



# 24 Die erste eigene Policy

In diesem Kapitel werden wir unsere erste eigene Policy für eine neue Applikation erzeugen. Nein, wir werden nicht eine komplette Policy für ein ganzes System aus der Taufe heben. Dies würde den Rahmen des Buches sprengen und macht auch für die meisten Anwender keinen Sinn (siehe jedoch Kapitel 29).

Basierend auf der mitgelieferten *Targeted-Policy* unserer Distribution werden wir nun für einen Befehl, der im Moment noch nicht überwacht wird, eine eigene Policy entwickeln und hierbei alle einzelnen Schritte durchlaufen. Diese Policy sollte aber anschließend auch bei Verwendung der *Strict-Policy* fast unverändert funktionieren (siehe Abschnitt 28.1).

Hierzu habe ich für dieses Kapitel eine ganz einfache Applikation ausgewählt: `date`. Der Befehl `date` hat zwei ganz einfache Aufgaben:

- Anzeige des Datums und der Uhrzeit
- Stellen des Datums und der Uhrzeit

Wir werden nun eine Policy erzeugen, die den Befehl `date` gesondert überwacht und seine Aktionen prüft.

Einige Leser mögen nun enttäuscht sein, dass ich nicht mit einer Policy für ein Oracle-Database-Management-System beginne. Die hier vorgestellten Schritte lassen sich aber so auf alle weiteren Programme übertragen. Im weiteren Verlauf werde ich Ihnen auch zeigen, wie Sie eine Policy für Netzwerkdienste und kompliziertere Applikationen erzeugen.

Die folgenden Schritte sollten unabhängig von der eingesetzten Distribution funktionieren. Wichtig ist jedoch, dass diese Distribution die modulare *Reference-Policy* verwendet. Für Anwender der Example-Policy (Fedora Core 3, 4 und Red Hat Enterprise Linux 4) werden Hinweise in dem Kapitel 31 gegeben. Dennoch ist es sinnvoll, dass Sie dieses Kapitel zunächst lesen.



### Achtung

Ich werde hier die *Fedora Core*-Distribution voraussetzen. Die Beispiele wurden unter *Fedora Core* umgesetzt. Falls Sie dies unter *Debian* durchführen möchten, sollten Sie zunächst den folgenden Punkt beachten.

Sie benötigen das Paket `selinux-policy-refpolicy-dev`. Dieses Paket enthält die Scripts und Befehle für die Erzeugung der Richtlinien. Wenn es noch nicht installiert sein sollte, benötigen Sie auch noch das Paket für den Makroprozessor *M4* (`m4`). Im Weiteren können sich einige Ausgaben der Befehle und einige Pfade unterscheiden. Wundern Sie sich bitte nicht.

Bei Einsatz der *Debian*-Distribution sollten Sie beachten, dass *Debian Etch* aktuell noch nicht den *Audit-Daemon* enthält. Daher erfolgt die SELinux-Protokollierung in der Datei `/var/log/syslog`.

Bevor Sie beginnen, stellen Sie bitte sicher, dass das Paket `selinux-policy-devel` installiert ist. Wir benötigen die in diesem Paket vorhandenen Dateien, um weitere Module für die geladene Policy zu entwickeln.

## 24.1 Start

Um nun mit der Entwicklung zu beginnen, benötigen wir zunächst drei Dateien, deren Namen sich lediglich in der Endung unterscheiden:

- `date.te`
- `date.fc`
- `date.if`

Die Datei `date.te` enthält später die *Type-Enforcement*-Regeln. In der Datei `date.fc` speichern wir die File-Context-Informationen, und die Datei `date.if` definiert eine Schnittstelle, sodass andere SELinux-Module die hier definierten Funktionen nutzen können. Zunächst beschäftigen wir uns nur mit den ersten beiden Dateien.

Erzeugen Sie sich hierzu an geeigneter Stelle ein Verzeichnis, und legen Sie zunächst die drei Dateien leer an:

```
[root@supergrobi ~]# mkdir selinux-date
[root@supergrobi ~]# cd selinux-date/
[root@supergrobi selinux-date]# touch date.te
[root@supergrobi selinux-date]# touch date.fc
[root@supergrobi selinux-date]# touch date.if
```

## 24.2 Domänen und Typen

Wir erzeugen zunächst nur ein Policy-Gerüst, das wir anschließend schrittweise mit den Richtlinien füllen. Damit der Prozess `date` von SELinux überwacht werden kann, müssen wir für diesen Prozess eine eigene Domäne definieren. erinnern Sie sich: Eine Domäne ist das Gleiche wie ein Typ. Der Unterschied ist zunächst nur sprachlicher Natur.

Die Definition des Typs `date_t` erfolgt in der Datei `date.te`. Wie gelingt es uns nun, dafür zu sorgen, dass bei dem Aufruf des Befehls `date` der entstehende Prozess in der Domäne `date_t` gestartet wird? Wir benötigen eine *Domänentransition*. Ruft ein Benutzer den Befehl `date` auf, so würde dieser per Default in der Domäne des Benutzers gestartet werden. Nun soll automatisch ein Wechsel in die Domäne `date_t` erfolgen. Um diese Domänentransition nur bei dem Aufruf des Befehls `/bin/date` auszulösen, benötigen wir noch einen weiteren Typ für die Binärdatei `/bin/date`. In Anlehnung an die restliche Policy verwenden wir hier `date_exec_t`.

Wir benötigen also zwei Typen in der Datei `date.te`:

```
policy_module(date,1.0.0)

type date_t;
type date_exec_t;
```

Die erste Zeile definiert den Namen und die Version des Moduls. Diese werden später von dem Befehl `semodule -l` angezeigt. Die beiden nächsten Zeilen definieren die Typen `date_t` und `date_exec_t`.

Nun müssen wir noch die Policy davon in Kenntnis setzen, dass der erste Typ eine Domäne ist. Der zweite Typ wird verwendet, um einen Wechsel der Domäne zu erzwingen. Dies könnten wir in expliziten SELinux-Regeln beschreiben. Schöner ist es jedoch, wenn wir Schnittstellen der geladenen Policy verwenden. Um auf diese Schnittstellen zuzugreifen, musste das Paket `selinux-policy-devel` installiert werden. In diesem Paket finden Sie unter `/usr/share/selinux/devel/include/*` die Schnittstellen der SELinux Policy. Für den Umgang mit Domänen befindet sich die Schnittstelle in `/usr/share/selinux/devel/include/kernel/domain.if`<sup>1</sup>. Wenn Sie diese Datei öffnen und lesen, finden Sie dort reichlich Kommentare und Schnittstellen-Abschnitte (*Interface*). Zunächst benötigen wir eine Möglichkeit, um einen Typ als Domäne einsetzen zu können. Wir finden die Schnittstelle mit dem Namen `domain_type`:

```
#####
## <summary>
##     Make the specified type usable as a domain.
## </summary>
## <param name="type">
```

<sup>1</sup> Wie Sie weitere Schnittstellen finden und zuordnen, zeige ich Ihnen weiter unten.

```

###      <summary>
###      Type to be used as a domain type.
###      </summary>
### </param>
#
interface('domain_type', '
    # start with basic domain
    domain_base_type($1)

...

```

Wir benötigen eine zweite Schnittstelle, um den Typ *date\_exec\_t* als Eintrittspunkt in die Domäne *date\_t* zu definieren. Hier benutzen wir die Schnittstelle *domain\_entry\_file*. Eine dritte Schnittstelle (*domain\_auto\_trans*) erlaubt der Domäne *unconfined\_t* tatsächlich den Wechsel in die Domäne *date\_t* bei dem Aufruf einer Datei vom Typ *date\_exec\_t*.

Unsere Datei *date.te* sieht nun folgendermaßen aus:

```

policy_module(date,1.0.0)

type date_t;
type date_exec_t;

domain_type(date_t)
domain_entry_file(date_t, date_exec_t)
domain_auto_trans(unconfined_t, date_exec_t, date_t)

```



### Achtung

Achten Sie auf die Syntax in der TE-Datei. Echte SELinux-Policy-Anweisungen werden mit einem Semikolon abgeschlossen, während Schnittstellenaufrufe ohne Semikolon geschrieben werden.



### Hinweis

Sie können sich diese Schnittstellen merken. Diese benötigen Sie bei jeder neuen Policy.

## 24.3 File-Contexts

Nun müssen wir noch dafür sorgen, dass die Binärdatei `/bin/date` den richtigen *Security-Context* erhält. Hierfür ist die Datei `date.fc` verantwortlich. In dieser Datei werden alle Dateikontexte definiert, die für die Funktion der Applikation erforderlich sind. Da der Befehl `date` ohne Protokoll-, Konfigurations- und weitere temporäre Dateien auskommt, genügt hier ein Eintrag in der Datei `date.fc`:

```
# date executable will have:
# label: system_u:object_r:date_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/bin/date -- gen_context(system_u:object_r:date_exec_t,s0)
```

Diese Datei ist aus drei Spalten aufgebaut. In der ersten Spalte befindet sich der Name der Datei. Hierbei kann es sich auch um einen regulären Ausdruck handeln. In der zweiten Spalte wird der Dateityp angegeben:

- `--`: Eine einfache Datei
- `-d`: Ein Verzeichnis
- `-l`: Eine Verknüpfung

Bleibt die zweite Spalte leer, so wird der Dateityp nicht eingeschränkt, sondern die Datei darf einen beliebigen Typ besitzen. In der dritten rechten Spalte wird der Context angegeben, den die Datei aufweisen soll. Auf einem *MLS/MCSystem* wird hierzu das Makro `gen_context` verwendet. Unterstützt das SELinux-System kein *MLS/MCS*, kann der Context auch direkt eingetragen werden.

## 24.4 Übersetzung

Nun müssen die von uns erzeugten Textdateien in ein binäres SELinux-Policy-Modul übersetzt werden. Hierzu ist in dem Entwicklungspaket ein `Makefile` enthalten, das wir hierzu verwenden.

```
[root@supergrobi selinux-date]# make -f /usr/share/selinux/devel/Makefile
file
Compiling targeted date module
/usr/bin/checkmodule: loading policy configuration ◀
    from tmp/date.tmp
/usr/bin/checkmodule: policy configuration loaded
/usr/bin/checkmodule: writing binary representation ◀
    (version 6) to tmp/date.mod
Creating targeted date.pp policy package
rm tmp/date.mod tmp/date.mod.fc
```



### Achtung

Dieses Makefile ist bei der Debian-Distribution nicht ganz vollständig. Daher erzeugen Sie sich in Ihrem aktuellen Verzeichnis selbst einen kleinen Zweizeiler, der dann die Aufgabe übernimmt. Erzeugen Sie die Datei `Makefile` mit folgendem Inhalt:

```
HEADERDIR:=/usr/share/selinux/refpolicy-targeted/include
include $(HEADERDIR)/Makefile$
```

Achten Sie bitte auf die Groß- und Kleinschreibung. Anschließend können Sie dann den Befehl `make` ohne den Verweis mit der Option `-f ...` verwenden.

Achten Sie darauf, dass Sie sich bei dem Aufruf weiterhin in dem Ordner befinden, in dem Sie Ihre Dateien abgespeichert haben. Der Verweis auf das Makefile erfolgt mit der Option `-f /usr/share/selinux/devel/Makefile`. Bei dem Aufruf wird der Befehl `make` entsprechend dem `Makefile` versuchen, sämtliche `*.te`-Dateien in dem aktuellen Verzeichnis zu übersetzen und für jede `*.te`-Datei ein Policy-Package zu bauen. Daher ist es sinnvoll, in der Zukunft für jedes Package ein eigenes Verzeichnis zu verwenden.

Nun haben Sie Ihr erstes *Policy!-Package* erzeugt: `date.pp`. Bei dem Übersetzen erkennt das Kommando `make`, welche Policy Sie geladen haben. In meinem Fall handelt es sich um die *Targeted-Policy*. Falls Sie gerade die *Strict-Policy* verwenden, werden Sie an den entsprechenden Stellen in der Ausgabe des Befehls diese referenziert finden. Bitte wechseln Sie aber für diesen Test in die *Targeted-Policy*. Hiermit ist die Erstellung zunächst wesentlich einfacher. Wir werden später das Modul auch für die *Strict-Policy* erzeugen (siehe Kapitel 28).

## 24.5 Laden der Policy und Labeln der Dateien

Nun müssen wir diese Policy erstmals laden. Hierfür verwenden Sie den Befehl `semodule`. Nach dem Laden können Sie mit dem gleichen Befehl überprüfen, ob das Modul erfolgreich geladen wurde:

```
[root@supergrobi selinux-date]# semodule -i date.pp
[root@supergrobi selinux-date]# semodule -l | grep date
date      1.0.0
```

Nachdem das Modul geladen worden ist, kann mit dem Befehl `restorecon` der Kontext der binären Datei `/bin/date` entsprechend eingestellt werden:

```
[root@supergrobi selinux-date]# ls -Z /bin/date
-rwxr-xr-x root root system_u:object_r:bin_t /bin/date
[root@supergrobi selinux-date]# restorecon /bin/date
[root@supergrobi selinux-date]# ls -Z /bin/date
-rwxr-xr-x root root system_u:object_r:date_exec_t /bin/date
```

Nun besitzt die Datei den richtigen Kontext. In Kombination mit dem TE-Richtlinien-gerüst wird beim Aufruf dieses Befehls SELinux den Prozess in der Domäne *date\_t* überwachen.

## 24.6 Test und Analyse der Fehlermeldungen

Nun ist es an der Zeit, unser Policy-Package zu testen. Ob der Befehl *date* weiterhin funktioniert, hängt vom Zustand des SELinux-Systems (Permissive oder Enforcing) ab. Im Enforcing-Modus sollte der Befehl nicht funktionieren, da wesentliche TE-Regeln noch fehlen. Im Permissive-Modus sollte der Befehl funktionieren. Alle noch nicht erlaubten Zugriffe sollten jedoch protokolliert werden.

Am einfachsten erfolgt die Analyse tatsächlich im *Permissive*-Modus. Sie erhalten dann auf einen Schlag sämtliche Fehlermeldungen in den Protokollen für die weitere Analyse. Diesen Weg wollen wir hier auch gehen. Aktivieren Sie daher den Permissive-Modus für diesen Test. Hierfür müssten Sie natürlich beim Einsatz der *Strict-Policy* vorher in die Rolle *sysadm\_r* wechseln<sup>2</sup>.

```
[root@supergrobi ~]# setenforce 0
[root@supergrobi ~]# getenforce
Permissive
```

Führen Sie nun ein *Reload* der SELinux-Policy durch. Dies erleichtert später die Analyse der Fehlermeldungen, da der Reload protokolliert wird. So müssen wir uns nur auf die Meldungen nach dem letzten Reload konzentrieren.

```
[root@supergrobi ~]# semodule -R
```

Nun rufen wir den Befehl *date* auf. Im Permissive-Modus sollte der Befehl uns auch die aktuelle Uhrzeit und das Datum anzeigen.

```
[root@supergrobi ~]# date
Mo 6. Nov 19:16:26 CET 2006
```

Wenn Sie nun die in der Datei */var/log/audit/audit.log* protokollierten Fehlermeldungen analysieren, stellen Sie fest, dass der Befehl *date* tatsächlich in der Domäne *date\_t* ausgeführt wird. Der Source-Context bei den meisten Fehlermeldungen ist *root:system\_r:date\_t*:

```
type=AVC msg=audit(1168283757.301:91): avc: denied { read write }
      for pid=2890 comm="date" name="2" dev=devpts ino=4
      scontext=root:system_r:date_t:s0-s0:c0.c1023 tcontext=
      root:object_r:devpts_t:s0 tclass=chr_file
```

Eine Vielzahl von Fehlermeldungen wird protokolliert. Auf meinem Fedora Core 6-System handelt es sich um die folgenden Meldungen des *Access-Vector-Cache* (AVC):

<sup>2</sup> `newrole -r sysadm_r`

```

type=AVC msg=audit(1168283757.303:93): avc: denied { read } for
pid=2890 comm="date" name="ld.so.cache" dev=hda2
ino=352549 scontext=root:system_r:date_t:s0-s0:c0.c1023
tcontext=system_u:object_r:ld_so_cache_t:s0 tclass=file
type=AVC msg=audit(1168283757.303:94): avc: denied { getattr }
for pid=2890 comm="date" name="ld.so.cache" dev=hda2
ino=352549 scontext=root:system_r:date_t:s0-s0:c0.c1023
tcontext=system_u:object_r:ld_so_cache_t:s0 tclass=file
type=AVC msg=audit(1168283757.303:95): avc: denied { search }
for pid=2890 comm="date" name="lib" dev=hda2 ino=160161
scontext=root:system_r:date_t:s0-s0:c0.c1023 tcontext=
system_u:object_r:lib_t:s0 tclass=dir
type=AVC msg=audit(1168283757.303:95): avc: denied { read } for
pid=2890 comm="date" name="librt.so.1" dev=hda2 ino=
160214 scontext=root:system_r:date_t:s0-s0:c0.c1023
tcontext=system_u:object_r:lib_t:s0 tclass=lnk_file
type=AVC msg=audit(1168283757.303:95): avc: denied { read } for
pid=2890 comm="date" name="librt-2.5.so" dev=hda2
ino=162740 scontext=root:system_r:date_t:s0-s0:c0.c1023
tcontext=system_u:object_r:lib_t:s0 tclass=file
type=AVC msg=audit(1168283757.304:96): avc: denied { getattr }
for pid=2890 comm="date" name="librt-2.5.so" dev=hda2
ino=162740 scontext=root:system_r:date_t:s0-s0:c0.c1023
tcontext=system_u:object_r:lib_t:s0 tclass=file
type=AVC msg=audit(1168283757.304:97): avc: denied { execute }
for pid=2890 comm="date" name="librt-2.5.so" dev=hda2
ino=162740 scontext=root:system_r:date_t:s0-s0:c0.c1023
tcontext=system_u:object_r:lib_t:s0 tclass=file
type=AVC msg=audit(1168283757.304:98): avc: denied { read } for
pid=2890 comm="date" name="ld-2.5.so" dev=hda2
ino=162736 scontext=root:system_r:date_t:s0-s0:c0.c1023
tcontext=system_u:object_r:ld_so_t:s0 tclass=file
type=AVC msg=audit(1168283757.305:99): avc: denied { search }
for pid=2890 comm="date" name="/" dev=dm-2 ino=2
scontext=root:system_r:date_t:s0-s0:c0.c1023
tcontext=system_u:object_r:usr_t:s0 tclass=dir
type=AVC msg=audit(1168283757.305:99): avc: denied { search }
for pid=2890 comm="date" name="locale" dev=dm-2
ino=259845 scontext=root:system_r:date_t:s0-s0:c0.c1023
tcontext=system_u:object_r:locale_t:s0 tclass=dir
type=AVC msg=audit(1168283757.305:99): avc: denied { read } for
pid=2890 comm="date" name="locale-archive" dev=dm-2
ino=262709 scontext=root:system_r:date_t:s0-s0:c0.c1023
tcontext=system_u:object_r:locale_t:s0 tclass=file

```

```

type=AVC msg=audit(1168283757.305:100): avc: denied { getattr }
      for pid=2890 comm="date" name="locale-archive" dev=dm-2
      ino=262709 scontext=root:system_r:date_t:s0-s0:c0.c1023
      tcontext=system_u:object_r:locale_t:s0 tclass=file
type=AVC msg=audit(1168283757.307:101): avc: denied { getattr }
      for pid=2890 comm="date" name="2" dev=devpts ino=4
      scontext=root:system_r:date_t:s0-s0:c0.c1023
      tcontext=root:object_r:devpts_t:s0 tclass=chr_file

```

Diese Fehlermeldungen können nun mit dem Befehl `audit2allow` (siehe Abschnitt 18.3) analysiert werden. Rufen Sie hierzu den Befehl wie folgt auf:

```

[root@supergrobi ~]# audit2allow -l -i /var/log/audit/audit.log
allow date_t devpts_t:chr_file getattr read write ;
allow date_t etc_t:dir search;
allow date_t ld_so_cache_t:file getattr read ;
allow date_t ld_so_t:file read;
allow date_t lib_t:dir search;
allow date_t lib_t:file execute getattr read ;
allow date_t lib_t:lnk_file read;
allow date_t locale_t:dir search;
allow date_t locale_t:file getattr read ;
allow date_t usr_t:dir search;

```

So zeigt der Befehl alle benötigten *allow*-Regeln für unser Modul an. Leider berücksichtigt der Befehl bei diesem Aufruf nicht die modulare Struktur der Reference-Policy. Würden Sie diese Regeln so ohne weitere Anpassung in Ihre TE-Datei übernehmen und versuchen, das Modul neu zu übersetzen, würden Sie den folgenden Fehler erhalten:

```

[root@supergrobi selinux-date]# make -f /usr/share/selinux/devel/
      Makefile
Compiling targeted date module
/usr/bin/checkmodule: loading policy configuration from
      tmp/date.tmp
date.te:9:ERROR 'unknown type devpts_t' at token ';' on line 76741:
allow date_t devpts_t:chr_file getattr read write ;

/usr/bin/checkmodule: error(s) encountered while parsing
      configuration
make: *** [tmp/date.mod] Fehler 1

```

Der Typ `devpts_t` wird zwar durch die *Reference-Policy* definiert, ist aber in deren Modulen verborgen. Ein direkter Zugriff auf diesen Typ ist nicht erlaubt.

Es existieren zwei Auswege aus diesem Dilemma:

1. Sie verlangen spezifisch, dass die entsprechenden Typen bei der Übersetzung und beim Laden Ihres Moduls vorhanden sind. Falls sich jedoch später die Policy in einem Update ändert, besteht die Gefahr, dass auch die von Ihnen verwendeten Typen sich ändern.
2. Oder Sie benutzen die von der Policy zur Verfügung gestellten Schnittstellen. Diese sollten sich auch bei einem späteren Upgrade der Policy nicht ändern.

## 24.7 Policy mit Require-Block

Die erste Variante ist die Quick-and-dirty-Methode. Um diese einzusetzen, genügt es, den Befehl `audit2allow` mit der Option `-r` aufzurufen und das Ergebnis in der TE-Datei zu verwenden:

```
[root@supergrobi selinux-date]# audit2allow -r -l -i /var/log/ ◀
    audit/audit.log
require
    class chr_file getattr read write ;
    class dir search;
    class file execute getattr read ;
    class lnk_file read;
    type date_t;
    type devpts_t;
    type etc_t;
    type ld_so_cache_t;
    type ld_so_t;
    type lib_t;
    type locale_t;
    type usr_t;
    role system_r;
;

allow date_t devpts_t:chr_file getattr read write ;
allow date_t etc_t:dir search;
allow date_t ld_so_cache_t:file getattr read ;
allow date_t ld_so_t:file read;
allow date_t lib_t:dir search;
allow date_t lib_t:file execute getattr read ;
allow date_t lib_t:lnk_file read;
allow date_t locale_t:dir search;
allow date_t locale_t:file getattr read ;
allow date_t usr_t:dir search;
```

Nun erzeugt der Befehl zusätzlich einen `require`-Block, in dem die Verfügbarkeit der entsprechenden Typen und Klassen erzwungen wird.

Wenn Sie diesen Block in Ihre Datei `date.te` eintragen, hat diese den folgenden Inhalt:

```
policy_module(date,1.0.1)

type date_t;
type date_exec_t;
domain_type(date_t)
domain_entry_file(date_t, date_exec_t)
domain_auto_trans(unconfined_t, date_exec_t, date_t)

require {
    class chr_file { getattr read write };
    class dir search;
    class file { execute getattr read };
    class lnk_file read;
    type date_t;
    type devpts_t;
    type etc_t;
    type ld_so_cache_t;
    type ld_so_t;
    type lib_t;
    type locale_t;
    type usr_t;
    role system_r;
};

allow date_t devpts_t:chr_file { getattr read write };
allow date_t etc_t:dir search;
allow date_t ld_so_cache_t:file { getattr read };
allow date_t ld_so_t:file read;
allow date_t lib_t:dir search;
allow date_t lib_t:file { execute getattr read };
allow date_t lib_t:lnk_file read;
allow date_t locale_t:dir search;
allow date_t locale_t:file { getattr read };
allow date_t usr_t:dir search;
```

Nun gelingt auch die Übersetzung des Policy-Package. Achten Sie darauf, die *Versionnummer* Ihres Paketes anzupassen. Dies wird leider nicht automatisch vorgenommen.

Nun können Sie Ihre neue Version des Policy-Package `date.pp` laden:

```
[root@supergrobi selinux-date]# semodule -u date.pp
[root@supergrobi selinux-date]# semodule -l | grep date
date      1.0.1
```

Die Kontrolle der Versionsnummer zeigt Ihnen auch, dass der Austausch Ihres Moduls erfolgreich geglückt ist.

Testen Sie nun den Befehl `date` erneut. Finden Sie in den Protokolldateien AVC-Meldungen, die auf weitere Regelverstöße hinweisen? Falls dies der Fall ist, wiederholen Sie bitte die letzten Schritte.

Versetzen Sie SELinux wieder in den *Enforcing*-Modus. Funktioniert der Befehl immer noch? Melden Sie sich zum Test als einfacher Benutzer an. Kann auch ein unprivilegierter Benutzer den Befehl verwenden?

Herzlichen Glückwunsch. Sie haben Ihr erstes Modul erfolgreich geschrieben!

## 24.8 Policy unter Zugriff auf die Schnittstellen

Wie ich bereits beschrieben habe, handelt es sich bei der letzteren Vorgehensweise häufig um die schnellere, aber nicht so zukunftsicherere Variante. Um von Änderungen der Policy unabhängig zu sein, sollten Sie die von der Policy zur Verfügung gestellten Schnittstellen nutzen. Auch hierbei unterstützt Sie der Befehl `audit2allow`.

Entfernen Sie die in dem letzten Abschnitt hinzugefügten Zeilen wieder aus Ihrer Datei `date.te`, falls Sie die Schritte nachvollzogen haben, und übersetzen Sie erneut das Paket. Denken Sie daran, wieder die Versionsnummer zu erhöhen. Führen Sie wieder ein Upgrade des Moduls durch, und testen Sie den Befehl im *Permissive*-Modus:

```
[root@supergrobi selinux-date]# setenforce 0
[root@supergrobi selinux-date]# semodule -u date.pp
[root@supergrobi selinux-date]# date
Mo 8. Jan 21:36:15 CET 2007
```

Wenn Sie nun den Befehl `audit2allow` mit der Option `-R` aufrufen, wird der Befehl versuchen, die notwendigen Schnittstellen in der *Reference-Policy* selbst zu ermitteln und zu verwenden. Leider gelingt dies nicht immer in allen Fällen. Diese Option setzt natürlich auch die Installation des Pakets `selinux-policy-devel` voraus. In dem folgenden Listing habe ich die überflüssigen kommentierten Zeilen gelöscht. Ab Fedora Core 6 und bei Debian Etch sieht die Ausgabe auch ein wenig anders aus<sup>3</sup>. Wundern Sie sich daher nicht, falls bei Ihnen die Ausgabe auch etwas anders aussehen sollte.

---

<sup>3</sup> Die `optional_policy`-Container sind dort verschwunden.

**Achtung**

Falls Sie die Fehlermeldung `could not open interface info [/var/lib/sepolgen/interface_info]` bekommen, müssen Sie vorher noch den Befehl `sepolgen-ifgen` aufrufen. Dieser Befehl erzeugt die benötigte Datei.

```
[root@supergrobi selinux-date]# audit2allow -R -l -i /var/log/ ◀
audit/audit.log
allow date_t devpts_t:chr_file { getattr read write };

#allow date_t etc_t:dir search;
optional_policy('files', '
    files_search_etc(date_t)
');
allow date_t ld_so_cache_t:file { getattr read };

#allow date_t ld_so_t:file read;
optional_policy('libraries', '
    libs_use_ld_so(date_t)
');

#allow date_t lib_t:dir search;
optional_policy('libraries', '
    libs_search_lib(date_t)
');
allow date_t lib_t:file { execute getattr read };

#allow date_t lib_t:lnk_file read;
optional_policy('libraries', '
    libs_read_lib_files(date_t)
');

#allow date_t locale_t:dir search;
optional_policy('miscfiles', '
    miscfiles_read_localization(date_t)
');
allow date_t locale_t:file { getattr read };

#allow date_t usr_t:dir search;
optional_policy('files', '
    files_search_usr(date_t)
');
```

Wenn wir nun diese Zeilen an unsere rudimentäre Type-Enforcement-Datei `date.te` anhängen und versuchen, das Paket neu zu übersetzen, erscheinen auf Fedora Core 5 und 6 (ohne Updates) mehrere Fehlermeldungen:

```
[root@supergrobi selinux-date]# make -f /usr/share/selinux/devel/
Makefile Compiling targeted date module
date.te:15: Warning: deprecated use of module name (files) as
first parameter of optional_policy() block.
date.te:21: Warning: deprecated use of module name (libraries) as
first parameter of optional_policy() block.
date.te:26: Warning: deprecated use of module name (libraries) as
first parameter of optional_policy() block.
date.te:32: Warning: deprecated use of module name (libraries) as
first parameter of optional_policy() block.
date.te:37: Warning: deprecated use of module name (miscfiles) as
first parameter of optional_policy() block.
date.te:43: Warning: deprecated use of module name (files) as
first parameter of optional_policy() block.
/usr/bin/checkmodule: loading policy configuration from tmp/date.tmp
date.te:21:ERROR 'syntax error' at token '}' on line 76837:
#line 21
                } else {
/usr/bin/checkmodule: error(s) encountered while parsing
configuration
make: *** [tmp/date.mod] Fehler 1
```

Diese Meldungen weisen uns zunächst darauf hin, dass hier Probleme zwischen dem Kommando `audit2allow` und dem Compiler bestehen. Die einfachste Vorgehensweise ist zunächst das Löschen des Modulnamens in jedem `optional_policy`-Konstrukt.



### Achtung

Wenn Sie sämtliche Updates unter Fedora Core 6 installieren, taucht diese Fehlermeldung inzwischen nicht mehr auf. Auch Fedora 7 weist diesen Fehler nicht mehr auf.

Hier sehen Sie ein Beispiel, bei dem der Modulname `files` gelöscht wurde. Achten Sie darauf, dass weiterhin die richtigen Anführungszeichen gesetzt sind:

```
#allow date_t etc_t:dir search;
optional_policy('
    files_search_etc(date_t)
');
```

Dennoch treten weiterhin Fehler auf:

```
[root@supergrobi selinux-date]# make -f /usr/share/selinux/devel/
      Makefile Compiling targeted date module
/usr/bin/checkmodule: loading policy configuration from tmp/date.tmp
date.te:10:ERROR 'unknown type devpts_t' at token ';' on line 76742:
allow date_t devpts_t:chr_file { getattr read write };

/usr/bin/checkmodule: error(s) encountered while parsing
      configuration
make: *** [tmp/date.mod] Fehler 1
```

Der Compiler beschwert sich, dass der Typ `devpts_t` unbekannt sei. Dieser Typ steht nicht frei zur Verfügung, und eine Schnittstelle muss genutzt werden. Leider hat der Befehl `audit2allow` nicht selbst diese Schnittstelle gefunden. Um die richtige Schnittstelle zu finden, müssen Sie alle verfügbaren durchsuchen. Hier bietet sich der Befehl `grep` an:

```
# grep -R devpts_t /usr/share/selinux/devel/include/
/usr/share/selinux/devel/include/admin/su.if: dontaudit $1_su_t
      initrc_devpts_t:chr_file getattr ioctl ;
/usr/share/selinux/devel/include/admin/portage.if: allow $1
      portage_devpts_t:chr_file rw_file_perms setattr ;
/usr/share/selinux/devel/include/admin/portage.if:
      term_create_pty($1,portage_devpts_t)
/usr/share/selinux/devel/include/kernel/terminal.if:
      type devpts_t;
/usr/share/selinux/devel/include/kernel/terminal.if: allow $1
      devpts_t:filesystem associate;
...
```

Die Ausgabe habe ich gekürzt, um Platz zu sparen. Ansonsten hätte ich hier mehr als zwei Seiten benötigt. Wir müssen die Suche einschränken, um schneller zum Erfolg zu kommen. Eine genaue Analyse der Fehlermeldung zeigt, dass wir eine Schnittstelle benötigen, die folgende Funktion enthält:

```
allow date_t devpts_t:chr_file
```

Deshalb können wir unser Suchmuster konkretisieren: `allow $1 devpts_t:chr_file`. Falls wir hiermit keinen Erfolg haben, können wir immer noch den einen oder anderen Parameter weglassen. Der Parameter `$1` wird in der Schnittstelle als Platzhalter verwendet. Wenn wir die Schnittstelle benutzen, wird er durch unseren Typ `date_t` ersetzt.

Wenn wir nun suchen, erhalten wir:

```
[root@supergrobi selinux-date]# grep -R 'allow $1 devpts_t: chr_file' /usr/share/selinux/devel/include/
/usr/share/selinux/devel/include/kernel/terminal.if: allow $1 devpts_t:chr_file ioctl;
/usr/share/selinux/devel/include/kernel/terminal.if: allow $1 devpts_t:chr_file setattr;
/usr/share/selinux/devel/include/kernel/terminal.if: allow $1 devpts_t:chr_file rw_term_perms lock append ;
```

Dies sind nur noch drei Fundstellen. Nun müssen wir in der Datei `/usr/share/selinux/devel/include/kernel/terminal.if` die richtige Schnittstelle finden. Da wir Lese- und Schreibberechtigungen benötigen, handelt es sich um die Schnittstelle, in der `rw_term_perms` zugewiesen werden. Dies ist das `term_use_generic_ptys`-Interface:

```
#####
### <summary>
### Read and write the generic ptys
### type. This is generally only used in
### the targeted policy.
### </summary>
### <param name="domain">
### <summary>
### Domain allowed access.
### </summary>
### </param>
#
interface('term_use_generic_ptys', '
    gen_require('
        type devpts_t;
    ')

    dev_list_all_dev_nodes($1)
    allow $1 devpts_t:dir list_dir_perms;
    allow $1 devpts_t:chr_file { rw_term_perms lock append };
')
```

Wir können nun unsere Datei `date.te` editieren und die Zeile

```
allow date_t devpts_t:chr_file { getattr read write };
```

durch die Zeilen

```
#allow date_t devpts_t:chr_file { getattr read write };
optional_policy(
    term_use_generic_ptys(date_t)
);
```

ersetzen. Bei einer erneuten Übersetzung stellen wir fest, dass auch die Zeile

```
allow date_t ld_so_cache_t:file { getattr read };
```

Probleme erzeugt. Auch hier ist die Anwendung einer Schnittstelle erforderlich. Wenn wir genauso wie vorhin suchen, erhalten wir:

```
[root@supergrobi selinux-date]# grep -R 'allow $1 ld_so_cache_t:
    file' /usr/share/selinux/devel/include/
/usr/share/selinux/devel/include/system/libraries.if:
    allow $1 ld_so_cache_t:file r_file_perms;
```

Eine Suche nach dem Interface-Namen in der Datei `libraries.if` ergibt: `libs_use_ld_so`. Bei Anwendung dieser Schnittstelle darf der Prozess auf Bibliotheken zugreifen. Eine Analyse der Datei `date.te` zeigt, dass diese Schnittstelle bereits verwendet wird. Es genügt daher, die entsprechende Zeile einfach auszukommentieren:

```
...
#allow date_t ld_so_cache_t:file { getattr read };
#allow date_t ld_so_t:file read;
optional_policy(
    libs_use_ld_so(date_t)
);
...
```

So nähern wir uns iterativ unserer Policy. Ein erneuter Übersetzungsversuch schlägt immer noch fehl. Diesmal stört sich der Compiler an der folgenden Zeile:

```
allow date_t lib_t:file { execute getattr read };
```

Scheinbar versucht der Befehl `date` Bibliotheken zu lesen und auszuführen. Ein Blick in die Schnittstellendatei `libraries.if` zeigt, dass wir noch die Schnittstelle `libs_use_lib_files` benötigen. Also editieren wir die Datei `date.te` wieder entsprechend:

```
#allow date_t lib_t:file { execute getattr read };
optional_policy(
    libs_use_lib_files(date_t)
);
```

Die letzte Zeile, die von dem Compiler noch mit einer Fehlermeldung quittiert wird, ist:

```
allow date_t locale_t:file { getattr read };
```

Eine Analyse der Schnittstelle `miscfiles_read_localization` in der Datei `misc-files.if` zeigt jedoch, dass diese Zeile auch in der schon geladenen Schnittstelle enthalten ist. Diese Zeile kann daher auskommentiert werden.

```
#allow date_t locale_t:dir search;
#allow date_t locale_t:file { getattr read };
optional_policy('
    miscfiles_read_localization(date_t)
');
```

Wenn wir nun das Policy-Package wieder bauen, gelingt dies auch:

```
[root@supergrobi selinux-date]# make -f /usr/share/selinux/devel/
Makefile
Compiling targeted date module
/usr/bin/checkmodule: loading policy configuration from tmp/date.tmp
/usr/bin/checkmodule: policy configuration loaded
/usr/bin/checkmodule: writing binary representation (version 6)
to tmp/date.mod
Creating targeted date.pp policy package
rm tmp/date.mod tmp/date.mod.fc
```

Nun können Sie das Modul wieder laden.

```
[root@supergrobi selinux-date]# semodule -u date.pp
libsemanage.semanage_direct_upgrade: Previous module date is same
or newer.
semodule: Failed on date.pp!
```

Uups. Hier haben wir wohl vergessen, die Versionsnummer im Modul anzupassen. Dies lässt sich aber schnell nachholen. Dann ist das Laden des Moduls erfolgreich.

Wenn Sie nun das SELinux-System wieder in den *Enforcing*-Modus versetzen und den Befehl `date` aufrufen, sollte dieser funktionieren. Im Protokoll dürfen keine Fehlermeldungen auftauchen.

Herzlichen Glückwunsch. Sie haben erneut erfolgreich ein SELinux- Policy-Package gebaut, das lediglich die Schnittstellen der Reference-Policy verwendet.

## 24.9 Setzen der Uhrzeit erlauben

Bisher erlaubt die Policy lediglich das Auslesen der Uhrzeit. Ein Setzen der Uhrzeit mit dem `date`-Kommando wird nicht erlaubt:

```
[root@supergrobi selinux-date]# date -s 07:00
date: das Datum kann nicht gesetzt werden: Die Operation ist
nicht erlaubt
Mi 10. Jan 07:00:00 CET 2007
```

Im Audit-Log taucht folgende Meldung auf:

```
type=AVC msg=audit(1168436786.621:432): avc: denied { sys_time } ←
      for pid=20569 comm="date" capability=25 scontext=root: ←
      system_r:date_t:s0-s0:c0.c1023 tcontext=root:system_r: ←
      date_t:s0-s0:c0.c1023 tclass=capability
```

Der Prozess mit dem Source-Context *root:system\_r:date\_t* versucht, auf die Capability (*tclass=capability*) *CAP\_SYS\_TIME* zuzugreifen. SELinux verhindert diesen Zugriff bisher.

Um das Setzen der Uhrzeit zu erlauben, müssen wir unsere Policy entsprechend erweitern. Die entsprechende Regel lautet:

```
allow date_t self:capability sys_time;
```

Hierfür benötigen wir keinen Schnittstellenzugriff. Wenn wir die Regel hinzufügen und unser Policy-Package neu übersetzen, müssen wir daran denken, dass wir auch die Versionsnummer inkrementieren. Nach dem Laden des neuen Package ist auch ein Setzen der Zeit möglich.

```
[root@supergrobi selinux-date]# semodule -u date.pp
[root@supergrobi selinux-date]# date -s 07:00
Mi 10. Jan 07:00:00 CET 2007
[root@supergrobi selinux-date]# date
Mi 10. Jan 07:00:02 CET 2007
```

## 24.10 Ist die Policy nun fertig?

Ist unsere Policy nun fertig? Nun, wir haben die beiden wesentlichen Anwendungen des Befehls *date* getestet. Er funktioniert in dieser Form sowohl für den Systemadministrator als auch den unprivilegierten Benutzer. Dennoch können wir nicht ausschließen, dass auch noch andere Programme auf dem System den Befehl *date* für ihre Funktionen nutzen. Häufig wird dieser Befehl in Scripts verwendet. Um sicherzustellen, dass das System trotz unserer Policy-Erweiterung so funktioniert, wie wir uns das vorstellen, sollten wir das System neu booten. So können wir prüfen, ob während des Bootvorgangs ein Script oder ein Dienst auf diesen Befehl zugreift und dieser Zugriff aktuell von unserer Policy unterbunden wird.

Auf einem Fedora Core 6-System ist das nicht der Fall. Daher scheinen wir tatsächlich mit der Policy fertig zu sein.

Falls wir später aber weitere Policys erzeugen, die genau dieses Problem aufweisen, müssen wir jetzt vorsorgen. Wir müssen eine Schnittstelle schaffen, sodass wir später in einem weiteren Policy-Package diese Schnittstelle zum *date.pp* nutzen können.

## 24.11 Interface

Jedes vollständige Policy-Package besteht normalerweise aus drei Dateien:

- `date.te`
- `date.fc`
- `date.if`

Während wir die ersten beiden Dateien bereits mit Inhalt gefüllt haben, haben wir bisher die letzte Datei ignoriert. Hierbei handelt es sich um die Interface-Datei. In der *Interface*-Datei werden Zugänge zu den in dem Policy-Package definierten Typen offengelegt. Hierbei handelt es sich meist um Zugänge zu folgenden Typen:

- Die Domäne, in der der Prozess ausgeführt wird. Hierfür wird eine Domänen-transition in dem Interface definiert.
- Logdateien. Um die Protokolldateien von Diensten lesen zu dürfen, wird eine Schnittstelle definiert, die dies anderen Domänen erlaubt.
- Lesen und Schreiben von Daten. Wenn ein Dienst Daten erzeugt oder anbietet, dann existieren häufig Schnittstellen, sodass auch andere Programme diese Daten lesen oder verändern dürfen.

Da wir in unserem Fall abgesehen von dem Hilfstyp (*date\_exec\_t*) nur einen neuen Typ (*date\_t*) definiert haben, benötigen wir auch nur eine Schnittstelle.

Die Schnittstelle hat folgendes Aussehen und folgende Syntax:

```
#####
## <summary>
##     Execute a domain transition to run date.
## </summary>
## <param name="domain">
##     Domain allowed to transition.
## </param>
#
interface('date_domtrans', '
    gen_require('
        type date_t, date_exec_t;
    ')

    domain_auto_trans($1,date_exec_t,date_t)

    allow $1 date_t:fd use;
    allow date_t $1:fd use;
    allow $1 date_t:fifo_file rw_file_perms;
    allow $1 date_t:process sigchld;
')
```

Hier wird zunächst die Dokumentation in XML angegeben. Damit der Compiler nicht über die Dokumentation stolpert, wird diese mit Kommentarzeichen versehen. Entsprechende Dokumentationsbrowser können diese jedoch extrahieren und anzeigen. Dann wird die Schnittstelle `date_domtrans` definiert. Diese Schnittstelle erwartet genau einen Parameter. Dieser Parameter ist der Name der Domäne, von der aus der Wechsel in die Domäne `date_t` beim Aufruf eines Binärprogramms vom Typ `date_exec_t` (`/bin/date`) erfolgen soll. Die Variable `$1` in der Schnittstelle wird durch den Parameter ersetzt. In der Schnittstelle wird mit dem `gen_require`-Makro zunächst sichergestellt, dass die Typen existieren. Dann wird die automatische Domänentransition definiert und die Kommunikation zwischen den beiden Domänen über File-Descriptors (`fd`) und Signale erlaubt.

In unserem Fall ist die Schnittstelle von geringer Bedeutung, da wir festgestellt haben, dass keine weitere Policy diese Schnittstelle benötigt.

## 24.12 Fazit

Ich hoffe, Sie haben mit diesem Kapitel einen kleinen ersten Einblick in die Erzeugung einer Policy für eine neue Applikation gewonnen. Die Erstellung der Policy ist nicht so einfach und so automatisiert wie bei AppArmor. Dafür bietet SELinux aber auch eine wesentlich weiter gefasste Überwachung und ist auf Fedora-, Red Hat Enterprise Linux- und Debian Etch-Systemen die erste Wahl.

Lassen Sie sich nicht verunsichern, wenn Sie meine Vorgehensweise in diesem Kapitel nicht direkt vollkommen verstanden haben. Wir werden in den folgenden Kapiteln auf einige Punkte genauer eingehen. Bei der täglichen Anwendung werden Ihnen viele Dinge in Fleisch und Blut übergehen, und Sie werden sich über viele jetzt unverständliche Aspekte bald keine Gedanken mehr machen.

Fassen wir noch einmal zusammen. Ein *Policy!-Package* besteht aus drei Dateien: einer Type-Enforcement-Datei mit der Endung `.te`, einer File-Context-Datei mit der Endung `.fc` und einer *Interface*-Datei mit der Endung `.if`. Diese Dateien haben auch immer einen typischen Aufbau. Im Folgenden zeige ich noch einmal den Inhalt der Dateien bei einer Targeted-Policy. Typisch für die Type-Enforcement-Datei ist der folgende Inhalt:

```
policy_module(myapp,1.0.0)

type myapp_t;
type myapp_exec_t;
domain_type(myapp_t)
domain_entry_file(myapp_t, myapp_exec_t)
domain_auto_trans(unconfined_t, myapp_exec_t, myapp_t)

term_use_generic_ptys(myapp_t)
files_search_etc(myapp_t)
libs_use_ld_so(myapp_t)
```

```

libs_search_lib(myapp_t)
libs_use_lib_files(myapp_t)
miscfiles_read_localization(myapp_t)

```

Die FC-Datei hat meist folgenden Inhalt:

```

# myapp executable will have:
# label: system_u:object_r:myapp_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/bin/myapp          --          gen_context(system_u:object_r: ◀
                        myapp_exec_t,s0)

```

Hier können aber auch weitere Typen definiert werden, wenn die Applikation auch temporäre Dateien oder Protokolle benötigt.

Schließlich gibt es noch die Interface-Datei. Hier werden die Schnittstellen definiert. In vielen Fällen existiert hier nur eine Schnittstelle:

```

#####
### <summary>
###     Execute a domain transition to run myapp.
### </summary>
### <param name="domain">
###     Domain allowed to transition.
### </param>
#
interface('myapp_domtrans', '
    gen_require('
        type myapp_t, myapp_exec_t;
    ')

    domain_auto_trans($1,myapp_exec_t,myapp_t)

    allow $1 myapp_t:fd use;
    allow myapp_t $1:fd use;
    allow $1 myapp_t:fifo_file rw_file_perms;
    allow $1 myapp_t:process sigchld;
')

```

Dies sind die Bausteine, aus denen alle Policy-Packages gebaut werden. Im nächsten Kapitel werden wir sehen, wie boolesche Variablen in der Policy definiert werden. Dann werden wir kompliziertere Packages für Netzwerkdienste bauen.