# SSH

## Die Secure Shell am Beispiel von OpenSSH

Dirk Geschke

Linux User Group Erding

26. Oktober 2011

# Gliederung


1. Historisches
2. Details
3. Keys
4. SSH-Optionen
5. Praxis
6. Sonstiges

# Geschichte

- Verschlüsselte Alternative für `telnet`, Ersatz für Berkeley-Tools `rlogin, rcp, rsh`
- 1995 von Tatu Ylönen als Freeware erstellt, er gründete *SSH Communications Security*, heute heißt die Firma *Tectia*.
- SSH wurde kommerziell, es gab keine freien Sourcen mehr.
- 1999 entstand basierend auf der letzten freien Version das Projekt *OpenSSH* mit einem Open-Source-SSH.
- SSH-Protokoll Version 1 hatte Sicherheitsprobleme, daher entstand Version 2
- OpenSSH Version 2 ist eine reine Neu- und Eigenentwicklung.

# Überblick

- Anwendungen von SSH sind
  - sichere Administration von Systemen
  - sicheres Ausführen von Kommandos auf entfernten Systemen
  - sicherer Datentransfer
  - sichere Verbindungstunnel: Absicherung von Kommunikationswegen
- basiert auf *public-key*-Verfahren zur Aushandlung der Verschlüsselung
- Authentisierung über viele Verfahren: wichtigste sind per Passwort und SSH-Key
- Tunneln von *X11*–Anwendungen leicht möglich

# Bedeutung des Host-Keys

- Wichtig ist der *Host-Key*: Über diesen authentisiert sich der Server!
- Übertragen wird der *Public-Key*, dieser wird mit dem lokal gespeicherten für diesen Server verglichen.
- Existiert dieser nicht, wird ein *Fingerprint* gezeigt und nachgefragt, ob dieser akzeptiert wird:

```
$ ssh vserver.lug-erding.de
The authenticity of host 'vserver.lug-erding.de
(62.75.188.56)' can't be established.
RSA key fingerprint is
5a:71:7b:5c:65:3d:fb:9b:44:98:48:2c:00:18:1b:b7.
Are you sure you want to continue connecting (yes/r
```

- Warnung bei geändertem Key: Vorsicht ist geboten Man-In-The-Middle–Angriff?

# Bedeutung des SSH-Keys

- Authentisierung per Passwort ist meist unsicher: Wahl des Passworts zu einfach!
- Bessere Lösung: Verwendung von SSH-Keys.
- Erzeugung mittels `ssh-keygen`, Wichtig die Zahl der Schlüsselbits, RSA oder DSA, Passphrase zum Schutz des *Private-Keys*
- *Public-Keys* muss auf Server hinterlegt werden, gewöhnlich in `~/.ssh/authorized_keys`
- Verteilung entweder manuell oder automatisch mit Hilfe von `ssh-copy-id`

# Verwaltung der SSH-Keys

- Automatische Verwaltung der SSH-Keys mit Hilfe von `ssh-agent`
- Keys werden mit `ssh-add` hinzugefügt, es wird nach der Passphrase gefragt und die SSH-Programme verwenden dann diese Keys.
- Keys können per *ssh-agent-forwarding* weitergeleitet (*mitgenommen*) werden, Option `-A`.
- Es werden alle geladenen Keys versucht, danach wird gewöhnlich nach dem Passwort gefragt.
- Vorsicht: Bei mehreren Keys werden alle durchprobiert, mitunter schließt der Server die Verbindung nach x-Versuchen.

# Wichtige SSH-Optionen

- Können auf der Kommandozeile angegeben werden oder per Konfigurationsdatei
- Dateien sind:
  1. Angabe per Kommandozeile: `-C` *Datei*
  2. `~/.ssh/config`
  3. `/etc/ssh/ssh_config`
- Wichtige Parameter für Konfigurationsdatei
  - Es gibt globale Einstellung und pro System
  - `Host` *Hostname* als Einleitung für bestimmtes Ziel
  - Danach sind zielspezifische Angaben möglich.

# Wichtige SSH-Optionen

- `Protocol` (`-1` oder `-2`)
- `Cipher` (`-c `), z.B. `blowfish` oder `arcfour`
- `Port` (`-p`)
- `Compression` (`-C `)
- `LocalForward` (`-L`) oder `RemoteForward` (`R`)
- `User` (`-l` oder *user@<hostname>*)
- `AgentForwarding` (`-A`), `ForwardX11` (`-X`)
- `LogLevel` (`-v`): Debugging, mehrere `-v` möglich (max 3)
- `ServerAliveInterval`: Senden von *Keep-Alives* innerhalb der SSH-Verbindung!
- Optionen auch mit `-o` direkt angebbar.

# Wichtige SSH-Server-Optionen

- `AllowTcpForwarding`, `PermitOpen`
- `AllowUsers`, `DenyUsers`, `AllowGroups`, `DenyGroups`, `PermitRootLogin`
- `ClientAliveInterval`
- `ForceCommand`: Festlegen des Kommandos, dass beim Einloggen ausgeführt wird
- `ListenAddress` *host:port*, `Port`
- `UseDNS`, versucht den DNS-Namen zur IP-Adresse des Clients zu finden.
- `LogLevel` oder `-d`

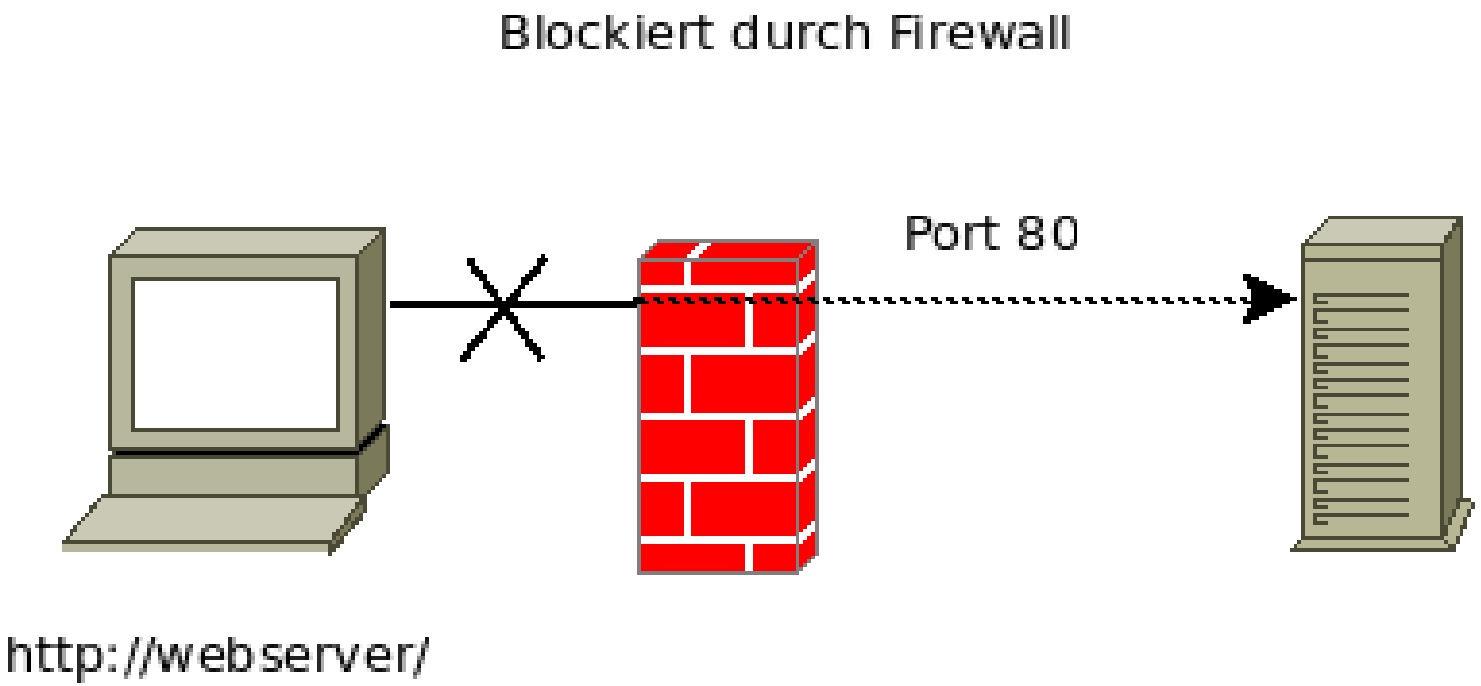
# Praktisches Vorgehen

- Einloggen per ssh: `ssh` *Servername*
- Ausführen von Kommandos: `ssh` *Servername Kommando*

```
$ ssh majestix ¨uname -r¨
2.6.32-5-686
```

- Datenübertragung mit scp: `scp` *Datei Servername*:*Pfad*
- oder umgekehrt: `scp` *Servername*:*Pfad Datei*
- `-r` hilfreich: *recursives* kopieren von Verzeichnissen
- `-p` hilfreich: erhält Zeitstempel, (großes `-P` für Port)
- Cipher / Compression beschleunigen mitunter, aber Vorsicht!
- `sftp` funktioniert analog zu `ftp`, nicht zu verwechseln mit *ftps*!

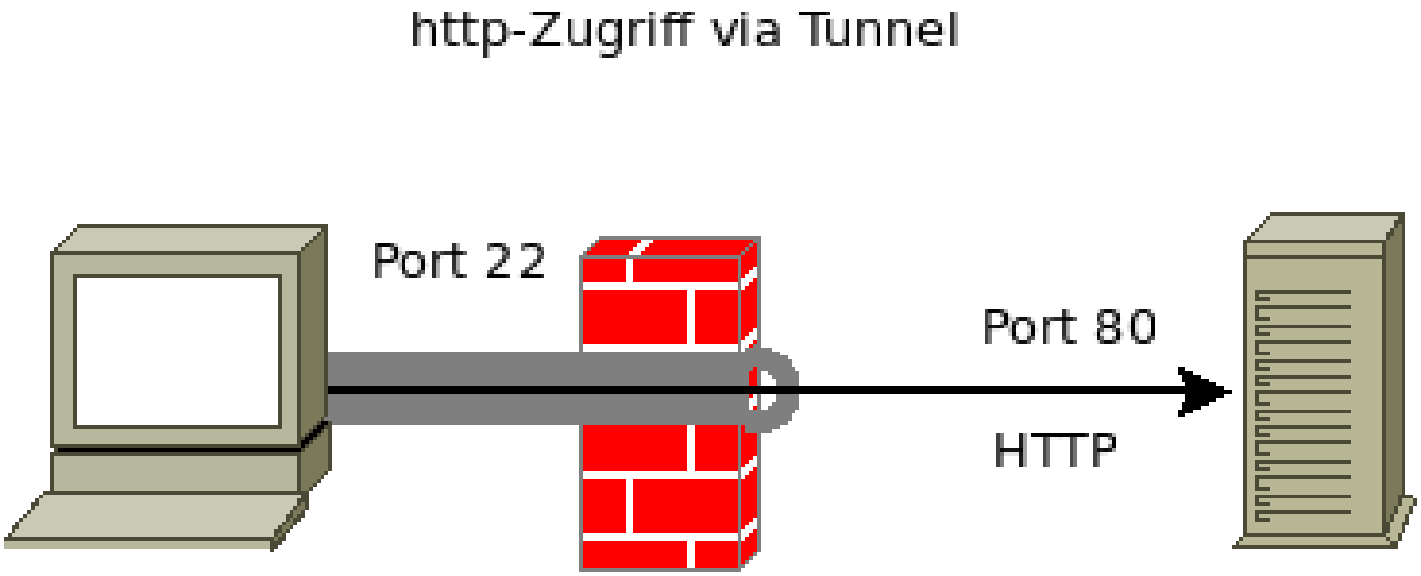
# Vorwärts-Tunnel: Zugriff auf internen Webserver



# Vorwärts-Tunnel: Zugriff auf internen Webserver


ssh Firewall mit Tunnel

# Vorwärts-Tunnel: Zugriff auf internen Webserver




Verbindung mit localhost

# Vorwärts-Tunnel: Zugriff auf internen Webserver

# Beispiel eines Local Forwardings

**ssh -L 1234:Tunnelziel:80 Servername**

- ssh-Verbindung zum Server
- Erstellung des Tunnels mit der Option `-L`
- Angabe des lokalen Ports, hier `1234`
- Angabe des Zielservers
- sowie des Zielports
- Nach dem Einloggen kann über das Ziel `localhost:1234` der Server `Tunnelziel` auf Port `80` erreicht werden.
- Die Verbindung bis zum SSH-Server `Servername` ist dabei verschlüsselt!

# Beispiel eines Reverse Forwardings

**ssh -R 1234:Tunnelziel:80 Servername**

- ssh-Verbindung zum Server
- Erstellung des Tunnels mit der Option `-R`
- Angabe des remote Ports, hier `1234`
- Angabe des Zielservers
- sowie des Zielports
- Nach dem Einloggen kann auf dem SSH-Server das Ziel `localhost:1234` der Server `Tunnelziel` auf Port `80` erreicht werden.
- Die Verbindung zum SSH-Server `Servername` ist dabei verschlüsselt!

# Besonderheiten

- Es können mehrere Tunnel parallel geöffnet werden.
- Tunnel können verkettet werden, also über mehrere Server gehen
- Lokale Ports können mit der Option `-g` anderen freigegeben werden
- Beachte: Den Tunnel kann jeder verwenden der Zugriff auf den Port hat!

# Weitere Funktionen

- SOCKS4/SOCKS5-Proxys sind möglich
- VPN per `tun`-Interface ist möglich
- Mit `sshfs` kann sicher auf entfernte Verzeichnisse mit üblichen Unix-Tools zugegriffen werden.
- `ssh-argv0` kann verwendet werden um das Ziel im Link-Namen festzulegen
- `ssh-keyscan` kann *Public-Keys* einsammeln
- `ssh-keygen` kann mit der Option `-l` Fingerprints von Keys ausgeben.

# Andere Implementierungen

- PuTTY ist eine mächtige GUI-Version die sowohl unter Windows als auch Linux verfügbar ist.
- OpenSSH ist via Cygwin auch unter Windows verwendbar
- WinSCP ist ein praktische SCP-Alternative für Windows
- Mit `sshfs` kann sicher auf entfernte Verzeichnisse mit üblichen Unix-Tools zugegriffen werden.
- `Webbasierte` SSH-Lösungen existieren auch wie z.B. Ajaxterm oder Anyterm
- Es existieren auch zahlreiche Lösungen für Smartphones und Tablets.
- `ssh-keygen` kann mit der Option `-l` Fingerprints von Keys ausgeben.

# Das war es schon!