Kernel härten mit Openwall

JOSEF BRUNNER, SECURITY CONSULTANT

Wer sich auch nur ein wenig mit IT-Security und den täglich neu auftauchenden Sicherheitslücken beschäftigt, wird sehr bald merken, daß über neunzig Prozent der Lücken auf Buffer Overflows zurückzuführen sind. Der Openwall-Patch stellt eine Möglichkeit dar, diese Löcher zu stopfen.

Obwohl sich die meisten Administratoren über die Gefahren eines Buffer Overflows in den eingesetzten Programmen im Klaren sind, können sie aus Zeitgründen nicht täglich nach neuen Patches suchen, sie testen und gegebenenfalls einspielen. Es kann also durchaus vorkommen, daß ein Produktivsystem mit einer bekannten Sicherheitslücke im Einsatz ist. Um diese Situation zu entschärfen, kann der Administrator zumindest den Kernel so härten, daß Buffer-Overflow-Attacken im System wenig bis keinen Schaden mehr anrichten können.

Funktionsweise und Features

Dafür gibt es verschiedene Ansatzpunkte. In der Ausgabe 4'02 der freeX wurde bereits gezeigt, wie solche Lücken entstehen, wie man sie ausnutzt und welche Folgen sie haben können. Dieser Beitrag beschäftigt sich im folgenden mit dem Openwall-Patch [1] für den Linuxkernel und zeigt, wie man ihn einspielt, konfiguriert und welche Vorzüge er mit sich bringt. Im folgenden wird von der Linux-Kernelversion 2.4 ausgegangen.
Angriffsprogramme, sogenannte Exploits, versuchen, bösartigen Maschinencode auf dem Stack auszuführen und die Kontrolle über das System zu übernehmen. Der Openwall-Patch erschwert dies, da er den Stack (genauer gesagt, den User Stack) auf »Non-Executable« setzt. Dort kann dann kein Code mehr ausgeführt werden.
Eine weitere bei weitem nicht so verbreitete Angriffsmethode ist die Rücksprungadresse auf eine Funktion in der libc (zumeist *system()*) zu setzen und so Schaden anzurichten. Auch das wird mit dem Patch verhindert. Der Patch bietet außerdem die Möglichkeit, das Umgehen des Prozeßlimits (falls denn gesetzt) zu verhindern. Aber seien Sie mit dieser Option vorsichtig: Auf der RedHat-8.0-Standardinstallation des Autors verlief die Modifikation einwandfrei, SuSE 8.1 jedoch hatte danach einige Probleme.
Das Ausbrechen des Limits und noch weitaus gefährlichere Vorgänge können dadurch verhindert werden, daß man die laufenden Prozesse in ein enges Gewand steckt. Eine Möglichkeit ist das chroot-ing. Damit setzt man aber lediglich auf der Verzeichnisebene an. Zusätzlich sollte man den Diensten verbieten, auf nicht benötigte Systemaufrufe zuzugreifen oder unter Umständen ein eigenes Regelsystem dafür einführen. Beiden Methoden werden in der nächsten freeX ausführlich besprochen werden.
Openwall-Patch kann auch nicht gebrauchtes Shared Memory freigegeben werden. Auch hier sei darauf hingewiesen, daß einige Applikationen unter Umständen daraufhin nicht mehr richtig arbeiten. Getestet wurden erfolgreich die Server Apache, MySQL und vsftpd.

Installation

Nun zur Installation des Patches. Zuerst ist zu prüfen, ob der Quelltext der gewünschten Kernelversion auf der Festplatte vorliegt. Ist das der Fall, kann die Installation beginnen. Dazu muß der Kernel neu gebaut werden:
Im ersten Schritt besorgt man sich den Openwall-Patch aus dem Internet (*www.openwall.com/linux*) und fügt den Patch zu den Kernelquellen hinzu. Falls aus den Quellen, die für das Erstellen des neuen Kernels benutzt werden, schon einmal ein Kernel gebaut wurde, sollte man vor dem Neukompilieren mit dem Befehl *make clean* die Überreste des letzten Kompilierens beseitigen.

```
# cd /usr/src/linux
#cp /download/linux-2.4.20-ow0.tar.gz .
```

```
#tar xzf linux-2.4.20-ow0.tar.gz
#cd http://linux-2.4.20-ow0/
#mv linux-2.4.20-ow0.diff ../
#cd ..
#patch -p0 < linux-2.4.20-ow0.diff
patching file arch/alpha/config.in
patching file arch/alpha/defconfig
patching file arch/arm/config.in
patching file arch/arm/defconfig
patching file arch/cris/config.in
patching file arch/cris/defconfig
patching file arch/i386/config.in
patching file arch/i386/defconfig
patching file arch/i386/kernel/head.S
patching file arch/i386/kernel/signal.c
patching file arch/i386/kernel/traps.c
patching file arch/ia64/config.in
patching file arch/ia64/defconfig
patching file arch/m68k/config.in
patching file arch/m68k/defconfig
patching file arch/mips/config-shared.in
patching file arch/mips/defconfig
...
```

Nun wird der Kernel mit *make mrproper* kompiliert. Dann wird er konfiguriert. Zu empfehlen ist eigentlich die textbasierende Methode (*make config*). Aus Übersichtlichkeitsgründen wird aber gern die grafische Methode mit *make xconfig* (Bild 1) gewählt.

Eigentliche Härtung

Tätigen Sie bitte nun die gewohnten Einstellungen und rufen Sie danach den Optionspunkt *Security Options* auf (Bild 2). Dort sind alle verfügbaren und aktivierbaren Features zu sehen.
Nachdem alle Einstellungen getroffen sind, verläßt man mit *Save and Exit* die grafische Konfigurationsoberfläche und gibt folgende Befehle ein:

```
#make dep
#make bzImage
```

Jetzt hängt die weitere Vorgehensweise vom Bootloader ab. Im folgenden wird davon ausgegangen, daß dies LILO ist. Die Datei »bzImage« (der Kernel) wird in das Bootverzeichnis kopiert und die »/etc/lilo.conf« editiert:

```
#cp /usr/src/linux/arch/i386/boot/\
bzImage /boot/vmlinux-openwall-2-4-20
# vi /etc/lilo.conf

prompt
timeout=50
default=linux
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
restricted
password=**********
message=/boot/message
linear

image=/boot/vmlinuz-2.4.20
  label=linux
  initrd=/boot/initrd-2.4.20.img
  read-only
  append="hdd=ide-scsi root=LABEL=/"

image=/boot/vmlinuz-openwall-2-4-20
  label=linux-openwall
  read-only
  root=/dev/hda2
```

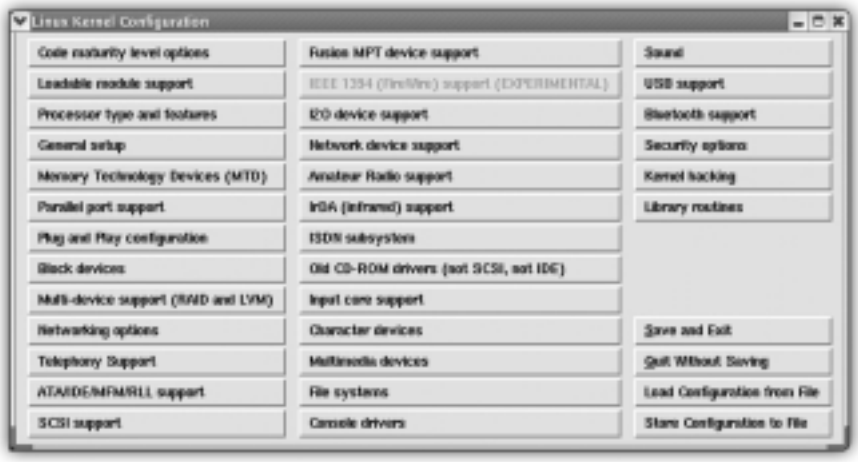


Bild 1: Die grafische Kernelkonfiguration

Nachdem die LILO-Konfigurations-Datei editiert ist, wird *lilo* gestartet und der Rechner mit dem neuen Kernel gebootet:

```
#/sbin/lilo
Added linux *
Added linux-openwall-2-4-20
#init 6
```

Jetzt kann man den neuen Kernel auswählen, booten und ist der bestmöglichen Sicherheit wieder ein Stückchen nähergekommen. Jetzt sollte man noch eine Logdatei erstellen, die aktuelle Angriffe anzeigt und die Möglichkeit bietet, den neuen Kernel zu testen. Dazu bedient man sich des Syslog-Servers.
Es wird dafür (falls noch nicht vorhanden) die Zeile

```
kern.alert          /var/log/kernalert
```

in die Konfigurationsdatei »/etc/

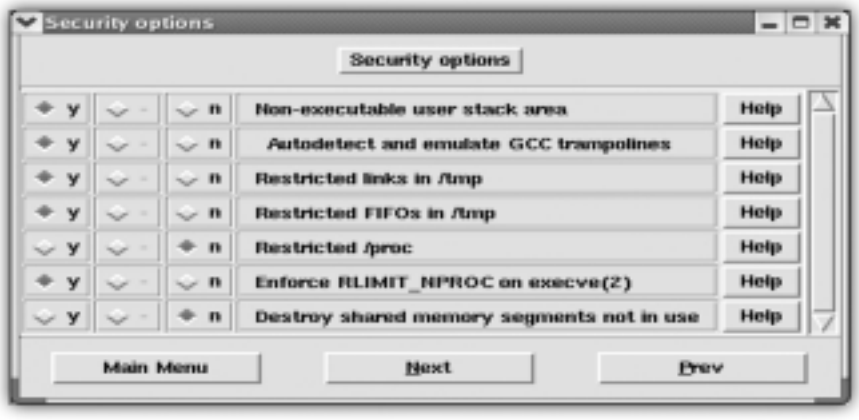


Bild 2: Die Sicherheitseinstellungen in der Kernelkonfiguration

Buffer Overflow

Ein Buffer Overflow entsteht dadurch, daß ein Programmierer übersehen hat, Pufferzuweisungen im Programmcode korrekt zu prüfen. Ein Angreifer, der weiß, daß ein bestimmtes Programm eine solche Lücke aufweist, kann ein Angriffsprogramm schreiben, das den Puffer zum Überlauf bringt. Damit hat er die Chance, die Rücksprungsadresse des Hauptprogramms (das ist die Stelle, an der die fehlerhafte Unterfunktion wieder in das Hauptprogramm springt) zu überschreiben und bösartigen Maschinencode auszuführen. Wer weitere Informationen zu diesem Thema findet man in [2]. Dort werden Buffer Overflows detailliert erläutert.

syslog.conf« des *syslog*-Servers eingetragen.
Nun legt man die Logdatei mit entsprechenden Rechten an:

```
#touch /var/log/kernalert
#chmod 640 /var/log/kernalert
```

Danach gilt es, den *syslog*-Server neu zu starten:

```
#/etc/init.d/syslog stop
#/etc/init.d/syslog start
```

Um den neuen Kernel testen zu können, kompiliert man das Zusatzprogramm *stacktest* aus dem Contrib-Verzeichnis des Patches. Das geht über folgende Befehle:

```
#cd /download/linux-2.4.20-ow0/optional
#gcc -o stacktest stacktest.c
```

Nachdem die Datei erstellt ist, sollte man sie in ein Verzeichnis im Suchpfad kopieren, um stets darauf zugreifen zu können:

```
#cp stacktest /usr/sbin
```

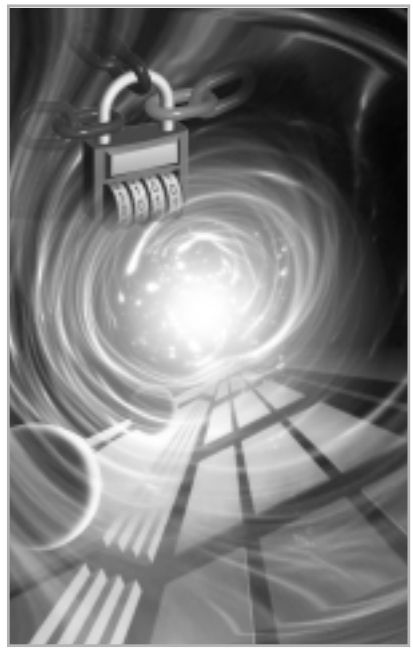
Jetzt kann man mit Hilfe des Programms *stacktest* testen, ob der Kernel tatsächlich sicherer ist. Als erstes erfolgt die Prüfung, ob die Emulation für GCC Trampolines vorhanden ist. Dafür wird das Testprogramm folgendermaßen aufgerufen:

```
# stacktest -t
Attempting to call a trampoline...
Succeeded.
```

Das hat also geklappt. Weitaus wichtiger ist aber, daß der Schutz vor Buffer Overflows gegeben ist. Der Test erfolgt mit folgendem Befehl:

```
#stacktest-e
Attempting to simulate a Buffer Overflow
 exploit...
Segmentation fault
```

Auch das scheint zu funktionieren. Vorsichtshalber sollte man noch ei-



nen Blick in die Logdatei werfen, um sicherzugehen, daß der Kernel den Angriff bemerkt und notiert hat:

```
#more /var/log/kernalert
Dec 23 11:52:05 freex kernel: Security:
return onto stack running as UID 0, EU
ID 0, process stacktest:1339
```

Auch das geht einwandfrei. Zu empfehlen ist aber, diese Prozedur mit einem wirklichen Exploit [2] durchzuführen, denn nur so hat man Sicherheit.

Ein weiteres Feature des Openwall-Patches ist, daß symbolische Links auf eine Datei, die von einem Benutzer, der normalerweise darauf keinen Zugriff hat, erstellt wurden, auch von priviligierten Usern nicht genutzt werden können. Hierzu ein Beispiel:

```
$ cd /dummyverzeichnis
$ ln -s /etc/shadow
$ su root
# more /dummyverzeichnis/shadow
more: /dummyverzeichnis/shadow: Keine
Berechtigung
```

Auch hier der Blick in die Logdatei:

```
#more /var/log/kernalert
Dec 23 11:56:12 freex kernel: Security:
not followed symlink of 500.500 by UID 0
, EUID 0, process bash:1374
```

Auch Hardlinks können nicht mehr erstellt werden:

```
$ln /etc/shadow
ln: Erzeugen der harten Verknüpfung
"./shadow" zu "/etc/shadow": Keine
Berechtigung
```

Dies ist die Ausgabe der Logdatei:

```
#more /var/log/kernalert
Dec 23 12:02:36 freex kernel: Security:
denied hard link to 0.0 for UID 500, E
UID 500, process ln:1500
```

Fazit

Der Openwall-Patch bietet viele nützliche Möglichkeiten, um das System zu schützen. In diesem Beitrag wurden nur die die Hauptfunktionen besprochen. Es gibt noch einige mehr, die der Readme-Datei zu entnehmen sind. Man sollte aber immer das Augenmerk darauf legen, daß keine Anwendungen beeinträchtigt werden. ◆

Literatur und Links

[1] Openwall-Patch: www.openwall. com/linux

[2] Josef Brunner: Security-Report, freeX 4'02, S. 42 ff.