Installation, Test und Bewertung von Internet-Sicherheitsmechanismen

Diplomarbeit

von

Jürgen Mayerhofer
und
Christian Rotter

Betreuer: Prof. Dr. Herbert Kopp

Fachhochschule Regensburg
Fachbereich Informatik

12. Mai 1998

Vorwort

Christian Rotter und Jürgen Mayerhofer

Ziel der Diplomarbeit ist es, verschiedene Internet-Sicherheits-Systeme auf Einsatzfähigkeit und Funktionalität zu prüfen. Wir beschränken uns dabei auf frei verfügbare Systeme, da kommerzielle Produkte meist sehr teuer, weniger flexibel und häufig nicht wesentlich sicherer sind. Auch sind die Folgekosten bei kommerziellen Produkten (Wartungsverträge, Updates, Lizenzgebühren, Schulungen) oft von beträchtlicher Höhe.

Im Endeffekt erfordern jedoch alle Sicherheitssysteme ein ständiges „am Ball bleiben“, da ständig neue Sicherheitsprobleme auftauchen, die vorher einfach nicht vorhanden waren und durch die rasante Weiterentwicklung im Hard- und Software-Bereich hervorgerufen werden. Ab einer gewissen Netzwerkgröße sollte für den Sicherheitsbereich ein eigener Administrator als Spezialist vorhanden sein, der seinen Schwerpunkt auch auf diesen Bereich legt. Das beste (und teuerste) Sicherheitssystem ist nämlich nutzlos, wenn es nicht voll ausgenutzt und ständig auf dem aktuellsten Stand gehalten wird. In der heutigen Zeit, in der eine Firma ohne funktionierende EDV in wenigen Tagen nicht mehr fähig ist, den Geschäftsbetrieb weiter aufrecht zu erhalten, hat die Sicherheit ohnehin einen besonderen Stellenwert. Ein Einbruch eines Unbefugten in interne Datenbestände kann hier verheerende Folgen haben.

Der nun schon seit einigen Jahren anhaltende Internet-Boom tut ein übriges, da der Vernetzungsgrad (sowohl intern als auch extern) weiter ansteigen wird und das gefahrbergende Umfeld sich deshalb ständig vergrößert. Durch die steigende Rechenleistung, die zunehmend auch für Privatpersonen erschwinglich wird, ist auch dem „Spieltrieb“ von eigentlich harmlosen Computerfreaks keine Grenze mehr gesetzt. In Systeme einzudringen, die noch vor fünf Jahren als sicher galten, oder zumindest ihre Funktion zu beeinträchtigen, ist nur eine Frage des Wissens, der Rechenleistung und des Zeitaufwandes. Hier muß ein Unternehmen in die eigene Zukunft investieren und sich absichern, so gut es geht. Ein Verzicht auf die Vorzüge des Internets ist zwar eine Lösung, jedoch wird die Zukunft ein weiteres Anwachsen des Internets bringen und damit das Internet auch für den Geschäftsbereich immer wichtiger werden.

Besonderer Dank gilt Herrn Prof. Dr. Herbert Kopp für die Stellung des überaus interessanten Themas dieser Diplomarbeit und für die Unterstützung bei der Durchführung. Seiner unkomplizierten Vorgehensweise bei der Beschaffung von Informationen, Hard- und Software ist es zu verdanken, daß wir immer wieder neu motiviert wurden. Des

weiteren möchten wir uns bei Herrn Hans Buberger für die Anbindung an das Fachhochschulnetz und Betreuung der Routingwege bedanken. Wir bedanken uns bei Herrn Prof. Werling und Herrn Pulina vom Rechenzentrum der Universität Regensburg für die Bereitstellung eines Cisco-Routers und diversen Netzwerkkomponenten.

Weiterhin gilt unser Dank Hubert Feyrer, sowohl für die guten technischen Hilfestellungen und Ratschläge, als auch für die kleinen Tips und belebenden Kommentare, Christoph Metten und Ingo Reckziegel für das Probelesen und Klaus Schmidt für die Bereitstellung einer 3GB Festplatte zum Erstellen von Logfiles sowie bei Herrn Kurt Spörl für die Anbindung an **sein** Netzwerk.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

Jürgen Mayerhofer

Die Diplomarbeit befaßt sich mit der Installation von Sicherheitshilfsmitteln. Es wird bewußt darauf verzichtet, die verschiedenen Konfigurationsmöglichkeiten im „Secured Firewall Subnet“ zu beschreiben, da es hierfür genügend Dokumentation an anderer Stelle gibt (siehe Literaturverzeichnis z.B. [10] und [11]). Stattdessen wird detailliert beschrieben, wie die einzelnen Hilfsprogramme zu installieren, konfigurieren und zu bedienen sind, die in den Firewallsubnetzen eingesetzt werden. Zusätzlich wird großer Wert darauf gelegt, daß die Übertragungen aller Daten möglichst sicher und meist verschlüsselt erfolgt.

Dem Leser soll ermöglicht werden, die Installation der eingesetzten Tools selbst vorzunehmen. Grundkenntnisse in der Administration des Betriebssystemes UNIX sowie die Bedienung von „vi“, „tar“ und ähnlicher Kommandos werden dabei vorausgesetzt.

Die Installation der getesten Software erfolgte größtenteils unter Solaris, teilweise unter Linux. Dies sind in der Regel die Umgebungen, die die wenigsten Probleme bereiten. Jedoch sollte mit ausreichend Know-How über andere UNIX-Derivate die Portierung kaum Probleme bereiten. Angesichts der großen Palette verschiedener UNIX-Systeme kann jedoch nicht immer gewährleistet werden, daß alles funktioniert.

Weiterhin wird ein funktionierender C-Compiler benötigt. Der frei verfügbare GNU-Compiler *gcc* liefert hier hervorragende Dienste. In einem „Firewall Subnet“ hat aber ein Compiler nichts zu suchen. Clevere Hacker haben es bereits wiederholt geschafft, eigene Systembefehle über Sourcecode auf fremden Rechnern zum Laufen zu bringen, indem sie die Quellen auf diesem Computer compilierten. Dies reichte bis hin zum kompletten eigenen Betriebssystem. Deshalb sollte man bei der Wahl der Betriebssysteme berücksichtigen, daß eine Entwicklungsumgebung im gesicherten Netz existiert, man dort compiliert, um die Binaries dann im Firewallnetz einzusetzen.

Viele Scanner und Logfile-Auswertungsprogramme benutzen *perl*. Informationen darüber, die Quellen und wie es installiert wird sind unter http://perl.com/ zu finden. Außerdem wurden häufig *sed* und *awk* verwendet. Kostenlose Versionen sind über jeden Mirror von prep.ai.mit.edu – besser bekannt als GNU – zu finden.

Kapitel 2

Proxy-Systeme

2.1 Was ist ein Proxy-System?

Christian Rotter

Ein Proxy-Server ist im Normalfall ein dedizierter Computer mit spezieller Software, der für eine Vielzahl von anderen Computern eine Verbindung ins Internet herstellt. Dabei gibt es – von der Performance her gesehen – zwei Grundtypen (es gibt auch andere Unterscheidungsmöglichkeiten, dazu später mehr):

- Der eine Typus stellt nur – nach Überprüfung der Rechtmäßigkeit des gewünschten Zielcomputers bzw. Zieldienstes – Zugriffsmethoden zur Verfügung. Diese Proxy-Systeme eignen sich z.B. für Telnet, RLogin und andere Dienste, bei denen die aufgebauten Verbindungen einen „einzigartigen“ und interaktiven Charakter haben.

- Der andere Typus macht zwar (vom Anwender aus gesehen) nichts anderes, jedoch ist dieser Proxy-Server so „intelligent“, bereits übertragene Daten in einem lokalen Cache zwischenzuspeichern. Bei wiederholten Anfragen erfolgt (fast) kein Zugriff auf das Internet, sondern die Daten werden aus diesem Cache gelesen. Dies ist natürlich nicht für alle Dienste sinnvoll, aber für HTTP und FTP ist es ein sehr performanter Ansatz, da die mit diesen Diensten übertragenen Daten häufig sehr groß sind, und bei langsamen Leitungen und wiederholter Übertragung sehr hohe Kosten entstehen können. Ein Problem bei Caching ist immer die Aktualität der Daten, weshalb ein Cache-Proxy-Server meist zuerst den Zeitpunkt der letzten Modifikation der realen Datenquelle überprüft. Man kann deshalb hier nicht von „Offline-Betrieb“ sprechen. Telnet und RLogin sind zum Beispiel keine geeigneten Kandidaten für solche Cache-Proxy-Systeme, da diese Dienste interaktiv arbeiten und keine zwei Sessions gleich ablaufen dürften.

2.1.1 Vorteile

Zusätzlich zu der oben genannten Funktionalität können Proxy-Systeme noch einige andere Vorteile für sich verbuchen:

- Was beide Typen gemeinsam haben, ist die Fähigkeit, Zugriffe über den Proxy auf entfernte Computer sehr effizient mitzuprotokollieren und dabei den Benutzern keine besondere Arbeitsmethoden aufzuzwingen, da die verwendeten Clientprogramme meist weiterhin funktionieren und somit eine langwierige Neuschulung entfällt.

- Weiterhin unterstützen manche Proxy-Systeme IP-Masquerading, was bedeutet, daß die „hinter“ dem Proxy verwendeten IP-Adressen nicht im „echten“ Internet sichtbar werden. Somit kann man theoretisch mit einer einzigen gültigen IP-Adresse sehr vielen Computern Zugriff auf das Internet ermöglichen. Dies trägt dazu bei, die (momentan) bestehende Knappheit von IP-Adressen und Domains zu beheben, bis sich IP Version 6 etabliert hat und solche Probleme der Vergangenheit angehören.

- Auch der Aufbau von VPNs (virtual private networks) ist mit Proxy-Systemen realisierbar (zwar aufwendig, aber ohne die sonst anfallenden Hardwarekosten für hochspezialisierte Netzwerkkomponenten).

- Einen weiterer Vorteil haben Proxy-Systeme in Zusammenhang mit Firewall-Systemen: Es ist nicht erforderlich, jeden Rechner einzeln abzusichern; alle sicherheitsrelevanten Tätigkeiten beziehen sich auf das Proxy-System, das alle Anfragen sammelt und die Schnittstelle zum externen Netzwerk darstellt.

- Den letzten Vorteil kann man auch als Nachteil sehen: Bei manchen Proxy-Systemen (z.B. Socks) sind Rechner *hinter* dem Proxy vom externen Netzwerk aus gar nicht zu erreichen, alle Verbindungen müssen vom internen Netzwerk aus aufgebaut werden.

2.1.2 Nachteile

Diesen Vorteilen stehen natürlich auch Nachteile gegenüber (siehe auch [5], S. 192 ff):

- Die Zugriffsgeschwindigkeit ist aufgrund der zwischengeschalteten Proxy-Software im Normalfall etwas geringer, was jedoch relativ zu sehen ist. So bemerkt man bei interaktiven Diensten meist keine Verzögerung (außer, der Proxy ist unterdimensioniert), und bei cache-fähigen Diensten ist der Zeitgewinn des lokalen Zugriffs im Durchschnitt meist größer als die Wartezeit, bis der Proxy die Anfrage beantworten kann.

- Ein weiteres Problem ist die Einführung eines neuen Dienstes, da es meist eine gewisse Zeit dauert, bis ein Proxy-Modul entwickelt worden ist (dies ist im Normalfall möglich, kann jedoch sehr aufwendig sein). Man hat also bei manchen neuen Diensten keine Möglichkeit, diese sofort über den Proxy laufen zu lassen und allen Benutzern anzubieten, was bei einigen Diensten (z.B. Real Audio oder Videokonferenz-Systemen) sicher ein Nachteil ist.

- Durch die Verwendung einer Vielzahl von Diensten wird die Proxy-Software zunehmend komplexer und eventuell reicht es nicht mehr aus, nur ein einziges Softwarepaket zu installieren und zu pflegen. In diesem Fall ist der Administrator gezwungen, für spezielle Dienste eigene Proxy-Module zu beschaffen und sich in die (möglicherweise vollkommen andere) Bedienungs- und Benutzungsphilosophie einzuarbeiten, was sehr zeitaufwendig sein kann.

- Ist ein Proxy-Mechanismus in einem Client-Programm nicht bereits vorgesehen, kann es sehr kompliziert sein, das Programm zur Zusammenarbeit mit dem Proxy zu bewegen. Sind in so einem Fall keine Sourcecodes verfügbar, muß man meist auf ein anderes Produkt ausweichen. Dieser Fall tritt bei den häufigsten Proxy-Einsatzfeldern (HTTP und FTP) jedoch nicht auf.

- Es gibt einige Dienste (wie z.B. Talk), bei denen keine vorhersagbaren Interaktionen stattfinden und für die ein Proxy-Mechanismus schwer oder garnicht entwickelt werden kann. Solche Dienste muß man entweder komplett deaktivieren oder durch vergleichbare, proxy-fähige Dienste (im Falle von Talk z.B. IRC) ersetzen.

- Ein Proxy erlaubt zwar, die Zugriffe zu protokollieren und eventuell zu unterbinden, jedoch werden vorhandene Schwächen von Diensten (wie etwa bei RLogin oder dem X-Window System) nicht beseitigt, was unter Sicherheitsaspekten betrachtet natürlich ein Problem darstellt. Allerdings ist die Abschottung von Computernetzen selbst durch Firewalls nicht vollkommen sicher, die für diesen Bereich eine erweiterte Funktionalität bieten. Auch bei anderen Diensten ist die Gefahr groß, daß die übertragenen Daten Schäden verursachen. Als Beispiel mögen hier Postscript-Dateien (Postscript ist eine komplette Programmiersprache) und ActiveX von Microsoft dienen. Bei derartigen Daten muß die verarbeitende Logik des Clientprogramms den Anwender vor Schäden bewahren, denn für einen universellen Proxy ist ein komplettes „Verstehen“ und Filtern wohl derzeit nicht realisierbar.

2.1.3 Custom Client Software und Custom User Procedures

Es gibt auch eine Unterscheidung von Proxy-Systemen in Bezug auf die Art und Weise des Zugriffs. Auf der einen Seite gibt es spezielle Client-Programme, die für

Proxy-Betrieb konfiguriert werden können (heute die häufigste und für den Anwender bequemste Methode); auf der anderen Seite gibt es besondere Vorgehensweisen (z.B. muß sich der Anwender mit einer speziellen Kennung und der Angabe eines Zielrechners auf dem Proxy einloggen, und dann seine Anfragen starten). Im zweiten Fall kann es durch unerfahrene oder unsichere Anwender durchaus zu Fehlbedienungen/Fehlfunktionen kommen, außerdem ist diese Vorgehensweise durch die aktuelle Software schon fast verdrängt worden, weshalb in dieser Abhandulung nicht weiter darauf eingegangen wird.

2.1.4 Application-Level und Circuit-Level Proxy-Systeme

Eine weitere gängige Klassifizierung von Proxy-Systemen soll nicht unerwähnt bleiben (siehe auch [5], S. 195 ff):

- Auf der einen Seite gibt es *application-level* Proxy-Systeme, die das zugrundeliegende Protokoll der Anwendung verstehen und die verschiedenen Befehle interpretieren. Als (extremes) Paradebeispiel einer solchen Arbeitsweise kann *sendmail* gelten, das im Endeffekt das *store-and-forward* Protokoll SMTP implementiert. Per Definition ist das ein Proxy-System, da ein dedizierter Rechner (der Mailhost) den Datenversand für eine Vielzahl von Client-Rechnern übernimmt. In gewisser Weise lassen sich auch HTTP-Cache-Proxies in diese Katagorie einteilen.

- Auf der anderen Seite haben *circuit-level* Proxy-Systeme eine andere Arbeitsweise; sie verstehen das Protokoll nicht, sondern bauen lediglich die Verbindung vom Client zum Server auf. Diese Arbeitsweise ist typisch für Multiprotokoll-Proxies wie z.B. *Socks*, hier werden lediglich die Quell- und Zieladressen des übergeordneten Protokolls verwendet, um die Verbindung aufbauen zu können. Allerdings ist der Übergang zum *application-level* Proxy-System fließend, da *Socks* beim FTP-Protokoll in den passiven Modus schalten muß, um den Datentransfer abzuwickeln, was natürlich bereits einen Eingriff in das FTP-Protokoll darstellt.

2.1.5 Ausblick

Was bei all diesen Unterscheidungen jedoch wirklich zählt, sind die Resultate. Ein Proxy muß zuverlässig funktionieren und möglichst transparent arbeiten. Der Administrator des Proxies muß entweder die bestehenden Programme umkonfigurieren oder neue Programme installieren. Dieser Arbeitsaufwand ist nicht zu vermeiden. Betrachtet man die Vorteile, die ein Proxy bietet, so ist diese Arbeitszeit sicher eine gute Alternative zu einem Datenverlust durch einen Einbruch in das interne Netzwerk und die dadurch entstehenden Kosten. Jedes Proxy-System kann seinen Teil zur Sicherheit der Daten beitragen; ganz egal, wie es funktioniert und wie man es klassifiziert.

Das weitere Vordringen des Internets in neue Bereiche wird ein weiteres Anwachsen der weltweiten Netzwerke zur Folge haben. Proxy-Systeme sind ein Weg, allzu starken „Wildwuchs“ zu vermeiden und zusätzlich noch Zeit und Geld zu sparen.

2.2 Socks

Christian Rotter

Quelle:

http://www.socks.nec.com/

Socks ist ein Proxy-Server, der mittels angepaßter Client-Programme einen zentralen Übergangspunkt vom internen Netzwerk zum externen Netzwerk (meist dem Internet) darstellt. Die Funktionen reichen von Filterung und Protokollierung bis hin zu IP-Masquerading.Der Test wurde mit einer älteren Socks-Version begonnen, als jedoch die neue Version 5 erschien, wurde dieser der Vorzug gegeben, da sie wesentlich leistungsfähiger ist.

2.2.1 Funktionsbeschreibung

In seiner ursprünglichen Form war Socks ein Proxy für TCP/IP-basierte Dienste, der angepaßte Client-Programme benötigte. Auf dem Proxy-Server wird der Socks-Daemon gestartet, der Anfragen von Client-Programmen nach einer Gültigkeitsprüfung an den Zielhost weiterleitet. Diese Gültigkeitsprüfung ist durch Konfigurationsdateien beeinflußbar. Socks vollzieht außerdem automatisches IP-Masquerading, da alle Pakete scheinbar vom Socks-Proxy kommen. Eine Anbindung von vielen Computern ans Internet mit einer einzigen gültigen IP-Adresse wird somit möglich. Die aktuelle Version (socks5-v1.0r1) unterstützt endlich auch UDP/IP-Proxying und hat eine eigene DNS-Verwaltung, der früher erforderliche DNS-Server für Rechner *hinter* dem Proxy kann somit entfallen. Computer, die *hinter* einem Socks-Proxy liegen, sind auf direktem Weg von externen Netzwerken aus nicht erreichbar (außer, der Proxy ist sozusagen redundant und es gibt eine Netzwerkverbindung, die ihn umgeht), sämtliche Anfragen ans externe Netzwerk (meist das Internet) müssen deshalb von Computern *hinter* dem Proxy initiiert werden. Ein Nachteil ist, daß nicht alle Client-Programme mit Socks zusammenarbeiten, jedoch sollten sich für alle gängigen Dienste entsprechende Client-Programme finden lassen.

Socks ist momentan auf Solaris, SunOS, OSF/1, IRIX und Linux lauffähig, eine Portierung auf andere Unix-Derivate sollte möglich sein.

2.2.2 Installation und Konfiguration von Socks

Socks verwendet den GNU-*autoconf*-Machanismus, die Vorgehensweise bei der Übersetzung des Pakets ist in Kapitel 2.3.2 genauer beschrieben.

2.2.2.1 Betriebsarten

Da der Daemon *socks5* verschiedene Betriebsarten und Authentifizierungsmethoden kennt, kann man beim Aufruf von *configure* sehr viele Optionen angeben. Eine Übersicht aller möglichen Optionen erhält man durch die Angabe von `--help` in der Kommandozeile oder in der umfangreichen Socks-Dokumentation (siehe [6]).

Die Betriebsarten seien hier kurz erläutert, da nicht alle Möglichkeiten sinnvoll bzw. auf jedem System realisierbar sind:

- inetd:

  Bei dieser Betriebsart wird *socks5* bei Bedarf über den inetd gestartet, was relativ lange dauert und für Einsatzzwecke geeeignet ist, bei denen der Proxy nur selten benutzt wird.

- standalone:

  Diese Variante ist das Gegenteil von *inetd*, da hier der Proxy immer gestartet wird und auf Verbindungen wartet. Problematisch ist dabei, daß für jede neue Verbindung mittels *fork()* eine neue Serverinstanz erzeugt werden muß, was zeitaufwendig ist.

- pre-forking:

  Diese auch schon etwas ältere Betriebsart 'behebt' die Probleme der *standalone*-Version, indem Serverprozesse im voraus erzeugt werden. Da man vorher in etwa wissen muß, wie stark die Auslastung der Proxies sein wird, ist auch diese Methode nicht mehr ganz zeitgemäß. Man gibt die Anzahl der Serverprozesse an, die gestartet werden. Übersteigt der Bedarf dann diese Anzahl, müssen neue Prozesse erzeugt werden, was zeitaufwendig ist, ist der Bedarf geringer, wird kostbares RAM verschwendet, da dann Prozesse nutzlos im Speicher gehalten werden.

- threaded:

  Diese modernere Betriebsart behebt alle obigen Probleme, da sie intern mit Multithreading arbeitet. Zudem ist sie wesentlich performanter als die *pre-forking*-Variante, allerdings muß dafür Unterstützung im Betriebssystem vorhanden sein (POSIX-Threads).

Die optimale Betriebsart ist sehr stark von den lokalen Gegebenheiten abhängig, außerdem haben fast alle Betriebsarten Beschränkungen, so ist z.B. die Anzahl von Child-Prozessen (*standalone* und *pre-forking* Betriebsart) genauso begrenzt wie die Anzahl von Threads bei der *threaded* Betriebsart. Das sollte man bedenken, wenn man bei hoher Auslastung des Proxies Probleme bekommt.

2.2.2.2 Konfigurations-Dateien

Der socks5 Server-Daemon liest seine Konfiguration aus der Datei *socks5.conf* (das Verzeichnis ist konfigurierbar).

Alle Konfigurationsanweisungen akzeptieren Parameter, deren Schreibweise und Bedeutung kurz erklärt werden:

Parameter	möglich Werte
source-host dest-host	**IP/Netzmaske** **-** steht für alle Rechner **a.** Klasse A-Netzwerk a **a.b.** Klasse B-Netzwerk a.b **a.b.c.** Klasse C-Netzwerk a.b.c **.a.b** Domain .a.b, z.B. .kde.org **a.b.c** Host a.b.c, z.B. keks.franken.de
source-port dest-port	Servicename, z.B. **ftp** Portnummer, z.B. **80** **-** steht für alle Ports **[a,b]** Ports von a bis b **(a,b)** Ports von a+1 bis b-1 **(a,b]** Ports von a+1 bis b
auth-methods	**n** keine Authentifizierung **u** Benutzername/Kennwort **k** Kerberos 5 **-** steht für alle Möglichkeiten
interface-address	**IP-Adresse** oder **Name** des Interfaces
proxy-list	**Host:Port**, getrennt durch Komma
cmd	**c** *connect* **b** *bind* **u** UDP **p** *ping* **t** *traceroute* **-** steht für alle Befehle
user-list	Benutzernamen, getrennt durch Komma

Es gibt sechs verschiedene Konfigurationsanweisungen, die mehrfach verwendet werden können:

- Verbotene Hosts

  Von verbotenen Rechners werden keine Verbindungen akzeptiert.

  ban <**source-host**> <**source-port**>

- Authentifizierung

Diese Anweisung gibt an, auf welche Art und Weise eine entsprechende Verbindung authentifiziert wird.

auth <**source-host**> <**source-port**> <**auth-methods**>

- Routing

  Für Socks-Hosts mit mehreren Netzwerkanschlüssen kann festgelegt werden, über welches Netzwerkinterface externe Rechner kontaktiert werden.

  route <**dest-host**> <**dest-port**> <**interface-address**>

- Socks-Proxies

  Werden mehrere Socks-Proxies kaskadiert, kann hier der zuständige Proxy für bestimmte Hosts angegeben werden. Interessant ist diese Funktion bei großen Netzwerken mit mehreren Firewall-Ebenen, die jeweile mit einem Socks-Proxy traversiert werden. Auch der Aufbau von VPNs (virtual private networks) wird damit möglich.

  proxy-type <**dest-host**> <**dest-port**> <**proxy-list**>

- Variablen

  Mit solchen Einträgen kann das Verhalten von Socks beeinflußt werden. Mögliche Variablen und ihre Funktion sind in der Man-Page von *socks5* aufgelistet.

  set variable value

- Zugriffsberechtigung

  permit <**auth-methods**> <**cmd**> <**src-host**> <**dest-host**> <**src-port**> <**dest-port**> **[user-list]**

  deny <**auth-methods**> <**cmd**> <**src-host**> <**dest-host**> <**src-port**> <**dest-port**> **[user-list]**

Socks5-Clientprogramme lesen die zugehörige Konfigrationsdatei *libsocks5.conf*. In dieser Datei stehen nur Proxy-Einträge, anhand derer die Client-Programme entscheiden, ob sie eine Verbindung direkt aufbauen oder einen Socks-Proxy verwenden. Die Schreibweise der Konfigurationszeilen ist wie oben beschrieben.

2.2.3 Client-Programme für Socks

Da *Socks* ein Proxy ist, der *Custom Client Software* erfordert, muß auch eine Auswahl der zu verwendenden Client-Programme unter diesen Bedingungen erfolgen. Die aktuelle *Socks*-Version ist in dieser Beziehung sehr flexibel, so daß einem Einsatz meist nichts im Wege steht.

Im Socks-Paket sind bereits einige Client-Programme enthalten:

- rping
- rtraceroute
- rftp
- rfinger
- rwhois
- rarchie (nutzt das neue UDP/IP-Proxying von *Socks5*)
- rtelnet

Alle Client-Programme sind größtenteils kompatibel zu den mitgelieferten Programmen des Betriebssystems. Sie funktionieren jedoch zusätzlich durch den Socks-Proxy. Ist kein Proxy verfügbar, greifen diese Programme einfach auf die normalen Versionen der Funktionen zurück (siehe unten) und arbeiten z.B. im internen Netzwerk ohne Probleme mit anderen Hosts zusammen, auch wenn dort kein Socks installiert ist. Es ist deshalb vorgesehen, die Systemprogramme durch die Socks-Versionen zu ersetzen, dies erfolgt am besten durch Umbenennung der Systemprogramme und anschließendes Kopieren der Socks-Versionen in die selben Verzeichnisse. Damit erfolgt die Umstellung für die Benutzer transparent. Für andere Client-Programme gibt es mehrere Möglichkeiten, diese zusammen mit Socks zu verwenden:

- Umkonfigurierung von socks-tauglichen Programmen
- Einbettung in *runsocks*
- Recompilierung
- Austausch

2.2.3.1 Umkonfigurierung von socks-tauglichen Programmen

Sind verwendete Client-Programme bereits für *Socks* vorbereitet, ist nur eine Änderung von Programmoptionen erforderlich. Das wohl bekannteste Programm dieser Kategorie ist wohl der *Netscape Navigator*, bei dem in *Options-> Network Preferences-> Proxies-> Manual Proxy Konfiguration* nur Name und Port des Socks-Proxy eingetragen werden müssen, um die Zusammenarbeit sicherzustellen. Andere Programme verfügen ebenfalls über eingebaute Socks-Unterstützung, eine Anfrage bei AltaVista ergibt meist eine brauchbare Sammlung von Programmen im Internet.

Name	Funktion
connect()	Initialisierung einer Socket-Connection
getsockname()	Rückgabe des vergebenen Namens für einen Socket-Descriptor
bind()	Vergabe eines Namens für einen Socket-Descriptor
accept()	Akzeptierung einer Socket-Connection
listen()	Warten auf eine Socket-Connection
select()	synchrones Input/Output-Multiplexing

Tabelle 2.1: Socks - Funktionsaufrufe

2.2.3.2 Einbettung in runsocks

Runsocks ist ein Shell-Skript, das der Socks5-Distribution beiliegt und während der Compilierung an das jeweilige Betriebssystem angepaßt wird. Durch einen kleinen Trick ist es möglich, viele Programme ohne Änderungen durch den Socks-Proxy hindurch zu benutzen. Dies wird durch eine Veränderung der Zugriffsreihenfolge auf *shared libraries* erreicht, indem die Library von Socks vor die Systemlibrary *libc* gestellt wird und somit Funktionen, die normalerweise in der *libc* gefunden und ausgeführt werden, mit den Socks-Äquivalenten überladen werden. Dieses eigentlich primitive Verfahren funktioniert in der Praxis zufriedenstellend, allerdings ist ein solcher Eingriff in die Systemabläufe schon etwas gewagt, weshalb man besser vorhandene Programme durch Recompilierung oder durch Austausch gegen *socks*-fähige Programme anpaßt.

2.2.3.3 Recompilierung

Im Normalfall ist die Anpassung sehr einfach, solange man den Sourcecode des Clients besitzt, denn es reicht meist aus, wenige (momentan sechs) Funktionsaufrufe durch eine Präprozessoranweisung umzubenennen und die entstehenden Objekt-Dateien mit der Socks-Library zu linken, siehe [6][How_to_SOCKSify]. Diese Aufrufe sind in Tabelle 2.1 beschrieben.

Die Umbenennung auf die Socks-Implementationen dieser Funktionen (deren Namen fast gleich sind, es wurde einfach ein 'R' vorangestellt) erfolgt normalerweise im Makefile, dort werden in den Compilerflags (CFLAGS) einfach die entsprechenden Defines eingetragen:

```
CFLAGS=  -Dconnect=Rconnect \
         -Dgetsockname=Rgetsockname \
         -Dbind=Rbind \
         -Daccept=Raccept \
         -Dlisten=Rlisten \
         -Dselect=Rselect
```

Für das erforderliche Linken mit der Socks-Library werden die Linkerflags (LFLAGS)

angepaßt:

LFLAGS= -L<verzeichnispfad der socks-library> -lsocks

Mit diesen beiden Änderungen läßt sich bereits ein Großteil der frei verfügbaren Client-Programme mit dem Socks-Proxy verwenden. Wenn Probleme auftreten, ist meist das Client-Programm unsauber programmiert. In manchen Fällen kann man durch „Umstrukturierung“ des Sourcecodes eventuell doch noch ein proxyfähiges Programm erhalten, siehe [6][What_SOCKS_expects].

2.2.3.4 Austausch

Sollten alle obigen Methoden nicht funktionieren, ist man entweder gezwungen, ein anderes Proxy-System zu wählen oder die Client-Programme auszuwechseln. Das Auswechseln von Client-Programmen hört sich zwar schlimm an, aber im Internet ist meist ein funktional ähnliches Programm zu finden; man muß nur danach suchen.

2.2.4 Erfahrungsbericht

2.2.4.1 Testumgebung und Testlauf

Der Aufbau der Testumgebung ist in Kapitel A.1 beschrieben. Zerberus wurde als Socks-Proxy konfiguriert, der Socks-Daemon wurde bereits beim Systemstart aktiviert, die Vorgehensweise ist in Kapitel A.3 beschrieben. Als Client diente Mozart, ein Linux-PC, auf dem die erforderlichen Client-Programme installiert waren. Das ganze Socks-Proxy-System wurde über mehrere Monate hinweg benutzt, wobei auf Client-Seite auch der Netscape Navigator eingesetzt wurde, der Socks direkt unterstützt.

2.2.4.2 Bewertung

Socks ist in der aktuellen Version 5 als sehr brauchbar einzustufen. Rechner hinter dem Proxy sind von externen Angriffen sicher, allerdings auch für reguläre Benutzer nicht erreichbar (außer bei VPN-Konfigurationen). Socks läuft sehr stabil, ist ein gut gepflegtes Paket und damit neben Squid (2.3) ein Highlight im Freeware-Bereich. Das einzige Problem ist die relativ komplizierte Konfiguration beim Einsatz in großen Netzwerken, aber dort sind kommerzielle Produkte auch nicht mehr intuitiv zu bedienen. Socks kann nur empfohlen werden.

2.3 Squid

Christian Rotter

Quelle:
http://squid.nlanr.net/Squid/

Squid, ein Beispiel für einen Proxy-Server mit Cache-Mechanismus, ist aus der *cached* Software des Harvest Forschungsprojektes hervorgegangen. Informationen über Harvest sind auf folgender Webseite zu finden:

http://harvest.cs.colorado.edu/

Proxy-Systeme mit Cache-Mechanismen dienen meist der Performancesteigerung von Datentransfers, die bei langsamer Internet-Anbindung sehr groß sein kann, wenn Daten wiederholt angefordert werden.

2.3.1 Funktionsbeschreibung

Squid, der *Internet Object Cache*, ist jedoch mehr als ein einfacher Cache-Proxy. Durch die Unterstützung von *Neighbor-Caches*, die entweder als *parent* oder *sibling* verwendet werden, kann man einen Cache-Verbund aufbauen, der einen hohen Prozentsatz von Anfragen beantworten kann, ohne langwierige Internet-Zugriffe zu erfordern. Dieses Verfahren verwenden zum Beispiel einige der großen Internet-Provider (wie ECRC in München), um den Datentransfer wenn möglich auf der eigenen Netzwerkstruktur abzuwickeln und damit Kosten für externe Zugriffe zu umgehen.

Die Arbeitsweise von Squid ist einfach:

- Eine Anfrage eines Clients wird zuerst auf die Zugriffsberechtigung untersucht, so kann man z.B. gewissen Rechnern einen Zugriff auf den Proxy ganz verbieten (siehe Konfiguration).
- Es wird versucht, die gewünschten Daten im lokalen Cache zu finden. Sollte dies fehlschlagen, werden andere Proxies im Cache-Verbund (sofern vorhanden), abgefragt.
- Sind die Daten vorhanden, wird ihre Aktualität geprüft. Sind die Daten veraltet, werden neue in den Cache geholt und an den Client weitergegeben. Bei aktuellen Daten wird das Neuladen übersprungen.
- Sind die Daten nicht vorhanden, werden sie in den Cache geladen und an den Client weitergereicht. Wie lange Daten im Cache verbleiben, ist abhängig von der Konfiguration (siehe dort).

Eine Beschreibung der internen Abläufe wurde im Sinne der Übersichtlichkeit weggelassen.

2.3.2 Squid-Installation

Durch die Verwendung des *autoconf* -Paketes von GNU ist die Installation von Squid sehr einfach. Im Archiv ist das Shellscript *configure* enthalten, mit dem das ansonsten komplizierte Compilieren von Sourcecode auf einer anderen Unix-Plattform als der Entwicklungsplattform zum Kinderspiel wird:

- Zuerst wird das Archiv entpackt, was mit der GNU-Version von *tar* sehr einfach geht:

  Prompt> tar xzvf squid-1.1.11-src.tar.gz

  alternativ (d.h. ohne den komfortablen GNU-tar):

  Prompt> gzip -d squid-1.1.11-src.tar.gz | tar xvf -

Ist auch *gzip* nicht vorhanden, sollte man ein Archiv mit der Endung **.Z** suchen,welches dann mit *uncompress* oder *zcat* so aufbereitet werden kann, daß der normale *tar* es entpackt.

- Danach wird der *autoconf*-Mechanismus angestoßen:

  Prompt> ./configure –prefix=/usr/local/squid

  prefix ist hierbei das Verzeichnis, in dem die Unterverzeichnisse für die Dateien des Pakets angelegt werden (wie *etc* für Konfigurationsdateien oder *bin* für Binaries). Dies ist bei einem späteren Programmupdate (oder bei Entfernung des Pakets) sehr praktisch, da in einem Verzeichnis alle zugehörigen Dateien gesammelt sind und nicht in der ganzen Dateihierarchie verteilt werden.

- *configure* prüft nun das System auf vorhandene Include-Dateien, Libraries und eventuelle Besonderheiten, die zu beachten sind. Mit diesen Informationen wird dann ein plattformspezifisches *Makefile* generiert.

- Durch die Eingabe von:

  Prompt> make all

  wird dann das Paket übersetzt.

- Anschließend kann mit:

  Prompt> make install

  die Installation in das oben angegebene Verzeichnis erfolgen.

Nun muß noch die Konfigurationsdatei an die lokalen Gegebenheiten angepaßt werden, und danach kann *squid* gestartet werden.

2.3.3 Squid-Konfiguration

Squid wird mittels einer normalen Textdatei konfiguriert (im Normalfall befindet sich diese Datei im Installationsverzeichnis und heißt `<prefix>`*/etc/squid.conf*), die logisch in mehrere Teilbereiche unterteilt werden kann. Bei der Compilierung von Squid wird eine Default-Konfiguration erzeugt, die mit minimalen Änderungen bereits einen ersten Testlauf ermöglicht. Für die Ausnutzung der vollen Leistung von Squid sollte jedoch eine individuelle Anpassung erfolgen.

Alle Zeilen von *squid.conf*, die mit # beginnen, sind Kommentare bzw. nicht aktivierte Optionen.

Als Minimalkonfiguration genügen die in Schnellstart beschriebenen Einstellungen. Für die genaue Syntax von Optionen sei ebenfalls auf *squid.conf* verwiesen, die folgenden Kapitel geben nur einen groben Überblick über die Möglichkeiten von Squid.

2.3.3.1 Schnellstart

Für einen ersten Testlauf sind nur einige Einstellungen vorzunehmen, die jedoch in der Konfigurationsdatei etwas verstreut sind. Die Optionen werden hier kurz erläutert:

- cache_host

  Dieser Eintrag ist erforderlich, wenn der lokale Proxy mit einem *Neighbor-Cache* verbunden werden soll. Die nötigen Angaben erhält man normalerweise vom Administrator des *Neighbor-Cache*, ein unbefugtes „Anzapfen“ eines anderen Squid-Proxies ist Bandbreitendiebstahl und gehört nicht zum guten Ton im Internet. *Neighbor-Caches* werden später genauer erläutert.

- cache_mem

  Dieser Parameter gibt an, wieviel physikalisches RAM zum Caching verwendet werden soll. *Squid* belegt in der Regel jedoch die zwei- bis vierfache Speichermenge, deshalb sollte der Wert nicht allzu großzügig bemessen sein. Hier sollte man bedenken, daß es für die Performance äußerst ungünstig ist, wenn *Squid* und seine Unterprozesse ausgelagert werden, da Festplattenzugriffe wesentlich langsamer als Hauptspeicherzugriffe sind. Andererseits sollte man aber bedenken, daß *Squid* WWW-Objekte zuerst im RAM puffert, um sie nach Bedarf dann auf die Festplatte auszulagern. Die optimale Einstellung der Hauptsspeichergröße hängt also stark von lokalen Gegebenheiten ab.

- cache_swap

  Die maximale Größe des permanenten Cache-Bereiches auf der Festplatte wird hier angegeben. Man muß jedoch in Betracht ziehen, daß *Squid* beim ersten Aufruf seine komplette Cache-Struktur in Form einer Verzeichnishierarchie bereits auf der Platte abbildet, was den verfügbaren Plattenplatz beschränkt. Dieser Wert

acl	Funktion
acl manager proto cache_object	Standard-*acl* für Management-Dienste
acl localhost src \ 127.0.0.1/255.255.255.255	Definition von *localhost* (Unix-Konvention)
acl all src 0.0.0.0/0.0.0.0	universelle *acl* (trifft für jede Adresse zu)
acl allowed_hosts src \ 194.95.109.176/255.255.255.240	*acl* für Test-Subnetz
http_access deny manager all	kein Remote-Management möglich
http_access allow allowed_hosts	Cache-Zugriff für lokales Test-Subnetz
http_access deny all	kein Cache-Zugriff für den Rest der Welt
icp_access allow allowed_hosts	Anwort auf ICP (Requests von *Neighbors*)
icp_access deny all	keine Antwort an den Rest der Welt

Tabelle 2.2: Squid - access control list

sollte deshalb ca. 10% kleiner als der real verfügbare freie Festplattenspeicher sein, den man verwenden will – die Prozentangabe kann jedoch mit der verwendeten Inode-Dichte variieren. *Squid* reagiert etwas empfindlich, wenn der verfügbare Platz kleiner ist als der zugeteilte. Programmabbrüche können die Folge sein.

- acl, http_access, icp_access

  Unter *acl* versteht man *access control lists*, also Filter, die den Zugriff auf den Proxy regeln.

  Der wichtigste Eintrag ist wohl *allowed_hosts*, der auf die Netzwerkadresse der Domain verweisen sollte, für die Squid cachen soll. Nach der Adressangabe (im Format www.xxx.yyy.zzz), folgt ein Schrägstrich und die verwendete Netzwerkmaske (bei einem Class C Netz also 255.255.255.0). Für den verwendeten Testaufbau (siehe Kapitel A.1) wurden die in Tabelle 2.2 beschriebene *acl* verwendet.

- cache_mgr

  Hier trägt man die E-Mail-Adresse des Betreuers ein, damit bei Bedarf ein Ansprechpartner vorhanden ist.

- cache_effective_user

  Dieser Eintrag ist erforderlich, wenn *squid* vom Administrator *root* gestartet wird (z.B. beim Booten). Hier sollte entweder ein sicherer Benutzer und eine sichere Gruppe eingetragen werden, die im System nur sehr beschränkte Rechte haben (wie *nobody/nogroup*), oder für *squid* wird ein neuer Benutzer und eine neue Gruppe angelegt. Nach dem Starten von *squid* ändert der Prozeß dann seine eigenen Zugriffsrechte, womit eventuelle Sicherheitsprobleme vermieden

werden, die auftreten könnten, wenns*quid* als root-Prozeß läuft. Zu beachten ist hierbei, daß das Verzeichnis für die Cache-Daten sowie die Installationsverzeichnisse ebenfalls dem neuen Benutzer gehören müssen, sonst gibt das Programm etwas irreführende Fehlermeldungen aus, die zwar bei anderen Betriebssystemen gang und gäbe sind, einen Unix-Benutzer aber eher verwirren dürften.

- visible_hostname

  Unter diesem Namen macht sich der Proxy bekannt, dieser Name ist aber nicht frei wählbar, sondern sollte mit dem DNS-Eintrag des Computers übereinstimmen.

Mit diesen Einträgen kann Squid testweise gestartet werden. Dazu ruft man

Prompt> <***prefix*** >**/bin/squid**

auf. Squid beginnt daraufhin, seine Cache-Verzeichnisse anzulegen, was einige Zeit dauert. Anschließend kann man den Cache-Proxy in einem WWW-Client (z.B. *Netscape Navigator*) eintragen und testen.

Die aktuelle Konfiguration stellt das Minimun dar, für effiziente Ausnutzung sollte Squid individuell angepaßt werden.

In den folgenden Kapiteln können die verwendeten Optionen erneut auftauchen, da sich der Aufbau an der automatisch generierten Squid-Konfigurationsdatei orientiert.

2.3.3.2 Allgemeine Einstellungen

- http_port

  Hiermit wird der von Squid verwendete Port für eingehende HTTP-Requests konfiguriert. Dieser Wert ist auf 3128 voreingestellt, die Einstellung des *HTTP Accelerators* ist Port 80 und kann durch die Kommandozeilenoption -*a* `<port>` beim Starten von Squid eingestellt werden.

- icp_port

  Dieser Port wird zur Kommunikation mit *Neighbor-Caches* verwendet. Die Standard-Einstellung ist 3130. Wird Squid unabhängig von einem Cache-Verbund betrieben, kann mit einer Portangabe von 0 ein Standalone-Betrieb erzwungen werden.

- mcast_groups

  Multicast-Groups sind eigene Netzwerkbereiche, die in einem großen Cache-Verbund verwendet werden. Dies wurde im Rahmen der Diplomarbeit jedoch nicht getestet.

Die folgenden vier Optionen sind für Proxy-Systeme mit mehreren Netzwerkanschlüssen von Bedeutung:

- tcp_incoming_address

  Die IP-Adresse des Netzwerkinterfaces für eingehende Anfragen von Clients und Neighbor-Caches.

- tcp_outgoing_address

  Der Netzwerkanschluß für die Verbindung zu WWW-Servern und anderen Neighbor-Caches.

- udp_incoming_address

  Die Adresse, über die Steuerinformationen von Neighbor-Caches empfangen werden.

- udp_outgoing_address

  Das Interface für Anfragen an Neighbor-Caches. Dieser Wert muß ein anderer sein als *udp_incoming_address*, da beide den selben Port verwenden.

Sind diese vier Optionen nicht angegeben, werden sie bei Bedarf automatisch zugewiesen. Eine Festlegung kann jedoch den Datenverkehr auf vorhersagbare Netzwerkverbindungen verlagern, was für große Netzwerke zur Lasttrennung nützlich ist.

2.3.3.3 Festlegung von Neighbor-Caches

- cache_host

  Dieser Eintrag ist erforderlich, wenn der lokale Proxy mit einem *Neighbor-Cache* verbunden werden soll. Die nötigen Angaben erhält man normalerweise vom Administrator des *Neighbor-Cache*, ein unbefugtes „Anzapfen“ eines anderen Squid-Proxies ist Bandbreitendiebstahl und gehört nicht zum guten Ton im Internet.

- cache_host_domain

  Mit dieser Option kann man den Bereich (Domain) einschränken, für den ein *Neighbor-Cache* abgefragt werden soll.

- neighbor_type_domain

  Man kann die Rangordnung von *Neighbor-Caches* auch von der Lage des betroffenen Objekts abhängig machen, die dann als Domain eingetragen wird. So kann derselbe Cache für eine Domain *parent* und für eine andere *sibling* sein.

- inside_firewall

  Man kann alle eigenen Domains, die hinter einer gemeinsamen Firewall liegen, hier angeben, um den Algorithmus zur Objektanforderung zu beeinflussen.

- local_domain

  Alle hier angebenen Domains werden immer direkt nach Objekten abgefragt, ohne den Umweg über *Neighbor-Caches* zu gehen.

- local_ip

  Funktioniert wie local_domain, nur werden hier IP-Adressen benutzt.

- firewall_ip

  Funktioniert wie inside_firewall, nur werden hier IP-Adressen benutzt.

- single_parent_bypass

  Ist als Optimierung diese Option aktiviert, wird beim Nachfragen von Objekten im Falle eines einzigen möglichen *parents* der übliche Ablauf zur Objektfindung umgangen und direkt dieser *parent* befragt. Dabei spart man ein paar Pakete, die sonst über das Netzwerk gehen würden. Allerdings kann es auch sein, daß der betreffende *parent* nicht aktiv ist, was zu einer Fehlermeldung führt. Jeder sollte selbst wissen, was ihm lieber ist.

- source_ping

  Diese Option dient dazu, Cache-Netzwerke ohne manuelle Aktionen aufzubauen. Das Problem dabei ist, daß die Automatik relativ viel Netzlast erzeugt und nicht immer optimal arbeitet.

- neighbor_timeout (seconds)

  Nach dem Ablauf dieser Zeitspanne wird eine von Neighbors nicht beantwortete Nachfrage von der normalen Datenquelle bezogen.

- hierarchy_stoplist

  Hier angegebene URL-Textmuster werden vom lokalen Cache bearbeitet und nicht an Neighbors weitergegeben. Ein Anwendungsfall sind *cgi*-Links, bei denen das Caching unsinnig wäre.

- cache_stoplist

  URL-Textmuster hinter dieser Option bewirken, daß das betreffende Objekt nicht im Cache verbleibt, sondern sofort wieder entfernt wird. Auch hier sind *cgi*-Programme der erste Kandidat.

- cache_stoplist_pattern

  Wie cache_stoplist, nur daß hier *regular expressions* unterstützt werden.

- cache_stoplist_pattern/i

  Wie cache_stoplist_pattern, nur ist Groß-/Kleinschreibung hier nicht mehr relevant.

2.3.3.4 Cache-Dimensionierung

- cache_mem (megabytes)

  Dieser Parameter gibt an, wieviel physikalisches RAM zum Caching verwendet werden soll. *Squid* belegt in der Regel jedoch die zwei- bis vierfache Speichermenge, deshalb sollte der Wert nicht allzu großzügig bemessen werden. Hier sollte man bedenken, daß es für die Performance äußerst ungünstig ist, wenn *Squid* und seine Unterprozesse ausgelagert werden, da Festplattenzugriffe wesentlich langsamer als Hauptspeicherzugriffe sind. Andererseits sollte man aber bedenken, daß *Squid* WWW-Objekte zuerst im RAM puffert, um sie nach Bedarf dann auf die Festplatte auszulagern. Die optimale Einstellung der Hauptsspeichergröße hängt also stark von lokalen Gegebenheiten ab.

- cache_swap (megabytes)

  Die maximale Größe des permanenten Cache-Bereiches auf der Festplatte wird hier angegeben. Man muß jedoch in Betracht ziehen, daß *Squid* beim ersten Aufruf seine komplette Cache-Struktur in Form einer Verzeichnisstruktur bereits auf der Platte abbildet, was den verfügbaren Plattenplatz beschränkt. Dieser Wert sollte deshalb ca. 10% kleiner als der real verfügbare freie Festplattenspeicher sein, den man verwenden will - die Prozentangabe kann jedoch mit der verwendeten Inode-Dichte variieren. *Squid* reagiert etwas empfindlich, wenn der verfügbare Platz kleiner ist als der zugeteilte. Programmabbrüche können die Folge sein.

- cache_swap_low (percent, 0-100)

  Squid verwendet LRU-Mechanismen, um Festplattenplatz freizugeben. Dieser Wert gibt an, bei welchem Füllgrad die Freigabe beendet werden soll. Der Wert wird in Prozent angegeben.

- cache_swap_high (percent, 0-100)

  Wie cache_swap_low, nur wird hier der Wert angegeben, bei dem das Freigeben beginnen soll.

- cache_mem_low (percent, 0-100)

  Dieser Wert gibt an, bei welchem Füllgrad des RAM-Cache das Auslagern auf die Festplatte beendet werden soll. Auch dieser Wert wird in Prozent angegeben.

- cache_mem_high (percent, 0-100)

  Analog zu cache_mem_low wird hier der Füllgrad angegeben, bei dem das Auslagern beginnen soll.

- maximum_object_size

Die maximale Größe von Objekten, welche noch im Cache aufbewahrt werden. Die Angabe ist in KByte.

- ipcache_size (number of entries)

  Größe für den IP Cache, der zum Aufbewahren von Adressen benutzt wird.

- ipcache_low (percent), ipcache_high (percent)

  Wie bei cache_mem und cache_swap kann man hier Schwellenwerte (in Prozent) angeben.

- cache_dir

  Hier wird das Verzeichnis angegeben, in dem Squid seinen Festplattencache anlegt. Werden mehrere *cache_dir*-Einträge angegeben, verteilt Squid seinen Cache auf alle Verzeichnisse; dies wirkt sich positiv auf die Performance aus, wenn die Verzeichnisse auf unterschiedlichen Festplatten liegen, da dann parallele Zugriffe möglich werden. Die optimale Lösung ist sicher ein RAID-Array, was jedoch nicht gerade preiswert ist.

- cache_access_log

  Pfad und Name der Logdatei für Anfragen von Client-Rechnern.

- cache_log

  Pfad und Name der Datei für Debug-Ausgaben von Squid.

- cache_store_log

  Die Logdatei für den *Storage Manager* von Squid wird hier angegeben.

- cache_swap_log

  Diese Datei enthält die Metadaten für die Cache-Verwaltung. Normalerweise liegt sie im ersten *cache_dir*-Verzeichnis, man kann jedoch auch eine alternative Datei angeben. Sollte diese Datei gelöscht oder beschädigt werden, hat dies Auswirkungen beim Neustart, da Squid aus dieser Datei den Inhalt des Festplatten-Cache rekonstruiert.

- emulate_httpd_log

  Man kann Squid anweisen, für das Logfile ein Format zu verwenden, das z.B. vom CERN *httpd* verwendet wird, wenn man für dieses Format bereits Tools zur Auswertung besitzt.

- log_mime_hdrs

  Die MIME-Header von HTTP-Transaktionen können mit dieser Option protokolliert werden.

- useragent_log

  Der Name des verwendeten Clients kann mit dieser Option protokolliert werden.

- pid_filename

  In diese Datei schreibt *squid* seine Prozeß-ID, was für einige Automatisierungen praktisch ist.

- debug_options

  Die verschiedenen Debug-Level können hier eingestellt werden.

- ident_lookup

  Auf Wunsch verwendet *squid* RFC 931-Anfragen, um den jeweiligen Benutzer mitzuprotokollieren.

- log_fqdn

  Mit dieser Option werden die kompletten Rechnernamen von jedem Zugriff mit ins Logfile geschrieben.

- client_netmask

  Man kann die Host-Bits von geloggten IP-Adressen mit dieser Maske ausblenden, um die Privatsphäre seiner Anwender sicherzustellen.

2.3.3.5 Externe Zusatzprogramme

- ftpget_program

  Hier kann das Programm, das für FTP-Aktionen verwendet werden soll, angegeben werden.

- ftpget_options

  Eventuell erforderliche Parameter für das ftpget_program kann man hier angeben.

- ftp_user

  Die zu verwendende Login-Kennung für FTP-Aktionen wird hier eingestellt.

- cache_dns_program

  Das Programm für DNS-Auflösung wird mit dieser Option spezifiziert.

- dns_children

  Die Anzahl von DNS-Serverprozessen wird hier angegeben, ist sie zu gering, nimmt die Performance von *squid* rapide ab.

- dns_defnames

  Ermöglicht die Auflösung von Rechnernamen, die nur aus der Namenskomponente bestehen (d.h. kein Domainsuffix haben).

- unlinkd_program

  Mit diesem Programm werden nicht mehr benötige Cache-Dateien gelöscht.

- pinger_program

  Dieses Programm wird verwendet, um Neighbors zu kontaktieren.

- redirect_program

  Das Programm für URL-Redirection kann hier eingestellt werden. Momentan ist dem Squid-Paket noch kein solches Programm beigelegt, man muß also selbst kreativ werden, wenn man diese Funktionalität benötigt.

- redirect_children

  Gibt an, wie viele Instanzen von redirect_program gestartet werden sollen.

2.3.3.6 Cache-Tuning

- wais_relay

  Dient dazu, WAIS (wide area information service) Anfragen an eine konfigurierbaren Rechner weiterzuleiten.

- request_size

  Diese Option dient zur Einstellung der maximalen Größe für Objekte, vor allem bei POST-Operationen (d.h. Upload-Vorgängen) sollte dieser Wert ausreichnend groß dimensioniert sein.

- refresh_pattern

  Hier kann der Refresh-Algorithmus von *squid* für die eigenen Bedürfnisse optimiert werden.

- refresh_pattern/i

  Wie refresh_pattern, nur, daß Musterangaben die Groß-/Kleinschreibung ignorieren.

- reference_age

  Diese Angabe verändert den Wert den maximalen Wert für den LRU-Mechanismus des Cache-Systems, was bedeutet, daß Objekte, die im vergangenen Zeitraum reference_age nicht mehr angefordert wurden, aus dem Cache entfernt werden. Diese Option ist nicht gültig für das Entfernen von älteren Objekten, wenn der

Cache zu voll wird. Hier werden immer die nach LRU ältesten Einträge zuerst entfernt, was sich auch durch hohe Werte von reference_age nicht unterbinden läßt.

- quick_abort

  Hier kann eingestellt werden, ob und wann ein vom Benutzer abgebrochener Download eines Objekts trotzdem noch weiter in den Cache geladen werden soll.

- negative_ttl (minutes)

  Hier kann eine Zeit angegeben werden, in der fehlerhafte Requests nicht neu geholt, sondern sozusagen vom Cache als fehlerhaft eingestuft werden, was auch dem Benutzer mitgeteilt wird. TTL bedeutet *time to live*.

- positive_dns_ttl (minutes)

  Der Zeitraum, für den eine erfolgreiche DNS-Auflösung gespeichert wird, kann hier eingestellt werden.

- negative_dns_ttl (minutes)

  Wie bei negative_ttl kann hier die Speicherdauer von fehlgeschlagenen DNS-Auflösungen parametriert werden.

Timeouts sind erforderlich, um nicht bestätigte Netzwerkanfragen nach einer gewissen Zeitspanne abzubrechen. Würde man das unterlassen, wären sehr bald alle verfügbaren Ports verbraucht und der Proxy wäre nicht mehr fähig, seine Aufgaben zu erfüllen.

- connect_timeout (seconds)

  Gibt die Zeit an, in der der Verbindungsaufbau zum Cache-Server abgeschlossen sein muß. Bei Überschreitung dieser Zeitspanne wird die Verbindung abgebrochen.

- read_timeout (minutes)

  Eine bereits aufgebaute Verbindung wird nach dieser Zeit der Inaktivität wieder beendet.

- client_lifetime (minutes)

  Die maximale Verbindungszeit für einen Client bei einem Objekttransfer. Bei Modemverbindungen sollte dieser Wert nicht zu knapp bemessen werden, da sonst der Download von größeren Dateien immense Probleme verursacht.

- shutdown_lifetime (seconds)

  Gibt die Zeit an, die *squid* nach dem Erhalt von entsprechenden Signalen noch wartet, bis alle Verbindungen getrennt werden.

2.3.3.7 Access Control Lists

Dieser Konfigurationspunkt ist in der Datei *squid.conf* so gut beschrieben, daß man es nicht besser machen kann. Bei Bedarf suche man in dieser Datei im Abschnitt ACCESS CONTROLS nach diesbezüglichen Informationen.

2.3.3.8 Administrationsparameter

- cache_mgr

  Hier trägt man die E-Mail-Adresse des Betreuers ein, damit bei Bedarf ein Ansprechpartner vorhanden ist.

- cache_effective_user

  Dieser Eintrag ist erforderlich, wenn *squid* vom Administrator *root* gestartet wird (z.B. beim Booten). Hier sollte entweder ein sicherer Benutzer und eine sichere Gruppe eingetragen werden, die im System nur sehr beschränkte Rechte haben (wie *nobody/nogroup*), oder für *squid* wird ein neuer Benutzer und eine neue Gruppe angelegt. Nach dem Starten von *squid* ändert der Prozeß dann seine eigenen Zugriffsrechte, womit eventuelle Sicherheitsprobleme vermieden werden, die auftreten könnten, wenn*squid* als root-Prozeß läuft. Zu beachten ist hierbei, daß das Verzeichnis für die Cache-Daten sowie die Installationsverzeichnisse ebenfalls dem neuen Benutzer gehören müssen, sonst gibt das Programm etwas irreführende Fehlermeldungen aus, die zwar bei anderen Betriebssystemen gang und gäbe sind, einen Unix-Benutzer aber eher verwirren dürften.

- visible_hostname

  Unter diesem Namen macht sich der Proxy bekannt. Dieser Name ist aber nicht frei wählbar, sondern sollte mit dem DNS-Eintrag des Computers übereinstimmen.

2.3.3.9 Cache Registration Service

Dieser Service dient dazu, andere Squid-Betreiber zu finden, um Cache-Hierarchien aufzubauen.

- cache_announce

  Hier wird die Häufigkeit von Registrierungsmeldungen eingestellt.

- announce_to host[:port] [filename]

  Die genaue Adresse vom Empfänger der Registrierungsmeldungen kann hier angegeben werden. Der Inhalt von **filename** wird, sofern vorhanden, an die Meldung angefügt.

2.3.3.10 HTTP Acceleration

Hinter dieser etwas irreführenden Bezeichnung versteckt sich ein Proxy, der völlig anders funktioniert als man es erwarten würde:

- Der Accelerator dient dazu, den Zugriff von externen Netzwerken auf die lokalen WWW-Daten zu beschleunigen. Die eigenen Benutzer profitieren davon also nicht.
- Die Arbeitsweise ist ähnlich wie beim Proxy-Cache, nur, daß die Cache-Daten eine Kopie der lokalen Webseiten darstellen.

Wegen dieser Arbeitsweise ergeben sich folgende Vorteile:

- Es ist möglich, den Accelerator außerhalb der Firewall (sofern vorhanden) zu betreiben, interne statische WWW-Daten müssen dann nicht bei jedem externen Request durch die Firewall geleitet werden, womit die Firewall entlastet wird.
- Der Accelerator ermöglicht es, mehrere interne WWW-Server logisch zusammenzufassen und so den Zugriff über eine einzige URL-Hierarchie zu ermöglichen.
- Die statischen Web-Objekte (z.B. Bilder) werden vom Cache des Accelerators geliefert, dynamische Objekte (z.B. CGI) weiterhin vom WWW-Server, der damit von den I/O-intensiven Ladevorgängen entlastet wird.

Die Nachteile seien jedoch auch kurz erwähnt:

- Sind Proxy-Cache und Accelerator auf dem selben Rechner installiert, relativiert sich die Entlastung der Firewall, da alle internen Zugriffe auf den Proxy-Cache durch die Firewall geleitet werden. Legt man den Rechner hinter die Firewall, entfällt die Entlastung ebenfalls, da alle externen Requests wiederum vom Accelerator *hinter* der Firewall beantwortet werden.
- Will man dies vermeiden, muß man Proxy-Cache und Accelerator jeweils auf einem eigenen Rechner betreiben, wodurch natürlich der Administrationsaufwand steigt, außerdem benötigt man noch einen zweiten Rechner für den Accelerator, der ähnlich ausgestattet ist wie der Proxy-Cache. Dies wirkt sich zwar positiv auf die Performance aus, man sollte aber auch den Kostenfaktor in Betracht ziehen.

Folgende Optionen steuern den HTTP Accelerator:

- httpd_accel [real_httpd_host] [real_httpd_port]

  Gibt die Adresse und den Port des richtigen HTTP-Servers an.

- httpd_accel_with_proxy

  Bei Aktivierung ist *squid* gleichzeitig Accelerator und Proxy.

- httpd_accel_uses_host_header

  Diese Option aktiviert den Accelerator für mehrere verschiedene HTTP-Server, allerdings ist diese Option etwas unsicher, da keine Überprüfung stattfindet. Deshalb sollte diese Option deaktiviert werden.

2.3.3.11 Zusätzliche Optionen

- dns_testnames

  Diese Option enthält Einträge für den Test der DNS-Server, welcher beim Starten automatisch initiiert wird.

- logfile_rotate #

  Hier kann angegeben werden, wie viele ältere Logfiles aufzubewahren sind, wenn zyklisch neue Logfiles angelegt werden. Ein Eintrag von 0 ermöglicht die Verwendung diverser Logfile-Auswertetools (siehe 6.2), denn dann kann das Logging kurzzeitig deaktiviert werden.

- append_domain

  Der hier angebene Domain-Name (der mit einem Punkt beginnen muß), wird als Domain-Suffix an einzelne Rechnernamen angefügt.

- tcp_recv_bufsize

  Die Größe der Empfangspuffer für TCP-Sockets kann hier angeben werden.

- ssl_proxy

  Ein eventuell vorhandener SSL (*secure socket layer*) -Proxy kann hier eingebunden werden.

- passthrough_proxy

  Dieser Eintrag gibt an, ob alle nicht-GET-Anfragen (wie POST und PUT) weitergeleitet werden sollen, und wenn ja, auf welchen Rechner.

- proxy_auth

  Man kann den Zugriff auf den Proxy auch beschränken und einzelnen Benutzern Paßwörter zur Freischaltung zuweisen.

- err_html_text

  Dieser HTML-Text wird in Fehlermeldungen eingebunden, um z.B. einen Link auf übergeordnete Seiten oder eine Mail-Funktion einzubauen.

- deny_info

  Hier kann im Falle von nicht-öffentlichen Seiten eine Information für alle hinterlegt werden, die keinen Zugriff haben.

- udp_hit_obj

  Diese Option bestimmt, wie sich *squid* gegenüber seinen *Neighbors* verhalten soll.

- udp_hit_obj_size

  Hier kann die Puffergröße für UDP-Hit-Verbindungen eingestellt werden.

- memory_pools

  Diese Option beeinflußt das Speichermanagement von *squid*.

- forwarded_for

  Die Angabe der Client-Adresse in weitergeleiteten Requests kann mit dieser Option unterdrückt werden.

- log_icp_queries

  Steuert das Verhalten beim Logging von ICP-Anfragen.

- minimum_direct_hops

  Bei der Verwendung der ICMP-Optionen werden Objekte, die auf nahegelegenen (d.h. durch weniger als minimum_direct_hops Netzwerkkomponenten getrennten) Servern vorhanden sind, direkt geladen.

- cachemgr_passwd

  Bei der Verwendung des *cachemgr.cgi*-Programmes können hier die einzelnen Abfragemöglichkeiten durch Paßwörter geschützt werden.

- swap_level1_dirs

  Im Verzeichnis für den Festplattencache wird die hier angegebene Anzahl von Unterverzeichnissen angelegt.

- swap_level2_dirs

  In jedem Unterverzeichnis im Verzeichnis des Festplattencache werden nochmals Verzeichnisse angelegt, die Anzahl kann hier eingestellt werden. Man sollte Bedenken, daß jedes Verzeichnis Platz auf der Festplatte belegt, was den Speicherplatz für Cache-Objekte verringert.

- store_avg_object_size

  Die durchschnittlich erwartete Objektgröße, dieser Wert wird benötigt, um diverse Statistiken aufstellen zu können.

- store_objects_per_bucket

  Objektanzahl in jeder logischen Einheit der Hashtable-Verwaltung für die Cacheobjekte.

- http_anonymizer

  Verändert die Angaben, die vom Client-Programm zum jeweiligen Webserver weitergegeben werden. Hier kann die Funktionalität von WWW-Anonymizern nachgebildet werden.

- client_db

  Diese Option faßt alle Statistiken zusammen und unterscheidet nicht nach einzelnen Clients.

- netdb_low

  Diese Angaben steuern die ICMP-Verwaltung, im Gegensatz zu ähnlichen Optionen sind hier Werte, keine Prozentangaben, anzugeben.

- netdb_high

  Diese Option gibt die obere Schwelle für die ICMP-Verwaltung an und gehört damit zu netdb_low.

- netdb_ping_rate

  Gibt den Zeitabstand von ICMP-Pings an.

- query_icmp

  Fordert von Rechnern im Cacheverbund ICMP-Informationen an, um Laufzeitoptimierungen zu ermöglichen.

- icp_hit_stale

  Aktiviert die Anzeige von Cache-Hits für bereits „abgelaufene“ Objekte, was manchmal bei lokalen Cache-Hierarchien praktisch sein kann.

2.3.4 Proxy-Aktivierung beim Systemstart

Ein manuelles Starten von Squid ist für erste Tests und nach Änderungen der Konfiguration kein Problem, im produktiven Einsatz empfiehlt es sich jedoch, bereits beim Wechseln in den Multiuser-Modus Squid automatisch zu starten. Bei dem auf dem Testsystem verwendeten Solaris-Betriebssystem wird dazu im Verzeichnis */etc/rc3.d* bzw. */etc/rc2.d* eine Batchdatei abgelegt, die sowohl das Starten als auch das saubere Herunterfahren von Squid (und einigen anderen getesteten Programmen) übernimmt. Dieser Vorgang ist auf anderen System-5-Unix-Derivaten fast identisch, nur das Verzeichnis kann variieren. Bei anderen Unix-Betriebssystemen sollte es ebenfalls kein Problem sein, Squid bereits beim Booten zu aktivieren.

Ein Beispiel einer solchen Batchdatei ist in Kapitel A.3 zu finden.

2.3.5 Cache-Manager

Der Cache-Manager ermöglicht die Überwachung des laufenden *squid*-Prozesses. Da der Zugriff über ein WWW-Interface erfolgt, liegt dem Paket ein CGI-Programm bei, das die Abfrage verschiedener Cache-Parameter (aus der Konfiguration und auch Laufzeitvariablen) ermöglicht. Das CGI-Programm erfordert logischerweise einen laufenden WWW-Server, in dessen *cgi-bin*-Verzeichnis das Programm *cachemgr.cgi* kopiert wird. In der Testkonfiguration war der WWW-Server auf einem internen Interface des Testrechners aufgesetzt; eine Abfrage eines nicht-lokalen *Squid* ist jedoch ebenfalls möglich (sofern die Zugriffsrechte vorhanden sind).

- Cache Host

  Der Name des Rechners, auf dem *Squid* läuft. In der Testkonfiguration war dieser Rechner ebenfalls der Web-Server, so daß die Eingabe von *localhost* ausreichte.

- Cache Port

  Der *Squid*- Port ist per default auf 3128 gesetzt. Da in der Testkonfiguration keine Änderungen erfolgten, wurde hier ebenfalls dieser Wert eingetragen.

- Password

  Hier muß das in der Datei *prefix* /etc/squid.conf eingestellte Paßwort eingegeben werden. Wofür ein Paßwort erforderlich ist, kann in der Konfigurationsdatei eingestellt werden.

- URL

  Falls für die gewählte Operation eine Objekt-Adresse (URL) erforderlich sein sollte, ist sie hier anzugeben.

- Operation

  Hier kann die gewünschte Abfrage gewählt werden. Momentan gibt es 18 unterschiedliche Bereiche, die im folgenden kurz vorgestellt werden.

2.3.5.1 Cache Information

Diese Option gibt eine Vielzahl von Informationen aus, die ein Feintuning von *Squid* erleichtern:

- Gesamtzahl der TCP- und UDP-Connections sowie den Stundendurchschnitt

  Dies ist für den Cache-Betreiber wichtig, um den Grundbedarf besser einschätzen zu können.

- Verbrauch von Speicherplatz im RAM und im Dateisystem

  Damit ist eine leichte Kontrolle des Proxies möglich, ohne auf Systemebene manuell nachzusehen.

- beanspruchte Rechenzeit für System- und Prozeß-Aktivität

  Falls die CPU unterdimensioniert ist, steigt die *Usage* stark an, ein hoher Anteil an aufgewendeter Systemzeit ist Indikator für einen nicht optimierten Kernel und zu langsames Festplattensubsystem. Ein sehr hoher Wert der *page faults* deutet auf zu geringen Hauptspeicherausbau hin.

- Status der Filedescriptoren (geöffnete Dateien und Netzwerkverbindungen)

  Ist die Zahl der verfügbaren Descriptoren auf Dauer sehr gering, sollte eine Erhöhung des Maximums in Erwägung gezogen werden. In der Praxis jedoch sind vom Betriebssystem Grenzen gesetzt, der Solaris-Kernel macht ab 4096 manchmal Probleme, Linux benutzt normalerweise nur 256 (was etwas wenig ist und deshalb mit einem Patch auf 1024 gesetzt werden sollte). Andere Unix-Derivate (wie NetBSD/FreeBSD) haben weiter gesteckte Grenzen, jedoch kann man der Performance auch schaden, wenn die Tabellen im Kernel zu groß werden und die Suchzeit deshalb ansteigt. Die Erhöhung setzt meist eine Neucompilierung bzw. Neukonfigurierung des Kernels voraus, was nicht jedermanns Sache ist.

  Bei den Testläufen war eine Zahl von 1024 nicht der Flaschenhals des Proxies, der begrenzende Faktor waren Festplatte und CPU.

- Anzahl der internen Speicherobjekte

  Ein mehr statistischer Wert, da man hier keine Aussage über Größe und Art der Objekte erhält; *Objects* und *VM Objects* sind in diesem Bereich aussagekräftiger. Wenn die Anzahl der Objekte interessant ist, sollte man auch einen Blick auf *Utilization* werfen.

- Aufschlüsselung der Speicherverteilung innerhalb des *squid*-Prozesses

  Sofern man nicht gerade den Speichermanager optimieren will, ist nur der belegte Gesamtspeicher von Bedeutung. Die anderen Werte sind zwar interessant, aber für Normalbenutzer nicht so wichtig.

2.3.5.2 Cache Configuration File

Hier wird lediglich eine Kopie der Konfigurationsdatei angezeigt, was bei der Fehlersuche in einem Cache-Verbund von Bedeutung sein kann. In unserer Testumgebung war ein direkter Zugriff auf die Konfigurationsdatei meist praktikabler.

2.3.5.3 Cache Parameters

Gibt die relevanten Parameter des Proxies aus, eine Art Extrakt aus der vorhergehenden Option.

2.3.5.4 Utilization

Aufgeschlüsselt nach dem Protokoll erhält man Informationen über Objektanzahl, Speicherbedarf, Trefferrate und einige Übertragungsstatistiken.

2.3.5.5 I/O

Eine Statistik über Ein- und Ausgabeoperationen läßt sich hier abrufen.

2.3.5.6 HTTP Reply Headers

Für interessierte Administratoren ist dieser Punkt vorhanden. Der Nutzen ist manchmal zweifelhaft.

2.3.5.7 Filedescriptor Usage

Hier erhält man Statusinformationen über die belegten Filedescriptoren. Jede aufgebaute Netzwerkverbindung belegt einen Descriptor. Sind keine mehr frei, schlagen Verbindungsversuche fehl.

2.3.5.8 Network Probe Database

Der Inhalt der Netzwerk-Datenbank kann hier angezeigt werden.

2.3.5.9 Objects

Diese Option generiert eine Webseite mit Referenzen zu allen Objekten, die sich momentan im Festplatten-Cache von Squid befinden. Da es sich um sehr viele Objekte handeln kann – im Testbetrieb waren es teilweise über 50.000 –sollte man den Umfang der Seite (meist einige MByte) nicht unterschätzen; ebenso ist die Belastung der CPU während der Generierung sehr hoch. Angezeigt werden schließlich die verschiedenen Parameter der Cache-Objekte (wie Größe und Aufbewahrungszeitraum) sowie ein HTML-Link, mit dem man sich das Objekt direkt im Browser anzeigen lassen kann.

2.3.5.10 VM Objects

Diese Option funktioniert ähnlich wie *Objects*, nur, daß hier eine Liste der gerade im Speicher vorhandenen Cache-Objekte im HTML-Format angezeigt wird.

Flags	C: Cached D: Dispatched N: Negative Cached L: Locked
lstref (last referenced)	Zeit seit der letzten Verwendung
TTL (Time-To-Live)	Gültigkeitsdauer
N	Zähler für Adressen

Tabelle 2.3: Squid – IP Cache Contents Schlüsselwörter

2.3.5.11 Cache Server List

Eine Statistik der Neighbor-Caches. Da im Testbetrieb ein *stand-alone*-Proxy eingesetzt wurde, war diese Option nicht von Bedeutung.

2.3.5.12 Cache Client List

Eine Auflistung der Computer, die gerade auf den Proxy zugreifen, zusätzlich wird noch eine Statistik über die Trefferart und -häufigkeit ausgegeben.

2.3.5.13 IP Cache Contents

Hier wird ein Liste angezeigt, die DNS-Namenseinträge und die zugehörige(n) IP-Adressen beschreibt. Diese Daten werden von den *dnsserver*-Prozessen gesammelt, die Squid automatisch startet. Die Bedeutung der einzelnen Spalten ist in Tabelle 2.3 beschrieben. Die Anzahl der IP-Adressen pro DNS-Name ist bei einigen Content-Providern (z.B. *www.geocities.com*) oft im zweistelligen Bereich, auch Digitals Altavista verfügt über eine erstaunliche Anzahl an IP-Adressen.

2.3.5.14 FQDN Cache Contents

Der FQDN-Cache ist die Umkehrung des IP Cache, d.h. es werden IP-Nummern auf Hostnamen abgebildet.

2.3.5.15 DNS Server Statistics

Eine Auslastungsanzeige der von Squid gestarteten DNS-Server, sind alle DNS-Server beschäftigt, kann es zu Wartezeiten kommen, da Squid dann bei der Namensauflösung blockiert, was natürlich die Performance negativ beeinflußt.

2.3.5.16 Redirector Statistics

Der Redirector war in der getesteten Version noch nicht enthalten. Auf einen Test von selbstgebastelten Ersatz-Programmen wurde verzichtet.

2.3.5.17 Shutdown Cache

Diese Option bewirkt ein Herunterfahren des Proxies. Für diese Option muß in der Konfigurationsdatei auf jeden Fall ein Paßwort angegeben werden.

2.3.5.18 Refresh Object

Für diese Option muß eine URL angegeben werden. Das betroffene Objekt wird dann neu in den Cache transferiert.

2.3.6 Erfahrungsbericht

2.3.6.1 Testumgebung und Testlauf

Der Aufbau der Testumgebung ist in Kapitel A.1 beschrieben. Die Konfiguration des Proxy-Cache war folgende:

- Als Proxy-Server wurde Zerberus verwendet, eine Sun Sparc 4 mit 110 MHz und 96 MByte Hauptspeicher. Dieser Rechner wurde uns dankenswerterweise von Herrn Professor Kopp zur Verfügung gestellt. Diese Workstation kann als zuverlässig gelten, die einzigen Schwachpunkte waren die etwas langsame interne Festplatte und die nicht ganz aktuelle CPU. Für den Anwendungszweck im Testaufbau war die Workstation jedoch ausreichend dimensioniert.
- Als WWW-Server wurde ein Linux-Rechner verwendet, der mit einer doppelten ISDN-Standleitung ans Internet angebunden war. Dies was erforderlich, um einen Flaschenhals in die Transferstrecke einzubauen, denn im Netzwerk auf dem Campus wäre kein bemerkenswerter Geschwindigkeitsgewinn möglich gewesen.
- Als WWW-Client verwendete ich ebenfalls die Zerberus, was durch die Testumgebung bedingt war.

Der Testlauf bestand aus dem Zugriff auf alle WWW-Seiten des WWW-Servers. Mittels *wget* wurden alle Seiten des WWW-Servers unter verschiedenen Voraussetzungen mehrmals auf den Testrechner heruntergeladen und die erforderliche Zeit mit *time* gemessen. Bei Abweichungen in den gemessenen Zeiten wurde jeweils aus drei Testzeiten der Durchschnitt bestimmt.

Der Ablauf des Tests ist hier protokolliert:

- Test 1: Ohne Proxy-Cache direkt auf die Festplatte des Testrechners.

```
Zerberus:~/FTP >> time wget -F -r 194.95.203.129
...
Downloaded: 718,158 bytes in 86 files
0.92u 1.47s 0:44.11 5.4%
```

Der fettgedruckte Wert war die Referenz-Zeit, die es zu verbessern galt.

- Test 2: Mit aktiviertem Proxy-Cache, der vorher durch einen Neustart mit entsprechenden Optionen komplett geleert wurde.

  Dies diente dazu, den Proxy-Cache mit den WWW-Seiten zu füllen.

  Zerberus:~/FTP >> time wget -F -r 194.95.203.129
  ...
  Downloaded: 718,158 bytes in 86 files
  0.81u 1.40s **0:51.11** 4.3%

Die Zeit war erwartungsgemaßschlechter als der direkte Ladevorgang, da einiger Verwaltungsaufwand betrieben werden muß, wenn die Daten **durch** den Proxy geladen werden. Ein **einmaliger** Zugriff auf eine Webseite dauert also mit Proxy sogar länger als ohne.

- Test 3: Dann wurde die Messung wiederholt, wobei ein hoher Prozentsatz der WWW-Seiten nun jedoch aus dem Proxy-Cache (im Hauptspeicher , d.h. VM-Cache) gelesen werden konnte.

  Zerberus:~/FTP >> time wget -F -r 194.95.203.129
  ...
  Downloaded: 718,158 bytes in 86 files
  0.88u 1.12s **0:08.81** 22.7%

Der beachtliche Geschwindigkeitsgewinn wäre noch größer gewesen, wenn die verwendete CPU schneller gewesen wäre.

- Test 4: Die Messung in Test 3 war ein Idealwert, da beim Zugriff auf die Daten im Proxy-Cache nicht auf die Festplattte zugegriffen wurde. Es wurden deshalb weitere WWW-Seitenüber den Proxy geladen, um eine Verlagerung von Proxy-Daten in den Festplatten-Cache zu erzwingen, dann wurde der Test gestartet.

  Zerberus:~/FTP >> time wget -F -r 194.95.203.129
  ...
  Downloaded: 718,158 bytes in 86 files
  1.07u 1.22s **0:09.70** 23.6%

Das Ergebnis war wegen der erforderlichen Suche auf der Festplatte langsamer, was aber vom Benutzer kaum bemerkbar sein dürfte.

- Test 5: Als nach einigen Stunden erneut eine Messung vorgenommen wurde, waren alle Daten des betroffenen Servers auf die Festplatte ausgelagert, was die Zugriffszeit weiter verschlechterte. Dieser Test wurde nur einmal ausgeführt, da nach dem Ladevorgang die betroffenen Objekte ja wieder im VM-Cache waren.

```
Zerberus:~/FTP >> time wget -F -r 194.95.203.129
...
Downloaded: 718,158 bytes in 86 files
1.02u 1.29s 0:11.06 20.8%
```

Selbst nach der Verschlechterung in Test 5 war der Zugriff über den Proxy wesentlich schneller als der direkte Zugriff auf den Webserver. Eine selbst im schlechtesten Fall vierfache Geschwindigkeit bei wiederholten Zugriffen auf Webseiten ist ein Indikator für die Brauchbarkeit von Proxy-Systemen wie *squid*. Sicher waren die Testbedingungen nicht ganz realistisch, aber selbst eine Verdoppelung von Zugriffszeiten macht sich beim Anwender bereits bemerkbar.

2.3.6.2 Weitere Erkenntnisse

Beim Betrieb von Squid über Modemleitungen ist zu beachten, daß Squid einen aktiven DNS-Server benötigt, um selbst als DNS-Server für die Client-Rechner tätig werden zu können. Squid kann also erst nach dem erfolgreichen Verbindungsaufbau gestartet werden.

2.3.6.3 Bewertung

Squid lief fast während der gesamten Dauer der Diplomarbeit in der Testumgebung und wurde von einigen Personen als Proxy (HTTP, FTP) benutzt. In dieser Zeit belegte Squid zweimal den gesamten ihm zur Verfügung gestellten Cache-Bereich auf den Festplatten, wobei durch ungünstige Voraussetzungen jedesmal ein Absturz auftrat, da der Cache mehr Platz belegen wollte als vorhanden war. Nachdem das Problem erkannt und behoben war, traten keine Instabilitäten mehr auf. Squid ist somit als sehr stabil und funktionell zu bezeichnen. Die Performance beim WWW-Zugriff konnte deutlich gesteigert werden, weshalb ich nur empfehlen kann, nach Möglichkeit einen Proxy-Cache einzusetzen. Der einzige Nachteil ist die längere Zugriffszeit, wenn eine Webseite erstmalig geladen wird, was in der Praxis aber kein Problem darstellt.

2.3.6.4 Hardware-Empfehlung

Der für den Proxy-Cache verwendete Rechner sollte über ausreichend Hauptspeicher und mehrere Festplatten bzw. ein RAID-Array verfügen, auf denen die Cache-Verzeichnisse verteilt werden können. Die im Test verwendete Festplatte (Seagate-ST43400N) war durch die langsame Zugriffszeit sicher ein begrenzender Faktor für die Leistungsfähigkeit von Squid. Die Geschwindigkeit der CPU ist eher zweitrangig; allerdings sollte man nicht erwarten, mit veralteter Hardware Spitzenleistungen zu erzielen.

Kapitel 3

Filter-Systeme

3.1 Drawbridge 2.0

Jürgen Mayerhofer
Quelle:
ftp://ftp.cert.dfn.de/pub/tools/net/drawbridge

3.1.1 Überblick

3.1.1.1 Was ist Drawbridge?

Drawbridge ist ein DOS-basierter TCP/UDP-Paketfilter, der IP/Port-orientiert Pakete erlaubt oder verbietet. Im Sommer 1992 wurde die A&M Universität in Texas von Hackern unter Beschuß genommen. Mit Hilfe der Tools *Drawbridge, tiger* und *netlog* gelang es, den Angriff abzuwehren.

Tiger ist eine Sammlung von leicht zu bedienenden Unix-Shellscripten, die Unix-Rechner auf Veränderungen im Dateisystem überprüfen.

Netlog ist ein Satz von Netzwerküberwachungsprogrammen, die Eindringlinge entdecken können. Die DOS-Box betreibt kein Packet-Logging, d.h. hierfür wird ein eigener Rechner außerhalb des Paketfilters benötigt.

Drawbridge-Filterregeln werden in einer ASCII-Datei definiert und durch einen Filtercompiler in Binärdaten umgewandelt. Diese können entweder über Diskette oder über den Filtermanager auf den DOS-Rechner übertragen werden.

Der Filtercompiler und der Filtermanager sind im Drawbridgepaket als ausführbare Binaries für Sun4-Solaris und als C-Sourcecode enthalten. Die Makefiles erlauben das Übersetzen auf Solaris, SunOS und AIX. Uns ist zusätzlich eine Portierung auf Linux gelungen.

3.1.1.2 Anforderungen

Für ein komplettes Firewallsystem sind erforderlich:

1. Außerhalb des gesicherten Netzes ein Unix-Rechner zum Loggen der Angriffe mit *Netlog* (tcpdump, snoop, etc.).

2. ein DOS-PC mit zwei Netzwerkkarten, die nach Möglichkeit SMC8013 oder zu solchen kompatibel sind. Auf diesem läuft die 'Drawbridge'-Software. Ein 486-er mit 4-8 Megabyte RAM ist dafür ausreichend. Der Datendurchsatz beträgt 5,5 MBit/s. Wem dies zu langsam ist, der nimmt zwei FDDI-Karten, die NDIS unterstützen; der Durchsatz steigt dann auf bis zu 18 MBit/s an (dies sind theoretische Werte, wir konnten das mangels Hardware nicht nachprüfen).

3. innerhalb des gesicherten Netzes ein Sun-Solaris, SunOS, IBM-AIX oder Linux Rechner zum Erzeugen der Filtertabellen.

3.1.1.3 Versuchsaufbau

Der Versuchsaufbau ist im Anhang (Kapitel A.1) beschrieben.

3.1.2 Details

Die DOS-Box, auf der Drawbridge läuft, benützt NDIS-Protokollteiber. Es ist also bei der Auswahl der Netzwerkkarten darauf zu achten, daß NDIS unterstützt wird.

Das Drawbridge-Paket wird mit Treibern und Konfigurationsdateien, die für SMC 8013 Netzwerkkarten gültig sind, ausgeliefert. Man kann sich eine Menge Arbeit ersparen, wenn man zwei Netzwerkkarten dieses Typs verwendet.

Zur Konfiguration von NDIS nehme man die Fachliteratur zur Hand.

Um der Drawbridge (das eigentliche Programm verbirgt sich hinter filter.exe) Regeln beizubringen, muß folgendes Schema durchlaufen werden:

1. Eine Quelldatei mit den Filterregeln erzeugen / bearbeiten.

2. Diese Datei mit dem Filter Compiler compilieren.

3. Mit Hilfe des Filtermanagers oder auf Diskette die beim Compilerlauf erzeugten Dateien auf die Drawbridge uploaden.

3.1.2.1 Filtersprache

Die Filtersprache erlaubt das Gruppieren von Hosts/Ports. Die Dokumentation, die mit der Software mitgeliefert wird, ist sehr leicht verständlich. Die Regeln werden in einer ASCII-Datei gespeichert.

Die Syntax ist sprechend und für den Fachmann leicht zu durchschauen. Einzelne Regeln können gruppiert werden. Es reicht, die mitgelieferte Beispieldatei an den geeigneten Stellen abzuwandeln.

3.1.2.2 Filtercompiler

Der Filtercompiler (*fc*) erzeugt aus den ASCII-Filterregeln binären Output für die Drawbridge in Form von mehreren Dateien: eine „Erlaube“-Tabelle, eine „Verbiete“-Tabelle, eine Classes-Tabelle und pro definiertem Netzwerk eine Netzwerk-Tabelle.

Der Aufruf erfolgt durch

fc [-b] [-c] file

Die Option **-b** steht für Byteswap und erzeugt die Binärdateien für das Turnschuhnetz, also zum Übertragen auf die Drawbridge via Diskette. Ohne **-b** dient der Output dem Filtermanager.

-c gibt die Copyright-Message aus.

Als Parameter „file“ wird die ASCII-Datei übergeben, die die Filterregeln enthält.

3.1.2.3 Filtermanager

Christian Rotter und Jürgen Mayerhofer

Der Filtermanager (*fm*) dient zum Managen der *drawbridge* und läuft auf einem Rechner, dem der Zugriff über einen Eintrag in der Datei PROTOCOL.INI gestattet wurde.

Prinzipiell ist es möglich, den *fm* auf mehreren Rechnen laufen zu lassen, um aber Inkonsistenzen bei den Filterregeln zu vermeiden, sollten die Regeln nur von einem gewartet werden.

Im folgenden ist eine kurze Filtermanagersitzung beschrieben:

```
   jmayerho@mozart:/home/jmayerho/da/drawbridge-2.0/fm > fm
   Filter Manager 2.0
   Copyright (C) 1993, 1995 Texas A&M University
   fm> help
   commands:
     set (verbose|target|password|retries|timeout) <args>
     load (network|classes|allow|reject) <filename>
     show (host|class|allow|reject|target|verbose|password|retries|timeout)
[
   <args>]
     query (host|class|allow|reject|stats) [<args>]
     upload (networks|classes|allow|reject)
     write
     release (classes|allow|reject|network) [<args>]
     reboot clear (tables|stats)
     reset
     newpwd <password>
     quit
   '!' as the first character of a command passes the command to
a shell
   '#' is a comment
```

```
fm> query stats
--- DRAWBRIDGE STATS ---
                            Inside       Outside
Packets filtered                43          4657
Packets received           6747638      10136808
Packets transmitted       10136644       6720056
Up for 135 days 8 hours 48 minutes 26 seconds
Cache Accesses:    1353153  Cache Misses:     4  Hit Ratio: 100%
Dropped packets due to lack of packet buffers:      19697
Cache Accesses:    1353154  Cache Misses:     4  Hit Ratio: 100%
Dropped packets due to lack of packet buffers:      19697
--- CARD STATS ---
                            Inside       Outside
Rx Frames                195083648      10168952
Rx Bytes                 572794147    3398635575
Rx Multicast                   N/A          N/A
Rx Broadcast                 28193         7961
Rx CRC Errors                    0          N/A
Rx Buffer Drops                N/A          N/A
Rx Hardware Error Drops          0            0
Tx Frames                 10156157       6726653
Tx Bytes                3397204356     892669019
Tx Multicast                   N/A          N/A
Tx Broadcast                  7910        26790
Tx Timeout Drops               N/A          N/A
Tx Hardware Error Drops          0            0
fm> load classes class_tables
fm> upload classes
fm> write
fm> quit
jmayerho@mozart:/home/jmayerho/da/drawbridge-2.0/fm >
```

Der Filtermanager liest das Dotted-File „.fmrc“ aus dem Home-Verzeichnis des Benutzers. Folgende Optionen waren für uns geeignet:

set target 194.95.109.190

set verbose yes

Das angegebene Ziel ist die IP-Adresse unserer Drawbridge.

3.1.3 Installation und Konfiguration von Drawbridge

3.1.3.1 Installation auf dem PC

Die Installation besteht aus dem Entpacken einer Zip-Datei auf dem Drawbridge-PC und der Konfiguration, die über diverse INI-Dateien vorgenommen wird. Die wichtig-

ste hierbei ist die PROTOCOL.INI, in der die Netzwerkkarten angemeldet und wichtige Parameter für das Management eingestellt werden.

Die Sektion „Drawbridge“ in der Datei PROTOCOL.INI sieht folgendermaßen aus:

```
[DRAWBRIDGE] DriverName = DRAWBRDG
Listen = both
Inside = SMC80002_NIF
Outside = SMC8000_NIF
Ipaddress = "194.95.109.190"
Subnetmask = "255.255.255.0"
Manager = "194.95.109.177","194.95.109.180"
Gateway = "194.95.109.177"
```

Die verwendeten Schlüsselworte sind selbsterklärend, interessant ist lediglich, daß in der Testkonfiguration das Gateway (=Verbindung zur Außenwelt) gleichzeitig auch als Manager eingetragen ist, was es ermöglicht, das Gateway (Sun Sparc 4 mit 2 Netzwerkkarten) als Router fungieren zu lassen. Dies geschah ausschließlich zu Testzwecken, im Produktionsbetrieb sollten alle Management-Stationen *hinter* der Drawbridge liegen, um diese potentiellen Schwachstellen auszuschalten.

3.1.3.2 Konfiguration mit Diskette

Durch Verwendung der Option **-b** beim Aufruf von *fc* (der Filtercompiler von Drawbridge) werden Dateien erzeugt, die per Diskette auf den Drawbridge-Host übertragen werden können (zu Fuß, deshalb in der Beschreibung deshalb auch „Turnschuhnetzwerk“ genannt). Diese Dateien werden dann per MS-DOS in das jeweilige Verzeichnis kopiert und dann die Drawbridge über das Kommando *filter.exe* neu gestartet.

In der Praxis kam es dabei zu Problemen, da die verwendeten Treiber (NDIS) nach Abbruch des Filterprogramms zwar im Speicher blieben, beim Neustart jedoch die Zusammenarbeit verweigerten. Es mußte deshalb bei einem manuellen Updaten der Filterregeln per Diskette anschließend neu gebootet werden, um wieder einen lauffähigen Filter zu erhalten.

Es erwies sich deshalb als günstiger, die Drawbridge über Netzwerk mit dem zum Paket gehörenden Filtermanager *fm* zu konfigurieren und den nicht mehr zeitgemäßen Weg über den Datentransport mit Disketten zu vermeiden.

3.1.3.3 Konfiguration mit Filtermanager

Der Filtermanager *fm,* der dem Paket als C-Sourcecode beiliegt, wurde unter Solaris 2.5 und unter Linux übersetzt und ergab in beiden Fällen ein voll funktionsfähiges Management-Programm für die Drawbridge.

Um eine Konfiguration der Drawbridge über Netzwerk zu ermöglichen, muß auf dem Drawbridge-PC in der Datei PROTOCOL.INI, Sektion [DRAWBRIDGE] ein Eintrag folgender Form vorhanden sein: Manager = „<IP-Adresse>“

Es können auch mehrere Management-Stations angegeben werden, dazu werden einfach mehrere IP-Adressen (jeweils eingeschlossen in Anführungszeichen) getrennt durch Kommas angegeben.

Aus Sicherheitsgründen ist es ratsam, nur Rechner, die ***hinter*** der Drawbridge liegen, als Management-Hosts anzugeben, da sonst bei einem Einbruch auf einem vor der Drawbridge liegenden Management-Host das ganze Filtersystem sehr einfach umkonfiguriert und somit umgangen werden kann.

Ansonsten kann das Drawbridge-System als sehr sicher gelten, da der Drawbridge-Host ein sehr „dummer“ Rechner ist, der keine weiteren Dienste anbietet und somit auch keine Angriffspunkte für eventuelle Angreifer darstellt.

Beim Aufstellen von Filterregeln muß beachtet werden, daß DNS-Anfragen durchgelassen werden.

3.1.4 Sicherheitsaspekte

Zum Scannen verwendeten wir verschiedene Produkte. Diese Diplomarbeit befaßt sich jedoch damit, daß die Daten sicherer übertragen werden. Sie soll keine Anleitung zum Hacken sein. Deshalb ist lediglich das Ergebnis der Angriffsversuche angegeben.

3.1.4.1 Scan mit satan

Der Scan mit *satan* erwies sich als schwieriger als zunächst angenommen: *satan* ist ein Speicherfresser auf UNIX-Seite und verabschiedete sich beim Versuch, die Drawbridge zu Scannen, mit einem asynchronus Memory-Fault auf Seite der Sun. Die Sun mußte neu gebootet werden.

3.1.4.2 Portscan-Angriff

Der Angriff mit Portscan auf die Drawbridge selbst lieferte kein Ergebnis, d.h. sie liefert keine Pakete auf Requests gleich welcher Art zurück. Der Scan auf einen Rechner mit gesetzter Filterregel durch die Drawbridge hindurch bleibt bei der ersten verbotenen Regel stehen. Die Drawbridge hat also diesen „Angriff“ erfolgreich unterbunden.

3.1.4.3 Overload (Flooding) Attacks

Die Drawbridge konnte nicht mehr alle Pakete zu beantworten (wohl aufgrund ihrer langsamen CPU), blieb aber stabil am Laufen. Ebenso störte es die Drawbridge nicht, als die Rechner dahinter gefloodet, d.h. mit Netzwerk-Überlast angegriffen wurden.

3.1.4.4 Remote-Management mit fm

Der Datentransfer im *fm* erfolgt in der außerhalb der vereinigten Staaten erhältlichen Version unverschlüsselt! Es ist also möglich, die Daten abzuhören. Für die USA existiert zusätzlich die Möglichkeit, den Datenverkehr mit DES zu verschlüsseln.

Der oder die Rechner, die den Filter-Manager benützten, sollen sich innerhalb des zu schützenden Netzes befinden. Dadurch wird die Sicherheit erhöht, kann aber nicht mehr garantiert werden, wenn sich ein Angreifer eingeschlichen hat und innerhalb des geschützten Netzes arbeitet.

3.1.5 Fazit

Der gewonnene Eindruck sagt uns, daß Drawbridge ein durchdachtes Paket-Filtering System ist. Die Installation und Bedienung ging flott und ohne größere Probleme vonstatten. Der Verwaltungsaufwand ist angemessen. Wer aber erwartet hat, ein Firewall-System auf einem DOS-Rechner betreiben zu können, der wird enttäuscht: Drawbridge ist eine von drei Komponenten eines Sicherheitssystemes, für das ein Unix-Fachmann erforderlich ist.

Das Hauptaugenmerk liegt nicht auf DOS-Ebene, sondern auf dem Logging der Angriffe. Nur gegen festgestellte Attacken kann man vorgehen. Wenn jedoch namhafte Routerhersteller behaupten, ihr Produkt könne „Firewallen“, dann verbirgt sich jedoch auch meist nur ein Paket-Filter dahinter. Solche Router können durchaus durch eine Drawbridge ersetzt werden (dürfte billiger sein als ein Firewall-Router).

Deshalb geben wir der Drawbridge die Note „gut“.

3.2 ipfwadm

Jürgen Mayerhofer

Quelle:

http://simba.xos.nl/linux/ipfwadm

Das frei verfügbare Betriebssystem Linux beinhaltet eine Anzahl vom Möglichkeiten, auf Kernel-Ebene schnell und einfach IP-Pakete zu filtern und auszusortieren. Das Annehmen und Weiterleiten von Paketen kann wie bei Drawbridege durch Filterregeln definiert werden, die durch folgende Verbindungscharakteristika definiert sind:

- IP-Adresse
- Port-Nummer
- IP-Flags
- Verbindungsrichtung.

Außerdem bietet Linux die Möglichkeit, als Bestandteil des Firewall-Modules mit kleinen Erweiterungen als transparenter Proxy-Server zu arbeiten, so daß keine Veränderungen auf Appilikationsebene vorgenommen werden müssen. Dies kann durch einen ähnlichen Regelsatz erreicht werden.

3.2.1 ipfwadm - Ein Linux-Paketfilter

Die Benutzerschnittstelle, mit der die Filterregeln für den Kernel definiert werden, besteht hauptsächlich aus zwei Teilen:

- Firewall-Regeln können mit Hilfe des *setsockopt*- Systemcalls generiert und gelöscht werden. Mit dem selben Mechanismus kann das Default-Verhalten eines Filters verändert werden. Details hierzu sind in der Manualpage von *ipfw* zu finden.
- Die festgelegten Filter können geprüft werden, indem man die folgenden pseudo-Dateien im */proc* Filesystem ausliest:

  /proc/net/ip_input
  /proc/net/ip_output
  /proc/net/ip_forward

  Jede dieser Dateien beschreibt das Default-Verhalten, gefolgt von den definierten abweichenden Regeln für den entsprechenden Filter, in einem kompakten Format.

Das *ipfwadm*-Kommando ist ein Komanndozeilen-Interface zum Verwalten der Linux-Firewall-Optionen. Es kann dazu benützt werden, die Kernel-Filterregeln anzuzeigen oder zu verändern.

3.2.2 Installation

Ein linux-tauglicher Rechner mit zwei Netzwerkinterfaces wird benötigt. Dabei kann es sich durchaus um zwei verschiedene Schnittstellen (z.B. seriell und Ethernet) handeln.

3.2.2.1 ipfwadm

Ipfwadm ist Bestandteil jeder Linuxdistribution. Sollte es einmal nicht vorhanden sein, so kann es über die oben genannte Quelle bezogen werden. Da unser Linux-System ausreichend jung war, blieb uns das Compilieren erspart.

3.2.2.2 Linux Kernel -Parameter

Zunächst muß dem Kernel beigebracht werden, Firewalling zu unterstützen. Abbildung 3.1 zeigt die dafür erforderlichen Parameter, die in der Section „Networking Options“ bei der Konfiguration mit „make menuconfig“ angegeben werden müssen. Anschließend wird der Kernel wie gewohnt compiliert und gestartet. Hierfür ziehe man im Zweifelsfall die einschlägige Literatur zu Rate.

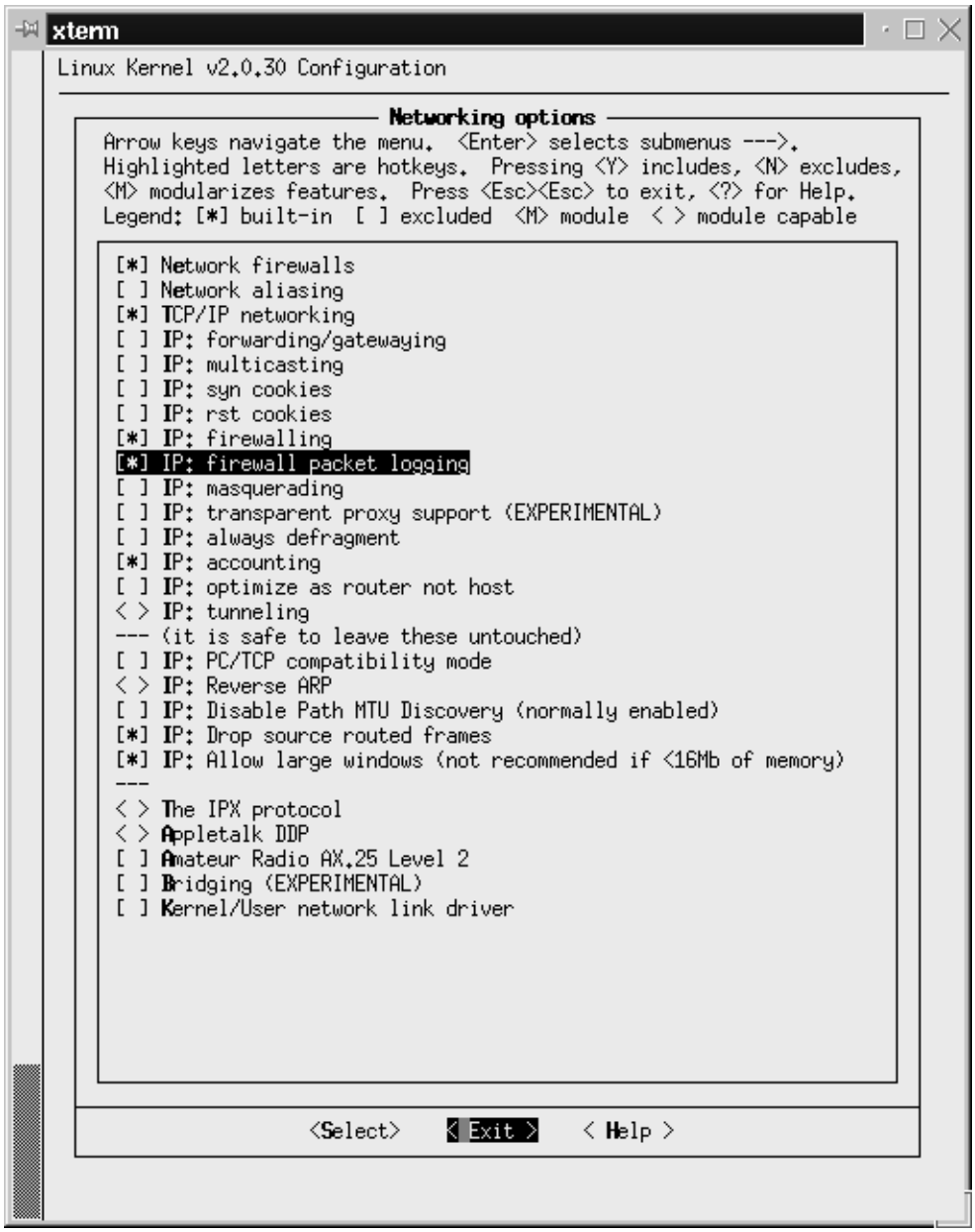

Abbildung 3.1: Linux Kernel-Parameter für Firewalling

3.2.3 Konfiguration

Konfiguriert wird am besten beim Booten vor dem Initialisieren der Netzwerkkarten, damit kein Paket dem Filter entwischt. Im Anhang ist im Kapitel A.4 ein Beispiel für ein Masquerade System V - Init-Skript zu finden, in das sich folgende Paradekonfiguration angepaßt einbinden läßt.

```
# Some definitions for easy maintenance.
LOCALHOST="gw.foo.com"
LOCALNET="192.168.37.0/24"
IFEXTERN="192.168.22.15"
IFINTERN="192.168.37.1"
ANYWHERE="any/0"
UNPRIVPORTS="1024:65535"
# ====== Basic rules.
# Sure we're paranoid, but are we paranoid enough?
ipfwadm -I -p deny
ipfwadm -O -p deny
ipfwadm -F -p deny
# Refuse spoofed packets.
ipfwadm -I -a deny -V $IFEXTERN -S $LOCALNET
ipfwadm -I -a deny -V $IFEXTERN -S $IFEXTERN
# Unlimited traffic within the local network.
ipfwadm -I -a accept -V $IFINTERN
ipfwadm -O -a accept -V $IFINTERN
# Unlimited ICMP traffic (not recommended).
ipfwadm -I -a accept -P icmp
ipfwadm -O -a accept -P icmp
ipfwadm -F -a accept -P icmp
# ====== External use of our system.
# Public access for e-mail, ftp, WWW, and DNS.
ipfwadm -I -a accept -P tcp \
        -D $LOCALHOST smtp ftp www domain
ipfwadm -I -a accept -P udp -D $LOCALHOST domain
ipfwadm -I -a accept -k -P tcp \
        -D $LOCALHOST ftp-data
ipfwadm -O -a accept -P tcp -S $LOCALHOST smtp ftp \
           ftp-data www domain
ipfwadm -O -a accept -P udp -S $LOCALHOST domain
# ====== Internal use of the Internet.
# Outgoing packets.
ipfwadm -O -a accept -P tcp -S $LOCALNET $UNPRIVPORTS \
        -D $ANYWHERE smtp ftp ftp-data www telnet gopher \
           z3950 domain
ipfwadm -O -a accept -P tcp -S $IFEXTERN $UNPRIVPORTS \
```

```
        -D $ANYWHERE smtp ftp ftp-data www telnet gopher \
           z3950 domain
ipfwadm -O -a accept -P udp -S $LOCALNET $UNPRIVPORTS \
        -D $ANYWHERE z3950
ipfwadm -O -a accept -P udp -S $LOCALHOST $UNPRIVPORTS \
        -D $ANYWHERE z3950 domain
ipfwadm -F -a accept -P tcp -S $LOCALNET $UNPRIVPORTS \
        -D $ANYWHERE ftp ftp-data www telnet gopher z3950
ipfwadm -F -a accept -P udp -S $LOCALNET $UNPRIVPORTS \
        -D $ANYWHERE z3950
# Incoming packets.
ipfwadm -I -a accept -k -P tcp \
        -S $ANYWHERE ftp www telnet gopher z3950 domain \
        -D $LOCALNET $UNPRIVPORTS
ipfwadm -I -a accept -k -P tcp \
        -S $ANYWHERE ftp www telnet gopher z3950 domain \
        -D $IFEXTERN $UNPRIVPORTS
ipfwadm -I -a accept -P tcp \
        -S $ANYWHERE ftp-data -D $LOCALNET $UNPRIVPORTS
ipfwadm -I -a accept -P tcp \
        -S $ANYWHERE ftp-data -D $IFEXTERN $UNPRIVPORTS
ipfwadm -I -a accept -P udp \
        -S $ANYWHERE z3950 -D $LOCALNET $UNPRIVPORTS
ipfwadm -I -a accept -P udp -S $ANYWHERE z3950 domain \
        -D $LOCALHOST $UNPRIVPORTS
ipfwadm -F -a accept -k -P tcp \
        -S $ANYWHERE ftp www telnet gopher z3950 \
        -D $LOCALNET $UNPRIVPORTS
ipfwadm -F -a accept -P tcp \
        -S $ANYWHERE ftp-data -D $LOCALNET $UNPRIVPORTS
ipfwadm -F -a accept -P udp \
        -S $ANYWHERE z3950 -D $LOCALNET $UNPRIVPORTS
```

3.2.4 Parameter

Die Optionen, die von *ipfwadm* angenommen werden, können in verschiedene Gruppen unterteilt werden. Folgende Kombinationen sind gültig:

ipfwadm -A Kommando Parameter [Optionen]
ipfwadm -I Kommando Parameter [Optionen]
ipfwadm -O Kommando Parameter [Optionen]
ipfwadm -F Kommando Parameter [Optionen]
ipfwadm -M -l [options]

3.2.4.1 Kategorieen

-A [richtung]: Beschreibt die Accounting-Regel. Optional kann eine Richtung (Schlüsselwörter „in“, „out“ oder „both“) angegeben werden.

-I: Firewall-Regeln für eingehende Verbindungen

-O: Firewall-Regeln für ausgehende Verrbindungen

-F: Forwarding Regeln

-M: IP-Masquerading Administration. Diese Kategorie wird nur mit dem Switch **-l** unterstützt.

3.2.4.2 Kommandos

-a [Regel]: *append*– hängt eine oder mehrere Regeln an das Ende der ausgewählten Liste an.

-i [Regel]: *insert*– fügt eine oder mehrere Regeln am Anfang der ausgewählten Liste an.

-d [Regel]: *delete*– Löscht einen oder mehrere Regel-Einträge aus der ausgewählten Liste an.

-l: *list*– zeigt alle aktuellen Regel-Einträge der ausgewählten Liste an. Dieses Kommando kann mit **-z** (Zurücksetzen der Zähler auf Null) kombiniert werden. Das geschieht unmittelbar nach dem Auflisten der momentanen Werte. Sofern die Option **-x** nicht angegeben ist, werden die Paket- und Byte-Zähler als *anzahl* K und *anzahl* M angezeigt, wobei 1K 1000 bedeutet und 1M 1000K (gerundet zum nächsten Integer-Wert). Mehr dazu bei den Optionen **-e** und **-x**.

-z: *zero*– Zurücksetzen der Paket- und Byte-Zähler der ausgewählten Liste auf Null. Dieses Kommando kann mit **-l** kombiniert werden.

-f: *flush*– löscht die ausgewählte Regelliste.

-p [Regel]: Ändert die Default-Regel für den ausgewählten Firewall-Typ. Die übergebene **Regel** muß entweder *accept, deny* ode *reject* lauten. Die Default-Regel kommt zum Tragen, wenn keine spezielle Regel gefunden wird. Dieses Kommando ist nur gültig für IP-Firewalls, d.h. in Kombination mit der **-I**, **-O** oder **-F** Kategorie.

-c: *check*– Prüft, ob das IP-Paket vom ausgewälten Firewall-Typ angenommen (accept), verboten (deny) oder zurückgewiesen (reject) wird. Dieses Kommando ist nur gültig für IP-Firewalls, d.h. in Kombination mit der **-I**, **-O** oder **-F** Kategorie.

-h: *help*– gibt eine kurze Beschreibung der Kommando-Syntax aus.

3.2.4.3 Parameter

Die folgenden Parameter können in Kombination mit den *append*, *insert*, *delete* oder *check* - Kommandos verwendet werden:

-P Protokoll: Das Protokoll der Regel oder des Paketes, das geprüft werden soll. Das angegebene Protokoll kann entweder *tcp*, *udp*, *icmp* oder *all* sein. Das Protokoll *all* beinhaltet alle Protokolle und ist der Standardwert, wenn dieser Parameter angegeben wird. *All* darf nicht zusammen mit *check* verwendet werden.

-S Adresse[/mask] [port ...]: Source – Angabe der Quelladresse (Pflicht). Die **Adresse** kann entweder ein Rechnername, ein Netzwerkname oder nur eine IP-Adresse sein. Die *mask* kann entweder eine Netzwerkmaske oder nur eine Zahl sein, die die Anzahl der Einsen auf der linken Seite der Netzwerkmaske angibt. So ist zum Beispiel 24 gleichbedeutend mit 255.255.255.0.

Die Quelle kann eine oder mehrere Port- oder ICMP- Angaben beinhalten. Gültig sind ein service-Name, eine Port-Nummer oder ein (numerischer) ICMP-Typ. Im restlichen Absatz bedeutet *port* entweder die Port-Nummernangabe oder den ICMP-Typ. Diese Angabe kann einen Bereich von Portnummern bedeuten und wird dann mit *port* :*port* spezifiziert. Außerdem darf die Anzahl der ports, die mit der Quell- und Ziel-Adresse übergeben werden, nicht größer sein als **IP_FW_MAX_PORTS** (momentan 10). Ein Bereich von Portnummern zählt als zwei Ports.

Pakete, die nicht als erstes Fragment eines TCP-, UDP- oder ICMP-Paketes ankommen, werden von der Firewall akzeptiert. Zum Zählen (accounting) werden diese zweiten und weiteren Pakete besonders gehandhabt. Die Portnummer 0xFFFF (65535) wird zum Zählen von zwiten und weiteren Fragmenten von TCP- und UDP-Paketen verwendet. Diese Paktet werden behandelt, wie wenn beide Portnummern 0xFFFF gewesen wären. Die Portnummer 0xFF (255) wird für Fragmente von ICMP-Pakten verwendet. Ports dürfen nur in Kombination mit mit dem *tcp*, *udp* oder *icmp* Protokoll angegeben werden. Wenn das check-Kommando angegeben wird, darf nur genau ein Port angegeben werden.

-D Adresse[/mask] [port ...]: Destination – Ziel-**Adresse**. Eine detaillierte Beschreibung der Syntax ist unter **-S** (source) zu finden. ICMP-Typen dürfen nicht im **-D** Flag angegeben werden: sie dürfen nur nach dem **-S** Flag verwendet werden.

-V Adresse: Optionale **Adresse** der Netwerkkarte, über das ein Paket empfangen wird oder über das ein Paket gesendet wird. Die **Adresse** kann entweder ein Rechnername oder nur eine IP-Adresse sein. Wen ein Rechnername angegeben wird, so sollte er zu genau einer IP-Adresse aufgelöst werden können. Wenn diese Option ausgelassen wird, so wird die **Adresse** 0.0.0.0 angenommen, was eine spezielle Bedeutung hat und alle Netzwerkkarten beschreibt. Für das *check*-Kommando ist dieser Parameter Pflicht.

-W name: Optionaler **Name** der Netwerkkarte, über das ein Paket empfangen wird oder über das ein Paket gesendet wird. Wenn diese Option ausgelassen wird, so wird eine leere Zeichenfolge angenommen, was eine spezielle Bedeutung hat und alle Netzwerkkarten beschreibt.

3.2.4.4 Optionen

Die folgenden zusätzlichen Optionen können angegeben werden:

-b: Bidirektionaler Modus

-e: erweiterte Ausgabe

-k: TCP-Pakete mit ACK-Bit

-m: Masquerade

-n: Numerische Ausgabe. Die IP-Adressen und Port-Nummern werden in numerischem Format ausgegeben. Normalerweise wird versucht die Adressen in Rechnernamen, Netzwerknamen und Servicenamen aufzulösen (soweit möglich).

-o: Kernel-Logging

-r port: Redirect pakets.

-v: verbose

-x: Expand Numbers.

3.2.5 Erfahrungsbericht

Aufgrund der Tatsache, daß der Paketfilter auf einem UNIX-System läuft, ist es möglich, das Logging auf dem gleichen Rechner zu starten. Man kann also den Paketfilter zu einem kompletten Firewall-System erweitern.

Damit die aufgestellten Regeln keine ungewollten Pakete durchlassen muß das aktivieren der Regeln *vor* dem Konfigurieren der Netzwerkinterfaces geschen. Ein System V Startup-Skript ist im Anhang im Kapitel A.4 zu finden.

Unseren Erfahrungen zufolge ist das Ansprechen der Hardwareadressen der Netzwerkinterfaces unter UNIX weit weniger problematisch als bei der Drawbridge unter DOS.

3.2.6 Bewertung

Für kleine Netze ist *ipfwadm* als Paketfilter ohne Probleme geeignet. Für große Netze fehlen uns Referenztests. Obwohl die Möglichkeit besteht, das Logging auf dem gleichen Rechner laufen zu lassen, ist dies trotzdem nur dann sinvoll, wenn aus Kostengründen Hardwaremangel besteht. Fällt der Rechner nämlich in „Feindeshand“, so ist auch das aktuelle Logging gefährdet und der Angreifer kann nicht ausfindig gemacht werden.

Kapitel 4

Daemon-Programme

4.1 logdaemon

Christian Rotter

Quelle:
ftp://ftp.cs.uni-sb.de/pub/net/logdaemon-5.6.tgz

Die bei den meisten Unix-Plattformen mitgelieferten Server für diverse Netzdienste sind nicht besonders kommunikativ, wenn es darum geht, Fehlermeldungen oder Einbruchsversuche zu protokollieren. Das LogDaemon-Paket ersetzt die wichtigsten Serverprogramme durch wesentlich gesprächigere Versionen. Die am besten getestete Plattform ist Solaris, aber auch die Unix-Derviate von HP, SGI und DEC werden unterstützt, wenn auch nicht komplett. Das Paket wurde nur sehr kurz getestet, da die Funktionalität nicht mehr den aktuellen Anforderungen entspricht und durch die Verwendung von Proxy- und Verschlüsselungs-Systemen der gleiche Effekt erzielt werden kann, ohne Systemprogramme zu ersetzen und dadurch neue Probleme bei Supportfragen aufzuwerfen.

4.1.1 Funktionsbeschreibung

Das Paket ersetzt folgende System-Serverprozesse:

- rshd
- rlogind
- ftpd
- rexecd
- login
- telnetd

Zusätzlich sind noch einige Besonderheiten im Paket enthalten, z.B. Unterstützung für S/Key und das Digital Pathways SecureNet Key Card System, das jedoch in Deutschland nicht sehr gebräuchlich ist und deshalb nicht weiter getestet wurde.

Alle Ersatzserver verfügen über erweiterte Funktionalität für Zugriffskontrolle und Logging. Die bekannte Schwächen von *rlogin* und *rsh* werden jedoch nicht behoben, so daß die Verwendung weiterhin gefährlich bleibt. Für weitere Information sei auf die Man-Pages in den jeweiligen Verzeichnisse verwiesen.

4.1.2 Installation

Da *logdaemon* ein älteres Paket ist, wird noch ein normales Makefile verwendet, was natürlich etwas mehr Erfahrung erfordert als der *autoconf*-Mechanismus der anderen Pakete. Auch nach erfolgter Übersetzung der einzelnen Server muß man zur Installation viele Arbeitsschritte manuell ausführen, so daß man schon umfangreiche Kenntnisse über sein Betriebssystem haben sollte, um das Paket erfolgreich anwenden zu können. Ich verweise hier auf die zahlreichen vorhandenen README-Dateien und Man-Pages des Pakets, anstatt einfach alles abzuschreiben. Hingewiesen sei aber auf die möglichen Anpassungen in den jeweiligen Makefiles, um erweiterte Optionen zu aktivieren.

4.1.3 Bewertung

In der nur kurzen Testphase wurde lediglich die Logging-Funktionalität untersucht. Je nach Serverprozess werden via *syslogd* Meldungen abgesetzt, die dann mit geeignete Werkzeugen (siehe Kapitel 6.1) aufbereitet werden müssen. Vor allem bei Systemen, wo Resourcen knapp sind, kann man mit *logdaemon* die Sicherheit verbessern, allerdings gibt es heutzutage wohl elegantere Lösungen, die leichter zu installieren und zu konfigurieren sind. Die bekannte Schwächen der BSD „r“-Befehle (siehe [5][S.240 ff]) werden nicht vollständig behoben, weshalb man für diesen Bereich SSH (siehe Kapitel 5.2) den Vorzug geben sollte.

4.2 sfingerd

Jürgen Mayerhofer

Quelle:

ftp://ftp.cert.dfn.de/pub/tools/net/sfingerd/sfingerd-1.8.tar.gz

Der auf den meisten UNIX-Systemen enthaltene Finger-Daemon gibt dem anfragendem Finger-Prozeß Auskunft über die Anzahl der momentan angemeldeten Benutzer, den Benutzernamen, dessen Account und das Datum des letzten Logins. Diese Informationen können dazu verwendet werden, wenig oder garnicht benutzte Kennungen aufzuhacken um so ins System zu gelangen. Besonders auf Rechnern, die eine große Anzahl von Accounts verwalten ist es schwierig für den Administrator herauszufinden,

welcher Login rechtmäßig stattgefunden hat und welcher nicht. Auf solchen Rechnern soll vermieden werden, den Hackern einen Ansatzpunkt durch Finger-Informationen zu bieten. Ein unsicherer Fingerdaemon liefert z.B. folgende Auskunft:

```
jma@jmpc01:/devel/temp/da/lyx > finger rvogl@localhost
[localhost]

Welcome to Linux version 2.0.30 at jmpc01.fh-regensburg.de !

 11:51am  up  1:30,  5 users,  load average: 0.14, 0.47, 4.39

Login: rvogl                            Name: Ruediger Vogl
Directory: /home/rvogl                  Shell: /soft/bin/bash
Last login Thu Apr 17 16:27 (GMT) on ttyp7 from birdy.fh-regens
Mail last read Wed Jul  9 09:57 1997 (GMT)
No Plan.
jma@jmpc01:/devel/temp/da/lyx >
```

sfinger von Laurent Demailly unterbindet die Ausgabe kritischer Informationen. Es ist mit *sfingerd* nicht mehr möglich eine Anfrage auf alle Benutzer auf dem Zielsystem mittels „finger @hostname“ zu stellen, lediglich die Benutzerinformationen für einzelne Benutzer sind erlaubt.

4.2.1 Installation

Die Installation erfolgt durch Compilieren des C-Sourcecodes. Anpassungen werden im Sourcefile „sfingerd.c“ getroffen. Die wichtigste Anpassung betrifft den Eintrag „CHROOT_PATH“. Das Kontaktieren des *sfingerd* bewirkt, daß Anfragen nicht mehr aus den Informationen des Homedirectories gelesen werden, sondern aus dem Verzeichnis der CHROOT-Umgebung.

Nach dem erfolgreichen Compilieren muß der Administrator den neuen *fingerd* ins System integrieren. Er wird an die vom Autor vorgeschlagene Stelle in das Verzeichnis „/usr/local/etc/“ kopiert. Die Anfrage wird über *inetd* gestellt. Es muß somit ein Eintrag in „/etc/inetd.conf“ vorgenommen werden:

```
finger       stream tcp nowait root /usr/local/etc/fingerd  fingerd
```

Es ist zu beachten, daß der neue *fingerd* als „root“ laufen muß, damit er den „chroot()“-Aufruf durchführen kann. Der *inetd* wird nun dazu gezwungen seine Konfigurationsdatei „/etc/inetd.conf“ neu z7u lesen. Es wird die Prozeß-ID des *inetd* ermittelt:

```
root@jmpc01:/usr/local/etc/fingerdroot # ps -ef | grep inetd
   96  ?  S     0:00 /usr/sbin/inetd INIT_VERSION=sysvinit-2.60f2
```

Abbildung 4.1: sfinger - Keine sicherheitskritischen Informationen

```
root@jmpc01:/usr/local/etc/fingerdroot #
```

Unser Ergebnis war also „96“. Die Option „ps -ef“ gehört hier zu einem SVR4-System. Unter einem BSD-System müßte „ps -aux“ aufgerufen werden. Dem *inetd* wird nun ein „HUP“-Signal geschickt. Der Parameter heißt also „-HUP“ oder „-1“:

```
root@jmpc01:/usr/local/etc/fingerdroot # kill -1 96
```

Eine Anfrage an den User „beispiel“ liest nun den Eintrag aus der Datei „/usr/local/etc/fingerdroot/beispie
Damit im „CHROOT_PATH“-Verzeichnis Dateien für jeden User mit deren User-ID erzeugt werden, hat Laurent Demailly im Quell-Paket zwei C-Shell-Skripten mitgeliefer. Das eine – „make-files“ – erzeugt die Dateien, das andere – „update-files“ – hält die Informationen auf dem neuesten Stand und kann als Cron-Job regelmäßig gestartet werden.

4.2.2 Test

- Eine Finger-Anfrage nach einem User ohne „.plan“-Datei ist in Abbildung 4.1 zu sehen.

- Zusätzlich wird ein Eintrag über „syslog“ angelegt:

  ```
  Jul 30 13:43:38 jmpc01 fingerd: 127.0.0.1 [localhost] : 'cmetten'
  ```

- Ein Versuch, die Liste aller angemeldeten Benutzer zu erfragen wird von *sfingerd* mit der Meldung

  ```
  jma@jmpc01:/home/jma > finger @localhost
  ```

```
[localhost]
sorry, generic fingerd is not running on this host.
try finger <username>@... for user specific informations
jma@jmpc01:/home/jma >
```

zurückgewiesen.

4.2.3 Bewertung

Die Angriffspunkte, die „finger“ liefert, werden mit *sfingerd* erfolgreich blockiert. Laurent Demailly weist daruf hin, daß sein *sfingerd* möglicherweise nicht den RFC-Ansprüchen genügt. Sein Ziel war es jedoch, einen überschaubaren, sicheren Finger-Daemon zu realisieren, was ihm mit 197 Zeilen C-Source (davon ca. 50 Zeilen Kommentar) sicherlich gut gelungen ist. Ein beliebtes Sicherheitsloch kann so gestopft werden.

4.3 rfingerd

Christian Rotter

Quelle:

ftp://ftp.cso.uiuc.edu/pub/security/coast/unix/rfingerd/rfingerd.tar.gz

Es gibt im Unix-Bereich diverse Dienste, die eigentlich eine sinnvolle Arbeitserleichterung darstellen, jedoch manchmal auch Sicherheitslücken beinhalten. Der *finger*-Dienst ist davon auch betroffen, er kann dazu verwendet werden, von Rechnern, die irgendwo ans Internet angebunden sind, diverse interessante Informationen zu erhalten, z.B. welche Benutzer gerade angemeldet sind und von wo aus die Anmeldung erfolgt ist. Aus diesen Hinweisen kann ein potentieller Eindringling einen günstigen Augenblick erkennen und dann aktiv werden, wenn gerade kein Administrator anwesend ist, der einen Einbruchsversuch schnell erkennen würde und Gegenmaßnahmen einleiten könnte. Zusätzlich ist der *finger*-Dienst relativ anonym, da Abfragen normalerweise nicht mitprotokolliert werden.

Ein kleines *perl*-Paket mit dem Namen *rfingerd* behebt diese bekannten Probleme. Durch die Verwendung von *perl* sind die beiden Programme des Pakets sehr kurz (wenige KByte) und sehr leicht an die eigenen Bedürfnisse anzupassen.

4.3.1 Installation

Das Entpacken des Archivs funktioniert auf gleiche Art und Weise wie in Kapitel 2.3.2 beschrieben.

Da es sich um ein *perl*-Skript handelt, entfallen die Konfiguration und Compilation, jedoch ist ein installiertes *perl* (ab Version 4.036) erforderlich. Man kann bei Bedarf noch ein paar Pfade anpassen, was mit jedem Editor problemlos möglich ist.

Im Normalfall protokolliert *rfinger* seine Aktionen in der Logdatei namens */var/log/trap/fingerd*.

Die eigenliche Installation besteht darin, den *fingerd* des Betriebssystems durch den *fingerd* des Paketes zu ersetzen und *authuser* nach z.B. */usr/libexec/* zu kopieren (je nach Einstellung in *fingerd*). Dann sollte man noch das Verzeichnis für die Logdatei anlegen und mit den entsprechenden Zugriffsrechten versehen. Nach einem kurzen Testlauf ist *rfinger* dann einsatzbereit. Da der *finger*-Dienst vom *inetd* gestartet wird, sind keine Einträge in System-Konfigurationsdateien/Batchdateien erforderlich.

4.3.2 Bewertung

Durch die Verwendung von *perl* ist der Speicherplatzbedarf sehr gering (sofern man nicht zuerst *perl* installieren muß !). Der eher anfällige *fingerd* des Betriebssystems wird durch eine leistungsfähigere und leichter erweiterbare Version ersetzt, die bekannte Schwächen behebt. Alles in allem ist *rfinger* ein Beweis dafür, daß leistungsfähige Utilities nicht mehrere MByte auf der Festplatte belegen müssen, um ihren Zweck zu erfüllen.

4.4 tcpd

Christian Rotter

Quelle:

ftp://ftp.uni-paderborn.de/SUGD/sol25sparc/security/

Einen anderen Ansatz als die bisher vorgestellten Programme verfolgt *tcpd*, es werden keine Systemprogramme durch neue Versionen ersetzt, sondern der *tcpd* wird über den *inetd* aufgerufen und startet dann (nach Verifizierung und Logging dieses Vorgangs) den eigentlich aufgerufenen Dienst. Der Einsatz ist jedoch auf System-Daemons beschränkt, die über den *inetd* gestartet werden, bei dauerhaft laufenden Prozessen kann dieser Ansatz nicht funktionieren. Diese Funktionalität ist bei jeder gängigen Firewall-Software auch vorhanden und wird auch als „TCP-Wrapping“ bezeichnet, da der jeweilige Dienst über eine Zwischenschicht, in die er „eingewickelt“ ist, aufgerufen wird.

4.4.1 Installation

Der Wrapper *tcpd* ist im Internet in verschiedenen Ausführungen verfügbar:

- Als Sourcecode, was durch den verwendeten *autoconf*-Mechanismus einfach installierbar, aber auf Dauer doch etwas eintönig ist, da fast alle getesteten Pakete dieser Diplomarbeit dieser Philosophie folgen.

- Als Alternative zum Standardablauf wurde ein vorkompiliertes Paket verwendet, das über den Package-Mechanismus von Solaris installiert werden kann. Dazu wird das Archiv zunächst entpackt (siehe auch Kapitel 2.3.2) und danach mit

  Prompt> pkgadd -d <zielverzeichnis>

  die Installation gestartet, die dann wie gewohnt textmenügesteuert abläuft und deshalb nicht weiter behandelt wird.

Dann kann man beginnen, *tcpd* zu konfigurieren und sein System sicherer zu machen.

4.4.2 Funktionsbeschreibung und Konfiguration

Die Funktionalität ist in mehrere Bereiche zu untergliedern, wobei jeder Bereich unterschiedlich konfiguriert wird:

- **Logging**

  Alle Log-Vorgänge werden über den *syslog*-Mechanismus abgewickelt, weshalb eine Konfiguration nicht erforderlich ist. Man sollte nur wissen, wo *syslogd* seine Informationen abspeichert (siehe [7]), um dann seine Schlüsse ziehen zu können.

- **Zugriffskontrolle**

  Man kann *tcpd* für eine einfache Zugriffskontrolle, die auf Mustererkennung beruht, konfigurieren. Ähnlich wie bei *swatch* (siehe Kapitel 6.2) können auch hier Aktionen definiert werden, wenn ein bestimmtes Muster auftritt. Die Konfiguration erfolgt über Textdateien. Dies ist in [9] sehr ausführlich erläutert, so daß hier nicht weiter darauf eingegangen wird.

- **Überprüfung von Rechnernamen**

  Der normale Authentifizierungsvorgang von *rhosts*-basierten Diensten verwendet als Zugangsberechtigung Rechnernamen, was von findigen Spezialisten sehr leicht für Angriffsversuche ausgenutzt werden kann, indem einfach ein passender Rechnername vorgetäuscht wird. Bei *tcpd* wird zusätzlich noch der Rechnername (der aus der IP-Adresse ermittelt wird) verwendet, um wiederum die zugehörige IP-Adresse vom DNS anzufordern. Diese Adresse sollte natürlich der ersten entsprechen, deshalb wird dieser Vergleich jedes Mal durchgeführt.

  Je nach den verwendeten Compileroptionen kann auf einen fehlgeschlagenen Vergleich mit Verbindungsabbruch (-DPARANOID) oder mit einer besonderen Aktion, ausgelöst durch die PARANOID-Regel, reagiert werden.

- **Abfangen von source-routing Paketen**

  Durch weitere Optionen beim Compilieren kann das im TCP/IP-Protokollstack vorhandene *source routing* deaktiviert werden. Dadurch wird verhindert, daß durch ausgeklügelte IP-Paketheader der vorhandene Sicherheitsmechanismus umgangen werden kann.

- **Benutzeridentifizierung**

  Durch eine letzte Compileroption kann noch die Benutzeridentifizierung aktiviert werden, die nach RFC 931 (siehe [12]) abläuft. Das Problem ist hierbei, daß auf Client-Seite ein entsprechender Daemon (z.B. *identd*) aktiv sein muß, was nicht immer der Fall ist. Außerdem versagt dieser Mechanismus bei UDP/IP-basierten Diensten und bei den meisten nicht Unix-basierten PCs.

4.4.3 Testlauf

Besonders ausführlich wurde *tcpd* nicht getestet, da der Ablauf und das Ergebnis ohnehin immer identisch gewesen wären:

- Es gibt Dienste, bei denen *tcpd* wegen seiner Arbeitsweise nicht funktionieren kann, dazu gehören alle Daemons, die nach dem Aufruf noch eine gewisse Zeit aktiv bleiben, um eventuelle Neuanfragen noch zu beantworten. In so einem Fall können nicht alle Verbindungen protokolliert werden, da nur der erste Verbindungsaufbau über den *inetd* und damit über *tcpd* vorgenommen wird.

  Eine andere Problemgruppe sind ständig laufende Server-Dienste, z.B. moderne HTTP-Server, die aber häufig die benötigte Funktionalität bereits implementiert haben.

- Bei anderen Diensten (siehe Bewertung) war die Funktion nach erfolgter Konfiguration einwandfrei.

4.4.4 Bewertung

Für die Absicherung von *telnet*, *finger*, *ftp*, *exec*, *rsh*, *rlogin*, *tftp*, *talk*, *comsat* (die gängigsten wurden kurz getestet) und einigen weiteren über *inetd* gestarteten Diensten bietet *tcpd* eine Verbesserung in bezug auf Zugriffskontrolle und Logging. Die einfache Konfiguration und das funktionelle Konzept sind die Vorteile von *tcpd*, welche von den Nachteilen (funktioniert nicht für alle Dienste, einige Probleme bei der Adreßauflösung) nicht in den Hintergrund gedrängt werden können. Für kleinere Anwendungsbereiche mit seltenen Änderungen ist *tcpd* durchaus zu empfehlen, für größere Netzwerke mit vielen Servern und dynamischen Anforderungen ist eine vollwertige Firewall eine kostspielige, aber bessere Lösung, da der Administrationsaufwand in diesem Umfeld sehr hoch wäre.

Kapitel 5

Verschlüsselungssysteme

5.1 PGP

Christian Rotter

Quelle:

http://www.ifi.uio.no/pgp/

In der heutigen Welt kommt niemand auf die Idee, wichtige Informationen wie Preisabsprachen oder Veröffentlichungstermine auf eine Postkarte zu schreiben und in den nächsten Briefkasten zu werfen. Um die Informationen zu schützen, wird man zumindest einen Brief verwenden, um unbefugtes Lesen zu verhindern. Auch bei Telefongesprächen wünscht man sich oft, keinen unerwünschten Mithörer zu haben.

Der allgemeine Wunsch nach Privatsphäre ist also allgegenwärtig, nur im Bereich der weltweiten Computernetze war eine solche Abschirmung nicht so einfach zu erreichen. Das ist natürlich ein Problem, denn wenn alle Daten im Klartext übertragen werden, ist es beim Internet besonders leicht, aus übertragenen Datenpaketen Informationen zu extrahieren und vielleicht gewinnbringend einzusetzen. Durch die unvorhersagbaren Wege, die ein Paket im Internet über diverse Router und andere Netzwerkkomponenten einschlägt, ist zudem eine Aufdeckung von unsicheren Übertragungswegen erschwert oder überhaupt nicht möglich. Deshalb versuchte man schon länger, die Daten vor der Übertragung zu verschlüsseln, was allerdings häufig an den nötigen Voraussetzungen (z.B. sichere Übertragung des Schlüssels) scheiterte. Einen Ausweg bieten Systeme mit öffentlichen Schlüsseln, die den Anforderungen eines weltweiten Netzwerkes gewachsen sind.

5.1.1 Vorwort: Kryptosysteme mit öffentlichen Schlüsseln

Bei herkömmlichen Verschlüsselungssystemen ist das Übertragen der Botschaft (Klartext) kein allzu großes Problem, jedoch muß der zugehörige Schlüssel, der zum Dekodieren erforderlich ist, auf einem sicheren Kanal (bzw. Übertragungsmedium) transferiert werden, was bei einigen bis mehreren Kommunikationspartnern realisierbar ist,

nicht jedoch bei einer breiten Anwendergruppe, die z.B. im Internet vorliegt (Electronic Mail ist ja das Haupteinsatzfeld von Verschlüsselungssystemen im privaten Bereich). Das Problem der sicheren Schlüsselverteilung wurde durch Public-Key-Systeme (wie Kryptosysteme mit öffentlichen Schlüsseln auch genannt werden) gelöst, die zur Ver- bzw. Entschlüsselung auf einen ausgefeilten Mechanismus zurückgreifen, der in [1][S. 391 ff] ausführlich erläutert wird.

Als bekanntestes Kryptosystem mit öffentlichen Schlüsseln baut RSA (benannt nach den Entdeckern Rivest, Shamir und Adlemann) auf genau diesem Prinzip auf, und die Implementationen von PGP, die noch heute in den USA verwendet werden dürfen, benutzen die Funktionsbibliothek RSAREF der Firma RSADSI Inc., die auch das Patent für den RSA Algorithmus in den USA innehat. Die internationale Version von PGP benutzt statt dessen die MPILIB Bibliothek von Philip R. Zimmermann, dem Entwickler von PGP [2].

In Public-Key-Systeme wird der Schlüssel in zwei Teilschlüssel aufgespalten: den privaten (private) und den öffentlichen (public) Schlüssel. Wird nun eine Botschaft mit dem System übertragen, benötigt der Absender nur den öffentlichen Schlüssel des Empfängers. Der Empfänger kann dann mit seinem privaten Schlüssel die mit seinem öffentlichen Schlüssel kodierte Botschaft wieder lesbar machen. Vorausgesetzt, daß der Empfänger seinen privaten Schlüssel geheimhalten kann, ist kein anderer in der Lage, an ihn gerichtete Botschaften zu entschlüsseln (diese Aussage relativiert sich natürlich mit der Tatsache, daß mit entsprechendem Einsatz von leistungsfähigen Computern, fähigen Kryptoanalytikern und genügend Zeit wohl auch derartige Systeme „geknackt“ werden könnten).

Die öffentlichen Schlüssel können (ähnlich einem Telefonbuch) einer breiten Masse zugänglich gemacht werden, womit einer allgemeinen Verwendung nichts mehr im Wege steht. Im Internet haben sich bereits einige Public Key Server (für PGP) etabliert, die eine sichere Kontaktaufnahme mit Leuten, die man nie zuvor gesehen hat, ermöglichen.

5.1.2 PGP – Pretty Good Privacy

PGP in seiner Ursprungsform ist ein textbasiertes, portables Programm, das auf allen erdenklichen Plattformen eingesetzt werden kann. Die Ursprungsplattform war MS-DOS, was die teilweise kryptisch kurzen Dateinamen und Dateiendungen erklärt, bei der Portierung für die Unix-Welt kamen noch einige Eigenheiten hinzu, so daß die Benutzungsphilosophie von PGP für heutige Verhältnisse ausgesprochen antiquiert wirkt.

Allerdings hat diese Entstehungsgeschichte auch Vorteile:

- Alle Arbeitsschritte werden mit Kommandozeilenparametern gesteuert, was eine weitgehende Automatisierung der Abläufe möglich macht.
- Der Betrieb kann auch in einer Filterfunktion erfolgen, d.h. lesen von der Standardeingabe, schreiben auf die Standardausgabe, Fehlermeldungen auf den Stan-

dardfehlerkanal, was die Einbettung in andere Programme unter Unix sehr vereinfacht, da dieses Verhalten dort Standard ist.

Der Nachteil dieser Batch-Philosophie ist die Vielzahl von möglichen Optionen, die man sich meist nur schwer merken kann.

Im Normallfall wird man also vom Befehlsprompt des Betriebssystems aus mit PGP arbeiten und sich auf die am häufigsten gebrauchten Optionen konzentrieren, was nach einer gewissen Einarbeitung auch problemlos funktioniert. Eine andere Möglichkeit ist, PGP so in andere Programme einzubauen, daß die Verwendung fast transparent wird. Diese Möglichkeit wird nun näher erläutert.

5.1.2.1 Anwendungsbeispiel: Verwendung von PGP mit dem Mailprogramm Pine

Die oben beschriebene Arbeitsweise ist zwar effizient, aber in der heutigen Zeit der mausbedienten graphischen Benutzeroberflächen nicht mehr ganz zeitgemäß. Es gibt jedoch im Public Domain Bereich schon viele Programme, die direkt (ohne Benutzerintervention) mit PGP zusammenarbeiten. Aber auch in andere Programme kann meist leicht eine PGP-Unterstützung eingebaut werden (zumindest im Unix-Bereich). Als gängiger Vertreter eines frei verfügbaren Mailprogramms ist *pine* zwar nur textorientiert, jedoch verfügt es über eine ausgefeilte Benutzerführung und eignet sich sehr gut für zügiges Lesen der Email (vor allem über Modemzugänge). Die Einbindung von PGP in *pine* wird deshalb näher erläutert, wobei die Vorgehensweise auf den Sun-Pool 511 der Fachhochschule Regensburg (Fachbereich Informatik, Sammelgebäude) abgestimmt ist. Bei anders verwalteten Pools müssen einige Pfadangaben angepaßt werden; dann sollte alles funktionieren.

5.1.2.2 Konfiguration von PGP

Zunerst muß PGP aktiviert werden, man ruft /soft/bin/pgp_user_install (siehe Kapitel A.3) auf. Diese Batchdatei führt einige Arbeitsschritte aus, unter anderem werden Verzeichnisse angelegt, die für den sinnvollen Einsatz von PGP erforderlich sind.

Danach wird man aufgefordert, die gewünschte Schlüsselgröße auszuwählen. Entweder nimmt man eine der drei angebotenen Größen, oder man gibt eine Zahl zwischen 384 und 2048 ein, wobei ein gewisser Sicherheitsgrad ab 1024 erreicht wird.

Der Schlüssel, der nun erzeugt wird, muß eindeutig einer bestimmten Person zugeordnet werden können, deshalb muß man als nächstes Name und Email-Adresse angeben. Hier sollte man nicht scherzweise irgendeinen Blödsinn eingeben, sondern seine eigenen Daten, als da wären:

Vorname Nachname Email-Adresse, eingeschlossen in spitze Klammern

Ein Beispiel gefällig ?

Hans Dampf <hans.dampf@stud.fh-regensburg.de>

Anmerkung: Das Mail-System in Raum 511 (Sun-Pool) verwendet als Empfängernamen die Login-Kennung, nicht vorname.nachname !

Der korrekte Eintrag für den Raum 511 lautet also

Hans Dampf <hdampf@rfhs8012.fh-regensburg.de>

Nun kommt die Ersatzsicherung: Für den Fall, daß so ein böser Bube den Account aufhackt und herumschnüffelt, kodiert man seinen Schlüssel noch mit einem Paßwort. Bei der Wahl des Paßworts gelten ähnliche Regeln wie beim Login-Paßwort, weshalb ich einfach auf einen früheren Artikel von mir verweise (http://rfhs8012.fh-regensburg.de/ rotter/Troja/passwd.html). Das Paßwort muß auch noch bestätigt werden, so wie man das gewohnt ist.

Jetzt kriegt der Rechner mal Arbeit; man wird aufgefordert, ein bischen auf der Tastatur herumzutippen, um den Zufallszahlengenerator ab und zu neu zu initialisieren. Dabei wird eine Zahl gegen Null gezählt, man muß also nicht bis zur totalen Erschöpfung Daten eingeben...

Danach malt der Rechner ein paar Punkte auf den Bildschirm, die den Fortschritt der Berechnung anzeigen. Je mehr Bits man verlangt hat, desto länger dauert die Berechnung. Nach einer Weile ist die Generierung abgeschlossen, das Ergebnis landet im Verzeichnis „`~/.pgp`“.

5.1.2.3 Konfiguration von Pine

pine hat ein umfangreiches Konfigurationsmenü, in dem alle Einstellungen vorgenommen werden.

- Pine starten.
- Setup->Configuration aufrufen (zuerst **s**, dann **c** drücken)
- Nach unten blättern bis zum Eintrag display-filters.
- **a** drücken, um einen neuen Filter anzumelden und

  LEADING(”—BEGIN PGP”) /soft/bin/pgp_display

  als neuen Filter eingeben (die Unterstriche gehören dazu, nach dem zweiten ist ein Leerzeichen). Damit kann man schon einmal PGP-verschlüsselte Mails lesen (sofern der öffentliche Schlüssel des Absenders bekannt ist, das kommt später noch genauer). Zum Dekodieren muß immer das Paßwort mit angegeben werden, um auf den eigenen privaten Schlüssel zugreifen zu können. Nach dem Entschlüsseln einfach auf *Return* drücken, dann wird die Mail angezeigt.
- Weiter blättern bis zum Eintrag sending-filters.

- **a** drücken, um einen neuen Filter anzumelden und

  /soft/bin/pgp_sign -fast

  als neuen Filter eingeben. Damit kann man Mails mit seiner PGP-Unterschrift versehen.

- **a** drücken, um noch einen neuen Filter anzumelden.

  /soft/bin/pgp_encrypt -feast _RECIPIENTS_

  als neuen Filter eingeben (für die Unterstriche gilt das selbe wie oben). Damit kann man Mails verschlüsselt an die Empfänger senden. Dazu müssen natürlich deren öffentliche Schlüssel bekannt sein (mehr dazu siehe unten).

- **e** drücken, um das Konfigurationsmenü zu verlassen, und die gemachten Änderungen mit **y** bestätigen.

Nach einem Neustart von Pine kann man nun beim Versenden einer Mail mit **ctrl-n** und **ctrl-p** zwischen den Versand-Filtern umschalten, wobei (unfiltered) die bisher verwendete (nichts verändernde) Standard-Einstellung ist.

5.1.2.4 PGP und der Schlüsselbund für die Kommunikation

Wie schon oben erwähnt, gibt es für jeden PGP-Anwender gleich zwei Schlüssel:

- Den öffentlichen, den der Kommunikationspartner kennen muß, um auf die sichere Art kommunizieren zu können.

- Den privaten, der zum Entschlüsseln gebraucht wird (nur Verschlüsseln wäre ja auch etwas unsinnig, da könnte man ja den Text auch löschen).

PGP verwaltet öffentliche und private Schlüssel getrennt voneinander in sogenannten Key Rings (Schlüsselring, Schlüsselbund), die in Dateien abgespeichert werden, zusätzlich kann man solche Key Rings auf mehreren Dateien aufteilen, um die Übersicht zu behalten. Zur Verwaltung all dieser Key Rings stellt PGP eine Vielzahl von Funktionen zur Verfügung.

5.1.2.5 Begriffserläuterungen

In der folgenden Kurzübersicht werden PGP-Standard-Begriffe verwendet, um die Parameter zu erläutern. Die verwendeten Begriffe sind in der Tabelle 5.1 erklärt.

Bezeichnung	Funktion
key ring, Schlüsselbund	Datei zum Speichern von Schlüsseln
Schlüsselpaar	zusammengehörige öffentliche und private Schlüssel
<datei>	ein Dateiname, der jeweils passende Daten enthält
<userid>	gültiger PGP-Datensatz (Vorname Nachname <Email-Adresse>) oder ein eindeutiger Teil daraus
<own_userid>	die eigene <userid>
<foreign_userid>	eine fremde <userid>
finger print	eindeutiger Hash-Wert eines Schlüssels, zur Verifizierung
Vertrauensparameter	Verläßlichkeits-Status eines Schlüsselinhabers legt z.B. fest, ob man von der entsprechenden Person verifizierte Schlüssel als korrekt übernehmen will

Tabelle 5.1: PGP - Begriffserläuterungen

5.1.2.6 Kurzübersicht über die Befehle von PGP 2.6.3(i)

PGP hat eine Vielzahl von Kommandozeilenoptionen (oder Befehle), um diesen Schlüsselwust unter Kontrolle zu halten, die wichtigsten seien hier kurz erklärt:

-kg: Diese Option wurde am Anfang der PGP-Konfiguration verwendet, um ein einzigartiges Schlüsselpaar zu erzeugen. Diese Option sollte man eigentlich nur einmal benötigen, außer man benutzt mehrere private Schlüssel.

-ka <datei>: Es wird der Inhalt von <datei> in die Schlüsseldatei eingefügt, dies ist für neue Schlüssel erforderlich, damit man dem Schlüsselinhaber PGP-Mails zuschicken kann.

-kr <userid>: Die <userid> wird aus der Schlüsseldatei entfernt.

-ke <own_userid>: Ermöglicht Änderungen am eigenen Schlüsseleintrag (z.B. Paßwort).

-kxa <userid> <datei>: Extrahiert einen Schlüssel und speichert ihn in einer Textdatei ab (z.B. wenn man seinen öffentlichen Schlüssel auf der Homepage zum Download anbieten oder per Mail verschicken will), das **a** steht für ASCII, was für Mail-Versand besser ist.

-kv: Zeigt den Inhalt der Schlüsseldatei an.

-kvc <userid> <datei>: Anzeigen des „finger prints“ eines öffentlichen Schlüssels, um ihn z.B. über Telefon mit seinem Besitzer besser vergleichen zu können.

-kc <**userid**> <**datei**>**:** Anzeigen des Inhalts zur Überprüfung der Beglaubigungsunterschriften des eigenen öffentlichen Schlüsselbunds.

-ke <**userid**> <**private datei**>**:** Bearbeiten der Benutzer-ID oder der Verwaltungsdaten für den eigenen privaten Schlüssel.

-ke <**userid**> <**datei**>**:** Bearbeiten der Vertrauensparameter eines öffentlichen Schlüssels.

-kr <**userid**> <**datei**>**:** Entfernen eines Schlüssels oder nur einer Benutzer-ID aus dem eigenen öffentlichen Schlüsselbund.

-ks <**foreign_userid**> <**-u own_userid**> <**datei**>**:** Unterschreiben und Beglaubigen des öffentlichen Schlüssels von jemand anderem im eigenen öffentlichen Schlüsselbund.

-krs <**userid**> <**datei**>**:** Entfernen ausgewählter Unterschriften einer Benutzer-ID aus einem Schlüsselbund.

-kd <**own_userid**>**:** Dauerhaften Zurückziehen des eigenen Schlüssels durch Veröffentlichung einer „Schlüssel-Widerrufs-Urkunde“.

-kd <**userid**>**:** Sperren oder Freigeben eines öffentlichen Schlüssels im eigenen öffentlichen Schlüsselbund.

-e <**datei**> <**foreign_userid**>**:** Verschlüsseln eines Klartextes mit dem öffentlichen Schlüssel des Empfängers.

-s <**datei**> <**-u own_userid**>**:** Unterschreiben eines Klartextes mit dem eigenen privaten Schlüssel.

-es <**datei**> <**foreign_userid**> <**-u own_userid**>**:** Unterschreiben eines Klartextes mit dem eigenen privaten Schlüssel und anschließendem Verschlüsseln des Klartextes mit dem öffentlichen Schlüssel des Empfängers.

-c <**datei**>**:** Verschlüsseln eines Klartextes nur mit herkömmlicher Verschlüsselung (keine Verwendung öffentlicher Schlüssel, sondern gleicher Schlüssel zum Ver- und Entschlüsseln).

<**datei**> <**-o datei**>**:** Entschlüsseln einer verschlüsselten Datei oder um die Echtheit einer Unterschrift einer unterschriebenen Datei zu prüfen.

-e <**datei**> <**foreign_userid_1**> <**foreign_userid_2**> **...:** Eine Nachricht für beliebig viele Empfänger verschlüsseln.

-e <**datei**> <**foreign_userid_1**> **... -@** <**datei**>**:** Wird verwendet, um weitere Empfänger-IDs für eine Nachricht aus einer Datei einzulesen (z.B. für Mailinglisten etc.).

5.1.2.7 Selten gebrauchte Kommandos

-d <**datei**>**:** Entschlüsseln einer Nachricht, wobei die Unterschrift intakt bleibt.

-sb <**datei**> <**-u own_userid**>**:** Erstellen einer Unterschriftsbescheinigung, die vom unterschriebenen Dokument getrennt ist.

-b <**datei**>**:** Trennen einer Unterschriftsbescheinigung vom unterschriebenen Dokument.

5.1.2.8 Zusätzliche Optionen

-a: Zur Herstellung eines verschlüsselten Textes im ASCII-radix-64-Format einfach den Schalter -a anhängen, wenn ein Dokument verschlüsselt oder unterschrieben bzw. wenn ein Schlüssel entnommen wird.

-w: Zum Löschen durch Überschreiben der Klartextdatei („wipe out“) nach der Herstellung der verschlüsselten Datei oder dem Unterschreiben .

-t: Zur Festlegung, daß eine Klartextdatei ASCII-Text und keine Binärdaten enthält, und daß sie beim Empfänger gemäß seiner Textdarstellungskonventionen angezeigt werden soll, die Option -t (Text) an andere Optionen anfügen

-m: Zum Anzeigen des entschlüsselten Klartextes auf dem Bildschirm, ohne ihn in eine Datei zu schreiben, die Option -m (More) benutzen

-steam <**datei**> <**foreign_userid**>**:** Zur Festlegung, daß der entschlüsselte Klartext dem Empfänger nur auf seinem Bildschirm angezeigt wird und nicht auf Diskette gesichert werden kann, beim Verschlüsseln/Unterschreiben die Option -m anhängen

-p: Zum Wiederherstellen des Original-Dateinamens beim Entschlüsseln die Option -p verwenden

-feast <**foreign_userid**> < <**datei**> > <**datei**>**:** Um den Unix-ähnlichen Filterbetrieb zu benutzen (Lesen von der Standardeingabe und Schreiben auf die Standardausgabe), kann man die Option -f anfügen.

-h: Diese vielleicht wichtigste Option ruft die Hilfefunktion auf, mit der man schnell die komplexeren Funktionen nachschlagen kann

5.2 SSH

Christian Rotter
Quelle:
http://www.cs.hut.fi/ssh/

Will man weltweite Netzwerke gewinnbringend nutzen, ist es oft erforderlich, auf entfernten Computern Befehle auszuführen. Die dazu verwendeten Programme, die als BSD „r“-Befehle [5][S.240 ff] bekannt sind, haben jedoch diverse Sicherheitslücken und übertragen alle Daten vollkommen unverschlüsselt. Auch das weitverbreitete Telnet hat dieses Problem, und dadurch wird der Gebrauch solcher Programme zum Risiko.

SSH bietet eine Lösung für diese Problematik, denn jede Kommunikation wird über verschlüsselte Kanäle abgewickelt.

5.2.1 SSH – Secure Shell

SSH kann verwendet werden, um sich in andere Computer per Netzwerk einzuloggen, Befehle auszuführen oder Dateien auf entfernte Computer zu kopieren. Zusätzlich verfügt SSH über sichere Methoden zur Authentifizierung und zur Kommunikation über potentiell unsichere Übertragungswege im Netzwerk. Diese Eigenschaften lassen SSH zu einem brauchbaren Ersatz für diverse Programme (wie telnet, rlogin, rsh, rcp, rdist) werden.

Durch einen Schlüsselaustausch und die damit verbundene starke Authentifizierung bei jedem Verbindungsaufbau werden außerdem die bekannten Schwachstellen von Routing und DNS (Spoofing) verbessert.

SSH besteht aus diversen Client-Programmen und dem SSH-Server *sshd*, der beim Starten den Host Key (Schlüssel des Servers) aus einer Datei ausliest und damit sämtliche Möglichkeiten zur Verschlüsselung benutzen kann. Die internen Abläufe beim Aufbau von SSH-Connections sind in [3] genau erläutert, weshalb an dieser Stelle nicht weiter darauf eingegangen wird.

5.2.2 Installation

Auch SSH verwendet das *autoconf*-Paket von GNU, weshalb die Installation ebenso funktioniert wie bei Squid (siehe dort).

Zusätztlich gilt jedoch: Bei Verwendung von SSH in einem Netzwerk mit einem gemeinsamen Verzeichnis für ausführbare Dateien genügt es, nur auf einem Computer (idealerweise dem Server)

Prompt> make install

aufzurufen, auf allen anderen (meist den Client-Rechnern) ist ein

Prompt> make hostinstall

ausreichend. Beim hostinstall werden lediglich die erforderlichen Host Keys generiert und die Standard-Konfigurationsdateien installiert.

5.2.3 Konfiguration

Nach der Installation können bei Bedarf die Konfigurationsdateien angepaßt werden. Dem Unix-Standard folgend sind Zeilen, die mit # beginnen, Kommentare.

5.2.3.1 Begriffserläuterung

- ~ („tilde“)

  Eine gebräuchliche Abkürzung für $HOME, die von den Programmen csh (C shell), tcsh (Tenex C shell) und bash (bourne again shell) verwendet wird.

- Forwarding

  Unter Forwarding versteht man die Paketweiterleitung eines bestimmten Ports durch einen anderen Port auf den Zielrechner, der diesen Vorgang dann wieder umkehrt. Forwarding wird häufig verwendet, um unsichere Dienste über abgesicherte Kanäle laufen zu lassen.

- rhosts-Mechanismus

  Durch die Datei *~/.rhosts* (und einige andere wie */etc/hosts*) wird es ermöglicht, mit den BSD „r“-Befehlen [5][S.240 ff] Kommandos auf entfernte Computer abzusetzen, ohne zuerst ein Paßwort eingeben zu müssen. Dies ist zwar sehr angenehm für den Benutzer, allerdings hat dieser Mechanismus eine Menge Sicherheitslücken, die sehr leicht ausgenutzt werden können. Dabei wird meist die Tatsache, daß in der rhosts-Datei Computernamen und Benutzernamen stehen, ausgenutzt, indem dem jeweiligen Server einfach vorgegaukelt wird, man sei eben der Benutzer X auf Computer Y, der Zugriffsrechte besitzt. Besonders schwerwiegend ist, daß die meisten Unix-Neulinge häufig Fehler bei den Zugriffsberechtigungen von Dateien machen, so ist z.B. eine für Fremde beschreibbare *~/.rhosts*-Datei eine Einladung zum Herumstöbern in privaten Daten.

- RSA

  Gebräuchliches Verschlüsselungsverfahren (siehe [1], S. 391 ff und [2]) mit öffentlichen Schlüsseln.

- Syslog

  In System-5-Unix-Systemen wird für Logmeldungen der Syslog-Dienst verwendet. Durch Betriebssystemfunktionen wird ein einheitliches Format der Logdatei erreicht, um automatische Auswertungen zu ermöglichen.

- Syslog Facility

  Facility bedeutet in diesem Zusammenhang die Programmart, von der der Eintrag stammt. Dies dient lediglich der Übersicht im Syslog und der gezielten Auswertung der oft recht umfangreichen Logdateien.

- X, X Window System, X11

  X ist der Standard im Unix-Bereich für graphische Benutzeroberflächen. Das X Window System ist ein Client-Server System, wobei die X-Clients (die auf beliebigen Computern laufen können) alle benötigten graphischen Resourcen vom X-Server erhalten.

- X-Server

  An diesem Computer sind der Bildschirm und die Eingabegeräte angeschlossen, die die Schnittstelle zum Benutzer darstellen.

- X-Session

  Eine Verbindung von einem X-Client zu einem X-Server, die lokal oder über ein Netzwerk erfolgen kann. Das normale X Window System hat beträchtliche Sicherheitslücken in diesem Bereich, so daß fast alle Hersteller von Unix-Systemen besondere Mechanismen verwenden, um bekannte Probleme zu beheben.

5.2.3.2 sshd-Konfiguration

- Die verwendete Datei ist */etc/sshd_config*, dies kann mit der Kommandozeilenoption **-f** <**datei**> geändert werden. Es gibt noch weitere Kommandozeilenoptionen, die bei Verwendung die eingestellten Werte der Konfigurationsdatei überlagern. Da *sshd* im Normalfall bereits beim Booten des Computers gestartet wird und dabei die Konfiguration sinnvollerweise aus einer Datei liest, wird hier auf weitere Kommandoeilenoptionen nicht speziell eingegangen. Bei Bedarf kann man in [4] weitere Informationen erhalten.

- **AllowHosts [pattern] ...**

  Die DNS-Namen der Hosts, die zum SSH-Server eine SSH-Verbindung aufbauen können. Mehrere Namen werden durch Leerzeichen getrennt, die Verwendung von ***** und **?** als Jokerzeichen (wildcard) ist möglich. Normalerweise werden die hier angegebenen Namen über Nameserver aufgelöst, ist ein Server nicht im DNS, kann auch die IP-Adresse angegeben werden. In der Standardeinstellung ist der Zugriff für alle Rechner gestattet.

- **DenyHosts [pattern] ...**

  Wie bei AllowHosts, nur, daß hier angegebe Computer vom Zugriff ausgeschlossen sind.

- **FascistLogging [yes|no]**

  Hier kann mit **yes** excessives Logging eingeschaltet werden; dies verletzt die Privatsphäre der Benutzer und ist deshalb standardmäßig ausgeschaltet.

- **HostKey** <**datei**>

  In dieser Datei wird der private Schlüssel für den Server gespeichert; normalerweise ist das */etc/ssh_host_key*.

- **IgnoreRhosts [yes|no]**

  Man kann die Verwendung von rhost- und shost-Mechanismen (d.h. Einloggen ohne Paßwort) durch **yes** verbieten, **no** ist die Voreinstellung.

- **KeepAlive [yes|no]**

  Gibt an, ob Meldungen verschickt werden sollen, durch die man Verbindungsabbrüche und Systemcrashes von Client-Rechnern aufspüren kann. KeepAlive sollte aktiviert werden, um hängengebliebene Verbindungen sauber zu beenden und die Vergeudung von Serverresourcen zu vermeiden. Wird diese Option deaktiviert, müssen die Client-Programme ebenfalls entsprechend konfiguriert werden.

- **KeyRegenerationInterval** <**seconds**>

  Nach der eingestellten Sekundenzahl (oder nach **3600**, wenn diese Option fehlt) wird der Schlüssel des Servers neu generiert. Dies ist eine Sicherheitsmaßnahme, um mitgeschnittene Sitzungen später mit gestohlenen Schlüsseln nicht decodieren zu können.

- **LoginGraceTime** <**seconds**>

  Bei erfolglosen Einlogvorgängen wird nach der eingestellten Sekundenzahl (oder **600**, wenn diese Option fehlt) die Verbindung vom Server geschlossen.

- **PasswordAuthentication [yes|no]**

  Gibt an, ob die Authentifizierung mit Paßwörtern erlaubt ist. Sinnvollerweise ist hier **yes** die Voreinstellung, sonst wäre ein Einloggen nur möglich, wenn die shost-Datei (oder ein Äquivatent dazu) vorhanden wäre.

- **PermitEmptyPasswords [yes|no]**

  Gibt an, ob ein Einloggen in Accounts, deren Paßwort nicht gesetzt ist, erlaubt ist. Dies sollte auf jeden Fall unterbunden werden, also ist hier **no** einzutragen, obwohl die Normaleinstellung **yes** ist (das ist jedoch eine gewaltige Sicherheitslücke im System).

- **PermitRootLogin [yes|nopwd|no]**

  yes (Voreinstellung) gestattet dem Superuser ein Einloggen über SSH, **no** verbietet es, **nopwd** verbietet nur die Authentifizierung über Paßwörter, weshalb dann z.B. die shost-Datei eingesetzt werden muß.

- **PidFile** <**datei**>

  In dieser Datei (normalerweise */etc/sshd.pid* oder */var/run/sshd.pid*) steht die Prozeß-ID des *sshd*.

- **Port** <**port**>

  Der von *sshd* verwendete Port; im Normalfall **22**. Wird der Port geändert, muß man natürlich auch die Client-Programme anpassen.

- **PrintMotd [yes|no]**

  Gibt an, ob */etc/motd* (message of the day) bei interaktiven Einloggvorgängen ausgegeben werden soll. Dies ist standardmäßig aktiviert, um Informationen über das System schnell für jedermann zugänglich zu machen.

- **QuietMode [yes|no]**

  Bei aktiviertem QuietMode werden keine Einträge ins Systemlog geschrieben (außer fatale Fehler), dieser Modus ist in der Grundeinstellung auf **no**, um eine Kontrolle von *sshd* zu ermöglichen.

- **RandomSeed** <**datei**>

  In dieser Datei steht der Initialisierungswert für den Zufallszahlengenerator, die Vorbelegung ist */etc/ssh_random_seed*.

- **RhostsAuthentication [yes|no]**

  Gibt an, ob eine Authentifizierung nur über den Computernamen (wie beim rhosts-Mechanismus) ausreichend ist. Da rhosts eine Sicherheitslücke darstellt, ist dieser Wert mit **no** vorgelegt, was man auch nicht ändern sollte.

- **RhostsRSAAuthentication [yes|no]**

  RhostsRSAAuthentication ist eine Kombination von rhosts mit einer RSA-gestützten Computer-Authentifizierung. Diese Kombination ist wesentlich sicherer als der reine rhosts-Mechanismus und sollte nach Möglichkeit verwendet werden. Die Voreinstellung ist deshalb **yes**.

- **RSAAuthentication [yes|no]**

  Wie RhostsRSAAuthentication, jedoch wird der zusätzliche rhosts-Mechanismus nicht verwendet.

- **ServerKeyBits** <**bitzahl**>

  Der Mindestwert ist 512, voreingestellt sind **768** Bit, was nach den RSA-Richtlinien [2] ausreichend sicher ist. Verwendet man größere Werte, sollte man beachten, daß dadurch die benötigte Rechenzeit stark ansteigt.

- **StrictModes [yes|no]**

  Vor dem Aufbau einer SSH-Sitzung kann geprüft werden, ob die am rhosts-Mechanismus beteiligten Dateien vor Fremdzugriffen geschützt sind. Da Neueinsteiger und andere Anfänger in dieser Richtung häufig Fehler machen, wird diese Option auf **yes** gesetzt, was man beibehalten sollte.

- **SyslogFacility [DAEMON | USER | AUTH | LOCAL(0-7)]**

  Unter diesem Stichwort tauchen Einträge von *sshd* im Syslog auf. Voreingestellt ist **DAEMON**.

- **X11Forwarding [yes|no]**

  Der einzig sinnvolle Eintrag ist **yes**, da **no** ohnehin sehr leicht umgangen werden kann. X11-Sessions können durch SSH getunnelt werden, was der Sicherheit von X11 sehr zuträglich ist, tut man das nicht, wird der ganze SSH-Mechanismus bei Verwendung von X-Clients unterlaufen und der gewonnene Sicherheitsvorsprung zunichte gemacht.

5.2.3.3 Client-Konfiguration

Die Konfiguration von Client-Programmen erfolgt in drei Stufen:

- Kommandozeilen-Optionen werden zuerst abgearbeitet.
- Danach werden die benutzerspezifischen Konfigurationsdaten aus der Datei *~/.ssh/config* gelesen.
- Am Ende kommen die systemweiten Standardeinstellungen aus der Datei */etc/ssh_config*.

Die Konfiguration hat jedoch einige Besonderheiten, die man kennen sollte:

- Jede Option wird beim ersten Auftreten gesetzt und in folgenden Stufen nicht mehr geändert, dies sollte man beachten, wenn man auf Fehlersuche ist. Außerdem sollten rechnerspezifische Optionen eher am Dateianfang und allgemeine Funktionen am Dateiende stehen, um unerwünschtes Überschreiben von Optionen zu vermeiden.
- Alle Konfigurationsdateien sind in einzelne Unterbereiche unterteilt, die jeweils aus einer Liste von Computernamen (oder auch Computernamen mit Jokerzeichen für mehrere Computer bzw. ganze Netzwerke) und den zugehörigen Einstellungen bestehen. Wichtig ist hierbei, daß der Hostname, der den jeweiligen Client-Programmen übergeben wird, für diese Einteilung benutzt wird, und nicht ein vom DNS (oder anderen Mechanismen) aufgelöster Computername.
- Bei allen Einträgen in die Konfigurationsdateien ist die Groß-/Klein-Schreibung von Bedeutung.

Die folgenden Optionen können angegeben werden:

- **Host** <**namensmuster** | **name** | **...**>

  Dieser Eintrag bedeutet, daß alle bis zur nächsten **Host**-Zeile folgenden Spezifikationen nur auf Comuter angewendet werdern sollen, die genau diesen **Namen** haben bzw. in das verwendete **Namensmuster** passen. Das **Namensmuster** darf die bekannten Unix-Jokerzeichen **?** und ***** in beliebiger Kombination enthalten.

  Ein ***** gilt für alle Computer und kann für global geltene Optionen verwendet werden.

  Wie bereits erwähnt, wird der als Argument an das Client-Programm übergebene Hostname verglichen, nicht ein in irgendeiner Form aufgelöster Name.

  Hinter **Host** können mehrere **Namen** bzw. **Namensmuster** folgen, diese sind durch Leerzeichen zu trennen.

- **BatchMode [yes|no]**

  Bei der Verwendung von Befehlsskripten und Batch-Jobs kann die interaktive Eingabe von Paßwörtern unterdrückt werden. Die Angabe von **yes** bewirkt dieses Verhalten. Man sollte allerdings sicherstellen, daß dann auch keine Paßwörter erforderlich sind, sonst wird wohl nicht alles wie gewollt funktionieren.

- **Cipher [idea | des | 3des | arcfour | tss | none]**

  Die zu verwendende Verschlüsselungsart ist auf **idea** voreingestellt (ersatzweise **3des**, wenn **idea** nicht verfügbar ist). Die Verwendung von **none** ist nur für Debugging-Zwecke brauchbar, da dann nicht verschlüsselt wird.

- **Compression [yes|no]**

  Schaltet die Datenkomprimierung ein bzw. aus. Voreingestellt ist keine Komprimierung, da in heutigen Netzwerken der Nutzwert fraglich wäre. Empfehlenswert ist hingegen, Modem-Verbindungen zu komprimieren, um einen höheren effektiven Durchsatz zu erreichen. Dies belastet zwar die CPU, aber der verwendete Algorithmus (GNU ZIP) ist recht leistungsfähig und zudem noch konfigurierbar (siehe **CompressionLevel**).

- **CompressionLevel [(1-9)]**

  Bei aktivierter Komprimierung kann hier die zu verwendende Kompressionstiefe eingestellt werden. **1** ist sehr schnell, aber wenig effizient, **9** ist langsam und komprimiert sehr stark. Die Voreinstellung ist **6**, was einen guten Kompromiß darstellt.

- **ConnectionAttempts** <**anzahl**>

  Gibt die Anzahl der Versuche an, nach denen entweder auf *rsh* zurückgeschaltet oder die Verbindung abgebrochen wird. Diese Option ist für Batch-Betrieb

interessant, wenn der entfernte Rechner manchmal schwer erreichbar ist. Der Verbindungsaufbau wird jede Sekunde erneut versucht, bis die eingestellte **Anzahl** erreicht ist.

- **EscapeChar [zeichen | ^zeicher | none]**

  Mit diesem **Zeichen** (voreingestellt ist ~) kann man zurück in den Kommandomodus des jeweiligen Programms wechseln – ähnlich wie bei *telnet*. Die Verwendung mit normalen Programmen macht nicht allzuviel Sinn, weshalb hier **none** eingetragen werden sollte.

- **FallBackToRsh [yes|no]**

  Gibt an, ob bei fehlgeschlagenem SSH-Verbindungsaufbau (meist ist *sshd* nicht aktiv) auf*rsh* zurückgeschaltet werden soll. Die Voreinstellung ist **yes**, was jedoch eine unverschlüsselte Verbindung ermöglicht und deshalb mit Vorsicht einzusetzen ist. SSH gibt im Falle einer unverschlüsselten Verbindung jedoch eine Warnungsmeldung aus, um auf diesen Mißstand hinzuweisen.

- **ForwardAgent [yes|no]**

  Ist der Authentication Agent aktiv, kann hier eingestellt werden, ob ein Forwarding der Connection auf den entfernten Rechner stattfinden soll.

- **ForwardX11 [yes|no]**

  Schaltet das automatisch Forwarding von X-Sessions ein oder aus.

- **GlobalKnownHostsFile <datei>**

  Soll eine andere Datei als */etc/ssh_known_hosts* für die Speicherung von bekannten SSH-Hosts benutzt werden, kann man sie hier angeben.

- **HostName [name | ip-nummer]**

  Hier kann man für den übergebenen Hostnamen einen Alias definieren, der dann den Computer angibt, mit dem die Verbindung in Wirklichkeit aufgebaut wird. Als Voreinstellung wird versucht, den übergebenen Hostnamen aufzulösen und zu kontaktieren.

- **IdentityFile <datei>**

  In dieser Datei ist der RSA-Key des Benutzers gespeichert. Es können mehrere **IdentityFile**-Zeilen angegeben werden, deren Dateien dann sequentiell durchsucht werden. Im Normalfall befindet sich die verwendete Datei im Homedirectory des Benutzers und heißt *~/.ssh/identity*.

- **KeepAlive [yes|no]**

  Diese Option ist das Äquivalent von **KeepAlive** in der *sshd*-Konfiguration und entsprechend zu setzen.

- **LocalForward** <**lokaler_port**> <**entfernter_rechner:port**>

  Der **lokale Port** wird von SSH durch Forwarding auf den **entsprechenden Port des entfernten Rechners** verbunden. Dieses Forwarding wird vom entfernten Rechner durchgeführt, mit dem die SSH-Verbindung aufgebaut wurde. Die Kommunikation läuft dann über den verschlüsselten Kanal von SSH. Dieser Eintrag kann mehrfach vorhanden sein, um mehrere Dienste forwarden zu können. Das Forwarding privilegierter Ports (d.h. Portnummer kleiner als 1024) ist nur dem Superuser (*root*) erlaubt.

- **PasswordAuthentication [yes|no]**

  Gibt an, ob zur Authentifizierung Paßwörter verwendet verden.

- **Port** <**port**>

  Der verwendete Port von SSH auf Serverseite (normalerweise **22**) kann hier geändert werden.

- **ProxyCommand** <**befehlszeile**>

  Wird ein Proxy (z.B. SOCKS) verwendet, kann hier die zu verwendende **Befehlszeile** angegeben werden. Die **Befehlszeile** wird mit */bin/sh* ausgeführt, als Platzhalter können dabei **%h** für den Hostnamen und **%p** für diePortnummer verwendet werden. Die **Befehlszeile** kann aus beliebigen Kommandos bestehen, sofern diese von der Standardeingabe lesen und auf die Standardausgabe schreiben. Ein Anwendungsfall wäre z.B., den *sshd* mit der Option **-i** (d.h. Start via *inetd*) aufzurufen, um eine gesicherte Verbindung zu initiieren.

  Bei der Verwendung von SOCKS (siehe 2.2) kann man in SSH bereits beim Compilieren durch die *configure*-Option **–with-socks** eine direkte Unterstützung einbauen.

- **RemoteForward** <**entfernter_port**> <**entfernter_rechner:port**>

  RemoteForward funktioniert wie **LocalForward**, nur daß das Forwarding für den entfernten Rechner vom lokalen Rechner ausgeführt wird.

- **RhostsAuthentication [yes|no]**

  Gibt an, ob der *rhosts*-Mechanismus zur Authentifizierung verwendet werden soll. Da diese Methode diverse Unsicherheiten aufweist, sollte man **no** eintragen.

- **RhostsRSAAuthentication [yes|no]**

  Die Kombination von *rhosts*- und RSA-Mechanismus kann als zuverlässig gelten und sollte deshalb auf **yes** gesetzt werden.

- **RSAAuthentication |yes|no]**

  Diese Option wird nur dann wirksam, wenn entweder *$HOME/.ssh/identity* existiert oder wenn ein *Authentication Agent* aktiv ist.

- S**trictHostKeyChecking [yes|no]**

  Normalerweise fügt SSH Rechner, wenn sie zum ersten Mal erfolgreich kontaktiert wurden, automatisch in die Datei *$HOME/.ssh/known_hosts* ein. Wird **yes** angegeben, müssen diese Einträge vom Benutzer selbst vorgenommen werden. Zusätzlich wird der Angriff durch *Trojanische Pferde* erschwert, da ein geänderter Host Key auf dem kontaktierten Server einen Verbindungsabbruch zur Folge hat.

- **User** <**benutzername**>

  Der zu verwendende **Benutzername** auf dem jeweiligen **Host** kann hier fest eingegeben werden. Dies ist praktisch, wenn man unterschiedliche Kennungen auf vielen Rechnern hat.

- **UserKnownHostsFile** <**datei**>

  Soll eine andere Datei als $HOME/.ssh/known_hosts zur Speicherung von bekannten Rechnernamen benutzt werden, kann man hier den Namen angeben.

- **UseRsh [yes|no]**

  Hier kann man einstellen, ob der jeweilige **Host** direkt mit *rlogin* /*rsh* kontaktiert werden soll. Dies ist nur dann sinnvoll, wenn auf dem **Host** kein *sshd* aktiv ist.

5.2.3.4 Client-Programme

Im SSH-Paket sind folgende Client-Programme enthalten:

- *ssh* (secure shell)

  Das eigentliche Arbeitspferd von SSH ist das Programm gleichen Namens. Mit *ssh* kann man alle erdenklichen Befehle auf anderen Computern ausführen lassen, wird kein Befehle angegeben, findet faktisch ein *remote login* zum interaktiven Arbeiten statt. Damit ist *ssh* ein guter Ersatz für unsichere Client-Programme wie *rsh, rlogin oder telnet*. Durch Verschlüsselung und einstellbare Kompression ist für fast jeden Anwendungsfall eine geeignete Einstellmöglichkeit vorhanden.

  Zusätzlich kann *ssh* noch X-Sessions forwarden und somit ebenfalls verschlüsseln, was der Sicherheit von X sehr zugute kommt. Dieses Forwarding ist auch noch für andere Ports (und damit Dienste) konfigurierbar, was das Arbeiten über Netzwerkverbindungen wesentlich sicherer macht.

  Als Arbeitserleichterung setzt *ssh* folgende Umgebungsvariablen:

  – DISPLAY

    In dieser Standard-Variable ist eine Angabe über den zu verwendenden X-Server gespeichert. Profis sollten sich nicht über die verwendete Nummer des X-Servers (die auch bei *single-display*-Computern größer als 0

ist) wundern, denn SSH verwendet diesen Eintrag zum Forwarding von X-Sessions. Eine manuelle Änderung dieser Variable umgeht bei X-Sessions die sicheren Übertragungswege von SSH und sollte deshalb unterbleiben.

- HOME

  Das Standard-Arbeitsverzeichnis des Benutzers wird hier abgelegt.

- LOGNAME

  Synonym für USER, einige Unix-Derivate verwenden diese Variable aus historischen Gründen.

- MAIL

  Die Mailbox des Benutzers wird hier gespeichert.

- PATH

  Diese Variable enthält den Suchpfad für ausführbare Dateien.

- SSH_AUTHENTICATION_FD

  Der Authentication Agent verwendet diese Variable; sie gibt den benutzten Dateideskriptor an, wenn eine Sitzung durch Forwarding aufgebaut wurde.

- SSH_CLIENT

  Die IP-Nummer des Clients und die verwendeten Ports auf Cient- und Serverseite werden hier abgelegt.

- SSH_TTY

  Das verwendete Terminal (real oder pseudo) steht in dieser Variablen.

- TZ

  Eine weitere Standard-Variable; diese enthält die Zeitzonen-Information.

- USER

  Die Login-Kennung des Benutzers wird hier gespeichert.

- zusätzliche Variableneinstellungen

  Weitere Umgebungsvariablen, die gesetzt werden sollen, können in */etc/environment* (systemweite Einstellungen) und für jeden Benutzer in *$HOME/.ssh/environment* angegeben werden, die Schreibweise ist wie in *sh*-Skripten.

Eine genaue Beschreibung der Kommandozeilenoptionen von *ssh* ist in [4] zu finden.

- *scp* (secure copy)

  Dieses Programm ist der Ersatz für den *rcp*-Befehl (remote copy), mit dem man wie bei *ftp* Dateien auf andere Rechner übertragen kann. Allerdings geht die Funktionalität beim Kopieren weit über die vom normalen *ftp* hinaus, so kann man ganze Verzeichnisse rekursiv kopieren oder von einem entfernten Rechner auf einen anderen entfernten Rechner kopieren. Außerdem können Dateien auch

komprimiert übertragen werden. Für eine detaillierte Funktionsbeschreibung sei auf [4] verwiesen.

- *slogin* (secure login)

  Dieses Programm ist nur ein Link auf *ssh*, das Programm entspricht einem *login* über einen sicheren Kanal bzw. einem Aufruf von *ssh* ohne nachfolgenden auszuführenden Befehl.

5.2.3.5 Hosts-Dateien

Beim normalen *rhosts*-Mechanismus ist es möglich, sich von bestimmten Rechnern ohne Paßwort auf bestimmten Kennungen einzuloggen. Normalerweise wird das von SSH aus Sicherheitsgründen unterbunden. Durch die Verwendung von speziellen Dateien kann jedoch ein sicherer Betrieb trotz *rhosts* gewährleistet werden.

- /etc/ssh_known_hosts

  Diese systemweit gültige Datei wird am besten durch das Perl-Programm *make-ssh-known-hosts* erzeugt, das dem SSH-Paket beiliegt. Nach dem Aufruf erzeugt dieses Programm (für dessen Einsatz *perl* in der Version 5 oder höher installiert sein muß) vollautomatisch eine Datei, die die Host Keys von allen Rechnern der jeweiligen Domain enthält. Die vielfältigen Optionen von *make-ssh-known-hosts* sind in [4] genauer beschrieben.

- $HOME/.ssh/known_hosts

  Diese Datei ist benutzerspezifisch und wird automatisch aktualisiert, wenn man mit *ssh* bzw. *slogin* einen Host erstmalig kontaktiert. In dieser Datei werden also bestenfalls manuell Einträge gelöscht, was mit jedem Editor möglich ist.

5.2.4 SSH-Aktivierung beim Systemstart

Da die SSH-Clients einen laufenden SSH-Server benötigen, um richtig arbeiten zu können, sollte auf allen potentiellen Servern *sshd* gestartet werden. Da *sshd* auf privilegierte Ports zugreift, muß dieser Prozeß mit root-Rechten gestartet werden, tut man das nicht, erfolgt **keine** Fehlermeldung (bis auf den Hinweis, das der jeweilige Host Key nicht gelesen werden kann).

Es empfiehlt sich, den *sshd* bereits beim Wechseln in den Multiuser-Modus automatisch zu starten. Bei dem auf dem Testsystem verwendeten Solaris-Betriebssystem wird dazu im Verzeichnis */etc/rc3.d* bzw. */etc/rc2.d* eine Batchdatei abgelegt, die sowohl das Starten als auch das saubere Herunterfahren von *sshd* (und einigen anderen getesteten Programmen) übernimmt. Dieser Vorgang ist auf anderen System-5-Unix-Derivaten fast identisch, nur das Verzeichnis kann variieren. Bei anderen Unix-Betriebssystemen sollte es ebenfalls kein Problem sein, *sshd* bereits beim Booten zu aktivieren.

Ein Beispiel einer solchen Batchdatei ist in Kapitel A.3 zu finden.

5.2.5 Erfahrungsbericht

5.2.5.1 Testumgebung und Testlauf

Als Testumgebung wurde der bereits mehrfach erwähnte Versuchsaufbau verwendet.

Die Testläufe wurden in zwei Bereichen durchgeführt:

- Administierung eines WWW-Servers in München

  SSH wurde anfangs benutzt, um einen WWW-Server, der beim Provider ECRC in München ans Internet angebunden war, zu administrieren. Da die Auslastung der dabei verwendeten Netzwerkleitungen sehr hoch war, war eine Geschwindigkeitsmessung nicht sinnvoll. Allerdings kann davon ausgegangen werden, daß der WWW-Server (Sun Ultrasparc 167 MHz, 128 MByte RAM, Solaris 2.5) die verbrauchte Rechenzeit für den SSH-Server kaum bemerkt hat, denn dieser Prozeß belegte nie mehr als ein Prozent der CPU-Leistung.

- In der verwendeten Testumgebung (siehe Kapitel A.1) und im lokalen Netzwerk der Fachhochschule wurde SSH ebenfalls eingesetzt über einen längeren Zeitraum eingesetzt.

  In beiden Bereichen gab es bei der Verwendung von SSH keine Probleme.

5.2.5.2 Performance

Auf Messungen für interaktive Dienste wurde verzichtet, da hierbei der Flaschenhals meist der Benutzer selbst ist.

Zu Testzwecken wurde eine 6 MByte große Datei mit *scp* von Mozart nach Zerberus kopiert und die dabei auftretende CPU-Belastung mittels *top* beobachtet. Die relavanten Ausgaben von *top* sind hier wiedergegeben (es wurden zwei Testläufe durchgeführt):

```
   last pid: 23461;  load averages:  0.39,  0.43,  0.43         13:23:28
   85 processes:  82 sleeping, 2 stopped, 1 on cpu
   CPU states: 62.7% idle, 23.6% user, 13.6% kernel,  0.0% iowait,  0.0%
swap
   Memory: 90M real, 548K free, 78M swap, 249M free swap
     PID USERNAME PRI NICE  SIZE   RES STATE   TIME   WCPU    CPU
COMMAND
   23459 root      26    0 1624K 1204K sleep   0:07  5.77% 19.12%
sshd
   23461 jmayerho  33    0 1264K  832K sleep   0:00  1.26%  1.52%
scp
   13521 root      23    0 1508K  880K sleep   1:01  0.02%  0.01%
sshd

   last pid: 23473;  load averages:  0.48,  0.39,  0.41         13:27:15
```

```
   83 processes:  79 sleeping, 1 running, 2 stopped, 1 on cpu
   CPU states: 62.7% idle, 24.5% user, 10.9% kernel,  1.8% iowait,  0.0%
swap
   Memory: 90M real, 516K free, 81M swap, 246M free swap
     PID USERNAME PRI NICE  SIZE   RES STATE   TIME   WCPU    CPU
COMMAND
   23471 root       7    0 1632K 1212K run     0:12  9.66% 20.43%
sshd
   23473 jmayerho  33    0 1264K  832K sleep   0:00  0.98%  0.93%
scp
   13521 root      23    0 1508K  880K sleep   1:01  0.02%  0.01%
sshd
```

Der Anteil an verbrauchter CPU-Zeit ist mit 20% zwar schon im System bemerkbar, aber bei der Datenmenge nicht weiter schlimm. Ein Vergleich mit *rcp*, das etwa 8% CPU belegt, fällt nicht einmal schlecht aus, da bei *rcp* weder Komprimierung noch Verschlüsselung vorgenommen werden.

Die durchgeführte Verschlüsselung und die einstellbare Komprimierung aller Daten bedeutet allerdings, daß sowohl Client- als auch Serverseite über einigermaßen aktuelle Hardware (vor allem CPU) verfügen sollten, um reibungslose Abläufe sicherzustellen. Bei veralteter Hardware wird man unweigerlich eine gewisse Zeitverzögerung feststellen. Die CPU von Mozart (Intel 486DX33) benötigte für *rcp* ganze 13 Sekunden (reale Zeit), für *scp* jedoch 88 Sekunden.

5.2.5.3 Bewertung

Die abgefangenen und analysierten Pakete ließen keinen Rückschluß auf die übertragenen Daten zu, so daß SSH gegenüber halbherzigen Angriffsversuchen als sicher gelten kann. Wie immer ist jedoch die Sicherheit von Daten immer in Relation zum Preis des Aufwandes zu setzen, der zum Dechiffrieren erforderlich ist. Erfahrene Kryptoanalytiker könnten sicher mit beträchtlichem Aufwand den verwendeten Schlüssel knacken und die Daten im Klartext lesen, dies gilt jedoch für **alle** bekannten verwendbaren Verschlüsselungssysteme und sollte deshalb kein Grund sein, SSH nicht einzusetzen. Der Verbrauch von Rechenzeit bei aktivierter Kompression und Verschlüsselung macht sich auf älteren Systemen zwar bemerkbar, jedoch sollte man die Sicherheit höher einschätzen als ein paar gesparte Minuten bei eher seltenen Datenübertragungen mit *scp*. Interaktives Arbeiten wird mit SSH wesentlich sicherer, ohne die beteiligten Rechner zu stark zu belasten. Die leichte Installation und Konfiguration sowie der problemlose Einsatz des SSH-Paketes weisen auf ein gut gepflegtes und stabiles Programm hin, so daß SSH empfohlen werden kann, wenn es darum geht, bekannte Sicherheitslücken von Telnet und ähnlichen Programmen zu schließen.

Kapitel 6

Logfile/Tracefile-Auswertung

6.1 Logfile/Tracefile-Auswertung

Christian Rotter

Log- bzw. Tracefiles sind Textdateien, in denen ständig laufende Programme die Resultate ihrer Arbeit und Debugging-Ausgaben ablegen. Der Übergang vom Trace- zum Logfile ist fließend und eigentlich ist die Bezeichnung nur Geschmackssache.

Programme, die den laufenden Betrieb des Rechners überwachen, haben zwei Möglichkeiten, sich dem interessierten Administrator mitzuteilen:

- Eine Möglichkeit ist, über ein Fenster ständig eine Anzeige der relevanten Daten vorzunehmen. Dieses Online-Verfahren hat viele Vorteile, aber es ist nicht so leicht, die Geschehnisse von letzten Monat nochmals anzuzeigen, da diese Daten dann nicht mehr vorhanden sind.

- Die andere Möglichkeit besteht darin, in den oben genannten Log-/Tracefiles die Statusmeldungen zur späteren Verwendung abzuspeichern. Mit diesen Dateien kann dann jede gewünschte Auswertung gemacht werden, sofern man über ein entsprechendes Analysetool verfügt.

Im Rahmen dieser Diplomarbeit sind zwei Bereiche als besonders wichtig einzustufen:

- Die Abläufe im Prozeßgeschehen der Rechnerumgebung müssen ständig überwacht werden, um einen reibungslosen Ablauf über mehrere Wochen hinweg sicherstellen zu können. Für diesen Bereich benutzt fast jedes Unix-Derivat den *syslog*-Mechanismus, der dazu benutzt wird, Meldungen von diversen Programmen an zentraler Stelle zu sammlen. Es gibt natürlich noch spezialisierte Programme für besondere Einsatzbereiche, deren Behandlung jedoch den Rahmen dieser Diplomarbeit sprengen würden.

- Das Datenaufkommen und damit zusammenhängende Bereiche (wie Protokolle und Dienste) im angeschlossenen Netzwerk müssen ebenfalls überwacht werden, um Sicherheitslücken im System möglichst schnell aufspüren und beheben zu können, denn ein Einbruchsversuch über das Netzwerk kann vielleicht zurückverfolgt und damit diese Gefahr beseitigt werden (natürlich kann es jederzeit zu neuen Problemen dieser Art kommen). Für diesen Zweck ist *tcpdump* gut geeignet, das in Kapitel 8.3 genauer beschrieben ist.

Für beide Bereich gibt es entsprechende Programme, die mehr oder weniger einfach zu bedienen sind und mehr oder weniger komplexe Ausgaben erzeugen, die der Administrator dann auswerten muß, besonders die Netzwerk-Monitore sind in diesem Bereich als recht kryptisch einzustufen, darum wird auch in diesem Kapitel der Auswertung von tcpdump-Tracefiles relativ viel Platz eingeräumt.

6.2 swatch

Christian Rotter

Quelle:

ftp://ftp.stanford.edu/general/security-tools/swatch/

Ein wichtiger Sicherheitsaspekt für jedem Rechner, auf dem mehrere Benutzer arbeiten, ist die Speicherung relevanter Vorkommnisse in einfachen Textdateien, den sogenannte Logfiles. Die dabei verwendeten Mechanismen sind sehr trickreich, da sichergestellt sein muß, daß bei einem Absturz oder gar einem Systemcrash keine Meldungen verloren gehen können. Außerdem ist es ratsam, die Statusmeldungen eines ganzen Workstation-Clusters zentral zu sammeln, um den erforderlichen Aufwand bei der Auswertung zu reduzieren. Auf den meisten Unix-Plattformen werden daher eigene Daemon-Programme eingesetzt, die diesen Anforderungen gewachsen sind, der bekannteste trägt den Namen *syslogd*.

6.2.1 Probleme mit syslogd

Wie fast alle Daemons, die irgendwelche Ausgaben generieren, hat auch *syslogd* das Problem, daß entweder

- der Umfang der Logfiles ins Unermeßliche wächst

oder

- durch einen zu geringen Log-Level wichtige Informationen verloren gehen können.

Man kann *syslogd* zwar so konfigurieren, daß die Größe der Logfiles kalkulierbar bleibt, aber damit ist noch nicht das eigentliche Problem behoben, das ganz einfach darin besteht, daß *syslogd* zwar im System aktiv ist, seine Ausgaben allerdings erst

dann Beachtung finden, wenn es wirklich ein schwerwiegendes Problem gab und man die Ursache sucht. Interessanterweise ließen sich viele schwerwiegende Probleme aber vermeiden, wenn man bereits erste Anzeichen, z.B. vermehrte fehlgeschlagene Login-Versuche, bemerken und entsprechend reagieren würde, anstatt nach einem erfolgten Einbruch in das System festzustellen, daß dies bereits seit längerer Zeit versucht und nur nicht bemerkt wurde, weil niemand Zeit hatte, die Logfiles durchzusehen.

6.2.2 Problembehebung

Es gibt aber eine ganze Menge von kleinen Hilfsprogrammen, die versuchen genau dieses Problem zu beheben.

Dabei lassen sich zwei Typen unterscheiden:

- Der eine Typ generiert Reports aus den umfangreichen Logfiles des Sytems, was normalerweise bedeutet, daß alle weniger wichtigen Meldungen herausgefiltert werden. Damit hat man in gewisser Weise ein ähnliches Ergebnis wie vorher, wenn man den Report-Generator falsch konfiguriert.

  Das Problem ist damit zwar nicht behoben, aber es ist zumindest eine Verbesserung zu verzeichnen, da ja auch Mehrfach-Meldungen zusammengefaßt werden können und der Umfang der durchzusehenden Textdateien geringer wird. Je nach verwendeter *syslogd*-Version ist diese Funktionalität aber manchmal schon vorhanden und muß nur noch aktiviert werden, weshalb auf diesen Typ nicht näher eingegangen wird.

- Der andere Typ reagiert direkt auf bestimmte Einträge und löst vorher im System definierte Aktionen aus, was natürlich eine gewaltige Verbesserung bedeutet. Es ist zum Beispiel möglich, bei wiederholten fehlerhaften Login-Versuchen direkt den Administrator zu pagen oder ihm eine E-Mail zukommen zu lassen, die auf das Problem hinweist. Auch diese Funktionalität ist in manchen *syslogd*-Versionen enthalten, jedoch ist besonders bei älteren Systemen die Wahrscheinlichkeit sehr gering, eine solche Version vorzufinden.

  Auch bekannte Firewall-Systeme (z.B. Solstice Firewall-1) verfügen über solche Mechanismen, allerdings bezahlt man dafür auch einen relativ hohen Preis.

Durch die kleinen Hilfsprogramme läßt sich aber auf fast jedem System die gewünschte Funktionalität nachbilden, weshalb in diesem Kapitel eines
dieser Utilities genauer untersucht wird, nämlich *swatch*.

6.2.3 swatch – der Simple WATCHer

Das Programm wurde an der Stanford University in Kalifornien entwickelt. Als Programmiersprache wurde *perl* verwendet, was bedeutet, daß zum Einsatz von *swatch* auch *perl* installiert sein muß. Zusätzlich kann man sehr leicht Änderungen und Erweiterungen einbauen, sofern man sich mit *perl* auskennt.

6.2.4 Installation

Zur Installation von *swatch* wird einfach das Archiv entpackt (siehe auch Kapitel 2.3.2), in das neue Verzeichnis gewechselt und mit

Prompt> sh install.sh

die Installationsroutine aufgerufen, die interaktiv einige Eingaben (meist Pfadangaben) erfordert. Das erste Verzeichnis, das man angibt, sollte man sich merken, da man dort nach der Installation die Datei *swatch* editieren sollte, um die Pfade zum *tail*-Befehl und zum Logfile des *syslogd* anzupassen. Alternativ kann man diese beiden Angaben zwar auch beim Aufruf von *swatch* machen (neben einigen anderen Kommandozeilenoptionen, siehe auch [8]), jedoch muß dieser Aufwand nicht sein.

6.2.5 Konfiguration

Um die Konfiguration zu vereinfachen, ist im Originalverzeichnis ein Unterverzeichnis *config_files* enthalten, in dem einige Beispielkonfigurationen als Textdatei abgelegt sind. Man kann sich eine dieser Dateien nach *$HOME/.swatchrc* kopieren und modifizieren, oder man erzeugt die Datei *$HOME/.swatchrc* manuell.

Die Konfigurationsdatei von swatch besteht aus einzelnen Textzeilen mit jeweils maximal vier Spalten, die mit Tabulatoren getrennt sein müssen.

Eine einzelne Zeile sieht folgendermaßen aus:

/Muster/[,/Muster/,...] Befehl[,Befehl,...] [[HH:]MM:]SS Beginn:Länge

Die einzelnen Einträge sind nach Unix-Konventionen anzugeben:

- **[]** bedeutet, daß der enthaltene Parameter optional ist.
- **...** weist darauf hin, daß mehrere Einträge dieser Art folgen können.
- **/** und **,** müssen mit angegeben werden.
- **Muster** ist eine *regular expression* von *perl*, deren Syntax größtenteil der *egrep* (siehe [7]) *regular expression* entspricht, einfach gesagt also der Triggertext im *syslogd*-Logfile, auf den *swatch* reagiert.
- **Befehl** weist *swatch* an, eine entsprechende Aktion einzuleiten:
  - **echo[=mode]** gibt die Zeile auf dem Bildschirm aus, die auf das Muster gepaßt hat.

    für **mode** kann bei Bedarf **normal**, **bold** (fettgedruckt), **underscore** (unterstrichen), **blink** (blinkend), oder **inverse** (inverse Darstellung) angegeben werden, wobei **normal** die Voreinstellung ist. Man sollte beachten, daß nicht alle Terminals (bzw. Terminalemulationen) alle Modi darstellen können.

  - **bell[=N]** gibt die Zeile aus und erzeugt einen **N** Piep-Töne (oder ein Bildschirmblitzen, je nach Konfiguration des Terminals). Die Voreinstellung bei fehlendem **N** ist 1.
  - **exec=command** führt **command** auf Betriebssystemebene aus, als Parameter kann man **$0** bzw. $* bzw. beliebig viele **$N** angeben, wobei **N** jeweils für die **N.** Spalte der Zeile steht, die auf das Muster von **exec** gepaßt hat. **$0** und **$*** stehen für die komplette Zeile.
  - **ignore** ignoriert die Zeile und macht nichts.
  - **mail[=address:address:...]** schreibt eine E-Mail an jede **address** die angegeben wird. Der Inhalt der E-Mail ist die Zeile, die auf Muster gepaßt hat. Als Voreinstellung ist der Benutzer eingetragen, der *swatch* gestartet hat.
  - **pipe=command** ruft **command** auf und schreibt die Zeile in die Standardeingabe von **command**.
  - **write[=user:user:...]** funktioniert wie **mail**, nur wird hier der *write*-Befehle verwendet.

- **HH:MM:SS** gibt eine Zeitspanne an, während deren Ablauf identische Einträge keine Mehrfachaktionen erzeugen, sondern eine einzige Zeile mit einem Zählerstand für die jeweilige Aktion verwendet wird. Man kann entweder nur Sekunden (**SS**) oder Sekunden und Minuten (**MM:SS**) oder Stunden und Minuten und Sekunden (**HH:MM:SS**) angeben oder überhaupt nichts.
- **Beginn:Länge** gibt die Startposition und die Länge des Zeitstempels an, der mit jeder Zeile ausgegeben wird. Dieser Eintrag darf nur dann erscheinen, wenn die dritte Spalte (Zeitspanne) ebenfalls vorhanden ist.

Um einen sinnvollen Betrieb von *swatch* zu ermöglichen, sollte man sich vor der Konfiguration einmal seine Logfiles ansehen, um festzustellen, was für Muster auftreten können und was man beim Auftreten eigentlich unternehmen will.

6.2.6 Hilfsprogramme für swatch

Im *swatch*-Paket ist noch ein Verzeichnis *utils* enthalten, in dem einige Beispiel-Kommandos enthalten sind, die als Argument für die **exec**-Aktion angegeben werden können. Dies sind jedoch nur Beispiele, man kann beliebige ausführbare Programme oder Batchdateien angeben und so jede nur denkbare Anforderung umsetzen.

6.2.7 Bewertung

Gemessen an seiner Größe ist *swatch* sicher ein brauchbares Programm, wenn man bereits *perl* installiert hat. Die Optimierung von *perl* für die Verarbeitung von Textdateien wird anschaulich unter Beweis gestellt, *swatch* ist sehr leicht an die jeweiligen

Anforderungen anzupassen und funktioniert zuverlässig. Dem Administrator wird ein Tool zur Verfügung gestellt, das ihm hilft, sein System besser im Griff zu haben und Probleme frühzeitig zu erkennen. *Swatch* ist für den täglichen Einsatz zu empfehlen.

6.3 tracelook

Jürgen Mayerhofer

Quelle:

ftp://ftp.cert.dfn.de

Eine graphische Auswertung des Netzwerkverkehrs über *awk, tcl/tk* und *xgraph* (siehe Kapitel 8.4.1.1) wird durch *tracelook* ermöglicht. Es benützt als Grundlage ein Logfile, das mit *tcpdump* generiert wurde.

6.3.1 Installation

xgraph, tcl/tk, awk und *tcpdump* müssen bereits auf dem System installiert sein. Da es sich bei *tracelook* um ein tk-Skript handelt, wird in der ersten Zeile „wish“ aufgerufen. Unter Umständen muß der Pfad erst ermittelt („which wish“) und angepaßt werden.

Ähnlich muß mit „bg.tcl“ vorgegangen werden. Dieses Skript ruft in der ersten Zeile „tclsh“ auf und wird von *tracelook* angestoßen, um einen Aufruf von *xgraph* vorzubereiten.

6.3.2 Test

Zum Starten muß dem Frontend *tracelook* als Parameter der Dateiname eines mittels *tcpdump* erzeugten Files übergeben werden. Darauf wird ein typisches tcl/tk-Fenster mit dem graphisch aufbereiteten Qutput des dumps geöffnet (Abb. 6.1).

Hier sind links die Rechner zu sehen, von denen eine Übertragung initiiert wurde, d.h. ein Paket gesendet wurde. Durch einen Einfachklick auf einen der Rechner auf der linken Seite wird ein Filter auf den von ihm ausgehenden Verkehr gesetzt(Abbildung 6.2).

Die so gefilterten Daten können mit „Save“ abgespeichert werden. mit „Deselect“ wird der Filter aufgehoben. Der Button „View“ zeigt die ausgewählte Übertragung detailliert an (Abbildung 6.3).Mit einem Doppelklick auf einen Filter (rechte Seite) wird die Ausgabe in xgraph angestoßen (Abbildung 6.4).

6.3.3 Bewertung

Durch die Handhabung mittels Graphischer Auswertung läßt sich schnell ein Überblick über die Verbindungen ermitteln.

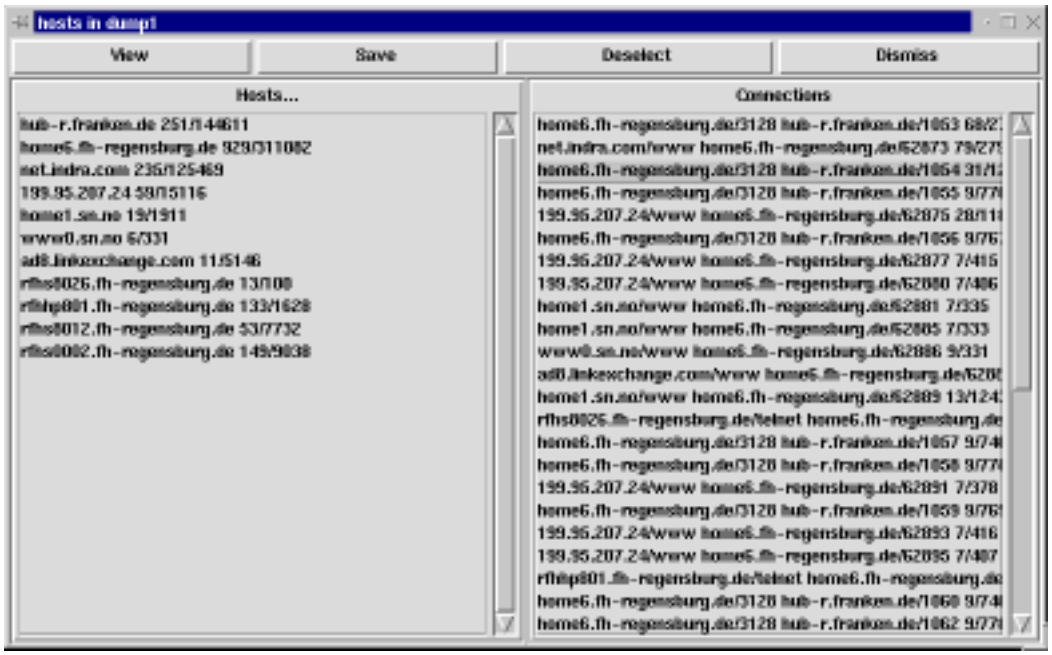

Abbildung 6.1: tracelook - Hauptmenü

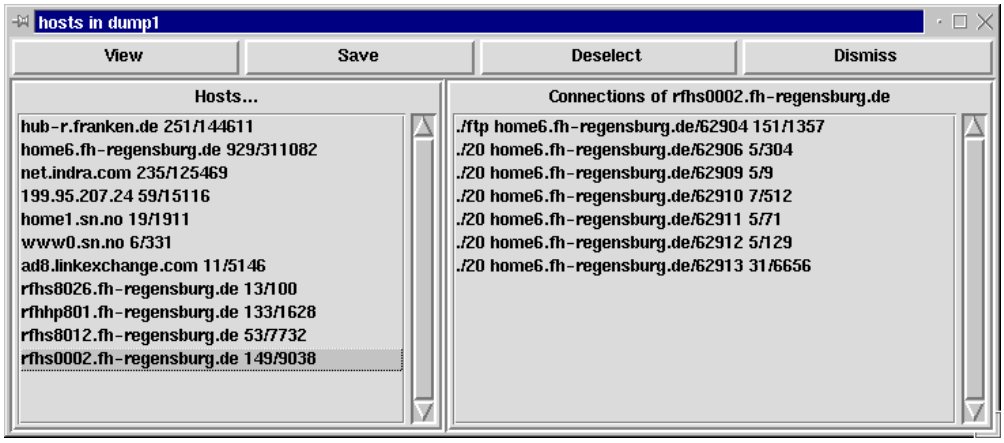

Abbildung 6.2: tracelook - Die Pakete eines Rechners im Trace-Zeitraum

Abbildung 6.3: tracelook - Ein Filter auf „finger“

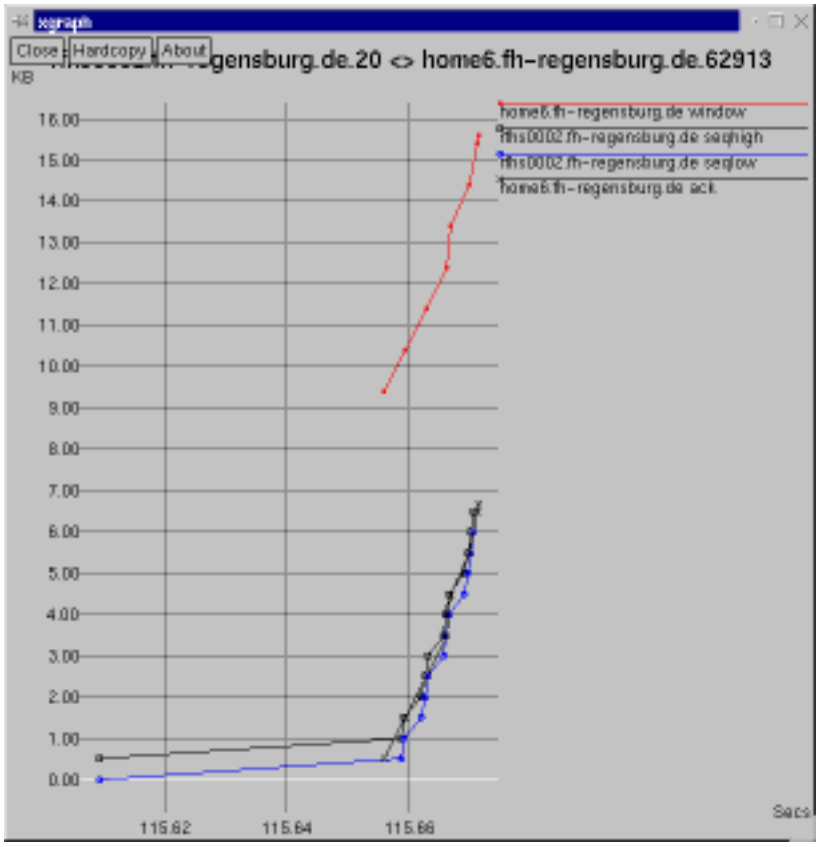

Abbildung 6.4: Datendurchsatz einer ftp-Sitzung

6.4 trimlog

Christian Rotter

Quelle:

ftp://coast.cs.purdue.edu/pub/tools/unix/trimlog/

Es gibt auch noch eine weitere Möglichkeit, den ständig wachsenden Log- und Tracefiles Einhalt zu gebieten. Das Programm *trimlog* hat allerdings eine etwas andere Arbeitsweise als die anderen in diesem Kapitel vorgestellten Programme, es wertet nämlich nicht aus und reagiert auch nicht auf bestimmte Textmuster, sondern es verkürzt Textdateien, indem es wahlweise vom Dateianfang oder vom Dateiende her beginnend Einträge entfernt. Dabei werden entweder einzelne Bytes oder ganze Zeilen als Grundlage für die Löschaktion verwendet. *Trimlog* eignet sich deshalb mehr für periodische Aufräumaktionen (z.B. mit *cron*), die von Zeit zu Zeit erforderlich sind. Man sollte aber beim Einsatz von *trimlog* nicht vergessen, vorher die Logfiles auf irgendeine Art und Weise auszuwerten, denn nachher sind wohl die meisten Informationen nicht mehr vorhanden.

6.4.1 Vorbereitungen

Nach dem Entpacken des Pakets (Vorgehensweise siehe Kapitel 2.3.2) muß man bei *trimlog* etwas mehr Aufwand betreiben als mit anderen Paketen. Solange man SunOS bzw. Solaris verwendet, sollte es aber keine Probleme geben. Bei anderen Unix-Derivaten muß man aber Funkionsaufrufe in der Datei *util.c* anpassen (und zwar *getrlimit* und *setrlimit*), die abweichende Parameter haben können. Das sollte aber kein allzu großes Problem darstellen. Im *Makefile* kann man noch die Einträge CONFIG (für den kompletten Dateinamen der Konfigurationsdatei), MANDIR (wo die Manuals von trimlog installiert werden sollen) und TRIMLOG (wo und unter welchem Namen trimlog installiert werden soll) an das eigene System anpassen.

6.4.2 Konfiguration

Nun muß man wieder einmal überlegen, was man mit *trimlog* eigentlich alles verkürzen will. Alle in Frage kommenden Dateien sollte man dann in der Konfigurationsdatei*trimlog.conf.Sun* eintragen oder die dort vorhanden Einträge übernehmen. Logfiles, die man nicht trimmen will, sollte man aus der Konfigurationsdatei entfernen.

Einträge in der Konfigurationsdatei haben folgenden Aufbau:

- # Kommentar (Unix-Standard)
- **sendsig Datei Signal**

  Wenn man bestimmten Daemons mitteilen will, daß ihr Logfile verändert wird, und der jeweilige Daemon diese Mitteilung über ein Signal entgegennehmen

kann, kann *trimlog* die Prozeß-ID des Daemons aus **Datei** auslesen und das angegebene **Signal** (das eine Zahl sein muß, symbolische Konstante wie beim *kill*-Befehl werden nicht unterstützt) an den Prozeß senden. Dabei wird natürlich vorausgesetzt, daß es eine Datei gibt, in der die Prozeß-ID abgelegt ist, was bei einigen Daemons (z.B. *httpd*) der Fall ist. Meistens muß nach **sendsig** zuerst ein Änderungsbefehl folgen, und dann muß mit einer weiteren **sendsig**-Anweisung der Daemon wieder aktiviert werden.

- **trimbylines Datei zeilenzahl**

  Diese Anweisung kopiert die letzten **zeilenzahl** Zeilen von **Datei** an den Dateianfang und entfernt alle anderen Zeilen.

- **trimbybytes Datei byteanzahl**

  Funktioniert wie **trimbylines**, nur werden Bytes als Ausgangsbasis verwendet, keine kompletten Zeilen.

- **truncate Datei bytezahl**

  Dieser Befehl löscht alle Daten in **Datei**, die nach dem **bytezahl**-ten Byte kommen. Meistens wird die **bytezahl** 0 sein, was bedeutet, daß die Datei geleert wird (nicht gelöscht).

6.4.3 Installation und Verwendung

Nach der Konfiguration kann mit

Prompt> make

der Übersetzungsvorgang gestartet und in Erfolgsfall mit

Prompt> make install install.man install.conf

die Installation abgeschlossen werden.

Es empfiehlt sich, *trimlog* über einen *cron*-Job regelmäßig zu starten, z.B einmal pro Woche. Die Verwendung von *cron* ist in [7] beschrieben.

6.4.4 Bewertung

Da *trimlog* ein sehr kompaktes Paket ist, gibt es nicht viel zu bewerten. Die Funktionalität ist als brauchbar einzustufen, und dem Administrator bleibt es erspart, Logfiles manuell zu löschen. Bei Systemen mit hohem Log-Level ist der Einsatz von *trimlog* aber durchaus sinnvoll, da die dort anfallenden Datenmengen enorm groß sein können. Das Problem ist allerdings, daß die Informationen in gelöschten oder getrimmten Logfiles wirklich weg sind, weshalb man sich vorher einige Gedanken über eine sinnvolle Reportgenerierung machen sollte.

Kapitel 7

Suchprogramme für Sicherheitslücken

7.1 satan

Jürgen Mayerhofer

Quelle:

ftp://ftp.cert.dfn.de/pub/tools/net/satan/

Zum Testen der Netzwerksicherheit insbesondere auf UNIX-Rechnern wurde das „Security Administrator Tool for Analyzing Networks“, kurz *satan,* ins Leben gerufen. Durch Abprüfen von offenen Ports und anschließendes Testen auf bekannte Sicherheitslöcher bei den gefundenen offenen Ports ermitteln *satan* seine Reports.

7.1.1 Installation

Die Installation ist in der Datei README beschrieben. Aus dieser Datei waren für uns folgende Schritte zutreffen:

- Für die Benutzung von *satan* ist ein *perl* in der Version 5.000 oder neuer erforderlich.
- Nach dem Entpacken des Sourcecodes ist im dabei entstandenen Verzeichnis ein Skript namens „reconfig“ zu finden. Dieses Skript muß aufgerufen werden, damit in den Satan-Perl-Skripten der Pfad von *perl* in der ersten Zeile gefixt wird.
- Durch den Aufruf von „make“ wird anschließend ein Binary namens *satan* erzeugt.
- Es ist zu beachten, daß *satan* auf ausreichend schneller Hardware läuft. Auch die im README empfohlenen 32 MB RAM waren zu wenig.

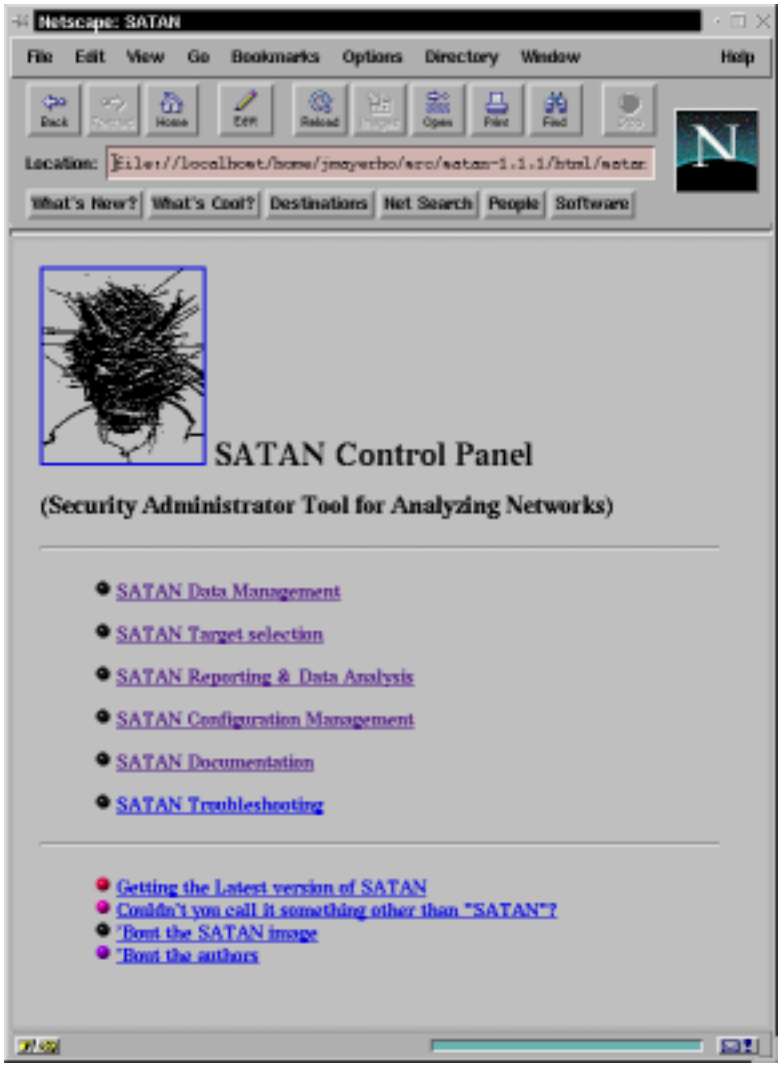

Abbildung 7.1: Satan - Menü

7.1.2 Test

Damit Daten gesammelt werden ist der Zugriff als „root“ auf den Netzwerktreiber im Kernel erforderlich; *satan* öffnet dabei einen HTML-Browser unter der Kennung von „root“. Wurde das X Window System als anderer Benutzer gestartet, so ist vorher dem Benutzer „root“ der Zugriff zu erlauben. Dies geschieht mittels *xhost*, besser aber – falls X mittels *xdm* gestartet wurde und einen .Xauthority-Eintrag generiert – mittels *xauth*. Abbildung 7.1 zeigt den Begrüßungsschirm von *satan*, der gleichzeitig das Ausgangsmenü darstellt.

Über den Menüpunkt „Target-Selection“ ist es möglich, einzelne Rechner sowie ganze Subnetze auf eventuell vorhandene Konfigurations-Fehler zu prüfen. In Abbildung 7.2 ist das Menü zu sehen, in dem die „Opfer“ einzutragen sind.

Während das Scannen eines einzelnen Rechners relativ flott vonstatten geht, dauert das Prüfen eines ganzen Netzes unter Umständen je nach Anzahl der darin enthaltenen Rechner ziemlich lange. Das Scannen eines Subnetzes erzeugt außerdem eine gehörige Netzlast, so daß dies zu Zeiten vorgenommen werden sollte, an denen der Produktivbetrieb nicht in Mitleidenschaft gezogen wird. Außerdem Zum Scannen eines kompletten Class-C Subnetzes sollte man:

1. unbedigt die Berechtigung zum Scannen mit den verantwortlichen Stellen abklären. Das Scannen erzeugt Einträge in den messages-Dateien der UNIX-Hosts!

2. viel Zeit mitbringen. Zirka 2-3 Stunden kann man für ein gut belegtes Class-C Netz rechnen.

Es ist dabei nicht möglich, mehrere Instanzen von satan aufzurufen, d.h. die Ausgabe blockiert solange, bis ein gestarteter Scan beendet ist.

Abbildung 7.3 zeigt *satan* bei der Arbeit.

Festgestellte Schwächen (Abbildung 7.4 und Abbildung 7.5) und welche Auswirkungen sie haben, sind über das Ergebnis-Protokoll über Mausklicks erfragbar.

Unser Netz ist keine Spielwiese für Sicherheitsfetischisten. Ich bitte darum, daß Übende mit *satan* ihr eigenes Netz scannen und nicht unseres. Die oben genannten Rechneradressen sollen nicht zur Nachahmung mit den gleichen Werten anregen!

7.1.3 Bewertung

Bei dem frei verfügbaren Security-Scanner handelt es sich – genau wie bei *iss*– um eine Version aus dem Jahr 1995. Lediglich an der Linux-Portierung, die bei uns zum Einsatz kam, wird noch gearbeitet, der Funktionsumfang bleibt aber dabei unverändert. Im Gegensatz zu *iss* ist *satan* aber keine eingeschränkte Demo-Version, sondern ein vollwertiges, ausgereiftes Produkt, das auch auf aktuellen Betriebssystemen Fehlkonfigurationen ausfindig macht.

Wie bei *pcops* muß auch bei *satan* dem Administrator bewußt sein, daß er nicht der einzige ist, der diesen Scanner einsetzt. Auch der „böse Bube“ kann die Fehler-Berichte zu illegalen Transaktionen vewenden. Aufgrund seiner Arbeitsweise ist aber

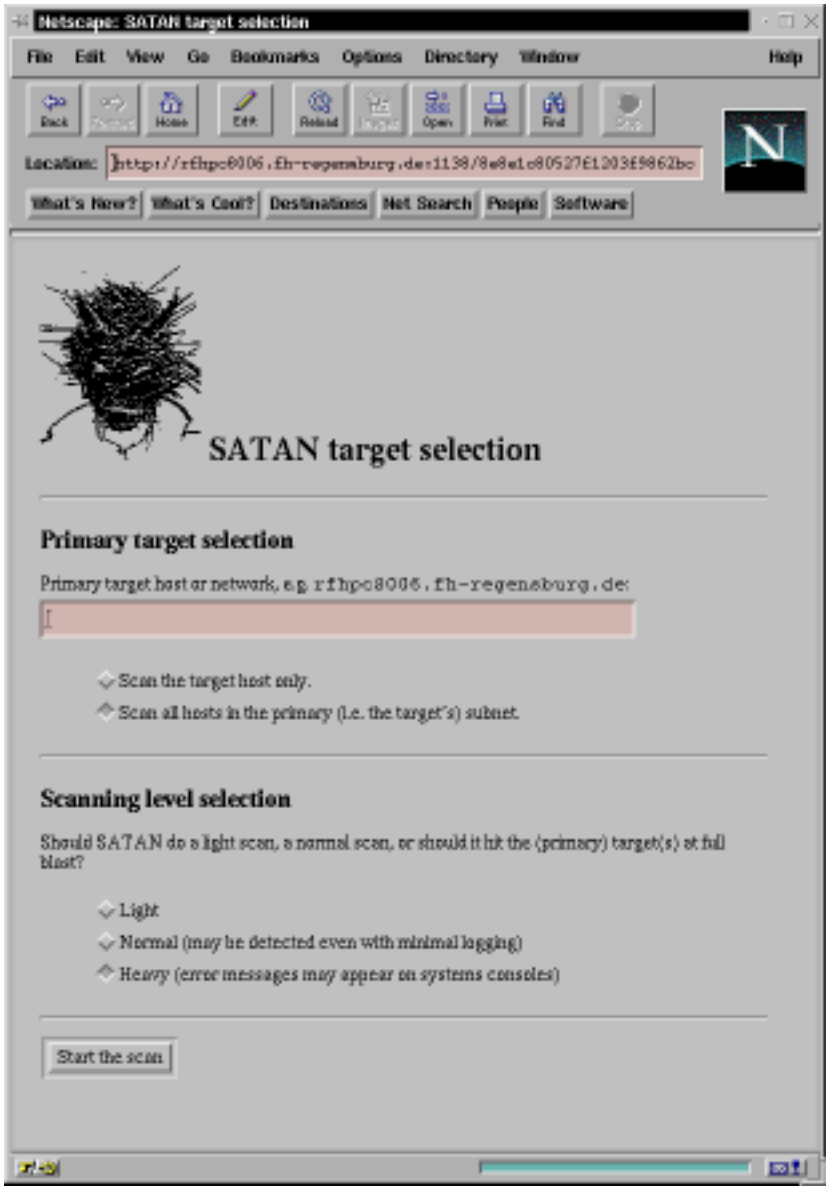

Abbildung 7.2: Satan - Welche Daten sollen ermittelt werden?

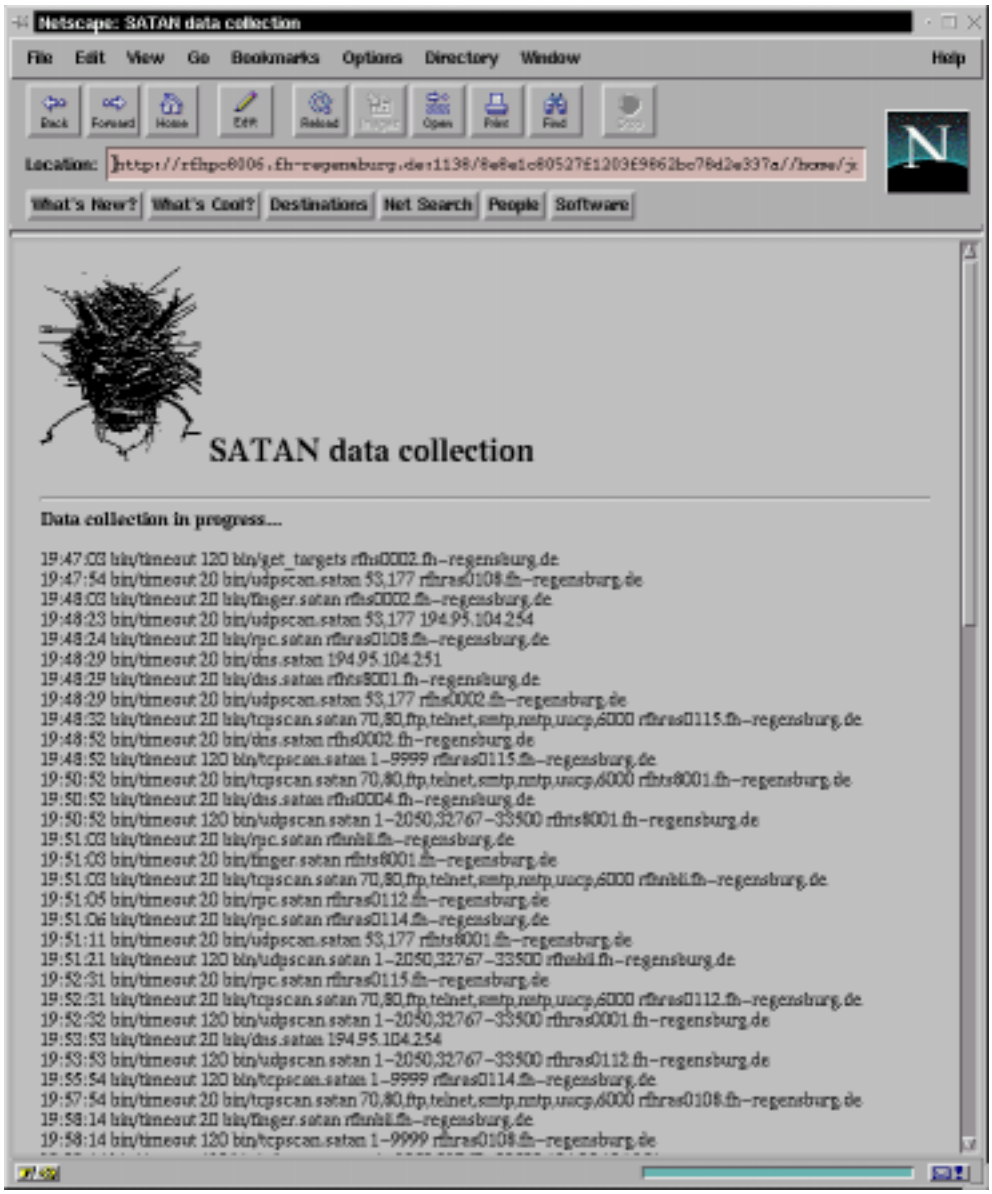

Abbildung 7.3: Satan beim Sammeln von Daten

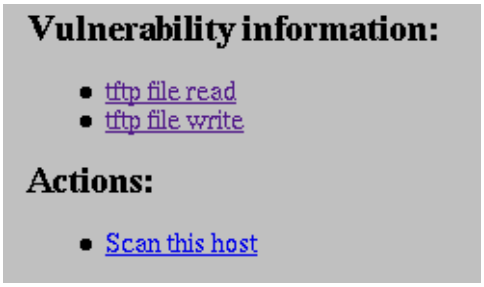

Abbildung 7.4: Ein Fehler in trivial-ftp wurde gefunden.

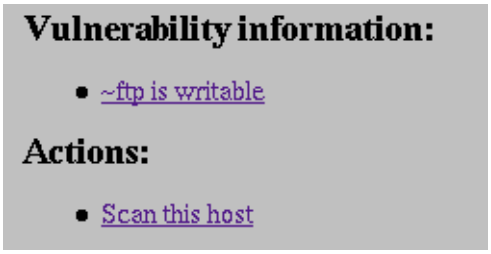

Abbildung 7.5: Ein Fehler in anonymous-ftp wurde gefunden.

bei *satan*– im Vergleich zu *tcpdump* – feststellbar, wer (immer root) damit gearbeitet hat und von welchem Rechner aus *satan* gestartet wurde.

Aufgrund seiner Bedienweise über die graphische Oberfläche mit einem HTML-Browser als Frontend ist *satan* kein Tool für den täglichen Einsatz. Dennoch sollte es regelmäßig – insbesondere nach der Inbetriebnahme von zusätzlichen Rechnern – vom Netzwerk- oder UNIX-Administrator verwendet werden. Trotz seines relativ hohen Alters ist *satan* ein Muß im sicheren Rechnernetz.

7.2 pcops

Jürgen Mayerhofer

Quelle:
ftp://ftp.umd.umich.edu/people/nhughes/banner/cops/

Pcops ist ein Scanner zum Aufspüren von bekannten Sicherheitslöchern auf dem Unix-Rechner. Jeder Administrator sollte von der Existenz dieses Werkzeuges wissen, da es nicht nur von ihm (dem Administrator) sondern auch von jedem Benutzer auf dem System ausgeführt werden kann. *Pcops* prüft

- Datei- Verzeichnis und Gerätedatei (/dev) Zugriffsberechtigungen
- schlechte Passwörter
- Inhalt, Format und Sicherheit von „passwd“- und „groups“-Dateien
- Programmen und Dateien, die von „/etc/rc*“-Skripten und „crontab“ gestartet werden
- das Vorhandensein von SETUID-root-Dateien, deren Schreibrechte und ob sie Shell-Skripten sind oder nicht
- CRC-Summen wichtiger Binaries oder Key-Dateien, um dort Veränderungen festzustellen
- Beschreibbarkeit der Home-Verzeichnisse der Benutzer und deren Startup-Dateien (.profile, .cshrc, .rhosts, etc.)

- anonymous-ftp Setup
- uneingeschränkets *tftp*, alias-Decodierung in sendmail, SUID *uuencode*-Probleme, versteckte Shells in „inetd.conf“, „rexd“ aus „inetd.conf“ aufgerufen
- verschiedene root-Tests – beinhaltet der Suchpfad das momentane Verzeichnis, ein „+“ in „/etc/host.equiv“, uneingeschränkte NFS-mounts, Sicherstellen, daß root in „/etc/ftpusers“ enthalten ist, etc.

und einige weitere Punkte. Es wird ein Bericht generiert, der auf die gefundenen Schwachstellen hinweist. *Pcops* versucht nicht die Schwachstellen auszunutzen oder zu beheben. Dies bleibt dem Administrator überlassen. *Pcops* ist eine Implementierung von COPS, deren Ziel es ist, auf möglichst vielen Systemen leicht handhabbar zu sein.

7.2.1 Installation

Wie so viele andere Tools benützt auch *pcops* die Skriptsprache perl. Da alle Skripten perl aus dem aktuellen Verzeichnis aus aufrufen, ist folgender Link anzulegen:

```
ln -s ‘which perl‘ perl
```

7.2.2 Konfiguration

Die Konfiguration des Paketes erfolgt über die Datei „cops.cf“. Die mitgelieferte Datei genügt den ersten Ansprüchen und ist gut dokumentiert. Da *pcops* nur testet, muß es nicht als „root“ gestartet werden. Lediglich der in der Konfigurationsdatei ausgeklammerte SUID-Checker erfordert „root“-Rechte, damit er alle Verzeichnisse erreichen kann. Es ist gedacht, daß *pcops* einmal täglich gestartet werden soll. Dies kann über *cron* automatisiert abgearbeitet werden. Für den Fall, daß der erstellte Bericht von dem des Vortages abweicht ist es möglich, sich über Mail benachrichtigen zu lassen. Die Einstellung wird in „cops.cf“ getroffen und betrifft die Zeilen:

```
   $MMAIL      = 1;   # send mail instead of generating
reports
   $SECURE_USERS  = ’jmayerho@mozart’; # users to receive
report
```

7.2.3 Test

Zum Starten wird „./cops“ aus dem beim Entpacken entstandenen Verzeichnis aufgerufen. Der Testlauf legt in der Konfiguration ohne Mailbenachrichtigung ein Unterverzeichnis mit dem Namen des Rechners an und legt dort den Bericht in einer Datei, dessen Name das aktuelle Datum ist, ab. Abbildung 7.6 zeigt einen Auszug aus einem Bericht.

Abbildung 7.6: Auszug aus einem Pcops-Bericht

7.2.4 Bewertung

Obwohl *pcops* ein relativ altes Scanner-Tool ist, darf es in keinem UNIX-Administrator-Werkzeugkasten fehlen. Wenn auch nicht alle Warnungen gefixt werden dürfen, so ist es doch sehr wichtig, auf die Veränderungen der Reports zwischen den Tagen zu achten. Ein Hacker und eine Fehlkonfiguration kann unproblematisch aufgespürt werden. Ich weise nochmals darauf hin, daß jeder Benutzer in der Lage ist, die Fehler im System zu finden und auszunutzen. Deshalb ist es ein Muß für den Administrator.

7.3 iss

Jürgen Mayerhofer

Quelle:

ftp://ftp.cert.dfn.de/pub/tools/net/iss/iss13.tar.gz

Das Aufspüren von bekannten Sicherheitslöchern über das Netz soll durch den Internet Security Scanner *iss* ermöglicht werden. Er wurde so entwickelt, daß er auf möglichst vielen UNIX-Systemen funktioniert. *iss* liefert dem Administrator Informationen, mit denen er offensichtliche Fehlkonfigurationen beheben kann. Über viele dieser Sicherheitslöcher wurde bereits in den CERT und CIAC advisories berichtet.

7.3.1 Installation

Die Installation erfolgt durch Compilieren des Sourcecodes, nachdem dieser erfolgreich entpackt ist. Durch den Aufruf von *make* wird ein Binary namens *iss* erzeugt.

7.3.2 Test

Die Handhabung von *iss* ist unkompliziert: Als Parameter werden die IP-Adresse des Rechners, bei dem die Fehlersuche beginnen soll und die IP-Adresse des Rechners, bei dem sie enden soll, angegeben. Der Aufruf von *iss* ohne Parameter zeigt eine Kurzübersicht der vorhandenen Optionen. So wird durch den Aufruf von

```
iss 128.128.128.1 128.128.128.25
```

der Bereich von 128.128.128.1 bis 128.128.128.25 durchsucht.

7.3.3 Bewertung

Der Autor Christopher Klaus weist ausdrücklich in dem mitgelieferten „readme.iss“ daruf hin, daß es sich bei dem frei verfügbaren Security-Scanner aus dem Jahr 1995 um eine eingeschränkte Demo-Version der kommerziellen Version handelt. Diese ist

über die Web-Page http://iss.net/iss erhältlich. Die dort erhältlichen Informationen versprechen viel über das ständig upgedatete Produkt. Auch hier wird wieder eine Testversion angeboten. Der Download scheiterte jedoch an einem nicht funktionierenden Eingabeformular.

Die frei verfügbare Version brachte in unserem Test keine Fehler an den gescannten Rechnern. Die Vermutung liegt nahe, daß das Produkt – nicht mehr auf dem heutigen Stand – keine Sicherheitslöcher aufspüren kann.

Kapitel 8

Diverse Hilfsprogramme

8.1 xtail

Jürgen Mayerhofer

Quelle:

ftp://ftp.crim.ca/lude-crim/xtail-2.1/src/orig

Wie betrachtet man Logfiles? Die gebräuchlichste Methode ist „tail -f dateiname“. Was aber nun, wenn man viele Logfiles hat? Unser Tip: *xtail*. Oder für nicht BSD-Systeme *uxtail*. Letzteres ist lediglich eine Abwandlung von *xtail* mit System V Anpassungen. Der Output erfolgt nicht, wie der Name vielleicht vermuten läßt, in einem Programm mit X-Window-Oberfläche (Menüleiste, etc...), sondern in der Terminalemulation, in der es aufgerufen wird.

8.1.1 xtail – Mehrere Logfiles in einem Output beobachten

xtail beobachtet eines oder mehrere Dateien und zeigt alle Daten an, die seit dem Aufruf von *xtail* in diese Dateien geschrieben worden sind. Wenn der übergebene Parameter ein Verzeichnisname ist, so werden alle Dateien in diesem Verzeichnis, auch jene, die erst nach dem Aufruf erstellt werden, beobachtet. Existiert der Name, der als Parameter übergeben wird nicht, so wartet *xtail* darauf, daß die Datei bzw. das Verzeichnis erzeugt wird.

8.1.2 Installation

Die Installation erfolgt durch Compilieren des Sourcecodes. Die Zielpfade werden durch Editieren des Makefiles konfiguriert.

8.1.3 Parameter

Als Parameter werden der zu überwachende Pfadname oder/und einer oder mehrere Dateinamen übergeben. Auch Wildcards sind erlaubt.

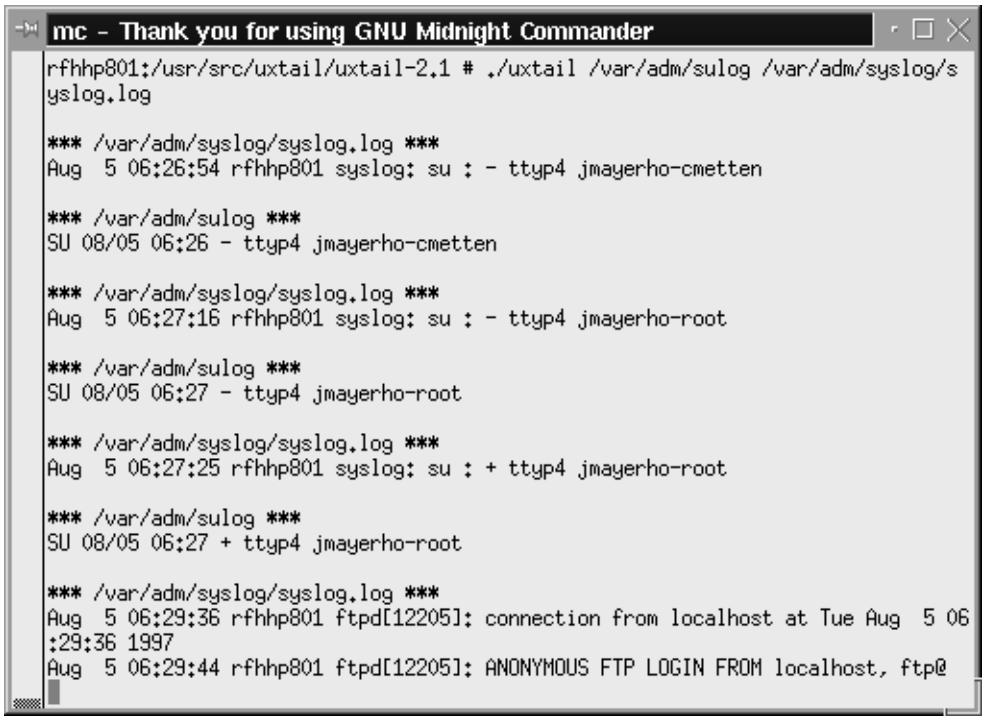

Abbildung 8.1: xtail - Ein Beispiel.

8.1.4 Beispiel

In Abbildung 8.1 werden auf einem HP-UX 10.10-Rechner die Dateien */var/adm/ sulog* und */var/adm/syslog/syslog.log* gleichzeitig betrachtet.

8.1.5 Erfahrungsbericht

Das Hauptproblem bei *xtail* ist das Finden der Abbruch-Tastenkombination. Hierfür muß man erst die Manpage um Rat fragen, bis man findet, daß CTRL-C lediglich die zuletzt veränderten Dateien anzeigt und zum Beenden nicht SIGINT sondern SIGQUIT verwendet wird. „stty -a“ gibt Auskunft über die aktuelle Tastaturbelegung. Meistens verbirgt sich CTRL-Backspace (auf deutschen Tastaturen die Tastenkombination Strg-AltGr-ß) hinter dem Quit-Signal. Sollte diese einmal nicht funktionieren, so kann man auch den Prozess mit CTRL-Z in den Hintergrund setzen und anschließend den Job killen.

8.1.6 Bewertung

Klein aber fein.

8.2 sudo

Jürgen Mayerhofer

Quelle:

ftp://ftp.uni-trier.de/pub/unix/security/cu-sudo

Viele Programme können nur als Superuser ausgeführt werden. Wenn mehrere User das Root-Paßwort besitzen, läßt sich nicht mehr nachvollziehen, wer wann welche Datei verändert hat. Es besteht zwar die Möglichkeit, eine zweite Kennung mit User-ID 0 einzurichten, aber außer an der Uhrzeit im lastlog und dem Datei-Zeitstempel läßt sich kein Rückschluß auf den Übeltäter einer Fehlkonfiguration ziehen.

sudo erlaubt einzelnen Usern das privillegierte Ausführen einzelner Kommandos, ohne Schreibzugriff auf alle Dateien im root-Dateibaum vergeben zu müssen.

Die offizielle *sudo*-Homepage befindet sich auf:

http://www.courtesan.com/courtesan/products/sudo/.

8.2.1 Installation

Die Installation erfolgt über *./configure* (siehe GNU-autoconf unter *squid* Kapitel 2.3.2). Dann *make*, danach *make install* als root aufrufen.

8.2.2 Konfiguration

Die Konfiguration erfolgt über die Datei */etc/sudoers*, welche als root mit dem Kommando *visudo* bearbeitet wird. Regeln können gruppiert werden. Ein erweiterter „vi“ wird aufgerufen, der beim Verlassen die Syntax der *sudoers*-Datei überprüft. Das Paradebeispiel aus der Homepage von *sudo* sieht folgendermaßen aus:

```
#
# Sample /etc/sudoers file.  (Assumes SunOS 4.x paths)
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the man page for the details on how to write a sudoers file.
#

##
# User alias specification
##
User_Alias      FULLTIMERS=millert,mikef,dowdy
User_Alias      PARTTIMERS=bostley,jwfox,mccreary

##
# Runas alias specification
##
Runas_Alias     OP=root,operator
```

```
   ##
   # Cmnd alias specification
   ##
   Cmnd_Alias      DUMPS=/usr/etc/dump,/usr/etc/rdump,/usr/etc/restore,\
                         /usr/etc/rrestore,/usr/bin/mt
   Cmnd_Alias      KILL=/usr/bin/kill
   Cmnd_Alias      PRINTING=/usr/etc/lpc,/usr/ucb/lprm
   Cmnd_Alias      SHUTDOWN=/usr/etc/shutdown
   Cmnd_Alias      HALT=/usr/etc/halt,/usr/etc/fasthalt
   Cmnd_Alias      REBOOT=/usr/etc/reboot,/usr/etc/fastboot
   Cmnd_Alias      SHELLS=/usr/bin/sh,/usr/bin/csh,/usr/bin/ksh,\
                          /usr/local/bin/tcsh,/usr/ucb/rsh,\
                          /usr/local/bin/zsh
   Cmnd_Alias      SU=/usr/bin/su
   Cmnd_Alias      VIPW=/usr/etc/vipw,/etc/vipw,/bin/passwd

   ##
   # Host alias specification
   ##
   Host_Alias      SUN4=bruno,eclipse,moet,anchor
   Host_Alias      SUN3=brazil,columbine
   Host_Alias      DECSTATION=wilkinson,soma,dendrite,thang
   Host_Alias      DECALPHA=widget,thalamus,foobar
   Host_Alias      HPSNAKE=boa,nag,python
   Host_Alias      CSNETS=128.138.243.0,128.138.204.0,128.138.242.0
   Host_Alias      CUNETS=128.138.0.0/255.255.0.0

   ##
   # User specification
   ##

   # root and users in group wheel can run anything on any machine
as any user
   root            ALL=(ALL) ALL
   %wheel          ALL=(ALL) ALL

   # full time sysadmins can run anything on any machine without
a password
   FULLTIMERS      ALL=NOPASSWD:ALL
   # part time sysadmins may run anything except root shells or su
   PARTTIMERS      ALL=ALL,!SU,!SHELLS

   # rodney may run anything except root shells or su on machines
in CSNETS
```

```
rodney          CSNETS=ALL,!SU,!SHELLS

# smartguy may run any command on any host in CUNETS
# (call B address)
smartguy        CUNETS=ALL

# operator may run maintenance commands and anything
# in /usr/oper/bin/
operator        ALL=DUMPS,KILL,PRINTING,SHUTDOWN,HALT,REBOOT,\
                /usr/oper/bin/

# joe may su only to operator
joe             ALL=/usr/bin/su operator

# pete may change passwords for anyone but root
pete            ALL=/bin/passwd [A-z]*,!/bin/passwd root

# bob may run anything except root shells or su on the sun3 and
# sun4 machines as any user in then Runas_Alias "OP"
# (contains root and operator)
bob             SUN4=(OP) ALL,!SU,!SHELLS:\
                SUN3=(OP) ALL,!SU,!SHELLS

# jim may run anything on machines in the biglab netgroup
jim             +biglab=ALL

# users in the secretaries netgroup need to help manage the printers
+secretaries    ALL=PRINTING

# fred can run /bin/ls as oracle by specifying -u oracle on
# command line;
# he can also run /bin/date as uid -2 without entering a password
fred            ALL=(oracle) /bin/ls,(#-2) NOPASSWD:/bin/date

# somedude may su to anyone but root and must give su an argument
somedude        ALL=/usr/bin/su ?*, !/usr/bin/su root
```

8.2.3 Beispiel

Der User jmayerho soll auf dem Rechner jmpc03 *tcpdump* ausführen können. Zunächst ist nur root zum Ausführen berechtigt. Abbildung 8.2 zeigt das Scheitern des Versuches von jmayerho, tcpdump auszuführen.

Der Superuser definiert nun mit Hilfe von *visudo* eine einfache Regel, die jmayerho den Zugriff erlaubt (siehe Abbildung 8.3).

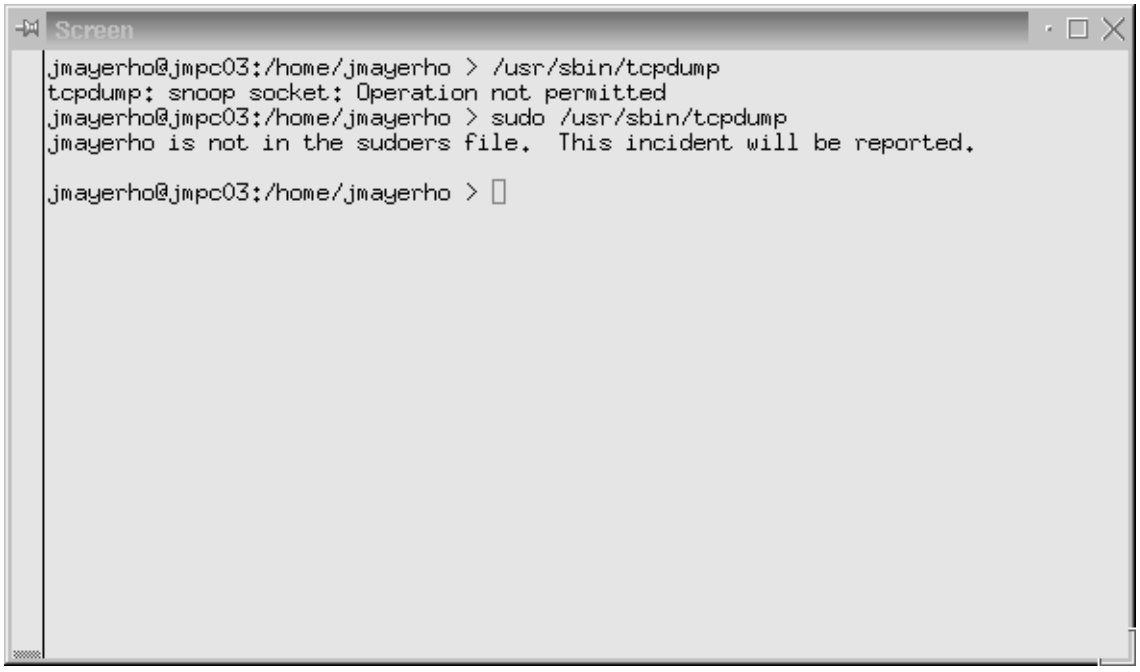

Abbildung 8.2: sudo ohne Berechtigung

Abbildung 8.3: visudo

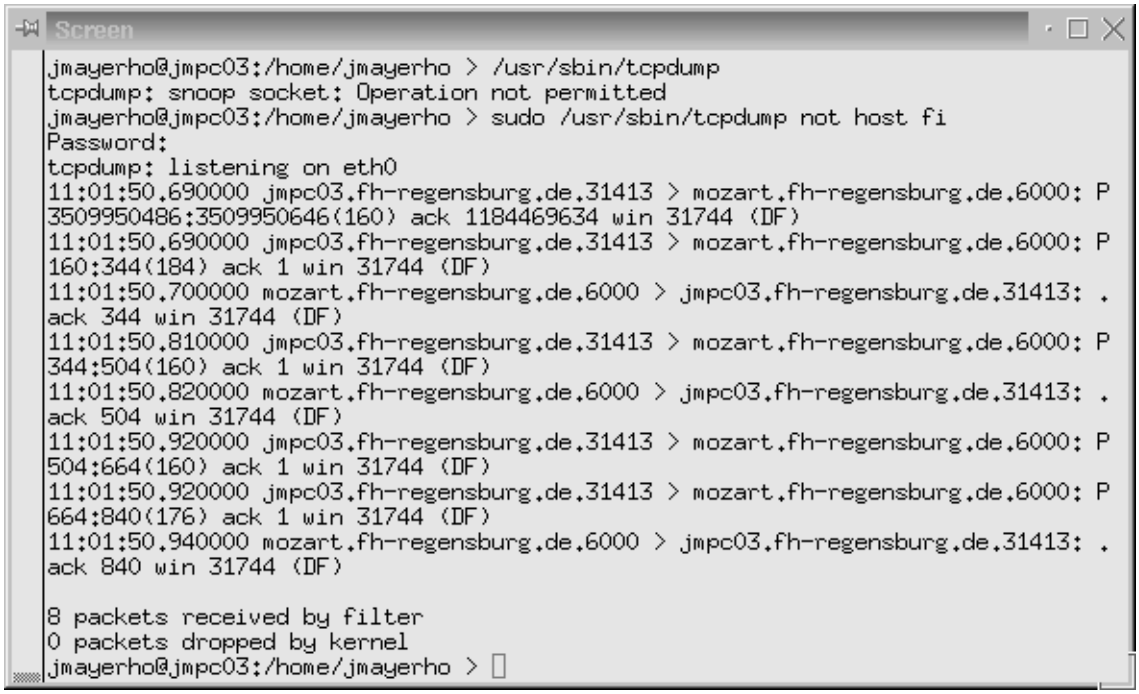

Abbildung 8.4: sudo mit Berechtigung

Jetzt versucht es jmayerho noch einmal. Abbildung 8.4 zeigt den erfolgreiche Ausführung von *sudo*. Beim jeweils ersten Versuch muß sich der Benutzer mit seinem Passwort authentifizieren.

Für die Syntax von *tcpdump* siehe Kapitel 8.3.4. Der Host „fi“ wurde ausgeklammert, weil von dort die Telnet-Session lief, deren Daten nicht beobachtet werden sollten. Der Output zeigt, daß von jmpc03 nach mozart eine X-Appliktion umgelenkt wurde.

8.2.4 Erfahrungsbericht

sudo ist ein kleines, nützliches Tool, das schnell konfiguriert ist. Jedoch kann man an dem kleinen Detail verzweifeln , daß der aufzurufende Befehl (hier *tcpdump*) sich entweder im PATH befinden muß oder mit kompletten Pfad aufgerufen werden muß. Hier bin ich „unnötigerweise“ ein paar Stunden daran gesessen, habe aber daraus als Konsequenz gezogen, mehr Beispiele anzugeben.

8.2.5 Bewertung

Mit Hilfe von *sudo* lassen sich viele Sicherheitslücken schließen. Der jeweils erste Aufruf von *sudo* nach der Bearbeitung der Datei *sudoers* setzt die Leseberechtigung so, daß niemand darin spionieren kann, welche Berechtigungen andere User haben.

```
root@jmpc03:/home/jmayerho # ls -la /etc/sudoers
-r--------   1 root     root     280 Aug  4 11:05 /etc/sudoers
root@jmpc03:/home/jmayerho #
```

Es soll so vermieden werden, daß ein Anwender versucht eine Kennung aufzuhacken, die geegnete Zugriffe auf das restliche System erlaubt.

8.3 tcpdump

Jürgen Mayerhofer

Quelle:

ftp://ftp.cert.dfn.de/pub/tools/net/tcpdump

Die Computer, die an das Internet angebunden sind, benützen hierfür ein Netzwerkinterface. Dies kann entweder die serielle oder parallele Schnittstelle, ein Ethernet, ein Tokenring, FDDI, ATM oder eine beliebige Kombination der angegebenen Interfaces sein. Hin und wieder ist es erforderlich, daß der Verkehr, der an einer bestimmten Schnittstelle stattfindet, verfolgt und analysiert wird. Sei es zur Fehlerverfolgung oder um eine bestimmte (hier feindseelige) Verbindung zu ermitteln.

tcpdump hilft solche Pakete zu finden und einzugrenzen.

8.3.1 tcpdump – Datenverkehr auf Netzwerkinterfaces beobachten

tcpdump ist ein kostenlos erhältliches, einfaches Software-Paket, mit dessen Hilfe es möglich ist, den Datenverkehr auf einem Ethernet LAN zu beobachten. *tcpdump* arbeitet, indem es jedes Paket, das das Netzwerkinterface des Computers passiert, andumpt. Die Größe des Dumps kann vom Anwender eingestellt werden und beträgt im Ausgangszustand 64 bytes. Bei einem Ethernet LAN werden Quell- und Ziel-MAC-Adresse, das Ethernet-Typ- und Längen-Feld und ein paar Byte vom Ethernet-Daten-Feld gedumpt. *tcpdump* dekodiert anschließend das Ethernet-Typ-Feld, um herauszufinden, welches Protokoll im Daten-Feld übertragen wurde.

Weil das Ethernet Daten-Feld ausgewertet wird, ist *tcpdump* somit nicht nur auf den IP-Verkehr beschränkt, sondern beherrscht darüberhinaus auch DECnet und andere Protokolle. Wie der Name aber schon sagt, wurde *tcpdump* dafür optimiert, IP-Datenverkehr zu beobachten. Deshalb beziehen sich die meisten Filter auf IP-Pakete.

8.3.2 Installation

Die Installation von *tcpdump* setzt *libpcap* vorraus.

8.3.2.1 libpcap-Installation

Quelle:

ftp://ftp.cert.dfn.de/pub/tools/net/tcpdump

Libpcap ist ein system-unabhängiges Interface zum Abfangen von Paketen auf Anwenderebene auf unterer Netzwerkebene. Weil verschiedene Hersteller unterschiedliche Interfaces zum Dumpen von Datenpaketen anbieten und weil es mehrere Tools gibt, die diese Funktionalität erfordern, hat die Network Research Group vom Lawrence Berkeley National Laboratory *libpcap* als systemunabhängige Programmierschnittstelle geschaffen.

Die Installation erfolgt mit Hilfe des *autoconf*-Paketes von GNU (siehe *squid*, Kapitel 2.3.2). Sollen mehrere Pakete installiert werden, die *libpcap* benützen, so empfiehlt sich die Installation auf ein gemeinsam nutzbares Verzeichnis.

Weil wir *libpcap* lediglich zum Übersetzen von *tcpdump* benötigten, genügte folgende Vorgehensweise:

```
rfhhp801:/usr/src/tcpdump $ ls
libpcap-0.4a1.tar.gz  tcpdump-3.4a3.tar.gz
rfhhp801:/usr/src/tcpdump $ tar xvfz libpcap-0.4a1.tar.gz
...
rfhhp801:/usr/src/tcpdump $ cd libpcap-0.4a1
rfhhp801:/usr/src/tcpdump/libpcap-0.4a1 $ ./configure
...
rfhhp801:/usr/src/tcpdump/libpcap-0.4a1 $ make
...
```

Es wird nun die statisch gelinkte Library *libpcap.a* erzeugt, die in das Binary von *tcpdump* eincompiliert wird.

8.3.2.2 tcpdump-Installation

Auch *tcpdump* benützt GNU-*autoconf* (siehe *squid* Kapitel 2.3.2). So wird *tcpdump* installiert:

```
rfhhp801:/usr/src/tcpdump $ tar xvfz tcpdump-3.4a3.tar.gz
...
rfhhp801:/usr/src/tcpdump $ cd tcpdump-3.4a3
rfhhp801:/usr/src/tcpdump/tcpdump-3.4a3 $ ./configure
...
rfhhp801:/usr/src/tcpdump/tcpdump-3.4a3 $ make
...
```

Der `--prefix`-Parameter ist optional und beschreibt das Zielverzeichnis (siehe Kapitel 2.3.2 bei Squid). Es wird ein statisch gelinktes Binary namens *tcpdump* erzeugt. Leider werden bei diesem Paket bei der Angabe von `--prefix /soft/tcpdump-3.4a3`

die Installationsverzeichnisse nicht über das Makefile erzeugt, so daß vor dem Aufruf von *make install* und *make install-man* das Anlegen dieser Verzeichnisse erforderlich ist:

```
rfhhp801:/usr/src/tcpdump/tcpdump-3.4a # mkdir /soft/tcpdump-3.4a3
rfhhp801:/usr/src/tcpdump/tcpdump-3.4a # cd /soft/tcpdump-3.4a3
rfhhp801:/soft/tcpdump-3.4a3 # mkdir sbin
rfhhp801:/soft/tcpdump-3.4a3 # mkdir man
rfhhp801:/soft/tcpdump-3.4a3 # mkdir man/man1
rfhhp801:/soft/tcpdump-3.4a3 # cd /usr/src/tcpdump/tcpdump-3.4a
rfhhp801:/usr/src/tcpdump/tcpdump-3.4a # make install
rfhhp801:/usr/src/tcpdump/tcpdump-3.4a # make install-man
```

Der Aufruf von make install erfolgt als Superuser (root). Die Rechte sehen wie folgt aus:

```
-r-xr-x---   1 bin      root       287764 Jul 29 14:52 tcpdump
```

8.3.2.3 iptraf

Quelle:

ftp://unixfe.rl.ac.uk/pub/ppncg/iptraf.tar.Z

iptraf ist eine Sammlung von *perl*-Skripten, mit denen es möglich ist, komplette Netzstatistiken für den lokalen Datenverkehr zu erstellen. Diese Skripten sind sehr nützlich zum Aufspüren von Fehlern lokaler Stationen. Für unseren Anwendungsbedarf ist jedoch keine Statistik der Netzlast gefragt, sondern die unberechtigten Zugriffe von fremden Netzen. Mit kleinen Abwandlungen und geeigneteten Filterregeln läßt sich das relativ schnell erreichen.

8.3.3 Konfiguration

tcpdump erfordert den Zugriff auf das Netzwerkdevice. Dieses trägt auf verschiedenen Betriebssystemen unterschiedliche Namen und kann meistens nur von „root“ gelesen werden. Dabei sollte es auch belassen werden. Die Mächtigkeit dieses Werkzeuges reicht bis hin zum Mitlesen von unverschlüsselten Paßwörtern. Das soll keinem Normalverbraucher erlaubt werden.

8.3.4 Parameter

Die wichtigsten habe ich im folgenden beschrieben. Für detailliertere Dokumentation sei auf die Manpage verwiesen.

tcpdump [-adeflnNOpqStvx] [-c count] [-F file] [-i interface] [-r file] [-s snaplen] [-T type] [-w file] [expression]

-a: versucht die Netzwerkadresse und Broadcastadresse in Namen umzuwandeln.

-c: Beendet tcpdump nach **count** Paketen.

-e: Schreibt den link-level header auf jede Zeile.

-F: Benützt **file** anstelle des Filter-Ausdruckes (expression). Ein zusätzlicher Filter in der Komandozeile wird ignoriert.

-l: erzeugt die Ausgabe auf stdout zeilengepuffert (für zeilenweises Lesen z.B. nach Umlenken der Ausgabe auf Datei und Lesen mit tail: „tcpdump -l > dat & tail -f dat“.

-n: Unterbindet das Umwandeln von Adressen (z.B. Hostnamen, Portnummern, etc.) in Namen.

-N: Läßt bei der Ausgabe den Namen der Domain weg.

-q: Kurzer Output mit weniger Information.

-r file: Liest die Pakete von dem **file**, das vorher mit **-w** erzeugt wurde. Falls **–** als **file** angegeben wird, erfolgt die Eingabe über stdin.

-S: schreibt absolute statt relative TCP-sequence-Nummern.

-t: unterdrückt das Schreiben eines Zeitstempels vor jeder Zeile.

-v: mehr (verbose) Informationen

-vv: noch mehr (verbose) Informationen. Z.B. zusätzliche Felder bei der Übertragung von NFS-Antwortpaketen.

-w file: Schreibt die Pakete auf **file** im „Raw“-Format anstatt sie zu parsen und auszugeben. Das **file** kann anschließend mit Hilfe der Option **-r** geparst werden. Falls **-** als **file** angegeben wird, erfolgt die Ausgabe über stdout.

-x: Schreibt die Ausgabe im Hex-Format ohne linklevel-Header.

8.3.5 Beispiel

Eine geeignete Filterregel sieht z.B. wie folgt aus:

```
tcpdump -n -q -t ip and not net 194.95.108.0 and not host rfhhp801
and not port 3128 and not port 80
```

und bedeutet im Einzelnen:

Abbildung 8.5: tcpdump - Die Netzwerkschnittstelle abhorchen

Gib mir die IP-Adressen und Portnummern aller Verbindungen, die nicht aus dem lokalen Netz stammen und weder die Proxy-Server-Portnummer noch die HTTP-Portnummer verwenden. Der Output ist in Abbildung 8.5 zu sehen.

Am Anfang der Zeile steht jeweils die fortlaufende Uhrzeit. Die Dauer der Messung hat also nur 9 Sekunden gedauert. Dann wurde mit CTRL-C abgebrochen.

8.3.6 Erfahrungsbericht

Der Output von *tcpdump* ist äußerst gesprächig. Es ist unbedingt erforderlich, zusätzliche Tools zum Auswerten von *tcpdump*-Logfiles zu installieren.

Leider funktioneiert *tcpdump* nicht auf Token-Ring, was sich mit seiner Arbeitsweise begründen läßt: Es erzeugt keine eigenen Pakete, sondern horcht lediglich die Leitung ab. Auf Token-Ring kann der Datenverkehr theoretisch nur dann abgehorcht werden, wenn der Agent zwischen dem Sender und dem Empfänger sitzt.

Auf seriellen Verbindungen mit *ppp* bzw. *slip* funktioniert *tcpdump* ebenso unproblematisch wie auf Ethernet.

8.3.7 Bewertung

Die Möglichkeiten, die dieses Tool bietet, sind nahezu grenzenlos. Mit Hilfe von *tcpdump* läßt sich sehr leicht ein kostenloser RMON-Agent implementieren. Die mitgedumpten Pakete können komplett zerlegt und aufbereitet werden.

Dies bietet gleichzeitig Gefahren. Jeder, der *tcpdump* benützen kann, ist in der Lage den Inhalt der Daten (bis hin zu übertragenen Paßwörtern) auszulesen. Es ist nicht möglich festzustellen, wer im Netzwerk *tcpdump* benützt. Theoretisch ist dazu jeder in der Lage, der einen root-Account auf einem beliebigen UNIX-Rechner im Netz besitzt. D.h. man muß den Personenkreis der UNIX-Administratoren auf die Personen einschränken, denen genügend Vertrauen geschenkt werden kann.

Von vielen wird *tcpdump* für das beste Netzwerktool gehalten, das frei erhältlich ist. So auch von uns.

8.4 traceroute

Jürgen Mayerhofer

Quelle:

ftp://ftp.ee.lbl.gov/traceroute.tar.Z

Hat man ein verdächtiges Paket oder gar einen Angreifer ausfindig gemacht, so will man wissen, wo dieser herkommt. *traceroute* verfolgt den Weg eines Paketes, das an den Rechner geschickt wird, von dem das verdächtige Paket kam.

8.4.1 traceroute-Installation

traceroute benötigt zwar nicht unbedigt *xgraph*, aber es ist nützlich zum Visualisieren der Mittelwerte der Hops von Routen.

8.4.1.1 xgraph

Quelle:

ftp://ftp.cs.toronto.edu/pub/radford/

xgraph erfordert das X Window System und dessen *imake*. Das Makefile wird mittels *xmkmf -a* erzeugt. Dann compilieren mit *make,* installieren mit *make install.*

8.4.1.2 traceroute

traceroute „gehorcht“ dem GNU-*autoconf* System (siehe *squid*, Kapitel 2.3.2). Der Aufruf von *make install* muß als „root“ erfolgen, damit es setuid root installiert werden kann.

8.4.2 Parameter

Auch hier sind wieder nur die gebräuchlichsten Parameter zusammengefaßt. Für mehr Informationen dient die manpage.

traceroute [options] host [packetsize]

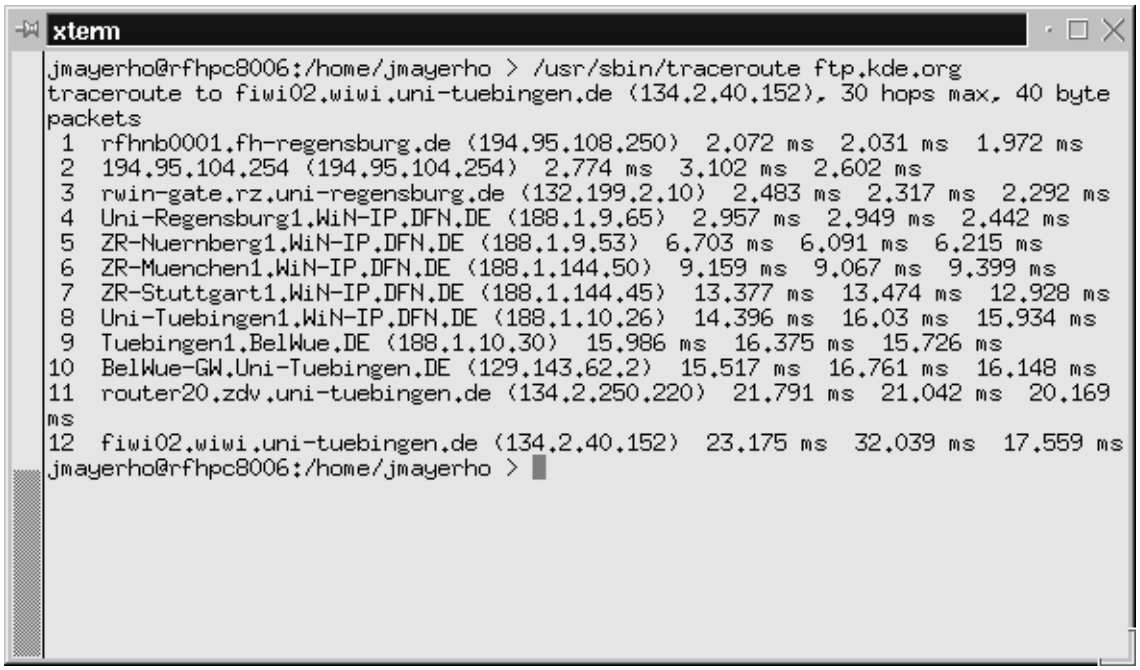

Abbildung 8.6: traceroute - Eine Route durchs Netz

-l: Der time-to-live Wert wird mit angegeben.

-m max_ttl: Setzt die größtmögliche Anzahl von ausgehenden Paketen auf max_ttl hops. Der Defaultwert beträgt 30 hops.

-n: löst die Rechnernamen, die auf dem Weg liegen nicht auf, sondern beläßt die IP-Adressen.

-r: umgeht die normalen Routing-Tabellen und sendet direkt an einen Host, der im lokalen Netz liegt.

-v: Für jedes gesendete Paket werden ICMP-Pakete empfangen. Diese Option zeigt die empfangenen Pakete an.

-w wait: Anzahl der Sekunden, die auf die Antwort gewartet werden soll.

8.4.3 Beispiele

1. Uns interessiert der Weg, den die Route nach „ftp.kde.org“ einschlägt. Abbildung 8.6 zeigt das Ergebnis.

2. Wir wollen die durchschnittliche Übertragungszeit einer getracten Route zwischen den einzelnen Routern ermitteln, um zu sehen, zwischen welchen Geräten die Übertragung besonders lange dauert. Als Hilfsmittel wird das beim Source von

traceroute mitgelieferte *awk*-Skript *median.awk* verwendet. Die Messung wird mit „traceroute -q 7 ftp.kde.org > t" durchgeführt, der Output von sieben Messungen wird also auf die Datei namens *t* umgelenkt. Das *awk*-Skript erzeugt folgendes Ergebnis:

```
jmayerho@Zerberus:/export/home/jmayerho >  awk -f median.awk
t
1 1.948
2 2.701
3 2.249
4 2.414
5 6.761
6 9.524
7 14.643
8 14.808
9 15.575
10 16.543
11 18.031
12 18.074
jmayerho@Zerberus:/export/home/jmayerho > awk -f median.awk t
> graph
```

Ich füge noch einen Titel und die Achsenbeschriftung ein.

```
jmayerho@Zerberus:/export/home/jmayerho >  cat graph
TitleText: Traceroute nach ftp.kde.org
YUnitText: Sendedauer im Schnitt
XUnitText: Router #

1 1.948
2 2.701
3 2.249
4 2.414
5 6.761
6 9.524
7 14.643
8 14.808
9 15.575
10 16.543
11 18.031
12 18.074
jmayerho@Zerberus:/export/home/jmayerho >
```

Abbildung 8.7 zeigt den Graphen, wenn mit *cat graph | xgraph* die Ausgabe umgelenkt oder als Parameter mit *xgraph graph* übergeben wird.

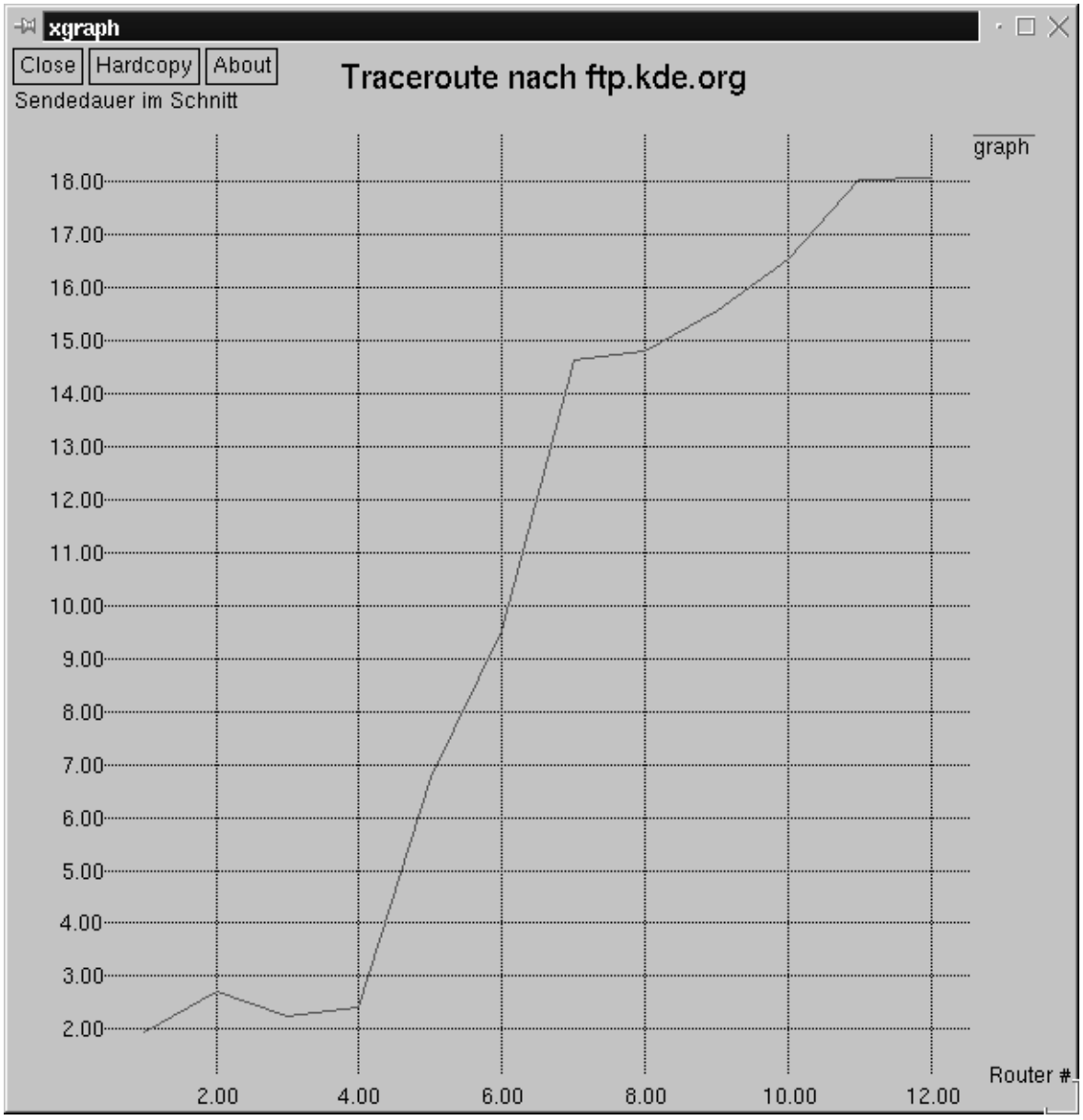

Abbildung 8.7: traceroute - Zeitmessungen als Graph

Es ist deutlich zu erkennen, daß der Flaschenhals der Übertragung zwischen den Routern 4 bis 7 liegt.

8.4.4 Erfahrungsbericht

Die Arbeit mit *traceroute* ist nicht nur für den Alltagsgebrauch geeignet um schlechte Routen zu finden. *traceroute* ist unkompilzert und leicht zu bedienen. Es ist somit sehr effektiv im Einstatz.

8.4.5 Bewertung

Normalerweise gehört *traceroute* zum Lieferumfang von allen Unix-Systemen. Sollte es einmal fehlen oder nicht wie gewünscht reagieren, so hilft die Implementierung der Network Research Group vom Lawrence Berkeley National Laboratory weiter.

Anhang A

Allgemeines

A.1 Versuchsaufbau

Die Grafik wurde mit *tkined* aus dem *scotty*-Paket erstellt und enthält im oberen Bereich den verwendeten Versuchsaufbau. Im unteren Bereich sind einige der an diversen Tests beteiligten Hosts aufgeführt. Die im Versuchsaufbau verwendeten Komponenten sind in der Tabelle A.1 genauer beschrieben.

Interessant ist, daß *db_net (intern)* und *db_net (extern)* jeweils identische IP-Adressen haben, ebenso wie die beiden Netzwerkkarten der Drawbridge. Dies ist in Kapitel 3.1 genauer erläutert. Die Begriffe **intern**, **extern**, **vor** und **hinter** beziehen sich auf die Lage der Netzwerkkomponenten, wobei **intern** und **hinter** auf besser geschützte Netzwerkbereiche hinweisen; anolog deuten **extern** und **vor** auf geringere Sicherheit hin.

Der dual homed Host Zerberus war in der ursprünglichen Aufgabenstellung als Firewall eingesetzt, was sich dann bei der Verwendung als Proxy-Server (für Socks und Squid) als praktisch erwies. Auf Zerberus wurde das Routing nicht aktiviert, so daß es keine direkte Verbindung vom *db_net (extern)* zum *fh_net* gab, dies wurde nur durch den Einsatz von Proxies bewerkstelligt.

Die Drawbridge wurde als Paketfilter eingesetzt, um eine preiswerte Alternative zu teurer Spezialhardware zu testen.

A.2 Dateien

A.3 Dateien (Solaris)

A.3.1 Automatischer Start von Serverprozessen

Um die Softwarepakete des Versuchsaufbaus nicht ständig manuell aufrufen zu müssen, wurden alle Serverdienste bereits beim Hochfahren des Betriebssystems gestartet. Da dies auch für produktiven Einsatz zu empfehlen ist, wird dieser Vorgang hier beschrieben.

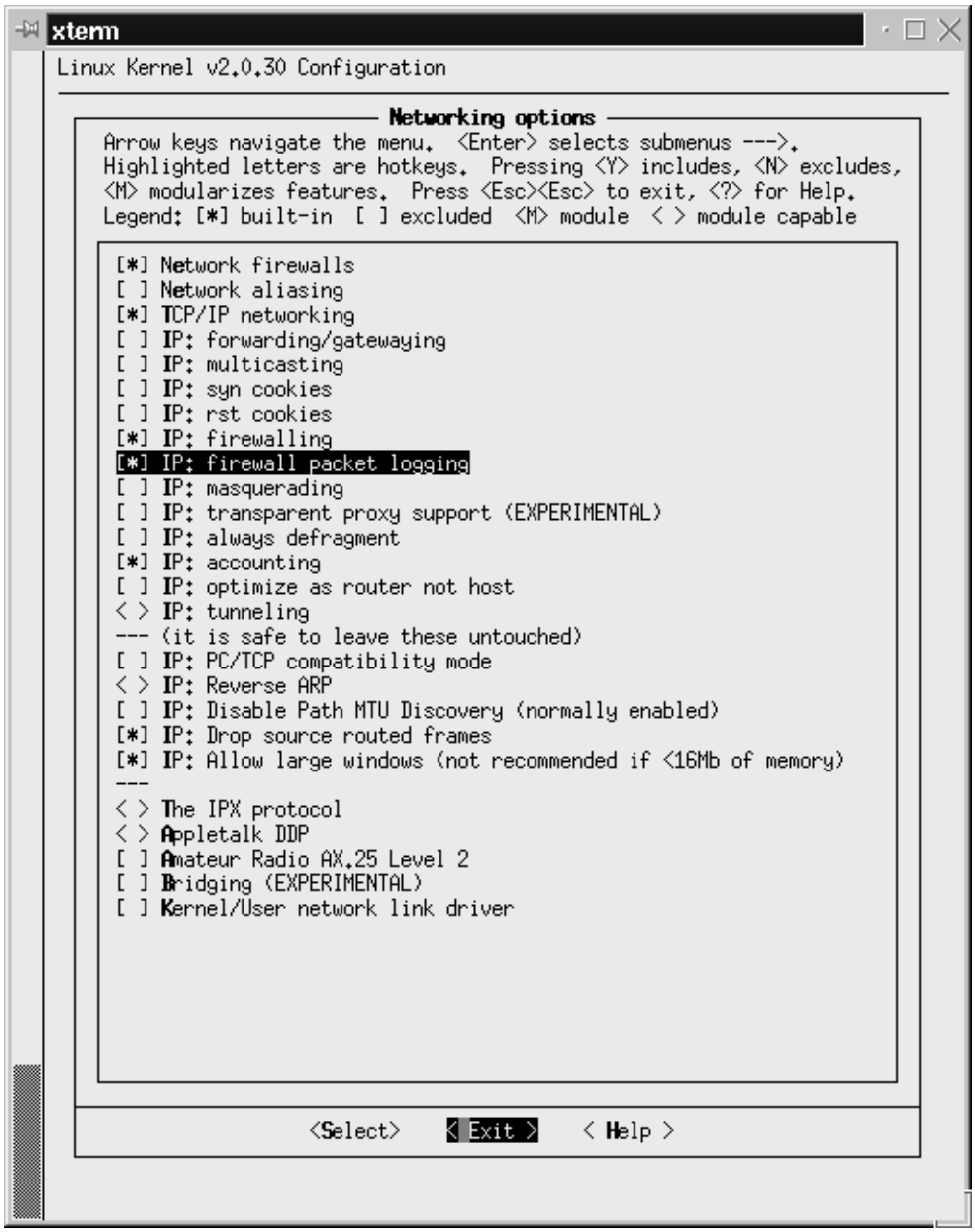

Abbildung A.1: Versuchsaufbau

Name	IP-Adresse(n)	Funktion
db_net (**intern**)	194.95.109.176	Netzwerk **hinter** der Drawbridge
db_net (**extern**)	194.95.109.176	Netzwerk **vor** der Drawbridge
fh_net	194.95.109.208	**externes** Firewall-Netzwerk
Zerberus (**intern**)	194.95.109.177	Firewall-Rechner (ursprünglich),
Zerberus (**extern**)	194.95.109.211	Socks-Proxy (siehe Kapitel 2.2), Squid-Proxy (siehe Kapitel 2.3), SSH-Server (siehe Kapitel 5.2), Drawbridge-Loghost (siehe Kapitel 3.1)
Drawbridge	194.95.109.190 194.95.109.190	'Filtering Bridge' mit zwei Adressen, von denen eine **hinter** der Bridge liegt
jmpc03	194.95.109.181	Entwicklungs-PC (Linux)
Mozart	194.95.109.180	Test-PC (Linux)
cisco (**intern**) cisco (**extern**)	194.95.109.210 194.95.109.226	Router vom fh_net zum Internet

Tabelle A.1: Versuchsaufbau

Unter Sun Solaris/sparc werden sämtliche Serverdienste beim Wechsel in den Multi-User-Level (runlevel 3) gestartet. Die verwendete Batchdatei muß also im Verzeichnis /etc/rc3.d liegen und der Dateiname mit einem **S** beginnen. Die beiden Zahlen, die hinter dem **S** stehen, geben die Reihenfolge der Abarbeitung an (je kleiner desto früher). Im Versuchsaufbau wurden diese Zahlen auf **9** gesetzt, um alle Standard-Server bereits vor dem Aktivieren der Testsoftware hochzufahren, was sich als zweckmäßig erwies. Auf die Zahlen folgt dann noch ein selbst vergebener, möglichst aussagekräftiger Name.

Beim Herunterfahren des Betriebssystems (entweder vollständig oder in den Single-User-Modus) werden in jedem Fall alle Batchdateien ausgeführt, die in */etc/rc2.d* liegen und mit **K** beginnen. Auch hier wird mit zwei Zahlen die Reihenfolge der Abarbeitung festgelegt, wobei **größere** Zahlen zuerst abgearbeitet werden, dann folgt wieder ein aussagekräftiger Name.

Zusätzlich sieht die System V Norm vor, daß alle Batchdateien entweder **start** oder **stop** als Argumente erwarten und entsprechend reagieren.

Im Normalfall wären also zwei Batchdateien erforderlich gewesen, und zwar */etc/rc2.d/K99daemons.server* und */etc/rc3.d/S99daemons.server.*

Wegen der erforderlichen Übergabe von **start** bzw. **stop** konnte jedoch in beiden Fällen das selbe Skript verwendet werden. Es bietet sich an, eine Datei im entsprechenden Verzeichnis zu erstellen und die andere mittels *ln* als Hardlink darauf zu setzen.

Im Versuchsaufbau wurden alle Testserver zusammen gestartet, es wäre für manche Anwendungen eventuell sinnvoller, für jedes Softwarepaket eigene Start- und Stop-Batchdateien zu haben. Die Vorgehensweise ist dabei jedoch gleich.

Für den verwendeten Versuchsaufbau sahen beide Dateien (eigentlich nur eine Datei) dann so aus:

```
#!/bin/sh

if [ ! -d /opt/bin ]
then                    # /opt not mounted
        exit
fi

killproc() {            # kill the named process(es)
        pid=‘/usr/bin/ps -e |
            /usr/bin/grep -w $1 |
            /usr/bin/sed -e ’s/^  *//’ -e ’s/ .*//’‘
        [ ”$pid” != ”” ] && kill $pid
}

case ”$1” in
’start’)
        /opt/SOCKS/bin/sockd
        /opt/SQUID-1.1.11/bin/squid &
        /opt/www/httpd/bin/httpd &
        ;;

’stop’)
        killproc sockd
        killproc httpd
        /opt/SQUID/bin/squid -k shutdown
        ;;
*)
        echo ”Usage: <script> { start | stop }”
        ;;
esac
```

A.3.2 PGP - User Installation

Das folgende kleine Shell-Skript erledigt alle erforderlichen Arbeiten.
Eventuell sollte man den Basispfad der Installation anpassen.

```
#!/bin/sh

BASE=/soft/pgp-2.6.3

mkdir $HOME/.pgp
cp $BASE/lib/system.pgprc $HOME/.pgp
ln -s /soft/pgp-2.6.3/doc $HOME/.pgp

pgp -kg
```

A.4 Dateien (Linux)

Folgendes System V Startup-Skript bietet ein Gerüst zum Starten und Beenden von IP-Masquerading:

```
#!/bin/sh
case $1 in
’start’)
        echo -n ”Starte Masquerading ...”
# Loesche alle Regeln
        ipfwadm -F -f
        ipfwadm -O -f
# Maskiere alle Pakete, die vom lokalen Netz kommen
        ipfwadm -F -i masquerade -S 192.168.208.0/24 -D 0.0.0.0/0
# Keine anderen Pakete durchlassen
ipfwadm -F -i deny -S 0.0.0.0/0 -D 192.168.208.0/24
# Keine Paketek vom lokalen Netz ins Internet
ipfwadm -O -i deny -S 192.168.208.0/24 -D 0.0.0.0/0
# Verbindung vom Router zum lokalen Netz
ipfwadm -F -i masquerade -S 194.95.203.0/24 -D 192.168.208.0/24
        modprobe ip_masq_irc
        echo done
        ;;
’stop’)
        echo -n ”Loesche Masquerade-Regeln”
```

```
        ipfwadm -F -f
        ipfwadm -O -f
        echo done
        ;;
esac
```

Anhang B

Erklärung

1. Hiermit erklären wir, daß wir diese Diplomarbeit selbständig verfaßt, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel verwendet, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet haben.
2. Uns ist bekannt, daß die Diplomarbeit als Prüfungsleistung in den Eigentum des Freistaates Bayern übergeht.
3. Hiermit erklären wir unser Einverständnis, daß die Fachhochschule Regensburg diese Prüfungsleistung von Studenten der Fachhochschule Regensburg einsehen lassen und unter Nennung unserer Namen als Urheber veröffentlichen darf.

Regensburg, den 06.08.1997

Jürgen Mayerhofer

Christian Rotter

Literaturverzeichnis

[1] Sedgewick, Robert:
Algorithmen (2. Nachdruck 1992)
Addison-Wesley, 1991, 1992
ISBN 3-89319-402-9

[2] div. Autoren [u.a. Philip R. Zimmermann, Stale Schumacher]:
Dokumentation von PGP 2.6.3(i)
Internet, 1996, 1997
ISBN keine

[3] div. Autoren [u.a. Tatu Ylonen]:
Dokumentation von SSH 1.2.17
Internet, 1995, 1996, 1997
ISBN keine

[4] div. Autoren [u.a. Timo Rinne, Tatu Ylonen]:
man-pages von SSH 1.2.17: scp, ssh, slogin, u.a.
Internet, 1995, 1996, 1997
ISBN keine

[5] D. Brent Chapman, Elizabeth D. Zwicky:
Building Internet Firewalls
O'Reilly & Associates, Inc., 1995
ISBN 1-56592-124-0

[6] div. Autoren [u.a. Ying-Da Lee,Wei Lu, Steven Lass, Ron Kuris]
Dokumentation von Socks 4.3 (Beta2) und Socks 5 (v1.0r1)
Internet, 1996, 1997
ISBN keine

[7] div. Autoren
Dokumentation (Man-Pages) des Linux- und Solaris-Betriebssystems
ISBN keine

[8] Atkins, Todd:
Dokumentation (Man-Pages) von SWATCH
ISBN keine

[9] Wietse Venema:
Dokumentation (Man-Pages) von TCPD
Internet, 1995, 1996, 1997
ISBN keine

[10] Karanjit Siyan, Chris Hare:
Internet Firewalls & Netzwerksicherheit
SAMS, 1995
ISBN 3-87791-845-X

[11] William R. Cheswick, Steven M. Bellovin
Firewalls und Sicherheit im Internet
Addison-Wesley, 1996
ISBN 3-89319-875-x

[12] RFC – request for comments
Internet, z.B. ftp://ftp.uni-regensburg.de/pub0/RFCs/
ISBN: keine

Index