LinuX Enterprise

LinuX Enterprise

Deutschland € 5,80
Österreich € 6,90
Schweiz sFr 10,90

S&S Magazin für Professional Linux & Open Source Solutions

3.05

Auf CD

- Pyramid BenHur 2
- Intel C++ Compiler for Linux Version 8.1
- Intel Fortran for Linux Version 8.1
- Terrashop-Katalog
  Terrashop-Katalog im PDF-Format
- Kernel & Software
  Linux-Kernel 2.6.11.5
  ActiveState Perl 5.8.6.811
  Gimp 2.2.4
  Python 2.4
  Ruby 1.8.2

CPAN Perl-Module
Python-Tools
Ruby-Tools

CD-Inhalt auf Seite 3

Web Security

- Wie Sie Apache besser schützen
- Sicherheit aus dem Kernel mit SELinux
- Honeypots als Lockvögel

Boost Your Productivity

Die Zukunft von C++

Special Softwarepatente

Gefahr oder Chance?

ASP.NET mit Apache

Entwicklung von Webanwendungen unter Mono und Linux

www.linuxenterprise.de

SQLObject

Object Relational Mapping in Python

Highway to Heaven?

SCM- und CRM-Lösungen unter Linux

Editoren

VIM, Emacs und mcedit Editoren im Vergleich

Ein sicheres Linux-System mit SELinux

Sicherheit aus dem Kernel

Das Thema Sicherheit bekommt in der heutigen Zeit einen immer größeren Stellenwert, denn die Zahl der Angriffe nimmt stetig zu. Daher sind sicherheitsbewusste Administratoren immer auf der Suche nach kostengünstigen Hilfsmitteln. Ein solches ist SELinux. In diesem Beitrag erfahren Sie, worum es sich handelt, warum es entwickelt wurde sowie Grundlagen der Administration eines mit SELinux gesicherten Systems.

von Oliver Drees

Worum handelt es sich bei SELinux?

SELinux wurde von der amerikanischen Behörde NSA in Zusammenarbeit mit anderen entwickelt. Dabei ging es in der ersten Linie darum, aufzuzeigen, dass ein hohes Maß an Sicherheit mit einem Mainstream-Betriebssystem zu erreichen ist. Dass die Wahl auf Linux fiel, liegt einerseits an der wachsenden Bedeutung von Linux und andererseits an dem frei verfügbaren Quellcode. Die von der NSA entwickelten Erweiterungen sind wiederum frei verfügbar. Es wurde eine Architektur entwickelt, die eine Zugriffskontrolle ermöglicht und im Subsystem des Kernels integriert wurde. Der Kernel wurde lediglich erweitert, es wurden keine Fehler behoben. Es können verschiedene Arten von Regeln für die Zugriffskontrolle verwendet werden: die Konzepte der Typen-Erzwingung (Type Enforcement, TE), rollenbasierter Zugriffskontrolle (Role Base Access Control, RBAC) und mehrstufiger Sicherheit (Multi-Level Security, MLS). Bei letzterem Konzept handelt es sich um experimentelle Erweiterungen, die in diesem Artikel nicht weiter betrachtet werden. Die Arbeit sollte nicht als vollständige Sicherheitslösung angesehen werden, sondern vielmehr als ein Baustein.

Quellcode

Der Quellcode zum Artikel befindet sich auf der beiliegenden CD.

Es kann bestimmt werden, welche Subjekte (Programme oder Benutzer) auf welche Objekte (Dateien oder Devices) zugreifen dürfen. Bekommt ein Angreifer über Fehler in einem Prozess Zugriff auf das System, so kann er nur auf die Daten und Programme zugreifen, für die der Ursprungsprozess berechtigt war. In einem normalen Linux-System hätte er Zugriff auf alle Daten, die der Benutzer des Ursprungsprozesses öffnen darf. Im Extremfall sind das bei root-Rechten alle Dateien. Durch die Einschränkungen aus SELinux ergibt sich eine erhöhte Sicherheit, denn auch root können Rechte verwehrt werden.

Welche Begriffe werden verwendet?

Zum Ersten wird der Begriff Identität verwendet. Wenngleich die Benutzer-ID im normalen Linux-System gleich der SELinux-Identität lauten kann, handelt es sich um unterschiedliche Dinge. Die SELinux-Identität ist ein Teil des sog. Security-Kontexts. Das Kommando *id -Z* zeigt Kontext eines Benutzers, z.B.: *oliver:user_r:user_t* Auch Dateien verfügen über einen Security-Kontext. Anzeigen lassen können Sie es sich mit *ls -Z*:

```
-rw-r--r-- root  root system_u:object_r:etc_t   /etc/passwd
```

Die Identität ist der erste Teil des Kontexts. Über sie wird festgelegt, welche Rollen und Domänen verwendet werden dürfen. Wurde der Identität das Recht gewährt, die Rolle zu ändern, sieht man in der Ausgabe von *id*, dass die Identität gleich geblieben ist, sich aber der Rest des Kontexts verändert hat (vergl. Listing 1).

Zum Zweiten kennen wir den Begriff der Domänen (Domains). Jeder Prozess läuft in einer Domäne, die bestimmt, welche Zugriffe ein Prozess machen und welche Aktionen er ausführen darf. Wird beispielsweise über eine Shell versucht, ein

Listing 1

Security Context nach newrole

```
[oliver@fedora3 ~]$ id -Z
oliver:staff_r:staff_t
[oliver@fedora3 ~]$ newrole -r sysadm_r
Authenticating oliver.
Password:
[oliver@fedora3 ~]$ id -Z
oliver:sysadm_r:sysadm_t
```

Programm mit root-Rechten auszuführen, muss der Benutzer über seine Rolle zum Ausführen des Programms berechtigt sein, denn die Shell läuft im Kontext des Benutzers. Die Verwendung von zum Beispiel *setuid*-root-Programmen wird dadurch wirksam eingeschränkt. In den Regeln wird dem Namen der Domain i.d.R. ein „_t“ angehängt. Beispiele sind *sysadm_t* (Systemadministration) und *user_t* (unprivilegierte Benutzer).

Der dritte Begriff des Typs (Type) ist in etwa gleich der Domäne. Der Unterschied besteht darin, dass hierbei Objekte angesprochen werden, so zum Beispiel Verzeichnisse, Dateien, Sockets, etc. Auch dem Typ-Namen wird normalerweise ein „_t“ angehängt. Die Lesbarkeit leidet jedoch, denn nicht immer ist klar, ob von einer Domain oder einem Typ gesprochen wird. Dateien im */proc*-Zweig haben dabei eine Sonderrolle, denn wenngleich sie für laufende Prozesse angelegt werden, gelten sie als Dateien und man spricht bei ihnen von Typen. Der dazugehörige Prozess läuft unter der gleich lautenden Domäne. Eine Rolle bestimmt, welche Domains ein Benutzer verwenden darf. Nur bei korrekter Definition einer Rolle in der Policy-Konfiguration darf ein Wechsel in die betreffende Domäne stattfinden. Den Security-Kontext haben wir oben bereits kurz angesprochen. Er kennt alle Attribute, die mit Elementen wie Dateien oder Prozessen verbunden sind. Er besteht aus der Identität, Rolle und Domäne bzw. Typ.

Beim Wechsel in eine andere Domain (Transition) unterscheidet man zwei Arten. Einerseits den Wechsel von Prozess-Domains, der beim Ausführen eines Prozesses stattfinden kann, und andererseits den Wechsel von Datei-Typen, der beim Anlegen von Dateien in einem bestimmten Verzeichnis stattfindet. Der Security-Kontext von Dateien wird per Default gleich dem Kontext des darüber liegenden Verzeichnisses. Legt ein Benutzer jedoch eine Datei im */tmp*-Ordner an, findet in der Fedora-Konfiguration ein Wechsel der Domain nach *tmp_t* statt. Beim Login per SSH passiert etwas Ähnliches, denn es findet ein Wechsel auf die Domain statt, die für die Benutzer-Shell festgelegt wurde. Bei Policies handelt es sich um einen Satz Regeln, die etwas bestimmen, zum Beispiel wer auf was zugreifen darf.

Wie installiere ich SELinux?

Eine Installationsanleitung würde den Rahmen dieses Beitrags sprengen, daher skizziere ich das hier lediglich. Die Kernel-Erweiterungen für das Linux-Security-Modul (LSM) und SELinux-Module sind seit Version 2.6.0-test3 im Standard-Kernel integriert. Falls notwendig, stehen über die Seiten der NSA Interims-Patches für die Fälle zur Verfügung, die noch nicht in den Kernel-Stream geflossen sind und dann manuell in den Kernel gepatcht werden müssen. Darüber hinaus sind noch die SELinux-Tools und eine Reihe Daemonen zu kompilieren. Abschließend müssen für alle Programme, die nicht in der Beispiel-Konfiguration enthalten sind, die Security-Kontexte ermittelt und konfiguriert werden. Ein ziemlich mühseliges Unterfangen.

In meine Augen greift man besser auf eine Distribution zurück, bei der SELinux bereits implementiert ist. Konkrete Hinweise zu verschiedenen Distributionen finden Sie über das SELinux-Projekt unter [2]. Empfehlenswert ist Fedora Core 3 [3]. Über das Sourceforge-Projekt werden entsprechend angepasste Pakete direkt bereitgestellt. Alle weiteren Pfadangaben in diesem Beitrag beziehen sich auf diese Distribution. Ansonsten können die Source-RPM-Pakete aber auch als Vorlage für Anpassungen an die eigene Distribution verwendet werden.

Verwenden Sie einen selbst übersetzten Kernel, müssen in der Kernel-Konfiguration unter FILESYSTEMS bei Ext2 und Ext3 die EXTENDED ATTRIBUTES und SECURITY LABEL-Optionen gesetzt werden. Unter FILESYSTEMS->PSEUDO FILESYSTEMS müssen die EXTENDED ATTRIBUTES- und SECURITY LABELS-Optionen für */dev/pts* aktiviert sein.

Daneben müssen unter SECURITY die folgenden Optionen gesetzt sein:

- ENABLE DIFFERENT SECURITY MODELS
- SOCKET AND NETWORKING SECURITY MODELS
- CAPABILITIES SUPPORT
- NSA SELINUX SUPPORT
- NSA SELINUX DEVELOPMENT SUPPORT (optional)
- NSA SELINUX BOOT PARAMETER (optional)

Wird die Option für die Boot-Parameter gesetzt, kann beim Systemstart über die Boot-Option *selinux=0* das System als War-

Verzeichnis	Beschreibung
appconfig	Standard-Typen und Security-Kontexte für verschiedene spezielle Umstände
domains	Domains für das Type-Enforcement
file_contexts	Security-Kontexte für persistente Dateien
flask	Symbole, die vom SELinux-Kernel verwendet werden
macros	*m4*-Makros, die in den Quell-Dateien verwendet werden können
tmp	verwendet während der Erstellung der einzelnen *policy.conf*-Datei
tunables	verschiedene Definitionen, die das System durch Aktivieren bzw. Deaktivieren von Booleans anpassen
types	generelle Typen, die nicht mit einer speziellen Domain verbunden sind

Tabelle 1: Verzeichnisstruktur der Policy-Quellen

Listing 2

Beispiel *file_contexts/program/dhcpd.fc*

```
1 # dhcpd
2 /etc/dhcpd.conf    --    system_u:object_
                                    r:dhcp_etc_t
3 /etc/dhcp3(/.*)?        system_u:object_
                                    r:dhcp_etc_t
4 /usr/sbin/dhcpd.*    --    system_u:object_
                                  r:dhcpd_exec_t
5 /var/lib/dhcp(3)?/dhcpd\.leases.* --
                    system_u:object_r:dhcpd_state_t
6 /var/run/dhcpd\.pid  -d    system_u:object_
                                 r:dhcpd_var_run_t
7 ifdef(`dhcp_defined', `', `
8 /var/lib/dhcp(3)?    -d    system_u:object_
                                    r:dhcp_state_t
9 define('dhcp_defined')
10')
```

tungssystem ohne ein aktives SELinux gestartet werden. Falls das nicht gewünscht ist, sollte ein separater Wartungs-Kernel mit den genannten Einstellungen für die Dateisysteme erstellt werden.

Wie wird das Ganze konfiguriert?

Die Kernel-Erweiterung arbeitet mit Policies in einem Binärformat. Wenngleich für die notwendige Übersetzung nur eine einzige Datei (*policy.conf*) verwendet wird, erfolgt die Konfiguration in einer Reihe separater Dateien. Die Quellen kommen aus den Paketen „selinux-policy-targeted-sources“ bzw. „selinux-policy-strict-sources“ und liegen unter */etc/selinux/[targeted|strict]/src/policy*. Bei „targeted“, sprich gezielt, werden nur die Programme von SELinux überwacht, die entsprechend konfiguriert wurden. Die meisten Prozesse werden dabei nicht überwacht. Im anderen Fall werden strikt alle Programme betrachtet. Bei der Übersetzung hilft der *m4*-Präprozessor, der auch für die *sendmail*-Konfiguration verwendet wird. Er hilft dabei, aus den einzelnen Dateien die Datei *policy.conf* zu erstellen, die dann in das Binärformat übersetzt wird. Die Übersichtlichkeit wird durch die vielen Dateien nach einer Eingewöhnungszeit durch die in Tabelle 1 dargestellte Struktur gegenüber einer einzelnen Datei deutlich erhöht.

In der Anfangsphase empfiehlt es sich, einen Kernel zu verwenden, der mit NSA SELinux Development Support übersetzt wurde. Dadurch ist es möglich, zur Laufzeit zwischen Warn-Modus (*echo 0 > /selinux/enforcing*) und „richtiger“ Prüfung (*echo 1 > /selinux/enforcing*) umzuschalten.

Um das System mit einem solchen Kernel beim Start direkt den Einschränkungen zu unterwerfen, muss noch der Eintrag *SELINUX=enforcing* in der Datei */etc/selinux/config* gesetzt werden. Möglich sind hier „permissive“ (es werden Policy-Verletzungen protokolliert), „enforcing“ (die „richtige“ Prüfung inkl. Abweisung bei Policy-Verletzungen findet statt) oder „disabled“ (es findet keine Prüfung und auch keine Protokollierung statt). In dieser Datei legen Sie auch fest, ob eine strikte Prüfung oder eine gezielte Überwachung stattfinden soll.

Bei den nachfolgenden Pfadangaben gehe ich davon aus, dass sie wie bei Fedora Core 3 unter */etc/selinux/[strict|targeted]/src/policy* liegen. Über die Datei *users* legen Sie fest, welche Benutzer vom System erkannt und in welchen Rollen sie agieren dürfen. Stellen Sie dabei sicher, dass mindestens ein Benutzer in einer Rolle für die Systemadministration (*staff_r*, *sysadm_t* und *system_r*) autorisiert wird. Alle anderen Benutzer sollten Sie der normalen Benutzerrolle (*user_r*) zuordnen, bzw. keine Angaben für diese machen. Meldet sich ein Benutzer am System an, für den es keinen Eintrag in dieser Datei gibt, so wird er automatisch unter dem Kontext *username:user_r:user_t* angemeldet. Falls Sie das nicht wünschen, entfernen Sie den Benutzer *user_u*. Entfernen Sie alle Beispielbenutzer und verändern Sie auf keinen Fall den

Listing 3

Beispiel *domains/program/dhcpd.te*

```
10 daemon_domain(dhcpd)
11
12 allow dhcpd_t dhcpd_port_t:udp_socket name_bind;
13
15 ifdef('pxe.te', `
16 type pxe_port_t, port_type;
17 ')
18 allow dhcpd_t pxe_port_t:udp_socket name_bind;
19
20 type dhcp_etc_t, file_type, sysadmfile, usercanread;
21 typealias dhcp_etc_t alias { etc_dhcp_t etc_dhcpc_
                                t etc_dhcpd_t };
22
24 can_network(dhcpd_t)
25 can_ypbind(dhcpd_t)
26 allow dhcpd_t self:unix_dgram_socket create_
                                socket_perms;
27 allow dhcpd_t self:unix_stream_socket create_
                                socket_perms;
28
29 allow dhcpd_t var_lib_t:dir search;
30
31 allow dhcpd_t devtty_t:chr_file { read write };
32
34 allow dhcpd_t self:capability { net_raw net_bind_
                                service };
35
37 type dhcp_state_t, file_type, sysadmfile;
38 type dhcpd_state_t, file_type, sysadmfile;
39 allow dhcpd_t dhcp_etc_t:file { read getattr };
40 allow dhcpd_t dhcp_etc_t:dir search;
41 file_type_auto_trans(dhcpd_t, dhcp_state_t,
                                dhcpd_state_t, file)
42
43 allow dhcpd_t etc_t:lnk_file read;
44 allow dhcpd_t { etc_t etc_runtime_t }:file r_file_
                                perms;
45
47 can_exec(dhcpd_t, { dhcpd_exec_t bin_t })
48
50 allow dhcpd_t self:packet_socket create_socket_
                                perms;
51 allow dhcpd_t self:rawip_socket create_socket_
                                perms;
52
54 allow dhcpd_t { bin_t sbin_t }:dir r_dir_perms;
55 allow dhcpd_t { bin_t sbin_t }:{ file lnk_file } rx_
                                file_perms;
56 allow dhcpd_t self:fifo_file { read write getattr };
57
59 allow dhcpd_t proc_t:{ file lnk_file } r_file_perms;
60 tmp_domain(dhcpd)
```

Listing 4

Makro daemon_domain aus *macros/global_macros.te*

```
385 define(`daemon_domain', `
386 daemon_base_domain($1, `$2', $3)
387 # Create pid file.
388 allow $1_t var_t:dir { getattr search };
389 var_run_domain($1)
390
391 allow $1_t devtty_t:chr_file rw_file_perms;
392
393 # for daemons that look at /root on startup
394 dontaudit $1_t sysadm_home_dir_t:dir search;
395
396 # for df
397 allow $1_t fs_type:filesystem getattr;
398 allow $1_t removable_t:filesystem getattr;
399
400 read_locale($1_t)
401
402 # for localization
403 allow $1_t lib_t:file { getattr read };
404 ')dnl end daemon_domain macro
```

Listing 5

Beispiel Syslog-Eintrag bei Regelverstößen

```
avc: denied { getattr } for pid=11129
exe=/usr/sbin/sendmail.sendmail path=/etc/hosts
    dev=dm-0 ino=405078 scontext=system_u:system_
        r:system_mail_t tcontext=system_u:object_
                r:unlabeled_t tclass=file
```

Benutzer *system_u*, dem eine besondere Bedeutung zukommt!

Generell werden die Policies, die eine Domain bilden, in zwei Dateien gruppiert. Im Verzeichnis *filecontexts/program* mit der Dateieendung *.fc* werden die Security-Kontexte der Dateien und Verzeichnisse der Domain festgelegt. Ein Beispiel zeigt Listing 2. Dateien im Verzeichnis *domains/program* legen dagegen die Zugriffsregeln und Übergänge einer Domain fest (vergl. Listing 3). Dabei sind in der *.te*-Datei in der Regel mehr Einträge als in der *.fc*-Datei. Sie finden daneben aber auch noch eine Anzahl anderer Dateitypen, die wir an dieser Stelle nicht weiter betrachten. Die meiste Arbeit steckt in den beiden genannten Dateitypen.

Die erste Spalte einer Zeile in einer *.fc*-Datei enthält einen regulären Ausdruck. Dateien und Verzeichnisse, die diesem Ausdruck entsprechen, werden mit den Informationen der anderen Spalten gelabelt, d.h. in die erweiterten Attribute der Dateisysteme werden die Informationen zum Security-Kontext einer Datei abgelegt. Die Meta-Zeichen entsprechen denen des vi-Editors und anderer Linux-Programme, die solche Ausdrücke verwenden. Die Klammern bedeuten eine Gruppierung und das Fragezeichen bedeutet, dass die vorangegangene Gruppierung bzw. das vorangegangene Zeichen optional ist. Bei einem Stern bedeutet das, dass diese beliebig oft vorhanden sein können (0 bis n-mal). Ein Punkt steht für ein beliebiges Zeichen. Der Schrägstrich steht für die Trennung zwischen Pfadangaben. Kommentare verbergen sich hinter dem #-Zeichen. Der Ausdruck */etc/dhcpd3(/.*)?* beschreibt folglich das Verzeichnis */etc/dhcpd3* sowie alle darin vorhandenen Dateien, sofern es welche geben sollte. In der zweiten Spalte stehen die sog. Flags, die bestimmen, ob Verzeichnisse (-d), Dateien (--) oder Verzeichnisse oder Dateien (zwei Leerzeichen) vom Ausdruck erfasst werden sollen. In der letzten Spalte steht der Security-Kontext, mit dem die Dateien und Verzeichnisse gelabelt werden sollen. In diesen Dateien werden keine anderen Zeilen verwendet.

Die *.te*-Datei ist etwas komplizierter. Es können sich darin Type-Zeilen (definiert einen Typ oder eine Domain), Allow-Zeilen (definieren Zugriffsregeln) und andere Zeilen (für Macro-Aufrufe oder Alias-Definitionen) befinden. In Zeile 10 des Beispiels in Listing 3 wird das Macro „daemon_domain“ aufgerufen. Diese Makro-Definition finden Sie in Listing 4. Wie Sie dort sehen können, werden weitere Makros verwendet. In der fertigen *policy.conf*-Datei werden alle Makros expandiert. So wird beispielsweise aus der Zeile 388

TE-Deklaration	Syntax	Beschreibung
type	type Identifikation [alias Namen] [, Attribute] Bsp.: type ping_t, domain, privlog;	Deklaration eines Typs bzw. einer Domain
typealias	typealias Identifikation Alias Name; Bsp.: typealias cupsd_etc_t alias etc_cupsd_t;	Definitionen eines Typ-Alias
attrib	attrib Identifikation; attrib admin	Definition eines Typ-Attributs, das für einen oder mehrere Typen verwendet werden kann
allow	allow Quell-Typ Ziel-Typ(en):Objekt-Klasse(n) Genehmigungen	Operation wird erlaubt.
auditallow	auditallow Quell-Typ Ziel-Typ(en):Objekt-Klasse(n) Genehmigungen	Operation wird erlaubt aber bei Auftreten protokolliert.
auditdeny	auditdeny Quell-Typ Ziel-Typ(en):Objekt-Klasse(n) Genehmigungen	Operation wird verboten und protokolliert. Wird nicht wirklich benötigt, da im Standard nicht autorisierte Aktionen verboten und protokolliert werden.
dontaudit	dontaudit Quell-Typ Ziel-Typ(en):Objekt-Klasse(n) Genehmigungen	Operation wird verboten aber nicht protokolliert.
neverallow	neverallow Quell-Typ Ziel-Typ(en):Objekt-Klasse(n) Genehmigungen	Erzwingt Beschränkungen auf die Policy selber. Das dient dazu, sicherzustellen, dass mit fehlerhaften Änderungen die Policy aufgeweicht wird. Selbst allow-Regeln können daran nichts ändern!

Tabelle 2: Deklarationen für das Type-Enforcement

```
allow dhcpd_t var_t:dir { getattr search };
```

Das „$1“ aus der Makro-Definition wird durch den ersten dem Makro übergebenen Parameter ersetzt. Der zweite Parameter, durch Komma getrennt, ist dann „$2“, etc.

Zeilen 15 bis 17 besagen, dass falls die Datei *pxe.te* existieren sollte, der Typ *pxe_port_t* mit dem Attribut *port_type* angelegt wird. Auf Attribute komme ich gleich zu sprechen. Die „allow“-Zeilen, wie beispielsweise in Zeile 27, legen fest, was der Domain erlaubt wird. In diesem Fall darf die Domain *dhcpd_t* verschiedene Lese- und Schreibzugriffe auf Sockets ausführen, die für dieselbe Domain gelabelt wurden. Die allgemeine Syntax für eine allow-Zeile lautet:

```
allow Domain|Attribut Domain_des_Ziels:Objekt-Klassen
                                        Genehmigungen
```

Typen-Attribute können in den Domain-Konfigurationsdateien auf die gleiche Weise verwendet werden wir normale Typen-Namen. Wenngleich Typen-Namen normalerweise nur in einer Domain verwendet werden, werden Typ-Attribute in mehreren verwendet. Attribute bilden einen Zugriffs-Vektor mit Regeln, der an einen Typ gebunden werden kann. Das bedeutet, dass alle Domains mit einem bestimmten Attribut diese Regeln gemeinsam haben.

Genehmigungen bestimmen, was den Objekt-Klassen ermöglicht werden soll. Bei den Objekt-Klassen handelt es sich um Gruppierungen von Objekten wie alle Character Devicefiles. Durch diese Kombination wird genau festgelegt, wie eine Domain auf Objekte zugreifen darf. Für Zeile 31 bedeutet das, dass es der *dhcpd_t*-Domain erlaubt wird, auf Character Devices

Zeichen	Bedeutung
*	alle zugehörigen Typen, Klassen oder Genehmigungen
~	Negation (bspw. nicht Typ Domain = ~domain)
-	Subtraktion
self	Ziel ist das Gleiche wie der Quell-Typ.

Tabelle 3: Zeichen für Typen, Klassen und Genehmigungen

lesend und schreibend zuzugreifen, die für die Domain *devtty_t* gelabelt wurden. Wie Sie an diesem Beispiel sehen können, werden Mehrfachzuordnungen durch geschweifte Klammern eingeschlossen. Das trifft für die meisten Teile der Konfiguration (vergl. Zeilen 21, 44 und 55) zu.

Zeile 21 zeigt eine Alias-Deklaration. In diesem Fall werden die Namen in geschweiften Klammern alternative Namen für den Typ *dhcp_etc_t*. Die Namen können mit gleicher Bedeutung an allen Stellen verwendet werden, bei denen ein Typ vorkommen kann.

Erzeugung einer neuen Binär-Datei der Policies

Verschieben Sie alle *.te*-Dateien für die Dienste und Anwendungen, die auf dem System betrieben werden, aus dem Verzeichnis *domains/program/unused* in das darüber liegende. Alles aus dem „unused"-Verzeichnis erscheint nicht in der übersetzten Policy!

Nach jeder Änderung an einer der Konfigurations-Dateien müssen Sie dafür sorgen, dass eine neue Binär-Datei der Policies erzeugt wird. Das erreichen Sie, indem Sie das *make*-Kommando im Verzeichnis */etc/selinux/[strict|targeted]/src/policy* mit einem der in Tabelle 4 genannten Targets ausführen, z.B. *make reload*.

Wird versucht, auf nicht erlaubte Weise auf ein Objekt zuzugreifen, wird ein entsprechender Eintrag im Syslog erzeugt (vergl. Listing 5). Aus dem Beispiel wird ersichtlich, dass das Programm *sendmail* (exe) davon abgehalten wurde, die Attribute (*{ getattr }*) für die Datei */etc/hosts (path)* zu ermitteln. Daneben wird noch der Quell-Kontext des ausführbaren Programms (*scontext*), der Ziel-Kontext (*tcontext*) sowie die Objekt-Klasse (*tclass*) ausgegeben.

Welche Hilfsprogramme gibt es?

Bevor selbst root wichtige Aktionen ausführen kann, muss er u.U. zunächst seine Rolle wechseln. Dies geschieht mit Hilfe des Kommandos *newrole*. Mit *newrole -r sysadm_r* wird die Rolle auf *sysadm_r* gewechselt, sodass Änderungen an System-Dateien gemacht werden können. Um sicherzustellen, dass Daemonen, die normalerweise über den init-Prozess gestartet werden, auch richtig arbeiten, müssen die init-Skripte mit dem Kommando *run_init Skriptname [Argumente]* gestartet werden. Mit *runcon Kontext Dateiname* starten Sie Kommandos in einem anderen als Ihrem Kontext.

Den aktuellen Security-Kontext eines Benutzers zeigt das Kommando *id -Z*. Dabei handelt es sich um das von der NSA gepatchte *id*. Analog dazu zeigt *ls -Z*, mit welchem Kontext die betreffende Datei gelabelt wurde. *ps -Z* zeigt schließlich zu den angezeigten Prozessen den jeweiligen Kontext.

Mit Hilfe des Kommandos *make relabel* im Verzeichnis der SELinux-Policy-Quelldateien werden alle Dateisysteme gemäß den Vorgaben gelabelt. Über das Kommando *chcon* werden einzelne Dateien oder Verzeichnisse mit dem angegebenen Security-Kontext gelabelt. Dabei kann auch der Kontext einer Datei-Referenz verwendet werden.

Make Target	Übersetzung der Policy	Installation der Policy	Laden oder Nachladen der Policy
policy	ja	nein	nein
install	ja	ja	nein
load	ja	ja	ja
reload	ja	ja	ja
relabel	nein	nein	nein
Bemerkungen	Syntax-Prüfung und Prüfung, dass keine Regelzwänge verletzt werden	Erstellen der binären Policy	Aktuell arbeiten beide gleich: Laden der binären Policy in den Kernel. Die neue Policy wird sofort verwendet.

Tabelle 4: Targets für das make-Kommando

Das *fixfiles*-Kommando labelt alle verfügbaren Dateisysteme gemäß den Vorgaben in den Standard-Konfigurationsdateien. *restorecon Pfadname* labelt dagegen ein oder mehrere Dateien gemäß den Standard-Konfigurationsdateien. Der Unterschied bei *setfiles* besteht darin, dass eine andere als die Standard-Konfigurationsdatei für das Labeln verwendet werden kann.

Das Programm *audit2allow* liest die Log-Einträge aus der Datei */var/log/messages* oder auch aus einer anderen Datei und erstellt daraus Regeln, die diese Fehler beheben würden. Man sollte sich aber nicht hinreißen lassen, diese Regeln ohne Hinterfragen zu übernehmen! Dadurch könnten Regeln, die bewusst protokolliert werden, auch außer Kraft gesetzt werden.

Fazit

Bevor ein SELinux-System sicher aufgesetzt werden kann, ist es notwendig, die Beispielregeln zu verstehen. Das Erstellen eigener Regeln für spezielle Programme ist ein schrittweiser Prozess: Regeln erstellen, in den Kernel laden, testen, Logfiles betrachten, Fehler beheben und wieder von vorne. Selbst für die strikte Prüfung bei Fedora Core 3 müssen noch einige Fehler behoben werden. Wegen der Komplexität eignet sich das System in erster Linie für Server, die Aufgaben bündeln und einen erhöhten Anspruch an die Systemsicherheit stellen.

In diesem Beitrag konnte ich Ihnen lediglich die Grundlagen aufzeigen. Als weiterführendes Buch empfehle ich [4]. Es ist allerdings nur in englischer Sprache verfügbar. Die Erklärungen sind dennoch sehr gut verständlich und gerade die Übersichten im Anhang zeigen kurz und knapp alle wichtigen Objektklassen, Operationen, Makros, Typen und Typ-Attribute. Bei [5] gibt es grafische Open-Source-Tools, die beim Erstellen der Policies unterstützen.

Links & Literatur

[1] SELinux-Homepage: www.nsa.gov/selinux

[2] SELinux-Sourceforge-Projekt: selinux.sourceforge.net

[3] Fedora-Homepage: fedora.redhat.com

[4] Bill McCarty: SELinux – NSA's Open Source Security Enhanced Linux. O'Reilly Verlag

[5] Tresys-Homepage: www.tresys.com