Sicherheitsmanagement mit SELinux unter CentOS 5

Angenendt, Ralph
Ridlerstr. 11
80339 München
ralph@centos.org

1 Der klassische Weg des Rechtemanagements

rwxr-xr-x – also die Vergabe von Schreib-, Lese- und Ausführrechten in Verbindung mit Username, Gruppenzugehörigkeit und den Rechten von „anderen“ war unter Linux lange Zeit der einzige Weg, Dateien für berechtigte Nutzer im Zugriff zu halten und unberechtigten Nutzern den Zugang zu verwehren. Funktioniert das auf Systemen mit wenigen Nutzern und vor allem wenigen Gruppen noch halbwegs zuverlässig und übersichtlich, so ergeben sich bei größeren Systemen einige Nachteile.

1.1 Vorteile der klassischen Rechtevergabe

Durch die eingeschränkte Anzahl von Parametern bleibt die klassische Rechtevergabe auf den ersten Blick sehr übersichtlich. Simplifiziert: Den Usernamen kann man aus `/etc/passwd` entnehmen, die Gruppenzugehörigkeit ist in der Datei `/etc/group` definiert. Natürlich können diese Attribute auch in einem firmenweiten LDAP-Server oder einem NIS-Verzeichnis definiert sein, die Grundlagen sind die Gleichen.

```
[angenenr@shutdown tmp]$ls -l rechnung.pdf
-rw-r----- 1 angenenr users 120090 Feb 14 10:38 rechnung.pdf
[angenenr@shutdown tmp]$getent group users
users:x:100:angenenr,mueller,meier
```

Die Datei `rechnung.pdf` darf vom Nutzer `angenenr` gelesen und verändert werden, die Gruppe `users` darf lesen und alle anderen müssen draußen bleiben. In der Gruppe `users` sind die Nutzer `angenenr`, `mueller` und `meier`, diese haben eine Leseberechtigung. Zumindest theoretisch, dürfen sie nicht auf das Verzeichnis zugreifen, in dem diese Datei liegt, nutzt ihnen auch die Zugehörigkeit zur Gruppe `users` nichts.

Diese Art der Rechtevergabe ist einfach und leicht zu vermitteln, so dass auch Anfänger schnell damit klarkommen. Anfängerfehler wie „Schreib-, Lese- und Ausführrechte für alle“ lassen sich durch eine Einführung sehr einfach vermeiden.

1.2 Probleme des klassischen Modells

Leider hat dieses Modell einige Probleme, die unter anderem in der Einfachheit der Rechtevergabe verwurzelt sind – und dem Mangel an Attributen. In komplexeren Systemen reichen User und Gruppe nicht aus, das Attribut „Andere“ kann ebenfalls nur selten vernünftig genutzt werden, so dass im Endeffekt nur zwei Attribute zur Rechtedefinition übrig. Sehr wenig.

Hinzu kommen weitere Beschränkungen. So ist die Anzahl an Gruppen, der ein User angehören darf, auf vielen unixoiden Systemen beschränkt. Auf Linuxsystemen mit Kernel 2.6 darf ein User 65535 Gruppen

angehören. Bei Systemen mit Kernel 2.4 sind es nur 32. Auf Solaris sind es 16 bis 32, auf anderen Systemen sieht es ähnlich aus. Kommt dann noch NFS ins Spiel, hat man plötzlich nur noch 16 Gruppen (neben der primären Gruppe des Users), in denen sich ein Nutzer befinden darf. So lässt sich folgendes Beispiel mit dem klassischen Modell nicht ohne Probleme abbilden:

Gegeben sei eine mittelständische Firma mit 20 Abteilungen. Neben den 20 Abteilungen gibt es noch das Management und eine Abteilung QA, die sich um die ISO9001-Dokumentation der Firma kümmert. Der Dokumentenbaum der Firma liegt auf einem Filer, den sich die Mitarbeiter per NFS mounten. Jede Abteilung soll nur ihre eigenen Dokumente schreiben können, wobei Mitarbeiter durchaus in mehr als einer Abteilung sein können. Management und QA sollen aber in allen Bereichen des Dokumentenbaums schreiben können. Die offensichtliche Lösung: Jede Abteilung wird durch eine Gruppe abgebildet, Management und QA sind in allen Gruppen vorhanden, Mitarbeiter in mehreren Abteilungen sind Mitglied in allen entsprechenden Gruppen. Leider funktioniert das nicht, aus den oben angebenden Limits. QA und Management müssten Mitglied in mehr als 16 Gruppen sein, das verbietet das System allerdings.

Aber schon das folgende, einfachere Beispiel, lässt sich mit der klassischen Rechtevergabe nicht zufriedenstellend lösen:

Gegeben seien zwei Benutzergruppen „Content“ und „Backup“. Weiterhin sei ein Verzeichnis `/var/www/html` gegeben. Da sich in diesem Verzeichnis Dateien befinden, die der Webserver ausliefern soll, gehört es dem Benutzer Apache, der lesend zugreifen darf. Benutzergruppe „Content“ soll nun in der Lage sein, neuen Content einzustellen, muss also lesend und schreibend zugreifen. Die Gruppe „Backup“ soll lesend zugreifen können, um die Daten, die Content einstellt, auf Band sichern zu können. Alle anderen Nutzer haben keinerlei Rechte an den Dateien im Verzeichnis. Der Nutzer Apache kann nicht in die Gruppe Backup aufgenommen werden, da Backup auch in anderen Bereichen des Systems lesen darf. Backup darf aber auch nicht in die Gruppe Apache aufgenommen werden, da sie sonst Zugriff auf etwaige Konfigurationsdateien hat.

Und selbst wenn eine der beiden Bedingungen erlaubt sein sollte: Dann müsste aus der Gruppe „Content“ ein Nutzer „Content“ werden, dem das Verzeichnis dann gehört.

1.3 Fazit

Die klassische Rechtevergabe unter Unix ist sehr einfach zu erlernen und erfüllt ihre Zwecke auf einfachen Systemen recht zuverlässig. Sobald die Systeme komplexer werden, muss der Administrator etliche Klimmzüge machen, um mit den gegebenen Mitteln sein System sicher zu bekommen und allen Nutzern und Gruppen den Zugriff auf die Dateien zu geben, die sie im Zugriff haben sollen. Sicherer wird das System dadurch nicht, da durch die Komplexität Fehler wahrscheinlicher werden.

2 Access Control Lists

2.1 Filesystem Access Control Lists – ein Überblick

Filesystem Access Control Lists (kurz: facl) sind Informationen im Dateisystem, die eine erweiterte Schnittstelle zur Rechtevergabe in Dateisystemen bereitstellen, um einige Limitierungen – siehe oben – des klassischen Rechtemodells zu umgehen. Um FACLs zu nutzen, muss das drunterliegende Dateisystem Extended Attributes unterstützen. Extended Attributes nehmen beliebige Key-Value-Paare auf, die eine Datei genauer beschreiben, also einen Metadatensatz. So lassen sich zum Beispiel Informationen über den Autor eines Textes speichern:

```
[angenenr@shutdown tmp]$setfattr -n user.Autor -v Ralph foo
[angenenr@shutdown tmp]$getfattr -n user.Autor foo
# file: foo
user.Autor="Ralph"
```

FACLs werden ebenfalls als Extended Attribute gespeichert. Mit FACLs lassen sich zusätzliche Zugriffsberechtigungen auf Dateien und Verzeichnisse vergeben, so dass sich die Gruppenzugehörigkeit einer Datei auf mehrere Gruppen erstrecken kann oder dass zusätzlich zum „Owner“ der Datei weitere Nutzer lesenden, schreibenden oder ausführenden Zugriff auf eine Datei bekommen. Der Kernel wertet diese Attribute neben den klassischen Rechten aus.

Mit FACLs lassen sich also weitaus feiner granulierte Rechte vergeben, als es klassisch möglich ist, so dass auch komplexe Setups abbildbar sind.

FACLs und Extended Attributes sollten auf allen modernen Linuxsystemen nutzbar sein, viele andere Unixe wie Solaris oder Darwin/MacOS unterstützen diese Schnittstelle ebenfalls. Leider können noch nicht alle Backupprogramme mit Extended Attributes umgehen, so dass der Administrator bei einem Backup bzw. einem Restore dafür Sorge tragen muss, dass dieses Attribute ebenfalls gesichert werden. star, einige gepatchte Versionen von tar und neuere Versionen von rsync können allerdings mit Extended Attributes umgehen und speichern diese.

Extended Attributes werden unter Linux von den Dateisystemen ext2/3, reiserfs und XFS unterstützt. NFS wird leider nicht komplett unterstützt, für NFSv4 unter Linux wird es zumindest eine rudimentäre Unterstützung geben.

2.2 Beispiele

Nehmen wir das Beispiel des HTML-Verzeichnisses: Es gehört dem User apache, die Gruppe content soll lesend und schreibend zugreifen können, die Gruppe backup nur lesend. Alle anderen dürfen nicht zugreifen können.

```
[angenenr@shutdown tmp]$ls -ld html
dr-xr-x--- 2 apache apache 4096 Feb 17 20:58 html
[angenenr@shutdown tmp]$
```

Die Rechte für die Gruppen content und backup legen wir als FACL ab:

```
[angenenr@shutdown tmp]$sudo setfacl -m d:g:content:rwx,d:g:backup:rx
html
[angenenr@shutdown tmp]$getfacl html
# file: html
# owner: apache
# group: apache
user::r-x
group::r-x
other::---
default:user::r-x
default:group::r-x
default:group:content:rwx
default:group:backup:r-x
```

Die Gruppenrechte wurden als Defaultrechte angelegt, so dass jede Datei, die unterhalb von html erzeugt wird, die gleichen FACLs bekommt:

```
[angenenr@shutdown tmp]$touch html/foo
[angenenr@shutdown tmp]$getfacl html/foo
```

```
# file: html/foo
# owner: angenenr
# group: angenenr
user::r--
group::r-x                      #effective:r--
group:content:rwx               #effective:rw-
group:backup:r-x                #effective:r--
```

Mit dieser Herangehensweise lässt sich auch das Problem der Firma mit den 20 Abteilungen lösen: Die Gruppen management und QA bekommen als Defaultrecht rwx auf den kompletten Dokumentenbaum.

3 SELinux

3.1 Eine Einführung

Bis hierhin hat jeder Nutzer die vollständige Kontrolle über alle Dateisystemsobjekte, die er erstellt – er entscheidet darüber, wer außer ihm auf Dateien oder Verzeichnisse zugreifen darf, er entscheidet darüber, ob Dateien les- oder schreibbar sind oder ob sie ausgeführt werden dürfen. SELinux bietet ein im Kernel integriertes „Mandatory Access Control System“ (eine regelbasierte Zugriffskontrolle). Dieses Zugriffssystem schränkt die Kontrolle, die der Nutzer über seine eigenen Dateien hat, ein, indem allen Objekten im Dateisystem ein Label (bzw. ein Kontext) hinzugefügt wird. Um eine Datei im System ändern zu können, muss der Nutzer Zugriff auf diesen Kontext haben. Hat er ihn nicht, so ist es ihm nicht mögliche, ein Attribut einer Datei zu ändern, egal ob er im klassischen System alle Rechte auf diese Datei hat oder nicht.

Wird nun zum Beispiel ein Daemon auf dem System kompromittiert, so darf dieser Prozess bei weitem nicht alle Dateien lesen, auf die er klassischerweise Zugriff hätte, so z.B. alle Dateien, die für die Rolle „Andere“ Leserechte hätten (rwxr-xr-x).

Weiterhin implementiert SELinux ein Role Based Access Control (RBAC) System, in dem Zugriffsrechte auf ein Objekt im Dateisystem nach Rollen gegliedert werden. Diese Rollen können dann unterschiedliche Zugriffsberechtigungen auf ein Objekt haben. Diese Rollen lassen sich zum Beispiel nach Geschäftsprozessen modellieren, siehe z.B. erneut das Beispiel unseres Mittelständlers. Für den Administrator bietet dieses System eine Vereinfachung seiner Arbeit, da er über Zugriffe über Gruppenzugehörigkeit reglementieren kann.

Weiterhin bietet SELinux eine Multi-Level-Security-Implementierung (MLS), die sich an die Klassifizierung „Unclassified -> Confidential -> Secret -> Top Secret“ der Militärs anlehnt. In diesem Modell bekommen die Objekte (Dateien im System) „Classifications“, während die Subjekte (Nutzer) „Clearances“ zugewiesen bekommen.

Wie funktioniert nun SELinux? Abbildung 1 zeigt den Entscheidungsprozess, der abläuft, wenn ein Subjekt – also zum Beispiel ein Prozess – versucht, auf ein Objekt – hier eine Datei – zuzugreifen. Der Policy Enforcement Server im Kernel sucht im Access Vector Cache (AVC), ob das Subjekt die passenden Berechtigungen hat, auf das Objekt zuzugreifen. Kann anhand des Caches nicht entschieden werden, ob das so ist, dann wird der SELinux Security-Server angefragt. Dieser schlägt in der Policy-Database nach. Passen der Securitykontext des Subjekts und des Objekts zusammen, so wird der Zugriff auf das Objekt erlaubt. Ist der Zugriff nicht erlaubt, dann wird ein aussagekräftiger „avc: denied“-Eintrag im message log oder im Audit-System des Kernels erzeugt.

Anhand dieses Eintrags kann der Administrator unzulässige Zugriffsversuche erkennen, kann aber auch die Policy für das Subjekt oder das Objekt so verändern, damit dieser Zugriff danach erlaubt wird. Tools wie `audit2allow` und `auditwhy` lesen diese Einträge aus um die Erstellung von Policies zu erleichtern.

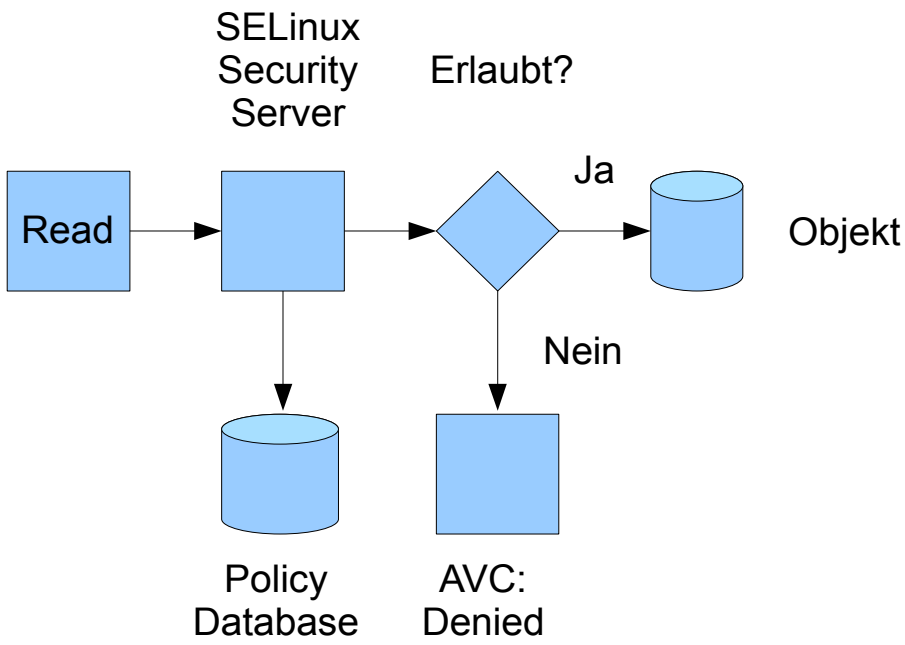


Abbildung 1: SELinux Entscheidungsprozess

3.2 SELinux starten

SELinux kennt drei Betriebsmodi:

Enforcing: Die definierte Policy wird geladen und vom Policy Enforcement Server erzwungen
Permissive: Die Policy wird geladen, aber nicht erzwungen. Dennoch werden „Denials“ geloggt. Dieser Modus eignet sich zum Debuggen von selbstgeschriebenen Policymodulen oder nicht funktionierenden Applikationen.
Disabled: Das System verhält sich, als ob es ohne SELinux läuft.

Im Enforcing- oder Permissive-Modus kennt SELinux die Policytypen „strict“ und „targeted“. Während bei strict alle Aktionen aller Prozesse vom Policy-Enforcement-Server überwacht werden, so werden bei der targeted policy nur ein Teil der auf dem System laufenden Prozesse überwacht. Unter CentOS/RHEL 5 sind das die folgenden Dienste: dhcpd, httpd, named, nscd, ntpd, portmap, snmpd, squid, syslogd und vsftpd. Alle anderen Prozessen laufen im Kontext „unconfined_t“. Alle in diesem Kontext laufenden Prozesse können (und müssen) mit den klassischen Rechten unter Unix geschützt werden. Die Standard-Policy unter CentOS/RHEL 5 ist die targeted policy. Wer weitere Dienste durch SELinux überwachen möchte, muss den Kontext für diese Prozesse selber erstellen. Das System erleichtert diese Aufgabe durch mitgelieferte Tools, mit denen Policy-Module geschrieben werden können.

Die grundlegenden Tools, mit denen der Administrator in seiner täglichen Arbeit konfrontiert wird, sollen im folgenden kurz beschrieben werden:

setenforce: Mit setenforce lässt sich der Betriebsmodus von SELinux im laufenden Betrieb verändern. **setenforce enforcing** schaltet den Enforcing-Modus ein, mit **setenforce permissive** lässt sich SELinux kurzfristig in den permissive Modus versetzen, um z.B. Probleme mit Applikationen oder Policies zu debuggen. SELinux lässt sich ohne Reboot nicht ausschalten, für den Disabled-Modus muss das System rebooted werden.

chcon: Mit chcon kann der Administrator den Kontext von Dateien und Verzeichnissen im Dateibaum verändern. Die Handhabung ist ähnlich wie die der bekannten Tools chmod und chown.

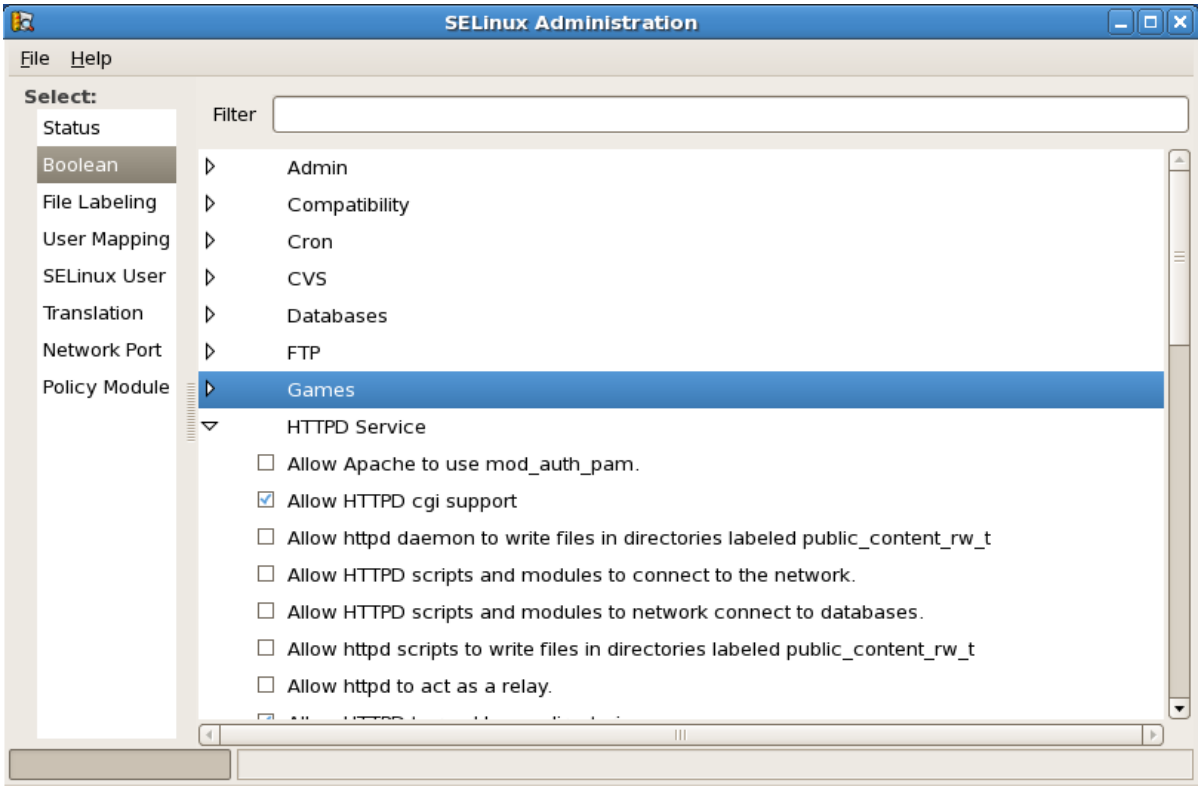
restorecon: Sollte der Administrator mit chcon (oder aus anderen Gründen) den SELinux-Kontext von z.B. /var/www/html verändert haben, so dass Probleme beim Betrieb des Webservers auftreten, kann mit **restorecon /var/www/html** der in der Policy definierte Zustand wiederhergestellt werden.

fixfiles: Hat der Administrator Dateien oder Policies geändert, während SELinux ausgeschaltet war, so stimmen wahrscheinlich einige Kontexte im System nicht mehr. Beispiel: Wurde das Paket httpd zu einem Zeitpunkt installiert, an dem SELinux ausgeschaltet war, so werden keine SELinux-Kontexte im Dateisystem angelegt. Mit **fixfiles -R httpd relabel** werden alle Dateien im Paket httpd mit dem korrekten Kontext versehen, der für dieses Paket festgelegt ist.

semodule: Mit semodule lassen sich Policymodule verwalten – sie können im laufenden Betrieb in den Kernel geladen werden, entladen werden, upgedatet werden.

semanage: Mit semanage können Teile der Policy neu konfiguriert werden, ohne die Sourcen anfassen oder neu kompilieren zu müssen. Weiterhin können mit semanage Linux-Usernamen auf SELinux-Identities gemappt werden oder Netzwerkports für Applikationen verwaltet werden. **semanage port -a -t http_port_t -p tcp 81** ermöglicht es, den Apache Webserver auch an Port 51 zu binden, was normalerweise von der Policy für httpd unterbunden wird.

system-config-selinux: Grafisches Tool mit dem sich verschiedenste Aspekte der aktiven Policy verändern lassen. **system-config-selinux** fasst die meisten oben genannten Tools unter einer grafischen Oberfläche zusammen. Hier findet sich auch ein eher natursprachlicher Überblick über die in SELinux vorhandenen Booleans:



3.3 SELinux im Betrieb

Wie wirkt sich SELinux nun im laufenden Betrieb aus? Mit der targeted Policy sollten User und Administratoren eines Systems keinen Unterschied zu einem System ohne SELinux merken, so sie sich an

die Vorgaben der Policy halten. Dazu zählt, dass sich von Prozessen benötigte Dateien an den dafür vorgesehen Orten befinden.

So werden die vom Webserver httpd ausgelieferten Dateien unter `/var/www/html` erwartet. Liegen sie z.B. unter `/data/www/pages`, dann verbietet der Kontext, in dem der Webserver läuft, das Ausliefern von Seiten aus diesem Verzeichnis. Im Dateisystem sieht das wie folgt aus:

```
[root@shutdown ~]# ls -Zd /data/www/pages
drwxr-xr-x  root root user_u:object_r:root_t           /data/www/pages
[root@shutdown ~]# ls -Zd /var/www/html/
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t /var/www/html/
```

ls -Z gibt den SELinux-Kontext von Dateien und Verzeichnissen an. /var/www/html hat den Kontext **httpd_sys_content_t** – mit diesem Kontext versehen Dateien und Verzeichnisse dürfen vom Webserver ausgeliefert werden, fehlt dieser Kontext, dann verbietet der Policy Enforcement Server den Zugriff auf diese Dateien. Sollen die Dateien aber aus Partitionierungsgründen unter `/data/www/pages` liegen, da dort mehr Platz im Dateisystem verfügbar ist, so lässt sich mit **chcon** der passende Securitykontext herstellen:

```
[root@shutdown ~]# chcon -R --reference /var/www/html/ /data/www/pages/
[root@shutdown ~]# ls -Zd /data/www/pages/
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t /data/www/pages/
```

Ab sofort können auch aus `/data/www/pages` HTML-Dokumente ausgeliefert werden. Neu in diesem Verzeichnis angelegte Dokumente bekommen automatisch den korrekten SELinux-Kontext zugewiesen. CGI-Skripte werden so dennoch nicht ausgeführt, diese werden in einem anderen Kontext ausgeführt:

```
[root@shutdown ~]# ls -Zd /var/www/cgi-bin/
drwxr-xr-x  root root system_u:object_r:httpd_sys_script_exec_t /var/www/cgi-bin/
```

Das bedeutet auch, dass ohne weitere Verrenkungen CGI-Skripte nicht nach /var/www/html schreiben dürfen. Um das zu erreichen, müssen die Dateien, die geschrieben werden sollen, mit dem Kontext `httpd_sys_script_rw_t` versehen werden. So können fehlerhafte Skripte nicht dazu missbraucht werden, an Orten zu schreiben, an denen zum Beispiel im klassischen Modell „Other“ Schreibrechte hätte.

3.3.1 Booleans

Um die geladene Policy zu verändern, muss diese Policy normalerweise nach der Änderung neu geladen werden. Um dem Administrator zu ermöglichen, einige Aspekte der Policy zu ändern ohne diese neu laden zu müssen, sind **booleans** unter SELinux eingeführt worden. Eine Policy kann Regeln enthalten, die bestimmte Aspekte der Policy ein- oder ausschalten. Eine Liste der definierten Booleans kann man mit **getsebool -a** erhalten. Ein Beispiel:

Auf vielen Servern dürfen Shell-Nutzer auch aus ihren Homeverzeichnissen heraus HTML-Dateien ausliefern. Ist ein Verzeichnis `~/public_html` vorhanden, in das der Prozess httpd wechseln darf, so werden von dort Dateien ausgeliefert, wenn der URL `http://www.example.com/~username/` aufgerufen wird. Unter SELinux muss dieses Verzeichnis natürlich auch den passende Kontext haben.

Einer der Booleans für den httpd-Kontext ist **httpd_enable_homedirs**. Mit **setsebool httpd_enable_homedirs on** kann der Administrator die Auslieferung von Content aus erlauben, mit **setsebool httpd_enable_homedirs off** schaltet er sie wieder aus:

```
[root@shutdown ~]# wget -nv http://localhost/~schmitz/
http://localhost/~schmitz/:
00:36:13 ERROR 403: Forbidden.
[root@shutdown ~]# setsebool httpd_enable_homedirs on
[root@shutdown ~]# wget -nv http://localhost/~schmitz/
00:36:22 URL:http://localhost/~schmitz/ [4/4] -> "index.html" [1]
```

Diese Änderung ist nicht permanent, sie wird beim nächsten Systemneustart wieder zurückgenommen. Mit setsebool -p sind diese Veränderungen an Booleans auch über Reboots hinweg aktiv.

Booleans sind also ein schneller Weg, bestimmte Aspekte der aktiven Policy zu verändern. Leider gibt es keine erklärende Liste aller vorhandenen Booleans, man muss also anhand der mit **getsebool -a** einzusehenden Liste nach eventuell interessanten Booleans suchen – und dann anhand einer Suchmaschine feststellen, ob es wirklich das passende Modul ist.

3.3.2 Policymodule

In den ersten Versionen von SELinux benötigte man die Sourcen der aktiven Policy, um Änderungen an ihr vornehmen zu können. Diese Änderungen wurden dann zusammen mit den alten Sourcen kompiliert und es musste die komplette Policy neu geladen werden. Neuere Versionen bieten die Möglichkeit, über semodule und semanage neue Policies als Modul zu laden, ohne dass die komplette Policy neu gebaut und geladen werden muss. Neu geladene Policymodule werden nach /etc/selinux/<policymode>/modules geschrieben und sind nach einem Neustart des Rechners sofort wieder aktiv.

Wie kann man sich nun ein eigenes Policymodul bauen? Als Beispiel soll ein Webserver dienen, der als Mirror einer bekannten Distribution dient. Nachdem festgestellt wurde, dass in der geographischen Nähe kein Mirror via ftp erreichbar ist, soll nun auf diesem Server ein FTP-Server so eingerichtet werden, dass er auf die gleichen Daten zugreifen darf, auf die auch der Webserver zugreift. Anonyme User werden direkt in das Webverzeichnis „gechrooted“, so dass kein Sicherheitsproblem entsteht, wenn der FTP-Server alle Daten lesen darf, die auch der Webserver lesen darf.

Ein erster Versuch endet mit – unter anderem – den folgenden Fehlermeldungen im Messagelog:

```
type=AVC msg=audit(1199361965.500:1661): avc:  denied  { search } for
pid=8729 comm="vsftpd" name="/" dev=md0 ino=2
scontext=root:system_r:ftpd_t:s0
tcontext=system_u:object_r:httpd_sys_content_t:s0 tclass=dir
type=SYSCALL msg=audit(1199361965.500:1661): arch=40000003 syscall=12
success=no exit=-13 a0=9194c60 a1=bfbc9814 a2=e352ec a3=0 items=0
ppid=8556 pid=8729 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0
sgid=0 fsgid=0 tty=(none) comm="vsftpd" exe="/usr/sbin/vsftpd"
subj=root:system_r:ftpd_t:s0 key=(null)
```

Diese Fehlermeldung sieht sehr kryptisch aus, bedeutet aber eigentlich nur, dass „search“ für den Prozess vsftpd im Securitykontext ftpd_t verboten wurde, und zwar als versucht wurde eine Datei zu lesen, die mit dem Kontext httpd_sys_content_t versehen ist. Der Prozess vsftpd darf also nicht in dem Verzeichnis suchen. Das Tool audit2why hilft uns hier auch nicht viel weiter, da es nur darauf verweist, dass eine lokale Regel fehlt, um das zu erlauben (das ist der Output, wenn obiger Fehler an audit2why übergeben wird):

> Was caused by:
>
> Missing or disabled TE allow rule.
>
> Allow rules may exist but be disabled by boolean settings; check boolean settings.
>
> You can see the necessary allow rules by running audit2allow with this audit message as input.

Hätte man die manual page von selinux_httpd gelesen, dann wüsste man, dass man für die Dateien im Webbaum nur einen Dateikontext **public_content_t** setzen muss, damit vsftpd, httpd, rsync und samba diese Daten lesen dürfen. Dies ist nicht geschehen, deshalb muss ein lokales Policymodul gebaut werden, um dem FTP-Server das Lesen zu erlauben.

Um alle Fehlermeldungen zu sammeln, die benötigt werden, um ein Policy-Modul zu schreiben, wird SELinux mit **setenforce 0** in den permissive Mode geschaltet. Dadurch werden Denials mitgeloggt, aber die Anfragen nicht hart abgewiesen – dadurch kann das Programm weiter laufen. Man kann solche Probleme auch im enforcing Mode debuggen, dadurch, dass das Programm nach einem AVC-Denial nicht

mehr weiterläuft, muss man allerdings das Policymodul Denial für Denial erweitern, was recht zäh sein kann. Allerdings sollte man sich bewusst sein, dass man durch ein **setenforce 0** den kompletten Schutz verliert, den SELinux bietet – daher bietet sich ein Testsystem an, um Policymodule zu bauen. Auf anderen Systemen gebaute Module lassen sich ohne Probleme auf anderen Systemen in den Kernel laden, solange die gleiche Policy-Version geladen ist.

Nach dem Sammeln der Denials werden die gefundenen Fehlermeldungen an das Tool **audit2allow** übergeben, welches Policyregeln aus AVC-Denials erstellt.

```
grep vsftpd /var/log/audit/audit.log | audit2allow -m local
module local 1.0;

require {
        type ftpd_t;
        type httpd_sys_content_t;
        class dir { read search getattr };
        class file { read getattr };
}

#============= ftpd_t ==============
allow ftpd_t httpd_sys_content_t:dir { read search getattr };
allow ftpd_t httpd_sys_content_t:file { read getattr };
```

Diese – lesbaren – Regeln besagen, dass ein Programm mit dem Securitykontext ftpd_t in Verzeichnissen mit dem Kontext httpd_sys_content lesen, suchen und Verzeichnisattribute lesen darf, bei gleich gelabelten Dateien darf die Datei gelesen werden und die Attribute der Datei. Diese Policy erlaubt keinen schreibenden Zugriff auf den Dateibaum, dieser wird in diesem Fall auch nicht benötigt.

Mit dem Output von audit2allow kann SELinux direkt noch nichts anfangen, die Ausgabe muss noch in ein für semodule lesbares Format gepackt werden. Dabei bietet sich an, das Modul umzubenennen – momentan heißt es „local“. Zum kompilieren wird das Tool checkmodule benutzt, welches aus der obigen Repräsentation ein Binärformat baut – und dabei praktischerweise die erstellte Policy auch auf Korrektheit überprüft.

```
[root@centos ~]# checkmodule -M -m -o local.mod local.te
checkmodule:  loading policy configuration from local.te
checkmodule:  policy configuration loaded
checkmodule:  writing binary representation (version 6) to local.mod
```

local.mod muss dann noch in das Paketformat für semodule überführt werden, das erledigt das Tool semodule_package:

```
[root@centos ~]# semodule_package -o local.pp -m local.mod
```

Die Ausgabedatei local.pp kann dann mit **semodule** in den Kernel geladen werden.

Der Prozess lässt sich mit **audit2allow** abkürzen, da dieses checkmodule und semodule_package direkt aufrufen kann:

```
[root@centos ~]# cat foo | audit2allow -M myvsftpd
******************** IMPORTANT ***********************
To make this policy package active, execute:

semodule -i myvsftpd.pp
```

Nachdem die Policy geladen ist, kann man mit **semodule -l** feststellen, ob das Modul geladen ist:

```
[root@centos ~]# semodule -l | grep vsftpd
```

```
myvsftpd        1.0
```

Hier ist das der Fall, dieses Modul wird bei jedem Neustart des Systems erneut geladen.

3.4 Weiterer Schutz des Systems

SELinux kennt 4 weitere Booleans, die Speicherschutz ermöglichen:

allow_execmem verbietet das Ausführen von Code im Datenspeicher der Applikation.
allow_execstack verbietet das Ausführen von Code auf dem Stack.

Java- und Mono-Applikationen laufen nicht, wenn diese beiden Booleans ausgeschaltet sind – also Ausführung verbieten. Diese Applikationen können dann mit mono_exec_t oder java_exec_t gelabelt werden, dann laufen sie mit eingeschaltetem allow_execstack, auch wenn diese Booleans auf „off“ stehen. Default ist „on“ - diese beiden Booleans sollte man aber ausschalten, wenn möglich.

allow_execheap verbietet das Ausführen von Code auf dem Heap. Default ist off.
allow_execmod verbietet shared libraries das relozieren von Code. Default ist on.

4 Fazit

SELinux ist ein sehr komplexes System, sollte in den vom Distributor ausgelieferten Zustand allerdings genügend Schutz bieten, ohne im Alltag Probleme bei Arbeitsabläufen zu bieten. Es lohnt sich aber, sich mit den Grundlagen des Systems zu beschäftigen, um eventuell auftretende Fehler analysieren und reparieren zu können.

Einfache Applikationen lassen sich via system-config-selinux sehr einfach unter den Schutz einer Policy stellen, durch die vorhandenen Booleans lässt sich erwünschtes Verhalten oft ohne Eingreifen in die Policy selber erreichen.

Auf keinen Fall sollte man SELinux auf exponierten Rechnern einfach ausschalten, weil man die Fehlermeldungen nicht mehr lesen möchte. Meist gibt es einfache Lösungen für die auftretenden Probleme. Dafür sollte man sich allerdings – wie schon gesagt – zumindest in die Grundlagen von SELinux eingearbeitet haben.

5 Literaturverzeichnis

[1] http://www.suse.de/~agruen/acl/linux-acls/online/ - Posix ACLs

[2] http://kris.koehntopp.de/artikel/rwx_sonst_nix/ - Klassisches Unix-Rechte-Modell

[3] http://danwalsh.livejournal.com/ - Dan Walshs SELinux-Blog

[4]http://www.centos.org/docs/5/html/5.1/Deployment_Guide/selg-overview.html – RHEL Deployment Guide – SELinux Overview

[5]http://www.redhatmagazine.com/2007/05/04/whats-new-in-selinux-for-red-hat-enterprise-linux-5/ - Was ist neu in SELinux in Enterprise 5?

[6]http://www.redhatmagazine.com/2007/08/21/a-step-by-step-guide-to-building-a-new-selinux-policy-module/ - Policymodule selber schreiben.