

3

Abhörsicherer Shellzugriff mit OpenSSH

Inhalt

3.1	Einleitung	30
3.2	Der Client	30
3.3	Authentisierungsmethoden und Zugangsbeschränkungen. . .	31
3.3.1	Benutzerbeschränkungen	31
3.3.2	Kennwortloser Zugang mit Public-Key-Verfahren	32
3.3.3	Der SSH-Agent	33
3.3.4	Obsolet: Die rechnerbasierte Authorisierung	34
3.4	Port-Weiterleitung	35
3.4.1	X11-Forwarding	35
3.4.2	Allgemeine Port-Weiterleitung	36
3.5	Sonstige Einstellungen	38
3.5.1	Protokollversionen	38

Lernziele

- Fortgeschrittene Konfiguration von OpenSSH beherrschen
- Die Authentisierungsmechanismen von SSH kennen
- Port-Weiterleitung mit OpenSSH konfigurieren können

Vorkenntnisse

- Grundlagen von SSH und OpenSSH

3.1 Einleitung

Ersatz für TELNET Die *Secure Shell*, SSH, ist ein durch asymmetrische Verschlüsselung abgesicherter direkter Ersatz für TELNET und die *r-tools* (*rlogin*, *rsh*, *rcp* usw.); mit Einschränkungen lässt sich damit auch FTP ersetzen. Der SSH-Server lauscht auf dem TCP-Port 22.

Versionen Das SSH-Protokoll existiert derzeit in zwei Versionen: 1 und 2. Die alte Version 1 sollte *keinesfalls* benutzt werden, da sie herstellerunabhängig gravierende Schwächen aufweist. Insbesondere ist es einem Angreifer möglich, eigenen Text in eine existierende Verbindung einzuschleusen, da die Integrität der übertragenen Daten ungenügend überprüft wird.

OpenSSH-Paket Das frei verfügbare OpenSSH-Paket enthält neben dem Server-Programm *sshd* die Client-Programme *ssh*, *scp* und *sftp* sowie Werkzeuge zur Schlüssel-Erzeugung und -Verwaltung. Dabei verhält sich *ssh* wie *telnet* oder *rsh*, *scp* wie *rcp* und *sftp* wie *ftp*, sie basieren aber alle auf dem SSH-Protokoll (*sftp* hat nichts mit FTPS, also FTP über TLS, zu tun).

In der Kursunterlage *Linux-Netzwerkadministration I – Lokale Netze* wurde OpenSSH bereits vorgestellt, so dass wir uns hier auf eine kurze Wiederholung der Grundlagen beschränken und uns auf exotischere Aspekte konzentrieren.

3.2 Der Client

Programme wie *ssh* und *scp* nehmen als Clients Kontakt mit dem SSH-Daemon (*sshd*) auf einem entfernten Rechner auf. Dabei muss zunächst die Authentizität des Servers sichergestellt werden; in einem zweiten Schritt authentisiert der Server den Client-Benutzer.

Authentisierung beim Verbindungsaufbau Zu seiner Authentisierung beim Verbindungsaufbau überträgt der Server seinen öffentlichen Schlüssel an den Client. Der Client vergleicht diesen Schlüssel mit den in `~/.ssh/known_hosts` und `/etc/ssh/ssh_known_hosts` hinterlegten. Sollte der Schlüssel in keiner dieser Dateien auftauchen, so reagiert der Client mit einer Meldung wie der folgenden:

```
$ ssh 192.0.2.42
```

```
The authenticity of host '192.0.2.42 (192.0.2.42)' can't be established.
```

```
RSA key fingerprint is 6a:41:ea:2d:05:54:89:10:64:c6:88:6c:0b:8c:6a:56.
```

```
Are you sure you want to continue connecting (yes/no)? _
```

Sie sollten nun auf *alternativem* Wege, etwa indem Sie den Administrator des entfernten Rechners per Telefon – nicht Mail! – fragen, den Fingerabdruck des Server-Schlüssels überprüfen, bevor Sie den Schlüssel durchwinken.



Nur für den Fall, dass jemand bei Ihnen anruft und den Fingerabdruck *Ihres* Server-Schlüssels vergleichen möchte: Sie erhalten den mit einem Kommando wie

```
# ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
```

(der Dateiname des öffentlichen Schlüssels kann je nach Ihrer Distribution abweichen).

Schlüssel anders? Stimmt die Version des Server-Schlüssels in Ihrer *known_hosts*-Datei nicht mit der vom Server übermittelten überein, so sehen Sie eine Meldung der Art

```
$ ssh 192.0.2.42
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@   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
6a:41:ea:2d:05:54:89:10:64:c6:88:6c:0b:8c:6a:56.
Please contact your system administrator.
Add correct host key in /home/hugo/.ssh/known_hosts to get rid of this message.
Offending key in /home/hugo/.ssh/known_hosts:9
RSA host key for 192.0.2.42 has changed and you have requested strict checking.
Host key verification failed.
```



Sollten Sie einen anderen SSH-Client benutzen, der bei unbekanntem oder veränderten Server-Schlüsseln nicht Alarm schlägt, dann schmeißen Sie ihn weg – er taugt nichts.

Das Verhalten des Clients können Sie global in der Datei `/etc/ssh/ssh_` Verhalten des Clients `config` festlegen (der Parameter für die Schlüsselprüfung heißt `StrictHostKeyChecking`). Darüberhinaus kann jeder Benutzer das Verhalten konfigurieren, und das sogar separat für jeden Server. Details erfahren Sie in den Handbuchseiten `ssh(1)` und `ssh_config(5)`.

Übungen



3.1 [!2] Testen Sie den `StrictHostKeyChecking`-Parameter. Setzen Sie ihn auf `no` und versuchen Sie, eine Verbindung zu einem entfernten System aufzubauen, dessen öffentlicher Server-Schlüssel nicht in einer lokalen `known_hosts`-Datei steht. Was passiert? Setzen Sie `StrictHostKeyChecking` anschließend auf `yes` und wiederholen Sie den Versuch. Prüfen Sie den Inhalt Ihrer `.ssh/known_hosts`-Datei.

3.3 Authentisierungsmethoden und Zugangsbeschränkungen

3.3.1 Benutzerbeschränkungen

Wenn Sie wollen, können Sie einzelne Benutzer oder Benutzergruppen vom Shellzugang über SSH ausschließen. Dazu dienen in der Server-Konfigurationsdatei `/etc/ssh/sshd_config` die Direktiven `DenyUsers`, `DenyGroups` und `AllowUsers`. Standardmäßig wird jeder Benutzer zugelassen, aber beispielsweise durch

```
DenyGroups      ftpusers pop3users
```

werden Mitglieder dieser beiden Gruppen ausgeschlossen. Umgekehrt können Sie mit `AllowUsers` eine Positivliste angeben.

Um nur vorübergehend, etwa während eines Wartungsfenster, Benutzer vom System auszuschließen, können Sie auch die vom Programm `shutdown` bekannte Datei `/etc/nologin` anlegen: existiert sie, so wird ihr

Benutzer-Ausschluss

Vorübergehender
Ausschluss

Tabelle 3.1: Server- und Benutzer-Schlüssel

Typ	SSH	Server	Benutzer
RSA	2	/etc/ssh/ssh_host_rsa_key	~/.ssh/id_rsa
DSA	2	/etc/ssh/ssh_host_dsa_key	~/.ssh/id_dsa
RSA1	1	/etc/ssh/ssh_host_key	~/.ssh/identity

Inhalt einem Anmeldewilligen angezeigt, der Zugang wird jedoch verweigert. Eine Ausnahme bildet natürlich root; für ihn spielt diese Datei keine Rolle.

Beschränkungen für root Wollen Sie hingegen Beschränkungen für root einrichten, so hilft Ihnen die Direktive PermitRootLogin weiter. Ihr Argument ist entweder yes, no, without-password und forced-commands-only. Dabei bedeutet without-password, dass root sich nicht mit Kennwort anmelden kann (er muss Public-Key-Verfahren benutzen), während forced-commands-only das Ausführen eines Kommandos erzwingt; eine direkte Shell wird nicht erlaubt.

3.3.2 Kennwortloser Zugang mit Public-Key-Verfahren


Bei SSH wird der Server immer dadurch authentisiert, dass festgestellt wird, ob er im Besitz des passenden privaten Schlüssels ist. Aus diesem Grund ist es so wichtig, dass Sie keine »falschen« öffentlichen Schlüssel fremder Server akzeptieren.

Schlüsselpaar erzeugen Wollen Sie, dass sich auch der clientseitige Benutzer gegenüber dem Server über asymmetrische bzw. Public-Key-Kryptographie authentisiert, müssen Sie zunächst ein Schlüsselpaar (öffentlicher/privater Schlüssel) erzeugen. Zur Schlüssel-Erzeugung dient das Programm ssh-keygen, dabei gibt es separate Schlüssel für die Algorithmen RSA und DSA sowie RSA1 (RSA für die Protokollversionen 1 von SSH):


```
$ ssh-keygen -t dsa -b 2048 -f ~/.ssh/id_dsa
Generating public/private dsa key pair.
Enter passphrase (empty for no passphrase): geheime passphrase
Enter same passphrase again: geheime passphrase
Your identification has been saved in /home/hugo/.ssh/id_dsa.
Your public key has been saved in /home/hugo/.ssh/id_dsa.pub.
The key fingerprint is:
b7:a6:39:41:ef:cf:b1:6b:82:bf:37:89:bf:4b:11:cb hugo@example.com
```

erzeugt einen 2048 Bit (Option -b) langen DSA-Schlüssel (Option -t; andere Werte: rsa und rsa1) und legt ihn in der angegebenen Datei (Option -f) ab; der öffentliche Schlüssel wird zusätzlich in id_dsa.pub abgelegt. Die richtigen Pfade für Benutzer- und Serverschlüssel sind in Tabelle 3.1 angegeben. Wenn Sie als normaler Benutzer einen Schlüssel generieren, können Sie jedoch auf die Pfadangabe verzichten; ssh-keygen schlägt Ihnen dann den richtigen vor.

~/.ssh/authorized_keys Hängen Sie den öffentlichen Schlüssel – also die Datei mit der Endung .pub – auf dem Server-Rechner an die Datei ~/.ssh/authorized_keys an. Ab jetzt können Sie sich auf dem betroffenen Rechner anmelden, ohne nach Ihrem Kennwort gefragt zu werden. Allerdings werden Sie nach Ihrer *passphrase* gefragt.

 Einen neuen Server-Schlüssel legen Sie genau so an wie einen für einen normalen Benutzer, allerdings mit den in Tabelle 3.1 angegebenen Pfaden. Bei einem Server kann die *passphrase*, die den privaten Schlüssel auf der Festplatte schützt, auch entfallen. Das ist weniger sicher, aber dafür kann der SSH-Server automatisch gestartet werden.

Server-Schlüssel

 Bei einem Schlüsselpaar für Benutzerauthentisierung (statt Serverauthentisierung) sollten Sie, wenn möglich, nicht auf die *passphrase* verzichten. Es kann sein, dass Sie eine automatische Anmeldung in Shellskripten für cron-Jobs o.ä. realisieren wollen; in diesem Fall bleibt Ihnen kaum etwas anderes übrig, als Schlüsselpaare ohne *passphrase* zu erzeugen. Dann sollten Sie allerdings dafür sorgen, dass der öffentliche Schlüssel auf dem Zielrechner so in die `authorized_keys`-Datei eingetragen wird, dass dort nur ein bestimmtes Kommando ausgeführt wird (egal was der Client gerne hätte). Das nötige Format des `authorized_keys`-Eintrags können Sie `sshd(8)` entnehmen.

Benutzerschlüssel ohne *passphrase*?

3.3.3 Der SSH-Agent

Bei der Authentisierung mit Public-Key-Verfahren sollten Sie unbedingt Ihren privaten Schlüssel vor neugierigen Blicken schützen. Um ihn auch gegen Benutzer mit `root`-Rechten abzusichern, reichen einfache Dateirechte nicht; Sie müssen den Schlüssel mit einer *passphrase* verschlüsseln. Beim intensiven Arbeiten mit SSH, beispielsweise wenn Sie viele Dateien mit `scp` transferieren, kann das dauernd erforderliche Eingeben der *passphrase* jedoch schnell lästig werden.

Abhilfe schafft hier der SSH-Agent `ssh-agent`, der sich für Sie die *passphrase* merkt und bei Anfrage einer Anwendung an Ihrer Statt antwortet. Damit Anwendungen überhaupt wissen, dass ein laufender SSH-Agent existiert, müssen entsprechende Umgebungsvariablen gesetzt sein. Sie können also den SSH-Agent durch

```
$ ssh-agent xterm &
```

o.Ä. aufrufen; das durch ihn gestartete Programm erbt dann die Umgebungsvariablen. Häufig passender ist jedoch eine andere Methode, die sich insbesondere für die Startskripte `~/.profile` und `~/.xsession` eignet. Rufen Sie `ssh-agent` nämlich ohne Programm-Argumente auf, so gibt er ein kleines Shell-Skript aus, das die entsprechenden Variablen setzt:

```
$ ssh-agent -s
SSH_AUTH_SOCKET=/tmp/ssh-nThebJ1030/agent.1030; export SSH_AUTH_SOCKET;
SSH_AGENT_PID=1031; export SSH_AGENT_PID;
echo Agent pid 1031;
```

Deswegen müssen Sie nur am Anfang Ihres Start-Skriptes durch den Aufruf

```
eval `ssh-agent -s`
```

dafür sorgen, dass dieses kleine Shell-Skript ausgeführt wird. (Die Anführungszeichen sind »einfache Anführungszeichen rückwärts«; in der Bash können Sie deswegen auch »`eval $(ssh-agent -s)`« schreiben. Bevorzugen Sie die C-Shell, dann müssen Sie die Option `-s` durch `-c` ersetzen.)

Mit dem Starten des SSH-Agent ist es aber noch nicht getan. Sie müssen ihm vor der Benutzung noch durch das Programm `ssh-add` die *passphrase* mitteilen. Durch `ssh-add` können Sie den SSH-Agent auch vorübergehend sperren (Option `-x`; entsperren mit `-X`) oder ihn die *passphrase* vergessen lassen (Option `-D` für sofortiges Vergessen oder beim Anlegen die Option `-t` für Amnesie nach der angegebenen Anzahl an Sekunden).



Die Verwendung von `ssh-agent` hat Sicherheitsimplikationen, die es zu bedenken gilt. Zum einen kommunizieren SSH-Anwendungen und `ssh-agent` über ein Socket, das von `root` oder einer anderen Instanz des Benutzers abgehört werden kann. – Zwar überträgt der SSH-Agent den privaten Schlüssel nicht über das Socket, sondern führt die notwendigen Berechnungen im Auftrag der Anwendung aus, aber ein Schädlingsprogramm könnte in die Rolle einer legitimen Anwendung schlüpfen. Zum anderen hält der SSH-Agent sensitive Daten im Speicher vor. Unter Linux kann ein Prozess die Auslagerung in den Swap-Bereich der Festplatte nur verhindern, wenn er `root`-Rechte besitzt. Somit besteht bei normaler Verwendung immer die Gefahr, dass die *passphrase* im Klartext auf der Platte landet. – Das Mindeste ist somit, dass Sie nach Gebrauch durch `ssh-add -D` den SSH-Agent alle *passphrases* vergessen lassen.

3.3.4 Obsolet: Die rechnerbasierte Authorisierung

SSH ist angetreten, einen kompletten Ersatz für Telnet und die R-Tools (`rlogin` usw.) bereit zu stellen. Aus diesem historischen Grund sind in OpenSSH noch Authentisierungsmechanismen vorhanden, die weit hinter den Möglichkeiten von SSH zurückbleiben und deshalb nicht mehr benutzt werden sollten. Wir erwähnen sie hier der Vollständigkeit halber und weil sie möglicherweise in der LPI-202-Prüfung auftauchen:

Historische Authentisierung
 Mit `rlogin` usw. konnten Sie sich auf einem anderen Rechner *ohne Kennwort-Abfrage* anmelden, wenn Ihr Rechner auf dem entfernten Rechner in der Datei `/etc/hosts.equiv` vermerkt war. Der entfernte Rechner vertraute also dem lokalen Rechner, dass eine Anmeldung hier wie dort äquivalent sei. Diese Vertrauensstellung konnte auch von normalen Benutzern hergestellt werden: ein Benutzer musste nur in der Datei `~/.rhosts` eintragen, von welchem Rechner aus (und mit welchem dortigen Benutzernamen) eine Anmeldung ohne Kennwort-Abfrage durchgewunken werden sollte.

OpenSSH kann so konfiguriert werden, dass es sich wie `rlogin` usw. verhält, also `/etc/hosts.equiv` und `~/.rhosts` auswertet (mit OpenSSH können auch die Dateien `/etc/ssh/ssh_known_hosts` bzw. `~/.ssh/known_hosts` benutzt werden). Um zumindest gegen (triviale!) IP- und DNS-Spoofing gefeit zu sein, wird der Client-Rechner nicht über die IP-Adresse authentisiert, sondern wie der Server-Rechner über seinen Server-Schlüssel. Dazu muss der öffentliche Schlüssel des lokalen (Client-)Rechners auf dem entfernten (Server-)Rechner hinterlegt sein (in `/etc/ssh/known_hosts` oder in `~/.ssh/known_hosts`); damit der Client an den privaten Schlüssel gelangen kann, muss `ssh` auf dem lokalen Rechner mit Root-Rechten laufen.

Die relevanten Direktiven der Server-Konfiguration sind:

<code>HostbasedAuthentication</code>	<code>yes</code>	<i>Protokoll-Version 2</i>
<code>RhostsRSAAuthentication</code>	<code>yes</code>	<i>Protokoll-Version 1</i>

zum Einschalten. Mit

```
IgnoreRhosts      no
IgnoreUserKnownHosts no
```

können Sie das Durchsuchen von `~/.rhosts` und `~/.shosts` bzw. von `~/.ssh/known_hosts` für die rechnerbasierte Authentisierung unterbinden.



Mit den hier beschriebenen Verfahren werden immer nur Rechner, nie jedoch Benutzer authentisiert (der Server vertraut anschließend dem authentisierten Client, dass dessen Benutzernamen dieselben sind wie auf dem Server). Rechnerbasierte Authentisierung können Sie in praktisch allen Fällen (einschließlich automatisch ablaufender Skripte) durch kennwortlosen Zugang mit Public-Key-Verfahren ersetzen.

Übungen



3.2 [!1] Welchen Vorteil könnte es haben, auf einem entfernten System das direkte Anmelden als `root` über die SSH mit `PermitRootLogin no` zu unterbinden?



3.3 [!3] Diskutieren Sie die Sicherheitsaspekte der drei Authentisierungsmethoden der OpenSSH: Mit Kennwort des entfernten Systems, mit Public-Key-Authentisierung und *passphrase*, und mit Public-Key-Authentisierung und `ssh-agent`.



3.4 [2] Warum verwendet die OpenSSH Dateien wie `.shosts` und `/etc/ssh/shosts.equiv` zusätzlich zu `.rhosts` und `/etc/hosts.equiv`?

3.4 Port-Weiterleitung

3.4.1 X11-Forwarding

Ein Design-Kriterium des X11-Protokolls war Netzwerktauglichkeit – Sicherheit hingegen gehörte nicht dazu. Auf den Einsatz von X11 zwischen zwei Rechnern sollten Sie daher unbedingt verzichten.

Sie können jedoch das unsichere X11-Protokoll mit durch eine SSH-Verbindung tunneln; dadurch ist es aus dem Netz zwischen den beiden Rechnern nicht mehr angreifbar. Dieses X11-Forwarding erreichen Sie durch den Aufruf von `ssh` mit der Option `-X` (großes »x«; mit `-x` wird es deaktiviert, sollte es z. B. durch eine globale Konfigurationsdatei angeschaltet worden sein).

X11-Forwarding

Allerdings muss der SSH-Server X11-Forwarding auch unterstützen. Dies wird in der Server-Konfigurationsdatei `/etc/ssh/sshd_config` durch die Zeile

SSH-Server

```
X11Forwarding  yes
```

erreicht. In einem solchen Fall gaukelt er den entfernten X-Clients vor, er sei ein X-Server, der auf dem Display `localhost:10.0` (oder `localhost:11.0` usw., sollte es besetzt sein) lauscht.



Das Sicherheitsniveau von X11-Forwarding ist geringer als das einer SSH-Sitzung ohne X11-Forwarding. Das liegt daran, dass auf dem

Sicherheitsniveau

entfernten Rechner die Verbindung zwischen X11-Client und SSH-Server mitgehört werden kann bzw. vom entfernten Rechner aus Ihr lokaler X-Server angezapft werden kann (für beides werden root-Rechte benötigt). Das trifft jedoch auch für eine lokale X11-Sitzung zu.

Wenn Sie dem (Administrator des) entfernten Rechner nicht trauen, sollten Sie deshalb X11-Forwarding unterbinden. – Oder besser noch: Sie verzichten ganz auf den Kontakt mit diesem Rechner (wer weiß, was der nicht vertrauenswürdige Administrator mit dem SSH-Daemon oder anderen Programmen angestellt hat?).

Als Administrator haben Sie wenig Chancen, X11-Forwarding generell zu unterbinden, da sich Benutzer immer auch selbst helfen können, vgl. Übung 3.9.

3.4.2 Allgemeine Port-Weiterleitung

Beim X-Forwarding handelt es sich nur um einen modifizierten Spezialfall der allgemeinen Port-Weiterleitung (vgl. Übung 3.9).

Dazu wird eine normale SSH-Verbindung aufgebaut. Zusätzlich gibt eine Option an, ob Sie lokal (Option `-L`) oder entfernt (engl. *remote*, Option `-R`) auf eine Anfrage lauschen wollen und auf welchen Port. – Dabei ist mit Port im Folgenden immer TCP-Port gemeint; nur solche werden hier weitergeleitet. – So wird durch

```
$ ssh -L 2023:<Rest> telnet.example.org
```

von hier zum Rechner `telnet.example.org` eine sichere Verbindung aufgebaut. Außerdem lauscht *lokal* der SSH-Client auf Port 2023. Eine Anfrage an den lokalen Port 2023 wird also durch den SSH-Tunnel nach `telnet.example.org` weitergeleitet.

Aber was passiert dann? Das bestimmt der Parameter `<Rest>`, der aus Rechnerbezeichnung und Portnummer besteht, also beispielsweise `192.0.2.1:23`: Am Ende des Tunnels wird eine *unverschlüsselte* Verbindung vom SSH-Server zum Rechner `192.0.2.1` und dort Port 23 aufgebaut. Zusammen ergibt dies dann

```
$ ssh -L 2023:192.0.2.1:23 telnet.example.org
```

Die Idee dahinter ist, dass der SSH-Tunnel zu einem Gateway-Rechner aufgebaut wird, von dem ab Sie sich im »sicheren Netz« befinden. Aber natürlich kann der Zielrechner auch `localhost` sein, so dass der Aufruf sich zu

```
$ ssh -L 2023:127.0.0.1:23 telnet.example.org
```

verändert. Damit können Sie statt dem unsicheren direkten TELNET mit

```
$ telnet telnet.example.org
```

die sichere Variante

```
$ telnet 127.0.0.1 2023
```

benutzen, die TELNET durch den Tunnel schickt. Der unprivilegierte Port 2023 gestattet es Ihnen dabei, dass Sie für den sicheren TELNET-Tunnel keine lokalen root-Rechte benötigen.



Sie müssen die Ports numerisch angeben; Portnamen aus `/etc/services` sind leider nicht erlaubt.



Angaben wie »ssh -L 23:localhost:23 telnet.example.org« können problematisch sein (unter SUSEs SLES 8 beispielsweise sind sie es definitiv), weil hierbei »localhost« von sshd auf telnet.example.org aufgelöst wird. Ergibt die Namensauflösung als erstes eine IPv6-Adresse (:::1), so wird eine Verbindung nur zu dieser hergestellt, selbst wenn der TELNET-Server nur auf IPv4-Localhost (127.0.0.1) lauscht. Geben Sie in einem solchen Fall immer 127.0.0.1 an und bedenken Sie auch die Konsequenzen der *entfernten* Namensauflösung.



Mit der neuen Version OpenSSH 4.0 lässt sich die Port-Weiterleitung noch besser steuern: es ist jetzt auch das Lauschen auf einzelnen Netzwerkschnittstellen möglich.

Übungen



3.5 [2] Erlauben Sie mit `xhost +` den Zugriff von jedem Rechner auf Ihren X-Server. Erstellen Sie nun mit `xwd` von einem anderen Rechner aus einen Schnappschuss des Bildschirminhalts dieses Rechners. Den Schnappschuss können Sie mit `xwud` betrachten. (Damit diese Übung funktioniert, muss der X-Server natürlich über das Netz erreichbar sein – manche Distributionen wie Debian GNU/Linux oder neuere SUSE-Versionen schalten das mit »-no!isten tcp« beim Serveraufruf ab. Starten Sie notfalls einen X-Server von Hand.)



3.6 [3] Stellen Sie eine SSH-Verbindung zu einem entfernten Rechner her, wobei Sie X11-Forwarding aktiviert haben. Starten Sie auf dem entfernten Rechner eine `xclock`, deren grafische Anzeige auf Ihrem lokalen Rechner zu sehen sein sollte.

Überprüfen Sie den Wert der Umgebungsvariable `DISPLAY` auf dem entfernten Rechner und die lokalen und entfernten Netzverbindungen mittels `netstat`. Setzen Sie beide Daten (Display und Netzverbindungen) in Beziehung zueinander.



3.7 [2] Skizzieren Sie an einem Schaubild mit jeweils drei Rechnern, was die Aufrufe

```
$ ssh -L 10080:192.0.2.3:80 192.0.2.2
```

und

```
$ ssh -R 10080:192.0.2.4:80 192.0.2.2
```

jeweils vom Rechner 192.0.2.1 aus bedeuten.



3.8 [2] Warum lässt sich FTP nur sehr mühselig sicher über SSH mit Port-Weiterleitung tunneln?



3.9 [4] Realisieren Sie X11-Forwarding »von Hand« durch normale Port-Weiterleitung. Denken Sie dabei an die folgenden Punkte:

1. Der lokale X-Server muss IP-Verbindungen annehmen; starten Sie ggf. einen X-Server von Hand.
2. Auf dem entfernten System müssen Sie die DISPLAY-Variable mit einem geeigneten Wert exportieren.
3. Der lokale X-Server muss X11-Verbindungen akzeptieren. Gegebenenfalls müssen Sie mit `xauth list` die lokalen MIT-Cookies erfragen und auf dem entfernten Rechner mit `xauth add` setzen.



3.10 [6] Tunneln Sie eine FTP-Sitzung mit aktivem FTP, wobei Sie den FTP-Client durch `netcat` oder `telnet` ersetzen. (Setzt Protokollkenntnisse von FTP voraus.)

3.5 Sonstige Einstellungen

3.5.1 Protokollversionen

SSH liegt, wie erwähnt, in zwei Protokollversionen vor: 1 und 2. Einen Vergleich der beiden Versionen können Sie unter <http://www.snailbook.com/faq/ssh-1-vs-2.auto.html> finden. Wir gehen hier darauf nicht weiter ein, denn obwohl OpenSSH beide Versionen unterstützt, sollten Sie die Benutzung von SSH 1 keinesfalls zulassen: *Protokollversion 1 ist unsicher*. Sie erlaubt unter bestimmten Umständen das Einfügen von fremden Daten in den verschlüsselten Datenstrom (CVE-1999-1085¹; eine Beschreibung dieses Angriffs findet sich unter <http://www1.corest.com/common/showdoc.php?idxseccion=10&idx=131>).

Client-seitig können Sie die Protokollversion in der Datei `/etc/ssh/ssh_conf` durch

```
Host *
  Protocol 2
```

deaktivieren. Server-seitig erfolgt die Deaktivierung ebenfalls durch den Eintrag `Protocol 2`, allerdings ohne vorangestellte Rechnerbeschränkung (`Host *`). Wenn Sie unbedingt müssen, so können Sie durch `Protocol 1,2` oder `Protocol 2,1` beide Versionen freischalten. Allerdings drückt die Reihenfolge nur auf dem Client eine Präferenz aus, da der Server jede Version akzeptiert, die der Client verlangt.



Da die Client-Konfiguration von den Benutzern *immer* überstimmt werden kann, können Sie die Protokollversion (und andere Einstellungen) nur auf dem Server erzwingen.

Übungen



3.11 [2] Richten Sie einen SSH-Server so ein, dass er nur Protokollversion 2 unterstützt.

Versuchen Sie sich mit diesem Server zu verbinden, wobei Sie durch die Option `-1` von `ssh` Protokollversion 1 erzwingen.

¹*Common Vulnerabilities and Exposures (CVE)* ist eine Klassifizierung von Schwachstellen (<http://cve.mitre.org/>).

Kommandos in diesem Kapitel

scp	Sicheres Dateikopierprogramm auf SSH-Basis.	scp(1)	30
sftp	Sicheres FTP-artiges Programm auf SSH-Basis.	sftp(1)	30
ssh	»Secure Shell«, erlaubt sichere interaktive Sitzungen auf anderen Rechnern.	ssh(1)	30
ssh-add	Akkreditiert private Schlüssel beim ssh-agent.	ssh-add(1)	33
ssh-agent	Verwaltet private Schlüssel und Kennwörter für die SSH.	ssh-agent(1)	33
ssh-keygen	Generiert und verwaltet Schlüssel für die SSH.	ssh-keygen(1)	32
sshd	Server für das SSH-Protokoll (sicherer interaktiver Fernzugriff).	sshd(8)	30
xwd	Kopiert den Inhalt eines X-Fensters in eine Datei.	xwd(1x)	37
xwud	Zeigt den Inhalt einer mit xwd geschriebenen Datei an.	xwud(1x)	37

Zusammenfassung

- Die *Secure Shell* (SSH) ist ein sicherer Ersatz für die TELNET und die *r-tools* und (mit gewissen Einschränkungen) FTP.
- OpenSSH ist eine frei verfügbare SSH-Implementierung.
- Über die `known_hosts`-Dateien können Sie öffentliche Schlüssel entfernter Rechner vorgeben. Die Direktive `StrictHostKeyChecking` regelt, wie ein ssh-Client mit den öffentlichen Schlüsseln entfernter Rechner umgeht.
- SSH-Zugriff kann bei der OpenSSH auf die Mitglieder bestimmter Gruppen eingeschränkt werden. Direktes Anmelden als `root` kann erlaubt, verboten oder von bestimmten Bedingungen abhängig gemacht werden.
- Schlüsselpaare für asymmetrische Kryptographie können Sie mit `ssh-keygen` erzeugen.
- Der `ssh-agent` erlaubt die (halbwegs sichere) Speicherung von *passphrases* zur Arbeitsvereinfachung.
- Die SSH unterstützt aus Kompatibilitätsgründen zu den *r-tools* eine rechnerbasierte Authentisierung mit Äquivalenz von Benutzernamen, aber das sollten Sie, wenn irgend möglich, nicht verwenden.
- Port-Weiterleitung erlaubt den Zugriff auf beliebige TCP-Server über eine verschlüsselte und authentifizierte SSH-Strecke.
- SSH gibt es in den Protokollversionen 1 und 2. Aus Sicherheitsgründen sollte nur Version 2 benutzt werden.

Literaturverzeichnis

- BS01** Daniel J. Barrett, Richard Silverman. *SSH, The Secure Shell: The Definitive Guide*. Sebastopol, CA: O'Reilly & Associates, 2001. ISBN 0-596-00011-1. <http://www.oreilly.com/catalog/sshtdg/>

