Mandatory Access Control für Linux-Systeme

Gut bewacht

Thorsten Scherf

Klassischerweise kommt auf Unix- und Linux-Rechnern das Prinzip der Discretionary Access Control zum Einsatz, das eine Zugriffsentscheidung lediglich anhand der Benutzeridentität und den damit verknüpften Berechtigungen fällt. Systeme mit Mandatory Access Control haben diese Schwäche nicht; die populärste Implementierung ist SELinux.

Security-Enhanced Linux (SELinux) ist mittlerweile ein fester Bestandteil des Linux-Kernels, einige Distributionen verwenden es dazu, eine Zugangskontrolle auf Basis der Mandatory Access Control zu implementieren. Allerdings ist seine Komplexität auch gefürchtet.

Dieser erste Teil des Tutorials stellt die Architektur und Administration eines SELinux-basierten Systems vor. Im zweiten Teil geht es um das Anpassen respektive Schreiben eines neuen Regelwerks (Policy), bevor sich der letzte Teil mit der nächsten SELinux-Generation in Gestalt einer Referenz-Policy mit binären Modulen beschäftigt. Der Kasten „Einige Voraussetzungen“ erklärt, was zum Nachvollziehen erforderlich ist.

Auf klassischen DAC-basierten Systemen steht ein Objekt vollkommen unter der Kontrolle seines Eigentümers. Dieser kann beispielsweise eine Datei löschen, modifizieren oder die Rechte ändern. Prozesse laufen unter dem User-Kontext des Benutzers, der den Prozess gestartet hat. Somit macht das System auch bei Prozessen den Zugriff auf Objekte lediglich anhand der Benutzeridentität beziehungsweise der UID fest. Es existiert neben der Kombination UID/Berechtigungen keine weitere Instanz, die den Zugriff von Subjekten (Benutzer, Prozesse) auf Objekte (Dateien, Sockets, Interfaces) überwacht, was Angreifern im Fall eines Root-Exploit das gesamte System zu Füßen legen kann.

MAC-basierte Systeme setzen neben den klassischen Unix-Berechtigungen auf eine weitere Zugriffskontrolle, die so genannte Mandatory Access Control (MAC). Anhand einer zentralen Instanz (der Security-Policy) und einem Security-Label – man spricht hier oft von einem Security-Kontext, den das System sowohl an Objekte als auch an Subjekte vergibt – findet eine fein granulierte Zugangskontrolle statt. Der Security-Server im Kernel erstellt aus den beiden Komponenten Policy und Label eine Zugangsmatrix, anhand derer er einen Zugriff gestattet oder verbietet. Vereinfacht gesagt, entscheidet SELinux, wie eine Quelle auf ein bestimmtes Target zugreifen darf. Die SELinux-Terminologie bezeichnet dies als Type-Enforcement (TE). SELinux unterstützt daneben die Role-based-Access-Control (RBAC) sowie die Multi-Level/Category-Security (MLS beziehungsweise MCS). Der erste Teil des Tutorials wird sich allerdings ausschließlich mit Type-Enforcement beschäftigen.

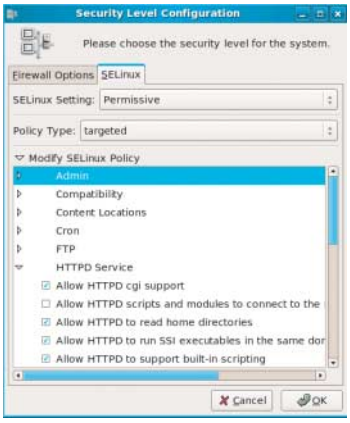
Mit TE in den Sandkasten

Oft bezeichnet man Type-Enforcement auch als Prozess-Sandboxing. Wie schon angesprochen, beschränkt diese Implementierung den Zugang von Prozessen auf bestimmte Objekte. Die mit RHEL4 ausgelieferte Targeted-Policy benutzt TE zur Absicherung gängiger Netzdienste wie Web-, DNS- oder Datenbank-Server. Daneben existiert die so genannte Strict-Policy, die jedoch nicht Bestandteil der RHEL4-Distribution ist. Im Gegensatz zur Targeted-Policy überwacht sie jedes einzelne Programm und jeden Benutzer.

iX-TRACT

- Klassische unixoide Systeme nutzen zur Überprüfung von Zugriffsberechtigungen ausschließlich die Identität des Benutzers.
- Läuft ein Programm im Benutzerkontext von Root, ist bei einem Sicherheitsloch die Sicherheit des gesamten Systems in Gefahr.
- Mandatory Access Control – MAC – verheißt einen Ausgleich der Unzulänglichkeiten – unter Inkaufnahme deutlich größerer Komplexität.
- Für Linux liefert das zum Standard-Kernel gehörende SELinux eine leistungsfähige, wenn auch komplexe MAC-Implemetierung.

In der SELinux-Terminologie ausgedrückt heißt das: Bei der Targeted-Policy laufen nur ausgesuchte Prozesse in einer eigenen Domäne, alle anderen Programme und Benutzer arbeiten in der ungeschützten Domäne *unconfined_t*, für die lediglich die DAC-basierten Kontrollmechanismen gelten. Bei der Strict-Policy existiert für jedes einzelne Programm und für jeden Benutzer eine eigene Domäne, das heißt, sie sichert das komplette System und nicht nur einzelne Teile.



Kommandozeilenmuffel können für SELinux-spezifische Einstellungen auch auf das grafische *system-config-securitylevel* zurückgreifen.

/etc/selinux/config beherbergt die Default-Einstellungen. Parameter legen den Modus (*SELINUX= … enforcing*, *permissive* oder *disabled*) sowie die beim Systemstart zu ladende Policy (*SELINUXTYPE= … targeted* oder *strict*) fest.

Über das Kommandozeilen-Tool *getenforce* kann sich der Anwender ebenfalls den gerade aktiven SELinux-Modus anzeigen lassen. Es sei an dieser Stelle noch einmal darauf hingewiesen, dass nur der *enforcing*-Modus wirklichen Schutz bietet, der *permissive*-Modus protokolliert lediglich nicht autorisierte Aktionen, verbietet sie jedoch nicht. Wer es gerne grafisch mag, dem bietet das Tool *system-config-securitylevel* die gleichen Konfigurationsmöglichkeiten unter X11 an.

Security-Label für Dateien und Prozesse

Wie erwähnt, versieht ein MAC-basiertes System jedes Objekt (Datei) beziehungsweise jedes Subjekt (Prozess) mit einem zusätzlichen Label. Die Aufrufe in Listing 1 zeigen das Security-Label eines Objekts beziehungsweise Subjekts an.

Listing 1: Ausgabe von Security-Label

```
[root@tiffy ~]# ls -lZ /var/www/html/index.html
-rw-r--r--  root  root  system_u:object_r:httpd_sys_content_t /var/www/html/index.html
[root@tiffy ~]# ps -AZ |grep httpd

system_u:system_r:httpd_t          4854 ?        00:00:00 httpd
system_u:system_r:httpd_t          4856 ?        00:00:00 httpd
system_u:system_r:httpd_t          4857 ?        00:00:00 httpd
system_u:system_r:httpd_t          4858 ?        00:00:00 httpd
```

Listing 2: SELinux-bezogener Log-Eintrag

```
May 19 00:22:56 grobi kernel: audit(1147990976.183:4): avc:  denied  { getattr }
for  pid=27392 comm="httpd" name="index.html" dev=dm-0 ino=616162
scontext=root:system_r:httpd_t tcontext=root:object_r:tmp_t tclass=file
```

Ein Security Label besteht aus den drei Komponenten <SELinux User-Identität>:<Rolle>:<Type oder Domäne>. Ist das Label an ein Objekt gebunden, bezeichnet man den dritten Teil des Labels als Type, gehört es zu einem Subjekt, spricht man von einer Domäne. Technisch gesehen sind jedoch beide Bezeichnungen identisch. In der SELinux-Terminologie laufen Prozesse in einer bestimmten Domäne, Dateien haben einen File-Type gesetzt. Beim TE spielt lediglich die dritte Komponente eine zentrale Rolle, also der Type respektive die Domäne. Bei der RBAC nehmen auch die ersten beiden Komponenten des Security-Labels eine zentrale Rolle ein, dazu später mehr.

Das Security-Label von permanenten Objekten (Dateien) speichert SELinux in den erweiterten Attributen einer Datei. Der Befehl *getfattr* gibt sie aus:

```
[root@tiffy ~]# getfattr -m security.selinux -d \
    /etc/shadow
getfattr: Removing leading ,/' from absolute \
    path names
# file: etc/shadow
security.selinux="system_u:object_r:shadow_t\000"
```

Selbst Dateisysteme, die keine erweiterten Attribute unterstützen, lassen sich in einer SELinux-Umgebung benutzen. Beim Einbinden kann der Administrator dem *mount*-Aufruf eine *context=user:rolle:type*-Option mitgeben, wodurch das System alle Dateien des Filesystems mit dem aufgeführten Label versieht. Die SELinux-Quellen lassen ebenfalls über die Datei *genfs_context* die Definition von Security-Labeln für nicht unterstützte Dateisysteme zu – auch dazu später mehr.

Die Aufgabe des Security-Servers im Kernel ist es nun, anhand von Security-Label und Policy eine Zugriffsentscheidung zu treffen. Aus Performance-Gründen speichert er alle Zugriffsentscheidungen in einem Cache zwischen, dem so genannten Access-Vector-Cache (AVC). In der Policy Datei *policy.conf* (alle in diesem Teil beschriebenen Quelldateien liegen im Verzeichnis */etc/selinux/targeted/src/policy/*) befindet sich unter anderem folgende Zeile:

```
allow httpd_t httpd_sys_content_t:file { getattr \
    read write append }
```

Diese Regel gestattet es allen in der Domäne *httpd_t* laufenden Prozessen, auf Dateiobjekte vom Typ *httpd_sys_content_t* zuzugreifen. Die genauen Zugriffsrechte finden sich in der geschweiften Klammer. Wie man sieht, stehen hier viel feiner granulierte Rechte zur Verfügung als bei einem klassischen DAC-System. Beispielsweise lässt sich exakt definieren, ob ein Benutzer lediglich in eine Datei schreiben oder ob er die Datei auch erzeugen darf. Die Datei *access_vectors* enthält eine genaue Übersicht aller zur Verfügung stehenden Rechte. SELinux differenziert auch bei den einzelnen Objekten sehr genau. Es unterschiedet unter anderem zwischen regulären, Link- und Block-Geräte-Dateien oder Sockets (TCP und UDP). Jedem Objekt stehen gewisse Rechte zur Verfügung, die der Administrator beim Schreiben eigener Regeln einsetzen kann. Die oft genutzte Objektklasse *file* verfügt beispielsweise über die folgenden Rechte: *ioctl*, *read*, *write*, *create*, *getattr*, *setattr*, *lock*, *relabelfrom*, *relabelto*, *append*, *unlink*, *link*, *rename*, *execute*, *swapon*, *quotaon*, *mounton*, *execute_no_trans*, *entrypoint* und *execmod*.

Findet ein nicht autorisierter Zugriff statt – beispielsweise von einem Prozess auf eine Datei –, protokolliert SELinux den Versuch in */var/log/messages* und unterbindet den Zugriff. Listing 2 zeigt einen beispielhaften Log-Eintrag.

Es handelt sich hier um eine Deny-Meldung des Access-Vector-Cache

Tutorialinhalt

Teil I:	Einführung, Architektur und Administration
Teil II:	Policies anpassen beziehungsweise schreiben
Teil III:	Frontends, Reference Policy, Binärmodule

Einige Voraussetzungen

Alle in diesem Tutorial vorgestellten Beispiele basieren auf einer RHEL-4-Installation (Red Hat Enterprise Linux), lassen sich jedoch leicht auf anderen Distributionen nachvollziehen, solange diese eine vollständige Unterstützung für SELinux mitbringen. Dazu zählen nicht nur der Support im Kernel (CONFIG_SECURITY_SELINUX =y), ein entsprechendes Regelwerk (selinux-policy-targeted.<version>.rpm sowie die dazu gehörigen Sourcen selinux-policy-targeted-sources.<version>.rpm), sondern auch ein um SELinux-Unterstützung erweitertes coreutils-Paket, das unter anderem modifizierte Versionen von ls, ps, cp und mv enthält. Letztere sind in der Lage, den an den Objekten beziehungsweise Subjekten gesetzten Security-Kontext anzuzeigen. Die dazu gehörende Kommanozeilenoption lautet meistens -Z. So zeigt ls -Z die Label von Dateien an, ps -AZ die von Prozessen. Auf der Sourceforge-Seite des Projekts (siehe „Quellen“) befindet sich eine Übersicht der Distributionen, die SELinux unterstützen.

Da Fedora Core 5 (FC5) bereits einen Technology-Preview der kommenden SELinux-Implementierung enhält (den auch das für Ende des Jahres erwartete RHEL5 verwenden wird), funktioniert das Anpassen der Policy dort etwas anders. selinux-policy-targeted-sources.<version>.rpm sucht man unter FC5 vergebens, stattdessen enthält das Paket selinux-policy-targeted.<version>.rpm einige binäre Basis-Module, die eine gewisse Anzahl von Programmen unter den Schutz von SELinux stellt. Mit Policy-Modulen lassen sich weitere Programme schützen. Wer die Basis-Module anpassen möchte, kann hierfür das zugehörige Quellcode-Paket (selinux-policy.<version>.srpm) herunterladen und daraus ein neues RPM-Paket bauen. Mehr dazu bringt der dritte Teil dieses Tutorials.

(*avc*), der zum Security-Server im Kernel gehört. Der Prozess *httpd* mit der Prozess-ID *27392* versucht, auf die Attribute der Datei *index.html* zuzugreifen. Die Datei hat die Inode-Nummer *616162* und liegt auf dem Device *dm-0*. Der Security-Context des Prozesses lautet *root:system_r:httpd_t*, der des Objekts *root:object_r:tmp_t*. Es scheint also keine Zugriffsregel für die Domäne *httpd_t* auf Dateien mit dem Type *tmp_t* zu geben. Somit wurde der Zugang verboten und der Zugriffsversuch im Syslog festgehalten.

Protokollverhalten und Policy

Allerdings gibt es neben dem Zugriffsvektor *allow* noch zwei weitere, die sich in der Policy verwenden lassen: *auditallow* und *dontaudit*.

Ersterer bewirkt, dass eine Aktion protokolliert wird, wenn der Security-Server sie gestattet – normalerweise ist das Gegenteil der Fall. *auditallow* lässt sich beispielsweise einsetzen, will man das Laden einer neuen Policy festhalten.

dontaudit hingegen verhindert das Logging bei verbotenen Aktionen. Dies kann für immer wieder auftretende, nicht erlaubte Aktionen sinnvoll sein, um ein Überlaufen der Log-Datei zu vermeiden.

In der Tat ist es so, dass zwei unterschiedliche Versionen der Policy existieren. Der Kernel lädt beim Systemstart eine Binärversion des Regelwerks in den Speicher. Diese Binärdatei trägt den einfachen Namen *policy.18*, wobei die Zahl die unterstützte Policy-Sprache des Kernels angibt. Neben der Binärvariante gibt es eine Source-Version, die man mit Hilfe des Makroprozessors *m4* aus diversen Einzeldateien zu einer *policy.conf* zusammensetzt. Hieraus lässt sich mit Hilfe des SELinux-Compilers *checkpolicy* wiederum eine Binärversion erstellen, die der Systemverwalter per *load_policy* in den Kernel laden kann. Dazu mehr im zweiten Teil des Tutorials in der nächsten Ausgabe.

Konfigurationsänderungen über M4-Makros

Zur leichteren Anpassung der Policy basieren die meisten Regeln und Anweisungen auf *m4*-Makros. Der *m4*-Makroprozessor ruft sie in den einzelnen Policy-Dateien auf und wertet sie aus. In der fertigen *policy.conf* befinden sich dann die expandierten Makro-Anweisungen, die noch zu kompilieren und in den Kernel zu laden sind.

Die zweite Aufgabe des Security-Servers ist es, zu bestimmen, welches Label neu erstellte Objekte beziehungsweise Subjekte erhalten sollen; zunächst zu den Objekten. Wird eine Datei in einem Ordner erzeugt, erbt diese üblicherweise das Security-Label des übergeordneten Verzeichnisses:

```
[root@tiffy ~]# ls -ldZ /tmp
drwxrwxrwt root  root  system_u:object_r:tmp_t \
   /tmp
[root@tiffy ~]# touch /tmp/test
[root@tiffy ~]# ls -lZ /tmp/test
-rw-r--r-- root  root  user_u:object_r:tmp_t   \
   /tmp/test
```

Dieses Verhalten ist nicht immer wünschenswert. Beispielsweise ist es sinnvoll, im Log-Verzeichnis nicht alle Dateien mit dem gleichen Label zu versehen, sondern dieses in Abhängigkeit vom Programm zu erzeugen, das die Log-Datei schreiben möchte. Dies lässt sich mit einem *m4*-Makro leicht erreichen. Die allgemeine Syntax hierfür lautet *file_type_auto_trans(creator_domain, parent_directory_type, file_ type, objectclass)*.

Möchte man also, dass Dateien, die der Webserver in */tmp* generiert, den Typ *httpd_tmp_t* anstelle von *tmp_t* bekommen, müsste der Makroaufruf folgendermaßen lauten:

```
file_type_auto_trans(httpd_t, tmp_t, httpd_tmp_t, file)
```

Dies bedeutet, dass SELinux bei allen Dateien eines Prozesses, der in der Domäne *httpd_t* läuft, als File-Type *httpd_tmp_t* setzt, solange diese in einem Ordner des Typs *tmp_t* erzeugt werden. Soll dies nicht nur für Dateien, sondern auch für Verzeichnisse gelten, ist als Objektklasse hier zusätzlich *dir* anzugeben.

Für Prozesse sieht es ähnlich aus. Startet ein Benutzer ein Programm, läuft es in derselben Domäne, in der sich der Benutzer befindet – der Targeted-Policy üblicherweise *unconfined_t*. Startet ein Programm einen anderen Prozess, läuft dieser ebenfalls in der gleichen Domäne wie das aufrufende Programm.

Ein kleines Beispiel zur Verdeutlichung: Wie weiter oben gezeigt, läuft der Webserver in der Domäne *httpd_t*. Startet nun der Webserver ein PHP-Skript, würde dies ebenfalls in der Domäne *httpd_t* laufen und hätte somit die gleichen Rechte wie der Webserver. Dies ist in den meisten Fällen nicht wünschenswert. Es wäre schön, wenn das durch den Webserver gestartete PHP-Skript in einer eigenen Domäne liefe, damit sich in dieser mit entsprechenden Regelsätzen der Zugang auf bestimmte Objekte einschränken ließe. Hierfür existiert das Makro *domain_auto_trans(parent_domain, programm_type, child_domain)*.

Für den gerade geschilderten Fall der PHP-Skripte sieht der Makro-Aufruf so aus:

```
domain_auto_trans(httpd_t, httpd_php_exec_t, \
   httpd_php_t)
```

Wobei *httpd_php_exec_t* den am PHP-Skript gesetzten File-Type darstellt.

Via *chcon* File-Context setzen

Es gibt viele Möglichkeiten, einen File-Type an einer Datei zu setzen. Über das Tool *chcon* kann dies manuell erfolgen:

```
[root@tiffy html]# ls -lZ skript.php
-rw-r--r-- root  root  user_u:object_r: \
    httpd_sys_content_t skript.php
[root@tiffy html]# chcon -t httpd_php_exec_t \
    skript.php
[root@tiffy html]# ls -lZ skript.php
-rw-r--r-- root  root  user_u:object_r: \
    httpd_php_exec_t skript.php
```

Ebenso lassen sich über die Option *-u* beziehungsweise *-r* die SELinux-User-Identität respektive die Rolle an einer Datei ändern. Da die Targeted-Policy aber fast ausschließlich mit dem letzten Teil des Security-Labels arbeitet, reicht hier die Option *-t*, mit der sich der File-Type an einer Datei setzen lässt.

Natürlich darf man nicht beliebige File-Types verwenden, sondern nur solche, die eine so genannte Type-Enforcement-Datei (**.te*) vorher definiert. Eine Definition in der *apache.te*-Datei (wird später per *m4* und allen anderen Policy-Dateien zu einer einzelnen, globalen *policy.conf* zusammengesetzt) für den Webserver könnte folgendermaßen aussehen:

```
type httpd_config_t, file_type, sysadmfile
```

Dies definiert den File-Type *httpd_config_t* mit den beiden Attributen *file_type* und *sysadmfile*. Letztere lassen sich später in Zugangsregeln dazu verwenden, den Zugriff von Prozessen auf Objekte mit einem bestimmten Attribut statt eines Types zu erlauben. Die möglichen Attribute finden sich in der Datei *attrib.te*.

Per *chcon* lässt sich dieser Type nun leicht an den Konfigurationsdateien des Webservers setzen:

```
[root@tiffy ~]# chcon -R -t httpd_config_t /etc/httpd
[root@tiffy ~]# ls -ldZ /etc/httpd/
drwxr-xr-x root  root  \
    system_u:object_r:httpd_config_t /etc/httpd/
```

Mit einer Reihe von Tools kann man das Label von Dateien automatisch setzen. Diese ziehen zur Bestimmung des einem bestimmten Objekt zugeordneten Labels die zugehörige File-Context-Datei (**.fc*) der Policy heran. *apache.fc* (wird später per *m4* und allen anderen File-Context-Dateien zu einer einzelnen, globalen Datei *file_context* zusammengesetzt) für den Webserver könnte folgendermaßen aussehen:

```
/etc/httpd(/.*)?    system_u:object_r:httpd_config_t
/etc/httpd/modules/ -d \
    system_u:object_r:httpd_modudules_t
```

Bei der Syntax der Datei standen reguläre Ausdrücke Pate. Die erste Zeile verknüpft das Verzeichnis */etc/httpd* und alle darin enthaltenen Dateien mit einem Security-Context, die zweite versieht lediglich das Verzeichnis (Option *-d*) */etc/httpd/modules/* mit einem Label.

Mit dem Tool *restorecon* lässt sich der Security-Context einer Datei leicht setzen, ohne dass man ihn explizit angeben muss. Das Tool sucht in der Policy nach einem passenden Eintrag für das Objekt:

```
[root@tiffy ~]# restorecon -R /etc/httpd
```

Listing 3: Ausgabe von *sestatus*

```
[root@tiffy ~]# sestatus
SELinux status:         enabled
SELinuxfs mount:        /selinux
Current mode:           enforcing
Mode from config file:  enforcing
Policy version:         18
Policy from config file:targeted

Policy booleans:
allow_syslog_to_console inactive
allow_ypbind            inactive
dhcpd_disable_trans     inactive
httpd_builtin_scripting inactive
httpd_disable_trans     inactive
httpd_enable_cgi        inactive
httpd_enable_homedirs   inactive
httpd_ssi_exec          active
httpd_tty_comm          inactive
httpd_unified           active
mysqld_disable_trans    inactive
named_disable_trans     inactive
named_write_master_zonesinactive
nscd_disable_trans      inactive
ntpd_disable_trans      inactive
pegasus_disable_trans   inactive
portmap_disable_trans   inactive
postgresql_disable_transinactive
snmpd_disable_trans     inactive
squid_disable_trans     inactive
syslogd_disable_trans   inactive
use_nfs_home_dirs       inactive
use_samba_home_dirs     inactive
use_syslogng            inactive
```

Ähnlich arbeiten die beiden Tools *setfiles* und *fixfiles*. Ersteres dient primär dazu, initial das Dateisystem korrekt mit einem Label zu versehen. Mit *fixfiles* lässt sich zusätzlich ein Check durchführen, sodass das System mit einem falschen Label ausgestattete Dateien lediglich anzeigt. Natürlich lassen sich diese per *restore*-Aufruf wieder mit dem korrekten Label versehen. Die Anweisung *fixfiles restore* würde beispielsweise das komplette Filesystem neu mit Labeln versehen. Eine weitere interessante *fixfile*-Option lautet *-R*; sie überprüft Dateien eines RPM-Pakets und erneuert gegebenenfalls deren Label.

Security-Context bei Netzwerk-Objekten

Bisher ging es nur um das Setzen von Labeln an Dateien. Natürlich lassen sich auch Netzwerkobjekte wie Ports oder Interfaces mit einem Security-Context versehen. Genau das erfolgt über die Datei *net_context*. In dieser kann der Administrator Netzwerk-Ports und -Interfaces mit einem (vorher definierten) Label verknüpfen.

```
[root@tiffy policy]# grep http_port_t net_contexts
portcon tcp 80 system_u:object_r:http_port_t
portcon tcp 443  system_u:object_r:http_port_t
```

Diese beiden Einträge weisen Port 80 und 443 den Typ http_port_t zu. Damit kann man seinen Webserver nun lediglich auf diesen beiden Ports betreiben. Die entsprechende Regel könnte so aussehen:

```
allow httpd_t http_port_t:tcp_socket name_bind;
```

Sie gestattet dem Webserver, sich an den mit dem Label *http_port_t* verknüpften Port zu binden. Würde der Server für einen anderen Port als 80 oder 443 konfiguriert, würde dieser erst gar nicht starten, da hierfür eine passende Regel fehlt.

Listing 4: Booleans in *apache.te*

```
### Den Default-Wert auf false setzen
bool httpd_enable_homedirs false;

### httpd_enable_homedirs Definition
if (httpd_enable_homedirs) {
allow { httpd_t httpd_sys_script_t httpd_suexec_t } user_home_dir_t:dir { getattr search };
}
```

Anpassungen über Booleans

Eine einfache Methode, ein SELinux-basiertes System an die eigenen Bedürfnisse anzupassen, stellen Booleans dar. Das Tool *sestatus* zeigt neben einigen weiteren Informationen alle zur Verfügung stehenden Booleans an, Listing 3 zeigt eine beispielhafte Ausgabe.

Über Booleans lassen sich diverse Eigenschaften eines SELinux-Systems verändern, ohne hierfür das eigentliche Regelwerk, die Policy, anpassen zu müssen. Beispielsweise kann der Administrator über *httpd_enable_homedirs* bestimmen, ob der Webserver auf die Home-Verzeichnisse der Benutzer zugreifen darf. Arbeitet man mit Usereigenen Webseiten und hat die *UserDir*-Anweisung in der Konfiguration des Webservers aktiviert, sollte dieser zumindest in der Lage sein, die Benutzerverzeichnisse nach einem Ordner *public_html* durchsuchen zu können. Genau dies erlaubt die Aktivierung des Boolean *httpd_enable_homedirs*:

```
[root@tiffy ~]# setsebool httpd_enable_homedirs=1
```

GUI-Liebhaber können alle Booleans über das schon erwähnte grafische Tool *system-config-securitylevel* verwalten. Alternativ kann das auch über das via */selinux* erreichbare virtuelle Dateisystem erfolgen:

```
[root@tiffy selinux]# echo 1 > \
   /selinux/booleans/httpd_enable_homedirs
[root@tiffy selinux]# getsebool httpd_ \
   enable_homedirs

httpd_enable_homedirs --> inactive pending: active
```

Allerdings muss man die Änderung erst bestätigen, bevor sie wirksam wird. Der Vorteil: Es lassen sich mehrere Booleans anpassen und zeitgleich durch ein abschließendes Commit aktivieren:

```
[root@tiffy selinux]# echo 1 > \
   /selinux/commit_pending_bools
[root@tiffy selinux]# getsebool \
   httpd_enable_homedirs
httpd_enable_homedirs --> active
```

Die Definition des Boolean erfolgt in *apache.te*, *m4* übernimmt sie beim Erzeugen der eigentlichen Policy in *policy.conf*. Listing 4 zeigt den relevanten Abschnitt.

Beim Aktivieren des Boolean erhalten die Domänen *httpd_t*, *httpd_sys_script_t* sowie *httpd_suexec_t* Zugriff auf Verzeichnisse mit dem Type *user_home_dir_t* und haben dort die Rechte *getattr* und *search*. Der Anwender kann dem Regelwerk beliebige eigene Booleans hinzufügen. Wie er eine bestehende Policy ändern oder neue Programme unter den Schutzschild von SELinux stellen kann, wird Thema des zweiten Tutorialteils sein. (avr)

THORSTEN SCHERF

Thorsten Scherf arbeitet als Security-Experte für Red Hat EMEA.

Quellen

SELinux
www.nsa.gov/selinux/

SELinux for Distributions
selinux.sourceforge.net

FC5-SRPMS
download.fedora.redhat.com/pub/fedora/linux/core/5/source/SRPMS/

Literatur

[1] Oliver Tennert; Linux-Kernel; Gut gerüstet; Sicherheit via Linux Security Modules; *iX* 2/2003, S. 105

[2] Oliver Tennert; Sicheres Linux; Hinter Schloss und Riegel; Mandatory Access Control mit SE-Linux; *iX* 12/2003, S. 114