Vorwort

Als Tatu Ylönen vor mehr als zehn Jahren die erste Version von SSH veröffentlichte, rechnete niemand mit einem solchen Erfolg: Auch wenn SSH vor allem hinter den bunten Kulissen präsent ist – SSH ist aus dem heutigen Internet nicht mehr wegzudenken, es bildet einen wesentlichen Teil der Infrastruktur.

Woher kommt dieser Erfolg? Wahrscheinlich hat es zwei Gründe. SSH ist einfach – und kompliziert.

Zunächst einmal ermöglicht es auf sehr einfache Weise kryptographisch gesicherte Logins auf entfernte Rechner. Der Benutzer muß nichts besonderes tun, er tippt lediglich `ssh` statt `telnet`. Moderne Kryptographie wird benutzbar gemacht, ohne daß man sich mit Zertifikaten, PKI-Hierarchien oder anderen komplexen Konfigurationhindernissen auseinandersetzen muss: SSH funktioniert – einfach.

SSH ist aber auch kompliziert. Der abgesicherte Kanal kann nicht nur von SSH selbst benutzt werden, sondern anderen Diensten zur Verfügung gestellt werden, Diensten, die von sich aus keine Verschlüsselung unterstützen. SSH bietet dabei eine Vielzahl von Tunnelmöglichkeiten, neben X11 können beliebige TCP-Verbindungen über SSH transportiert werden.

Beides zusammen, die Einfachheit und die zahllosen Verwendungsmöglichkeiten, sind Grund dafür, daß SSH sich auf so vielen Rechnern findet. Auch auf OpenBSD-Systemen ist SSH fast immer als erstes Zusatz-Paket nachinstalliert worden – nachinstalliert, da die restriktive Lizenz der SSH Software keine uneingeschränkte Weitergabe erlaubt hat.

Eine der Prinzipien von OpenBSD ist es aber, daß Software so wenig Nutzungsbeschränkungen hat wie irgend nur möglich. Das gilt besonders für Software, wie SSH, die für die Sicherheit von zentraler Bedeutung ist. Nur so können sich die Ideen über Open-Source-Systeme hinaus verbreiten und Einzug in die kommerzielle Softwarewelt finden.

Mit OpenSSH versuchen wir genau das zu erreichen: jedes Netzwerkgerät soll das SSH-Protokoll unterstützen. OpenSSH basiert ursprünglich auf der letzten freien SSH-Implementierung von Tatu Ylönen und wird kontinuierlich moder-

nisiert und erweitert. OpenSSH ist inzwischen nicht nur zusammen mit vielen freien Unix-System erhältlich, sondern auch in die unterschiedlichsten Produkte integriert: in kommerzielle Unix- und Nicht-Unix-Systeme, aber auch in Router oder Switches.

Doch freie Software gibt es nicht *umsonst*. Mit der Zahl der Benutzer wächst die Verantwortung der Entwickler. Um die Qualität von OpenSSH sicherstellen zu können, ist das OpenSSH-Projekt weiterhin auf die freiwillige, auch finanzielle, Unterstützung angewiesen. Informationen hierzu finden sich unter `www.openssh.com`.

Daß die Zahl der Benutzer und Unterstützer von OpenSSH weiter zunimmt, dazu leistet dieses Buch einen wertvollen Beitrag: Es zeigt viele praxisnahe Einsatzmöglichkeiten von SSH, erklärt Hintergründe und Zusammenhänge.

Markus Friedl München, im November 2006

Teil I

Grundlagen

Kapitel 1

Ziel dieses Buches

Dieses Buch setzt sich mit dem Einsatz von SSH – der Secure Shell – in Netzwerken beliebiger Größe auseinander. Die betrachtete Plattform bildet dabei das frei verfügbare Betriebssystem Linux.

Wenn Sie dieses Buch in Händen halten, haben Sie die Notwendigkeit für geschützten Datentransfer bereits erkannt. Es soll Ihnen ergänzend Lücken in Ihren Sicherheitskonzepten entdecken helfen und Sie in die Lage versetzen, die hier vorgestellten Vorschläge zu realisieren. Auf Basis des vermittelten Wissens sollen Sie ferner die Vorschläge an Ihre eigenen Bedürfnisse anpassen oder eigene Lösungen entwickeln und anwenden können. Nach Studium dieses Buches sollen Sie unterschiedliche Lösungsalternativen gegeneinander abwägen, Vorteile und Nachteile bewerten sowie die jeweiligen Konsequenzen abschätzen können.

In Zeiten des Auftretens von täglich neuen Viren, Würmern und Trojanischen Pferden läßt sich ein steigendes Bedürfnis an Sicherheit beobachten. In dem Maß, in dem die geschäftliche oder auch private Nutzung elektronischer Kommunikationsmedien zunimmt, steigt auch der Bedarf an Schutz dieser Kommunikation und verläßlicher Sicherheit. Gleichzeitig steigt die Akzeptanz von Firewalls und sogar Personal Firewalls auf Rechnern ohne Einbindungen in ein lokales Netzwerk. Trotzdem tauschen Nutzer aber weiterhin vertrauliche Informationen unverschlüsselt über Instant Messenger aus und kleben immer noch Notizzettel mit ihren Paßwörtern an den Monitor. Während wir wenig Möglichkeiten haben, an diesen fahrlässigen aber bequemen Verhaltensweisen etwas zu ändern, können wir in einem anderen Bereich Schlimmeres verhindern beziehungsweise Unterstützung für zusätzliche Sicherheit leisten.

Dem verantwortlichen Systemadministrator kommt deswegen die Aufgabe zu, so weit es durchführbar ist, verläßliche Rahmenbedingungen zu schaffen, so daß sich auch der unbedarfte Nutzer auf sichere Kommunikation im Netzwerk abstützen kann. Mit den verschiedenen SSH-Produkten stehen Ihnen dafür Werk-

zeuge zur Verfügung, mit denen sich ein großer Teil des Datentransfers sicher gestalten läßt. Eine Vielzahl von Kommunikationswegen läßt sich durchaus flexibel absichern.

In einem Umfeld, in dem Sicherheit nichts – oder zumindest fast nichts – kosten darf, entfalten besonders die Open Source-Implementierungen der Secure Shell ihren Charme. Mit ihnen sind vergleichsweise einfache Maßnahmen im Hinblick auf Anschaffung (kostenlos) und Umsetzung (Arbeitszeit) günstig zu bekommen. Zusätzlich geben sie Zugang zu einem großen Leistungsspektrum. Die Summe dieser positiven Eigenschaften sollte ein gutes Argument für eine Beschäftigung mit dem Thema Secure Shell und einer ihrer Implementierungen sein.

OpenSSH ist der zentrale Bezugspunkt dieses Buchs und die Beispiele beziehen sich darauf.

Ohne allzuweit vorzugreifen läßt sich sagen, daß SSH ein Client/Server-Protokoll ist. Client-Anwendungen stehen nicht nur für Unix zur Verfügung, sondern auch für eine Vielzahl anderer Plattformen. Trotzdem beschränken wir uns in diesem Buch auf die Darstellung und Verwendung der Unix-Clients. Anhand dessen läßt sich die Funktionalität am besten erklären, da Unix sozusagen die „natürliche Umgebung“ von SSH ist.

Wir haben uns entschlossen, auf die Darstellung anderer, insbesondere grafischer, Clients zu verzichen. Weil diese in ihrem Aussehen und ihrer Handhabung doch recht unterschiedlich sind, würde eine Betrachtung einiges an Raum einnehmen. Diesen nutzen wir lieber, um Ihnen die Grundlagen in Form von Erläuterungen zur Protokollspezifikation nahezubringen.

Auf diese Weise sind die Inhalte des Buches auch weniger abhängig von Software-Produkten und veralten nicht so schnell. Programme sind Release-Zyklen unterworfen und können sich von Version zu Version grundlegend ändern. Von diesen Änderungen ist das Buch – abgesehen von OpenSSH – unabhängig.

Erkenntnisse lassen sich aber auch bis zu einem gewissen Grad auf die verschiedenen Clients übertragen. Die konkrete Ausprägung hängt natürlich von der Plattform und dem Stand der Implementierung des jeweiligen Produkts ab.

Der Aufbau dieses Buches

Kapitel 2 sensibilisiert Sie für die Thematik und führt Ihnen die Notwendigkeit von Maßnahmen zum Schutz des Datentransfers vor Augen. Weil Sie nicht der einzige sind, der sich für Ihre Daten interessiert, gehen wir auf einige allgemeine Anforderungen ein, die es zu erfüllen gilt.

Einiges an Hintergrundwissen zu vermitteln, ist die Aufgabe von Kapitel 3. Das beinhaltet einen Abriß über Kryptographie und die Vorstellung von einigen typi-

schen Angriffsmethoden im Netz. Sie erfahren außerdem, wie SSH in den Kontext der Internet-Protokollfamilie TCP/IP einzuordnen ist.

In den Kapiteln 4 bis 6 erhalten Sie einen Überblick über das SSH-Protokoll. Mit der Vorstellung der unterschiedlichen Protokollversionen – SSHv1 in Kapitel 5 und SSHv2 in Kapitel 6 – der einzelnen Implementierungen und ihrer Einsatzgebiete legen Sie den Grundstein für das weitere Verständnis.

In Kapitel 7 wird es konkreter: Das Kapitel beschäftigt sich mit Konfiguration und Setup sowie der Client- und Server-Konfiguration von OpenSSH. Dabei lernen Sie die jeweiligen Grundgedanken kennen, erhalten erste Praxistipps und werden vor typischen Fehlern gewarnt. Dieser Schwerpunktteil schildert alle Konfigurationsmöglichkeiten im Detail.

Einen weiteren wichtigen Bereich des Buches bildet „SSH im Einsatz“, abgedeckt von Kapitel 8. Es startet mit einem Überblick über die verschiedenen Unterprogramme von OpenSSH und ihre Anwendungsgebiete.

Zusammen mit dem bis dahin angesammelten Wissen sind Sie dann bereit, sich den Herausforderungen des Alltags zu stellen. Weitere Abschnitte geben einige Vorschläge zur Umsetzung in Ihrem Netzwerk. Neben grundlegenden Anwendungen, die nirgends fehlen sollten, werden auch weiterführende Einsatzmöglichkeiten angeregt.

Das Kapitel 9 ist für Einsteiger und Umsteiger interessant. Wer seine bisherige Netzinfrastruktur für den Einsatz von SSH fit machen, von der alten Version SSHv1 auf SSHv2 oder von einem anderen SSH-Produkt auf OpenSSH umsteigen möchte, findet hier die nötigen Informationen. Bei der Migration bestehender Dienste nehmen wir Sie an die Hand und ergänzen die Beschreibung durch einige Fallbeispiele. NX/FreeNX und CVS über SSH sind dafür nur zwei Vorschläge.

Mit dem Aufsetzen von Virtuellen Privaten Netzen (VPN) setzt sich Kapitel 10 auseinander. Die zugehörige Funktionalität ist noch recht jung in OpenSSH und bietet interessante Möglichkeiten.

Den Abschluß des Buches bildet das Kochbuchkapitel 11. Es enthält einfache Vorschläge für das schrittweise Umsetzen typischer Aufgabenstellungen mit OpenSSH.

Sowohl die Ausführungen zur VPN-Thematik als auch die Kochrezepte wurden von Alexander von Gernler als Gastkapitel verfaßt. Aufgrund seiner mehrjährigen Mitarbeit im OpenBSD-Projekt, unter anderem als Committer und Betreiber des OpenBSD Mirrors an der Universität Nürnberg-Erlangen, ist er prädestiniert für diese Themen.

Technische Ausrüstung

Basis für die Ausführungen dieses Buches bildet das Betriebssystem SUSE LINUX 10.x. Alle Erkenntnisse sollten sich aber auch auf andere Unix-Derivate übertragen lassen. Zum Einsatz kommt OpenSSH 4.5.

Kolophon

Zur Zusammenarbeit haben wir die Versionsverwaltung CVS genutzt. In ihr wurden sowohl die eigentlichen Texte und Bilder als auch Protokollspezifikationen und andere Elemente von allgemeinem Interesse abgelegt. Letztendlich handelte es sich um alle Elemente, auf die alle beteiligten Personen Zugriff haben sollten.

Basierend auf Erfahrungen aus der Software-Entwicklung haben wir eine *Continuous-Integration*-Umgebung aufgesetzt. Alle Änderungen sollten regelmäßig gesetzt und idealerweise alle Projektbeteiligten über den Fortschritt informiert werden. Das leistet das Open Source Werkzeug *CruiseControl* (`http://cruisecontrol.sourceforge.net/index.html`). Obwohl es eher auf Software-Entwicklungsprojekte abzielt, hat es uns gute Dienste geleistet. Insbesondere der automatische Versand von E-Mails über Neuerungen hat alle Beteiligten auf dem laufenden gehalten.

Die Grafiken entstanden mit Hilfe von OmniGraffle auf einem G4 Apple PowerBook 1,5 GHz unter Mac OS X. Den Text haben wir unter Verwendung von LaTeX 2ε verfaßt, sowohl mit *Kile* als auch *Texshop*.

Konventionen

Wenn ein komplexes Thema mit vielen Facetten und zahlreichen Details behandelt wird, ist es unter Umständen schwer, den Überblick zu behalten. Einheitliche Strukturen und Konventionen haben dann einen Wiedererkennungseffekt zur Folge und unterstützen damit das Verständnis.

Aus diesen Gründen verwenden wir in diesem Buch einige Konventionen, die in den nächsten Abschnitten kurz vorgestellt werden.

Diagramme

Insbesondere in den Kapiteln des Grundlagenteils tauchen zahlreiche Diagramme auf, die Abläufe veranschaulichen sollen. Diese Bilder stellen Sequenzdiagramme der *Unified Modeling Language 2 (UML)* dar. Wir glauben, daß die Orientierung an einem Standard sinnvoll ist und außerdem zu einer einheitlichen

Gestaltungsweise der Graphiken führt. Auch ohne Kenntnis der UML sollten die Diagramme verständlich sein.

Begriffe

Beschäftigt man sich mit der Secure Shell, so stößt man schnell auf ähnliche Begriffe und Namen, die aber durchaus Unterschiedliches bedeuten können. Um hier Mißverständnisse zu vermeiden und auch im weiteren das Verständnis zu erleichtern, grenzen wir hier die einzelnen Begriffe voneinander ab.

1. SSH oder Secure Shell bezeichnet eigentlich ein Protokoll, kein konkretes Produkt. Es definiert, wie sichere Netzwerkkommunikation stattzufinden hat.
2. Das SSH-Protokoll wurde und wird weiterentwickelt. Deswegen haben sich bis jetzt zwei größere Versionen etabliert. Diese bezeichnen wir in Anlehnung an die Schreibweisen von IP (IPv4, IPv6) als SSHv1 und SSHv2. Beide sind inkompatibel zueinander, und bei SSHv2 handelt es sich um die neuere Version (Kapitel 5 und 6).
3. Von SSHv1 wiederum gibt es mehrere Versionen. SSHv1 umfaßt die Protokollversionen 1.3 und 1.5. SSHv2 entspricht zur Zeit der Version 2.0. Die Versionsnummer 1.99 stellt eine Konvention dar und bedeutet, daß sowohl SSHv1 als auch SSHv2 unterstützt wird.
4. Oft sprechen wir von SSH oder der Secure Shell. Dann sind immer beliebige Produkte gemeint, die das SSH-Protokoll implementieren.[1] In solchen Fällen geht es allgemein um den Einsatz eines SSH-Produkts, ohne daß dabei das Produkt selber von Bedeutung ist.
5. SSH1 und SSH2 sind kommerzielle Produkte. Sie sind Implementierungen des SSHv1- beziehungsweise SSHv2-Protokolls.

Typographische Konventionen

Tabelle 1.1 zeigt einige Beispiele für die verwendeten typographischen Konventionen.

Beispiele

Beispiele finden Sie in grau hinterlegten Bereichen. Diese Beispiele zeigen Kommandos beziehungsweise Nutzereingaben und die resultierenden Ausgaben. Sie enthalten aber auch etwa den Inhalt von Konfigurationsdateien.

[1] OpenSSH ist eines davon und bildet die Basis für die Ausführungen dieses Buchs

Tabelle 1.1: Typographische Konventionen

Beispiel	Bedeutung
Kursiv	Die *kursive* Schreibweise wird für Produktnamen, Protokolle und Fachbegriffe bei ihrer ersten Verwendung genutzt.
`typewriter`	Befehle, Programme, Optionen, Dateien, Verzeichnisse und Internetadressen erscheinen in dieser `Schweibweise`.
F1	Sind bestimmte Tasten auf der Tastatur gemeint, so werden sie über ein separates Symbol dargestellt.
fett	Eingaben von Kommandos innerhalb von Beispielen erscheinen **fett**.

```
user@linux: > ssh -D 1234 sshserver
Enter passphrase for key '/home/user/.ssh/id_rsa':
user@linux: >
```

Website

Begleitend zu diesem Buch existiert eine Website, die Sie unter der URL `http://www.sshbuch.de/` erreichen. Hier finden Sie einige Dinge rund um die Secure Shell, OpenSSH und dieses Buch. Unter `tdotzauer@sshbuch.de` und `tluetticke@sshbuch.de` können Sie direkt Kontakt mit uns aufnehmen.

Danksagung

Wir möchten einer ganzen Reihe von Personen danken, die zum Gelingen dieses Buches beigetragen haben. *Nicolaus Millin* hat mit seinem Verlag dem OpenSSH-Buch eine Heimat gegeben. Unseren Frauen *Irina* und *Christiane* sei Dank für ihre unendliche Geduld, Unterstützung und Verständnis. *Alexander von Gernler* und *Markus Friedl* haben im Fachlektorat sehr gründlich gearbeitet und uns mit vielen kritischen Hinweisen weitergeholfen. Zusätzlich haben sie uns einige Einblicke hinter die Kulissen von OpenSSH ermöglicht. Ihre Mitarbeit im OpenBSD-Projekt und insbesondere ihr Wissen zu OpenSSH – Alexander als ambitionierter Anwender und Markus als einer der Hauptentwickler – hat zu einigen neuen Erkenntnissen verholfen. *Tobias Dussa* hat ebenfalls etliche gute Hinweise, insbesondere zur Grafikgestaltung, beigesteuert. *Bodo Farnsteiner* und *Jörg Ruckelshauß* haben die Website zum Buch gestaltet, wie wir es nie geschafft hätten.

Timo Dotzauer, Tobias Lütticke Stuttgart & Karlsruhe, Dezember 2006

Kapitel 2

Einführung

Dieses einführende Kapitel will für sicherheitstechnische Aspekte sensibilisieren. Dazu streift es einige Sachverhalte und reißt einige Themen aus dem großen Bereich Sicherheit und Datenübertragung an.

Einige einleitende Gedanken dazu, was Sicherheit genau bedeutet und welche Anforderungen sich daraus ableiten, bilden die Grundlagen für die folgenden Teile. In diesen stellen wir die Secure Shell-Protokoll und seine Implementierung OpenSSH als eine Möglichkeit vor, diesen Ansprüchen gerecht zu werden. Dazu gehen wir in allgemeiner Form auf die Secure Shell ein und erläutern ihre allgemeinen Leistungsmerkmale. Ergänzend geben wir einen Überblick über verschiedene verfügbare SSH-Implementierungen (Clients und Server) auf unterschiedlichen Plattformen (Abschnitt 2.2.4). Von Interesse sind außerdem einige lizenzrechtliche Aspekte, die es trotz der Verfügbarkeit freier und damit kostenloser Produkte zu berücksichtigen gilt.

Einer kurzen Einführung in die Entwicklungsgeschichte von SSH folgt ein Überblick über verwandte Technologien aus dem Kontext von Sicherheit und Verschlüsselung. Ihre Abgrenzung zu den Einsatzgebieten von SSH runden das Gesamtbild ab.

2.1 Meine Daten sind sicher – oder?

Achten Sie darauf, daß niemand Ihre Tagebücher liest? Legen Sie Wert darauf, daß Ihr Briefgeheimnis gewahrt bleibt? Vermeiden Sie es, Ihre Gehaltsabrechnung offen in der Firma liegen zu lassen? Bestimmt tun Sie das. Aber was ist mit ihren E-Mails? Wenn Sie persönliche Daten verschicken, sind diese dann verschlüsselt? Bei den meisten Menschen sind sie es nicht. Aber warum? Sind die Informationen unwichtig? Wohl kaum. Und es ist Ihnen auch sicher nicht gleichgültig, wenn ihr Chef erfährt, daß Sie sich soeben bei einer anderen Firma be-

worben haben. Warum schließen Sie dann Ihr Auto ab, lassen aber ihren Arbeitsplatzrechner ungesperrt, wenn Sie weggehen?

Es ist ein menschliches Empfinden, persönliche, private Daten für sich zu behalten und darüber zu wachen, daß dieses so bleibt. Das Recht auf informationelle Selbstbestimmung liefert dazu die rechtliche Grundlage[1]. Das Recht auf den Schutz der Privatsphäre ist ein Grundrecht und jeder möchte selbst entscheiden, was er wann und wem mitteilt.

Warum werden viele Menschen dann aber so leichtsinnig, wenn es um Computer- und insbesondere um Internet-Nutzung geht? Sie installieren Anhänge von E--Mails unbekannter Absender, nur weil in Ihnen vor einem neuen Virus gewarnt wird. Sie speichern sensible private Daten auf Firmenrechnern, ohne zu wissen, welcher Personenkreis Zugang dazu hat.

Kernaufgabe des Internets und aller anderen Netzwerke ist der Datentransfer. Computer werden vernetzt, um Informationen austauschen zu können. Dabei können diese Computer im selben Raum, im selben Gebäude, in derselben Stadt stehen oder auf dem ganzen Globus verstreut sein. Gleichgültig, ob die übertragenen Daten spielerischer Natur sind und der Unterhaltung dienen oder ob sie professionelle Verwendung finden, ihre Nutzung steht und fällt mit ihrer Qualität. Unvollständige Informationen können unbrauchbar sein oder aber eine völlig andere Bedeutung erlangen. Noch schlimmer: die Folgen von Manipulationen sind schier endlos. Dazu muß man noch bedenken: Menschen sind neugierig! Sie studieren die Einkaufswagen der anderen Kunden im Supermarkt, sie lesen Postkarten Fremder, sie hören Gesprächen im Bus zu und sie beobachten Menschen im Café. Auch ohne daß man Mitmenschen kriminelle Energie unterstellt, gilt es deswegen, Informationen zu schützen. Das gilt speziell für die Datenübertragung in lokalen und Weitverkehrsnetzen (LAN/WAN). Diese sollte deswegen den sogenannten kryptographischen Grundaufgaben genügen [Sch95]:

- Authentifizierung (*authentication*)
- Integrität (*integrity*)
- Vertraulichkeit (*privacy*)
- Nichtabstreitbarkeit (*non-repudiability*)

Nicht zu den Grundaufgaben gehörend, aber in diesem Themenkomplex auch von Interesse ist die Autorisierung, die sich mit der Frage „Wer darf was?“ beschäftigt.

2.1.1 Authentifizierung

Mit Hilfe der *Authentifizierung* lassen sich bestimmte Informationen beweisen. Wenn Sie Geld am Automaten abheben, findet eine Authentifizierung über Ihre

[1] Urteil des BVerfG vom 15.12.1983; Az.: 1 BvR 209/83; NJW 84, 419 [Bun]

PIN statt. Damit weisen Sie sich als Berechtigten aus. Auch beim Bezahlen mit Kreditkarte findet eine Authentifizierung statt, sobald der Kassierer die Unterschrift auf der Karte mit der auf dem Beleg vergleicht. Insgesamt unterscheidet man drei Möglichkeiten der Authentifizierung [Sch95]:

- Eigenschaft: man ist etwas (Biometrische Merkmale)
- Besitz: man hat etwas (Schlüssel, Codekarte)
- Wissen: man weiß etwas (Paßwort)

Gelegentlich wird als Teil der Authentifizierung die Identifikation separat betrachtet. Der Hintergedanke dabei ist, daß bei der Identifikation die Aufgabe eines Prüfers ist, die verfügbaren Informationen mit allen anderen bekannten Daten zu vergleichen, um zu einer Übereinstimmung zu gelangen. Der Polizist vergleicht einen konkreten Fingerabdruck mit allen gesammelten. Bei einer Übereinstimmung ist die Identität festgestellt. Authentifizierung setzt voraus, daß die Identifikation bereits stattgefunden hat. Die Identifikation muß deshalb eindeutig sein, Authentifizierung jedoch nicht.

Die Authentifizierung als übergeordnetem Konzept beinhaltet Identifikation. Authentifizierung ist der Nachweis der Indentität in zwei möglichen Ausprägungen:

- Identität als Person (Max Mustermann)
- Identität als Angehöriger eines Personenkreises (Polizei)

Authentifizierung kann außerdem noch einen Herkunftsnachweis beinhalten. Nachrichtenauthentifizierung über Digitale Signaturen erlaubt den Beweis, daß eine bestimmte Nachricht von Person X stammt. Während Geheimhaltung den Kreis möglicher Empfänger einschränkt – zumindest derer, die mit der Nachricht etwas anfangen können – schränkt eine solche Authentifizierung die Menge der Absender ein. Nicht mehr irgend jemand, sondern nur der angegebene Absender ist auch wirklich der Absender. Dank Quellenauthentizität kann sich niemand mehr Wissen dadurch erschleichen, daß er vorgibt, jemand anders zu sein[2]. Vermeintlich seriöse Anfragen der Art „Hallo, hier ist Marco, ich bin im Urlaub ausgeraubt worden und habe nichts mehr. Bitte überweise mir 2000 Euro auf folgendes Konto, damit ich nach Hause kommen kann.“ gehen damit ins Leere.

2.1.2 Vertraulichkeit

Vertraulichkeit oder *Geheimhaltung* bedeutet, daß kein Dritter Kenntnis vom Inhalt der Nachricht erlangt. Sie schützt die Privatsphäre des einzelnen und das

[2]Quellenauthentizität ist nur im Zusammenhang mit geeigneten Techniken wie der Digitalen Signatur gewährleistet. Authentifizierung alleine leistet dieses noch nicht.

Kapitel 3

Allgemeine Grundlagen

Nach einer allgemeinen Einführung in Kapitel 2 und einer Einordnung in den technologischen Gesamtzusammenhang schlagen die folgenden Abschnitte die Brücke hin zu den SSH-spezifischen Themen in Kapitel 7.

Der erste Abschnitt beschäftigt sich mit den kryptographischen Grundlagen, die für ein Verständnis der Funktionsweise von SSH und einiger damit verbundener Abläufe erforderlich sind. Hierbei werden einzelne Konzepte und Verfahren vorgestellt und kurz auf ihre Vor– und Nachteile eingegangen. Dazu zählen unter anderem Public Key-Kryptographie und Hash-Funktionen. Einige Hintergrundinformationen runden das Gesamtbild ab.

Die beiden folgenden Abschnitte stellen in gewisser Weise einen Vorgriff auf das folgende Kapitel dar, das sich mit Aufbau und Struktur der Secure Shell befaßt. Wir stellen aber hier schon die beiden SSH-Protokollversionen gegenüber und befassen uns später nur noch mit der neueren. Der letzte Abschnitt stellt dann den Zusammenhang zu den Protokollen aus der Netzwerkwelt her und beleuchtet deren Zusammenspiel mit SSH. Dazu gehen wir hier auch kurz auf die nötigen Grundlagen ein.

3.1 Kryptographische Grundlagen

Verschlüsselung ist ein integraler Bestandteil von SSH, denn mit ihr wird der Schutz vor unbefugtem Zugriff sichergestellt. Wir geben deswegen an dieser Stelle eine Einführung in die Grundbegriffe der Kryptographie. Sie bildet die Basis für das spätere Verständnis von Aufbau, Struktur und Verwendung von SSH und vermittelt Bedeutung der verwendeten Verfahren und Hilfsmittel.

Kryptographie ist eine in unserer Informationsgesellschaft anerkannte und sich immer weiter verbreitende Methode, Informationen vor unbefugtem Zugriff zu

schützen. Spätestens seit Brieftauben abgefangen und reitenden Boten ihre Nachricht abgenommen werden konnte, hat sich der Bedarf nach einem besonders zuverlässigen Grad an Geheimhaltung ergeben. Mit einer explodierender Menge an auszutauschenden Daten hat sich sicher dieser Bedarf immer weiter erhöht und Kryptographie ist längst keine militärische Domäne mehr. Sie ist aus den Bereichen wie elektronischem Zahlungsverkehr, Zugangskontrolle und Schutz der Privatsphäre nicht mehr wegzudenken.

Kryptographie und Kryptoanalyse sind Teilgebiete der Kryptologie. Kryptographie beschäftigt sich mit der Anwendung mathematischer Methoden und Techniken zum Schutz von Informationen gegen unbefugten Zugriff. Im Kontext von SSH dient sie dazu, die Sicherheit von elektronischen Transaktionen zu erhöhen. Im Gegensatz zu Mauern, Gräben und Vorhängeschlössern ist ihr Schutz mathematisch-logischer Natur. Ziel ist es, eine Nachricht über einen unsicheren Kanal zu versenden, ohne daß jemand anderes als der gewünschte Empfänger sie lesen kann. Bei dem Kanal kann es sich um ein beliebiges Transportmedium handeln, zum Beispiel das gute alte Netzwerkkabel oder aber auch durch die Luft bei drahtloser Übertragung.

Im Gegensatz zur Kryptographie verfolgt die Kryptoanalyse das Ziel, Verschlüsselungen ohne Kenntnis der Schlüssel rückgängig zu machen. Es gibt hier eine ganze Reihe von Ansatzpunkten, die später kurz beleuchtet werden. Zusätzlich führen wir im folgenden vor, wie ein verschlüsselter Nachrichtentransfer aussieht, erklären die Unterschiede zwischen symmetrischen und asymmetrischen Verfahren und gehen kurz darauf ein, welchen Sinn Hash-Funktionen und Digitale Signaturen haben.

3.1.1 Kryptographische Algorithmen und Schlüssel

Um die Nachricht so zu behandeln, daß sie wirklich nur Sender und Empfänger lesen können, kommt ein sogenannter kryptographischer Algorithmus zu Einsatz. Hierbei handelt es sich um einen mathematischen Rechenvorgang, der aus mehreren Schritten besteht. Er kann in Form von Software oder Hardware umgesetzt sein und seine Wirksamkeit beruht darauf, daß ein Angreifer ein gewisses mathematisches Problem nicht zu lösen vermag. Unlösbar, weil er nicht etwa unfähig ist, sondern vielmehr weil ihm bestimmte Informationen fehlen. Diese Informationen sind die oder der sogenannte Schlüssel, mit dessen Hilfe die Nachricht verschlüsselt und für andere unlesbar wird.

Verschlüsselungsalgorithmen können beliebige Daten verschlüsseln. Es ist ihnen egal, ob sie auf die Bauanleitung für das neue Regal oder auf die Personalplanung der Bundesregierung für die nächste Kabinettsumbildung angewendet werden. Darüber hinaus sind sie so flexibel, daß sie die unterschiedlichsten Schlüssel verwenden können (die aber bestimme Eigenschaften haben müssen). Ein Schlüssel

Kapitel 4

Die Secure Shell im Überblick

Seit der Entstehung der Secure Shell hat diese sich kontinuierlich weiterentwickelt und bis zum jetzigen Zeitpunkt etliche Erweiterungen und Verbesserungen erfahren. Aus SSHv1 ist SSHv2 hervorgegangen und ein revidiertes und erweitertes Protokoll hat die ursprüngliche Spezifikation abgelöst. Es sind Implementierungen zu den unterschiedlichen Protokollversionen im Einsatz. In der Masse sind dieses Umsetzungen der SSH-Protokollversionen 1.5 und 2.0.

Im Schwerpunkt geht dieses Buch auf die aktuelle Protokollversionen 2.0 (SSHv2) der Secure Shell ein. Weil sich SSHv1-Produkte aber immer noch großer Beliebtheit erfreuen beziehungsweise verbreitet sind, soll auch diese Protokollversionen in den folgenden Abschnitten vorgestellt werden. Unser Ziel ist es, ein Verständnis für die grundlegenden Unterschiede und die Weiterentwicklungen der Protokollversionen 2 zu vermitteln. Aufbauend auf diesem Wissen kann dann zum Beispiel eine Migration von SSHv1 auf SSHv2 in Angriff genommen werden. Auch in den späteren Kapiteln des Buches werden wir auf Unterschiede zwischen SSHv1 und SSHv2 hinweisen. Um im Protokoll- und Versionsdschungel den Überblick zu behalten, sei noch ein Hinweis gestattet. Es muß zwischen den Versionen von SSH-Implementierungen und der Protokollversion, die sie umsetzen, unterschieden werden. Das Produkt OpenSSH ist in der Version OpenSSH 4.5 aktuell, implementiert aber SSH in der Version 2.0 und 1.5, so daß Abwärtskompatibilität gewährleistet bleibt. Gleiches gilt für andere Implementierungen, etwa den SSH-Client PuTTY.

Weiterhin sollen die folgenden Erläuterungen den theoretischen Unterbau für den SSH-Einsatz liefern. Auch wenn nicht jedes Detail in der Praxis von Bedeutung ist, so lassen sich doch viele Sachverhalte oder spätere Entscheidungen mit diesem Wissen leichter nachvollziehen. Der Schwerpunkt in diesem Abschnitt liegt daher auf der Darstellung des Protokollaufbaus. Dinge mit ganz konkretem Praxisbezug finden sich dann in Kapitel 8.

SSHv2 ist nicht nur eine neuere Version des SSHv1-Protokolls, es ist vielmehr eine Neuentwicklung von Grund auf. Das hat den Sprung auf die Protokollversion 2 gerechtfertigt. Dieses hat außerdem zur Folge, daß beide Protokolle strukturell sehr unterschiedlich und inkompatibel zueinander sind. Nichtsdestotrotz sind viele Client- und Server-Implementierungen in der Lage, Unterstützung für beide Versionen zu leisten. Wie wir noch sehen, wird das verwendete Protokoll zwischen Client und Server ausgehandelt. Der Client kann dabei aus der Menge ihm angebotener Protokolle wählen. Auf diese Weise ist gewährleistet, daß nicht schlagartig alle SSHv1-Clients von der Kommunikation abgeschnitten sind, wenn ihr gewohnter Kommunikationspartner auf der Server-Seite ein Upgrade auf SSHv2 erfährt. Auf keinen Fall sollen unterschiedliche und inkompatible Versionen – wie es SSHv1 und SSHv2 ja sind – die sichere Kommunikation verhindern. Nichts wäre schlimmer, als wenn jemand nach einer gescheiterten Verbindungsaufnahme zum Beispiel frustriert zu `rlogin` zurückwechselt. OpenSSH unterstützt deshalb beide Protokollversionen[1], so daß Kompatibilität gewährleistet und ein endgültiger Umstieg auf SSHv2 durch Deaktivierung von SSHv1 in den Konfigurationsdateien möglich ist.

4.1 Protokolldefinition

Das – oder besser die – Secure Shell-Protokolle, welche allen Implementierungen zugrundeliegen, sind in Spezifikationen niedergelegt. Diese Dokumente beschreiben das Sollverhalten, und nach diesen Richtlinien sind SSH-Produkte entwickelt. Die Produkte unterscheiden sich darin, welche Protokollversion sie unterstützen. Das bedeutet, daß sie unterschiedlich aktuell sind und einen unterschiedlichen Stand der Technik repräsentieren. Auch besitzen sie eine verschiedene Menge an Funktionen. Zusätzlich fügen Hersteller noch eigene Logik hinzu, um sich von Konkurrenten abzusetzen. Die in der Protokollspezifikation beschriebene Funktionalität ist aber der gemeinsame Nenner aller SSH-Implementierungen.

Eine Protokollspezifikation legt im Detail das gewünschte Verhalten fest. Alle Eventualitäten, Sonder- und Grenzfälle sind (beziehungsweise sollten) berücksichtigt sein. Wer also genau wissen will, was im Hintergrund abläuft oder warum etwas auf eine bestimmte Art und Weise gehandhabt wird, der sollte sich mit der Spezifikation beschäftigen. Wer Produkte wie OpenSSH einfach nur anwenden möchte, braucht aber in dieser Tiefe nicht einzusteigen. Für das Gesamtverständnis und einen Überblick über Zusammenhänge und technische Details ist aber ein Blick in die Spezifikation zu empfehlen. Das dort zu findende Wissen ist auch hilfreich bei der Fehlersuche oder bei der Entscheidung, ob ein scheinbar falsches Verhalten tatsächlich einen Fehler darstellt oder nicht doch in Ordnung ist.

[1] Genau genommen unterstützt OpenSSH drei Versionen: 1.3, 1.5 und 2.0

4.1.1 Der Internet-Standardisierungsprozeß

Wenn man sich mit den Spezifikationen zu bestimmten Protokollen befaßt, kommen früher oder später einige Fragen auf. Wer verfaßt diese Spezifikationen? Wer entwickelt sie weiter? Wann wird aus einer Idee und einem Vorschlag ein Internet-Standard? Um das Gesamtbild der Vorgänge um Standards und Spezifikationen zu vervollständigen, geben wir einen Abriß von den Zusammenhängen.

Ihr formales Leben beginnen Ideen als sogenannter *Internet Draft*. Dabei handelt es sich um den Entwurf für etwas, das später mal ein Standard werden soll. Ein solcher Entwurf ist ein Arbeitsdokument der *Internet Engineering Task Force (IETF)* oder einer ihrer Teilorganisationen. Eine davon ist die *Secure Shell (SECSH) Working Group*, die sich um die Weiterentwicklung der Secure Shell kümmert. Ergänzend können aber auch andere Gruppen Drafts herausgeben. Ein Internet-Draft weist bestimmte Charakteristika auf:

- Bei einem Internet-Draft handelt es sich um einen Entwurf und nicht um ein offizielles Dokument.
- Ein Internet-Draft hat eine beschränkte Lebensdauer von sechs Monaten. Bis zu deren Ablauf muß ein Entwurf aktualisiert werden, oder wird gelöscht.
- Entwürfe haben zum Ziel, als RFC (*Request For Comment*) und später auch als Standard verabschiedet zu werden.
- Im Laufe ihrer Bearbeitung durchleben Internet-Drafts eine Reihe von unterschiedlichen Zuständen. Sie werden verfaßt, veröffentlicht, mehrfach durch verschiedene Personen und Gremien geprüft und schließlich als RFC veröffentlicht.
- Internet-Draft haben ein standardisiertes Namensformat:
  draft-<quelle>-<name>-<nummer>
  Sie beginnen mit dem Wort *draft*. Es folgt der Ersteller, zum Beispiel *ietf*. Der eigentliche Name zusammen mit einer laufenden Nummer bildet den Abschluß. Bei Dokumenten von Working Groups taucht ihre Name zu Beginn auf. Insgesamt ergibt sich zum Beispiel: *draft-ietf-secsh-architecture-22.txt*

4.1.2 Secure Shell Protokolle

Für SSHv2 sind eine ganze Reihe von Teilprotokollen relevant (Abschnitt 6.1.1). Viele von ihnen liegen als Internet-Draft vor, alle wichtigen jedoch als RFC[2]:

- RFC 4250: *SSH Protocol Assigned Numbers*
- RFC 4251: *SSH Protocol Architecture*
- RFC 4252: *SSH Authentication Protocol*

[2] Abschnitt 6.1.1 stellt sie kurz vor.

Kapitel 5

SSHv1: Die Grundlagen

Die erste Implementierung des SSHv1-Protokolls, Version 1.0, wurde 1995 veröffentlicht. Die wohl am weitesten verbreitete und auch letzte Version der Einser-Protokollfamilie ist 1.5. Das SSHv1-Protokoll wurde nie abschließend standardisiert, es existiert lediglich ein Protokollentwurf, der die Basis aller Implementierungen ist[1]. Die Abschnitte dieses Kapitels beziehen sich vollständig auf SSHv1. Wenn von SSH-Client, SSH-Server, SSH-Tunnel oder SSH-Agent gesprochen wird, so sind SSHv1 implementierende Produkte gemeint.

5.1 Protokollstruktur

Das SSHv1-Protokoll (im Folgenden auch kurz SSHv1) weist eine monolithische Architektur auf. Das bedeutet, daß es sich nicht aus Komponenten zusammensetzt, sondern vielmehr alle benötigte Funktionalität vollständig in einem Block vereint. Ein Austausch einzelner Funktionen oder eine bedarfsabhängige Erweiterung ist damit nicht möglich.

SSHv1 ist ein paketbasiertes Binärprotokoll, das auf einer beliebigen Transportschicht aufsetzt. Das bedeutet, Informationen werden über Datenpakete in einem bestimmten Format ausgetauscht. Ein Binärprotokoll ist SSHv1, weil die ausgetauschten Informationen nicht rein textueller Natur sind. SSHv1 (und SSHv2) ist außerdem ein Client-Server-Protokoll. Wir haben es also mit einem zentralen Server-Dienst zu tun, der die Anfragen einer Vielzahl von Clients bedient.

Bei der SSHv1 zugrundeliegenden Schicht handelt es sich in der Regel um TCP/IP. Die Kommunikation zwischen SSH-Client und SSH-Server findet über sogenannte Kanäle statt. Details zu den Abläufen des Protokolls finden sich in Abschnitt 5.2. Abbildung 5.1 zeigt den schematischen Aufbau eines SSHv1-Pa-

[1] Im Gegensatz zu SSHv2, dessen wesentliche Teilprotokolle als RFCs verfügbar sind.

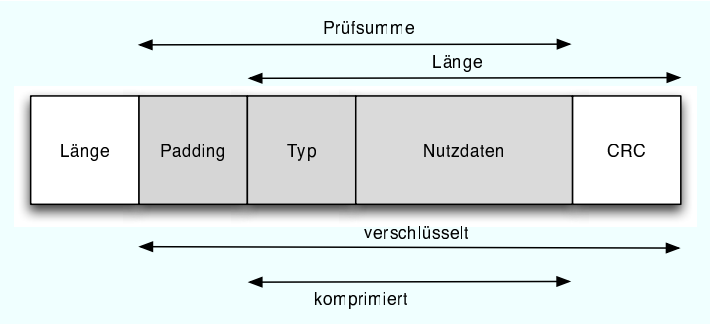

Abbildung 5.1: Aufbau eines SSHv1-Pakets

kets. Solche Pakete kommen im Anschluß an die Phase der Protokollidentifizierung zum Einsatz. In dieser initialen Phase tauschen Client und Server Informationen darüber aus, welche SSH-Versionen sie unterstützen.

Zu Beginn der Kommunikation ist keine Verschlüsselung aktiv. Der später zu verwendende Algorithmus wird vom Client aus dem Server-Angebot ausgewählt[2].

Die einzelnen Felder eines solchen SSHv1-Pakets haben dabei die folgende Bedeutung:

1. *Länge des Datenpakets*
   Das erste Feld enthält die Länge des Pakets und ist 32 Bit groß. Diese Längenangabe versteht sich als Summe aus Typ, Nutzdaten und Prüfsumme. Das Längenfeld selber und die Stopfdaten (Padding) sind also nicht berücksichtigt. Die maximale Länge eines SSHv1-Pakets ist 262 144 Byte (wieder ohne Längenfeld und Padding).
2. *Zufällige Stopfbits (Padding)*
   Im folgenden Feld sind zufällige Füllinformationen eingefügt, das sogenannte *Padding*. Dieses können 1 bis 8 Byte sein, die *known plaintext*-Angriffe erschweren sollen. Aus diesem Grund sind Padding-Daten immer vorhanden. Sie dienen nicht nur dazu, die Daten auf ein Vielfaches von acht zu ergänzen. Die notwendige Padding-Menge berechnet sich so: Padding = (8 - (Paketlänge modulo 8)) Byte.
3. *Pakettyp*
   Das Typ-Feld hat eine Länge von 8 Bit. Es identifiziert den aktuellen Typ des Pakets, so daß klar wird, wozu die enthaltenen Daten verwendet werden müssen. SSHv1 kennt auch Pakete, die nur vom Server oder nur vom Client gesendet werden dürfen. Andere wiederum dürfen beide verschicken. Diese Unterscheidung erfolgt über den Typ.

[2] Dieses ist in SSHv2 noch ausgefeilter, die verwendeten Verfahren werden ausgehandelt

Kapitel 6

SSHv2: Die Grundlagen

SSHv1 und SSHv2 wollen beide dasselbe Ziel erreichen – allerdings auf unterschiedliche Art und Weise. Insgesamt ist SSHv2 sicherer und flexibler. Der größte Unterschied liegt in der Struktur beider Protokolle. Während SSHv1 ein einziges, großes Protokoll darstellt, gliedert sich SSHv2 in mehrere Teilspezifikationen. Diese Kapselung der SSH-Funktionalität in einzelne Komponenten erlaubt eine mächtigere und flexiblere Herangehensweise an die Aufgabe, sicheren Datenaustausch zu ermöglichen.

Dieses Kapitel schildert den Aufbau und die Funktionsweise von SSHv2. Die Basis für diese Erläuterungen bilden die Protokollspezifikationen der einzelnen Protokolle, aus denen sich das SSH-Protokoll zusammensetzt (Abschnitt 6.1). Alle wesentlichen Spezifikationen liegen als RFC mit dem Status: *PROPOSED STANDARD* vor. Zu weiterführenden und ergänzenden Bereichen existieren Entwürfe, die aber recht stabil sind.

Die Kapitel des Praxisteils beschäftigen sich konkret mit OpenSSH und damit ausschließlich mit real verfügbaren Funktionalitäten. Die Abschnitte dieses Kapitels beziehen sich allein auf SSHv2. Wenn von SSH-Client, SSH-Server, SSH-Tunnel, SSH-Agent oder einfach SSH gesprochen wird, so sind SSHv2 implementierende Produkte gemeint.

6.1 Protokollstruktur

Die drei Teilprotokolle, aus denen sich der Kern von SSHv2 zusammensetzt:

- *Verbindungsprotokoll* (RFC 4254)
  Aufgabe des Verbindungsprotokolls ist es, eine flexible Handhabung der sicheren Verbindung zu erlauben.

- *Authentifizierungsprotokoll* (RFC 4252)
  Das Authentifizierungsprotokoll beschreibt, wie sich Clients gegenüber einem SSH-Server authentifizieren.
- *Transportprotokoll* (RFC 4253)
  Die Grundlage von SSHv2 bildet das Transportprotokoll. Es ist zuständig für den anfänglichen Verbindungsaufbau, Server-Authentifizierung und Verschlüsselung.

Der Überblick über die SSHv2-Protokollarchitektur, die Aufgaben der drei Teilprotokolle und ihr Zusammenspiel ist in der Protokollspezifikation *SSH Protocol Architecture* (RFC 4251) [YC06b] zusammengefaßt.

Über die Modularisierung in Einzelprotokolle hinaus setzt SSHv2 einige weitere Maßnahmen um, mit denen es zusätzliche Flexibilität erreicht. Dazu gehören:

- Abwärtskompatibilität
- Erweiterbarkeit
- Namensgebung
- Verbindungseigenschaften

Die folgenden Abschnitte greifen diese Eigenschaften noch einmal auf und beschreiben im Detail, was sie für SSHv2 bedeuten.

Abbildung 6.1 verdeutlicht, wo die Protokolle von SSHv2 im Hinblick auf die Struktur von TCP/IP einzuordnen sind. SSHv2 ist als Ganzes auf Ebene der Anwendungsschicht angesiedelt. Das Transportprotokoll von TCP/IP etwa erfüllt ganz andere Aufgaben als das Transportprotokoll von SSHv2. Tabelle 6.1 grenzt die beiden Transportprotokolle von einander ab.

Innerhalb von SSHv2 ist das Transportprotokoll für die sichere Übertragung der SSH-Pakete zuständig. Das bedeutet, es muß sich um Verschlüsselung und Integrität kümmern. Dazu kommen noch weitere Aufgaben wie Kompression und Host-Authentifizierung. Die Transportschicht von TCP/IP hingegen ist für deutlich grundlegendere Aufgaben zuständig. Ihr fällt es zu, eine Infrastruktur für den allgemeinen Datentransport zur Verfügung zur stellen (zusammen mit IP),

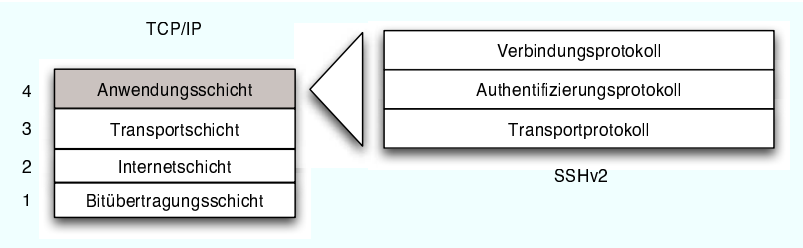

Abbildung 6.1: Einordnung der SSHv2-Protokolle in das TCP/IP-Referenzmodell

Tabelle 6.1: Aufgaben der Transportschichten in TCP/IP und SSHv2

TCP/IP (TCP)	SSHv2
Flußkontrolle	Integrität
Reihenfolgetreue	Verschlüsselung
Zuverlässigkeit	Kompression
	Host-Authentifizierung

unabhängig davon, ob es sich um SSH-Pakete oder Daten anderer Anwendungen oder Protokolle handelt. In der Transportschicht können auch andere Protokolle als TCP eingesetzt werden. Die Regressionstests von OpenSSH etwa nutzen eine *pipe* und OpenSSH unterstützt auch eine serielle Leitung statt TCP.

TCP als Protokoll der Transportschicht kümmert sich um Flußkontrolle, Reihenfolgetreue und Zuverlässigkeit. Datenpakete gehen also nicht verloren bzw. werden dann erneut gesendet, erreichen in der Sender-Reihenfolge den Empfänger und überschwemmen den Empfänger auch nicht (UDP leistet dieses genau nicht).

Bild 6.2 zeigt eine schematische SSHv2-Verbindung. Datenpakete von SSH-Client oder SSH-Server marschieren zuerst durch die Schichten des SSHv2-Protokolls. Dort werden sie verschlüsselt, ggf. komprimiert, eine Prüfsumme wird angehängt und einiges mehr. Anschließend gibt SSH sie weiter an TCP/IP, um den eigentlichen Transport abzuwickeln. Dort werden die Pakete ggf. zerlegt, Routing-Informationen ausgewertet und schließlich werden sie über das physikalische Übertragungsmedium in Richtung Empfänger weitergegeben.

Auf der Gegenseite nehmen die Pakete den umgekehrten Weg, werden wieder zusammengesetzt, dekomprimiert, entschlüsselt und schließlich der Empfängeranwendung übergeben.

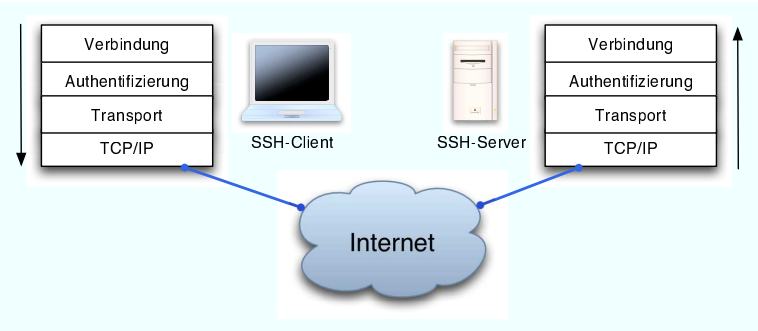

Abbildung 6.2: Datenfluß einer SSHv2-Verbindung durch die Schichten

Kapitel 7

Konfiguration und Setup

In diesem Kapitel gehen wir von der Theorie zur Praxis über. Wir werden intensiv die Server- und Client-Konfiguration betrachten. Am Ende dieses Kapitels öffnen die vermittelten Grundlagen Ihnen die Tür zum Praxiseinsatz (Kapitel 8).

Auch wenn sich dieses Kapitel ausführlich mit der Konfiguration von OpenSSH beschäftigt, so muß vorweg angemerkt werden: Notwendig ist eine solche aufwendige Konfiguration für die Nutzung keineswegs. OpenSSH ist ein Musterbeispiel dafür, wie sich innerhalb kürzester Zeit ein lauffähiges und gleichzeitig recht sicheres Setup erreichen läßt. Der Umfang dieses Kapitels ist lediglich ein Indiz dafür, daß sich OpenSSH zusätzlich flexibel und feingranular konfigurieren läßt.

Die Konfiguration für den OpenSSH-Server und OpenSSH-Client ist jeweils in einen einführenden und einen weitergehenden Abschnitt unterteilt. Die Einführung stellt den Unterbau dar und hilft, eine lauffähiges Client/Server-System auf die Beine zu stellen. Dieses erleichtert einen OpenSSH-Schnellstart, ohne sich mit den Details im einzelnen beschäftigen zu müssen. So können die Beispiele nachvollzogen werden und eine umfangreiche Konfiguration ist nicht nötig. Grundsätzlich sind folgende Schritte wichtig:

- Installation von OpenSSH (falls nötig)
- Starten des OpenSSH-Servers
- Aufbau einer Client/Server-Verbindung

Die Erweiterungsabschnitte vertiefen das eingeführte Wissen umfassend. Wer nur erste Gehversuche mit OpenSSH wagen möchte, kann diese Erweiterungen getrost überspringen und später nachholen. Wer aber alle Details von OpenSSH kennenlernen möchte, sollte sich in den Vertiefungsabschnitten gut aufgehoben fühlen.

Anschließend beschäftigen wir uns mit weiteren Möglichkeiten von OpenSSH. Das beinhaltet den Umgang mit den verschiedenen Authentifizierungsmethoden, um den unterschiedlichen Sicherheitsansprüchen gerecht zu werden.

Das Kapitel schließt mit einem empfohlenen Setup. Hier erfahren Sie, auf welche Einstellungen Sie getrost verzichten können und welche im Gegenteil geradezu unverzichtbar sind.

7.1 Hallo, Welt!

Genug der trockenen Theorie, hier beginnt der Praxisteil. Bevor wir mit der systematischen Besprechung der OpenSSH richtig durchstarten, machen wir zur Motivation ein paar kleine Fingerübungen. Die Perspektive auf das, was mit der SSH alles möglich ist, hilft einem dabei, sich durch den großen Umfang an Funktionen durchzuarbeiten, mit dem das Softwarepaket aufwartet. Der Anspruch dieses Abschnitts ist nicht, bereits alle vorgeführten Funktionen umfassend zu erklären. Vielmehr folgt nun eine Kurzvorstellung in Schlaglichtern, um ein Gefühl dafür zu vermitteln, womit wir uns im Rest des Buchs beschäftigen werden.

Die Voraussetzung, um diesen Abschnitt auch praktisch nachvollziehen zu können, ist ein Rechner, der unter der eigenen administrativen Kontrolle steht und auf dem ein Secure-Shell-Dienst läuft[1]. Das realistische Szenario besteht zwar aus Client und Server, aber wahrscheinlich steht nicht jedem Leser spontan ein zweiter Rechner zur Verfügung, auf dem die Experimente durchgeführt werden können. Daher ist es auch kein Problem, die beschriebenen Beispiele mit nur einem Rechner nachzustellen. Hierbei befinden sich Secure-Shell-Client und -Server auf demselben Rechner, und die Vorstellungskraft des Lesers ist gefragt, um das Szenario auf ein echtes Netzwerk abzubilden.

7.1.1 Der einfachste Fall

Das häufigste Szenario ist gleichzeitig das simpelste: Wir loggen uns auf einem entfernten Rechner ein, auf dem wir ebenfalls einen Benutzerzugang mit dem gleichen Login-Namen besitzen. Der entfernte Rechner heißt hier `zielhost`.

```
user@linux: > ssh zielhost
user@zielhost's password:
user@zielhost: >
```

Und schon sind wir eingeloggt! Abhängig von der Konfiguration des Zielrechners sehen wir noch begrüßenden Text, haben aber am Schluß eine Shell zur Verfügung, mit der wir dort Befehle absetzen können.

[1] Informationen zur Installation finden Sie in diesem Kapitel in Abschnitt 7.2

Dieses Beispiel ist gleichzeitig ein hervorragender Test, ob die prinzipielle Konfiguration von Client und Server in Ordnung ist. Das werden wir nämlich bei allen folgenden Beispielen voraussetzen.

7.1.2 Nur ein Befehl

Manchmal möchte man nur einen einzigen Befehl auf dem entfernten Rechner absetzen. Hier wäre eine eigens geöffnete interaktive Shell möglicherweise umständlicher, als den Befehl einfach direkt beim Aufruf mit zu übergeben.

```
user@linux: > ssh zielhost echo Hallo Welt
user@zielhost's password:
Hallo Welt
user@linux: >
```

Mit diesem Beispiel haben wir zwei Fliegen mit einer Klappe geschlagen: Wir haben das für jedes Computerbuch traditionell so wichtige Hallo-Welt-Programm ausgeführt, und außerdem gezeigt, daß die Ausführung von Einzeilern auf entfernten Rechnern problemlos möglich ist. Bedeutend ist bei diesem Beispiel, daß das `Hallo Welt` tatsächlich auf dem entfernten Rechner ausgeführt wurde, und nicht lokal. Nur die Ausgabe des Befehls wurde auf den lokalen Rechner geleitet. Um diese Tatsache zu überprüfen, können Sie ja einfach einen Befehl absetzen, der eine Datei anlegt – diese Datei wird ebenfalls auf dem entfernten Rechner entstehen, nicht auf dem lokalen.

7.1.3 Echte Shell mit Pipes

Die SSH ist tatsächlich eine echte Shell, das heißt die Ein- und Ausgabeumleitung funktioniert ebenso wie im lokalen Fall. Nehmen wir beispielsweise an, es soll ein Verzeichnis mit wichtigen Daten rekursiv auf einen anderen Rechner gesichert werden, aber es stehen nur Betriebssystem-Bordmittel zur Verfügung, und keine Zusatzsoftware. Eine gangbare Möglichkeit wäre dann das lokale Starten eines Archivierers (`tar`), die Umleitung seiner Ausgabe durch die Secure Shell hindurch, und das Wiedereinbringen des Ausgabestroms in einen `tar` auf der entfernten Maschine.

```
user@linux: > find daten/
daten
daten/foo
daten/bar
daten/bar/baz
user@linux: > tar -cvf - daten/ | ssh zielhost tar -xpf -
user@zielhost's password:
daten
```

1. Allgemeingültige Standardeinstellungen mit der Wahrscheinlichkeit zur häufigsten Nutzung gehören in die systemweite `/etc/ssh/ssh_config`. Dabei handelt es sich oft um Vorgaben des Administrators, die mit Kenntnis und Rücksicht auf die gesamte Netzstruktur gewählt wurden. Sie gelten auch für Nutzer, die keine eigene Konfigurationsdatei besitzen.
2. Einstellungen, die dem Nutzungsprofil einzelner Anwender Rechnung tragen, finden ihren Eingang in die nutzerspezifische Datei `~/.ssh/config`. Häufig genutzte Ziele mit Parametern, die von den globalen Einstellungen abweichen, sind hier gut aufgehoben.
3. Kommandozeilenparameter finden ihren Einsatz bei Bedarf und meistens im Einzelfall. Werden bestimmte häufig genutzt, ist ein Eintrag in die `~/.ssh/config` zu prüfen.

7.4 Grundlegende Server-Konfiguration

In Kapitel 6 haben wir die theoretischen Grundlagen für die folgenden Sachverhalte geschaffen. Mit ihnen wenden wir uns nun zuerst dem OpenSSH-Server und im anschließenden Abschnitt dem OpenSSH-Client zu. Der OpenSSH-Server bringt schon einige sinnvolle Standardeinstellungen mit. Darüber hinaus ist er sehr differenziert konfigurierbar. Die Konfiguration des Servers ist auf mehreren Ebenen möglich. Die beiden Möglichkeiten sind:

1. Serverweite Konfiguration
2. Account-bezogene Konfiguration

Wir setzen uns in diesem Abschnitt zunächst mit der serverweiten Konfiguration auseinander, die das globale Laufzeitverhalten des Servers beeinflußt. Später gehen wir im Rahmen der erweiterten Konfiguration noch auf die Account-orientierte Server-Konfiguration ein (Abschnitt 7.7.3).

Die folgenden Ausführungen sind so gegliedert, daß zuerst die grundlegende und im späteren Abschnitt 7.7 die erweiterte Server-Konfiguration betrachtet wird. Die Grundlagen umfassen dabei im wesentlichen das Starten und einen Funktionstest des Servers. Die Erweiterungen beschäftigen sich unter anderem mit einer detaillierten Betrachtung der Konfigurationsoptionen. Die gleiche Gliederung findet sich auch bei der Behandlung der Clients.

Ziel der folgenden Erklärungen ist es, mit möglichst wenig Aufwand ein lauffähiges OpenSSH-System zu erhalten. Einige grundlegende Einstellungen sind erforderlich, im Vordergrund steht aber, schnell die ersten Gehversuche mit einem sinnvoll eingestellten OpenSSH-System unternehmen zu können.

7.4.1 Einstellen der Basiskonfiguration

Etliche Parameter werden bei der Installation schon gesetzt beziehungsweise haben sinnvolle Standardwerte. So ist zum Beispiel die Option `PermitEmptyPasswords` standardmäßig auf auf `no` gesetzt, denn niemand sollte sich ohne Paßwort sofort einloggen können. Grundlegend ist der OpenSSH-Server so konfiguriert, daß nach dem Starten des Servers sofort ein Verbindungsaufbau möglich ist. Natürlich sind erweiterte Authentifizierungsmethoden wie zum Beispiel `GSSAPIAuthentication` dann noch nicht konfiguriert oder aktiviert.

Mit den sinnvollen Standardwerten ist OpenSSH auch ohne Konfiguration schon gut einsetzbar. Ganz besonders wird dies deutlich am Beispiel der *secure by default*-Mentalität des OpenBSD-Projekts, in dem OpenSSH seine Herkunft hat. Die Authentifizierung über Public Keys etwa ist von Beginn an aktiviert.

Für den Anfang und die ersten Schritte beschränken wir uns auf die Authentifizierung über Paßwörter. Grundsätzlich raten wir aber zur Verwendung von Public Keys. Diese Technik wird in späteren Abschnitten vorgestellt.

Auch wenn die Standardwerte sinnvoll vorgegeben sind, möchten wir doch noch auf einige Parameter eingehen, die nach der Installation angepaßt oder geprüft werden sollten. Sie finden sich in der Konfigurationsdatei `/etc/ssh/sshd_config`. Wir gehen auf diese Parameter hier aber nicht im Detail ein, das geschieht in der erweiterten Konfiguration.

Die in diesem Moment interessanten Parameter aus `/etc/ssh/sshd_config` nennt Tabelle 7.8. Sie stellt den Standardwert und unsere Empfehlung gegenüber. In einigen Fällen stimmt die Empfehlung mit dem Standard überein. Hier ist es schlicht sinnvoll, dies noch einmal zu prüfen und die gewünschte Einstellung sicherzustellen.

Tabelle 7.8: Anfang einzustellende Optionen in `sshd_config`

Option	Standardwert	Empfehlung
`Protocol`	`2,1`	`2`
`ListenAddress`	<leer>	<IP Adresse>
`PermitRootLogin`	`yes`	`no`
`MaxAuthTries`	6	`3`
`PasswordAuthentication`	`yes`	`no`
`PermitEmptyPasswords`	`no`	`no`
`UsePAM`	`no`	`yes`
`X11Forwarding`	`no`	`no`

```
linux: # cd /etc/init.d
linux:/etc/init.d # ls -l *sshbuch*
lrwxrwxrwx  1 root root 10 May  1 12:00 K21sshbuch -> ../sshbuch
lrwxrwxrwx  1 root root 10 May  1 12:00 S01sshbuch -> ../sshbuch
```

Wie Sie sehen, wurden zwei Links erstellt, die den Prozeß beenden bzw. starten. Über den Befehl `insserv` mit der Option `-r` läßt sich die Registrierung der Testdatei wieder entfernen.

```
linux:/etc/init.d # insserv -r sshbuch
```

Wir haben Ihnen drei Varianten vorgestellt, mit denen Sie Skripte in verschiedene Runlevel eintragen können. Sicherlich gibt es noch andere Methoden, wie zum Beispiel das Setzen eines Links. Eine generelle Empfehlung, welches die beste Methode ist, gibt es nicht. Das müssen Sie von Fall zu Fall entscheiden. Generell sollten Sie unter SUSE LINUX die Verwendung von `rc`-Skripts bevorzugen (Vorlagen dazu finden Sie in `/etc/init.d/skeleton`). Diese Variante ist allerdings auch aufwendiger und ergibt sicherlich nur bei Programmen Sinn. Wenn Sie in einem Skript einfach nur eine Hand voll Befehle ausführen, dann sollte ein Link ausreichend sein.

7.5 Grundlegende Client-Konfiguration

Das Gegenstück des Servers im SSH-Protokoll ist der Client. Ebenso wie für den Server gilt auch auf Seiten des Client, daß etliche Konfigurationseinstellungen möglich sind und sich so im Zusammenspiel zwischen den beiden eine nicht zu unterschätzende Komplexität ergibt.

Wie die Server-bezogenen Beschreibungen haben wir unsere Erklärungen zum Client in einen grundlegenden und einen weiterführenden Teil gegliedert. Diese Struktur soll es Ihnen ermöglichen, nur mit Hilfe der Grundlagen schon etliche Möglichkeiten ausloten zu können. Dazu gehört das Aufbauen von Sitzungen oder das Kopieren von Dateien über eine verschlüsselte Verbindung. Die Erläuterungen der erweiterten Konfiguration gehen dann stärker ins Detail und können bei Bedarf intensiver studiert werden.

Neben dem eigentlichen SSH-Client `ssh` liefert des OpenSSH-Paket noch weitere Client-Programme mit. Dazu gehören:

- `scp`
- `sftp`
- `ssh-add`
- `ssh-agent`
- `ssh-keygen`

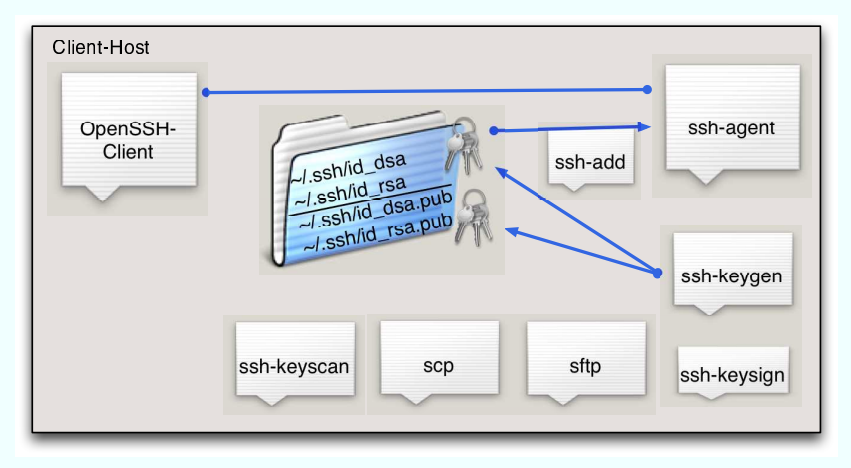

Abbildung 7.3: Client-Seite der OpenSSH-Anwendungen

- `ssh-keyscan`
- `ssh-keysign` (für interne Zwecke)

Diesen speziellen Basisanwendungen widmet sich Kapitel 8.

Abbildung 7.3 zeigt die Client-seitigen Komponenten, die im Zusammenhang mit OpenSSH eine Rolle spielen. Abbildung 7.1 in Abschnitt 7.3 hat bereits einen Überblick gegeben. Die Anwendungen `ssh-agent`, `ssh-add` und `ssh-keygen` kommen im Rahmen der *publickey*-Authentifizierung zum Einsatz.

`ssh-keysign` wird bei der *hostbased*-Authentifizierung benötigt und die Clients `scp` und `sftp` dienen der sicheren Dateiübertragung (im Fall von `sftp` auch mehr).

`ssh-keyscan` schließlich prüft Systeme im Netzwerk und sammelt öffentliche Host-Schlüssel. Damit läßt sich einfach eine Known-Hosts-Datei aufbauen, die Teil der Server-Host-Authentifizierung ist.

Für die ersten Schritte muß ein OpenSSH-Client nicht umfangreich konfiguriert werden. Schon mit den Standardwerten läßt sich eine Verbindung zum OpenSSH-Server herstellen. Dennoch gehen wir zu Beginn auf einige Parameter ein, die zu der Grundkonfiguration eines OpenSSH-Clients zählen. Die Parameter finden Sie in der globalen Konfigurationsdatei `/etc/ssh_config` oder in der benutzerspezifischen Konfigurationsdatei `~/.ssh/config`. Es sind im einzelnen:

- `Port`
- `Host`
- `Protocol`

Kapitel 8

OpenSSH im Einsatz

Im Kapitel 7 haben wir uns ausführlich die Grundlagen der Server- und der Client-Konfiguration angesehen. Ergänzt wurde dieses durch erste Gehversuche mit dem Client `ssh`.

Darauf aufbauend folgt jetzt eine detailliertere Betrachtung des OpenSSH-Clients `ssh`. Einen weiteren wichtigen Bereich bildet der Abschnitt über Weiterleitungen. Zusätzlich lernen Sie in diesem Kapitel weitere Clients von OpenSSH kennen. Einige davon beschäftigen sich mit dem Management von Schlüsseln. Dazu kommt noch die Betrachtung von SFTP als Werkzeug zum Datentransfer.

Dieses Kapitel endet mit einigen gängigen Beispielen zum Einsatz von OpenSSH in Unternehmen.

8.1 Schlüsselverwaltung und Agenten

Schlüssel spielen im Zusammenhang mit der Secure Shell eine zentrale Rolle. Sie dienen der Authentifizierung, einem Aspekt, mit dem die Vertrauenswürdigkeit steht und fällt. Bei der Authentifizierungsmethode *publickey* kommt das Schlüsselpaar des Nutzers zum Einsatz, und Host-Schlüssel sind beim Verbindungsaufbau zwischen Client und Server immer beteiligt.

OpenSSH bringt zur Verwaltung der Schlüssel einige Werkzeuge mit. Dazu gehören unter anderem Programme zum Erzeugen von neuen Schlüsselpaaren, dann sogenannte Agenten, die den Umgang mit Schlüsseln durch Caching erleichtern sollen und schließlich Tools zur Konvertierung und zum Auffinden von Schlüsseln im Netzwerk. Diesen und anderen Werkzeugen widmen sich die folgenden Abschnitte.

Die Reihenfolge der Abschnitte orientiert sich am Vorgehen in der Praxis. Der Vorstellung einiger allgemeiner Informationen zu Beginn folgt die Schlüsseler-

zeugung, deren lokale Verwaltung und ihr Transfer zum OpenSSH-Server. Den Abschluß bildet die Schlüsselkonvertierung und das Scannen von Hosts nach verfügbaren Schlüsseln. Diese Abschnitte lassen sich also gut der Reihe nach lesen. Wer allerdings nur Informationen zu einem bestimmten Aspekt benötigt, kann genauso gut auch dort direkt einsteigen.

OpenSSH kennt die folgenden Clients, die auch Basisanwendungen genannt werden:

- `ssh-keygen`
  Erzeugung von Schlüsselpaaren und Konvertierung vom und in das *Public Key File Format*
- `ssh-agent`
  Verwaltung privater Schlüssel und Caching im Speicher. Die wiederholte Eingabe der Passphrase entfällt.
- `ssh-add`
  Hinzufügen von privaten Schüsseln zum Agenten
- `ssh-keyscan`
  Suche nach Host-Schlüssels im Netzwerk

8.1.1 Schlüssel-Management

Schlüssel in verschiedenen Ausprägungen sind ein zentrales Element bei der Verwendung der Secure Shell. Im Hinblick hierauf klären die folgenden Abschnitte einige wichtige Begriffe, die für das weitere Verständnis von Bedeutung sind.

8.1.1.1 Identitäten

Ein Schlüsselpaar der Public Key Kryptographie (Abschnitt 3.1.3) bezeichnet man im Zusammenhang mit SSH auch als *Identität*. Eine Identität ist damit aus einem öffentlichen und einem privaten Schlüssel aufgebaut.

Der private Schüssel repräsentiert die Identität eines Nutzers für ausgehende Verbindungen. Sobald ein Nutzer unter seinem Account einen OpenSSH-Client, wie `ssh` oder `scp` ausführt und damit eine Verbindung zum OpenSSH-Server aufbaut, wird dieser Schlüssel verwendet. Damit beweist der Nutzer seine Identität gegenüber dem Server.

Der öffentliche Schüssel bildet das Gegenstück zum privaten. Er repräsentiert die Nutzer-Identität bei eingehenden Verbindungen. Fordert ein OpenSSH-Client mit dem entsprechenden privaten Schlüssel eine Verbindung zu einem Server-Account an, so untersucht der OpenSSH-Server den zugehörigen öffentlichen Schlüssel. Passen diese beiden Schlüssel zueinander – sie sind Schlüssel desselben Paares – so ist der Benutzer authentifiziert und die Verbindung wird auf-

gebaut. Öffentliche Schlüssel müssen nicht geheimgehalten werden, da sie nicht genutzt werden können, um in ein Account einzubrechen. Sie müssen vielmehr auf den OpenSSH-Servern verfügbar sein, damit eine Verbindung überhaupt aufgebaut werden kann.

Ein OpenSSH-Schlüsselpaar wird normalerweise in Form von zwei Dateien im Verzeichnis `~/.ssh/` mit ähnlichen Namen abgelegt. Beim öffentlichen Schlüssel wird per Konvention die Endung `.pub` angehängt. Wenn also der private Schüssel `mykey` heißt, so ist der dazugehörige öffentliche Schlüssel `mykey.pub` benannt. Standardmäßig werden SSHv1 Schlüsselpaare `identity/identity.pub` genannt und SSHv2 Schlüsselpaare `id_rsa/id_rsa.pub` beziehungsweise `id_dsa/id_dsa.pub`.

Identitäten unter SSHv1 Von der Verwendung von SSHv1 raten wir aufgrund von Sicherheitsbedenken ab. Für eine Migration zu SSHv2 (Abschnitt 9.2) ist es aber sinnvoll, die Verfahrensweisen beider Versionen zu kennen.

Eine SSHv1-Identität ist in zwei Dateien gespeichert, die unter `~/.ssh/` (Standardeinstellung) abgelegt sind. Sofern nicht anders gewünscht, wird der private Schlüssel in der Datei `identity` und der öffentliche Schlüssel in der Datei `identity.pub` gespeichert.

Bevor der öffentliche Schlüssel für die Authentifizierung verwendet werden kann, muß er noch auf den OpenSSH-Server kopiert werden. Wie eine Identität erzeugt wird, erklärt Abschnitt 8.1.2.

Identitäten unter SSHv2 Ähnlich wie bei SSHv1 wird das SSHv2-Schlüsselpaar in zwei Dateien im Verzeichnis `~/.ssh/` abgelegt. Auch die Namensform der beiden Dateien weist dieselbe Struktur auf. Es wird der Suffix `.pub` an den öffentlichen Schlüssel angehängt. Der Name des Schlüsselpaares beinhaltet den verwendeten Verschlüsselungsalgorithmus. So wird zum Beispiel ein privater RSA-Schlüssel in der Datei `id_rsa` abgelegt oder ein DSA-Schlüssel in der Datei `id_dsa`.

Sowohl SSHv1- als auch SSHv2-Schlüsselpaare werden mit Hilfe des Befehls `ssh-keygen` generiert. Bevor der öffentliche Schlüssel aber zur Authentifizierung verwendet werden kann, muß er noch auf den OpenSSH-Server kopiert werden.

Seit der OpenSSH Version > 2.9.9p1 wird der öffentliche Schlüssel in der Datei `~/.ssh/authorized_keys` abgelegt und nicht in der Datei `~/.ssh/authorized_keys2`. Wenn Sie allerdings auf einen OpenSSH-Server Version < 2.9.9p1 zugreifen wollen, so ist hier der öffentliche Schlüssel in der Datei `~/.ssh/authorized_keys2` abzulegen.

Inhalt der `passwd` verglichen. Die allgemeinen Eigenschaften und den Ablauf beim Hashing beschreibt Abschnitt 3.1.5.

Die Beispiele dieses Abschnitts zeigen diese Aktionen:

- Hashing der Datei `~/.ssh/known_hosts`
- Entfernen von Hosts aus der Datei im Hash-Format

Um die Known-Hosts-Datei – auch jederzeit nachträglich – durch Hashing umzustellen, muß `ssh-keygen` mit der Option `-H` verwendet werden.

```
user@linux: > ssh-keygen -H
/home/user/.ssh/known_hosts updated.
Original contents retained as /home/user/.ssh/known_hosts.old
WARNING: /home/user/.ssh/known_hosts.old contains unhashed entries
Delete this file to ensure privacy of hostnames
```

Die Datei `known_hosts.old` beinhaltet die vorherige Version, so daß Sie noch die Möglichkeit haben, die Änderungen rückgängig zu machen. Sie sollten allerdings die alte Known-Hosts-Datei (`~/.ssh/known_hosts.old`) löschen, ansonsten ist nichts gewonnen.

Ein Blick in die neue Known-Hosts-Datei zeigt die Veränderungen.

```
user@linux: > cat /home/user/.ssh/known_hosts
|1|dTIaYhNULFLXQsnVKkTxs9GYDfA=|pLOLcxqtVPtQ/x3GKQNeajx/aO4= ss➥
h-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAvUD70Ml4wKANuCIW6iPMbXrcQbZlg➥
vh3ykgIs9tswEs7jU9KIFE2FT6cxYcwTcZ9bEvxt452I8pjEpexsmq+tAgaFi4Q➥
wdu8jfYxJ/gwXQFG0tOXU1B/Stt1fu61efFrUd76qA8w5xMR+Y5nUNOgBwcecJ5➥
s7zKHCL2Lnqe0gkE=
```

Es ergibt sich die Frage, wie und warum man nicht mehr gültige Hosts aus der Datei entfernt. Das ist dann der Fall, wenn diesen Hosts nicht mehr vertraut werden kann, etwa weil sie eine neue Adresse bekommen haben oder ganz vom Netz genommen wurden. Die manuelle Entfernung ist nicht mehr möglich, da wegen der Hash-Darstellung die Einträge nicht mehr unmittelbar verständlich sind. Als Hilfsmittel liefert `ssh-keygen` die Optionen `-R`, so daß ein Aufruf von `ssh-keygen -R sshserver` den Host `sshserver` aus der Known-Hosts-Datei entfernt. Neue Server hingegen werden automatisch eingetragen.

```
user@linux: > ssh-keygen -R sshserver
/home/user/.ssh/known_hosts updated.
Original contents retained as /home/user/.ssh/known_hosts.old
```

8.1.3 SSH-Agenten

Der SSH-Agent ist ein Programm, das private Nutzerschlüssel zwischenspeichert und auf authentifizierungsbezogene Anfragen antwortet (s. Abschnitt 6.7). Es

sorgt dafür, daß nach dem erstmaligen Laden eines Schlüssel bei der weiteren Benutzung keine Passphrase mehr eingeben werden muß. Das wäre sonst immer dann nötig, wenn eine Operation mit Hilfe des privaten Schlüssels auszuführen ist. Ein Agent dient damit der Bequemlichkeit des Nutzers. Besonders nützlich ist er, wenn mehrere Schlüsselpaare im Einsatz sind und der Nutzer sonst immer die passende Passphrase zu jedem Schlüssel parat haben müßte.

Da der Agent in der lokalen Umgebung des Nutzers aktiv ist, wandern niemals private Schlüssel über eine Netzverbindung. Alle Operationen, die den Einsatz des privaten Schlüssels erfordern, nimmt der Agent außerdem selber vor. Er gibt den Schlüssel nicht aus der Hand, sondern leitet nur das Arbeitsergebnis zurück. Eine typische Aufgabe ist das Signieren von Daten.

Abschnitt 6.7 schildert Agenten und ihre Arbeitsweise ausführlich. Abbildung 8.2 vermittelt noch einmal einen kurzen Eindruck vom Szenario eines Agenteneinsatzes.

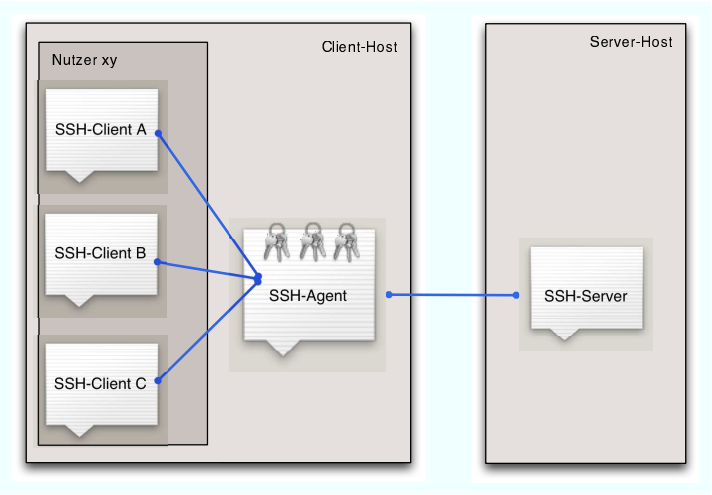

Abbildung 8.2: Einsatz von Agenten

Agenten erfüllen im wesentlichen drei Aufgaben:

- Schlüsselverwaltung (Cache)
- Beantworten von SSH-Client-Anfragen
- Operationen mit dem privaten Schlüssel (etwa Erstellung von Signaturen)

Damit diese Aufgaben erfüllt werden können, muß zum einen der SSH-Agent gestartet und zum anderen ein Schlüssel zum Agenten hinzugefügt sein.

Die beiden OpenSSH-Basisanwendungen `ssh-agent` und `ssh-add` arbeiten im Zusammenhang mit Agenten und Schlüsselverwaltung eng zusammen. Die fol-

genden Abschnitte stellen sie und ihre Befehlsoptionen nacheinander vor, die sich anschließenden Beispiele betrachten sie jedoch gemeinsam.

8.1.3.1 Befehlsoptionen: ssh-agent

Der `ssh-agent` ist ein Programm zur Verwaltung von privaten Schlüsseln für die Authentifizierungsmethode *publickey*. `ssh-agent` verwaltet alle ihm mit `ssh-add` zugewiesen Schlüssel, denn der Agent selber lädt keine Schlüssel. Alle Schlüssel müssen dem Agenten übergeben werden, er startet immer „leer“. Ein Agent ist pro Login-Sitzung aktiv. Sollen Agenten in mehreren Sitzungen verwendet werden, ist jeweils ein eigener erforderlich.

`ssh-agent` legt beim Start einen Socket, an über den es kommuniziert. Zu diesem Zweck belegt es zwei Umgebungsvariablen (Abschnitt 7.8.1.1):

1. `SSH_AUTH_SOCK`: Socket zur Kommunikation mit dem Agenten
2. `SSH_AGENT_PID`: Prozeß-ID des Agenten

Der OpenSSH-Client `ssh` wertet diese Umgebungsvariablen für die Interaktion mit dem Agenten aus.

Es gibt zwei Alternativen, einen Agenten zu betreiben:

1. Starten einer neuen Umgebung über den Agenten
   Gibt man einem Agenten ein Kommando für eine neue Umgebung mit, setzt er in dieser die genannten Umgebungsvariablen. Die Man-Page nennt als Beispiel `ssh-agent xterm &`.
2. Starten des Agenten in aktueller Umgebung
   `ssh-agent` erzeugt passende Shell-Ausgaben (Bourne oder C-shell): `eval 'ssh-agent -[c|s]'`.

Die Tabelle 8.2 zeigt einen Überblick über die Parameter von `ssh-agent`.

Die Parameter haben im einzelnen die im folgenden beschriebenen Bedeutungen.

Tabelle 8.2: Befehlsoptionen von `ssh-agent`

Option	Bedeutung
`-a`	Bindung des Agenten an ein Socket
`-c`	Generiert C-Shell Ausgaben auf `stdout`
`-d`	Debug-Modus
`-k`	Beendet den Agenten
`-s`	Generiert Bourne Shell Ausgaben auf `stdout`
`-t`	Speicherdauer der Schlüssel

```
user@linux: > cat batchdatei
cd /tmp
rm *sess*
```

Das Kommando `sftp -b batchdatei sshserver` führt `sftp` im Batch-Modus aus und verwendet dazu die angegebene Datei. Die zugehörige Ausgabe in `stdout` bei direkter Ausführung ist:

```
user@linux: > sftp -b batchdatei sshserver
sftp> cd /tmp
sftp> rm *sess*
Removing /tmp/sessioID769890
Removing /tmp/sessioID769891
Removing /tmp/sessioID769892
Removing /tmp/sessioID769893
sftp>
```

8.5 OpenSSH in Unternehmensnetzen

Dieser Teil möchte einige Beispiele für den Einsatz von OpenSSH in Unternehmensnetzen vorführen. Gewissermaßen wird hier aus dem Nähkästchen geplaudert und einige Aspekte aus der Praxis beleuchtet. Die hier vorgestellten Beispiele sollen aufzeigen, wie tägliche Probleme mit OpenSSH gelöst werden können. Natürlich stellen sie nur eine kleine Auswahl aus einer Vielzahl denkbarer Einsatzgebiete dar.

8.5.1 SSH-Tunnel

SSH-Tunnel haben wir bereits in Abschnitt 8.2 vorgestellt. Die grundsätzlichen Einsatzmöglichkeiten sollen jetzt durch Beispiele veranschaulicht werden. Das soll die Basis für Ihre eigenen Ideen zu individuellen Lösung von Problemen in Ihrer Umgebung sein.

Viele Administratoren benutzen zum Service Monitoring das Open Source Tool *Nagios* (`http://www.nagios.org`). Nagios basiert auf verschiedenen Plugins, die verwendet werden, um die einzelnen Dienste zu analysieren. Unter anderem ist es darüber möglich, *PostgreSQL*-Datenbanken abzufragen. Diese Datenbank bindet standardmäßig den Port 5432, über den man sich zu den einzelnen Datenbank-Instanzen verbinden kann. Diese Verbindung ist jedoch nicht verschlüsselt. Was innerhalb des eigenes Netzwerkes schon ein Problem sein kann, wächst zu einem nicht akzeptablen Zustand heran, wenn man entfernte Server überwachen muß. Aus unterschiedlichen Gründen ist ein VPN Zugang nicht immer möglich, aber unter Umständen man hat zur Administration des Servers

einen SSH-Zugang zur Verfügung. Was liegt hier näher, als einen SSH-Tunnel zu eröffnen, um darüber die nötigen Informationen zu beziehen?

Nehmen wir also an, Sie haben einen Server-Host namens `nagios`, der die verschiedenen Dienste und Server beobachtet und auf Störungen reagiert. Ferner müssen Sie die PostgreSQL-Datenbank auf dem Server-Host `customer` überprüfen. Wie müßte der SSH-Tunnel aussehen, wenn Sie sich auf dem Server `nagios` befinden?

Die Lösung lautet: `ssh -L 5432:127.0.0.1:5432 customer`

Dieser Befehl eröffnet eine lokale Weiterleitung, die sich an das lokale Interface (`localhost`) und den Port 5432 bindet. Nun müssen Sie den Nagios-Server noch so konfigurieren, daß dieser anstatt auf `customer:5432` auf `localhost:5432` die PostgreSQL-Datenbank abfragt (aber dem Rechner `customer` zuordnet).

Weil es sich beim obigen Befehl aber um einen ssh-Login handelt, den man als Benutzer mit korrekter Authentifizierung macht, liegt es nahe, daß für einen automatischen Betrieb im Nagios noch ein bißchen mehr zu tun ist. Wir wollen kurz darauf eingehen, worum es sich handelt, ohne allzusehr in die Tiefe zu gehen, weil die Bedienung von Nagios nicht im Fokus dieses Buches liegt.

Damit Nagios jedes Mal, wenn eine Überprüfung des SQL-Portes des Rechners `customer` ansteht, auch Zugriff auf den Port hat, müssen wir dem Nagios-Proze"ß auch den Zugang ermöglichen. Folgende Schritte sind dazu nötig:

1. Erstellen eines ssh-Keys ohne Passphrase, der nur für den Nagios-Zugriff genutzt werden soll.
2. Anlegen eines Benutzeraccounts `nagios` oder (falls dieser schon existiert und anderweitig genutzt wird) eines anderen passenden Namens.
3. Kopieren des ssh-pubkeys von obigem generiertem Schlüssel in das Homeverzeichnis des gerade angelegten Nutzers unter `~/.ssh/authorized_keys`
4. Ganz wichtig: Einschränken der Möglichkeiten dieses Schlüssels durch Voranstellen von ssh-Keyoptionen in der `authorized_keys`. Ein Vorschlag wäre:
   ```
   from="nagioshost",command="",permitopen="127.0.0.1:5432",
   no-X11-forwarding,no-agent-forwarding,no-pty
   ```
   Hierbei wird angenommen, daß es sich beim Rechner `nagioshost` um den Nagios-Rechner handelt, von dem aus die Verbindung geöffnet wird.
   Bitte beachten Sie, daß alle diese Optionen und der ssh-pubkey in einer einzigen Zeile stehen müssen!
5. Hinzufügen eines ssh-Befehls in die Nagios-Skripte, so daß jedem Check des Ports 5432 auf dem lokalen Rechner folgender Befehl vorausgeht:
   ```
   ssh -l nagios -L 5432:127.0.0.1:5432 -f sleep 20
   ```

Kapitel 9

Migration bestehender Systeme

In diesem Kapitel betrachten wir die Migration bestehender Systeme. Wie werden dabei zwei Arten der Migration im Zusammenhang mit OpenSSH beleuchten:

- Ersetzen bestehender unsicherer Werkzeuge durch sichere. Das beinhaltet die Ablösung der `r`-Kommandos durch OpenSSH.
- Wechsel von SSHv1- zu SSHv2-Implementierungen

Die Beschreibung des Migrationsprozesses beginnt dabei ganz am Anfang und schreitet schrittweise voran. Was muß beachtet werden? Welche Probleme können auftauchen? Diese Fragen werden in diesem Kapitel beantwortet.

Ein Punkt ist noch offen: Warum sollten die genannten Migrationen überhaupt durchgeführt werden?

- Die `r`-Kommandos verfügen über kein Verschlüsselungsverfahren und sind somit ein gefundenes Fressen für jeden Angreifer.
- SSHv1 ist bereits in die Jahre gekommen, und es existieren bekannte Sicherheitslücken (Abschnitt 5.4). SSHv2 hat im Vergleich zu SSHv1 deutliche Fortschritte gemacht, die sich in Form einer vollständigen Überarbeitung niederschlagen.

9.1 Ersetzen der r-Kommandos

Zunächst werfen wir einen Blick auf das Ersetzen der `r`-Kommandos . `r`-Kommandos waren früher weit verbreitet, doch nach und nach schwinden die Einsatzgebiete. Ihre Nutzung wird mehr und mehr ersetzt durch SSH-Produkte. Auch denjenigen, die sich bis heute nicht von den `r`-Kommandos trennen konn-

ten, ist ein Umstieg anzuraten. Folgende r-Kommandos betrachten wir stellvertretend:

- rcp
- rsh
- Sonderfall rsync

Es existieren sicher noch weitere r-Kommandos, die aufgezählten sollten aber den Großteil der Verwendungen abdecken. rsync unterscheidet sich in zwar in mehrerleich Hinsicht von den r-Kommandos, der Einfacheit halber handeln wir es aber in diesem Zusammenhang mit ab. Historisch gesehen ist rsync kein Relikt aus der r-Zeit und unterstützt auch schon länger das Zusammenspiel mit der Secure Shell. Im Unterschied zu den anderen r-Kommandos kennt rsync auch dasn rsync://-Protokoll.

9.1.1 Ersetzen von rcp und rsh

Das Ersetzen von rcp durch das OpenSSH-Werkzeug scp ist im recht einfach. Die beiden Kommandos rcp und scp sind sich sehr ähnlich und verwenden größtenteils dieselbe Syntax. Eine Datei kopieren Sie unter Verwendung der r-Kommandos mit

```
rcp <Datei> <RemoteHost>:<SpeicherOrt>.
```

Bei der Betrachtung der Syntax von scp stellen Sie fest, daß die Syntax übereinstimmt. Diese Tatsache macht die Umstellung mehr zur einer Fleißaufgabe als zu einem schwierigen Unterfangen. Wichtig beim Ersetzen der r-Kommandos ist deswegen ein strukturiertes Vorgehen, um sicherzustellen, daß Sie alle Verwendungen entdecken und umstellen. Zum Beispiel muß auch an alle automatisierten Skripte gedacht werden.

Ähnlich wie rcp lässt sich auch rsh ersetzen. Das Pendant ist hier der OpenSSH-Client ssh, der eine ähnliche Syntax verwendet. Ein rsh-Befehl zur Anmeldung auf einem Host hat den Aufbau:

```
rsh -l <Benutername> <Host> <Kommando>
```

Die Struktur von des ssh-Befehls für denselben Zweck ist mehr als ähnlich:

```
ssh -l <Benutername> <Host> <Kommando>
```

Dieses bedeutet, daß Sie Ihre rsh-Aufrufe meist 1-zu-1 ersetzen können.

Folgende Punkte geben eine Reihe von Aspekten wieder, die bei der Ablösung von rcp berücksichtigt werden sollten. Sie sind gleichzeitig ein Vorschlag für eine Reihenfolge bei der Migration. Der Vorschlag gilt gleichermaßen auch für die Ablösung von rsh.

- Analyse des aktuellern Zustands

Kapitel 10

Virtuelle Private Netze mit OpenSSH

von Alexander von Gernler

Dieses Kapitel behandelt ein relativ neues Merkmal von OpenSSH: Ab Version 4.3 ist es möglich, direkt mit der SSH virtuelle private Netzwerke aufzubauen.

10.1 Hallo, Welt!

Zunächst einmal für alle, die es gar nicht erwarten können: Hier ist die Anleitung, wie man sein eigenes VPN mit dem Host `server` unter OpenSSH aufsetzt:

```
# ssh -fN -w 0:0 server
# ifconfig tun0 10.0.50.1 pointopoint 10.0.99.1 netmask 255.255.255.252
```

Es ist prinzipiell wirklich so einfach. Was jetzt alles genau dahintersteckt, was die Befehle bewirken, und was man noch alles damit machen kann, wird in diesem Kapitel beschrieben.

10.2 Motivation

In diesem Abschnitt erfolgt eine grundlegende Klärung des Begriffs VPN als allgemeines Muster der Netzwerktechnik sowie eine Beschreibung der Gründe, die für den Einsatz dieser Techik sprechen.

10.2.1 Billige private Netze für alle

Denkt man zurück an die frühen Zeiten des Internets, so war es bei großen Konzernen durchaus üblich, einzelne Standorte zum Zwecke einer einheitlichen

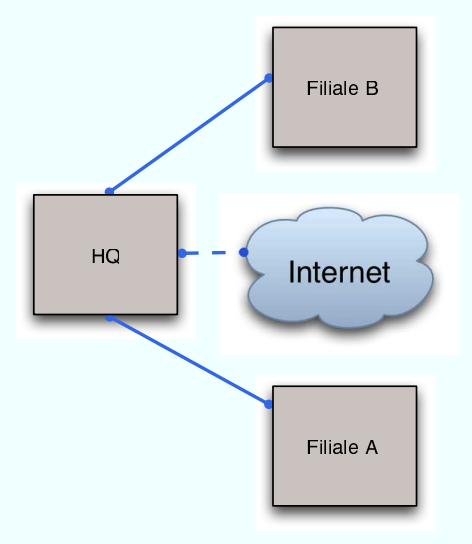

Abbildung 10.1: Traditioneller Intranet-Ansatz: Die Filialen sind mit dem Hauptsitz eines Konzerns über private, eigens gemietete Standleitungen verbunden. Es existiert (wenn überhaupt) ein einzelner definierter Übergang ins Internet.

EDV-Organisation von Warenwirtschaft, Auftragsführung, Buchhaltung und anderen Gegenständen des Geschäftsbetriebs untereinander zu vernetzen. Diese Netze bildeten dann ein firmenweites *Intranet*, das nicht zwingend mit dem echten Internet verbunden sein mußte, auch wenn die eingesetzten Techniken und Protokolle denen des Internets entsprachen. Zur Verbindung der Standorte wurden bei großen Telekommunikationsanbietern (sog. *Carriern*) für viel Geld Standleitungen eingekauft, die tatsächlich von einer Niederlassung der Firma zur anderen durch das Netz des Anbieters gelegt wurden (Abbildung 10.1).

Diese Maßnahme blieb daher einige Zeit nur den zahlungskräftigen Betrieben vorbehalten – kleinere Unternehmen und vor allem Privatleute konnten sich solche dedizierten Leitungen kaum leisten.

Dies änderte sich mit der Einführung des Konzepts virtueller privater Netzwerke. Diese nutzen einen bereits vorhandenen, kostengünstigeren Verbindungsweg (zum Beispiel das Internet), um über diesen einen Tunnel zu betreiben, durch den dann wiederum Netzwerkverkehr der beteiligten Einrichtungen geleitet wird (Abbildung 10.2).

Damit die Technik funktioniert, muß das Netz, das zur Durchleitung benutzt wird, keine Kenntnis von diesen virtuellen Netzen haben. Umgekehrt muß das unterliegende Netz nicht einmal IP sprechen, damit im Tunnel IP eingepackt sein

dem Weg (also derjenige, der noch direkt im eigenen Subnetz erreichbar ist) das Gateway in mehrere oder alle anderen Netze.

Üblicherweise existiert in kleinen Installationen aber nur ein Gateway im Netz, über das der gesamte weitergehende Verkehr geroutet wird. Existieren mehrere Gateways, so entscheidet das Betriebssystem des versendenden Rechners, über welches davon das betreffende Paket verschickt wird.

Meist wird diese Entscheidung anhand einer sog. *routing table* im Kern getroffen. Diese ordnet bestimmten Netzen bestimmte Gateways zu. Existiert für ein Zielnetz keine explizite Zuordnung, wird das Paket über das sog. *default gateway* verschickt. Existiert auch ein solches nicht, so wird das Paket verworfen und eventuell eine entsprechende Benachrichtigung an den Absender zurückgegeben[3]. Eine sehr einfache Routing-Tabelle könnte unter Linux beispielsweise so aussehen:

```
Destination     Gateway         Genmask         Flags  ..  Iface
192.168.100.0   0.0.0.0         255.255.255.0   U      ..  0 eth0
0.0.0.0         192.168.100.1   0.0.0.0         UG     ..  0 eth0
```

Die Tabelle wird ausgewertet, indem die Zieladresse des weiterzuleitenden Pakets inspiziert wird, und derjenige Eintrag greift, der mit der kleinsten Netzmaske greift (sog. *most specific match*). Findet sich gar kein passender Eintrag, so wird die Default-Route verwendet, denn diese greift immer, aber wegen ihrer größtmöglichen Netzmaske, zuletzt.

In unserem Fall bedeutet das: Ist ein Paket für das Netz 192.168.100.0/24 bestimmt, so kann das Paket direkt auf dem Interface eth0 abgeschickt werden, und der Empfänger ist im lokalen Netz zu finden. Für alle anderen Empfänger muß das Gateway 192.168.100.1 benutzt werden. Das Gateway hat nicht nur zufällig eine Adresse im gleichen Subnetz, sondern es muß vom lokalen Rechner direkt erreichbar sein, und diese Forderung erfüllen eben nur Adressen im eigenen Subnetz.

Je komplizierter die Umgebung, desto umfangreicher ist auch die Routing-Tabelle des lokalen Systems. Auf Systemen, wo Routing-Einträge je nach Verfügbarkeit dynamisch wegbrechen und wieder hinzustoßen können, übernimmt in der Regel ein spezieller Routing-Dienst die Veränderung der Einträge in der Tabelle. Solche Dienste sind beispielsweise gated, Quagga, GNU Zebra, OpenBGPD oder OpenOSPFD, aber es gibt noch viele mehr.

10.6 Einrichten eines VPNs mit OpenSSH

Nachdem die notwendige Theorie jetzt behandelt wurde, können wir daran gehen, unser VPN zu entwerfen und umzusetzen. Hierzu besprechen wir nachein-

[3] Dem einen oder anderen kommt bestimmt die Meldung „No route to host" bekannt vor.

ander zwei Beispiele, anhand derer eigene Konfigurationen einfach aufgesetzt werden können.

10.6.1 Bestandsaufnahme

Es ist äußerst hilfreich beim Einrichten einer komplexeren Netzwerkstellung, wenn man sich vorher Notizen anfertigt und die bekannten Daten sowie den gewünschten Endzustand für sich selbst skizziert.

Für unser Beispiel gelte folgendes Szenario, das wir auf Grundlage von Abbildung 10.3 erstellen: Eine Firma besitzt einen Hauptsitz (Netz A) und eine Filiale (Netz B). Beide Standorte haben über ihren örtlichen Provider eine DSL-Flatrate mit fester IP und ohne Trennung nach 24 Stunden. Das ist zwar etwas teurer als eine Flatrate für den Hausgebrauch, aber für eine Firma dennoch sehr leicht erschwinglich, und immer noch weit unter den Kosten einer Standleitung zwischen den zwei Orten.

Als Bestandsaufnahme würde Abbildung 10.3 zusammen mit diesen Informationen schon fast ausreichen, es fehlt lediglich noch die Zusammenfassung der Ziele:

- Der Hauptsitz möchte Verbindungen in das Filialnetz über das VPN leiten, und alle restlichen Verbindungen, die nicht für das lokale Netz bestimmt sind, weiterhin einfach ins Internet routen.
- Für die Filiale soll ebenfalls Routing zum Hauptsitz per VPN erfolgen und der Rest ins Internet.
- Die bereits existierenden Gateway-Rechner, die bisher nur als DSL-Router fungiert haben, sollen jetzt auch VPN-Router werden.
- Der VPN-Tunnel wird vom Hauptsitz aus initiiert.

10.6.2 Rechner bereitmachen

Ein VPN mit OpenSSH benötigt einige Voraussetzungen, die noch nichts mit der eigentlichen Konfiguration zu tun haben. Um später keine bösen Überraschungen zu erleben, gehen wir einfach die folgenden Punkte als Checkliste durch.

10.6.2.1 Sicherstellen, daß SSH aktuell ist

Weil das Tunneling-Merkmal erst ab OpenSSH 4.3 verfügbar ist, sollten wir zuerst prüfen, ob die installierte SSH auf beiden Rechnern auch aktuell genug ist. Nichts ist enttäuschender, als umfangreiche Konfiguration aufzubauen, und dann alles an einer veralteten Softwareversion scheitern zu sehen.

Um die installierte Version herauszufinden, kann entweder auf den Paketmanager des installierten Betriebssystems zurückgegriffen werden, oder man fragt einfach die installierten Programme selber:

```
$ ssh -V
OpenSSH_4.4, OpenSSL 0.9.7j 04 May 2006
$ telnet localhost 22
Connected to localhost.
Escape character is '^]'.
SSH-1.99-OpenSSH_4.4
^]
telnet> quit
Connection closed.
$ telnet vpngw-b 22
Connected to vpngw-b.
Escape character is '^]'.
SSH-1.99-OpenSSH_4.4
^]
telnet> quit
Connection closed.
$
```

10.6.2.2 tun-Unterstützung sicherstellen

Unter OpenBSD sind die tun-Devices immer vorhanden, weil sie per Default in den Kernel einkompiliert sind. OpenBSD-Benutzer können den Rest des Abschnitts also getrost überspringen.

Je nach verwendeter Linux-Distribution kann allerdings der tun-Treiber auch als Modul vorliegen und muß deshalb nachgeladen werden. Sollte einer der späteren Schritte am fehlenden tun-Device scheitern, so ist die Abhilfe hierfür meistens:

```
# modprobe tun
```

Ob der tun-Treiber schon als Modul geladen ist, kann man unter Linux folgendermaßen feststellen:

```
# lsmod | grep '^tun'
tun                    9696  2
```

Dieser Befehl muß natürlich auf beiden Rechnern ausgeführt werden. Ist er für das Funktionieren nötig, so ist es sehr empfehlenswert, ihn in das Framework der Startskripten der jeweiligen Maschine einzutragen. Die Startskripte unterscheiden sich jeweils von Distribution zu Distribution, so daß hier keine allgemeine Lösung präsentiert werden kann.

Kapitel 11

Kochbuch

von Alexander von Gernler

In diesem Kapitel präsentieren wir reale Problemstellungen, die den Autoren tatsächlich so oder zumindest ähnlich schon untergekommen sind. Nach einer detaillierten Beschreibung des Ausgangszustands zeigen wir jeweils eine Lösung des Problems mit OpenSSH und anderen einfach verfügbaren Unix-Bordmitteln.

11.1 Verzicht auf die lästige Passphrase-Eingabe!

Wie dieses Buch umfangreich zeigt, hat die Verwendung der Secure Shell enorme Vorteile gegenüber unverschlüsselten Varianten des Logins, und daher sollte man sie wo immer nur möglich benutzen. Schnell fällt jedoch auf, daß das ständige Eingeben von Paßwort oder Passphrase doch recht umständlich ist.

Um sich durch diese notwendige Sicherheitsmaßnahme nicht von der produktiven Arbeit abhalten zu lassen und trotzdem weiterhin von den Vorteilen der Secure Shell zu profitieren, stellen wir hier ein paar Kochrezepte für den effizienten Einsatz des bereits im Buch angesprochenen `ssh-agent` vor.

Hierbei machen wir uns eine wesentliche Tatsache beim Arbeiten am Computer zunutze: Der Benutzer beginnt seine Arbeit mit dem Starten einer Sitzung (gleich welcher Art: X-Session, login auf der Textkonsole, Benutzung des Befehls `screen`) und authentifiziert sich bei diesem Start. Verläßt der Benutzer den Rechner, so sperrt er die Sitzung (falls er nur eine kurze Pause einlegt), oder er meldet sich gleich komplett ab. In jedem Fall kann zu keiner Zeit ein unbefugter Dritter auf die Sitzung des Benutzers zugreifen, wenn dieser auch nur ein minimales Sicherheitsbewußtsein an den Tag legt.

Wenn der Benutzer nun aber sowieso schon authentifiziert ist, wieso sollte der Rechner ihn dann jedes Mal aufs Neue überprüfen wollen?

Der Gedanke ist also, den Start des Agenten bequem in den Start der Sitzung einzubauen, denn auf diese Weise kann der Benutzer während seiner Arbeit ganz einfach SSH-Verbindungen aufbauen. Es wird also versucht, ein sog. *single sign on*-Verhalten herzustellen.

Je nachdem, welche Art von Sitzung der Benutzer für seine Arbeit bevorzugt, ergeben sich unterschiedliche Möglichkeiten, den Agenten mitwirken zu lassen.

11.1.1 Agent beim Start der X-Session

Einige Distributionen bringen den hier beschriebenen Mechanismus schon mit, deshalb können Sie diesen Abschnitt getrost überspringen, falls dies bei Ihnen der Fall ist.

Der beste Weg, den `ssh-agent` beim Start der X-Session mitzustarten ist, ihn in die X-Skripte des Systems global einzubinden. Auf diese Art und Weise wird jeder Benutzer, der in seinem Heimverzeichnis potenziell aktivierbare SSH-Keys hat, am Anfang nach der Passphrase gefragt.

Der beschriebene Weg geht davon aus, daß die Skripte unter `/etc/X11/` zu finden sind. Je nach Distribution kann der Pfad variieren, das Prinzip bleibt aber gleich. Weil es unter X mehrere Wege zum Start einer Sitzung gibt, gibt es auch mehrere Skripte, die modifiziert werden müssen.

- `/etc/X11/xdm/Xsession` ist das Shellskript, das beim Login über den graphischen Display-Manager `xdm` ausgeführt wird.
- `/etc/X11/xinit/xinitrc` ist das Shellskript, das ausgeführt wird, wenn Sie X von der Konsole mit `startx` starten.
- Wenn Sie einen anderen Display-Manager als `xdm` einsetzen, kann es sein, daß sie auch noch in dessen Konfiguration einen entsprechenden Zusatz für den Sitzungsstart machen müssen.

Wir stellen hier die Zeilen vor, die Sie in alle oben aufgezählten Skripte einfügen sollten, damit der Agent unabhängig von Ihrem Weg, sich anzumelden, gestartet wird. Sollte der `ssh-agent` auf Ihrem System nicht unter `/usr/bin/` zu finden sein, müssen Sie den Pfad noch anpassen.

```
# if we have private ssh keys, start ssh-agent and add the keys
id1=$HOME/.ssh/identity
id2=$HOME/.ssh/id_dsa
id3=$HOME/.ssh/id_rsa
if [ -x /usr/bin/ssh-agent ] && [ -f $id1 -o -f $id2 -o -f $id3 ]
then
```

```
        eval ‘ssh-agent -s‘
        ssh-add < /dev/null
fi
```

Das kurze Skript (das übrigens von der Heimat der OpenSSH, dem Betriebssystem OpenBSD direkt entnommen ist) tut der Reihe nach folgendes:

1. In der `if`-Bedingung wird geprüft, ob sowohl der `ssh-agent` verfügbar ist, als auch mindestens ein potenziell brauchbarer privater Schlüssel existiert. Ist dies nicht der Fall, braucht der Agent nicht gestartet zu werden, und das Skript ist beendet.
2. Falls die Bedingung jedoch erfüllt ist, so wird in der Zeile mit dem Befehl `eval` zunächst der Agent innerhalb der Backticks (‘) gestartet. Der Schalter `-s` verlangt, daß die Ausgabe der Kommandos, die der Agent zur Ausführung empfiehlt, in einem zur Bourne-Shell kompatiblen Format passieren soll. Die Ausgabe sähe etwa so aus, ist aber im obigen Beispiel für den Benutzer nicht sichtbar:
   ```
   SSH_AUTH_SOCK=/tmp/ssh-QcfDA28856/agent.28856;
   export SSH_AUTH_SOCK;
   SSH_AGENT_PID=12584; export SSH_AGENT_PID;
   echo Agent pid 12584;
   ```
   Bei den Befehlen handelt es sich um Umgebungsvariablen, mit deren Hilfe später die `ssh` oder verwandte Programme überhaupt erst zum Agenten finden können. Es ist daher sehr wichtig, daß diese Variablen auch tatsächlich gesetzt werden.
3. Dies passiert dadurch, daß `eval` das kurze Skript, das von `ssh-agent -s` zurückgeliefert wird, in der aktuellen Shell ausführt. Damit sind die Variablen von nun an in diesem Kontext und auch in allen daraus gestarteten Programmen, sprich in dieser Sitzung, verfügbar.
4. Damit ist aber immer noch kein Schlüssel geladen, der Agent läuft also noch leer. Um diesen Zustand zu ändern, wird noch `ssh-add` ausgeführt. Das Programm merkt, daß es kein Terminal zur Verfügung hat, und startet deshalb ein kleines X-Fenster (`ssh-askpass`), das den Benutzer nach der Passphrase fragt.

11.1.2 Agent beim Login auf der Textkonsole

Das oben präsentierte kurze Shellskript ist so vielseitig, daß Sie es ohne weitere Änderungen auch in die `rc`-Datei Ihrer Shell eintragen können. Verwenden Sie keine Bourne-, sondern eine C-Shell-kompatible Shell, werden Sie mindestens noch den Aufruf des Agenten in ein `ssh-agent -c` verwandeln müssen.