

# Labreport #5

Patrick Eickhoff, Alexander Timmermann

## Aufgabe 1: Netzwerkeinstellungen

### 2. – Client-VM

**IP-Adresse:** 192.168.254.44

**Gateway:** 192.168.254.2

**Nameserver:** 10.1.1.1

### – Router-VM

**IP-Adresse eth0:** 172.16.137.222

**IP-Adresse eth1:** 192.168.254.2

### – Server-VM

**IP-Adresse:** 172.16.137.144

## Aufgabe 2: Absichern eines Einzelplatzrechners mit iptables

1. Auf der Client-VM sind keine iptables-Regeln vorhanden, die man löschen könnte. Zum Löschen könnte man sonst folgende Befehle benutzen:

# iptables -F	# flush chains in 'filter' table
# iptables -t nat -F	# flush chains in 'nat' table
# iptables -t mangle -F	# flush chains in 'mangle' table
# iptables -X	# delete custom chains

Mit

# apt-get update
# apt-get install openssh-server

installieren wir den OpenSSH Server.

2. Um das Surfen auf Webseiten zu erlauben, müssen wir den Datenverkehr über die Ports 80 (HTTP), 443 (HTTPS) und 53 (DNS) der RouterVM erlauben:

# iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
# iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
# iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT
# iptables -A INPUT -p udp --dport 53 -j ACCEPT
# iptables -A INPUT -p tcp --dport 80 -j ACCEPT
# iptables -A INPUT -p tcp --dport 443 -j ACCEPT

Desweiteren wollen wir sowohl als ICMP-Nachrichten senden und empfangen, als auch SSH-Verbindungen (Port 22) aufbauen können:

```
# iptables -A INPUT -p icmp -j ACCEPT
# iptables -A INPUT -p tcp --dport 22 -j ACCEPT
#
# iptables -A INPUT -p udp --dport 53 -j ACCEPT
# iptables -A INPUT -p tcp --dport 80 -j ACCEPT
# iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

Desweiteren wollen wir sowohl als ICMP-Nachrichten senden und empfangen, als auch SSH-Verbindungen (Port 22) aufbauen können:

```
1 iptables -A INPUT -p icmp -j ACCEPT
2 iptables -A INPUT -p tcp --dport 22 -j ACCEPT
3
4 iptables -A OUTPUT -p icmp -j ACCEPT
5 iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT
```

Letzendlich wollen wir jeglichen anderen Traffic unterbinden:

```
1 iptables -A INPUT -j REJECT
2 iptables -A OUTPUT -j REJECT
```

3. Die SSH-Verbindung von der ClientVM auf die RouterVM (`ssh user@192.168.254.2`) wird verweigert ("refused"), während die Verbindung von RouterVM auf ClientVM (`ssh user@192.168.254.1`) problemlos funktioniert.

Per `nc -l 5555` setzen wir einen Server auf der ClientVM auf. Wenn wir diesen jedoch von der RouterVM mit `nc 192.168.254.2 5555` ansprechen wollen, wird die Verbindung verweigert ("refused").

Wenn wir statt REJECT DROP für unsere Firewall verwenden, bekommen wir bei einem Verbindungsversuch keine Refused-Nachricht mehr zurück, da die Firewall das Packet einfach ohne weitere Aktion verwirft, statt eine Abweisungsnachricht zurückzusenden.

4. Mithilfe dynamischer Regeln können wir einfach definieren, dass ein- und ausgehende Pakete, die zu bereits etablierten Verbindungen gehören (ESTABLISHED,RELATED), automatisch akzeptiert werden:

```
# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Die restlichen Regeln definieren sich dann wie folgt:

```
# iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
# iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
# iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT
# iptables -A INPUT -p tcp --dport 22 -j ACCEPT
# iptables -A INPUT -p icmp -j ACCEPT
# iptables -A INPUT -j REJECT
```

Dynamische Regeln sind sehr angenehm, da sie erlauben Pakete abhängig von ihrem Zustand zu behandeln. So werden deutlich weniger Regeln benötigt, um die Kommunikation bereits aufgebauter Verbindungen zu erlauben.

## Aufgabe 3: Absichern eines Netzwerks

1. Folgenden Befehl haben wir in der `rc.local` auf der RouterVM gefunden:

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

Die *NAT-Tabelle* gibt die Firewall-Regeln für Pakete an, die neue Verbindungen aufbauen wollen. Die Postrouting-Chain behandelt Pakete nachdem sie geroutet wurden. Der Befehl maskiert, bildet die Source-IP, der Pakete, die aus dem internen Netzwerk von Ip-Adressen 192.168.254.0/24 gesendet und über das Interface eth0 geroutet wurden, auf eine interne IP-Adresse ab.

- Die ServerVM von der ClientVM anzupingen funktioniert problemlos, da die Pakete über die RouterVM per Masquerading weitergeleitet werden. Wenn dann die ICMP-Antwort an der RouterVM ankommt, wird die Destination-IP auf die zugehörige IP im internen Netzwerk abgebildet. Die ClientVM können wir jedoch nicht von der ServerVM direkt ansprechen, da die das Netzwerkinterface der ClientVM nur mit dem Netz unter der RouterVM verbunden ist. Da wir das interne Mapping der RouterVM nicht kennen, können wir auch nicht die gemappte IP-Adresse der ClientVM ansprechen.

- Die minimale Firewallkonfiguration sollte wie folgt aussehen:

```
# iptables -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
# iptables -A FORWARD -p tcp -d 172.16.134.144 --dport 80 -j DROP
# iptables -A FORWARD -p tcp -d 10.1.1.2 -j DROP
# iptables -A FORWARD -p tcp -d 10.0.0.0/8 -j DROP
# iptables -A FORWARD -i eth1 -o eth0 -p udp -m udp --dport 53 -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -p tcp -m tcp --dport 80 -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -p tcp -m tcp --dport 443 -j ACCEPT
# iptables -A FORWARD -j DROP
```

- Damit eine SSH-Verbindung aufgebaut werden kann, müssen wir Forwarding zu Port 22 erlauben:

```
# iptables -I FORWARD 2 -i eth1 -o eth0 --dport 22 -j ACCEPT
```

- Zum forwarden der SSH-Verbindung an die ClientVM brauchen wir nur eine Regel:

```
1 iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 5022 -j DNAT \
2     --to-destination 192.168.254.44:22
```

Die Pakete die an Port 5022 des äußeren Interfaces eth0 eingehen, werden vorm Routing zur ClientVM (192.168.254.44) auf Port 22 weitergeleitet.

- Nachdem wir eth0 eine weitere IP-Adresse 172.16.137.223 eingerichtet haben, geben wir an, dass alle Pakete, die an diese IP-Adresse geschickt werden, zur ClientVM weitergeleitet werden:

```
1 iptables -t nat -A PREROUTING -i eth0 -d 172.16.137.223 -j DNAT \
2     --to-destination 192.168.254.44
```

## Aufgabe 4: SSH-Tunnel

- Als erstes konfigurieren wir die Firewall:

```
# iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -p udp --dport 53 -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 22 -j ACCEPT
# iptables -A FORWARD -j DROP
```

- Ein SSH-Tunnel lässt sich einfach über den ssh-Befehl aufbauen<sup>1</sup>:

---

<sup>1</sup>Quelle: <http://blog.trackets.com/2014/05/17/ssh-tunnel-local-and-remote-port-forwarding-explained-with-examples.html>

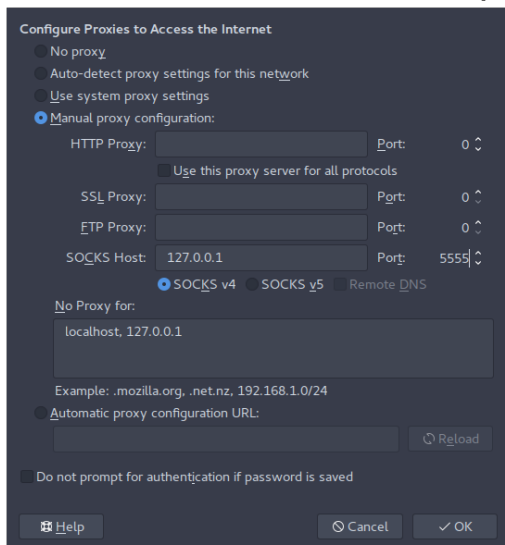
```
$ ssh -L 9000:localhost:80 user@172.16.137.144
```

Mit obigem Befehl binden wir Port 80 der ServerVM an Port 9000 der ClientVM per SSH-Verbindung. Dann lässt sich über *localhost:9000* auf den Webserver der ServerVM zugreifen. Über Wireshark sehen wir nur die SSH-Pakete.

3. Local Forwarding zum Browsen wäre sehr aufwendig, da jeder Port, den wir zum Surfen brauchen, einzeln geforwarded werden muss. Stattdessen benutzen wir Dynamic Forwarding, um unsere SSH-Verbindung als SOCKS-Proxy zu verwenden<sup>2</sup>:

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

Im Browser müssen wir nun den Proxy konfigurieren:



4. Zuerst starten wir auf der ClientVM einen Server mittels Netcat `nc -L 5555` und bauen dann unseren SSH-Tunnel zum Remote-Forwarding auf

```
$ ssh -R 9000:localhost:5555 user@172.16.137.144
```

Nun können wir uns auf der ServerVM mittels `nc localhost 9000` verbinden.

## Aufgabe 5: OpenVPN

1. Filterregeln:

```
# iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
# iptables -A FORWARD -i eth0 -o eth1 -j DROP
```

2. Der Server und Client sind wie in der *client.conf* (App. A) und *server.conf* (App. B) angegeben, konfiguriert.
3. Die Firewall erlaubt Pakete die Port 1194 für OpenVPN ansteuern und Pakete die vom Interface *tun0*, welches von der VPN genutzt wird, eingehen. Ausgehen dürfen nur Pakete die zu einer bereits etablierten VPN-Verbindung gehören:

<sup>2</sup>Quelle: <https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding>

```
# iptables -A INPUT -i tun0 -j ACCEPT
# iptables -A INPUT -p tcp --dport 1194 -j ACCEPT
# iptables -A INPUT -j DROP
# iptables -A OUTPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
# iptables -A OUTPUT -j DROP
```

- Wir starten erst VPN-Client und -Server: `openvpn client.conf` und `openvpn server.conf`. Auf der ClientVM sehen wir dann bei erfolgreicher Verbindung:

```
Peer Connection Initiated with [AF_INET]172.16.137.222:54286
Initialization Sequence Completed
```

Auf der ServerVM respektive:

```
Peer Connection Initiated with [AF_INET]172.16.137.144:1194
Initialization Sequence Completed
```

- Für den Tunnel der VPN wird das Interface `tun0` verwendet. Der Webserver der ServerVM lässt sich einfach im Browser über `10.8.0.1:80` aufrufen. Der Aufbau einer SSH-Verbindung ist möglich, so lange er über das VPN erfolgt. Der Client kann in unserer Konfiguration unter `10.8.0.2` erreicht werden, also:

```
$ ssh user@10.8.0.2
```

## Aufgabe 6: HTTP-Tunnel

- Firewall:

```
# iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 80 -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth0 -p udp --dport 53 -j ACCEPT
```

- Es gibt mehrere Wege, die Firewall zu überlisten.

Ein Weg ist es, Pakete die bei der ServerVM auf Port 80 eingehen, an Port 22 Weiterleiten:

```
# iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 22
```

Eine andere Möglichkeit ist, `sshd` auf der ServerVM auf Port 80 lauschen zu lassen, indem man in `/etc/ssh/sshd_config` die Definition `Port 22` durch `Port 80` ersetzt. Nun muss man `sshd` neustarten und Apache, was bisher Port 80 belegt, stoppen:

```
# service apache2 stop
# service sshd restart
```

- Firewall:

```
# 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
```

4. In Firefox müssen wir als HTTP-Proxy nur *192.168.254.2:3128* einstellen.

5. Als erstes verbinden wir uns über Netcat mit dem SSH-Server der ServerVM:

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

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

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

6. Wir überlisten die Firewall einfach, indem wir Pakete die bei der ServerVM auf Port 80 eingeht an Port 22 Weiterleiten:

```
1 iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 22
```

7. Firewall:

```
1 iptables -A INPUT -i eth1 -p tcp --dport 3128 -j ACCEPT
2 iptables -A INPUT -i eth1 -j DROP
3 iptables -A FORWARD -j DROP
4 iptables -A OUTPUT -o eth0 -p udp --dport 53 -j ACCEPT
5 iptables -A OUTPUT -o eth0 -p tcp --dport 80 -j ACCEPT
6 iptables -A OUTPUT -o eth0 -p tcp --dport 443 -j ACCEPT
7 iptables -A OUTPUT -o eth0 -j DROP
```

8. In Firefox müssen wir als HTTP-Proxy nur *192.168.254.2 3128* einstellen.

9. Als erstes verbinden wir uns über Netcat mit dem SSH-Server der ServerVM:

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

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

```
1 Host *
2   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
```

10. 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`

## Anhang A Server-Config

Listing 1: server.conf

```
1 port 1194
2 dev tun
3 ifconfig 10.8.0.1 10.8.0.2
4 secret static.key
5 keepalive 10 120
6 verb 3
```

## Anhang B Client-Config

Listing 2: client.conf

```
1 dev tun
2 remote 172.16.137.144 1194
3 ifconfig 10.8.0.2 10.8.0.1
4 secret static.key
5 verb 3
```