OpenSSH
Teil 1: Grundlagen

Johannes Franken
<jfranken@jfranken.de>

Viele Leute glauben, dass **`ssh`** bloss **`telnet`** mit Verschlüsselung sei. Auf dieser Seite demonstriere ich die über **`telnet`** hinaus gehenden Fähigkeiten von OpenSSH 3.x.

Inhalt

OpenSSH - was ist das?

OpenSSH ist eine Implementierung der ssh-Protokoll-Suite. Der Quellcode ist offen und wird ständig weiterentwickelt vom OpenSSH-Projekt-Team, siehe http://www.openssh.org

Bezugsquellen

OpenSSH ist für sämtliche Betriebssysteme compiliert worden und im Lieferumfang der meisten Linux- oder BSD-Distributionen enthalten.

Ein exzellenter Windows-Port unter Verwendung der CygWin32 Bibliotheken ist erhältlich auf http://www.networksimplicity.com/openssh/

Bestandteile

Die OpenSSH-Distribution enthält folgende Programme:

Name	Verwendung
ssh	ist der ssh-Client. Er baut die Verbindung zu einem ssh-Server auf. Er wird in den folgenden Kapiteln detailliert beschrieben.
sshd	ist der ssh-Server. Er nimmt die Verbindung von ssh-Clients auf. Details: siehe Konfigurationshinweise
ssh-keygen	erstellt und konvertiert Schlüssel. Details: siehe unten.
ssh-agent, **ssh-add**	hält den entschlüsselten Privatekey im Arbeitsspeicher, wo er von ssh-clients angesprochen werden kann. Details: siehe unten.
ssh-keyscan	zeigt den Hostkey eines ssh-Servers an. Anwendungsbeispiel: siehe unten.

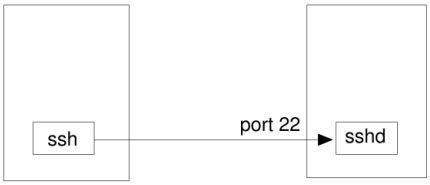
Alternativen zu OpenSSH

Putty ist eine freie Implementierung eines ssh-Clients, welche im Gegensatz zu dem in OpenSSH enthaltenen Client eine grafische Benutzeroberfläche bietet. Eine Windows-Version ist auf http://www.chiark.greenend.org.uk/~sgtatham/putty/ verfügbar, und es gibt bereits die ersten Linux-Ports.

Dann gibt es noch einige kommerzielle SSH-Varianten, z.B. auf http://www.ssh.com und http://www.f-secure.com Zusätzlich zu den Möglichkeiten von OpenSSH enthalten sie Programme zur zentralen Konfiguration von Servern, Schlüsseln und Tunnels.

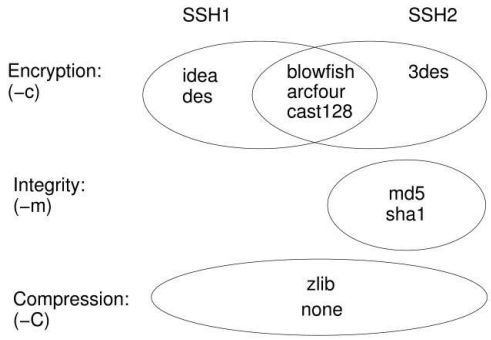
ssh-Protokolle

Beim Einsatz von OpenSSH wird meist ein ssh-Client (**/usr/bin/ssh**) einen ssh-Server (**/usr/sbin/sshd**) ansprechen, der auf einem anderen Rechner laeuft und auf Port 22 lauscht. Im Laufe der Kommunikation kommen die verschiedensten Protokolle zum Einsatz.




```
jfranken@hamster:~ $ echo SSH-1.99-jfranken| nc -w 1 gate 22
SSH-1.99-OpenSSH_3.4p1Debian1:3.4p1-1\n[...]
diffie-hellman-group-exchange-sha1,diffie-hell
man-group1-sha1\0\0\0017ssh-rsa,ssh-dss\0\0\0faes128-cbc
,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc
,rijndael-cbc@lysator.liu.se\0\0\0faes128-cbc,3des-cbc,blowfish-cbc
,cast128-cbc,arcfour,aes192-cbc,aes256-cbc,rijndael-cbc@lysator.
liu.se\0\0\0Uhmac-md5,hmac-sha1,hmac-ripemd160,hmac-ripemd160@opens
sh.com,hmac-sha1-96,hmac-md5-96\0\0\0Uhmac-md5,hmac-sha1,hmac-ripem
d160,hmac-ripemd160@openssh.com,hmac-sha1-96,hmac-md5-96\0\0\0\tnone
,zlib\0\0\0\tnone,zlib\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
```

Die Liste zeigt die in der ssh-Protokollversion 2 enthaltenen Algorithmen. Eine Untermenge war bereits in der ssh-Protokollversion 1 enthalten, die heute aus Sicherheitsgründen nicht mehr verwendet werden sollte. OpenSSH 3 spricht wahlweise die Protokollversionen 1 oder 2.



Details zu den einzelnen Algorithmen gibt es hier:

- draft-galb-secsh-gssapi-00.txt
- draft-galb-secsh-publickey-channel-00.txt
- draft-ietf-secsh-architecture-06.txt
- draft-ietf-secsh-auth-kbdinteract-01.txt
- draft-ietf-secsh-connect-08.txt
- draft-ietf-secsh-transport-08.txt
- draft-ietf-secsh-userauth-08.txt
- draft-nisse-secsh-srp-00.txt
- draft-salowey-secsh-kerbkeyex-00.txt

Konfiguration

Client-Konfiguration

Der ssh-Client liest seine Konfiguration aus

- Kommandozeilenparametern,
- **`~/.ssh/config`** und
- **`/etc/ssh/ssh_config`**.

Der erste Treffer zählt. Die config-Dateien erlauben die Definition von Host-Bereichen. z.B.:

```
Host stunnel.our-isp.org
  ProxyCommand ~/.ssh/ssh-https-tunnel %h %p
  Port 443

Host hera 141.*
  User franken
  Protocol 1

Host 192.* gate mausi hamster tp
  Compression no
  Cipher blowfish
  Protocol 2

Host *
  ForwardAgent yes
  ForwardX11 yes
  Compression yes
  Protocol 2,1
  Cipher 3des
  EscapeChar ~
```

Mehr zum Thema:
siehe: ssh(1), ssh_config(5) manpages.

Server-Konfiguration

Der ssh-Server liest seine Konfiguration aus

- Kommandozeilenparametern und
- **`/etc/ssh/sshd_config`**

Der erste Treffer zählt.

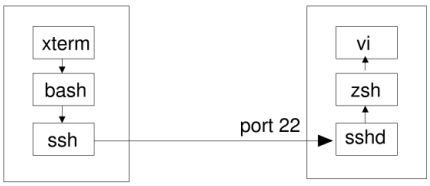
Mehr zum Thema:
siehe: sshd(8), sshd_config(5) manpages.

Login Sessions

Durch Eingabe von

```
ssh hostname
```

eröffnet man eine Shell auf dem Rechner **hostname**.



Die Anmeldung erfolgt mit dem lokalen Usernamen

Wenn man sich auf dem anderen Rechner unter einem anderen Usernamen anmelden möchte, hilft entweder

- **ssh -l username hostname**
- **ssh username@hostname**
- **ssh -o User=username hostname**
- **ssh hostname** und **User hostname** in der config-Datei.

Escape-Kommandos

Während der ssh-Sitzung kann man nach Eingabe von **<Enter><Tilde>** mit den folgenden Tasten Funktionen des ssh-Clients aufrufen:

.	Beendet die Sitzung inkl. aller Tunnels.
&	beendet die Sitzung. Die Tunnels bleiben bestehen.
C	Zeigt einen Prompt (**ssh>**), an dem man mit den Befehlen **-L** und **-R** nachträglich Portforwarding einrichten kann (Details: siehe Teil 2).
<Strg-Z>	Suspended die ssh. Man landet in der Shell, aus der man ssh aufgerufen hatte. Mit **fg** geht's weiter.
#	zeigt alle Verbindungen an, die gerade über die aktuelle ssh tunneln.
?	Hilfemenü. Zeigt auch die hier nicht genannten, eher unwichtigen Befehle an.

Wer ein Keyboard-Layout mit deadkeys verwendet, drückt für ein Tildezeichen zweimal auf die Tilde-Taste.

Wer mehrere ssh-Sitzungen ineinander geschachtelt hat, erreicht den Escape-Modus der n-ten Shell (von außen gezählt) durch **<Enter>** und n-fachen Druck (bei deadkeys 2n-fach) auf die Tilde-Taste. Wem das zu kompliziert ist, der kann mit dem **-e** Parameter für jede ssh einen eigenen Escape-Charakter (z.B. **<Strg-B>**) bestimmen:

```
jfranken@gate:~ $ ssh -e ^B hamster
jfranken@hamster:~ $ <Strg-B>.
Connection to hamster closed.
jfranken@gate:~ $
```

Interaktive Programme automatisch starten

```
$ ssh tp -t vim /etc/hosts
```

Verschlüsselung

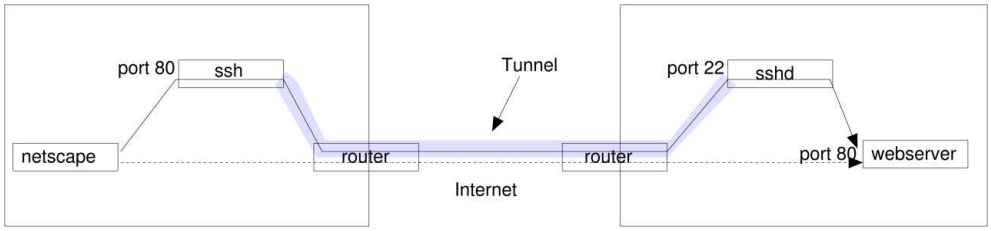
ssh verschlüsselt die gesamte Kommunikation zum sshd, inkl. der Anmeldung und aller Tunnels. Man kann dabei zwischen verschiedenen Verschlüsselungsalgorithmen wählen, z.B.

- **`3des`** gilt als besonders sicher, benötigt aber viel Rechenzeit
- **`blowfish`** arbeitet besonders schnell

Hier sind einige Möglichkeiten zur Auswahl des **`blowfish`**-Algorithmus im lokalen Netz, der die beteiligten Prozessoren entlastet und somit die Kommunikation beschleunigt:

- **`ssh -c blowfish`** ...
- **`ssh -o Cipher=blowfish`**...
- Eintrag **`Cipher blowfish`** für einige oder alle Hosts in der **`~/.ssh/config`** oder **`/etc/ssh/ssh_config`**.

In Verbindung mit Portforwarding lassen sich unsichere Netzabschnitte für den Client völlig unsichtbar verschlüsseln, etwa um zwei Firmenstandorte über das Internet zu verbinden:



Komprimierung

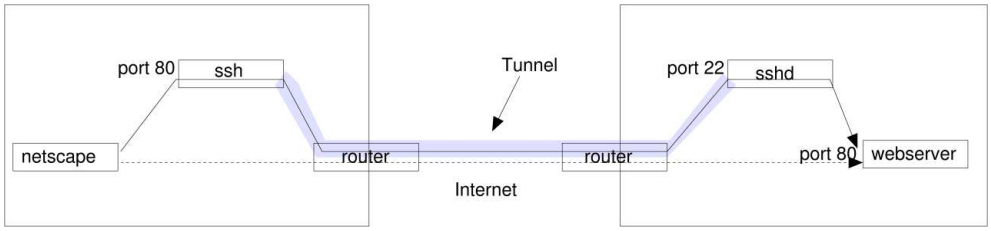
Wenn sich das Netz als Engpaß herausstellt, kann man die Kommunikation zwischen ssh und sshd komprimieren. Das benötigt auf beiden Seitem etwas mehr Rechenzeit, spart jedoch etwa 50% der Netz-Pakete ein.

Um die Komprimierung zu aktivieren:

- **`ssh -C ...`**
- **`ssh -o Compression=yes ...`**
- **`Compression yes`** in **`~/.ssh/config`** oder **`/etc/ssh/ssh_config`**einbauen.

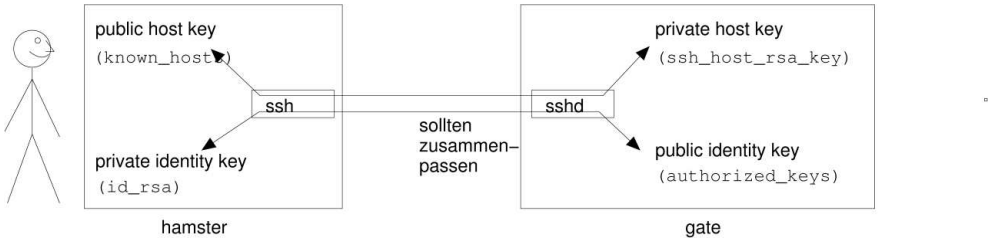
Zusätzlich sollte auf dem Server in **`/etc/ssh/sshd_config`** die Zeile **`Compression yes`** stehen.

In Verbindung mit Portforwarding lassen sich langsame Netzabschnitte (z.B. ISDN- oder DSL-Strecken) für den Client völlig unsichtbar beschleunigen:



Public key authentication

Die Publickey-Authentifizierung beruht auf dem Prinzip asymmetrischer Verschlüsselung: Der Client und der Server besitzen einen Schlüssel und können damit Nachrichten so verschlüsseln, dass nur der andere sie entschlüsseln kann. Wenn einer nun seinen Schlüssel geheim hält ("private key"), kann der andere von der Authentizität seines Gegenüber ausgehen, sobald er eine Nachricht mit einem seiner eigenen Schlüssel ("public key") entschlüsseln kann.



Hierzu stehen drei verschiedene Algorithmen zur Verfügung:

Algorithmus	Parameter für `ssh-keygen`	Dateinamen	Publickey beginnt mit
RSA für SSH ab Version 2	**`-t rsa`**	**`~/.ssh/id_rsa[.pub]`**	**`ssh-rsa`**
RSA für SSH Version 1	**`-t rsa1`**	**`~/.ssh/identity[.pub]`**	**`1024 35`** o.ä, (bits exponent)
DSA	**`-t dsa`**	**`~/.ssh/id_dsa[.pub]`**	**`ssh-dss`**

Auf die Spezialitäten der einzelnen Algorithmen gehe ich hier nicht weiter ein, als dass DSA die meiste Rechenlast erzeugt und RSA1 besonders leicht verwundbar ist. In vielen Fällen bietet RSA einen guten Kompromiss.

Jeder Publickey (z.B. **`~/.ssh/id_rsa.pub`**) setzt sich wie folgt zusammen:

a) Der Key-Type (z.B. **`ssh-rsa`**, siehe Tabelle oben)
b) Ein Leerzeichen
c) Der Modulus (das ist die lange Folge von Buchstaben/Ziffern)
d) optional: Ein Leerzeichen, gefolgt von einem Kommentar

Er wird mittels uuencode auf 6 bit Ascii codiert. Damit sieht er wie eine lange Text-Zeile aus und lässt sich leicht im Text einer Mail verschicken:

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAxtWFO8XbyT6+IlBAWYyOb
/VWraJ7iymKVsb0TNmieBSzF6fgustkT0nX3udbSqTqiEC/wXFKqeyl27
bkd+rEcFba+s+wgv9MKRaiV0kOFVQrAvwrKnS1QI6YZWZIhSP7KS5QE0H
Rra+gy/47vGwHUn0RxksGOQ6YsKP5lNN8H3E= jfranken@hamster
```

ssh-keygen

Jeder Authentifikationsalgorithmus benötigt ein eigenes Schlüsselpaar, das man mit dem Befehl **ssh-keygen** erstellen kann. Mit dem **-t**-Parameter bestimmt man den Algorithmus, zu dem man das Schlüsselpaar erzeugen möchte.

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/export/home/jfranken/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /export/home/jfranken/.ssh/id_rsa.
Your public key has been saved in /export/home/jfranken/.ssh/id_rsa.pub.
The key fingerprint is:
7c:02:29:1c:d4:8a:90:ad:f2:b0:65:9e:71:92:ef:0f jfranken@hamster
```

Der Fingerprint ist eine gemeinsame Eigenschaft von Private- und Publickey und ermöglicht es später festzustellen, welcher Privatekey zu einem Publickey passt:

```
$ ssh-keygen -l -f .ssh/id_rsa
1024 5a:b6:c4:50:ce:ec:18:78:e9:b2:e7:5b:04:c2:e4:c7 .ssh/id_rsa.pub
```

Man kann die Passphrase des Privatekey nachträglich ändern:

```
$ ssh-keygen -p -f ~/.ssh/id_rsa
Enter old passphrase:
Key has comment '/export/home/jfranken/.ssh/id_rsa'
Enter new passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved with the new passphrase.
```

Wer seinen Publickey verlegt hat, kann sich aus dem Privatekey einen neuen Publickey erzeugen:

```
$ ssh-keygen -y -f ~/.ssh/id_rsa
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEA5+UaqG/zI...
```

Aber nicht vergessen, es gibt keine Möglichkeit, einen verlorenen Privatekey aus dem Publickey zu erstellen.

Hostkeys verwenden

known_hosts

Wer beim Aufbau einer ssh-Verbindung ganz sicher gehen möchte, dass der erreichte Server auch der richtige ist, gibt die Option **`-o StrictHostkeyChecking=yes`** an oder trägt in der **`~/.ssh/config`** für die betreffenden Hosts **`StrictHostkeyChecking yes`** ein.

Der Client bricht dann die Verbindung ab, wenn der geheime Hostkey des Servers nicht zu dem Public-Key passt, der bereits auf dem Client in **`~/.ssh/known_hosts`** oder **`/etc/ssh/ssh_known_hosts`** abgelegt worden ist:

```
jfranken@hamster $ ssh -o StrictHostkeyChecking=yes gate
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
5c:6e:b2:99:3d:44:03:32:fb:e8:c1:ca:4f:cb:9e:8f.
Please contact your system administrator.
Add correct host key in /export/home/jfranken/.ssh/known_hosts to get rid of this message.
Offending key in /export/home/jfranken/.ssh/known_hosts:45
RSA host key for gate has changed and you have requested strict checking.
Host key verification failed.
jfranken@hamster $
```

oder, wenn dieser Host noch gar nicht in der **`known_hosts`** eingetragen ist:

```
jfranken@hamster $ ssh -o StrictHostkeyChecking=yes gate
No RSA host key is known for gate and you have requested strict checking.
Host key verification failed.
jfranken@hamster $
```

Wer nicht alle Hostkeys manuell in die **`known_hosts`**-Datei aufnehmen möchte, sondern nur bei Veränderungen gewarnt werden möchte, sollte **`StrictHostkeyChecking`** auf **`ask`** setzen:

```
jfranken@hamster $ ssh -o StrictHostkeyChecking=ask gate
The authenticity of host 'gate (192.168.42.1)' can't be established.
RSA key fingerprint is 5c:6e:b2:99:3d:44:03:32:fb:e8:c1:ca:4f:cb:9e:8f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'gate,192.168.42.1' (RSA) to the list of known hosts.
jfranken@gate $
```

Die Zeilen der **`known_hosts`** haben folgendes Format:

1. Liste von Hostnamen oder IP-Adressen eines Servers, durch Komma getrennt. Wildcards **`*?`** und Negation **`!`** sind erlaubt
2. Ein Leerzeichen
3. Der Publickey dieses Servers (ohne Kommentar)
4. optional: ein Leerzeichen, gefolgt von einem Kommentar

Die Hostkeys befinden sich auf dem Server an folgenden Stellen:

- **`/etc/ssh/ssh_host_dsa_key[.pub]`** (DSA-Format)
- **`/etc/ssh/ssh_host_rsa_key[.pub]`** (RSA-Format f. SSH version 2+)
- **`/etc/ssh/ssh_host_key[.pub]`** (im RSA-Format f. SSH version 1)

Man kann diese Schlüssel mit **ssh-keygen** herstellen. Dabei ist zu beachten. dass man dem Privatekey eine leere Passphrase gibt. Bei vielen Distributionen wird diese Aufgabe beim Einspielen des sshd-Pakets automatisch erledigt.

Ändert sich der Hostkey eines Servers (z.B. nach einem SSH Versions-Update), muß man auf dem Client die entsprechenden Zeile aus der **known_hosts** entfernen oder aktualisieren.

Mehr zum Thema:
siehe: sshd(8) manpage

ssh-keyscan

Wer viele Hostkeys in eine **known_hosts**-Datei eintragen muß, kann diese mit **ssh-keyscan** von jedem Client aus abfragen und die Ausgabe direkt an die **known_hosts** anhängen.

Da Fehler beim Auslesen des Hostkeys auf Netz- oder Serverprobleme hinweisen, kann man **ssh-keyscan** zur schnellen Diagnose solcher Probleme einsetzen: Wenn es den Hostkey nicht anzeigt, besteht ein Problem.

Mehr zum Thema:
siehe: ssh-keyscan(1) manpage

Identity-Keys verwenden

authorized_keys

Alternativ zu der Eingabe eines Passwortes kann man ein Schlüsselpaar zur Authentifikation verwenden, wobei der Privatekey wiederum mit einer Passphrase geschützt sein kann.

Vorteile der Publickey- gegenüber Passwort-Authentifikation:

- Die Schlüssel sind mit Brute-Force wesentlich aufwendiger zu erraten als Passwörter
- Wer meinen Publickey hat, kann mich leicht auf seinen Server einladen.
- Ich muß mir für alle Zugänge nur eine einzige Passphrase merken, die ich jederzeit mit einem Kommando ändern kann
- Auf dem Server liegt nur mein Public-Key, mit dem allein noch niemand auf andere Systeme zugreifen kann. Es besteht insbesondere nicht das Risiko, dass jemand mit Brute-Force auf **`/etc/shadow`** mein Passwort ertestet.
- Wenn mehrere User auf den selben Account zugreifen, kann jeder seine eigene Passphrase verwenden.

Die Verwendung von Public-keys ist einfach: Zunächst erstellt man auf dem Client beispielsweise mit **`ssh-keygen -t rsa`** ein Schlüsselpaar, wie oben beschrieben. Den Public-key kopiert man dann als eine lange Zeile in die Datei **`~/.ssh/authorized_keys`** auf dem Server.

Am Zeilenanfang der Einträge in der **`~/.ssh/authorized_keys`** kann man zusätzlich dem sshd jeweils mitteilen, von welchen Rechnern man sich mit diesem Schlüssel einloggen darf und welche Besonderheiten (Environmentvariablen, Shell, ssh-Optionen) dabei gelten. Die genaue Syntax der Zeilen lautet:

1. Optional eine Liste von Optionen, durch Komma getrennt, gefolgt von einem Leerzeichen. Beispiele:
   a) **`environment="SSHKEY=jfranken"`**
      (ab openssh ≥3.5p1 muß man hierzu **`PermitUserEnvironment=yes`** in **`/etc/ssh/sshd_config`** setzen.)
   b) **`command="./menu.pl"`**
   c) **`from="*.jfranken.de,egal.our-isp.org"`**
   d) **`no-port-forwarding`**
   e) **`no-X11-forwarding`**
   f) **`permitopen="host:port"`**
   g) **`no-agent-forwarding`**
   h) **`no-pty`**
2. Der Public-Key (siehe *.pub-Datei) ohne Kommentar
3. optional: Ein Leerzeichen und ein Kommentar

Wenn dann noch auf dem Server in der **`/etc/ssh/sshd_config`** die Option **`PasswordAuthentication No`** eingestellt ist, kann man sich nur noch mit dem Schlüssel anmelden, was ziemlich gute Sicherheit gegen Brute-ForceAngriffe bietet.

Mehr zum Thema:
siehe: sshd(8) manpage

ssh-copy-id

Wer öfters seinen Publickey auf Servern platzieren muß, wird gerne auf ein kleines Shellscript namens **`ssh-copy-id`** zurückgreifen, das über eine ssh-pipe einfach den Publickey an die **`authorized_keys`** auf dem Server anhängt. Leider verwendet es defaultmäßig den Publickey für RSA1 (**`~/.ssh/identity.pub`**). Wer lieber mit RSA(2)-keys arbeitet, kann entweder die Parameter **`-i ~/.ssh/id_rsa.pub`** angeben, oder die von mir gepatchte Version verwenden:

```
jfranken@hamster $ ./ssh-copy-id2 franken@hera.cs.uni-frankfurt.de
franken@hera.cs.uni-frankfurt.de's password:
Now try logging into the machine,
with "ssh 'franken@hera.cs.uni-frankfurt.de'", and check in:
  .ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
jfranken@hamster $
```

Mehr zum Thema:
siehe: ssh-copy-id(1) manpage

Login ohne Passwortabfrage

Leere Passphrase

Wenn die Passphrase eines Privatekey leer und der Publickey in der **~/.ssh/authorized_keys** auf dem Server enthalten ist, lässt sich **ssh** wunderbar in Scripte einbauen. Dann sollte man jedoch darauf achten, dass kein Unbefugter Zugriff auf die Privatekey-Datei hat, insbesondere wenn dieser auf einem NFS- oder CIFS-Fileserver liegt.

ssh-agent, ssh-add

Der **ssh-agent** bietet eine Lösung für User, die es leid sind, immer wieder die selbe Passphrase einzugeben, aber aus Sicherheitsgründen auch nicht ganz auf eine Passphrase verzichten möchten. Damit muß ein Hacker zumindest Zugriff auf den lokalen Rechner und die Socketdatei erlangen, wenn er den Privatekey benutzen möchte.

Wenn man den **ssh-agent** ohne Parameter aufruft, erstellt er einen Unix-Domain-Socket, teilt mit, wie man ihn dort erreichen kann und lauscht im Hintergrund darauf:

```
$ ssh-agent > myagent.sh
$ cat myagent.sh
SSH_AUTH_SOCK=/tmp/ssh-XXcMloql/agent.21753; export SSH_AUTH_SOCK;
SSH_AGENT_PID=21754; export SSH_AGENT_PID;
echo Agent pid 21754;
```

Mit **ssh-add** bringt man dem **ssh-agent** seine Privatekeys bei:

```
$ . ./myagent.sh
Agent pid 21754;
$ ssh-add ~/.ssh/id_rsa
Enter passphrase for /export/home/jfranken/.ssh/id_rsa:
Identity added: /export/home/jfranken/.ssh/id_rsa (/export/home/jfranken/.ssh/id_rsa)
$
```

Wenn man keinen Privatekey angibt, werden die Standard-keys (**identity**,**id_rsa** und **id_dsa**) geladen.

Eine Übersicht der in den agent geladenen Keys erhält mit **ssh-add -l**, und die zugehörigen Publickeys mit **ssh-add -L**.

Der **ssh**-Client wird dann bei jedem Verbindungsaufbau die Privatekeys aus dem **ssh-agent** probieren, statt denen aus den Identity-Dateien:

```
jfranken@hamster $ . ./myagent.sh
Agent pid 21754;
jfranken@hamster $ ssh gate
[...]
debug1: userauth_pubkey_agent: testing agent key /export/home/jfranken/.ssh/id_rsa
debug1: input_userauth_pk_ok: pkalg ssh-rsa blen 149 lastkey 0x80926f8 hint -1
[...]
jfranken@gate $
```

Alternativ zu der **myagent.sh** kann man die Werte für die Environment-Variablen auch mit **lsof** bestimmen:

```
$ /usr/sbin/lsof -a -u jfranken -U -c ssh-agent
COMMAND   PID     USER   FD   TYPE     DEVICE SIZE NODE NAME
ssh-agent 477 jfranken    3u  unix 0xc1da1aa0      1155 /tmp/ssh-XXWzoqlO/agent.452
$ export SSH_AUTH_SOCK=/tmp/ssh-XXWzoqlO/agent.452 SSH_AGENT_PID=477
```

Wer möchte, dass die Variablen sich automatisch bei jedem Login auf den laufenden ssh-agent einstellen, kann das Programm keychain in seine **`~/.bash_profile`** aufnehmen.

Mehr zum Thema:
siehe: ssh-agent(1) manpage
ssh-add(1) manpage

Weiterführendes

- Teil 2: SSH-Tunnels
- Teil 3: Firewalls durchboren
- The Secure Shell TM Frequently Asked Questions
- OpenSSH Projekt Seite