



12 SELinux-Grundlagen

12.1 Was ist SELinux?

SELinux ist ein Mandatory-Access-Control-System (MAC). Es wird von dem Linux-Kernel zusätzlich zum Discretionary-Access-Control-System (DAC) verwendet, um den Zugriff auf eine Ressource zu gewähren oder zu verweigern. Während das Linux-DAC-System lediglich die Identität des Benutzers oder Prozesses und die Rechte der Datei auswertet, nutzt SELinux einen Security-Context. Dieser *Security-Context* besteht sowohl für den Prozess als auch für das angesprochene Objekt aus Benutzer, Rolle und Typ. Der Zugriff wird dann über spezielle Regeln erlaubt. Jedes Subjekt (Benutzer, Prozess) und jedes Objekt (Datei, Verzeichnis, Socket) erhält einen Security-Context.

12.2 Der Security-Context: SELinux-Benutzer, -Rollen und -Typen

SELinux unterscheidet Benutzer (*user*), Rollen (*role*) und Typen (*type*). Jedes Subjekt und jedes Objekt erhält einen Security-Context aus diesen drei Informationen.

- *SELinux User*: Der SELinux-Benutzer ist nicht identisch mit dem Linux-Benutzer. SELinux verwendet eine eigene Benutzerdatenbank. Mehrere Linux-Benutzer können denselben SELinux-Benutzer verwenden. Daher sind die doppelten Benutzerdatenbanken nicht besonders störend. Häufig existieren auf einem SELinux-System nur wenige SELinux-Benutzer.
- *SELinux Role*: Jeder SELinux-Benutzer hat Zugriff auf mindestens eine Rolle. Stehen mehrere Rollen zur Verfügung, kann der Benutzer zwischen diesen Rollen mit dem Befehl `newrole` (siehe 18.15) wechseln. Die unterschiedlichen Rollen erlauben dann unterschiedlichen Zugriff auf das System.
- *SELinux Type*: Dies ist das wesentliche SELinux-Attribut. Jedes Subjekt und jedes Objekt erhält einen Typ. Da SELinux im Wesentlichen auf *Type-Enforcement*-Regeln basiert, definiert dieses Attribut, ob ein Zugriff erlaubt ist oder nicht. Mehrere gleichartige Subjekte und Objekte lassen sich über dieses Attribut auch gruppieren und so mit gleichen Berechtigungen ausstatten. Bei Subjekten spricht man übrigens nicht von dem Typ, sondern von der Domäne. Technisch handelt es sich aber um ein und dasselbe. Die Unterscheidung ist nur sprachlicher Natur. Da diese sprachliche Unterscheidung aber durchaus zur Klarheit beiträgt, werde ich im Weiteren auch bei Prozessen von der *Domäne* sprechen.

Ein modernes SELinux-System definiert üblicherweise nur wenige Rollen und Benutzer, während es mehrere 100 unterschiedliche Typen verwendet. Die Benutzer und die Rollen haben auch nur sehr geringe Auswirkungen auf die SELinux-Policy. Im Moment können wir sie fast vernachlässigen.

An der Schreibweise können die verschiedenen Parameter unterschieden werden. Benutzer bestehen entweder aus einem einfachen Benutzernamen wie *root* oder erhalten ein Suffix *_u*: *user_u*. Rollen werden mit dem Suffix *_r* versehen: *sysadm_r* und Typen erhalten das Suffix *_t*: *httpd_t*.

Zusammen erzeugen diese Attribute den Security-Context, den Sie sich im Dateisystem mit dem Befehl `ls` ansehen können:

```
[root@supergrobi ~]# ls -Z /etc/shadow /etc/passwd
/home/student/.bash_profile
-rw-r--r-- root root system_u:object_r:etc_t
/etc/passwd
-r----- root root system_u:object_r:shadow_t
/etc/shadow
-rw-r--r-- student student user_u:object_r:user_home_t
/home/student/.bash_profile
```



Hinweis

SELinux benötigt für den Betrieb ein angepasstes Linux-Betriebssystem. Es genügt nicht, über einen entsprechenden Kernel zu verfügen. Viele Befehle, wie auch der Befehl `ls` müssen angepasst werden. Daher sollten Sie SELinux nur dann einsetzen, wenn Ihre Distribution SELinux unterstützt oder Sie sehr viel Zeit haben, die Unterstützung selbst einzubauen.

Hier haben die Dateien `/etc/passwd` und `/etc/shadow` den SELinux-User *system_u*. Dies ist typisch für Systemdateien. Die Datei des Benutzers *student* hat den SELinux-User *user_u*. Alle Dateien haben die SELinux-Rolle *object_r*. Dies ist typisch für alle Objekte. Tatsächlich unterscheiden sich die Dateien in ihrem Typ. Die Datei `/etc/passwd` hat den Typ *etc_t*. Dies ist typisch für die meisten Dateien im Verzeichnis `/etc`. Die Datei `/etc/shadow` besitzt den Typ *shadow_t*. So kann SELinux für diese Datei andere Zugriffsbeschränkungen erzwingen. Die Datei des Benutzers *student* hat schließlich den Typ *user_home_t*. Dieser Typ wird auf Fedora Core 5-Systemen verwendet, um die Dateien in den Heimatverzeichnissen der Benutzer auszuzeichnen.

Auch Prozesse besitzen diesen Security-Context. Sie können den Security-Context eines Prozesses mit dem Befehl `ps` anzeigen:

```
[root@supergrobi ~]# ps -eZ
...
system_u:system_r:xfs_t          2077 ?          00:00:00 xfs
```

12.3 Type Enforcement am Beispiel: Squid

```
system_u:system_r:cron_d_t      2094 ?          00:00:00 atd
system_u:system_r:getty_t      2213 tty1         00:00:00 mingetty
...
root:system_r:unconfined_t    2516 pts/1      00:00:00 bash
```

Auch hier erkennen Sie, dass jeder Prozess einen eigenen Security-Context besitzt. Auch hier unterscheiden sich die Security-Contexts nicht in der Rolle (*system_r*) und nur wenig im Benutzer (*system_u* oder *root*). Das wesentliche Unterscheidungsmerkmal hier ist wieder der Typ oder besser gesagt die Domäne. Erinnern Sie sich: Der Typ eines Prozesses wird als *Domäne* bezeichnet. In diesem speziellen Fall handelt es sich um ein Fedora Core 5-System mit einer *Targeted-Policy*¹.

Ob der Zugriff eines Subjekts auf ein Objekt nun erlaubt wird, hängt von den *Type-Enforcement*-Regeln ab. Diese Regeln definieren, wie eine Domäne auf Objekte mit einem bestimmten Typ zugreifen darf:

```
allow passwd_t shadow_t:file rw_file_perms;
```

12.3 Type Enforcement am Beispiel: Squid

Damit Sie SELinux, die Security-Contexts und die Richtlinien leichter verstehen, spielen wir das Ganze an einem Beispiel durch. Ich hoffe, Ihnen ist der Caching-Webproxy *Squid* bekannt. Der Squid ist ein Webproxy, der die Anfragen von Webbrowsern entgegennimmt und die Antworten der Webserver an die Clients ausliefert und zwischenspeichert. Nach seiner Installation und seinem Start auf einem *Fedora Core 5*-System mit *Targeted-Policy* läuft der Prozess in einer eigenen *Domäne squid_t*:

```
[root@supergrobi ~]# ps -eZ | grep squid
root:system_r:squid_t          3444 ?          00:00:00 squid
root:system_r:squid_t          3446 ?          00:00:00 squid
root:system_r:squid_t          3448 ?          00:00:00 unlinkd
```

Die verschiedenen Dateien und Verzeichnisse, auf die der Squid Webproxy zugreift, besitzen ebenfalls eigene Security-Contexts:

```
[root@supergrobi ~]# ls -dZ /var/spool/squid /var/log/squid ◀
    /etc/squid/squid.conf
-rw-r----- root squid system_u:object_r:squid_conf_t ◀
    /etc/squid/squid.conf
drwxr-x--- squid squid system_u:object_r:squid_log_t ◀
    /var/log/squid
drwxr-x--- squid squid system_u:object_r:squid_cache_t ◀
    /var/spool/squid
```

¹ Neben der *Targeted-Policy* gibt es auch noch eine *Strict*- und eine *MLS-Policy*. Die Unterschiede werden später im Buch erläutert. Die *Targeted-Policy* ist der Default und wird zunächst hier im Buch vorausgesetzt.

SELinux benötigt nun Regeln, die den Zugriff der Domäne *squid_t* auf die Dateien vom Typ *squid_conf_t*, *squid_log_t* und *squid_cache_t* erlauben. Dabei benötigt der Squid teilweise schreibende oder nur lesende Rechte. Damit Sie einen ersten Eindruck von der Regelsyntax erhalten, ist hier ein Auszug der Squid-Regeln abgedruckt:

```
allow squid_t squid_cache_t:dir create_dir_perms;
allow squid_t squid_cache_t:file create_file_perms;
allow squid_t squid_cache_t:lnk_file create_lnk_perms;

allow squid_t squid_conf_t:file r_file_perms;
allow squid_t squid_conf_t:dir r_dir_perms;
allow squid_t squid_conf_t:lnk_file read;
```

Diese Regeln zu lesen, ist nun nicht ganz so einfach wie bei AppArmor. Wir beginnen mit der ersten Regel. Diese erlaubt (`allow`) den Zugriff der Domäne *squid_t* auf Objekte mit der Klasse *dir* (Verzeichnis) und dem Typ *squid_cache_t*. Der erlaubte Zugriff lautet *create_dir_perms*². Die zweite Regel erlaubt einen entsprechenden Zugriff auf Dateien (*file*) vom Typ *squid_cache_t*, während die dritte Regel den Zugriff auf Verknüpfungen (*lnk_file*) erlaubt. Bei dem Zugriff auf Dateien, Verzeichnisse und Verknüpfungen mit dem Typ *squid_conf_t* erlaubt SELinux nur lesende Operationen (*r_file_perms*, *r_dir_perms* und *read*).

Jede `allow`-Regel besteht also aus dem Schlüsselwort `allow`, der zugreifenden Domäne (*source_t*), dem Objekttyp (*target_t*), der Objektklasse (*class*) und den Berechtigungen (*permissions*).

```
allow source_t target_t:class {permissions};
```

Wenn nun der Squid-Prozess auf die Datei `/etc/squid/squid.conf` zugreifen möchte, prüft zunächst das klassische Linux-Rechtssystem (*DAC*), ob der Zugriff erlaubt ist. Der Squid-Prozess läuft mit dem Benutzer *squid*. Dieser muss Leserechte an der Datei `squid.conf` besitzen:

```
-rw-r----- 1 root squid 122530  8. Jun 12:53 /etc/squid/squid.conf
```

Dies ist erfüllt, da der Benutzer *squid* Mitglied der Gruppe *squid* ist und diese Gruppe Leserechte erhält. Dann prüft SELinux, ob der Zugriff erlaubt ist. Da der Prozess in der Domäne *squid_t* läuft und die Datei den Typen *squid_conf_t* besitzt und für diese Kombination eine *allow*-Regel existiert, wird der Zugriff erlaubt.

Ein Prozess, der nicht in der Domäne *squid_t* läuft, darf nicht auf die Datei zugreifen. Auch wenn der Prozess den Benutzer *root* verwenden würde, würde SELinux den Zugriff ablehnen, da SELinux grundsätzlich jeden Zugriff ablehnt, der nicht explizit erlaubt wird.

² Dies ist ein Makro, und es enthält die Rechte, die zum Erzeugen von Verzeichnissen erforderlich sind: `create read getattr lock setattr ioctl link unlink rename search add_name remove_name reparent write rmdir`.

12.4 Welche Ressource erhält welchen Context?

Natürlich gibt es in SELinux auch Regeln, die den Zugriff auf Ports, IP-Adressen und UNIX-Sockets regeln. Diese werden wir in einem späteren Kapitel genauer besprechen. Hier soll nur kurz ein Beispiel gezeigt werden.

```
corenet_tcp_bind_all_nodes(squid_t)
corenet_tcp_bind_http_cache_port(squid_t)
```

Diese einfache Zeile in der Squid-Regeldatei erlaubt es der Domäne *squid_t*, sämtliche IP-Adressen (*nodes*) zu verwenden und sich auf den HTTP-Cache-Port zu binden. Bei beiden Aufrufen handelt es sich um SELinux-Interfaces. Hiermit werden von anderen Regelsätzen exportierte Regeln aufgerufen. Der zweite Aufruf wird folgendermaßen aufgelöst:

```
interface('corenet_tcp_bind_http_cache_port', '
    gen_require('
        type http_cache_port_t;
    ')

    allow $1 http_cache_port_t:tcp_socket name_bind;

')
```

Das Interface verlangt zunächst die Existenz des Typs *http_cache_port_t* und erlaubt dann dem übergebenen Typ (*squid_t*) den Zugriff auf entsprechende TCP-Sockets. Woher weiß nun SELinux, dass eine neu angelegte Datei einen bestimmten *Security-Context* bekommt oder ein *TCP-Socket* einen bestimmten Context aus User, Role und Type hat?

12.4 Welche Ressource erhält welchen Context?

Die SELinux-Policy entscheidet auch, welchen *Security-Context* eine Datei oder ein Netzwerkport erhält. Sie können sich die geladenen Einstellungen mit dem Befehl `semanage anzeigen` lassen.

```
[root@supergrobi ~]# semanage fcontext -l | grep squid
/etc/squid(/.*)?      all files    system_u:object_r:squid_conf_t:s0
/var/log/squid(/.*)? all files    system_u:object_r:squid_log_t:s0
...
```

Hier können Sie erkennen, dass alle Dateien in dem Verzeichnis `/etc/squid` und das Verzeichnis selbst den Security-Context *system_u:object_r:squid_conf_t:s0* erhalten. Wir ignorieren für einen kurzen Moment den vierten Parameter *s0*. Dieser wird in einem späteren Abschnitt (siehe Abschnitt 12.7) angesprochen. Wie bereits erwähnt, ist der einzige wesentliche Bestandteil des Security-Contexts der Typ: *squid_conf_t*.

Mit dem Befehl `semanage` können Sie auch den Typ eines Ports ermitteln:

```
[root@supergrobi ~]# semanage port -l | grep cache
http_cache_port_t      tcp      3128, 8080, 8118
http_cache_port_t      udp      3130
```

Hier finden Sie den `http_cache_port_t` wieder, den wir weiter oben bereits kennengelernt haben. Die oben besprochenen Regeln erlauben es dem Prozess, in der *Domäne* `squid_t` auf die Ports 3128/tcp, 8080/tcp, 8118/tcp und 3130/udp zuzugreifen. Weitere Ports werden von SELinux verweigert. Dies zeigt bereits die sehr detaillierte Konfiguration von SELinux und gibt einen Eindruck von dem sehr großen Aufwand, der bei der Erstellung einer SELinux Policy betrieben werden muss. Die Policy soll nur das Nötigste, aber Notwendige erlauben. Bei vielen installierten Applikationen kann die Policy schnell mehrere 1000 Regeln umfassen.

12.5 Das klassische Beispiel: passwd

Eine klassische Anwendung für SELinux ist die Überwachung der Datei `/etc/shadow`. Um SELinux in diesem Zusammenhang zu verstehen, schauen wir uns zunächst einmal an, welche Aufgabe diese Datei hat und wie sie verwaltet wird.

Die Datei `/etc/shadow` enthält die verschlüsselten³ Kennwörter jedes Benutzers. Wenn jeder Benutzer diese Datei lesen dürfte, könnte er die verschlüsselten Kennwörter der anderen Benutzer lesen und mit einem Crack-Programm wie `john`⁴ knacken. Daher darf nur der Benutzer `root` diese Datei lesen und schreiben:

```
[root@supergrobi ~]# ls -l /etc/shadow
-rw----- 1 root root 1174 18. Aug 16:34 /etc/shadow
```

Teilweise verfügt die Datei auch über noch weniger Rechte. Dann basiert der Zugriff auf der Tatsache, dass sich `root` über die Dateirechte hinwegsetzen darf (Capability: `CAP_DAC_OVERRIDE`, siehe Abschnitt 2.3). Jeder Benutzer soll nun sein Kennwort ändern dürfen, während `root` sämtliche Kennwörter ändern darf. Hierzu hat der Befehl `passwd` einen ausgeklügelten Mechanismus, mit dem er ermittelt, ob es sich um den Benutzer `root` oder um einen normalen Benutzer handelt.

Dennoch reicht dies nicht für eine Funktion des Kommandos `passwd`. Ruft ein normaler Benutzer diesen Befehl auf, so verfügt der Prozess nicht über die ausreichenden Rechte, um die Datei `/etc/shadow` zu editieren. Hierzu wird ein zusätzliches Recht benötigt:

```
[root@supergrobi ~]# ls -l /usr/bin/passwd
-r-s--x--x 1 root root 21944 12. Feb 2006 /usr/bin/passwd
```

³ Das ist nicht ganz korrekt. Die Kennwörter werden nicht verschlüsselt, sondern gehasht. Es soll nicht möglich sein, die Kennwörter zu entschlüsseln.

⁴ <http://www.openwall.com/john>

Dabei handelt es sich um das *SetUID*-Recht (*s*). Dieses Recht erlaubt es dem Prozess *passwd*, die Identität zu wechseln. Jeder Prozess auf einem Linux-System kennt zwei Identitäten: *uid* und *euid*. Die *uid* ist die Identität des aufrufenden Benutzers, während die *euid* die effektive Identität des Prozesses darstellt. Die *uid* kann ein Prozess nicht ändern, wohl aber die *euid*. Das *SetUID*-Recht erlaubt es einem Prozess, die *euid* auf den Besitzer der Binärdatei zu ändern. Im Fall des Befehls *passwd* ist das *root*. So erhält der Prozess *passwd* das Recht, die Datei */etc/shadow* zu modifizieren.

So weit, so gut. Wo ist nun die Anwendung für SELinux? Es gibt noch weitere Befehle mit dem *SetUID*-Recht. Sie können auf Ihrem Linux-System leicht sämtliche Befehle im Verzeichnis */bin* mit diesem Recht anzeigen:

```
[root@supergrobi ~]# find /bin -perm -4000 -ls
672716   72 -rwsr-xr-x   1 root root 62236 Mai 24 21:03 /bin/umount
672753   32 -rwsr-xr-x   1 root root 25152 Jul 13 12:09 /bin/su
672692   40 -rwsr-xr-x   1 root root 36608 Feb 24 16:42 /bin/ping
672721   96 -rwsr-xr-x   1 root root 87820 Mai 24 21:03 /bin/mount
672693   36 -rwsr-xr-x   1 root root 31796 Feb 24 16:42 /bin/ping6
```

Für uns bedeutet das, dass auch der Befehl *ping* bei einem Aufruf über die notwendigen Privilegien verfügt, um die Datei */etc/shadow* zu editieren und einen Benutzer hinzuzufügen. Das Discretionary-Access-Control-System (*DAC*) von Linux ist nicht in der Lage, hier die Datei */etc/shadow* zu schützen. Verfügt nun das Kommando *ping* über eine Sicherheitslücke, die ein Angreifer – vielleicht sogar über das Netz – ausnutzen kann, kann er die Benutzerdatenbank nach seinem Geschmack editieren. SELinux kann dies verhindern.

Da SELinux grundsätzlich jeden Zugriff ablehnt (*Default Deny*), sind die Zutaten hierfür sparsam. Wir benötigen dazu:

- einen Security-Context für die Datei */etc/shadow*
- einen Security-Context für den Prozess *passwd*
- eine Allow-Regel, die den Zugriff erlaubt

Die Datei */etc/shadow* erhält den Security-Context *system_u:object_r:shadow_t*:

```
[root@supergrobi ~]# s -Z /etc/shadow
-rw----- root root system_u:object_r:shadow_t /etc/shadow
```

Der Prozess *passwd* erhält bei dem Aufruf durch einen normalen Benutzer den Security-Context *user_u:user_r:passwd_t*:

```
[root@supergrobi ~]# ps -Z | grep passwd
user_u:user_r:passwd_t 8322 pts/1 00:00:00 passwd
```

Nun fehlt noch die Allow-Regel:

```
allow passwd_t shadow_t:file rw_file_perms;
```

Der Parameter `rw_file_perms` ist wieder ein Makro, das alle Rechte enthält, die für den lesenden und schreibenden Zugriff benötigt werden. Diese Regel erlaubt es, Prozessen in der Domäne `passwd_t` auf Objekte mit der Klasse `file` und dem Typ `shadow_t` lesend und schreibend zuzugreifen. Wie gelangt nun der Prozess `passwd` in die Domäne `passwd_t`?

12.5.1 Domänentransition

Wenn der Prozess `passwd` von dem Benutzer aufgerufen wird, läuft er zunächst in der Domäne des Benutzers. Dies ist üblicherweise die Domäne `user_t`. Wie gelangt er jetzt in die Domäne `passwd_t`?

Hierfür benötigen wir eine *Domänentransition*. Das ist ein Wechsel von einer Domäne in eine andere. Diese Wechsel sind grundsätzlich verboten und müssen über entsprechende Regeln erlaubt werden. Hierzu wird ein Trick genutzt. Die Binärdatei `/usr/bin/passwd` erhält den Typ `passwd_exec_t`.

```
[root@supergrobi ~]# ls -Z /usr/bin/passwd
-r-s--x--x root root system_u:object_r:passwd_exec_t /usr/bin/passwd
```

Jetzt definieren wir Regeln, die es einem Prozess, der aus einer Binärdatei mit diesem Typ entsteht, erlauben, in die Domäne `passwd_t` zu wechseln:

```
allow user_t passwd_exec_t:file { getattr execute };
allow passwd_t passwd_exec_t:file entrypoint;
allow user_t passwd_t:process transition;
```

Die erste Regel erlaubt es zunächst dem Benutzer, dessen Shell in der Domäne `user_t` läuft, Dateien vom Typ `passwd_exec_t` auszuführen. Ohne diese Regel wäre selbst diese Operation bereits verboten. Denken Sie immer daran: SELinux verbietet grundsätzlich alles!

Die zweite Regel ist nun die wichtigste Regel in diesem Konstrukt. Hiermit öffnen Sie eine Tür in die Domäne `passwd_t`. Ein *entrypoint* ist eine binäre Datei, die als Tür in eine andere Domäne genutzt werden darf. Da wir nur diese eine Tür definieren, können lediglich Prozesse, deren Binärdatei den Typ `passwd_exec_t` besitzen, in die Domäne `passwd_t` wechseln. Keinem anderen Prozess erlaubt SELinux den Wechsel. Der `ping`-Befehl kann nun nicht in die Domäne `passwd_t` wechseln, und daher darf er auch nicht auf die Datei `/etc/shadow` zugreifen!

Die dritte Regel definiert schließlich noch, dass die Domäne `user_t` den Wechsel in die Domäne `passwd_t` durchführen darf. Der *Squid*-Proxy, der in der Domäne `squid_t` läuft, könnte genauso wie ein normaler Benutzer den Befehl `passwd` aufrufen. Natürlich hat auch die *entrypoint*-Regel für den Squid ihre Gültigkeit, jedoch darf aus der Domäne `squid_t` ohne eine explizite Regel kein Wechsel in die Domäne `passwd_t` erfolgen. Das wird hier nur für die Domäne `user_t` erlaubt. Erstaunlich ist bei dieser letzten Regel die Klasse des Objekts: *process*. Tatsächlich gibt es noch sehr viele weitere Klassen, die wir noch kennenlernen werden. Hier wird das Recht *transition* bei dem

Zugriff auf den Prozess gewährt. Das erlaubt es dem Prozess, die eigene Domäne zu wechseln.

Damit tatsächlich der Wechsel möglich ist, werden alle drei Regeln benötigt. Jede einzelne Regel genügt nicht, sondern erlaubt nur einen bestimmten Aspekt.

Hier noch einmal zusammengefasst:

1. Für die originale Domäne existiert eine Regel, die dem Prozess die Transition in die Zieldomäne erlaubt.
2. Für diese Transition wurde ein Entrypoint definiert.
3. Die originale Domäne darf die Entrypoint-Binärdatei ausführen.

Jedoch erlauben diese Regeln lediglich die *Domänentransition*, sie erzwingen sie nicht. Der Prozess muss die Transition anfordern, damit sie stattfindet. Dies kann aber nur bei neuer Software, die noch geschrieben wird, berücksichtigt werden. Daher erlaubt SELinux auch die Definition von automatischen Domänentransitionen. Die folgende Regel kümmert sich um unser Problem:

```
type_transition user_t passwd_exec_t:process passwd_t;
```

Mit dem Schlüsselwort `type_transition` wird automatisch eine Domänentransition in die Domäne `passwd_t` angefordert, sobald ein Prozess aus der Domäne `user_t` eine Datei mit dem Typ `passwd_exec_t` aufruft.



Achtung

Diese Regel erlaubt die Transition nicht, sondern verlangt sie nur. Die drei weiter oben definierten Regeln werden zusätzlich benötigt!

Da diese Vielzahl an Regeln zu unübersichtlichen und nicht wartbaren Regelsätzen führt, nutzen die Regelsätze an ihrer Stelle Makros.

```
domain_entry_file(passwd_t,passwd_exec_t)
domain_auto_trans(user_t,passwd_exec_t,passwd_t)
```

12.6 Rollen und Benutzer

Wozu werden nun die Rollen und Benutzer verwendet? Die *Type-Enforcement*-Regeln prüfen ja nur die Domäne des zugreifenden Prozesses und den Typ der Resource. Die Rolle definiert, welche Domänen für einen Prozess erreichbar sind. Im

letzten Abschnitt haben wir gesehen, dass ein Prozess eine Domänentransition durchführen kann, wenn entsprechende Regeln es erlauben. Zusätzlich muss aber auch die Rolle des aufrufenden Prozesses diese *Domänentransition* erlauben. So sind bestimmte Domänen nur für bestimmte Rollen erreichbar. Die Rolle limitiert also die möglichen Domänentransitionen. Es handelt sich also um eine Art von Role Based Access Control (RBAC). Entsprechend dem Beispiel im letzten Abschnitt muss die Rolle *user_r*, mit der der `passwd`-Prozess läuft, die Erlaubnis haben, in die Domäne *passwd_t* zu wechseln. Diese Erlaubnis wird mit einer `role`-Anweisung gegeben:

```
role user_r types passwd_t;
```

Diese Anweisung deklariert zunächst die Rolle *user_r*, falls sie noch nicht in der Policy existieren sollte. Außerdem erlaubt die Anweisung der Rolle *user_r*, die Domäne *passwd_t* zu nutzen.

Wie werden denn nun die Rollen verwaltet? Welcher Benutzer darf welche Rollen nutzen? Das ist die Aufgabe der SELinux-Benutzerverwaltung. Hierzu gibt es die Benutzerdeklarationen, die Sie mit dem Befehl `semanage` anzeigen können.

```
[root@supergrobi modules]# semanage user -l
```

	Labeling	MLS/	MLS/	
SELinux User	Prefix	MCS Level	MCS Range	SELinux Roles
root	user	s0	SystemLow-SystemHigh	system_r ←
	sysadm_r	staff_r		
staff_u	staff	s0	SystemLow-SystemHigh	sysadm_r ←
	staff_r			
sysadm_u	sysadm	s0	SystemLow-SystemHigh	sysadm_r
system_u	user	s0	SystemLow-SystemHigh	system_r
user_u	user	s0	SystemLow-SystemHigh	user_r

Wir ignorieren für einen Moment (bis zum nächsten Abschnitt) noch die Spalten *MLS/MCS*. Die weiteren Spalten definieren, welcher Benutzer welche Rollen nutzen darf. Der SELinux-Benutzer *root* darf die SELinux-Rollen *system_r*, *sysadm_r* und *staff_r* verwenden. Der SELinux-Benutzer *system_u* hat lediglich Zugriff auf die Rolle *system_r*, genauso wie der SELinux-Benutzer *user_u* auch nur Zugriff auf die Rolle *user_r* hat.

SELinux ordnet jetzt die Linux-Benutzer den SELinux-Benutzern zu. Systemdienste erhalten den Benutzer *system_u*. Die weiteren Benutzer werden bei ihrem Login zugewiesen. Hierzu verwendet SELinux das folgende Mapping:

```
[root@supergrobi ~]# semanage login -l
```

Login Name	SELinux User	MLS/MCS Range
__default__	user_u	s0
root	root	SystemLow-SystemHigh

Bei dem Benutzer *root* entspricht der SELinux-Benutzername dem Linux-Benutzernamen. Alle weiteren Benutzer erhalten den SELinux-Benutzer *user_u*. Damit ist festgelegt, welche Rollen sie benutzen dürfen. Möchten Sie zum Beispiel, dass ein bestimmter Benutzer auch die Rolle *sysadm_r* benutzen darf, können Sie dies folgendermaßen erreichen:

```
[root@supergrobi ~]# semanage user -R "sysadm_r user_r" -P user -a ralf_u
[root@supergrobi ~]# semanage login -s ralf_u -a ralf
```

Der erste Befehl erzeugt den SELinux-Benutzer *ralf_u* und weist diesem die Rollen *sysadm_r* und *user_r* zu. Der zweite Befehl erzeugt das Mapping für den Login, sodass SELinux dem Linux-Benutzer *ralf* bei der Anmeldung den SELinux-Benutzer *ralf_u* zuweist. Dies lässt sich nach einer Anmeldung auch verifizieren:

```
[ralf@supergrobi ~]$ id -Z
ralf_u:user_r:user_t
[ralf@supergrobi ~]$ newrole -r sysadm_r
Authentifiziere ralf.
Passwort:
[ralf@supergrobi ~]$ id -Z
ralf_u:sysadm_r:sysadm_t
```

Die hier verwendeten Befehle *id* und *newrole* (siehe auch Abschnitt 18.15) wurden noch nicht erwähnt und sollen kurz erklärt werden:

- *id*: Dieser Befehl zeigt die aktuelle Identität des Benutzers. Mit der Option *-Z* zeigt er die SELinux-Identität. Nach der Anmeldung hat der Benutzer den SELinux-Benutzer *user_u* und die Rolle *user_r*.
- *newrole*: Mit diesem Befehl wechselt ein Benutzer seine Rolle. Dieser Befehl ist nur erfolgreich, wenn der Benutzer auf die neue Rolle zugreifen darf.

12.7 Multi Level Security

Multi Level Security (MLS) wurde bereits in Abschnitt 2.1.3 beschrieben. Daher soll hier nicht erneut darauf eingegangen werden. SELinux hat immer schon *MLS* unterstützt, jedoch war der Einsatz lange Zeit experimentell und für kommerzielle Anwendungen nicht geeignet. Ab dem Kernel 2.6.12 ist ein neues und überarbeitetes Modell im Linux-Kernel vorhanden, das sowohl Multi-Level- als auch Multi-Category-Security (*MCS*) anbietet.

Die aktuellen SELinux-Implementierungen unterstützen *MLS* und *MCS*. Wenn Sie prüfen möchten, ob dass auch für Ihr System zutrifft, können Sie das mit dem Befehl *semanage* oder *id tun*:

```
[root@supergrobi ~]# id -Z
root:staff_r:staff_t:SystemLow-SystemHigh
[root@supergrobi ~]$ /usr/sbin/semanage translation -l
```

Level	Translation
s0	
s0-s15:c0.c255	SystemLow-SystemHigh
s0:c0.c255	SystemHigh

Der *Security-Context* enthält dann ein zusätzliches Feld. Dieses Feld besteht in Wirklichkeit aus bis zu zwei Feldern: *Sensitivity* und *Compartment* (oder *Category*). Jedes Objekt besitzt nun genau eine MLS-Markierung. Diese Markierung ist für die meisten Objekte *SystemLow* oder auch *s0*. Besondere Objekte wie `/dev/mem` erhalten eine andere Markierung (z.B. *SystemHigh*):

```
[root@supergrobi ~]# ls -Z /dev/mem
crw-r----- root kmem system_u:object_r:memory_device_t:SystemHigh /dev/mem
```

Jeder Prozess erhält einen MLS-Bereich (*Range*) zugewiesen. Dieser wird auch bei neuen Prozessen vererbt. Allerdings kann es zu Bereichstransitionen (*range_transition*) kommen.

Damit nun ein Prozess auf eine Ressource zugreifen darf, müssen die MLS-Eigenschaften des Prozesses und der Ressource ein bestimmtes Verhältnis aufweisen. Damit ein Prozess eine Datei lesen oder ausführen darf, gilt die folgende SELinux-Regel:

```
mlsconstrain { dir file lnk_file chr_file blk_file
  sock_file fifo_file }
{ read getattr execute }
(( l1 dom l2 ) or
  (( t1 == mlsfilereadtoclr )
    and ( h1 dom l2 )) or
  ( t1 == mlsfileread ) or
  ( t2 == mlstrustedobject ));
```

Diese Regel ist im Moment sicherlich noch nicht einfach nachzuvollziehen. Aber ich möchte dennoch versuchen, den wesentlichen Teil zu erläutern. Ein *Constraint* ist eine Einschränkung, die immer erfüllt sein muss, damit ein Zugriff erlaubt wird. Hier handelt es sich um den lesenden (*read*) oder ausführenden (*execute*) Zugriff auf Dateien (*file*) oder Verzeichnisse (*dir*). Dieser wird nur gewährt, wenn $(l1 \text{ dom } l2)$. Die Platzhalter *l1* und *l2* stehen für den MLS-Wert des Subjekts und des Objekts. Von *Dominanz* ($l1 \text{ dom } l2$) spricht man, wenn die Sicherheitsstufe *l1* höher oder gleich der Stufe *l2* ist. Gelesen werden dürfen Dateien also nur von Prozessen, die mindestens dieselbe Sicherheitsstufe aufweisen. Umgekehrt ist ein Schreibzugriff nur erlaubt, wenn das Subjekt von dem Objekt dominiert wird. Die Sicherheitsstufe des Objekts *l2* ist also mindestens dieselbe Stufe. Ein Schreiben und Lesen ist damit nur möglich, wenn das Subjekt (Prozess) und das Objekt über identische Sicherheitsstufen verfügen.

12.8 Multi Category Security

Die Multi Category Security (MCS) wurde in der *Reference-Policy* eingeführt, um Multi Level Security benutzerfreundlicher und genereller zu gestalten. MCS baut komplett auf *MLS* und damit auf *Type-Enforcement* Regeln auf. Bei einem *MLS*-System entscheidet das System, welche Sicherheitstufe ein Objekt erhält. Ein Benutzer hat hier keinen Einfluss. Bei MCS erhalten alle Objekte dieselbe Sicherheitssensitivität (*s0*) und unterscheiden sich nur in ihrer Kategorie (*c0-c255*). Diese Kategorien können von den Benutzern selbst verwaltet werden.

Wendet der Benutzer diese Möglichkeiten nicht an, arbeitet SELinux so, als wäre die MCS-Funktionalität nicht aktiviert. Nutzt der Benutzer aber MCS, kann er den Zugriff auf bestimmte Dokumente noch weiter einschränken. Diese Einschränkung kann unabhängig von Benutzerrechten nur für einzelne Dateien und Dienste gelten.

Obwohl die MCS-Funktionalitäten aktuell von keiner Distribution genutzt werden⁵ sind aufregende Anwendungen möglich:

- Beim Druck von Dokumenten kann das Drucksystem in Abhängigkeit der MCS-Kategorie den Druck verbieten, nur auf bestimmten Druckern erlauben oder mit Kopfzeilen versehen.
- Beim E-Mail-Versand kann das Mail-Programm das Anhängen von bestimmten Dokumenten verbieten oder nur verschlüsselt erlauben.

Diese Anwendungen sind im Moment aber noch Zukunftsmusik. Vielleicht bringen zukünftige Fedora-Versionen hier neue Ansätze.

⁵ Fedora Core 5 bietet MCS an, jedoch kann jeder Benutzer und jeder Prozess momentan auf alle Kategorien zugreifen. FC5 implementiert mit MCS also bisher keinen Schutz.

