SSH (Secure Shell)

Modul 346, BBZW

Patrick Bucher

Problem

Zwei (entfernte) Rechner wollen miteinander kommunizieren.

- Fernwartung eines Servers
- Zugriff auf zentrale Ressourcen
- Peer-to-Peer-Dateiübertragung

Lösungsansatz 1: RDP (Remote Desktop Protocol)

Man verbindet sich mit dem Desktop eines anderen Rechners.

ermöglicht:

- Fernwartung (Jump Host)
- Ressourcenzugriff
- Dateiübertragung

Problem:

- proprietäre Lösung von Microsoft
- nur für Zugriff auf Windows
  - Clients für Linux, macOS usw. existieren
- erfordert einen Desktop (grafische Oberfläche)
- erschwert/verunmöglicht Automatisierung

Server mit GUI?

Windows XP (2001) *mit* Desktop:

- 512 MB RAM
- 1500 MHz CPU
- Anwendungen: Office 2000, Return to Castle Wolfenstein

Debian 10 “Buster” (2019) *ohne* Desktop:

- 512 MB RAM
- 2198 MHz CPU
- Anwendungen: Gitea (inkl. PostgreSQL & nginx)
  - code.frickelbude.ch

Verzichten wir auf Servern besser auf ein GUI!

Lösungsansatz 2: Telnet

Man erstellt eine interaktive, *textorientierte* Verbindung zu einem anderen Rechner.

```
$ telnet [address] [port]
```

```
$ telnet www.whatever.xy 80
GET /index.html HTTP/1.1
...
```

Problem:

- unsicher: Datenverkehr wird nicht verschlüsselt
- veraltet: wird kaum mehr weiterentwickelt

Man kann `telnet` immer noch als manuellen Port-Scanner einsetzen.

Lösungsansatz 3: SSH (Secure Shell)

Abbildung 1: OpenSSH (Quelle: openssh.com)

SSH

- Authentifizierung mittels Schlüsselpaar
  - Client und Server haben je ein Schlüsselpaar
  - asymmetrische Verschlüsselung
  - `ssh-keygen -t ed25519 -C joe.doe@acme.org`
- Verschlüsselung der Kommunikation
  - Client und Server verhandeln einen Schlüssel
  - symmetrische Verschlüsselung
  - interaktive Sitzung: `ssh [user]@[host] -p [port]`
  - einzelner Befehl: `ssh [user]@[host] -p [port] '[command]'`
- Client/Server-Prinzip
  - `ssh`: Client
  - `sshd`: Server (Daemon)

Anwendungsfälle

- Fernwartung
- Arbeit auf einem entfernten System
- Dateiübertragung: `scp`, `sftp`
- Jump Host: Client -> Client/Server -> Server
- SSH-Tunnel:
  - Local Forwarding: Client-Aufrufe an Server weiterleiten
  - Remote Forwarding: Server-Aufrufe an Client weiterleiten
  - VPN

Fernwartung

Aktualisierung von Debian-Packages:

```
$ ssh code.frickelbude.ch 'sudo apt update -y && sudo apt upgrade -y'
```

Neustarten von Gitea:

```
$ ssh code.frickelbude.ch 'sudo systemctl restart gitea.service'
```

Dateiübertragung per scp

Gitea-Backup ins persönliche `$HOME`-Verzeichnis kopieren:

```
$ ssh code.frickelbude.ch 'sudo cp /backup/weekly.tar.gz ~/'
$ ssh code.frickelbude.ch 'chown patrick:patrick ~/weekly.tar.gz'
```

Dateien herunterkopieren:

```
$ scp code.frickelbude.ch:weekly.tar.gz .
```

SSH-Tunnel

SSH-Tunnel können sehr nützlich sein – aber auch ein Sicherheitsrisiko!

Zu Risiken und Nebenwirkungen fragen Sie den Sysadmin Ihrer Organisation.

Local Forwarding

Auf einen Webserver (`srv`) läuft eine Anwedung auf dem Port `8080`

Port `8080` ist aber *nicht* freigegeben.

- **Problem**: Auf die Anwendung kann von aussen nicht zugegriffen werden.
- **Lösung**: SSH Local Forwarding

Local Forwarding: Umsetzung

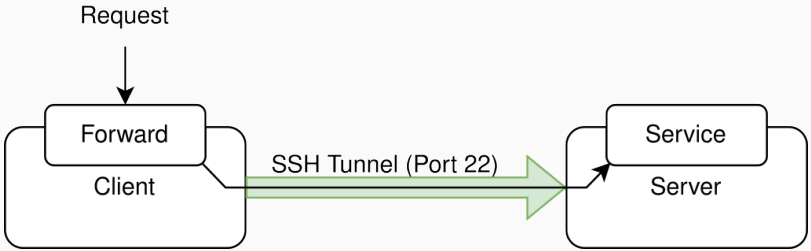

Abbildung 2: Local Forwarding

```
$ ssh -L localhost:1234:srv:8080 user@srv
```

Der lokale Request auf Port `1234` wird zum Server auf Port `8080` umgeleitet.

Remote Forwarding

Auf einem Client (`localhost`) läuft eine Anwendung auf dem Port `1234`.

Der Server (`srv`) kann nicht auf die Anwendung auf dem Client zugreifen.

- **Problem**: Der Server kann keine Verbindung zum Client erstellen.
- **Lösung**: SSH Remote Forwarding

Remote Forwarding: Umsetzung

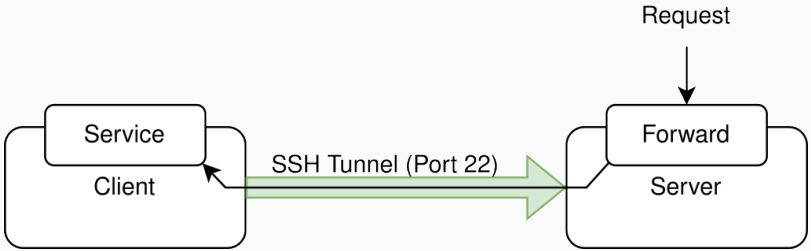

Abbildung 3: Remote Forwarding

```
$ ssh -R srv:8080:localhost:1234 user@srv
```

Der Server-Request auf Port `8080` wird zum lokalen Computer auf Port `1234` umgeleitet.