Rootkits

Grundlagen unter Unix / Linux,
Erkennung, Entfernung und
Schutzmaßnamen

Wilhelm Dolle, Head of Networking Unit & IT-Security (wilhelm.dolle@interActive-Systems.de)
www.interActive-Systems.de/security

Nach einem erfolgreichen Angriff

Rootkits

- Was geschieht nach einem erfolgreichen Einbruch?
  - Überwachungsmethoden analysieren
  - Verwischen von Spuren / Entdeckung vermeiden
  - Installation eines Rootkits
  - Einrichtung von Backdoors
  - Angriff auf weitere Systeme

Was sind Rootkits?

- Oft leichte Installation
- Verbergen Spuren (Dateien, Prozesse, ...) des Einbrechers
- Bringen trojanisierte Systemprogramme und andere Tools mit („Admin Bausatz“)
- Schaffen dauerhaften Zugang zum kompromittierten System (Backdoor)

- Dateibasierte/-modifizierende Rootkits
- Kernelbasierte/-modifizierende Rootkits

Dateibasierte Rootkits

- Ersetzen bzw. verändern Systembefehle und Überwachungsprogramme
- Klassische Rootkits (z.B. lrk3, lrk4, lrk5, t0rnkit)

Dateibasierte Rootkits - t0rnkit

- Weit verbreitetes Rootkit für Unix und Linux
- Ersetzt unter anderem die Programme du, find, ifconfig, ls, netstat, ps, top und login
- Über Konfigurationsdatei anpassbar (z. B. /dev/.hidden/psconf)
- login: bei bestimmten IP-Adressen root-Zugang ohne Passwort und Logging möglich
- In Standardinstallation TCP-Port 47017 offen

- LKM oder direkte Modifikation des Kernels im Speicher
- Vorteile:
  - Privilegierter Zugriff auf das System
  - Behandlung von Netzwerkpaketen vor lokaler Firewall
  - Manipulation der Systemsprungtabelle oder direkt der Systemfunktionen
  - Keine Änderung von Systemprogrammen nötig

Interessante Systemfunktionen (1)

Rootkits

- open() – lesender Zugriff Original, ausführender Zugriff trojanisierte Datei
- getdents(), mkdir(), chdir(), rmdir() – Verstecken von Verzeichnissen/Dateien

Interessante Systemfunktionen (2)

Rootkits

- execve(), clone(), fork() – Ausführen von Programmen mit bestimmten Eigenschaften (verstecken), und Vererbung an Kindprozesse
- stat() – Manipulation der Dateieigenschaften
- ioctl() – Device-Kontrolle, z.B. kein promisc-Bit (Sniffer) zu sehen

Rootkits die LKM benutzen

- Erstes Auftauchen vor etwa 5 Jahren
- Rootkits benutzen ladbare Kernelmodule (benötigen: insmod, lsmod, rmmod)

- Verbreitete Exemplare
  - Knark (für Kernel 2.2) – speziell zur Täuschung von Programmen wie Tripwire oder md5sum
  - Adore (für Kernel 2.2 und 2.4)
    - Module: adore.o und cleaner.o
    - Komandozeilentool: ava

Rootkits direkt über den Hauptspeicher

- 2001: KIS
- 2002: SucKIT

- Benötigen keine LKM Unterstützung
- Greifen direkt auf den im Hauptspeicher befindlichen Kernel über /dev/kmem zu
- Technik wurde schon 1998 beschrieben
- Monolitischer Kernel hilft nicht mehr

Kernel Intrusion System (KIS)

- DefCon 9 / 2001: Kernel Intrusion System (KIS) von optyx
- Bringt eigenen Modullader mit und benötigt keine LKM (Kernel-Memory-Patching)
- Versteckte Hintertür: lauscht erst auf einem Port nachdem ein spezielles Paket (beliebiger Port) an den Rechner geschickt wurde (Stealth-Backdoor)
- Graphischer Client und Server
- Interface für Plug-ins (leicht erweiterbar)

Rootkits starten

- Modifikation des *init*-Programms
- Modifikation der Startscripte
- Modifikation von Serverprogrammen die oft beim Systemstart aufgerufen werden (*sshd*, *httpd*, ...)

Wie fällt ein kompromittiertes System auf?

Rootkits

- Intrusion Detection Systeme (host- / netzwerkbasierend)
- Tripwire

- Erhöhter Netzwerkverkehr
- Verstoß gegen die Sicherheitsrichtlinien (nicht erlaubte Protokolle, Zugriffszeiten, ...)
- Dateisystem wird stärker genutzt
- Höhere Prozessorlast
- Geänderte Passwörter / neue Benutzer

System untersuchen (1)

Rootkits

- Wichtige Dateien überprüfen (inetd.conf / xinetd, services, Startdateien in rc.d, ...)
- Oft „trojanisierte“ Programme
  - ps, ls, find, ifconfig, netstat, du, df
  - sshd, httpd
  - login, passwd
  - inetd, tcpd
- Verdächtige MACtimes in /sbin/ oder /usr/sbin? (verdächtige Binaries mit „strings“ ansehen)
- lsof +L1

System untersuchen (2)

Rootkits

- Geladene Module prüfen
- Nach Interfaces im „promiscous mode“ suchen (Sniffer)
- Analyse der installierten Pakete mit RPM

  rpm --verify --all

- Logfiles auf verdächtige Einträge durchsuchen; Beispiel:

  Oct 31 16:52:33 opfer telnetd: connect from x.x.x.x

  Oct 31 16:52:34 opfer telnetd: ttloop: peer died: ...

  - => Buffer-Overflow-Angriff auf telnetd

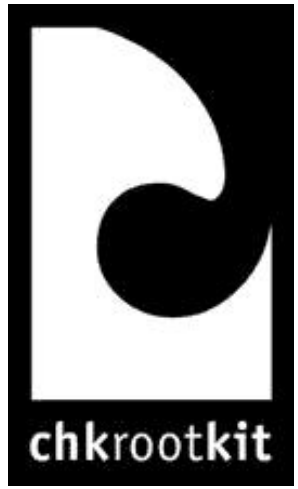
Rootkits finden (Konzepte)

- Offene Ports (Scan von außen mit netstat vergleichen – hidden Backdoor?)
- Modulliste (wenn nicht versteckt)
- Vergleich der Systemsprungtabelle mit System.map
- Vergleich der Kernelsymboltabelle (/proc/ksyms) mit der eines „sauberen“ Kernels
- Signatur (wenn bekannt und nicht verschlüsselt) im Speicher suchen (z.B. strings /dev/kmem)
- PID-Test (versuchen alle „freien“ PIDs durch einen Testprozess zu belegen)

- www.s0ftpj.org
- Ursprünglich für Linux Kernel 2.2, inzwischen portiert auf 2.4
- Kompilieren mit sauberem Kernel (passend zum Zielsystem)
- Findet geladene Module (die zB. mit *lsmod* nicht angezeigt werden) und kann sie entfernbar machen
  - kstat –M
- Zeigt veränderte Systemcalls an
  - kstat –S

- www.chkrootkit.org
- Lokal ausführbares Script
- Erkennt nahezu alle bekannten Rootkits
- Benutzt lokal vorhandene Befehle
- Probleme zB. mit *ifconfig* (besser *ip link show*)
- Evtl. Einsatz von einem schreibgeschützten Medium mit allen notwendigen (statisch gelinkten) Befehlen:

  chkrootkit –p /mnt/cdrom/bin



Entfernen von Rootkits

- Backup einspielen erst nach digitaler Forensik (wann war das letzte „saubere“ Backup?)

- Manipulierte Dateien durch Originale ersetzen
- Module entfernen
- Startscripte säubern / Startpunkte des Rootkits entfernen

- Wenn möglich: System komplett neu aufsetzen!

Proaktive Schutzmaßnamen (1)

Rootkits

- Gut ausgebildete Systembetreuer (sowie ausreichend Zeit)
- Verhindern des Ausnutzens von Buffer-Overflows und Format-String Angriffen
- Auditing von in den Kernel eingefügten Modulen (Modifikation von insmod)
- Tripwire
- Snort (Download oder Kommunikation von Rootkits erkennen)

Proaktive Schutzmaßnamen (2)

Rootkits

- Problem der Unix-Architektur: der allmächtige Administrator *root*

- LIDS
- NSA Linux
- GrSecurity
- RSBAC

- LIDS (www.lids.org) ist ein Patch für den Kernel
- Es werden Features implementiert, die Linux im Vergleich zu anderen OS fehlen
  - Schutz/Kontrolle/Verstecken von Dateien (z.B. /etc/passwd), Geräten (z.B. /dev/hda), Speicher (/dev/kmem) und Prozessen(z.B. tripwire) (auch vor root!)
  - Integrierter Portscan-Detector
  - ACLs für User, Einschränkungen für root (z. B. keine Module laden, kein Neustart)
- Administration erst nach RIPE-MD Kennwort oder Neustart

Zusammenfassung

- Dateibasierte/-modifizierende Rootkits
- Kernelbasierte/-modifizierende Rootkits

- Rootkits finden (Konzepte, kstat, chkrootkit)

- Proaktive Schutzmaßnamen