

SSH - dem Lauscher keine Chance

Im Normalfall wird heutzutage jede TCP/IP-Verbindung im Klartext übertragen. Dies betrifft nicht nur die Daten und die Bildschirmausschnitte, die transportiert werden, auch Login und Paßwort gehen so ungeschützt über das Netz. Jede Station, die sich zwischen dem eigenen Rechner und dem Zielrechner befindet, ist in der Lage, diese Informationen mitzuschneiden und zu mißbrauchen. Heutzutage sind in fast jeder UNIX-Distribution dafür geeignete Werkzeuge vorhanden. Viele davon sind sogar frei erhältlich. Daß dies jeden auch nur etwas sicherheitsbewußten Anwender schaudern läßt, dürfte wohl klar sein.

Wie in diesem Heft an anderer Stelle zu lesen ist, kann man E-Mail und HTTP-Verbindungen ohne Probleme verschlüsseln. Wie sieht es aber mit `telnet`, `rlogin`, `rsh`, `rdist` und `rcp` aus? Seit 1995 existiert ein finnisches Programm, das es sich zur Aufgabe gemacht hat, diese Problematik wirksam zu behandeln. Dieses Programm heißt **Secure Shell (SSH)**; sein Einsatz und die Installation sollen im folgenden Artikel näher beschrieben werden.

Mit SSH kann man sich über ein Netzwerk bei einem anderen Rechner anmelden, um dort Befehle auszuführen und Dateien zu transportieren. Es benutzt dabei eine starke Authentifizierung und ermöglicht eine sichere Kommunikation durch unsichere Kanäle.

Wovor SSH schützt

Durch den Einsatz von starker Authentifizierung werden folgende Gefahren eliminiert:

- *IP Spoofing*: Ein entfernter Host sendet Pakete, die vorgeben, von einem anderen eventuell vertrauens-

würdigen Host zu kommen. SSH schützt auch vor gefälschten Paketen, die aus dem eigenen LAN stammen und vorgeben, daß sie vom eigenen Router kommen.

- *DNS spoofing*: Ein Angreifer fälscht Antworten eines DNS-Servers.
- *Sniffing*: Mitschneiden und Protokollieren von Verbindungsdaten und Paßwörtern (Abb. 1).
- *Hijacking*: Manipulation von Verbindungen durch Hosts, die als Vermittlungsstelle dienen oder im gleichen Segment liegen, z. B. durch Übernahme einer bestehenden Verbindung.
- *X attacks*: Attacken, die auf Abhören und Fälschen von X-Windows-Authentifizierungsdaten beruhen.

Mit anderen Worten, SSH traut niemals „dem Netz“. Jemand, der in ein Netzwerk eindringt, kann höchstens die SSH-Verbindung unterbrechen, aber nicht die Verbindung übernehmen, die übertragenen Daten mitlesen/mitschneiden und wieder einspielen. Dieser ganze Schutz beruht im wesentlichen auf Verschlüsselung. Daher sollte die vorhandene Option `-nocrypt` im Echtbetrieb niemals benutzt werden!

Wovor SSH nicht schützt

SSH wird niemals den normalen Schutz der Workstation oder des Servers übernehmen. Das heißt, wenn es jemandem gelingt, über bekannte Lücken oder durch Administrationsfehler Administratorrechte zu erlangen, kann er auch SSH kompromitieren. Wenn jemand unberechtigten Zugang zu HOME-Verzeichnissen hat, kann die Sicherheit, die SSH dem Nutzer bietet, wirkungslos sein. Dies betrifft vor allem HOME-Verzeichnisse, die per NFS zur Verfügung gestellt werden. NFS ist höchst unsicher und kann leicht kompromittiert werden. Im HOME-Verzeichnis eines jeden Nutzers werden im Unterverzeichnis `.ssh/` eigene Client-Konfigurationsdateien, die Public Keys der vertrauenswürdigen Server und der eigene Private Key gespeichert.

Es wird eine zusätzliche Authentifizierungsmethode eingeführt. Der bekannte und bekanntermaßen unzulängliche `.rhosts`-Mechanismus wird durch RSA Server-Authentifi-

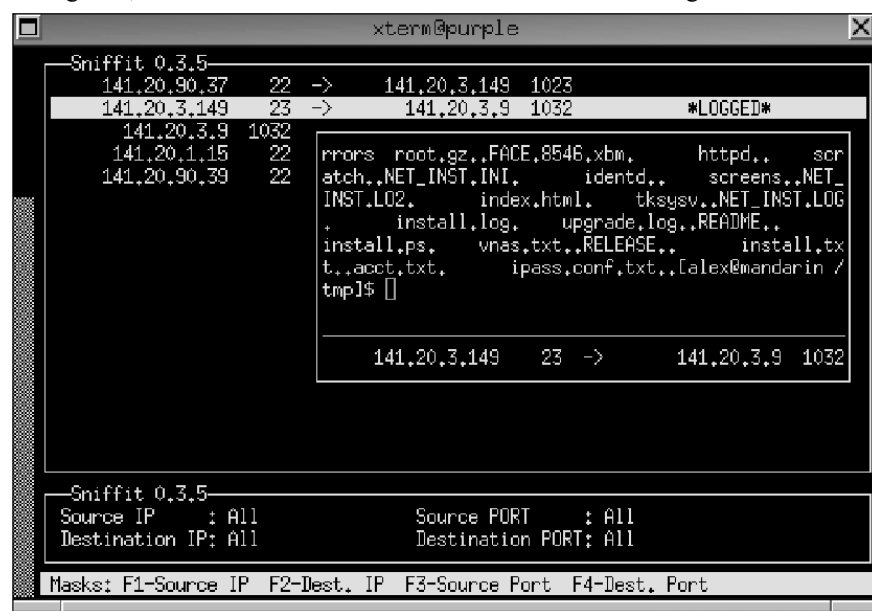


Abb. 1: Netzwerkmitschnitt ohne SSH

zierung ergänzt. Es ist auch eine pure RSA-Authentifizierung möglich.

Die gesamte Kommunikation zwischen Client und Server wird automatisch und – für den Anwender – transparent verschlüsselt. Diese Verschlüsselung schützt außerdem vor gefälschten Paketen und Versuchen, die Verbindung zu kapern. Unter X11 ist darüberhinaus ein verschlüsseltes X-Forwarding möglich. Durch die RSA Host-Authentifizierung wird der Client bei jedem Verbindungsaufbau vor trojanischen Pfer-

platte gespeichert; er wird ständig neu erstellt und befindet sich ausschließlich im Hauptspeicher.

Wie läuft der Login-Prozess ab?

Der SSH-Daemon kontrolliert den Login-Prozess folgendermaßen:

1. Es wird die Lastlogin-Zeit und der Inhalt von `/etc/motd` (wenn nicht abgestellt) angezeigt.
2. Es wird die Loginzeit gespeichert.
3. Wenn die Datei `/etc/nologin` existiert, wird deren Inhalt angezeigt und die Verbindung beendet. Dies gilt nicht für root.
4. Es wird die UID des Nutzers angenommen.
5. Es werden die vorgegebenen globalen System-Umgebungsvariablen gesetzt.
6. Wenn `/etc/environment` und `~/.ssh/environment` existieren, werden auch die darin enthaltenen Umgebungsvariablen gesetzt.
7. Es wird in das Home-Verzeichnis des Nutzers gewechselt.
8. Wenn `~.ssh/rc` oder `/etc/sshrc` existieren, werden Sie ausgeführt.
9. wird `xauth` ausgeführt und `$DISPLAY` gesetzt (wenn nicht abgestellt).
10. Es wird die Nutzer-Shell gestartet.

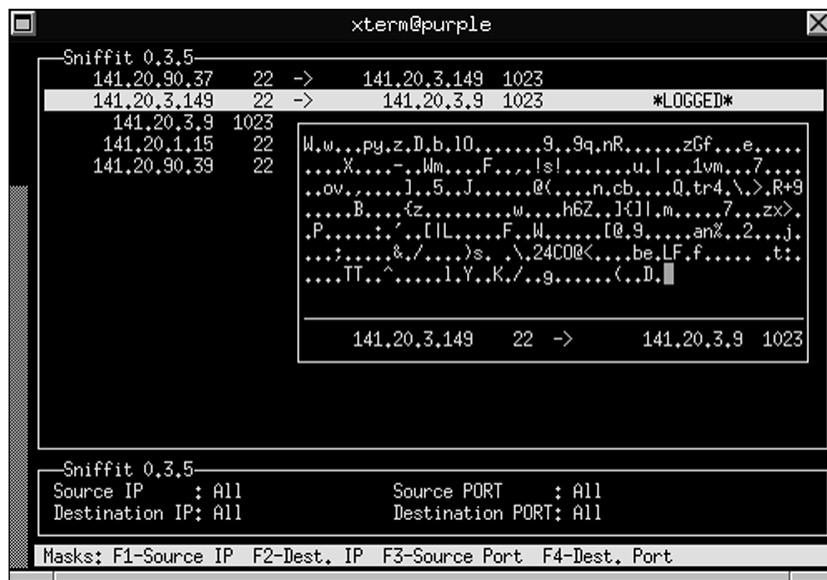


Abb. 2: Netzwerkmitschnitt mit SSH

den, die durch gefälschte Routing- und DNS-Informationen eingeschleust werden können, geschützt. Durch die RSA Client-Authentifizierung wird der Server vor gefälschten Paketen geschützt, noch bevor er auf `.rhosts` oder `/etc/hosts.equiv` basierende Authentifizierung benutzt.

SSH ist als wirksamer, kompletter Ersatz für `rlogin`, `rsh`, `rcp` und `rdist` gedacht. In den meisten Fällen ist SSH auch ein `telnet`-Ersatz!

Funktionsweise

SSH benutzt ein paket-orientiertes, binäres Protokoll. Es wartet auf Port 22, der offiziell für SSH reserviert ist, auf Verbindungen. Das benutzte Protokoll tauscht zufällige Session Keys unter Einsatz des RSA-Algorithmus aus. Der Rest der Verbindung wird – je nach Konfiguration – mit 3DES, DES RC4-128, TSS oder Blowfish verschlüsselt. RSA wird auch für die Authentifizierung benutzt. Der benutzte Session Key wird aber niemals auf der Fest-



Abb. 3: Anmeldung mit slogin

Die wesentlichen Eigenschaften von SSH sind:

starke Authentifizierung

SSH kann reine RSA Host-Authentifizierung oder `.rhosts` zusammen mit RSA Host-Authentifizierung benutzen.

geschützte Privatsphäre

Automatische, für den Nutzer transparente Verschlüsselung der gesamten Verbindung. Host und Client Keys werden mit RSA verschlüsselt. Die Session wird je nach Konfiguration mit 3DES, DES oder IDEA verschlüsselt. Die Verschlüsselung beginnt vor der Paßwortübertragung!

sichere X11 Sessions

`$DISPLAY` wird auf dem Host gesetzt, und X11-Verbindungen werden automatisch über den verschlüsselten Kanal übertragen.

Port Forwarding

Bidirektionale Umlenkung von normalen TCP/IP-Ports auf die verschlüsselte Verbindung.

Automation

Die unsicheren alten `r*`-Programme werden komplett durch die neuen ersetzt. Der Nutzer kann wie gewohnt mit `.rhosts` arbeiten.

Vertraue niemals dem Netz

Ist RSA „eingeschaltet“, wird nur noch dem Privaten Schlüssel getraut.

Host Authentication Key

Der Client kann anhand des gespeicherten Server Keys den Server identifizieren.

User Authentication Key

Der Server kann anhand des gespeicherten Client Keys den Client identifizieren.

Server Key Regeneration

Der Session Key wird regelmäßig neu erstellt und niemals auf der Festplatte gespeichert.

Konfigurierbar

Client wie auch Server sind global und individuell nutzerbezogen konfigurierbar.

rsh Fallback

Wenn kein SSH-Daemon auf dem Zielsystem läuft, kann automatisch das alte (unsichere) `r*`-Protokoll benutzt werden.

Kompression

Daten können bei Bedarf vor der Übertragung automatisch komprimiert werden.

Installation und Konfiguration

SSH ist zum gegenwärtigen Zeitpunkt in der Version 1.2.20 vorhanden. Die Entwickler geben folgende Minimalplattformen für den Einsatz an:

386BSD 0.1; i386

AIX 3.2.5, 4.1, 4.2; RS6000, PowerPC

BSD 4.4; weitere Plattformen

BSD/OS 1.1, 2.0.1; i486

BSD/386 1.1; i386

BSDI 2.1; x86

ConvexOS 10.1; Convex

Digital Unix 4.0, 4.0A, 4.0B; Alpha

DGUX 5.4R2.10; DGUX

FreeBSD 1.x, 2.x; Pentium

HPUX 7.x, 9.x, 10.0; HPPA

IRIX 5.2, 5.3; SGI Indy

IRIX 6.0.1; Mips-R8000

Linux 1.2.x, 2.0.x Slackware 2.x, 3.x, RedHat 2.1, 3.0;

i486, Sparc

Linux 3.0.3, 4.0; Alpha

Linux/Mach3, Macintosh(PowerPC)

Linux/m68k (1.2.x, 2.0.x, 2.1.x)

Mach3; Mips

Mach3/Lites; i386

Machten 2.2VM (m68k); Macintosh

NCR Unix 3.00; NCR S40

NetBSD 1.0A, 1.1, 1.2; Pentium, Sparc, Mac68k,

Alpha

OpenBSD 2.0; x86.

NextSTEP 3.3; 68040

OSF/1 3.0, 3.2, 3.2; Alpha

Sequent Dynix/ptx 3.2.0 V2.1.0; i386

SCO Unix; i386 (Client)

SINIX 5.42; Mips R4000

Solaris 2.3, 2.4, 2.5; Sparc, i386

Sony NEWS-OS 3.3 (BSD 4.3); m68k

SunOS 4.1.1, 4.1.2, 4.1.3, 4.1.4; Sparc, Sun3

SysV 4.x; verschiedene Plattformen

Ultrix 4.1; Mips

Unicos 8.0.3; Cray C90

Windows (3.x/95/NT), MacOS, Amiga, OS/2

Das Programmpaket besteht aus folgenden Komponenten:

`sshd`

Server-Programm, „lauscht“ auf Port 22 auf Verbindungen von Clients, authentifiziert die Verbindungen und startet den Dienst

`ssh`

Client-Programm, dient zum Verbindungsaufbau mit `sshd` als `rlogin`- und `rsh`-Ersatz

`slogin`

symbolischer Link zu `ssh` (`rlogin`-Ersatz)

`scp`

kopiert Dateien zwischen SSH-Systemen (`rcp`-Ersatz)

`ssh-keygen`

erstellt Authentifizierungsschlüssel für Server und Nutzer

`ssh-agent`

Authentifizierungsagent; verwaltet online die RSA-Schlüssel

ssh-add
 registriert neue Schlüssel beim ssh-agent
 make-ssh-known-hosts
 Perl-Script, das im Netzwerk nach öffentlichen
 Schlüsseln sucht und sie in
 /etc/ssh_known_hosts oder
 \$HOME/.ssh/known_hosts speichert

Wenn ein Nutzer eine SSH-Verbindung zu einem SSH-System herstellen möchte, kann er u. a. folgende Kommandos auf der Befehlszeile eingeben:
 % ssh Befehl_der_auf_dem_Zielsystem_ausgefuehrt_werden_soll

oder
 % ssh Zielsystem
 oder
 % xterm -e ssh Zielsystem &

Wenn das Zielsystem kein SSH installiert haben sollte, werden automatisch die adäquaten r* -Programme benutzt. Durch die Angabe der Option -C kann man eine Komprimierung der Verbindungsdaten erreichen. Das Starten der Programme mit der Option -v ermöglicht eine genaue Fehleranalyse durch erweiterte Bildschirmausgabe.

Die folgenden Konfigurationsdateien sind für den Einsatz von SSH von Bedeutung:

<i>Verzeichnis</i>	<i>Dateiname</i>	<i>Verwendung</i>
/etc/	ssh_host_key	privater Schlüssel des Servers; nur für root zugänglich
	ssh_host_key.pub	öffentlicher Schlüssel des Servers
	ssh_random_seed	Verweis für den systemweiten Zufallsnummerngenerator; nur für root zugänglich
	ssh_known_hosts	systemweite Liste mit bekannten öffentlichen Schlüsseln anderer Systeme; ein System pro Zeile
	ssh_config	systemweite Konfigurationsdatei für die SSH-Clients
	sshd_config	Konfigurationsdatei für den SSH-Server
	sshd.pid	Process-ID des letzten sshd-Prozesses
	nologin	wenn diese Datei existiert, darf sich nur root auf dem System anmelden; der Inhalt dieser Datei wird allen anderen Nutzern angezeigt, wenn sie abgewiesen werden
	environment	Umgebungsvariablen, die bei der SSH-Anmeldung gesetzt werden
	hosts.equiv	systemweite Definition von Nutzern und Hosts, die sich via rlogin/rsh anmelden dürfen, wenn RhostAuthentication oder RhostRSAAuthentication in der /etc/sshd_config gesetzt ist
\$HOME/	shosts.equiv	gleiche Funktion wie hosts.equiv, aber nur für SSH
	sshrc	beinhaltet Befehle, die bei der Anmeldung ausgeführt werden, bevor die Shell eines Nutzers gestartet wird
	.rhosts	ermöglicht nutzerbezogene RhostAuthentication, wenn es in der /etc/config_sshd gesetzt ist
\$HOME/.ssh/	.shosts	gleiche Funktion wie .rhosts, aber nur für SSH
	.Xauthority	wird von SSH benutzt, um das Authorization Cookie für den X11-Server zu speichern; SSH überprüft, daß die X11-Forward-Verbindungen dieses Cookie betreffen; wenn die X11-Verbindung aufgebaut wurde, wird dieses Cookie durch das richtige X11-Cookie ersetzt; alle X11-Verbindungen gehen automatisch durch diesen verschlüsselten Kanal, den der SSH-eigene X11-Proxy-Server bereitstellt; SSH setzt \$DISPLAY auf den Server mit einer Displaynummer, die größer als 0 ist
	known_hosts	nutzerbezogene Liste mit bekannten öffentlichen Schlüsseln anderer Systeme; ein System pro Zeile; gilt als Ergänzung der systemweiten /etc/ssh_known_hosts, wenn StrictHostKeyChecking in /etc/sshd_config abgestellt ist.
	identity	privater Schlüssel des Nutzers; wird mit ssh-keygen erstellt und ist durch eine Passphrase geschützt
	identity.pub	öffentlicher Schlüssel des Nutzers; entsteht bei der Erstellung des privaten Schlüssels

<code>authorized_keys</code>	nutzerbezogene Liste öffentlicher Schlüssel (<code>identity.pub</code>) von Nutzern, die ohne Angabe eines Paßwortes Zugang zu diesem Nutzer-Account haben
<code>random_seed</code>	Verweis für den nutzerbezogenen Zufallsgenerator; sollte nur für den Nutzer lesbar / schreibbar sein und von ihm nicht verändert werden
<code>ssh_config</code>	nutzerbezogene Konfigurationsdatei für den SSH-Client
<code>environment</code>	Umgebungsvariablen, die bei der SSH-Anmeldung gesetzt werden; wird nach <code>/etc/environment</code> abgearbeitet
<code>rc</code>	nutzerbezogene Variante von <code>/etc/sshdrc</code>

Angesichts der Gefahren, die herkömmliche `telnet`- oder `rlogin`-Verbindungen beinhalten, sollte sich jeder überlegen, diese überholten Programme durch verschlüsselungsfähige abzulösen. Installation, Konfiguration und Einsatz der Programme wie SSH sind ein-

fach und gewährleisten Erfolg bei der sicheren Übertragung von Daten über unsichere Kanäle.

Alexander Geschonneck
geschonneck@rz.hu-berlin.de
<http://www.hu-berlin.de/~h0271cbj/>