FORSCHUNGSZENTRUM JÜLICH GmbH

Jülich Supercomputing Centre
52425 Jülich, ☎ (02461) 61-6402

Beratung und Betrieb, ☎ (02461) 61-6400

Technische Kurzinformation

FZJ-JSC-TKI-0330
E. Grünter

03.01.2023

OpenSSH sicher konfigurieren

1. Einleitung

Secure Shell (SSH) wurde 1995 mit dem Ziel entwickelt, Programme wie telnet und rsh zu ersetzen, da diese Programme nicht nur die Sitzungsdaten, sondern auch Anmeldedaten wie Benutzer-ID und Passwort im Klartext übertragen. Diese Schwachstelle in der Übertragung der Daten wird von SSH beseitigt. Die Daten werden für den Benutzer transparent vor der Übertragung ver- und nach der Übertragung entschlüsselt, so dass SSH die Vertraulichkeit gewährleistet. Weiterhin sichert SSH die Integrität durch kryptografische Prüfsummen und die am Datentransfer beteiligten Parteien müssen einander authentifizieren.

SSH ist dennoch nicht fehlerfrei. Es gibt Schwachstellen und bei Angreifern sehr beliebt sind automatisierte Anmeldungen mit verschiedenen Benutzernamen- und Passwortkombinationen, um Passworte zu erraten, die dann in so genannten Wörterbüchern gesammelt werden. Sehr häufig werden hier Benutzerkennungen erraten, die standardmäßig vom System eingerichtet werden oder auch solche, die schwache Passworte (z.B. test123) verwenden.

Des Weiteren können implementationsabhängige Schwachstellen existieren, die ein mittleres oder hohes Sicherheitsrisiko bedeuten. Zum Zeitpunkt des Erscheinens dieser TKI gilt die Version OpenSSH8.4 als sicher.

Schwerpunkt dieser technischen Kurzinformation ist die sichere Konfiguration und Verwendung von OpenSSH. Hinweise und Beispielkonfigurationen werden, den Administratoren von JuNet-Rechnern einen sicheren Betrieb eines SSH-Servers ermöglichen. Die SSH-Protokollversion 1 und die nicht-interaktive Nutzung von SSH sind nicht Gegenstand dieser TKI.

2. Der SSH-Client

a. Allgemeines

Auf der Kommandozeile wird für eine interaktive Nutzung der folgende Befehl abgesetzt:

```
ssh Optionen [-l Benutzername] Rechnername
```

```
ssh Optionen Benutzername@Rechnername
```

Wird kein Benutzername angegeben, so nutzt SSH den Benutzernamen, der in der Umgebungsvariablen USER gespeichert ist. Nach einer erfolgreichen Authentifizierung wird vom SSH-Server eine Shell auf dem entfernten Rechner gestartet. Beim Logout wird die SSH-Sitzung beendet und die Verbindung zwischen den beteiligten Rechnern wird abgebaut.

Die Authentifizierung ist bei SSH mehrstufig. Zunächst authentifiziert sich der Server gegenüber dem Klienten mittels asymmetrischer Kryptographie. Während der Installation des SSH-Paketes wurde ein Schlüsselpaar (öffentlicher u. privater Schlüssel) für den SSH-Server erstellt. Während der Authentifizierung sendet der Server seinen öffentlichen Schlüssel an den Klienten. Meldet sich der lokale Benutzer zum Ersten Mal beim Server an, so ist eine Authentifizierung schwierig. Dem Benutzer wird der Fingerprint, ein Hash-Wert des öffentlichen Server-Schlüssels, angezeigt und er soll entscheiden, ob diesem Schlüssel vertraut wird oder nicht:

```
The authenticity of host ‘localhost(127.0.0.1)’ can’t be established. RSA key fingerprint is
65:92:9e:d5:78:a4:23:a7:10:b4:67:ac:4d:99:b3:cc. Are you sure you want to continue?
```

Wird die Frage mit "no" beantwortet, so ist die Verbindung beendet. Andernfalls wird der öffentliche Schlüssel des SSH-Servers zur Datei `$HOME/.ssh/known_hosts` hinzugefügt. Dieses Verhalten kann über die Option `StrictHostKeyChecking` in der Datei `/etc/ssh/ssh_config` beeinflusst werden (s. Beispielkonfiguration).

Diese Entscheidung ist letztlich nur dann "sicher" zu treffen, wenn der Fingerprint über einen anderen, gesicherten Kommunikationskanal verifiziert werden kann. Für JuNet-Rechner bietet das JSC den Service, die öffentlichen Host-Schlüssel zu verwalten. Die öffentlichen Schlüssel können unter *https://junet-portal.fz-juelich.de/cgi-bin/public/auth/ssh_send.cgi* gemeldet werden. Dazu muss der Inhalt der Dateien in die entsprechenden Felder eingetragen werden. Das JSC stellt dann die gesammelten öffentlichen Schlüssel in der Datei `ssh_known_hosts` auf dem internen FTP-Server zur Verfügung.

Für weitere Logins auf diesem SSH-Server wird keine Entscheidung mehr erfragt, solange der öffentliche Schlüssel des SSH-Servers nicht verändert wurde. Ist der Schlüssel verifiziert, so generieren Server und Client mit dem Diffie-Hellmann-Verfahren einen Sitzungsschlüssel, mit dem alle weiteren Informationen verschlüsselt werden.

Ist der öffentliche Schlüssel des Servers erfolgreich verifiziert worden, so kann der Benutzer sich anmelden und eine Shell starten. Es gibt mehrere Authentifizierungsverfahren (s. FAQ).

Meldet sich der Server mit einem anderen als dem gespeicherten öffentlichen Schlüssel, so generiert SSH folgende Warnmeldung:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@ WARNING: REMOTE HOST IDENTIFICATION HAS CAHNGED! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eaves-dropping on you right now (man-in-the-middleattack)!
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
65:92:9e:d5:78:a4:23:a7:10:b4:67:ac:4d:99:b3:cc
Please contact your system administrator.
Add correct host key I n /home/gruenter/.ssh/known_hosts to get rid of this message.
Offending key in /home/gruenter/.ssh/known_hosts:37
RSA host key for 127.0.0.1 has changed and you have requested strict checking.
Host key verification failed.
```

Diese Änderung ist nach einer Neuinstallation eines Systems durchaus legitim. Daher sollten vor einer Neuinstallation die Host-Schlüssel aus dem Verzeichnis `/etc/ssh` gesichert werden, um anschließend

wiederhergestellt werden zu können. Allerdings kann die Meldung auch in böswilliger Absicht erzwungen worden sein, in dem ein Angreifer vortäuscht, der gewünschte SSH-Server zu sein. In jedem Fall ist es sinnvoll, den Administrator des Rechners zu fragen, ob der Host-Schlüssel verändert wurde.

Der SSH-Client bietet eine Vielzahl von Optionen, die auf der Kommandozeile gesetzt werden können. Mittels `ssh -h` werden die Optionen angezeigt.

b. Wichtige Dateien

Im Folgenden werden Dateien aufgelistet, die von den SSH-Klienten genutzt werden. Es gibt systemweite und benutzerspezifische Dateien, die in den meisten Fällen unter $HOME/.ssh/ zu finden sind. Sind Werte in der benutzerspezifischen Konfigurationsdatei gesetzt, so haben diese Vorrang vor den systemweiten Einträgen.

`/etc/ssh/ssh_known_hosts`

In dieser systemweiten Datei sind die öffentlichen Schlüssel von SSH-Servern gespeichert. Sie stehen allen Benutzern eines Systems zur Verfügung. Diese Datei muss manuell vom Administrator bearbeitet werden, da standardmäßig neue Schlüssel in `$HOME/.ssh/known_hosts` abgelegt werden.

`/etc/ssh/ssh_config`

Diese Datei stellt die systemweite Konfigurationsdatei des SSH-Clients dar.

`$HOME/.ssh/known_hosts`

Wie `/etc/ssh/ssh_known_hosts`, allerdings sind die öffentlichen Schlüssel nur dem Benutzer zugänglich. Schlüssel von Servern, auf denen sich der Benutzer erstmalig eingeloggt hat, werden automatisch an diese Datei angehangen.

`/etc/ssh/sshrc`

Der Inhalt dieser systemweiten Datei wird nach Authentifizierung des Benutzers, aber vor dem Ausführen eines Befehls oder der Shell ausgeführt.

`$HOME/.ssh/identity , $HOME/.ssh/id_rsa , $HOME/.ssh/id_dsa`

Diese Dateien stellen die privaten Schlüssel des Benutzers dar. Sie bezeichnen die Schlüssel für die Protokollversion 1, Protokollversion 2 RSA und Protokollversion 2 DSA. Diese Schlüssel müssen geheim gehalten werden und sollten nur vom Benutzer lesbar sein. SSH ignoriert die Datei, sobald andere außer dem Eigentümer Leserechte für die Datei besitzen. Die Erstellung eines Schlüsselpaares wird in Kapitel 6 FAQ, Abschnitt e beschrieben.

`$HOME/.ssh/identity.pub , $HOME/.ssh/id_rsa.pub , $HOME/.ssh/id_dsa.pub`

Diese Dateien enthalten die öffentlichen Schlüssel zur Authentifizierung eines Benutzers.

`$HOME/.ssh/authorized_keys`

Diese Datei enthält eine Liste aller öffentlichen Schlüssel, mit denen sich ein Benutzer für diesen Account anmelden darf.

`$HOME/.ssh/config`

Diese Datei ist die Konfiguration des SSH-Clients für den jeweiligen Benutzer. Sie ergänzt oder überschreibt die Werte aus der systemweiten Konfigurationsdatei `/etc/ssh/ssh_config`. Ist die benutzereigene Konfigurationsdatei nicht vorhanden, so wird die Datei `/etc/ssh/ssh_config` ausgeführt.

`$HOME/.ssh/sshrc`

Wie `/etc/ssh/sshrc`, jedoch wird der Inhalt dieser Datei nur für den Benutzer ausgeführt.

`$HOME/.ssh/environment`

Enthält zusätzliche Umgebungsvariablen, die auf dem Server beim Einloggen über SSH definiert werden.

c. Beispielkonfiguration

Die folgende Beispielkonfiguration kann als systemweite Konfigurationsdatei

`/etc/ssh/ssh_config` verwendet werden. Sie gibt sinnvolle Standeinstellungen vor, die durch benutzerdefinierte Konfigurationsdateien (`$HOME/.ssh/config`) ersetzt werden können.

```
#Konfigurationsdatei fuer den ssh-client
#Einleitung eines Konfigurationsblocks fuer alle Rechner, zu denen sich verbunden wird.
Host *
#Maximal ein Verbindungsversuch soll gestartet werden (Standardwert).
 ConnectionAttempts 1
#Bekanntgabe der systemweiten known_hosts-Datei (Standardwert).
 GlobalKnownHostsFile /etc/ssh/ssh_known_hosts
#Bekanntgabe der benutzereigenen known_hosts-Datei (Standardwert)
 UserKnownHostsFile ~/.ssh/known_hosts
#Level fuer Syslog einstellen; moegliche Werte: QUIET,FATAL,ERROR,INFO,VERBOSE, DEBUG1,
#DEBUG2 (Standardwert)
 LogLevel INFO
#Timeout fuer den Verbindungsaufbau in Sekunden setzen (ab OpenSSH 3.8).
 ConnectTimeout 20
#DNS-Namen und IP-Adressen vergleichen als Schutz gegen IP-Spoofing.
CheckHostIP yes
#Hostbasierte Authentifizierung wird ausgeschaltet.
 HostbasedAuthentication no
#Anzahl der Password-Prompts einstellen (Standardwert)
NumberOfPasswordPrompts 3
#Authentifizierung ueber Passwort einschalten
 PasswordAuthentication yes
#Public-Key-Authentifizierung einschalten
 PubkeyAuthentication yes
#Identitaetsdateien fuer Public-Key-Autehntifizierung angeben
IdentityFile ~/.ssh/idententiy
IdentityFile ~/.ssh/id_rsa
IdentityFile ~/.ssh/id_dsa
#Bevorzugte Authentifizierungsmethoden setzen, Werte: hostbased, publickeys,keyboard
#interactive,password.
 PreferredAuthentications publickey,password
#Schlüssel von unbek. Hosts nur auf Anfrage an known_hosts-Datei haengen, Werte: yes, no, ask
(Standardwert).
 StrictHostKeyChecking ask
#Protokollversion einstellen (Standardwert 2,1).
 Protocol 2
#Alle X11-Verbindungen werden durch die SSH-Verbindung getunnelt
 ForwardX11 yes
#Weiterleitungen des SSH-Agenten aktivieren, falls ssh zum Ziel ueber Zwischenstationen
gefuehrt wird. Standardwert fuer diese Option ist no.
# ForwardAgent yes
```

3. Der SSH-Server

a. Allgemeines

In diesem Kapitel wird der SSH-Server sshd beschrieben. Er besitzt eine eigene Konfigurationsdatei, die das Verhalten des Servers bestimmt. Der Server nimmt Verbindungsanfragen der Clients entgegen und stellt dem Benutzer eine Shell für interaktive Sitzungen zur Verfügung.

Die Konfiguration des Servers besitzt eine Vielzahl von Optionen, die meistens als Direktiven in der Konfigurationsdatei eingesetzt werden. Kommandozeilenoptionen werden nur zur Fehlersuche genutzt.

b. Wichtige Dateien

Der SSH-Server benötigt die folgenden Dateien:

`/var/run/sshd-pid`

Diese Datei speichert die Prozess-ID (PID) des Server-Prozesses.

`/etc/ssh/sshd_config`

Die Konfigurationsdatei des `sshd`. Die Unix-Dateirechte für diese Datei sollten so gesetzt sein, dass nur der Super-User (root) die Datei schreiben kann.

`/etc/ssh/ssh_host_key, /etc/ssh/ssh_host_rsa_key, /etc/ssh/ssh_host_dsa_key`

Diese Dateien speichern die privaten Schlüssel des Rechners. Die Dateirechte sollten nur dem Super-User lesenden und schreibenden Zugriff gewähren.

`/etc/ssh/ssh_host_key.pub, /etc/ssh/ssh_host_rsa_key.pub, /etc/ssh/ssh_host_dsa_key.pub`

Diese Dateien speichern die öffentlichen Schlüssel des Rechners. Ihr Inhalt kann unter *https://junet-portal.fz-juelich.de/cgi-bin/public/auth/ssh_send.cgi* gemeldet werden, so dass die öffentlichen Schlüssel einer FZJ-weiten `known_hosts`-Datei zugefügt werden. Die Datei sollte für alle Benutzer eines Systems lesbar, aber nur für *root* schreibbar sein.

`/etc/ssh/moduli`

Enthält Diffie-Hellmann-Gruppen, die zur Schlüsselgenerierung genutzt werden.

`/var/run/sshd`

Dieses Verzeichnis wird als CHROOT-Umgebung des SSH-Servers genutzt. Das Verzeichnis muss leer sein und darf nur für *root* schreibbar sein.

`/etc/hosts.allow, /etc/hosts.deny`

Die Dateien werden für Zugangskontrollen genutzt, wenn der SSH-Server mit TCPWrapper-Unterstützung kompiliert wurde (s. FAQ).

c. Beispielkonfiguration

```
#Beispielkonfiguration fuer einen ssh-server
#Umfang des Syslog, Werte: QUIET, FATAL, ERROR, INFO, VERBOSE, DEBUG1, DEBUG2, DEBUG3
LogLevel VERBOSE
SyslogFacility AUTH
#MaxStartUps begrenzt die max. Anzahl nicht authentifizierter Verbindungen. Das sind
#Verbindungen, die z.B. auf die Passworteingabe des Benutzerrs warten.
MaxStartups 3
#Datei, welche die Prozess-ID speichert (Standardwert)
PidFile /var/run/sshd.pid
#Zeigt beim Login den Zeitpunkt des letzten erfolgreichen Login an (Standardwert).
PrintLastLog yes
#Gibt an, ob der Inhalt der Datei /etc/motd (Message of the day) angezeigt
#wird (Standrdwert).
PrintMotd yes
#StrictModes prueft die Rechte wichtiger Dateien, die vom sshd benutzt werden.
#Bei unsicherer Rechtevergabe wird der Server nicht gestartet.
StrictModes yes
#Aktiviert das Login-Programm fuer interaktive Sitzungen (Standardwert).
UseLogin no
```

```
#Ein unprivilegierter sshd-Prozess wird gestartet, der nur den Netzwerkverkehr verwaltet.
#Nach erfolgreicher Authentifizierung startet dann ein Prozess mit den Rechten des
#Benutzers (Standardwert).
UsePrivilegeSeparation yes
#Bennent den vollstaendigen Pfad zum xauth-Programm.
XAuthLocation /usr/bin/X11/xauth
#Nach 60 Sekunden der Inaktivität schickt der Server dem Client eine Nachricht,
#die bestätigt werden #muss. Auf diese Art koennen unterbrochene Verbindungen
#festgestellt werden.
ClientAliveInterval 60
#Max. 3 der durch ClientAliveInterval gesendeten Nachrichten duerfen unbestaetigt bleiben,
#bevor die Verbindung abgebaut wird.
ClientAliveCountMax 3
#Ab OpenSSH3.8 verwendbar, bestimmt die Option ob TCP-Keep-Alives zur Erkennung
#unterbrochener Verbindungen genutzt werden. Damit wird die Erkennung unterbrochener
#Verbindungen sensibilisiert, so dass eine Verbindung auch bei kleineren Stoerungen
#abgebaut wird.
TCPKeepAlive no
#Vor dem Senden werden die Daten komprimiert (Standardwert).
Compression yes
#Gibt die Portnummer des Servers an (Standardwert).
Port 22
#ListenAddress benennt die Interfaces eines Rechners, an die sich der SSH-Server binden
#soll. Der Wert 0.0.0.0 kennzeichnet jedes Interface mit einer IPv4-Adresse. Der Wert ::
#kennzeichnet jedes Interface mit einer IPv6-Adresse.
ListenAddress 0.0.0.0
ListenAddress ::
#Legt fest, welche SSH-Protokollversion verwendet wird (Standardwert 2,1)
Protocol 2
#Ab OpennSSH3.6 ist die Option so benamt (vorher VerifyReverseMapping). Bei eingehenden
#SSH-Verbindungen wird die Quell-IP-Adresse in einen DNS-Namen umgewandelt und umgekehrt.
#Sind die Client-IP-Adresse und das Ergebnis der Namensaufloesung gleich, so wird
#die Verbindung zugelassen.
UseDNS yes
#Legt fest, dass X11-Verbindungen ueber SSH getunnelt werden.
X11Forwarding yes
#Gibt den ersten Display-Wert fuer virtuelle X11-Displays an.
X11DisplayOffset 10
#Alle virtuellen X11-Displays werden an die Loopback-Adresse (127.0.0.1) des Rechners
#gebunden.
X11UseLocalhost yes
#Benutzerkontrolle: Nur Mitglieder der Gruppen „users“ und „root“ duerfen sich per ssh
#einloggen. Alle anderen Gruppen wird der Login verwehrt. ’?’ und ’*’ duerfen
#als Jokerzeichen verwendet werden.
#Alternativ: DenyGroups. ACHTUNG: DenyGroups hat immer Vorrang vor AllowGroups
AllowGroups users root
#Benutzerkontrolle: Nur die Benutzer snoopy und root duerfen sich per ssh einloggen. Allen
#uebrigen wird der Login verboten. Alternativ koennen hier Benutzernamen in der Form
#root@beispiel.rechner.de angegeben werden, so dass der Benutzer root nur vom angegebenen
#Rechner einen Zugang erhaelt. ’?’ und ’*’ duerfen als Joklerzeichen verwendet werden.
#Alternativ: DenyUsers. ACHTUNG: DenyUsers hat immer Vorrang vor AllowUsers.
AllowUsers snoopy root
#Legt fest, dass der Super-User keine Passwort-Authentifizierung, sondern nur Public-Key
#Authentifizierung nutzen darf, Werte (yes, without-password, force-commands-only,no).
PermitRootLogin no
#Legt fest, dass weder die Datei $HOME/.ssh/environment, noch die ENVIRONMENT-Direktiven
#aus der Datei $HOME/.ssh/authorized_keys ausgewertet werden.
PermitUserEnvironment no
#Benennt die Datei aller autorisierten Schluessel. Die Pfadangabe ist entweder absolut oder
#(wie hier) relativ zum Home-Verzeichnis des Benutzers.
AuthorizedKeysFile .ssh/authorized_keys
#Gibt an, dass Challenge-Response-Authentication genutzt wird.
ChallengeResponseAuthentication yes
#Legt fest, dass Passwort-Authentifizierung genutzt wird (Standardwert)
PasswordAuthentication yes
#Legt fest, dass keine leeren Passwoerter verwendet werden duerfen.
PermitEmptyPasswords no
```

```
#Legt fest, dass Public-Key-Authentication verwendet werden darf.
PubkeyAuthentication yes
#HostKey der Protokoll-Version 1
#HostKey /etc/ssh/ssh_host_key
#Host-Schluessel der Protokoll-Version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
#Startet Secure-FTP (SFTP) als Subsystem des SSH-Servers.
Subsystem sftp /usr/lib/ssh/sftp-server
```

4. Formate wichtiger Dateien

Die Dateien `known_hosts` und `authorized_keys` Dateien unterliegen einem bestimmten Format, welches hier genauer beschrieben werden soll.

/`etc/ssh/ssh_known_hosts, /etc/ssh/known_hosts`

Für OpenSSH haben die Dateien ein einheitliches Format. Sie werden zeilenweise betrachtet, wobei eine Zeile den folgenden Aufbau hat:

Rechnernamen Schlüsseltyp Schlüssel Kommentar

Die Rechnernamen stellen eine Liste von Namensmustern dar, in denen die Jokerzeichen '*' und '?' vorkommen dürfen. Ebenso können IP-Adressen angegeben werden. Das '!' negiert die folgenden Einträge.

Als Schlüsseltyp kann einer der folgenden Werte angegeben werden: ssh-dss, ssh-rsa.

Der Schlüssel selbst ist im Base64-Format gespeichert.

Das Kommentarfeld am Ende der Zeile wird ebenso überlesen wie Leerzeilen und Zeilen die mit einem '#' beginnen.

Alle Felder einer Zeile werden durch Leerzeichen getrennt, so dass insbesondere zwischen den Mustern der Rechnernamen und im Schlüssel keine Leerzeichen erlaubt sind.

`$HOME/.ssh/authorized_keys`

Die Datei `$HOME/.ssh/authorized_keys` wird für Authentifizierung über Public-Keys genutzt. In jeder Zeile steht der öffentliche Schlüssel eines berechtigten Benutzers. Die Zeilen sind folgendermaßen aufgebaut:

`[Optionen] Schlüsseltyp Schlüssel Kommentar`

Das Feld Optionen muss nicht angegeben werden. Sinnvoll ist der Einsatz der Option `from="Rechnername"`. Sie benennt eine Liste von Rechnernamen oder IP-Adressen, auf die der Hostname des Clienten passen muss. Die Jokerzeichen '*' und '?' dürfen ebenso genutzt werden wie die Negation '!'.

5. scp und sftp

Secure Copy, kurz scp, ist als Ersatz für rcp gedacht. Die Daten werden durch eine verschlüsselte Verbindung transferiert. Ansonsten ist scp ein normaler ssh-Client, der die Authentifizierungsmechanismen und Konfigurationen des SSH-Klienten aus Kapitel 2 nutzt.

Das Format der scp-Aufrufs ist folgendes:

`scp [Optionen] [[nutzer@]quellhost:]quellen [[nutzer@]zielhost]:ziel`

Zur Angabe des Benutzernamens nutzt scp das "eMail-Format" (user@host). Wird kein Benutzername angegeben, so wird, wie bei ssh, der Inhalt der Variablen $USER genutzt. scp arbeitet wie der Unix-Befehl cp. Quelldateien können in Verzeichnisse kopiert werden, bestehende Dateien werden überschrieben, und werden mehrere Dateien kopiert, so muss das Ziel ein Verzeichnis sein. Folglich

können also auch Shell-Jokerzeichen (z.B. '*') genutzt werden. Sollen ganze Verzeichnisse kopiert werden, so muss die Option -r gesetzt werden, da ansonsten der Kopiervorgang abbricht. Symbolische Links werden im Zielverzeichnis nicht erstellt, sondern der Inhalt der verknüpften Datei wird unter dem Namen des Links abgelegt. Soll in das Home-Verzeichnis des Benutzers auf dem Server kopiert werden, so kann der Zielpfad weggelassen werden. Relative Pfadnamen interpretiert scp immer in Bezug auf das Heimatverzeichnis des Benutzers. Der Doppelpunkt hinter dem Hostnamen darf allerdings nicht weggelassen werden, da scp das Ziel sonst als lokale Datei oder Verzeichnis interpretiert und dementsprechend kopiert.

Sonderzeichen müssen maskiert werden. Die Shell-Joker-Zeichen müssen vor der lokalen Shell versteckt werden. Bei Dateinamen, die Leerzeichen enthalten und von entfernten Rechnern kopiert werden sollen, müssen die Leerzeichen für die entfernte Shell maskiert werden. Das folgende Beispiel zeigt äquivalente Aufrufe:

```
scp user@zam001:Datei\\\ mit\\\ Leerzeichen lokale_Datei
```

```
scp 'user@zam001:Datei\ mit\ Leerzeichen' lokale_Datei
```

sftp dient ebenfalls, wie scp, dem verschlüsselten Kopieren von Dateien und Verzeichnissen zwischen zwei Rechnern. Im Gegensatz zu scp ist sftp interaktiv und nutzt die Befehle des bekannten FTP-Protokolls. Der sftp-Server wird standardmaessig installiert und sollte nicht deinstalliert werden, da scp auf dessen Dienste zur Dateiübertragung zurückgreift.

Der sftp-Client kann wie folgt aufgerufen werden:

```
sftp [benutzer@]rechner:[:verzeichnis]
```

Mit diesem Befehl, meldet man sich als Benutzer an und wechselt sofort in das angegebene Verzeichnis. Wie bei allen ssh-Clients wird der Inhalt der USER-Umgebungsvariablen genutzt, wenn kein Benutzername angegeben wurde. Nach erfolgreichem Login kann der Benutzer interaktive Befehle erteilen.

6. FAQ

a. Können X11-Verbindungen über SSH weitergeleitet werden?

Ja. Dazu muss in der Konfigurationdatei des SSH-Klienten die Option ForwardX11 und in der Konfiguration des Servers die Option X11Forwarding auf yes gesetzt sein.

b. Können X11-Verbindungen nach einem „su -“-Kommando weitergeleitet werden?

Ein weiteres Problem ergibt sich häufig, wenn ein Login als unprivilegierter Benutzer vorgenommen wird und mittels `su` Super-User-Rechte angenommen werden. Die X11-Weiterleitung kann dann folgendermaßen erreicht werden:

```
ksh> su –
Password:
ksh> export DISPLAY=localhost:10.0
ksh> xauth –f /home/`who am i | awk{‘print $1’}`/.Xauthority nextract – $HOSTNAME/unix:10.0
| xauth nmerge –
ksh> xlogo
```

Nach diesen Schritten, sollte das X-Logo auf dem Bildschirm erscheinen.

Erfolgte der Login von einer Windows-Maschine und ist dort X11-Weiterleitung eingeschaltet, so braucht nach su die obige Befehlsfolge nicht ausgeführt zu werden.

c. Wie kann der SSH-Server sicher gegen unbefugte Zugriffe konfiguriert werden?

Die Antwort auf diese Frage ist sehr vielfältig und Abhängigkeit von der Einsatzumgebung des Servers. Daher werden hier verschiedene Szenarien beschrieben:

1.Wird kein SSH-Server benötigt, so ist der sinnvollste die Abschaltung des Dienstes. Dieses erreicht man entweder unter Yast2, System, Runlevel Editor, sshd aus der Liste wählen und die Schaltfläche „Deaktivieren“ wählen.

2.Falls der SSH-Server benötigt wird, so kann man den Zugang über iptables oder eine andere Personal Firewall einschränken. So ist es zum Beispiel sinnvoll, den SSH-Zugriff nur für JuNet-IP-Adressen (134.94.xxx.xxx) zu erlauben. Damit kann über den Einwahlzugang und die VPN-Zugänge weiterhin auf den Rechner zugegriffen werden. Weitere detaillierte Anweisungen kann in der TKI-402 gefunden werden.

3.Meistens wird der SSH-Server mit den Bibliotheken des TCPWrappers übersetzt. Auf diese Art und Weise kann der Zugang eingeschränkt werden, wenn iptables oder eine andere Personal Firewall nicht vorhanden sind. Dazu müssen die Dateien `/etc/hosts.allow` und `/etc/hosts.deny` editiert werden. Soll der Zugriff nur für JuNet-Rechner und von localhost erlaubt werden, so können folgende Einstellungen gesetzt werden:

```
/etc/hosts.allow:
sshd : 134.94.0.0/255.255.0.0 127.0.0.1/255.255.255.255 : ALLOW /etc/hosts.deny:
sshd : ALL : DENY
```

4.Falls kein Personal Firewall und keine TCPWrapper-Unterstützung auf dem Rechner vorliegt und eine externe Kommunikation des Rechners nicht notwendig ist, so kann das Routing des Rechners vom Administrator eingeschränkt werden. Dazu wird die Default-Route gelöscht und eine neue Route für das Forschungszentrum eingetragen. Die folgenden Schritte sind durchzuführen:

```
ksh> route –n

 Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
134.94.168.0 0.0.0.0 255.255.248.0 U 0 0 0 eth0
169.254.0.0 0.0.0.0 255.255.0.0 U 0 0 0 eth0
127.0.0.0 0.0.0.0 255.0.0.0 U 0 0 0 lo
0.0.0.0 134.94.168.1  0.0.0.0 UG 0 0 0 eth0
ksh> route add –net 134.94.0.0/16 gw 134.94.168.1 metric 1
ksh> route delete –net default
```

Fett gedruckt in der letzten Zeile der Routing-Tabelle findet sich das sog. Default-Gateway. Diese IP-Adresse wird dann in der Zeile route add als Gateway hinter der Option gw eingetragen.

Der root-Zugriff per ssh sollte nach Möglichkeit unterbunden oder auf wenige Rechner eingeschränkt eingeschränkt werden (vgl. AllowUsers in Kapitel 3 Abschnitt c). Super-User-Rechte sollten über den Befehl „su“ angenommen werden. FAQ b zeigt, wie man dennoch X11-Verbindungen weiterleiten kann.

d. Soll Public-Key-Authentifizierung genutzt werden?

Ja. SSH-Identitäten identifizieren einen Benutzer eindeutig. Sie basieren auf Public-Key-Kryptographie und werden durch eine Passphrase geschützt. Sie bieten eine höhere Sicherheit beim Login als Passworte. Zusammen mit dem SSH-Agenten stellen sie eine Arbeitserleichterung dar.

e. Wie wird ein Schlüsselpaar erstellt?

Ein Schlüsselpaar wird mit dem Programm ssh-keygen generiert:

```
ksh> ssh-keygen –b 2048 –t rsa –f myid
```

Mit diesem Befehl wird ein 2048-Bit großer Schlüssel generiert vom Typ rsa. Die Option –t nimmt als mögliche Werte rsa oder das an. Der private Schlüssel wird in der Datei myid des aktuellen Verzeichnisses gespeichert, der öffentliche Schlüssel in der Datei myid.pub. Abschließend wird der Benutzer zur Eingabe einer Passphrase aufgefordert.

f. Wie benutze ich meine SSH-Identität?

Damit die erzeugt SSH-Identität genutzt werden kann, muss der öffentliche Schlüssel auf dem SSH-Server in der Datei `$HOME/.ssh/authorized_keys` eingetragen werden. Dazu gibt es zwei verschiedene Möglichkeiten:

Mit dem Programm `ssh-copy-id user@host` wird der öffentliche Schlüssel nach einer erfolgreichen Passwortauthentisierung automatisch an die Datei der autorisierten Schlüssel angehangen.
Die zweite Möglichkeit erfordert mehr „Handarbeit“. Zunächst wird der öffentliche Schlüssel mit scp zum Server kopiert. Danach erfolgt ein Login auf dem Server, so dass der Inhalt der myid.pub mit cat an die Datei `authorized_keys` angehangen werden kann. Das folgende Beispiel illustriert das Vorgehen:

```
ksh> scp myid.pub user@host:.ssh/.
ksh> ssh user@host
user@host’s password:
[…]
ksh@host> cd .ssh
ksh@host> cat myid.pub >> authorized_keys
ksh@host> logout
ksh> ssh user@host
Enter passphrase for key ‘myid’:
[…]
ksh>
```

Die öffentlichen SSH-Schlüssel können unterschiedliches Format haben. Man unterscheidet zwischen OpenSSH und SECSH. Das Program ssh-keygen ist in der Lage, Schlüssel zwischen beiden Formaten zu konvertieren. Mittels der Option –e wird eine angegebene Datei vom OpenSSH-Format in das SECSH-Format übertragen. Mit der Option –i geschieht dies in der umgekehrten Richtung.

g. Was ist der SSH-Agent und wie verwendet man ihn?

ssh-agent, der sogenannte SSH-Agent, lädt die privaten Authentifizierungsschlüssel eines Benutzers und übernimmt für ihn die Authentifizierung. Schlüssel werden mit dem Hilfsprogramm `ssh-add` zum Agenten hinzugefügt. Ist der Schlüssel über eine Passphrase geschützt, so wird diese vor dem Laden abgefragt. Beide Programme ermöglichen eine einfache und passwortlose Authentifizierung. Dazu muss der SSH-Agent gestartet werden und mittels `ssh-add` muss ein Schlüssel hinzugefügt werden. Versucht ein Client sich nun mit einem Server zu verbinden und erfolgt die Authentifizierung über öffentliche Schlüssel, so übernimmt der Agent die notwendigen Aktionen zur Authentifizierung. Wichtig ist, dass auf dem Server der öffentliche Schlüssel in der Datei `$HOME/.ssh/authorized_keys` abgelegt ist. Die Schlüssel werden vom Agenten in einem geschützten, privaten Speicherbereich gehalten, so dass von außen nicht darauf zugegriffen werden kann. Alle Kindprozesse der Shell haben somit Zugriff auf den Schlüssel.
Anwendungsbeispiel:

```
ksh> ssh-agent $SHELL
ksh> ssh-add
Enter passphrasse for /home/user/.ssh/myid:
ksh> ssh-add –L
ssh-rsa ……
```

Der letzte Befehl `ssh-add –L` listet alle Identitäten, die dem SSH-Agenten bekannt sind auf. Der nächste SSH-Login auf einem Rechner wird dann vom SSH-Agenten automatisch ausgeführt.

h. Wo finde ich weitere Dokumentation?

An dieser Stelle wird eine Liste für weitergehende Literatur empfohlen:

SSH, The Secure Shell – The Definitive Guide, Daniel J. Barrett & Richard E. Silverman, O’Reilly, 2001, ISBN: 0596008953

SSH kurz & gut, Sven Riedel, O’Reilly, 2004, ISBN:3897212692

Manual Pages http://www.openssh.org/manual.html