Zurück zur Startseite

SSH - Secure SHell

Präsentation

Inhalte 2021

Standardfeature ist bekannt: verschlüsselte Remote-Shell, Aufruf mit `ssh username@host`

Dateien in ~/.ssh

`~/.ssh/config` enthält individuelle Einstellungen für den jeweiligen User auf diversen Hosts.

```
# Host * sind Defaulteinstellungen für alle Hosts
Host *
        IdentityFile ~/.ssh/id_ed25519
Host algol
        Hostname algol.cmkrueger.lan
        AddressFamily inet6
Host pihole
        User adminuser
        Hostname 10.0.12.105
        AddressFamily inet
        IdentityFile ~/.ssh/id_rsa
```

`~/.ssh/known_hosts` speichert die Host-Keys von bereits verbundenen Hosts. Früher in Klartext, heute in einem base64-kodierten Format.

```
|1|36mHoZFzfn561LnkAwWb8z3UcFA=|xqq9PBFD4W1FdyBcGaY1NigBm0o= ecdsa-sha2-
nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBFnHB9j85nHQ2KC0EA31QMYl
Mq22NO5P/qIZk51QSW1oWZCsMdEDCf2zzBJr/4LxVS7xWQ3xk+gUss5Sy+ZH3xc=
|1|onRNMPB7MA37Wx72yZRVi9Rv6UE=|NBvO2Zfvq2efJl7xDVfL/awk6Gk= ecdsa-sha2-
nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBFnHB9j85nHQ2KC0EA31QMYl
Mq22NO5P/qIZk51QSW1oWZCsMdEDCf2zzBJr/4LxVS7xWQ3xk+gUss5Sy+ZH3xc=
```

Um heutzutage einen veralteten Eintrag zu entfernen, kann man (überraschenderweise) `ssh-keygen -R hostname` verwenden.

Key-Auth statt Passwörtern mit authorized_keys

Statt Passwörtern kann man sich auch mit einem asymmetrischen Verschlüsselungsverfahren anmelden. Dazu muss man für den eigenen User einen SSH-Key generieren: `ssh-keygen -t ed25519`. Statt `ed25519` kann man auch `ecdsa`, `rsa` oder (veraltet) `dsa` verwenden. Die Key-Länge kann manuell mit `-b` festgelegt werden.

`ssh-keygen` erzeugt zwei Dateien in `~/.ssh`: einmal den eigentlichen Key `id_ed25519` und zusätzlich den öffentlichen Teil unter `id_ed25519.pub`.

Der öffentliche Schlüssel muss *im Zielhost beim Zieluser* in einer Zeile der `~/.ssh/authorized_keys` stehen. Dies kann man von Hand machen (nicht empfohlen) oder mittels `ssh-copy-id zieluser@zielhost`.

Passphrase und ssh-agent

Ein Key kann mit einer Passphrase versehen werden, die dann bei jeder Verwendung des Keys abgefragt wird. `ssh-keygen` fragt beim Erstellen standardmäßig nach einer Passphrase.

Um bei wiederholten Zugriffen nicht immer wieder die Passphrase eingeben zu müssen, kann diese für 15 Minuten (Defaulteinstellung) mit Hilfe von `ssh-agent` und `ssh-add` gecached werden.

```
eval `ssh-agent`
ssh-add
```

One-Shot-Sessions

Sobald Key-Auth aktiv ist, kann man eine SSH-Sessions, um einen einzigen Befehl auszuführen. Dabei funktioniert sogar die Filterung mit Pipes!

```
ssh zielhost uptime

ssh zielhost getent passwd | grep username

cat lokaledatei | ssh zielhost tee -a remotedatei
```

Dateien kopieren mit scp

`scp` (secure copy) arbeitet wie `cp`, bietet aber auch SSH-Remote-Ziele.

```
scp lokaledatei user@zielhost:/tmp/remotedatei
scp user@quellhost:/etc/passwd /tmp/lokalekopiederpasswd
```

Local Port Forward (Tunnel)

Mit `-L` kann man einen lokalen Port (d.h. beim SSH-Client) öffnen, der dann über die SSH-Verbindung verschlüsselt auf den SSH-Server weitergeleitet wird. Das Format ist `-L lokaleIP:lokalerPort:zielIP:zielPort`, wobei die lokale IP weggelassen werden kann.

Beispiele (beziehen sich darauf, dass man sich vom Rechner `lfbdebian` auf dem Rechner `ziel` einloggt

- `ssh -L 10.0.0.2:1234:127.0.0.1:80 user@ziel` → Leitet den Port 1234 auf dem Interface mit der IP 10.0.0.2 von `lfbdebian` über SSH so weiter, dass es für den Webserver auf `ziel` so aussieht, als ob die Verbindung von sich selbst kommt. Der Nutzer verbindet sich lokal zu 10.0.0.2 Port 1234 und landet auf `ziel` Port 80
- Das Ziel der Weiterleitung muss nicht der localhost sein. `ssh -L 44444:www.heise.de:80 user@ziel`. Wenn man sich auf `lfbdebian` (beliebige IP) auf Port 44444 verbindet, landet man beim Webserver auf `www.heise.de`. Dieser wiederum sieht die Verbindung von `ziel` aus kommend.

Merke: **lokale** Weiterleitung bedeutet, ein Port wird **lokal** (beim SSH-Client) geöffnet, und das Ziel ist aus der Remote-Sicht (beim SSH-Server) erreichbar.

Remote Port Forward (Tunnel)

Logischerweise das Gegenstück zur lokalen Weiterleitung :)

Hier gilt: **remote** Weiterleitung bedeutet, ein Port wird **remote** (beim SSH-Server) geöffnet, und das Ziel ist aus der lokalen Sicht (beim SSH-Client) erreichbar.

- `ssh -R 3.4.5.6:3333:127.0.0.1:22 user@ziel` bedeutet, auf dem SSH-Server (`ziel`) wird an der IP 3.4.5.6 der Port 3333 zum Lauschen geöffnet. Jeder der sich auf diesen Port verbindet, landet auf dem SSH-Server vom Rechner `lfbdebian` (der für die urspüngliche SSH-Verbindung als Client fungiert).
- `autossh -M 9998 -o “ServerAliveInterval 10” -o “ServerAliveCountMax 3” -o “ExitOnForwardFailure yes” -R 443:localhost:443 -R 80:localhost:80 -N cloudhetzner` öffnet auf `cloudhetzner` die Ports 80 und 443 und leitet diese auf den gleichen Port auf dem lokalen Host weiter. So ist ein interner Webserver über eine öffentliche IP erreichbar.

autossh mit automatischem Tunnel

`autossh` kann statt `ssh` verwendet werden, wenn die Verbindung immer wieder automatisch neu aufgebaut werden soll. Dies ist insbesondere sinnvoll, um einen automatischen Reverse-Tunnel aufgebaut zu lassen, z.B. als Ersatz für einen Reverse-Proxy oder als festen Zugangspunkt bei dynamischen IPs und/oder NAT.

Beispiel: der Host `internalhost` ist hinter NAT mit dynamischer IP, soll aber immer per SSH über den öffentlich erreichbaren Host `xyz.cmkrueger.de` auf Port 4321 erreichbar sein. Dazu reicht ein

per SSH mit Key-Auth erreichbarer Standarduser auf `xyz.cmkrueger.de` (der Port 4321 muss dazu frei sein). Die komplette Konfiguration findet auf `internalhost` statt!

```
# /home/internaluser/.ssh/config
Host xyz
  Hostname xyz.cmkrueger.de
  RemoteForward 4321 127.0.0.1:22
  ServerAliveInterval 30
  ServerAliveCountMax 2
  ExitOnForwardFailure yes
```

Damit sollte sich der `internaluser` via `ssh xyz` ohne Fehlermeldung und ohne Passphrase einloggen können.

Um das Ganze zu automatisieren, legt man eine Datei namens `/etc/systemd/system/autossh-tunnel-xyz.service` an:

```
[Unit]
Description=AutoSSH tunnel zu heka
After=network.target

[Service]
User=internaluser
Environment="AUTOSSH_GATETIME=0"
ExecStart=/usr/bin/autossh -M 0 -N xyz

[Install]
WantedBy=multi-user.target
```

Das war's! Bei Änderung der IP bekommt `xyz` einen Timeout und schließt die Verbindung, während `autossh` darauf achtet, dass die Verbindung von Clientseite bestehen bleibt.

Ab sofort ist der `internalhost` aus dem Internet via `ssh -p 4321 internaluser@xyz.cmkrueger.de` erreichbar. Von da kann man wiederum weitere Ports weiterleiten.

Key-Typen: dsa, rsa, ed25519, ecdsa

Allgemein gilt: wenn man sich da nicht genau auskennt, lieber die Defaults nehmen. Elliptische Kurven haben gegenüber RSA den Vorteil von gleicher Sicherheit bei deutlich reduzierter Key-Länge. 256 Bit EC gelten als mindestens so sicher wie 2048 Bit RSA [citation needed].

`ed25519` ist eine wissenschaftlich anerkannte elliptische Kurve, die von Dan J. Bernstein, Tanja Lange et al. entwickelt wurde und allgemeine Anerkennung besitzt. Siehe
https://ed25519.cr.yp.to/papers.html
Es kann sein, dass ältere SSH-Clients oder -Server kein ed25519 unterstützen. Dann ist `rsa` angesagt.

`ecdsa` ist DSA, nutzt aber statt langer Zahlen eine elliptische Kurve vom NIST, bei der (entgegen aller Wahrscheinlichkeit) gewisse Leute Bedenken haben, die NSA könnte da ihre Finger im Spiel haben.

`rsa` ist das klassische, bekannte RSA von Rivest, Shamir, Adleman. Hier sind lange Keys >= 2048 Bit nötig.

Remote X authentication

Mit `ssh -X user@host` werden austomatisch bestimmte Variablen gesetzt (u.a. XSESSION), damit auf dem Client grafische Anwendungen des Hosts angezeigt werden können. In der SSH-Session muss man dann einfach xterm, firefox oder was auch immer starten.

Parallel SSH

Mit `parallel-ssh` (Paket `pssh`) kann man Kommandos parallel auf mehreren Hosts ausführen lassen. Die Hosts müssen in einer Host-Liste stehen.

```
berta@lfbdebian:~$ cat ziele
sirprize@gibip.de
algol
berta@lfbdebian:~$ parallel-ssh -i -h ziele uptime
[1] 17:52:29 [SUCCESS] algol
 17:52:29 up 16 days,  4:01,  0 users,  load average: 0,00, 0,00, 0,00
[2] 17:52:29 [SUCCESS] sirprize@gibip.de
 15:52:29 up 105 days,  7:48,  0 users,  load average: 0.23, 0.22, 0.13
```

SSL/SSH Multiplexing mit sslh

siehe https://www.rutschle.net/tech/sslh/README.html

Statt Fail2ban: Tarpit mit endlessh

siehe https://github.com/skeeto/endlessh

Key-Auth mit Zertifikaten (PKI)

Achtung kommerzielles Produkt: https://smallstep.com/blog/use-ssh-certificates/

Zurück zur Startseite

From:
http://dwiki.jdsr.de/ - **wiki**

Permanent link:
http://dwiki.jdsr.de/doku.php?id=informationstechnik:ssh&rev=1728973919

Last update: **15/10/2024 06:31**