Rootkits: Techniken und Abwehr

Andreas Bunten
DFN-CERT GmbH
Heidenkampsweg 41
D-20097 Hamburg
bunten@cert.dfn.de

Zusammenfassung

Ein Rootkit ermöglicht einem Angreifer auf einem kompromittierten System unentdeckt zu bleiben, um sich nach Belieben wieder mit dem System zu verbinden und die durch den Einbruch im System erlangten Privilegien wiederherzustellen. Die dabei eingesetzten Techniken haben sich von der Filterung von Logfiles bis zur umfangreichen Manipulation des Kerns des Betriebssystems entwickelt. In diesem Beitrag wird ein Überblick bzgl. der eingesetzten Techniken und der im Gegenzug entstandenen Methoden zur Entdeckung von Rootkits aus dem Unix- und Microsoft-Umfeld gegeben. Exemplarisch wird die Wirksamkeit von Abwehrmaßnahmen in Experimenten untersucht und ein Ausblick über die zu erwartenden Entwicklungen gegeben.

1 Einleitung

Ein typischer Angriff auf ein mit dem Internet verbundenes System kann wie folgt aussehen: Das Ziel wird mit Hilfe von Portscans untersucht, um Dienste mit bekannten Schwachstellen zu erkennen. Kann eine Sicherheitslücke erfolgreich ausgenutzt und können infolge Befehle auf dem System ausgeführt werden, gilt dieses als kompromittiert. Über weitere lokale Schwachstellen kann der Angreifer ggf. seine Zugriffsrechte bis zu denen des Systemadministrators erweitern.

Der Angreifer will das kompromittierte System für seine Zwecke nutzen und dazu Änderungen am System durchführen. Dies kann z.B. die Installation einer *Backdoor* sein, um wieder Zugriff auf das System zu bekommen, ohne dass erneut Schwachstellen ausgenutzt oder die regulären Anmeldeverfahren durchlaufen werden müssen. Bei Entdeckung des Einbruchs wird der Administrator versuchen, das System gegenüber dem Angreifer zu schließen. Dies kann z.B. durch Beseitigung der Sicherheitslücke, eine Neuinstallation oder durch die Trennung des Systems vom Netz geschehen.

Der Angreifer ist daher bestrebt seine Anwesenheit auf dem System, die Spuren des Angriffs und die durchgeführten Modifikationen zu verschleiern. Es werden dazu Sammlungen von speziellen Werkzeugen und Programmen eingesetzt - sog. *Rootkits*. Dieser Beitrag behandelt, unabhängig davon wie der eigentliche Einbruch in das System bewerkstelligt wurde, die Techniken und Methoden von Angreifern, sich den Zugiff auf ein bereits kompromittiertes System mit Hilfe von Rootkits zu bewahren, um dort unentdeckt zu agieren.

In der Einleitung wird im Folgenden die Entwicklungsgeschichte der Rootkits zusammengefasst und einige technische Begriffe schematisch erklärt. Im zweiten Abschnitt werden die relevanten Techniken im Detail zusammen mit Beispielen zu ihrer Abwehr vorgestellt. Die Methoden zur Abwehr und Entdeckung von Rootkits werden im dritten Abschnitt zusammengefasst und bewertet. Im abschließenden vierten Abschnitt werden die Ergebnisse diskutiert und eine Prognose kommender Entwicklungen vorgestellt.

1.1 Eine kurze Geschichte der Rootkits

Die Entwicklung von Rootkits [Ditt2002] begann Ende der 80’er Jahre und betraf hauptsächlich UNIX-artige Systeme wie z.B. SunOS. Es kursierten Programme zur Manipulation der Logfiles, um die Anwesenheit bestimmter User zu verheimlichen. Hierzu sind die Rechte des Systemadministrators - auf Unix-Systemen *root* - nötig. Die Manipulation von Logfiles führte dazu, dass der Ausgabe von Systembefehlen wie `who` oder `last`, welche die angemeldeten User anzeigen, nicht unbedingt vertraut werden konnte.

Austausch von Systembefehlen

Infolge entstand die Praxis, Befehle des Systems auszutauschen, um die Aktivitäten der Angreifer zu verbergen. Es wurde z.B. `ls` ersetzt, um bestimmte Dateien und Verzeichnisse nicht auszugeben, oder `netstat`, um bestimmte Verbindungen zu verheimlichen. Diese ersten explizit als Rootkit bezeichneten Programmsammlungen wurden zuerst für SunOS 4 bekannt, aber bald auf Linux und andere UNIX Varianten portiert. Hierdurch konnten Administratoren Systembefehlen nicht mehr vertrauen, anstatt dass nur Einträge in Logfiles misstraut werden musste.

Erste Rootkits im Kernel

Viele Betriebssyteme besitzen die Möglichkeit Treiber zur Laufzeit in den Kern einzubinden. Mitte der 90’er Jahre kamen *Kernel Rootkits* auf, die durch zur Laufzeit ladbare Kernelmodule den Kern des Betriebssystems manipulierten. Diese neuen Rootkits wurden bald von Linux auf andere UNIX-artige Systeme wie Solaris und die freien BSD Varianten portiert. Auf diese Weise wurde eine Manipulation der Informationen ermöglicht, die der Kernel an die Systembefehle weitergibt, so dass ein Austausch einzelner Programme nicht mehr nötig war. Auf einem anderen System statisch gebundene Programme konnten trojanisierte Systembefehle aufdecken, aber sind auf die gleichen Systemaufrufe angewiesen. Die Entdeckung eines Kernel Rootkits ist am laufenden System nur schwer möglich, da eine beliebige Manipulation des Systems möglich ist.

Ende der 90'er Jahre verbreiteten sich auch Rootkits für die Windows Betriebssysteme von Microsoft. Diese Rootkits wurden auch in den Kern des Betriebssystems eingebunden und nahmen die fortgeschrittenen Methoden der Kernel-Rootkits unter UNIX auf.

Die neue Generation von Kernel-Rootkits

Neuere Entwicklungen von Rootkits unter Linux und den freien BSD Varianten ermöglichen die Installation von Kernel-Rootkits, ohne dass Kernelmodule vom angegriffenen System unterstützt werden müssen. Dazu werden zentrale Strukturen innerhalb des Kernels durch heuristische Suche gefunden und gezielt manipuliert, um den Programmcode der Angreifer zur Ausführung zu bringen. Die im Gegenzug entstehenden Detektionsmethoden müssen zum Teil speziell auf Merkmale einzelner Rootkits zugeschnitten werden.

1.2 Verwendete technische Begriffe

Im Folgenden werden einige Begriffe vorgestellt, die für das Verständnis der Techniken von Rootkits benötigt werden. Die Definitionen sind anwendbar auf die meisten UNIX Varianten und zum Teil auf Windows. Tiefergehende Informationen zu Windows [SR2000], UNIX [Bach1986] und Linux [BC2002] sind in der Literatur zu finden.

Speicherschutz: Ein laufendes Programm kann aufgrund des Speicherschutzes nicht den Speicherbereich eines anderen Programms einsehen oder verändern. Insbesondere kann ein Programm nur unter speziellen Umständen auf den Speicher des Kerns des Betriebssystems zugreifen (siehe auch *Zugriff auf Gesamtspeicher*). Bzgl. der Zugriffsrechte eines Programms wird zwischen *Usermode* und *Kernelmode* unterschieden. Jedes von einem Nutzer gestartete Programm wird im Usermode ausgeführt. Die Funktionen des Betriebssystems können nur über *Systemaufrufe* verwendet werden. Die Ausführung eines Systemaufrufs findet im Kernelmode statt, in dem ohne Einschränkung auf das gesamte System und die Hardware zugegriffen werden kann.

Ablauf eines Systemaufrufs: Wie ein Systemaufruf durchgeführt wird, soll am Beispiel des Kommandos `ls /tmp` unter Linux betrachtet werden, das den Inhalt des Verzeichnisses `/tmp` anzeigt. Das Programms `ls` führt dazu den Systemaufruf `open()` aus, um das betreffende Verzeichnis zum lesen zu öffnen. Dieser Aufruf ist schematisch in Abb. 1 dargestellt und verläuft wie folgt:

1. Das Programm `ls` ruft `open(/tmp, O_RDONLY)` auf. Der erste Parameter gibt an, welches Verzeichnis geöffnet werden soll. Der zweite gibt an, dass lediglich gelesen und nicht geschrieben wird. Die Parameter und die Nummer des gewünschten Systemaufrufs (`open()` hat die Nummer 5) werden in den dafür vorgesehenen Registern des Prozessors abgelegt und der für Systemaufrufe vorgesehene Interrupt `0x80` wird ausgelöst.
2. Das System wechselt zur Interruptbehandlung in den Kernelmode und wählt aus der *Interrupt Descriptor Table* (IDT) den zum Interrupt (`0x80`) gehörenden *Interrupt Handler* aus, und führt diesen aus.

3. Der Interrupt Handler (`0x80`) ist unter Linux für die Ausführung von Systemaufrufen zuständig. Es wird die übergebene 5 in der *Syscall Tabelle* referenziert, in der alle Systemaufrufe verzeichnet sind, und die Funktion `sys_open()` des Kernels mit den übergebenen Parametern aufgerufen.

Als Ergebnis der Funktion wird ein gültiges Filehandle auf ähnliche Weise zurückgegeben. Das Programm `ls` liest dann mit dem Systemaufruf `getdents()` die Einträge des Verzeichnisses `/tmp`, um diese schließlich auszugeben.

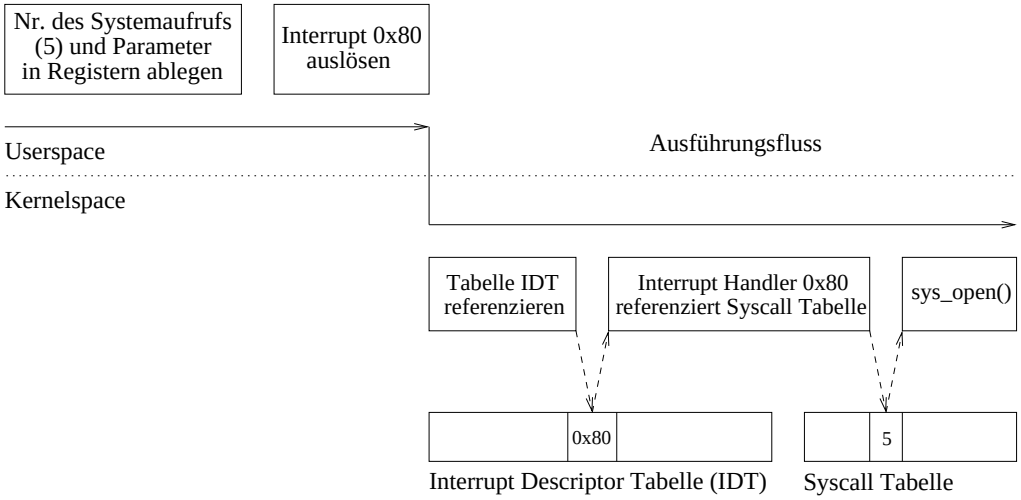

Abbildung 1: Schematischer Ablauf eines Systemaufrufs

Die beschriebenen Grundprinzipien finden sich in allen gängigen Betriebssystemen wieder. Die Verwendung eines Interrupts für Systemaufrufe ist sehr verbreitet, aber es existieren auch andere Methoden. Relevant ist, dass die Auswahl der bei einem Systemaufruf auszuführenden Funktion im Kernel von wenigen zentralen Ressourcen abhängt - u.a. der Interrupt Descriptor Tabelle und der Syscall Tabelle.

Kernelmodule: Die meisten gängigen Betriebssysteme besitzen die Möglichkeit zur Laufzeit zusätzlichen Programmcode in From von *Kernelmodulen* in den Kern zu integrieren. Dies erleichtert die Arbeit des Administrators, da z.B. Treiber für neue Hardware ohne aufwendiges Übersetzen oder Binden des Kernels genutzt werden können. Über die Symboltabellen des Kernels kann das zur Laufzeit hinzugefügte Modul auf die Funktionen und Datenstrukturen des Kernels zugreifen. Die Unterstützung von Modulen lässt sich auch deaktivieren. Es werden dann keine Symboltabellen im Kernel vorgehalten, da die Funktionsaufrufe der festen Bestandteile des Kernels bereits zur Übersetzungszeit feststehen. Die Liste der gerade geladenen Module kann über Befehle im Userspace angezeigt werden. Unter Linux ist auch direkter Zugriff auf diese Liste unter `/proc/modules` möglich.

Zugriff auf Gesamtspeicher: Über die Devices `/dev/kmem` und `/dev/mem` kann auf UNIX-artigen Systemen ein Programm, das die Zugriffsrechte des Administrators besitzt, auf den Speicher des Kernels zugreifen. Je nach Konfiguration des Kernels, ist auch schreibender Zugriff möglich. Wenige Programme im Userspace, wie der X-Server XFree86, benutzen den schreibenden Zugriff für die Ansteuerung von Hardware.

2 Techniken der Rootkits

Im Folgenden werden die im vorherigen Abschnitt kurz vorgestellten Techniken genauer beschrieben. Dabei werden Beispiele für relevante Techniken aufgeführt, ohne Anspruch auf Vollständigkeit erheben zu können.

2.1 Die Manipulation von Logfiles

Programme zur Maniplulation von Logfiles werden seit den ersten Tagen der Rootkits eingesetzt, um den Angriff selber oder die darauf folgenden Aktivitäten zu verheimlichen [BTA1989]. Typischerweise geschieht dies zum Teil automatisch, so dass der Angreifer lediglich Stichworte als Parameter angeben muss, die gesucht und aus den Logfiles gelöscht werden. Dies ist i.d.R. der eigene Loginnamen oder die IP des Systems von dem der Angriff erfolgt. Implementierungen von Programme zur automatischen Manipulation von Logfiles existieren für alle gängigen Betriebssysteme. In Abb. 2 ist die Ausgabe des Programms zum manipulieren von Logfiles aus dem Rootkit tornkit zu sehen.

```
linux:/home/rks # tail -3 /var/log/messages
Jan  3 14:16:10 linux kernel: VFS: Disk change detected on device fd(2,0)
Jan  3 14:19:39 linux sshd[1517]: WARNING: /etc/ssh/primes does not exist,\
   using old prime
Jan  3 14:19:43 linux sshd[1517]: Accepted password for seg from\
   192.168.1.17 port 36262 ssh2
linux:/home/rks # sh t0rnsb seg
 * sauber by socked [07.27.97]
 *
 * Cleaning logs.. This may take a bit depending on the size of the logs.
 * Cleaning messages (414 lines)...1 lines removed!
 * Cleaning boot.log (7mlines)...0 lines removed!
(...)
 * Alles sauber mein Meister !
linux:/home/rks # tail -3 /var/log/messages
Jan  3 14:16:10 linux kernel: VFS: Disk change detected on device fd(2,0)
Jan  3 14:19:39 linux sshd[1517]: WARNING: /etc/ssh/primes does not exist,\
   using old prime
Jan  3 14:20:21 linux syslogd 1.3-3: restart.
```

Abbildung 2: Manipulation der Logfiles

Im Logfile `/var/log/messages` ist zuerst das Login des Users `seg` zu sehen. Das Programm `t0rnsb` wird mit dem Schlüsselwort `seg` gestartet und durchsucht eine Reihe von Dateien. Danach ist die Zeile mit dem angegebenen Schlüssel aus dem Logfile verschwunden. In diesem Beispiel werden die Unzulänglichkeiten solcher Hilfsmittel deutlich: Zu der eigentlichen Meldung gehört auch die vorhergehende Zeile bzgl. der Datei `/etc/ssh/primes`. Diese Zeile bleibt nach der Anwendung des Programms alleine im Logfile zurück, was einem aufmerksamen Administrator bei der Kontrolle der Logfiles auffallen kann. Weiterhin wird der Neustart des Syslog Daemon gemeldet. Dies sollte nicht außerhalb des üblichen Turnus geschehen und ist ebenso verdächtig.

Den vollständigen Kontext um die zu löschenden Meldungen auszuwählen und alle Verdacht erregenden Meldungen zu löschen ist insgesamt eine komplexe Aufgabe. Oft lassen sich daher Hinweise auf den Angriff und die Manipulation des Systems in den Logfiles finden. Die Suche nach derartigen Hinweisen wird z.B. durch das Paket `chkrootkit` [MSJ2002] vorgenommen, das eine Reihe allgemeiner und viele spezielle, auf bestimmte Rootkits zugeschnittene, Tests durchführt, um ein installiertes Rootkit zur Laufzeit zu erkennen. Die oben beschriebenen Unregelmäsigkeiten durch unvollständiges Löschen lassen sich i.d.R. nur durch manuelle Analyse der Logfiles aufdecken. Werden die Daten der Logfiles nicht lokal aufbewahrt, sondern direkt an einen zentralen Loghost geschickt, wird die Entdeckung eines Rootkits erheblich erleichtert. Sobald der Angreifer die Rechte des Administrators erlangt hat, kann das Logging auf dem kompromittierten System gestoppt werden. Alle Spuren, die bis dahin erzeugt wurden, liegen dann bereits auf dem Loghost.

2.2 Austausch von Systemprogrammen

Die Manipulation der Logfiles reicht meistens nicht aus, damit der Angreifer auf einem kompromittierten System unentdeckt bleibt. Laufende Prozesse und die Dateien des Angreifers sind z.B. deutliche Hinweise für den Administrator. Der Austausch von Systembefehlen ermöglicht es, diese Spuren zu verheimlichen. Diese Vorgehensweise wurde zuerst auf SunOS 4 bekannt und bald als Linux Rootkit (LRK) portiert. Mittlerweile existieren Implementierungen solcher Rootkits für jedes gängige, UNIX-artige Betriebssystem. Ausgetauscht werden vor allem die Befehle `ps`, `ls` und `netstat`, um bestimmte Prozesse, Dateien und Netzwerkverbindungen aus der Ausgabe dieser Standardbefehle zu löschen.

Im Folgenden soll das Rootkit tornkit in der Version 6.66 als Beispiel dienen. Das Rootkit kam im Jahr 2000 verstärkt zum Einsatz und wird auch gegenwärtig noch verwendet [CERT2000]. Die trojanisierten Programme liegen i.d.R. übersetzt und nicht als Quellcode vor. Bei der Installation wird ein Passwort und ein Port für die Backdoor angegeben, und es ist zur Laufzeit konfigurierbar, welche Dateien, Prozesse und Verbindungen versteckt werden sollen. Etwas vereinfacht läuft die Installation wie folgt ab:

1. Der `syslogd` wird gestoppt und damit das Logging abgeschaltet.
2. Ein bei der Installation anzugebendes Passwort wird in `/etc/ttyhash` gespeichert.
3. Ein trojanisierter SSH Daemon wird unter dem Namen `nscd` gestartet und in `/etc/rc.d/rc.sysinit` eingetragen, damit er bei einem Reboot gestartet wird. Der `nscd` ist normalerweise ein kleines Hilfsprogramm für DNS-Anfragen und fällt dadurch nicht unbedingt auf. Die Schlüssel des Daemons werden in `/usr/info/.t0rn` abgelegt.
4. Das Arbeitsverzeichnis `/usr/src/.puta` wird angelegt und die Konfigurationsdateien dort abgelegt.
5. Es werden die folgenden Systembefehle ausgetauscht: `login`, `ifconfig`, `ps`, `du`, `ls`, `netstat`, `in.fingerd`, `find` und `top`.
6. Die Zeitstempel werden mit denen der Original-Befehle abgeglichen und durch das Anhängen von Nullbytes erhalten die trojanisierten Programme die gleich Größe wie die Originale.

7. Es wird u.a. ein Sniffer in `/usr/src/.puta` installiert und die Dienste `telnet`, `shell` und `finger` für den Internet Daenmon `inetd` in `/etc/inetd.conf` aktiviert.

8. Der `inetd` und der `syslogd` werden neu gestartet.

Eine gängige Methode ist das Kopieren der Original-Befehle in ein verstecktes Verzeichnis. Die trojanisierten Versionen prüfen bei Aufruf, ob spezielle Umstände vorliegen, die ein Filtern der Ausgabe nötig machen und rufen ggf. die Original-Befehle auf. Das tornkit dagegen ersetzt die auzutauschenden Programme komplett - es wird keine Kopie der Originale angelegt. Die Konfigurationsdateien in `/usr/src/.puta` geben an, was versteckt wird. Es wird die Ausgabe von Dateien, Verzeichnissen und Prozessen unterdrückt, die eines der angegebenen Schlüsselwörter im Namen enthalten.

Da es prinzipiell viele Wege gibt sich z.B. die Dateien eines Verzeichnisses anzeigen zu lassen, enthalten neuere Rootkits auch immer mehr trojanisierte Befehle. Neben den vom tornkit trojanisierten Programmen, tauschen manche Rootkits noch folgende Befehle aus: `lsof`, `rpm`, `md5sum`, `tar`, `crontab`, `killall`, `passwd`, `pidof`, `rshd`, `syslogd`, `tcpd` und `tripwire`. Die Liste kann naturgemäß nie vollständig sein und es wird nur eine zufällige Entdeckung verhindert.

Es existieren eine Reihe von Möglichkeiten Rootkits zu erkennen, die wie das tornkit Systembefehle austauschen. Im Folgenden sollen die gängigsten Ansätze kurz vorgestellt werden:

Checksummen: Die trojanisierten Programme haben die gleiche Länge und die gleichen Zeitstempel wie die Originale, aber werden Checksummen mit Algorithmen wir MD5 und SHA-1 gebildet, wird die Manipulation offensichtlich.

Solche Tests können mit Programmen wie `tripwire` oder `aide` [Aide2002] automatisiert durchgeführt werden. Letzteres ist freie Software und erlaubt die freie Wahl des Checksummen generierenden Algorithmus. Um die Checksummen zu überprüfen müssen vorher Referenzwerte erzeugt werden. Der erste Einsatz eines solchen Programms ist nicht sinnvoll, wenn bereits der Verdacht besteht, dass ein Rootkit installiert sein könnte. Ein Vergleich der wichtigsten Programme mit den Versionen auf der CD, von der das Betriebssystem installiert wurde, kann allerdings schon in vielen Fällen helfen, um z.B. den Austausch von `ls` oder `ps` zu erkennen.

Etwas ähnliches kann auch mit Paket-Systemen wie z.B. `rpm` erreicht werden. Diese enthalten i.d.R. schon Mechanismen, um installierte Pakete auf Vollständigkeit und Konsistenz zu prüfen. Wird aber nicht nur mit dem Paket-System, sondern auch manuell Programme installiert, so ist die Ausgabe evtl. irreführend bzw. mit vielen falschen Treffern versehen. In Abb. 3 ist die gekürzte Ausgabe eines Tests der Checksummen per `aide` und `rpm` zu sehen, nachdem das *tornkit* installiert wurde. Manche Hersteller bieten weiterhin an, die Checksummen regulär installlierter Programme über eine Website des Herstellers nachzuschlagen [SUN2002].

Bei allen auf Checksummen basierenden Methoden muss bedacht werden, dass sowohl die Programme zur Erstellung der Checksummen, als auch die Datenbanken mit den Werten manipuliert werden können. Sowohl die nötigen Programme als auch die Checksummen sollten daher auf einem schreibgeschützten, externen Medium gelagert werden.

```
linux:/home/rks # aide -C
AIDE found differences between database and filesystem!!
Start timestamp: 2003-01-03 21:32:11
Summary:
Total number of files=17520,added files=16,removed files=0,changed files=43
[...]
changed:/bin
changed:/bin/ls
changed:/bin/netstat
changed:/bin/ps
changed:/bin/login
[...]
linux:/home/rks # rpm -Vva
[...]
S.5....T   /bin/ls
S.5.....   /bin/netstat
SM5.....   /bin/ps
.M5....T   /bin/login
[...]
```

Abbildung 3: Test der Checksummen mit `aide` und `rpm`.

Rootkit Detektoren: Programme wie das bereits erwähnte `chkrootkit` suchen nach allgemeinen Anzeichen für die Installation eines Rootkits und nach Signaturen bestimmter Rootkits. Existiert z.B. das Verzeichnis `/usr/src/.puta`, so ist dies ein deutlicher Hinweis auf die Installation des tornkit. Bei neuen oder stark modifizierten Rootkits reichen die Signaturen allerdings oft nicht zur Erkennung aus. Weiterhin sollte auch `chkrootkit` und die davon verwendeten Programme auf einem externen, schreibgeschützten Medium gelagert werden. Wie in Abb. 4 zu sehen, wird das tornkit von `chkrootkit` erkannt, sobald die ausgetauschten Systembefehle und bekannte Arbeitsverzeichnisse von Rootkits getestet werden.

```
linux:/home/tools/chkrootkit-0.38 # ./chkrootkit
[...]
Checking ‘login’... INFECTED
Checking ‘ps’... INFECTED
[...]
Searching for t0rn’s default files and dirs... Possible t0rn rootkit installed
[...]
```

Abbildung 4: `chkrootkit` erkennt das tornkit.

Manuelle Suche: Da nur eine begrenzte Anzahl von Systembefehlen ausgetauscht wird, kann z.B. der Inhalt eines verdächtig erscheinenden Verzeichnisses auf andere Weise angezeigt werden. In Abb. 5 wird dies mit `tar`, `rm` und dem in der Shell implementierten Befehl `echo` realisiert. Die mit `strings` anzeigbaren, lesbaren Zeichenketten können auch Hinweise auf Rootkits geben. Die in Abb. 5 gefundene Referenz auf die Datei `/usr/src/.puta/.1file` verrät dabei das Arbeitsverzeichnis des Rootkits.

```
linux:/usr/src # ls -al
total 8
drwxr-xr-x   3 root     root         4096 Jan  3 21:34 .
drwxr-xr-x  25 root     root         4096 Jan  3 21:34 ..
linux:/usr/src # echo .*
. .. .puta
linux:/usr/src # tar -cf - . | tar -tvf - | grep "^d"
drwxr-xr-x root/root         0 2003-01-03 21:34:51 ./
drwxr-xr-x root/root         0 2003-01-03 20:19:24 ./.puta/
linux:/usr/src # rm -i .*
rm: cannot remove `.' or `..'
rm: cannot remove `.' or `..'
rm: `.puta' is a directory
linux:/usr/src/.puta # which ls
/bin/ls
linux:/usr/src/.puta # strings /bin/ls | grep puta
/usr/src/.puta/.1file
```

Abbildung 5: Die manuellen Suche mit Alternativen zum Befehl `ls`.

Einer Suche nach ausgetauschten Systembefehlen kommt unter Solrais entgegen, dass es i.d.R. verschiedene Versionen der grundlegenden Systembefehle gibt. Es kann z.B. zwischen den nicht identischen Programmen `/usr/bin/cp`, `/usr/sbin/static/cp` und `/usr/xpg4/bin/cp` gewählt werden, um eine Datei zu kopieren.

Untersuchen der Systembefehle: Eine genauere Untersuchung von Programmen ist möglich, indem alle zur Laufzeit getätigten Systemaufrufe aufgelistet werden. Unter Solaris ist dies mit dem Befehl `ptrace` möglich. In Abb. 6 werden die vom trojanisierten `ls` getätigten Systemaufrufe unter Linux mit `strace` angezeigt. Hierbei reichen die Aufrufe von `open()` aus, um zu sehen, dass neben den üblichen Dateien die Konfiguration für das trojanisierte `ls` aus `/usr/src/.puta/.1file` gelesen wird.

```
linux:/usr/src/.puta # which ls
/bin/ls
linux:/home/rks # strace /bin/ls / 2> /tmp/tracefile
linux:/home/rks # cat /tmp/tracefile | grep open
open("/etc/ld.so.cache", O_RDONLY)      = 3
open("/lib/libc.so.5", O_RDONLY)        = 3
open("/usr/local/share/locale/locale.alias", O_RDONLY) = -1 ENOENT
open("./locale.alias", O_RDONLY)        = -1 ENOENT
open("/usr/share/locale/de_DE/LC_CTYPE", O_RDONLY) = -1 ENOENT
open("/usr/share/locale/de/LC_CTYPE", O_RDONLY) = -1 ENOENT
open("/usr/src/.puta/.1file", O_RDONLY) = 3
open("/", O_RDONLY)                     = 3
```

Abbildung 6: Die genauere Untersuchung von `ls` per `strace`.

Werden die trojanisierten Systembefehle nicht auf dem Zielsystem übersetzt, muss der Angreifer auf Kompatibilität aller bei der Übersetzung verwendeter Bibliotheken mit dem Zielsystem achten, damit die Befehle lauffähig sind. Statische Bindung scheidet i.d.R. aus, da

Systembefehle aus Effizienzgründen dynamisch gebunden werden und eine Abweichung von dieser Norm schon aufgrund der Größe des resultierenden Programms auffällt. Funktionieren manche Kommandos nicht mehr, oder versagen Skripte, da die Ausgabe von Systembefehlen sich scheinbar etwas geändert hat, so ist dies oft ein Hinweis auf ein nachlässig installiertes Rootkit.

Eine Verallgemeinerung des Austauschs von Systembefehlen ist die Manipulation von dynamischen Bibliotheken [Half1997b] [Cesa2000]. Es wird dabei eine zusätzliche Bibliothek geladen oder eine bestehende verändert, um Systembefehle zu trojanisieren ohne diese direkt zu ändern. Es besteht z.B. die Möglichkeit Anwendungen von Systemaufrufen in der libc zu manipulieren, um so Dateien und Prozesse zu verstecken. Diese Verallgemeinerung ist damit eine Vorstufe zu den Kernel-Rootkits. Die Manipulation von Systembibliotheken wie der `glibc` ist dabei leichter auf verschiedene Betriebssysteme portierbar als ein an einen bestimmten Kernel gebundenes Rootkit. Eine Erkennung ist durch statisches Binden von Programmen möglich. Weicht die Ausgabe des auf dem System befindlichen Befehls `ps` von der Ausgabe einer extern statisch gebundenen Version ab, so ist dies ein deutlicher Hinweis auf die Installation eines Rootkits.

2.3 Kernel-Rootkits

Die Kontrolle von Checksummen ist eine wirkungsvolle Methode, um Rootkits zu entdecken. Im Phrack Magazin erscheint 1997 eine Anleitung zum Schreiben von Kernel-Rootkits für Linux, mit denen der von Checksummen ausgehende Schutz wirkungsvoll umgangen werden kann [Half1997]. Detailierte Anleitungen zur Portierung auf andere Systeme wie Solaris [PlT1999] und FreeBSD [PrT1999] folgen bald.

Ähnlich dem Vorgehen bei der Trojanisierung von Systembibliotheken findet die Manipulation auf einer tieferen Ebene als bei einzelnen Befehlen statt. Das Rootkit wird als Modul in den Kernel geladen und Einträge der Syscall Tabelle werden gegen eigene Routinen getauscht, die dann ggf. die Originale aufrufen. Anstatt z.B. das Programm `ls` zu manipulieren, um bestimmte Dateien aus dessen Ausgabe zu löschen, werden nun die Systemaufrufe der Familien `getdents()` und `stat()` manipuliert, um das Gleiche zu bewerkstelligen. Damit zeigt dann nicht nur `ls` sondern auch alle anderen Programme, die diese Systemaufrufe benutzen (z.B. `tar` oder `rm`) die betreffenden Dateien nicht mehr an. Für die Manipulation des Systems existieren dabei keine Grenzen. Durch existierende Kernel-Rootkits wird beispielsweise implementiert:

- Verstecken von Dateien, Prozessen und Verbindungen
- Verheimlichen des Promiscous Mode des Netzwerkinterfaces
- Umleitung der Ausführung von Programmen (sog. *Executable Redirection*): Nur bei Ausführung wird ein anderes Programm verwendet, als beim Einlesen des Programms z.B. zur Bildung von Checksummen.
- Bereitstellung lokaler Backdoors, z.B. durch Aufruf eines bestimmten Syscalls mit einem bestimmten Parameter

- Bereitstellung von Backdoors über das Netzwerk, z.B. durch direkte Manipulation des TCP/IP Stacks.
- Manipulation beim Auslesen von Speicherseiten, um das Kernel-Rootkits unauffindbar zu machen.
- Zufälliges Einfügen von Fehlern bei Netzwerk- oder Dateisystem-Operationen.

Das Rootkit Adore in der Version 0.42 für Linux soll im Folgenden als Beispiel dienen. Die Installation läuft wie folgt ab:

1. Das Modul `adore.o` wird in den Kernel geladen und tauscht die folgenden Systemaufrufe aus: `write()`, `getdents()`, `kill()`, `fork()`, `clone()`, `close()`, `open()`, `stat()`, `lstat()`, `oldstat()`, `oldlstat()`, `stat64()`, `lstat64()`, `mkdir()`, `getdents64()`.
2. Durch ein zweite Modul wird das Kernelmodul `adore.o` aus der Liste der geladenen Module herausgetrennt und dadurch versteckt. Das Hilfsmodul wird wieder konventionell entladen.
3. Das Programm `ava` wird als Schnittstelle zum Rootkit im Kernel verwendet.

Das Rootkit Adore ermöglicht die Tarnung von Dateien, Prozessen und Verbindungen. Dazu werden 15 Systemaufrufe manipuliert, da von vielen Funktionen unterschiedliche Varianten existieren. Das erneute Laden des Rootkits bei Neustart des Systems und eine Backdoor über das Netzwerk wird nicht von Adore selber realisiert. Die Backdoor z.B. wird oft durch einen manipulierten `sshd` bewerkstelligt, der durch das Kernel-Rootkit verheimlicht wird.

Die Executable Redirection bewirkt das Laden eines anderen Befehls, sobald der Systemaufruf `execve()` zum Ausführen des Programms verwendet wird. Bei einer Überprüfung der Checksummen wird die ursprüngliche Version der Datei mit `open()` geladen. So kann z.B. ein Programm zum Laden des Rootkits nach einem Neustart des Systems vor Programmen getarnt werden, die Checksummen überprüfen. Da aber bei der Installation eines Kernel-Rootkits leicht vom Angreifer eine manipulierte Datei vergessen werden kann, ist der Einsatz von Programmen wie `aide` und `tripwire` genauso empfehlenswert wie die Anwendung von `chkrootkit`, das nach Anzeichen eines installierten Rootkits sucht. Zur Entdeckung von Kernel-Rootkits existieren noch weitere Möglichkeiten:

Suchen nach unbekannten Modulen: Tauchen bei der Abfrage der geladenen Kernel-Module unbekannte Namen auf, so sollte diesen nachgegangen werden. Unter Linux gibt der Befehl `cat /proc/modules` manche Kernel-Rootkits aus, die gegen die Anzeige durch das normale Interface geschützt sind. Das Rootkit Adore trägt sich aus der Liste der geladenen Module aus und taucht daher auch nicht in in `/proc/modules` auf. Unter Solaris ist das Unterdrücken der Anzeige eines Moduls durch das Setzen eines leeren Namens möglich. Es kann daher nicht davon ausgegangen werden, dass geladene Kernel-Rootkits angezeigt werden.

Deaktivieren der Kernelmodule: Es besteht die Möglichkeit die Unterstützung von Kernelmodulen auf einem System zu deaktivieren. Der administrative Aufwand für das so geschützte System steigt dadurch an, dass alle benötigten Treiber fest in den Kern eingebunden sein müssen. Herkömmliche Kernel-Rootkits wie Adore haben dadurch keine Möglichkeit sich zu installieren. Wie in Abschnitt 2.5 erläeutert wird, existieren noch andere Möglichkeiten für den Angreifer, Programmcode im Kernel zur Ausführung zu bringen. Als Grundsicherung ist die Deaktivierung von Kernelmodulen an Systemen, deren Hardware sich nur selten ändert, trotzdem empfehlenswert.

Überprüfung zentraler Strukturen im Kernel: Bei der Manipulation des Kernels werden i.d.R. wenige Strukturen gezielt verändert, um den Programmcode des Angreifers in Systemaufrufen zur Ausführung zu bringen. Es liegt nahe bei der Übersetzung des Kernels zentrale Strukturen wie die Interrupt Descriptor Tabelle und die Syscall Tabelle zu sichern, damit diese zu einem späteren Zeitpunkt kontrolliert werden können. Das Programm `kstat` [Fusy2002] kann eine derartige Sicherung und einen Vergleich mit Referenzwerten durchführen. In Abb. 7 wird durch `kstat` die Manipulation der Syscall Tabelle durch das Rootkit Adore entdeckt und daraufhin rückgängig gemacht.

```
linux:/home/tools/KSTAT24/2.4.16 # ./kstat -s 0
sys_fork                        0xf880c7a0 WARNING! should be at 0xc01058fc
sys_write                       0xf880ca30 WARNING! should be at 0xc013193c
sys_open                        0xf880dbd0 WARNING! should be at 0xc01312c8
sys_close                       0xf880cb80 WARNING! should be at 0xc0131450
sys_oldstat                     0xf880cff0 WARNING! should be at 0xc0137dd4
sys_kill                        0xf880c900 WARNING! should be at 0xc011e1c8
sys_mkdir                       0xf880ccb0 WARNING! should be at 0xc013bfc8
sys_oldlstat                    0xf880d1e0 WARNING! should be at 0xc0137eb4
sys_stat                        0xf880d3d0 WARNING! should be at 0xc0137e44
sys_lstat                       0xf880d5c0 WARNING! should be at 0xc0137f24
sys_clone                       0xf880c850 WARNING! should be at 0xc0105918
sys_getdents                    0xf880c3c0 WARNING! should be at 0xc013ea58
sys_stat64                      0xf880d7b0 WARNING! should be at 0xc0138274
sys_lstat64                     0xf880d9c0 WARNING! should be at 0xc01382e4
sys_ni_syscall                  0xf880c5b0 WARNING! should be at 0xc013ec5c
linux:/home/tools/KSTAT24/2.4.16 # ./kstat -s 1
Restoring system calls addresses...
```

Abbildung 7: Die Manipulation der Syscall Tabelle wird entdeckt.

Laufzeituntersuchng von Systemaufrufen: Werden bestimmte Strukturen auf Manipulation untersucht, besteht immer die Möglichkeit, dass ein Rootkit sich an einer anderen Stelle in den Ausführungsfluss des Kernels bringt. Ein allgemeinerer Ansatz ist daher die Messung der Anzahl der Instruktionen, die bei der Abarbeitung eines bestimmten Systemaufrufs vom Prozesser ausgeführt werden [Rutk2002]. Tatsächlich erzeugen die trojanisierten Systemaufrufe durch die Sonderbehandlung, z.B. zum Verstecken mancher Dateien, einen messbaren Mehraufwand. Es werden auch hier zuerst Referenzwerte bestimmt und bei Verdacht vergleichende Messungen durchgeführt.

Die im Phrack Magazin vorgestellte Implementierung benötigt dazu die Unterstützung eines Kernelmoduls, da die gängigen Debug Mechanismen nicht ohne weiteres im Ker-

nelmode funktionieren. Welche Systemaufrufe mit welchen Parametern untersucht werden, kann leicht im Programmcode angepasst werden. Einzelne Messungen sind nicht reproduzierbar, da der Einfluss der momentanen Arbeitslast und der abzuarbeitenden Interrupts nicht vorhersehbar ist. Die Messwerte sind aber in erster Linie von der Version des Kernels und den verwendeten Treibern abhängig, so dass auch Referenzwerte herangezogen werden können, die von einem anderen System mit dem gleichen Kernel und nicht grundverschiedener Hardware stammen.

In Abb. 8 wird eine Laufzeitmessung an einem Linux System mit Kernel 2.4.16 durchgeführt, auf dem nach der Erstelllung der Referenzwerte das Rootkit Adore installiert wurde. Das Programm führt mehrere Messungen durch (hier 1000 Stück) und gibt die Ergebnisse gemittelt in Form einer Tabelle wieder. Die erste Spalte gibt den untersuchten Systemaufruf an, in der zweiten stehen die aktuellen Messwerte und in der dritten die Referenzwerte. Die Differenz und deren Bewertung sind in den Spalten vier und fünf zu sehen. Zur Abarbeitung des Systemaufrufs `readdir_proc()` wird nach Installation des Rootkits um einen Faktor fünf mehr Befehle ausgeführt. Diese Abweichung ist deutlich vom Schwanken der Messwerte aufgrund variierender Systemlast zu unterscheiden.

Obwohl das Rootkit Adore detektiert wird, können sehr einfache Kernel-Rootkits der Entdeckung entgehen. Werden nur wenige Anweisungen für die Sonderbehandlung gebraucht, z.B. der Test auf eine bestimmte UID als Parameter in einer trojanisierten Version von `setuid()`, so ist die Abweichung der Laufzeit nicht zuverlässig von den normalen Schwankungen der Messwerte zu unterscheiden. Die Laufzeituntersuchung ist daher ein sehr elegantes und allgemeines Mittel zur Detektion, aber gibt dennoch keine Erfolgsgarantie.

```
linux:/home/tools/rktest # ./patchfinder -c referenz_2.4.16
* FIFO scheduling policy has been set.
* each test will take 1000 iteration
* testing... done.
* dropping realtime schedulng policy.

    test name      | current | clear | diff  | status
-----------------------------------------------------
open_file          |    7110|    1442|   5668| ALERT!
stat_file          |    7050|    1255|   5795| ALERT!
read_file          |     608|     608|      0|  ok
open_kmem          |    7124|    1510|   5614| ALERT!
readdir_root       |    6497|    2750|   3747| ALERT!
readdir_proc       |   14422|    2401|  12021| ALERT!
read_proc_net_tcp  |   11750|   11750|      0|  ok
lseek_kmem         |     220|     220|      0|  ok
read_kmem          |     327|     327|      0|  ok
```

Abbildung 8: Kontrolle der Laufzeiten von Systemaufrufen.

Suche nach speziellen Merkmalen: Auch ohne spezielle Programme können Kernel-Rootkits entdeckt werden. Eine Schwachstelle aus Sicht der Rootkits ist oft das Verstecken von Prozessen im Verzeichnis `/proc`. Für jeden Prozess existiert in `/proc` ein

Verzeichnis, dessen Name der ID des Prozesses entspricht. Die Einträge von versteckten Prozessen in `proc` sind i.d.R. mittels `chdir()` zu betreten, aber werden durch `ls /proc` nicht angezeigt. Das Programm `chkproc` aus dem Paket `chkrootkit` vergleicht eine Liste mit betretbaren Unterverzeichnissen in `/proc` mit einer Liste der sichtbaren. Dies kann auch in einem kurzen Shell-Skript implementiert werden.

Wie jedes Rootkit hat auch Adore noch weitere Merkmale, anhand derer es erkannt werden kann. In Abb. 9 ist zu sehen, wie sich die Ausgabe von `lsof` durch die Installation des Rootkits Adore verändert. Der `sshd` mit der ID 153 stellt dabei eine durch das Rootkit getarnte Backdoor dar. Der Befehl `netstat -an` zeigt den offenen Port auch nach der Aktivierung des Rootkits an. Der versteckte Prozess wird daraufhin mit `chkproc` erkannt.

```
[... Vor Installation des Rootkits Adore ...]

linux:/home/tools # lsof -i
COMMAND PID USER   FD   TYPE DEVICE SIZE NODE NAME
portmap 146 root    3u  IPv4    557       UDP *:sunrpc
portmap 146 root    4u  IPv4    558       TCP *:sunrpc (LISTEN)
sshd    153 root    3u  IPv4    627       TCP *:ssh (LISTEN)

[... Nach Installation des Rootkits Adore ...]

linux:/home/tools # lsof -i
COMMAND PID USER   FD   TYPE DEVICE SIZE NODE NAME
portmap 146 root    3u  IPv4    557       UDP *:sunrpc
portmap 146 root    4u  IPv4    558       TCP *:sunrpc (LISTEN)
linux:/home/tools # netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
udp        0      0 0.0.0.0:111             0.0.0.0:*

linux:/home/tools/chkrootkit-0.37 # ./chkproc -v -v
CWD   153: /
EXE   153: /usr/sbin/sshd
linux:/home/tools/chkrootkit-0.37 # ls -al /proc | grep 153
linux:/home/tools/chkrootkit-0.37 # cd /proc/153
linux:/proc/153 # cat cmdline ; echo
/usr/sbin/sshd
```

Abbildung 9: Entdeckung des Rootkit aufgrund spezieller Eigenschaften.

Anti-Rootkit Kernelmodule: Eine spezielle präventive Maßnahme ist die Nutzung von Anti-Rootkit Kernelmodulen wie z.B. StMichael [Lalw2002], das die Installation eines Rootkits erheblich erschweren kann. Die Systemaufrufe zum Laden von Kernelmodulen werden dazu verändert und zusätzlich Funktionen in regelmäßigen Abständen aufgerufen, welche die Integrität des Anti-Rootkit Moduls überprüfen.

Die aufgezählten Techniken und die Methoden zur Detektion sind auf den meisten UNIX-artigen Betriebssystemen anwendbar, da die grundsätzlichen Mechanismen identisch sind. Das Rootkit Adore existiert in Versionen für Linux und FreeBSD.

2.4 Rootkits unter Windows

Erst mit Windows NT wurden Speicherschutz und Sicherheitsmodelle in Microsoft Windows implementiert. Windows Systeme wurden lange kaum als Server oder im Mehrbenutzerbetrieb verwendet. Die im Vergleich zu den meisten UNIX-artigen Systemen spärliche Dokumentation der Systeminterna erschwerten zusätzlich die Erstellung von Rootkits. Mit einem Grundlagen-Artikel 1999 im Phrack Magazin werden Rootkits aber auch unter Windows bekannt [Hogl1999]. Mittlerweile existiert eine Reihe von Rootkits für Windows NT, XP und 2000 und die Anpassung an den Entwicklungsstand der Kernel-Rootkits auf UNIX-artigen Betriebssystemen schreitet zügig voran.

Es existieren erhebliche Unterschiede zwischen UNIX und Microsoft Windows, aber einige grundlegenden Konzepte sind vergleichbar. Unter Windows NT existiert z.B. mit der *Native API* eine Schnittstelle zum Kern des Betriebssystems ähnlich der Systemaufrufe in UNIX [Russ1998]. Im Gegensatz zu UNIX wird diese Schnittstelle nicht direkt, sondern über verschiedene dazwischen liegende APIs (Application Programming Interfaces) genutzt - z.B. der Win32 API. Die Systemaufrufe unter Windows werden *System Services* genannt und über *System Service Decsriptor Tables* referenziert. Hier können Rootkits unter Windows ähnlich wie Kernel-basierte Rootkits unter UNIX zur Manipulation ansetzen. Die von Windows Rootkits implementierten Funktionen sehen daher auch denen unter UNIX sehr ähnlich.

Als Beispiel wird das NT Rootkit in Version 0.40 betrachtet, welches in übersetzter Form und als Quellcode kursiert. Zur Installation wird das Programm `deploy.exe` gestartet. Daraufhin kann das Rootkit mit dem Befehl `net start _root_` aktiviert werden. Die folgenden Funktionen sind implementiert:

- Verstecken von Dateien, Prozessen, Verbindungen und Registry Einträgen, die mit dem Schlüsselwort `_root_` beginnen
- Umleitung der Ausführung von Programmen (Executable Redirection)
- Abhören der Tastatur an der Console
- Bereitstellung einer Backdoor über das Netzwerk
- Gezielter Crash des Systems durch einen Befehl über das Netzwerk.

Das Schlüsselwort, das dem Rootkit zu erkennen gibt, was versteckt werden soll, ist fest im Quellcode gewählt. Die über das Netzwerk zur Verfügung gestellte Backdoor wird über einen im Rootkit integrierten, rudimentären TCP/IP Stack realisiert. Dieser benutzt die ebenfalls fest gewählte IP `10.0.0.166` und ist damit nur im lokalen Netzwerk nutzbar. Die Implementierung ist allerdings nicht sehr stabil, so dass ein Angreifer wahrscheinlich eine Backdoor im Userspace, z.B. mit dem Programm `netcat`, realisieren wird und diese wiederum durch das Rootkit schützen kann. Folgende Möglichkeiten zur Detektion des Rootkits bieten sich an:

Suche nach speziellen Merkmalen: Die mit Hilfe des Rootkits versteckten Dateien werden sichtbar, wenn das entsprechende Verzeichnis im Windows Netzwerk freigegeben und auf einem entfernten System betrachtet wird. Wird der im Rootkit enthaltene TCP/IP Stack benutzt, kann nach der fest kodierten IP im Netzwerk gesucht und z.B. anhand der MAC-Adresse das kompromittierte System gefunden werden. Wird eine Backdoor mit `netcat` realisiert, so kann der geöffnete Port mit den üblichen Werkzeugen wie `netstat` oder `fport` gefunden werden.

Antiviren-Software: Auf Windows Systemen ist die Benutzung von Antiviren-Software sehr verbreitet. Es liegt nahe, diese auch zur Erkennung von Rootkits zu verwenden. Die Schadprogramme unter Windows sind allerdings vermehrt dazu in der Lage, die Viren-Definitionen zu manipulieren oder die Antiviren-Software insgesamt abzuschalten.

Anti-Rootkit Treiber: Analog zu den Anti-Rootkit Kernelmodulen unter UNIX existieren Programme, die als Treiber in den Kern des Betriebssystems geladen werden, um das Laden eines Rootkits zu verhindern. Der *Integrity Protection Driver* geht z.B. so vor. Leider sind diese Mechanismen teilweise noch nicht ausgereift und leicht zu umgehen [Rutk2002b].

2.5 Im Kernel jenseits von Syscall Tabelle und Kernelmodulen

Einige Abwehrmaßnahmen gehen davon aus, dass Kernel-Rootkits unter UNIX die Syscall Tabelle manipulieren. Tatsächlich hat ein Angreifer noch andere Möglichkeiten zur Manipulation des Kernels. Beispielsweise können die Funktionen des Virtual Filesystems unter Linux direkt manipuliert werden [Palm2001]. Eine andere Möglichkeit ist die Manipulation des Interrupt Handlers für Systemaufrufe. Es kann eine veränderte Kopie der Syscall Tabelle benutzt werden während die ursprüngliche Tabelle unverändert an der alten Position verbleibt. Eine weiterführende Möglichkeit ist die Manipulation der Interrupt Descriptor Tabelle [Kad2001].

Rootkits müssen auch nicht als Modul in den Kernel geladen werden. Im Folgenden wird das Rootkit SucKIT in der Version 1.3b unter Linux als Beispiel vorgestellt [SD2001]. Zur Installation des Rootkits ist kein Modul-Support im Kernel notwendig, da schreibender Zugriff auf `/dev/kmem` ausreicht. Die Installation verläuft wie folgt:

1. Ein Schlüsselwort (im Folgenden `.sk12`) zum Verstecken von Dateien und ein Port für die Backdoor über das Netzwerk werden vom installierenden Angreifer erfragt.

2. Das Pogramm `/sbin/init` wird nach `/sbin/init.sk12` kopiert und durch das Programm zur Installation des Rootkits im Kernel ersetzt.

3. Im Verzeichnis `/usr/share/locale/sk` wird das Arbeitsverzeichnis `.sk12` angelegt und dort ein Kopie des Installationsprogramm und die Logdatei des Sniffers abgelegt.

4. Durch Untersuchung des Interrupt Handlers wird der Start der Syscall Tabelle bestimmt.

5. Die Adresse der Kernel Funktion `kmalloc()` wird durch heuristische Suche gefunden.

6. In der Syscall Tabelle wird der Eintrag eines Systemaufrufs durch die Adresse von `kmalloc()` ersetzt. Speicher wird durch Aufruf des ersetzten Systemaufrufs alloziert und die Syscall Tabelle wieder restauriert.
7. Der Programmcode wird in den allozierten Speicher geschrieben und die weitere Manipulation kann im Kernelmode erfolgen.
8. In einer Kopie der Syscall Tabelle werden die Adressen der Folgenden Systemaufrufe verändert: `clone()`, `fork()`, `vfork()`, `getdents()`, `getdents64()`, `kill()`, `open()`, `close()`, `read()`, `write()`, `execve()`, `utime()`, `oldstat()`, `oldlstat()`, `oldfstat()`, `stat()`, `lstat()`, `fstat()`, `stat64()`, `lstat64()`, `fstat64()`, `creat()`, `unlink()` und `readlink()`.
9. Der Interrupt Handler für Systemaufrufe wird derart manipuliert, dass die Kopie der Syscall Tabelle benutzt wird.
10. Die Backdoor wird gestartet.

Die Suche nach der Syscall Tabelle und der Funktion `kmalloc()` ist notwendig, da ein Linux-Kernel, in dem Kernelmodule deaktiviert sind, keine Symbol-Informationen enthält. Der Austausch eines Systemaufrufs in Schritt 6 ist notwendig, da Funktionen des Kernels nur über das Interface der Systemaufrufe benutzt werden können

Bei einem Neustart des Systems startet der Kernel als ersten Prozess das durch das Rootkit ausgetauschte `/sbin/init`. Dieses installiert erneut das Rootkit ab Schritt 4 und startet danach die nicht trojanisierte Version in `/sbin/init.sk12`. Das Rootkit enthält damit bereits einen Mechanismus, um über einen Neustart hinweg persistent zu bleiben sowie eine Backdoor. Die Datei `/sbin/init.sk12` und das Verzeichnis`/usr/share/locale/sk/.sk12` werden durch Befehle wie `ls` nicht angezeigt, da ihre Namen das Schlüsselwort `.sk12` enthalten. Gleichzeitig wird aufgrund der implementierten Executable Redirection, bei der Überprüfung der Checksumme von `/sbin/init`, die Checksumme des Originals in `/sbin/init.sk12` erzeugt. Das Rootkit ist in übersetzter Form relativ portabel und kann in Linux Kernel der Versionen 2.2.x und 2.4.x installiert werden.

Durch die Manipulation einer Kopie der Syscall Tabelle erkennt `kstat` das Rootkit nicht. Die Laufzeituntersuchung von Systemaufrufen scheitert in diesem Fall, da für die Messung ein unbenutzter Systemaufruf belegt wird. Dieser kann aber trotz scheinbar erfolgreicher Installation nicht verwendet werden, da der Interrupt Handler eine Kopie der Syscall Tabelle benutzt. Dies kann ebenso als Detektionsmethode verstanden werden. Könnte eine Messung durchgeführt werden, ist eine deutliche Veränderung der Laufzeiten nach Installation des Rootkits wahrscheinlich. Anti-Rootkit Module wie StMichael verhindern in erster Linie das Laden von unerwünschten Kernelmodulen und können daher die Manipulation nicht verhindern. Folgende Methoden sind aber anwendbar:

Deaktivieren des Schreibzugriffs auf /dev/kmem: Parallel zum Abschalten der Modulunterstützung kann der schreibende Zugriff auf `/dev/kmem` durch einen Patch des Kernels deaktiviert werden. Dies verhindert eine Installation von SucKIT und ähnlicher Rootkits. Es existieren allerdings systemnahe Programme, die schreibenden Zugriff auf `/dev/kmem` im Usermode verwenden. Der X-Server XFree86 kann beispielsweise nicht mehr eingesetzt werden.

Suche nach speziellen Merkmalen: Die in 2.3 vorgestellte Suche nach Verzeichnissen in `/proc`, die zwar zu betreten sind, aber nicht bei der Auflistung des Inhalts auftauchen, ist auch hier erfolgreich. Weiterhin wird das ursprüngliche `/sbin/init` nicht überschrieben sondern durch eine Kopie ersetzt. Programmen wie `aide` und `tripwire` fällt dabei die Änderung der Inode-Nummer auf. Dies ist ein deutlicher Hinweis auf die Installation eines Rootkits.

```
linux:/sbin # ls -il init*
  68269 -rwxr-xr-x    1 root     root       429356 Jan  3 18:32 init
linux:/sbin # md5sum init
225e82f77d74dc8326335381d47f87a9  init
linux:/sbin # mv init renamedinit
linux:/sbin # ls -il renamedinit
  69665 -rwxr-xr-x    1 root     root        28984 Jan  3 18:32 renamedinit
linux:/sbin # md5sum renamedinit
89e5bcb0ef33614bb8c3a45c59d5e471  renamedinit
```

Abbildung 10: Das Rootkit wird durch Umbenennung entdeckt.

Es werden insgesamt 24 Systemaufrufe manipuliert, um die Funktionen des Rootkits zu realisieren. Der Aufruf `rename()` ist allerdings nicht dabei. Wird die Checksumme von `/sbin/init` überprüft so erscheint diese korrekt aufgrund der Execution Redirection. Wie in Abb. 10 zu sehen kann durch einfaches Umbennen mit dem Befehl `mv`, der den Systemaufruf `rename()` benutzt, auf das trojanisierte Programm `/sbin/init` zugegriffen werden. Es ändern sich, scheinbar durch das Umbennenen, Inode-Nummer, Größe und Checksumme der Datei.

2.6 Aktuelle Entwicklungen

Statt simpler Manipulation der Syscall Tabelle geht der Trend dahin, Programmcode in tiefer liegenden Funktionen im Kernel einzubetten [Palm2001b]. Ein einfacher Vergleich nur dieser Tabelle mit einer vorher gespeicherten Referenz verliert damit an Sinn, aber bleibt ein einfacher Test für alte Kernel-Rootkits. Infektion des Kernels und von Bibliotheken [Anon2002] zur Laufzeit, zusammen mit der Manipulation des auf der Festplatte liegenden Abbild des Kernel [Jbtz2002] stellen weitere Optionen der Angreifer dar. Die Entdeckung der Teile des Rootkits, die sich im Userspace und im Dateisystem befinden, wird dadurch erheblich erschwert. Es werden dabei vermehrt Techniken aus dem Bereich der Computerviren in Rootkits übertragen.

Bei der Codeflow Analyse werden die möglichen Verzweigungen des Programmflusses mit graphentheoretischen Algorithmen untersucht. Eine solche Analyse kann benutzt werden, um automatisch zur Laufzeit Punkte im Programmcode eines Systemaufrufs zu finden, an denen effektiv trojanisierter Code eingefügt werden kann [Teso2003]. Da der Programmcode dadurch nicht am Anfang des Systemaufrufs angesprungen werden muss, ist eine Entdeckung der Manipulation deutlich erschwert. Die Codeflow Analyse wird weiterhin benutzt, um Anti-Rootkit Module wie StMichael im Kernel ausfindig zu machen und zu deaktivieren.

3 Abwehrtechniken

Die in den vorhergehenden Abschnitten erläuterten Techniken zur Abwehr und Entdeckung von Rootkits sollen im Folgenden zusammengefasst und bewertet werden:

Kontrolle der Logfiles: Werden die Logfiles vom Angreifer manipuliert, sind trotzdem oft noch Spuren zu finden. Allgemeine Unregelmäßigkeiten im Betrieb, wie spontan versagende Skripte oder ungewöhnliche Auslastung von System und Netzwerk, sind ebenso wichtige Hinweise auf Missbrauch. Voraussetzung für das Erkennen dieser Spuren ist eine genaue Kenntnis der regulär anfallenden Logdaten.

Sehr effektiv ist die Verwendung eines dedizierten Loghosts, an den die Logdaten aller anderen Systeme geschickt werden. Ein Angreifer kann nach erfolgreicher Übernahme eines Systems das Logging komplett deaktivieren. Die Logdaten, die im Laufe des Angriffs an den Loghost übertragen wurden, sind aber bereits seinem Zugriff entzogen. Organisatorisch ergibt sich ein zusätzlicher Vorteil, da auf dem Loghost die Daten verschiedener Systeme in Korrelation gebracht werden können. Umfangreiche Referenzen zur Analyse von Logfiles und zum Aufbau einer zentralen Log-Infrastruktur sind bei Counterpane zu finden [Bird2002].

Integritätstests durch Checksummen: Die Kontrolle der Checksummen durch Programme wie `aide` [Aide2002] oder `tripwire` ist aufwendig, da nach jeder Änderung der Systemkonfiguration die Datenbank mit den Checksummen auf den neuen Stand gebracht werden muss. Dieser Aufwand ist aber lohnend, da jede Änderung sehr genau verfolgt werden kann und damit nicht nur eine Detektion sondern auch eine Schadensabschätzung möglich ist. Kernel-Rootkits können zur Laufzeit einer Entdeckung durch die Kontrolle von Checksummen entgehen. Findet die Untersuchung aber in einer kontrollierten Umgebung statt, sind die Checksummen wiederum sehr hilfreiches bei der Spurensuche (siehe auch *Offline-Analyse*).

Programme zur Detektion von Rootkits: Es existiert eine Reihe von Programmen zur Detektion von Rootkits, die im Falle eines Verdachts oder regelmäßig zur Prävention angewandt werden können. Diese Programme benutzen zur Detektion i.d.R. Signaturen in Form von bestimmten Merkmalen einzelner Rootkits. Nur wenige Programme suchen nach allgemeinen Unregelmäßigkeiten oder Inkonsistenzen im Verhalten des Systems. Beispiele für Programme, die zur Entdeckung von Rootkits eingesetzt werden können, sind:

Chkrootkit: Das Programm sucht mit Hilfe von Signaturen nach bestimmten Rootkits und allgemeinen Anzeichen zum Vorhandensein eines Rootkits. Ein allgemeiner Hinweis auf die Existenz eines Rootkits sind z.B. versteckte Prozesse in `/proc` [MSJ2002].

KSTAT: Zentrale Resourcen des Kernels wie die Syscall Tabelle werden mit vorher gespeicherten Referenzwerten verglichen. Weiterhin ist eine Suche nach nicht registrierten Kernelmodulen durchführbar. Leider ist die aktuelle Version nur mit Anpassungen im Quellcode auf aktuellen Linux Kerneln lauffähig [Fusy2002].

Patchfinder: Die Messung der Laufzeiten von Systemaufrufen ist eine sehr robuste Methode zur Detektion von Rootkits. Die im Phrack Magazin vorgestellte Implementierung ist noch experimentell, kann aber leicht ausgebaut werden [Rutk2002].

Präventive Patches und Kernelmodule: Auf Systemen, die unter Linux ohne X-Server betrieben werden und deren Konfiguration selten geändert wird, ist das Deaktivieren von Kernelmodulen und das Abschalten des schreibenden Zugriffs auf `/dev/kmem` empfehlenswert.

Neben `/dev/kmem` existiert noch `/dev/mem`, das sich dadurch unterscheidet, dass direkt auf die physikalische Anordnung des Speichers zugegriffen wird und noch keine Umsetzung auf den virtuellen Adressraum vorgenommen wird. Eine Anwendung der in Abschnitt 2.5 vorgestellten Techniken ist theoretisch über diesen Umweg möglich, wenn auch deutlich komplizierter. Ggf. reicht auch schon die Benutzung der Funktion `mmap()` aus, um Speicher des Kernels beschreiben zu können [Pela2002]. Die Deaktivierung des Schreibzugriffs stellt also kein unüberwindbares Hindernis dar, aber erschwert die Installation eines Kernel-Rootkits erheblich.

Alle oben beschriebenen Detektionsmethoden können durch absehbare Weiterentwicklungen der Rootkits umgangen werden. Ein Administrator hat allerdings den Vorteil, eine Analyse auch offline unter kontrollierten Bedingungen vornehmen zu können und das lokale Netzwerk zu beherrschen:

Offline-Analyse: Besteht der Verdacht auf Installation eines Rootkits, kann das System vom Netz getrennt werden, um die Festplatte auf einem anderen System unter kontrollierten Bedingungen zu untersuchen. Mit Hilfe von vorher erzeugten Checksummen ist es möglich, die Manipulation genau zu rekonstruieren. Der Vorteil bei der Benutzung eines Analyse-System liegt darin, dass dem System, den Befehlen und den Prüfprogrammen vertraut werden.

Wird das Rootkit nur im Hauptspeicher des Systems installiert, sind bei einer derartigen Offline-Analyse evtl. keine Spuren mehr zu finden. Der Wurm CodeRed und seine Varianten infizieren seit 2001 mit dem Internet verbundene Windows Systeme ohne den eigenen Code auf einen Datenträger zu schreiben. Die Angreifer sind aber i.Allg. daran interessiert, ein System auch nach einem Neustart kontrollieren zu können und installieren dazu Mechanismen, die eine Aktivierung des Rootkits beim Start des Systems erwirken. Diese Mechanismen können durch eine Offline-Analyse gefunden werden.

Der Einsatz von forensichen Werkzeugen wie des Coroner’s Toolkit [Vene2002] eignet sich, um auch den momentanen Zustand des Systems durch ein Speicherabbild in die Analyse einzubeziehen.

Kontrolle des lokalen Netzwerks: Es ist schwer abzuschätzen, was ein Angreifer mit einem kompromittierten System zu tun gedenkt. Oft wird das System benutzt, um illegal kopierte Software oder Daten in Umlauf zu bringen, oder um weitere Angriffe zu starten. Beides verursacht speziellen Verkehr im Netzwerk, der schon bei der Erstellung von nach Protokollen sortierten Statistiken auffallen wird. Die Kontrolle des Netzwerkverkehrs ist damit ein guter Indikator, um kompromittierte Systeme zu entdecken. Der Einsatz eines Intrusion Detection Systems im Netzwerk ist der nächste logische Schritt,

um schon den Angriff auf ein System zu entdecken und ggf. zu unterbinden. Das regelmäßige Scannen des eigenen Netzwerks nach installierten Backdoors ist eine weitere wichtige Informationsquelle.

Obwohl es viele verschiedene Werkzeuge und Programme zur Entdeckung von Rootkits gibt, ist die wichtigste präventive Maßnahme, sich *bevor* das Problem auftritt überlegt zu haben, was im Angriffsfall getan werden muss. Ein solcher Notfallplan sollte eine gute Backupstrategie umfassen und alle wichtigen Schritte und Maßnahmen aufführen. Auf diese Weise kann schnell auf einen Sicherheitsvorfall reagiert werden. Nur wenn ein aktuelles Backup existiert, kann im Notfall das betroffene System schnell ersetzt werden.

4 Diskussion und Ausblick

Die Techniken der Rootkits haben sich seit dem Ende der 80'er Jahre von der Filterung von Logfiles bis zur detailierten Manipulation des Kerns des Betriebssystems entwickelt. Im Gegenzug haben die Methoden zur Entdeckung der Angreifer sich darauf einstellen müssen, dem zu untersuchenden System immer weniger vertrauen zu können. In der Praxis tauchen sowohl aktuelle als auch alte Rootkits auf. Es ist unbedingt zu empfehlen, die eigenen Systeme gegen alle Varianten zu schützen. Die beschriebenen Abwehrmaßnahmen und Methoden zur Detektion stellen dabei einen guten Grundschutz dar, aber können keine Sicherheit garantieren.

Auffallend ist, dass die Entwicklung der Techniken der Angreifer und der Abwehrmaßnahmen hauptsächlich auf Betriebssystemen wie Linux und den freien BSD Varianten vorangetrieben wird. Der Transfer auf andere Systeme ohne freien Quellcode findet aber zunehmend schneller statt. Die Entwicklungsspirale von Angriffstechniken und lokalen Abwehrmaßnahmen muss allerdings nicht hingenommen werden. Offline-Analyse, die genaue Kontrolle des lokalen Netzwerkes und der Einsatz von IDS-Systemen bieten Administratoren zusätzliche Möglichkeiten, seine Systeme vor Angreifern zu schützen und Rootkits zu entdecken.

Kernel-Rootkits stellen auch in Zukunft die größte Bedrohung für die Integrität von mit dem Internet verbundenen Systemen dar. Es ist absehbar, dass die Programme zur Entdeckung von Rootkits eine bessere Prüfung der Integrität des Kernels vornehmen werden. Anstatt der momentanen Beschränkung auf einen Vergleich von zentralen Ressourcen mit gespeicherten Referenzwerten muss die Konsistenz des gesamten Ausführungspfads von Systemaufrufen abgesichert werden. Die beschriebene Laufzeitmessung ist ein Schritt in diese Richtung. Die Auswertung von Ausführungspfaden mit graphentheoretischen Mitteln bietet die Möglichkeit, abstrakte Beschreibungen von Systemaufrufen zu erstellen. Diese können dann zur Überprüfung der Integrität genutzt werden.

Literaturverzeichnis

[Aide2002] R. Lehti; P. Virolainen, *Homepage Aide*, http://www.cs.tut.fi/~rammer/aide.html, 2002.

[Anon2002] Anonymous, *Runtime Process Infection*, Phrack Magazin, Issue 59, Vol. 11, File 8. http://www.phrack.org/phrack/59/p59-0x08, 2002.

[BC2002] D. Bovet; M. Cesati, *Understanding the Linux Kernel, 2nd Edition*. O‘Reilly. ISBN-0596002130, 2002.

[BTA1989] “Black Tie Affair”, *Hiding Out Under Unix*. Phrack Magazin, Issue 25, Vol. 3, File 6. http://www.phrack.org/phrack/25/P25-06, 1989.

[Bach1986] M. Bach, *The Design of the UNIX Operating System*. Prentice-Hall. ISBN-0132017997, 1986.

[Bird2002] T. Bird, *Log Analysis Resources*. Homepage Counterpane Internet Security. http://www.counterpane.com/log-analysis.html, 2002.

[CERT2000] CERT Coordination Center, *CERT/CC Incident Note IN-2000-10: Widespread Exploitation of rpc.statd and wu-ftpd Vulnerabilities*, http://www.cert.org/incident_notes/IN-2000-10.html, 2000.

[Cesa2000] S. Cesare, *Shared library call redirection via ELF PLT infection*. Phrack Magazin, Issue 56, Vol. 10, File 7. http://www.phrack.org/phrack/56/p56-0x07, 2000.

[Ditt2002] D. Dittrich, *“Root Kits” and hiding files/directories/processes after a break-in*. http://staff.washington.edu/dittrich/misc/faqs/rootkits.faq, 2002.

[Fusy2002] “Fusys”, *Homepage kstat*, http://www.s0ftpj.org/en/tools.html, 2002.

[Half1997] “Halflife”, *Abuse of the Linux Kernel for Fun and Profit*. Phrack Magazin, Issue 50, Vol. 7, File 5. http://www.phrack.org/phrack/50/P50-05, 1997.

[Half1997b] “Halflife”, *Shared Library Redirection Techniques*. Phrack Magazin, Issue 51, Vol. 7, File 8. http://www.phrack.org/phrack/51/P51-08, 1997.

[Hogl1999] G. Hoglund, *A real NT Rootkit, patching the NT Kenerl*. Phrack Magazin, Issue 55, Vol. 9, File 5. http://www.phrack.org/phrack/55/P55-06, 1999.

[Jbtz2002] “jbtzhm”, *Static Kernel Patching*. Phrack Magazin, Issue 60, Vol. 11, File 8. http://www.phrack.org/phrack/60/p60-0x08, 2002.

[Kad2001] “kad”, *Handling Interrupt Descriptor Table for fun and profit*. Phrack Magazin, Issue 59, Vol. 10, File 4. http://www.phrack.org/phrack/59/p59-0x04.txt, 2001.

[Lalw2002] T. Lalwess, *Homepage des Anti-Rootkit Kernel Modul StMichael*, http://sourceforge.net/projects/stjude/, 2002.

[MSJ2002] N. Murila; K. Steding-Jessen, *Homepage chkrootkit*. http://www.chkrootkit.org, 2002.

[Palm2001] "Palmers", *Advances in Kernel Hacking*. Phrack Magazin, Issue 58, Vol. 11, File 6. http://www.phrack.org/phrack/58/p58-0x06, 2001.

[Palm2001b] "Palmers", *Advances in Kernel Hacking II*. Phrack Magazin, Issue 59, Vol. 11, File 5. http://www.phrack.org/phrack/59/p59-0x05, 2001.

[Pela2002] G. Pelat, Artikel auf Mailingliste BugTraq: *Grsecurity problem - modifying 'read-only' kernel*, http://securityfocus.com/archive/1/273002, 2002.

[PlT1999] "Plasmoid"; "THC", *Solaris Loadable Kernel Modules: "Attacking Solaris with loadable kernel modules"*. http://packetstormsecurity.nl/groups/thc/slkm-1.0.html, 1999.

[PrT1999] "Pragmatic"; "THC", *Attacking FreeBSD with Kernel Modules*, http://www.thehackerschoice.com/papers/bsdkern.html, 1999.

[Russ1998] M. Russinovich, *Inside the Native API*. Artikel auf Sysinternals Homepage. http://www.sysinternals.com/ntw2k/info/ntdll.shtml, 1998.

[Rutk2002] J. K. Rutkowski, *Execution Path Analysis: finding kernel based rootkits*. Phrack Magazin, Issue 59, Vol. 11, File 10. http://www.phrack.org/phrack/59/p59-0x0b.txt, 2002.

[Rutk2002b] J. K. Rutkowski, Artikel auf Mailingliste BugTraq: *Bypassing Pedestal Software Integrity Protection Driver*. http://online.securityfocus.com/archive/1/301866, 2002.

[SD2001] "SD"; "Devik", *Linux on-the-fly kernel patching without LKM*. Phrack Magazin, Issue 58, Vol. 10, File 7. http://www.phrack.org/phrack/58/p58-0x07, 2001.

[SR2000] D. Solomon; M. Russinovich, *Inside Microsoft Windows 2000, Third Edition*. Microsoft Press. ISBN-0735610215, 2000.

[SUN2002] SUN Microsystems, *Solaris Fingerprint Database*, http://http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl, 2002.

[Teso2003] "team teso", *Codeflow Analyse*. Vortrag auf dem 19. Chaos Communications Congress, Berlin. http://www.team-teso.net/articles/19c3-speech/, 2003.

[Vene2002] W. Venema, *Homepage: The Coroner's Toolkit*, http://www.porcupine.org/forensics/, 2002.

Alle angegeben Referenzen auf Webpages wurden im Januar 2003 überprüft. Die Jahreszahlen stehen dabei für das Erstellungsdatum oder die letzte angegebene Änderung.