbewertung verwendet.

3.2 UML in Betrieb nehmen

Starten Sie ein Terminal und wechseln Sie ins Verzeichnis */securitylab*. Entzippen Sie das File *firewall_nftables.zip*:

```
unzip firewall_nftables.zip
```

Dies erzeugt ein Verzeichnis *firewall_nftables*. Dieses können Sie nach Beendigung des Labs auch wieder entfernen, falls Sie Probleme mit der Grösse des Image haben.

Starten Sie nun drei weitere Terminals und wechseln Sie in allen vier Terminals ins Verzeichnis */securitylab/firewall_nftables*. Starten Sie in einem Terminal die virtuelle Netzwerkkonfiguration:

```
./prepare_uml gruppennummer
./net_config
```

Dabei entspricht `gruppennummer` Ihrer Gruppennummer. Gruppe 6 führt also `./prepare_uml 6` aus.

Starten Sie anschliessend die folgenden vier Befehle in je einem Terminal:

```
./run_dmz
./run_internal
./run_external
./run_firewall
```

In jedem Terminal wird nun eine UML-Instanz gestartet. In jedem Fenster wird es unter Anderem eine Zeile geben «`virtual console 1 started on /dev/pts/12`» oder ähnlich. Merken Sie sich diesen Device-Namen.

Starten Sie nun ein weiteres Terminal. In diesem Terminal starten Sie nun `screen`, um mit den verschiedenen Linux-Systemen in Kontakt zu treten. Hat beispielsweise das internal-UML u.a. auf `/dev/pts/5` ein Terminal, dann starten Sie

```
./screen internal 5
```

Hier sollten Sie dann den login-Prompt des «internal»-Systems sehen. Mit «Control-r c» (im Weiteren «`C-r c`» geschrieben) erzeugen Sie ein neues Fenster. Starten Sie dort ebenfalls screen, z.B. für die DMZ. Verwenden Sie das Terminal, das sie Sich beim Start des DMZ-Systems gemerkt haben. Ist es beispielsweise `/dev/pts/12`, dann tippen Sie:

```
./screen dmz 12
```

Erzeugen Sie nun mit `C-r c` ein neues Fenster und starten Sie darin screen auf den «external»-Rechner und ein weiteres Fenster mit screen auf «firewall». Sie haben nun vier screen-Fenster erzeugt, zwischen denen Sie mit `C-r SPC` (Control-R space) weiterschalten können. Mit `C-r p` kommen Sie ins vorherige (*previous*) und mit `C-r n` ins nächste (*next*) Fenster. Wenn Sie die UML-Instanz in Ihrem screen-Fenster herunterfahren, wird das screen-Fenster noch nicht geschlossen. Um dieses Fenster zu beenden müssen Sie die Shell, die nach Herunterfahren der UML-Instanz zu sehen ist, ebenfalls beenden. Beim Herunterfahren der letzten UML-Instanz wird screen insgesamt beendet.

Hinweis: Auch mit UML werden völlig normale Kernel gebootet und Filesysteme gemounted. Gehen Sie also mit den UML-Instanzen genau so «sorgfältig» um wie mit einem «richtigen» Linux-System. Insbesondere sollten Sie die Instanzen nicht einfach durch Schliessen des Terminal-Fensters terminieren, sondern Sie sollten sauber heruntergefahren werden (mit `halt`).

Hinweis: Möglicherweise stellen Sie irgendwann fest, dass Sie sich die UML-Instanz zerschossen haben und Sie zum ursprünglichen Zustand zurückkehren wollen. Wenn *x* der Name Ihrer UML-Instanz ist (also «internal», «external», «dmz» oder «firewall»), dann können Sie nach dem Herunterfahren der UML-Instanz im Verzeichnis *firewall_nftables* die Datei `cow_`*x* löschen. Das in diesem Lab entwickelte Firewall-Skript ist davon *nicht* betroffen.

Hinweis: Wenn Sie mit den verschiedenen Windows von screen nicht zurecht kommen, können Sie auch einfach vier weitere Terminals auf dem Ubuntu-Image starten und auf jedem Terminal screen einzeln aufrufen.

Melden Sie sich auf allen vier UML-Instanzen als *root* an (Passwort `root`). Prüfen Sie dann mit `ip addr`, ob alle Netzwerkinterfaces die richtigen Adressen erhalten haben (gemäss der Abbildung weiter oben) und testen Sie, ob Sie alle Hosts von jedem Host aus anpingen können. Verwenden Sie sowohl IPv4 als auch IPv6.

Hinweis: Auf den UML-Hosts sind die Interfaces der verschiedenen Rechner mit symbolischen Namen ausgestattet, was es einem erspart, die IP-Adressen anzugeben. Das ist insbesondere bei IPv6 von Vorteil. Die Interfaces sind:

Interface-Name	Adressen
firewall-int	10.0.x.1, 2001:db8:2:x:1
int	10.0.x.2, 2001:db8:2:x::2
firewall-dmz	192.168.x.129, 2001:db8:1:x::1
dmz	192.168.x.130, 2001:db8:1:x::2
firewall-ext	192.168.x.1, 2001:db8:3:x::1
ext	192.168.x.2, 2001:db8:3:x::2

Will man also das IPv6-Interface des internal-Hosts anpingen, tippt man `ping -6 int`.

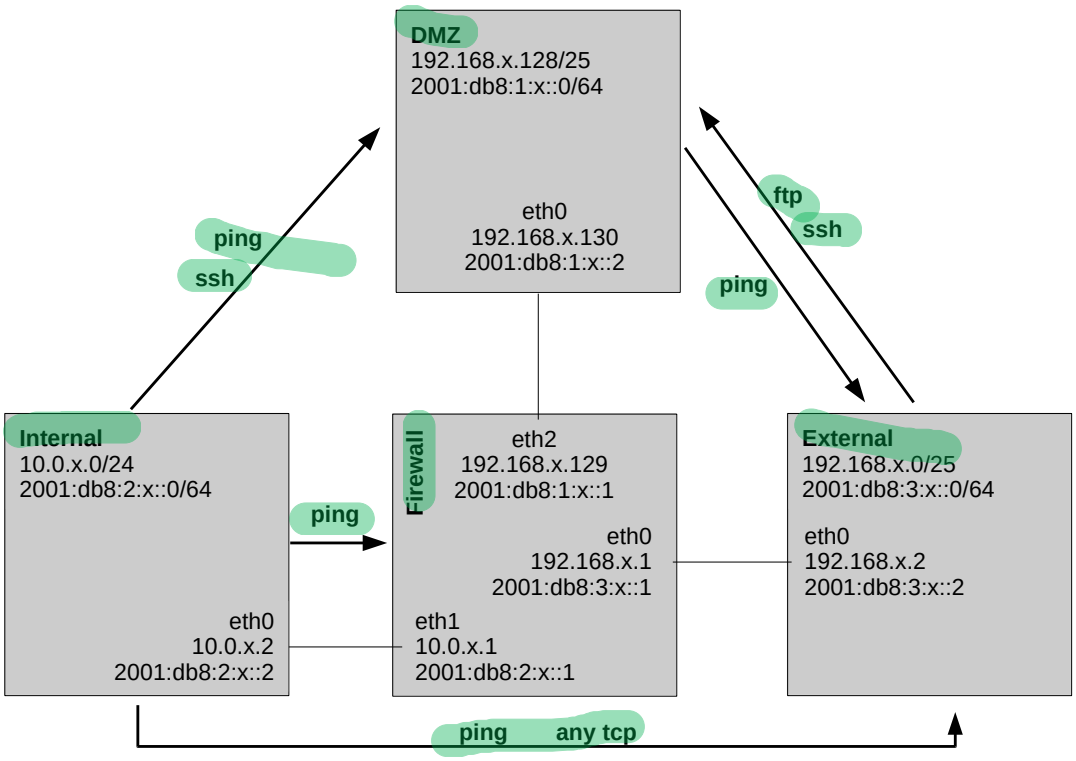
Prüfen Sie auch, ob auf dem DMZ-Host ein ssh- und ftp-Server laufen (am besten mit `netstat -l`) und überprüfen, ob auf den Ports 21 (ftp) und 22 (ssh) TCP-Server am „listenen“ sind). Prüfen Sie analog, ob auf dem externen Host ein ssh-Server läuft. Wenn all dies funktioniert, sind Sie für die Aufgaben bereit.

Hinweis: Der `netstat`-Befehl läuft aus irgendeinem Grund erst dann korrekt, wenn auf der jeweiligen UML-Instanz der Zufallszahlengenerator korrekt initialisiert ist. Dann steht im Konsolenfenster “`random: crng init done`”. Falls das bei Ihnen noch nicht der Fall sein sollte, erhalten Sie beim Aufruf von `netstat` Fehlermeldungen. Schauen Sie in dem Fall mit `ps aux`, ob ssh- oder ftp-Prozesse laufen.

3.3 Aufgaben

Sobald mehrere Firewall-Regeln mit nftables konfiguriert werden, ist es sinnvoll, ein Script zu verwenden, in welches die iptables-Befehle eingetragen werden. Auf dem Firewall-Host finden Sie unter */root* ein Skript *firewall*, das bereits vorbereitet wurde und das Sie im Laufe der folgenden Aufgaben erweitern sollen. Verwenden Sie dazu am besten den Editor *vi*. Das Skript selbst führen Sie mit *./firewall* aus. Nach jeder Änderung müssen Sie das Skript neu ausführen. Wird irgendeine Meldung ausgegeben, so haben Sie in einer Regel einen syntaktischen Fehler gemacht, den Sie zuerst korrigieren müssen.

Die nachfolgende Abbildung zeigt die Konfiguration, die wir am Ende erreichen wollen. Ein Pfeil bedeutet, dass der entsprechende Traffic zugelassen ist und die Richtung des Pfeils zeigt, in welche Richtung ein Verbindungsaufbau bzw. ein Ping möglich sein soll. Alles, was nicht in der Abbildung eingezeichnet ist, soll blockiert werden. Jeglicher erlaubter Traffic soll sowohl mit IPv4 als auch mit IPv6 möglich sein.



Arbeiten Sie zu Beginn mit stateless Firewall Regeln; wir werden später stateful Regeln dazufügen.

Überlegen Sie sich bei allen Aufgaben genau, ob das erhaltene Resultat eines nmap-Scans oder Pings Ihren Erwartungen entspricht! Wenn nein, dann überlegen Sie sich, ob Sie einen Denkfehler gemacht haben oder ob Sie einen Konfigurationsfehler gemacht haben. Denken Sie auch daran, dass Sie die aktuellen nftables-Einträge mit `nftables -a list ruleset` anschauen können.

Teil 1 - Allgemeine Voraussetzungen schaffen

Zum jetzigen Zeitpunkt ist das ganze Netzwerk noch völlig offen, die Firewall lässt also alles durch. Wenn Sie das firewall-Skript ansehen, werden Sie zudem sehen, dass die einzigen bereits vorhandenen nftables-Befehle dazu dienen, alle bestehenden Firewall-Regeln zu entfernen (flush). Dies ist sinnvoll, weil damit die nachfolgenden nftables-Befehle (die Sie bald eintragen werden) immer basierend auf einer nicht konfigurierten (d.h. komplett offenen) Firewall ausgeführt werden.

Hinweis: nftables-Skripte werden immer *atomar* geladen. Das bedeutet, dass der alte Zustand bestehen bleibt, wenn das neu zu ladende Skript einen Fehler enthält. Das ist ungemein praktisch und einer der wesentlichen Unterschiede zum vorherigen Framework «iptables». So kann es beispielsweise nicht passieren, dass durch einen Konfigurationsfehler in der Skriptdatei ein vorher geschlossenes Netzwerk nun völlig geöffnet wird.

Führen Sie im ersten Teil folgende Aufgaben durch:

1. Scannen Sie die TCP-Ports auf dem DMZ-Host vom external-Host aus. Um den Scan zu beschleunigen, sollten Sie den TCP-Portbereich bei allen Scans in diesem Praktikum auf die Ports 1-100 einschränken. Verwenden Sie ebenfalls bei allen Aufgaben die `-Pn`-Option. Scannen Sie IPv4 und IPv6. Wie sieht der Output aus, insbesondere bzgl. den Port(s), die bei diesem Scan als offen, geschlossen und gefiltert gemeldet wurden? Entspricht das Verhalten Ihren Erwartungen? Erklären Sie zudem, was *offen*, *geschlossen* und *gefiltert* im Zusammenhang mit den TCP-Ports ganz generell bedeutet.

   Hinweis: Sollten einige Ports als «closed» angezeigt werden, die eigentlich «open» sein sollten, liegt das vermutlich am noch nicht fertig initialisierten Zufallszahlengenerator auf der DMZ. Bitte warten Sie dann so lange, bis im Fenster der DMZ-UML-Instanz die Meldung «`random: crng init done`» erscheint. Wir können dieses Phänomen zur Zeit nicht erklären.

```
root@external:~# nmap -v -Pn -sT -p1-100 dmz
Starting Nmap 7.70 ( https://nmap.org ) at 2020-04-24 05:07 UTC
Initiating Connect Scan at 05:07
Scanning dmz (192.168.21.130) [100 ports]
Discovered open port 22/tcp on 192.168.21.130
Discovered open port 21/tcp on 192.168.21.130
Discovered open port 7/tcp on 192.168.21.130
Discovered open port 13/tcp on 192.168.21.130
Discovered open port 9/tcp on 192.168.21.130
Completed Connect Scan at 05:07, 0.11s elapsed (100 total ports)
Nmap scan report for dmz (192.168.21.130)
Host is up (0.0065s latency).
Other addresses for dmz (not scanned): 2001:db8:1:15::2
Not shown: 95 closed ports
PORT   STATE SERVICE
7/tcp  open  echo
9/tcp  open  discard
13/tcp open  daytime
21/tcp open  ftp
22/tcp open  ssh

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 1.07 seconds
```

Befehl: nmap -v -PN -sT -p1-100 dmz

offen: Zugriff auf diesen Port wird zugelassen

geschlossen: Verbindung zu diesem Port wird durch die Firewall blockiert

gefiltert: Die Firewall gibt keine Antwort an den Sender zurück und es läuft somit in ein Timeout (drop-Action)

2. Erweitern Sie das firewall-Skript, indem Sie die Policies der myinput und myoutput Chains auf drop setzen und führen Sie es aus. Können Sie die Firewall vom external-Host aus anpingen? Verändert sich der nmap-Scan des DMZ-Hosts im Vergleich zur obigen Aufgabe? Erklären Sie das Verhalten.

```
root@external:~# ping firewall-ext
PING firewall-ext(firewall-ext (2001:db8:3:15::1)) 56 data bytes
From ext (2001:db8:3:15::2): icmp_seq=9 Destination unreachable: Address
 unreachable
From ext (2001:db8:3:15::2): icmp_seq=10 Destination unreachable: Addres
s unreachable
From ext (2001:db8:3:15::2): icmp_seq=11 Destination unreachable: Addres
s unreachable
From ext (2001:db8:3:15::2): icmp_seq=12 Destination unreachable: Addres
s unreachable
From ext (2001:db8:3:15::2): icmp_seq=13 Destination unreachable: Addres
s unreachable
From ext (2001:db8:3:15::2): icmp_seq=14 Destination unreachable: Addres
s unreachable
From ext (2001:db8:3:15::2): icmp_seq=15 Destination unreachable: Addres
s unreachable
From ext (2001:db8:3:15::2): icmp_seq=16 Destination unreachable: Addres
s unreachable
From ext (2001:db8:3:15::2): icmp_seq=17 Destination unreachable: Addres
s unreachable
^C
--- firewall-ext ping statistics ---
20 packets transmitted, 0 received, +9 errors, 100% packet loss, time 79
9ms
pipe 4
```

Das Pingen geht nicht mehr aufgrund der drop-action verwirft es die Pakete und gibt keine Rückmeldung. Entsprechend erscheint beim Ping "unreachable"

Das Verhalten des nmap-Scan ist noch identisch, da myforward noch nicht angepasst wurde

3. Setzen Sie nun auch noch die Policy der myforward Chain auf drop. Wie sieht der nmap-Scan jetzt aus? Erklären Sie wiederum das Verhalten.

```
Initiating Connect Scan at 06:39
Scanning dmz (192.168.21.130) [100 ports]
Discovered open port 22/tcp on 192.168.21.130
Discovered open port 21/tcp on 192.168.21.130
Discovered open port 9/tcp on 192.168.21.130
Discovered open port 13/tcp on 192.168.21.130
Discovered open port 7/tcp on 192.168.21.130
Completed Connect Scan at 06:39, 0.18s elapsed (100 total ports)
Nmap scan report for dmz (192.168.21.130)
Host is up (0.033s latency).
Other addresses for dmz (not scanned): 2001:db8:1:15::2
Not shown: 95 closed ports
PORT   STATE SERVICE
7/tcp  open  echo
9/tcp  open  discard
13/tcp open  daytime
21/tcp open  ftp
22/tcp open  ssh

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.47 seconds
root@external:~# nmap -v -Pn -sT -p1-100 dmz
Starting Nmap 7.70 ( https://nmap.org ) at 2020-04-24 06:42 UTC
Initiating Connect Scan at 06:42
Scanning dmz (192.168.21.130) [100 ports]
Completed Connect Scan at 06:42, 21.07s elapsed (100 total ports)
Nmap scan report for dmz (192.168.21.130)
Host is up.
Other addresses for dmz (not scanned): 2001:db8:1:15::2
All 100 scanned ports on dmz (192.168.21.130) are filtered

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 22.02 seconds
```

Nun werden sämtliche 100 gesetzte Ports auf gefiltert gesetzt. Dies aufgrund der Anpassung von myforward auf drop

Teil 2 – Konnektivität mit IPv6 wiederherstellen

Möglicherweise haben Sie bemerkt, dass Sie über IPv6 den nmap-Scan nicht mehr durchführen können, wenn Sie die Policy der myinput- und myoutput-chains auf drop stellen, obwohl der Scan über IPv4 noch funktioniert. Das scheint nicht logisch zu sein. In diesem Teil finden wir heraus, was da passiert und was man tun muss, um das erwünschte Verhalten wiederherzustellen.

4. Beschreiben Sie zunächst kurz, was passieren muss, damit ein Rechner A *über IPv4* ein Paket an einen Rechner B im selben Ethernet schicken kann. Nehmen Sie der Einfachheit halber an, dass A seit dem Boot noch kein Paket an B geschickt hat. Beachten Sie: Hier geht es *nur* um IPv4!

5. Beschreiben Sie jetzt, wie derselbe Vorgang in IPv6 abläuft.

6. Erklären Sie nun, wieso der nmap-Scan über IPv4 funktioniert, über IPv6 aber manchmal nicht.

7. Die UML-Instanzen sind alle so konfiguriert, dass ihre Default-Gateways fest kodiert sind (siehe dazu die Datei `/etc/network/interfaces`). Das ist aber nicht der Normalfall, denn im Normalfall wird ein IPv6-Host selbst herausfinden, wo sein Router sitzt. Beschreiben Sie kurz, wie das geht.

Teil 3 - Ping auf die Firewall zulassen

In Teil 2 haben Sie die Firewall so konfiguriert, dass alles blockiert wird. In diesem Teil wollen wir erst einmal die erlaubten Pings auf die Firewall zulassen. Lösen Sie dazu die folgende Aufgabe:

8. Erlauben Sie Pings auf die Firewall vom internen Netz und von der DMZ aus. Beachten Sie, dass Sie dazu jeweils ICMP echo-request Meldungen zur Firewall (myinput Chain) zulassen müssen und echo-reply Meldungen in der Gegenrichtung (myoutput Chain). Dies bedeutet, dass jeweils zwei Einträge nötig sind. Prüfen Sie anschliessend, ob Sie sowohl vom internal- als auch vom DMZ-Host die Firewall anpingen können (IPv4 und IPv6). Prüfen Sie ebenfalls, ob dies vom external-Host her nicht geht, um sicherzustellen, dass Sie die Firewall nicht irrtümlicherweise „zuviel“ geöffnet haben.

   Hinweis: Sie benötigen für eine vollständige Konfiguration die Parameter `iifname` (input interface name), `oifname` (output interface name), `ip saddr` (IPv4 source address), `ip daddr` (IPv4 destination address), `ip6 saddr` (IPv6 source address), `ip6 daddr` (IPv6 destination address), `icmp type` und `icmpv6 type`, sowie die ICMP-Nachrichtentypen `echo-request`, `echo-reply`, `nd-neighbor-advert` und `nd-neighbor-solicit`. Wir haben unten noch `nd-router-solicit` und `nd-router-advert` zusätzlich angegeben, wobei das in diesem Fall strenggenomen nicht nötig sein sollte, weil ja, wie oben angegeben, die Rechner ihr Gateway alle kennen. Da das aber nicht der Normalfall ist, haben wir das hier eingebaut. Als Beispiel sind die sechs(!) Einträge für den Ping vom internal-Netz auf die Firewall angegeben; die Einträge für den Ping vom DMZ-Netz zur Firewall müssen Sie analog erstellen:

```
chain myinput {
  chain myinput {
```

}

```
        type filter hook input priority 0; policy drop;
        #meta nftrace set 1
        icmpv6 type { nd-neighbor-advert, nd-neighbor-solicit, nd-router-so-
licit } accept
        iifname $iifc ip saddr $i4nw icmp type echo-request accept
        iifname $iifc ip6 saddr $i6nw icmpv6 type echo-request accept
      }
      chain myoutput {
        type filter hook output priority 0; policy drop;
        #meta nftrace set 1
        icmpv6 type { nd-neighbor-advert, nd-neighbor-solicit, nd-router-ad-
vert } accept
        oifname $iifc ip daddr $i4nw icmp type echo-reply accept
        oifname $iifc ip6 daddr $i6nw icmpv6 type echo-reply accept
      }
```

9. Erklären Sie, wozu die Einträge für `nd-neighbor-advert` und `nd-neighbor-solicit` dienen und warum diese Einträge nicht mit den `echo-request`- und `echo-reply`-Einträgen zusammengefasst sind.

   nd-neighbor-advert: Geht es um die IP Auflösung von IPv4 nach IPv6 und, dass sich die Router sporadisch im Netzwerk melden

   nd-neighbor-solicit: Alle Router im Netz werden aufgefordert alle

   echo-request: Wird explizit pro Netz für jede Anforderung gesetzt

   echo-Reply: wird explizit pro Netz für jede Antwort gesetzt

10. Beachten Sie, dass mit diesen beiden Regeln *alle* Hosts im gesamten Netz 10.0.x.0/24, bzw. 2001:db8:2:x::/64 die Firewall anpingen können, auch wenn nur einer vorhanden ist. Der Vorteil der Angabe des Netzes ist, dass die Firewall nicht erweitert werden müsste, wenn weitere Hosts im Intranet hinzugefügt würden. Sie können aber auch einfach nur den Host (10.0.x.2) angeben. Beachten Sie auch, dass die Regeln immer so genau spezifiziert werden sollen, wie möglich. Geben Sie also nach Möglichkeit immer die Interfaces (`iifname`, `oifname`) *und* die Source- und Destination IP-Adressen (`saddr`, `daddr`) an, denn damit wird IP-Spoofing effektiv unterbunden. Spezifizieren Sie ebenfalls den Protokolltyp möglichst genau, damit z.B. wie im obigen Beispiel nicht einfach alle ICMP-Nachrichten sondern wirklich nur Echo-Requests und Replies in der jeweils entsprechenden Richtung durchgelassen werden.

Teil 4 - SSH vom internen Netz auf den DMZ-Host zulassen

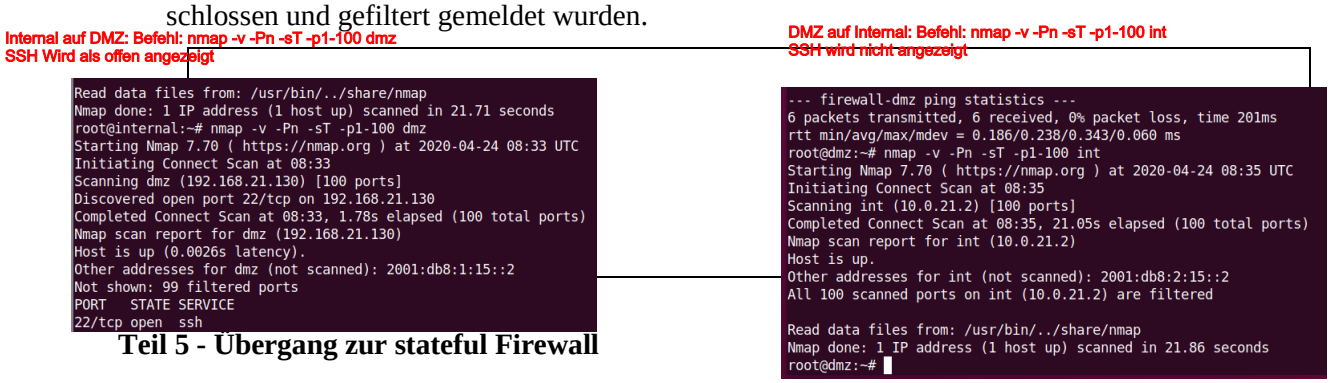
In diesem Teil werden Sie eine der beiden ssh-Verbindungen gemäss obiger Abbildung zulassen. Lösen Sie dazu die folgende Aufgabe:

11. Erlauben Sie Zugriff vom internen Netz auf den ssh-Server in der DMZ über IPv4 und IPv6. Loggen Sie sich anschliessend mit ssh vom internal-Host auf der DMZ ein (als Benutzer *root*), um zu testen, ob dies funktioniert. Prüfen Sie dann mit einem nmap-Scan, ob Sie nicht aus Versehen auch andere Services zugelassen haben. Prüfen Sie ebenfalls, mit einem nmap-Scan, dass Sie *nicht* vom DMZ-Host auf den ssh-Server auf dem internal-Host (auch da läuft ein ssh-Server) zugreifen können, also ob Sie nicht aus Versehen auch die Gegenrichtung geöffnet haben.

    Hinweis: Sie benötigen für eine vollständige Konfiguration die Parameter `iifname`, `oifname`, `{ip,ip6} {saddr,daddr}`, `tcp {sport, dport}` und `tcp flags != syn`. Die vier Einträge (je einen für IPv4 und IPv6 und je einen pro Richtung) müssen Sie in der myforward Chain vornehmen. Sie sollten den symbolischen Namen «`ssh`» für den ssh-Port verwenden statt des numerischen Äquivalents 22.

- **Praktikumspunkt**: Für den korrekten Output der beiden nmap-Scans vom internal- auf den DMZ-Host (je einen für IPv4 und einen für IPv6) erhalten Sie den ersten Punkt. Speichern Sie deshalb den Output (z.B. als Screenshot). Geben Sie zudem in der folgenden Box die wichtigsten

Informationen im Scan-Output an, insbesondere welche(r) Port(s) bei diesem Scan als offen, geschlossen und gefiltert gemeldet wurden.



Teil 5 - Übergang zur stateful Firewall

Sie haben sicher bemerkt, dass die ständige Konfiguration von paarweisen Regeln irgendwie mühsam ist. Dies liegt daran, dass wir bisher alle Regeln stateless konfiguriert haben. Ebenfalls wissen Sie aus der Vorlesung, dass eine stateless Firewall Probleme mit komplexeren Protokollen wie ftp hat. Dies liegt daran, dass der ftp-Server für jede Anfrage vom Client einen Datenkanal zum Client hin öffnet. Aus diesem Grund verwenden wir ab jetzt die stateful-Möglichkeiten von iptables und verwenden entsprechend die nftables Option `ct` (für «connection tracking»). In diesem Teil werden Sie die Vorbereitungen treffen, damit wir den Rest der Firewall stateful konfigurieren können. Natürlich würde man in der Realität gleich alle Regeln von Beginn an stateful konfigurieren.

12. Fügen Sie in der myforward-Chain anfangs eine protokollunabhängige Regel ein, welche alle Pakete passieren lässt, die in Bezug zu einer bestehenden Kommunikationsbeziehung stehen. Diese Regel lautet wie folgt:

    ```
    ct state established,related accept
    ```

    Hat man diese Regel erst einmal konfiguriert, so muss bei TCP-Verbindungen nur noch das initiale SYN-Segment einer TCP-Verbindung zugelassen werden, und zwar mit dem Zusatz «`ct state new`»; alles andere übernimmt die obige Regel. Ebenfalls wird die Firewall bei UDP-Kommunikationsbeziehungen und bei einem ICMP Meldungsaustausch die Antworten zulassen, die zu einem initialen Paket „passen“.

Teil 6 - Stateful Regeln konfigurieren

In diesem Teil werden Sie die restlichen Regeln stateful konfigurieren. Wie oben bereits angetönt müssen Sie im Gegensatz zu den vorhergehenden Teilaufgaben nur noch das initiale Paket in der „Vorwärtsrichtung“ durchlassen, alles andere übernimmt die stateful Firewall. Generell spezifizieren Sie das initiale Paket mit der iptables Option `ct state new`. Lösen Sie in diesem Teil die folgenden Aufgaben:

13. Erlauben Sie die weiteren zugelassenen Pings gemäss obiger Abbildung. Testen Sie auch hier, ob nur die erlaubten Pings funktionieren.

14. Erlauben Sie ebenfalls Zugriff vom external-Netz auf den ssh-Server in der DMZ. Dies ermöglicht es z.B. einem Administrator, von zu Hause aus gewisse Serverkonfigurationen vorzunehmen. Überprüfen Sie wiederum mit nmap, ob Sie alles korrekt konfiguriert haben.

15. Konfigurieren Sie die ssh-Regeln für den ssh-Zugriff auf die Firewall von stateless auf stateful um.

16. Lassen Sie nun ftp-Traffic vom external-Netz auf den ftp-Server in der DMZ zu. Damit das funktioniert, müssen Sie auf der Firewall den conntrack-helper aktivieren. Führen Sie dazu das Kommando «`sysctl -w net.netfilter.nf_conntrack_helper=1`» aus und testen Sie anschliessend mit nmap, ob Sie vom external-Host sowohl auf den ssh- als auch auf den ftp-Server zugreifen können und dass dies vom internal-Host nach wie vor nur den ssh-Server erreichen. Loggen Sie sich zudem als user *test* (Passwort *test*) vom external-Host auf dem ftp-Server ein und führen Sie ein Listing (`ls`) aus, um zu testen, dass wirklich ein ftp-Datenkanal geöffnet und durchgelassen wird. Mittels ausführen von `netstat -t` auf dem DMZ-Host können Sie zudem

sehen, dass ftp wirklich einen Control-Kanal (Port 21) und dann für jede nachfolgende Datenübertragung einen Datenkanal (Port 20) verwendet.

- **Praktikumspunkt**: Für den korrekten Output der beiden nmap-Scans vom external- auf den DMZ-Host (je einen für IPv4 und einen für IPv6) erhalten Sie den zweiten Punkt. Speichern Sie deshalb den Output. Geben Sie zudem in der folgenden Box die wichtigsten Informationen im Scan-Output an, insbesondere welche(r) Port(s) bei diesem Scan als offen, geschlossen und gefiltert gemeldet wurden.

```
root@external:~# nmap -v -Pn -sT -p1-100 dmz
Starting Nmap 7.70 ( https://nmap.org ) at 2020-04-24 13:46 UTC
Initiating Connect Scan at 13:46
Scanning dmz (192.168.21.130) [100 ports]
Discovered open port 22/tcp on 192.168.21.130
Discovered open port 21/tcp on 192.168.21.130
Completed Connect Scan at 13:46, 1.73s elapsed (100 total ports)
Nmap scan report for dmz (192.168.21.130)
Host is up (0.0091s latency).
Other addresses for dmz (not scanned): 2001:db8:1:15::2
Not shown: 98 filtered ports
PORT   STATE SERVICE
21/tcp open  ftp
22/tcp open  ssh

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 2.89 seconds
```

```
root@external:~# nmap -6 -v -Pn -sT -p1-100 dmz
Starting Nmap 7.70 ( https://nmap.org ) at 2020-04-24 13:49 UTC
Initiating Connect Scan at 13:49
Scanning dmz (2001:db8:1:15::2) [100 ports]
Discovered open port 22/tcp on 2001:db8:1:15::2
Discovered open port 21/tcp on 2001:db8:1:15::2
Completed Connect Scan at 13:49, 1.81s elapsed (100 total ports)
Nmap scan report for dmz (2001:db8:1:15::2)
Host is up (0.012s latency).
Other addresses for dmz (not scanned): 192.168.21.130
Not shown: 98 filtered ports
PORT   STATE SERVICE
21/tcp open  ftp
22/tcp open  ssh

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 2.43 seconds
```

Wie in der obigen Abbildung gewünscht erreichen wir vom external aus, den DMZ mittels SSH oder FTP (beides funktioniert, sowohl als ipv4, wie auch ipv6)

Vergleichen wir das mit dem nmap ext aus der vorherigen Aufgabe so sehen wir das die Ports 7, 9, und 13 nicht mehr aufgelistet werden

17. In einem realistischen Szenario dürfen meist alle Benutzer im internal-Netz auf das Internet zugreifen. Lassen Sie deshalb sämtlichen TCP-Traffic vom internal-Netz zum external-Netz zu. Verifizieren Sie dies mit einem nmap-Scan des external-Hosts vom internal-Host aus. Wie sieht der Output aus, insbesondere bzgl. den Port(s), die bei diesem Scan als offen, geschlossen und gefiltert gemeldet wurden? Entspricht das Verhalten Ihren Erwartungen?

```
root@internal:~# nmap -v -Pn -sT -p1-100 ext
Starting Nmap 7.70 ( https://nmap.org ) at 2020-04-24 13:52 UTC
Initiating Connect Scan at 13:52
Scanning ext (192.168.21.2) [100 ports]
Completed Connect Scan at 13:53, 1.75s elapsed (100 total ports)
Nmap scan report for ext (192.168.21.2)
Host is up (0.0015s latency).
Other addresses for ext (not scanned): 2001:db8:3:15::2
Not shown: 99 filtered ports
PORT   STATE  SERVICE
80/tcp closed http

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 2.45 seconds
```

Insofern deckt es meine Erwartung, dass nur http explizit aufgelistet wird. Jedoch habe ich erwartet, dass es als "open" deklariert wird. Jedoch sehen wir hier, dass es als "closed" aufgelistet wird

18. Um Netzwerkprobleme aufzuspüren soll auch ein Traceroute vom internal- zum external-Netz möglich sein. Traceroute arbeitet mit UDP-Datagrammen, die Destination-Ports im Bereich 33434-33523 verwenden. Die zurückkommenden Antworten sind ICMP Meldungen, entweder `time-exceeded` (von Zwischenknoten) oder `destination-unreachable` (vom Zielhost). Diese Antworten werden von der Firewall in den meisten Fällen durch die oben spezifizierte extablished,related-Regel bereits durchgelassen, weil sie zu ausgehenden UDP-Datagrammen gehören (related). Deshalb müssen Sie nur die vom internal-Netz herausgesendeten UDP-Datagramme korrekt durch die Firewall durchlassen. Spezifizieren Sie diese Regel und kontrollieren Sie, ob der Traceroute vom internal-Host zum external-Host funktioniert (Befehl: `traceroute ext`, bzw. `traceroute -6 ext`.)

- **Praktikumspunkt**: Für den korrekten Output dieses Traceroute-Befehls erhalten Sie den dritten Punkt. Speichern Sie deshalb den Output. Geben Sie zudem in der folgenden Box diesen Output an und liefern Sie eine Erklärung, wieso in der ersten Zeile nur „Sternchen“ gemeldet werden.

```
root@internal:~# traceroute ext
traceroute to ext (192.168.21.2), 30 hops max, 60 byte packets
 1  * * *
 2  ext (192.168.21.2)  0.667 ms  0.355 ms  0.295 ms
```

Teil 7 - NAT einrichten

Damit „gegen Aussen“ nicht die internen IPv4-Adressen sichtbar werden, wenden wir im letzten Teil auch noch Network Address Translation (NAT oder besser gesagt Source NAT) an. Dabei agiert die Firewall als NAT-Box und wird alle Pakete vom internal-Netz mit der eigenen, externen IPv4-Adresse versehen (192.168.x.1). (Für IPv6 gibt es andere Mechanismen; NAT ist für IPv6 nicht vorgesehen.) Lösen Sie dazu die folgende Aufgabe:

19. Fügen Sie eine Regel an, dass NAT wie oben angegeben richtig funktioniert. Dabei müssen Sie eine neue Table erzeugen (z.B. namens `mynat`), die auf der `ip` Address Family arbeitet. In dieser Table erzeugen Sie die eine Source-NAT-Chain (z.B. namens mysnat), die dem postrouting-Hook zugeordnet ist. Innerhalb dieser Chain erzeugen Sie nun eine Regel, die bei Paketen aus dem internen Netz ins externe Netz Source-NAT anwendet (option `snat` mit der IPv4-Adresse des externen Interfaces als Argument). Beachten Sie, dass nur die internen Adressen übersetzt werden sollen, und auch nur dann, wenn die Verbindung auf das externe Netz geht; Sie müssen dies also durch `oifname`, `ip saddr` und `ip daddr` Parameter geeignet eingrenzen (nicht jedoch durch `iifname`, das funktioniert aus irgendeinem Grund nicht). Sie können den korrekten nftables Eintrag überprüfen, indem Sie vom internal-Host eine ssh-Connection auf den external-Host aufbauen und auf dem external-Host die bestehenden Verbindungen anschauen (`netstat -t`). Die ssh-Connection ist als ESTABLISHED eingetragen und die „Foreign Address“ sollte die der Firewall sein.

    Hinweis: Verwenden Sie die Option «`-4`» beim ssh-Client, sonst wird manchmal IPv4 und manchmal IPv6 verwendet.

- **Praktikumspunkt**: Für den korrekten Output dieses netstat-Commands erhalten Sie den vierten Punkt. Speichern Sie deshalb den Output, wobei die ersten paar Zeilen bis und mit der entscheidenden ESTABLISHED-Zeile genügen. Tragen Sie die relevante ESTABLISHED-Zeile zudem in der folgenden Box ein.

```
root@external:~# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 ext:ssh                 firewall-ext:36912      ESTABLISHED
```

Teil 8 – aufräumen

Wenn Sie alles richtig gemacht haben, wird Ihre Firewall genau das tun, was sie soll, aber das Skript wird unordentlich aussehen. Ausserdem wird es einiges an Regelduplikation geben, einmal für IPv4 und dann noch einmal die fast identische Regel für IPv6. Das wollen wir am Schluss noch beseitigen.

In der Vorlesung haben wir die Möglichkeit kennengelernt, die Bearbeitung eines Pakets in einer anderen Chain fortzusetzen, einer sogenannten non-base-chain. Dazu dient die action `goto`.

Erzeugen Sie eine (vorerst leere) Chain namens `int-to-dmz`, indem Sie zu den bereits bestehenden Chains diese Chain hinzufügen:

```
chain int-to-dmz {

}
```

Lokalisieren Sie nun die Regeln, die den Verkehr vom internen Netz zur DMZ betreffen. Das sollten vier Regeln sein: zwei für ping (IPv4 und IPv6) und zwei für ssh (IPv4 und IPv6). Ersetzen Sie diese vier Regeln nun durch zwei Regeln:

```
# Internal to DMZ
iifname $iifc ip saddr $i4nw oifname $difc ip daddr $d4nw ct state
```

```
new goto int-to-dmz
  iifname $iifc ip6 saddr $i6nw oifname $difc ip6 daddr $d6nw ct
state new goto int-to-dmz
```

In der int-to-dmz-Chain fügen Sie nun folgende Regeln ein:

```
  chain int-to-dmz {
    # Allow ping
    icmp type echo-request accept
    icmpv6 type echo-request accept

    # Allow ssh
    tcp dport ssh accept
  }
```

Prüfen Sie, ob alles immer noch so funktioniert, wie gedacht.

Auf den ersten Blick haben wir nun vier Regeln durch fünf ersetzt und damit wenig gewonnen, aber das stimmt nicht. Denn die Prüfung auf `iifname`, `oifname`, `ct state` usw. muss nur noch ein- oder zweimal gemacht werden, nicht viermal bei jedem zutreffenden Paket. Dadurch kann die Bearbeitung von Regeln deutlich beschleunigt werden. Ausserdem gibt es bei dieser Organisation für jeden Typ von Traffic (internal nach DMZ, internal nach extern, …) genau eine chain, in der alle Regeln stehen. Bei grossen Regelsätzen vereinfacht das die Administration deutlich.

20. Erstellen Sie nun für die Traffic-Typen «intern nach extern», «DMZ nach extern» und «extern nach DMZ» analoge non-base-Chains. Am Ende sollte die myforward-Chain nur noch den `ct state established,related` und `jump`s auf die entsprechenden non-base-Chains enthalten.

Praktikumspunkte

In diesem Praktikum können Sie **6 Praktikumspunkte** erreichen. Dazu müssen Sie die vier Outputs (siehe Teil 3, 5 und 6) und das fertige Firewall-Skript in einer E-Mail an den Betreuer senden oder dem Betreuer während der Praktikumssession zeigen. Verwenden Sie *Security Lab - iptables - Gruppe X - Name1 Name2* als Subject; entsprechend Ihrer Gruppennummer und den Namen der Gruppenmitglieder.

Anhang: Einfrierende UML Systeme und Abhilfe

Falls die UML Systeme bei Ihnen nach kurzer Zeit einfrieren (kein Input mehr möglich), dann könnte dies an einem Problem mit der Zeitsynchronisation resp. mit den VMware Tools liegen. Abhilfe kann hier das Deaktivieren der Zeitsynchronisation bringen. Dies können Sie wie folgt machen:

- Im vmx-File folgenden Eintrag auf *FALSE* setzen: *tools.syncTime = "FALSE"*
- Oder mit der VMware Software die entsprechende Checkbox in den Settings deaktivieren.

Das ist allerdings mit neueren UML-Versionen und neueren Versionen zumindest von VirtualBox nicht aufgetreten.

1.

21.

22.

23.

24.

25.

26.

27.

28.

29.

30.