

Regel-Praxis

SE Linux regelt die Zugriffe im System: Welcher Benutzer und welcher Prozess darf unter welchen Bedingungen worauf zugreifen? Die einzelnen Regeln sind in einer Policy festgelegt. Der Artikel erklärt, wie ein neues Regelwerk entsteht und was der Admin dabei beachten muss. Carsten Grohmann



Mit SE Linux schränkt der Admin die Zugriffe im System ein. Welcher Benutzer darf welches Programm starten, welches Programm darf welche Dateien lesen, welcher Benutzer darf in welcher Rolle mit welchem Programm welches File öffnen – es gibt viel zu regeln [5]. Der Backup-Client von Amanda [1] dient im Folgenden als Beispiel dafür, wie ein solches Regelwerk in der Praxis entsteht. SE Linux soll die Rechte von Amanda so weit einschränken, dass es nur noch auf die wirklich notwendigen Dateien zugreifen kann. Die fertige Policy ist bereits in SE Linux enthalten.

Falsche oder fehlende Regeln höhlen die zusätzliche Sicherheit aus, SE Linux schützt dann nicht besser als ein Standard-Linux. Beispielsweise laufen Prozesse ohne eigene Domäne in der aufrufenden Domäne. Das ist kritisch, wenn etwa ein Systemdienst weiterhin in der »initrc_t«-Domäne läuft, denn gerade sie

hat weit reichende Rechte. Ihre eigentliche Aufgabe endet nach dem Start der Skripte unter »/etc/init.d/«. Problematisch sind auch vom Benutzer gestartete Programme, die wichtige Fähigkeiten wie »setuid« oder »sys_nice« nutzen, aber in der Benutzerdomäne laufen. Dadurch erhalten sie Zugriff auf Objekte, die sie nicht benötigen und daher nicht lesen dürften.

Voraussetzungen

Um die Typen und Regeln für ein neues Programmpaket korrekt zu vergeben, ist einiges an Hintergrundwissen nötig. Als Erstes ist zu klären, ob die ausführbaren Komponenten statisch oder dynamisch gelinkt sind. Statische Programme müssen keine dynamischen Bibliotheken öffnen und laden, somit kann im Regelwerk das Makro »uses_shlib« entfallen, denn es würde den Zugriff auf die dynamischen Bibliotheken gestatten.

Der nächste Schritt betrifft das Startverhalten des Programms: Wird es automatisch vom Init-Prozess gestartet, bei Bedarf vom Inetd, regelmäßig vom Crond, oder manuell von den Anwendern? In jedem Fall sind die berechtigten Benutzer und Rollen zu konfigurieren. Anschließend benötigt der Admin die Datei- und Verzeichnisliste der Software. Sie dient als Grundlage für die File-Context-Datei (»*.fc«). Aus RPM-Paketen lässt sich diese Liste mit Hilfe von »rpm -ql Paketname« erzeugen.

Gedanken über die Berechtigungen für die Programmdomäne schließen diesen Punkt ab. Welche Dateitypen darf das Programm lesen? Benötigt es umfassenden Netzwerkzugriff? Sind Setuid- und Setgid-Aufrufe erlaubt? Welche anderen

Prozesse darf das Programm starten? Ist dazu der Wechsel der Domäne nötig? Darf das Programm seine Domäne überhaupt wechseln? Diese und ähnliche Fragen geben Anregungen, welche Berechtigungen notwendig sind und welche das Programm zwar gern hätte, aber nicht braucht und folglich nicht erhält.

Dateien und Verzeichnisse klassifizieren

In Listing 1 ist ein Auszug der Dateinamen im Amanda-Paket zu sehen. Mit dieser Liste und zusätzlichen Programmkenntnissen kann der Admin die Dateien und Verzeichnisse zu Gruppen zusammenfassen und klassifizieren. Wichtig ist die Gruppierung der Dateien, um den Files später ihren richtigen Typ zuzuordnen. So lassen sich zum Beispiel folgende Einträge zusammenfassen:

```
/etc/amanda
/etc/amanda/example
/etc/amanda/example/amanda.conf
/etc/amanda/example/disklist
/var/lib/amanda/.amandahosts
```

Sie dienen der Konfiguration, folglich wäre »amanda_config_t« ein guter Name für den Typ dieser Dateien und Verzeichnisse. Um unerwünschte Änderungen zu verhindern, sollten die Files nur lesbar sein, Amanda versucht auch nicht, in diese Dateien zu schreiben. Sonst ließen sich die Files in verschiedene Typen mit unterschiedlichen Lese- und Schreibrechten einteilen. Oder man untersagt das Schreiben und unterdrückt die Logmeldungen mit »dontaudit«. Letzteres ist sinnvoll, wenn das Programm trotz des Verbots funktioniert.

Legt ein Prozess eigene Dateien oder Verzeichnisse an, können diese mit dem

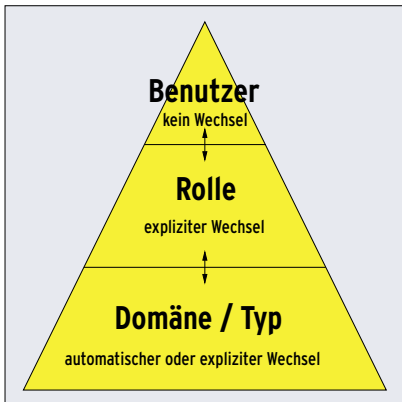


Abbildung 1: Der Benutzer als konstantes Element, er kann aber in mehreren Rollen auftreten. Domänen und Typen regeln die erlaubten Zugriffe.

»file_type_auto_trans«-Makro einen anderen Typ als das übergeordnete Verzeichnis erhalten. Andernfalls erben in SE Linux die untergeordneten Einträge die Rechte ihrer Eltern. Die Verzeichnisse unter »/var« erfordern besondere Aufmerksamkeit: Speziell die Files in »/var/log« und in »/var/run« sollten bis auf wenige Ausnahmen immer eigene Typen erhalten. Somit ist der Zugriff von fremden Domänen auf diese Dateien eingeschränkt.

Um Regeln übersichtlich zu gestalten, bietet SE Linux mehrere Mechanismen an. Der erste und einfachste sind Listen, deren Elemente in geschweiften Klammern stehen und durch Leerzeichen getrennt sind:

```
{ Element_1 Element_2 Element_X }
```

Mit Attributen und Makros stehen in der Policy auch komplexere Mechanismen zur Verfügung.

Attribute bilden Gruppen

Attribute im Sinne von SE Linux vereinigen mehrere Typen. Die Deklaration eines Typs kann ein oder mehrere Attribute enthalten. Dadurch wird dieser Typ ein Element jedes der angegebenen Attribute. Statt mühsam alle zulässigen

Typen in geschweifte Klammern zu setzen, kann der Admin auch das passende Attribut verwenden. In der Policy gibt es technisch gesehen keinen Unterschied zwischen einem Attribut und der Aufzählung von Typen in geschweiften Klammern.

Die Vorteile der Attribute lassen sich am besten an einem Beispiel erkennen. Folgendes Listing gibt der Inirc-Domäne das Recht, alle PID-Dateien zu lesen und zu löschen. Die erste Variante führt die Typen einzeln auf – gezeigt ist nur eine kleine Auswahl der Typen:

```
allow inirc_t {
    var_run_slapd_t var_run_snmpd_t
    sendmail_var_run_t var_run_named_t
    var_run_ncsd_t var_run_ntpd_t
    var_run_mysql_d_t lpd_var_run_t
}:file { getattr read unlink };
```

Mit dem geeigneten Attribut schrumpft die Regel auf eine erträgliche Länge:

```
allow inirc_t
    pidfile:file { getattr read unlink };
```

Der Administrator muss mit Hilfe der Attribute nicht mehr die ganze Policy überarbeiten, um die Berechtigungen für einen Typ zu entfernen oder hinzuzufügen, auch sind die kürzeren Regeln leichter zu lesen und zu verstehen. Eine

Übersicht über einige wichtige Attribute ist in **Tabelle 1** zu finden. Alle Attribute sind in »selinux/policy/attrib.te« detailliert beschrieben.

Entstehung von Typen

Alle Zugriffe sind über Typen geregelt. Über diese Typen werden später die Berechtigungen auf die einzelnen Objekt-klassen erlaubt. Jeder Typ muss zunächst deklariert werden:

```
type foo_t [, Attribute];
```

Die Deklaration kann optional auch Attribute für diesen Typ vergeben (durch Komma getrennt). Wie erwähnt lässt sich ein Attribut stellvertretend für alle Typen einsetzen, denen es zugewiesen wurde. Leider gibt es keine allgemein anerkannte Namenskonvention für Typen, daher ist die Benennung noch sehr uneinheitlich. Für die Datei »foo« im Verzeichnis »/var/log/«, das selbst den Typ »var_log_t« hat, existieren folgende drei Varianten:

- var_log_foo_t
- foo_log_t
- foo_var_log_t

Als Vorlage sollten bestehende Typen des gleichen oder eines ähnlichen Ver-

Listing 1: Amandas Dateiliste

```
01 /etc/amanda
02 /etc/amanda/example
03 /etc/amanda/example/amanda.conf
04 /etc/amanda/example/disklist
05 /etc/amandates
06 /etc/dumpdates
07 /usr/lib/amanda
08 /usr/lib/amanda/amandad
09 /usr/lib/amanda/amcat.awk
10 /usr/lib/amanda/amcleanupdisk
11 /usr/lib/amanda/amidxtaped
12 /usr/lib/amanda/amindexd
13 /usr/lib/amanda/amlogroll
14 /usr/lib/amanda/ampdot.awk
15 /usr/lib/amanda/ampdot.g
16 /usr/lib/amanda/ampdot.gp
17 /usr/lib/amanda/amtrmidx
18 /usr/lib/amanda/amtrmllog
19 /usr/lib/amanda/calcsiz
20 /usr/lib/amanda/chg-chio
21 ...
22 /usr/sbin/amadmin
23 /usr/sbin/amcheck
24 /usr/sbin/amcheckdb
25 /usr/sbin/amdump
26 /usr/sbin/amflush
27 ...
28 /var/lib/amanda
29 /var/lib/amanda/.amandahosts
30 /var/lib/amanda/.bashrc
31 /var/lib/amanda/.profile
32 /var/lib/amanda/DailySet1
33 /var/lib/amanda/disklist
34 /var/lib/amanda/gnutar-lists
35 /var/lib/amanda/index
```

Tabelle 1: Häufig verwendete Attribute

Attribut	Verwendung
domain	Domänentypen
file_type	Dateitypen
exec_type	ausführbare Dateien
sysadmfile	Dateitypen, auf welche die »sysadm_t«-Domäne Vollzugriff hat
pidfile	PID-Dateien
tmpfile	temporäre Dateien
privlog	Domänen, die via Socket mit Syslog kommunizieren dürfen

zeichnisses dienen. Sofern vorhanden bieten sich Makros wie »log_domain« oder »tmp_domain« als elegante Lösung an: Sie kümmern sich um die Namensgebung und erlauben gleichzeitig das Anlegen und Schreiben des neuen Typs.

Bäumchen wechsele dich

Der Wechsel zwischen Typen und Domänen ist nur möglich, wenn er zuvor in der Policy erlaubt oder als Default-Fall vorgegeben wurde. Eine »type_change«-Regel erlaubt den Wechsel, das Programm muss ihn aber explizit anfordern. Das erledigt ein neuer SE-Linux-Syscall, das Programm muss jedoch dafür angepasst sein. Eine »type_transition«-Regel sorgt dafür, dass SE Linux den Wechsel automatisch durchführt. Die allgemeine Syntax lautet:

```
type_transition Quelltyp Zieltyp : 🔧
    Objektklasse Neuer_Typ;
```

Ausgehend von dieser allgemeinen Form kommt meist eine von zwei spezielleren Varianten zum Einsatz. Für Domänenwechsel (Typwechsel von Prozessen) empfiehlt sich folgende Zeile:

```
type_transition Quelldomäne Exec-Typ : 🔧
    process Neue_Domäne;
```

Startet ein Prozess, der in der Quelldomäne läuft, ein Programm, dessen Typ dem Exec-Typ entspricht, dann führt SE Linux den neu entstehenden Prozess in der neuen Domäne aus. Beispielsweise starten die Init-RC-Skripte den SSH-Daemon. Die RC-Skripte laufen in der Domäne »initrc_t«, der SSH-Server soll aber die Domäne »sshd_t« erhalten. Die Programmdatei hat den Typ »sshd_exec_t«.

Die folgende Regel erlaubt den gewünschten Domänenwechsel – andernfalls würde der SSH-Daemon die Domäne der Init-RC-Skripte erben.

```
type_transition initrc_t sshd_exec_t : 🔧
    process sshd_t;
```

Für einen Wechsel der Dateitypen ist folgende Regel zuständig:

```
type_transition Quelldomäne Zieltyp : 🔧
    Objektklasse_für_Dateien Neuer_Typ;
```

Hier führt SE Linux bei neu angelegten Dateien einen Wechsel in den neuen Typ aus – ohne diese Regel würde die Datei den Typ des Verzeichnisses erben. Es müssen drei Bedingungen erfüllt sein:

- Der Prozess, der das File erzeugt, muss in der Quelldomäne laufen.
- Das Verzeichnis, in dem die neue Datei liegt, muss den Zieltyp haben.
- Die Datei muss die angegebene Objektklasse aufweisen.

Tabelle 2 zeigt die unter SE Linux verfügbaren Objektklassen für Dateien. Die Tabelle enthält auch Makros, die mehrere Objektklassen zusammenfassen. Es ist auch möglich, mehrere Typen oder Klassen als Listen zusammenzuführen. Komfortabler ist der Einsatz der beiden Makros »file_type_auto_trans« und »domain_auto_trans«, da diese auch gleich die typischerweise erforderlichen

Berechtigungen setzen. Beispielsweise erlaubt »domain_auto_trans«, dass der Vater sein Kind startet und mit ihm über Pipes kommuniziert. »file_type_auto_trans« erlaubt dem Prozess, die neue Datei anzulegen.

Ein Domänenwechsel startet einen neuen Prozess in einer neuen Domäne, der Typwechsel für Dateien gilt nur für neu angelegte Files. Vorhandene Dateien erhalten ihren Security Context beim Einrichten des Systems auf Basis der »*.fc«-Dateien.

Techtelmechtel der Typen

Eine »allow«-Anweisung gibt einem Quelltyp das Recht, auf den Zieltyp zuzugreifen. Das Zielobjekt muss dabei der angegebenen Objektklasse entsprechen, auch die Berechtigung ist detailliert festgelegt. Die Anweisung erhält vier Parameter, zwischen dem zweiten und dritten steht ein Doppelpunkt, am Ende ein Semikolon:

```
allow (Quelltyp|Attribut|Liste|Makro) 🔧
    (Zieltyp|Attribut|self) : 🔧
    (Objektklasse|Liste|Makro) 🔧
    (Berechtigung|Liste|Makro);
```

Das Schlüsselwort »self« eignet sich als Zieltyp, wenn er mit dem Quelltyp identisch ist. Beim Einsatz mehrere Objekt-

Tabelle 2: Objektklassen von Dateien

Einzelne Objektklassen	
Objektklasse	Beschreibung
blk_file	blockorientierte Geräte
chr_file	zeichenorientierte Geräte
dev_file	Geräte-dateien
dir	Verzeichnisse
fifo_file	Pipes
file	reguläre Dateien
filesystem	Dateisysteme
lnk_file	Verweise (Symlinks)
sock_file	Sockets
Makros für Objektklassen	
Makro	Beschreibung
dir_file_class_set	alle Verzeichnis- und Dateiklassen
file_class_set	alle Dateiklassen, ohne Verzeichnis
notdevfile_class_set	alle Dateiklassen, ohne Verzeichnisse und Gerätedateien
socket_class_set	alle Socket-Klassen
devfile_class_set	alle Gerätedateien
dgram_socket_class_set	Datagram-Socket-Klassen
stream_socket_class_set	Stream-Socket-Klassen
unpriv_socket_class_set	unprivilegierte Socket-Klassen

klassen muss man darauf achten, dass die Rechte auch für jede Klasse definiert sind. **Tabelle 2** zeigt eine Übersicht der Objektklassen, **Tabelle 3** enthält Makros für Berechtigungen. Werden mehrere »allow«-Regeln für Typen oder Klassen verwendet, kommen alle vergebenen Rechte zum Tragen.

Die folgende Allow-Regel erlaubt es dem Quelltyp »amanda_t«, Konfigurationsdateien (Zieltyp »amanda_config_t«, Objektklasse »file«) zu lesen (Liste von Berechtigungen mit »read« und »getattr«):

```
allow amanda_t amanda_config_t : 2
    file { getattr read };
```

Die Anweisung »dontaudit« ist zwar nicht das Gegenteil von »allow« – SE Linux verweigert alle nicht ausdrücklich erteilten Rechte –, sie unterbindet aber Logmeldungen über fehlende Berechtigungen und hat die gleiche Syntax wie »allow«. Ihr Einsatz sollte mit Bedacht erfolgen: Fehlende Rechte können ein Programm nutzlos machen, dann sind die Logmeldungen eine wichtige Hilfe. Unnötige Operationen lassen sich mit Hilfe von »dontaudit« aber unterbinden, ohne die Logfiles mit erwarteten Meldungen zu füllen.

Typenparade

Im nächsten Schritt gilt es, die Basisregeln in »amanda.te« zu erstellen. Am Anfang erhält Amanda eine eigene Domäne mit passenden Attributen. Diese Domäne wird mit einer oder mehreren Rollen verbunden: User müssen sich in einer autorisierten Rollen befinden, um Amanda nutzen zu dürfen.

Der Konvention folgend besteht der Typenbezeichner der Domäne aus dem Namen des Programms und dem Zusatz »_t«. Typen für ausführbare Programme enden auf »_exec_t«. Die Bezeichner der verbleibenden Typen sind frei wählbar, ihr Name sollte aber auf den Zweck schließen lassen. Im folgenden Beispiel erhält die Domäne »amanda_t« die Attribute »domain« und »privlog«:

```
type amanda_t, domain, privlog;
role system_r types amanda_t;
```

Die Rollendeklaration in der zweite Zeile ordnet die Domäne »amanda_t« der »system_r«-Rolle zu. Damit gestattet SE Linux dieser Rolle, einen Prozess in der Amanda-Domäne zu starten. Ein Benutzer oder Prozess muss sich in einer erlaubten Rolle befinden, wenn er Amanda startet, andernfalls wird der Aufruf fehlschlagen. Die Zuordnung der »amanda_t«-Domäne zur »system_r«-Rolle ist erforderlich, weil der Inetd die Prozesse startet und dieser sich als Systemdienst immer in der »system_r«-Rolle befindet. Für neue Benutzerdomänen muss die Zuordnung zur Rolle des Benutzers erfolgen.

Das »role«-Kommando stellt die Verbindung zwischen der Domäne und der Rolle her. Enthalten mehrere »role«-Kommandos die gleiche Rolle oder sind in einer Rollendeklaration mehrere Domänen angegeben, dann erlaubt SE Linux dieser Rolle den Zugang zu allen aufgeführten Domänen.

Listing 2 zeigt die restlichen Typdeklarationen für Amanda, sie betreffen die ausführbaren Programme sowie die verwendeten Dateien und Verzeichnisse. Dabei kommen die Gedanken vom Anfang des Artikels über die Programm- und Verzeichnisstruktur von Amanda zur Anwendung. Alle mit dem Dateisystem verbundenen Typen haben zusätz-

Benutzer, Rollen, Typen

Der dreischichtige Aufbau eines Security Context mit seinen Wechselbeziehungen ist in **Abbildung 1** schematisch dargestellt. Eine Konstante im Security Context ist der Benutzer, er wechselt nie seine Identität. In der zweiten Schicht befinden sich die verschiedenen Rollen, die für den Benutzer freigegeben sind. Ein Wechsel zwischen den Rollen ist möglich, wenn er auch explizit mit »newrole« erfolgen muss. Die unterste Schicht sammelt alle Domänen und Typen der darüber liegenden Rolle. Im Gegensatz zu den oberen beiden Schichten finden in ihr die Wechsel zwischen Domänen und Typen im Rahmen der Policy automatisch statt. Zusätzlich kann ein Programm den Wechsel auch ausdrücklich anfordern, es muss dafür jedoch speziell angepasst sein: Der Wechsel erfolgt über den neuen »execve_secure«-Systemaufruf.

Der Grund für den Aufwand mit Typen, deren Wechsel und den Berechtigungen untereinander liegt in der besseren Kontrolle des Admins. Die nötigen Zugriffe lassen sich so übersichtlich und zentral festlegen. Die fehlenden Berechtigungen kann ein User nicht durch Tricks und Kniffe umgehen.

Tabelle 3: Makros für Dateiberechtigungen

Makro	Beschreibung
x_file_perms	Programme ausführen
r_file_perms	Dateien lesen
rx_file_perms	Dateien lesen und ausführen
rw_file_perms	Dateien lesen und schreiben
ra_file_perms	Dateien lesen und anhängen
link_file_perms	Symbolische Links anlegen
create_file_perms	Vollzugriff auf Dateien
r_dir_perms	Verzeichnisse (nicht ihre Inhalte) und ihre Attribute lesen
rw_dir_perms	Wie »r_dir_perms«, jedoch zusätzlich Ändern von Einträgen
ra_dir_perms	Wie »r_dir_perms«, jedoch zusätzlich Hinzufügen von Dateien
create_dir_perms	Vollzugriff auf Verzeichnisse
mount_fs_perms	Verzeichnisse einbinden (mounten)

lich die Attribute »file_type« und »sysadmfile«. Die ausführbaren Typen sind mit »exec_type« gekennzeichnet und die temporären Dateien haben das Attribut »tmpfile«.

Automatischer Wechsel der Domäne

Nach der Deklaration der Typen muss das Regelwerk einige Wechsel zwischen Typen und Domänen anweisen. Am einfachsten beginnt man mit dem Domänenwechsel. Ein Makro spart Arbeit: »domain_auto_trans(Aktuelle_Domäne, Programmtyp, Neue_Domäne)« wechselt beim Start der neuen Programme automatisch die Domäne. Für Amanda, gestartet von »inetd_t«, lautet der Aufruf folgendermaßen:

```
domain_auto_trans(
    inetd_t, amanda_inetd_exec_t, amanda_t)
```

Der Erfolg dieses Makros ist in [Abbildung 2](#) zu sehen. Auch zwischen Dateitypen sind Übergänge nötig. Verschiedene Dateitypen sind in vielen Fällen sinnvoll, denn so lassen sich Daten auf einfachem Weg vor unerwünschten Zugriffen schützen. Die Unterscheidung zwischen schreibbar und nur lesbar gilt auch für Prozesse mit UID »0«.

Mit dem anderen Makro »file_type_auto_trans(Aktuelle_Domäne, Typ_des_Verzeichnisses, Neuer_Dateityp)« ist der Wechsel von Dateitypen leichter zu konfigurieren: Zusätzlich zum Typwechsel setzt es auch die Berechtigungen für die Datei passend; der aktuellen Domäne gewährt es beliebige Zugriffe (definiert

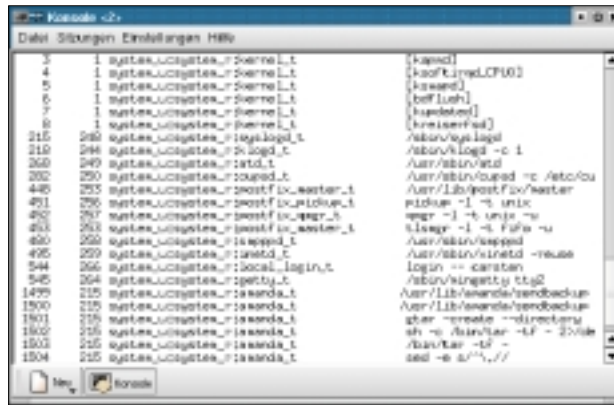


Abbildung 2: SE Linux gibt Amandas Prozessen (die letzten sechs Zeilen dieser »ps«-Ausgabe) einen eigenen Security Context.

im Makro »create_file_perms«). Auf den Amanda-Regelsatz bezogen lautet der Makro-Aufruf konkret:

```
file_type_auto_trans(
    amanda_t, tmp_t, amanda_tmp_t)
```

Für den Amanda-Client ist nur ein Typwechsel nötig, der die Verzeichnisse unterhalb von »/tmp« schützt. Alle anderen neu angelegten Dateien erben die Berechtigungen von ihren Vaterverzeichnissen und brauchen keinen expliziten Typwechsel.

Elementare Rechte

Abschließend sind noch einige elementare Berechtigungen erforderlich. Eine davon ist das Recht, die Systembibliotheken zu nutzen. Dieser Punkt war bereits ein Teil der Betrachtungen am Anfang des Projekts. Da Amanda dynamische Bibliotheken nutzt, sollte der Regelschreiber dies mit dem Makro »uses

_shlib(Domäne)« erlauben. Ein weiteres Makro kann er – so erwünscht – auch schon am Anfang einfügen: »can_network(Domäne)« gestattet der Domäne beliebige Netzwerkzugriffe. Der Schritt ist aber sehr offensiv und scheint den SE-Linux-Prinzipien zu widersprechen.

Auch für SE Linux gilt, dass die Sicherheit von der subjektiven Einschätzung des Administrators geprägt ist. Darf eine Domäne mehr, als sie minimal benötigt, weicht das zunächst die Sicherheit etwas auf. Allerdings lassen sich dadurch die Regeln oft klarer und übersichtlicher formulieren. Das wiegt in vielen Fällen den kleinen Nachteil wieder auf. Alternativ könnte der Admin die Regeln für den Netzwerkzugriff einzeln auführen und am Anfang schon grundlegende »allow«-Regeln für einen Teil der Typen einfügen, sofern er die nötigen Detailkenntnisse hat. Ein weiterer Vorteil der vordefinierte Regeln ist die geringere Anzahl von Logmeldungen, aus denen sich die restlichen Regeln ableiten lassen.

Dateiliste erstellen

Nun kann man die einzelnen Typen mit dem Dateisystem verknüpfen. Wichtig ist, dass für den ersten Testlauf die ge-

Listing 2: Amanda-Typdeklarationen

```
01 # Die Amanda-Domäne
02 type amanda_t, domain, privlog;
03
04 # Von Amanda selbst gestartete Programme:
05 type amanda_exec_t, file_type, sysadmfile, exec_type;
06 # Von Inetd gestartete Programme:
07 type amanda_inetd_exec_t, file_type, sysadmfile, exec_type;
08 # Vom User nutzbare Programme:
09 type amanda_user_exec_t, file_type, sysadmfile, exec_type;
10 # Ausführbare Skripte:
11 type amanda_script_exec_t, file_type, sysadmfile, exec_type;
12
13 # Konfigurationsdateien:
14 type amanda_config_t, file_type, sysadmfile;
15 # Shell-Konfigurationsdateien:
16 type amanda_shellconfig_t, file_type, sysadmfile;
17 # /etc/amandates:
18 type amanda_amandates_t, file_type, sysadmfile;
19 # /etc/dumpdates:
20 type amanda_dumpdates_t, file_type, sysadmfile;
21
22 # Nicht gekennzeichnete Dateien in /usr/lib:
23 type amanda_usr_lib_t, file_type, sysadmfile;
24 # Nicht gekennzeichnete Dateien in /var/lib:
25 type amanda_var_lib_t, file_type, sysadmfile;
26 # Alle Dateien in /var/lib/amanda/gnutar-lists:
27 type amanda_gnutarlists_t, file_type, sysadmfile;
28 # Alle temporären Dateien unter /tmp:
29 type amanda_tmp_t, file_type, tmpfile, sysadmfile;
30
31 # Alle von Amanda änderbaren Daten:
32 type amanda_data_t, file_type, sysadmfile;
```

starteten Programme den richtigen Security Context besitzen, da sonst die Logmeldungen von SE Linux fehlerhaft sind und es schwieriger wird, daraus fehlerfreie Regeln zu erzeugen.

Für den Start von Amanda sollten die Dateien »/usr/lib/amanda/amandad«, »/usr/lib/amanda/amidxtaped« sowie »/usr/lib/amanda/indexd« mit dem Typ »amanda_inetd_exec_t« verbunden sein. Die Reihenfolge der Einträge ist wichtig: immer vom Allgemeinen zum Speziellen, damit nicht die spezielleren die allgemeinen Einträge überschreiben und das System unerwartet reagiert. Die Kennzeichnung der verbleibenden Dateien ist in »selinux/policy/file_contexts/program/amanda.fc« zu finden.

Durch reguläre Ausdrücke lässt sich die File-Context-Datei übersichtlich halten. »(/.*)?« am Ende eines Musters bezeichnet das vorher angegebene Verzeichnis und alle ihm untergeordneten Einträge. So erhalten alle Einträge unterhalb eines Verzeichnisses den gleichen Security Context wie das Verzeichnis selbst. Hilfreich ist auch »[^/]*«, das genau eine

Tabelle 4: Mögliche Dateitypen	
Option	Beschreibung
-	reguläre Dateien
-b	blockorientierte Geräte
-c	zeichenorientierte Geräte
-d	Verzeichnisse
-l	Verweise
-p	Fifos
-s	Sockets

Verzeichnisebene beschreibt. Im folgenden Beispiel erhalten »~/.ssh« und alle untergeordneten Einträge den Security Context »system_u:object_r:user_home_ssh_t«. Verzeichnisse mit dem gleichen Namen, die aber nicht direkt im Wurzelverzeichnis eines Benutzers stehen, sind somit nicht betroffen:

```
/home/[^/]*/.ssh(/.*)? 2
system_u:object_r:user_home_ssh_t
```

Die selten verwendete mittlere Spalte bestimmt den Dateityp, auf den der Eintrag zutrifft. **Tabelle 4** führt die sieben Typen auf. Meist sind nur »-« oder »-d« nötig.

Ein Versuch macht klüger

Die ».te«-Datei muss sich im Verzeichnis »selinux/policy/domains/program/« befinden und die ».fc«-Datei in »selinux/policy/file_contexts/program/«. Dann erstellt das Kommando »make load« die Policy neu und lädt sie. Um den Security Context der Dateien neu zu erstellen, bieten sich zwei Wege an: Wenn nur wenige Einträge für die Dateien existieren, lohnt es sich, für jedes einzelne Verzeichnis »setfiles« aufzurufen. Andernfalls ist »make relabel« komfortabler, aber langsamer.

Nachdem die Policy das Programm nun rudimentär unterstützt, helfen die Logmeldungen des Systems, um die Regeln zu komplettieren und zu verfeinern. Im Permissive-Mode schreibt SE Linux alle Regelverstöße ins Syslog und erlaubt dennoch die Operationen. Im Enforcing-

Mode würde meist nur eine Fehlermeldung erscheinen, da sich das Programm aufgrund mangelnder Rechte beendet. Aus den Logs kann man halbautomatisch die fehlenden Regeln generieren. Das funktioniert sogar für Closed-Source-Programme: Das System meldet die fehlenden Rechte, daher ist kein Blick in den Quellcode notwendig.

Am einfachsten lässt sich der erste Lauf des Backupserver mit »amcheck Konfiguration« anschreiben. Das erzeugt mehrere Fehlermeldungen, ähnlich denen in **Listing 3**. Ihre Syntax erläutert der **Kasten „Aufbau der Logmeldungen“**.

Aus zwei mach eins: Weniger Regeln

Da das manuelle Schreiben der Regeln oft recht aufwändig ist, kommt das Skript »newrules.pl« aus »selinux/scripts« zum Einsatz. Mit der Option »-d« verwendet es die Einträge des Kernel-Ringpuffers. So muss der Admin die schon bearbeiteten Einträge nicht erst entfernen, die beim Nutzen des Syslog

Aufbau der Logmeldungen

Der Eintrag einer Fehlermeldung besteht aus mehreren Elementen. Die letzte Meldung aus **Listing 3** (Zeile 5) hat folgende Abschnitte:

- »Jun 25 15:39:12«: Datum und Uhrzeit der Meldung
- »re4«: Name der Meldungsquelle
- »kernel«: Kernelmeldung
- »avc: denied«: Access Vector Cache »avc« meldet, dass er einen Zugriff verweigert hat (»denied«)
- »{ fork }«: verweigerter Operation
- »pid=1324 exe=/usr/lib/amanda/amandad«: Prozess-ID und das ausgeführte Programm
- »scontext=system_u:system_r:amanda_t«: Security Context des Quelltyps
- »tcontext=system_u:system_r:amanda_t«: Der Security Context des Zieltyps
- »tclass=process«: Objektklasse des Ziels

Listing 3: Syslog-Auszüge

```
01 Jun 25 15:39:12 re4 kernel: avc: denied { search }
for pid=1324 exe=/usr/lib/amanda/amandad path=/var
dev=03:05 ino=48577 scontext=system_u:system_r:amanda_t
tcontext=system_u:object_r:var_t tclass=dir
02 Jun 25 15:39:12 re4 kernel: avc: denied { execute }
for pid=1324 exe=/usr/lib/amanda/amandad
path=/usr/lib/amanda/selfcheck dev=03:05 ino=728830
scontext=system_u:system_r:amanda_t
tcontext=system_u:object_r:amanda_exec_t tclass=file
03 Jun 25 15:39:12 re4 kernel: avc: denied { read } for
pid=1324 exe=/usr/lib/amanda/amandad
path=/var/lib/amanda/.amandahosts dev=03:05 ino=340034
scontext=system_u:system_r:amanda_t
tcontext=system_u:object_r:amanda_config_t tclass=file
04 Jun 25 15:39:12 re4 kernel: avc: denied { getattr }
for pid=1324 exe=/usr/lib/amanda/amandad
path=/var/lib/amanda/.amandahosts dev=03:05 ino=340034
scontext=system_u:system_r:amanda_t
tcontext=system_u:object_r:amanda_config_t tclass=file
05 Jun 25 15:39:12 re4 kernel: avc: denied { fork } for
pid=1324 exe=/usr/lib/amanda/amandad
scontext=system_u:system_r:amanda_t
tcontext=system_u:system_r:amanda_t tclass=process
```

mehrfach auftreten würden. »-v« ergänzt noch zusätzliche Informationen über das aufrufende Programm, das Zielobjekt und das benötigte Recht. Aus der letzten Meldung von Listing 3 (Zeile 5) generiert »newrules.pl -v« die Regel:

```
allow amanda_t amanda_t:process { fork };
#EXE=/usr/lib/amanda/amanda : fork
```

Aus diesen Regelvorschlägen muss der Administrator sinnvolle Regeln schaffen. Sei es durch Zusammenfassen oder Ignorieren (»dontaudit«) von Vorschlägen. Dabei sollte er die folgenden Punkte beachten:

- Er sollte nur sinnvolle Berechtigungen vergeben. Beispielsweise sind Schreibrechte für fremde Files oder gar Systemdateien nur selten erwünscht.
- Der Admin sollte Berechtigungen zusammenfassen und, wenn möglich, durch vorhandene Makros ersetzen (siehe Tabelle 5). »daemon_domain«, »log_domain« und »tmp_domain« leisten hier gute Dienste, da sie ihre verwendeten Typen gleich mit anlegen. Die bessere Übersichtlichkeit der Regeln und die zusätzlichen Rechte, die manche Makros vergeben, sollten sich aber die Waage halten.
- Logmeldungen für Rechte, die der Admin verweigern möchte, kann er mit »dontaudit« unterdrücken. Das schont die Logdateien und beugt dem Anschein vor, dass die Regeln fehlerhaft oder unvollständig seien.
- Zur besseren Lesbarkeit kann er bei identischen Quell- und Zieltypen Letztere durch »self« ersetzen.

Tipps und Tricks für Regeln

- Die Regel »type_transition init_t foo_t:process foo_t« dient zum Wechsel in die Domäne »foo_t«, wenn SE Linux nicht statisch im Kernel eingebunden ist, sondern zu einem späteren Zeitpunkt als Modul nachgeladen wird. Diese Regel ist optional, denn die meisten Benutzer folgen der Empfehlung und nutzen die statische Variante von SE Linux.
- Beim Hinzufügen von Typen zur Policy sollte man immer zuerst die Policy erneuern, bevor man den Security Context für die Dateien anpasst, da sonst der verwendete Typ noch unbekannt ist.
- Im Gegensatz zu SE-Linux-Kommandos werden Makros nicht mit einem Semikolon

abgeschlossen. Zwischen dem Makro-Namen und der öffnenden Klammer darf kein Leerzeichen stehen.

- Sollte trotz einer mit »newrules.pl« erzeugten Regel die gleiche Fehlermeldung erneut auftreten, kann dies an unterschiedlichen Rollen im Quell- und Zieltyp liegen.
- Startet ein Programm nicht in der erwarteten Domäne, dann helfen möglicherweise folgende Schritte: Prüfen, ob der Security Context für das Laufwerk angelegt ist. Vergleichen, ob die Angaben in »foo.fc« mit denen im Dateisystem übereinstimmen. Auch ein Blick ins Syslog kann sich dabei lohnen.

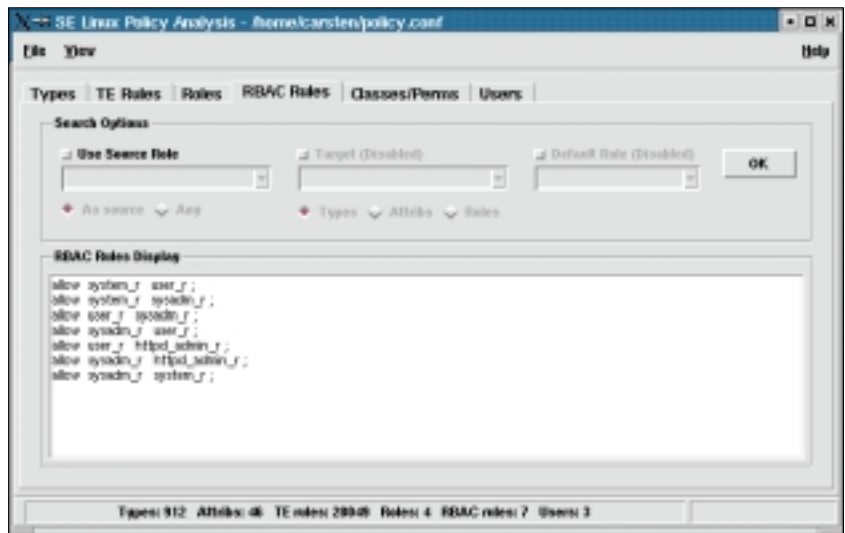


Abbildung 3: Apol, der SE-Linux-Policy-Analyser von Tresys. Die Registerkarte »RBAC Rules« (Role-Based Access Control) zeigt einen Überblick über die erlaubten Rollenwechsel.

- Teile der Regeln können entfallen, wenn der Admin Attribute wie »priv-log« nutzt.

Es gibt jedoch kein allgemein gültiges Rezept, wie man Regeln am besten erstellt. Aufschlussreich ist ein Blick in die vorhandenen Regelsätze.

Feintuning für mehr Sicherheit

In vielen Fällen ist zum Abschluss noch etwas Feintuning nötig. Die Verzeichnisse »/proc/PID/« haben den Typ der Domäne des jeweiligen Prozesses in ihrem Security Context. Sollte ein Prozess auf sein eigenes Proc-Verzeichnis zugreifen müssen, kann man das mit »allow foo_t ...« erlauben. Ein Zugriff auf die Verzeichnisse anderer Prozesse ist be-

denklich und sollte daher nur in Ausnahmefällen durch das Attribut »domain« erlaubt werden.

Die Datei »selinux/policy/assert.te« ist ein weiteres wichtiges Element von SE Linux. Ihre »neverallow«-Regeln fließen nicht direkt in die Policy ein, sie dienen nur zum Prüfen der einzelnen Anweisungen auf ihre elementare Gültigkeit. Dadurch ist eine grundlegende Sicherheit gewährleistet. So prüft das Programm »checkpolicy« mit Hilfe der Assert-Datei, dass nur die vier Domänen »kmod_t«, »insmod_t«, »kernel_t« und »ifconfig_t« die Berechtigung zum Laden von Kernelmodulen haben. Aufgrund ihrer Bedeutung sollte man diese Datei nicht ändern.

Tipps für ein Backup mit Amanda

- Für Amanda sollte man »/bin/tar« durch die SE-Linux-Variante unter »/usr/local/selinux/bin/tar« ersetzen.
- Zurzeit kann Amanda den Security Context nicht sichern. Obwohl »tar« diese Unterstützung bietet, sieht Amanda keine Möglichkeit vor, zusätzliche Schalter für »tar« anzugeben. Deshalb sollten die Dateien mit dem File Context so gestaltet sein, dass sie eine automatische Wiederherstellung des Sicherheitskontexts mit »setfiles« beziehungsweise »make-relabel« erlauben.
- Das Wiederherstellungsverzeichnis sollte vom Typ »amanda_recover_dir_t« und beschreibbar sein.
- Nicht in das originale Verzeichnis zurück-sichern, da Amanda Dateien, die nicht auf dem Backup sind, ohne Nachfrage löscht.

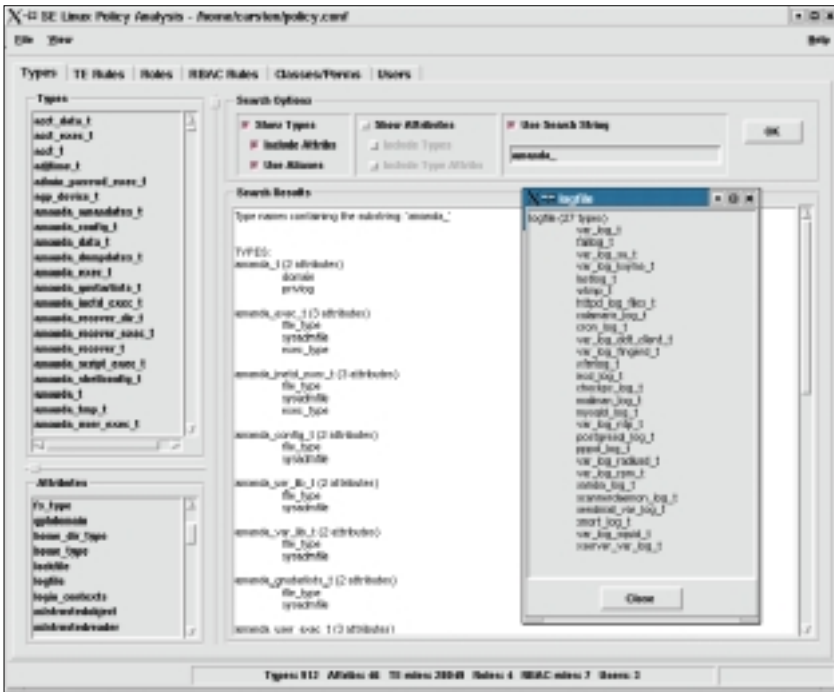


Abbildung 4: Unter »Types« zeigt Apol eine Übersicht aller Typen und Attribute; der Suchstring »amanda« grenzt die Auswahl ein. Das Logfile-Fenster enthält alle Typen, die das Attribut »logfile« verwenden.

Der M4-Präprozessor bearbeitet alle »te«-Dateien, sodass in »policy.conf« alle Makros durch ihren Inhalt ersetzt sind. Glücklicherweise sind keine speziellen Kenntnisse über M4 erforderlich, SE Linux verwendet nur die beiden Anweisungen »define« und »ifdef«. Erstere definiert Makros und Namen. Für alle Files unter »policy/domains/program« und »policy/domains/misc« werden automatisch die Dateinamen mit »define« als M4-Namen definiert. Die zweite Anweisung »ifdef« ermöglicht es, Regeln in Abhängigkeit von Typen zu verwenden, die in anderen Regelsätzen deklariert

sind. Ein gutes Beispiel für beide Anweisungen ist in »policy/domains/program/postfix.te« zu finden.

Grafische Policy-Oberfläche

Mit der Zeit wird die Policy durch ihren Umfang immer unübersichtlicher. Dem kann der Admin durch entsprechende Regeln und Makros gegensteuern. Dennoch ist es schwierig, die Regeln in ihrer Gesamtheit zu überblicken. Für Abhilfe sorgt das Programm Apol von Tresys Technology [4]. Es zeigt die komplette Policy aus der Datei »policy.conf« in

unterschiedlichen Darstellungen, aufbereitet nach verschiedenen Kriterien (Abbildung 3 und 4).

Die erste Einarbeitung in SE Linux ist nicht einfach – es gibt sehr viel zu regeln. Es sind aber gerade die mächtigen Zugriffsregeln, durch die SE Linux ein bestehendes Linux-System passgenau abdichtet. Bei näherer Betrachtung zeigt sich, dass das Regelwerk durchaus übersichtlich und verständlich ist. GUI-Werkzeuge helfen, die Auswirkung der Makros nachzuvollziehen, zudem geben die ausführlichen Logmeldungen bei der Fehlersuche wichtige Hinweise. Wer die ersten Hürden genommen hat, kommt mit SE Linux zu einem widerstandsfähigen Linux. (fjl)

Infos

- [1] Amanda-Homepage: [<http://www.amanda.org>]
- [2] Stephen Smalley, Configuring the SELinux Policy: [<http://www.nsa.gov/selinux/policy2-abs.html>]
- [3] SE-Linux-Download: [<http://www.nsa.gov/selinux/download.html>]
- [4] SE-Linux-Tools von Tresys Technology: [<http://www.tresys.com/selinux.html>]
- [5] Carsten Grohmann, Konstantin Agouros und Achim Leitner: „Regel-recht, Security Enhanced Linux im Einsatz“, Linux-Magazin 01/03, S. 38

Der Autor

Carsten Grohmann ist seit dem KC87 von Computern begeistert und arbeitet bereits seit 1997 mit Linux. Seit 2000 ist er beruflich als Systemadministrator tätig.

Tabelle 5: Best of Makros

Makro	Bedeutung
domain_auto_trans(Aktuelle_Domäne, Programmtyp, Neue_Domäne)	Wechselt beim Aufruf eines Programms mit dem angegebenen Programmtyp automatisch von der aktuellen Domäne in die neue Domäne.
file_type_auto_trans(Aktuelle_Domäne, Typ_des_Verzeichnisses, Neuer_Dateityp [, Objektklasse])	Wechselt automatisch den Typ für neue Verzeichniseinträge.
uses_shlib(Domäne)	Erlaubt der angegebenen Domäne den Zugriff auf alle dynamischen Bibliotheken.
can_exec(Domäne, Programmtyp)	Erlaubt das Ausführen von Programmen des angegebenen Typs in der übergebenen Domäne.
can_exec_any(Domäne)	Erlaubt das Ausführen von Programmen aller Programmtypen ohne Domänenwechsel.
can_network(Domäne)	Vollzugriff auf das Netzwerk.
daemon_domain(Domänenpräfix [, Attribute])	Anlegen einer Domäne für Dienste, die der Init-Prozess startet. Das Makro erzeugt Typen für die Domäne, die Programmdatei und das PID-File. Es erlaubt auch die Benutzung von dynamischen Bibliotheken und den Zugriff auf diverse Typen.
tmp_domain(Domänenpräfix)	Deklariert einen eigenen Typ für temporäre Daten unter »/tmp« und weist den Typwechsel an.
log_domain(Domänenpräfix)	Deklariert einen eigene Typ für Logdateien unter »/var/log« und weist den Typwechsel an.
Hinweis: Einige Makros erwarten nur das Domänenpräfix (ohne »_tk«) als Parameter.	