---

ssh - Secure Shell

Bernd Lehle / Oliver Reutter

Zur Zeit werden im Internet meistens Protokolle verwendet, die leider zwei Schwächen aufweisen. Zum einen wird die gesamte Kommunikation unverschlüsselt durchgeführt, so daß bestimmte Programme (Packet Sniffer) bei login-Vorgängen, wie telnet und ftp, das Paßwort und den Benutzernamen abhören und aufzeichnen können. Das zweite Problem, das bei herkömmlichen Verfahren vor allem beim Einsatz in großen Netzwerkverbünden auftritt, ist die fehlende Authentifizierung beim Verbindungsaufbau. Hacker können DNS-Namen und die IP-Adressen von Rechnern fälschen (sogenanntes DNS- und IP-Spoofing) und sich so für jemand völlig anderen ausgeben. Zu den von diesem Hacking gefährdeten Protokollen zählen beispielsweise`rsh` und `rlogin`. Ein Programmpaket des Finnen Tatu Ylonen mit dem Namen ssh (Secure Shell) verspricht hier Abhilfe zu schaffen.

Das Gesamtpaket `ssh` besteht aus einem Server `sshd` und verschiedenen Klienten, die als vollwertiger Ersatz für `rsh`, `rcp` und `rlogin` gedacht sind. `ssh` führt im Gegensatz zu den anderen keine Authentifizierung mit IP-Adressen, sondern mit Hilfe des RSA-Verschlüsselungsverfahrens durch. Zugriff auf den Rechner wird nur der Maschine erlaubt, die im Besitz des geheimen (privaten) Teils des Schlüssels ist. Daneben kann dieser Zugriff nicht nur pro Maschine, sondern auch pro Benutzer gewährt werden. Nachdem die Kontaktaufnahme erfolgreich war, wird die Kommunikation zwischen beiden Maschinen verschlüsselt, sodaß ein Abhören nicht mehr möglich ist. Man kann selbst die Verbindungen von X11-Klienten zum X11-Server via `ssh` abwickeln, sodaß ein Abhören des Magic Cookies wirkungsvoll verhindert wird. Auch das verschlüsselte Tunneln einer beliebigen TCP-Verbindung (z.B. durch einen Firewall) ist möglich.

Verbindungsaufbau und Möglichkeiten der Authentifizierung

Bei den klassischen Protokollen `rsh`, `rlogin` und `rcp` funktionierte die Authentifizierung nach folgendem einfachen Schema. Einige Betriebssysteme schieben ein paar weitere Riegel vor, wenn versucht wird, sich als root einzuloggen.

1. Wenn die Maschine, von der der User kommt, auf der entfernten Maschine in `/etc/hosts.equiv` enthalten ist und der Login-Name auf beiden Seiten gleich ist, wird der Nutzer umgehend eingeloggt.
2. Wenn auf der entfernten Maschine `.rhosts` im Stammverzeichnis des Users vorhanden ist und dieses File den Namen der Maschine und den Login-Namen enthalten, von der der Nutzer kommt, wird der Benutzer eingeloggt. Ein Pluszeichen dient als Wildcard.

Diese beiden Methoden der Authentifizierung sind normalerweise nicht sicher und vom Server `sshd` ohne Verbindung mit anderen Formen der Authentifizierung nicht gestattet. Sie werden im

einfachsten Fall mit RSA-gestützter Rechner-Authentisierung verbunden. Bevor die beiden o.g. Möglichkeiten erfolgreich sind, wird also geprüft, ob der Host Key des Rechners, an dem der Nutzer sitzt verifiziert werden kann; nur dann wird der Login gestattet. Es wird also nur Zugriff auf Maschinen gestattet, die bekannt sind, was Name- und IP-Spoofing verhindert. Eine Möglichkeit des Spoofings bleibt dann natürlich der Erstkontakt. Gibt ein Eindringling seine eigenen Schlüssel an den Server, bevor dies der rechtmäßige Benutzer tun kann, wird er korrekt authentifiziert. Daher sollte ein Systemverwalter in seinem Netz darauf achten, daß nach der Installation des `ssh-` Pakets alle beteiligten Rechner als erstes ihre Schlüssel austauschen. Hierfür reicht ein einziger `ssh-` Kontakt zwischen den Rechnern.

Es gibt jedoch auch ein bisher nicht unterstütztes Authentisierungsverfahren, bei dem der einzelne Benutzer sich durch ein Schlüsselpaar ausweist, um ohne Paßwortkontrolle auf einen entfernten Rechner zu kommen. Dazu erzeugt der Benutzer mit dem Befehl `ssh-keygen` ein RSA-Schlüsselpaar. Die Schlüssel werden dann in einem Unterverzeichnis `$HOME/.ssh` / unter den Namen `identity` (geheimer Schlüssel) und `identity.pub` (öffentlicher Schlüssel) abgelegt.

Um nun einem entfernten Rechner die Authentifizierung mit diesem Verfahren zu erlauben, muß der öffentliche Schlüssel dorthin kopiert werden. Er muß auf dem Rechner in der Datei `$HOME/.ssh/authorized_keys` liegen. Diese Datei kann mehrere öffentliche Schlüssel von verschiedenen Maschinen enthalten, die auf diesen Rechner zugreifen dürfen.

Das `ssh`-Programm, das der Benutzer aufruft, gibt dem Server `sshd` das gewünschte Schlüsselpaar an. Nachdem der Server überprüft hat, ob der Schlüssel zugelassen ist, schickt er eine Zufallszahl mit dem öffentlichen Schlüssel verschlüsselt an den Klienten. Dieser kann diese Zahl nur wieder entschlüsseln, wenn er den privaten Schlüssel hat, allerdings ohne seinen privaten Schlüssel via Netz übertragen zu müssen.

Bei dieser Methode steht das File `$HOME/.ssh/authorized_keys` in Korrespondenz zu `.rhosts`, bietet aber im Gegensatz zum alten Verfahren sehr hohe Sicherheit. Man sollte sich allerdings bewußt sein, daß ein geknackter Account, dem auf dieser Art vertraut wurde, trotz aller Sicherheit sofort den Zugang zu allen Accounts erlaubt, die auf den geheimen Schlüssel vertrauen. Man kann den geheimen Schlüssel zwar analog zu PGP mit einer Passphrase sichern, was allerdings wenig Sinn macht, wenn man sich ohne Paßwort auf den entfernten Rechner einloggen will.

Falls alle obigen Methoden versagen sollten, wird der Benutzer nach seinem Paßwort gefragt. Dieses wird verschlüsselt übertragen, sodaß ein Packet Sniffer wirkungslos bleibt. Sollte der Vorgang vom Server gestattet werden, führt die entfernte Maschine entweder das - mitangegebene - Kommando aus oder öffnet eine normale Kommando- Shell, die man auch beim Kommando `rsh` oder `rlogin` erhalten hätte.

Für Nutzer die X11 einsetzen, bietet `ssh` große Vorteile. Die DISPLAY-Variable wird automatisch gesetzt und ein Cookie wird automatisch generiert. Auf der entfernten Maschine können somit sofort X11-Anwendungen gestartet werden, die verschlüsselt übertragen werden.

Welche Authentifizierungsverfahren der `sshd` akzeptiert und von welchen Rechnern er dies tut, kann in der Datei `/etc/sshd_config` eingestellt werden. So kann der Systemverwalter beispielsweise die Verwendung von `.rhosts` (oder dem `ssh`-Äquivalent `.shosts`) auf seinem System verbieten und Paßwort- bzw. RSA-Authentifizierung verlangen. Auch die Anbindung an den TCP-Wrapper (siehe BI. 3 96) kann realisiert werden, wenn beim Kompilieren die `libwrap` mitverwendet wird.

Installation des Pakets

ssh kann auf verschiedenen Systemen/Betriebssystemarchitekturen eingesetzt werden:

- Solaris 2.X (ab 2.3)
- SunOS 4.1.X
- AIX 3.2.5, 4.1
- HPUX 7.X, 9.X, 10.X
- IRIX 5.2, 5.3, 6.0
- SYS V 4.X
- BSD 4.X
- LINUX
- OS/2 ab Warp
- WIN NT/95
- MacOS

Diese Liste ist natürlich nicht vollständig und kann noch auf weitere Betriebssysteme ausgedehnt werden. Unter http://www.cs.hut.fi/ssh/ erhalten Sie weitere Informationen. Die zur Zeit (7/97) aktuelle Version ist ssh-1.2.20.

Nach dem Auspacken mit...

```
gunzip ssh-1.2.20.tar.gz
tar xvf ssh-1.2.20.tar
```

...erfolgt die Übersetzung und Installation der Programme mit

```
cd ssh-1.2.20
./configure
make
make install
```

Die Übersetzung sollte man am besten schon als `root` ausführen, spätestens jedoch bevor man `make install` startet, da sonst die Programme nicht korrekt installiert, der Host Key nicht erzeugt und die Zugriffsrechte nicht richtig gesetzt werden.

Noch ein Kommentar zum verwendeten Compiler: Es kann im Prinzip jeder Compiler eingesetzt werden, der den ANSI-Standard beherrscht. Bei den meisten muß dies explizit vorgeben werden. Ist auf der Maschine der GNU Compiler `gcc` nicht installiert, sollte in jedem Fall die Variable `CC` auf den zu verwendenden Compiler setzen und die Variable `CC-OPT` mit der entsprechenden Option füttern, sodaß der Compiler ANSI übersetzen kann. Dies muß auf jeden Fall vor der Benutzung von `configure` geschehen.

Damit die Maschine auch mit `ssh` erreicht werden kann, muß der `sshd` zuvor gestartet werden. Am besten sorgt man dafür, daß `sshd` schon beim Booten geladen wird. Normalerweise geschieht dies durch Anpassung der Datei `/etc/rc.local`

Unter Solaris stehen jedoch die Startup-Skripte in `/etc/init.d`

Hier ein Beispiel für `/etc/init.d/sshdaemon`

```
#! /bin/sh
# start/stop the secure shell deamon
case "$1" in
'start')
        # Start the ssh daemon
        if [ -f /usr/local/sbin/sshd ]; then
                echo "starting SSHD daemon"
                /usr/local/sbin/sshd &
```

```
        fi
        ;;
'stop')
        # Stop the ssh daemon
        PID=`/usr/bin/ps -e -u 0|/usr/bin/fgrep sshd|/usr/bin/awk '{print $1}'`
        if [ ! -z "$PID" ] ;  then
                /usr/bin/kill ${PID} 1>/dev/null 2>&1
        fi

        ;;
esac
```

Danach muß nur in `/etc/rc/03` ein symbolischen Link auf das Startup-Skript gesetzt werden:

```
ln -s /etc/init.d/sshdaemon /etc/rc2/S89sshdaemon
ln -s /etc/init.d/sshdaemon /etc/rc2/K89sshdaemon
```

Nach dem nächsten Shutdown steht dann die volle Funktionalität zur Verfügung (der Daemon kann natürlich auch mit `/etc/init.d/sshdaemon start` gestartet werden).

Es ist auch möglich, den `sshd` über den `inetd` zu starten. Dies erfolgt mit folgendem Eintrag in `/etc/inetd.conf` :

```
ssh stream tcp nowait root /usr/etc/sshd sshd
```

In `/etc/services` muß `ssh` dann auf Port 22 gesetzt sein. Grundsätzlich ist davon abzuraten den `sshd` über `inetd` zu starten, da dann bei jedem login-Vorgang der Server einen neuen Server-Key erzeugt, was etwa 30 Sekunden dauert.

Die öffentlichen Schlüssel der gewünschten Remote-Maschinen sollten der Datei `/etc/ssh_known_hosts` hinzugefügt werden. Da dies bei größeren Netzen unpraktisch und sehr zeitaufwendig sein kann, empfiehlt sich die Nutzung des Perl-Skriptes `make-ssh-known-hosts` :

```
make-ssh-known-hosts rus.uni-stuttgart.de > ssh_known_hosts
```

Diese Datei muß dann noch nach `/etc` verschoben werden bzw. auf allen gewünschten Rechnern im Verzeichnis `/etc` abgelegt werden.

Da `ssh` die Programme `rsh`, `rcp` und `rlogin` vollständig ersetzen kann, empfiehlt es sich, diese in `/etc/inetd.conf` zu deaktivieren. Es ist sogar möglich, diese Programme komplett zu löschen und einfach durch einen symbolischen Link auf `ssh` zu ersetzen. Die Benutzer merken dann keine Änderung. Lediglich wenn sie mit rlogin zu einem Rechner verbinden wollen, auf dem kein `sshd` läuft, bekommen sie eine Warnung, daß die Verbindung nicht verschlüsselt wird. Dann läuft das normale unverschlüsselte Protokoll ab.

Bedienung

Der login-Vorgang sieht nun wie folgt aus:

```
ssh -l bGates anywhere.somewhere.com -c 3des

bGates's password:

Last login: Mon Jun 30 11:06:12 1997 from winnie.somewhere.com

Sun Microsystems Inc. SunOS 5.5.1 Generic May 1996

Erase is Ctrl-H
```

```
anywhere>
```

Mit der Option `-l` gibt man `ssh` zu verstehen, daß man einen anderen Benutzernamen auf der neuen Maschine besitzt. Danach folgt der Name des Rechners . `-c 3 des` bedeutet, daß `ssh` das Verschlüsselungsverfahren Triple-DES einsetzen soll. In der gegenwärtigen Version unterstützt ssh die Verschlüsselungsverfahren IDEA, DES, Triple-DES, tss und arcfour. Durch `-c none` kann die Verschlüsselung komplett abgestellt werden. Das ist dann sinnvoll, wenn lediglich die Authentifizierung durch `ssh` notwendig ist. Bei der Übertragung von großen Datenmengen kann Kompression sinnvoll sein, was durch die Option `-C` erreicht wird. Die Option `-v` zeigt alle Aktionen an, die `ssh` gerade ausführt.

Weitere Features

Zusätzlich zu `ssh`, `sshd` und `ssh-keygen` enthält das Paket noch einige weitere Programme. Das wichtigste davon ist `scp` (Secure Copy). Es erlaubt das Kopieren von Dateien analog `rcp`. Dabei werden aber alle Sicherheitsmechanismen der Secure Shell verwendet und die Bedienung ist einfacher. Soll eine Datei auf einen bestimmten Account kopiert werden, der auch über `ssh` erreichbar ist, sieht das Kommando wie folgt aus: `scp file.name username@host.somewhere.edu:new.file.name`

Die Datei `file.name` wird damit ins Homeverzeichnis des Benutzers `username` auf dem Rechner `host.somewhere.edu` kopiert. Die Authentisierungsmechanismen sind dabei dieselben, wie bei einem login-Vorgang, d.h. man wird u.U. nach einem Paßwort gefragt oder muß Schlüssel hinterlegen. Das Kopieren funktioniert auch in der Gegenrichtung und mit absoluten Pfadnamen. Will man von dem Rechner die Paßwortdatei haben, sieht das Kommando so aus:

```
scp username@host.somewhere.edu:/etc/passwd
```

Zusätzlich zu nennen wäre dann noch der `ssh-agent` . Dieser erlaubt es, die Authentisierung der Verbindungen noch weiter zu verfeinern und für einzelne Verbindungen einzelne Schlüsselpaare zu verwalten und an andere Rechner weiterzugeben. Wir haben uns damit mangels Anwendung noch nicht beschäftigt.

Ansprechpartner in sicherheitsrelevanten Fragen

- sneakers@rus.uni-stuttgart.de
- dfncert-request@cert.dfn.de

Literatur

[1] Schneier, Bruce, Applied Cryptography, Wiley & Sons, 1996

[2] manpages zu `ssh` (1)

[3] Lehle, Bernd / Reutter, Oliver, PGP, BI. 3 96

Bernd Lehle, NA-5531
E-Mail: lehle@rus.uni-stuttgart.de

Oliver Reutter, NA-4513
E-Mail: Oliver.Reutter@rus.uni-stuttgart.de