Labreport 05

Julian Deinert, Tronje Krabbe

23. 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 ACCEP
# 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 NAT-Abläufe

- 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 IP-Tables

Wir filtern HTTP-Requests auf Port 80 und 443 nach `10.1.1.2` und `172.16.137.144` raus, und lassen alle anderen durch:

```
# iptables -A FORWARD -p tcp -d 10.1.1.2 --dport 80 -j DROP
# iptables -A FORWARD -p tcp -d 10.1.1.2 --dport 443 -j DROP
# iptables -A FORWARD -p tcp -d 172.16.137.144 --dport 80 -j DROP
# iptables -A FORWARD -p tcp -d 172.16.137.144 --dport 443 -j DROP
# iptabels -A FORWARD -p tcp -d 10.0.0.0/8 -j DROP
# iptables -A FORWARD -p tcp --dport 80 -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
# iptables -A FORWARD -j DROP
```

3.4 Anpassung für SSH-Verbindung

Wir möchten eine Verbindung zur ServerVM unter `172.16.137.144` auf Port `22` ermöglichen. Hierzu fügen wir an Stelle 7 eine neue Regel in die *FORWARD CHAIN* ein. Wir lassen hier außerdem das Protokoll weg, damit auch UDP-Verbindungen möglich sind.

```
# iptables -I FORWARD 7 -d 172.16.137.144 --dport 22 -j ACCEPT
```

3.5 Weiterleitung 5022 auf 22

Wir leiten den Port `5022` der RouterVM auf den Port `22` der ClientVM weiter.

```
# iptables -t nat -A -p tcp --dport 5022 -j DNAT --to-destination 192.168.254.44:22
```

3.6 Statische NAT-Regel

Wir wählen uns eine neue IP-Adresse aus dem gleichen Subnetz in der auch die ServerVM ist aus.

```
# ifconfig eth0:1 172.16.137.225 netmask 255.255.255.0
```

Alle Anfragen auf die IP `172.16.137.225` sollen nun zur ClientVM durchgereicht werden.

```
# iptables -t nat -A PREROUTING -d 172.16.137.225 -j DNAT --to-destination 192.168.254.44
```

4. SSH-Tunnel

4.1 Konfigurieren der Firewall

```
# iptables -A FORWARD 0s 192.168.254.44 -d 172.16.137.144 -p tcp --dport 22 -j ACCEPT
# iptables -A FORWARD -s 192.168.254.44/32 -p udp -m udp --dport 53 -j ACCEPT
# iptables -A FORWARD -s 192.168.254.44/32 -j DROP
```

4.2 SSH-Tunnel

Wir richten den SSH-Tunnel wie folgt ein.

```
$ ssh -L 8000:localhost:80 user@172.16.137.144
```

Wireshark bestätigt die Korrektheit des Befehls.

4.3 Tunnel ins Internet

```
$ ssh -D lcalhost:5555 user@172.16.137.144
```

Konfiguration von Firefox im Appendix.

4.4 Netcat

Auf der ClientVM

```
$ nc -l 5555
$ ssh -R 8000:localhost:5555 user@172.16.137.144
```

Um netcat auf Port `5555` lauschen zu lassen und einen Tunnel von ServerVM Port `8000` auf ClientVM Port `5555` einzurichten.

Dann, auf der ServerVM:

```
# nc localhost 8000
```

Nun können wir beliebig Daten versenden.

5. OpenVPN

5.1 Filter-Regeln

```
# iptables -A FORWARD -s 192.168.254.0/24 -d 172.16.137.0/24 -j ACCEPT
# iptables -A FORWARD -j DROP
```

5.2 OpenVPN-Server einrichten

Auf der ServerVM

```
$ openvpn --genkey --secret static.key
```

5.3 Filterregeln auf der ServerVM

```
# iptables -A INPUT -p udp --dport 1194 -j ACCEPT
# iptables -A INPUT -j DROP
# iptables -A OUTPUT -p udp --dport 1194 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
# iptables -A OUTPUT -j DROP
```

5.4 VPNClient

Clientconfig im Appendix

5.5 Starten des VPN-Tunnels

Auf der ServerVM

```
$ sudo openvpn server.conf
```

Auf der ClientVM

```
$ sudo openvpn client.conf
```

5.6 Funktionskontrolle

Es wird das Interface `tun0` verwendet.

```
Wed Jun 22 19:50:12 2016 Peer Connection Initiated with [AF_INET]172.16.137.222:1194
Wed Jun 22 19:50:12 2016 Initialization Sequence Completed
```

Über 10.8.0.1 können wir auf den Server zugreifen. Wir könne von der ServerVM auf die ClientVM zugreifen, da der Tunnel bidirektional ist. Dies erreichen wir über die VPN-IP der ClientVM(10.8.0.2).

6. HTTP-Tunnel

6.1 Neue Filterregeln

Wir erlauben nur tcp Port `80` und udp Port `53`

```
# iptables -A FORWARD -s 192.168.254.44 -p tcp --dport 80 -j ACCEPT
# iptables -A FORWARD -s 192.168.254.44 -p udp --dport 53 -j ACCEPT
# iptables -A FORWARD -s 192.168.254.44 -j DROP
```

6.2 Umgehen der Filterrregeln

Wir ändern in `etc/ssh/sshd_config` den Port von `22` auf `80` und starten den Service mit `service ssh restart` neu, nachdem wir den bereits auf Port `80` laufenden Apache HTTP-Server gestoppt haben.

6.3 Squid Firewallregeln

Wir erstellen folgende Firewallregeln:

```
# iptables -A INPUT -i eth1 -p tcp --dport 3128 -j ACCEPT
# iptables -A INPUT -i eth1 -j DROP
# iptables -A FORWARD -j DROP
# iptables -A OUTPUT -o eth0 -p udp --dport 53 -j ACCEPT
# iptables -A OUTPUT -o eth0 -p tcp --dport 80 -j ACCEPT
# iptables -A OUTPUT -o eth0 -p tcp --dport 443 -j ACCEPT
# iptables -A OUTPUT -o eth0 -j DROP
```

6.4 Proxy eintragen

Wir stellen in Firefox `192.168.254.2:3128` als HTTP-Proxy ein.

6.5 HTTP-CONNECT

Zunächst verbinden wir uns via *netcat* mit dem SSH-Server auf der ServerVM:

```
nc -x192.168.254.2:3128 -Xconnect 172.16.134.144 80
```

Danach konfigurieren wir unseren ssh-Befehl mittels Corkscrew in */.ssh/config*:

```
Host *
    ProxyCommand corkscrew 192.168.254.2 3128 %h %p
```

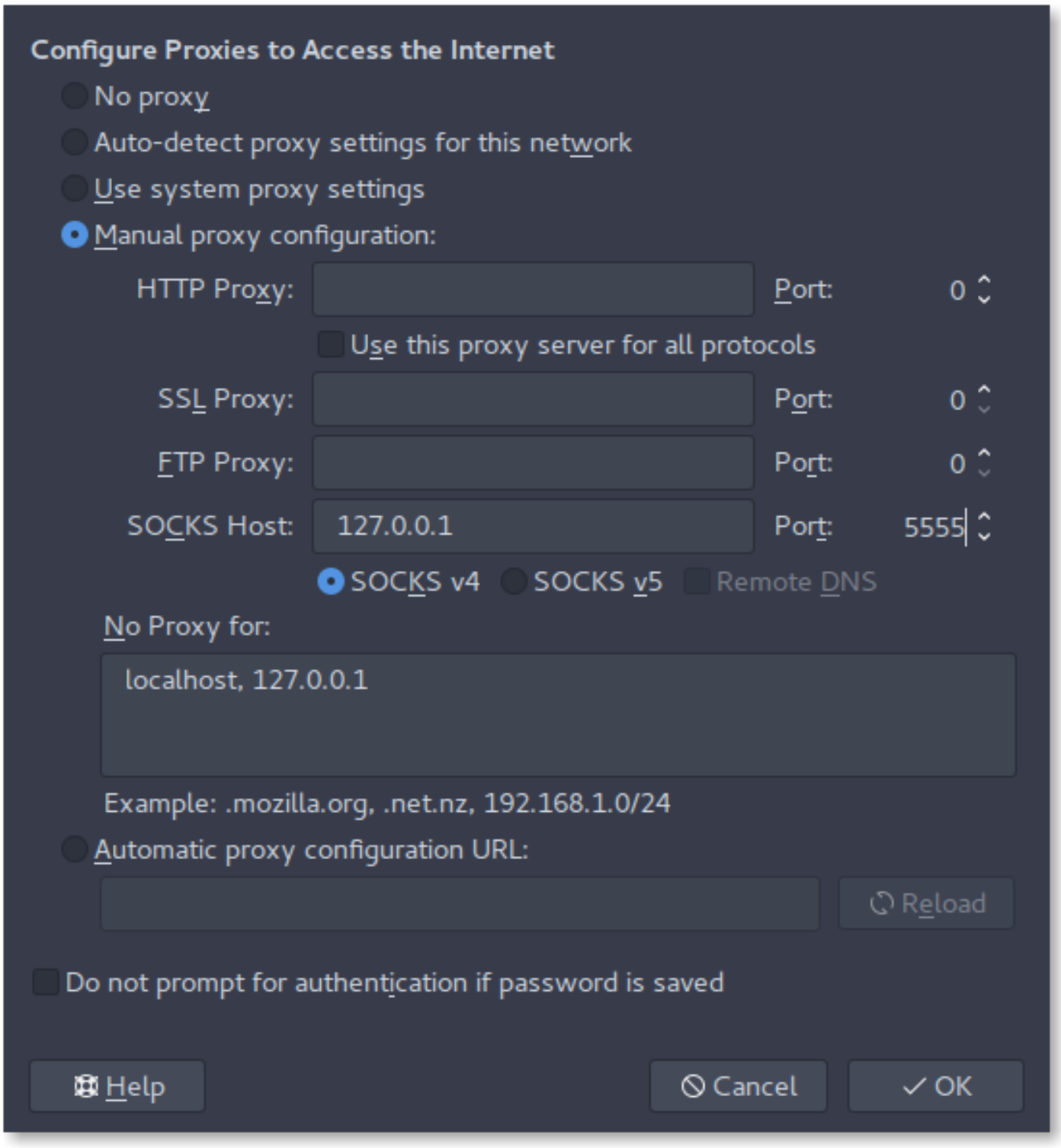
Jetzt können wir einfach einen HTTP-Tunnel via SSH aufbauen: `ssh -p 80 user@172.16.134.144`

6.6 HTTP-Tunnel

Nachdem wir httptunnel installiert haben, müssen wir nur mit `hts -F localhost 80` den Server auf der ServerVM öffnen und den Client mittels `htc -P 192.168.254.2:3128 -F 8888 172.16.134.144 80` auf der ClientVM starten. Danach können wir einen HTTP-Tunnel von der ClientVM zur ServerVM aufbauen: `ssh -p 8888 user@localhost`

Appendix

4.3 Konfiguration von Firefox



5.2 OpenVPN Server Config

```
port 1194
dev tun
ifconfig 10.8.0.1 10.8.0.2
secret static.key
keepalive 10 120
;status openvpn-status.log
verb 3
```

5.2 OpenVPN Client Config

```
dev tun
remote 172.16.137.144 1194
;resolve-retry infinite
;nobind
ifconfig 10.8.0.2 10.8.0.1
secret static.key
verb 3
```

Hinweis: Wir haben bei diesem Blatt, sowie bei jedem Blatt, eng mit der Gruppe 11 zusammengearbeitet. Etwaige Übereinstimmungen der Lösungen sind auf diese Zusammenarbeit zurückzuführen.