Zehn SSH-Tricks

Der Vortrag behandel das Programm `ssh` aus der OpenSSH-Suite, mit dem man verschlüsselte Verbindungen zu anderen Rechnern aufbauen kann. Über diese verschlüsselte Verbindung kann man per Shell (Kommandozeile) den Rechner administrieren, aber auch Dateien übertragen, Verbindungen tunneln und VPNs (Virtual Private Networks) aufsetzen. Während des Vortrags bespreche ich jedes der Features theoretisch und führe es dann live vor, bzw. zeige weitere Anwendungsbereiche auf.

Vorkenntnisse: Der Zuhörer sollte nach Möglichkeit schon einige Male SSH benutzt haben. Ideal zum ausprobieren der gezeigten Szenarien ist es, wenn der Zuhörer Zugriff auf einen Rechner außerhalb des Netzes der TU Chemnitz hat (beispielsweise einen eigenen Server oder einen zu Hause stehenden Rechner, der über ein DynDNS-Alias auf Port 22 erreichbar ist).

Weitere Informationen: OpenSSH-Webseite: `http://openssh.org/`
Die Manual-Seiten geben einen sehr guten Überblick über die Fähigkeiten von SSH:
`http://man.cx/ssh`
`http://man.cx/ssh_config`
`http://man.cx/sshd_config`

Überblick

1) **Interaktive Kommandos:** Während eine SSH-Verbindung besteht, werden alle Tastatureingaben weitergeleitet. Über sogenannte Escape-Sequenzen ist es aber möglich, SSH selbst zu steuern.
2) **Dateien kopieren:** Per `scp` (secure copy, Unterprogramm von SSH) kann man Dateien und Verzeichnisse übertragen. Entfernte Verzeichnisse einzubinden per wird per `sshfs` bzw. `shfs` ermöglicht.
3) **Kompression:** Gerade bei langsamen Leitungen sehr von Vorteil (für die Übertragung von Plaintext ca. 50% Datenreduktion).
4) **Connection Sharing:** Die Einstellung einer *Master-Connection* erlaubt es SSH, mehrere Dateien/Shells über eine einzige Verbindung zu übertragen/öffnen.
5) **X11-Forwarding:** Es ist möglich, X11-Programme auf einem Rechner laufen zu lassen, die grafische Ausgabe aber auf einem anderen Rechner anzeigen zu lassen. (Sehr langsam!)
6) **Local-Port-Forwarding:** Verbindungen, die auf einem bestimmten lokalen Port eingehen, werden vom Server an einen beliebigen anderen Rechner weitergeleitet. (⟶ HTTP-Proxy)
7) **Remote-Port-Forwarding:** Genau wie oben, nur dass die Verbindung auf den Computer, von dem die SSH-Verbindung ausgeht, getunnelt wird.
8) **Dynamic Forward:** SSH imitiert einen SOCKS4/5-Server, der eingehende Verbindungen über den Server tunnelt. `tsocks` erlaubt auch Programmen, die nativ kein SOCKS unterstützen, Verbindungen zu einem SOCKS-Server aufzubauen.
9) **Encrypted Tunnel:** SSH erstellt je ein `tun`-Device auf der Client- und Serverseite, die verschlüsselt beliebigen Level-2- bzw. Level-3-Netzwerk-Traffic austauschen können (PPP bzw. Ethernet/IP).
10) **Poor Man's VPN:** Über einen solchen verschlüsselten Tunnel kann nun ein VPN aufgebaut werden, indem die Routing-Tabellen manipuliert werden und evtl. IPTables-Regeln erstellt werden.

Julius Plenz <julius@plenz.com>

Zehn SSH-Tricks – Eingegebene Befehle

Im Folgenden eine kurze Sammlung der Befehle, die ich eingegeben habe:

1) **Interaktive Kommandos:**
   Eingabe der Sequenzen erst nach einem [Return] wirksam:

```
Supported escape sequences:
~.  - terminate connection
~B  - send a BREAK to the remote system
~C  - open a command line
~R  - Request rekey (SSH protocol 2 only)
~^Z - suspend ssh
~#  - list forwarded connections
~&  - background ssh (when waiting for connections to terminate)
~?  - this message
~~  - send the escape character by typing it twice
(Note that escapes are only recognized immediately after newline.)
```

2) **Dateien kopieren:**
   `scp` *`host`*`:file.txt .` (Datei herunterladen)
   `scp bild.jpg programm.c` *`host`*`:` (Dateien hochladen)

   `sshfs` *`user@host:[directory] mountpoint`*
   (*directory* (falls nicht angegeben, $HOME) in *mountpoint* einbinden. (fuse-Kernelmodul und libfuse benötigt, mehr Informationen unter http://fuse.sourceforge.net/sshfs.html)

3) **Kompression:**
   `scp` **`-C`** *`user@host`*

4) **Connection Sharing:**

```
ControlMaster   auto
ControlPath     ~/.ssh/master-%r@%h:%p
```

5) **X11-Forwarding:**
   `ssh -X` *`host`* `xlock &` (entfiel)

6) **Local-Port-Forwarding:**
   `ssh -NL 8118:localhost:8118` *`host`* (Web-Proxy auf localhost:8118 einstellen)
   `LocalForward 8118 localhost:8118` (Config-Eintrag)

7) **Remote-Port-Forwarding:** `ssh -NR 8080:localhost:80` *`host`* (Port 8080 auf *host* leitet auf localhost:80 weiter)
   `RemoteForward 8080 localhost:80` (Config-Eintrag)

8) **Dynamic Forward:**
   `ssh -D 2323` *`host`* (SOCKS-Proxy auf localhost:2323 starten)
   `DynamicForward 2323` (Config-Eintrag)

Julius Plenz <julius@plenz.com>

9) **Encrypted Tunnel:**

Lokal	Remote
`sudo ssh -vw 0:any host true`	
`sudo ifconfig tun0 up 10.0.23.2 \` `netmask 255.255.255.252`	`sudo ifconfig tunX up 10.0.23.1 \` `netmask 255.255.255.252`
`ping 10.0.23.1`	`ping 10.0.23.2`
`nc -v 10.0.23.1 3434`	`nc -vlp 3434`

Mit den letzten beiden `nc`-Kommandos kann man bspw. beliebige Daten über den Tunnel auf Port 3434 schicken.

Damit dies Funktioniert, muss die Direktve `PermitTunnel` auf `yes` gesetzt werden (in der sshd-Config des Servers).

10) **Poor Man's VPN:**
So würde man ein VPN zwischen kreieren, so dass alle Daten vom lokalen Rechner *immer* über den entfernten Rechner getunnelt werden:

Lokal	Remote
`route add -host host gw GW`[1] `dev eth0`	`iptables -t nat -A POSTROUTING \` `-s 10.0.23.2 -j SNAT \` `--to-source IP_des_Servers`
`route add -net 0.0.0.0/0 dev tun0`	`iptables -A FORWARD -d 10.0.23.2 \` `-j ACCEPT`

[1] Hier muss das Gateway des lokalen Netzes eingetragen werden, also beispielsweise die IP, die der DHCP-Server verteilt hat. nur so können noch Verbindungen nach aussen aufgebaut werden.