UNIX und Linux basierte Kernel Rootkits

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

Abstract: Ein Rootkit ermöglicht einem Angreifer auf einem kompromittierten System unentdeckt zu bleiben, um dieses ohne das Wissen des Administrators nach Belieben zu seinen Zwecken missbrauchen zu können. In diesem Beitrag wird ein Überblick über die aktuell von Angreifern auf UNIX und Linux Systemen eingesetzten Techniken gegeben. Für die gängigsten Rootkits werden Abwehrmaßnahmen vorgestellt und deren Wirksamkeit diskutiert.

Keywords: Malicious Agents, Malware, Kernel Rootkits, Incident Response.

1 Motivation

Ein Angriff auf ein mit dem Internet verbundenes System sieht typischerweise wie folgt aus: Mit Hilfe von Portscans wird versucht, Dienste mit bekannten Schwachstellen zu finden. Kann eine Sicherheitslücke erfolgreich ausgenutzt und können Befehle auf dem System ausgeführt werden, gilt das System als *kompromittiert*. Über weitere, lokale Schwachstellen kann der Angreifer dann die Zugriffsrechte des System Administrators erhalten.

Wird der Einbruch entdeckt, versucht der Administrator die Kontrolle über das System zurückzugewinnen. Dies kann durch Beseitigung der Sicherheitslücke, eine Neuinstallation oder die Trennung des Systems vom Netz geschehen. Will der Angreifer länger auf das System zugreifen, muss er daher den Einbruch und seine Anwesenheit verbergen.

Zu diesem Zweck setzen Angreifer auf kompromittierten Systemen oft Sammlungen von speziellen Werkzeugen und Programmen ein - so genannte *Rootkits*. Dieser Beitrag behandelt die Techniken der Angreifer, sich den Zugriff auf ein bereits kompromittiertes System mit Hilfe von Rootkits zu bewahren. Weiterhin werden Strategien zur Aufdeckung dieser Manipulation vorgestellt und deren Wirksamkeit an Beispielen getestet.

1.1 Die Entwicklung der Rootkits

Die Entwicklung von Rootkits begann Ende der 80'er Jahre mit Programmen zur Manipulation von Log Files [Bl89]. Für diese Operationen sind die Rechte des System Adminis-

trators nötig, der auf Unix-Systemen üblicherweise *root* heißt. Diese Manipulation führte dazu, dass der Ausgabe von System Befehlen wie `who` oder `last` nicht mehr vertraut werden konnte, da diese zur Anzeige der angemeldeten Nutzer auf Log Files angewiesen sind. Bestand der Verdacht einer Kompromittierung, konnten Angreifer jedoch leicht durch alternative System Befehle entdeckt werden.

Bald entstand die Praxis, Befehle des Systems auszutauschen, um die Aktivitäten der Angreifer zu verbergen. Es wurden z.B. `ls` oder `netstat` ersetzt, um bestimmte Dateien oder Verbindungen in der Ausgabe des jeweiligen Programms nicht auftauchen zu lassen. Diese ersten explizit als Rootkit bezeichneten Programm Sammlungen wurden zuerst für SUN OS bekannt, aber bald auf Linux und andere UNIX Varianten portiert. Zur Entdeckung eines solchen Angriffs kann die Integrität der System Befehle durch Checksummen kontrolliert werden und es können gegebenenfalls statisch gebundene Programme von externen Datenträgern zur Untersuchung ins System gebracht werden. Da diese nicht vom Angreifer ausgetauscht wurden, sind sie in der Lage, die Manipulation aufzudecken.

Mitte der 90'er Jahre kamen *Kernel Rootkits* auf Linux Systemen auf, die durch zur Laufzeit ladbare Module den Kern des Betriebssystems manipulieren [Ha97]. Diese neuen Rootkits wurden bald auf andere UNIX-artige Systeme wie Solaris [Pl99] und verschiedene BSD Varianten [Pr99] portiert. Durch die Manipulation des Kernels wird der Austausch einzelner Programme überflüssig, da alle Programme auf Funktionen des Kernels angewiesen sind. Kernel Rootkits haben heutzutage herkömmliche Rootkits weitgehend ersetzt.

2 Technische Grundlagen

Im Folgenden wird der Unterschied zwischen User und Kernel Mode, sowie der Ablauf von Systemaufrufen kurz erläutert, um eine Grundlage zum Verständnis der in Rootkits eingesetzten Techniken zu bilden. Eine genauere Beschreibung dieser Mechanismen ist in der Literatur zu finden [Ba86].

2.1 User und Kernel Mode

UNIX-artige Betriebssysteme unterscheiden zwischen Prozessen im Kernel (*Kernel Mode*) und Prozessen, die von Nutzern (einschließlich `root`) gestartet wurden (*User Mode*). Auf Speicherbereiche des Kernels kann von einem Prozess im User Mode nur über die Device Files `/dev/kmem` und `/dev/mem` zugegriffen werden. Dieser Zugang wird allerdings selten genutzt, um Daten zwischen User und Kernel Mode zu transportieren, da seine Anwendung schwierig ist und z.B. Symbol Tabellen zum Auffinden von Funktionen und Datenstrukturen im Kernel nicht unbedingt vorhanden sind. Der X-Server XFree86 benutzt z.B. je nach Betriebsmodus `/dev/kmem`, um Daten in den Kernel zu transportieren.

Der Aufruf von Funktionen innerhalb des Kernels ist nur über die *Systemaufrufe* möglich, die ein wohldefiniertes, statisches Interface zur Verfügung stellen. Auch der User `root` kann nicht beliebigen Programmcode innerhalb des Kernels ausführen. Allerdings können alle modernen UNIX-artigen Systeme Module zur Laufzeit in den Kernel laden. Dies wird z.B. benutzt, um Treiber für Multimedia oder Hotplug-Hardware nachzurüsten.

2.2 Ablauf eines Systemaufrufs

Der Befehl `ls /tmp` listet alle Dateien im Verzeichnis `/tmp` auf. Hierbei wird der Systemaufruf `open()` verwendet, um das Verzeichnis zum Lesen zu öffnen (siehe Abbildung 1). Der Aufruf beinhaltet folgende Schritte:

- Das Programm `ls` ruft `open()` auf. Dazu werden die Parameter in den entsprechenden Registern abgelegt und der für Systemaufrufe vorgesehene Interrupt ausgelöst, damit das System in den Kernel Mode wechselt.
- Im Kernel Mode wird die *Interrupt Descriptor Tabelle* (IDT) referenziert, um den entsprechenden *Interrupt Handler* zu finden.
- Der Interrupt Handler bestimmt anhand der *Syscall Tabelle* die aufzurufende Funktion `sys_open()` des Kernels und ruft diese mit den übergebenen Parametern auf.

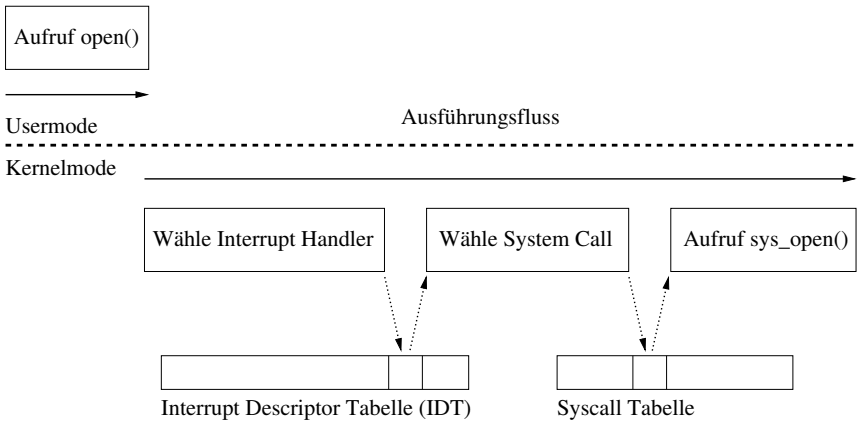

Abbildung 1: Der Systemaufruf `open()` wird verwendet und der Ausführungsfluss wechselt dabei in den Kernel Mode. Die Interrupt Descriptor Tabelle und die Syscall Tabelle werden referenziert, um die auszuführende Funktion zu bestimmen.

Das Programm `ls` erhält ein Filehandle als Ergebnis und tätigt weitere Systemaufrufe wie `getdents()`, um den Inhalt von `/tmp` auszulesen und auszugeben. Welche Funktionen des Kernels bei den jeweiligen Systemaufrufen ausgeführt werden, hängt stark von zentralen Ressourcen wie der IDT und der Syscall Tabelle ab.

3 Kernel Rootkits

Kernel Rootkits können jegliche Information manipulieren, die vom betroffenen System verarbeitet wird. Dies erlaubt theoretisch eine perfekte Simulation des nicht kompromittierten Systems. Es existieren z.B. Überlegungen, Emulatoren wie User Mode Linux zu benutzen, um in der simulierten Umgebung ein Replikat des ursprünglichen Systems zu betreiben [Sk03]. In der Praxis ist das für einen Angreifer aber kaum möglich.

Die meisten Kernel Rootkits geben dem Angreifer die Möglichkeit, Prozesse, Dateien, Verzeichnisse und Netzwerkverbindungen zu verstecken. Dies wird oft durch Manipulation von Systemaufrufen realisiert. Damit das Rootkit aktiv werden kann müssen zwei grundsätzliche Erfordernisse umgesetzt werden: Der Transfer des Rootkits in den Kernel und die Umleitung des Programm Flusses innerhalb des Kernels. Die Umsetzung dieser Erfordernisse können als Kriterium zur Klassifikation von Rootkits verwendet werden. Nach diesem Schema werden im Folgenden drei verbreitete Kernel Rootkits vorgestellt.

3.1 Adore

Das Rootkit Adore war eines der ersten Kernel Rootkits, aber wird noch immer auf kompromittierten Systemen gefunden. Es existieren Versionen für Linux und verschiedene BSD Systeme. Im Folgenden wird die Version 0.42 unter Linux betrachtet. Adore ermöglicht die Tarnung von Dateien, Prozessen und Verbindungen, wobei ein User Mode Programm namens `ava` als Schnittstelle zum Rootkit im Kernel dient. Ein Mechanismus zum Laden des Rootkits bei Neustart des Systems und eine Backdoor über das Netzwerk sind nicht direkt ein Teil von Adore. Eine solche Backdoor wird oft durch einen manipulierten SSH-Daemon realisiert, der mit Hilfe des Kernel Rootkit versteckt wird.

3.1.1 Transfer in den Kernel Mode

Das Rootkit Adore wird als Modul in den Kernel geladen, wobei das reguläre Interface des Systems verwendet wird. Die Methode ist gut dokumentiert und wird von den meisten Kernel Rootkits verwendet. Ein zweites Modul wird geladen und dann wieder entladen, um die Strukturen zur Verwaltung der installierten Module im Kernel zu manipulieren. Das Rootkit kann dadurch mit den üblichen System Befehlen weder angezeigt noch entladen werden.

3.1.2 Umleitung des Programm Flusses

Das Rootkit manipuliert die Syscall Tabelle, um dort die Einträge von 15 Systemaufrufen auf Funktionen des Rootkits umzuleiten. Dies ist die Vorgehensweise der meisten Kernel Rootkits und relativ leicht zu implementieren. Die Manipulation ist schematisch in Abbildung 2 dargestellt.

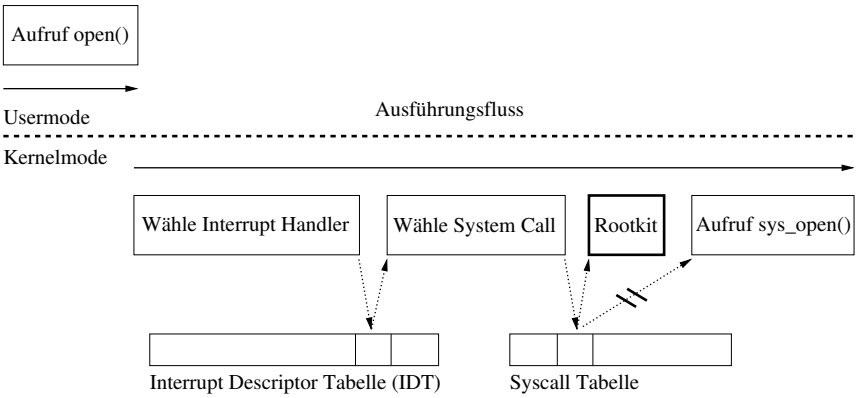

Abbildung 2: Das Rootkit Adore manipuliert die Syscall Tabelle, um Einträge von 15 Systemaufrufen auf Funktionen des Rootkits umzuleiten.

3.2 SucKIT

Das Rootkit SucKIT wurde 2001 im Phrack Magazin ausführlich beschrieben [SD01] und ist gegenwärtig oft auf kompromittierten Systemen zu finden. Im Folgenden wird Version 1.3b unter Linux betrachtet. Portierungen auf andere UNIX-artige Systeme sind möglich, aber noch nicht beobachtet worden.

Das Rootkit umfasst einen Sniffer zur Protokollierung von ungesichert im Netzwerk übertragenden Passwörtern und eine Backdoor über das Netzwerk. Die Backdoor wird erst aktiviert nachdem ein bestimmtes Paket das kompromittierte System erreicht hat. Dies soll die Backdoor vor Entdeckung durch Portscans schützen.

Weiterhin ist ein Mechanismus zum Laden des Rootkits bei Neustart des Systems enthalten. Dazu wird das Programm `/sbin/init` bei der Installation des Rootkits durch die Lade Routine des Rootkits ersetzt. Beim Start des Systems wird das manipulierte `/sbin/init` ausgeführt, welches das Rootkit in den Kernel lädt und danach das umbenannte Original `/sbin/init_XY`[1] ausführt. Sobald das Rootkit aktiv ist, wird das umbenannte Original durch das Rootkit versteckt. Außerdem werden Zugriffe auf `/sbin/init` durch die manipulierten Systemaufrufe auf das umbenannte Original umgeleitet. Wird eine Checksumme von `/sbin/init` erstellt nachdem das Rootkit aktiviert wurde, so scheint sich die Datei nicht verändert zu haben. Das Laden des Rootkits ist schematisch in Abbildung 3 dargestellt.

Das Rootkit ist in übersetzter Form sehr portabel und kann in beliebigen Linux Kerneln der Versionen 2.2.x und 2.4.x installiert werden. Läuft das Linux System in einer emulierten Umgebung wie `vmware`, versagt die Lade Routine.

[1] Anstatt des 'XY' kann eine beliebige Zeichenkette bei der Konfiguration des Rootkits gewählt werden.

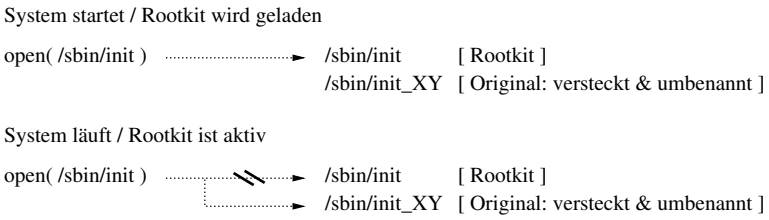

Abbildung 3: Das Rootkit SucKIT ersetzt `/sbin/init` durch die eigene Lade Routine. Um dies zu verbergen werden die Zugriffe auf `/sbin/init` umgeleitet auf das versteckte, umbenannte Original sobald das Rootkit aktiv ist.

3.2.1 Transfer in den Kernel Mode

Das Rootkit SucKIT verwendet nicht das Modul-Interface des Kernels, sondern benötigt zur Installation lediglich schreibenden Zugriff auf `/dev/kmem`. Der Programmcode des Rootkits wird durch ein mehrstufiges Verfahren in den Kernel transferiert:

- Im Speicher des Kernels wird nach den Adressen der Syscall Tabelle und der Funktion `kmalloc()` gesucht.
- In der Syscall Tabelle wird der Eintrag eines unbenutzten Systemaufrufs durch die Adresse von `kmalloc()` ersetzt.
- Die Funktion `kmalloc()` wird als Systemaufruf gestartet, um Speicher im Kernel zu reservieren.
- Die Programmcode des Rootkits wird über `/dev/kmem` in den gerade reservierten Speicher übertragen.
- Der unbenutzte Systemaufruf wird erneut manipuliert, um den übertragende Programmcode im Kernel Mode aufzurufen.

An diesem Punkt kann die weitere Manipulation direkt im Kernel stattfinden. Die anfängliche Suche nach der Syscall Tabelle und der Funktion `kmalloc()` ist notwendig, da deren Adressen unbekannt sind, wenn der Kernel keine Symbol Tabellen vorhält. Auf den umständlichen Austausch des Systemaufrufs kann nicht verzichtet werden, da Funktionen des Kernels nur über das Interface der Systemaufrufe verwendet werden können

3.2.2 Umleitung des Programm Flusses

Das Rootkit SucKIT manipuliert ebenfalls die Syscall Tabelle, allerdings wird zuerst eine Kopie angelegt, um in dieser 24 Systemaufrufe auf Funktionen des Rootkits umzuleiten. Bei der Durchführung eines Systemaufrufs referenziert der Interrupt Handler die Syscall Tabelle, um die aufzurufende Funktion zu bestimmen (vgl. 2.2). Das Rootkit verändert den Interrupt Handler für Systemaufrufe derart, dass die manipulierte Kopie der Syscall

Tabelle anstatt des Originals verwendet wird. Vergleicht ein Programm die Syscall Tabelle mit einer vorher gespeicherten Version, so scheint sich diese nicht verändert zu haben. Da der Interrupt Handler nun die Kopie verwendet, haben weitere Änderungen der Syscall Tabelle, z.B. durch legitime Module, keinen Effekt auf das System. Die Manipulation ist schematisch in Abbildung 4 dargestellt.

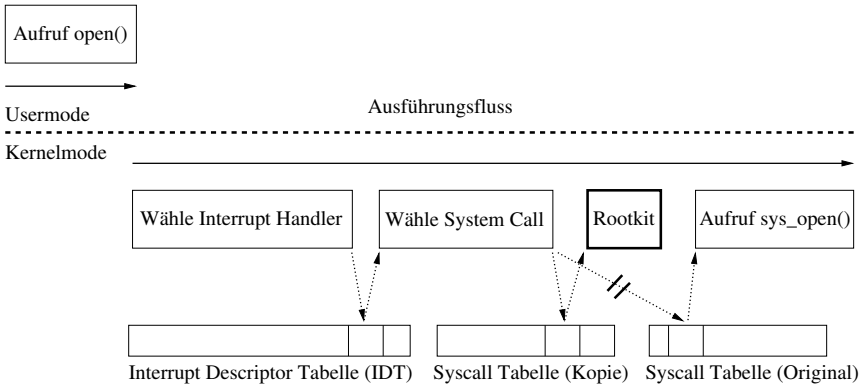

Abbildung 4: Das Rootkit SucKIT kopiert die Syscall Tabelle und manipuliert lediglich die Kopie. Der Interrupt Handler für Systemaufrufe wird derart verändert, dass die manipulierte Kopie zur Bestimmung der aufzurufenden Funktion verwendet wird.

3.3 Adore-NG

Das Rootkit Adore-NG unterscheidet sich erheblich von seinem Vorgänger Adore und wird daher separat in der Version 1.31 unter Linux betrachtet.

3.3.1 Transfer in den Kernel Mode

Ähnlich wie das ursprüngliche Adore wird der Programmcode des Rootkits als Modul in den Kernel geladen. Hier besteht aber die Möglichkeit, den Programmcode des Rootkits an ein bestehendes, dem Administrator bekanntes Modul anzuhängen. Durch Austausch von Einträgen in der Symbol Tabelle des Moduls wird zuerst das Rootkit und danach das eigentliche Modul ausgeführt [tr03].

3.3.2 Umleitung des Programm Flusses

Adore-NG leitet den Programm Fluss des Kernels an einem späteren Zeitpunkt um als die meisten anderen Rootkits. Anstatt die Syscall Tabelle oder ähnlich zentrale Strukturen des Kernels zu manipulieren, wird das virtuelle Dateisystem manipuliert, von dem alle Systemaufrufe Gebrauch machen, die in irgendeiner Form Dateien manipulieren. Da fast

alle Informationen auf UNIX-artigen Systemen als Dateien dargestellt werden, sind auf diese Weise die gleichen Funktionen wie bei herkömmlichen Rootkits realisierbar.

4 Abwehrmaßnahmen

Die eingesetzten Abwehrmaßnahmen sollten eine geschlossene Kette von Prävention über Detektion zur Reaktion bilden. Im Folgenden werden einige Ansätze zur Prävention und Detektion vorgestellt. Die Reaktion ist ein umfangreiches eigenständiges Thema, welches den Rahmen dieses Artikels sprengen würde.

4.1 Prävention

Es existiert eine Fülle von Ratgebern und Anleitungen, wie ein mit dem Internet verbundenes System abzusichern ist. Im Folgenden werden drei speziell auf Kernel Rootkits ausgerichtete Maßnahmen vorgestellt.

4.1.1 Deaktivierung von Kernel Modulen

Die Unterstützung von Kernel Modulen kann insgesamt auf einem System deaktiviert werden, um das Laden eines Rootkits zu verhindern. Kernel Rootkits, die als Modul geladen werden, haben dadurch keine Möglichkeit sich zu installieren. Alle Rootkits, die über `/dev/kmem` oder andere Wege in den Kernel gelangen sind davon nicht betroffen (vgl. 3.2). Der administrative Aufwand für das so geschützte System steigt dadurch an, dass alle benötigten Treiber fest in den nun statischen Kernel eingebunden werden müssen. Als Grundsicherung ist die Deaktivierung von Kernel Modulen an Systemen damit sinnvoll, solange der statische Kernel keinen zu großen Mehraufwand bedeutet.

4.1.2 Einsatz von Mandatory Access Controls

Der einzige effektive Weg einem Rootkit den Zugriff auf `/dev/kmem` zu verwehren, ist der Einsatz von Mandatory Access Controls und damit die strenge Reglementierung der Zugriffsrechte auch für den User `root`. Mandatory Access Controls können je nach System durch separate Patches eingerichtet werden, oder sind in einigen UNIX-artigen Systemen bereits enthalten. Beispiele für solche Systemen sind SELinux [NS04], Trusted BSD [BS04] und Trusted Solaris [SU04]. Um die Installation eines Rootkits zu verhindern muss eine sehr genaue Konfiguration stattfinden. Weiterhin müssen entsprechende Verfahren bei der täglichen Administration eingehalten werden.

Schreibender Zugriff auf `/dev/kmem` wird eventuell von manchen Programmen benötigt, so dass dort auf Alternativen ausgewichen werden muss. Je nach Konfiguration und Hardware verlangt z.B. der X-Server XFree86 schreibenden Zugriff auf `/dev/kmem`.

4.1.3 Laden von Anti-Rootkit Modulen

Spezielle Anti-Rootkit Module können die Installation eines Rootkits erschweren, indem die Systemaufrufe zum Laden von Modulen verändert werden und z.B. so genannte *Watchdog Funktionen* in regelmäßigen Abständen aufgerufen werden. Diese testen die Integrität des Systems im Allgemeinen und speziell des Anti-Rootkit Moduls, um im Falle einer Kompromittierung Alarm zu schlagen.

Ein Beispiel eines solchen Anti-Rootkit Moduls ist `StMichael`. Einige Rootkits wurden in der Zwischenzeit speziell zur Umgehung und Deaktivierung von `StMichael` angepasst.

4.2 Detektion

Die Detektion von Rootkits findet oft aufgrund eines unbestimmten Verdachtes statt. Dies hat zur Folge, dass je nach zu untersuchendem System weder Offline-Analyse noch ein Neustart in Frage kommen. Weiterhin ist der maximale Zeitaufwand für eine Untersuchung oft recht gering. Diese Randbedingungen schränken die einsetzbaren Methoden erheblich ein, da in der Praxis immer ein Kompromiss zwischen Aufwand und Sicherheit gemacht werden muss. Im Folgenden werden die wichtigsten Methoden zur Detektion vorgestellt.

4.2.1 Integritätstests durch Checksummen

Der Einsatz von Integritätstests durch Checksummen entstand als Reaktion auf Rootkits, die einzelne Befehle des Systems austauschen. Dabei werden Checksummen von wichtigen System Dateien erzeugt und extern gelagert. Der Test besteht aus der erneuten Erzeugung dieser Checksummen und dem Vergleich mit den gespeicherten Werten. Obwohl solche Rootkits immer seltener von Angreifern benutzt werden, ist dieser Ansatz auch bei Kernel Rootkits hilfreich. Diese sind zwar in der Lage beliebig Zugriffe auf einzelne Dateien durch die manipulierten Systemaufrufe zu kontrollieren, aber oft bleiben trotzdem Spuren des Rootkits im System zurück. Entweder durch Konfigurationsfehler der Angreifer, oder durch unvollständige Tarnung durch das Rootkit. Alle Möglichkeiten zur Tarnung eines Kernel Rootkits werden außer Kraft gesetzt, sobald das verdächtigte System offline untersucht wird. Checksummen sind in diesem Fall sehr hilfreich, um die Vorgänge auf dem betroffenen System zu rekonstruieren.

Integritätstest dieser Art werden unter anderem durch die Programme `tripwire` und `aide`[LV04] durchgeführt. Ein Nachteil dieses Tests ist die Notwendigkeit die Listen mit Checksummen auf aktuellem Stand zu halten, was einen erheblichen organisatorischen und zeitlichen Aufwand bedeuten kann. Eine einfache Version von Integritätstests kann auch mit einem Paket-Systemen wie z.B. `rpm` durchgeführt werden, das bereits auf vielen Systemen vorhanden ist. Diese enthalten oft Mechanismen zur Konsistenz-Prüfung der installierten Programme. Die Kontrolle von Checksummen mit Hilfe von `aide` und `rpm` ist beispielhaft in Abbildung 5 dargestellt.

```
linux:/home/rks # aide -C
AIDE found differences between database and filesystem!!
Start timestamp: 2003-01-13 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 5: Das Programm `aide` testet, ob die Checksummen verschiedener Programme noch mit den gespeicherten Werten übereinstimmen. Der gleiche Test kann mit Hilfe von `rpm` durchgeführt werden. Beide Programme stellen eine Diskrepanz fest; bei `rpm` ist dies an der Ausgabe von '5' bei veränderter MD5 Checksumme zu erkennen.

4.2.2 Kontrolle kritischer Kernel Strukturen

Eine Reihe zentraler Strukturen im Kernel eines UNIX-artigen Systems bieten sich zur Manipulation durch ein Rootkit an, um den Programm Fluss des Kernels möglichst einfach zu manipulieren. Beispiele für solche Strukturen sind die Interrupt Descriptor Tabelle, die Syscall Tabelle, der Interrupt Handler und der Syscall Handler. Der Inhalt dieser Strukturen und ihre Position im Speicher können ähnlich wie bei den oben beschriebenen Integritätstests auf einem externen Medium gespeichert werden, um bei Bedarf diese Referenzwerte mit den aktuellen Strukturen zu vergleichen. Auch hier müssen die Vergleichswerte vor dem Verdachtsfall erhoben werden, damit ein sinnvoller Test stattfinden kann. Programme wie `kstat` [Fu04] und `ksec` [Pi04] können derartige Tests durchführen (vgl. Abbildung 6).

4.2.3 Laufzeit Untersuchung von Systemaufrufen

Die von Kernel Rootkits eingesetzten Algorithmen zur Tarnung der Anwesenheit des Angreifers sind derart komplex, dass ein nicht unbeträchtlicher Anteil der Ausführungszeit eines Systemaufrufs darauf verwendet wird. Ein allgemeiner Ansatz zur Detektion von Kernel Rootkits ist daher die Messung der Anzahl der Instruktionen, die bei der Abarbeitung bestimmter Systemaufrufe vom Prozessor des Systems ausgeführt werden. Hierbei können wie bei den oben beschriebenen Integritätstests im Vorfeld Referenzwerte bestimmt werden, um diese mit aktuellen Messungen zu vergleichen. Die Veränderung der Laufzeit durch manche Rootkits ist allerdings so groß, dass generische Werte für die je-

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

linux:/home/tools/KSTAT24/2.4.16 # ./kstat -s 0
No System Call Address Modified

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

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_ni_syscall                  0xf880c5b0 WARNING! should be at 0xc013ec5c
linux:/home/tools/KSTAT24/2.4.16 # ./kstat -s 1
Restoring system calls addresses...
```

Abbildung 6: Der Inhalt der Syscall Tabelle ist mit Hilfe des Programms `kstat` gespeichert worden, um diesen nun mit den Werten vor und nach Installation eines Rootkits vergleichen zu können. Die Manipulation der Einträge mehrerer Systemaufrufe wird dabei entdeckt.

weiligen Betriebssysteme bzw. Kernel-Versionen oft ausreichen. Eine solche Messung ist mit der Referenz-Implementierung `patchfinder` von Jan Rutkowski möglich [Ru02], wie sie in Abbildung 7 zu sehen ist.

```
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!
(...)
```

Abbildung 7: Die Anzahl der ausgeführten Instruktionen bei der Abarbeitung verschiedener Systemaufrufe wurde zuvor gemessen und ist in der Spalte 'clear' zu sehen. Übersteigt die Differenz zu den aktuellen Werten der Spalte 'current' eine gewisse Schwelle, wird eine Warnung ausgegeben.

4.2.4 Forensische Untersuchung

Besteht die Möglichkeit das System vom Netz zu trennen, kann eine forensische Untersuchung durchgeführt werden. Hierbei können Manipulationen aufgedeckt werden, die oben beschriebenen Integritätstests entgehen würden. Eine solche Untersuchung ist allerdings zeitaufwendig und erfordert weitere spezielle Kenntnisse.

Beschränkt sich ein Rootkit auf die Manipulation des Systems zur Laufzeit, ohne dass

Daten auf der Festplatte manipuliert werden, liefert eine forensische Untersuchung keine Erkenntnisse. In einem solchen Fall ist es notwendig auch den momentanen Zustand des Systems durch ein Speicher Abbild in die Analyse einzubeziehen, wie es rudimentär z.B. mit dem Coroner's Toolkit [Ve04] möglich ist.

4.2.5 Kontrolle des lokalen Netzwerks

Bei dem Einbruch in das System wird bestimmter Netzwerk Verkehr erzeugt, der bereits einen Alarm auslösen kann. Das kompromittierte System wird dann in der Regel vom Angreifer in einer Art missbraucht, die neuen Verkehr erzeugt. Die Kontrolle des Netzwerkes ist damit ein guter Indikator, um ein kompromittiertes Systeme zu entdecken und um eine erste Schätzung bzgl. der installierten Software abzugeben. Der Einsatz eines Intrusion Detection Systems kann der nächste Schritt sein, um den Angriff und den Missbrauch des Systems gezielt zu entdecken. Diese Systeme stellen ein umfangreiches Thema dar, welches nicht innerhalb dieses Artikels ausreichend diskutiert werden kann.

4.2.6 Rootkit Detektoren und manuelle Suche

Kein Rootkit ist in der Lage das unkompromittierte System perfekt nachzuahmen. In irgendeiner Form werden immer Anomalien erzeugt, die mehr oder weniger leicht zu finden sind. Ist ein Rootkit erstmal bekannt und wird hinreichend untersucht, werden solche Anomalien gefunden und dokumentiert. Diese von Rootkit zu Rootkit unterschiedlichen Anomalien werden z.B. von den Programmen `chkrootkit` und `Rootkit Hunter` genutzt, um eine ganze Reihe von Rootkits zu erkennen. Damit ein Rootkit erfolgreich auf diese Weise gefunden wird, muss es bereits bekannt sein und es muss eine Routine zur Erkennung seiner speziellen Anomalien existieren.

Besteht bereits der Verdacht auf Installation eines bestimmten Rootkits, kann dieses oft relativ leicht gefunden werden. Wird z.B. das Rootkit SucKIT erwartet, kann dieses anhand der Lade Routine in `/sbin/init` leicht enttarnt werden, wie in Abbildung 8 zu sehen ist. Eine allgemeine Methode zum Auffinden eines Kernel Rootkits ist weiterhin eine gezielte Suche im Dateisystem `/proc`, um versteckte Prozesse zu finden.

5 Anwendung der Abwehrmaßnahmen

Die vorgestellten Methoden zur Prävention und Detektion werden im Folgenden an den Beispiel Rootkits getestet. Als Test Plattform wird Linux 2.4.16 auf einem 32 Bit Intel System verwendet. Die Effektivität der Präventionsmaßnahmen ist stark von der lokalen Konfiguration abhängig. In Tabelle 1 wird daher nur das Abschalten des Modul-Supports im Kernel und der Einsatz von Mandatory Access Controls zur Unterbindung des schreibenden Zugriffs auf `/dev/kmem` betrachtet. Ein Eintrag von 'reicht aus' impliziert dabei die korrekte Anwendung, welche insbesondere bei den Mandatory Access Controls nicht trivial ist.

```
linux:/sbin # ls -al init*
-rwxr-xr-x    1 root     root        392124 Jan  6  2003 init
linux:/sbin # mv init init.bak
linux:/sbin # ls -al init*
-rwxr-xr-x    1 root     root         28984 Jan  6  2003 init.bak
linux:/sbin # ./init.bak
/dev/null
Detected version: 1.3b
use:
./init.bak <uivfp> [args]
u       - uninstall
i       - make pid invisible
v       - make pid visible
f [0/1] - toggle file hiding
p [0/1] - toggle pid hiding
linux:/sbin #
```

Abbildung 8: Es besteht der Verdacht auf Installation des Rootkits SucKIT. Das Rootkit manipuliert 24 Systemaufrufe, aber der von `mv` verwendete Aufruf `rename()` wird nicht manipuliert. Bei Ausgabe der Größe der Datei `/sbin/init` wird durch die manipulierten Systemaufrufe auf das versteckte, umbenannte Original zugegriffen. Der Aufruf von `mv` greift auf das tatsächliche, manipulierte Programm zu, welches danach seine Größe geändert zu haben scheint. Ein Aufruf des so enttarnten Programms bestätigt nochmal den Verdacht der Installation von SucKIT.

Präventionsmaßnahme	Adore 0.42	SucKIT 1.3b	Adore-NG 1.31
Modul-Support abschalten	Reicht aus	Reicht nicht aus	Reicht aus
Mandatory Access Controls	Reicht aus	Reicht aus	Reicht aus

Tabelle 1: Wirksamkeit der Präventionsmaßnahmen im Falle der Beispiel Rootkits.

Von den vorgestellten Detektionsmaßnahmen werden lediglich die zur Laufzeit anwendbaren bzgl. der Beispiel Rootkits getestet. Vor der Installation der Rootkits sind Checksummen von wichtigen System Dateien und Verzeichnissen erstellt worden. Die Rootkits verstecken jeweils ein Verzeichnis innerhalb des so überwachten Bereiches. Weiterhin wird ein SSH-Daemon versteckt, falls das Rootkit keine eigene Backdoor bereitstellt. Die Ergebnisse des Tests der Detektionsmaßnahmen sind Tabelle 2 zu entnehmen.

Das Rootkit Adore wird von allen vorgestellten Detektionsmaßnahmen gefunden. Das Programm `aide` findet das Rootkit SucKIT unter anderem, weil sich die Inode-Nummer der Datei `/sbin/init` ändert. Der Vergleich der Syscall Tabelle mit gespeicherten Werten mit Hilfe von `kstat` führt bei SucKIT nicht zur Detektion, da eine manipulierte Kopie der Tabelle verwendet wird. `Chkrootkit` detektiert SucKIT, indem die Existenz der Lade Routine in `/sbin/init` überprüft wird. Die Laufzeitmessung mit Hilfe von `patchfinder` schlägt beim Rootkit SucKIT fehl, da das dafür notwendige Modul aufgrund der Manipulation von Syscall Tabelle und Handler nicht geladen werden kann. Dieser Fehler sollte allerdings schon zu einem Alarm führen.

Art des Tests	Adore 0.42	SucKIT 1.3b	Adore-NG 1.31
Integritätstests mit `aide 0.7`	Wird erkannt	Wird erkannt (`/sbin/init`)	Wird erkannt
Kerneltest mit `kstat 2.4`	Wird erkannt	Wird nicht erkannt	Wird nicht erkannt
Merkmalssuche mit `chkrootkit 0.43`	Wird erkannt (verst. Prozesse)	Wird erkannt (`/sbin/init`)	Wird nicht erkannt
Laufzeit Test mit `patchfinder`	Wird erkannt	Modul kann nicht geladen werden	System Crash
Manuelle Suche in `/proc`	Wird erkannt	Wird erkannt	Wird nicht erkannt

Tabelle 2: Die zur Laufzeit anwendbaren Detektionsmaßnahmen werden an den Beispiel Rootkits getestet.

Das Programm `aide` erkennt bei Adore-NG wie auch bei den beiden anderen Rootkits, dass ein Verzeichnis versteckt wurde. Da Adore-NG die Syscall Tabelle nicht manipuliert, kann `kstat` auch keine Veränderung feststellen. `Chkrootkit` erkennt wie `aide`, dass ein Verzeichnis versteckt wurde. Eine Laufzeitmessung mit Hilfe von `patchfinder` führte wiederholt zum Absturz des Systems. Auch dieser Fehler sollte in der Praxis eine genauere Analyse des Systems nach sich ziehen. Eine manuelle Suche im Dateisystem `/proc` enttarnt keine von Adore-NG versteckten Prozesse.

6 Diskussion

Es sind die grundsätzlichen Konzepte und Vorgehensweisen moderner Rootkits vorgestellt worden, wobei besonders auf zwei grundsätzliche technische Erfordernisse eines Kernel Rootkits eingegangen wurde: Transfer in den Kernel Mode und Manipulation des Programm Flusses innerhalb des Kernels. Anhand von drei Beispiel Rootkits ist eine Klassifikation gemäß der Umsetzung dieser Erfordernisse möglich. Gängige Abwehrmaßnahmen wurden vorgestellt und deren Wirksamkeit an den Beispiel Rootkits getestet.

Tabelle 1 gibt Auskunft über die Wirksamkeit der vorgestellten Präventionsmaßnahmen. Mandatory Access Controls sind in der Lage die Installation eines Rootkits wirkungsvoll zu verhindern. Allerdings ist ihr Einsatz auch mit erheblichem administrativem Mehraufwand verbunden. Fehler bei Konfiguration und Betrieb können einem Angreifer Möglichkeiten zur Installation eines Rootkits eröffnen. Weiterhin erlauben Schwachstellen in elementaren Programmen oder dem Kernel die Umgehung jeglicher Präventionsmaßnahmen. Tabelle 2 stellt die Wirksamkeit der vorgestellten Detektionsmaßnahmen dar. Es war möglich Hinweise auf die Installation aller getesteten Rootkits zu finden. Wie zu erwarten sind die aktuelleren Rootkits schwerer zu entdecken als das ältere Adore. Es ist allerdings zu bemerken, dass für die von den Rootkits manipulierten Verzeichnisse zuvor Checksummen als Referenzwerte erstellt wurden. Entscheidet sich der Angreifer keine

Verzeichnisse und Prozesse zu verstecken, muss das Rootkit anhand der Manipulation im Kernel erkannt werden. Verändert das Rootkit den Ausführungsfluss im Kernel nicht an den offensichtlichen Stellen, wie z.B. Adore-NG, ist die Detektion extrem schwer.

Mit Ausnahme der Laufzeit Untersuchung von Systemaufrufen sind die meisten Detektionsmethoden nicht generisch genug, um neue oder leicht modifizierte Rootkits zu erkennen. Jedes Rootkit kann theoretisch erkannt werden, wenn ausreichend Ressourcen aufgewendet werden. In der Regel wird allerdings aufgrund eines vagen Verdachts nach einem Rootkit gesucht und es steht nur wenig Zeit zur Verfügung. Daraus ergibt sich ein Bedarf an allgemeinen Detektionsmethoden, die schnell und zur Laufzeit angewandt werden können. Eine Möglichkeit dafür stellen Konsistenz Tests des laufenden Systems dar, die auf abstrakte Beschreibungen beruhen anstatt z.B. lediglich gespeicherte Tabellen des Kernels mit aktuellen Versionen zu vergleichen. Die Autoren von Adore-NG verfolgen einen ähnlichen Ansatz: Der Ausführungsfluss des Kernels wird zur Laufzeit untersucht, um das Rootkit im virtuellen Filesystem zu verstecken [TE03]. Einerseits wird dadurch deutlich, dass aktuelle Rootkits viel generischere Techniken benutzen als die gängigen Werkzeuge zur Detektion. Andererseits wird ein Weg für die weitere Entwicklung von besseren Werkzeugen zur Detektion von Rootkits aufgezeigt. Die Analyse des Ausführungsflusses kann z.B. auch genutzt werden um ein laufendes System abstrakt zu beschreiben und so Konsistenz Tests zu definieren. Dies sind Themen weiterführender Forschung mit dem Ziel, Werkzeuge zu entwickeln, die Administratoren eine allgemeine Detektion von Rootkits zur Laufzeit zu ermöglichen.

Literatur

[Ba86] Bach, M.: *The Design of the UNIX Operating System*. Prentice Hall. 1986.

[Bl89] Black Tie Affair: Hiding out under unix. *Phrack*. 25. 1989. http://www.phrack.org/phrack/25/P25-06.

[BS04] BSD Projekt. Trusted BSD. 2004. http://www.trustedbsd.org/.

[Fu04] Fusys. kstat. 2004. http://www.s0ftpj.org/en/tools.html.

[Ha97] Halflife: Abuse of the linux kernel for fun and profit. *Phrack*. 50. 1997. http://www.phrack.org/phrack/50/P50-05.

[LV04] Lehti, R. und Virolainen, P. Aide. 2004. http://www.cs.tut.fi/~rammer/aide.html.

[NS04] NSA. SE Linux. 2004. http://www.nsa.gov/selinux/.

[Pi04] Pigpen. ksec. 2004. http://www.s0ftpj.org/en/tools.html.

[Pl99] Plasmoid. Attacking solaris with loadable kernel modules. 1999. http://packetstormsecurity.nl/groups/thc/slkm-1.0.html.

[Pr99] Pragmatic. Attacking freebsd with kernel modules. 1999. http://www.thehackerschoice.com/papers/bsdkern.html.

[Ru02] Rutkowski, J. K.: Execution path analysis: finding kernel based rootkits. *Phrack*. 59. 2002. http://www.phrack.org/phrack/59/p59-0x0b.txt.

[SD01] SD: Linux on-the-fly kernel patching without lkm. *Phrack*. 58. 2001. http://www.phrack.org/phrack/58/p58-0x07.

[Sk03] Skoudis, E.: *Malware. Fighting malicious Code*. Prentice Hall. 2003.

[SU04] SUN Microsystems. Trusted Solaris. 2004. http://wwws.sun.com/software/solaris/trustedsolaris/.

[TE03] TESO. Codeflow Analyse, Vortrag auf dem 19. Chaos Communications Congress, Berlin. 2003. http://www.team-teso.net/articles/19c3-speech/.

[tr03] truff: Infecting loadable kernel modules. *Phrack*. 61. 2003. http://www.phrack.org/phrack/61/p61-0x0a_Infecting_Loadable_Kernel_Modu%les.txt.

[Ve04] Venema, W. The coroner's toolkit. 2004. http://www.porcupine.org/forensics/.