

# Was tun, wenn man schon drin ist?

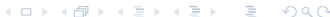
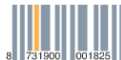
## Rootkits nutzen, Spuren finden und verwischen

losTrace (David Rasmus Piegdon)

Chaos Computer Club Cologne e.V.

U23 2006

23. September 2006



The likelihood of getting arrested for computer hacking has increased to an unprecedented level. No matter how precautionary or sage you are, *you're bound to make mistakes*. And the fact of the matter is if you have trusted anyone else with the knowledge of what you are involved in, you have made your first mistake.

(phrack 52.8, emph by author)



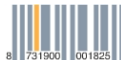
# Gliederung

- 1 warum Rootkits?
- 2 Eindringlinge finden
- 3 Local Info Hiding
- 4 Userspace
- 5 Kernel
- 6 Kernelspace



# Systemeinbruch und Folgen

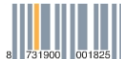
## Systemeinbruch



# Systemeinbruch und Folgen

Systemeinbruch

→ Spuren

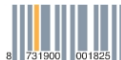


# Systemeinbruch und Folgen

## Systemeinbruch

→ Spuren

→ Entdeckung



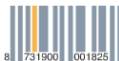
# Systemeinbruch und Folgen

## Systemeinbruch

→ Spuren

→ Entdeckung

- Rausschmiss
- entfernen der Lücke
- Strafrechtliche Verfolgung



# Strafrechtliche Verfolgung?

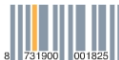
Wir wollen nicht bestraft werden! Ein bisschen Kybernetik:



# Strafrechtliche Verfolgung?

Wir wollen nicht bestraft werden! Ein bisschen Kybernetik:

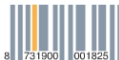
- Nicht einbrechen  
(ist immer eine Wahl!)



# Strafrechtliche Verfolgung?

Wir wollen nicht bestraft werden! Ein bisschen Kybernetik:

- Nicht einbrechen  
(ist immer eine Wahl!)
- Keine Spuren hinterlassen  
(ist unmöglich!)

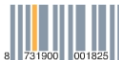




# Strafrechtliche Verfolgung?

Wir wollen nicht bestraft werden! Ein bisschen Kybernetik:

- Nicht einbrechen  
(ist immer eine Wahl!)
- Keine Spuren hinterlassen  
(ist unmöglich!)
- Nicht Entdeckt werden  
ist möglich: So wenig Spuren wie möglich; Spuren verwischen, verschleiern; sich unsichtbar machen



# Eindringlinge finden

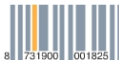
Welche Möglichkeiten als Administrator habe ich, nach Eindringlingen ausschau zu halten?



# Eindringlinge finden

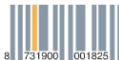
Welche Möglichkeiten als Administrator habe ich, nach Eindringlingen ausschau zu halten?

- Logfiles, Syslog
- Honeypots, Honeynets
- Intrusion Detection Systems (IDS), Netzwerkverkehr beobachten, NIDS
- Dateien beobachten
- Prozesse beobachten



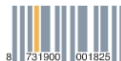
# Logs, Syslog

- Syslog-Printer (vollkommen offline)
- Syslog-Server (sehr schwer erreichbar für Angreifer)
- automatische Log-Watcher, zB *logsentry* (Whitelist auf Logs, alles Unbekannte wird per Email an den Admin geschickt)



# Honeypots, Honeynets

- Angreifer in eine Falle laufen lassen
- Angriffsfläche bieten, wo keine ist
- bisher unbekannte Exploits finden



# Honeypots, Honeynets

- Angreifer in eine Falle laufen lassen
- Angriffsfläche bieten, wo keine ist
- bisher unbekannte Exploits finden

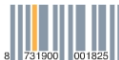
## als Angreifer:

- beware the honey! Vorsichtig sein (immer...)!
  - Laufen wir in VM?
    - qemu, UML, ... lassen sich teilweise identifizieren: kernel commandline, System.map, Kernel-Symbole, ...



# IDS, NIDS

- Matching auf seltenes/seltsames Verhalten, Pattern Matching (eventuell sogar whitelist)
- oft unsichtbar, da Sniffer/NIDS auch auf Routern oder Firewalls laufen können



# IDS, NIDS

- Matching auf seltenes/seltsames Verhalten, Pattern Matching (eventuell sogar whitelist)
- oft unsichtbar, da Sniffer/NIDS auch auf Routern oder Firewalls laufen können

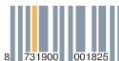
## als Angreifer:

- bietet eventuell neue Angriffsfläche (zB *ethereal* bzw. *wireshark* (Sniffer) hatte diverse Exploits)
- bekannte Pattern vermeiden, Shellcodes obfuscaten (ASCII-only Shellcode)
- nicht von zuhause
- TOR-Netzwerk ist auch nicht die feine Art (ist nicht für sowas gedacht)



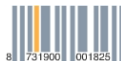
# Dateien beobachten

- Timestamps (*ctime*: creation, *mtime*: modification, *atime*: access(!))
- *tripwire*, *AIDE* bilden Checksummen über Dateien und vergleichen mit Originalwerten
  - besonders auf Programme und Configfiles (`/etc` `/bin` `/sbin` `/usr/bin` `/usr/sbin...`)
- shell-history (wird oft vom Angreifer vergessen)
- `wtmp`, `btmp` (*ssh* benutzt? `bash -login?`)



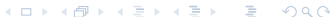
# Prozesse beobachten

- `/bin/ps aux`, `/bin/ps -elf, ...`
- `lsOf`: list all open files (`-i -n`: all sockets)
- `netstat`
- basieren alle auf Kernelinterfaces:
  - `/proc`, `/sys`, `syscalls`



# Local Information Hiding

- als Angreifer: Informationen geheim halten. Wie?
- wir (sollten) wissen, welche Informationen wo abgegriffen werden können.
- verstecken wir sie also





# komische Datei- und Prozessnamen

- zB durch Verzeichnis namens „. \_“, „. . \_“, „\_“, usw.
- oder in `/usr/share/man/pl/man5`
- und Prozesse namens „apache“, „sh“, usw.

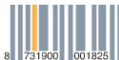
## als Admin:

- *(s)locate, find, AIDE*
- `/proc/$PID/exe`



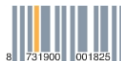
# Rootkits

- Beeinflussen und Filtern der Interfaces, die der Admin benutzt
- mehr dazu im nächsten Kapitel
- `chkrootkit` sucht nach diversen Spuren. Findet aber nicht alles!



# Lowlevel-Zugriff auf Datenträger

- Bad-Blocks (als solche markieren)
- unbenutzter Platz in Inodes/Sektoren (zB hinter dem Ende von Dateien)
- Erster Zylinder der Festplatten (Vorsicht mit grub u.ä.)
- Flash, nvram, BIOS(?), ...
- im RAM

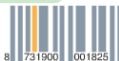


# Lowlevel-Zugriff auf Datenträger

- Bad-Blocks (als solche markieren)
- unbenutzter Platz in Inodes/Sektoren (zB hinter dem Ende von Dateien)
- Erster Zylinder der Festplatten (Vorsicht mit grub u.ä.)
- Flash, nvram, BIOS(?), ...
- im RAM

als Admin:

- *sleuthkit* und andere Autopsy-Tools



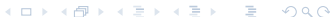
# Steganografie

- niederwertigstes Bit von WAV-files
- Bilder



# Steganografie

- niederwertigstes Bit von WAV-files
- Bilder
- Wenn man Steganografie wirklich sucht, kann man die meisten Implementationen auch finden!



# Steganografie

- niederwertigstes Bit von WAV-files
- Bilder
- Wenn man Steganografie wirklich sucht, kann man die meisten Implementationen auch finden!
- aber wer sucht schon danach?



# Definition Userspace-Rootkits

Userspace-Rootkits  
sind Rootkits, die komplett im Userspace residieren und  
vorhandene Programme ersetzen oder beeinträchtigen.



# Programme austauschen

- alle wichtigen Programme, womit uns ein Admin finden könnte:
  - `ps ls netstat lsof ...`



# Programme austauschen

- alle wichtigen Programme, womit uns ein Admin finden könnte:
  - `ps ls netstat lsof ...`

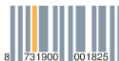
## als Admin:

- testen, ob die sich geändert haben:
- `md5sum` (auch mit austauschen?)
- `tripwire`, `AIDS`, `sleuthkit`, ...
- statisch gelinkte Binaries auf Read-Only Volumes



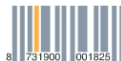
# LD\_PRELOAD

- Programme wie `ps` `ls` ... benutzen meist Library Calls und nicht direkt Syscalls
- LIBCs *open, close, read, write, fread, fwrite, ...*
- LD\_PRELOAD erlaubt das Laden eines beliebigen Shared Objects vor zB der LIBC.
- `LD_PRELOAD="/home/lostrace/my_filter.so" \`  
`ls -la`
- LD\_PRELOAD gibt uns viele Möglichkeiten :-)



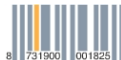
# LD\_PRELOAD entdecken

- aber schnell Entdeckbar:
- `echo $LD_PRELOAD` kann schon helfen
- *lsOf*
- `/proc/$PID/maps`
- `/proc/$PID/environ`



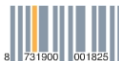
# ELF-Infection

- Infektion von vorhandenen Programmen (ähnlich wie echte Viren)
- ebenfalls findbar via Tripwire et al.



# Runtime Process Infection

- Laufende Prozesse infizieren
- keine Änderung auf Festplatte → nicht entdeckbar via Tripwire o.ä.
- eventuell keine Änderung der offenen Dateien, eventuell (bei `.so`-Injection) entdeckbar via *mmap* oder *open files*



# Definition Kernel

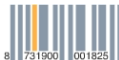
Ein *Betriebssystemkern* oder Systemkern ist der zentrale Bestandteil eines Betriebssystems. In ihm ist die Prozess- und Datenorganisation festgelegt, auf der alle weiteren Softwarebestandteile des Betriebssystems aufbauen.

(wikipedia)



# Monolithischer Kernel

Beim *monolithischen Kernel* arbeitet der gesamte Kernel mit allen Treibern in einem Adressraum. Daher gibt es keine Privilegien-Separierung: der gesamte Kernelcode kann auf alles zugreifen.



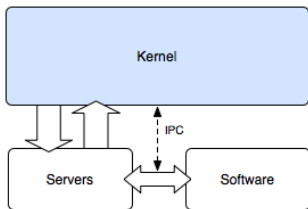
Monolithisch vs. Microkernel

# Monolithischer Kernel



# Microkernel

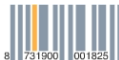
Beim *Microkernel* wird nur das allernötigste implementiert, was i.A. auf IPC und eventuell Policy-Enforcements reduziert werden kann. Alles andere wird in separierten und weniger privilegierten „Servern“ oder Userspace-Prozessen ausgeführt. Prozesse können **nur** via IPC interagieren.



# Microkernel

*A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e., permitting competing implementations would prevent the implementation of the systems' required functionality.*

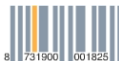
(wikipedia.en L4\_microkernel\_family)



# und Linux?

Alle häufig benutzten *Normalanwender*-Betriebssysteme sind monolithische Systeme oder „Hybridkernel“, bei denen beides benutzt werden kann. Die Realität zeigt aber: wenn der Hersteller die Wahl hat, wird der Treiber im monolithischen Stil geschrieben.

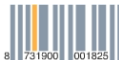
- Monolithisch: System V, Linux, {Free,Open,Net}BSD, Win98/ME, OS2
- Hybrid: Windows NT (XP?, Vista?), Darwin, BeOS
- Micro: QNX Neutrino, BeOS, Tru64, Amoeba



# Definition Kernel-Rootkits

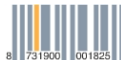
## Kernel-Rootkits

sind Rootkits, die in den Kernel-Space eindringen und sich dort festsetzen, um den Kernel zu beeinflussen.



# Einbruch in den Kernelspace

Code im laufenden Kernel ändern? Geht das?  
Wir brauchen nur Zugriff auf den Kernelspace.



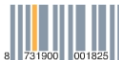
# Einbruch in den Kernelspace

Code im laufenden Kernel ändern? Geht das?

Wir brauchen nur Zugriff auf den Kernelspace.

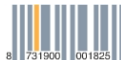
Diverse Methoden:

- Loadable Kernel Module (LKM)
- `/dev/kmem` Runtime Kernelpatching (obsolete since 2.6.17.irgendwas?)
- `/dev/mem` (ebenso?)
- Defekte im Code, der im Kernelspace läuft. (z.B. „böses“ Filesystem mounten)
- Hardware-Zugriff (zB auf RAM via FireWire)



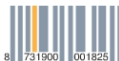
# Loadable Kernel Modules

- gedacht zum Nachladen von Treibern, erweitern der Funktionalität des Kernels
- Codebasis vorhanden
- eigenes Module schreiben und zum speziellen Kernel passend compilieren.
- sehr einfach zu erstellen



# LKM finden

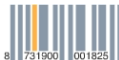
- finden via *lsmod* oder `/proc/modules`
- als Angreifer: verstecken, indem Liste der geladenen Module manipuliert wird
- als Admin: trotzdem teilweise noch entdeckbar, da bestimmtes Speicherverhalten
- Rootkits generell: finden via weiterer Methoden, später besprochen



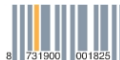
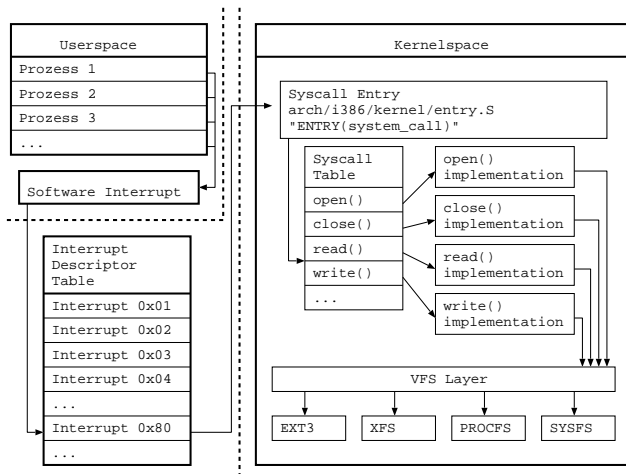
# Interessante Filtermethoden

- Prozesse verstecken
- Dateien verstecken
- Infos über offene Dateien, Sockets etc verstecken
- Andere Programme ausführen, als der Admin glaubt (READ auf Datei zeigt einen Inhalt, EXECVE auf Datei führt ganz andere Datei aus)
- much more...

Wenn wir ordentlich Filtern wollen, müssen wir wissen, wie die relevanten Interfaces funktionieren.



# Kernelspace Angriffsvektoren

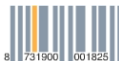


# IDT

In Interrupt Descriptor Table (IDT): Sprungadresse austauschen, also eigenen Interrupt-Handler implementieren, der den Ursprünglichen wrapped.

## Finden

- Regelmässiger vergleich der IDT mit ursprünglichen Werten.
- Check ob Sprungadressen im Kernelcode oder irgendwohin anders?



# entry.S

Syscall Entry Code mit unserem Code überschreiben. Etwas schwieriger, da Prozesse während dem Austausch hineinspringen könnten. → Vorher LOCKen. Dann Code mit Sprung in unsern Code austauschen, den Code wrappen o.ä.

## Finden, Abwehren

- MD5sum o.ä. über Syscall-Entry-Code
- FAR-JUMP im Entry-Code?
- Execute-No-Write Pages (Lässt sich auch umgehen)

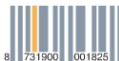


# Syscall-Table

Syscall-Table Einträge überschreiben (ab 2.5er Kernel ist die Adresse der Syscall-Sprungtabelle nicht mehr exportiert)

## Finden, Abwehren

- Symbol nicht mehr exportieren (lässt sich trotzdem finden...)
- Regelmässig mit Original vergleichen

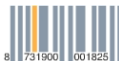


# Syscalls

## Einzelne Syscalls überschreiben

### Finden, Abwehren

- MD5sum o.ä. über Syscalls
- FAR-JUMP im Entry-Code?
- Execute-No-Write Pages (Lässt sich auch umgehen)

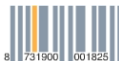


# VFS

VFS Layer manipulieren  
(dazu hab ich nicht grossartig viel gelesen ;)

## Finden, Abwehren

- MD5sum über Code, Pointer checken.
- Execute-No-Write Pages (Lässt sich auch umgehen)

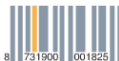


# FS-Treiber

Einzelne Filesystem-Treiber beeinflussen (besonders PROCFS und SYSFS)

## Finden, Abwehren

- MD5sum über Code, Pointer checken.
- Execute-No-Write Pages (Lässt sich auch umgehen)



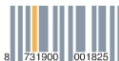
# Advanced Topics

## Advanced Topics



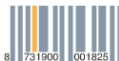
# Userlevel-Threading

- Infektion eines laufenden Prozesses
- implementierung eines Userlevel-Threading Algorithmus
- Userlevel-Threading: zwei Threads, in einem Userspace-Process gemerged



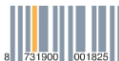
# Hidden Network Channels

- Hidden Information Channel in TCP-Timestamps und anderen Random basierten Protokollspezifischen Feldern
- Eventuell entdeckbar, weil die meisten Implementationen der Protokollstacks nicht pures Random für die Felder benutzen und so Identifizierbar ist, wenn plötzlich pures Random für die Felder benutzt wird



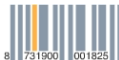
# Local Information Hiding? Warum?

- Warum überhaupt Informationen auf der Festplatte verstecken?
- Immer mehr Systeme mit hoher Uptime
- Rootkit ausschliesslich im RAM im Kernelspace
- Keinen Code sondern nur „unwichtige“ Datenstrukturen im Kernel ändern
- so nur noch extrem schwer entdeckbar



# Virtualisierungstechnologie

- Hypervisormode moderner Prozessoren: mehrere virtuelle PCs simulieren, mit Hardwareunterstützung
- zB XEN mit HWunterstützung kann unveränderte Betriebssysteme laufen lassen
- BluePill: ein solches Rootkit wurde kürzlich von Joanna Rutkowska vorgestellt
- RedPill: Programm zur Erkennung, ob in solcher VM
- Möglichkeit zur Entdeckung aber teilweise noch unklar
- sobald Hardwareunterstützung an der Stelle komplett (RedPill nutzt fehlende Unterstützung aus), wirds echt schwierig.



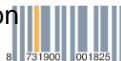


Fragen?



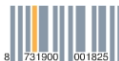
# Referenzen (1)

- <http://phrack.org>
  - Phrack 50.5: ttylogger, LKM
  - Phrack 51.5: File Descriptor Hijacking
  - Phrack 51.8: Shared Lib. Redirect. (LD\_PRELOAD, libdl)
  - Phrack 51.9: Bypassing Integrity Checking Systems
  - Phrack 52.6: Hardening the Linux Kernel
  - Phrack 52.8: Steganography Thumbprinting
  - Phrack 52.18: Weakening the Linux Kernel
  - Phrack 58.6: Advances in kernel hacking
  - Phrack 58.7: Linux on-the-fly kernel patching without LKM
  - Phrack 59.5: Advances in kernel hacking II
  - Phrack 59.8: Runtime Process Infection
  - Phrack 63.8: Raising The Bar For Win. Rootkit Detection
  - and much more...



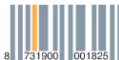
## Referenzen (2)

- [http://packetstormsecurity.nl/docs/hack/LKM\\_HACKING.html](http://packetstormsecurity.nl/docs/hack/LKM_HACKING.html)
- <http://packetstormsecurity.nl/UNIX/penetration/rootkits/>
- <http://www.chkrootkit.org/>
- Syscall-Proxying:  
<http://www.coresecurity.com/files/files/11/SyscallProxying.pdf>
- Kernel-space Modification via System-Management-Mode  
<http://www.cansecwest.com/slides06/csw06-duflot.ppt>



## Referenzen (3)

- <http://invisiblethings.org/papers.html>
- Passive Covert Channels in the Linux Kernel  
<http://invisiblethings.org/papers/passive-covert-channels-linux.pdf>
- Virtualisierungstechnik für Rootkit:
  - <http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html>
  - <http://www.theinvisiblethings.blogspot.com/>
  - RedPill: <http://invisiblethings.org/papers/redpill.html>



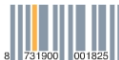
# Hands On

## Hands On



# Hands On: 1. (~15 min)

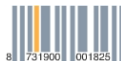
- schaut euch `/proc` und `/sys` an
  - was gibt es für Informationen? (Besonders über Prozesse)
  - welche Rechte haben die Files, wer kann was lesen?



# Hands On: 2. (~1 h)

In zwei Teams:

- ihr bekommt eine Box, die ihr aufmacht
- installiert ein Rootkit eurer Wahl
- verwischt eure Spuren, so gut ihr könnt



Hands On 3

# Hands On: 3. (~1 h)

- kurze erläuterung aller mir bekannten Lücken
- tauscht die Box
- findet das andere Team



# Abschlussbesprechung

- Was wurde gefunden? Warum?
- Was wurde nicht gefunden?
- Habt ihr mich gefunden?
- Never assume that you are the first!

