Installation und Konfiguration von LIDS

Simon Adler - MI4457
Florian Stumpf - WI3964

19. Juni 2002

Inhaltsverzeichnis

1 Einführung **2**
- 1.1 Was ist LIDS? . . . 2
- 1.2 Was kann LIDS? . . . 2
- 1.3 Was kann LIDS nicht? . . . 2

2 Installation **4**
- 2.1 Der Kernel . . . 4
- 2.2 Der Userspace-Teil . . . 5
  - 2.2.1 Installation per make . . . 5
  - 2.2.2 Installation per Debian-Paket . . . 5

3 Konfiguration **7**
- 3.1 Grundsätzliches . . . 7
- 3.2 Fallen . . . 7
  - 3.2.1 READONLY . . . 7
  - 3.2.2 Geschützte Prozesse . . . 8
  - 3.2.3 Versteckte Prozesse . . . 8
- 3.3 Die Tools . . . 9
  - 3.3.1 Einschränkungen im Dateisystem . . . 9
  - 3.3.2 Prozesse und Capabilities . . . 10
  - 3.3.3 Vererbung von Capabilities . . . 10
  - 3.3.4 Verstecken und Schützen von Prozessen . . . 11
- 3.4 lids.conf updaten . . . 12
- 3.5 Übersicht anzeigen lassen . . . 12
- 3.6 Ändern des LIDS-Passwortes . . . 12

4 Administration **14**
- 4.1 Versiegeln des Kernels . . . 14
- 4.2 Aktivieren und Deaktivieren von Capabilities . . . 14
- 4.3 Neu laden der lids.conf . . . 15
- 4.4 Vollständiges Deaktivieren von LIDS . . . 15
- 4.5 Einrichten einer LFS . . . 15

5 Skripte **16**
- 5.1 Paketfilterskript . . . 16

6 chroot-User **18**
6.1 Theoretische Überlegungen 18
6.1.1 Linux 18
6.1.2 Linux mit LIDS 18
6.2 Ausführungen und Probleme 19
6.3 Warum es nicht sinnvoll ist 19
6.4 Wie es doch geht 20
6.4.1 Mehrfach gemountetes proc-Dateisystem 23

7 Fazit **24**

8 Quellen **25**

Kapitel 1

Einführung

1.1 Was ist LIDS?

LIDS[1] ist ein System zur Erhöhung der Sicherheit in einem Linux-System. LIDS besteht aus einem Kernel-Patch und zwei Userspace-Programmen, die zur Konfiguration und Administration des Systems dienen. LIDS implementiert die sogenannten Capabilities, die bereits in dem normalen Linux-Kernel verwendet werden[2]. Mit diesen Capabilities werden diverse Rechte zur Ausübung von Aufgaben an Dateien beziehungsweise Prozesse vergeben.
In einem normalen System hat man, sobald man eine „Root-Shell“ geöffnet hat, volle Rechte über sämtliche Prozesse und Devices des Systems. Genau hier setzt LIDS an.

1.2 Was kann LIDS?

- Dateien und Verzeichnisse schützen (Zugriff verweigern/read-only)
- Prozesse verstecken
- Prozesse vor sämtlichen Signalen schützen
- Direkten Zugriff auf Geräte verhindern
- Port-Scans bereits vom Kernel entdecken und melden lassen
- Binden an Ports unterhalb 1024 verbieten
- Durch diverse „Capabilities“ die Sicherheit weitestmöglich erhöhen
- Loggen der Sicherheitsverletzungen und eventuell versenden einer Mail

1.3 Was kann LIDS nicht?

1. „Jails“ im engeren Sinne

[1] Linux Intrusion Detection System
[2] siehe /usr/src/linux/include/linux/capabilities.h

2. Erweiterte Gruppeneinteilung
3. Programme, die nicht durch LIDS READONLY geschüzt sind, Capabilities zuweisen[3]
4. Sicherheitseinstellungen Benutzerunabhängig durchführen

[3]durch kleine Veränderung doch möglich

Kapitel 2

Installation

Da die Veränderungen, die LIDS am Sicherheitssystem von Linux vornimmt, sehr tiefgreifend sind, muß man auch einen neuen Kernel kompilieren.
Trotzdem ist die Installation von LIDS im Prinzip relativ einfach:
Man benötigt einen Kernel-Source, möglichst ohne Patches und die dazugehörigen LIDS-Sourcen[1]. Wobei LIDS relativ unempfindlich ist, wenn man z.B. die Debian-Version eines Kernels verwendet.

2.1 Der Kernel

Kurze Vorgehensweise für alle, die noch nie einen Kernel kompiliert und installiert haben:

- Man entpackt die Kernel-Sourcen:
  `cd /usr/src && tar xvzf linux-2.4.16.tar.gz`
- Den LIDS-Patch in die Sourcen einbringen:
  `patch -p1 < ../lids-1.1.1-2.4.16/lids-1.1.1-2.4.16.patch`
- Konfiguration und Installation werden mit folgenden Befehlen abgearbeitet:
  ```
  make menuconfig
  make dep clean bzImage modules modules_install && cp
  arch/i386/boot/bzImage /boot/vmlinuz-2.4.16_lids
  ```
- Danach muß man die Bootloader-Konfiguration anpassen:
  `vi /etc/lilo.conf` und z.B. folgendes eintragen:

  ```
  image=/boot/vmlinuz-2.4.16_lids
        label=Linux_lids
        read-only
  ```

- Lilo aufrufen:

[1] Wir haben einen 2.4.16-Kernel, gepatched von den Debian-Entwicklern und LIDS-1.1.1 verwendet

```
soLIDState:~# lilo
Added Linux
Added Linux_lids *
soLIDState:~#
```

- **Vor** dem Neustart die Konfiguration durchführen, da sonst die Änderungen nicht übernommen werden, was zu massiven Problemen führen kann.

2.2 Der Userspace-Teil

Den Userspace-Teil von LIDS kann man, zumindest unter Debian, auf zwei verschiedene Arten installieren:

2.2.1 Installation per make

Dieser Teil ist der einfachere, aber nicht ganz saubere. Man wecheslt in das Verzeichnis, in dem man die LIDS-Sourcen entpackt hat.

```
soLIDState:~# cd /usr/src/LIDS/lids-1.1.1-2.4.16/
soLIDState:/usr/src/LIDS/lids-1.1.1-2.4.16# ./configure && make && make install
[...]
make[1]: Leaving directory `/usr/src/LIDS/lids-1.1.1-2.4.16'
soLIDState:/usr/src/LIDS/lids-1.1.1-2.4.16#
```

Daraufhin werden die LIDS-Tools konfiguriert, übersetzt und installiert. Da bei einigen Programmen aber kein Ziel „uninstall“ in dem Makefile vorhanden ist, ist es sehr zeitaufwendig, das gesamte Programm und eventuell vorhandene zusätzliche Dateien zu deinstallieren.
Die eindeutig bessere Art ist die Installation per Debian-Paket, beschrieben in dem nachfolgenden Abschnitt:

2.2.2 Installation per Debian-Paket

Dieser Ansatz ist ein wenig komplizierter als der vorher beschriebene, lohnt sich aber insofern, da man sämtliche zu LIDS gehörenden Dateien in einem Paket hat. Man kann sie also ohne großen Aufwand deinstallieren und auch auf andere Rechner installieren, die allerdings die gleiche Ausstattung haben müssen.
Die Installation läuft wie folgt:

```
soLIDState:/usr/src/LIDS/lids-1.1.1-2.4.16# dh_make

Type of package: single binary, multiple binary, or library? [s/m/l] s

Maintainer name : root
Email-Address   : root@soLIDState
Date            : Fri, 17 May 2002 14:58:06 +0200
Package Name    : lids-1.1.1
Version         : 2.4.16
Type of Package : Single
Hit <enter> to confirm:
```

```
Done. Please edit the files in the debian/ subdirectory now. lids-1.1.1
uses a configure script, so you probably don't have to edit the Makefiles.
soLIDState:/usr/src/LIDS/lids-1.1.1-2.4.16#
```

Mit diesem Befehl wurde ein Verzeichnis "`debian`" angelegt, in dem sich einige mehr oder weniger wichtige Konfigurationsdateien befinden. Unbedingt benötigt werden die Dateien changelog, control und rules. Die restlichen Dateien kann man löschen, sofern man nicht auf die von ihnen angebotene zusätzliche Funktionalität – wie z.B. welche Befehle nach dem Installieren ausgeführt werden sollen etc. – angewiesen ist.
Die Datei debian/rules muß man allerdings noch ein wenig modifizieren. Hier muß man die benötigten Parameter an das configure-Skript übergeben. Das gleiche muß man auch noch einmal für die Installation machen, da sonst die Dateien in den normalen Verzeichnisbaum einkopiert werden. Wenn man die nötigen Veränderungen gemacht hat, sollte man möglichst als normaler User das eigendliche Paket kompilieren, da man hierbei ggf. auftretende Fehler besser sieht.

```
user@soLIDState:/usr/src/LIDS/lids-1.1.1-2.4.16$ fakeroot dpkg-buildpackage
dpkg-buildpackage: source package is lids-1.1.1
dpkg-buildpackage: source version is 2.4.16-1
[...]
dpkg-deb: building package 'lids-1.1.1' in '../lids-1.1.1_2.4.16-1_i386.deb'.
 dpkg-genchanges
dpkg-genchanges: including full source code in upload
dpkg-buildpackage: full upload; Debian-native package (full source is included)
user@soLIDState:/usr/src/LIDS/lids-1.1.1-2.4.16$
```

Damit hat man alles in einem Schritt gemacht - die Konfiguration, Kompilierung und die Erstellung des Paketes. Das Paket liegt in der Verzeichnishierarchie nun eine Ebene höher. Zusätzlich zu dem eigentlichen Paket befinden sich dort einige Dateien mehr - eine Datei mit der Beschreibung des Paketes, eine mit den Quellen des Paketes, und eine mit den Änderungen von den ursprünglichen Quellen zu den Debian-Quellen – in diesem Fall lediglich mit dem zusätzlichen Verzeichnis.
Um nun mit LIDS arbeiten zu können, muß man das Paket noch installieren:

```
soLIDState:/usr/src/LIDS# dpkg -i lids-1.1.1_2.4.16-1_i386.deb
Selecting previously deselected package lids-1.1.1.
(Reading database ... 57948 files and directories currently installed.)
Unpacking lids-1.1.1 (from lids-1.1.1_2.4.16-1_i386.deb) ...
Setting up lids-1.1.1 (2.4.16-1) ...

soLIDState:/usr/src/LIDS#
```

Kapitel 3

Konfiguration

3.1 Grundsätzliches

Das Grundprinzip ist eigentlich ähnlich einer Firewall: Mal verbietet alles und erlaubt einigen wenigen Programmen einiges. Spezifisch für LIDS sieht das dann so aus, daß man alle Capabilities ausschaltet und für einige Programme – eventuell auch mit Vererbungs-Level – freigibt.

3.2 Fallen

3.2.1 READONLY

Da man /etc in LIDS read-only eintragen sollte und einige Prozesse trotzdem in dieses Verzeichnis schreiben wollen, muß man bei einigen Dateien ein paar kleine Tricks anwenden. Dazu verschiebt man die Dateien in ein beliebiges nicht schreibgeschütztes Verzeichnis[1] und verlinkt einfach diese Dateien nach /etc.
Eine Sonderrolle nimmt dabei die Datei /etc/mtab ein. In dieser Datei werden alle Dateisysteme eingetragen, sobald sie gemountet werden. Das wird aber auch parallel in den Kernelvariablen verarbeitet, man kann also einfach die Eintragung vereinheitlichen. Dazu verlinkt man die mtab auf /proc/mounts. Das Einzige, was man jetzt noch ändern muß, befindet sich in /etc/init.d: in allen Dateien, in denen mount aufgerufen wird, muß man mount durch die Option -n ergänzen. Dadurch werden Dateisysteme, sobald sie gemountet werden, nicht mehr in die /etc/mtab eingetragen, was jetzt von dem Kernel übernommen wird.
Weitere Dateien, die man verschieben und verlinken muß:
/etc/adjtime, /etc/ioctl.save und /etc/mtab /etc/network/ifstate

Weiterhin muß man bei /lib aufpassen. Hier werden unter anderem die Module für die Kernel gespeichert. Bei jedem Start werden allerdings die Anhängigkeiten zwischen den Modulen neu berechnet. Diese Abhängigkeiten werden in der Datei `modules.dep` abgelegt. Einige andere Dateien beinhalten noch spezielle Abhängigkeiten, wie z.B. für Plug'n'Play der verschiedenen Busse. Man muß

[1] Wir haben hierfür /var/run benutzt

also dem Programm, das diese Abhängigkeiten berechnet, Schreibzugriff auf dieses Verzeichnis gewähren. Das Programm heißt depmod und befindet sich in dem Verzeichnis /sbin.

3.2.2 Geschützte Prozesse

LIDS bietet die Möglichkeit, Prozesse vor sämtlichen Signalen zu schützen (`CAP_PROTECTED`). Dabei ist zu erwähnen, daß es wirklich **alle** Signale sind, nicht nur die, die den Prozess auf mehr oder weniger freundliche Art beenden würden. Das eine Problem kann man sich dann schon fast denken: die Prozesskommunikation über Signale wird dadurch auch verhindert.
Es gibt aber eine Möglichkeit, bestimmten Prozessen zu erlauben, die geschützten Prozesse zu beenden oder, allgemeiner gesagt: ihnen Signale zu senden (`CAP_KILL_PROTECTED`). Dabei ist das aber nur möglich, wenn der Prozess, der die Signale sendet, unter der gleichen User-ID läuft, wie der Prozess, der die Signale empfangen soll. Sonst werden diese Signale abgefangen, als ob der sendende Prozess nicht das Recht dazu hätte. Dazu ist ein sehr schönes Beispiel der apache. Der eigentliche Elternprozess läuft mit der UID 0, also als *root*. Seine Kindprozesse hingegen laufen als User *www-data*. Damit ist es also schon klar: Wenn der apache von LIDS geschützt wird, kann er seine eigenen Kind-Prozesse nicht stoppen und behindert damit den Shutdown-Vorgang. Eine Möglichkeit, diesen Prozessen doch noch zu erlauben, ihre Kind-Prozesse zu stoppen, ist, ihnen die Capability `CAP_KILL` zu geben. Damit haben sie auch die Erlaubnis, Prozesse zu beenden, die nicht mit ihrer eigenen User-ID laufen. Damit hat man aber wieder eine relativ große Sicherheitslücke geschaffen. Die Vergabe dieser Capability ist für geschützte Prozesse nur mit der gleichzeitigen Vergaben von `CAP_KILL_PROTECTED` sinnvoll, da diese Möglichkeit nicht in der `CAP_KILL` eingeschlossen ist.

3.2.3 Versteckte Prozesse

LIDS bietet auch die Möglichkeit, Prozesse aus dem normalen proc-System zu verstecken. Dadurch sind sie durch die Befehle `top` oder `ps` nicht zu sehen. Lediglich die Tatsache, daß die meisten Prozesse ihre PID's unter /var/run ablegen, läßt erkennen, daß ein solcher Prozess läuft. Das Problem, was hierbei auftaucht, ist, daß der start-stop-daemon diese Prozesse nicht erkennt. Er versucht, indem er in den Kernelvariablen nachsieht, die ausführbare Datei zu dem Prozess zu finden. Dabei verwendet er natürlich die PID, die er unter /var/run findet. Da aber diese PID in den Kernelvariablen nicht vorhanden ist, behauptet der start-stop-daemon, dieser Prozess wäre nicht vorhanden. Das ist besonders unangenehm, wenn ein versteckter Prozess auf eine noch gemountete Partition zugreift. Ein solcher Prozess ist zum Beispiel der klogd. Er greift auf die /boot Partition zu. Es ist allerdings auch sinnvoll, diesen Prozess zu verstecken, da er dafür sorgt, die Kernel-Meldungen zu speichern.
Als Lösung für dieses Problem haben wir das shutdown-Skript /etc/init.d/umountfs so verändert, daß an alle Prozesse, die ihre PID unter /var/run in eine entsprechende Datei gespeichert haben, erst ein `kill -15`, dann ein `kill -9` gesendet wird.

3.3 Die Tools

Unter Lids gibt es zwei Tools mit denen sich LIDS konfigurieren läßt. Das eine Programm ist **lidsadm** und das Andere ist **lidsconf**. Dieser Abschnitt beschäftigt sich mit lidsconf, was für die Konfiguration von LIDS gedacht ist und stellt seine unterschiedlichen Optionen vor. Hier wird erklärt, was die einzelnen Optionen bewirken und welche Probleme sich daraus ergeben können, bzw. warum es wichtig ist, diese Schritte zu gehen. Mit lidsconf kann man:

- Regeln hinzufügen (Option -A)
- Regeln aus dem Regelsatz entfernen (Option -D)
- Den gesamten Regelsatz löschen (Option -Z)
- Die lids.conf updaten (Option -U)
- Sich eine Regelübersicht anzeigen lassen (Option -L)
- Das LIDS-Passwort ändern (Option -P)

Die einzelnen Optionen werden im weiteren zusammen mit der Anwendung von dem Tool lids.conf erklärt.

3.3.1 Einschränkungen im Dateisystem

Der Aufbau der Optionen von lidsconf ähnelt auf den ersten Blick denen von diversen Paketfilter-Administrations-Tools.
Man kann mit der Angabe der Option -A Regeln zu dem Regelsatz hinzufügen und mit der Option -D diese Regeln auch wieder löschen.
lidsconf -[A|D] [-s source] -o object -j target

Für die Rechtevergabe für das Dateisystem kann man als Objekt jedes Verzeichnis angeben, oder jede beliebige Datei. Als Ziel kommen dann DENY, APPEND, READONLY oder WRITE in Frage.
Mit DENY kann man den Zugriff auf eine Datei oder ein Verzeichnis verbieten.
Mit APPEND ermöglicht man das Schreiben in Dateien, aber nicht das Löschen von Inhalten, was einem eine Freigabe von LOG-Dateien so erlaubt, das sie von einem Angreifer nicht manipuliert werden können. Das Prinzip ähnelt dem von *BSD Betriebssystemen.
Mit READONLY kann man verbieten, auf die angegebenen Dateien oder Verzeichnisse schreibend zuzugreifen.
Bis auf die Append-Funktion scheint diese Funktionalität auch durch das Dateisystem von Unix gegeben zu sein, aber hier bietet LIDS eine weitere Absicherung dagegen, das dieses System mutwillig umgangen wird, was in dem normalen Linuxsystem ja auch durch den Superuser möglich ist
Zu diesen Regeln kann man dann weitere Regeln definieren mit denen man den Zugriff eines Prozesses auf eine Datei erlaubt. Als Beispiel sei hier einmal /bin/login genannt. Angenommen man hat die Datei /etc/shadow per `lidsconf -o /etc/shadow -j DENY` auf DENY gesetzt, so wird es unmöglich, sich einzuloggen, da /bin/login hierzu auf /etc/shadow lesend zugreifen muß.
Für diesen Fall kann man für login eine Außnahme definieren, so daß man

vollständig sagen kann:

```
soLIDState:~# lidsconf -A -o /etc/shadow -j DENY
ADD
soLIDState:~# lidsconf -A -s /bin/login -o /etc/shadow -j READONLY
ADD
soLIDState:~#
```

Hiermit wäre der Login in das System möglich, obwohl die /etc/shadow sonst nicht einmal für den Superuser lesbar ist.

3.3.2 Prozesse und Capabilities

Um Prozesse zu beeinflussen, muß erst definiert werden, welche Eigenschaften, bzw. Capabilities als global aktiv oder inaktiv gelten sollen. Hierzu muß man die Datei /etc/lids/lids.cap editieren. Hier sind alle Capabilities und ihre Eigenschaften aufgelistet. Vor jeder Capability steht führend ein + bzw. ein -. Wenn for der Capability ein Plus steht, dann kann diese Eigenschaft global ohne Sondergenehmigung genutzt werden. Bei einem Minus ist diese Option nicht nutzbar.
Capabilities gibt es 31, wovon 29 als Beschränkungen gelten und zwei als zusätzliche Eigenschaft. Die Capabilities sind in der Datei /etc/lids/lids.cap definiert und kurz erklärt, daher werden wir sie hier nicht noch einmal aufführen.
Nun wird man sicherlich bestrebt sein, alle Capabilities zu deaktivieren, um eine möglichst hohe Sicherheit zu erzielen. Auch an dieser Stelle wird es wieder notwendig, Ausnahmeregeln zu definieren. So zum Beispiel, wenn man allgemein die `CAP_NET_BIND_SERVICE` deaktiviert hat. Hier wird es dann grundsätzlich verboten, daß Prozesse sich an Ports ≤ 1023 binden dürfen. In der Regel wird man jedoch einen MTA (Mail Transport Agent) nutzen, der sich an Port 25 binden soll, um den Emailtransfer zu regeln. Als Beispiel sei hier der exim angenommen. Um dem exim das zu erlauben gibt es für lidsconf das Ziel GRANT.
Beispiel:
In der lids.cap steht die Zeile:

```
-:CAP_NET_BIND_SERVICE
```

Ausnahmeregel für exim: `lidsconf -A -s /usr/sbin/exim -o CAP_NET_BIND_SERVICE 25 -j GRANT`
Hiermit wird also dem Prozess der aus dem Programm /usr/sbin/exim hervor geht erlaubt Aktionen durchzuführen, die die Capability `CAP_NET_BIND_SERVICE` benötigen.

3.3.3 Vererbung von Capabilities

Nachdem nun die Möglichkeit bekannt ist, wie man Rechte für das Dateisystem freigibt und wie man Prozesse beschränkt wird schnell klar, daß man mit LIDS dem Linux-System eine Art Korsett anzieht. Dieses Korsett muß jedoch sehr eng definiert werden. Diese Definitionen können aber mühsam werden, wenn man z.B. einmal einen Prozess wie den crond betrachtet, der diverse andere Programme ausführt. Diese Programme müssen natürlich alle die

entsprechenden Capabilities haben, damit sie erfolgreich arbeiten können, wenn sie von cron aufgerufen werden. Das wird sehr mühsam, wenn man betrachtet, daß diese Programme auch Skripte sein können, und das diese vielleicht wieder andere Programme oder Skripte aufrufen. Außerdem möchte man vielleicht manchen Programmen nicht einmal die vollen Rechte geben, sondern eigendlich nur für die Ausführung von cron.
Hierfür kann man bei lidsconf die Option **-i** mitgeben. Mit ihr kann man einen Vererbungslevel vergeben.

Beispiel

```
soLIDState:~# lidsconf -A -s /usr/sbin/cron -o CAP_SYS_ADMIN -i 2 -j GRANT
ADD
soLIDState:~#
```

Was bedeutet das jetzt genau? Wie bisher geben wir einer Datei (hier: /usr/sbin/cron) das Recht auf eine Capability (hier: `CAP_SYS_ADMIN`). Das besondere ist jedoch das CRON diese Capability an die von ihm aufgerufenen Prozesse vererben kann. Diese können dann diese Eigenschaft wieder vererben. Bei jeder Stufe der Vererbung wird der Wert, der mit -i angegeben wurde um eins reduziert. Ist der Wert 0, so wird keine Vererbung mehr zugelassen. Wenn man die Weitergabe auf Dauer ermöglichen will, kann man das mit dem Vererbungswert -1.

```
soLIDState:~# lidsconf -A -s /usr/sbin/cron -o CAP_SYS_ADMIN -i -1 -j GRANT
ADD
soLIDState:~#
```

3.3.4 Verstecken und Schützen von Prozessen

LIDS bietet noch ein paar Sondermöglichkeiten. Es gibt außer den Capabilities, die verwendet werden, um Rechte anzupassen, auch solche, die für die angegebenen Prozesse Zusatzeigenschaften darstellen. Diese Capabilities sind `CAP_HIDDEN` und `CAP_PROTECTED`. Sie werden ähnlich wie die Außnahmeregeln für Capabilities gesetzt, also:

```
lidsconf -A -s /usr/sbin/apache -o CAP_HIDDEN -j GRANT
```

bzw.

```
lidsconf -A -s /usr/sbin/apache -o CAP_PROTECTED -j GRANT
```

`CAP_HIDDEN` versteckt Prozesse. Hier wird also der Prozess, der von /usr/sbin/apache gestartet wird weder in top noch in ps oder im Verzeichnis /proc zu sehen sein. Leider kann man die PIDs noch feststellen, wenn man in das Verzeichnis /var/run sieht. Hier findet man Dateien mit allen PIDs der laufenden Prozesse und leider auch die der Versteckten. So z.B. die Datei apache.pid, in der man die PID des eigentlich versteckten apache Prozesses findet. Dieser Prozess kann jetzt wie gewohnt mit dem Befehl kill beendet werden.
Hier greift die zweite Capability. `CAP_PROTECTED` verhindert, daß an einen Prozess Signale gesendet werden können. Damit kann er nicht mehr ohne weiteres

beendet werden. Leider werden alle Signale unterbunden, so daß die Kommunikation zwischen Prozessen über Signale unmöglich wird.

Um den Rechner trotz dieses Schutzes beenden zu können, muß man dem entsprechenden Programm das Beenden dieser Prozesse erlauben, damit sie ordnungsgemäß beendet werden können. Hierzu muß man den Programmen die Capability `CAP_KILL_PROTECTED`, und, wenn sich die UID's der Prozesse unterscheiden, noch `CAP_KILL` zuweisen.

3.4 lids.conf updaten

Alle Regeln, die mit dem Tool definiert werden, werden in der Datei /etc/lids/lids.conf festgehalten. Zusammen mit den Dateinamen werden hier die I-Nodes der Dateien notiert, die von LIDS spezielle Rechte erhalten.
Aufgrund von Änderungen am Dateisystem kann es jedoch dazu kommen, daß sich die I-Node einer Datei ändert. Anstatt jetzt den vollen Regelsatz neu definieren zu müssen, kann man die Option -U von lidsconf nutzen. Sie aktualisiert den I-Node zu jedem Dateinamen.

3.5 Übersicht anzeigen lassen

Mit der Option -L kann man sich bei LIDS den gesamten Regelsatz anzeigen lassen der definiert ist. Hierbei werden die Regeln anhand einer Tabelle angezeigt.

Beispiel

lidsconf -L

Subject	ACCESS(inherited)	time	Object
/usr/bin/ssh	READONLY(domain):0	0000-0000	/etc/shadow
/bin/login	GRANT(domain):0	0000-0000	CAP_FOWNER
Any file	READONLY(domain):0	0000-0000	/usr/bin

3.6 Ändern des LIDS-Passwortes

Mit der Option -P kann man unter LIDS das Administrator-Passwort von LIDS ändern. Eine große Schwachstelle ist dabei, daß man nicht zuerst aufgefordert wird, das alte Passwort einzugeben. Damit ist es möglich, ohne Kenntnis des alten Passwortes das Passwort zu ändern. Sofern /etc durch LIDS READONLY geschützt ist, muß man natürlich LIDS lokal oder gar global deaktiviert haben, damit das Passwort in die lids.pw geschrieben werden kann.
Nachdem das neue Passwort eingegeben wurde, wird es noch keine Gültigkeit haben, sondern erst, wenn man einmal die Konfigurationsdateien neu lädt.

```
soLIDState:~# lidsconf -P
```

```
[..]
soLIDState:~# lidsadm -S -- +RELOAD_CONF
```

Kapitel 4

Administration

LIDS gibt neben dem Programm lidsconf dem Administrator noch ein weiteres Programm an die Hand, das während der Laufzeit dem Administrator die Möglichkeit gibt in LIDS einzugreifen. Die Möglichkeiten sind:

- Versiegeln des Kernels
- Aktivieren und Deaktivieren einzelner Capabilities
- Neu laden der lids.conf
- Vollständiges deaktivieren von LIDS
- Einrichten einer LFS[1]

4.1 Versiegeln des Kernels

Nachdem der Rechner gestartet wurde, müssen irgendwann die Regeln aus der lids.conf geladen werden. Dazu gibt es von dem Programm lidsadm die Option -I. lidsadm lädt dann alle Regeln und merkt sich, daß die Regeln geladen wurden. Dieser Vorgang nennt sich „Versiegeln des Kernels“, da man ohne Passwort die Regeln nur ein einziges Mal laden kann. Man kann das Laden der Regeln in ein Sart-Stop-Skript schreiben, das man dann so spät wie möglich ausführt, da dann noch Dienste starten können, die einige der Sonderrechte von LIDS benötigen. Ohne das Versiegeln wären nur die Einschränkungen, nich aber die Ausnahmen aktiv. Man kann sich also nicht mehr einloggen, wenn man die Datei /etc/shadow auf DENY gesetzt hat.

4.2 Aktivieren und Deaktivieren von Capabilities

Capabilities können von dem Administrator jederzeit deaktiviert oder aktiviert werden, was dann jedoch global gültig ist. Hierbei wird die Option -S genuzt. Sie

[1] LFS: LIDS-Free-Session

wird als Schalter beschrieben der Funktionen von LIDS umschaltet, nachdem das LIDS Passwort angegeben wurde.

`lidsadm -S -- -CAP_SOMETHING`

4.3 Neu laden der lids.conf

Wenn man im Laufe des Betriebes Regeln zur Konfiguration hinzufügt, oder andere Regeln entfernt, dann ist es nötig die lids.conf neu zu laden. Hier bietet LIDS die Möglichkeit, einen Reboot zu umgehen. Man benötigt dafür den Befehl:
`lidsadm -S -- +RELOAD_CONF`
Zwar sieht dieser Befehl aus, als würde man eine Capability aktivieren, jedoch ist diese scheinbare Capability extra angelegt, um ein erneutes Laden der Konfigurationsdatei ohne Neustart des Rechners zu ermöglichen.

4.4 Vollständiges Deaktivieren von LIDS

Dem Administrator sind durchaus auch Möglichkeiten gegeben, LIDS zu deaktivieren. Hierbei kann er entweder LIDS für eine Konsole deaktivieren, was im nächsten Abschnitt näher beleuchtet wird, oder er kann LIDS vollkommen deaktivieren. Der Aufruf von lidsconf sieht ebenfalls aus, als würde man eine Capability ändern. Um LIDS vollkommen zu deaktivieren muß man folgenden Befehl nutzen:

`soLIDState:~# lidsadm -S -- -LIDS_GLOBAL` zum deaktivieren bzw.
`soLIDState:~# lidsadm -S -- +LIDS_GLOBAL` um es wieder zu aktivieren.

4.5 Einrichten einer LFS

Falls man sich in der Administration in Bereichen bewegen muß, die von LIDS geschützt werden, dann wäre es schlecht, wenn man dafür den Schutz des gesamten Systems aufheben müßte. Hierfür kann man auf einer Konsole den Schutz von LIDS deaktivieren. Auf allen anderen Konsolen bleibt LIDS weiterhin aktiv. Diese Konsole läuft dann unter einer so genannten **L**IDS **F**ree **S**ession. Wichtig ist, daß immer nur eine Konsole zur Zeit als LFS eingerichtet werden kann. Dies geschieht analog zur globalen Deaktivierung von LIDS über den Befehl:

`lidsadm -S -- -LIDS` zum deaktivieren von LIDS auf der Konsole, bzw.
`lidsadm -S -- +LIDS` um die LFS wieder zu beenden und um LIDS wieder auf der Konsole zu aktivieren.

Kapitel 5

Skripte

5.1 Paketfilterskript

Bei jedem System, bei dem ein hohes Sicherheitsniveau angestrebt wird, kann man erwarten, daß der Netzverkehr gefiltert wird.
Wir haben iptables verwendet, um einen Paketfilter aufzubauen. Hierbei bot sich iptables wegen seiner Fähigkeit, Verbindungen nach ihrem Status zu behandeln (Statefull-Filtering) an. Um den Paketfilter zu konfigurieren, sind Variablen in dem Paketfilter-Skript deklariert, die die Konfiguration ermöglichen. Das Skript selber befindet sich in `/etc/init.d/firewall`.
Die Variablen lassen sich in Netze und Dienste einteilen.

Netze:
ME: Die IP des Rechners, auf dem der Paketfilter installiert ist.
NETZE: Netze, mit denen der Rechner in Kontakt ist.
TRUSTED: Gruppe von Rechnern, denen spezielle Dienste angeboten werden
PROXIES: Proxies, die von dem Rechner aus genutzt werden
DNS_SERVER: Der Server, der den Namensdienst anbietet

Dienste:
OFFERED_SERVICES: Dienste, die der Rechner allen Rechnern anbietet.
TO_SERVICES: Dienste, die von dem Rechner aus genutzt werden können
PROXY_SERVICES: Dienste, die von den PROXIES angeboten werden
ICMP_IN und ICMP_OUT: Ein- und Ausgehende ICMP-Typen, die zugelassen werden
FROM_TRUSTED_SERVICES: Dienste, die nur den TRUSTED angeboten werden

Der Paketfilter ist in Funktionen aufgeteilt, so daß jeder einzelne Teil des Paketfilters für sich ausgeführt werden kann. Antwortpakete auf Dienste werden in der Regel durch die Eigenschaft des Statefull-Filtering erlaubt, so daß nur Pakete angenommen werden, die aufgrund einer ausgehenden Verbindung des Hosts als Antwort ankommen.
Als Parameter zu dem Firewallskript sind erlaubt:
help: Zeigt die möglichen Parameter an

restart, reload, start: Löscht alle alten Regeln und lädt die Regeln des Skriptes neu.

stop: Löscht alle alten Regeln und setzt alle Regelketten auf DROP

clear: Löscht alle alten Regeln und setzt alle Regelketten auf ACCEPT.

Der Parameter `clear` ist nur für den Notfall gedacht und sollte aus Sicherheitsgründen auskommentiert werden.

Für das Loopback-Device gilt, daß alle Pakete erlaubt sind, die auf diesem Interface eintreffen und von dort kommen, beziehungweise alle Pakete, die von diesem Interface ausgehen und dieses Interface wieder als Ziel haben. Es wird also Spoofing auf diesem Gerät verhindert.

Kapitel 6

chroot-User

6.1 Theoretische Überlegungen

6.1.1 Linux

Wenn man nun schon mal ein so sicheres System hat, kann man ja auch diesen Rechner als Arbeitssystem für diverse Leute nehmen, denen man sogar root-Rechte geben kann, ohne daß sie damit viel anfangen können. Dazu muß man in das neue Root-Verzeichnis eigentlich das gesamte „normale“ Root-Verzeichnis kopieren[1]. /etc, /lib und /boot muß man nicht vollständig oder gar nicht kopieren, da der Rechner ja in dem neuen Root-Verzeichnis nicht booten oder gar Module laden muß.
Problematisch wird es bei /proc. Man kann ja nicht einfach die Kernelvariablen von einem bestimmten Zeitpunkt in ein Verzeichnis kopieren, da in diesem Fall nicht die aktuellen Kernelvariablen angezeigt werden, sondern die eines vorherigen Zeitpunktes. Man muß also das Proc-Dateisystem jeweils in die Home-Verzeichnisse der User mounten.

6.1.2 Linux mit LIDS

Wenn man Programmen Zugriff auf Capabilities geben möchte, müssen diese Programme von LIDS Read-Only geschützt sein. Das heißt, man muß einen ähnlichen Regelsatz wie den für das Grundsystem nochmal für jeden Nutzer einrichten. Damit vergibt man aber die Möglichkeit, daß der Nutzer seine Root-Rechte ausnutzen kann, indem er zum Beispiel neue Programme installiert oder nicht benötigte Programme entfernt. Um volles Root-Recht in seinem Bereich nutzen zu können, müßte man ihm also auch das Passwort für LIDS geben. Damit hätte er aber auch Zugriff auf das gesamte System, diese Möglichkeit fällt also aus Sicherheitsgründen aus.

[1] Verlinken geht nicht, da chroot nicht mehr auf Bereiche außerhalb des neuen Root-Vezeichnisses zugreifen kann

6.2 Ausführungen und Probleme

Als erstes benötigt man für die Personen, die den Rechner als Arbeitssystem nutzen sollen (im folgenden Chroot-User genannt), eine eigene Login-Shell. Diese muß für den User die Chroot-Umgebung aufbauen und ihm eine echte Shell zur Verfügung stellen. Dazu haben wir ein kleines Shell-Skript geschrieben:

```
#!/bin/sh
# /usr/local/bin/local_login
#
# This is going to be a new login-shell for almost all users
#   except the admin

CHROOT=/usr/sbin/chroot

$CHROOT $HOME /usr/local/bin/login/saba
```

Dieses Skript wird als Login-Shell in die /etc/shells eingetragen, um dem System mitzuteilen, daß es sich um eine gültige Login-Shell handelt. Weiterhin muß man diese Shell auch als Standard-Shell für die Chroot-User festlegen. Entweder macht man das für alle neu anzulegenden Benutzer[2], oder man ändert die Shell des Users direkt[3]. Da man ja vielleicht auch noch normale Benutzer anlegen möchte, haben wir uns für die zweite Lösung entschieden, die allerdings durch ein Skript, welches den adduser-Befehl erweitert, wesentlich vereinfacht wird.

In diesem Skript wird `chroot` mit dem eigentlichen Home-Verzeichnis des Users als neuem Root-Verzeichnis und `/usr/local/bin/local_login` als Befehl ausgeführt. Der Befehl, der ausgeführt wird, ist wiederum ein Shell-Skript, was als Aufgabe hat, eine gewisse Zeit nichts zu tun, so daß sich die danach ausgeführte Shell, in diesem Falle die bash, nicht mehr auf die ursprüngliche Root-Umgebung zugreift. Das Skript sieht wie folgt aus:

```
#!/bin/bash
# /usr/local/bin/login/saba
# Sleep And BAsh

/bin/sleep 1
/bin/bash --login
```

Weiterhin muß man an der etc/passwd noch einige Veränderungen vornehmen. Will der Chroot-User – warum auch immer – ein "`su - $USER`" machen, so darf in der passwd ja nicht mehr die vorherige Login-Shell stehen. Desweiteren brauchen in der lokalen passwd auch die übrigen User des Systems nicht mehr enthalten sein.

6.3 Warum es nicht sinnvoll ist

- User kann mögliche Root-Rechte nicht nutzen

[2] in der /etc/adduser.conf die Variable DSHELL auf den entsprechenden Wert setzen
[3] chsh Username

- Regelsatz muß für jeden zusätzlichen User erneut hinzugefügt werden
- Jeder User braucht für sich fast den Plattenplatz einer kompletten Distribution
- Kein Sicherheitsgewinn durch chroot

6.4 Wie es doch geht

Da LIDS auch irgendwo überprüfen muß, ob die Programme, denen man zusätzliche Rechte geben will, auch geschützt sind, kann man ja in den Quellcode gucken und diese Überprüfung etwas abändern. Man sollte allerdings die Abbruchmeldungen zumindest durch eine Warnung ersetzen und die Meldung nicht völlig wegfallen lassen. Das haben wir allerdings nur in dem Überprüfungsteil der lidsconf für nötig gehalten, da sonst bei jedem Aufruf eines nicht-geschützten Programmes eine Fehlermeldung beziehungsweise eine Warnung auftritt. Die erste Überprüfung findet bereits in der lidsconf statt, also in dem Programm, welches die Regeln erstellt. Das zweite Mal wird im Kernel selber überprüft, ob die Datei ausgeführt werden darf. Die Überprüfung im Kernel findet in der Datei fs/exec.c statt. Wir haben als logische Folge den Teil, der für das Nicht-Ausführen von nicht-geschützten Dateien verantwortlich ist, auskommentiert.

Um nicht jedes Mal, wenn man einen neuen Benutzer anlegen will, seine Chroot-Umgebung per Hand einrichten zu müssen, haben wir wiederum ein Skript geschrieben, das diese Aufgaben für den Administrator übernimmt. Der Befehl, mit dem man normalerweise Benutzer einrichtet (adduser) ist in dieser Hinsicht sehr flexibel, da er durch ein selbst erstelltes Skript, welches man in /usr/local/sbin legt und adduser.local nennt, sehr einfach erweiterbar ist.
Hier das Skript:

```
#!/bin/sh
#
# This script copies all files needed to let the new user be chroot'ed
#

ROOT_TAR="/var/local/root.tar.gz"


add_chroot_user() {

CAPPIES="CAP_DAC_OVERRIDE CAP_DAC_READ_SEARCH CAP_FOWNER CAP_FSETID \
CAP_SETGID CAP_SETUID CAP_NET_RAW CAP_SYS_CHROOT CAP_SYS_PTRACE \
CAP_SYS_NICE"


echo -n "Extracting new root-files to $4: "
tar xpzf $ROOT_TAR -C $4 >/dev/null
if [ $? -eq 0 ]
        then echo done
```

```
        else echo FAILED!
fi


echo -n "Building new home: "
mkdir $4/home/$1
FILES=‘find /etc/skel -type f | sed "s|/etc/skel/\(.*\)|\1|"‘
for i in $FILES
do
        rm $4/$i
        cp /etc/skel/$i $4/home/$1
done
echo "done"


echo -n "Inserting proc in fstab: "
echo "proc      $4/proc proc    defaults        0       0">>/etc/fstab
mount -a -t proc
echo "done"


echo -n "Adding user in local passwd/shadow: "
tail -1 /etc/passwd |sed "s|\(.*\):.*|\1:/bin/bash|" >> $4/etc/passwd
tail -1 /etc/shadow >> $4/etc/shadow
echo "done"



echo "Changing chroot-login-shell: "
chsh -s /usr/local/bin/local_login $1
echo "done"


echo -n "Generating LIDS-rules: "
lidsconf -A -o $4/usr/local/bin/login -j READONLY
echo "done"
echo "Remeber to reload your LIDS-config!!"



for CAP in $CAPPIES
do
        /sbin/lidsconf -A -s $4/usr/local/bin/login/local_login -o $CAP\
-j GRANT >/dev/null
done

}


echo
```

```
echo "This script adds a new chroot-user."
echo "Do you want to continue? [Y/n]"
read continue
if [ -z "$continue" -o "$continue" = "Y" -o "$continue" = "y" ]
        then add_chroot_user $*
fi
```

Das Skript erfüllt mehrere Aufgaben: Das wohl wichtigste ist, daß nicht jeder Benutzer, der eingerichtet wird, ein Chroot-User ist. Das heißt, daß man die Ausführung dieses Skriptes auch verhindern können muß. Soll doch ein Chroot-User hinzugefügt werden, wird ein bereits vorhandenes Tar-Archiv in das neue Home-Verzeichnis entpackt, das proc-System gemountet, im neuen System die /etc/passwd und /etc/shadow aktualisiert, seine Login-Shell im Hauptsystem geändert, die LIDS-Regel für seine login-Shell erstellt, und die gewünschten Capabilities auf die Login-Shell zugewiesen.
Ebenso haben wir für das Löschen eines Benutzers ein Skript geschrieben, das wie folgt aussieht:

```
#!/bin/sh
# /usr/local/sbin/deluser.local
#
# Written for LIDS-Project on FH-Wedel

echo "Delete remaining home from $1? [Y/n]"
read continue
if [ -z "$continue" -o "$continue" = "Y" -o "$continue" = "y" ]
then

        echo -n "Removing proc-line in /etc/fstab: "
        mv /etc/fstab /etc/fstab.old
        cat /etc/fstab.old | grep -v "$4/proc" >> /etc/fstab
        echo "cat /etc/fstab.old | grep -v $4/proc >> /etc/fstab"
        echo "done"

        echo -n "Removing lines in lids.conf: "
        mv /etc/lids/lids.conf /etc/lids/lids_conf.old
        cat /etc/lids/lids_conf.old | grep -v "$4" >> /etc/lids/lids.conf
        echo "cat /etc/lids/lids_conf.old | grep -v $4 >> /etc/lids/lids.conf"
        echo "done"


        echo -n "Unmounting proc: "
        if [ -n "`mount|grep $4`" ]
                then umount "$4"/proc
        fi

        echo "Removing home-directory of user $1: "
        rm -rf "$4"
        echo "done"
fi
```

```
exit 0
```

Dieses Skript hilft dem Administrator, die Dateien und Einträge eines Chroot-Users, die nicht von dem normalen deluser-Befehl gelöscht werden, aus dem System zu entfernen. Im Einzelnen wird der proc-Eintrag aus der /etc/fstab entfernt, Die Einträge aus der /etc/lids/lids.conf gelöscht, ein eventuell gemountetes proc-Dateisystem aus dem System ausgeklinkt und das restliche Home-Verzeichnis des Benutzers gelöscht.

6.4.1 Mehrfach gemountetes proc-Dateisystem

Wie schon erwähnt, muß man in jedem Home-Verzeichnis das proc-Dateisystem mounten, so daß die entsprechenden User mit ihrem System arbeiten können. Ein Problem entsteht dabei allerdings: Wenn der Rechner heruntergefahren werden soll, werden in dem Skript /etc/init.d/umountfs alle Dateisysteme außer proc aus dem System ausgeklinkt. Dabei ist es für das Skript irrelevant, wo das proc-System gemountet wurde.

```
# We leave /proc mounted.
echo -n "Unmounting local filesystems... "
umount -tnoproc $FORCE -a -r
echo "done."
```

Dieser Teil wurde von uns in ein eigenes Skript ausgelagert und ein wenig modifiziert:

```
#!/bin/sh
#
# /usr/local/sbin/umountproc
#
# This is a little skript for unmounting all mounted proc-systems
# Written for LIDS-project on FH-Wedel
#

# finding all homes from users with additionally mounted proc
HOMES=‘grep "/usr/local/bin/local_login" /etc/passwd | cut -d ":" -f 6‘

for home in $HOMES
        do umount $home/proc >/dev/null 2>&1
done
```

/etc/init.d/umountfs sieht dann folgendermaßen aus:

```
# We leave /proc mounted.
echo -n "Unmounting local filesystems... "
# For unmounting all mounted procs except /proc
/usr/local/sbin/umountproc
echo procs unmounted
```

Dabei ist es wichtig, erst die mehrfach gemounteten proc-Systeme auszuklinken, so daß man die anderen gemounteten Partitionen ohne Probleme unmounten kann. Der Hauptmountpunkt des proc-Systems wird dabei nicht tangiert.

Kapitel 7

Fazit

Das Konzept von Lids scheint auf den ersten Blick zu überzeugen. Es ist jedoch aufgrund der mangelnden flexibilität von LIDS sehr problematisch, dynamische und stark verändernde Systeme zu schützen. Es scheint nur der Einsatz in nicht stark veränderlichen Systemen möglich, wie es bei einem DNS- oder Webserver der Fall ist.
Leider ist der Aufwand zum Einrichten von LIDS ziemlich hoch, so daß es einem Administrator eine große zusätzliche Belastung auferlegen wird. Dieser zusätzliche Aufwand ist jedoch kein Garant für mehr Sicherheit. Es arbeiten sich nur zwei Entwickler an dem Projekt, die beide auf getrenntem Weg entwicklen. Hierbei ist z.B. ein Bug entstanden, der fatale Auswirkungen auf die Sicherheit hatte, die ein normales Linuxsystem von Grund auf bietet, so daß die scheinbare Sicherheit von LIDS sich kurze Zeit gewahrt wurde.
Man sollte momentan die Verwendung von LIDS vermeiden, bis das Konzept besser umgesetzt wurde und der Konfigurationsaufwand eventuell geringer ausfällt.

Kapitel 8

Quellen

- LIDS-FAQ (http://www.lids.org/lids-faq/LIDS-FAQ.html)
- Kernel-Patch und LIDS-Sourcen (http://www.lids.org/download/lids-1.1.1-2.4.16.tar.gz)
- Mailing Liste (http://www.lids.org/maillist.html)
- capabilities.h (/usr/src/linux/include/linux)
- ftp://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.2/README