Labreport 04

Julian Deinert, Tronje Krabbe

20. Juni 2016

Inhaltsverzeichnis

1. Netzwerkeinstellungen

1.2 Ermitteln der Netzwerkkonfiguration

- Die ClientVM hat die IP-Adresse `192.168.254.44` und das Standardgateway `192.168.254.2` außerdem verwendet sie den DNS-Server `10.1.1.1`.
- Die RouterVM besitzt für das Interface *eth0* die IP-Adresse `172.16.137.222` und für das Interface *eth1* die IP-Adresse `192.168.254.2`.
- Die ServerVM hat die IP-Adresse `172.16.137.144`.

2. Absichern eines Einzelplatzrechners mit iptables (ClientVM)

2.1 Löschen aller Firewallregeln

Wir richten die default policy wieder ein und flushen alle Chains in der *filter-*, *nat-* und *mangle-table*.

```
# iptables -P INPUT ACCEPT
# iptables -P FORWARD ACCEPT
# iptables -P OUTPUT ACCEPT

# iptables -t nat -F
# iptables -t mangle -F
# iptables -F
# iptables -X
```

Danach installieren wir mit `apt-get` das Paket `openssh-server`.

2.2 Entwerfen eines Konzepts

Wir wollen Traffic durch Port `80` und `443` generell erlauben und Traffic durch Port `22` nur aus dem lokalen Netzwerk zulassen. Hierzu setzen wir die folgenden IP-Table Einträge: Zunächst ändern wir die defaults targets der drei chains `INPUT`, `OUTPUT` und `FORWARD` zu `DROP` (`REJECT` geht nicht; der Versuch, `REJECT` als target zu nutzen endet mit der Fehlermeldung *iptables: bad policy name. Run 'dmesg' for more information.*).

```
# iptables -P INPUT DROP
# iptables -P OUTPUT DROP
# iptables -P FORWARD DROP
```

Nun ist also gar keine Kommunikation mehr möglich. `ping` z.B. schlägt fehl mit *ping: sendmsg: Operation not permitted*. So weit, so gut. Nun wollen wir aber einige Kommunikation erlauben. Zunächst kümmern wir uns um *ssh*:

```
# iptables -A INPUT -i eth0 -p tcp -s 192.168.254.0/24 --dport 22 -j ACCEPT
# iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT
```

Nach diesen Befehlen kann man sich aus dem lokalen 192.168.254er Netz auf die Client-VM per *ssh* verbinden. Lässt man den `-s` Switch weg, kann man sich von überall auf die VM *ssh*en.

Nun brauchen wir noch HTTP (und HTTPS); wir erlauben eingehenden tcp traffic, der von einem Port 80 kommt, und ausgehenden tcp traffic, der zu einem Port 80 will (bzw. Port 443 für HTTPS):

```
# iptables -A OUTPUT -o eth0 -p tcp --dport 80 -j ACCEPT
# iptables -A INPUT -i eth0 -p tcp --sport 80 -j ACCEPT

# iptables -A OUTPUT -o eth0 -p tcp --dport 443 -j ACCEPT
# iptables -A INPUT -i eth0 -p tcp --sport 443 -j ACCEPT
```

Jetzt brauchen wir noch Regeln für ICMP:

```
# iptables -A INPUT -p icmp -j ACCEPT
# iptables -A OUTPUT -p icmp -j ACCEPT
```

Wenn man lediglich z.B. pings (und pongs) erlauben will, würde man noch die `-icmp-type` flag auf einen entsprechenden Wert setzen. In der Aufgabe wurde allerdings nur "ICMP Nachrichten" spezifiziert, also erlauben wir alles, was ICMP ist.

2.3 Prüfen der Korrektheit der Regeln

- Von der RouterVM aus können wir eine SSH-Verbindung zur ClientVM aufbauen. Von Client- zu RouterVM geht dies nicht. Es soll ja nur eingehendes SSH erlaubt sein.
- Wir starten einen netcat server in der ClientVM auf Port 5555 mit `nc -l 5555` (mit GNU netcat hätten wir noch die -p Flag nehmen müssen). Jetzt versuchen wir, uns damit von der RouterVM aus zu verbinden: `nc 192.168.254.44 5555`. Komischerweise ergibt dies keine Fehlermeldung, aber der netcat 'client' terminiert nach einer kurzen Zeit. In dieser Zeit können keine Nachrichten ausgetauscht werden. Dies ist natürlich gewollt; wir wollen nur HTTP(S), SSH und ICMP zulassen. Port 5555 hat mit keiner dieser Protokolle zu tun, und wird deshalb nicht durchgelassen.
- Da REJECT nicht als target für die default chains erlaubt ist, können wir dies nicht beobachten. REJECT verhält sich aber so: Während DROP das Paket einfach fallen lässt (Zitat man-page: "DROP means to drop the packet on the floor."), sendet REJECT noch eine Nachricht, dass das Paket eben rejected wurde.

2.4 Stateful Filtering

Zunächst: `# iptables -F`. Nun erlauben wir wieder SSH:

```
# iptables -A OUTPUT -o eth0 -p tcp -d 192.168.254.0/24 --dport 22 -m state --state NEW,ESTABLISHED -j
# iptables -A INPUT -i eth0 -p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT
```

Nun beachten wir auch den state der Verbindung, und akzeptieren eingehende Pakete nur, wenn sie Teil einer bereits bestehenden Verbindung sind. Ähnlich für HTTP(S):

```
# iptables -A OUTPUT -o eth0 -p tcp --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
# iptables -A INPUT -i eth0 -p tcp --sport 80 -m state --state ESTABLISHED -j ACCEPT

# iptables -A OUTPUT -o eth0 -p tcp --dport 443 -m state --state NEW,ESTABLISHED -j ACCEPT
# iptables -A INPUT -i eth0 -p tcp --sport 443 -m state --state ESTABLISHED -j ACCEPT
```

Die Regeln für ICMP bleiben gleich; hier gibt es keine bestehenden Verbindungen o.ä.

Stateful filtering kann flexibler und übersichtlicher sein. Man beachte folgende Regel: `iptables -A INPUT -m conntrack -ctstate RELATED,ESTABLISHED -j ACCEPT`. Zusammen mit einer 'DROP' policy sichert sie ein System relativ gut ab. Eingehende Pakete werden nur akzeptiert, wenn sie bereits Teil einer etablierten oder

verwandten Verbindung sind.
Will man jedoch spezifischer filtern, kommt es immer darauf an, was genau erreicht werden soll; dann ist der state eines Pakets nur eine weitere Variable, nach der gefiltert werden kann, und sollte nicht pauschal bevorzugt werden, wenn es nicht nötig ist.

3. Absichern eines Netzwerks (RouterVM)

3.1 Dynamisches NAT

Der `iptables` Aufruf in */etc/rc.local* lautet:

```
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.254.0/24 -j MASQUERADE
```

Der `-t nat` Switch wählt die *nat* Table aus, der `-A POSTROUTING` Switch erstellt eine neue Regel, um ausgehende Packets zu verändern. `-o eth0` wählt das Out-Interface aus, der `-s` Switch die Quell-Adresse. `-j MASQUERADE` spezifiziert das `MASQUERADE` target, welches dafür sorgt, dass Traffic unverändert an das Interface fließt. Insgesamt sorgt die Regel also dafür, dass Traffic aus dem `192.168.254.0/24` Netz zum `eth0` Interface durchgelassen wird.

3.2

- Wir können die ServerVM von der ClientVM aus *ping*en.
- Von der ServerVM aus können wir nicht die ClientVM ansprechen, da die ServerVM im `172.16.137.0/24` Netz liegt. Tatsächlich kommen pings von der ServerVM in das `192.168.254.0/24` Netz scheinbar gar nicht in der RouterVM an. WireShark zeigt nämlich keine entsprechenden Pakete an. Von der ClientVM zur ServerVM werden die Request und Response Pakete korrekt aufgelistet (von Client zu Router, Router zu Server, und wieder zurück).

3.3

Als erstes blockieren wir pauschal alles, was durch die RouterVM durchgehen soll:

```
iptables -P FORWARD DROP
```

Zunächst das einfachste: wenn man auf 10.1.1.2 zugreifen möchte, soll dies nicht möglich sein:

```
# iptables -A FORWARD -d 10.1.1.2 -j DROP
```

Jetzt wird alles gedroppt, was durch die RouterVM dort hin möchte.

Appendix