Security Lab – Linux Firewall mit nftables

Dieses Lab besteht aus zwei Teilen, welcher jeder 2 Punkte gibt.

Teil 1: Page 1-12

Teil 2: Page 13-15

Setup

Das Lab wird in unserem «cloud-lab» in Gruppen gelöst:

https://sec.cloudlab.zhaw.ch

Die Infrastruktur ist so aufgesetzt, dass Sie nach dem login direkt zu den VMs, welche Ihrer Gruppe zugeordnet sind, gelangen.

Achtung: alle Mitglieder der Gruppe arbeiten auf denselben VMs.

Pro Gruppe sind dies die folgenden VMs (wobei ID der Gruppennummer entspricht):

- Internal: `its-group-<GROUP_ID>-internal`
- External: `its-group-<GROUP_ID>-external`
- DMZ: `its-group-<GROUP_ID>-dmz`
- Firewall: `its-group-<GROUP_ID>-firewall`

Die VMs befinden sich alle in verschiedenen Subnetzen. Die IP-Adressvergabe ist wie folgt:

1. Internal VM: IP: `10.x.1.10`
2. DMZ VM: IP: `10.x.2.10`
3. External VM: IP: `10.x.3.10`
4. Firewall VM (Es hat drei IP-Addressen, da er über drei Netzwerkschnittstellen verfügt):
   - IP: `10.x.1.5 (ens3)`
   - IP: `10.x.2.5 (ens4)`
   - IP: `10.x.3.5 (ens5)`

Bitte beachten Sie, dass "x" für die GROUP_ID + 100 steht. Wenn Ihre group_ID zum Beispiel 42 ist, dann ist Ihre Internal VM IP 10.142.1.10. Weitere Informationen über die Hosts, die Netzwerktopologie und die IP-Addressen finden Sie in Kapitel 3.

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 wir unsere Cloudlab-Umgebung nutzen, um eine virtuelle, praxisnahe Umgebung zu simulieren, in der Sie nftables und nmap anwenden können.

Hinweis. Für dieses Praktikum benötigen Sie eine *Gruppennummer*. Diese haben Sie bereits für Ihre früheren Praktika erhalten. Falls nicht, fragen Sie Ihren Dozenten.

2 Grundlagen

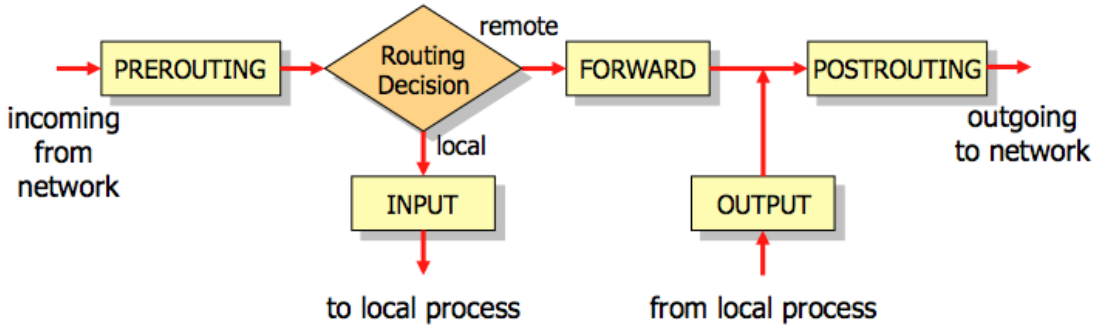
2.1 nftables

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
```

Regeln 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.

Das Debuggen von Regeln ist oft knifflig. Dazu kann man in einer Chain

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

angeben, wodurch in den Metadaten der Pakete, die durch diese Chain gehen, das nftrace Debuggin-Bit gesetzt wird. Fortan werden die weiteren Stationen des Pakets durch nftables geloggt. Matcht das Paket später eine Regel (oder geht an eine Chain) mit «meta nftrace set 0» wird das logging für dieses Paket deaktiviert und dessen weiter Stationen nicht mehr geloggt. 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 VMs 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.

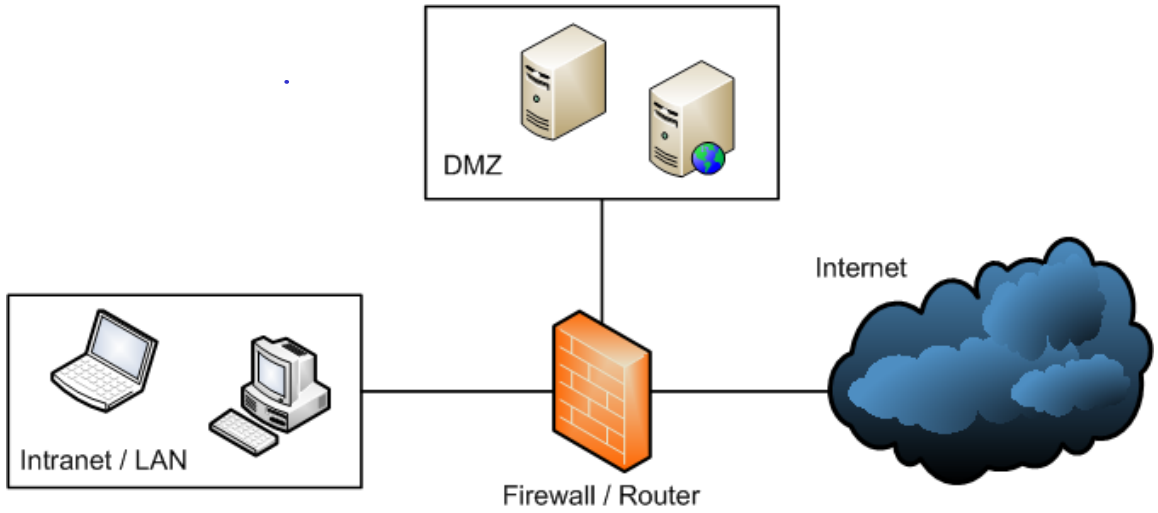
2.4 `nmap` Beispiele

`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.

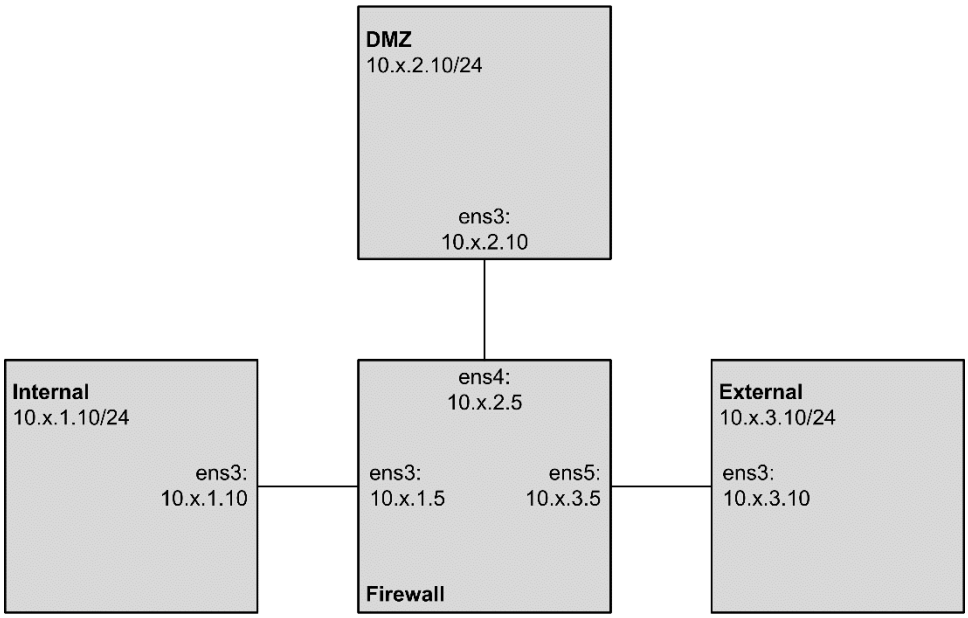


Wie in der Abbildung ersichtlich, möchten wir mit dieser Netzarchitektur eine vereinfachte Unternehmensumgebung mit folgenden Komponenten realisieren:

- 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.

3.1 Praktikum Setup im Cloudlab

Für einen einfachen Setup in unserer Cloudlab-Umgebung 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 GROUP_ID + 100; es werden also alle Gruppen eigene private Adressbereiche verwenden. Die Firewall hat also auf ens3 bei Gruppe 12 die IPv4-Adresse 10.12.1.5.

Melden Sie sich auf allen vier VM-Instanzen. 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.

Hinweis: Auf den Hosts sind die Interfaces der verschiedenen Rechner mit symbolischen Namen ausgestattet, was es einem erspart, die IP-Adressen anzugeben

Interface-Name	Adressen
firewall-int	10.x.1.5
int	10.x.1.10
firewall-dmz	10.x.2.5
dmz	10.x.2.10
firewall-ext	10.x.3.5
ext	10.x.3.10

Will man also das Interface des internal-Hosts anpingen, tippt man `ping -4 int`. (Man braucht -4 hier nicht anzugeben, weil die Interfaces zwar alle auch IPv6-Adressen haben, aber keine Namen. Trotzdem sollte man das machen, um sich anzugewöhnen, zwischen IPv4 und IPv6 zu unterscheiden.)

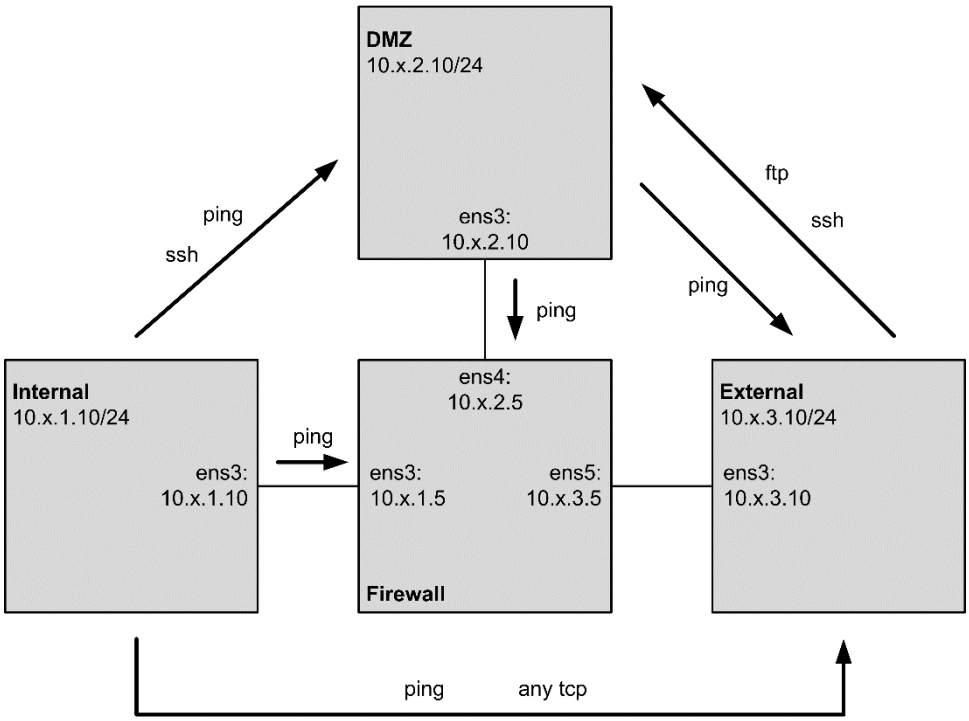
Prüfen Sie auch, ob auf dem DMZ-Host ein ssh- und ftp-Server laufen (am besten mit `netstat -l`) und überprüfen, ob auf den Ports 21 (ftp) und 22 (ssh) TCP-Server am „listenen“ sind). Prüfen Sie analog, ob auf dem externen Host ein ssh-Server läuft. Wenn all dies funktioniert, sind Sie für die Aufgaben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

Hinweis: Manchmal dauert der Verbindungsaufbau von z.B. ssh sehr lange. Brechen Sie ihn also nicht ab, sondern warten Sie, bis er entweder funktioniert hat, oder sie eine Fehlermeldung erhalten.

3.2 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.



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 `nft -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:

> 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. Wie sieht die Kommandozeile und der zugehörige 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:** Erstellen Sie entweder einen Screenshot, der die Kommandozeile und den gesamten Output von nmap zeigt oder kopieren Sie die Kommandozeile und den gesamten Output von nmap in Ihre Lösung. Allgemeine Formulierungen wie «Ports 80, 81 und 82 offen, alle anderen geschlossen» geben *keine* Punkte. Die *Kommandozeile und der gesamte Output von nmap* muss zu sehen sein!

```
user@its-group-5-external:~$ nmap -Pn -p 1-100 10.5.2.10
Starting Nmap 7.80 ( https://nmap.org ) at 2024-04-21 11:25 CEST
Nmap scan report for 10.5.2.10
Host is up (0.0079s latency).
Not shown: 98 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
23/tcp open  telnet

Nmap done: 1 IP address (1 host up) scanned in 1.62 seconds
```

Es sollte so sein, in den Firewall Regeln, werden sie durchgelassen

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.

```
user@its-group-5-external:~$ nmap -Pn -p 1-100 10.5.2.10
Starting Nmap 7.80 ( https://nmap.org ) at 2024-04-21 11:42 CEST
Nmap scan report for 10.5.2.10
Host is up (0.18s latency).
Not shown: 98 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
23/tcp open  telnet
```

Es braucht noch myforward

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

```
user@its-group-5-external:~$ nmap -Pn -p 1-100 10.5.2.10
Starting Nmap 7.80 ( https://nmap.org ) at 2024-04-21 11:45 CEST
Stats: 0:00:23 elapsed; 0 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 45.00% done; ETC: 11:46 (0:00:12 remaining)
Stats: 0:00:24 elapsed; 0 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 50.00% done; ETC: 11:46 (0:00:11 remaining)
Stats: 0:00:28 elapsed; 0 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 70.00% done; ETC: 11:46 (0:00:06 remaining)
Nmap scan report for 10.5.2.10
Host is up.
All 100 scanned ports on 10.5.2.10 are filtered

Nmap done: 1 IP address (1 host up) scanned in 34.09 seconds
```

Jetzt sucht er so lange bis der Timer abgelaufen ist. Wenn reject wäre dann ginge es schneller.

Teil 2 - Ping auf die Firewall zulassen

In Teil 1 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:

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. 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), `icmp type`, sowie die ICMP-Nachrichtentypen `echo-request` und `echo-reply`. Als Beispiel sind die zwei 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
    iifname $iifc ip saddr $i4nw icmp type echo-request accept
  }
  chain myoutput {
    type filter hook output priority 0; policy drop;
    #meta nftrace set 1
    oifname $iifc ip daddr $i4nw icmp type echo-reply accept
  }
```

Beachten Sie, dass mit diesen beiden Regeln *alle* Hosts im gesamten Netz 10.x.1.10/24 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.x.1.10) 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.

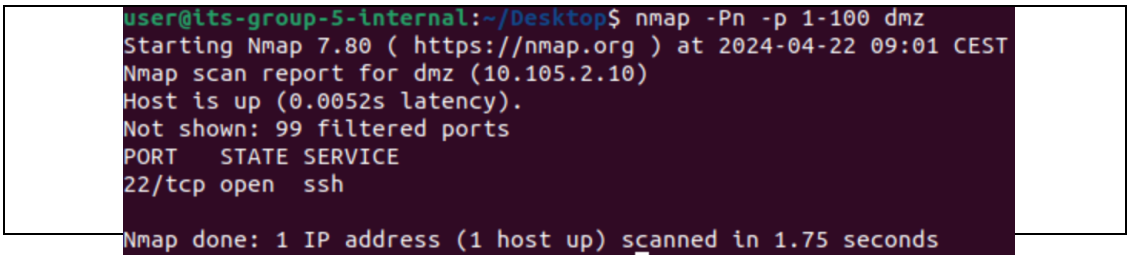
Teil 3 - SSH vom internen Netz auf den DMZ-Host zulassen

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

Erlauben Sie Zugriff vom internen Netz auf den ssh-Server in der DMZ. 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 {saddr,daddr}`, `tcp {sport, dport}` und `tcp flags != syn`. Die zwei Einträge (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 des nmap-Scans vom internal- auf den DMZ-Host 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 4 - 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.

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 5 - 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.

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

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.

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

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 *user* (Passwort *securityzhaw*) 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 des nmap-Scans vom external- auf den DMZ-Host 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.

```
user@its-group-5-external:~$ nmap -Pn -p 1-100 dmz
Starting Nmap 7.80 ( https://nmap.org ) at 2024-04-22 10:02 CEST
Nmap scan report for dmz (10.105.2.10)
Host is up (0.0061s latency).
Not shown: 98 filtered ports
PORT   STATE SERVICE
21/tcp open  ftp
22/tcp open  ssh
```

---------- Der erste Teil ist abgeschlossen (Woche 1). ----------

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?

```
PORT     STATE SERVICE
7/tcp    open  echo
9/tcp    open  discard
13/tcp   open  daytime
22/tcp   open  ssh
3389/tcp open  ms-wbt-server
```

alle mit TCP sind open.
also alle bekommen ack response.
es sollte so sein.

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`.)

- **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. In dieser Erklärung muss *der Lauf der Pakete genau beschrieben* werden: welche Pakete werden von wem an wen geschickt, was lösen sie dort aus, was passiert dann usw. Allgemeine Antworten wie «Die Firewall droppt traceroute» geben *keine* Punkte (und sind zudem falsch).

```
user@its-group-5-internal:~/Desktop$ traceroute ext
traceroute to ext (10.105.3.10), 30 hops max, 60 byte packets
 1  * * *
 2  ext (10.105.3.10)  8.450 ms  8.426 ms  8.416 ms
```

Internel versucht External zu erreichen.
Zeilenzahl entspricht die Time To Live (wieviele Hops (Netzwerkgeräte) die Traceroute überspringen kann bevor es stirbt (Sternchen))
Bei der ersten Zeile erreicht das Traceroute erst die Firewall.
bei der zweite Zeile schafen die 3 Packete bis external.

Teil 6 - 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 (10.x.3.5). Lösen Sie dazu die folgende Aufgabe:

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 machmal 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.

```
user@its-group-5-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:47952      ESTABLISHED
tcp6       0      0 ext:ms-wbt-server       10.99.0.189:39900       ESTABLISHED
user@its-group-5-external:~$ netstat -4 -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 ext:ssh                 firewall-ext:40298      ESTABLISHED
```

Teil 7 – aufräumen

Wenn Sie alles richtig gemacht haben, wird Ihre Firewall genau das tun, was sie soll, aber das Skript wird unordentlich aussehen. 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 zwei Regeln sein: eine für ping und eine für ssh. Ersetzen Sie diese zwei Regeln nun durch eine Regel:

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

  # Allow ssh
  tcp dport ssh accept
}
```

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

Auf den ersten Blick haben wir nun zwei Regeln durch drei ersetzt und damit wenig gewonnen, aber das stimmt nicht. Denn die Prüfung auf `iifname`, `oifname`, `ct state` usw. muss nur noch einmal gemacht werden, nicht zweimal bei jedem zutreffenden Paket. Dadurch kann---besonders bei grösseren Regelsätzen---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.

> 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 **2 (für den ersten Teil) + 2 (für den zweiten Teil) = 4 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.