AutoSSH

- a cool solution to connect home

Author: Peter Murr
für Linux Usergroup Mittereich
April 2016

Linux User Group Mitterteich

http://linux-mitterteich.de

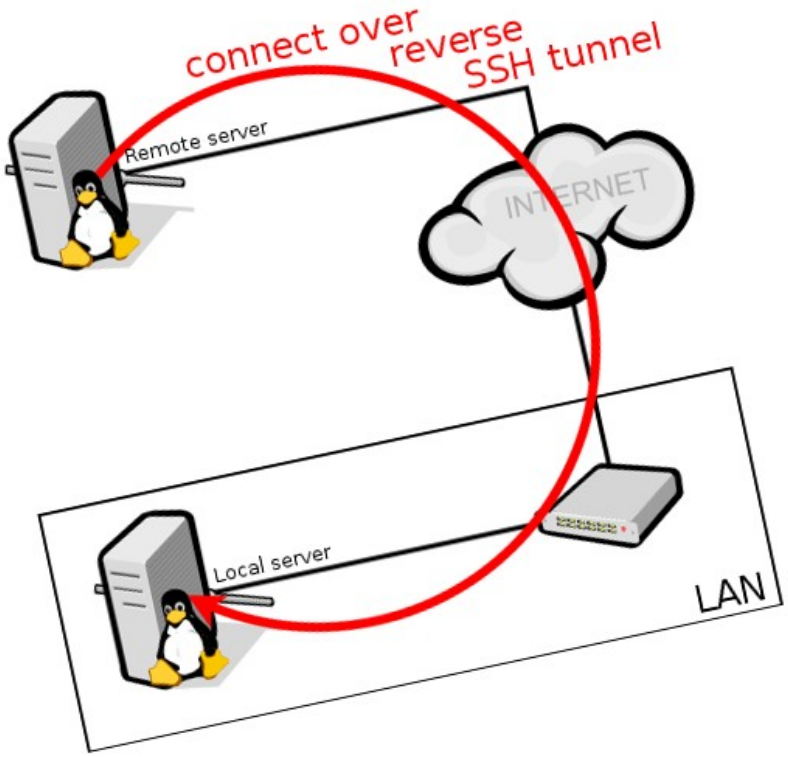
Inhaltsverzeichnis

Beschreibung

AutoSSH ist ein Programm welches einen SSH-Tunnel starten, überwachen und - sollte der Tunnel zusammenbrechen - wieder starten kann.
Die ursprüngliche Idee und der Mechanismus stammen von *rstunnel (reliable SSH Tunnel)*. Mit der Version 1.2 von AutoSSH wurde der Verbindungsmechanismus geändert: AutoSSH benutzt SSH um mehrere SSH-Weiterleitungen zu erzeugen (eine local -> remote, eine remote -> local) und schickt dann Testdaten, die es dann fehlerfrei zurückerwartet.
(Die Idee dazu hatte Terrence Martin - *Danke Martin*)

Ab Version 1.3, wurde wieder eine neue Methode eingeführt (*Danke an Ron Yorston*):
Es kann ein Port für den entfernten *ECHO-Dienst* angegeben werden, der dann die Testdaten zurücksendet. Das verhindert Staus und dass es auf der Remote-Maschine Port-Kollisionen gibt.

Die *"Ring-der-Weiterleitungen-Methode"* ist jedoch immer noch möglich für Situationen bei denen der *ECHO-Dienst* nicht eingesetzt werden kann.



Setup

Schritt 1 - Remote Server (z.B. Heimserver im LAN in der Wohnung - CentOS 7.2) - User für AutoSSH anlegen

Wir legen zunächst einen Benutzer *sshtunnel* mit Home-Verzeichnis an, der aber keine Shell *(/usr/sbin/nologin)* hat.
Das Home-Verzeichnis ist wichtig, da wir dort die SSH-Konfiguration (*known_hosts* und *authorized_keys*) für die Verbindung speichern werden.

```
[root@homeserver ~]# useradd -s /usr/sbin/nologin -m sshtunnel
```

Dieser User braucht grundsätzlich kein eigenes SSH-Schlüsselpaar, da er später eine ausgehende Verbindung aufbauen wird. Minimale Voraussetzung ist aber, dass es im Home-Verzeichnis des Benutzers *(/home/sshtunnel)* einen Unterordner *.ssh* gibt, also */home/sshtunnel/.ssh*, in dem wir später den öffentlichen SSH-Schlüssel des "Road-Warriors" hinterlegen können.

Es spricht aber nichts dagegen, ein Schlüsselpaar zu erzeugen (was automatisch auch den entsprechenden *.ssh* Unterordner im Heimatverzeichnis des Benutzers anlegt.

Wir beschränken uns aber auf das minimale Setup und legen nur den *.ssh* Unterordner an:

Benutzerkontext wechseln und .ssh Unterordner anlegen

```
[root@homeserver ~]# su -c "mkdir ~/.ssh" -s /bin/sh sshtunnel
```

Obiger Befehl wechselt temporär in den Benutzerkontext des Benutzers *sshtunnel* und legt mit diesem den Unterordner *.ssh* in dessen Heimatverzeichnis an. (Es gibt auch andere Möglichkeiten dies zu tun, aber so kann man als *root* alles in einem Befehl erledigen)

Schritt 2 - Lokale Maschine (z.B. Road Warrior) - User für AutoSSH anlegen und Schlüssel erzeugen

Auf der lokalen Maschine legen wir ebenfalls einen Benutzer *sshtunnel* mit Home-Verzeichnis, aber ohne shell an:

```
[root@roadwarrior ~]# useradd -s /usr/sbin/nologin -m sshtunnel
```

Nun wechseln wir durch nachfolgenden Befehl temporär in den Benutzerkontext des neu angelegten Benutzers *sshtunnel* und erzeugen ein RSA SSH-Schlüsselpaar mit dem Namen *tunnel_key_a*, einer Länge von 2048 bytes ohne Passwort im Ordner */home/sshtunnel/.ssh*.

(RSA sollte überall funktionieren, DSA Schlüssel werden seit OpenSSH 7.0 nicht mehr akzeptiert und ECDSA-Support ist noch zu neu!)

```
[root@roadwarrior ~]# su -c 'ssh-keygen -t rsa -b 2048 -q -N "" -f
~/.ssh/tunnel_key_a' -s /bin/sh sshtunnel
```

Das Anlegen des Schlüsselpaars prüfen wir kurz:

```
[root@roadwarrior ~]# ls -l /home/sshtunnel/.ssh/

insgesamt 8
-rw------- 1 sshtunnel sshtunnel 1675  5. Apr 14:16 tunnel_key_a
-rw-r--r-- 1 sshtunnel sshtunnel  400  5. Apr 14:16 tunnel_key_a.pub
```

Schritt 3 - Öffentlichen SSH-Schlüssels der lokalen Maschine auf den Remote Server in authorized_keys einpflegen

Wir müssen nun irgendwie den erzeugten öffentlichen Schlüssel *(tunnel_key_a.pub)* in die Datei *authorized_keys* (sshtunnel@homeserver:/home/sshtunnel/.ssh/authorized_keys) auf dem Remoteserver bringen. Zusätzlich wollen wir aber für diesen Schlüssel noch folgende Features verbieten

- agent-forwarding (no-agent-forwarding)
- user-rc (no-user-rc)
- X11-forwarding (no-X11-forwarding)
- pty (no-pty)

da diese für den AutoSSH-Tunnel nicht benötigt werden und wir so einen möglichen Angriffsvektor minimieren.

Für weitere Optionen z.B. das Begrenzen der Verbindung auf bestimmte Ports, oder die Möglichkeit, beim Versuch Befehle abzusetzen eine Fehlermeldung auszugeben uvm., lesen sie bitte die manpage von sshd(8), (suchen sie nach *AUTHORIZED_KEYS FILE FORMAT*) für weitere Optionen.

Mit folgendem Befehl kann man die Textausgabe erzeugen, welche wir auf dem Remote Server in die *authorized_keys* Datei schreiben müssen:

```
[root@roadwarrior ~]# echo 'no-agent-forwarding,no-user-rc,no-X11-
forwarding,no-pty' $(cat /home/sshtunnel/.ssh/tunnel_key_a.pub) >
/tmp/add_to_authorized_keys_on_home_server
```

Fügen sie nun, mit der für sie am einfachsten durchzuführenden Methode, den Inhalt der Datei

```
/tmp/add_to_authorized_keys_on_home_server
```

am Ende der Datei *sshtunnel@homeserver:/home/sshtunnel/.ssh/authorized_keys* auf dem Remote-Server (Homeserver) an.

In meiner Testumgebung sieht die Datei *sshtunnel@homeserver:/home/sshtunnel/.ssh/authorized_keys* dann so aus:

```
no-agent-forwarding,no-user-rc,no-X11-forwarding,no-pty ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCqfcJA8YaH5D/CV7eRoq4WU/ARsG55jFhASxLc5uV7dB0aqGe
yqRWXlDeOCOXa70WE6+Aew5mTMMhcYizlsWx8l66UH6gGm3GGlPYY0/tDn3jWGKjJGwzK9+konKFerr
5yzWoX9QXXM1ajBl/R1yjgvqOmUhyBz4lptQ8iK+
+TuqBbNW9aV+jszUBt9qO1bDCNf36erqENQjHFriVuw2Drzf4fCm7e6tel5BXQ7ehX1jm9NPv9D+ip2
6UXAiy5uNG9DDXhGOQK1erfWZLYnEeTYX5P0whbpKUtuJFUQfQOkiAVCfed10ivzXQWfs/gBt9excLh
8SkLdRmXJdzC+EAx sshtunnel@roadwarrior
```

Zusätzlich sollten die Datei noch die Berechtigung *-rw------- (600)* erhalten.

Nach der Übertragung können sie die Datei

```
/tmp/add_to_authorized_keys_on_home_server
```

auf der lokalen Maschine (Road Warrior) löschen.

Schritt 4 - Remote Server zur Liste der bekannten Hosts auf der lokalen Maschine (Road Warrior) hinzufügen

Bitte ersetzen sie im folgenden Kommando den Text ‘REMOTEHOST’ durch die IP-Adresse ihres Remote-Hosts (Home-Server-IP).

```
[root@roadwarrior ~]# -s /bin/sh sshtunnel
[su@ip ~] ssh -c "ssh-keyscan -H -t rsa REMOTEHOST|tee >> ~/.ssh/known_hosts"
```

Schritt 5 - Test des Reverse-SSH-Tunnels zum Remote Server

Damit die Verbindung mit AutoSSH auch klappt, muss der Remote-Server in den „Known_Hosts“ gelistet sein. Am einfachsten erreicht man dies, in dem man sich einmal als User *(hier sshtunnel)* per ssh mit dem Server verbindet und die Verbindung akzeptiert.

Um AutoSSH nutzen zu können, muss das Paket erst noch installiert werden.

Bei CentOS gibt es die Besonderheit, dass das Paket „autossh“ nicht in den Standard-Repositories zu finden ist, sondern in den EPEL-Paketen (Extra Pakete für Enterprise-Linux).

```
[root@roadwarrior ~]# yum install epel-release
[root@roadwarrior ~]# yum install autossh
```

Danach kann der AutoSSH-Befehl gestartet werden
Bitte ersetzen sie im folgenden Kommando den Text ‘REMOTEHOST’ durch die IP-Adresse ihres Remote-Hosts (Home-Server-IP).

```
[root@roadwarrior ~]# su -c "autossh -M 0 -v -i ~/.ssh/tunnel_key_a REMOTEHOST
-N -R 9000:localhost:22" -s /bin/sh sshtunnel
```

```
OpenSSH_7.2p2, OpenSSL 1.0.2g  1 Mar 2016
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Connecting to REMOTEHOST [192.168.0.14] port 22.
debug1: Connection established.
debug1: identity file /home/sshtunnel/.ssh/tunnel_key_a type 1
debug1: key_load_public: No such file or directory
debug1: identity file /home/sshtunnel/.ssh/tunnel_key_a-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_7.2
debug1: Remote protocol version 2.0, remote software version OpenSSH_6.6.1
debug1: match: OpenSSH_6.6.1 pat OpenSSH_6.6.1* compat 0x04000000
debug1: Authenticating to 46.101.240.82:22 as 'sshtunnel'
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: curve25519-sha256@libssh.org
debug1: kex: host key algorithm: ssh-rsa
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC:
<implicit> compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC:
<implicit> compression: none
...
```

```
...
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: ssh-rsa
SHA256:UiOgkSdzpx/a3ZlY3OFecM7CCIkz6f6sD5Anw4f5T18
debug1: Host '192.168.0.14' is known and matches the RSA host key.
debug1: Found key in /home/sshtunnel/.ssh/known_hosts:1
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEY
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS received
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,gssapi-keyex,gssapi-with-
mic,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /home/sshtunnel/.ssh/tunnel_key_a
debug1: Server accepts key: pkalg ssh-rsa blen 279
debug1: Authentication succeeded (publickey).
Authenticated to REMOTEHOST ([192.168.0.14]:22).
debug1: Remote connections from LOCALHOST:9000 forwarded to local address
localhost:22
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: pledge: network
debug1: Remote: Agent forwarding disabled.
debug1: Remote: User rc file execution disabled.
debug1: Remote: X11 forwarding disabled.
debug1: Remote: Pty allocation disabled.
debug1: Remote: Agent forwarding disabled.
debug1: Remote: User rc file execution disabled.
debug1: Remote: X11 forwarding disabled.
debug1: Remote: Pty allocation disabled.
debug1: remote forward success for: listen 9000, connect localhost:22
debug1: All remote forwarding requests processed
```

Beenden sie den Test mit *Ctrl+C*

Die Verbindung zum Remote Server (Homeserver) sollte ähnlich wie oben aufgeführt aufgebaut werden. Wir wissen nun also, dass der RoadWarrior per AutoSSH eine Verbindung zum Homeserver aufbauen kann und wir darüber Verbindungen tunneln können.

Übrigens kommt es darauf an, welche Version von AutoSSH bzw. SSH bei ihnen installiert ist. Der Syntax kann je nach Version gravierend abweichen, z.B. wäre obiger Befehl bei einer etwas älteren Ubuntu-Version folgendermaßen:

```
su -c "autossh -v -i ~/.ssh/tunnel_key_a remote-server -N -R 9000:localhost:22"
-s /bin/sh sshtunnel
```


Schritt 6 - AutoSSH Befehl auf dem RoadWarrior automatisch starten

Damit der RoadWarrior, wie der Name *AutoSSH* schon erahnen lässt automatisch die Verbindung zum Homeserver aufbaut, muss noch dafür gesorgt werden, dass der AutoSSH-Befehl automatisch abgesetzt wird.

Hierzu gibt es viele Möglichkeiten. Bei Systemen mit *systemd* empfiehlt es sich z.B. wie in dieser Anleitung vorzugehen: http://blog.philippklaus.de/2013/03/start-autossh-on-system-startup-using-systemd-on-arch-linux/

In meinem Beispiel nutzt der RoadWarrior *Arch Linux* und ich beschreibe hier kurz mein Setup:

systemd .service Konfigurationsdatei erzeugen

```
# /etc/systemd/system/autossh-tunnel-a.service
# you could also us a systemd/user unit

[Unit]
Description=AutoSSH service for a reverse tunnel from some.example.com to this
machine (tunnel_key_a)
After=network.target

[Service]
# specify the user for the command
User=sshtunnel
# what to execute
ExecStart=/usr/bin/autossh -M 0 -q -i /home/sshtunnel/.ssh/tunnel_key_a
192.168.0.14 -N -o "ServerAliveInterval 60" -o "ServerAliveCountMax 3" -R
9000:localhost:22

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

Nun den neuen Daemon bekanntmachen

```
[root@roadwarrior ~]# systemctl daemon-reload
```

starten

```
[root@roadwarrior ~]# systemctl start autossh-tunnel-a.service
```

Status prüfen

```
[root@roadwarrior ~]# systemctl status autossh-tunnel-a.service
```

und für autostart enablen

```
[root@roadwarrior ~]# systemctl enable autossh-tunnel-a.service
```

Nun ist der Road-Warrior **permanent** soweit ein Verbindungsaufbau möglich mit dem Remote Server (Home Server) verbunden.
Auf dem Home Server kann man nun jederzeit eine Verbindung zum RoadWarrior aufbauen indem man auf dem Home Server eine SSH-Verbindung zum lokalen Port 9000 (das ist der Port den wir dafür konfiguriert haben) aufbauen

Wichtig ist hierbei, dass der User, mit dem man sich verbinden möchte, auf dem Roadwarrior auch vorhanden ist.

Bei der erstmaligen Verbindung (später nicht mehr) muss ECDSA fingerprint einmal akzeptiert werden

```
[root@homeserver ~]# ssh -p 9000 user@localhost

The authenticity of host '[localhost]:9000 ([::1]:9000)' can't be established.
ECDSA key fingerprint is e7:bc:94:f9:a6:f6:4b:53:21:f1:49:45:0d:aa:05:65.
Are you sure you want to continue connecting (yes/no)? Yes
Warning: Permanently added '[localhost]:9000' (ECDSA) to the list of known
hosts.
Password: <your ssh password for the above connection>
Last login: Tue Apr  5 08:44:36 2016 from 192.168.0.111
```

Zusammenfassung

Mit AutoSSH ergeben sich ungeahnte Möglichkeiten, z.B. könnte man nun vom HomeServer aus regelmäßig Daten vom RoadWarrior automatisiert mit *rsync* oder *rsnapshot* sichern, der RoadWarrior bekäme davon nichts mit.

Ein eingehender Port muss nur in der Firewall des Homeservers geöffnet werden, der RoadWarrior baut ja die Verbindung per AutoSSH (ausgehend) auf. Den “Rück-Kanal” dieser Verbindung nutzt man dann vom Server aus (user@localhost port 9000)

Links

https://chemnitzer.linux-tage.de/2016/media/programm/papers/371_Sicheres_Backup_rsync_ssh_und_LUKS_im_Team_folien_Vortrag_CLT_2016.pdf

https://blog.sleeplessbeastie.eu/2014/12/23/how-to-create-persistent-reverse-ssh-tunnel/