RRZN

Die Benutzung von ssh unter Unix

1 Überblick

ssh (secure remote shell, im folgenden *secure shell*) ist ein Programm, das dazu dient, sich über das Netz auf einen entfernten Rechner aufzuschalten, dort Kommandos abzusetzen oder Daten von einem Rechner zum anderen zu übertragen. Dabei bietet *secure shell* Authentisierung und eine verschlüsselte und dadurch sichere Kommunikation über das Netz.

1.1 Kommandos

secure shell deckt folgende remote Anwendungen ab:

- remote login,
- remote execute,
- remote copy,
- remote X11 clients
- und andere TCP/IP-Anwendungen.

secure shell bietet folgende Kommandos:

`ssh` zum Ausführen von Kommandos auf entfernten Rechnern, unter anderem X11-Clients → ersetzt `rsh`

`slogin` für Aufschalten auf entfernte Rechner → ersetzt `telnet` und/oder `rlogin`

`scp` Kopieren von Dateien → ersetzt `rcp` (`ftp`)

Zur Unterstützung der Authentisierungsvorgänge gibt es die Kommandos `ssh-keygen`, `ssh-agent` und `ssh-add` .

1.2 Sicherheit

Kommandos wie `telnet` und `ftp` sind unsicher. Bei ihrer Benutzung geht beispielsweise das Paßwort des Benutzers im Klartext über das Netz und kann „abgehört“ werden. Auch die remote Nutzung von X11-Clienten birgt Unsicherheiten. Wenn man z.B. über das Kommando `xhost` Zugriffsrechte zuteilt, können andere Nutzer auf dem Client-Rechner den Bildschirminhalt lesen.

secure shell beseitigt diese Unsicherheiten, indem es kryptographische Authentisierung mittels *host keys* und *user keys* unterstützt. X11-Verbindungen werden automatisch umgeleitet. Alle *secure shell*-Sitzungen laufen verschlüsselt ab.

Dadurch gewährleistet *secure shell* einen Schutz vor

- IP spoofing,
- Netzschnüfflern,
- Abhören von Passwörtern,
- Hijacking von Sitzungen,
- Abhören von X11.

1.3 Vorteile

Die Verwendung von *secure shell* bietet Ihnen eine einfache Möglichkeit, Ihre Kennung vor Fremdzugriffen zu schützen. Es findet eine RSA-Authentisierung (s. 2.3) statt und die Kommunikation wird verschlüsselt.

Wenn Sie remote X11-Clienten mit *secure shell* starten, können Sie sich über die bequeme Handhabung freuen. Im Gegensatz zu den Verfahren mit `xhost` oder `xauth` müssen Sie keinen Befehl zur Autorisierung eingeben. Auch die Variable `DISPLAY` müssen/dürfen Sie nicht setzen, sie wird automatisch angepaßt. *secure shell* realisiert dies über einen virtuellen X-Server auf dem Client-Rechner. Von dort läuft die Kommunikation über einen sicheren Kanal zum lokalen X-Server.

Der Systemverwalter (und Benutzer) kann *secure shell* umfangreich konfigurieren. Die Konfigurationsdateien werden in Kapitel 3 beschrieben.

Wenn eine sichere Kommunikation nicht möglich ist, weil *secure shell* auf dem entfernten Rechner nicht installiert ist, greift der rsh-Fallback. In diesem Fall wird die Kommunikation – natürlich unsicher –mit dem (`rsh`, `rcp`)-Protokoll weitergeführt.

1.4 Nachteile

secure shell ist keine Standard-Software für Unix-Systeme. Daher fällt für den Systemverwalter ein gewisser Aufwand für Installation, Konfiguration und Pflege an.

secure shell unterstützt zunächst nur die oben genannten Dienste. Die Benutzung weiterer Dienste (beispielsweise ftp) ist zwar möglich, jedoch umständlich zu handhaben.

Ein Nachteil von *secure shell* ist der Zeitverlust durch die Ver- und Entschlüsselung. Dabei ist die Wahl des Verschlüsselungsverfahrens entscheidend. In Kapitel 3 sind die möglichen Verfahren aufgelistet. Vom Systemverwalter wird ein default vorgegeben, den Sie mit einer Option oder durch Konfigurierung übersteuern können. Zeitverluste von weniger als 10% sollten Sie jedoch bei der Rechengeschwindigkeit heutiger Computer in Kauf nehmen, wenn Sie als Gegenleistung mehr Sicherheit und Komfort erhalten.

2 Funktionsweise der ssh

2.1 Das Client-Server-Konzept

Die *secure shell* arbeitet nach einem Client-Server-Konzept. Der Client-Prozeß fordert einem entfernten Rechner Dienste ab. Auf dem entfernten Rechner läuft ein Server-Prozeß, der diese Dienste befriedigt.

Client Die Client-Prozesse der *secure shell* sind `ssh` (`slogin` ist ein symbolischer Link auf `ssh`) und `scp`.

Server Der Server-Prozeß ist der `sshd` (*secure shell daemon*). Er muß beim Boot gestartet werden. Er lauscht auf Port 22, ob von irgendwelchen Clienten Anforderungen kommen. Für jede Anforderung wird ein neuer `sshd`-Kindprozeß generiert.

2.2 Kryptographie

secure shell benutzt ein Verschlüsselungsverfahren zur Authentifizierung. Die Kommunikation selbst wird ebenfalls verschlüsselt. Wie dies genau vor sich geht, soll im folgenden beschrieben werden.

Bei einer Verschlüsselung wird der zu übertragende Text (*plain text*, im folgenden mit P bezeichnet) vom Sender verschlüsselt oder kodiert. Zur Verschlüsselung dient ein Schlüssel (*key*, im folgenden K), das Ergebnis ist der kodierte Text (*cipher text*, im folgenden C).

$P*K \rightarrow C$

Der verschlüsselte Text wird über das Netz übertragen und vom Empfänger wieder dekodiert. Der Empfänger muß zur Entschlüsselung den inversen Schlüssel (*inverse key*, im folgenden K^{-1}) kennen.

$C*K^{-1} \rightarrow P$

Für den Schlüssel und sein Inverses gilt also der Zusammenhang

$K*K^{-1}=1$

Es werden zwei grundsätzlich verschiedene Verschlüsselungsverfahren unterschieden, die symmetrische und die asymmetrische Verschlüsselung.

Symmetrisches Verfahren (Private Key Verfahren)

Charakteristisch bei diesem Verfahren ist, daß der Schlüssel und sein Inverses identisch sind. Das heißt es gilt:

$K = K^{-1}$

Beispiele: DES, IDEA, 3DES, blowfish, arcfour(rc4)

Schlüssellänge: 64 oder 128 bits

Dieses Verfahren ist effektiv und im Vergleich zum asymmetrischen schneller.

Das Problem am symmetrischen Verfahren ist die Übertragung des Schlüssels. Da sowohl Sender als auch Empfänger den gleichen Schlüssel benutzen, darf dieser natürlich nicht

im Klartext über ein unsicheres Netz übertragen werden.

Asymmetrische Verfahren (Public Key Verfahren)

Bei diesem Verfahren sind der Schlüssel und sein Inverses nicht identisch, das heißt es gilt

$K \neq K^{-1}$

Es existieren zwei Schlüssel, wobei der eine in der Regel zur Verschlüsselung (*encryption key*, im folgenden E) und der andere zur Entschlüsselung (*decryption key* oder *private key*, im folgenden D) verwendet wird. Der *encryption key* ist öffentlich und wird an andere Kommunikationspartner weitergegeben. Daher wird er auch als *public key* bezeichnet. Der *decryption key* ist nur dem Empfänger bekannt und wird als *private key* bezeichnet.

Für die Schlüssel gilt der Zusammenhang E*D = 1. Wichtig ist außerdem, daß der *decryption key* nicht aus dem *encryption key* hergeleitet werden kann.

$E \not\Rightarrow D$

Schlüssellänge: 512 bits

Beispiele: RSA, RSAREF

2.3 Ssh-Protokoll

secure shell realisiert eine sichere Verbindung, indem die beiden oben beschriebenen Verfahren kombiniert werden.

Client und Server benutzen zur Authentisierung ein asymmetrisches Verfahren (RSA). Der Client generiert einen Schlüssel, den *session key*, der (RSA-)verschlüsselt an den Server übermittelt wird. Die eigentliche Sitzung wird mit dem *session key* im symmetrischen Verfahren verschlüsselt.

Im einzelnen läuft das `ssh`-Protokoll in folgenden Schritten ab;

1. Der Client eröffnet die Verbindung zum Server.
2. Der Server sendet seinen *public host key* und einen *public server key*, der im Speicher steht und der sich stündlich ändert..
3. Der Client überprüft den *host key*, indem er ihn mit dem Eintrag in `/etc/ssh_known_hosts` oder `~/.ssh/known_hosts` vergleicht.
4. Der Client generiert einen *session key*, und verschlüsselt diesen mit den beiden erhaltenen *public keys*. Der *session key* dient zur Verschlüsselung der weiteren Kommunikation mit dem symmetrischen Verfahren.
5. Der Server entschlüsselt den *session key* mit seinen beiden *private keys*. Danach schickt er eine mit dem *session key* verschlüsselte Bestätigung.

Nach Empfang dieser Bestätigung ist der Server authentisiert und es besteht eine verschlüsselte Verbindung. Als nächstes muß sich der Benutzer mit seinem Client-Rechner beim Server authentisieren.

6. Der Benutzer meldet sich mit seinem Usernamen an. Für das weitere Vorgehen gibt es drei Alternativen, die in der folgenden Reihenfolge abgefragt werden.

a. RSA-Authentisierung auf Rechnerebene
Für dieses Verfahren muß der Client-Rechner auf dem Server entweder in der Datei `/etc/hosts.equiv` oder `/etc/shosts.equiv` vom Systemverwalter oder in der benutzereigenen Datei `~/.rhosts` oder `~/.shosts` eingetragen sein. Das systemweite Verfahren funktioniert nur mit gleichen Benutzernamen auf beiden Rechnern, in der benutzereigenen Datei kann ein abweichender Benutzername stehen.
Der Server generiert eine sogenannte Challenge und verschlüsselt diese mit dem *public key* des Client-Rechners. Der Client entschlüsselt diese mit seinem *private key* und bildet eine Prüfsumme (MD5-Quersumme) aus Challenge und *private key*. Dieser sogenannte Response wird vom Server verifiziert.

b. RSA-Authentisierung auf Benutzerebene
Sie können mit dem Kommando `ssh-keygen` ein eigenes Schlüsselpaar generieren. Der *private key* wird unter `~/.ssh/identity`, der *public key* in der Datei `~/.ssh/identity.pub` abgelegt.
Ihr *public key* muss auf dem Server in der Datei `~/.ssh/authorized_keys` abgelegt werden. Mit diesem Key wird dann wieder ein Challenge/Response zwischen Server und Client durchgeführt.
Bei der Schlüsselgenerierung wird nach einer *passphrase* gefragt. An dieser Stelle können Sie ein zusätzliches Paßwort angeben. Dieses Paßwort wird immer dann abgefragt, wenn Sie eine neue Verbindung zu einem Server aufbauen.
Wenn Sie als *passphrase* nichts angeben, wird kein zusätzliches Paßwort generiert. Die *passphrase* dient dazu, Ihren *private key* vor Mißbrauch zu schützen. Wenn Sie keine *passphrase* angeben, hat jeder Benutzer, der Zugriff auf Ihre lokale Kennung hat, automatisch Zugriff auf Ihre Kennung auf entfernten Rechnern.
Zur Vereinfachung können Sie die *passphrase* mit den Kommandos `ssh-agent` und `ssh-add` an einen Agenten übermitteln, so daß nicht bei jedem Zugriff die *passphrase* eingegeben werden muß.

c. Paßwort Authentisierung
Sie geben Ihr Unix- Paßwort an.

Theoretisch sind noch weitere Authentisierungsverfahren möglich.

7. Vorbereitungsphase: Der Client sendet seine requests zum Server.

8. Interaktive Phase. Kommando-Abarbeitung. Am Ende sendet der Server seinen exit status, danach beendet der Client die Sitzung.

3 Beispiele

Das Kommando `ssh` kann auf zwei Arten benutzt werden: Sie können sich auf einem Rechner aufschalten oder ein Kommando auf dem entfernten Rechner absetzen. Statt `ssh` können Sie auch das Kommando `slogin` verwenden.

3.1 Aufschalten auf einen entfernten Rechner

Zum Einloggen auf einem entfernten Rechner dient das Kommando

```
ssh host
```

`host` steht dabei für den Namen des Zielrechners. Wenn Sie auf dem entfernten Rechner eine andere Kennung haben, müssen Sie diese zusätzlich anfügen:

```
ssh username@host
```

oder

```
ssh - l username host
```

Wichtige Optionen von `ssh` sind:

-v	verbose, gibt Auskunft über Authentisierung
-c	gibt den Verschlüsselungsalgorithmus an, z.B.: idea,des,3des,blowfish
-C	komprimiert die zu übertragenen Daten
-L	ermöglicht Port-Umlenkung
-q	unterdrückt Warnungen und Diagnosemeldungen
-V	gibt die Versionsnummer aus

Sie können sich mit dem Kommando

ssh -l zzzzmq unics

als Benutzer `zzzzmq` auf den Rechner `unics` aufschalten. Das Kommando

xterm -e ssh sun222x &

startet ein `xterm`, in dem eine shell auf dem Rechner `sun222x` geöffnet wird.
Als Loginname wird der gleiche Benutzername wie auf dem Startsystem benutzt.

X-Clienten können nach dem Aufschalten mit `ssh` direkt gestartet werden. Sie brauchen keine Authentifizierung mit `xauth` oder Autorisierung mit `xhost` vorzunehmen. Ebenso brauchen bzw. dürfen Sie die Variable `DISPLAY` nicht setzen.

3.2 Kommandos auf entfernten Rechnern absetzen

Mit `ssh` können Sie einzelne Kommandos auf einem entfernten Rechner absetzen. Dazu rufen Sie `ssh` in der folgenden Form auf:

```
ssh host kommando
```

Gegebenenfalls müssen Sie auch hier Ihren Benutzernamen hinzufügen. Bei dieser Art des Aufrufs sind die folgenden Optionen wichtig:

`-f`	startet `ssh` im background nach Authentisierung, ermöglicht Passworteingabe

`-n` lenkt stdin von `/dev/null` um.
Dies ist nötig, um `ssh` im Hintergrund zu halten.

Das folgende Kommando startet den X-Clienten `netscape` im Hintergrund auf dem Rechner `xserv`:

```
ssh -n xserv netscape -install &
```

Statt der Option `-n` kann auch `-f` benutzt werden. Hier ist, falls erforderlich, die Eingabe des Paßwortes möglich, bevor der X-Client `emacs` im Hintergrund gestartet wird. Der X-Client wird unter dem Benutzernamen `nhzzmarq` auf dem Rechner `berte.zip.de` gestartet.

```
ssh -f nhzzmarq@berte.zip.de emacs
```

Es ist nicht nötig, den Zugriff über das Kommando `xhost` oder MIT-Magic-Cookies zu erlauben.

3.3 Daten kopieren

Das Kommando `scp` dient zum Kopieren von Daten zwischen Rechnern. Es wird in der Form

```
scp sourcefile destinationfile
```

aufgerufen. Dabei können Sie `sourcefile` und `destinationfile` in der Form `[[username@] host:]path` angeben. Das Kommando

```
scp -c blowfish datei zzzzmq@unics:datei
```

kopiert eine Datei namens `datei` in das Heimatverzeichnis des Benutzers `zzzzmq` auf dem Rechner `unics`. Dabei wird als Verschlüsselungsalgorithmus blowfish benutzt.

3.4 Konfigurationsdateien und Dateien für die Schlüssel

Die ssh-Kommandos können über Optionen bei der Eingabe des Kommandos sowie über benutzereigene oder systemweite Konfigurationsdateien beeinflußt werden. Dabei werden die Informationsquellen mit Prioritäten gemäß der oben angegebenen Reihenfolge berücksichtigt.

In den Dateien `/etc/ssh_config` und `$HOME/.ssh/config` wird die Konfiguration des Clienten (beispielsweise das Verschlüsselungsverfahren) festgelegt. Die Dateien `/etc/environment` und `$HOME/.ssh/environment` dienen zur Einstellung der Umgebung auf dem Zielrechner. Hier können Sie zum Beispiel Drucker oder Suchpfade festgelegen. Hinweise auf mögliche Einträge und deren Syntax gibt die Manualseite zu ssh (`man ssh`).

Die für die Authentisierung nötigen keys werden in verschiedenen Dateien systemweit bzw. benutzerspezifisch abgelegt.Die *host keys* für die Authentisierung der Server liegen für das gesamte System in `/etc/ssh_known_hosts`. Die Einträge müssen per Hand vom Systemverwalter vorgenommen werden. Ansonsten wird der *host key* automatisch in der benutzerspezifischen Datei `$HOME/.ssh/known_hosts` abgelegt, sobald eine Anfrage erfolgreich bearbeitet wurde.

Eine Client-Authentisierung kann wie für das rlogin-Kommando systemweit in den Dateien `/etc/hosts.equiv` oder `/etc/shosts.equiv` festgelegt werden. Entsprechend können Sie als Benutzer Einträge in die Dateien `$HOME/.rhosts` bzw. `$HOME/.shosts` vornehmen.

Wenn Sie sich als Benutzer mit dem Kommando `ssh-keygen` ein eigenes Schlüsselpaar generieren (siehe 2.3 Abschnitt 6 Alternative b), wird Ihr *public key* in der Datei `$HOME/.ssh/identity.pub` und Ihr *private key* in der Datei `$HOME/.ssh/identity` abgelegt.

Zusätzlich müssen Sie Ihren *public key* auf dem Zielrechner in der Datei `$HOME/.ssh/authorized_keys` ablegen.

3.5 Das RRZN-Kommando `sshtool`

Um die Benutzung der ssh zu vereinfachen und komfortabel zu gestalten, wurde am RRZN das `sshtool` entwickelt. `sshtool` ist in Tcl/Tk geschrieben und somit ein grafisches Programm. Durch den Aufruf

```
sshtool &
```

wird ein Fenster geöffnet, in dem sich je ein Auswahl-Menü für den Namen des Zielrechners (*Hostname*) und den Benutzernamen (*Username*) befinden, die noch um jeweils ein Textfeld erweitert sind. Sie haben die Möglichkeit entweder über das Auswahlmenü oder durch einen Eintrag in das Textfeld den Namen des Zielrechners und den Benutzernamen anzugeben.

Anschließend wird durch das Drücken des Button mit der Aufschrift *Run it* ein `xterm`-Fenster mit einer ssh-Verbindung zum ausgewählten Host generiert. Das `sshtool`-Fenster bleibt für weitere Aktionen am Bildschirm. Zusätzlich gibt es einen Button mit der Aufschrift *verbose* zum Ein- bzw. Ausschalten des verbose-Modus. Der *Exit*-Button dient zum Verlassen des `sshtools`.

Konfiguration über Ressourcen

Für die Menüs gibt es Default-Einstellungen, die Sie um eigene Einträge für Rechner- und Benutzernamen ergänzen können. Dazu müssen Sie in Ihrem Heimatverzeichnis eine Datei `Sshtool` angelegen mit Einträgen der Form:

```
*userlist: nhzzgdcm zzhiwi Klemens
```

für die Benutzernamen, bzw.

```
*hostlist: s5b001 s5b002 c34004 berte.zib.de studserv
```

für die Rechnernamen.

Optionen

Das `sshtool` unterstützt im Kommando-Modus noch einige Optionen, um den Gebrauch in Window-Manager-Konfigurationen zu erleichtern. Es gilt folgende Syntax:

```
sshtool [-verbose] [-e] [-r] [-l username] [hostname]
```

Die Optionen haben folgende Bedeutung:

`-verbose` schaltet den verbose-Mode an

`-e` bewirkt ein Verlassen von `sshtool` nachdem der *Run it*-Button betätigt wurde.

`-r` generiert eine ssh-Verbindung, ohne ein `sshtool`-Fenster zu öffnen.

`-l` dient dazu, einen Usernamen vorzugeben.

Folgende Einträge in Windowmanagerdateien wie .zB. `.ctwmrc, .mwmrc, .dtwmrc` zum Zugriff auf remote-Hosts sind sinnvoll und in den RRZN-Dateien aufgenommen:

```
"UniCS"             !"sshtool -r unics &"
```

entspricht in der Wirkung

```
"UniCS"             !"xterm -title UniCS -e ssh unics &"
```

Zugriff auf VPP mit Eingabemöglichkeit des Usernamens:

```
"VPP"               !"sshtool -e vpp &"
```

Zugriff auf VPP mit vordefiniertem Usernamen und automatischem Start der Verbindung:

```
"VPP"               !"sshtool -e -l username vpp &"
```

3.6 Das Kommando ssh-keygen

Wenn Sie die RSA-Authentisierung auf Benutzerebene verwenden, müssen Sie sich mit dem Kommando `ssh-keygen` ein eigenes Schlüsselpaar anlegen und den *public key* auf den jeweiligen Zielrechnern deponieren. Dieses Verfahren birgt die Gefahr, daß jemand, der Ihr Unix-Login kennt, automatisch per ssh auch Zugriff auf Ihre Kennung auf den entfernten Rechnern hat.

Um hier eine zusätzliche Sicherheit zu haben, können Sie bei der Generierung der Schlüssel eine sogenannte *passphrase* angeben. Dieses zusätzliche Paßwort wird jedesmal abgefragt, wenn Sie eine neue ssh-Verbindung zu einem remoten Server aufbauen.

Der *ssh agent* kann Sie bei der Verwendung einer *passphrase* unterstützen, indem er die Übermittlung der *passphrase* zu den Zielrechnern übernimmt. Sie müssen die *passphrase* auf dem lokalen Rechner dann nur einmal eintippen, um sie dem Agenten mitzuteilen.

Um den *ssh agent* zu benutzen, müssen Sie zunächst den Agenten mit dem Kommando `ssh-agent` starten. Anschließend übergeben Sie die *passphrase*, indem Sie das Kommando `ssh-add` aufrufen und das zusätzliche Paßwort eingeben.

Wenn Sie in jeder Sitzung mit dem Agenten arbeiten wollen, empfiehlt es sich, die Kommandos `ssh-agent` und `ssh-add` automatisch beim Einloggen ausführen zu lassen. Dazu können Sie die Zeilen

```
eval $(ssh-agent)
ssh-add
```

in die Datei `.profile` einfügen. Sie werden dann bei jedem Aufschalten auf ihrem lokalen Rechner zunächst aufgefordert, Ihre *passphrase* einzugeben.

In dem `.profile` des RRZN werden für die Workstations die beiden Kommandos aktivert,wenn Sie mit `ssh-keygen` ein eigenes Schlüsselpaar generiert haben.

Anhang A Informationen

Der Urheber von ssh ist Tatu Ylönen, die dazugehörige Homepage lautet:

```
http://www.ssh.fi/
```

Die neueste Version (zum Erstellungszeitpunkt dieser Dokumentation Version 1.2.26) finden Sie auf dem ftp-Server des DFN:

```
ftp://ftp.cert.dfn.de/pub/tools/net/ssh
```

Die Unix-Version ist für nicht-kommerzielle Zwecke frei verfügbar (siehe Lizenzbestimmungen).

Eine FAQ-Liste (Frequently Asked Questions) können Sie unter folgendem URL abrufen:

```
http://www.uni-karlsruhe.de/~ig25/ssh-faq/ssh-faq.html
```

Auf der Seite

```
http://www.tac.nyc.ny.us/~kim/ssh
```

finden Sie eine Einführung zu ssh, insbesondere das Arbeiten mit eigenen Keys ist hier ausführlich erläutert.

Während der Erstellung dieser Dokumentation steht für Unix eine vollständig neue Version (Version 2.10.0) mit einem anderen Protokoll, dem ssh2-Protokoll, zur Verfügung. Diese Dokumentation bezieht sich ausschließlich auf das ssh1-Protokoll. Am RRZN läuft auf allen gängigen Unix-Sytemen die ssh in der Version 1.2.26, d.h. mit dem ssh1-Protokoll.

Das ssh2-Protokoll bietet einige Verbesserungen, u.a. sftp (secure ftp). Es ist allerdings bisher nicht vollständig implementiert und nicht kompatibel zur Version 1. Daher ist ssh2 noch nicht weit verbreitet und wird auch im RRZN z. Z. nicht eingesetzt.

Installationshinweise zur ssh2 finden Sie unter:

```
http://www.yl.is.s.u-tokyo.ac.jp/~ymmt/ssh2.html
```