Seminar „Sicherheit in Datennetzen“ Sichere Kommunikationsprotokolle am Beispiel von SSH

Matthias Köppe*

27. November 1997

Zusammenfassung

Die TCP/IP-basierten Berkeley-*r*-Protokolle weisen Sicherheitsprobleme auf: Daten, insbesondere Paßwörter, werden unverschlüsselt übertragen, und die Authentizität des jeweiligen Kommunikationspartners wird nur schwach überprüft. Abhilfe schaffen zentrale Authentifizierungssysteme wie Kerberos oder der verteilte Einsatz sicherer Kommunikationsprotokolle. Die letztere Möglichkeit wird hier am Beispiel der Secure Shell (SSH) dargestellt.

1 Sicherheitsprobleme. Lösungsmöglichkeiten

Die Berkeley-*r*-Protokolle bzw. -Kommandos (`rsh`, `rlogin`, `rcp`) weisen Sicherheitsprobleme auf. Einerseits erfolgt keine Verbindungsverschlüsselung, so daß insbesondere Paßwörter auf dem Weg der Übertragung durch das Netzwerk mitgelesen werden können. Andererseits vertrauen die Kommunikationspartner auf die – relativ leicht verletzbare – Integrität von TCP/IP-Netzdiensten wie dem *Domain Name System*: Der Client erwartet, daß mit dem verwendeten Hostnamen bzw. der IP-Adresse der echte Server angesprochen wird; der Server entscheidet anhand der Adresse des Client, ob ohne weitere Nutzer-Authentifizierung Zugriff gewährt wird.

Um diese Sicherheitslücken zu schließen, sind kryptographische Verfahren anzuwenden. Für Netze innerhalb von Unternehmen oder Institutionen empfiehlt sich der Einsatz eines starken zentralen Authentifizierungssystems wie Kerberos; dessen Einrichtung ist jedoch recht aufwendig, da unter anderem der zentrale Authentifizierungsserver physisch stark gesichert werden muß. Für moderate Sicherheitsanforderungen empfiehlt sich deshalb der Einsatz sicherer Kommunikationsprotokolle mit verteilter Authentifizierung.

Eine solche Protokollfamilie für sicheres *remote login* und die sichere Nutzung anderer Netzdienste über ein unsicheres Netz ist die *Secure Shell* (SSH). Sie wurde ursprünglich entwickelt und implementiert von Tatu Ylönen (TU Helsinki); die Weiterentwicklung erfolgt kommerziell durch *SSH Communications Security* und *Data Fellows*. Für Unix-Systeme gibt es freie Versionen.

Die SSH-Protokollfamilie wird derzeit von der *Network Working Group* der *Internet Engineering Task Force* (IETF) als Netzstandard spezifiziert.

*`mkoeppe@csmd.cs.uni-magdeburg.de`

2 SSH-Protokolle

Die SSH-Protokollfamilie ist asymmetrisch konzipiert; in jeder Verbindung wird zwischen dem *Client* und dem *Server* unterschieden. Die folgenden Dienstschichten sind vorhanden:

- Das *transport layer protocol* sichert die Authentizität des Diensterbringers sowie die Vertraulichkeit und Integrität der Datenkommunikation. Außerdem kann es Datenkompression bereitstellen. Das Protokoll setzt auf einem verbindungsorientierten Netzdienst (typischerweise TCP) auf.

- Das *user authentication protocol* sichert die Authentizität des Dienstnutzers. Es setzt auf dem *transport layer protocol* auf.

- Das *connection protocol* setzt auf dem *user authentication protocol* auf und nutzt die gesicherte Verbindung, um Kanäle für verschiedene Netzdienste bereitzustellen.

2.1 *Transport layer protocol*

Das *transport layer protocol* von SSH setzt auf einem zuverlässigen (*reliable*) verbindungsorientierten Netzdienst auf; dieser ist typischerweise TCP. Der Server wartet am TCP-Port 22 auf Verbindungen.

Sobald ein Client eine Verbindung zu dem Server aufgebaut hat, erfolgt eine Verhandlung über die zu verwendende Protokollversion sowie über die zu verwendenden Algorithmen für die Realisierung verschiedener Sicherheitsaspekte:

Key-Exchange-Methoden Die Übertragung von Daten zwischen Client und Server soll verschlüsselt erfolgen, und der Server soll seine Authentizität nachweisen. Dazu werden Public-Key-Verfahren eingesetzt. Jeder Host, auf dem ein SSH-Server läuft, besitzt ein Schlüsselpaar, das der Host-Identifikation dient.

Die derzeit einzige definierte Key-Exchange-Methode ist ein Schlüsselaustausch nach Diffie–Hellman (`diffie-hellman-group1-sha1`). Im wesentlichen stellen Client und Server vermöge zahlentheoretischer Mittel ein *gemeinsames Geheimnis* her; die dabei ausgetauschten Daten bieten für Außenstehende keine Möglichkeit, das Geheimnis zu berechnen. Dieses Geheimnis wird als Schlüssel für die symmetrische Verbindungsverschlüsselung eingesetzt.

Tatsächlich aber geht in die Berechnung des gemeinsamen Geheimnisses server-seitig zusätzlich der *geheime*, client-seitig der *öffentliche* Schlüssel des Server-Hosts ein. Dadurch ist *explizite* oder *implizite* Authentifizierung des Servers möglich.

Der Schlüsselaustausch wird typischerweise nach einer Stunde Verbindungszeit oder 1 GB übertragener Daten wiederholt; die Schlüssel werden aus Sicherheitsgründen niemals in einer Datei abgespeichert.

Verschlüsselungsmethoden Verschiedene symmetrische Verschlüsselungsmethoden wie `3des-cbc`, `blowfish-cbc` und `idea-cbc` sind definiert. Es handelt sich hierbei um *block ciphers* im *cipher block chaining mode*.

Methoden zur Sicherung der Integrität Die Integrität der Daten wird gesichert, indem jedes Paket mit einem *message authentication code* (MAC) ausgestattet wird. Es handelt sich hierbei um eine kryptographisch gesicherte Prüfsumme; verschiedene Verfahren sind definiert.

Fernerhin können Kompressionsmethoden vereinbart werden.

2.2 *User authentication protocol*

Das *user authentication protocol* setzt auf dem *transport layer protocol* auf. Es kann daher vorausgesetzt werden, daß eine verschlüsselte Verbindung mit einem authentifizierten Server-Host besteht. Es gilt nun, einen Nutzer auf dem Server-Host anzumelden (*remote login*) und dabei die Authentizität des Nutzers zu beweisen. Dazu sind verschiedene Methoden definiert; Implementierungen bzw. Konfigurationen müssen nicht alle unterstützen:

User-orientierte RSA-Authentifizierung (`publickey`) Der Server verfügt für jeden Nutzer über eine Liste zulässiger öffentlicher Schlüssel (`~/.ssh/authorized_keys`). Will sich ein Nutzer anmelden, so deutet er an, mit welchem Schlüssel er sich authentifizieren will; daraufhin sendet der Server eine RSA-Challenge, die nur der Nutzer vermöge seines privaten Schlüssels (`~/.ssh/identity`) lösen kann.

Paßwort-Authentifizierung (`password`) Ein Nutzer meldet sich mit seinem gewöhnlichen Unix-Nutzerpaßwort an. Man beachte, daß dies kein Sicherheitsrisiko darstellt, da das *transport layer protocol* den Verbindungskanal verschlüsselt betreibt.

Host-orientierte RSA-Authentifizierung (`hostbased`) Der Server verfügt über eine Liste von vertrauenswürdigen Rechnern und zugehörigen öffentlichen Schlüsseln (`/etc/ssh_known_hosts`). Im Gegensatz zu den Berkeley-*r*-Protokollen wird nicht der Integrität von TCP/IP-Netzdiensten wie DNS vertraut, sondern die Authentizität des Client-Hosts mittels einer RSA-Challenge geprüft. (Der Client-Host hält seinen privaten Schlüssel in `/etc/ssh_host_key`.)

2.3 *Connection protocol*

Das *connection protocol* setzt auf dem *user authentication protocol* auf; somit steht ihm eine verschlüsselte Verbindung zwischen gegenseitig authentifizierten Kommunikationspartnern zur Verfügung. Auf diese Verbindung werden Kommunikationskanäle für verschiedene Dienste gemultiplext. Es sind im wesentlichen die folgenden Dienste definiert:

Remote shell Auf dem Server können Shell-Kommandos ausgeführt werden (`ssh`) oder ein Pseudo-Terminal angelegt werden (`slogin`).

X11 forwarding Jeder X-Client verfügt über eine Verbindung zu seinem Display (X-Server). Diese läuft im nicht-lokalen Fall über TCP/IP, und zwar unverschlüsselt. Die Client-Authentifikation findet typischerweise host-orientiert (`xhost`) statt und vertraut der Integrität der TCP/IP-Dienste. Es steht auch ein user-orientierte Authentifikationsmethode (`xauth`, `MIT-`

MAGIC-COOKIE-1) zur Verfügung, die aber in kryptographisch nicht gesicherten Verbindungen nicht sehr wirkungsvoll ist.

Diese Probleme behebt das *X11 forwarding* von SSH. Auf dem SSH-Server (wo X-Clients ausgeführt werden sollen) wird auf einem freien Port ein virtueller X-Server eingerichtet und die DISPLAY-Variable entsprechend gesetzt. Dieser X-Server leitet nun die X11-Protokoll-Pakete über den gesicherten Kanal zu dem echten X-Server auf dem SSH-Client; entsprechend wird auch die umgekehrte Richtung behandelt.

TCP/IP port forwarding Beliebige TCP/IP-Ports können über die gesicherte Verbindung weitergeleitet werden. Beispielsweise läßt sich so die Kommando-Verbindung eines FTP sichern, so daß Paßwörter nicht im Klartext übertragen werden.

Remote copy Derzeit noch nicht spezifiziert, aber in der Referenzimplementation verfügbar, sind Kopieroperationen zwischen Client und Server (scp).

3 Verbleibende und neue Probleme

Durch *TCP/IP port forwarding* kann die Funktionalität von Firewalls beeinträchtigt werden, da die durch die verschlüsselte Verbindung geleiteten Pakete von der Firewall nicht untersucht werden können.

Können unberechtigte Personen Zugriff auf das Homeverzeichnis (mit den privaten Schlüsseln) erlangen, so kann SSH keinerlei Sicherheit mehr bieten. Dies ist insbesondere der Fall, wenn das Homeverzeichnis mit ungesichertem NFS exportiert wird.

Literatur

[1] Koenig, Thomas: *Ssh (Secure Shell) FAQ – Frequently asked questions.* http://www.uni-karlsruhe.de/~ig25/ssh-faq, Juni 1997.

[2] Lai, Xuejia: *On the Design and Security of Block Ciphers*, Band 1 der Reihe *ETH series in Information Processing*. Hartung-Gorre Verlag, Konstanz, Erste Auflage, 1992.

[3] Ylönen, T., T. Kivinen und M. Saarinen: *SSH Authentication Protocol.* Internet Draft draft-ietf-secsh-userauth-03.txt, November 1997. Work in progress.

[4] Ylönen, T., T. Kivinen und M. Saarinen: *SSH Connection Protocol.* Internet Draft draft-ietf-secsh-connect-03.txt, November 1997. Work in progress.

[5] Ylönen, T., T. Kivinen und M. Saarinen: *SSH Protocol Architecture.* Internet Draft draft-ietf-secsh-architecture-01.txt, November 1997. Work in progress.

[6] Ylönen, T., T. Kivinen und M. Saarinen: *SSH Transport Layer Protocol.* Internet Draft draft-ietf-secsh-transport-03.txt, November 1997. Work in progress.