

hacking

Abwehrmethoden

Angriffe auf
MS SQL 2000
und
Oracle

Sniffing
für Anfänger

Trojaner für
FreeBSD
Kernelmodule

Übernahme von
FTP-Verbindungen

SELinux
im Dienst der Sicherheit

Ausgabe 1/2003

Deutschland: EUR 7,50

Österreich: EUR 8,70

Luxembourg: EUR 8,70

Schweiz: CHF 14,80



SELinux - im Dienst der Sicherheit

Grzegorz B. Prokopski

In den letzten Jahren entstanden viele Anwendungen für das Linux-System, deren Aufgabe ist, die Systemsicherheit zu steigern. Besteht also Bedarf für eine weitere Lösung? Worin steckt die Andersartigkeit und Stärke des SELinux? Woher, von wem und auf welcher Basis kommt es? Wie funktioniert es, wer kann und wer sollte es anwenden? Diese und andere Fragen versuche ich mit diesem Artikel über *Security Enhanced Linux*, also *Linux mit erweiterten Sicherheitsfunktionalitäten*, zu beantworten. Ich lade Sie zur Lektüre ein!

Was ist das und woher kommt das

Kurz und bündig: *SELinux* ist ein System mit *MAC* (engl. *Mandatory Access Control*), oder *Zugriffskontrolle mit Zugriffsklassen*, das eine Politik der *RBAC* (engl. *Role Based Access Control*), oder *Rollenbasierter Zugriffskontrolle* verfolgt, indem es *DTAC* (engl. *Dynamical-ly Typed Access Control*), also die *Domainbasierte Zugriffskontrolle*, benutzt. Die Definition klingt ziemlich kompliziert und im Folgenden werde ich sie genauer erklären.

Die Geschichte von SELinux ist dagegen viel einfacher. Seine Entwicklung wird von der amerikanischen *NSA* (engl. *National Security Agency*) finanziert und von der Firma *Secure Computing Corp.* realisiert, die auch Eigentümer der in SELinux angewendeter Patente ist. Weder die Idee noch die Implementierung, die heute als SELinux bezeichnet wird, sind aber neu. Seit 1992 wurden im Rahmen des Projekts *Distributed Trusted Mach (DTMach)* Lösungen entwickelt, die später in das Betriebssystem *Fluke* integriert wurden. Für *Fluke* wurde wiederum in dem Projekt *Flux* die Architektur *Flask* entworfen. Auch wenn das kompliziert klingt, ist das Fazit hier, dass die *Flask*-Architektur in den *Linux-Kernel* eingliedert und das Ergebnis *SELinux* genannt wurde.

Was unter der Haube steckt

SELinux besteht aus drei wesentlichen Elementen:

Kernel

SELinux verwendet zurzeit (es war nicht immer so) den Kernelpatch *LSM* (engl. *Linux Security Modules - Linux Sicherheitsmodule*). Sie stellt Interfaces bereit, die den Zugriff zu Systemobjekten bei Benutzeraktivitäten (Öffnen

einer Datei. Anlegen eines Verzeichnisses, Anbindung an einen Datenport) kontrollieren lassen. SELinux schließt sich an das Interface an und erzwingt eine eigene Sicherheitspolitik auf dem System. Vom Gesichtspunkt des Administrators, der SELinux installiert, ist es bloß ein Kernelpatch.

Modifizierte Schlüsselanwendungen

In den meisten Fällen erlaubt SELinux die Anwendungen nicht zu verstehen, dass sie nicht unter einem gewöhnlichen Linux-System laufen. Einige für die Systemsicherheit und -basisfunktionalität kritische Programme müssen es jedoch unterstützen können. Es sind u.a. Anwendungen wie: *ssh*, *ls*, *ps*, *xdm*, *login*... Der Administrator muss bei der Installation entweder ihre modifizierte Versionen herunterladen oder die Quellen der benutzten Programme entsprechen patchen.

Regeln, oder Policy

Sie legen die Berechtigungen auf Zugriff, auf Operationen im System, das Systemverhalten fest. Sie bilden den praktisch wichtigsten Teil des SELinux-Systems, denn es sind gerade diese *Regeln*, die über das fehlerfreie Funktionieren des ganzen Systems entscheiden. In einem Teil dieses Artikels erkläre ich, wie genau die *Policy*-Elemente gestaltet werden. Ein Systemadministrator sollte prinzipiell keine eigenen *Policy*-Regeln schreiben müssen (außer wenn irgendwelche besondere Anforderungen an die Sicherheitspolitik gestellt werden), aber er muss sie manchmal an seine Bedürfnisse anpassen und vor allem verstehen können.

Das Wichtigste (Werbung)

Man kann sagen, dass sich SELinux durch Folgendes kennzeichnet:

Integrität und Vollständigkeit

Es konzentriert sich nicht, wie viele andere Lösungen (z.B. Kernelpatches), darauf, ausgewählte Sicherheitslücken zu flicken oder Probleme der Systemgestaltung (wie POSIX-Normen im Bezug auf Eigenschaften der Unix-Systeme) zu lösen. Stattdessen bietet SELinux einen ganzheitlichen Ansatz.

Erweiterbarkeit und Flexibilität

Der Systemadministrator kann in der *Policy* neue Regeln zu neu installierten Softwarepaketen aufstellen und existierende modifizieren. So kann er die Sicherheitspolitik an die momentanen Bedürfnisse anpassen.

Standardkonformität

Die Norm POSIX 6 beschreibt den MAC-Mechanismus, der Objekten-*Labelling* verwendet. So wird er auch im SELinux implementiert. Darüber hinaus hält sich SELinux trotz seiner etwas revolutionären Natur an die Standards für Unix-Systeme.

Intensive Entwicklung

SELinux wird von einer kommerziellen, staatlich finanzierten Firma entwickelt. Es ist jedoch ein Open Source Produkt, und der NSA (davon scheint ihr Vorgehen zu zeugen) liegt es an einer weiten Verbreitung und einer wirklich offenen Entwicklung des SELinux. Darüber hinaus bilden die bezahlten Programmierer zusammen mit den freiwilligen Enthusiasten, die sich dem Projekt angeschlossen haben, eine Gruppe, die gerne bei Problemen mit der Installation und Anwendung von SELinux hilft, was besonders bei einem so bahnbrechenden Projekt sehr wichtig ist.

Einiges zu MAC (und ACL)

Da SELinux MAC implementiert, lohnt es sich zu erklären, was das genau bedeutet. Die wichtigste Regel der Systeme mit MAC lautet: *der Benutzer entscheidet NICHT über die Sicherheitsmaßnahmen und Zugriffsberechtigungen zu Objekten*. Die genannten Maßnahmen und Berechtigungen werden von oben der *Sicherheitspolitik* nach definiert und finden unter SELinux ihr Abbild in den *Policy*-Formeln.

Da SELinux mit ACL-implementierenden Systemen oder auch (völlig falsch) mit einem ACL-implementierenden (engl. *Access Control Lists*) System verglichen wird, ist Folgendes anzudeuten:

- MAC wird durch ACL nicht realisiert,
- ACL sind eigentlich viel komplizierter und erfordern mehr Aufwand bei der Konfiguration und Fehlerkontrolle.

Man kann auch sagen, ACL beschäftigen sich mit *Einzelfällen*, während SELinux - wie Sie im Kurzen sehen - MAC durch *allgemeine* Regeln realisiert.

ACL wurden in die Kernel der 2.4-Serie auch deshalb nicht eingegliedert, dass die Kernelentwickler sie für keine gute Methode, Benutzerberechtigungen im System zu kontrollieren, halten. In der Praxis werden sie meist zusammen mit dem *Samba*-Server angewendet, da man dann damit Berechtigungen auf Dateien aus der Windows-Clientebene festlegen kann.

Was DTAC ist

Auf Englisch bedeutet das *Dynamically Typed Access Control*, auch als *Domainbasierte Zugriffskontrolle* bezeichnet. Die wichtigsten Merkmale von DTAC sind:

- jedes Objekt (z.B. Datei, Datenport, Netzwerkgerät) ist von einem einzigen Typ,

- der *Objektyp* wird von oben durch Regeln erzwungen (hängt nicht von der Benutzerentscheidung ab, sondern von der vom Administrator gestalteten Policy),
- in einem System können fast beliebig viele *Regeln* aufgestellt werden (in einer Größenordnung von 100 Tausend) - das System ist darauf vorbereitet,
- für jeden *Typ* wird eine Reihe von *Regeln* erstellt, die über das Systemverhalten bei Operationen an *Objekten* des gegebenen *Typs* entscheiden (mehr dazu gleich),
- es ist leichter, die Korrektheit der Regel zu überprüfen und zu testen - IBM arbeitet an einem Tool, das die Integrität und Korrektheit der Df/AC-Sicherheitspolitik automatisch kontrollieren lässt.

Was RBAC ist

Auf Englisch bedeutet das *Role Based Access Control* oder *Rollenbasierte Zugriffskontrolle*. Obwohl der Begriff nicht sehr oft benutzt wird, lohnt es sich zu wissen, dass z.B. das Standard-Unix-System RBAC implementiert. Und wirklich, wenn man es sich genauer ansieht - es gibt die sog. *Systembenutzer*, die alle eine bestimmte Reihe von Operationen ausführen dürfen (also eine bestimmte *Rolle* im System erfüllen). Ein zweites Element, das das Standard-Unix-System zu den Systemen mit RBAC qualifiziert, sind die *SUIDs*, mit denen die *Rolle* geändert werden kann, durch die die *gegebene* Operation ausgeführt wird. Und was hat das mit SELinux zu tun?

- SELinux ist ein System, das RBAC mit DTAC realisiert,
- SELinux erweitert die SUIDs-Philosophie,
- SELinux erweitert das Verständnis von Benutzerrollen.

Im Allgemeinen ermöglicht es SELinux, viel flexibler, genauer und in einer besser an gegebene Bedürfnisse angepassten Weise *Rollen* im System zu bestimmen und zu realisieren.

SELinux - ein System im System

Jetzt, wo wir die Grundbegriffe schon kennen, können wir versuchen, das Betriebsprinzip von SELinux zu bestimmen:

- ein Benutzer kann im System viele *Rollen* zugewiesen bekommen, wobei eine davon die Standardrolle ist,
- jede Benutzerrolle verfügt über eine Reihe von *Zugriffs-Domains* und eine *Domain* ist für diese *Rolle* die Standarddomain; am gegebenen Zeitpunkt kann ein Programm, das als ein bestimmter Benutzer funktioniert, nur eine *Rolle* in einer *Domain* erfüllen; zwischen *Rollen* und *Domains* kann umgeschaltet werden,
- für gewählte Kreuzungen einer *Domain* und jeden *Typs* werden *Verhaltensregeln* definiert (Richtlinien und andere) - so hängen effektive Prozessberechtigungen im System von der jeweils aktiven *Domain* ab,

- SELinux kann auch mit dem Standardmechanismus *Capabilities* zusammenarbeiten, indem es den Satz von *Capabilities* des startenden Programms je nach der Domain des Prozesses gestalten lässt, der es auszuführen sucht.

Wichtig zu verstehen ist auch die Art und Weise, wie die Regeln von SELinux mit dem Unix-Standardmodell der Berechtigungen - UGO (engl. *User, Group, Others* - Benutzer, Gruppe, Andere) - zusammenwirkt. Das Grundprinzip ist, dass SELinux die Berechtigungen für Aufgabenausführung im Vergleich zu einem System ohne SELinux *einschränkt, ohne zusätzliche Berechtigungen zu erteilen*, die der Benutzer unter dem normalen System ohne SELinux nicht hatte. Aus der praktischen Sicht ist es eine sehr treffende Haltung, da bei eventuellen Fehlern in der SELinux-Sicherheitspolitik, wenn die Policy schlimmstenfalls allen alles erlauben würde, hätten wir es mit einem gewöhnlichen Unix-System zu tun, das gar nicht so schlecht abgesichert ist (was wir ja aus der Alltagserfahrung wissen).

Der Security Context der Prozesse und Objekte

Das Konstrukt SC, oder *Sicherheitskontext* des Prozesses, lässt sich am besten an einem Beispiel erklären. Nachdem sich der Benutzer *greg* im SELinux angemeldet hat, funktioniert sein Shellprozess im folgenden Sicherheitskontext:

```
greg:user_r:user_t
```

wo:

- *greg* - Bezeichner des Unixbenutzers,
- *user_r* - die Standardrolle des Benutzers nach der Anmeldung,
- *user_t* - die Standarddomain der Rolle *user_r*.

Eigentlich besitzen Systemobjekte auch einen vollen SC, für einen TCP-Port kann SC z.B. folgend definiert werden:

```
system_u:object_r:http_port_t
```

Dabei ist nur der letzte Teil von Bedeutung: *http_port_t* wird bei Objekten als *Typ* und nicht *Domain* bezeichnet, wie es bei Prozessen der Fall ist. Auch sollte klar sein, dass eine gegebene Rolle von mehreren Unix-Benutzern verwendet (laut *Policy*-Regeln) werden kann, so wie auch mehrere Rollen auf eine Domain zugreifen können.

Die Hauptarten der SELinux-Regeln

Die SELinux-Regeln werden für die Paare *Domain-Typ* aufgestellt. Die häufigsten Typen sind:

- Definition von zugelassenen Operationen, wie Lesen,

Schreiben, Anlegen, Ausführen, Anbindung an einen Datenport usw.

- für den Benutzer zugelassene Rollenwechsel (und überhaupt eine Liste der dem Benutzer erlaubten Rollen),
- innerhalb einer Rolle zugelassene Domain-Wechsel (und überhaupt eine Liste der dem Benutzer erlaubten Domains),
- automatische (erzwungene, wie bei SUID) Rollen- bzw. Domainwechsel bei Ausführung eines Programms oder einer Operation an einem Objekt (z.B. automatische Typänderung beim Anlegen einer Datei).

Zurzeit wurden aus der *Standard-Policy* alle erzwungenen Rollenwechsel zugunsten der erzwungenen Domainwechseln entfernt.

Beispiel: ein IRC-Client und ein Webbrowser

Wenn man sich Geschichte der IRC-Clients-Fehler ansieht, durch die eine Kontrollübernahme über die Client-Maschine erfolgen könnte (wenigstens im Rahmen der Benutzerberechtigungen), sollte es nicht wundern, dass die *Standard-Policy* diese Anwendungen speziell betrachtet:

- die Domain *user_irc_t* wurde eingeführt,
- beim Aufrufen des IRC-Clients (dessen ausführbare Datei auch vom Sondertyp *irc_exec_t* ist) wird ein Domainwechsel im Security Context des Prozesses erzwungen (*user:user_r.user_t => user:user_r:user_irc_t*),
- ein in der Domain *user_irc_t* arbeitende Prozess hat sehr beschränkte Berechtigungen - eigentlich kann er nur in seine Kontroll- und Konfigurationsdateien schreiben,
- weder der Benutzer noch die Rolle unterliegen keinen Modifizierungen (es ist kein SUID).

Ähnliche Gefahren verbirgt die Verwendung von Webbrowsern. Sie sind meist ausgebaut und kompliziert, was sich leider in der Anzahl von Fehlern und daraus folgenden Gefahren widerspiegelt. Deshalb gilt es für sie analog:

- ein Typ der ausführbaren Dateien *netscape_exec_t* wurde eingeführt,
- beim *exec()* einer Datei dieses Typs wird der Prozess in die Domain *user_netscape_t* umgeschaltet und so werden seine Berechtigungen eingeschränkt:
- er darf Dateien vom Typ *user_netscape_t* und *user_netscape_rw_t* anlegen (so werden auch Konfigurationsdateien gekennzeichnet),
- er darf heruntergeladene Programme ausführen, aber nur im Rahmen der momentanen Domain,

- er darf drucken (hat Recht auf die Ausführung des Programms `lpr`).

Diese Einschränkungen tragen zur wesentlichen Reduzierung der durch Einbrüche (z.B. mit der Pufferüberlauftechnik) verursachten Schäden bei.

Rollen und Domains im Dienste der Daemons

Die Separierung der Berechtigungen der von Benutzern gestarteten Programme ist eins der Anwendungsfelder von SELinux. Da aber Linux immer noch meist in Serversysteme benutzt wird, und da die Serversysteme an die Sicherheitspolitik die höchsten Anforderungen stellen, spricht SELinux zur Einteilung von Berechtigungen unter Dienste, Daemons usw. ein zweites, viel weiteres Anwendungsfeld dafür. An Hand des vorigen Beispiels können wir erraten (und zwar richtig), dass die Rollen und Domains der Server-Prozesse die einzelnen Daemons und falls nötig einzelne Daemon-Funktionen abgrenzen und zwar besser als es mit SUIDs und Systembenutzern geschieht, denn:

- die SELinux-Policy ist flexibler bei der Zuweisung der Berechtigungen,
- keine zusätzlichen Systembenutzer sind nötig, um Funktionen zu verteilen,
- wenn der Serverprozess die ganze Zeit die Root-Berechtigungen braucht, macht das Anlegen von Benutzern keinen Sinn und Programmierer versuchen das oft noch komplizierter zu umgehen, z.B. mit `chroot` usw.

Leider müssen die Policy-Elemente separat für jede Software definiert werden (oder sich im Rahmen einer Softwareklasse um die korrekten Einträge entwickeln lassen).

Eine Beispielimplementierung dafür ist der `Bind`-Server aus der Klasse `DNS-Server`. Er braucht `root`-Berechtigungen

Glossar

- **RBAC** – ein rollenbasiertes System der Zugriffskontrolle,
- **DTAC** – ein Low-Level-Mechanismus zur Definierung der Zugriffsregeln, von SELinux bei der Realisierung von RBAC verwendet,
- **Typ** – ein Attribut jedes Zugriffsobjekts, das als einziges zur Festlegung der Zugriffsberechtigungen aus der gegebenen *Domain* verwendet wird,
- **Benutzer** – entspricht einem Benutzer im Sinne von Unix, allerdings kann ein Benutzer unter SELinux mehrere *Rollen* zugewiesen bekommen,
- **Rolle** – eine der Benutzerfunktionen; damit sie effektiv erfüllt werden kann, sind manchmal Berechtigungen für eine oder mehrere *Domains* notwendig,
- **Domain** – definiert den Satz von Berechtigungen für Objekte des angegebenen Typs,
- **Policy** – die Regel-Basis unter SELinux,
- **Security Context - Sicherheitskontext**, bezieht sich sowohl auf Objekte (für sie ist nur der *Typ* von Bedeutung), als auch auf Prozesse.

wenigstens zum Starten und oft auch zum Funktionieren (wenn er sich an dynamische Interfaces anbinden muss, die während seiner Aktivität aufgestellt werden). Dann müssen wir - egal, ob er mit den `root`- oder normalen Benutzerberechtigungen läuft - seine Berechtigungen folgend ausdehnen:

- Anbindung an die Datenports 53 TCP und UDP,
- Ablesen von Konfigurations- und Domaindateien,
- Schreiben nur in die Auslagerungsdateien.

So kann der Angreifer nicht mal bei einem Einbruch, wo dieser Daemon als `Root` funktioniert, keine Daten im System zerstören (höchstens werden falsche Antworten auf DNS-Anfragen anderer Server abgeschickt).

Type Enforcement Definitionen in der SELinux-Policy

Die *TE*-Definitionen in der Policy beziehen sich auf: Typendeklaration, erzwungene Typänderung eines Objektes (z.B. einer Plattendatei), erlaubte Typänderung (z.B. einer Konsolendatei) oder die sog. *Zugriffsvektoren* zu verschiedenartigen Objekten. Beispiele solcher Definitionen finden Sie im Listing 1.

Typen, die von anderen *Policy*-Regeln verwendet werden, müssen zuerst definiert werden. Im ersten Teil des Listing sehen Sie die Definitionen von Typen wie: die Domain `sshd_t` (die Hauptdomain für Programme des Pakets `SSH`) und Typen der Dateien, die von den Programmen dieser Domain verwendet werden. Dadurch können z.B. die Berechtigungen auf temporäre `SSH`-Dateien (der Typ `sshd_tmp_t`) von anderen temporären Dateien (meist des Typs `tmp_t`) getrennt werden.

Im zweiten Teil des Listings sehen Sie die Definitionen von erzwungenen Transitionen. Hier ist die Definition dank Makrobefehlen wie `file_type_auto_trans` besser lesbar. Der Makrobefehlmechanismus der SELinux-Policy erleichtert den Prozess der *Policy*-Erstellung bzw. -änderung außergewöhnlich. Hier besagt die Datei `file_type_auto_trans`, dass wenn ein Prozess in der Domain `sshd_t` eine Datei öffnet (z.B. indem er sie zugleich anlegt), die normalerweise die Domain `tmp_t` hätte, wird die Dateidomain automatisch auf `sshd_tmp_t` geändert. Der erste Makrobefehl `domain_auto_trans` besagt wiederum, dass wenn ein Prozess in der Domain `initrc_t` (und darin werden alle Skripten aus `/etc/init.d/` gestartet) eine Programmdatei des Typs `sshd_exec_t` ausführt, funktioniert das gestartete Programm in der Domain `sshd_t`. So eine Deklaration (und ein spezieller Typ `*_exec_t`) ist praktisch für jedes Softwarepaket erforderlich, das in `/etc/init.d/` gestartet wird.

Der dritte Listingteil enthält die Erlaubnis zur Objekttypänderung durch ein Programm aus einer bestimmten Domain. Ein Programm in der Domain `user` kann etwa den Typ einer Character-Device-Datei von `tty_device_t` auf `_tty_device_t` ändern. Wie Sie sehen, werden Konsolen, auf denen ein

Listing 1. Type Enforcement -Definitionen in der SELinux-Policy

Typen:

```
type sshd_t, domain, privuser, privrole, privlog,
    privowner;
type sshd_exec_t, file_type, exec_type, sysadmfile;
type sshd_tmp_t, file_type, sysadmfile, tmpfile;
type sshd_var_run_t, file_type, sysadmfile, pidfile;
```

Erzwungene Transitionen (Makrobefehle):

```
domain_auto_trans(initrc_t, sshd_exec_t, sshd_t)
file_type_auto_trans(sshd_t, tmp_t, sshd_tmp_t)
domain_auto_trans{sshd_t, shell_exec_t, user_t}
```

Erlaubte Transitionen (mit und ohne Makrobefehle):

```
type_change user_t tty_device_t:
    chr_file user_tty_device_t;
type_change sysadm_t tty_device_t:
    chr_file sysadm_tty_device_t;
type_change user_t sshd_devpts_t:
    chr_file user_devpts_t;
type_change sysadm_t sshd_devpts_t:
    chr_file sysadm_devpts_t;
domain_trans(sshd_t, shell_exec_t, sysadm_t)
```

Zugriffsvektoren:

```
allow sshd_t sshd_exec_t:file {
    read execute entrypoint };
allow sshd_t sshd_tmp_t:file {
    create read write getattr setattr link unlink
    rename };
allow sshd_t user_t:process transition;
```

Benutzer in der Domain `sysadm_t` arbeiten soll, besonders behandelt - er kann den Typ der Character-Device-Datei (der zu verwendenden Konsole) auf `sysadm_tty_device_t` ändern. Wie Sie sehen, werden Konsolen, auf denen ein Benutzer in der Domain `sysadm_t` arbeiten soll, besonders behandelt - er kann den Typ der Character-Device-Datei (der zu verwendenden Konsole) auf `sysadm_tty_device_t` ändern. Dies soll als zusätzlicher Ablauch- bzw. Modifizierungsschutz der Kommunikation über dieses Device dienen, wenn es von einem Prozess in der Domain `sysadm_t` benutzt wird.

Der letzte Listingteil enthält Informationen dazu, was ein Prozess aus einer gegebenen *Domain* mit einem Objekt eines gegebenen Typs anfangen darf.

Role Based Access Control Definitionen in der SELinux-Policy

Eine zweite Sorte von *Policy*-Elementen unter SELinux sind die auf den RBAC-Mechanismus bezogenen Einträge. Beispiele dafür sehen Sie im Listing 2.

Da es - wie der Name besagt - ein rollenbasiertes System ist, müssen die Rollen definiert werden. In dem Fall definieren wir auch, welche Domains die gegebene Rolle je benutzen darf. Für die Rolle `user_r` zum Beispiel ist die Basis-, Standarddomain `user_t` zugänglich sowie die früher

besprochenen Hilfsdomains `user_netscape_t` und `user_irc_t`.

Unter Umständen ist es nützlich, zwischen den Rollen auf Wunsch wechseln zu können (dazu dient das Programm `newrole`). Beispiele der erlaubten Transitionen zwischen Rollen finden Sie im zweiten Teil des Listing 2.

Im Allgemeinen ist das RBAC-System von den Unixbenutzerkonten stark unabhängig. In viele Fällen wäre es jedoch wünschenswert, konkrete Benutzer an die für sie erlaubten Rollensätze zu binden. In dem Beispiel sehen Sie, wie etwa der Benutzer `root` standardmäßig mit der Rolle `user_r` arbeitet, obwohl er auch Zugang zur Rolle `sysadm_r` hat.

Security Context von Netzwerkobjekten

SELinux bietet die interessante Möglichkeit, den *Security Context* für Systemobjekte zu definieren, die mit der Netzwerkfunktionalität verbunden sind. Dies macht klar, dass die Systemobjekte tatsächlich einen vollen SC zugewiesen bekommen, auch wenn nur der Typ verwendet wird. Im ersten Teil des Listing 3 sehen Sie, wie der Typ `http_port_t` definiert wird, der den TCP-Ports 80 und 8080 zugewiesen wird. Dadurch kann in anderen *Policy*-Elementen festgelegt werden, dass nur Prozesse aus spezifizierten Domains auf diese Ports, und genauer gesagt, auf ihren Typ zugreifen dürfen (und sich z.B. an sie anbinden). Es ist z.B. die Domain `httpd_t`, in der von einer Binärdatei des Typs gestartete `httpd_exec_t` Webserver funktioniert, für den eine erzwungene Transition gerade in die Domain `httpd_t` beim Starten angeordnet wird, hnlich können die Berechtigungen für Netzwerkinterfaces geregelt werden.

Standardmäßige Security Contexts der Dateien

Es ist bekannt, dass jede Datei im Dateisystem als Objekt einen SC, und darin einen *Typ* zugewiesen hat. Sehen wir uns diese etwas näher an. Der Datei-SC (und vor allem der Typ) wird nicht

Listing 2. Definitionen der Role Based Access Control in der SELinux Policy

Rollen:

```
role system_r types {
    kernel_t initrc_t getty_t klogd_t };
role user_r types {
    user_t user_netscape_t user_irc_t };
role sysadm_r types { sysadm_t run_init_t };
```

Erlaubte Transitionen:

```
allow system_r { user_r sysadm_r };
allow user_r sysadm_r;
allow sysadm_r system_r;
```

Der erkannte Benutzer:

```
user system_u roles system_r;
user root roles { user_r sysadm_r };
user jdoe roles user_r;
```

Listing 3. Definitionen der Sicherheitskontexte (SC) der Systemobjekte

Netzwerkkontexte - Ports:

```
portcon tcp 80 system_u:object_r:http_port_t
portcon tcp 8080 system_u:object_r:http_port_t
```

Netzwerkkontexte - Netzwerkinterfaces:

```
netifcon eth0 system_u:object_r:netif_eth0_t
system_u:object_r:netmsg_eth0_t
netifcon eth1 system_u:object_r:netif_eth1_t
system_u:object_r:netmsg_eth1_t
```

Standardmäßige Dateienkontexte:

```
/home system_u:object_r:home_root_t
/home/[^/]+ -d system_u:object_r:
user_home_dir_t
/home/[^/]+/. + system_u:object_r:
user_home_t
/home/[^/]+/.ssh(/.*)? system_u:object_r:
user_home_ssh_t
```

automatisch beim Systemstart mit einem SELinux-Kernel vergeben. Sie müssen in dem Verfahren von *Labelling*, noch bevor wir das System mit SELinux zu benutzen beginnen, verliehen werden. Die Frage ist nun: *woher wissen wir, welchen Typ die gegebene Datei haben sollte?*

Wenn wir die SELinux-Quellen installieren, findet sich die Antwort in der Datei *file_contexts*, einem Baustein der *Policy*. Sie enthält Definitionen wie diese im zweiten Teil des Listing 3. Sie informieren das *Labelling*-Programm, welche Typen (oder genauer, welcher *Sicherheitskontexte*) es den verschiedenen Dateien zuweisen soll. Im präsentierten Beispiel sehen Sie, wie das Verzeichnis */home* den Typ *home_root_t* erhält, die Benutzerverzeichnisse - *user_home_dir_t*, und die darin enthaltenen Dateien und Verzeichnisse - den Typ *user_home_t*. Die von *SSH* benutzten Dateien in den Benutzerverzeichnissen werden speziell behandelt, indem sie einen eigenen Typ verliehen *user_home_ssh_t* bekommen. Ähnliche Definitionen gibt es z.B. für die von *IRC*-Clients benutzten Verzeichnisse mit Kontroll- und Konfigurationsdateien.

Die Installation von SELinux

Jetzt, mit dem Basiswissen über SELinux ausgerüstet, können wir es praktisch anwenden. Um SELinux zu benutzen, brauchen wir:

- einen Kernelpatch mit LSM und SELinux,
- Patches für einige wichtige Systemtools (*login*, *ssh*, *ls*, *ps*, *xm*,...),
- die SELinux-spezifischen Systemtools.

Anstatt sich mit Patches Umstände zu machen, können wir auch von der NSA-Website die vollen modifizierten Quellen des

Kernels und der Anwendungen. Für Debian-Distributionen (stabil und instabil) stehen APT-Quellen mit allen zur Installation und Nutzung von SELinux notwendigen Elementen zur Verfügung.

Im gepatchten Kernel sind die im Kasten genannten Optionen anzuschalten. Da wir den Kernel sowieso selbst kompilieren müssen, verwenden wir lieber nicht das Image von *initrd* – sonst muss die *Policy* nach jeder Bearbeitung auch in dieses Image eingegliedert werden, was zeitaufwändig und umständlich ist.

Nachdem der Kernel und die für SELinux modifizierten Tools installiert worden sind, muss noch das *Labelling* durchgeführt werden. Vorher ist aber noch zu überprüfen, ob die Pfade in der Datei *file_contexts* für unsere Distribution geeignet sind. Wenn wir die NSA-Quellen benutzten, sind die Pfade standardmäßig für RedHat aufbereitet. Bei Debian mit Pakete aus einer passenden APT-Quelle sind sie auch entsprechend korrigiert. Die Benutzer anderer Distributionen sollten den Inhalt dieser Datei unbedingt überprüfen. Dann kann das *Labelling* beginnen.

Die SELinux-Distribution beinhaltet die Datei *Makefile* mit folgenden Zielen:

- *install* - Kompilierung und Installation der *Policy*,
- *load* - Kompilierung, Installation und Laden der *Policy*,
- *reload* - Kompilierung, Installation und Laden (Neuladen) der *Policy*,
- *relabel* - wiederholtes *Labelling* des Dateisystems (FCs),
- *policy* - lokale Kompilierung der *Policy* für Testzwecke.

Das *Labelling* kann ziemlich lange dauern, da sie an jeder Datei im lokalen Dateisystem arbeitet. Nachdem sie abgeschlossen ist, bemerken Sie, dass im Wurzelverzeichnis jeder Partition das Verzeichnis *...security* mit einigen Binärdateien entstanden ist - die gerade die zusätzlichen Informationen zu allen Dateien auf der Partition enthalten. Mit dem installierten SELinux-Kernel und den passenden Programmen können wir uns jetzt die praktischen Tests vornehmen.

Der Erststart

Auch wenn alles mit gehöriger Sorgfalt durchgeführt wurde, werden beim Erststart bestimmt mehrere Fehlermeldungen der SELinux-Erweiterung angezeigt. Trotzdem sollte das System ganz starten (warum, sehen Sie gleich). Eine (eine!) Beispielfehlermeldung sehen Sie in Listing 4.

Sie informiert, dass der Prozess */bin/login* mit dem SC *system_u:system_r:local_login_t* versucht hat, in das Socket */dev/log* mit dem Typ *device_t* zu schreiben. Diese Meldung ist

Listing 4. Eine Beispielfehlermeldung beim Start des SELinux

```
avc: denied { write } for pid=1112 exe=/bin/login
path=/dev/log dev=03:01 ino=15 scontext=system_u:
system_r: local_login_t
tcontext=system_u:object_r:device_t tclass=sock_file
```

Optionen, die im gepatchten Kernel anzuschalten sind

Networking Options

```
[Y] Network Packet Filtering
```

Security Options

```
[Y] Enable different security models
[Y] Capabilities Support
[Y] NSA SELinux Support
[Y] NSA SELinux Development Support
```

wirklich aufgetreten und war durch die Nicht-Anpassung der Policy an den von mir benutzten Daemon `syslog-ng` verursacht. Nachdem ich der Policy zwei Zeilen beigefügt hatte (die mir Rüssel Coker, Debian- und SELinux-Entwickler, geraten hat), war das Problem gelöst.

Da bei der Kernelkonfiguration die Option `NSA SELinux Development Support` markiert wurde, startet SELinux im `permissive`-Modus, in dem zwar alle durch gesperrten Zugriff verursachten Fehler gemeldet werden, der Zugriff auf Objekte selbst aber nicht abgelehnt wird. Das System arbeitet dann wie eine normales Linux, meldet jedoch Fehler. Es gibt mehrere Methoden, es in den `enforcing`-Modus umzuschalten, wo die Policy-Regeln wirklich eingesetzt werden:

- `NSA SELinux Development Support` auszuschalten,
- mit dem Befehl `avc_toggle` zwischen den Modi zuwechseln (wenn die `Policy` das erlaubt - z.B. die Rückkehr in den `permissive`-Modus kann gesperrt werden),
- den Parameter `permissive=1` an den Kernel zu überreichen.

Das System können Sie im `enforcing`-Modus erst dann starten, wenn Sie sicher sind, dass die `Policy`-Regeln entsprechend modifiziert worden sind und das System in diesem Modus arbeiten kann. Versuchen Sie nicht alle gemeldeten Fehler zu beseitigen, da nicht alle kritisch sind und manche können sich aus der definierten Sicherheitspolitik ergeben. Ich mache Sie vor allem auf die Prozesse aufmerksam, die in der Domain `initrc_t` (`ps -e -context` zeigt die Prozesskontexte an) funktionieren bzw. zu funktionieren versuchen. Die damit verbundenen Fehlermeldungen bedeuten, dass für das Paket des gegebenen Programms höchst wahrscheinlich die `Policy`-Einträge fehlen und der SC beim Start aus den Skripten in `/etc/init.d/` nicht auf den für das Paket richtigen geändert wird. Überprüfen Sie auch, ob die Pfade in der Datei `file_contexts` korrekt sind. Wenn das in Frage kommende Paket in der `Policy` überhaupt nicht berücksichtigt wird, bitten Sie am besten die SELinux-Entwickler um Hilfe bei der Erstellung passender Einträge. Auch das Skript `scripts/newrules.pl`, das neue `allow`-Regeln anhand der gemeldeten Fehler zu erstellen versucht, kann manchmal behilflich sein.

Nach dem Systemstart können Sie sich anmelden.

Nützliche Befehle

- `avc_toggle` - schaltet `permissive/enforcing` um
- `avc_enforcing` - überprüft den aktuellen Modus
- `ps -e -context` - zeigt die Prozess-SCs an
- `ls -context` - zeigt die Dateien-SCs an
- `chcon` - lässt die Dateien-SCs ändern

Wenn Sie das als `root` aus einer lokalen Konsole tun, können Sie auf die Rolle `sysadm_r` (die anders als die standardmäßige `user_r` echte Möglichkeiten, das System zu verwalten, gibt) direkt zugreifen. Ein Fragment des Anmeldebildschirms wurde in Listing 5 dargestellt.

Die Rolle, mit der `root` arbeitet, kann auch später (innerhalb der Grenzen der `Policy`) mit dem Befehl `newrole -r sysadm_r` geändert werden. Der Zugang zur Rolle `sysadm_r` erfordert normalerweise die wiederholte Eingabe des `root`-Passworts.

Nützliche Informationen

Wenn wir einen grafischen Anmeldemanager verwenden, wie `xdm`, `gdm` oder `kdm`, muss er gepatcht werden, bevor er unter SELinux arbeiten kann. Wenn wir ihn nicht patchen wollen, können wir immer `startx` beim Start der grafischen Konsole aus der Text-Konsole benutzen (der Domainwechsel, der sonst von `*dm` ausgeführt werden müsste, erfolgte schon bei der Anmeldung).

Es ist gut zu wissen, ob wir im `enforcing`- oder `permissive`-Modus arbeiten. Dies kann mit dem Befehl `avc_enforcing` ermittelt werden.

Die Daemons, die im `permissive`-Modus in der Domain `initrc_t` starten, werden fast hundertprozentig sicher im `enforcing`-Modus nicht korrekt funktionieren. Die `Policy` muss so bearbeitet werden, dass sie in ihren eigenen Domains arbeiten.

Nützliche Befehle sind hier `ps -e -context` und `ls --context`. Der erste überprüft, mit welchen SCs die Prozesse arbeiten, der zweite - welche Typen die Dateien haben. Für Testzwecke können die SCs der Dateien mit `chcon` geändert werden. Denken Sie daran, dass wenn Sie keine passenden Einträge in die `file_contexts` Datei einfügen, gehen diese Änderungen beim nächsten Labelling verloren.

Wenn wir zwischen einem Kernel wechseln, der mit SELinux kompiliert wurde und einem ohne, muss vor dem Einschalten des Kernels mit SELinux ein erneutes Labelling durchgeführt werden, da inzwischen Dateien entstehen konnten, die keinen SC zugewiesen bekommen haben. Andererseits steht nichts im Wege, die ganze Zeit den Kernel mit einkompiliertem

Listing 5. Der Anmeldebildschirm des Benutzers `root`

```
Your default context is root:user_r:user_t
Do you want to choose a different one? [n]y
[1] root:user_r:user_t
[2] root:sysadm_r:sysadm_t
Enter number of choice: 2
```

SELinux im *permissive*-Modus zu benutzen.

Die Policy besteht aus mehreren Elementen, die meist für ein bestimmtes Softwarepaket zuständig sind, z.B. den DNS-Server. In den meisten Fällen sind in unserem System nicht alle Komponenten installiert, für die es Policy-Elemente gibt. Daher werden wir z.B. beim Installieren von Paketen unter Debian gefragt, welche Policy-Elemente wir wirklich verwenden wollen. Wenn wir alle benutzen, macht das ca. 200 Tausend Regeln. Vielleicht lohnt es sich, diese Zahl auf die zig Tausend der wirklich passenden Regeln zu beschränken.

Berlin, Patente und ein allgemein zugängliches Root-Account

Hier muss ich die Fans der grafischen Konsole enttäuschen. Leider kann SELinux nicht wesentlich zur Lösung der Sicherheitsprobleme des aktuellen X-Protokolls, also auch des X-Window, beitragen. Das Problem ist, dass auch wenn unser xterm, IRC-Client oder Webbrowser in separaten Domains funktionieren, können sie immer noch den Inhalt anderer Fenster und die Tastatur bespitzeln. Deshalb setzt uns ein Fehler im IRC-Client immer noch einer Gefahr aus. Wenn wir uns z.B. aus dem xterminal irgendwo anmelden wollen, kann der Angreifer unser Passwort ablauschen. Vielleicht werden in den künftigen Versionen des X-Protokolls z.B. zwei Vertrauensniveaus eingeführt: Das niedrigere ohne beschriebene Berechtigungen und das höhere so wie das jetzige. Der Druck, solche Änderungen einzuführen, scheint jedoch unbedeutend. Der Entwurf eines alternativen Protokolls und Servers, der u.a. erweiterte Sicherheitsfunktionalität beinhaltet, konnte leider keine große Masse von Fans und Entwicklern hinreißen und sich so durchsetzen.

Eine weitere Frage sind Patente. Zurzeit finanziert die NSA die Entwicklung von SELinux, die Patente gehören allerdings einer kommerziellen Firma. Es kann passieren, dass diese Firma eines Tages Geld für die Verwendung der von ihr patentierten Lösungen fordert. Die NSA hat gar nicht vor, die Patente z.B. abzukaufen (was eine unerwünschte Präzedenz wäre), und *Secure Computing Corp.* möchte wohl kaum einen Konflikt mit der einflussreichen Regierungsagentur (da sie das erste Ziel wäre). Trotzdem wurde bisher nicht deutlich erklärt, ob das Patent unter der GPL-Lizenz vertrieben wird und daher wird SELinux wahrscheinlich nicht in die 2.5er und 2.6er Kernel eingegliedert. Wenn die Lizenz- und Patentfragen geklärt werden, sollte die Software mit den ersten 2.7ern integriert werden.

Russell Coker, Debian-Entwickler und einer der wenigen unbezahlten SELinux-Developer, hat die Möglichkeiten des SELinux in einem interessanten Experiment demonstriert. Er hat nämlich einen Rechner im Internet bereitgestellt, dessen Root-Passwort allgemein bekannt war. Es hat sich erwiesen, dass man zwar als Root z.B. chroot ausführen, aber im System

praktisch nichts anstellen konnte. Die angewendeten Policy-Regeln waren nicht restriktiver als die Standardeinstellungen, und hier und da waren sie sogar loser, so dass jeder die Systemkontrollaufzeichnungen ablesen kann und nachlesen, warum die gegebene Operation nicht ausgeführt werden konnte. Meiner Meinung nach zeugt das von der Stärke der SELinux-Idee.

Und was jetzt? Benutzen!

Russell Coker, gefragt nach den Möglichkeiten, SELinux anzuwenden, antwortete, er wende es überall an: auf den Workstations, auf dem Laptop, auf den Servern. Ich persönlich glaube jedoch, das SELinux vor allem in Serversystemen angewendet werden sollte und wird, da dort die Sicherheitsanforderungen oft sehr hoch sind.

Zurzeit kommt SELinux ans Tageslicht. Es kann sich auf folgende Weise weiterentwickeln und verbreiten:

- Eingliederung in den offiziellen Kernel - *LSM* sind schon im 2.5er Zweig vorhanden, und SELinux ist (dem Footprint nach, im Vergleich zu *LSM*) eine kleine Ergänzung, die NSA unbedingt integrieren will; sobald die Lizenz- und Patentfragen geklärt sind, sehen wir SELinux möglicherweise sogar noch in den späteren 2.6er Kernein.
- Erweiterung der Softwaregruppe, die durch die Standard-*Policy* unterstützt wird (so dass die Administratoren keine Regeln mehr schreiben müssen) Integration mit den einzelnen Distributionen - am besten kommt hier Debian zurecht, dessen einige Pakete dank den Bemühungen Russel Cokers mit SELinux schon zusammen arbeiten, und in der Zukunft soll SELinux zum Standardelement (auch wenn optional) der Debian-Distribution werden

Sie sollten jedoch nicht missverstehen, dass diese Technologie erst interessant werden wird. Die Tatsache ist, jetzt schon ist SELinux ein völlig einsatzfähiges, getestetes, flexibles und mächtiges Werkzeug zur Sicherheitserhöhung unter Linux-Systemen. Es verfügt über eine starke und aktive Entwicklergemeinschaft, in Kurzem wird es mit dem Standardkernel und verbreiteten Distributionen integriert. Es wäre wohl ein gewichtiger Fehler, sich diese faszinierende und nützliche Technologie entgehen zu lassen! ■

Im Internet

NSA (Download, Dokumentation, Mailinglisten)

<http://www.nsa.gov/selinux>

Debian Stable - Eintrag in *sources.list*: *deb*

<http://www.microcomaustralia.com.au/debian/stable/selinux>

Debian Unstable - Eintrag in *sources.list*: *deb*

<http://www.coker.com.au/selinux/>