Inhaltsverzeichnis

Vorwort

Die Arbeit an diesem Buch hat sehr viel Spaß gemacht, obwohl es am Ende länger gedauert hat, als ich ursprünglich dachte. Mit der Erfahrung von vier Büchern glaubte ich, den Aufwand für dieses Buch besser abschätzen zu können.

Eigentlich sollte es nur ein Buch über SELinux werden. Als Novell AppArmor unter der GPL-Lizenz veröffentlichte, entschloss ich mich, auch dieses Thema aufzunehmen. Ich vermute, dass Novell nun bis auf Weiteres SELinux nicht unterstützen wird. Um den *SUSE*-Linux-Anwendern aber auch die notwendige Dokumentation und Hilfe zukommen zu lassen, wurde diesem Thema etwa 100 Seiten gewidmet.

Das Hauptthema dieses Buches ist jedoch SELinux. Die Dokumentation von SELinux stellt sich jedoch als besonders schwierig dar. Zum einen handelt es sich um einen komplett neuen Ansatz der Zugriffskontrolle. Die Hintergründe und Ideen verständlich aufzubereiten war nicht immer einfach. Erschwerend kommt hinzu, dass SELinux sich immer noch in der Erforschung und Entwicklung befindet. Sich bewegende Ziele lassen sich nur schwer treffen. Das hat dazu geführt, dass ich kurz vor der erwarteten Fertigstellung alles verworfen und erneut von vorn begonnen habe.

In langer Tradition habe ich auch in diesem Buch wieder die Rechnernamen aus der Fernsehserie «Sesamstraße» genommen. So wurde AppArmor auf Samson und SELinux auf Supergrobi umgesetzt.

Inzwischen ist das Buch fertiggestellt und ich hoffe, dass ich es geschafft habe, die beiden Themen verständlich und nachvollziehbar darzustellen[1]. Ich habe versucht, dieses Buch so zu schreiben, dass es sowohl als Arbeitsunterlage in Schulungen als auch als Nachschlagewerk eingesetzt werden kann.

Ich glaube, dass die Sicherheit moderner Rechnersysteme ohne Mandatory-Access-Control-Systeme nicht mehr gewährleistet werden kann. AppArmor und SELinux helfen Ihnen dabei, Ihre Rechner zu schützen. Ich wünsche Ihnen viel Spaß dabei.

[1] Falls Sie anderer Meinung sind, eine Anregung haben oder einen Fehler gefunden haben, würde ich mich über eine E-Mail an *ralf@spenneberg.net* freuen.

1 Computersicherheit

1.1 Übersicht

In diesem einleitenden Kapitel möchte ich Ihnen einige Hintergründe zum Thema Computersicherheit liefern. Hierbei soll es sich aber nur um kurze Einführungen handeln, die in keiner Weise erschöpfend sind.

Allgemein beschreiben vier grundlegende Begriffe die *Informationssicherheit*. Diese sind zum Teil sogar in einem Standard festgelegt (ISO 17799, BS 7799). Da Computer Informationen (Daten) verarbeiten, können diese Begriffe direkt auf die Computersicherheit übertragen werden:

- Confidentiality
- Integrity
- Availability

Diese Begriffe will ich kurz erläutern. Die *Confidentiality* beschreibt die *Vertraulichkeit* der Daten. *Vertraulichkeit* ist eines der vier wichtigsten Sachziele in der Informationssicherheit. Sie beschreibt die Eigenschaft eines Systems, berechtigten Subjekten den Zugriff auf bestimmte Objekte (häufig elektronische oder physische Dokumente) zu gestatten und unberechtigten Subjekten den Zugriff auf diese Objekte zu verwehren.

Unter Vertraulichkeit versteht man, dass eine Information nur für Befugte zugänglich ist, Unbefugte dagegen keinen Zugang zu der Information haben. So können beispielsweise nur der Sender und der Empfänger eine Nachricht im Klartext lesen.

Die *Integrity* (*Integrität*) beschreibt die Tatsache, dass Daten über einen bestimmten Zeitraum vollständig und unverändert sind. Eine Veränderung könnte absichtlich, unabsichtlich oder durch einen technischen Fehler auftreten.

Die *Integrität* von Daten ist also gewährleistet, wenn die Daten vom angegebenen Absender stammen und vollständig sowie unverändert an dem Empfänger übertragen worden sind.

Die *Availability* oder auch *Verfügbarkeit* bezeichnet die Fähigkeit, auf die benötigten Daten jederzeit zugreifen zu können. Sie wird daher sowohl durch die Zuverlässigkeit des Systems, das die Daten zur Verfügung stellt, als auch durch die Erreichbarkeit des Systems bestimmt.

Zusätzlich zu diesen drei Hauptzielen gibt es auch noch drei weitere Ziele, die hier nur kurz erwähnt werden sollen: Authentizität, Nichtabstreitbarkeit und Verbindlichkeit. Im Weiteren will ich auf diese jedoch nicht mehr eingehen.

Allgemein werden diese Schutzziele bedroht. So kann es zu unautorisiertem Zugang kommen, falsche Daten können vorgetäuscht werden, oder der Zugang zum System kann verhindert werden. Um diesen Bedrohungen zu begegnen, ist es zunächst erforderlich, diese genau zu ermitteln und dann eine *Security-Policy* (*Sicherheitsrichtlinie*) zu erzeugen, die beschreibt, was erlaubt und was verboten ist. Hierauf kann dann ein Sicherheitsmechanismus aufbauen und die Umsetzung der Sicherheitsrichtlinie erzwingen.

Computerbetriebssysteme verfügen seit vielen Jahren über die unterschiedlichsten Sicherheitsmechanismen, die die Umsetzung von Sicherheitsrichtlinien erlauben. Hierbei handelt es sich meist um Discretionary-Access-Control-Systeme[1]. In einigen Umgebungen genügen diese einfachen Sicherheitsmechanismen nicht. Die hier gewünschten Security Policys erfordern umfangreichere und komplizierte Sicherheitsmechanismen. Hierzu gehören zum Beispiel militärische Anwendungen mit sehr hohen Geheimhaltungsanforderungen. Novell AppArmor und SELinux sind zwei konkurrierende Systeme, die derartige Security Policys umsetzen können. Dieses Buch beschäftigt sich mit der Erzeugung, Administration, Anpassung und Wartung dieser Security Policys.

1.2 Anfänge der Computersicherheit

In diesem Kapitel stelle ich kurz die Geschichte der Computersicherheit in den letzten 40 Jahren dar. Computersicherheit wird erst seit den 70er-Jahren tatsächlich untersucht, obwohl auch frühere Systeme, wie MULTICS und Atlas davon beeinflusst wurden.

Die folgende Darstellung basiert unter anderem auf der Zusammenstellung von Dokumenten durch Matt Bishop vom Computer Security Laboratory der Universität von Kalifornien in Davis (UCD). Diese ist im Internet unter *http://csrc.nist.gov/publications/history/* zu finden.

1969 beschrieb Butler Lampson [4] bereits die Grundlagen moderner Zugriffskontrollsysteme. Von ihm wurden die Begriffe Capability und Domain erstmals beschrieben. 1970 legte Willis H. Ware [8] die Grundlagen für Multi-Level-Security-Systeme und beschrieb erstmals die Anforderungen an Betriebssysteme, wenn mehrere Benutzer Daten unterschiedlicher Sensitivität verarbeiten. 1972 erweiterte James P. Anderson [9] das Multi-Level- Modell von Ware in einer Studie für die United States Air Force. Im Jahr 1973 legten David E. Bell und Leonard J. LaPadula [10] mit ihrem Dokument »Secure Computer Systems: Mathematical Foundations« die Grundlagen für das Bell-LaPadula-Modell. Dies ist bis heute das bekannteste Multi-Level-Security-Modell. Dieses Modell wird in Abschnitt 2.1.3 genauer erläutert. 1974 defi-

[1] Was das ist, wird in Abschnitt 1.2 erklärt.

nierte Butler Lampson das Referenzmonitor-Modell. Dieses, auch als Zugriffsmatrix bekannte Modell, beschreibt anhand einer Matrix, wie ein Subjekt auf ein Objekt zugreifen darf.

1979 legten Peter G. Neumann et al. die Grundlagen für ein »Provably Secure Operating System (PSOS)« [11]. Dieses stellte später die Grundlage für das Betriebssystem LOCK dar, das 1987 von O. Sami Saydjari et al. vorgestellt wurde [12]. Dieses Betriebssystem führt als erstes das Type-Enforcement ein. 1987 versuchten Clark und Wilson [5] das Bell-LaPadula- und das Biba-Modell auf kommerzielle Systeme zu übertragen. Der wesentliche Aspekt dieses Modells ist die Integritätsicherung. Damit lässt sich das Clark-Wilson-Modell gut auf Geschäftsprozesse übertragen.

1992 beschrieben Ferraiolo und Kuhn [13] das Role-Based-Access-Control-Modell. Dieses wurde in den folgenden Jahren weiterentwickelt ([14, 15]) und von dem National Institute for Standards and Technology (NIST) standardisiert (American National Standard 359-2004).

2 Access Control Systeme

2.1 DAC-, MAC- und RBAC-Systeme

Es gibt drei verschiedene Zugriffskontrollstrategien. Eine Zugriffskontrollstrategie beschreibt, auf welche Art Rechte an Objekten vergeben werden.

2.1.1 Discretionary Access Control (DAC)

Die Discretionary Access Control ist die am weitesten verbreitete Zugriffskontrollstrategie. Die DAC wird häufig auch als *Identity Based Access Control* (IBAC) bezeichnet. Frei übersetzt werden kann dies mit benutzerbestimmbarer oder identitätsbasierter Zugriffskontrollstrategie. Ob der Zugriff auf ein Object (Datei, Ressource) erlaubt wird, wird allein auf der Basis der Identität des Subjektes (Benutzer, Prozess) entschieden. Die Zugriffsrechte werden also von den Benutzern selbst und je Benutzer festgelegt.

Während dieses Modell genügt, damit ein Benutzer seine eigenen Daten schützen kann, ist es häufig nicht ausreichend, um aus Unternehmenssicht oder aus Sicht des Administrators ein System ausreichend zu schützen.

So ist es zum Beispiel auf einem UNIX-System nur unter Anwendung des DAC-Modells nicht möglich, einem Benutzer das Recht zu geben, sein Kennwort zu ändern. Hierzu benötigt der Benutzer erweiterte Rechte. Diese erhält er in Form des *SetUID*-Bits, welches für den Befehl `/usr/bin/passwd` gesetzt ist. So erhält der von dem Benutzer gestartete Prozess die Rechte des Administrators *root*. Besitzt der Prozess eine Sicherheitslücke, besteht die Gefahr, dass der Benutzer nun auf alles zugreifen kann, auf das auch *root* zugreifen kann.

2.1.2 Mandatory Access Control (MAC)

Die Mandatory Access Control ist ein Konzept für die Kontrolle und Steuerung von Zugriffsrechten auf IT-Systeme, bei der die Entscheidung über den Zugriff nicht auf der Basis der Identität des Subjektes (Benutzers, Prozesses) und des Objektes (Datei, Ressource) gefällt wird, sondern aufgrund allgemeiner Regeln und Eigenschaften des Subjektes und Objektes. Voraussetzung ist, dass Subjekte niemals direkt, sondern nur durch einen *Referenzmonitor* auf Objekte zugreifen können. Die Regeln des Referenzmonitors können nicht von individuellen Benutzern verändert werden. Da dieses Modell auf Regeln basiert, wird es häufig auch als Rule(set) Based Access Control

bezeichnet. Dies sollte nicht mit der rollenbasierten *Role Based Access Control* (*RBAC*) verwechselt werden.

Modelle der Mandatory Access Control sind vor allem dazu geeignet, die Vertraulichkeit, Integrität und Verfügbarkeit der Daten zu garantieren. Sie dienen also dazu, den Informationsfluss zu kontrollieren und so das »Abfließen« geschützter Information zu verhindern, sowie zu gewährleisten, dass sich die Daten immer in einem konsistenten Zustand befinden.

Ursprünglich wurden die MAC-Systeme vor allem im Bereich des Militärs entwickelt. Hier handelt es sich bei der Datenverarbeitung primär um sensible Informationen. Auch in anderen Bereichen, in denen sensible Informationen verarbeitet werden, wurden früh MAC-Systeme eingesetzt (Nachrichtentechnik).

Allgemein werden zwei verschiedene Arten von MAC-Modellen unterschieden: Multi Level Security und Multilateral Security.

2.1.3 Multi Level Security

Die Multi-Level-Sicherheitssysteme (*MLS*) entsprechen der ursprünglichen Form der Mandatory Access Control, die in den 70er- Jahren zuerst von Willis H. Ware [8] beschrieben wurde. Meistens wurden Implementationen auf Mainframes im militärischen oder sicherheitstechnischen Bereich verwendet. Bis heute ist diese Art der Mandatory Access Control am weitesten verbreitet. Bei den MLS-Systemen wird der Zugriff immer anhand der Schutzstufen abgebildet. Die Zugriffssicherheit bezieht sich auf den Top-down- und Bottom-up-*Informationsfluss*.

Bell LaPadula

Das *Bell-LaPadula*-Modell stellt eine Sicherheit mit Schutzstufen dar: Objekte werden vertikal (nach Schutzstufe) unterteilt, Subjekte werden einer Schutzstufe zugeordnet. Objekte und Subjekte werden dabei in gemeinsame Schutzstufen (vertraulich, geheim, streng geheim) eingeordnet. Die Schutzstufe des Objektes wird auch als *Classification* (*Klassifizierung*) und die Schutzstufe des Subjektes als *Clearance* (*Freigabe*) bezeichnet. Ein lesender Zugriff kann nur erfolgen, wenn die Clearance des Subjektes nicht niedriger ist als die Classification des Objektes (no-read-up). Diese Regel garantiert die *Vertraulichkeit* und realisiert die Zugriffskontrolle. Das Bell-LaPadula-Modell regelt aber auch den *Informationsfluss*. Ein Schreibzugriff darf nur auf ein Objekt erfolgen, dessen Classification nicht niedriger als die Clearance des Subjektes ist (no-write-down).

Durch diese zwei Regeln treten bei dem Bell-La-Padula-Modell zwei besonders schwerwiegende Probleme auf:

- Ein Benutzer mit einer hohen Clearance kann keine Anweisung an einen Benutzer mit einer niedrigen Clearance verfassen, sodass dieser sie lesen darf.

- Ein Benutzer mit niedriger Clearance kann Dokumente höherer Classification ergänzen, aber diese Änderungen nicht selbst lesen. Dieses blinde Schreiben stellt ein großes Problem für die Datenintegrität dar.

Generell ist das Ziel dieses Modells nicht die *Integrität*, sondern die Vertraulichkeit.

Biba

Das *Biba*-Modell stellt eine Umkehrung des Bell-LaPadula-Modells dar. Es dient nicht der Informationsflusskontrolle, sondern der *Integrität*ssicherung. Hier werden Informationen nicht vor dem Lesen, sondern vor Manipulation durch Unbefugte geschützt. Das Biba-Modell wird einerseits in der Informationstechnik verwendet, z.B. als Gegenmaßnahme bei Angriffen auf sicherheitsrelevante Systeme wie Firewalls, andererseits auch bei militärischen Systemen, wo es grundlegend wichtig ist, dass ein Befehl in der Kommandokette nicht modifiziert werden kann und somit eine falsche Anweisung weitergegeben wird.

Auch hier gibt es zwei Regeln, die sich jedoch entgegengesetzt den Regeln des Bell-LaPadula-Modells verhalten:

- Jedes Subjekt darf nur auf der gleichen Integritätsstufe oder einer niedrigeren schreiben.
- Jedes Subjekt darf nur auf der gleichen oder einer höheren Integritätsstufe lesen.

LoMAC

Die *Low-Watermark Mandatory Access Control* ist eine Variation des Biba-Modells, die es erlaubt, dass Subjekte hoher Integrität lesend auf Objekte niedrigerer Integrität zugreifen. Dabei wird die Integrität des lesenden Subjekts heruntergesetzt, damit dieses nicht mehr schreibend auf Objekte mit hoher Integrität zugreifen kann.

Die *Mandatory Integrity Control*, die von *Microsoft Vista* eingesetzt wird, ist eine Variante des LoMAC-Modells.

2.1.4 Multilateral Security

Der Begriff *multilaterale Sicherheitsmodelle* wird für Sicherheitssysteme verwendet, die nicht nur Top-down- oder Bottom-up-Betrachtungen anstellen wie die MLS-Modelle, sondern alle Seiten betrachten. Die multilateralen Sicherheitsmodelle werden auch als *Policy-basierende Sicherheitsmodelle* oder *regelbasierte Sicherheitssysteme* bezeichnet.

Compartment- oder Lattice-Modell

Das *Compartment*-Modell basiert auf dem Bell-LaPadula-Modell und erweitert die Zugriffe um zusätzliche Schlüsselbegriffe. Wenn Benutzer A Lesezugriff auf die Klassifizierung *Streng Vertraulich* besitzt, kann er Informationen dieser Klassifizierung lesen. Derselbe Benutzer besitzt aber keinen Zugriff auf Daten, die als *Streng Ver-*

traulich-(*Krypto*) klassifiziert sind. Nur wenn der Benutzer Zugriff die Klassifizierungen *Streng Vertraulich* und *Krypto* besitzt, kann er auf die Daten zugreifen.

Clark Wilson

Das Clark-Wilson-Modell beschreibt die Integrität von kommerziellen Systemen und ist eine Variation des klassischen MAC-Ansatzes.

1. Das System befindet sich in einem gültigen (konsistenten) Anfangszustand.
2. Der Zugriff auf das System erfolgt nur mittels explizit erlaubter Transaktionen.
3. Nur solche Transaktionen sind erlaubt, die das System unter allen Umständen in einen (neuen) gültigen Zustand bringen.

Chinese Wall

Der Ausdruck *Chinese Wall* kommt aus der Finanzbranche und entstand in den USA nach dem Aktiencrash 1929. Damals wurden Gesetze erlassen, die die Interessenskonflikte zwischen den Investmentbanken und Emissionsgeschäften verhindern sollten.

Hierbei werden die Daten zunächst zu Datensätzen zusammengefasst. Diese werden dann in Conflict-of-Interest-(CoI-)Klassen eingeteilt. Ein einfaches Beispiel verdeutlicht das (Abbildung 2.1). Hier sind insgesamt fünf Firmen aufgeführt. Werden sämtliche Informationen von einem System verwaltet, so stellt das Chinese-Wall-Modell sicher, dass ein Subjekt (Berater) nur dann ein Objekt (Daten) lesen darf, wenn eine der beiden folgenden Bedingungen zutrifft:

- Das Objekt befindet sich in einem Datensatz (Firma), auf die bereits das Subjekt zugegriffen hat, oder

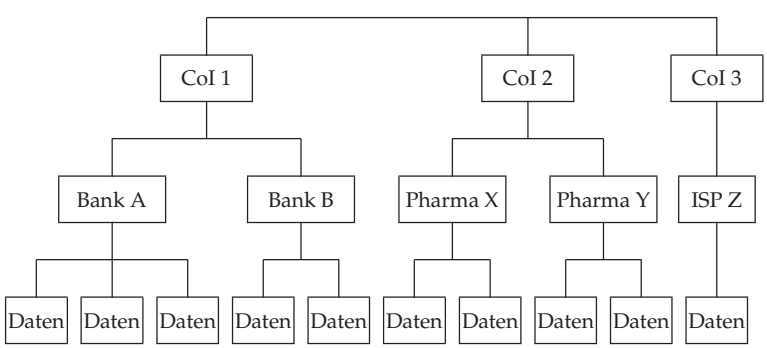

Abbildung 2.1: Chinese-Wall-Klassifizierung

- das Objekt befindet sich in einer CoI-Klasse, auf die das Subjekt bisher noch nicht zugegriffen hat.

So stellt das Chinese-Wall-Modell sicher, dass ein Subjekt nicht gleichzeitig auf die Daten zweier konkurrierender Firmen zugreifen darf.

2.1.5 Role Based Access Control (RBAC)

Role Based Access Control (*RBAC*) ist in Mehrbenutzersystemen oder Rechnernetzen ein Verfahren zur Zugriffssteuerung und -kontrolle für Dateien oder Dienste. Hierbei werden den Benutzern des Computers oder Netzwerks Rollen zugeordnet. Diese Rollen dienen der Abstraktion der Benutzer. Benutzer können dabei mehrere Rollen besitzen. Eine Rolle beschreibt dann die Funktion, die der Benutzer aktuell ausübt. Hierbei kann es sich um den Webmaster, Postmaster oder Systemadministrator handeln. Für die Ausübung dieser Funktion sind dabei je nach Rolle möglicherweise unterschiedliche Zugriffsberechtigungen notwendig.

Wegen der dreistufigen Gliederung in Benutzer, Rollen und Gruppen ist es möglich, Zugriffsrechte eines Benutzers über eine Rollenzuordung und daran gebundene Gruppenzuordnungen zu kontrollieren.

RBAC kann grundsätzlich in Kombination mit DAC- oder MAC- Systemen eingesetzt werden. In der Praxis wird es jedoch meist zur Vereinfachung der Administration von Mandatory-Access-Control-Systemen genutzt.

Diese Mandatory-Access-Control-Systeme werden heute meist in abgewandelter Form oder auch in Kombination eingesetzt. Lediglich im militärischen Bereich finden diese Systeme sich noch in ihrer Reinform.

2.2 Type Enforcement

Type-Enforcement ist ein Mandatory-Access-Control-Mechanismus. Er erlaubt die Implementierung der oben erwähnten Modelle. Beim Type Enforcement erhält jedes Subjekt (Prozess) ein Domänenattribut und jedes Objekt erhält ein Typattribut. Dieser Unterschied ist rein sprachlicher Natur. Das Type Enforcement ist eine Vereinfachung des *Domain and Type Enforcement* (*DTE*), das zusätzlich noch zwischen Domänen und Typen unterscheidet. Diese Attribute werden auch als Label bezeichnet. Diese Label werden üblicherweise der Funktion entsprechend gewählt.

Der Mechanismus verfügt dann über Regeln, die den Zugriff der Domänen auf die Typen steuern. Hierbei werden meist zwei wesentliche Ziele verfolgt:

- Die Prozesse können ihre Funktion erfüllen.
- Die Prozesse erhalten nur die maximal notwendigen Zugriffsrechte.

In der praktischen Anwendung existieren meist für jede Applikation eigene Domänen und Typen, da jede Applikation eigene Zugriffsrechte benötigt. Jede Applikation läuft dann in ihrer eigenen Domäne, und die Domäne bestimmt dann eindeutig die Rechte der Applikation. So arbeiten die Applikationen getrennt voneinander

und können nicht auf die Daten anderer Applikationen zugreifen, außer die Type Enforcement Policy (TE) erlaubt dies.

Hinweis

Die Firma Secure Computing Corporation besitzt ein Patent für die Technologie des Type Enforcement. Bezüglich der Verwendung dieser Technologie in SELinux hat die Secure Computing Corporation ein Statement veröffentlicht, in welchem sie auf die Anwendung ihrer Rechte in Bezug auf SELinux verzichtet[1]. Unabhängig von dieser Tatsache sind die betroffenen Patente inzwischen auch abgelaufen:

	Veröffentlichung	Eingereicht	Ablaufdatum
US4713753	15. Dezember 1987	21. Februar 1985	21. Februar 2005
US4621321	4. November 1986	16. Februrar 1984	16. Februar 2004
US4701840	20. Oktober 1987	20. Juni 1986	20. Juni 2006

Sie können die Patente auf der folgenden Webseite einsehen: *http://patft.uspto.gov/netahtml/PTO/srchnum.htm*

2.3 Linux-Capabilitys

2.3.1 Was ist eine Capability?

Wörtlich übersetzt bedeutet der Begriff *Capability* Fähigkeit. Es existieren viele verschiedene Definitionen für den Begriff *Capability* im Zusammenhang mit Computersystemen. Während man allgemein ein Token, das von einem Prozess verwendet wird, um den Zugriff auf ein Objekt zur erhalten, als Capability bezeichnen kann, verstehen wir unter diesem Begriff die Capability, wie sie bei der Definition der POSIX-Capabilitys beschrieben wurde.

Bei den POSIX-Capabilitys handelt es sich um den Versuch, die Allmacht von *root* in unterschiedliche Privilegien aufzuteilen. In dem POSIX-Draft 1003.1e sollten die Capabilitys standardisiert werden. Leider hat sich dieser Draft nie durchgesetzt.

Jeder Prozess besitzt nun drei Bitmaps, in denen die Capabilitys gespeichert werden:

- Vererbbare Capabilitys (Inheritable, I)
- Erlaubte Capabilitys (Permitted, P)
- Effektive Capabilitys (Effective, E)

Jede einzelne Capability wird als Bit in diesen Bitmaps dargestellt. Möchte nun ein Prozess eine privilegierte Operation durchführen, so prüft der Kernel, ob der Prozess über die entsprechende Capability verfügt. Vor Einführung der Capabilitys wurde

[1] *http://www.securecomputing.com/pdf/Statement_of_Assurance.pdf*

lediglich geprüft, ob der Prozess die effektive UID 0 (*root*) hatte. Möchte ein Prozess beispielsweise die IP-Adresse ändern, so benötigt er die Capability *CAP_NET_ADMIN*.

Der Satz der erlaubten Capabilitys definiert, welche Capabilitys der Prozess aktiv nutzen darf. Ein Prozess kann eine Capability vorübergehend in dem Satz der effektiven Capabilitys deaktivieren und später wieder aktivieren. Mehr Capabilitys, als sich im Satz der erlaubten Capabilitys befinden, kann er aber nicht benutzen.

Die vererbbaren Capabilitys geben an, welche Capabilitys der aktuelle Prozess auf neue Prozesse vererben kann. Dies bezieht sich lediglich auf Prozesse, die mit `exec()` aufgerufen werden. Prozesse, die mit `fork()` oder `clone()` gestartet werden, erhalten eine exakte Kopie der Capability-Sets des Elternprozesses.

Aktuell unterstützt Linux Capabilitys nur für Prozesse. Die POSIX- Capabilitys forderten auch diese Funktion für ausführbare Dateien. Dies ist aktuell unter Linux nur mit einem Patch erreichbar (siehe Abschnitt 2.3.4).

2.3.2 Welche Capabilitys existieren?

Insgesamt existieren in Abhängigkeit der Kernel-Version etwa 30 verschiedene Capabilitys. Diese werden im Folgenden in ihrer alphabetischen Reihenfolge aufgeführt. Dabei wird bei jeder Capability die Nummer des Bits im Capability-Set genannt.

- *CAP_CHOWN* (0): Diese Capability erlaubt die Änderung des Eigentümers und der Gruppe von Dateien. Hierbei können beliebige neue Eigentümer und Gruppen gewählt werden. Normalerweise darf lediglich der Eigentümer die Gruppe modifizieren und dabei eine Gruppe wählen, deren Mitglied er ist.
- *CAP_DAC_OVERRIDE* (1): Diese Capability erlaubt den Zugriff auf Dateien, ohne die hierzu notwendigen Rechte zu besitzen. Die einzige Ausnahme ist das Setzen der Immutable- Attribute (siehe *CAP_LINUX_IMMUTABLE*).
- *CAP_DAC_READ_SEARCH* (2): Dies erlaubt das Lesen und Durchsuchen von Verzeichnissen, ohne die erforderlichen Rechte zu besitzen. Auch hier ist die einzige Ausnahme der Zugriff, der durch *CAP_LINUX_IMMUTABLE* definiert wird.
- *CAP_FOWNER* (3): Mit dieser Capability dürfen Sie auf beliebigen Dateien die Änderungen durchführen, die normalerweise nur der Eigentümer durchführen darf. Gemeinsam mit der Capability *CAP_CHOWN* dürfen Sie bei beliebigen Dateien beliebige Besitzer und Gruppen eintragen.
- *CAP_FSETID* (4): Diese Capability erlaubt es Ihnen, die SetUID- und SetGID-Rechte auf Dateien zu setzen, die nicht Ihnen gehören.
- *CAP_KILL* (5): Diese Capability erlaubt das Senden von Signalen an Prozesse, deren Eigentümer Sie nicht sind.
- *CAP_SETGID* (6): Diese Capability erlaubt das Setzen der effektiven Gruppe des aktuellen Prozesses.
- *CAP_SETUID* (7): Diese Capability erlaubt es, den effektiven Benutzer des aktuellen Prozesses zu setzen.

- *CAP_SETPCAP* (8): Mit dieser Capability können Sie die Capabilitys aus ihrem erlaubten Satz bei jedem Prozess entfernen oder hinzufügen.
- *CAP_LINUX_IMMUTABLE* (9): Diese Capability erlaubt das Setzen und Entfernen der Attribute *immutable* und *append* bei Dateien auf einem Ext2/Ext3-Dateisystem mit dem Befehl `chattr`. Dies funktioniert auch bei XFS-Dateisystemen.
- *CAP_NET_BIND_SERVICE* (10): Diese Capability erlaubt die Verwendung von TCP- und UDP-Sockets < 1024.
- *CAP_NET_BROADCAST* (11): Mit dieser Capability darf ein Prozess Broadcast-Pakete verschicken und Multicast-Adressen registrieren.
- *CAP_NET_ADMIN* (12): Dies ist eine der mächtigsten Capabilitys. Sie umfasst die folgenden Fähigkeiten:
  - Konfiguration der Netzwerkkarten
  - Administration der Firewallregeln
  - Setzen der Debug-Option für Sockets
  - Administration der Routing-Tabellen
  - Ändern der Eigentümer der Sockets
  - Verwendung beliebiger IP-Adressen für transparente Proxies
  - Setzen der TOS-Bits
  - Aktivieren des Promiscuous-Modus einer Netzwerkkarte für das Sniffing
  - Löschen der Treiberstatistiken
  - Senden von Multicast-Paketen
  - Lesen und Schreiben der Register von Netzwerkkarten
- *CAP_NET_RAW* (13): Diese Capability erlaubt die Verwendung von RAW- und PACKET-Sockets. Der Befehl `ping` benötigt zum Beispiel diese Capability zum Versenden von ICMP-Paketen.
- *CAP_IPC_LOCK* (14): Diese Capability erlaubt das Locking von Shared-Memory-Segmenten. Außerdem dürfen die Aufrufe `mlock` und `mlockall` verwendet werden.
- *CAP_IPC_ONWER* (15): Dies erlaubt den Zugriff auf Message Queues, deren Besitzer nicht der Besitzer des Prozesses ist.
- *CAP_SYS_MODULE* (16): Diese Capability erlaubt die beliebige Modifikation des Kernels einschließlich des Ladens und Entfernens von Kernel-Modulen und der Modifikation des Capability-Bounding-Sets (siehe Abschnitt 2.3.3).
- *CAP_SYS_RAWIO* (17): Dies erlaubt direkten Zugriff auf Hardware mit `ioperm()` und `iopl()`. Außerdem erlaubt die Capability das Versenden von USB-Nachrichten via `/proc/bus/usb`.
- *CAP_SYS_CHROOT* (18): Dies erlaubt die Verwendung des System-Calls `chroot()`.
- *CAP_SYS_PTRACE* (19): Dies erlaubt die Verwendung des System-Calls `ptrace()` bei jedem beliebigen Prozess.
- *CAP_SYS_PACCT* (20): Mit dieser Capability darf das Process Accounting an- und abgeschaltet werden.

- *CAP_SYS_ADMIN* (21): Dies ist eine der weitreichendsten Capabilitys. Sie erlaubt unter anderem die folgenden Tätigkeiten:
  - Setzen des Secure Attention Keys. Dies ist eine Tastenkombination, die vor der Anmeldung eingegeben werden kann, um eine sichere Anmeldekonsole zu erhalten. Dies entspricht dem berühmten STRG+ALT+DEL unter Windows. Unter Linux ist dies mit SysRq+k möglich. Diese Tastenkombination tötet alle Prozesse auf der aktuellen Konsole, sodass der Login-Prozess neu gestartet wird.
  - Administration des Zufallszahlengenerators
  - Administration der Quota
  - Konfiguration des Kernel-Syslog.
  - Setzen des Host- und Domänennamens
  - Administration der Mounts
  - Administration des SWAP-Speichers
  - Administration der Software-RAID-Geräte
- *CAP_SYS_BOOT* (22): Reboot
- *CAP_SYS_NICE* (23): Diese Capability erlaubt das Anheben der Prioritäten von Prozessen und das Setzen der Priorität bei fremden Prozessen. Außerdem kann bei Mehrprozessorsystemen die CPU-Affinität von Prozessen definiert werden.
- *CAP_SYS_RESOURCE* (24): Dies erlaubt die Überschreitung und das Setzen von Ressourcengrenzen. Darüber hinaus erlaubt es das Überschreiten von Quoten und reservierten Speicherbereichen der Dateisysteme und die Konfiguration des Journaling-Modus bei Ext3-Dateisystemen.
- *CAP_SYS_TIME* (25): Diese Capability erlaubt das Setzen der System-Uhrzeit und der Real Time Clock in der Hardware.
- *CAP_SYS_TTY_CONFIG* (26): Dies erlaubt die Konfiguration von TTY-Geräten.
- *CAP_MKNOD* (27): Hiermit dürfen Sie Geräte mit dem Kommando `mknod` erzeugen.
- *CAP_LEASE* (28): Dies erlaubt die Erzeugung von Leases mit dem Aufruf `fcntl()` auf beliebigen Dateien. Normalerweise ist dies nur erlaubt, wenn der Besitzer des Prozesses auch Besitzer der Datei ist. Mit einer Lease wird der Prozess (Lease Holder) benachrichtigt, wenn ein weiterer Prozess (Lease Breaker) auf die Datei zugreift. Ursprünglich sind die Leases eingeführt worden, um Samba besser zu unterstützen.
- *CAP_AUDIT_WRITE* (29): Diese Capability benötigt ein Prozess, um eine Nachricht an das Audit-Subsystem im Kernel zu schicken. Lediglich vertrauenswürdige Applikationen dürfen Audit-Ereignisse protokollieren.
- *CAP_AUDIT_CONTROL* (30): Diese Capability erlaubt die Administration des Audit-Subsystems mit dem Befehl `auditctl` (siehe Abschnitt A.3).
- *CAP_FS_MASK* (31): Diese Capability entscheidet, ob die Funktion `suser()` oder `fsuser()` bei der Bestimmung der Identität genutzt wird. Jede dateisystemnahe Operation muss den `fsuser()`-Aufruf verwenden.

2.3.3 Wie setzt man die Capabilitys ein?

Die Capabilitys werden seit dem Linux-Kernel 2.2.11 unterstützt. Leider gab es in der Vergangenheit nur sehr wenige Möglichkeiten, diese einzusetzen, und nur wenige Administratoren, die diese Möglichkeiten genutzt haben.

Natürlich erlauben Mandatory-Access-Control-Systeme wie die in diesem Buch beschriebenen AppArmor und SELinux, aber auch LIDS, grsecurity, RSBAC etc. die Verwaltung dieser Capabilitys. Diese Verwaltung ist meist auch sehr flexibel und komfortabel. Aber Linux-Systeme, die diese Funktionen nicht besitzen, können auch von den Capabilitys profitieren. Die meisten Administratoren kennen nur nicht die Möglichkeiten.

Die Capabilitys können über `/proc/sys/kernel/cap-bound` gelesen und geschrieben werden. Dies ist das Capability-Bounding-Set. Dieses gibt die maximal ausübbaren Capabilitys für das gesamte System an. Hierbei wird jede Capability durch ein Bit repräsentiert. Diese Konfiguration ist aber mit Vorsicht zu nutzen. Selbst *root* kann eine Capability, die dem System entzogen wurde, nicht wieder zur Verfügung stellen. Lediglich der `init`-Prozess könnte dies tun. Da dieser aber diese Funktion nicht anbietet, können Sie über diese Datei die maximal verfügbaren Capabilitys nach dem Start des Systems konfigurieren. Eine Modifikation ist dann nur durch einen Reboot möglich. Um zum Beispiel eine Modifikation des Kernels zu verhindern, können Sie die Capability *CAP_SYS_MODULE* entfernen. Damit ein potenzieller Angreifer nicht direkt über `/dev/mem` ein Modul nachlädt oder das Capability-Bounding-Set verändert, sollten Sie auch immer die Capability *CAP_SYS_RAWIO* entfernen. Allerdings funktionieren dann einige Programme nicht mehr, die direkten Zugriff auf den Speicher und I/O-Ports benötigen. Hierzu gehört auch X.

Um das Capability-Bounding-Set zu editieren, gibt es mehrere Möglichkeiten. Zwei Programme sind auf der CD enthalten. Hierbei handelt es sich um den Befehl `lcap` der in der Debian-Distribution enthalten ist, und um das Perl-Script `syscapset`. Mit beiden Befehlen können Sie die aktuellen Capabilitys auslesen und setzen:

```
[root@supergrobi ~]# syscapset list
audit_control                   enabled
audit_write                     enabled
chown                           enabled
dac_override                    enabled
dac_read_search                 enabled
fowner                          enabled
fsetid                          enabled
....
[root@supergrobi ~]# lcap
Current capabilities: 0xFDEFFEFF
   0) *CAP_CHOWN                1) *CAP_DAC_OVERRIDE
   2) *CAP_DAC_READ_SEARCH      3) *CAP_FOWNER
   4) *CAP_FSETID               5) *CAP_KILL
```

```
 6) *CAP_SETGID                7) *CAP_SETUID
 8)  CAP_SETPCAP               9) *CAP_LINUX_IMMUTABLE
10) *CAP_NET_BIND_SERVICE     11) *CAP_NET_BROADCAST
12) *CAP_NET_ADMINyesyes           13) *CAP_NET_RAW
14) *CAP_IPC_LOCK             15) *CAP_IPC_OWNER
16) *CAP_SYS_MODULE           17) *CAP_SYS_RAWIO
18) *CAP_SYS_CHROOT           19) *CAP_SYS_PTRACE
20)  CAP_SYS_PACCT            21) *CAP_SYS_ADMIN
22) *CAP_SYS_BOOT             23) *CAP_SYS_NICE
24) *CAP_SYS_RESOURCE         25)  CAP_SYS_TIME
26) *CAP_SYS_TTY_CONFIG       27) *CAP_MKNOD
28) *CAP_LEASE                29) *CAP_AUDIT_WRITE
30) *CAP_AUDIT_CONTROL
  * = Capabilitys currently allowed
```

Um eine Capability zu entfernen, geben Sie diese einfach bei den Befehlen an:

```
[root@supergrobi ~]# lcap CAP_SYS_TIME
[root@supergrobi ~]# syscapset set sys_time
```

Sinnvoll ist der Einsatz dieser Befehle, nachdem der Bootvorgang abgeschlossen ist, sämtliche Module geladen worden sind und das System sich in einem funktionstüchtigen Zustand befindet. Capabilitys, deren Deaktivierung zu empfehlen ist, sind:

- *CAP_SYS_MODULE*
- *CAP_SYS_RAWIO*
- *CAP_LINUX_IMMUTABLE*

Natürlich müssen Sie die korrekte Funktion Ihres Systems anschließend prüfen. Auf Firewallsystemen, deren Netzwerkkonfiguration sich nicht ändert, können Sie auch überlegen, *CAP_NET_ADMIN* zu deaktivieren.

2.3.4 Filesystem-Capabilitys

Eine sehr interessante Alternative zu den klassischen und komplizierteren MAC-Systemen, die in diesem Buch vorgestellt werden, sind die *Filesystem-Capabilitys*. Diese können sicherlich kein komplexes MAC ersetzen, aber auf Systemen, wo kein MAC existiert, können die Filesystem-Capabilitys das System wesentlich sicherer machen. Ein großes Problem des UNIX-Systems ist die Tatsache, dass viele Befehle und Applikationen *root*-Privilegien verlangen, da sie einen privilegierten Port benötigen, die Datei `/etc/shadow` lesen oder schreiben müssen etc. In den meisten Fällen benötigen diese Programme aber nur wenige *root*-Privilegien. Dennoch müssen diese Applikationen mit sämtlichen *root*-Rechten ausgestattet werden. Eine schöne Alternative wäre die Zuweisung von einzelnen Capabilitys.

Für den Befehl `/bin/ping` würde dies bedeuten, dass er nicht mehr SetUID-*root* wäre, sondern lediglich die Capability *CAP_NET_RAW* erhalten würde. Dies erlauben die

Filesystem-Capabilitys. Diese benötigen einen Kernel-Patch, damit sie funktionieren. Dieser wurde von Serge E. Hallyn[2] auf der *linux-security-module*-Mailingliste veröffentlicht[3] und befindet sich auch auf der CD (`fsposixcaps.diff`). Zusätzlich benötigen Sie für die Funktion die entsprechenden Befehle. Eine modifizierte Bibliothek `libcap` für Fedora-Distributionen mit den entsprechenden Befehlen findet sich auf *http://www.kaigai.gr.jp/* und auf der CD (`libcap-1.10-25.kg.3.i386.rpm`).

Wurde die Unterstützung für die Filesystem-Capabilitys installiert, so kann anschließend der Befehl `/bin/ping` entsprechend angepasst werden. Hierzu entfernen Sie zunächst das SetUID-Bit von dem Befehl:

```
[spenneb@supergrobi ~]$ ls -l /bin/ping
-rwsr-xr-x 1 root root 41652 12. Apr 10:56 /bin/ping
[spenneb@supergrobi ~]$ ping -c 1 localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.314 ms
--- localhost ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.314/0.314/0.314/0.000 ms
[spenneb@supergrobi ~]$ sudo chmod 755 /bin/ping
[spenneb@supergrobi ~]$ ping -c 1 localhost
ping: icmp open socket: Die Operation ist nicht erlaubt
```

Bei dem anschließenden Test darf ein einfacher Benutzer den Befehl `/bin/ping` nicht mehr ausführen. Nun setzen wir die Capability *CAP_NET_RAW* im effektiven (e) und erlaubten (p) Capability-Set für den Befehl. Anschließend darf der Benutzer wieder den Befehl `/bin/ping` benutzen. Mit dem Befehl `attr` kann man sich die zusätzlich zur Datei gespeicherten erweiterten Attribute anzeigen lassen.

```
[spenneb@supergrobi ~]$ sudo setfcaps cap_net_raw=ep /bin/ping
[spenneb@supergrobi ~]$ ping -c 1 localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.314 ms
--- localhost ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.314/0.314/0.314/0.000 ms
[spenneb@supergrobi ~]$ attr -l /bin/ping
Attribute "capability" has a 16 byte value for /bin/ping
```

Chris Friedhoff hat auf seiner Webseite *http://www.friedhoff.org/fscaps.html* weitergehende Hinweise zur Konfiguration der Filesystem-Capabilitys zusammengetragen.

[2] Ein alternativer Ansatz wurde von Olaf Dietsche veröffentlicht (*http://www.olafdietsche.de/linux/capability/*).

[3] *http://marc.info/?l=linux-security-module&m=116287121812676&w=2*

Achtung

Mit den Filesystem-Capabilitys können Sie die Fähigkeiten eines Benutzers erweitern. Wenn Sie zum Beispiel dem Befehl `modprobe` die Capability *CAP_SYS_MODULE* zuweisen, kann jeder Benutzer mit diesem Befehl Kernel-Module nachladen. Dies kann von Vorteil sein, aber kann auch Sicherheitslücken erzeugen. Ein großer Vorteil von AppArmor und SELinux ist, dass jede Überprüfung, ob ein Vorgang erlaubt ist, von zwei Access-Control-Systemen durchgeführt wird. Sie müssen als normaler Linux-Benutzer zunächst die Rechte besitzen, und das MAC muss auch den Zugriff erlauben. Mit AppArmor oder SELinux können Sie nicht über das DAC-System hinaus weitergehende Rechte zuweisen. Bei Verwendung der Filesystem-Capabilities können Sie dagegen das Linux-Rechte-System aushebeln.

2.4 Alternative MAC-Systeme

Für den Linux-Administrator bieten sich viele verschiedene Mandatory-Access-Control-Systeme an. Diese Systeme verfolgen alle leicht unterschiedliche Philosophien, wie und in welcher Form der Zugriff gestattet wird.

Im Folgenden will ich die bekanntesten und am häufigsten eingesetzten Systeme, die im weiteren Buch nicht mehr besprochen werden, kurz beschreiben und vergleichen. Dieser Vergleich erhebt keinerlei Anspruch auf Vollständigkeit. In einigen Fällen mag ich auch bestimmte Eigenschaften eines Systems nicht erwähnen, da ich die Systeme teilweise nicht so gut kenne wie Novell AppArmor und SELinux. Ich würde mich über entsprechende Hinweise aber freuen und diese in zukünftige Versionen des Buches aufnehmen.

Genauer betrachtet werden:

- Linux Intrusion Detection System (*LIDS*)
- GetRewted Security (*grsecurity*)
- RuleSet Based Access Control (*RSBAC*)

2.5 Linux Intrusion Detection System (LIDS)

Das Linux Intrusion Detection System (*http://www.lids.org*) wurde am 15. Oktober 1999 von Xie Hua Gang auf der Linux Kernel Mailinglist vorgestellt (*http://lkml.org/lkml/1999/10/15/197*). Nachdem es mehrere Jahre still um das Projekt geworden war, existiert seit dem 09. Mai 2007 eine neue Version für den Kernel 2.6.21.

LIDS besteht aus einem Kernel-Patch und Userspace-Werkzeugen (`lidsadm` und `lidsconf`). Mit dem Befehl `lidsconf` definiert der Administrator die Zugriffsrechte. Am einfachsten erklärt sich die Verwendung an einem einfachen Beispiel:

```
/sbin/lidsconf -A -R -o /sbin -j READONLY
/sbin/lidsconf -A -R -o /boot -j READONLY
/sbin/lidsconf -A -R -o /bin -j READONLY
/sbin/lidsconf -A -R -o /lib -j READONLY
/sbin/lidsconf -A -R -o /usr -j READONLY
/sbin/lidsconf -A -R -o /etc -j READONLY
/sbin/lidsconf -A -R -o /etc/lids -j DENY
/sbin/lidsconf -A -R -o /etc/shadow -j DENY
/sbin/lidsconf -A -o /var/log/wtmp -j WRITE
/sbin/lidsconf -A -s /bin/login -o /etc/shadow -j READONLY
/sbin/lidsconf -A -s /bin/su    -o /etc/shadow -j READONLY
/sbin/lidsconf -A -s /bin/login -o CAP_SETUID -j GRANT
```

Hiermit wird allen Benutzern (auch *root*) nur lesender Zugriff auf die Dateien in den Verzeichnissen `/sbin`, `/boot`, `/bin`, `/lib`, `/usr` und `/etc` gegeben. Die Datei `/etc/shadow` und sämtliche Dateien in `/etc/lids` dürfen von keinem Benutzer (auch nicht von *root*) gelesen werden. Lediglich die Befehle `su` und `login` erhalten Leserechte an der Datei `/etc/shadow`.

LIDS kennt kein Default-Deny bei dem Zugriff auf die Dateien. Sie müssen den Zugriff auf jede einzelne Datei mit dem Befehl `lidsconf` definieren. Da Sie hier aber auch ganze Verzeichnisbäume angeben können, hält sich die Anzahl der Regeln in Grenzen. Häufig werden nicht mehr als 100 Regeln für ein kleines Linux-System benötigt.

Mit dem Befehl `lidsadm` erfolgt die Administration. Zunächst können Sie hiermit LIDS aktivieren. Dies erfolgt normalerweise nach dem Start sämtlicher Dienste als letzter Schritt. Dadurch müssen die Regeln nicht alle Vorgänge berücksichtigen, die bei dem Boot erfolgen. Hier ist LIDS noch nicht aktiv. Dies reduziert die Anzahl der Regeln erheblich. Natürlich können Sie LIDS auch wieder deaktivieren. Hier bietet LIDS aber einige ganz interessante Funktionen. Sie können LIDS zum Beispiel nur für die Konsole deaktivieren, an der Sie gerade angemeldet sind. Das restliche System und alle Prozesse werden dann noch überwacht. Dies können Sie zusätzlich einschränken und die Deaktivierung über das Netzwerk verbieten.

Um die Erzeugung der Regeln zu vereinfachen, können Sie mit dem Befehl `lidsadm` auch einen Lernmodus (ACL Discovery Mode) aktivieren. LIDS protokolliert dann jeden Zugriff. Für die Analyse der Protokolle und die Erzeugung der benötigten Regeln steht ein Perl-Skript (`lids_acl_discovery.pl`) zur Verfügung.

Mit einem zusätzlichen Patch kann LIDS mit `iptables` auch die Netzwerkpakete analysieren und überwachen.

2.5.1 GetRewted Security (grsecurity)

Das Projekt *grsecurity* (*http://www.grsecurity.net*) wurde 2001 begonnen und über mehrere Jahre von Brad Spengler als eigener Kernel-Patch für den Kernel 2.4 und 2.6 gepflegt. Es bietet neben den Funktionen eines Role-Based-Mandatory-Access-

Control-Systems noch viele weitere Funktionen. Die folgenden Funktionen wurden der grsecurity-Webseite entnommen und sinngemäß übersetzt:

- Härtung des Chroot
- Sicherung des `/tmp`-Verzeichnisses gegen Race Conditions
- Randomisierung des User-Stacks, der Bibliotheken und des Heap
- Randomisierung des Kernel-Stacks
- Einschränkung der Sicht der Benutzer auf ihre Prozesse
- Protokollierung der IP-Adresse der Benutzer, die die Richtlinien verletzen

Achtung

Das Projekt *grsecurity* ist leider nur ein Ein-Mann-Projekt. Dadurch gab es in der Vergangenheit bereits einmal ein Problem. Am 01.06.2006 gab der Entwickler bekannt, das Projekt bis zum 07.06.2006 einzustellen, da er verschuldet sei und die Sponsoren des Projekts ihn nicht finanziell unterstützen würden. Am 09.06.2006 war das Projekt dank neuer Sponsoren und Spenden doch gerettet worden. Jedoch zeigt dies, wie wacklig die Unterstützung sein kann.

Die Administration des RBAC-Systems erfolgt mit dem Befehl `gradm`. Zusätzlich ist eine Datei `/etc/grsec/policy` erforderlich, in der das *grsecurity*-RBAC-System konfiguriert wird. Im Folgenden ist ein Auszug aus einer Beispieldatei angegeben:

```
role admin sA
subject / rvka
        / rwcdmlxi

role default G
role_transitions admin
subject /
        /               r
        /opt            rx
        /home           rwxcd
        /mnt            rw
        /dev
        /dev/grsec      h
        /dev/urandom    r
        /dev/random     r
        ...

subject /usr/bin/ssh
        /etc/ssh/ssh_config r
```

Hier sind nun zwei Rollen dargestellt. Nach der Anmeldung als Benutzer arbeiten alle Anwender in der Rolle *default*. In dieser Rolle sind nur bestimmte Zugriffe erlaubt.

Der Zugriff mit beliebigen Befehlen (`subject /`) ist auf dem gesamten Rechner (`/`) zunächst nur mit Leserechten (`r`) erlaubt. Das Verzeichnis `/opt` darf gelesen und geschrieben werden. Das Verzeichnis `/dev/grsec` wird vor allen Prozessen versteckt (hidden, `h`). Wird das Kommando `/usr/bin/ssh` aufgerufen, darf es lediglich die Datei `/etc/ssh/ssh_config` lesen. Mehr Rechte erhält ein Benutzer nur, wenn er in die Rolle *admin* wechselt. Dieser Rollenwechsel ist erlaubt (`role_transitions admin`). Die Rolle *admin* erfordert eine Authentifizierung bei dem Rollenwechsel mit `gradm -a admin`. Dann hat der Benutzer uneingeschränkten Zugriff. Das hier verwendete Kennwort wird mit `gradm -P admin` gesetzt und muss nicht mit dem *root*-Kennwort übereinstimmen.

Interessant ist die Tatsache, dass *grsecurity* auch die Netzwerkverbindungen überwachen kann. Mit einem Patch kann hier auch das Kommando `iptables` genutzt werden.

2.5.2 RuleSet Based Access Control (RSBAC)

RSBAC wird seit 1996 von Amon Ott entwickelt. Ursprünglich begann RSBAC als Diplomarbeit im Fachbereich Informatik an der Universität Hamburg. Die erste Veröffentlichung als Version 0.9 für den Linux- Kernel 2.0.30 erfolgte im Januar 1998.

RSBAC geht einen anderen Weg, indem es zunächst nur ein Framework darstellt. Innerhalb dieses Frameworks werden die Zugriffsanfragen bearbeitet. Dabei können die unterschiedlichsten Zugriffsmodelle realisiert werden. Es können auch mehrere Modelle gleichzeitig geladen und aktiv sein. Aktuell unterstützt RSBAC die folgenden Modelle:

- *MAC*: Mandatory-Access-Control-Modell nach *Bell-LaPadula*
- *RC*: Role-Compatibility-Modell. Jeder Benutzer erhält eine Rolle, die alle von ihm gestarteten Prozesse erben. Dieses Modell hat eine gewisse Ähnlichkeit mit dem *Type-Enforcement* von SELinux.
- *FC*: Ein einfaches funktionales Rollenmodell, das zugunsten von RC aufgegeben wurde.
- *AUTH*: Dieses Modul überwacht die Benutzer und regelt, welcher Prozess welchen Benutzerwechsel durchführen darf.
- *UM*: Dieses User-Management-Modul im Kernel ergänzt oder ersetzt die Benutzerverwaltung unter Linux.
- *ACL*: Mit den Access Control Lists (ACLs) prüft RSBAC, ob ein Subjekt (Benutzer, Rolle, ACL-Gruppe) auf ein Objekt zugreifen darf.
- *PAX*: Dies ist der PaX(PageExec)-Patch, der auch Bestandteil von grsecurity ist. Hiermit werden Schwächen in der Speicherverwaltung vermieden.
- *DAZ*: Das Dazuko-Modul erlaubt einen On-Access-Scan von Dateien mit einem Virenscanner.
- *CAP*: Erlaubt die Definition von maximalen Capabilitys für einzelne Prozesse.
- *JAIL*: Dieses Modul härtet das Chroot.

- *RES*: Mit diesem Modul können für einzelne Prozesse Resource-Limits gesetzt werden.
- *FF*: Das File-Flags-Modul erlaubt das Setzen von Attributen auf Dateien, wie *read only*, *append only*, *write only* oder *execute only*. Diese können auch von *root* nicht modifiziert werden.
- *PM*: Das Privacy-Modul implementiert den Schutz der persönlichen Daten. Hierbei lehnt es sich das deutsche Gesetz zum Datenschutz an.

Da RSBAC derart viele Funktionen aufweist, ist es hier nicht sinnvoll, auch nur ansatzweise deren Administration aufzuzeigen. RSBAC bietet selbst genug Material für ein Buch. Bei Interesse bitte ich Sie, die Homepage *http://www.rsbac.org* aufzusuchen.

3 Benchmarks

Bei dem Einsatz von Mandatory-Access-Control-Systemen kommt immer wieder die Frage auf, wie viel *Overhead* diese Systeme erzeugen. Hierzu können Benchmarks zur Messung herangezogen werden. Auf *http://lbs.sourceforge.net/* werden verschiedene Benchmarks für Linux aufgeführt. Die meisten Benchmarks sind jedoch synthetischer Natur und können daher nicht direkt auf Applikationen übertragen werden, die in der Realität eingesetzt werden.

Ich verwende hier die Benchmarks LMBench[1] und UNIXbench[2]. Im Folgenden finden Sie die Vergleichswerte mit und ohne Mandatory-Access-Control-System.

Die Prüfungen fanden jeweils im Single-User-Modus statt, um möglichst alle Verfälschungen durch weitere laufende Applikationen zu vermeiden (Mailserver, Cron-Daemon etc.). Als Hardware wurde ein Dell Optiplex 170L eingesetzt. Dieses System hat einen Pentium-4-Prozessor mit 3 GHz und aktiviertem Hyperthreading und 512 Mbyte RAM.

Besonders der Micro-*Benchmark LMBench* zeigt bei SELinux durchaus eine Verringerung der Geschwindigkeit. Diese ist jedoch bei *UNIXbench* mit Ausnahme der gleichzeitigen Shellscript-Ausführung nicht nachzuvollziehen. Als Fazit kann man nur sagen, dass eine Applikation durch SELinux wie auch durch AppArmor gebremst wird. Ob dies aber signifikant ist, kann nur von Applikation zu Applikation durch einen Test geprüft werden. Ob ein Vergleich derartig unterschiedlicher MAC-Systeme überhaupt sinnvoll ist, sollten auch Sie selbst entscheiden. Hier stelle ich lediglich meine Ergebnisse vor. Diese mögen in Abhängigkeit der Architektur auch von System zu System unterschiedlich ausfallen.

3.1 Novell AppArmor

Die Benchmark-Prüfung erfolgte auf einer OpenSuse 10.2-Distribution mit allen Patches. Die Zeiten wurden mit und ohne Novell-AppArmor-Modul ermittelt.

Um tatsächlich auch AppArmor zu testen, wurden für die Applikationen jeweils Profile erzeugt, sodass die Benchmark-Applikationen unter der Kontrolle des AppArmor-Systems liefen. Die hierbei verwendeten Profile für die Applikationen sind im Anhang abgedruckt (siehe Abschnitt B.1).

---

[1] *http://www.bitmover.com/lmbench/*

[2] *http://www.tux.org/pub/tux/benchmarks/System/unixbench/*

Der UNIXbench-Benchmark ergab die folgenden Ergebnisse:

Test	ohne AppArmor	mit AppArmor	Verlust
Dhrystone 2 using register variables	343,9	345,5	0%
Double-Precision Whetstone	191,9	192,0	0%
Execl Throughput	694,2	477,5	32%
File Copy 1024 bufsize 2000 maxblocks	732,7	697,1	5%
File Copy 256 bufsize 500 maxblocks	506,4	476,3	6%
File Copy 4096 bufsize 8000 maxblocks	1118,9	1050,5	7%
Pipe Throughput	445,3	440,7	1%
Process Creation	792,8	785,6	1%
Shell Scripts (8 concurrent)	1048,3	934,3	11%
System Call Overhead	390,1	390,3	0%

Die Ergebnisse des LMBench-Benchmarks hier aufzuführen, möchte ich Ihnen ersparen. Sie befinden sich daher im Anhang (siehe Abschnitt C.1). Die wesentlichen Punkte möchte ich jedoch zusammenfassen. Der LMBench-Benchmark weist kaum Unterschiede mit und ohne AppArmor auf. Lediglich bei den folgenden Operationen erkennt man einen deutlichen Effekt durch AppArmor:

- AppArmor bremst die Ausführung eines Shellscriptes um den Faktor 1,2.
- Die Ausführung eines neuen Prozesses ist 35% langsamer.
- Das Öffnen und Schließen einer Datei dauert mit AppArmor doppelt so lange.

3.2 SELinux

Die Benchmark-Prüfung erfolgte auf einer Fedora Core 6-Distribution mit sämtlichen verfügbaren Patches. Die Benchmarks wurden mit und ohne SELinux-Unterstützung[3] durchgeführt. Die verwendeten Profile sind im Anhang abgedruckt (siehe Abschnitt B.2). So konnte der Geschwindigkeitsverlust durch die zusätzlichen Kontrollen durch SELinux gemessen werden. Der UNIXbench-Benchmark ergab die in der Tabelle auf der folgenden Seite gezeigten Ergebnisse.

Die Ergebnisse des LMBench sind ebenfalls im Anhang (siehe Abschnitt C.2) zu finden. Hier wieder nur in Auszügen:

- Das Öffnen und Schließen einer Datei benötigt mit SELinux 12% mehr Zeit.
- Der Start eines neuen Prozesses benötigt 5% mehr Zeit.
- Der Start eines Prozesses mit eigener Shell dauert 8% länger.
- Der Start einer Datei benötigt 23% mehr Zeit.

[3] Beim Booten wurde `selinux=0` angegeben.

- Auch das Erzeugen von Dateien dauert bis zu doppelt so lange. Das Löschen einer Datei dauert bis zu 58% länger.

Test	ohne SELinux	mit SELinux	Verlust
Dhrystone 2 using register variables	392,5	375,9	5%
Double-Precision Whetstone	198,9	192,4	4%
Execl Throughput	727,5	679,5	7%
File Copy 1024 bufsize 2000 maxblocks	612,7	585,7	5%
File Copy 256 bufsize 500 maxblocks	418,4	391,9	7%
File Copy 4096 bufsize 8000 maxblocks	1157,9	1124,7	3%
Pipe Throughput	366,4	327,5	11%
Process Creation	745,4	740,4	1%
Shell Scripts (8 concurrent)	1054,5	118,5	89%
System Call Overhead	286,2	285,4	1%

4 AppArmor-Geschichte

AppArmor wurde ursprünglich von der Firma *Wirex Communications*, Inc. (*http://www.wirex.com*) unter dem Namen *SubDomain* entwickelt. Wirex Communications wurde 1998 von Crispan Cowan gegründet. Die Firma beschäftigte sich mit der Entwicklung von freien und kommerziellen Sicherheitslösungen, die als *Immunix* bezeichnet wurden. Das erste Produkt in dieser Reihe war StackGuard. Bei *StackGuard* handelt es sich um einen modifizierten GNU C-Compiler (gcc), der den kompilierten Code so modifiziert, dass die resultierenden Applikationen immun gegen das Einschleusen und Ausführen von Code durch Buffer-Overflow-Angriffe sind. Der StackGuard wurde auf der Immunix-Homepage (*http://www.immunix.org*) veröffentlicht. Dort stellte Wirex unter dem Namen *ImmunixOS* auch eine von der Red Hat Linux 5.1-Distribution abgeleitete Distribution zur Verfügung, die komplett mit dem StackGuard Compiler übersetzt worden war.

Im Jahr 2000 stellte Wirex mit *FormatGuard* ein weiteres Werkzeug zur Abwehr zur Verfügung und reagierte damit auf die neuartigen Angriffe, die Formatstring-Schwächen ausnutzen. FormatGuard ist eine Erweiterung der Glibc-Bibliothek. Um diesen Schutz nutzen zu können, ist ebenfalls eine erneute Übersetzung der Programme erforderlich. Basierend auf Red Hat Linux 7.0 wurde mit ImmunixOS 7.0 eine Distribution zur Verfügung gestellt, die entsprechend übersetzt worden war.

Ebenfalls wurde die Kernel-Erweiterung SubDomain erstmals der Öffentlichkeit vorgestellt (*http://archives.neohapsis.com/archives/linux/immunix/2000-q4/0014.html*). Subdomain war ein Kernel-Patch für den Linux-Kernel 2.2.17 und modifizierte die System-Calls. Dadurch konnte SubDomain das Lesen und Schreiben von Dateien und das Ausführen weiterer Prozesse überwachen. Die Überwachung erfolgte mit Hilfe von Profilen, die bereits eine AppArmor-ähnliche Syntax verwenden:

```
foo {
  /etc/readme  r,
  /etc/writeme w,
  /usr/bin/bar x  {
    /usr/lib/otherread r,
    /usr/opt/otherwrite w,
  },
}
```

Ein wesentlicher Vorteil von SubDomain war die schlanke Implementierung. Die erste Version bestand aus 4500 Zeilen C-Code. Der Parser benötigte 825 Zeilen

C-Code. Die Firma Wirex Communications war maßgeblich an der Entwicklung der Linux-Security-Module-Schnittstelle (*LSM*) im Kernel beteiligt. SubDomain wurde daher im Weiteren auf diese Schnittstelle portiert. Im Jahr 2003 benannte sich Wirex Communications, Inc. nach seinem Hauptprodukt in Immunix, Inc., um. Im Jahr 2004 wurde SubDomain an die *SUSE*-Distribution angepasst. Anfang 2005 wurde SubDomain in *AppArmor* umbenannt und unter dem neuen Namen beworben. Im Mai 2005 erwarb *Novell* Inc. die Firma Immunix Inc. Das Produkt AppArmor wurde erstmals mit dem ServicePack 3 für den SUSE Linux Enterprise Server (SLES) 9 von Novell veröffentlicht. Im Januar 2006 stellte Novell AppArmor unter der GNU General Public License (GPL) zur Verfügung und gründete das AppArmor-Projekt auf *http://www.opensuse.org/Apparmor*, wo die weitere Entwicklung stattfindet.

5 AppArmor-Anwendung

In diesem Kapitel werde ich zunächst die erste Anwendung von AppArmor auf einer SUSE-Distribution vorstellen. Die Installation und Anwendung auf anderen Distributionen wird in Kapitel 7 besprochen. Weiterführende Informationen über die zur Verfügung stehenden Befehle, die Syntax und die manuelle Erzeugung von Profilen finden Sie ebenfalls in späteren Kapiteln.

5.1 Installation unter SUSE Linux

Zunächst sollten Sie sicherstellen, dass auf Ihrer SUSE-Installation alle benötigten Pakete vorhanden sind. Aktuell ist AppArmor verfügbar ab SUSE Linux 10.1 und dem *SUSE* Linux Enterprise Server 9 mit Servicepack 3.

Die benötigten Pakete bei der SUSE Linux-Distribution lauten:

- `apparmor-utils`
- `apparmor-profiles`
- `apparmor-docs`
- `libapparmor`
- `apache2-mod-apparmor`
- `apparmor-parser`

Für die Konfiguration von AppArmor über *Yast2* ist zusätzlich noch das Paket `yast2-apparmor` erforderlich. Stellen Sie zunächst sicher, dass diese Pakete installiert sind. Am einfachsten erfolgt die Installation mit Yast, das dann gleichzeitig einige Abhängigkeiten berücksichtigen kann.

Wenn Sie den SUSE Linux Enterprise Server einsetzen, haben die entsprechenden Pakete andere Namen. Die Umbenennung in AppArmor ist hier noch nicht so weit fortgeschritten wie bei der SUSE Linux-Distribution 10.1. Bei dem SLES 9 wurden die Pakete mit einem späteren Update zur Verfügung gestellt. Sie sind nicht in der original ausgelieferten Version enthalten.

- `subdomain-utils`
- `subdomain-profiles`
- `subdomain-docs`
- `mod-change-hat`
- `subdomain-parser-common`
- `yast2-subdomain`

5.2 Starten von AppArmor

Sie können jederzeit AppArmor starten und stoppen. Hierzu hat die SUSE Linux-Distribution 10.1 ein Startscript `/etc/init.d/boot.apparmor`. Dieses Startscript kann auch mit `rcapparmor` aufgerufen werden. Hierbei handelt es sich lediglich um eine Verknüpfung in dem Verzeichnis `/sbin` auf das originale Script.

Achtung

Bei der Enterprise-Version heißt dieses Script `/etc/init.d/boot.subdomain`. Entsprechend kann dieses Script auch mit dem Namen `rcsubdomain` aufgerufen werden. Diese Verknüpfung existiert aus Gründen der Rückwärtskompatibilität auch bei der SUSE Linux-Distribution.

Dieses Startscript unterstützt die folgenden Funktionen:

```
# rcapparmor
Usage: /sbin/rcapparmor start|stop|restart|try-restart|reload| ➥
       force-reload|status|kill
```

Mit `rcapparmor start` können Sie jederzeit AppArmor starten. Genauso können Sie mit `rcapparmor stop` jederzeit AppArmor anhalten.

Achtung

Dennoch müssen Sie beachten, dass nach dem Start von AppArmor zunächst kein Prozess durch AppArmor überwacht wird. Nur neu gestartete Prozesse können von AppArmor überwacht werden.

Nach einem `rcapparmor stop` ist das AppArmor-Kernel-Modul weiterhin geladen. Lediglich die Profile wurden gelöscht. Um auch das AppArmor-Kernel-Modul zu entladen, können Sie `rcapparmor kill` verwenden.

Den Zustand von AppArmor können Sie mit dem Argument `status` auslesen:

```
# rcapparmor status
apparmor module is loaded.
50 profiles are loaded.
50 profiles are in enforce mode.
0 profiles are in complain mode.
Out of 57 processes running:
```

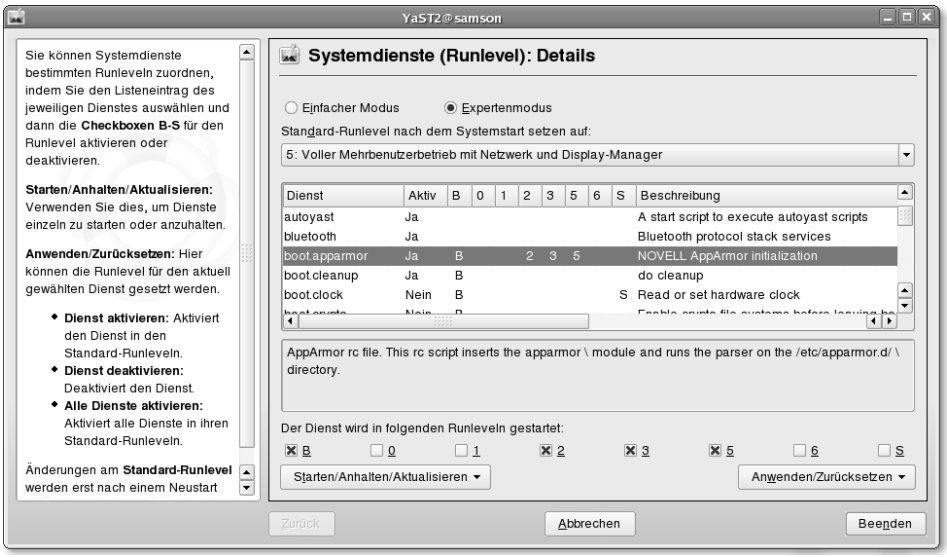

Abbildung 5.1: Mit dem Runlevel-Editor in Yast können Sie AppArmor zum Bootzeitpunkt aktivieren.

```
6 processes have profiles defined.
6 processes have profiles in enforce mode.
0 processes have profiles in complain mode.
```

Da AppArmor nur die Prozesse überwachen kann, für die es Profile besitzt und die nach dem Start von AppArmor geladen wurden, sollte AppArmor immer sehr früh zum Boot-Zeitpunkt gestartet werden. Dann kann AppArmor, entsprechende Profile vorausgesetzt, anschließend gestartete Prozesse überwachen. Um zu überprüfen, ob dies auf Ihrem System der Fall ist, können Sie in Yast den Runlevel-Editor starten oder das auf der Kommandozeile mit `chkconfig` überprüfen (siehe Abbildung 5.1).

```
# chkconfig boot.apparmor
boot.apparmor  on
```

Wird AppArmor so zum Bootzeitpunkt gestartet, werden die entsprechenden Applikationen überwacht.

5.2.1 Welche Prozesse werden überwacht?

Um nun schnell zu ermitteln, welche Prozesse bereits überwacht werden, gibt es eine Möglichkeit. Hierfür können Sie das Kommando `unconfined` benutzen. Dieses Kommando ruft den Befehl `netstat` auf und ermittelt so alle Netzwerkdienste. Wie Sie alle laufenden Prozesse auf ihren AppArmor-Status prüfen können, ist in Abschnitt 6.3.9 erläutert. Anschließend prüft `unconfined`, ob diese Netzwerkdienste von AppArmor überwacht werden. Sinnvoll ist es, dass alle über das Netzwerk er-

reichbaren Dienste als potenzielle Sicherheitslücken von AppArmor überwacht werden:

```
# unconfined
11314 /sbin/dhcpcd not confined
11470 /usr/sbin/mdnsd confined by '/usr/sbin/mdnsd (enforce)'
11470 /usr/sbin/mdnsd confined by '/usr/sbin/mdnsd (enforce)'
11487 /usr/lib/zmd/zmd-bin not confined
11527 /sbin/portmap not confined
11527 /sbin/portmap not confined
11646 /usr/sbin/cupsd not confined
11646 /usr/sbin/cupsd not confined
11728 /usr/lib/postfix/master confined by '/usr/lib/postfix/master ➥
        (enforce)'
11728 /usr/lib/postfix/master confined by '/usr/lib/postfix/master ➥
        (enforce)'
11840 /usr/sbin/sshd not confined
```

Wenn Sie lediglich die geladenen Profile anzeigen möchten, genügt der folgende Befehl:

```
# cat /sys/kernel/security/apparmor/profiles
/usr/sbin/traceroute (enforce)
/usr/sbin/squid (enforce)
/usr/sbin/sendmail (enforce)
...
```

Diese Liste zeigt Ihnen, welche Prozesse potenziell von AppArmor überwacht werden, falls das entsprechende Programm gestartet wird.

5.3 Analyse der Protokolle

AppArmor protokolliert alle Verstöße gegen die geladenen Profile in einem Protokoll. Hierfür nutzt AppArmor den *Auditd*-Daemon, der in einem eigenen Kapitel besprochen wird (siehe Anhang A). Der Auditd-Daemon protokolliert seine Meldungen in der Datei `/var/log/audit/audit.log`. Falls der Auditd-Daemon nicht aktiv ist, werden die Protokolle über den Syslogd-Daemon in die Datei `/var/log/messages` geschrieben.

Hier möchte ich Ihnen kurz den Aufbau der AppArmor-Protokollmeldungen beschreiben. Diese Protokollmeldungen helfen Ihnen sowohl bei der Fehlersuche als auch bei der Anpassung vorhandener Profile an neue Anforderungen und beim Erstellen kompletter neuer Profile.

AppArmor kann vier verschiedene Protokollmeldungen schreiben, die im Folgenden einzeln betrachtet werden sollen.

- `PERMITTING`: Diese Meldungen werden erzeugt, wenn AppArmor sich in dem Lernmodus befindet. Jeder Zugriff der Applikation im Lernmodus wird dann protokolliert.

```
type=APPARMOR msg=audit(1155035013.332:16): PERMITTING access ➥
        to capability 'net_raw' (nmap(24874) profile /usr/bin/ ➥
        nmap active /usr/bin/nmap)
type=APPARMOR msg=audit(1155035013.448:17): PERMITTING r access ➥
        to /usr/share/nmap/nmap-mac-prefixes (nmap(24874) ➥
        profile /usr/bin/nmap active /usr/bin/nmap)
```

In der ersten Meldung wird der Zugriff auf eine Capability erlaubt. Der zugreifende Prozess ist `nmap` mit der Prozessnummer 24874. Das aktive Profil `/usr/bin/nmap` überwacht dabei den Prozess `/usr/bin/nmap`. In der zweiten Meldung wird der lesende (`r`) Zugriff auf die Datei `/usr/share/nmap/nmap-mac-prefixes` demselben Prozess gewährt.

- `REJECTING`: Diese Meldungen werden erzeugt, wenn AppArmor einen Prozess überwacht und dabei den Zugriff nicht erlaubt. Häufig führt das auch zu einer Fehlfunktion der Applikation. Das Profil muss erweitert werden, um den Zugriff zu erlauben. Möglicherweise handelt es sich aber auch um einen unerwünschten bösartigen Zugriff, der von AppArmor erfolgreich abgewehrt wurde. Im Folgenden sind zwei typische Ablehnungen aufgeführt:

```
type=APPARMOR msg=audit(1155037095.706:54): REJECTING r access ➥
        to /proc/swaps (httpd2-prefork(28354) profile /usr ➥
        /sbin /httpd2-prefork active /usr/sbin/httpd2-prefork)
type=APPARMOR msg=audit(1155037095.710:57): REJECTING x access ➥
        to /bin/mount (httpd2-prefork(28354) profile /usr/sbin ➥
        /httpd2-prefork active /usr/sbin/httpd2-prefork)
```

Hier wird für den Prozess `/usr/sbin/httpd2-prefork` der lesende Zugriff auf die Datei `/proc/swaps` und der ausführende Zugriff auf die Datei `/bin/mount` abgelehnt.

- `LOGPROF-HINT`: Diese Meldungen tauchen nur im Lernmodus auf und weisen den Befehl `logprof` auf besondere Umstände hin. Ruft ein Programm im Lernmodus einen weiteren Prozess auf, so schreibt AppArmor die folgende Meldung:

```
type=APPARMOR msg=audit(1155037256.176:287): LOGPROF-HINT ➥
        changing_profile pid=28383
```

Diese Meldungen werden nur von `logprof` ausgewertet.

- `AUDITING`: Diese Meldung erscheint nur im Audit-Modus. Im Audit-Modus wird jeder Zugriff einer Applikation protokolliert. Erlaubte Zugriffe werden mit dem Schlüsselwort `AUDITING` protokolliert und verbotene Zugriffe mit dem Schlüsselwort `REJECTING`.

```
type=APPARMOR msg=audit(1155037692.339:315): AUDITING r access ➥
        to /etc/localtime (nmap(28408) profile /usr/bin/nmap ➥
        active /usr/bin/nmap)
```

5.4 AppArmor-Benachrichtigungen und -Berichte

Novell AppArmor kann auch Benachrichtigungen und Berichte erzeugen. Die Konfiguration erfolgt über Yast2. Bei den Benachrichtigungen handelt es sich um regelmäßige E-Mails, die Sie über AppArmor-Aktivitäten informieren.

Sie erreichen die Konfiguration der Benachrichtigungen über Yast2 im Menü NOVELL APPARMOR unter dem Menüpunkt APPARMOR-KONTROLLLEISTE (siehe Abbildung 5.2).

Abbildung 5.2: Über Yast2 können Sie die Benachrichtigung bei Sicherheitsereignissen aktivieren.

Dort haben Sie die Möglichkeit, zwischen drei verschiedenen Benachrichtigungen zu wählen (siehe Abbildung 5.3).

Leider ist die Konfiguration der Benachrichtigung in der aktuellen Ausgabe defekt[1] und zeigt nach erfolgter Konfiguration immer noch den Status DIE BENACHRICHTIGUNG IST DEAKTIVIERT. Die Benachrichtigungen werden jedoch versandt. Je nach Wahl der Benachrichtigung ist die versandte E-Mail sehr knapp oder ausführlicher. Eine typische ausführliche E-Mail hat den folgenden Inhalt:

```
Subject: Verbose Security Report for Samson.
Date: Sat, 19 Aug 2006 15:31:17 +0200 (CEST)
From: root@samson.spenneberg.net (AppArmor Security Notification)
```

[1] *https://bugzilla.novell.com/show_bug.cgi?id=177039*

```
The following security events occured since Thu Jan  1 01:00:00 1970:

type=APPARMOR msg=audit(1155994266.353:7635): REJECTING r access ➥
         to /var (ls(27036) profile /usr/sbin/sshd active ralf)
type=APPARMOR msg=audit(1155994267.361:7636): REJECTING x access ➥
         to /usr/bin/nail (bash(27037) profile /usr/sbin/sshd ➥
         active ralf)
type=APPARMOR msg=audit(1155994267.361:7637): REJECTING r access ➥
         to /usr/bin/nail (bash(27037) profile /usr/sbin/sshd ➥
         active ralf)
```

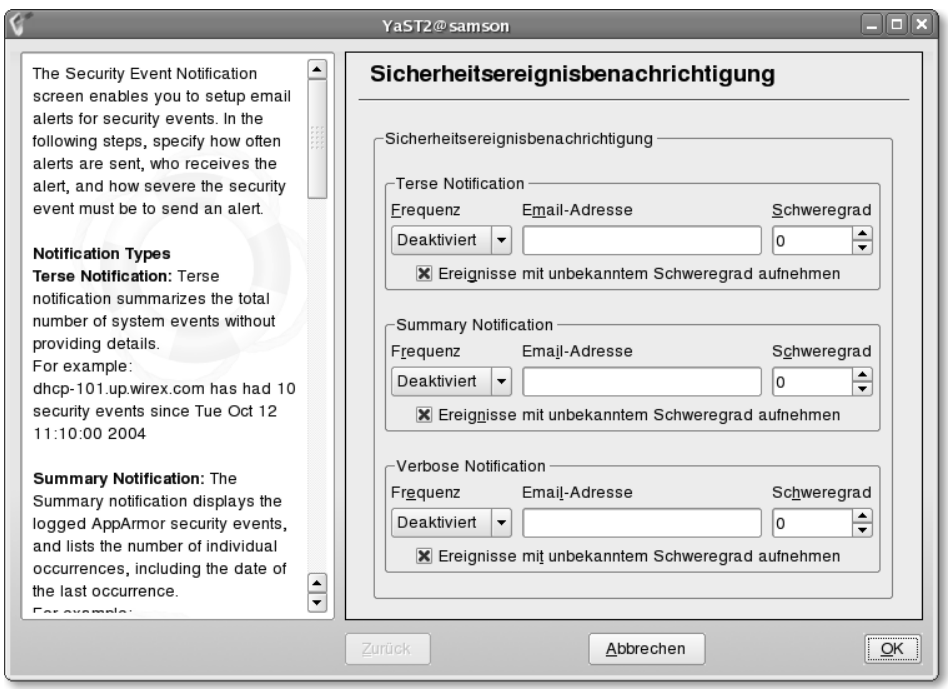

Abbildung 5.3: Über Yast2 können Sie drei verschiedene Benachrichtigungen aktivieren.

Mit der Berichtserstellung können Sie drei verschiedene Berichte erzeugen lassen:

- Executive Summary Report
- Applications Audit
- Security Incident Report

Alle Berichte lassen sich automatisch zu einem bestimmten Zeitpunkt ausführen und per E-Mail versenden oder auch bei Bedarf sofort ausführen.

5.4.1 Executive Summary Report

Der *Executive Summary Report* fasst die Ereignisse, die im Security Incident Report ausführlich dargestellt werden, in wenigen Worten zusammen, sodass die verantwortliche Person sich schnell einen Überblick über die vorgefallenen Aktivitäten machen kann.

5.4.2 Applications Audit

Der *Applications Audit* prüft, welche der laufenden Anwendungen von AppArmor überwacht werden und welche nicht. Der Bericht entspricht dem Befehl `unconfined`.

5.4.3 Security Incident Report

Der *Security Incident Report* führt alle Verstöße gegen die AppArmor-Richtlinien auf (siehe Abbildung 5.4). Dabei besteht die Möglichkeit, die Verstöße vorher zu filtern. Sie können den Zeitraum und den Namen des zu überwachenden Programms angeben.

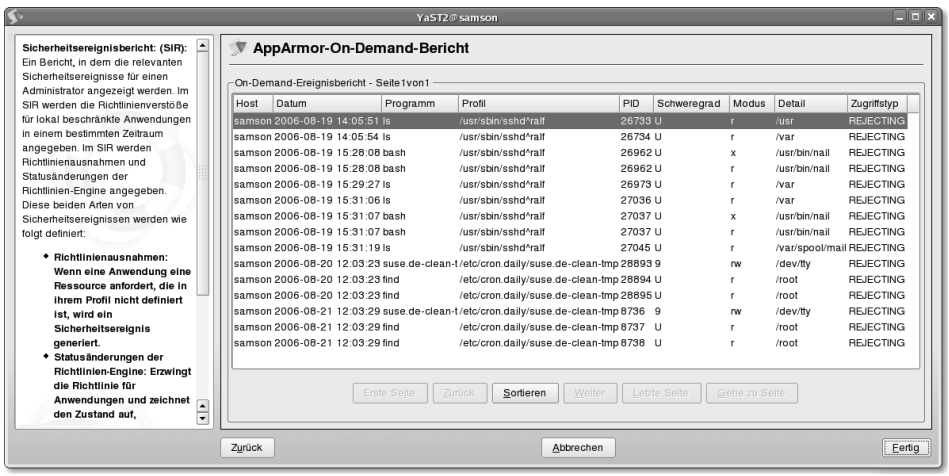

Abbildung 5.4: Mit Yast2 können Sie einen Security Incident Report anfertigen.

5.5 Erzeugen eines Profils mit Yast

SUSE Linux bietet eine Vielzahl von vorbereiteten Profilen. So ist ein gewisser Mindestschutz gewährleistet, wenn Sie AppArmor aktivieren. Enthalten sind Profile sowohl für exponierte Netzwerkdienste als auch für grafische Applikationen und Kommandozeilenbefehle, die Daten aus nicht vertrauenswürdigen Quellen verarbeiten. Letztere sind zum Beispiel der Realplayer, der Firefox-Browser und der Acrobat-Reader. Aber auch die Befehle `netstat` und `ping` gehören dazu. Obwohl die

Profile für diese Applikationen enthalten sind, sind sie nicht unbedingt aktiv. Um eine Liste aller geladenen Profile zu erhalten, können Sie diese entweder aus dem Kernel auslesen, oder Sie betrachten die in dem Verzeichnis `/etc/apparmor.d` vorhandenen Dateien. Weitere inaktive Profile befinden sich in dem Verzeichnis `/etc/apparmor/profiles/extras`. Um diese Profile zu aktivieren, müssen Sie lediglich das entsprechende Profil aus diesem Verzeichnis in das Verzeichnis `/etc/apparmor.d` kopieren. Bei dem nächsten Neustart des Systems wird das Profil aktiviert. Möchten Sie das Profil sofort aktivieren, so können Sie das mit dem Befehl `enforce` erledigen. Geben Sie lediglich den kompletten Namen des zu überwachenden Programms inklusive Pfad an.

```
# enforce /usr/bin/ethereal
Setting /usr/bin/ethereal to enforce mode.
```

Achtung

SUSE bezeichnet die in diesem Verzeichnis vorhandenen Profile als nicht ausreichend ausgereift. Die Profile wurden in vielen Fällen nicht an die aktuelle Distribution angepasst. Pfade stimmen daher nicht. Wenn Sie diese Profile verwenden möchten, müssen Sie meist die Profile noch anpassen.

Ich zeige Ihnen nun, wie Sie selbst ein Profil für ein Programm erzeugen können. Am einfachsten erfolgt das mit Yast. Ich werde in einem späteren Kapitel auch weitere Methoden vorstellen. Hier soll nun ein Profil für das Programm `nmap` erzeugt werden. Im Weiteren werde ich Ihnen die Schritte vorstellen, damit Sie diese auch auf Ihrem System direkt nachvollziehen können. Hierzu muss zunächst das Programm installiert werden. Dies erfolgt ganz einfach mit `yast -i nmap`.

Nun starten Sie *Yast2* und rufen den ASSISTENT ZUM HINZUFÜGEN VON PROFILEN auf (siehe Abb. 5.5).

In dem neuen Fenster geben Sie den Namen der zu überwachenden Anwendung mit ihrem kompletten Pfad an (siehe Abb. 5.6). Anschließend bestätigen Sie die Eingabe mit CREATE. Leider ist bei SUSE Linux 10.1 dieser Dialog noch nicht in die deutsche Sprache übersetzt worden. Nun müssen Sie den Befehl `nmap` benutzen. Rufen Sie zum Beispiel als *root* den Befehl folgendermaßen auf:

```
# nmap -sS localhost
# nmap -sS -O -v some_other_host
```

Es ist sinnvoll, möglichst alle Funktionen der Applikation zu nutzen. *Yast2* wird anschließend ein Profil erzeugen, das genau diese Funktionen erlaubt. Natürlich sollten Sie sicherstellen, dass der Befehl gerade jetzt nicht für einen Angriff genutzt werden kann. Wenn Sie den Befehl ausreichend getestet haben, klicken Sie SCAN SYSTEM LOG FOR APPARMOR EVENTS (Abbildung 5.7). Nun präsentiert Ihnen *Yast2* auf den folgenden Dialogen die Dateien und Funktionen, die der getestete Befehl genutzt hat.

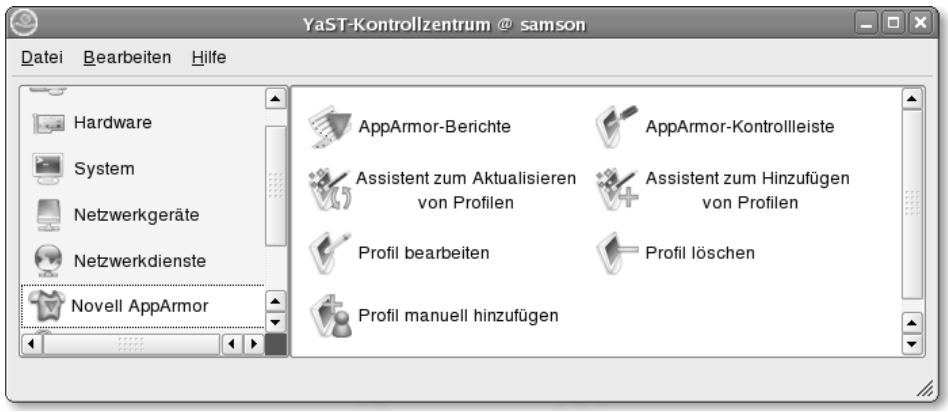

Abbildung 5.5: Über Yast2 können Sie die wichtigsten AppArmor-Funktionen steuern.

Abbildung 5.6: Geben Sie bei der Anwendung den kompletten Pfad mit an.

Sie haben hier die Möglichkeit zu entscheiden, ob das Profil den Zugriff erlauben soll oder nicht. Sie müssen nun in jedem Dialog einzeln entweder ALLOW oder DENY auswählen. Damit die Anwendung später erfolgreich funktioniert, ist ein Erlauben erforderlich (Abb. 5.8). In vielen Fällen erkennt der Assistent, dass es bereits vorgefertigte Abstraktionen gibt. Wenn Sie diese auswählen, verringert sich die Anzahl der Regeln ungemein. Hier (Abb. 5.9) erkennt der Assistent an dem Zugriff auf das Gerät `/dev/tty`, dass der Prozess auf das Terminal zugreifen möchte. Anstatt nun

Abbildung 5.7: Der Wizard fordert Sie auf, die Applikation zu verwenden.

alle notwendigen Zugriffe einzeln zu erlauben, schlägt er die Abstraktion `abstractions/consoles` vor. Diese befindet sich in dem Verzeichnis `/etc/apparmor.d`.

Allgemein bietet Ihnen der Dialog die folgenden Möglichkeiten (siehe auch Abschnitt 6.3.7):

- ALLOW erlaubt den angegebenen Zugriff auf die Ressource.
- DENY verweigert den angegebenen Zugriff auf die Ressource. Das Programm funktioniert möglicherweise nicht richtig.
- GLOB: Durch Anwählen dieses Buttons wird der letzte Pfadanteil der Ressource durch ein * ersetzt. Der Zugriff betrifft dann alle Dateien auf dieser Ebene. Ein weiteres Betätigen der Schaltfläche löscht eine weitere Ebene und ersetzt den * durch **. Hiermit wird der Zugriff auf alle Unterverzeichnisse und Dateien erlaubt.
- GLOB W/EXT: Dies entspricht dem GLOB mit dem Unterschied, dass die Dateien identische Endungen (Extensions) aufweisen müssen.
- EDIT erlaubt es Ihnen, die Pfadangabe der Ressource frei zu ändern.
- INHERIT: Wenn ein weiteres Programm aufgerufen wird, wird der entstehende Prozess ebenfalls von diesem Profil überwacht.
- PROFILE: Hiermit verlangen Sie, dass das aufgerufene Programm über ein eigenes Profil verfügt und dass dieses auch tatsächlich geladen ist. Verfügt das Programm über kein Profil, schlägt der Aufruf fehl!

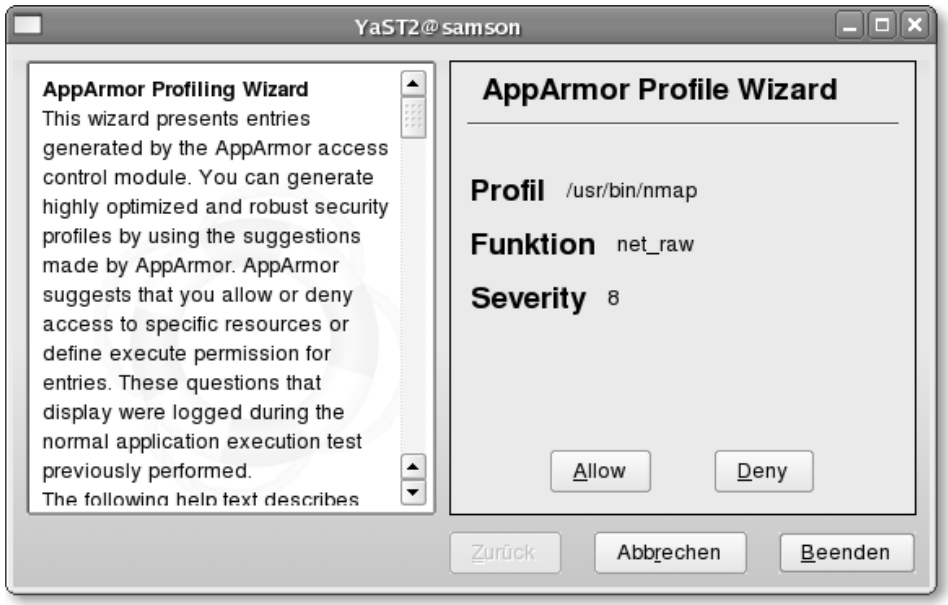

Abbildung 5.8: Nmap benötigt für die Erzeugung der Pakete für den Scan eine spezielle Capability: CAP_NET_RAW.

- UNCONFINED: Diese Funktion ist besonders gefährlich und startet den neuen Prozess ohne jede Überwachung.

Wenn Sie alle Dialoge beantwortet haben, beginnt der Assistent wieder von Neuem und bietet Ihnen an, die Protokolle zu analysieren. Sie können nun den Assistenten beenden. Der Assistent hat nun ein Profil für den Befehl Nmap erzeugt. Dieses Profil wurde in dem Verzeichnis `/etc/apparmor.d` unter dem Namen `usr.bin.nmap`[2] gespeichert und auch direkt geladen. Wenn Sie nun Nmap aufrufen, wird der Prozess von AppArmor überwacht. Im Folgenden sehen Sie das von dem Assistenten erzeugte Profil:

```
1 # vim:syntax=apparmor
2 # Last Modified: Mon Jun 26 12:10:10 2006
3 #include <tunables/global>
4
5 /usr/bin/nmap {
6  #include <abstractions/base>
7  #include <abstractions/consoles>
8  #include <abstractions/nameservice>
9
10  capability net_raw,
```

[2] Diese Namenskonvention ist gebräuchlich, aber nicht zwingend. Die Dateien dürfen jeden beliebigen Namen haben.

```
11
12  /usr/bin/nmap r,
13  /usr/share/nmap/nmap-mac-prefixes r,
14  /usr/share/nmap/nmap-os-fingerprints r,
15  /usr/share/nmap/nmap-services r,
16 }
```

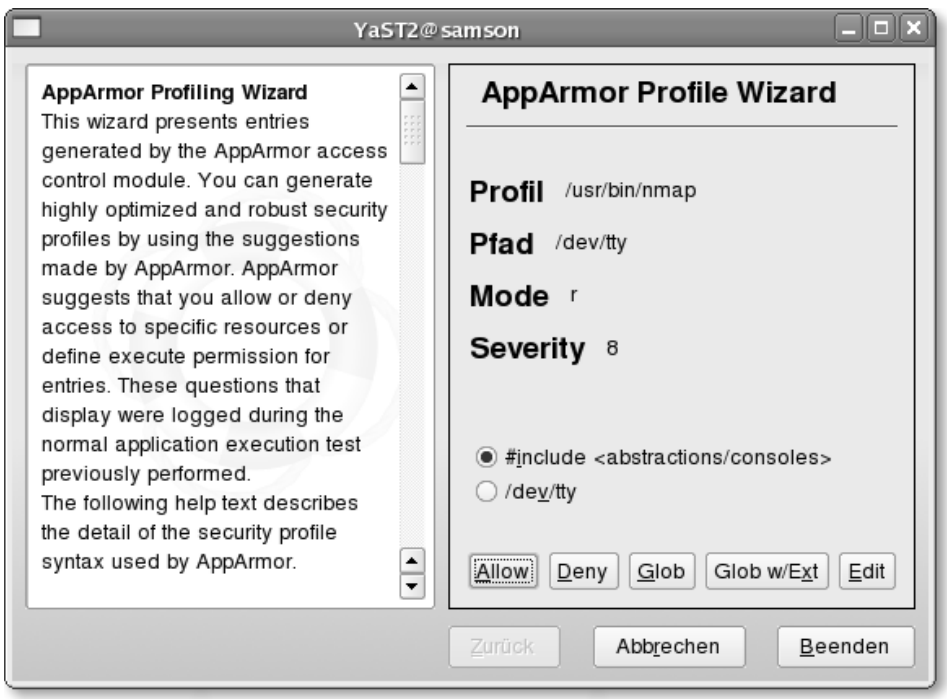

Abbildung 5.9: Stehen Abstraktionen zur Verfügung, bietet Ihnen der Assistent diese auch an.

Die Zeilennummern zu Beginn jeder Zeile wurden von mir der Lesbarkeit halber hinzugefügt. Die Zeilen 1-2 sind lediglich Kommentare und enthalten nur für den Vim einen Syntax-Highlighting-Hinweis. Leider ist in der Version SUSE Linux 10.1 keine Syntax-Beschreibung für AppArmor enthalten, obwohl die Manpage `apparmor.vim(5)` vorhanden ist. Wahrscheinlich wird ein Update dieses Problem beheben. Ab Zeile 3 beginnt das Profil. Hier wird über eine Include-Direktive der Inhalt der Datei `tunables/global` zu diesem Profil geladen. Alle Pfade in dieser Datei sind relativ zum Verzeichnis `/etc/apparmor.d`. Die geladene Datei ist daher `/etc/apparmor.d/tunables/global`. Anschließend beginnt das Profil für das Kommando `/usr/bin/nmap`. Zunächst werden hier einige Abstraktionen geladen. Wenn Sie wissen möchten, was sich in diesen Abstraktionen befindet, können Sie die entsprechenden Dateien in einem Editor öffnen.

Nun wird die Capability *CAP_NET_RAW* zur Verfügung gestellt. Insgesamt gibt es je nach Kernel bis zu 31 verschiedene Capabilitys, die in der Manpage `capabilities(7)` erläutert werden. Die Zeilen 12–15 erlauben dem Nmap-Prozess, auf die verschiedenen Dateien zuzugreifen.

Wenn Sie nun den Nmap-Befehl benutzen, wird er immer von AppArmor überwacht. Falls eine bestimmte Funktion nicht zur Verfügung steht, sollten Sie die AppArmor-Protokolle analysieren. Falls Sie dort Meldungen finden, die auf einen Fehler im Profil hinweisen, können Sie den gesamten Vorgang wiederholen. Wenn Sie vorher das bereits erzeugte Profil nicht löschen, erweitert der Assistent das Profil nur um die fehlenden Punkte.

5.6 Erzeugen eines Subprofils (Hat) mit Yast

AppArmor verfügt über die sehr interessante Funktion des Subprofils. Ein Prozess wird von AppArmor entsprechend einem Profil überwacht. Für bestimmte Funktionen kann zudem ein Subprofil verwendet werden. Dieses wird als *Hat* bezeichnet. Dieser Hat kann mehr oder weniger Funktionen als das übergeordnete Profil aufweisen.

Hinweis

Im diesem Kapitel wollen wir diese Funktion für den *Apache* Webserver nutzen. Hierzu soll für eine *PHP*-Applikation ein *Hat* erzeugt werden. Dadurch soll diese PHP-Applikation nur die durch das Profil erlaubten Funktionen nutzen dürfen.

Um diese Funktion zu nutzen, muss eine Applikation speziell an AppArmor angepasst worden sein. Aktuell ist das nur für den *Apache* Webserver der Fall. Auf der AppArmor-Homepage gibt es aber auch schon ein `pam_apparmor`-PAM-Modul, das für alle PAM-fähigen Programme ein benutzerabhängiges Subprofil aktivieren kann. Dieses PAM-Modul wird in Abschnitt 8.2.2 besprochen. Der Webserver ist insbesondere interessant, da bei Webapplikationen aus Geschwindigkeitsgründen die Scripts häufig direkt von dem Webserver ausgeführt werden. Hierfür besitzt der Apache zum Beispiel die Module `mod_php` und `mod_perl`, die direkt PHP-Scripts und Perl-Scripts ausführen können. Wenn diese Scripts nun mehr oder weniger Privilegien benötigen als der Webserver selbst, ist das nur mit einem Subprofil möglich.

Der Apache kann ein Subprofil auswählen in Abhängigkeit von

- der URI,
- der Location,
- des Verzeichnisses (Directory) oder
- des VirtualHosts.

Die Funktionen des VirtualHosts werden in einem späteren Kapitel erläutert. Hier werden wir zunächst Subprofile in Abhängigkeit von der URI erzeugen.

Stellen Sie zunächst sicher, dass sowohl der Apache 2.x-Webserver als auch die Pakete `apache2_mod_php5` und `apache2_mod_apparmor` installiert wurden. Achten Sie darauf, dass diese Namen der SUSE Linux 10.1-Distribution entnommen wurden. Ältere und neuere Distributionen verwenden hier möglicherweise andere Namen, wie `apache2_mod_subdomain` oder `apache2_mod_changehat`.

Um nun für dieses Kapitel eine sinnvolle, aber auch übersichtliche Webapplikation einzusetzen, habe ich *phpSysInfo* ausgewählt. phpSysInfo ist eine Webapplikation, die Systeminformationen übersichtlich in einer Webseite darstellt. Sie können die Webapplikation von Sourceforge herunterladen (*http://phpsysinfo.sourceforge.net*). Wenn Sie keinen Internetzugang besitzen, können Sie auch das Paket von der CD aus dem Verzeichnis `/Software` nutzen.

Installieren Sie zunächst die Software, indem Sie das Quelltextpaket extrahieren und in die DocumentRoot des Apache 2.x-Webservers kopieren. Für unsere Zwecke genügt es für die Konfiguration, die vorbereitete Datei umzubenennen. Sie können diese Datei natürlich anpassen. Für die Demonstration in diesem Kapitel ist das jedoch nicht erforderlich.

```
# tar xzf phpsysinfo-2.5.2-rc3.tar.gz
# mv phpsysinfo /srv/www/htdocs/
# cd /srv/www/htdocs/phpsysinfo
# mv config.php.new config.php
```

Starten Sie nun den Apache 2.x-Webserver. Wenn Sie nun in einem Browser die URL *http://localhost/phpsysinfo* eingeben, sollte ein Bild wie in Abbildung 5.10 erscheinen. Falls Sie ein anderes Bild erhalten, kann dies drei Gründe haben:

1. Ihr Webserver läuft nicht. Prüfen Sie in den Protokollen, ob Sie tatsächlich den Webserver gestartet haben.
2. Ihr Browser verbindet sich nicht mit Ihrem Webserver. Prüfen Sie, ob der Browser einen Proxy benutzt, und deaktivieren Sie den Proxy für den Zielrechner Localhost.
3. AppArmor überwacht bereits den Apache Webserver. Sie sehen ein Bild wie in Abbildung 5.11. Auf einer SUSE Linux 10.1-Distribution wird das Apache-AppArmor-Profil per Default nicht geladen. Lesen Sie in diesem Fall einfach weiter.

Nachdem wir nun wissen, wie phpSysInfo bei erfolgreicher Ausführung die Informationen anzeigt, werden wir nun die AppArmor-Überwachung für den Apache 2.x aktivieren.

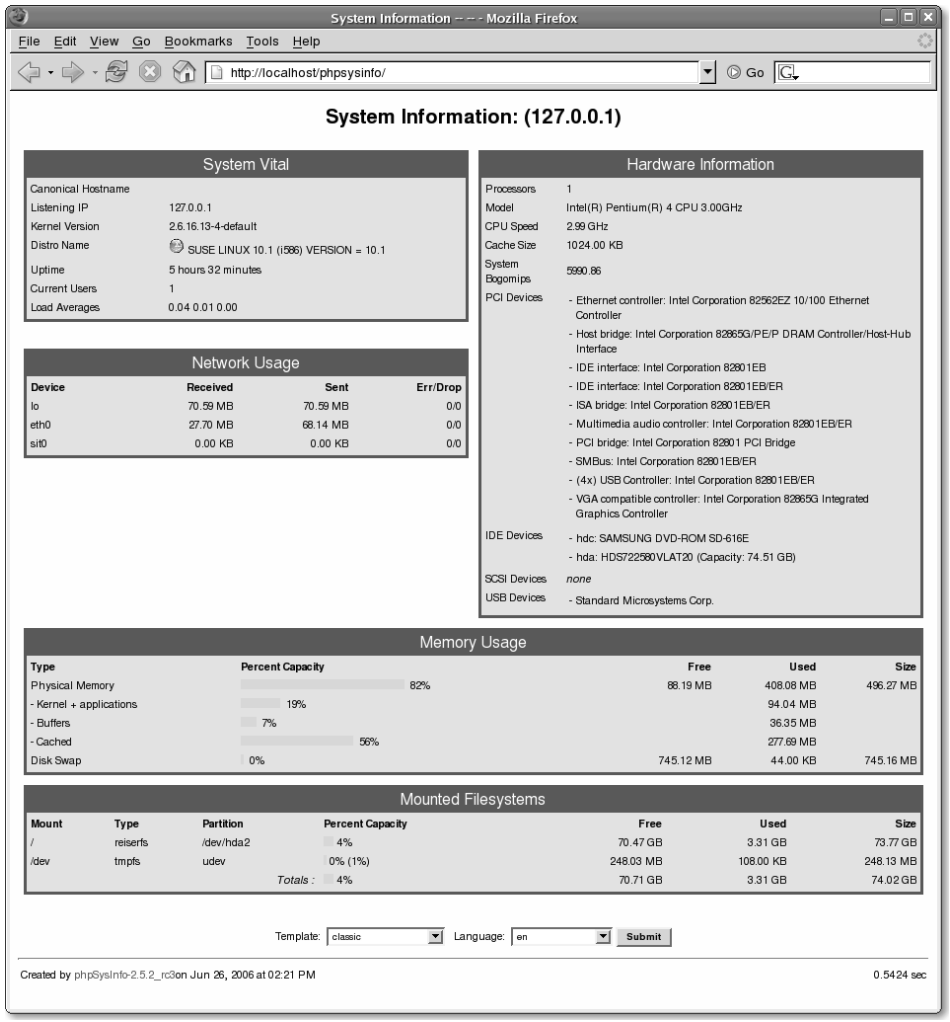

Abbildung 5.10: phpSysInfo zeigt Informationen über den Zustand des Systems an.

Kopieren Sie hierzu das Profil in das entsprechende Verzeichnis, aktivieren Sie es mit `enforce` und starten Sie den Apache erneut:

```
# cp /etc/apparmor/profiles/extras/usr.sbin.httpd2-prefork /etc/ ↵
        apparmor.d/
# enforce /usr/sbin/httpd2-prefork

Setting /usr/sbin/httpd2-prefork to enforce mode.
# rcapache2 restart
```

Wenn Sie nun erneut die phpSysInfo-Webseite aufrufen, sollten Sie sehr viele Fehlermeldungen sehen (Abb. 5.11). Um nun ein Subprofil für diese Webapplikation zu erzeugen, das phpSysInfo mit den nötigen Berechtigungen für die Ausführung versorgt, gehen wir zunächst genauso vor wie bereits in dem letzten Kapitel. Rufen Sie *Yast2* auf, und wählen Sie dort im Menü NOVELL APPARMOR den ASSISTENT ZUM HINZUFÜGEN VON PROFILEN aus. Wir wählen bewusst nicht den Assistenten zum Aktualisieren von Profilen aus. Auch der jetzt gewählte Assistent kann existierende

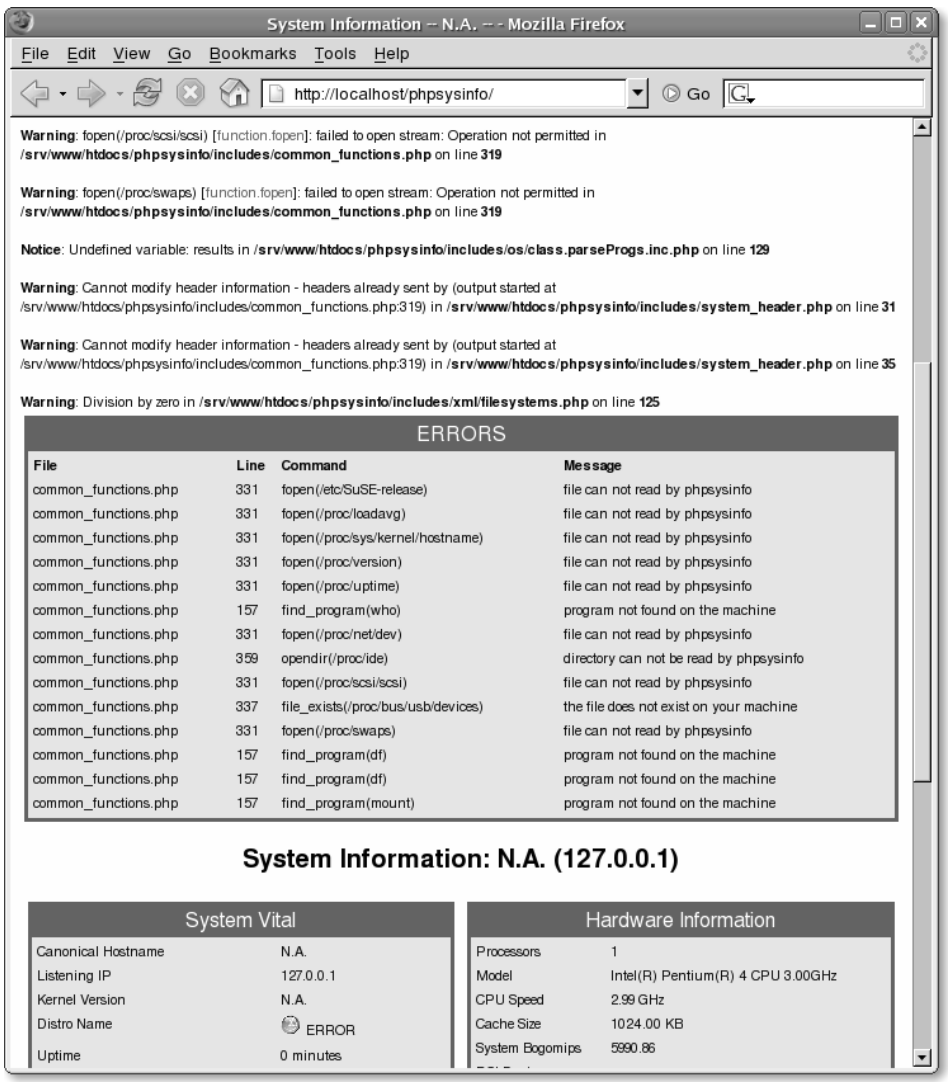

Abbildung 5.11: Wenn AppArmor aktiv ist, werden viele Zugriffe von phpSysInfo verhindert.

Profile aktualisieren. Dabei ist es aber möglich, die Aktualisierung auf ein bestimmtes Profil zu beschränken. Der andere Assistent aktualisiert alle Profile!

Bei dem Assistenten geben Sie nun als Applikation `/usr/sbin/httpd2-prefork` an. Benutzen Sie nun wieder die Applikation. Die Webapplikation sollte nun im Wesentlichen funktionieren, da jetzt wieder alle Zugriffe erlaubt, aber auch protokolliert werden. Nachdem Sie die Webapplikation genutzt haben, wählen Sie wieder im Assistenten SCAN SYSTEM LOG. Nun wird der Assistent Ihnen die erfolgten Zugriffe präsentieren, und Sie müssen entscheiden, ob der Zugriff erlaubt ist. Hierbei sollte der Assistent Ihnen sehr früh anbieten, einen zusätzlichen Hat hinzuzufügen. Bestätigen Sie das mit ADD REQUESTED HAT. Hiermit wird nun ein neues Profil für diese URI eröffnet (Abbildung 5.12). Alle weiteren Zugriffe werden nun, nach einer Bestätigung durch Sie, diesem Profil hinzugefügt. Achten Sie darauf, dass der Name des bearbeiteten Profils sich geändert hat. Er lautet nun `/usr/sbin/httpd2-prefork^/phpsysinfo/` (Abbildung 5.13).

Damit die von phpSysInfo aufgerufenen externen Programme unter der Kontrolle von AppArmor ausgeführt werden, wählen Sie für alle diese Zugriffe auf externe Kommandos INHERIT aus. Falls eine Datei gelesen werden muss, wählen Sie ALLOW. Eine Ausnahme ist der Befehl `mount`. Dieser Befehl muss `unconfined` ausgeführt werden. Wenn Sie dies nicht möchten, wählen Sie DENY. Sie müssen dann auf diese Funktion in phpSysInfo verzichten. Wenn Sie UNCONFINED auswählen, erhalten Sie eine Warnmeldung, die Sie auf die möglichen Gefahren hinweist.

Wenn Sie alle Fragen beantwortet haben und den Assistenten beenden, sollte die Webapplikation wieder so funktionieren wie vor der Aktivierung von AppArmor.

Abbildung 5.12: Yast bietet an, einen neuen Hat für die PHP-Applikation zu erzeugen.

Der einzige Unterschied ist die Tatsache, dass die Webapplikation nun von AppArmor überwacht wird und keine zusätzlichen Funktionen ausüben kann, die von einem Angreifer für einen Einbruch missbraucht werden könnten.

Hinweis

Wenn die Applikation noch nicht funktioniert und scheinbar allgemeine PHP-Fehler auftreten, dann müssen Sie möglichweise noch das Apache-Profil selbst anpassen. Rufen Sie erneut den Assistenten auf, und geben Sie wieder den kompletten Pfad `/usr/sbin/httpd2-prefork` an. Starten Sie nun den Webserver neu, und klicken Sie direkt anschließend den Button SCAN SYSTEM LOGS FOR APPARMOR EVENTS. Es fehlten wahrscheinlich in dem Profil noch die Zugriffe auf die PHP-Konfigurationsdateien. Diese fügen Sie hiermit hinzu. Jetzt sollte die Webapplikation fehlerfrei laufen.

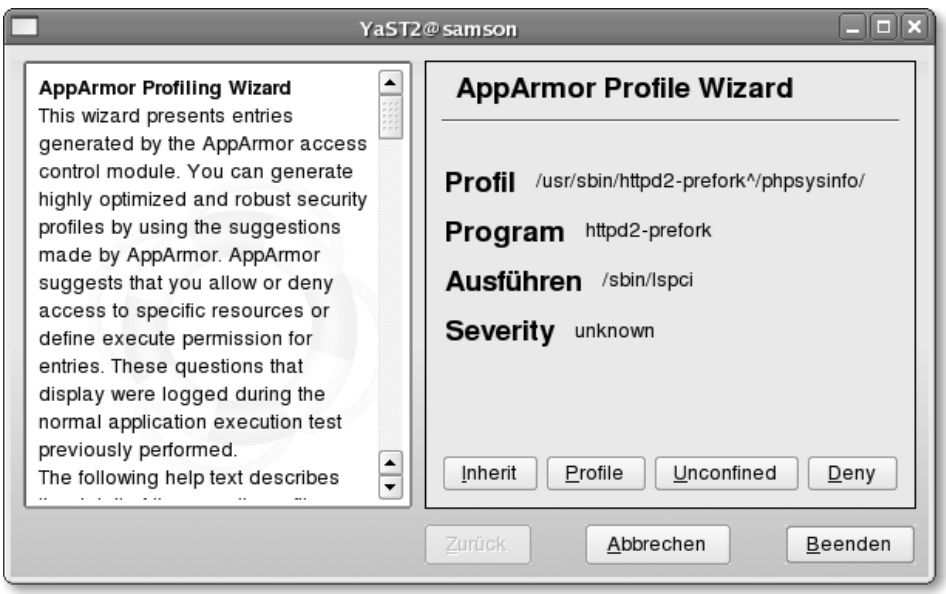

Abbildung 5.13: AppArmor bittet um die Bestätigung, in dem Hat Befehle (lspci) ausführen zu dürfen.

Das erzeugte Subprofil befindet sich ebenfalls in der Datei `/etc/apparmor.d/usr.sbin.httpd2-prefork`. Sie finden es am unteren Ende der Datei:

```
...

  ^/phpsysinfo/ {
    #include <abstractions/base>
    #include <abstractions/bash>
```

```
    #include <abstractions/consoles>

    /bin/bash ixr,
    /bin/df ixr,
    /bin/mount ux,
    /etc/SuSE-release r,
    /lib/ld-2.4.so ixr,
    /proc/** r,
    /sbin/lspci ixr,
    /srv/www/htdocs/phpsysinfo/** r,
    /sys/** r,
    /usr/bin/lsscsi ixr,
    /usr/bin/who ixr,
    /usr/sbin/lsusb ixr,
    /var/log/apache2/access_log w,
    /var/log/apache2/error_log w,
    /var/run/nscd/socket w,
  }

...
```

Die Namen von Subprofile beginnen mit dem `^`. Falls Ihr Subprofil einen anderen Aufbau hat, wundern Sie sich nicht. Der Aufbau hängt stark von dem Einsatz des Assistenten und der Wahl der File-Globbings[3] ab. Wahrscheinlich ist zu Beginn Ihr Profil wesentlich größer. Wenn Sie mehr Übung im Umgang mit dem Assistenten bekommen, werden Sie auch die Dialoge so beantworten, dass die entstehenden Profile kompakter werden. Natürlich können Sie auch anschließend das Profil editieren und mit `rcapparmor reload` die Profile neu laden.

Das Profil wird nun geladen, sobald als URI `/phpsysinfo/` im Browser ausgewählt wird.

Eine andere und möglicherweise einfachere Variante der Administration der Subprofile in Apache wird in Abschnitt 8.2.1 beschrieben.

[3] Als *Globbing* bezeichnet man die Verwendung von *Wildcards* bei der Angabe von Dateinamen. AppArmor erlaubt die Verwendung von `?`, `*` und `**`.

6 AppArmor-Funktion

Dieses Kapitel führt Sie tiefer in die Welt von AppArmor ein, nachdem Sie im letzten Kapitel erste Versuche mit AppArmor unternommen haben. Sie lernen die verschiedenen Programme kennen, erfahren, welche Programme Sie mit AppArmor überwachen sollten, und erhalten Hintergrundinformationen über die Syntax der AppArmor-Profile. Das Kapitel schließt mit der Erklärung der ChangeHat-Funktion von AppArmor.

6.1 Was sollte immunisiert werden?

Novell AppArmor bezeichnet den implementierten Schutz als *Immunisierung des Systems*. Diese Bezeichnung erklärt sich aus der Geschichte von AppArmor und seiner Herkunft von der Firma Immunix. Diese Immunisierung schützt das System vor den Schwächen der eingesetzten Software. Sobald AppArmor aktiviert und das System neu gestartet wurde, ist dieser Immunschutz aktiv.

Die verschiedenen SUSE Linux-Distributionen und SUSE Linux Enterprise Server verfügen über eine unterschiedliche Anzahl von mitgelieferten und aktivierten Profilen. Jedes Profil implementiert den Schutz für genau ein Programm. Novell stellt für die meisten Standarddienste auf einem Linux-System Profile zur Verfügung. Ihre Qualität unterscheidet sich leider stark in den unterschiedlichen Versionen. Die Profile definieren, welchen Zugriff Novell AppArmor den verschiedenen überwachten Programmen erlauben soll. Dabei gelten die von AppArmor implementierten Beschränkungen zusätzlich zu den allgemeinen Linux-Rechten und ergänzen diese. So stellt AppArmor sicher, dass jeder Prozess nur das tut, was er tun soll.

Wenn Sie sich nun fragen, welche Programme von AppArmor überwacht werden sollen, so ist es besonders wichtig, die Programme zu kontrollieren, die einem Benutzer erweiterten Zugriff auf das System erlauben. Dies sind:

- Netzwerkdienste
- Netzwerkclients
- Cron-Jobs

Im Folgenden wollen wir diese drei Gruppen ein wenig genauer betrachten und klären, warum diese tatsächlich überwacht werden sollten.

6.1.1 Netzwerkdienste

Netzwerkdienste erlauben es einem beliebigen Client, Kontakt mit Ihrem System aufzunehmen. Selbst wenn diese Netzwerkdienste eine Authentifizierung des Clients durchführen, verfügen die Dienste selbst über wesentlich mehr Rechte, als Sie einem anonymen oder auch authentifizierten Client zugestehen möchten. Ein Webserver darf zum Beispiel sehr viele Dateien Ihres Systems lesen. Ein IMAP-Server kann die E-Mail-Postfächer jedes Benutzers lesen und schreiben. Er soll meist auch andere Dateien lesen und schreiben. Hierfür benötigt er meist *root*-Privilegien. Das Gleiche trifft auf den *Secure-Shell*-Server zu[1]. Es sollte Ihnen nun sehr deutlich klar geworden sein, dass es sinnvoll ist, diese Dienste zu überwachen und ihnen nur die notwendigen Zugriffe zu erlauben. Sie können die in Frage kommenden Dienste mit dem Befehl `unconfined` ermitteln.

6.1.2 Netzwerkclients

Während die Notwendigkeit einer Überwachung der Netzwerkdienste leicht einleuchtet, ist das bei den Netzwerkclients nicht offensichtlich. Aber auch diese Programme sollten überwacht werden, da sie über sämtliche Rechte des aufrufenden Benutzers verfügen. Hat der Firefox-Webbrowser zum Beispiel einen Fehler in der Behandlung und Darstellung bestimmter Bildformate, wie PNG, so kann ein Angreifer auf einer Webseite manipulierte PNG-Bilder hinterlegen, die dann in dem Firefox Schadcode zur Ausführung bringen. In der Vergangenheit gab es zahllose Beispiele, bei denen auf diese Weise Trojaner auf betroffenen Clientsystemen installiert wurden. Überwacht AppArmor den Firefox-Browser und gestattet ihm nur das Lesen und Schreiben bestimmter ausgewählter Dateien und Verzeichnisse, ist die Wahrscheinlichkeit eines erfolgreichen Angriffs stark herabgesetzt.

Dies betrifft natürlich nicht nur Webbrowser, sondern jedes Programm, das Daten aus nicht vertrauenswürdigen Quellen anzeigen kann:

- Browser
- E-Mail-Clients
- Chat-Clients
- Bildbetrachter
- Dokumentenbetrachter (speziell PDF)
- Video- und Audioplayer
- ...

Leider gibt es keine Möglichkeit, einfach alle in Frage kommenden Programme zu ermitteln. Sie sollten überprüfen, welche Programme Sie einsetzen, und für die entsprechenden Programme Profile erzeugen.

[1] Der Secure-Shell-Server benötigt die *root*-Rechte, da er nach der Anmeldung des Benutzers einen entsprechenden Prozess mit den Rechten des angemeldeten Benutzers starten soll. Lediglich der Benutzer *root* ist in der Lage, seine Identität beliebig zu ändern. Diese Funktion benötigt der Secure-Shell-Server. Aus diesem Grunde benötigt auch der IMAP-Server *root*-Privilegien, wenn die E-Mails in dem Dateisystem abgelegt werden.

6.1.3 Cron-Jobs

Schließlich gehören auch *Cron*-Jobs zur Gruppe der Programme, die über erweiterte Privilegien verfügen. Cron-Jobs werden häufig mit *root*-Rechten ausgestattet, damit sie ungestört ihre Arbeit verrichten können. Dies hat in der Vergangenheit dazu geführt, dass Cron-Jobs häufig Angriffsvektoren für lokale Angreifer waren. Durch geschickte Modifikation der von den Cron-Jobs verarbeiteten Dateien konnte der Angreifer den Cron-Job übernehmen oder zur Ausführung anderer Funktionen zwingen.

Die Ermittlung der in Frage kommenden Cron-Jobs ist recht einfach. Diese befinden sich in den folgenden Verzeichnissen:

- `/etc/cron.d`
- `/etc/cron.daily`
- `/etc/cron.hourly`
- `/etc/cron.monthly`
- `/etc/cron.weekly`

Zusätzlich können Cron-Jobs in der Datei `/etc/crontab` definiert worden sein. Persönliche Cron-Jobs von *root* zeigt dieser mit dem Befehl `crontab -l` an.

6.2 Wie schützt AppArmor?

AppArmor betrachtet einzelne Prozesse und überwacht deren Zugriffe. Dabei entscheidet es, ob der Zugriff erlaubt oder abgelehnt wird. Die Überwachung des Prozesses wird von AppArmor zum Startzeitpunkt des Prozesses aktiviert. AppArmor muss also vor dem Prozess aktiviert werden, und das entsprechende Profil muss geladen worden sein. AppArmor unterstützt die Steuerung über ein Security-Dateisystem unterhalb von `/sys/kernel/security`. Bei älteren Versionen befindet sich diese Schnittstelle in `/subdomain`. Um den Schutz zu gewährleisten, liest AppArmor die Richtlinien aus den Profildateien ein. Unabhängig von diesen Dateien stellt AppArmor sicher, dass ein überwachter Prozess nie die folgenden Systemaufrufe durchführen kann:

- `create_module(2)`
- `delete_module(2)`
- `init_module(2)`
- `ioperm(2)`
- `iopl(2)`
- `mount(2)`
- `mount(2)`
- `ptrace(2)`
- `reboot(2)`
- `setdomainname(2)`
- `sethostname(2)`

- `swapoff(2)`
- `swapon(2)`
- `sysctl(2)`
- `mknod(2)` für Zeichen- oder Blockgeräte

6.3 AppArmor-Befehle

AppArmor verwendet viele verschiedene Befehle, um seine Funktion sicherzustellen. Bei den meisten Befehlen handelt es sich um Perl-Scripts, die auf die *Perl*-Bibliothek `Immunix::Subdomain` zugreifen. Sie können diese Perl-Bibliothek auch für eigene Scripts nutzen.

Im Einzelnen werde ich Ihnen nun die folgenden Befehle vorstellen:

- `apparmor_status`: Dieser Befehl zeigt Ihnen den aktuellen Status von AppArmor an (früher: `subdomain_status`).
- `audit`: Dieser Befehl schaltet ein Profil in den Audit-Modus. Alle erlaubten Zugriffe werden protokolliert.
- `autodep`: Hiermit können Sie ein Basis-Profil für eine Applikation erzeugen.
- `complain`: Hiermit schalten Sie ein Profil in den Complain-Modus. In diesem Lernmodus werden alle in dem Profil verbotenen Zugriffe erlaubt und protokolliert.
- `enforce`: Hiermit schalten Sie ein Profil wieder in den Enforce-Modus. Dies ist der Default. Alle nicht im Profil erlaubten Zugriffe werden abgelehnt.
- `genprof`: Hiermit erzeugen Sie für eine Applikation ein neues Profil oder aktualisieren es.
- `logprof`: Dieser Befehl ist vor allem sinnvoll, wenn viele Profile angepasst werden müssen. Versetzen Sie alle Profile in den Complain-Modus, nutzen Sie das System, und analysieren Sie es anschließend mit `logprof`.
- `apparmor_parser`: Dies ist das Herz der AppArmor-Profile. Hiermit können Sie Profile in den Kernel laden, ersetzen und löschen.
- `unconfined`: Dieser Befehl prüft, welche Applikationen aktuell von AppArmor überwacht werden.

Fast alle Befehle weisen eine Manpage auf. Daher werde ich teilweise auch auf diese Dokumentation verweisen.

6.3.1 apparmor_status

Dieser Befehl zeigt verschiedene Informationen über den aktuellen Status von AppArmor an. Seine Funktion erklärt sich am besten an einer Beispielausgabe:

```
# apparmor_status
apparmor module is loaded.
65 profiles are loaded.
```

```
56 profiles are in enforce mode.
9 profiles are in complain mode.
Out of 71 processes running:
10 processes have profiles defined.
10 processes have profiles in enforce mode.
0 processes have profiles in complain mode.
```

Der Befehl `apparmor_status` muss als *root* ausgeführt werden, damit der Befehl über die notwendigen Leserechte verfügt, um die Informationen anzuzeigen. Speziell für die Verwendung in Scripts unterstützt der Befehl auch die Option `--enabled` mit der ein Script prüfen kann, ob AppArmor aktiv ist. Ist dies nicht der Fall, wird ein Fehlercode zurückgegeben, der von dem Script ausgewertet werden kann. Die Optionen `--profiled`, `--enforced` und `--complaining` geben jeweils nur die Zahlen der oben angezeigten Ausgabe auf dem Bildschirm aus.

6.3.2 audit

Mit dem Befehl `audit` können Sie ein Programm in den Audit-Modus schalten. Leider ist für diesen Befehl keine Manpage vorhanden. Er verhält sich jedoch genauso wie auch der Befehl `enforce` und `complain`. Im Audit-Modus wird die Applikation von AppArmor genauso überwacht wie in dem Enforce-Modus. Die Applikation darf nur die von dem Profil erlaubten Zugriffe durchführen. Unerlaubte Zugriffe werden wie im Enforce-Modus protokolliert. Zusätzlich werden aber auch alle erlaubten Zugriffe protokolliert. Eine typische Protokollmeldung sieht so aus:

```
type=APPARMOR msg=audit(1151477803.302:3398): AUDITING r access to ↵
         /etc/mtab (df(13541) profile /usr/sbin/httpd2-prefork ↵
         active /phpsysinfo/)
```

Diese Protokollmeldungen benötigen natürlich Rechenzeit und Speicherplatz. Daher sollte auf einem Produktivsystem diese Funktion möglichst nicht genutzt werden. Dennoch kann auf diese Weise sehr einfach kontrolliert werden, worauf eine Applikation wann zugreifen möchte.

Die Syntax des Befehls ist sehr einfach. Sie geben lediglich den Namen des Programms an:

```
audit /usr/sbin/httpd2-prefork
```

Alternativ können Sie auch den Namen des Profils angeben. Um alle überwachten Profile in den Audit-Modus zu versetzen, genügt ein:

```
audit /etc/apparmor.d/*
```

Für den unwahrscheinlichen Fall, dass sich das Profil nicht an der üblichen Stelle befindet, können Sie mit der Option `-d <directory>` das Profilverzeichnis angeben.

6.3.3 autodep

Dieser Befehl erstellt ein minimales Profil für ein noch nicht von AppArmor überwachtes Programm. Hierzu rufen Sie den Befehl unter Angabe von einer oder mehreren ausführbaren Dateien auf. Der Befehl analysiert diese ausführbaren Dateien, indem er unter anderem jeden Befehl mit dem Kommando `ldd` analysiert. Der Befehl `ldd` ermittelt die von einem ausführbaren Programm benötigten Bibliotheken.

Der Befehl `autodep` kann sowohl bei binären Dateien als auch bei Scripts eingesetzt werden. Er erzeugt immer mindestens ein Profil, mit einer Include-Direktive für die Base-Abstractions (siehe Abschnitt 6.6).

Wird der Befehl `autodep` zum Beispiel für das Programm Nmap aufgerufen, erzeugt er die folgende Profildatei:

```
# autodep /usr/bin/nmap
Writing updated profile for /usr/bin/nmap.
# cat /etc/apparmor.d/usr.bin.nmap
# vim:syntax=apparmor
# Last Modified: Wed Jun 28 09:37:12 2006
#include <tunables/global>

/usr/bin/nmap flags=(complain) {
  #include <abstractions/base>

  /usr/bin/nmap r,
}
```

Basierend auf dieser Datei kann nun mit `genprof` (siehe Abschnitt 6.3.6) oder `logprof` (siehe Abschnitt 6.3.7) das Profil weiterentwickelt werden. Hierzu wurde das Profil automatisch bereits in den Complain-Modus geschaltet und von AppArmor geladen. Bei dem Einsatz von `genprof` ist der Einsatz des `autodep`-Kommandos aber nicht zwingend erforderlich.

6.3.4 complain

Der Befehl `complain` arbeitet ähnlich wie der `audit`- und `enforce`-Befehl. Er schaltet ein Profil in den Complain-Modus. Dieser Modus kann als Lernmodus bezeichnet werden, da nun alle Zugriffe (auch die in dem Profil verbotenen Zugriffe) erlaubt werden. Zugriffe, die von dem Profil normalerweise abgelehnt werden würden und in diesem Lernmodus erlaubt werden, werden zusätzlich protokolliert:

```
type=APPARMOR msg=audit(1151482060.120:3514): PERMITTING r access ⬅
         to /etc/resolv.conf (nmap(14613) profile /usr/bin/nmap ⬅
         active /usr/bin/nmap)
```

Diese Meldungen werden dann von `genprof` oder `logprof` verwendet, um das Profil anzupassen. Hierfür fügt AppArmor auch `LOGPROF-HINT`-Meldungen ein, die von `logprof` ausgewertet werden.

```
type=APPARMOR msg=audit(1151325947.843:1416): LOGPROF-HINT ➥
         fork pid=28870 child=29066
type=APPARMOR msg=audit(1151325947.843:1418): LOGPROF-HINT ➥
         changing_profile pid=29066
type=APPARMOR msg=audit(1151325948.119:1448): LOGPROF-HINT ➥
         unknown_hat /phpsysinfo/images/Suse.png pid=28871 profile ➥
         =/usr/sbin/httpd2-prefork active=/usr/sbin/httpd2-prefork
```

6.3.5 enforce

Dieser Befehl schaltet ein Profil wieder in den Enforce-Modus. Das ist nur erforderlich, wenn Sie vorher ein Profil manuell mit `complain` oder `autodep` in den Complain-Modus geschaltet haben. Die Default-Einstellung für alle Profile ist der Enforce-Modus. Sie können den Befehl auch nutzen, um den Audit-Modus für ein Profil wieder abzuschalten. Genauso wie bei `audit` und `complain` können Sie entweder das zu überwachende Programm mit seinem gesamten Pfad oder auch das Profil angeben.

Um zum Beispiel alle Profile in den Enforce-Modus zu versetzen, können Sie den folgenden Befehl verwenden:

```
# enforce /etc/apparmor.d/*
Setting /etc/apparmor.d/bin.netstat to enforce mode.
Setting /etc/apparmor.d/bin.ping to enforce mode.
Setting /etc/apparmor.d/lib.ld-2.2.so to enforce mode.
Setting /etc/apparmor.d/sbin.klogd to enforce mode.
...
```

Sie können auch ein neues Profil, das noch nicht von AppArmor geladen wurde, mit diesem Befehl laden. Dazu kopieren Sie das Profil in das Verzeichnis `/etc/apparmor.d` und aktivieren es dann mit dem `enforce`-Kommando. Das funktioniert natürlich auch mit dem `complain`- und `audit`-Kommando.

6.3.6 genprof

Mit diesem Befehl können Sie recht einfach auf der Kommandozeile ein Profil für eine neue Applikation erstellen. *Yast2* nutzt für seine Assistenten ebenfalls diesen Befehl, wenn ein neues Profil erzeugt werden soll.

Der Befehl erwartet von Ihnen die Angabe des Programms, für das ein Profil erzeugt werden soll. Sie müssen dieses Programm wie gewohnt mit seinem kompletten Pfad eingeben. `genprof` prüft dann zunächst, ob dieses Programm von einem Profil geschützt werden darf. Bestimmte Programme sind von dem Schutz durch AppArmor ausgenommen, da dadurch das gesamte System instabil werden kann. Diese Programme werden in der Konfigurationsdatei `/etc/apparmor/logprof.conf` definiert (siehe Abschnitt 6.4.1).

Anschließend prüft dieser Befehl, welches Protokollsystem aktuell für die Verarbeitung der AppArmor-Meldungen zuständig ist. Falls die AppArmor-Meldungen von dem Syslog verarbeitet und in der Datei `/var/log/messages` gespeichert werden, erzeugt `genprof` eine Markierung in dieser Datei:

```
kernel GenProf: <MD5-Pruefsumme des Datums>
```

Erfolgt die Protokollierung über den *Auditd*-Daemon, so liest `genprof` den letzten Eintrag in dem Protokoll, um später die neu hinzugefügten Meldungen erkennen zu können.

Als Nächstes prüft dieser Befehl, ob für das zu analysierende Programm bereits ein Profil existiert. Ist dies nicht der Fall, erzeugt der Befehl zunächst mithilfe des Kommandos `autodep` ein Basisprofil. Das Profil wird dann mit `complain` in den Lernmodus geschaltet, und Sie werden aufgefordert, nun das Programm, für das `genprof` das Profil erzeugen soll, zu benutzen (siehe Abbildung 6.1). Sobald Sie nun S für Scan auswählen, beginnt `genprof` mit der Analyse der Meldungen und fragt Sie bei jedem protokollierten Zugriff, ob dieser in Zukunft erlaubt sein soll. Für diese Funktion nutzt `genprof` den Befehl `logprof` und weist diesen an, die Protokolle ab

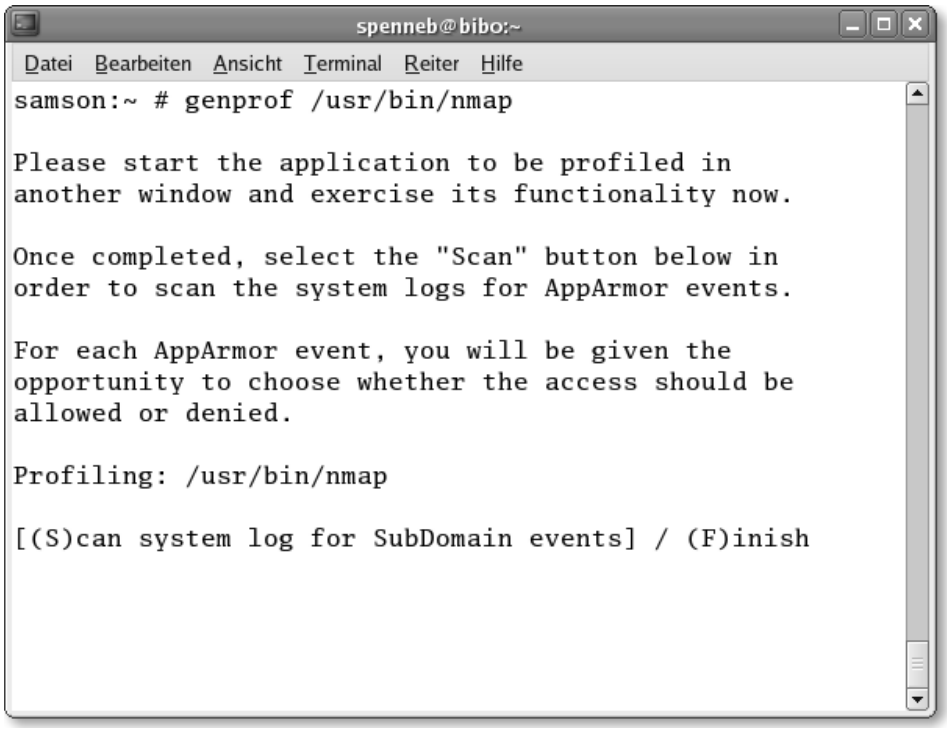

Abbildung 6.1: genprof fordert Sie auf, das Programm zu benutzen. Anschließend scannt genprof die erzeugten AppArmor-Protokollmeldungen.

der geschriebenen Markierung zu analysieren. Neu hinzugekommen gegenüber dem Yast-Dialog ist die Auswahl *New*. Diese entspricht dem EDIT-Button im Yast-Dialog (siehe auch Abschnitt 6.3.7).

Sobald alle Meldungen abgearbeitet sind und alle Fragen von Ihnen beantwortet wurden, erzeugt `genprof` das neue Profil, lädt es und versetzt es wieder in den Lernmodus. Sie können nun das Programm weiterverwenden und erneut den Scan durchführen, um neue Zugriffe in das Profil aufzunehmen. Wenn Sie fertig sind, beenden Sie `genprof` mit (F)inish. `genprof` wird nun das Profil in den Enforce-Modus schalten. Alle Zugriffe, die Sie nicht erlaubt haben, werden nun von AppArmor unterbunden.

Natürlich können Sie zu jedem späteren Zeitpunkt die Anpassung des Profils wieder aufnehmen. Dazu starten Sie erneut `genprof` für das betroffene Programm. Zugriffe, die bisher nicht erlaubt waren, aber von dem Programm benötigt werden, können dann wieder mit `genprof` dem Profil hinzugefügt werden.

6.3.7 logprof

Mit diesem Befehl können Sie die AppArmor-Protokollmeldungen analysieren und entsprechende Änderungen an den Profilen vornehmen. Dieser Befehl ist besonders interessant, wenn viele Profile betroffen sind. Sie versetzen mit dem Befehl `complain` die entsprechenden Profile in den Lernmodus und rufen nach der Benutzung `logprof` auf. Die Syntax für den Aufruf ist sehr einfach. Sie können drei optionale Parameter angeben:

- `-d --dir <Profilverzeichnis>`: Hiermit können Sie ein alternatives Profilverzeichnis angeben. Das ist aber eigentlich nie nötig.
- `-f --file <Protokolldatei>`: Hiermit können Sie die Protokolldatei auswählen, in der die AppArmor-Meldungen gespeichert werden. Dies ist notwendig, wenn `logprof` nicht selbst die richtige Datei findet. Das kann zum Beispiel der Fall sein, wenn Sie den Syslog-Daemon umkonfiguriert haben, sodass die Meldungen nicht in der Datei `/var/log/messages` landen.
- `-m --logmark "<Markierung>"`: Dieser Parameter ist der wichtigste von allen. Hiermit weisen Sie `logprof` an, die Analyse der Protokolldatei an einer bestimmten Stelle zu beginnen und ältere Einträge zu ignorieren. Wenn `logprof` von `genprof` aufgerufen wird, wird diese Option genutzt, um den Analysestart festzulegen. Wie Sie selbst diese *Protokollmarkierung* erzeugen können, wird bei der Protokollanalyse beschrieben (siehe Abschnitt 8.1).

`logprof` startet dann als interaktives Werkzeug und wird auch von `genprof` und von den Yast-Assistenten genutzt. Seine Darstellung auf der Kommandozeile ist in Abbildung 6.2 zu erkennen. `logprof` analysiert die Protokollmeldungen und schlägt Ihnen Modifikationen des Profils vor, die die gemeldeten Zugriffe in Zukunft erlauben. Häufig schlägt `logprof` Ihnen mehrere Möglichkeiten vor, wie Sie den Zugriff erlauben können. Sie können dann aus der Liste durch Auswahl der vorangestellten Nummer den besten Vorschlag auswählen. Wenn Ihnen kein Vorschlag gefällt, können Sie durch die Eingabe von (N) für New einen eigenen Vorschlag erstellen.

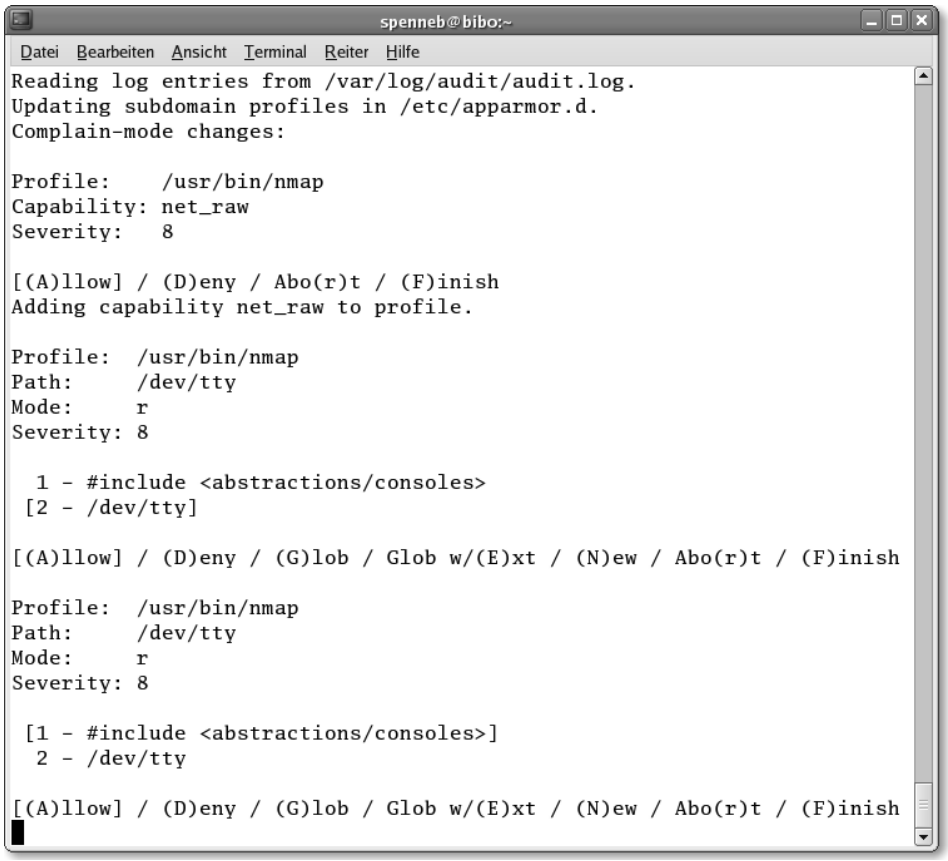

Abbildung 6.2: Ähnlich wie bei den Yast-Assistenten haben Sie die Möglichkeit auszuwählen, ob der Zugriff erlaubt werden soll oder nicht.

Die Analyse der Protokollmeldungen kann sowohl mit Profilen im Complain-Modus als auch im Enforce-Modus erfolgen. Der einzige Unterschied im Verhalten von `logprof` ist die Auswahl der Default-Aktion bei der Auswahl der verschiedenen Möglichkeiten. War das Profil im Complain-Modus, schlägt Ihnen `logprof` per Default (A)llow vor. Ansonsten ist es ein (D)eny. Natürlich funktioniert `logprof` besser im Complain-Modus. Im Enforce-Modus verhindert AppArmor den Zugriff auf die angeforderte Ressource. Häufig kann dann das Programm nicht weiterarbeiten. `logprof` kann daher immer nur wenige Meldungen analysieren.

Eine besondere Funktion von AppArmor und `logprof` ist das Globbing. AppArmor unterstützt ein Globbing der Dateinamen mit dem Wildcard `*`. Die hierfür erforderlichen Ausdrücke können sehr leicht mit `logprof` erzeugt werden. Wenn Sie einmal (G)lob auswählen, wird der letzte Anteil des Pfades entfernt und durch einen `*` ersetzt. Wählen Sie erneut (G)lob aus, so wird wieder der letzte Teil des Pfades entfernt, und es werden `**` angehängt. Während `*` ein Wildcard für jeden beliebigen Dateinamen ist,

ist ** ein Wildcard für den gesamten Verzeichnisbaum unterhalb des angegebenen Pfades. Die letzte Globbing-Variante wird auch automatisch vorausgewählt (siehe Abbildung 6.3).

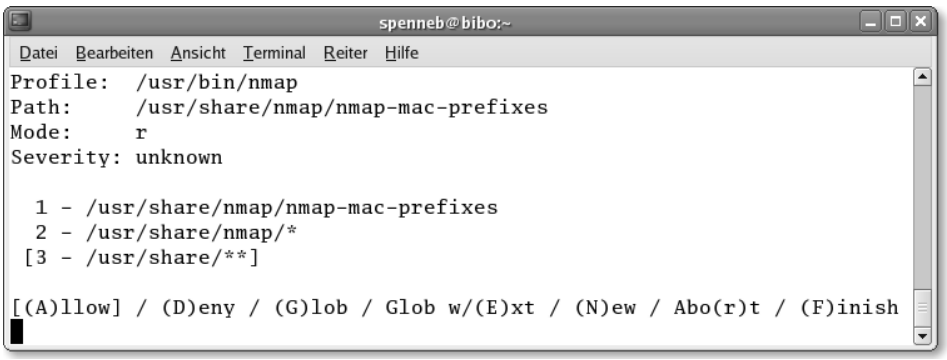

Abbildung 6.3: Um die Regeln übersichtlicher zu gestalten, erlauben AppArmor und logprof die Verwendung von Wildcards in den Regeln.

Sollten Sie in einem früheren Dialog bereits weitere Dateien in dem Verzeichnis, z.B. `/usr/share/nmap/nmap-protocols` aufgenommen haben, so werden diese automatisch von `logprof` entfernt, sobald eine entsprechende Globbing-Regel in das Profil aufgenommen wird.

Durch die Auswahl von Abo(r)t werden alle akzeptierten Änderungen vergessen und rückgängig gemacht. Mit (F)inish akzeptieren Sie die Änderungen.

Neben den einfachen Dateizugriffen analysiert `logprof` auch die Ausführung von neuen Prozessen, ChangeHat-Ereignissen und den Zugriff auf Capabilitys.

Ruft das analysierte Programm ein weiteres Programm auf, so schlägt Ihnen `logprof` vier verschiedene Möglichkeiten vor:

- `px`: Hier muss das aufgerufene Programm über ein eigenes Profil verfügen und wird von diesem überwacht. Existiert für das Programm bei der späteren Ausführung kein Profil, wird der Aufruf verweigert!
- `ix`: Das aufgerufene Programm erbt das Profil des aufrufenden Programms. Es darf lediglich auf die gleichen Dateien zugreifen, auf die auch der Mutterprozess zugreifen darf. Dies ist in vielen Fällen die beste und auch eine sehr sichere Lösung. Viele Programme, wie zum Beispiel der Pager `less`, werden von einer Vielzahl von Softwarepaketen genutzt. Wollte man ein eigenes Profil für den Befehl `less` erzeugen, müsste dieses alle aufrufenden Softwarepakete berücksichtigen. Da ist es einfacher und besser, das Elternprofil zu vererben (Inherit).
- `ux`: Diese Option ist sehr gefährlich. Das aufgerufene Programm wird von AppArmor nicht überwacht. Ein Angreifer kann versuchen, über den Elternprozess gefährliche Daten an dieses nicht überwachte (unconstrained) Programm zu übergeben. Jedoch gibt es bestimmte Befehle, die in der Datei `/etc/apparmor/log-`

`prof.conf` definiert werden (siehe Abschnitt 6.4.1), bei denen nur diese Option angeboten wird.

- `Deny`: Selbsterklärend.

Falls `logprof` ein *ChangeHat*Ereignis erkennt (aktuell nur für den Apache 2.0), werden Sie aufgefordert zu entscheiden, ob für diesen Zugriff der DefaultHat genutzt werden soll, ob Sie einen neuen *Hat* (Subprofil) erzeugen möchten, in dem Sie die weiteren Zugriffe dann erlauben, oder ob Sie den Zugriff verweigern möchten. Häufig ist es sinnvoll, wenn serverbasiertes Scripting eingesetzt wird, einen eigenen Hat zu definieren. Sie können dann den Zugriff des Scripts einschränken oder erweitern, ohne die restlichen Funktionen auf dem Webserver zu beeinflussen.

Das letzte Ereignis, das von `logprof` besonders behandelt wird, ist der Zugriff auf eine POSIX-*Capability* (siehe Abschnitt 2.3). Hier haben Sie dann die Möglichkeit zu entscheiden, ob der Prozess diese Capability nutzen darf oder nicht. Damit beschneiden Sie die Möglichkeiten der Prozesse, die mit *root*-Privilegien gestartet wurden. Obwohl der Prozess mit *root*-Privilegien gestartet wurde, darf er nur die speziell erlaubten Capabilitys nutzen. So darf ein `ping`-Kommando zunächst ein SetUID zu *root* durchführen, wie es die Linux-Dateirechte ihm erlauben, und dann sowohl RAW- als auch PACKET-Sockets öffnen und benutzen. Eine Administration der Netzwerkdienste (`NET_ADMIN`) oder das Lesen beliebiger Dateien (`DAC_OVERRIDE`) werden ihm nicht erlaubt, obwohl das `ping`-Kommando dies ohne AppArmor-Überwachung durchaus dürfte.

```
#include <tunables/global>

/bin/ping {
  #include <abstractions/base>
  #include <abstractions/consoles>
  #include <abstractions/nameservice>

  capability net_raw,
  capability setuid,

  /bin/ping ixr,

...
}
```

6.3.8 apparmor_parser

Mit dem Befehl `apparmor_parser` werden neue Profile in den Kernel importiert, dort vorhandene Profile ersetzt oder gelöscht. Er stellt das Herz der AppArmor-Userspace-Befehle dar. Bei älteren Versionen heißt dieser Befehl `subdomain_parser`.

Über diesen Befehl laden die AppArmor-Befehle neue Profile und ersetzen vorhandene Profile. Sie können diesen Befehl auch selbst benutzen, um ein geändertes Profil neu zu laden oder ein Profil zu entfernen.

Der Befehl bietet hierfür die folgenden Optionen an:

- `-a, --add`: Hiermit laden Sie ein Profil in den Kernel. Dies ist auch der Default, wenn keine Option angegeben wurde. Existiert bereits ein Profil mit identischem Namen im Kernel, wird eine Fehlermeldung ausgegeben. Bei Erfolg wird ebenfalls eine Meldung ausgegeben:

  ```
  # apparmor_parser -a usr.bin.nmap
  Addition succeeded for "/usr/bin/nmap".
  ```

  Achtung: Bereits gestartete Anwendungen werden von einem nachträglich geladenen Profil nicht überwacht.

- `-r, --replace`: Hiermit können Sie ein im Kernel geladenes Profil durch ein aktuelleres Profil ersetzen. Ein Neustart der überwachten Applikation ist nicht erforderlich!

  Befinden Sie sich in dem Verzeichnis `/etc/apparmor.d` so ist die Angabe des Pfades nicht erforderlich.

  ```
  # apparmor_parser -r /etc/apparmor.d/usr.sbin.httpd2-prefork
  Replacement succeeded for "/usr/sbin/httpd2-prefork".
  ```

- `-R, --remove`: Hiermit löschen Sie ein geladenes Profil. Eine aktuell laufende Applikation, die von diesem Profil überwacht wurde, wird anschließend nicht mehr überwacht. Auch ein erneutes Laden des Profils stellt die Applikation nicht wieder unter die Überwachung. Dies erfolgt dann erst nach einem Neustart.

- `-p, --preprocess`: Diese Option ist für Sie wahrscheinlich uninteressant, da der Parser hier lediglich alle Include-Direktiven analysiert und das originale Profil einschließlich aller Include-Dateien auf der Standardausgabe ausgibt.

- `-I, --Include`: Hiermit können Sie einen zusätzlichen Suchpfad für die Auswertung der absoluten Include-Direktiven angeben. Dies ist meist unnötig.

- `-b, --base`: Hiermit können Sie das Verzeichnis angeben, in dem relative Include-Direktiven gesucht werden. Auch diese Angabe ist meist unnötig.

- `-C, --Complain`: Dieser Schalter lädt das Profil im Complain-Modus.

- `-h, --help`: Hiermit erhalten Sie eine kleine Hilfe.

- `-v, --version`: Dieser Schalter gibt die Versionsnummer des Parsers aus.

- `-d, --debug`: Diese Option ist wieder sehr interessant. Wenn Sie diese Option einmal angeben, prüft der Parser die syntaktische Richtigkeit des Profils:

  ```
  # apparmor_parser -d /etc/apparmor.d/usr.bin.nmap
  Error: #include <abtractions/base> not found. line 6 in /etc/ ↵
           apparmor.d/usr.bin.nmap
  ```

  Dabei prüft der Parser, ob sämtliche Schlüsselwörter bekannt sind und sämtliche Include-Dateien existieren. Ob die in den Regeln definierten Dateien vorhanden sind, prüft er nicht.

  Wenn Sie die Debug-Option doppelt angeben, wertet der Parser alle Regeln und Include-Dateien aus und gibt Ihnen das Ergebnis auf dem Bildschirm aus. So kön-

nen Sie nachvollziehen, auf welche Dateien die Applikation tatsächlich zugreifen darf:

```
# apparmor_parser -dd /etc/apparmor.d/usr.bin.nmap
----- Debugging built structures -----
Name:           /usr/bin/nmap
Subname:        NULL
Type:           Default Allow
Capabilitys:   net_bind_service net_raw
--- Entries ---
Mode:   rw      Name:   (/dev/console)
Mode:   w       Name:   (/dev/log)
Mode:   rw      Name:   (/dev/null)
Mode:   r       Name:   (/dev/pts)
Mode:   rw      Name:   (/dev/pts/[0-9]*)
...
```

Gerade diese letzte Funktion kann durchaus bei der Fehlersuche behilflich sein. Sie können in der Ausgabe mit `grep` nach bestimmten Dateien suchen und die Rechte bei einem Zugriff auslesen.

6.3.9 unconfined

Dieser Befehl ermittelt mit `netstat -nlp` die aktuell laufenden Netzwerkdienste und überprüft, ob die entsprechenden Prozesse von AppArmor überwacht werden. Dabei prüft `unconfined` nicht, ob das entsprechende Profil geladen wurde, sondern ob tatsächlich der Prozess überwacht wird. Wurde das Profil erst nach dem Start des Prozesses geladen oder AppArmor erst nach dem Start des Prozesses aktiviert, wird der Prozess nicht überwacht. Unconfined erkennt das und meldet, dass der entsprechende Prozess nicht überwacht wird.

```
# unconfined
11314 /sbin/dhcpcd not confined
11487 /usr/lib/zmd/zmd-bin not confined
11527 /sbin/portmap not confined
11527 /sbin/portmap not confined
11840 /usr/sbin/sshd not confined
13163 /usr/sbin/cupsd not confined
13163 /usr/sbin/cupsd not confined
15584 /usr/lib/postfix/master confined by '/usr/lib/postfix/master ↵
        (enforce)'
15584 /usr/lib/postfix/master confined by '/usr/lib/postfix/master ↵
        (enforce)'
15623 /usr/sbin/mdnsd confined by '/usr/sbin/mdnsd (enforce)'
15623 /usr/sbin/mdnsd confined by '/usr/sbin/mdnsd (enforce)'
24997 /usr/sbin/sshd not confined
```

Achtung

Die Ausgabe von `unconfined` ist teilweise fehlerhaft, wenn ein Netzwerkdienst zum Zeitpunkt des Netstat-Aufrufes läuft und sich bei der Überprüfung der Überwachung durch AppArmor bereits beendet hat! Dann wird dieser Befehl von `unconfined` so angezeigt, als ob er nicht überwacht werden würde. Er lief, aber die Überwachung konnte nicht verifiziert werden.

`unconfined` unterstützt auch die Option `--paranoid` mit der Sie alle aktuell laufenden Prozesse auf ihren AppArmor-Status hin analysieren. Ohne diese Option werden nur, wie oben gezeigt, die Netzwerkdienste betrachtet. Mit dieser Option werden *alle* laufenden Prozesse analysiert.

```
# unconfined --paranoid
1 /sbin/init not confined
854 /sbin/udevd not confined
3103 /usr/bin/dbus-daemon not confined
3142 /sbin/acpid not confined
3150 /usr/sbin/hald not confined
3157 /usr/lib/hal/hald-addon-acpi not confined
3260 /usr/lib/hal/hald-addon-storage not confined
14868 /bin/bash (/bin/bash) not confined
15584 /usr/lib/postfix/master confined by '/usr/lib/postfix/ ➥
        master (enforce)'
15592 /usr/lib/postfix/pickup confined by '/usr/lib/postfix/ ➥
        pickup (enforce)'
15593 /usr/lib/postfix/qmgr confined by '/usr/lib/postfix/qmgr ➥
        (enforce)'
15623 /usr/sbin/mdnsd confined by '/usr/sbin/mdnsd (enforce)'
15662 /sbin/syslog-ng not confined
15665 /sbin/klogd confined by '/sbin/klogd (enforce)'
15666 /usr/bin/perl (/usr/bin/perl -w /usr/sbin/unconfined ➥
        --paranoid) not confined
24997 /usr/sbin/sshd not confined
...
```

6.4 AppArmor-Konfigurationsdateien

6.4.1 logprof.conf

Diese Konfigurationsdatei besteht aus vier Bereichen:

- `[qualifier]`: Hier können Einschränkungen für die Erzeugung von Regeln beim Aufruf externer Programme definiert werden.

- `[required_hats]`: Für diese Applikationen müssen mindestens die hier angegebenen Hats definiert werden.
- `[defaulthat]`: Hier wird der Default Hat für die verschiedenen Programme mit Changehat-Unterstützung definiert.
- `[globs]`: Hier werden in Abhängigkeit von regulären Ausdrücken Globbing-Vorschläge für `logprof` definiert.

Qualifiers

Hier können Sie Befehle definieren, bei deren Aufruf `logprof` nicht alle verschiedenen Möglichkeiten angeben soll. So ist es für die Systemstabilität gefährlich, wenn der Befehl `/bin/bash` von einem eigenen Profil überwacht wird. Wenn die Bash von einem anderen von AppArmor überwachten Prozess aufgerufen wird, soll `logprof` daher nur drei verschiedene Möglichkeiten anbieten: (I)nherit, (U)nconstrained und (D)eny.

```
[qualifiers]
  # things will be painfully broken if bash has a profile
  /bin/bash     = iu
```

Ähnliches gilt für einige andere Kommandos, die nicht richtig funktionieren, wenn sie von AppArmor überwacht werden. Hier dürfen nur die Möglichkeiten (U)nconstrained und (D)eny angeboten werden:

```
  /bin/mount    = u
  /etc/init.d/subdomain = u
  /sbin/cardmgr = u
  /sbin/subdomain_parser = u
  /usr/sbin/genprof = u
  /usr/sbin/logprof = u
  /usr/lib/YaST2/servers_non_y2/ag_genprof = u
  /usr/lib/YaST2/servers_non_y2/ag_logprof = u
```

Required Hats

Der Required-Hats-Abschnitt in der Datei `/etc/apparmor/logprof.conf` wird von dem Befehl `autodep` benötigt. Dieser Befehl ermittelt aus dieser Datei, welche Hats in einem Basisprofil für die entsprechende Applikation angelegt werden müssen. Obwohl in der Konfigurationsdatei auch Hats für den Secure Shell Daemon definiert wurden, ist die ChangeHat-Unterstützung für den SSHD im Moment (SUSE 10.1) nicht funktionstüchtig.

Default Hats

Hier definieren Sie den Default Hat, der bei den entsprechenden Applikationen nach einem ChangeHat-Ereignis von `logprof` für die Anpassung gewählt werden soll. Bei dem Apache Webserver ist das immer `DEFAULT_URI`. Lediglich wenn Sie einen eigenen Hat definieren, weicht `logprof` hiervon ab.

Globs

Damit `logprof` Ihnen intelligente Vorschläge für die Erzeugung der Regeln machen kann, haben die AppArmor-Entwickler hier eine Liste von Vorschlägen vorbereitet. Greift Ihre Applikation zum Beispiel auf ein Prozessunterverzeichnis in `/proc`, zum Beispiel `/proc/1234` zu, so ist es sehr wahrscheinlich, dass die PID sich bei dem nächsten Aufruf geändert hat. Eine Regel, die den Zugriff auf dieses Verzeichnis erlaubt, wäre daher unsinnig. Sinnvoller ist ein Zugriff auf alle Verzeichnisse in `/proc`[2].Um nun einen entsprechenden Vorschlag zu erzeugen, befindet sich Folgendes in dem Abschnitt Globs:

```
# strip pid numbers from /proc accesses
^/proc/\d+/                          = /proc/*/
```

6.4.2 subdomain.conf

Diese Datei enthält aktuell vier Variablen, die das Verhalten des AppArmor-Systems steuern. Hierbei handelt es sich um:

- `SUBDOMAIN_PATH`. Diese Variable definiert das Verzeichnis, in dem AppArmor seine Profile sucht. Bei aktuellen SUSE Linux-Distributionen ist dies das Verzeichnis `/etc/apparmor.d`, auch wenn die Dokumentation in der Konfigurationsdatei etwas anderes angibt.
- `SUBDOMAIN_MODULE_PANIC`. Diese Variable definiert, wie sich das System verhalten soll, wenn bei dem Laden des AppArmor-Kernel-Moduls ein Fehler auftritt:
  - `warn`: Der Fehler wird protokolliert. Dies ist der Default.
  - `build`: Es wird der Versuch unternommen, ein neues Kernel-Modul für den laufenden Kernel zu bauen. Schlägt dies fehl, erfolgt eine Protokollierung.
  - `panic`: Der Fehler wird protokolliert, und das System wechselt in den Runlevel 1. Damit wird sichergestellt, dass ein Angriff nicht möglich ist.
  - `build-panic`: Es wird zunächst der Versuch unternommen, ein neues Kernel-Modul zu bauen. Schlägt dieser Versuch fehl, erfolgt dieselbe Reaktion wie bei `panic`.
- `SUBDOMAIN_ENABLE_OWLSM`. Hiermit könnten Sie das `owlsm`-Kernel-Modul zusätzlich laden. Leider unterstützen aktuelle Kernel dieses LSM-Modul nicht mehr. Dieses Modul implementiert einige Ideen des Openwall-Linux-Patches bezüglich Race Conditions in temporären Verzeichnissen. Zwei Beschränkungen werden von diesem Modul erzeugt:
  - Hardlinks: Sie dürfen nur dann einen Hardlink auf eine Datei erzeugen, wenn Sie der Besitzer der Datei sind oder das Recht CAP_CHOWN besitzen.
  - Symbolische Links: Symbolische Links werden nur verfolgt, wenn diese von dem Besitzer des Verzeichnisses oder des aktuellen Prozesses erzeugt wurden.

[2] Wirklich sinnvoll wäre natürlich nur ein Zugriff auf `/proc/[0-9]*`, denn in dem Verzeichnis `/proc` befinden sich noch viele weitere Dateien, deren Zugriff meist nicht gewünscht wird. Obwohl AppArmor das unterstützt, wird es hier nicht genutzt.

- APPARMOR_ENABLE_AAEVENTD. Dieser Dienst wird für die Berichterzeugung benötigt, die Sie über Yast konfigurieren können (siehe Abschnitt 5.4). Sobald Sie diese Berichte aktiviert haben, wird der Dienst `aa-eventd` automatisch über sein Startscript `/etc/init.d/aaeventd` gestartet. Für dieses Perl-Script gibt es keine Manpage. Die ist aber auch nicht erforderlich, da das Script als einzige Option die Angabe einer PID-Datei erlaubt.

6.4.3 reports.conf

Diese XML-Datei ist nicht für die direkte Modifikation gedacht. Sie wird von Yast mit Informationen gefüllt und von dem Befehl `reportgen.pl` für die Berichterzeugung verwendet. Eine Dokumentation existiert nicht.

6.4.4 severity.db

Diese Datei definiert numerisch die potenzielle Gefahr, die von dem Zugriff ausgeht, auf einer Skala von 0 (keine Gefahr) bis 10 (Vollzugriff auf das System). Dieser Schweregrad wird von `logprof` in seinen Dialogen angezeigt, damit Sie besser entscheiden können, ob Sie den Zugriff erlauben möchten oder nicht. Die Textdatei ist sehr einfach aufgebaut. Für jede Datei ist der Schweregrad für das Lesen, Schreiben und Ausführen angegeben.

```
# filename      r w x
# 'hard drives' are generally 4 10 0
/**/lost+found/**       5 5 0
/boot/**        7 10 0
/etc/passwd*    4 8 0
```

6.4.5 reports.crontab

Hier wird definiert, wann die Berichte erzeugt werden müssen.

6.5 AppArmor-Syntax

Um manuelle Änderungen an den AppArmor-Profilen durchführen zu können oder diese komplett von Hand zu erstellen, ist es erforderlich, dass Sie die Syntax der Profildateien kennen. Diese Syntax ist auch in der Manpage `apparmor.d(5)` erläutert.

6.5.1 Kommentare

Zunächst die wichtigste Information zur AppArmor-Konfigurationssyntax: Mit einem # wird ein Kommentar eingeleitet. Alle weiteren Zeichen auf derselben Zeile werden ignoriert. Die einzige Ausnahme stellt die Include-Direktive dar:

```
#include <abstractions/base>
```

6.5.2 Include-Direktiven

Include-Direktiven beginnen immer mit der Direktive `#include`. Hieran schließt sich ein absoluter oder relativer Pfad zur einzulesenden Datei an. Diese Datei wird dann zusätzlich zum Profil eingelesen. Ein absoluter Pfad wird in doppelte Anführungszeichen eingeschlossen. Ein relativer Pfad wird in spitze Klammern eingeschlossen. Relative Angaben werden relativ zum Verzeichnis `/etc/apparmor.d` gesucht.

```
#include <abstractions/base>
#include "/etc/apparmor.d/abstractions/base"
```

6.5.3 Profil

Jedes Profil folgt in seinem Aufbau der folgenden Struktur:

```
[Include-Direktive]
[Variablen-Zuweisung]
  Programmname [flags=(complain|audit)] {
     [ (Regel | Kommentar | Include-Direktive | Capability) ... ]
     [Sub-Profil]
  }
```

Wie müssen Sie diese Struktur lesen? Nun, zu Beginn kann (die eckigen Klammern geben eine optionale Komponente an) eine Include-Direktive stehen. Anschließend können Variablen definiert werden. Hierauf folgt der Name des Programms. Dieses Programm muss mit seinem absoluten Pfad angegeben werden. Auf das Programm kann die Option `flags` folgen. Hiermit wird der Modus des Profils angegeben. Fehlt diese Option, ist das Profil im Enforce-Modus. Nun folgen in geschweiften Klammern einzelne Regeln (siehe Abschnitt 6.5.4), Kommentare (siehe Abschnitt 6.5.1) oder weitere Include-Direktiven (siehe Abschnitt 6.5.2). Vor der schließenden geschweiften Klammer kann noch ein Subprofil angegeben werden. In seltenen Fällen wird dies auch außerhalb der geschweiften Klammern (mit einem absoluten Pfad) angegeben.

6.5.4 Regel

Laut der AppArmor-Manpage (`apparmor.d(5)`) kennt AppArmor sowohl Regeln für den Zugriff auf Dateien als auch für den Zugriff auf das Netzwerk. Letztere wurden jedoch in der aktuellen Version nicht implementiert und stellen ein Relikt aus alten Zeiten dar. Zukünftige Versionen werden wieder diese Funktion mit einer abweichenden Syntax unterstützen. Im Moment kann daher nur der Zugriff auf Dateien beschränkt werden. Eine Regel besteht aus einem absoluten Dateinamen oder einem Dateiglobbing, gefolgt von den Zugriffsrechten. Sowohl der Dateiname als auch das Globbing kann Variablen (siehe Abschnitt 6.5.5) enthalten. Im Folgenden sehen Sie ein Beispiel mit zwei Regeln:

```
/dev/tty*      r,
/etc/passwd    r,
```

Für das Globbing stehen verschiedene Sonderzeichen zur Verfügung:

- `*` Platzhalter für jedes beliebige Zeichen außer `/`.
- `**` Platzhalter für jedes beliebige Zeichen einschließlich `/`. Hiermit werden also auch Unterverzeichnisse eingeschlossen.
- `?` Platzhalter für ein beliebiges Zeichen außer `/`.
- `[abc]` Platzhalter für ein Zeichen: `a`, `b` oder `c`.
- `[a-c]` Platzhalter für ein Zeichen: `a`, `b` oder `c`.
- `{ab,cd}` Platzhalter für entweder `ab` oder `cd`.

Als Zugriffsrechte können Kombinationen der folgenden Rechte angegeben werden[3]:

- `r`: Hiermit wird einem Programm erlaubt, die Datei zu lesen. Lesezugriff wird bei Scripts benötigt. Außerdem erlaubt dieses Recht bei ausführbaren Dateien ein Debugging mit dem `ptrace(2)`-Systemcall.
- `w`: Hiermit wird dem Programm ein Schreibzugriff auf die Datei erlaubt. Auch für das Löschen einer Datei wird dieses Recht benötigt.
- `ux`: Hiermit wird der Aufruf des Programms ohne AppArmor-Überwachung erlaubt. Häufig ist es sinnvoller, wenn ein überwachtes Programm eine privilegierte Aktion ausführen möchte, diese in einem eigenen externen Programm zu implementieren und dann den Aufruf dieses externen Programms mit `ux` zu erlauben. Bei der Verwendung von `ux` wird nicht die Ausführungsumgebung gelöscht. Umgebungsvariablen wie `LD_PRELOAD` bleiben erhalten. Darüber kann ein Angreifer bösartigen Code einschleusen.
- `Ux`: Dieses Recht arbeitet wie `ux`, jedoch wird die Ausführungsumgebung zuvor gelöscht. Hierbei wird derselbe Mechanismus wie bei SetUID-Programmen eingesetzt. Die Manpage `ld.so` liefert hier Hintergrundinformationen.
- `px`: Hiermit darf ein Programm aufgerufen werden. Dieses Programm muss über ein eigenes Profil verfügen. Ist kein Profil für die Applikation geladen, wird der Zugriff abgelehnt. Ähnlich `ux` wird die Ausführungsumgebung nicht zurückgesetzt.
- `Px`: Dieses Recht ist analog dem Recht `px` und stellt das Löschen der Umgebungsvariablen sicher.
- `ix`: Mit diesem Recht erbt das aufgerufene Programm das aktuelle Profil. Das aktuelle Profil muss dann alle Zugriffe des neu aufgerufenen Programms erlauben. Ein Löschen der Umgebung ist hier nicht vorgesehen und auch nicht nötig, da das neue Programm nicht über erweiterte Privilegien verfügt.
- `m`: Dieses Recht stellt sicher, dass das aufgerufene Programm unter Anwendung des `PROT_EXEC`-Flags von `mmap(2)` in den Speicher geladen wird. So werden die Speicherseiten, die ausführbaren Programmcode enthalten, als *executable* markiert. Dies verhindert im Gegenzug die Ausführung von Daten-Speicherseiten. Hiermit

[3] Achtung: Nicht alle Optionen stehen überall zur Verfügung. `Px`, `Ux` und `m` sind noch sehr neu. Möglicherweise existieren diese nicht auf Ihrer Plattform.

kann der Administrator einigen Buffer-Overflows und Formatstring-Fehlern Einhalt gebieten.

- `l`: Hiermit wird einem Programm erlaubt, den angegebenen Link zu erzeugen.

Sämtliche Rechte mit `x` schließen sich logischerweise gegenseitig aus. Sie dürfen nur eines dieser Rechte in der Kombination angeben.

6.5.5 Variablen

AppArmor erlaubt die Verwendung von Variablen in den Profildateien. Mehrfache Zuweisungen an Variablen sind erlaubt, jedoch müssen sie vor dem Start des Profils beginnen. Hiermit können sehr komplexe Setups realisiert werden (siehe Abschnitt 8.3). Aktuell verwendet AppArmor hier lediglich die Variablen `@{HOME}` und `@{HOMEDIR}`.

```
@{HOME}=@{HOMEDIRS}/*/ /root/
@{HOMEDIRS}=/home/
```

Diese werden in der Datei `<tunables/home>` gesetzt, die von den Profilen über Include-Direktiven eingelesen wird.

6.5.6 Capabilitys

Verschiedenste Programme müssen privilegierte Aktionen durchführen. Diese wurden bereits vor längerer Zeit in unterschiedliche Capabilitys unterteilt (siehe Abschnitt 2.3). AppArmor überwacht die Verwendung der Capabilitys. Damit ein Programm eine *Capability* verwenden darf, muss das Profil eine entsprechende Regel aufweisen. Diese Regeln bestehen aus dem Schlüsselwort `capability` und dem Namen der Capability (siehe die `capabilities(7)`-Manpage) in Kleinbuchstaben.

```
capability net_raw,
capability setuid,
```

Der Zugriff für die Systemaufrufe `mount(2)` und `umount(2)` kann so aber nicht gewährt werden. Die Capability *CAP_SYS_ADMIN* genügt hier nicht. Kein Programm, das von AppArmor überwacht wird, darf diese Systemcalls aufrufen. Ebenso kann das Laden der AppArmor-Richtlinien keinem von AppArmor überwachten Programm erlaubt werden.

6.6 Abstractions

Als *Abstractions* bezeichnet AppArmor große Regeldateien, die von mehreren Programmen gemeinsam genutzte Regeln enthalten. Diese Abstractions können dann von den entsprechenden Profilen mit Include-Direktiven geladen werden. Das erleichtert die Administration und sorgt für übersichtliche Regeldateien. Die Abstractions befinden sich in dem Verzeichnis `/etc/apparmor.d/abstractions`. Aktuell liegen dort die folgenden Dateien:

- `abstractions/audio`: Hier wird der Zugriff auf Gerätedateien für Audioanwendungen erlaubt.
- `abstractions/authentication`: Diese Datei erlaubt den Zugriff auf Dateien und Dienste, die für die Authentifizierung von Benutzern benötigt werden.
- `abstractions/base`: Diese Datei enthält Zugriffsregeln für Dateien, die von fast allen Programmen benötigt werden.
- `abstractions/bash`: Hier wird der Zugriff auf die typischen von der Bash benötigten Dateien erlaubt.
- `abstractions/consoles`: Hier wird der Zugriff auf Konsolen und Terminals erlaubt. Jedes Programm, das mit dem Benutzer interagiert, benötigt diese Zugriffe.
- `abstractions/fonts`: Diese Datei erlaubt den Zugriff auf Zeichensätze und die entsprechenden Bibliotheken.
- `abstractions/gnome`: Hier wird Lese- und Schreibzugriff auf die Gnome-Konfigurationsdateien und lesender Zugriff auf die Gnome-Bibliotheken gewährt.
- `abstractions/kde`: Hier wird analog der Zugriff auf die KDE-Konfigurationsdateien und -Bibliotheken gewährt.
- `abstractions/kerberosclient`: Diese Datei erlaubt den Zugriff auf die Kerberos-Bibliotheken und die Konfigurationsdateien.
- `abstractions/nameservice`: Hier werden die Regeln für die Namensauflösung via DNS, LDAP, NIS, SMB und den lokalen Dateien zusammengefasst.
- `abstractions/perl`: Diese Datei definiert den Lesezugriff auf die Perl-Module.
- `abstractions/user-(download|mail|manpages|tmp|write)`: Diese Dateien erlauben Anwenderprogrammen den Zugriff auf entsprechende Dateien. Dies wird sowohl von Browsern und Dateibetrachtern als auch von E-Mail-Programmen genutzt.
- `abstractions/wutmp`: Diese Datei erlaubt den schreibenden Zugriff auf die Dateien `/var/log/wtmp` und `/var/log/utmp`. Zum Beispiel der SSH-Daemon benötigt diese Zugriffe.
- `abstractions/X`: Diese Datei erlaubt lesenden Zugriff auf X-Bibliotheken, -Konfigurationsdateien, Sockets etc.

Wenn Sie das Werkzeug `logprof` einsetzen und `logprof` erkennt, dass ein Zugriff durch eine Abstraction leichter erlaubt werden kann, schlägt `logprof` den Einsatz der Abstraction vor. So können die Profile klein und übersichtlich gestaltet werden. Dennoch sollten Sie sich vor dem Einsatz der Abstraction einen Überblick über ihre Regeln verschaffen. Möglicherweise erlauben Sie mit einer Abstraction mehr, als Sie erlauben möchten.

7 AppArmor-Installation

In diesem Kapitel werde ich die Installation von AppArmor beschreiben. Hierbei werde ich zum einen auf die Ubuntu- und Debian-Distribution eingehen, für die es inzwischen fertige Pakete gibt, und zum anderen anhand der Fedora-Distribution die Installation von AppArmor aus den Quellen beschreiben. Falls Sie eine andere Distribution einsetzen, finden Sie hoffentlich hier genügend Hinweise, um auch auf Ihrer Distribution AppArmor einzusetzen.

Achtung

AppArmor kann nicht gleichzeitig mit SELinux eingesetzt werden. Falls Sie eine Distribution verwenden, die SELinux nutzt, so müssen Sie SELinux abschalten. Im Kernel können aber beide gleichzeitig vorhanden sein.

Nach der Installation stehen Ihnen alle in diesem Kapitel beschriebenen Funktionen zur Verfügung. Lediglich auf die Yast-Oberfläche müssen Sie verzichten.

7.1 Ubuntu und Debian

Für *Ubuntu* Dapper Drake gibt es unter *http://www.linuxalert.org/ubuntu/apparmor/* fertige Pakete. Diese wurden von Markus Runesson zur Verfügung gestellt. Diese Pakete funktionieren sowohl unter Ubuntu als auch unter *Debian* Etch. Um die Pakete aus dem Repository zu verwenden, sollten Sie zunächst die Liste der Repositories Ihrer Distribution anpassen. Diese Liste wird in der Datei `/etc/apt/sources.list` gespeichert. Tragen Sie dort zusätzlich die folgende Zeile ein:

```
deb http://www.linuxalert.org/ubuntu/apparmor/ /
```

Mit einem anschließenden `apt-get update` aktualisieren Sie die Paketlisten. Nun können Sie die Pakete installieren. Hierzu verwenden Sie:

```
# apt-get install apparmor-utils libapparmor1 apparmor-parser
```

Den Kernel müssen Sie manuell herunterladen und installieren:

```
# wget  http://www.linuxalert.org/ubuntu/apparmor/linux-image ➥
        -2.6.15-21-386_2.6.15-21.32mr1_i386.deb
# dpkg -i linux-image-2.6.15-21-386_2.6.15-21.32mr1_i386.deb
```

Anschließend müssen Sie noch den Kernel im Grub-Menü aktivieren. Tragen Sie hierzu den folgenden Block in der Datei `/boot/grub/menu.lst` ein:

```
title           Debian GNU/Linux AppArmor
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.15-21-386 root=/dev/hda1 ro
initrd          /boot/initrd.img-2.6.15-21-386
savedefault
boot
```

Nach einem Reboot wählen Sie im Grub-Menü den neuen Kernel aus. Prüfen Sie nach dem Boot, ob der richtige Kernel geladen und AppArmor aktiviert wurde:

```
ubuntu:~# uname -a
Linux station6 2.6.15-21-386 #1 PREEMPT Sun Apr 23 17:28:37 CEST ➥
         2006 i686
GNU/Linux
ubuntu:~# /etc/init.d/apparmor-parser status
apparmor module is loaded.
49 profiles are loaded.
49 profiles are in enforce mode.
0 profiles are in complain mode.
Out of 54 processes running:
0 processes have profiles defined.
0 processes have profiles in enforce mode.
0 processes have profiles in complain mode.
```

Herzlichen Glückwunsch. Sie haben AppArmor erfolgreich installiert.

7.2 Installation aus den Sourcen

Ich werde die Installation aus den Sourcen am Beispiel der *Fedora Core* 5-Distribution vorstellen. Dennoch sollte die hier beschriebene Vorgehensweise auch auf anderen Distributionen zum Erfolg führen.

Leider stellt Novell bisher keine aktuellen Kernel-Patches zur Verfügung. Dies mag sich mit Erscheinen dieses Buches geändert haben. Sie können daher nur die Patches vom Januar 2006 verwenden. Diese finden Sie unter *http://forge.novell.com/modules/xfcontent/downloads.php/apparmor/Development%20-%20January%20Snapshot/*. Für die Übersetzung benötigen Sie nun noch einen Vanilla-Linux-Kernel. Da der Patch für den Linux-Kernel 2.6.15 ist, laden Sie diesen auch herunter:

```
http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.15.7.tar.bz2
```

Entpacken Sie zunächst den Kernel, und patchen Sie ihn mit den AppArmor-Patches. Anschließend rufen Sie die Kernel-Konfiguration auf und aktivieren AppArmor unter SECURITY OPTIONS. Gleichzeitig können Sie SELinux deaktivieren.

```
# cd /usr/src
# tar xjf /tmp/linux-2.6.15.7.tar.bz2
# cd linux-2.6.15.7
# patch -p1 < /tmp/aa_2.0-2.6.15.patch
# patch -p1 < /root/aa_namespace_sem-2.6.15.patch
# make menuconfig
# make modules bzImage
# make modules_install install
```

Um anschließend die Bibliothek und die Userspace-Programme zu installieren, entpacken Sie die Quelltext-Pakete und kompilieren und installieren die Software.

```
# tar xzf /tmp/libapparmor-2.0.tar.gz
# cd libapparmor-2.0
# make
# make install
# tar xzf /tmp/apparmor-parser-2.0.tar.gz
# cd apparmor-parser-2.0
# make
# make install
# tar xzf apparmor-utils-2.0.tar.gz
# cd apparmor-utils-2.0
# make install
# tar xzf apparmor-profiles-2.0.tar.gz
# cd apparmor-profiles-2.0
# make install
```

Bei der Installation des AppArmor-Parsers wird automatisch für die Distribution das richtige Startscript in das Verzeichnis `/etc/init.d` kopiert. Dies funktioniert für die Distributionen von *SUSE*, *Fedora Core*, *Redhat*, *Slackware* und *Debian*. Aktivieren Sie den Dienst `subdomain`:

```
chkconfig --add subdomain
chkconfig subdomain on
```

Nach einem Reboot wird AppArmor automatisch gestartet.

Tipp

Wenn Sie die neuesten AppArmor-Funktionen nutzen möchten, können Sie auch die Patches aus den *SUSE*-Kerneln extrahieren. Diese sind in den Source-RPM-Paketen (`*.src.rpm`) der SUSE-Kernel enthalten. Dieses Vorgehen empfiehlt sich jedoch nur für den erfahrenen Administrator und kann aufgrund der vielen Unwägbarkeiten hier nicht erläutert werden.

8 AppArmor für Fortgeschrittene

8.1 Erzeugen der Log-Markierung für logprof

Der Befehl `logprof` kann unter Angabe einer *Protokollmarkierung* gestartet werden. Der Befehl analysiert dann nur die Protokolleinträge ab dieser Markierung. Der Befehl `genprof` erzeugt selbst die Markierung und ruft `logprof` anschließend mit der Markierung auf. Das können Sie aber auch selbst. Zunächst müssen Sie prüfen, ob Ihr System den Auditd-Daemon einsetzt. Ist dies der Fall, so müssen Sie lediglich den Zeitstempel der letzten Audit-Meldung auslesen:

```
type=APPARMOR msg=audit(1155038603.784:322): REJECTING access to ⬅
         capability 'net_raw' (nmap(28483) profile /usr/bin/nmap ⬅
         active /usr/bin/nmap)
```

Bei der angegebenen Protokollmeldung entspricht die Markierung also 1155038603.784:322. Der entsprechende Aufruf von `logprof` lautet dann:

```
$ logprof -m 1155038603.784:322
```

Wird der Auditd-Daemon nicht eingesetzt, werden die AppArmor-Meldungen in die Datei `/var/log/messages` geschrieben. Da die Zeitstempel hier nicht die notwendige Genauigkeit aufweisen, benötigt `logprof` an dieser Stelle eine Markierung. Diese können Sie selbst mit dem Kommando `logger` erzeugen. Zunächst benötigen Sie eine beliebige 16-stellige hexadezimale Zahl. `genprof` selbst verwendet hierzu die MD5-Prüfsumme des aktuellen Datums. Möchten Sie es genauso machen, verwenden Sie:

```
$ date | md5sum
7d5f01a70c0d465737a8dc8625016ce4  -
$ logger -p kern.warn 'GenProf: 7d5f01a70c0d465737a8dc8625016ce4'
```

Der anschließende Aufruf von `logprof` erfolgt dann mit:

```
$ logprof -m 7d5f01a70c0d465737a8dc8625016ce4
```

8.2 ChangeHat

Die *ChangeHat*-Funktionalität erlaubt es einem Programm, für die Ausführung von Programmteilen in ein AppArmor-Subprofil zu wechseln. Momentan wird die Unterstützung für den Apache2 ausgeliefert, und es existiert ein experimentelles Modul für alle Applikationen, die PAM verwenden.

8.2.1 Apache 2.0 und mod_apparmor

Viele Websites basieren heute auf Webapplikationen, die den Inhalt dynamisch durch den Einsatz von serverbasierten Scriptsprachen und Datenbanken erzeugen. Bei dem Einsatz des Apache Webservers wünschen sich viele Administratoren Möglichkeiten, um die einzelnen Scripts voneinander isolieren zu können. Jedes Script soll möglichst nur die benötigten Privilegien erhalten. Der Apache selbst bietet hierfür jedoch keine Funktionen. Jedes Script verfügt über sämtliche Privilegien, die auch der *Apache*-Prozess selbst besitzt. Mit AppArmor kann dieses Problem zunächst für externe Scripts, die über den CGI[1]-Mechanismus aufgerufen werden, gelöst werden. Jedoch wird dieser Mechanismus auf den meisten Webservern aus Geschwindigkeitsgründen nicht mehr eingesetzt. Anstelle von extern aufgerufenen Script-Interpretern und CGI-Scripts werden die Script-Interpreter in den Apache in Form von Modulen eingebettet. Übliche Module sind `mod_php`, `mod_perl` und `mod_python`. Der aufwendige Start des Interpreters fällt somit weg. Außerdem können bereits aufgerufene Scripts in einer vorkompilierten Form zwischengespeichert werden, sodass auch der Schritt der Kompilierung zum Beispiel bei Perl wegfällt. Dies beschleunigt die Ausführung, aber verlangt bei der Überwachung mit AppArmor einen anderen Ansatz.

Hierfür wurde das Modul `mod_apparmor` für den Apache 2.0 geschaffen. Wird dieses Modul von dem Apache 2.0 geladen, so versucht der Apache 2.0-Webserver, für jeden URI (z.B. */webapp/index.php*) in ein entsprechendes Subprofil zu wechseln. Existiert kein Subprofil mit dem Namen, verwendet der Apache das Subprofil mit dem Namen `DEFAULT_URI`. Die Erzeugung eines derartigen Profils wurde in 5.6 beschrieben.

Jedoch gibt es auch die Möglichkeit, ein Subprofil mit einem beliebigen Namen zu erzeugen und dieses für bestimmte Teile im Webserver zu aktivieren. Hierzu stellt das Modul `mod_apparmor` zwei Optionen zur Verfügung: `AAHatName` und `AADefaultHatName`. Bei älteren Versionen heißen diese beiden Optionen `ImmHatName` und `ImmDefaultHatName`.

Mit der Option `AADefaultHatName` können Sie ein Default-Subprofil definieren. Dieses kann für jeden virtuellen Host und den eigentlichen Apache-Server unterschiedlich konfiguriert werden. Jeder virtuelle Host kann also ein eigenes Default-Subprofil besitzen. Existiert das angegebene Subprofil nicht, verhält sich der Apache so, als ob die Direktive nicht gesetzt sei. Das bedeutet, er sucht zunächst nach einem Profil entsprechend der URI des Zugriffs und verwendet, falls auch ein derartiges Profil nicht existiert, das Subprofil `DEFAULT_URI`.

[1] Common Gateway Interface

Das mit dem Parameter `AADefaultHatName` gewählte Subprofil kann dann mit dem Parameter `AAHatName` überschrieben werden. Damit können Sie für bestimmte Bereiche innerhalb des `DocumentRoot` des Webservers oder des virtuellen Hosts vom `AADefaultHatName` abweichende Subprofile aktivieren. Der Apache Webserver bietet hierfür die Parameter `Location` und `Directory`. Wenn Sie den Ort als absoluten Pfad im Linux-Verzeichnisbaum angeben möchten, verwenden Sie die Option `Directory`. Falls Sie den Ort in Abhängigkeit von der URL angeben möchten, nutzen Sie `Location`. Normalerweise verwenden Sie diese Optionen, um den Zugriff auf einen bestimmten Bereich zu beschränken. Eine typische Konfiguration sieht folgendermaßen aus:

```
<Directory /srv/www/htdocs/webapp>
  Order Allow,Deny
  Allow from All
  Deny from 192.168.0.0/24

  Options None
</Directory>

<Location /server-status>
  SetHandler server-status
  Order Allow,Deny
  Allow from 10.0.0.5
</Location>
```

Achtung

Das `mod_apparmor`-Modul ist nicht in der Lage, Vermischungen der `Directory`- und `Location`-Direktiven sauber aufzulösen. Daher sollten Sie in Ihrer Konfiguration entweder nur `Directory` oder `Location` verwenden.

Um die Funktion in dem Beispiel aus dem Abschnitt 5.6 zu sehen, stoppen Sie den Apache und AppArmor. Editieren Sie anschließend das AppArmor-Profil für den Apache (`/etc/apparmor.d/usr.sbin.httpd2-prefork`), und benennen Sie das in Abschnitt 5.6 erzeugte Subprofil in `^info` um. Damit nun dieses Profil geladen wird, erzeugen Sie in dem Verzeichnis `/etc/httpd2/conf.d` eine neue Datei `sysinfo.conf`. Alle Dateien mit der Endung `.conf` in diesem Verzeichnis werden von dem Apache automatisch bei seinem Start geladen. Tragen Sie die folgenden Zeilen in dieser Datei ein:

```
<Directory /srv/www/htdocs/phpsysinfo>
  AAHatName info
  # Aeltere Versionen benoetigen ImmHatName info
</Directory>
```

Starten Sie AppArmor und den Apache neu, und prüfen Sie die Funktion.

Interessant ist insbesondere die Anwendung von AppArmor in einem *Shared-Hosting*-Umfeld. Eine typische Anwendung wird in Abschnitt 9.3 besprochen.

Tipp

Die mit den Parametern `AAHatName` und `AADefaultHatName` gewählten Hats haben Vorrang vor allen weiteren Hats. Wird ein entsprechender Hat nicht gefunden oder sind die Parameter nicht gesetzt, prüft AppArmor, ob ein Hat `^URI` existiert. Ist dieser nicht vorhanden, nutzt AppArmor `^DEFAULT_URI`. Fehlt auch dieser Hat, wird das globale Apache-Profil genutzt.

8.2.2 PAM und pam_apparmor

Mit diesem Modul, das noch nicht mit SUSE Linux Enterprise Server 9 oder SUSE Linux Professional 10.1 ausgeliefert wird, kann jede *PAM*-fähige Applikation bei dem Wechsel eines Benutzers in ein Subprofil wechseln. Eine typische Anwendung hierfür ist zum Beispiel die *Secure-Shell* oder auch der Login-Befehl. Bei der Anmeldung können diese Befehle direkt ein Subprofil aktivieren. Wenn die anschließend aufgerufene Shell das Profil erbt (`ixr`), ist gewährleistet, dass der Benutzer während seiner Arbeit von AppArmor überwacht wird.

Hierfür müssen Sie zunächst `pam_apparmor` installieren, falls dies auf Ihrer Distribution noch nicht der Fall ist. Hierzu laden Sie zunächst das Quelltextpaket `pam_apparmor-2.0.tar.gz` von der Homepage *http://forge.novell.com/modules/xfmod/project/?apparmor*. Für die erfolgreiche Übersetzung müssen Sie noch das `pam-devel`-Paket installieren. Dann können Sie das Quelltext-Archiv auspacken und mit `make` und `make install` übersetzen. Nun müssen Sie das PAM-Modul noch in der Konfiguration aktivieren. Editieren Sie hierzu unter SUSE die Datei `/etc/pam.d/common-session` und fügen Sie die folgende Zeile hinzu:

```
session required pam_apparmor.so
```

Achtung

Modernere Versionen von `pam_apparmor` erlauben die Auswahl der Eigenschaft, die für die Bestimmung des Profils verwendet werden soll. Während die alte Version 2.0 nach einem Profil entsprechend dem Benutzernamen sucht, suchen modernere Versionen per Default nach einem Profil entsprechend der primären Gruppe. Sie können das aber auch bei diesen Versionen einstellen und zunächst nach einem Profil für den Benutzer und dann nach einem Profil für die primäre Gruppe suchen. Wenn beides nicht gefunden wird, können Sie zusätzlich ein Default-Profil angeben.

Auf anderen Distributionen editieren Sie bitte die entsprechende Datei und fügen die Zeile hinzu. Sie können auch ein RPM-Paket erzeugen. Dann können Sie später die Installation leichter nachvollziehen, und die PAM-Konfigurationsdatei wird automatisch angepasst. Hierzu kopieren Sie das Quelltextarchiv nach `/usr/src/packages/SOURCES`[2]. Dann können Sie nach dem Entpacken des Archivs die Befehle `make` und `rpmbuild -bb pam_apparmor.spec` aufrufen. Der zweite Befehl baut das RPM-Paket und legt es in dem Verzeichnis `/usr/src/packages/RPMS/i586/` ab. Das Paket können Sie mit `rpm -i pam_apparmor-2.0-1.i586.rpm` installieren.

Um nun `pam_apparmor` mit der *Secure-Shell* zu benutzen, müssen Sie für die Secure-Shell ein Profil erzeugen. Dies kann manuell erfolgen, ist aber am leichtesten mithilfe des Befehls `genprof`. Hierzu geben Sie auf dem AppArmor-System den Befehl `genprof /usr/sbin/sshd` ein. So erzeugt der `genprof`-Befehl (siehe Abschnitt 6.3.6) ein minimales Profil für die *Secure-Shell* und aktiviert dieses im Lernmodus. Nun können Sie sich per SSH mit unterschiedlichen Benutzern anmelden und verschiedene Aktivitäten ausführen. Denken Sie daran, dass auch der Benutzer *root* von der Überwachung betroffen ist. Sie können so also zum Beispiel per SSH einen Zugang als *root* erlauben, der lediglich den Apache 2.0- Webserver neu starten darf.

Nachdem Sie sich wieder per SSH von dem System abgemeldet haben, werten Sie mit `genprof` die im Lernmodus aufgezeichneten Ereignisse aus. Dabei erzeugt `genprof` dann das Profil für die *Secure-Shell*. Ein mögliches Profil ist im Folgenden abgedruckt und befindet sich auch auf der CD im Verzeichnis `/Listings/`.

```
# vim:syntax=apparmor
# Last Modified: Thu Aug 10 13:00:13 2006
#include <tunables/global>

/usr/sbin/sshd {
  #include <abstractions/authentication>
  #include <abstractions/base>
  #include <abstractions/consoles>
  #include <abstractions/nameservice>
  #include <abstractions/wutmp>

  capability chown,
  capability net_bind_service,
  capability setgid,
  capability setuid,
  capability sys_chroot,
  capability sys_tty_config,

  /dev/ptmx rw,
  /etc/environment r,
  /etc/hosts.allow r,
```

[2] Auf Systemen, die auf Red Hat Linux basieren, ersetzen Sie `/packages/` im Pfad durch `/redhat/`.

```
/etc/hosts.deny r,
/etc/motd r,
/etc/ssh/* r,
/proc/*/fd r,
/usr/sbin/sshd ixr,
/var/run/sshd.init.pid w,

^AUTHENTICATED {
}

^EXEC {
}

^PRIVSEP {
}

^PRIVSEP_MONITOR {
}

^ralf {
  #include <abstractions/authentication>
  #include <abstractions/base>
  #include <abstractions/bash>
  #include <abstractions/consoles>
  #include <abstractions/nameservice>

  capability setuid,

  /bin/bash ixr,
  /bin/grep ixr,
  /bin/hostname ixr,
  /bin/sed ixr,
  /bin/sort ixr,
  /bin/uname ixr,
  /dev/ptmx rw,
  /etc/environment r,
  /etc/manpath.config r,
  /etc/motd r,
  /etc/nntpserver r,
  /etc/profile.d r,
  /etc/sysconfig/* r,
  /home/ralf r,
  /home/ralf/* rw,
  /lib/ld-2.4.so ixr,
  /sbin/yast2 ixr,
```

```
    /usr/bin/getopt ixr,
    /usr/bin/manpath ixr,
    /usr/bin/tty ixr,
    /usr/lib/YaST2/bin/yast2-funcs r,
    /usr/lib/jvm/java-1.4.2-gcj-1.4.2.0/jre/bin/java ixr,
    /usr/share/applications/YaST2 r,
    /usr/share/applications/YaST2/* r,
    /var/log/Xorg.0.log r,
  }

  ^root {
    #include <abstractions/authentication>
    #include <abstractions/base>
    #include <abstractions/bash>
    #include <abstractions/consoles>
    #include <abstractions/nameservice>
    #include <abstractions/php5>
    #include <abstractions/wutmp>

    capability dac_override,
    capability net_bind_service,
    capability setgid,
    capability setuid,

    / r,
    /bin/basename ixr,
    /bin/bash ix,
    /bin/grep ixr,
    /bin/hostname ixr,
    /bin/ln ixr,
    /bin/sed ixr,
    /bin/sort ixr,
    /bin/stty ixr,
    /bin/uname ixr,
    /bin/usleep ixr,
    /dev/blog w,
    /etc r,
    /etc/apache2/sysconfig.d/* w,
    /etc/environment r,
    /etc/init.d r,
    /etc/init.d/apache2 ixr,
    /etc/manpath.config r,
    /etc/nntpserver r,
    /etc/profile.d r,
    /etc/rc.status r,
```

```
    /etc/sysconfig/* r,
    /proc r,
    /proc/*/stat r,
    /proc/*/statm r,
    /sbin/startproc ixr,
    /sbin/yast2 ixr,
    /usr/bin/getopt ixr,
    /usr/bin/manpath ixr,
    /usr/bin/readlink ixr,
    /usr/bin/tty ixr,
    /usr/lib/YaST2/bin/* r,
    /usr/sbin r,
    /usr/sbin/httpd2 w,
    /usr/sbin/httpd2-prefork pxr,
    /usr/share/apache2/* r,
    /usr/share/apache2/find_mpm ixr,
    /usr/share/apache2/get_includes ixr,
    /usr/share/apache2/get_module_list ixr,
    /usr/share/applications/YaST2 r,
    /usr/share/applications/YaST2/* r,
    /var/log/Xorg.0.log r,
    /var/log/apache2/rcapache2.out w,
    /var/run/httpd2.pid r,
  }
}
```

Eine kurze Erläuterung des Profils ist sicherlich angebracht. Zunächst handelt es sich um ein ganz einfaches AppArmor-Profil mit einigen Subprofilen. Die folgenden Subprofile sind Relikte der AppArmor-Entwicklung, als eine *ChangeHat*-fähige *Secure-Shell* entwickelt werden sollte:

```
  ^AUTHENTICATED {
  }

  ^PRIVSEP {
  }

  ^PRIVSEP_MONITOR {
  }
```

Ob diese Entwicklung jemals abgeschlossen wird, ist heute noch nicht klar. Dennoch fügt `genprof` diese Hats grundsätzlich in ein *Secure-Shell*-Profil ein. Aktuell kann man diese einfach ignorieren und nach Belieben auch löschen. Ob zukünftige Versionen diese nutzen, bleibt abzuwarten. Möglicherweise werden dann die Hats auch umbenannt.

Anschließend erkennen Sie jeweils einen Hat für den Benutzer *ralf* und *root*. Der Benutzer *ralf* darf nur sehr wenige Funktionen ausüben. Bei allen Befehlen, die der Benutzer *ralf* aufrufen darf, wurde darauf geachtet, dass diese Befehle das Subprofil erben. Hierzu wurden die AppArmor-Rechte `ixr` zugewiesen.

Ähnlich verhält es sich mit dem Benutzer *root*. Auch dieser Benutzer darf nur einen begrenzten Satz an Befehlen aufrufen. Alle Befehle erben das Subprofil (`ixr`). Einzige Ausnahme ist der Webserver Apache 2.0. Für den Aufruf dieses Programms wird die Existenz eines eigenen Profils gefordert:

```
/usr/sbin/httpd2-prefork pxr,
```

So läuft der Webserver immer in seinem eigenen Profil. Existiert dieses Profil nicht, kann der Webserver nicht gestartet werden. Außerdem benötigt der Benutzer *root* nun keinen Lese- und Schreibzugriff auf die Dateien des Webservers. Diese Zugriffe werden in dem Profil des Webservers verwaltet.

Tipp

Möchten Sie nur für bestimmte Benutzer ein eigenes Subprofil anlegen und alle weiteren Benutzer in einem Profil zusammenfassen oder diesen gar ohne AppArmor-Schutz Zugriff auf das System gewähren, so können Sie manuell einen Hat mit dem Namen `^DEFAULT` anlegen. Hier können Sie entweder ein Default-Profil für alle Benutzer definieren oder auch die Ausführung der Bash-Shell ohne Einschränkung (`/bin/bash uxr`) erlauben. Leider legt die aktuelle Version von `genprof` das Default-Profil bei der *Secure-Shell* fälschlicherweise unter dem Namen `^EXEC` an, während das PAM-Modul, falls es kein Subprofil mit dem entsprechenden Namen findet, das Subprofil mit dem Namen `^DEFAULT` verwendet.

8.3 Einzelne Benutzer mit unterschiedlichen Profilen

AppArmor selbst ist nicht in der Lage, zwischen unterschiedlichen Benutzern zu unterscheiden. Es gibt jedoch häufig Bedarf für einen Schutz des Systems vor den Benutzern und der Benutzer untereinander.

Tipp

Im letzten Abschnitt (siehe Abschnitt 8.2.2) wurde eine alternative Vorgehensweise besprochen. Welche der beiden Vorgehensweisen Sie wählen, hängt sicherlich von dem Einsatzzweck und der Umgebung ab.

Eine klassische Anwendung ist ein *Shared-Hosting*-Webserver, auf dem unterschiedliche Kunden ihre Webpräsenz hosten. Jeder Kunde soll sich mit der *Secure-Shell* auf dem System anmelden können und seine Webpräsenz administrieren können. Das System und vor allem auch die Webpräsenzen der anderen Kunden sollen vor seinem Zugriff jedoch geschützt werden.

Dies kann nur erfolgen, indem Sie jedem Benutzer eine eigene Shell zuweisen und entsprechende Profile für die Shell erzeugen. Über die Datei `/etc/passwd` weisen Sie dann jedem Benutzer seine eigene Shell zu.

Erzeugen Sie zunächst ein Verzeichnis, in dem Sie die Shells sämtlicher Benutzer anlegen. Kopieren Sie dann als Erstes die von Ihnen ausgewählte Shell in das Verzeichnis. Anschließend legen Sie für jeden Benutzer einen Hardlink auf die Shell an und weisen dem Benutzer die entsprechende Shell zu.

```
# mkdir /opt/shells/
# cp /bin/bash /opt/shells
# ln /opt/shells/bash /opt/shells/ralf-bash
# ln /opt/shells/bash /opt/shells/claudia-bash
# usermod -s /opt/shells/ralf-bash ralf
# usermod -s /opt/shells/claudia-bash claudia
```

Legen Sie nach diesem Muster für jeden Benutzer einen Hardlink an.

Tipp

Wenn verschiedene Benutzer über identische Zugriffsrechte verfügen sollen, genügt natürlich eine Shell für diese Gruppe. Diese Shell nennen Sie dann vielleicht `/opt/shells/<group>-shell` und erzeugen hierfür das Profil. Dann müssen Sie natürlich das Profil anpassen, sodass der Zugriff auf die Heimatverzeichnisse gewährleistet ist.

Nun müssen Sie die Profile erzeugen. Ein einfaches Grundgerüst für diese Profile kann das folgende Listing[3] sein:

```
# vim:syntax=apparmor
# Last Modified: Tue Aug  8 17:51:36 2006

@{USER}=ralf
@{HOMEDIRS}=/home/
@{HOME}=@{HOMEDIRS}/@{USER}/

/opt/shells/ralf-bash {
  #include <abstractions/base>
  #include <abstractions/bash>
  #include <abstractions/consoles>
  #include <abstractions/nameservice>
```

[3] Auf der CD in dem Verzeichnis `/Listings`.

```
  #include <abstractions/user-manpages>
  #include <abstractions/user-tmp>

  /bin/grep ixr,
  /bin/uname ixr,
  /bin/vim ixr,
  /usr/bin/manpath ixr,
  /usr/bin/tty ixr,
  /usr/bin/wget ixr,
  /bin/hostname ixr,
  /etc/vimrc r,
  /etc/manpath r,
  /etc/nntpserver r,
  /home/@{USER} r,
  /home/@{USER}/** rwl,
  /opt/shells/@{USER}-bash r,
  /usr r,
  /usr/share/** r,
}
```

Dieses Profil erlaubt der Bash den Aufruf der üblichen Befehle `grep` `vim`, `wget`, etc. Dabei darf der Benutzer nur die Dateien in seinem eigenen Verzeichnis betrachten und verändern. Die Abstraction `abstractions/user-tmp` erlaubt auch noch den Zugriff auf die temporären Verzeichnisse. Bei einem Zugriff auf andere Verzeichnisse wird der Zugriff verweigert:

```
samson:/home/ralf$ cd /var
samson:/var$ ls
/bin/ls: .: Operation not permitted
```

Dieses Profil ist natürlich sehr eingeschränkt. Selbst das Umbenennen von Dateien ist nicht erlaubt. Hierzu muss der Benutzer den Befehl `/bin/mv` aufrufen dürfen. Die Anpassung des Profils in dieser Beziehung ist aber sehr einfach. Fügen Sie einfach eine Zeile hinzu:

```
  /bin/mv   ixr,
```

Sie können so sehr einfach die Privilegien der angemeldeten Benutzer verwalten und ihren Zugriff überwachen.

Achtung

Das abgebildete Profil verwendet für die einfache Anpassung des Benutzernamens Variablen. Leider kann bei den aktuellen Versionen von AppArmor die Variable nicht in dem Namen des überwachten Programms verwendet werden. Dieser Name muss ohne Verwendung von Variablen mit seinem absoluten Pfad angegeben werden.

Tipp

Gruppieren Sie die Befehle, die Sie den verschiedenen Benutzern zur Verfügung stellen möchten, in einzelnen Dateien. Sie können dann diese mit Include-Direktiven einlesen. Erzeugen Sie zum Beispiel eine Datei `abstractions/dateizugriff`. In dieser Datei hinterlegen Sie alle Befehle, die für den Dateizugriff benötigt werden:

```
# abstractions/dateizugriff
# Uebliche Befehle fuer den Dateizugriff

  /bin/awk      ixr,
  /bin/basename ixr,
  /bin/cat      ixr,
  /bin/chmod    ixr,
  /bin/cp       ixr,
  /bin/date     ixr,
  /bin/df       ixr,
  /bin/echo     ixr,
  /bin/egrep    ixr,
  /bin/fgrep    ixr,
  /bin/gawk     ixr,
  /bin/grep     ixr,
  /bin/gunzip   ixr,
  /bin/gzip     ixr,
  /bin/hostname ixr,
  /bin/ls       ixr,
  /bin/mkdir    ixr,
  /bin/more     ixr,
  /bin/mv       ixr,
  /bin/rm       ixr,
  /bin/rmdir    ixr,
  /bin/sed      ixr,
  /bin/sort     ixr,
  /bin/tar      ixr,
  /bin/touch    ixr,
  /bin/vim      ixr,
  /bin/vi       ixr,
  /bin/zcat     ixr,
```

Genauso können Sie eine Datei `abstractions/prozessverwaltung` anlegen, in der Sie den Zugriff auf die Befehle `ps`, `top`, `kill`, `nice`, `renice` und `pidof` erlauben. Dann können Sie einfach durch Auswahl der Abstraction den Benutzern unterschiedliche Funktionen zuweisen.

Tipp

Hiermit ist es auch möglich, mehreren Benutzern die UserID 0 zuzuweisen und gewisse Aufgaben als *root* auszuführen. Die Tragweite ihrer Aktionen wird von dem Profil eingeschränkt, wenn sie eine spezielle Shell bei ihrem Login zugewiesen bekommen und diese auch nicht ändern dürfen! Ein schreibender Zugriff auf die Datei `/etc/passwd` muss für diese Benutzer natürlich unterbunden werden!

9 Typische AppArmor-Administrationsvorgänge

9.1 Weiteres Cacheverzeichnis für den Squid-Proxy

Der *Squid*-Proxy ist ein häufig eingesetzter Caching-Proxy. Ein Caching-Proxy speichert die übertragenen Daten zusätzlich in einem Zwischenspeicher ab, um bei erneuter Anfrage die Daten schneller aus dem Cache liefern zu können. AppArmor verfügt für die Überwachung des Squid-Proxys über ein Profil, das für den Standardfall vollkommen ausreicht:

```
# $Id: usr.sbin.squid 12 2006-04-12 21:35:41Z steve-beattie $
# ----------------------------------------------------------------------
#
#    Copyright (C) 2002-2006 Novell/SUSE
#
#    This program is free software; you can redistribute it and/or
#    modify it under the terms of version 2 of the GNU General Public
#    License published by the Free Software Foundation.
#
# ----------------------------------------------------------------------
# vim:syntax=apparmor

#include <tunables/global>

/usr/sbin/squid
  #include <abstractions/base>
  #include <abstractions/consoles>
  #include <abstractions/kerberosclient>
  #include <abstractions/nameservice>

  capability setgid,
  capability setuid,

  /usr/lib/squid/* rix,
  /usr/sbin/squid rix,
  /usr/sbin/unlinkd ixr,
```

```
/var/cache/squid/** lrw,

/dev/tty rw,
/etc/mtab r,
/etc/squid/* r,
/proc/*/mounts r,
/proc/mounts r,
/usr/share/squid/** r,
/var/log/squid/access.log w,
/proc/sys/kernel/ngroups_max r,
/var/log/squid/cache.log rw,
/var/log/squid/store.log w,
/var/run/squid.pid lrw,
```

Aus Geschwindigkeitsgründen empfiehlt sich die Verwendung mehrerer Cache-Verzeichnisse auf getrennten Festplatten. Der Squid-Proxy verteilt dann den Cache gleichmäßig über alle Festplatten. So verteilen sich dann auch die Zugriffe entsprechend. Hierzu werden in der Squid-Konfiguration einfach mehrere Cache-Verzeichnisse angegeben:

```
cache_dir ufs /var/cache/squid1 100 16 256
cache_dir ufs /var/cache/squid2 100 16 256
```

Damit AppArmor den Zugriff erlaubt, müssen Sie das Profil anpassen. Zwei Varianten sind in diesem Fall denkbar:

1. Sie erlauben explizit den Zugriff für jedes Verzeichnis:

   ```
   /var/cache/squid1/** lrw,
   /var/cache/squid2/** lrw,
   ```

2. Sie benutzen File-Globbing:

   ```
   /var/cache/squid*/** lrw,
   ```

Anschließend müssen Sie das Profil neu laden. Am einfachsten erfolgt das mit einem Neustart von AppArmor über `rcapparmor reload`. Ein Neustart des Squid ist nicht zwingend erforderlich (siehe Abschnitt 9.6).

9.2 Eine neue PHP-Anwendung für den Apache

Sie haben einen Webserver, der bereits mit AppArmor überwacht wird, und möchten nun eine neue *PHP*-Applikation in die Überwachung integrieren? Oder Ihr Webserver soll nun erstmals mit AppArmor überwacht werden, und Sie verwenden bereits eine PHP-Applikation, die von AppArmor kontrolliert werden soll? Dann haben Sie grundsätzlich drei Möglichkeiten:

1. Sie verwenden `genprof` und erzeugen ein Profil für den Webserver und berücksichtigen bei der Erzeugung des Profils im Hat `^DEFAULT_URI` auch die PHP-Applikation. Die PHP-Applikation darf dann auf alle Dateien so zugreifen, wie es auch der Webserver darf.
2. Sie verwenden `genprof` und erzeugen, sobald `genprof` es Ihnen vorschlägt, einen Hat für Ihre PHP-Applikation. Dabei können Sie für jeden URI einen eigenen Hat erzeugen oder den URI in dem Default-Hat `^DEFAULT_URI` laufen lassen.
3. Bevor Sie `genprof` starten, tragen Sie in Ihrer Webserver-Konfiguration für das Verzeichnis, in dem sich die Applikation befindet, den Parameter `AAHatName <profil>` ein. `genprof` wird dann vorschlagen, die Regeln für die Zugriffe durch die PHP-Applikation diesem Profil zuzuordnen (siehe Kapitel 8.2.1).

Welche der drei verschiedenen Varianten Sie verwenden, bleibt Ihnen überlassen, wobei natürlich die letzten beiden Varianten die bessere Vorgehensweise darstellen. Welche der beiden letzten Varianten Sie wählen, hängt sicherlich von der Komplexität der PHP-Applikation und Ihrem persönlichen Geschmack ab. Bei komplexeren Applikationen würde ich immer die letzte Variante vorziehen, da ich dann sicher bin, dass jedes PHP-Script, das sich in dem Verzeichnis befindet, für das der Parameter `AAHatName` gesetzt wurde, auch von AppArmor entsprechend überwacht wird.

Tipp

Die mit den Parametern `AAHatName` und `AADefaultHatName` gewählten Hats haben Vorrang vor allen weiteren Hats (siehe Abschnitt 8.2.1).

9.3 VirtualHosts mit Apache

Ein klassisches Anwendungsszenario für AppArmor ist ein *Shared-Hosting*-Webserver, der von einem Anbieter betrieben wird und auf dem mehrere Kunden ihre Webpräsenz pflegen. Sobald der Anbieter den Kunden die Unterstützung von serverbasierten Scriptsprachen wie PHP, Perl oder Python anbietet, besteht die Gefahr, dass Fehler in den Scripts eines Kunden sich auf den gesamten Webserver auswirken. Da aber kaum noch ein Kunde mit statischen Webseiten arbeitet, sondern fast alle Webpräsenzen dynamisch mit Scripts datenbankgestützt erzeugt werden, ist der Einsatz von Scriptsprachen die Regel. Stellen Sie sich den folgenden Fall vor: Ein Apache Webserver wurde so konfiguriert, dass er zwei virtuelle Hosts anbietet. Einer enthält ein Content Management System (CMS). Dieses besitzt einige Konfigurationsdateien, in denen auch Benutzernamen und Kennwörter gespeichert wurden. Bisher sind keine Sicherheitslücken in dem CMS bekannt. Der zweite virtuelle Host verfügt über einige selbst geschriebene und vorgefertigte PHP-Scripts. Mindestens ein Script weist eine Sicherheitslücke auf, mit der es möglich ist, den Inhalt von Dateien

anzuzeigen und auszuführen. Ein anderes Script erlaubt es, Bilder auf den Webserver zu laden. Eine Prüfung der Dateien erfolgt nicht. Auch ein Upload von Scripts und ausführbaren Programmen istmöglich. Mithilfe dieser Scripts kann ein Angreifer nun auf alle Dateien lesend und ausführend zugreifen, die auch der Apache Webserver erreichen kann. Wurde der Apache Webserver nicht in einem Chroot-Verzeichnis installiert und gestartet, handelt es sich um fast sämtliche Dateien des darunterliegenden Linux-Betriebssystems. Dazu gehören dann natürlich auch die Dateien des anderen virtuellen Hosts inklusive der Dateien mit Benutzernamen und Kennwörtern. Bei einem Webshop gehören potenziell auch Dateien mit Bankverbindungen und Kreditkarteninformationen hierzu. Wurden diese Informationen in einer Datenbank gespeichert, kann der Angreifer ein eigenes PHP-Script hochladen, das die Datenbank ausliest, und das Script über den Browser starten (Abbildung 9.1). Der Kunde des sicheren CMS wird über einen derartigen Angriff nicht erfreut sein, und der Provider wird die Nachlässigkeiten des zweiten Kunden verantworten müssen.

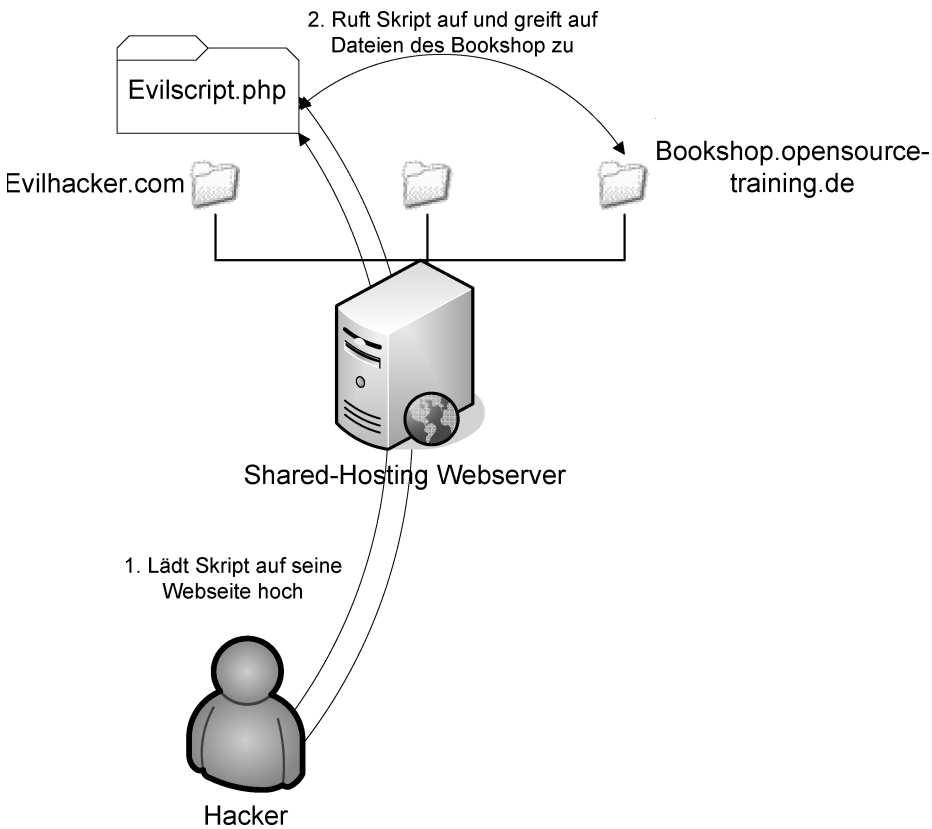

Abbildung 9.1: Bei virtuellen Hosts kann eine Sicherheitslücke alle weiteren virtuellen Hosts kompromittieren.

Der *Apache* Webserver bietet keine Möglichkeit, virtuelle Hosts voreinander zu schützen. Sämtliche Scripts werden bei dem Einsatz von `mod_php` oder `mod_perl` von einem einzigen immer gleichen Benutzer ausgeführt. UNIX-Dateirechte können hier also keinen Schutz erreichen. Werden die Scripts als CGI-Scripts ausgeführt, kann der Apache mit dem suEXEC-Mechanismus hier unterschiedliche Benutzer verwenden. So können die Scripts der verschiedenen virtuellen Hosts mit unterschiedlichen Benutzern gestartet werden. Das erlaubt eine Einschränkung der Zugriffe mit UNIX-Dateirechten. Eine andere Alternative ist der Einsatz des experimentellen Multi-Processing-Moduls (MPM) `perchild`. Dieses Apache-2-MPM erlaubt die Bindung eines virtuellen Hosts an einen Apache-Prozess (Worker). Die verschiedenen Worker können dann mit unterschiedlichen Benutzern gestartet werden. Dies würde dann auch die Einschränkung der Zugriffe beim Einsatz von `mod_php` und `mod_perl` erlauben.

Am einfachsten und wirkungsvollsten ist jedoch der Einsatz eines MAC-Systems wie AppArmor oder SELinux, das für diesen Fall eine Unterstützung bietet. Ein derartiges MAC-System schützt auch in einem weiteren Fall: Vielleicht erfolgt der Angriff gar nicht von außen, sondern ein Kunde des Shared-Hosting-Providers versucht, auf die Daten der anderen Kunden zuzugreifen und diese auszulesen. Bis zu einem gewissen Grad kann mit Dateirechten sichergestellt werden, dass auf der Kommandozeile kein Zugriff auf die Dateien der anderen Kunden möglich ist. Der Webserver muss jedoch auf alle Daten sämtlicher Kunden zugreifen können. Also installiert der Angreifer in seinem Webspace lediglich ein PHP- oder auch ein Perl-Script, das die Dateien der anderen Kunden ausliest und anzeigt.

Hier soll der Einsatz von AppArmor beschrieben werden. Die Anwendung von SELinux erfolgt in dem entsprechenden Kapitel.

Um AppArmor für unterschiedliche virtuelle Hosts nutzen zu können, müssen diese zunächst konfiguriert werden. Eine typische Konfiguration für zwei virtuelle Hosts könnte folgendermaßen aussehen:

```
# Bei SUSE ist dies die Datei /etc/apache2/vhosts.d/vhost.conf

NameVirtualHost *:80

<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host.example.com
    DocumentRoot /srv/www/evil
    ServerName evil-hacker.com
    ServerAlias www.evil-hacker.com
    ErrorLog /var/log/apache2/evil
    CustomLog /var/log/apache2/evil-access common
    <Directory /srv/www/evil>
       Order allow,deny
       allow from all
    </Directory>
</VirtualHost>
```

```
<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host.example.com
    DocumentRoot /srv/www/bookshop
    ServerName bookshop.opensource-training.de
    ErrorLog /var/log/apache2/bookshop
    CustomLog /var/log/apache2/bookshop-access common
    <Directory /srv/www/bookshop>
       Order allow,deny
       allow from all
    </Directory>
</VirtualHost>
```

Nun sollte zunächst für jeden virtuellen Host ein Default-Hat definiert werden. Der Apache Webserver wechselt dann bei Anfragen an einen virtuellen Host in das entsprechende Subprofil. Hierzu fügen Sie zu jedem virtuellen Host in der Konfiguration nur die folgende Zeile hinzu:

```
AADefaultHatName evil
```

Beziehungsweise:

```
AADefaultHatName bookshop
```

Nun rufen Sie den Profilassistenten `genprof` auf. Dies kann über Yast oder auf der Kommandozeile erfolgen:

```
genprof /usr/sbin/httpd2-prefork
```

Anschließend greifen Sie mit einem Browser auf beide Websites zu und versuchen, sämtliche erlaubten Funktionen wahrzunehmen. Sobald Sie zufrieden sind, werten Sie die Zugriffe mit `genprof` aus, indem Sie (S)can auswählen. `genprof` fragt zunächst, ob der neue Hat `^evil` hinzugefügt werden soll. Anschließend werden für die Zugriffe innerhalb des Hats Regeln hinzugefügt. Ein typischer Hat könnte so aussehen:

```
^evil {
  /srv/www/evil/** r,
  /usr/share/apache2/error/** r,
  /var/log/apache2/evil w,
  /var/log/apache2/evil-access w,
}
```

Dieser Hat erlaubt dem Apache Webserver, bei Anfragen an diesen virtuellen Host auf alle Dateien unterhalb von `/srv/www/evil` und `/usr/share/apache2/error/` zuzugreifen. Zusätzlich dürfen die zwei Protokolldateien geschrieben werden. Jeder weitere Zugriff ist nicht erlaubt.

Der Hat für den zweiten virtuellen Host wird analog konfiguriert. Existiert nun noch ein besonderes `/admin`-Verzeichnis, in dem durch ein Kennwort geschützt besondere privilegierte Scripts liegen, die erweiterten Schreibzugriff benötigen, um zum

Beispiel Dateien hochzuladen, kann auch dies berücksichtigt werden. Hierzu wird die Apache-Konfiguration zunächst angepasst:

```
<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host.example.com
    DocumentRoot /srv/www/bookshop
    ServerName bookshop.opensource-training.de
    ErrorLog /var/log/apache2/bookshop
    CustomLog /var/log/apache2/bookshop-access common
    <Directory /srv/www/bookshop>
       Order allow,deny
       allow from all
    </Directory>
    <Directory /srv/www/bookshop/admin>
       AAHatName bookshop-admin
    </Directory>
</VirtualHost>
```

Nun kann ein weiterer Hat definiert werden:

```
  ^bookshop-admin {
    /srv/www/bookshop/** rw,
    /usr/share/apache2/error/** r,
    /var/log/apache2/bookshop w,
    /var/log/apache2/bookshop-access w,
  }
```

Sobald nun ein Script aus dem Verzeichnis `/srv/www/bookshop/admin` aufgerufen wird, wechselt der Webserver aus dem Hat `^bookshop` in den Hat `^bookshop-admin`. In diesem Hat erlaubt AppArmor einen schreibenden Zugriff auf die Dateien in `/srv/www/bookshop`. Natürlich müssen auch die UNIX-Dateirechte dem Apache-Webserver den schreibenden Zugriff erlauben. Ein Schreibzugriff außerhalb des Verzeichnisses oder durch den anderen virtuellen Host wird aber wirksam unterbunden.

9.4 Überwachung von Snort mit AppArmor

Das Netzwerk-Intrusion-Detection-System *Snort* ist ein Programm, das bei seinem Einsatz auf jeden Fall mithilfe von AppArmor überwacht werden sollte. Die Aufgabe von Snort ist es, bösartige oder verdächtige Netzwerkaktivitäten zu erkennen und zu melden. Dabei kann allerdings nicht ausgeschlossen werden, dass aktuelle Snort-Versionen selbst Sicherheitslücken enthalten. Dies ist in der Vergangenheit schon mehrfach der Fall gewesen. Durch die Überwachung des Snort-Prozesses mit AppArmor ist sichergestellt, dass Snort nur auf die bei AppArmor definierten Dateien zugreifen darf.

Um ein AppArmor-Profil zu erzeugen, müssen Sie zunächst Snort installieren. Hier können Sie auf die Distributionspakete zurückgreifen. Sie sollten aber immer beden-

ken, dass neue Versionen von Snort häufig auch neue Angriffe erkennen können. Ein reines Update der Regeln genügt in vielen Fällen nicht, da ein großer Teil der Intelligenz in den Präprozessoren verborgen ist.

Nach der Installation können Sie zunächst den Befehl `genprof` aurufen:

```
# genprof /usr/bin/snort
Setting /usr/bin/snort to complain mode.

Please start the application to be profiled in
another window and exercise its functionality now.

Once completed, select the "Scan" button below in
order to scan the system logs for AppArmor events.

For each AppArmor event, you will be given the
opportunity to choose whether the access should be
allowed or denied.

Profiling: /usr/bin/snort

[(S)can system log for SubDomain events] / (F)inish
```

Nun starten Sie nach Anpassung der Konfiguration den Snort-Dienst über das Startscript:

```
# /etc/init.d/snort start
```

Anschließend können Sie direkt durch Auswahl von (S)can bei `genprof` das Profil erzeugen. Ein typisches Snort-Profil ist im Folgenden angegeben:

```
# vim:syntax=apparmor
# Last Modified: Tue Aug  8 17:00:49 2006
#include <tunables/global>

/usr/bin/snort {
  #include <abstractions/base>
  #include <abstractions/nameservice>

  capability dac_override,
  capability net_raw,
  capability setgid,
  capability setuid,

  /etc/snort/*.conf r,
  /etc/snort/*.config r,
  /etc/snort/rules/*.rules r,
  /etc/snort/unicode.map r,
```

```
  /usr/bin/snort r,
  /var/log/snort/* w,
  /var/run/snort_*.pid w,
}
```

Dieses Profil sollte für die meisten Einsatzzwecke mit Snort ausreichen. Unangenehm ist jedoch die Tatsache, dass der Snort-Prozess Zugriff auf die *Capability CAP_DAC_OVERRIDE* benötigt. Eine Analyse der Fehlermeldungen bei deaktivierter Capability durch AppArmor zeigt, dass diese Capability benötigt wird, damit der Benutzer *root* Zugriff auf das Verzeichnis `/var/log/snort` erhält. Dieses Verzeichnis ist bei SUSE nur mit Rechten für den Benutzer *snort* versehen. Dies kann bei dem Start von Snort aus folgender Protokollmeldung abgelesen werden:

```
Aug  8 17:15:14 samson snort: FATAL ERROR: log directory '/var/ ➥
         log/snort' does not exist
```

Folgender Trick löst das Problem:

```
# chown root.snort -R /var/log/snort/
# chmod 770 -R /var/log/snort/
```

So erhalten sowohl der Benutzer *root* als auch die Gruppe *snort*, zu der auch der Benutzer *snort* gehört, Schreibzugriff auf das Verzeichnis `/var/log/snort`. Dann kann in dem Profil auf die Capability *CAP_DAC_OVERRIDE* verzichtet werden.

9.5 Überwachung von Shellscripts

Bereits in Abschnitt 8.3 wurde gezeigt, wie AppArmor unterschiedliche Benutzer anhand der verwendeten Shell unterscheiden kann. Hier sollen nun Shellscripts überwacht werden. Auch das ist möglich. Sinnvoll ist das zum Beispiel bei Scripts, die automatisch durch den Benutzer *root* aufgerufen werden. Ein typisches Beispiel sind die *Cron*-Jobs, die von dem System automatisch über die Datei `/etc/crontab` gestartet werden:

```
SHELL=/bin/sh
PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
MAILTO=root
#
# check scripts in cron.hourly, cron.daily, cron.weekly, and cron. ➥
         monthly
#
-*/15 * * * *   root  test -x /usr/lib/cron/run-crons && /usr/lib/ ➥
         cron/run-crons >/dev/null 2>&1
```

Diese Cron-Tabelle startet alle 15 Minuten den Befehl `/usr/lib/cron/run-crons` als Benutzer *root*. Dieser Befehl prüft, ob einer der Cron-Jobs in den Verzeichnissen `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly` oder `/etc/cron.monthly` abgearbeitet werden muss. Auch diese werden dann mit dem Benutzer *root* gestartet.

In der Vergangenheit waren vor allem selbst geschriebene Cron-Scripts häufig Einfallstore für lokale Angreifer, die durch Unterwanderung der Scripts auf *root*-Privilegien zugreifen konnten.

Die Überwachung von Shellscripts ist aber immer interessant, wenn diese Scripts automatisch auch durch andere Programme, wie Webserver, Monitoring-Software (z.B. Nagios) oder remote per SSH gestartet werden.

Ein typisches Script, das für einen Angreifer interessant sein könnte, ist das Script `/etc/cron.daily/suse.de-clean-tmp`. Dieses Script prüft täglich die Dateien in den verschiedenen temporären Verzeichnissen und löscht nach Ablauf einer Schonfrist die Dateien, die sich in diesen Verzeichnissen befinden. Ein Angreifer könnte versuchen, dem Script zusätzliche Verzeichnisse unterzuschieben. Daher ist es sinnvoll, mit AppArmor den Dateizugriff zu überprüfen.

Hierzu starten Sie wieder den Profilassistenten `genprof` auf der Kommandozeile oder mit Yast: `genprof /etc/cron.daily/suse.de-clean-tmp`. Anschließend rufen Sie als *root* das Script auf. Nach seiner Arbeit überprüfen Sie mit `genprof` die Zugriffe und erzeugen das Profil. Ein typisches Profil für das Script könnte folgendermaßen aussehen:

```
# vim:syntax=apparmor
# Last Modified: Sat Aug 12 12:18:34 2006
#include <tunables/global>

/etc/cron.daily/suse.de-clean-tmp {
  #include <abstractions/base>
  #include <abstractions/bash>
  #include <abstractions/nameservice>

  / r,
  /bin/bash ix,
  /bin/pwd ixr,
  /bin/rm ixr,
  /etc/cron.daily/suse.de-clean-tmp r,
  /etc/sysconfig/cron r,
  /tmp r,
  /tmp/** wr,
  /usr/bin/find ixr,
  /usr/bin/safe-rm ixr,
}
```

Die Abstraction `abstractions/nameservice` wird benötigt, da das Script es erlaubt, die Dateien bestimmter Benutzer nicht zu löschen. Hierfür benötigt das Script Namensauflösung. So ist es mit AppArmor auch möglich, Scripts zu überwachen. Speziell bei Scripts aus eigener Entwicklung sollten Sie über den Einsatz von AppArmor nachdenken.

9.6 Modifikation eines Profils ohne Neustart der überwachten Applikation

Wenn Sie ein neues Profil für eine Applikation erzeugen, die bisher noch nicht überwacht wurde, ist es erforderlich, dass Sie nach dem Laden des Profils mit `enforce` oder `rcapparmor reload` auch die Applikation neu starten. Applikationen, die bereits gestartet wurden, bevor die entsprechenden Profile geladen wurden, werden nicht überwacht.

Etwas anders ist der Fall gelagert, wenn die Applikation bereits von AppArmor überwacht wird und Sie nur das Profil ändern möchten, während die Applikation weiterläuft. Dies ist möglich. Editieren Sie einfach das Profil, und rufen Sie anschließend `rcapparmor reload` (lädt alle Profile neu) oder `apparmor_parser <profil>` auf (lädt nur das eine Profil neu). Anschließend sind die Profiländerungen sofort aktiv! Ein Neustart der Applikation ist nicht erforderlich. Dies wirkt sich auch auf bereits geöffnete Dateien aus. Wenn der Zugriff vorher erlaubt war, aber nach der Änderung der Zugriff nicht mehr erlaubt ist, lehnt AppArmor den Zugriff ab, auch wenn die Datei nicht neu geöffnet wird.

Das betrifft auch Subprofile (*Hats*). Verfügt ein Profil über ein Subprofil, das vor dem Reload modifiziert wurde, wendet AppArmor auch die neuen Subprofile an.

Natürlich kann es bei einigen Applikationen zu Problemen kommen. Nicht jede Applikation reagiert gutmütig auf Änderungen der Profile. Teilweise stürzt die Applikation ab, wenn vormals offene Dateien plötzlich nicht mehr erreichbar sind, oder versucht einen zweiten Zugriff nicht, wenn dieser nach der Änderung vielleicht erlaubt ist. Dann können Sie die Applikation nur durch einen Neustart wieder aktivieren.

10 Kritische Betrachtung von AppArmor

10.1 Geschwindigkeit

Die Geschwindigkeit von Mandatory-Access-Control-Systemen bei realen Anwendungen zu messen ist nicht sehr einfach. Natürlich lassen sich die Operationen wie Dateiöffnen und -schließen in ihrer Geschwindigkeit messen, jedoch lassen sich hiervon nur sehr geringe Aussagen für spätere echte Anwendungen ablesen. Die Ergebnisse eines entsprechenden Benchmarks finden Sie in Abschnitt 3.1.

Hier wähle ich daher einen anderen Ansatz, der aber auch sicherlich nicht für alle Anwendungen geeignet ist. Die bereits vorgestellte Web-Applikation *phpSysInfo* soll hier genutzt werden. Für eine Analyse der Geschwindigkeit wird diese Applikation mehrfach (1000-mal) ohne und mit AppArmor-Überwachung über ein 100- Mbit/s-Netzwerk aufgerufen. Das getestete System war ein *SUSE* 10.1- System mit sämtlichen Updates auf einem Intel Pentium 4-Rechner mit 3 GHz und 512 Mbyte Arbeitsspeicher. Die Zugriffe erfolgen mit dem Apache-Benchmark-Werkzeug `ab`.

Lauf	Dauer
Ohne AppArmor	238.4s, 239.2, 239.5
Mit AppArmor	242.4s, 244.3, 241.4s

Es lässt sich leicht feststellen, dass die zusätzlichen Tests, die AppArmor durchführt, bei dieser Applikation nicht wesentlich ins Gewicht fallen. Der zusätzliche Overhead liegt bei 1,6%. Dies ist sicherlich noch applikationsabhängig und kann bei anderen Applikationen sowohl nach oben als auch nach unten abweichen. Hier zeigt sich, wie bei vielen anderen Applikationen auch, dass andere Kriterien die Ausführungsgeschwindigkeit beschränken (Netzwerkbandbreite etc.). Der zusätzliche Aufwand für AppArmor ist meist zu vernachlässigen.

10.2 Sicherheit

Wenn Sie bereits bis hierhin gelesen haben, kennen Sie AppArmor bereits recht gut. Falls Sie direkt hierhin geblättert haben, werden Sie vielleicht einige Begriffe nachschlagen müssen. Dennoch sollten Sie in der Lage sein, die folgenden Ausführun-

gen zu verstehen. AppArmor erhöht sicherlich die Sicherheit des Systems, auf dem es installiert wurde. Als Mandatory-Access-Control-System (*MAC*) muss es sich aber auch den Vergleich mit anderen ähnlichen Lösungen gefallen lassen. Alternative Lösungen sind:

- SELinux
- *LIDS* (*http://www.lids.org*)
- *grsecurity* (*http://www.grsecurity.org*)
- *RSBAC* (*http://www.rsbac.org*)
- etc.

Ich werde hier lediglich AppArmor betrachten. Eine Betrachtung von SELinux erfolgt in dem entsprechenden Kapitel.

Bei der Betrachtung von AppArmor fällt als Erstes die leichte Handhabung, die verständliche Syntax der Profil-Dateien und die gute Dokumentation auf. Dies sind alles Pluspunkte für AppArmor.

Aus Sicht der Sicherheit ist es jedoch bei AppArmor ungünstig, dass auf dem System vertrauenswürdige und nicht vertrauenswürdige Programme unterschieden werden müssen. Die vertrauenswürdigen Programme werden von AppArmor nicht überwacht und dürfen auf dem System jede durch die UNIX-Rechte erlaubte Tätigkeit durchführen. Lediglich die nicht vertrauenswürdigen Applikationen werden von AppArmor zusätzlich überwacht.

Ein zweiter Kritikpunkt ist die Tatsache, dass AppArmor als *Zugriffsattribut* den Dateinamen verwendet. Ein Umbenennen oder Kopieren der Datei entzieht eine Datei oder Applikation der Überwachung durch AppArmor. Änderungen der Installationsorte von Applikationen durch den Programmierer führen dazu, dass die Applikation plötzlich nicht mehr überwacht wird. Außerdem können nicht alle Objekte unter Linux über einen Dateinamen angesprochen werden. Netzwerkverbindungen, Prozesse und IPC-Aufrufe können nicht über Dateinamen angesprochen und daher nicht von AppArmor überwacht werden.

AppArmor kann nicht zwischen unterschiedlichen Benutzern unterscheiden. Applikationen, die auf Benutzerdaten zugreifen müssen, benötigen immer den Zugriff auf die Daten sämtlicher Benutzer. Dies löst AppArmor über die Variable `@HOME` und stellt dem Befehl den Zugriff auf alle Heimatverzeichnisse zur Verfügung.

Schließlich bleibt noch die Qualität der von Novell mitgelieferten Profile zu betrachten. Zunächst enttäuscht die geringe Menge der mitgelieferten Profile. Novell unterstützt in der aktuellen Version 10.1 mit allen Updates nur Profile für die folgenden Applikationen:

- `/bin/netstat`
- `/bin/ping`
- `/lib/ld-2.2.so`
- `/sbin/klogd`

- /sbin/syslogd
- /usr/bin/ldd
- /usr/sbin/identd
- /usr/sbin/mdnsd
- /usr/sbin/nscd
- /usr/sbin/ntpd
- /usr/sbin/traceroute

Alle weiteren Profile in dem Verzeichnis /etc/apparmor/profiles/extra gelten als nicht unterstützt. Ein Anwender, der diese Profile nutzen möchte, tut auch gut daran, diese zunächst zu analysieren und zu testen, denn leider enthalten diese Profile teilweise fehlerhafte Pfade oder logische Fehler.

So verweist das Profil etc.cron.daily.tmpwatch auf einen Cronjob tmpwatch den es auf dieser Version nicht gibt. In dem Profil usr.lib.RealPlayer10.realplay wird auf die Applikation /opt/MozillaFirefox/lib/firefox-bin verwiesen, die bei SuSE 10.1 unter /usr/lib/firefox/firefox-bin installiert wurde. Ähnlich hartcodierte Pfade befinden sich in dem Profil des Acrobat Readers. Ein Upgrade der Java-Virtual-Machine macht das Profil unbrauchbar.

Schließlich kann AppArmor den *Informationsfluss* nicht überwachen. Wenn eine Applikation eine Datei schreibt, kann AppArmor nicht einschränken, welche andere Applikation diese Datei wieder lesen darf, da der verwendete Dateiname beliebig ist und daher nicht von AppArmor als Attribut genutzt werden kann. Dies ist speziell problematisch bei Dateien in Verzeichnissen wie /tmp. Hier muss der Administrator den klassischen UNIX-Dateirechten (*DAC*) vertrauen. AppArmor bietet hier keinen zusätzlichen Schutz.

11 Hintergrund

11.1 Geschichte

SELinux ist das Ergebnis einer langen Entwicklung, an der unter der Federführung der *NSA* viele unterschiedliche Unternehmen und Institute mitwirkten. Wie bereits in Teil I erwähnt wurde, entwickelten die beiden Wissenschaftler David Bell und Leonard LaPadula 1973 das nach ihnen benannte Modell eines Multi-Level-Security-Systems. Später waren Bell und LaPadula auch an der Entwicklung des Orange Books[1] beteiligt. Das *Orange Book* definiert sechs verschiedene Klassen: C1, C2, B1, B2, B3 und A1. Die Betriebssysteme in den Klassen C1 und C2 setzen für die Sicherheit der Daten lediglich *DAC*-Systeme ein, während ab der Stufe B1 Mandatory-Access-Systeme verlangt werden.

Die ersten *MAC*-Systeme waren Multi-Level-Security-Systeme und wurden für militärische und geheimdienstliche Zwecke eingesetzt. Es handelte sich häufig um Betriebssysteme, die außerhalb dieser Einsatzbereiche nicht genutzt wurden. Die *MLS*-Systeme waren aber sehr starr und konnten nicht flexibel angepasst werden. Sie hatten nur ein Ziel: Die Vertraulichkeit der Daten musste gewährleistet werden. Flexiblere Betriebssysteme der Stufen B1 und aufwärts wurden benötigt. Forscher der Information Assurance Research Group der NSA entwickelten daraufhin zusammen mit dem Unternehmen *Secure Computing Corporation* (SCC) eine neue flexible Mandatory-Access-Control-Architektur, die auf dem Modell des *Type-Enforcement* basierte. Dieser Mechanismus wurde zuerst für das Betriebssystem Logical Coprocessing Kernel (*LOCK*) entwickelt. LOCK wurde noch von Honeywells Secure Computing Technology Center (SCTC) entwickelt, aus dem später die Firma SCC hervorging. In Zusammenarbeit mit der NSA entstanden zwei Mach-Kernel-basierte Prototypen: *DTMach* und *DTOS*[2]. An der Entwicklung von DTOS war auch die *Flux*-Forschungsgruppe der Universität in Utah beteiligt. Gemeinsam wurden die Techniken in das Betriebssystem *Fluke* implementiert. Dabei wurde die Architektur erweitert und ergänzt, sodass nun die Richtlinien dynamisch geladen werden konnten. Diese neue Architektur erhielt den Namen *Flask*[3]. Fluke war aber immer noch ein OS, das lediglich zu Forschungszwecken eingesetzt wurde. Um die Technologie einem breiteren Publikum zur Verfügung zu stellen und so mehr Erfahrung zu sammeln,

[1] Der eigentliche Titel lautet: Trusted Computer System Evaluation Criteria (TCSEC). Es wird aber allgemein als Orange Book bezeichnet.

[2] *http://www.cs.utah.edu/flux/dtos/*

[3] *http://www.cs.utah.edu/flux/flask/*

implementierte die NSA die Flask-Architektur in Linux. Dabei wurde die NSA von *Network Associates* und *MITRE* unterstützt. Im Dezember 2000 wurde SELinux auf der Basis des Linux-Kernels 2.2 als Open-Source-Software veröffentlicht.

Auf dem Linux Kernel Summit 2001 in Ottawa wurde das Linux-Security- Module-(*LSM*-)Projekt gestartet. Hiermit sollte erstmals eine standardisierte Sicherheitsschnittstelle im Linux-Kernel geschaffen werden. In den folgenden Jahren wurde SELinux auf die LSM-Schnittstelle portiert und ist seit dem Kernel 2.6 fest im Kernel enthalten.

Eine der ersten Distributionen, die SELinux aufgenommen hat, war *Fedora Core*. Ab Version 2 enthält diese Distribution die SELinux-Erweiterung. Während in der Version 2 die Unterstützung noch fehlerhaft war, konnte in den Versionen 3 bis 5 eine bessere Unterstützung erreicht werden. *Debian* Etch unterstützt ebenfalls SELinux.

Die erste kommerzielle Distribution mit SELinux-Unterstützung ist Red Hat Enterprise Linux. Die Version 4 unterstützt SELinux mit einer Targeted Policy (siehe Abschnitt 17.1).

Inzwischen wurde die Architektur auch in BSD[4] und Darwin[5] implementiert.

11.2 Architektur

In diesem Kapitel stelle ich Ihnen die Architektur des SELinux-Frameworks vor. Wahrscheinlich werden Sie nach dem ersten Lesen dieses Kapitels noch einige Fragen haben. Bitte lesen Sie dieses Kapitel erneut, wenn Sie sich ein wenig mit SELinux beschäftigt haben. Dann wird Ihnen einiges klarer werden.

SELinux basiert auf dem *Flask*-Kernel. Flask ist der Flux-Advanced-Security-Kernel. *Flux* ist eine Forschungsgruppe an der Universität von Utah, die sich allgemein mit Software-Systemen beschäftigt[6].

Die Flask-Architektur ist zuerst für einen *Microkernel* entwickelt worden und wurde von dem *Fluke*-Microkernel-Betriebssystem abgeleitet. Die Flask-Architektur beschreibt die Interaktionen zwischen den beiden verschiedenen Subsystemen, die die Entscheidungen treffen und umsetzen. Dies sind in der Flask-Architektur zwei voneinander getrennte Subsysteme. Während die Umsetzung von dem *Object-Manager* durchgeführt wird, wird die Entscheidung im *Security-Server* getroffen (siehe Abbildung 11.1).

Der Object-Manager wird bei den zu überwachenden Objekten implementiert. Dieser Object-Manager fängt jede Anfrage auf das Objekt ab und stellt diese an den Security-Server. Dieser prüft auf beliebige Weise (bei SELinux anhand der SELinux-Richtlinie), ob der Zugriff erlaubt ist, und liefert die Entscheidung an den Object-

[4] SEBSD: *http://www.trustedbsd.org/sebsd.html*
[5] SEDarwin: *http://www.trustedbsd.org/sedarwin.html*
[6] *http://www.cs.utah.edu/flux/*

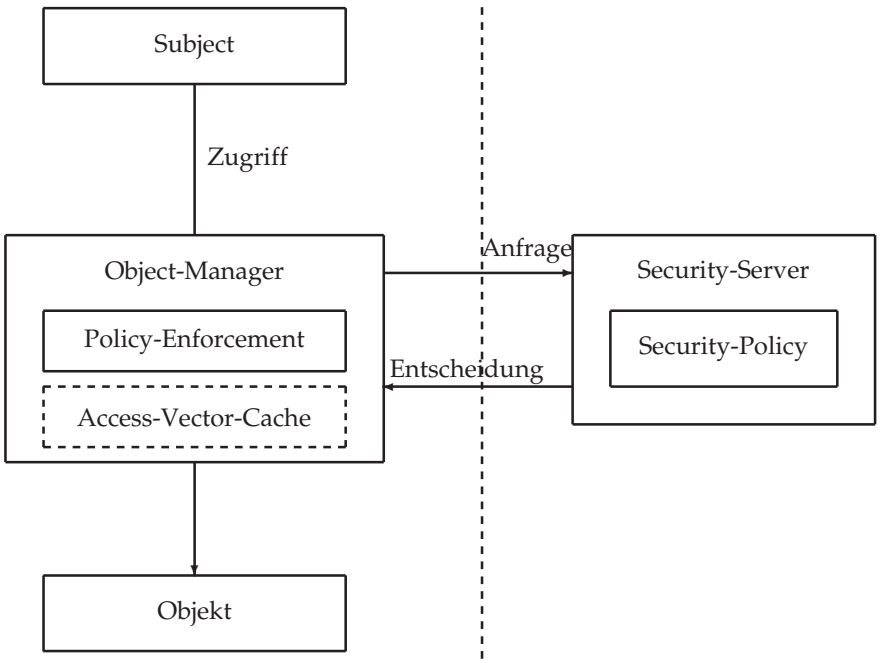

Abbildung 11.1: Flask stellt eine Trennung zwischen Policy Enforcement und Decision dar.

Manager zurück. Dieser setzt dann die Entscheidung um und gibt den Zugriff frei oder verweigert den Zugriff.

Der Object-Manager hat drei Aufgaben in der Flask-Architektur:

- Kommunikation mit dem Security-Server. Der Object-Manager muss mit dem Security-Server kommunizieren, um die Zugriffsanfragen weiterzuleiten und Labeling-Informationen zu erhalten.
- Speicherung der Entscheidungen in einem *Access-Vector-Cache* (AVC). Damit nicht jeder Zugriff erneut an den Security-Server weitergegeben werden muss, verfügt der Object-Manager über einen Access-Vector-Cache, in dem die Entscheidungen gespeichert werden.
- Registration bei dem Security-Server. Der Security-Server muss den Object-Manager benachrichtigen können, sobald sich die Policy ändert. Der Object-Manager muss dann die gespeicherten Entscheidungen im AVC löschen.

11.2.1 Access-Vector-Cache

Am einfachsten wäre es sicherlich, wenn der Object-Manager bei jedem Zugriff den Security-Server fragen würde, ob der Zugriff erlaubt ist. Dies wäre jedoch sehr lang-

sam und daher nicht sinnvoll. Die Flask-Architektur erlaubt daher das Caching der Entscheidungen in dem Object-Manager.

Das AVC-Modul wird von allen Object-Managern gemeinsam genutzt. So kommunizieren die Object-Manager nicht direkt mit dem Security-Server, sondern über den AVC.

Muss nun der Zugriff eines Subjekts auf ein Objekt evaluiert werden, sendet der Object-Manager einen Access-Vector, der aus dem Subject-*Security-Context*, dem Object-Security-Context und dem Zugriff besteht, an den AVC. Dieser prüft, ob die entsprechende Entscheidung bereits im AVC gespeichert ist. Ist das nicht der Fall, wird der Access-Vector an den Security-Server weitergesendet. Dieser prüft und entscheidet über den Zugriff und sendet die Entscheidung zurück an den AVC. Aus Geschwindigkeitsgründen kann der Security-Server mehrere (auch zusätzliche nicht angefragte) Entscheidungen gleichzeitig an den AVC senden. Dieser wird sie für zukünftige Zugriffe speichern. So wird zum Beispiel immer ein kompletter Access-Vector zurückgeliefert. Der Access-Vector enthält dann für das betroffene Objekt alle Zugriffe, die das Subjekt durchführen darf.

Hier erklärt sich dann auch, warum die Protokollmeldungen von SELinux nicht SELinux, sondern AVC im Namen tragen. Das Subsystem, das die Meldung auslöst, ist der AVC.

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
```

```
system_u:system_r:crond_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-Rechtesystem (*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*.

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 verschlüsselten[3] 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 Ressource. 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 dom l2 )`. Die Platzhalter `l1` und `l2` stehen für den MLS-Wert des Subjekts und des Objekts. Von *Dominanz* (`l1 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 werden[5] 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.

13 SELinux-Anwendung

In diesem Kapitel werden wir die ersten Schritte der Installation und Inbetriebnahme von SELinux unter verschiedenen Distributionen betrachten. Anschließend stelle ich Ihnen die wichtigsten Befehle vor, die Sie als Administrator benötigen, um Ihr SELinux-System zu verwalten und zu benutzen. Im nächsten Kapitel besprechen wir dann erste Anpassungen des Systems über boolesche Variablen, während Sie im übernächsten Kapitel kleine Anpassungen an der Richtlinie selbst vornehmen werden.

13.1 Distributionen

SELinux ist keine eigene Distribution. Daher benötigen Sie zunächst immer eine Linux-Distribution, die Sie anschließend um SELinux erweitern. Am besten werden aktuell die *Fedora Core*-Distributionen unterstützt. Die in diesem Buch besprochene SELinux-Referenz- Richtlinie kommt seit Fedora Core 5 zum Einsatz. Alternativ können Sie aber auch die *Debian*-Distribution Etch, *Ubuntu*, *Gentoo* oder *Slackware* einsetzen. Falls Sie eine andere Distribution wünschen, müssen Sie die notwendigen Anpassungen selbst vornehmen. Unterschätzen Sie den Aufwand nicht. Eine Vielzahl von Programmen müssen modifiziert werden. Es genügt nicht, einige wenige Bibliotheken und Befehle zusätzlich zu installieren.

Achtung

Dies trifft auch und besonders auf die *SUSE*-Distribution ab Version 10.1 zu, bei der die Unterstützung für SELinux wieder aus dem Kernel und aus den Applikationen entfernt wurde.

13.1.1 Fedora Core

Fedora Core kommt direkt mit SELinux-Unterstützung. Ab Fedora Core 5 ist die SELinux *Reference-Policy* im Einsatz (siehe Abschnitt 13.2).

13.1.2 Debian und Ubuntu

Debian hat für die aktuelle Version Etch (4.0) die Unterstützung für SELinux aufgenommen. Dies beinhaltet dann auch die Unterstützung für die SELinux *Reference-Policy* (siehe Abschnitt 13.2). Die Unterstützung für *Ubuntu* wird auf *https://wiki.ubuntu.com/SELinux* vorangetrieben und dokumentiert.

13.1.3 Gentoo

Das *Gentoo*-Projekt hat unter *http://www.gentoo.org/proj/en/hardened/selinux/selinux-handbook.xml* eine hervoragende Dokumentation darüber, wie eine Gentoo-Installation um SELinux erweitert wird. Eine weitere Erläuterung hier ist daher unnötig. Die Integration der SELinux Reference-Policy wird gerade vorbereitet und ist bei Veröffentlichung des Buches möglicherweise schon abgeschlossen.

13.1.4 Slackware

Die Unterstützung für *Slackware* ist nicht weit fortgeschritten. Es existieren jedoch einige Pakete mit den für Slackware notwendigen angepassten Dateien:

- *http://projects.dimensionalstorm.net/selinux/*
- *ftp://ftp.diyab.net/selinux/*

Diese Pakete sind aber durchaus schon älter, und ihr Betrieb ist daher problematisch.

13.2 Welche SELinux-Policy?

Dieses Buch betrachtet in erster Linie die SELinux-*Reference-Policy*, die auf *http://serefpolicy.sf.net* gepflegt wurde und nun unter *http://oss.tresys.com* zu finden ist. Ältere Distributionen (zum Beispiel Fedora Core 3, 4 und Red Hat Enterprise Linux 4) verwenden häufig auch noch die SELinux *Example-Policy*, die auf *http://selinux.sf.net* gepflegt wird. Sollten Sie eine derartige Distribution einsetzen wollen, sollten Sie auch den Teil V lesen. Hier wird auf die Unterschiede eingegangen.

Die wesentlichen Ziele, die zur Entwicklung der neuen Reference-Policy geführt haben, waren:

- einen Quelltext für die Erzeugung aller Policy-Varianten: Targeted, Strict und MLS
- integrierte Dokumentation innerhalb der Policy
- Modularität und Kapselung der einzelnen Bestandteile
- vereinfachte Verwaltung und Übersetzung
- integrierte Unterstützung von *MLS*

Obwohl die Modularität die tägliche Arbeit mit der Policy vereinfacht, kann dieses Ziel erst richtig verstanden werden, wenn auch der SELinux Policy Management Ser-

ver (siehe Kapitel 32) eingesetzt wird. Dieses Projekt befindet sich noch in der Entwicklung und wird die zentrale Administration der Policy durch unterschiedliche Benutzer erlauben.

Sowohl beim Einsatz der Example-Policy als auch bei der Reference- Policy bieten viele Distributionen sowohl eine Targeted als auch eine Strict-Policy als Variante an. Die meisten einführenden Beispiele basieren auf der Targeted-Variante. Diese Variante ist bei den meisten Distributionen der Default und erlaubt die verständlichere Darstellung der Beispiele. Die Eigenschaften der Targeted-Policy werden in Kapitel 17.1 besprochen. Viele Befehle sind jedoch nur bei Einsatz der Strict-Variante sinnvoll (z.B. `newrole`). Deren Unterschiede werden in Kapitel 17.2 besprochen.

13.3 Erste Schritte und SELinux-Befehle

Sobald Sie ein funktionstüchtiges Linux-System mit aktiviertem SELinux besitzen, sollten Sie sich als *root* anmelden[1].

Tipp

Wenn Sie nicht über ein entsprechendes System verfügen, befindet sich auf der CD ein Fedora Core 6-System mit installierter SELinux-Reference-Policy als VMWare-Image. Dieses können Sie in einer VMWare-Workstation oder mit dem beigelegten VMWare-Player betreiben. Die Anmeldung ist als Benutzer *root* mit dem Kennwort *kennwort* möglich.

Zunächst sieht direkt nach der Anmeldung alles normal aus. Möglicherweise erhalten Sie aber auch direkt Protokollmeldungen von SELinux auf der Konsole (siehe Abbildung 13.1).

Ihre erste Frage sollte nun sein: Wer bin ich? Dies beantwortet der Befehl `id`. Mit der Option `-Z` erhalten Sie nur die SELinux-Identität:

```
[root@supergrobi ~]# id
uid=0(root) gid=0(root) Gruppen=0(root),1(bin),2(daemon), ➥
         3(sys),4(adm),6(disk),10(wheel) context=root:staff_r: ➥
         staff_t:SystemLow-SystemHigh
```

Ihr Benutzer verfügt nun über die Rolle *staff_r*. Diese Rolle ist Personen vorbehalten, die administrative Tätigkeiten wahrnehmen. In dieser Rolle sind diese administrativen Tätigkeiten diesen Personen aber nicht gestattet. Das können Sie sogar recht leicht erkennen.

[1] Möglicherweise funktioniert die grafische Anmeldung nicht. Wechseln Sie dann mit STRG+ALT+F1-6 auf eine Konsole, und melden Sie sich dort an.

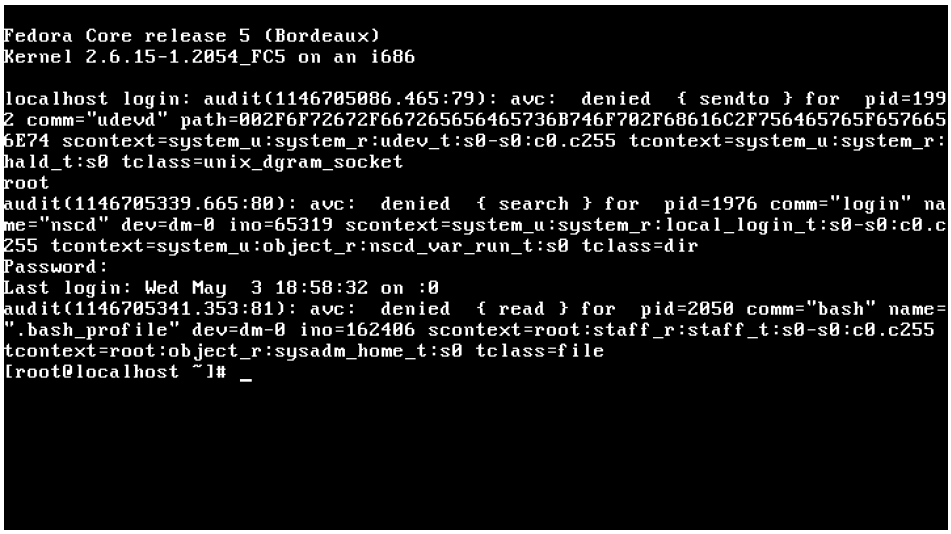

Abbildung 13.1: Bei der Anmeldung protokolliert SELinux möglicherweise direkt Verletzungen der Richtlinie.

Hierfür müssen Sie sich aber zunächst weitere Informationen über die SELinux-Konfiguration beschaffen. Der Befehl `getenforce` teilt Ihnen den aktuellen Zustand des SELinux-Systems mit:

```
[root@supergrobi ~]# getenforce
Permissive
```

Mögliche Rückgabewerte des Befehls sind:

- `Disabled`: SELinux ist abgeschaltet.
- `Permissive`: SELinux ist angeschaltet und wertet die Policy aus. Verletzungen werden aber nur protokolliert und nicht verhindert. Jede Aktion ist erlaubt, als ob SELinux abgeschaltet wäre.
- `Enforcing`: SELinux ist angeschaltet und erzwingt die Einhaltung der Policy.

Der Befehl `sestatus` liefert Ihnen die gleichen Informationen ein wenig ausführlicher:

```
[root@supergrobi ~]# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   permissive
Mode from config file:          permissive
Policy version:                 20
Policy from config file:        strict
```

Hier erkennen Sie, dass SELinux angeschaltet ist und wo das SELinux-Dateisystem gemountet wird. Wir werden das SELinux-Dateisystem später noch kennenlernen. Die hier vorgestellten Befehle kommunizieren mit SELinux im Kernel über dieses Dateisystem. Es handelt sich um ein virtuelles Dateisystem ähnlich `/proc` und `/sys`. Sie erkennen den aktuellen Modus (`Permissive`) und den in der Konfigurationsdatei hinterlegten Modus. Die angegebene Version bezeichnet die SELinux-Version im Kernel und ist ein Anhaltspunkt für die verfügbaren Sprachelemente. Schließlich erkennen Sie noch, dass auf meinem Rechner die *Strict-Policy* eingesetzt wird.

Die hier angesprochene Konfigurationsdatei ist `/etc/selinux/config`. Diese Datei definiert den SELinux-Modus:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - SELinux is fully disabled.
SELINUX=permissive
# SELINUXTYPE= type of policy in use. Possible values are:
#       targeted - Only targeted network daemons are protected.
#       strict - Full SELinux protection.
SELINUXTYPE=strict

# SETLOCALDEFS= Check local definition changes
SETLOCALDEFS=0
```

Diese Datei wird bei dem Boot des Systems ausgewertet. Nach einer Änderung in der Datei genügt daher ein Reboot. Wenn Sie jedoch die Policy ändern möchten (`strict` oder `targeted`), genügt kein einfacher Reboot, da diese Policys unterschiedliche Security-Contexts verwenden. Zusätzlich ist ein *Relabeling* des Dateisystems erforderlich[2]. Wir werden diesen Punkt noch ansprechen.

Kommen wir zurück zu unserem Ausgangspunkt. Der Benutzer *root* verfügt über die Rolle *staff_r*, die keine administrativen Arbeiten erlaubt. Testen Sie das zum Beispiel mit dem Befehl `ps -ef`. Dieser Befehl zeigt sämtliche Prozesse auf dem System an. Auf dem hier vorgestellten System funktioniert das zunächst, da es sich im `Permissive`-Mode befindet:

```
[root@supergrobi ~]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 12:37 ?        00:00:01 init [5]
root         2     1  0 12:37 ?        00:00:00 [migration/0]
root         3     1  0 12:37 ?        00:00:00 [ksoftirqd/0]
root         4     1  0 12:37 ?        00:00:00 [watchdog/0]
root         5     1  0 12:37 ?        00:00:00 [migration/1]
```

[2] Üblicherweise genügt es, eine Datei `/.autorelabel` vor dem Reboot anzulegen.

```
root         6     1  0 12:37 ?        00:00:00 [ksoftirqd/1]
...
root      4122  2239  0 18:23 tty1     00:00:00 -bash
root      4297  4092  0 18:57 pts/2    00:00:00 ps -ef
```

Wechseln Sie nun auf dem System in den `Enforcing`-Mode. Hierzu müssen Sie nicht rebooten. Es genügt, den Befehl `setenforce` zu verwenden:

```
[root@supergrobi ~]# setenforce 1
[root@supergrobi ~]# getenforce
Enforcing
[root@supergrobi ~]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root      4092  4089  0 18:21 pts/2    00:00:00 -bash
root      4122  2239  0 18:23 tty1     00:00:00 -bash
root      4305  4092  0 18:58 pts/2    00:00:00 ps -ef
```

Nun sehen Sie nur noch eine kleine Auswahl der laufenden Prozesse. SELinux verhindert den Zugriff auf alle Prozesse, die nicht von Ihnen gestartet wurden! Dies gilt auch für andere Prozesse, die den Benutzer *root* verwenden. Versuchen Sie, auf die Heimatverzeichnisse anderer Benutzer oder auf das `proc`-Dateisystem zuzugreifen:

```
[root@supergrobi ~]# ls -l /home/
insgesamt 8
?--------- ? ?    ?       ?            ? /home/lost+found
?--------- ? ?    ?       ?            ? /home/student
[root@supergrobi ~]# ls -l /proc/kcore
ls: /proc/kcore: Keine Berechtigung
```

Nebenbei: In den `Permissive`-Modus wechselt SELinux ebenfalls mit dem Befehl `setenforce`:

```
[root@supergrobi ~]# setenforce 0
setenforce:  setenforce() failed
```

Leider darf dies nur ein Benutzer mit der Rolle *sysadm_r*.

Sie sollten bis jetzt bereits einen Eindruck davon gewonnen haben, wie SELinux die Sicherheit eines Linux-System gewährleisten kann. Natürlich ist dafür eine fehlerfreie und umfangreiche Policy erforderlich. Diese wird jedoch bei den meisten Distributionen bereits mitgeliefert. Auch setzt dieses Beispiel die *Strict-Policy* voraus. Wir werden in diesem Buch uns in erster Linie mit der *Targeted-Policy* beschäftigen, da diese häufiger zum Einsatz kommt.

Wie wird nun das System administriert? Der Benutzer *root* hat auch Zugriff auf die Rolle *sysadm_r*, die über die notwendigen Privilegien verfügt. Um in die Rolle zu wechseln, verwendet er den Befehl `newrole`:

```
[root@supergrobi ~]# newrole -r sysadm_r
Authentifiziere root.
Passwort:
[root@supergrobi ~]# ls -l /proc/kcore
-r-------- 1 root root 527896576 23. Aug 19:06 /proc/kcore
```

Anschließend kann er auch wieder auf alle Dateien zugreifen.

Weitere wichtige Befehle auf der Kommandozeile sind:

- `chcon`: Hiermit können Sie den Kontext einer Datei ändern (siehe Abschnitt 18.6).
- `restorecon`: Dieser Befehl stellt den Security-Context einer Datei entsprechend der Policy wieder her (siehe Abschnitt 18.16).
- `selinuxenabled`: Dieser Befehl kann in Scripts eingesetzt werden, um den Status von SELinux zu prüfen (siehe Abschnitt 18.24).
- `getsebool/setsebool`: Hiermit können Sie boolesche Variablen lesen und setzen (siehe Abschnitt 15 und Abschnitt 18.34).
- `audit2allow/audit2why`: Diese Befehle erlauben die Analyse der SELinux-Protokollmeldungen und die Anpassung der Richtlinie (siehe Kapitel 16 und Abschnitt 18.3).

Dies soll als erste Einführung in SELinux genügen. Zunächst werden Sie das System genauso verwenden können wie bisher. Jedoch werden einige Funktionen nicht mehr zur Verfügung stehen. Dann müssen Sie die Protokollmeldungen analysieren (siehe Kapitel 14) und können die Richtlinie über boolesche Variablen (siehe Kapitel 15) oder über die Erweiterung der Richtlinien anpassen (siehe Kapitel 16). Eine typische Zusammenfassung der täglichen Aufgaben und ihre Lösung ist in Kapitel 19 zu finden. Wenn Sie aber tiefer einsteigen möchten und vielleicht sogar für komplett neue Dienste die Richtlinien entwickeln möchten, müssen Sie sich mehr mit der Sprache und ihren Bestandteilen beschäftigen. Dann sollten Sie Teil IV durcharbeiten.

14 SELinux-Protokollmeldungen

Eine der wichtigsten Informationsquellen im Zusammenhang mit SELinux sind die Protokolle. Hier erfährt der Administrator, wo SELinux den Zugriff unterbunden hat. Deswegen beschäftigen wir uns zunächst mit diesen Protokollen.

SELinux erzeugt bei fast allen Regelverletzungen Protokollmeldungen. Diese Meldungen werden von dem Audit-Subsystem des Kernels erzeugt und von dem *Syslogd*-Daemon normalerweise in der Datei `/var/log/messages` gespeichert:

```
Aug 24 10:35:23 supergrobi kernel: audit(1156408523.227:485): avc: ➥
         denied  write  for  pid=8430 comm="sshd" name="wtmp" ➥
         dev=dm-3 ino=6311880 scontext=system_u:system_r:sshd_t: ➥
         s0-s0:c0.c255 tcontext=system_u:object_r:var_log_t:s0 ➥
         tclass=file
```

Sobald jedoch der *Auditd*-Daemon auf Ihrem System aktiv ist, übernimmt dieser anstelle des Syslogd-Daemons die Meldungen und schreibt sie in die Datei `/var/log/audit/audit.log`. Der Auditd-Daemon und das Kernel-Audit-Subsystem werden genauer im Anhang besprochen (siehe Anhang A). Die gleiche Meldung im Audit-Log sieht folgendermaßen aus:

```
type=AVC msg=audit(1156408523.227:485): avc:  denied   write  ➥
         for  pid=8430 comm="sshd" name="wtmp" dev=dm-3 ino=6311880 ➥
         scontext=system_u:system_r:sshd_t:s0-s0:c0.c255 tcontext= ➥
         system_u:object_r:var_log_t:s0 tclass=file
```

Beide Meldungen sind ab dem Schlüsselwort `audit` identisch. Dass es sich um eine SELinux-Protokollmeldung handelt, erkennen Sie an dem Schlüsselwort `avc`. Diese Abkürzung steht für den *Access-Vector-Cache* (siehe auch Kapitel 11.2). Der Access-Vector-Cache wertet die Richtlinien aus und gewährt oder verweigert den Zugriff.

Protokollmeldungen können bei zwei Ereignissen erzeugt werden:

- Der Zugriff wird verweigert. Dann wird immer eine `denied`-Meldung erzeugt. In Ausnahmefällen kann die Meldung unterdrückt werden.
- Der Zugriff wird erlaubt. Dann wird normalerweise keine Meldung erzeugt. Es gibt jedoch Regeln, die eine Meldung speziell erzwingen. Diese erzeugen eine `granted`-Meldung.

Die `granted`-Meldungen sind eher selten und werden nur bei besonderen Ereignissen ausgelöst. Ein typisches Ereignis ist ein Neuladen der Policy:

```
type=AVC msg=audit(1156414643.446:697): avc:  granted ⏎
         { load_policy } for  pid=8972 comm="load_policy" ⏎
         scontext=root:sysadm_r:load_policy_t:s0-s0:c0.c255 ⏎
         tcontext=system_u:object_r:security_t:s0 tclass=security
```

Wie müssen Sie die Meldungen nun lesen?

```
type=AVC msg=audit(1156408523.227:485): avc❶:  denied❷ ⏎
         { write }❸ for  pid=8430❹ comm="sshd"❺ ⏎
         name="wtmp"❻ dev=dm-3❼ ino=6311880❽ scontext= ⏎
         system_u:system_r:sshd_t:s0-s0:c0.c255❾ tcontext= ⏎
         system_u:object_r:var_log_t:s0❿tclass=file⓫
```

Jede Meldung beginnt, wie bereits erwähnt, mit dem Schlüsselwort `avc` ❶. Anschließend kommt entweder `granted` oder `denied` ❷, gefolgt von dem Zugriff in geschweiften Klammern ❸. Hier handelt es sich um einen Schreibzugriff (`write`). Damit Sie prüfen können, welcher Prozess den Zugriff durchgeführt hat, folgen die Prozess-ID des Prozesses ❹ und der Name des Kommandos ❺. Anschließend finden Sie den Namen des Objekts ❻, auf das zugegriffen wurde, und bei Dateien, Verzeichnissen etc. den Namen des Geräts ❼, auf dem sich das Objekt befindet, und seine *Inode*-Nummer ❽. In allen Meldungen finden Sie dann wieder den *Security-Context* der Quelle (`s(ource)context`, ❾) und den Security-Context des Ziels (`t(arget)context`, ❿, gefolgt von der Objektklasse des Ziels (`t(arget)class`,⓫.

In diesem Fall hat also der *Secure-Shell*-Daemon (`sshd`) versucht, die Datei (`tclass=file`) `wtmp` auf dem Gerät `dm-3`[1] zu schreiben. Dabei verwendet der Secure-Shell-Daemon den Typ *sshd_t*, und die Datei hat den Typ *var_log_t*. Dieser Zugriff wird von den Regeln nicht erlaubt und wird daher abgelehnt. Eine entsprechende Regel, die diesen Zugriff erlauben würde, ist:

```
allow sshd_t var_log_t:file { write };
```

Für die Analyse der Protokollmeldungen stehen auch einige Befehle zur Verfügung, mit denen Sie die Analyse einfacher durchführen können. Der Befehl `audit2allow` (siehe auch Abschnitt 18.3) ermöglicht es Ihnen, direkt aus einer Protokollmeldung die entsprechende Allow-Regel zu erzeugen. Sie werden diesen Befehl in Kapitel 16 näher kennenlernen.

Der Befehl `audit2why` versucht, den Grund einer Meldung zu analysieren. Leider ist dieses Werkzeug nicht sehr intelligent. Im Falle unserer Meldung erhalten wir die folgende Ausgabe:

[1] Dies ist ein Logical Volume, das von dem Device-Mapper erzeugt wurde.

```
Aug 24 10:35:23 supergrobi kernel: audit(1156408523.227:485): avc: ➥
         denied   write  for  pid=8430 comm="sshd" name="wtmp" ➥
         dev=dm-3 ino=6311880 scontext=system_u:system_r:sshd_t: ➥
         s0-s0:c0.c255 tcontext=system_u:object_r:var_log_t:s0 ➥
         tclass=file
       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.
```

Schließlich steht auch noch ein grafisches Werkzeug für die Analyse der Protokollmeldungen zur Verfügung. Möglicherweise müssen Sie das Paket `setools-gui` nachinstallieren, um das Programm *seAudit* auf Ihrem System verwenden zu können. Existiert für Ihre Distribution dieses Paket nicht, können Sie die Werkzeuge auch von der Homepage von Tresys herunterladen (`http://oss.tresys.com/projects/setools`) und manuell installieren.

Mit `seaudit` lesen Sie die Protokollmeldungen aus der Datei `/var/log/messages` oder `/var/log/audit/audit.log` ein und stellen sie grafisch dar (siehe Abbildung 14.1). Dieses Werkzeug erlaubt es Ihnen, auch zwischen der Echtzeitüberwachung und der Offline-Analyse hin- und herzuschalten. Über TOGGLE MONITOR schalten Sie die Echtzeitanzeige an und ab. Dies ist insbesondere von Vorteil, da Sie auch die Sicht auf die Ereignisse modifizieren können. Wählen Sie MODIFY VIEW und Sie erhalten einen neuen Dialog (siehe Abbildung 14.2), in dem Sie der Sicht zunächst einen Namen geben können und Filter definieren können. Durch diese Filter erhalten Sie einen wesentlich besseren Überblick über die anfallenden Meldungen (siehe Abbildung 14.3). Sie können beliebig viele Filter definieren. Diese können Sie verwenden, um Meldungen zu verstecken (HIDE) oder anzuzeigen (SHOW). Dabei kön-

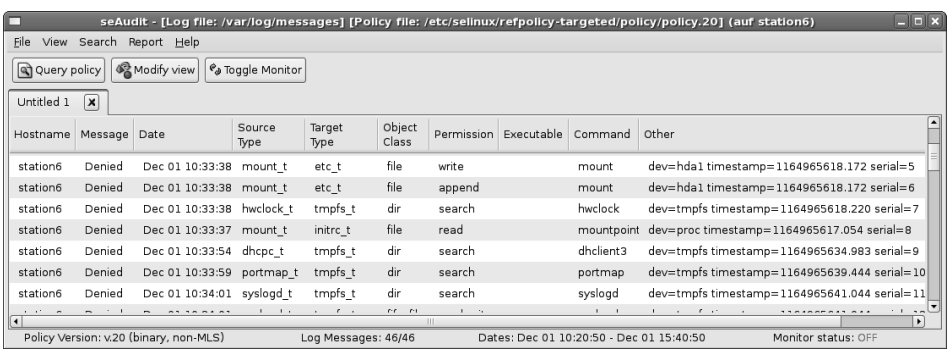

Abbildung 14.1: seAudit stellt die Logmeldungen grafisch dar. Für die Echtzeitüberwachung können Sie den Monitor-Modus aktivieren.

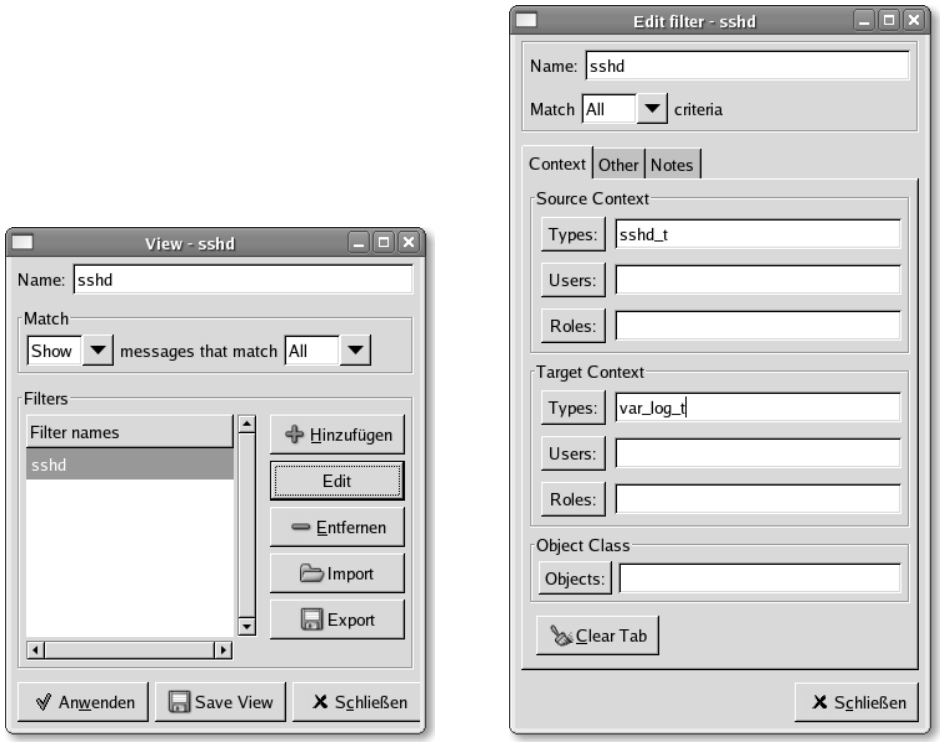

Abbildung 14.2: Die Sichten in Seaudit können mächtig editiert und mit Filtern versehen werden.

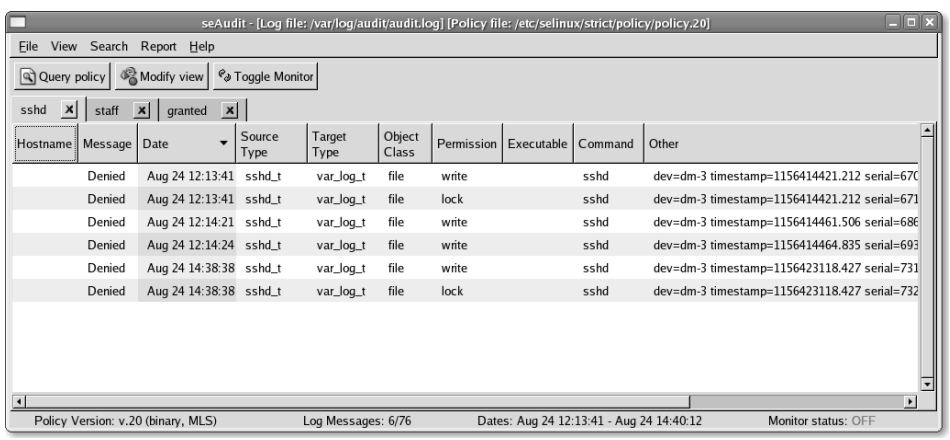

Abbildung 14.3: Mit den Sichten behalten Sie einen guten Überblick über die anfallenden Meldungen.

nen Sie die Filter Und- (ALL) und Oder-verknüpfen (ANY). Auch die Filter können mit Namen versehen werden (siehe Abbildung 14.2). Bei den Filtern können Sie den *Security-Context* des Subjekts und Objekts und seine Objektklasse spezifizieren. Zusätzlich können Sie auf weiteren Reitern IP-Adressen, Ports und Netzwerkschnittstellen, Rechnernamen, Kommandonamen und Pfade angeben. Auf der Registerkarte OTHER können Sie auch zwischen `Denied` und `granted`-Meldungen unterscheiden. Natürlich können Sie die fertigen Sichten auch abspeichern und die Filter auf der Registerkarte NOTES kommentieren.

Haben Sie eine Meldung gefunden, die Sie näher interessiert, können Sie direkt aus dem Programm seAudit die Policy laden und die entsprechenden Regeln anzeigen. Hierzu wählen Sie QUERY POLICY und spezifizieren die anzuzeigenden Regeln (siehe Abbildung 14.4). Hierfür benötigt seAudit lediglich die vorhandene binäre Policy.

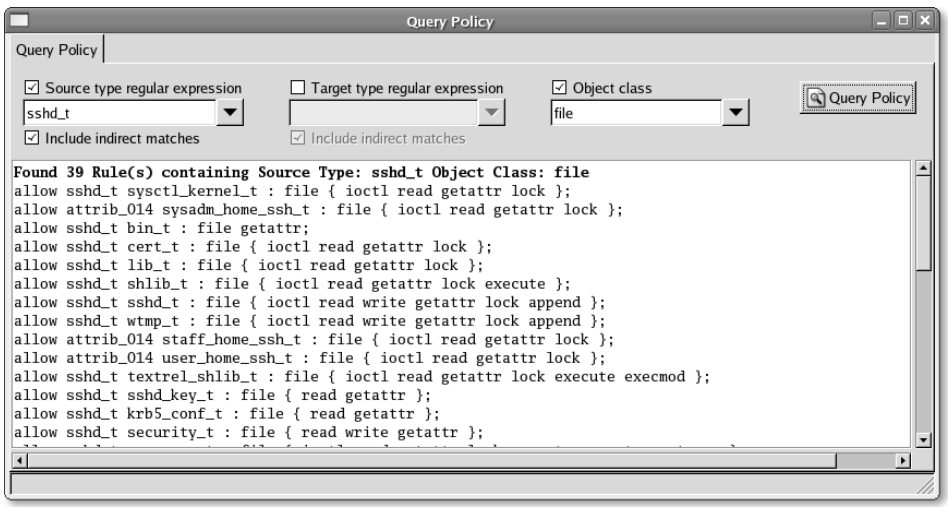

Abbildung 14.4: Mit Seaudit können Sie direkt auch die entsprechenden Regeln der Policy anzeigen.

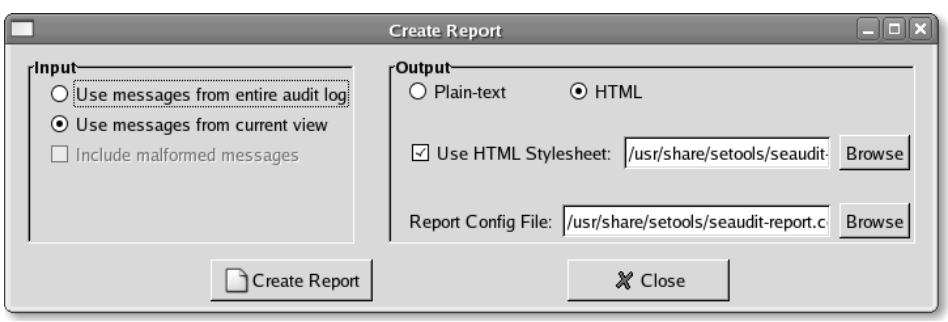

Abbildung 14.5: Seaudit erlaubt auch die direkte Erzeugung von Berichten.

Möchten Sie nicht die Standard-Policy laden, können Sie über das FILE-Menü natürlich auch eine andere Policy angeben.

Schließlich können Sie auch direkt über `seaudit` den Kommandozeilenbefehl `seaudit-report` aufrufen (siehe Abschnitt 18.20). Hiermit erstellen Sie einen Bericht, der die Meldungen zusammenfasst und sogar in HTML mit einem Stylesheet ansprechend formatieren kann (siehe Abbildung 14.5 und 14.6). Wenn Sie jedoch nur einige

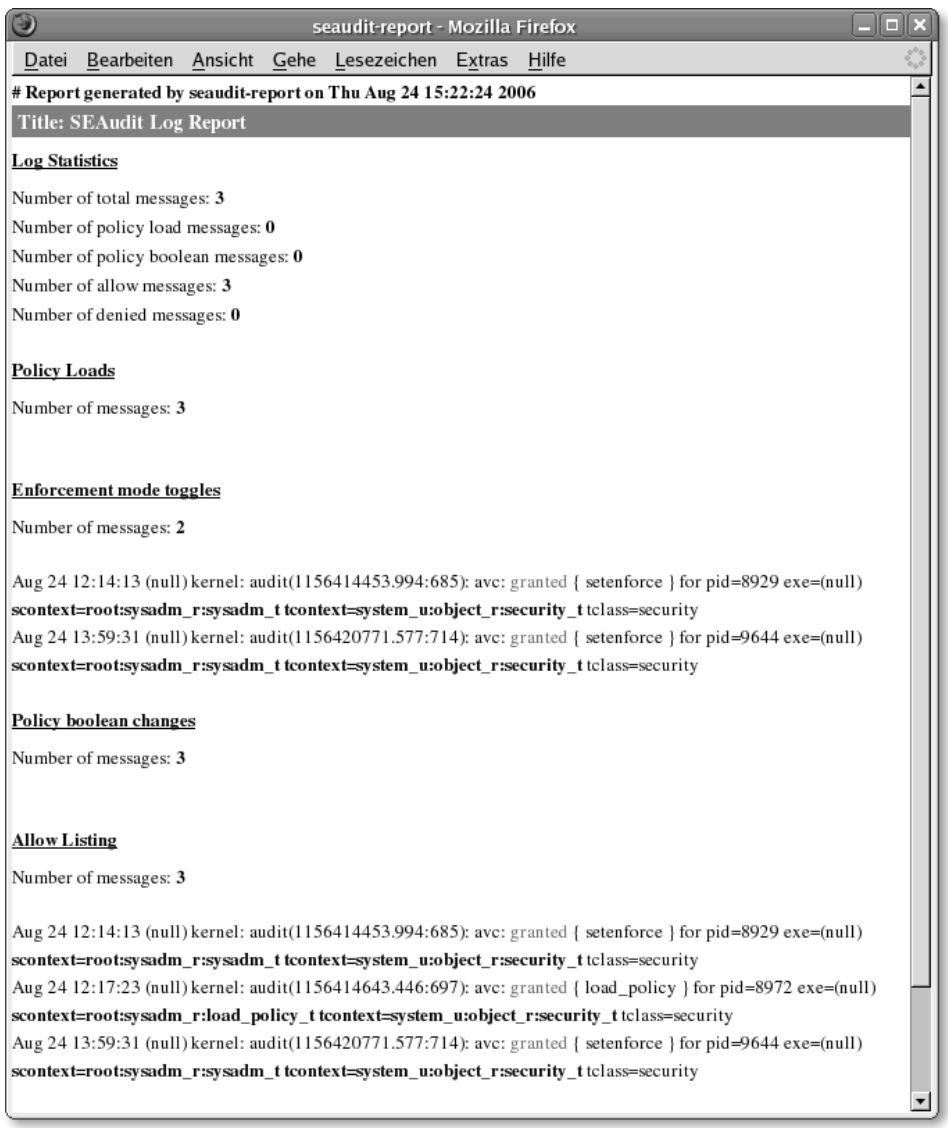

Abbildung 14.6: Die Berichte können in HTML mit Stylesheets formatiert werden.

Meldungen für eine spätere Analyse zum Beispiel mit `audit2allow` exportieren möchten, können Sie das über das Menü VIEW erreichen. Entweder exportieren Sie sämtliche Meldungen in der aktuellen Sicht (EXPORT VIEW), oder Sie wählen zuvor einzelne Meldungen aus und exportieren diese (EXPORT SELECTED MESSAGES).

15 SELinux und boolesche Variablen

Wenn SELinux allgemein eingesetzt werden soll, müssen einfache Änderungen an der Richtlinie leicht durch den Administrator durchzuführen sein. Obwohl sich viele Linux-Systeme gleichen, ist es dennoch so, dass die eingesetzten Applikationen sich im Detail doch unterscheiden.

Ein Beispiel sind zwei Linux-Systeme, die als Webserver eingesetzt werden. Ein *Apache* Webserver bietet auch die Heimatverzeichnisse der Benutzer an, während der zweite ein Content-Management-System mit einer MySQL-Datenbank im Hintergrund betreibt. Während der erste Zugriff auf die Heimatverzeichnisse der Benutzer benötigt, muss der zweite Netzwerkverbindungen zur *MySQL*-Datenbank aufbauen dürfen.

Es wäre nun unsinnig, eine SELinux-Richtlinie zu erzeugen, die alle diese Zugriffe erlaubt. Diese Richtlinie wäre zu löchrig. Daher wählten die SELinux-Entwickler einen anderen Weg und haben diese Zugriffe in Ihrer Richtlinie vorgesehen und *Boolesche Variablen* eingeführt, über die Sie die entsprechenden Regeln fein granuliert an- und abschalten können.

In diesem Kapitel werde ich Ihnen zeigen, wie Sie diese booleschen Variablen nutzen und konfigurieren. Die Verwendung der booleschen Variablen in eigenen Richtlinien wird in Kapitel 25 besprochen.

Zunächst sollten Sie sich einen Überblick darüber verschaffen, welche booleschen Variablen Ihr System unterstützt. Dies hängt von der eingesetzten Richtlinie ab und kann sich daher auch bei einem Update der Richtlinie ändern.

Der Befehl, mit dem Sie die booleschen Variablen und ihren aktuellen Wert anzeigen können, lautet `getsebool`. Mit der Option `-a` zeigt er alle Variablen an:

```
[root@supergrobi ~]# getsebool -a
allow_cvs_read_shadow --> off
allow_execheap --> off
allow_execmem --> off
allow_execmod --> off
allow_execstack --> off
allow_ftpd_anon_write --> off
```

```
allow_gpg_execstack --> off
allow_gssd_read_tmp --> on
...
allow_httpd_anon_write --> off
allow_httpd_apcupsd_cgi_script_anon_write --> off
allow_httpd_bugzilla_script_anon_write --> off
...
```

Einige dieser Variablen sind in einzelnen Manpages erläutert. So existiert eine Manpage `httpd_selinux(8)`, die die SELinux- Policy für den Apache Webserver einschließlich der hier wichtigen booleschen Variablen erläutert. Da diese Manpages leider nicht immer vollständig sind und auch nicht alle booleschen Variablen in Manpages erläutert werden, werde ich zunächst die aktuell in *Fedora Core* 5 vorhandenen Variablen erläutern.

- `allow_cvs_read_shadow`: Diese Variable entscheidet, ob der CVS-Dienst die Datei `/etc/shadow` lesen darf, um Benutzer zu authentifizieren.
- `allow_execheap`: Hiermit entscheiden Sie, ob Prozesse Code auf dem Heap ausführen dürfen.
- `allow_execmem`: Hiermit entscheiden Sie, ob Prozesse Speicher, der ursprünglich zur Speicherung von Daten genutzt wurde, als Code ausführen dürfen. Dies ist bei neu generiertem Code durch die Applikation der Fall.
- `allow_execmod`: Hiermit entscheiden Sie, ob im Speicher geladene (mmapped) modifizierte Dateien ausgeführt werden dürfen.
- `allow_execstack`: Hiermit entscheiden Sie, ob Daten auf dem Stapel ausgeführt werden dürfen (benötigt auch `allow_execmem`).
- `allow_ftpd_anon_write`: Hiermit entscheiden Sie, ob die Policy dem FTP-Server Schreibrechte an den öffentlichen Dateien einräumt. Als öffentliche Dateien werden Dateien mit dem Typ *public_content_t* und *public_content_rw_t* bezeichnet. Wenn Sie möchten, dass der FTP-Server die Daten mit dem zweiten Typ schreiben darf, müssen Sie diese Variable setzen.
- `allow_gpg_execstack`: Diese Variable erlaubt spezifisch der GnuPG-Domäne das Ausführen von Code auf dem Stack.
- `allow_gssd_read_tmp -> on`: Diese Variable erlaubt dem `rpc.gssd`-Daemon das Lesen der temporären Verzeichnisse.
- `allow_httpd_anon_write`: Hiermit erlauben Sie dem Apache die Modifikation der öffentlichen Dateien (siehe `allow_ftpd_anon_write`).
- `allow_httpd_staff_script_anon_write`: Hiermit verwalten Sie das entsprechende Recht für Scripts in der *httpd_staff*-Domäne.
- `allow_httpd_sys_script_anon_write`: Hiermit verwalten Sie das entsprechende Recht für Scripts in der *httpd_sys*-Domäne.
- `allow_httpd_sysadm_script_anon_write`: Hiermit verwalten Sie das entsprechende Recht für Scripts in der *httpd_sysadm*-Domäne.

- `allow_httpd_user_script_anon_write`: Hiermit verwalten Sie das entsprechende Recht für Scripts in der *httpd_user*-Domäne.
- `allow_java_execstack`: Dies erlaubt der Java Virtual Machine das Ausführen von Code auf dem Stack. Die Java Virtual Machine benötigt diese Funktion unter Umständen.
- `allow_kerberos`: Dies erlaubt es dem System, für die Authentifizierung Kerberos einzusetzen.
- `allow_ptrace`: Hiermit erlauben Sie auf dem System die Verwendung von `ptrace` zur Fehlersuche.
- `allow_rsync_anon_write`: Dies erlaubt dem Rsync-Daemon, die öffentlichen Dateien zu lesen (siehe `allow_ftpd_anon_write`).
- `allow_saslauthd_read_shadow`: Dies erlaubt dem Saslauthd-Daemon, die Datei `/etc/shadow` zu lesen.
- `allow_smbd_anon_write`: Mit dieser Variable darf der Samba-Dienst die öffentlichen Dateien schreiben (siehe `allow_ftpd_anon_write`).
- `allow_ssh_keysign`: Diese Variable erlaubt die Authentifizierung bei SSH mit Public Keys.
- `allow_user_mysql_connect`: Diese Variable erlaubt den Benutzern, eine Verbindung zum Mysql-DBMS aufzubauen.
- `allow_write_xshm`: Diese Variable erlaubt das Schreiben des gemeinsam genutzten Speichers des X-Servers (X-Server shared memory).
- `allow_ypbind`: Dies erlaubt es dem System, zur Authentifizierung einen NIS-Server zu verwenden.
- `cdrecord_read_content`: Dies erlaubt es dem Befehl `cdrecord` verschiedenste Dateien zu lesen.
- `cron_can_relabel`: Hiermit darf der Cron-Daemon Dateien einen neuen Security-Context zuweisen.
- `fcron_crond`: Diese Variable erweitert die Regeln für den Cron-Daemon, sodass auch der Fcron-Daemon[1] funktioniert.
- `ftp_home_dir`: Hier entscheiden Sie, ob der FTP-Daemon Benutzerverzeichnisse lesen und schreiben darf.
- `ftpd_is_daemon`: Diese Variable erlaubt den Betrieb des FTP-Daemons ohne Inetd.
- `httpd_builtin_scripting`: Diese Variable erlaubt dem Webserver die Verwendung von eingebauten Scriptsprachen (`mod_php`, `mod_perl` etc.).
- `httpd_can_network_connect`: Diese Variable erlaubt dem Webserver den Aufbau von Netzwerkverbindungen.
- `httpd_can_network_connect_db`: Hiermit darf der Webserver Netzwerkverbindungen zu MySQL und PostgreSQL-DBMS aufbauen.

[1] *http://fcron.free.fr/*

- `httpd_can_network_relay`: Der Apache darf Netzwerkverbindungen weiterleiten.
- `httpd_enable_cgi`: Diese Variable aktiviert die Regeln, die CGI-Programme erlauben.
- `httpd_enable_ftp_server`: Diese Variable erlaubt dem Webserver die Bindung auf dem FTP-Port.
- `httpd_enable_homedirs`: Diese Variable erlaubt dem Webserver den Zugriff auf die Heimatverzeichnisse der Benutzer.
- `httpd_ssi_exec`: Mit dieser Variable werden Server Side Includes in der *httpd_script_t*-Domäne ausgeführt.
- `httpd_tty_comm`: Mit dieser Variable darf der Webserver mit dem Terminal kommunizieren, von dem er aufgerufen wurde.
- `httpd_unified`: Mit dieser Variable werden CGI-Scripts in der *httpd_t*-Domäne gestartet.
- `named_write_master_zones`: Diese Variable erlaubt es dem Bind-Nameserver, die Masterzonen zu verändern. Dies wird bei dynamischen DNS benötigt.
- `nfs_export_all_ro`: Diese Variable erzwingt den Read-only-Export der Dateisysteme durch den NFS-Server.
- `nfs_export_all_rw`: Diese Variable erlaubt es dem NFS-Server, Verzeichnisse auch read/write zu exportieren.
- `pppd_can_insmod`: Mit dieser Variable erlauben Sie es dem PPP-Daemon, notwendige Kernel-Module nachzuladen.
- `pppd_for_user`: Diese Variable erlaubt es normalen Benutzern, den PPP-Daemon zu verwenden.
- `read_default_t`: Diese Variable erlaubt das Lesen von Dateien mit dem *default_t*.
- `read_untrusted_content`: Diese Variable erlaubt das Lesen von Dateien aus unbekannten Quellen (Internet). Ansonsten müssen die Dateien zunächst umgelabelt werden.
- `run_ssh_inetd`: Hiermit darf der SSH-Daemon über den Inetd gestartet werden.
- `samba_enable_home_dirs`: Diese Variable erlaubt es dem Samba-Dienst, auf die Heimatverzeichnisse der Benutzer zuzugreifen.
- `secure_mode`: Ist diese Variable gesetzt, dürfen Programme wie `newrole` nicht mehr in administrative Domänen wechseln.
- `secure_mode_insmod`: Ein Wechsel in die Domäne *insmod* ist nicht mehr erlaubt. Module können nicht mehr geladen werden.
- `secure_mode_policyload`: Wurde diese Variable gesetzt, ist ein Reload der Policy, ein Ändern der booleschen Variablen und das Deaktivieren von SELinux erst durch einen Reboot möglich.
- `spamassasin_can_network`: Diese Variable ist aus historischen Gründen erforderlich.
- `spamassassin_can_network`: SpamAssassin darf Netzwerkverbindungen zu DNS-Servern etc. aufbauen.

- `squid_connect_any`: Hiermit darf der Squid sich mit jedem beliebigen Port verbinden. Ist die Variable nicht gesetzt, beschränkt sich das auf die HTTP-, HTTPS-, FTP- und Gopher-Ports.
- `ssh_sysadm_login`: Mit dieser Variable entscheiden Sie, ob eine Anmeldung per SSH als `sysadm_r:sysadm_t` möglich ist.
- `staff_read_sysadm_file`: Hiermit entscheiden Sie, ob Personen in der Rolle *staff_r* Dateien vom Typ *sysadm_t* lesen dürfen.
- `stunnel_is_daemon`: Diese Variable erlaubt den Betrieb von `stunnel` als Daemon.
- `use_nfs_home_dirs`: Mit dieser Variable erlauben Sie den Betrieb der Heimatverzeichnisse über NFS.
- `use_samba_home_dirs`: Diese Variable erlaubt analog Heimatverzeichnisse auf einem Samba-Server.
- `user_direct_mouse`: Mit dieser Variable dürfen Benutzer direkt auf die Maus zugreifen.
- `user_dmesg`: Dies erlaubt es normalen Benutzern, den Befehl `dmesg` zu benutzen.
- `user_net_control`: Hiermit erlauben Sie es Benutzern, bestimmte Netzwerkgeräte (`USERCTL=true`) zu aktivieren.
- `user_ping`: Dies erlaubt es einfachen Benutzern, den Befehl `ping` zu verwenden.
- `user_rw_noexattrfile`: Mit dieser Variable erlauben Sie einfachen Benutzern, Dateien auf Dateisystemen zu lesen und zu schreiben, die keine erweiterten Attribute unterstützen. Auf diesen Dateisystemen kann SELinux keinen Schutz bieten.
- `user_rw_usb`: Hiermit erlauben Sie Benutzern das Lesen und Schreiben von USB-Geräten.
- `user_tcp_server`: Hiermit erlauben Sie einfachen Benutzern das Betreiben von TCP-Netzwerkdiensten. Dies ist zum Beispiel für aktives FTP erforderlich.
- `user_ttyfile_stat`: Hiermit definieren Sie das Verhalten der Befehle `w`, `who` etc. Die Variable entscheidet, ob alle Benutzer angezeigt werden können.
- `write_untrusted_content`: Diese Variable entscheidet, ob Applikationen-Dateien unbekannter Quelle speichern dürfen. Ist diese Variable nicht gesetzt, können Daten aus dem Internet nicht gespeichert werden.
- `xdm_sysadm_login`: Mit dieser Variable entscheiden Sie, ob grafische Anmeldungen als `sysadm_r:sysadm_t` möglich sind.

15.1 Administration der booleschen Variablen

Für die Administration der booleschen Variablen gibt es zwei verschiedene Varianten. Entweder setzen und lesen Sie die Variablen direkt über das virtuelle *SELinux!-Dateisystem* (`/selinux/booleans`), oder Sie verwenden die Befehle `getsebool` und `setsebool`.

Da die Handhabung der Befehle wesentlich einfacher ist, werde ich mich zunächst auf diese beschränken. Der Befehl `getsebool` macht genau das, was man von ihm erwartet. Er liest den Wert einer booleschen Variable aus:

```
[root@supergrobi policy]# getsebool allow_ssh_keysign
allow_ssh_keysign --> off
```

Mit der Option `-a` zeigt er alle Variablen und ihre Werte an.

Mit dem Befehl `setsebool` können Sie die Variablen verändern. Um die gerade gelesene Variable zu setzen, verwenden Sie:

```
[root@supergrobi policy]# setsebool allow_ssh_keysign=1
[root@supergrobi policy]# getsebool allow_ssh_keysign
allow_ssh_keysign --> on
```

Sie können auch mehrere boolesche Variablen gleichzeitig angeben. Diese werden dann auch gleichzeitig gesetzt.

Nach einem Reboot werden die booleschen Variablen aber wieder auf ihre Ausgangswerte gesetzt. Damit die Änderungen auch nach einem Reboot wieder aktiviert werden, müssen sie in der Datei `/etc/selinux/<policy>/modules/active/booleans.local` eingetragen werden. Da diese manuelle Tätigkeit fehleranfällig ist, unterstützt der Befehl `setsebool` Sie dabei. Mit der Option `-P` (für permanent) schreibt er die Änderung auch in diese Datei.

Die zweite Variante der Verwaltung der Variablen ist das *SELinux!-Dateisystem* in `/selinux/booleans`. Zunächst können Sie die booleschen Variablen auslesen:

```
[root@supergrobi policy]# cat /selinux/booleans/allow_ssh_keysign
1 1
```

Sie erhalten dabei immer zwei Werte angezeigt. Der erste Wert ist der aktuelle Wert der Variablen, während der zweite Wert die Änderung nach dem nächsten Commit angibt (*Pending*). Wenn Sie nun eine Variable verändern, wird diese Änderung zunächst nur gespeichert, aber noch nicht umgesetzt:

```
[root@supergrobi policy]# echo 0 > /selinux/booleans/allow_ssh_keysign
[root@supergrobi policy]# cat /selinux/booleans/allow_ssh_keysign
1 0
[root@supergrobi ~]# getsebool allow_ssh_keysign
allow_ssh_keysign --> on pending: off
```

Die Umsetzung erfolgt erst mit dem Commit der Änderungen. Hierzu müssen Sie die Datei `/selinux/commit_pending_bools` schreiben:

```
[root@supergrobi policy]# echo 1 > /selinux/commit_pending_bools ign
[root@supergrobi policy]# cat /selinux/booleans/allow_ssh_keysign
0 0
```

Damit besteht die Möglichkeit, mehrere Änderungen gleichzeitig an der Policy vorzunehmen. Der Befehl `setsebool` nutzt übrigens auch diese Schnittstelle. Wenn Sie für eine Variable eine Änderung definiert haben und anschließend mit `setsebool` eine weitere Änderung durchführen, wird auch die erste wartende Änderung durchgeführt!

In vielen Fällen können Sie die mitgelieferte Richtlinie bereits über die sinnvolle Anpassung der booleschen Variablen an Ihre Wünsche anpassen, ohne dass Sie tiefergreifende Änderungen vornehmen müssen.

16 SELinux-Anpassungen

Die von den Distributionen mitgelieferte SELinux-Richtlinie ist meist sehr universell einsetzbar und für die meisten Fälle ausreichend. Einfache Anpassungen können sehr leicht über die booleschen Variablen (siehe Kapitel 15) erfolgen. Ab und zu reicht das aber nicht.

Entweder möchten Sie, dass die Dateien in einem bestimmten Verzeichnis einen besonderen *Security-Context* erhalten, oder Sie möchten den *Apache* Webserver auf einem anderen zusätzlichen *Port* laufen lassen oder einer Applikation mehr Rechte einräumen, als sie aktuell besitzt.

Ich werde diese Aufgaben hier recht allgemein darstellen. Ausführlichere Beispiele der Anpassung finden Sie in Kapitel 19.

16.1 Verwaltung der SELinux-Benutzer

Mit dem Befehl `semanage` können Sie die wichtigsten Anpassungen mit einigen Einschränkungen vornehmen. Sie administrieren zum Beispiel die SELinux-Benutzer und ihre Rollen mit diesem Befehl. Wenn Sie möchten, dass ein bestimmter Benutzer besonders behandelt wird, können Sie ihm zunächst einen eigenen *SELinux!-User* zuordnen. Stellen Sie sich vor, der Benutzer *ralf* soll auch die neue Rolle *webadm_r* wahrnehmen dürfen. Dann können Sie das folgendermaßen erreichen:

```
[root@supergrobi policy]# semanage user --roles 'user_r webadm_r' ➥
        --prefix user --add ralf_u
```

Hiermit erzeugen Sie zunächst einen SELinux-Benutzer *ralf_u*. Der Befehl weist dem Benutzer die beiden Rollen *user_r* und *webadm_r* zu. Natürlich müssen diese beiden Rollen auch existieren. Sie können es alternativ auch mit der Rolle *sysadm_r* testen.

Anschließend weisen Sie bei dem Login dem Linux-Benutzer *ralf* den SELinux-Benutzer *ralf_u* zu.

```
[root@supergrobi policy]# semanage login --add --seuser ralf_u ralf
```

Mit dem Befehl `semanage user -l` bzw. `semanage login -l` sehen Sie die definierten SELinux-Benutzer und ihre Zuordnung zu echten Linux-Benutzern. Natürlich können Sie auch Benutzer löschen. Das funktioniert jedoch nur bei den Benutzern, die Sie selbst erzeugt haben. Benutzer, die in der Policy definiert wurden, können nicht gelöscht werden:

```
[root@supergrobi policy]# semanage user -d root /usr/sbin/semanage:
SELinux user root is defined in policy, cannot be deleted
```

Das ist allgemein der Fall auch bei Ports und Security-Contexts.

16.2 Verwaltung der Ports

Die Netzwerk-Ports können nun genauso verwaltet werden wie die Benutzer. Auch hier wird der Befehl `semanage` verwendet, und auch hier können Sie die durch die Policy definierten Ports nicht ändern oder löschen. Wenn Sie aber möchten, dass der *Apache* Webserver auf einem zusätzlichen *Port* horchen soll, können Sie das hiermit einfach erreichen.

Um dem Apache Webserver in der Policy auch den Port 81 zu erlauben, müssen Sie lediglich den folgenden Befehl benutzen:

```
[root@supergrobi policy]# semanage port --add --proto tcp --type ➥
         http_port_t 81
[root@supergrobi policy]# semanage port -l | grep http_port_t
http_port_t                    tcp      81, 80, 443, 488, 8008, 9050
```

Wichtig bei den Ports ist die Angabe des Protokolls. Sie müssen mit `-p` oder `--proto` das Transportprotokoll (`udp` oder `tcp`) angeben.

16.3 Verwaltung der Security-Contexts der Dateien

Schließlich können Sie mit dem Befehl `semanage` auch beeinflussen, welchen *Security-Context* welche Datei erhält (File-Context). Der zu verwendende Security-Context wird von der Policy definiert. Sie können sich alle definierten Security-Contexts mit `semanage fcontext -l` anzeigen lassen. Es ist wiederum nicht möglich, in der Policy vordefinierte File-Contexts zu ändern oder zu löschen. Sie können aber zusätzliche Definitionen hinzufügen.

Ein häufiges Problem stellt zum Beispiel beim Betrieb eines Webservers die Auslagerung der Webseiten dar. Häufig erzeugt der Administrator ein eigenes Verzeichnis als *DocumentRoot* (z.B. `/web`) in dem die Webseiten gespeichert werden. Dieses Verzeichnis erhält per Definition zunächst den File-*Security-Context system_u*:*object_r*:*default_t*:*s0*. Die Policy für den Apache erlaubt es ihm nicht, auf diese Dateien zuzugreifen. Anstatt nun dem Webserver zu erlauben, auf diese zusätzlichen Security-Contexts zuzugreifen, sollten diese Dateien so gelabelt werden, dass die vorhandene Policy den Zugriff erlaubt.

Das gelingt Ihnen mit dem Kommando `semanage`. Das Kommando unterstützt hierbei reguläre Ausdrücke. Damit nun jede Datei in diesem Verzeichnis und weiteren Unterverzeichnissen und das Verzeichnis `/web` selbst den richtigen Context erhalten, setzen Sie den folgenden Befehl ab:

```
[root@supergrobi policy]# semanage fcontext --add --type ➥
         httpd_sys_content_t '/web(/.*)?'
```

Dies setzt sowohl für das Verzeichnis als auch für seinen Inhalt den Context. Wenn Sie nun mit `restorecon` den *Security-Context* reparieren, erhält das Verzeichnis den richtigen Security-Context[1]. Jede weitere in dem Verzeichnis angelegte Datei erhält nun auch automatisch den richtigen Security-Context, da sich der Security-Context des Verzeichnisses vererbt.

```
[root@supergrobi policy]# restorecon -R /web
[root@supergrobi policy]# ls -Zd /web
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t /web
```

16.4 Customizable Types

Es gibt *Customizable Types*. Hierbei handelt es sich um eine Liste von Typen, die bei einem *Relabeling* des Systems oder bei Verwendung des Befehls `restorecon` nicht zurückgesetzt werden. Diese Typen werden in der Datei `contexts/customizable_types` definiert. Bei der *Fedora Core* 6-Distribution enthält diese Datei bei der *Targeted-Policy* die folgenden Typen:

```
cvs_data_t
httpd_bugzilla_content_t
httpd_bugzilla_htaccess_t
httpd_bugzilla_script_exec_t
httpd_bugzilla_script_ra_t
httpd_bugzilla_script_ro_t
httpd_bugzilla_script_rw_t
httpd_squid_content_t
httpd_squid_htaccess_t
httpd_squid_script_exec_t
httpd_squid_script_ra_t
httpd_squid_script_ro_t
httpd_squid_script_rw_t
httpd_sys_content_t
httpd_sys_htaccess_t
httpd_sys_script_exec_t
httpd_sys_script_ra_t
httpd_sys_script_ro_t
httpd_sys_script_rw_t
```

[1] Wenn dies bei Ihnen nicht funktioniert, lesen Sie bitte auch Abschnitt 16.4!

```
httpd_unconfined_script_exec_t
mount_loopback_t
public_content_rw_t
public_content_t
samba_share_t
swapfile_t
xen_image_t
```

Sobald eine Datei diesen Typ besitzt, wird sie bei einem *Relabeling* oder bei Verwendung der Befehle `restorecon`, `fixfiles` oder `setfiles` nicht modifiziert.

Sie können diese Liste natürlich auch selbst erweitern.

16.5 Erweiterung der Policy

Häufig können die Änderungen mit den Befehlen `setsebool` oder `semanage` so durchgeführt werden, dass eine Erweiterung der Policy nicht nötig ist. Wenn jedoch die Applikation grundsätzlich noch nicht über die benötigten Privilegien verfügt und diese auch über boolesche Variablen nicht anschaltbar sind, dann müssen Sie die Policy erweitern.

Hierzu sollten Sie zunächst die Applikation installieren und prüfen, ob für die Applikation bereits eine Policy existiert, die nicht Ihren Ansprüchen genügt, oder ob eine komplett neue Policy für die Applikation entwickelt werden muss. Im zweiten Fall möchte ich Sie auf Teil IV verweisen, in dem Ihnen genau erklärt wird, wie Sie eine Policy bauen. Kapitel 24 gibt Ihnen dabei eine schnelle Einführung.

Hier wollen wir uns ansehen, wie eine vorhandene Policy erweitert werden kann, um einer Applikation zusätzliche Rechte einzuräumen. Als Beispiel verwende ich hier wieder die Richtlinie für den Webserver und die *PHP*-Applikation *phpSysInfo* (siehe auch Abschnitt 5.6 und Abschnitt 9.2). Hierzu laden Sie die Applikation von ihrer Homepage (*http://phpsysinfo.sf.net*) herunter und entpacken sie im Document-Root des Apache Webservers:

```
# cd /var/www/html
# tar xzf /path/phpsysinfo-<version>.tar.gz
# cd phpsysinfo
# mv config.php.new config.php
```

Testen Sie nun, ob die Applikation sich so verhält, wie Sie es wünschen. Dabei sollte SELinux abgeschaltet sein oder sich mindestens im *Permissive*-Mode befinden. Sie sollten eine Webseite wie in Abbildung 16.1 erhalten. Diese ähnelt der Abbildung 5.10. Sobald Sie nun SELinux wieder mit `setenforce` in den *Enforcing*-Mode versetzen, wird die Applikation nicht mehr alle Informationen anzeigen (siehe Abbildung 16.2). Gleichzeitig können Sie viele Meldungen von SELinux in der Protokolldatei (`/var/log/messages` oder `/var/log/audit/audit.log`) verfolgen. Um nun die Richtlinien zu erweitern, müssen Sie aus den Fehlermeldungen entsprechende `allow`-Regeln erzeugen. Am einfachsten erfolgt das mit dem Befehl `audit2allow`. Dieser liest die Protokolldatei und erzeugt daraus entsprechende Allow-Meldungen.

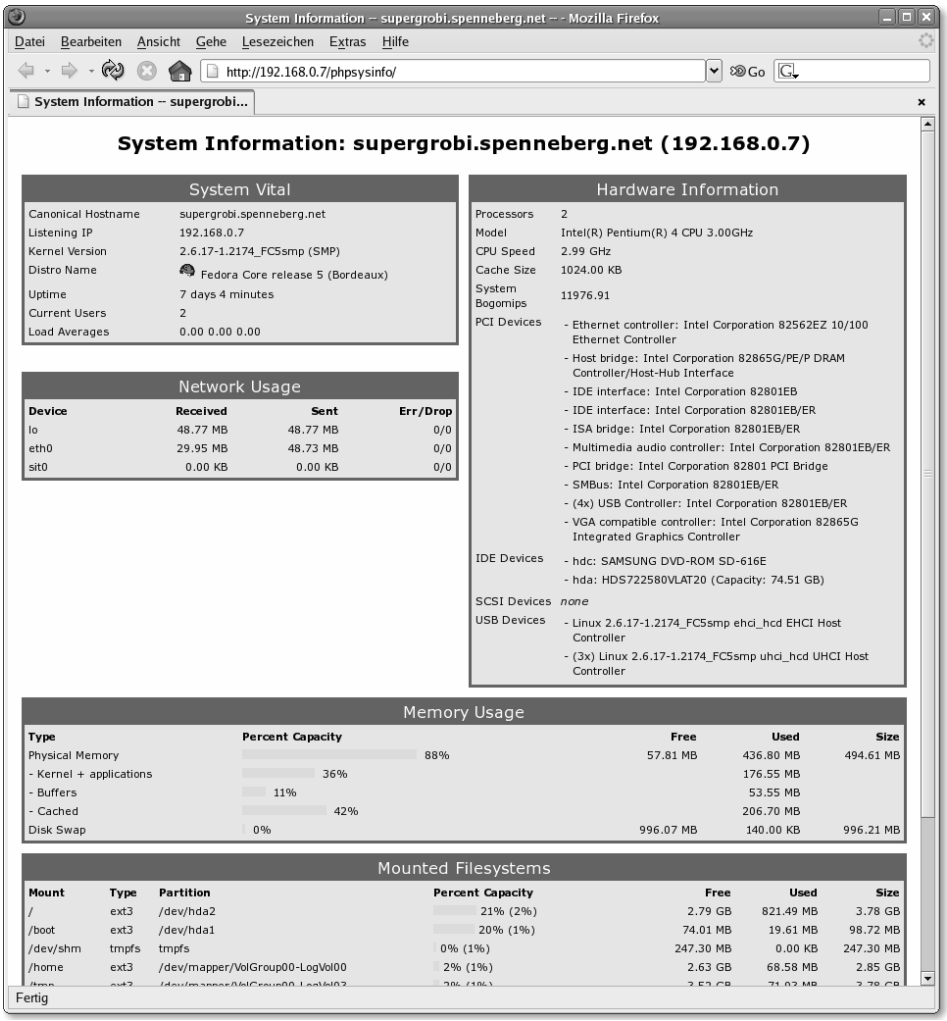

Abbildung 16.1: PHPSysInfo zeigt die Eigenschaften der Fedora Core- Installation an.

Wir werden hier einen schnellen und einfachen Weg zum Ziel nutzen. Später werde ich Ihnen noch eine bessere Alternative zeigen (siehe Kapitel 23).

Damit der Befehl `audit2allow` nicht auch noch Meldungen analysiert, die die *PHP*-Anwendung nicht betreffen, sollten Sie vor dem Einsatz die Policy neu laden. Dies führt zu einem Eintrag in dem Protokoll, und Sie können anschließend dem `audit2allow`-Kommando auftragen, nur die Meldungen zu verarbeiten, die seit dem letzten *Reload* hinzugekommen sind. Natürlich sollten Sie darauf achten, dass Sie nicht gleichzeitig auch noch andere Programme nutzen, damit nicht noch weitere unbeteiligte Meldungen verarbeitet werden.

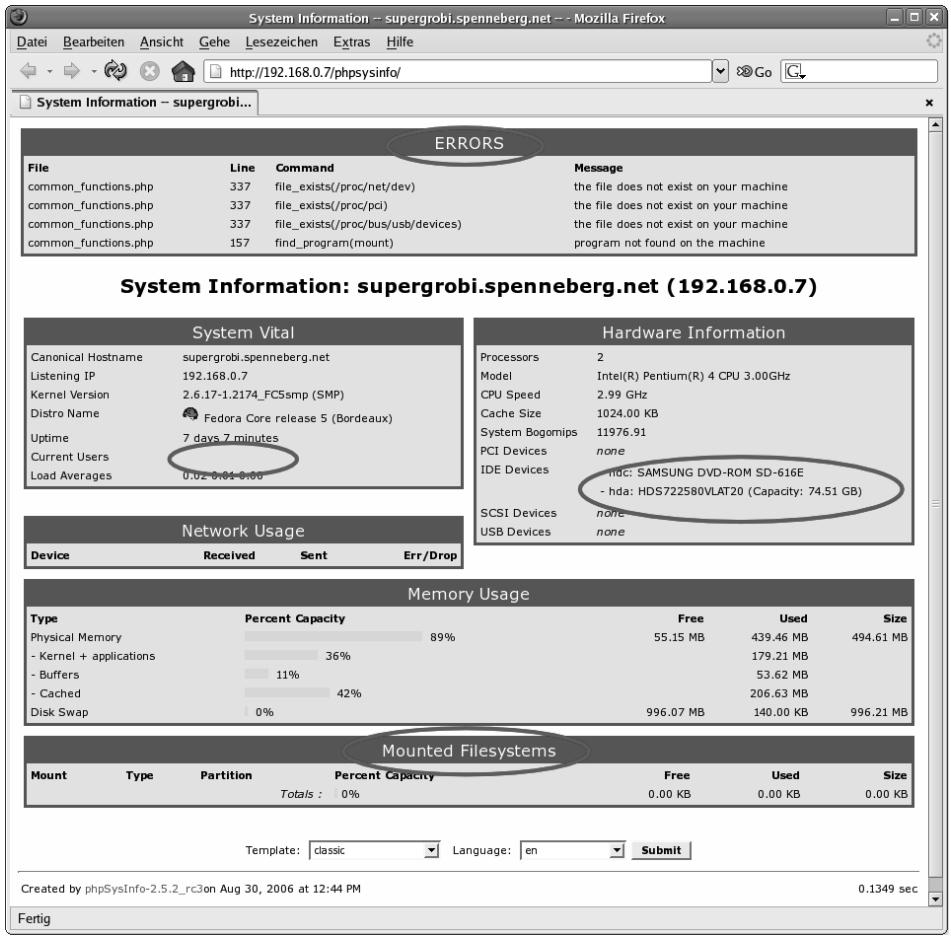

Abbildung 16.2: SELinux verhindert die Ausführung vieler Befehle.

Bevor wir die SELinux-Policy neu laden, versetzen wir SELinux in den *Permissive*-Mode (`setenforce 0`). In diesem Modus protokolliert SELinux jede Verletzung, verhindert sie aber nicht. Es handelt sich also um eine Art Lernmodus, wenn anschließend die Meldungen mit `audit2allow` ausgewertet werden.

Achtung

Achten Sie darauf, dass Ihr System nun überhaupt nicht von SELinux geschützt wird. Verzichten Sie daher auf Netzwerktätigkeiten, und unterbinden Sie den Zugriff von außen.

Um nun die SELinux-Policy komplett neu zu laden, benutzen Sie den Befehl `semodule`:

```
[root@supergrobi phpsysinfo]# semodule -R
[root@supergrobi phpsysinfo]# tail /var/log/audit/audit.log
...
type=AVC msg=audit(1156940576.026:3334): avc:  granted ⬅
         { load_policy } for  pid=29591 comm="load_policy" scontext= ⬅
         root:sysadm_r:load_policy_t:s0-s0:c0.c255 tcontext= ⬅
         system_u:object_r:security_t:s0 tclass=security
type=MAC_POLICY_LOAD msg=audit(1156940576.026:3334): policy ⬅
         loaded auid=0
...
```

Nun benutzen Sie erneut die entsprechende Applikation, deren Policy Sie erweitern möchten. Hier verwenden wir als Beispiel wieder *phpSysInfo*. Wenn Sie dabei gleichzeitig das SELinux-Protokoll beobachten, können Sie die hinzugefügten Meldungen direkt betrachten und analysieren.

Anstatt das jedoch manuell zu tun, werden wir nun den Befehl `audit2allow` verwenden. Die Anwendung ist kinderleicht. Mit der Option `-a` weisen Sie den Befehl an, sowohl die Datei `/var/log/messages` als auch die Datei `/var/log/audit/audit.log` auf SELinux-Meldungen hin zu durchsuchen[2]. Alternativ könnten Sie mit der Option `-i <datei>` auch eine bestimmte Datei auswählen. Wenn Sie weder `-a` noch `-i <file>` angeben, liest der Befehl von der Standardeingabe. Die Option `-l` benötigen Sie, weil der Befehl lediglich die Meldungen seit dem letzten *Reload* der Policy analysieren soll. Mit der Option `-M <modul>` weisen Sie den Befehl `audit2allow` an, direkt ein SELinux Policy-Modul zu bauen, das Sie später mit `semodule` laden können.

Für unseren Anwendungsfall bedeutet das:

```
[root@supergrobi phpsysinfo]# audit2allow -a -l -M phpsysinfo
Generating type enforcment file: phpsysinfo.te
Compiling policy
checkmodule -M -m -o phpsysinfo.mod phpsysinfo.te
semodule_package -o phpsysinfo.pp -m phpsysinfo.mod

******************** IMPORTANT ***********************

In order to load this newly created policy package into the kernel,
you are required to execute

semodule -i phpsysinfo.pp
```

Mit diesen Schritten haben wir nun ein fertiges *Modul* gebaut, das direkt von uns geladen werden kann. Wenn Sie die erzeugten `allow`-Regeln noch einmal kontrollieren

[2] Unter Debian müssen Sie die Datei `/var/log/syslog` mit der Option `-i` angeben.

möchten, können Sie sich die Datei `phpsysinfo.te` ansehen. Diese Datei enthält die *Type-Enforcement*-Regeln. Sie erkennen das an der Endung `*.te`. Bei diesem Beispiel enthält die Datei die folgenden Regeln:

```
module phpsysinfo 1.0;

require
        class dir  getattr search ;
        class file  execute execute_no_trans getattr lock read ;
        type etc_runtime_t;
        type httpd_t;
        type hwdata_t;
        type initrc_var_run_t;
        type mount_exec_t;
        type proc_net_t;
        type shell_exec_t;
        type sysctl_fs_t;
        type usbfs_t;
        type var_lib_nfs_t;
        role system_r;
;

allow httpd_t etc_runtime_t:dir search;
allow httpd_t hwdata_t:dir search;
allow httpd_t hwdata_t:file  getattr read ;
allow httpd_t initrc_var_run_t:file  lock read ;
allow httpd_t mount_exec_t:file  execute execute_no_trans read ;
allow httpd_t proc_net_t:dir  getattr search ;
allow httpd_t proc_net_t:file  getattr read ;
allow httpd_t shell_exec_t:file  execute execute_no_trans ➥
         getattr read ;
allow httpd_t sysctl_fs_t:dir search;
allow httpd_t usbfs_t:dir  getattr search ;
allow httpd_t usbfs_t:file  getattr read ;
allow httpd_t var_lib_nfs_t:dir search;
```

Falls Sie manuelle Anpassungen vornehmen möchten, können Sie das Modul anschließend auch mit `checkmodule` von Hand übersetzen. Sie sollten lediglich darauf achten, dass Sie bei jeder Modifikation die *Versionsnummer* am Anfang inkrementieren.

```
[root@supergrobi phpsysinfo]# checkmodule -M -m -o phpsysinfo.mod ➥
         phpsysinfo.te
checkmodule:  loading policy configuration from phpsysinfo.te
checkmodule:  policy configuration loaded
checkmodule:  writing binary representation (version 5) to ➥
         phpsysinfo.mod
```

```
[root@supergrobi phpsysinfo]# semodule_package -o phpsysinfo.pp ➥
         -m phpsysinfo.mod
```

Um nun dieses Modul zu laden, verwenden Sie den Befehl `semodule`. Ist bisher keine alte Version des Moduls geladen, genügt die Option `-i` für »Installation«. Ist das Modul bereits geladen und Sie möchten es durch ein Modul mit höherer Versionsnummer ersetzen, verwenden Sie die Option `-u`. Mit der Option `-r` entfernen Sie das Modul wieder.

```
[root@supergrobi phpsysinfo]# semodule -i phpsysinfo.pp
[root@supergrobi phpsysinfo]# semodule -l | grep php
phpsysinfo      1.0
```

Wenn Sie nun SELinux wieder in den `Enforcing`-Modus versetzen, sollte die Applikation weiterhin funktionieren. Ihr erzeugtes Modul versorgt SELinux nun mit den notwendigen zusätzlichen *Type-Enforcement*-Regeln.

17 Policys

17.1 Die Targeted-Policy

Aktuell liefern Debian, das Fedora-Projekt und Red Hat ihre Distributionen mit mindestens zwei Richtlinien aus: *Strict* und *Targeted*. Zusätzlich gibt es auch noch die *MLS*-Policy. Ich verwende in diesem Buch bei den Beispielen in erster Linie die *Targeted*-Policy, die auch der Default bei diesen Distributionen ist. Bei dem kommerziellen Red Hat Enterprise Linux wird auch nur für die *Targeted*-Policy Support angeboten.

Wie unterscheiden sich nun die Policys? Grundsätzlich überwacht bei allen Policys SELinux den gesamten Rechner und alle Prozesse. Grundsätzlich gilt auch bei der *Targeted*-Policy, dass jeder Zugriff, der nicht explizit erlaubt wird, von SELinux unterbunden wird. Jedoch verfügt die *Targeted*-Policy über eine besondere Domäne *unconfined_t*. Jeder Prozess, der diese Domäne verwendet, hat fast uneingeschränkten Zugriff auf das System. Bei der *Targeted*-Policy werden nur bestimmte Dienste in eigenen Domänen gestartet. Diese Domänen erlauben dann nur die von den Diensten tatsächlich benötigten Zugriffe. Sobald ein Benutzer sich lokal an dem System anmeldet, wird ihm die Domäne *unconfined_t* zugewiesen, und er wird von SELinux nicht wesentlich eingeschränkt. Benutzerspezifische Domänen wie *user_t* oder *sysadm_t* gibt es bei der *Targeted*-Policy nicht. Auch weisen die unterschiedlichen Benutzer nicht unterschiedliche Rollen auf, sondern verwenden alle die Rolle *system_r*. Die *Targeted*-Policy stellt so sicher, dass die exponierten Dienste von SELinux überwacht werden und das System vor deren Fehlverhalten geschützt wird, aber die Funktion des System für authentifizierte Benutzer nicht weiter eingeschränkt wird. Der authentifizierte Benutzer wird als vertrauenswürdig eingestuft. Seine Überwachung erfolgt nur über die üblichen Linux-Rechte (*DAC*).

Tipp

In einer gewissen Weise ähnelt die *Targeted*-Policy dem Ansatz von AppArmor. Dies wird von den AppArmor-Entwicklern auch immer wieder angeführt. Dennoch darf man nicht vergessen, dass auch im *Targeted*-Modus SELinux wesentlich mächtiger ist und immer noch zum Beispiel den *Informationsfluss* der Daten überwachen kann.

17.1.1 Deaktivierung von SELinux für einzelne Applikationen

Bei der Targeted-Policy können Sie SELinux für einzelne Applikationen »abschalten«. Natürlich wird nicht SELinux tatsächlich abgeschaltet, aber Sie können SELinux anweisen, die Applikation in der *Domäne unconfined_t* auszuführen und nicht in ihrer eigenen eingeschränkten Domäne. Dann schränkt SELinux die Applikation fast gar nicht mehr ein. Hierfür benötigen Sie die booleschen Variablen. Der nächste Abschnitt erläutert ihre Anwendung.

17.1.2 Die booleschen Variablen der Targeted-Policy

Die *Targeted*-Policy weist einige zusätzliche boolesche Variablen auf, die hier erwähnt werden sollen. Wenn Sie hier den Befehl `getsebool -a` aufrufen, wird Ihnen auffallen, dass es sehr viele Variablen gibt, die auf den Namen *disable_trans* enden. Diese Variablen kontrollieren, ob eine Applikation, die von SELinux überwacht werden kann, in ihrer eigenen Domäne läuft oder in der Domäne *unconfined_t*. Nur wenn die Applikation einen Domänenwechsel (*Domänentransition*) durchführt, gelten die Einschränkungen ihrer Policy. Werden diese booleschen Variablen gesetzt, so wird der Domänenwechsel für die entsprechende Applikation verhindert. Möchten Sie zum Beispiel, dass die Bluetooth-Anwendungen nicht von SELinux gesondert überwacht werden, sondern in der Domäne *unconfined_t* betrieben werden, führen Sie den folgenden Befehl aus:

```
setsebool -P bluetooth_disable_trans=1
```

Die Option `-P` stellt sicher, dass die boolesche Variable auch nach dem nächsten Neustart des Systems über diesen Wert verfügt.

17.1.3 Schutz zusätzlicher Applikationen mit SELinux

Vielleicht haben Sie dieses Buch gekauft, weil Sie SELinux im Strict-Modus einsetzen, und sind nun enttäuscht, da sich Teile des Buchs scheinbar nur um die *Targeted-Policy* drehen. Allerdings sollte Ihnen bei dem Lesen der Einleitung dieses Kapitels bereits aufgefallen sein, dass es eigentlich keinen Unterschied zwischen der Targeted- und der *Strict-Policy* gibt. Jeder Prozess wird von SELinux mit *Type-Enforcement*-Regeln überwacht. Nun gut, bei der Targeted-Policy laufen viele Prozesse in der Domäne *unconfined_t*. Diese Domäne wird aber dennoch überwacht. Sie benötigt explizite Allow-Regeln, damit die Zugriffe erlaubt werden.

Wenn Sie nun die Policy eines vorhandenen Dienstes anpassen möchten, gehen Sie genauso vor, wie in dem entsprechenden Kapitel beschrieben. Häufig kann sogar mit Anpassungen ein und dasselbe Policy-Package sowohl in der Targeted- als auch in der Strict-Policy verwendet werden.

17.2 Die Strict-Policy

Die *Strict*-Policy ist die »wahre« SELinux-Policy. Hier wird jeder Prozess in einer Domäne überwacht. Dabei stellen die unterschiedlichen Domänen sicher, dass jeder Prozess nur die erforderlichen Privilegien erhält (Prinzip des *Least-Privilege*). Dies führt in vielen Fällen dazu, dass die Applikationen, abweichend von ihrem Standard eingesetzt, nicht korrekt funktionieren, da SELinux die Funktion unterbindet. Weil dies in vielen Fällen zu Problemen führt, wurde die *Targeted*-Policy entwickelt. Dennoch kann die *Strict*-Policy sinnvoll eingesetzt werden. Es ist sicherlich nicht sinnvoll, die Policy auf einem System zu verwenden, das von mehreren Benutzern genutzt wird. Ein klassisches Desktop-System lässt sich nur sehr schwer mit der *Strict*-Policy überwachen. Meist sind im Verlauf der Verwendung doch wesentliche Anpassungen erforderlich. Jedoch erlaubt die *Strict*-Policy seit Fedora Core 5 eine grafische Anmeldung der normalen Benutzer über den `gdm` mit der GNOME-Oberfläche. Der Benutzer *root* kann sich nicht grafisch anmelden, wenn das System sich im *Enforcing*-Modus befindet.

17.2.1 Was unterscheidet die Strict-Policy?

Erste Unterschiede habe ich bereits in der Einleitung beschrieben. Am einfachsten erkennen Sie die Unterschiede, wenn Sie sich an einem SELinux-System mit Strict-Policy im Enforcing-Modus als *root* anmelden. Die folgenden Beispiele wurden auf einem Fedora Core 6-System durchgeführt.

Wenn Sie sich lokal anmelden, sieht das folgendermaßen aus:

```
Fedora Core release 6 (Zod)
Kernel 2.6.20-1.2944.fc6 on an i686

supergrobi login: root
Password:

[root@supergrobi ~]# id -Z
root:sysadm_r:sysadm_t
```

Wenn Sie sich mit der *Secure-Shell* anmelden, ergibt sich folgendes Bild:

```
$ ssh root@supergrobi
root@supergrobi's password:
Last login: Sun Apr 22 11:10:11 2007
-bash: /root/.bash_profile: Keine Berechtigung
-bash-3.1# id -Z
root:staff_r:staff_t
```

Wie Sie hoffentlich leicht erkennen können, besitzen Sie nach der Anmeldung lokal und remote unterschiedliche Rollen. Dies hat eine große Auswirkung auf die weitere Funktion. Am deutlichsten wird das bei dem Befehl `ps`. Wenn Sie mit `ps -ef` sämtliche Prozesse anzeigen, dann sehen Sie in der Rolle *staff_r* nur die eigenen Prozesse:

```
-bash-3.1# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root      2837     1  0 08:52 ?        00:00:00 /usr/libexec/gam_server
root      3154     1  0 11:06 ?        00:00:00 /usr/bin/esd -terminate ➥
          -nobeeps
root      3760  3756  0 11:13 pts/0    00:00:00 -bash
root      3787  3760  0 11:14 pts/0    00:00:00 ps -ef
```

In der Rolle *sysadm_r* können Sie dagegen alle Prozesse sehen[1]. Auch der Zugriff auf das Verzeichnis des Benutzers *root* ist unterschiedlich. In der Rolle *staff_r* dürfen Sie die Dateien in dem Verzeichnis noch nicht einmal lesen. Dies erklärt auch die Fehlermeldung und den ungewöhnlichen Prompt bei der Anmeldung über die SSH. In der Rolle *sysadm_r* können Sie die Dateien lesen und schreiben.

Um die Rolle zu wechseln, können Sie den Befehl `newrole` verwenden. Nach der Anmeldung können Sie also trotzdem die Rolle *sysadm_r* einnehmen:

```
-bash-3.1# newrole -r sysadm_r
Authentifiziere root.
Passwort:
[root@supergrobi ~]#
```

Hierbei verlangt der Befehl erneut das Kennwort des Benutzers *root*. Dies erlaubt es übrigens nicht einem normalen Benutzer, die Funktion von *root* zu übernehmen:

```
$ ssh benutzer@supergrobi
benutzer@supergrobi's password:
[benutzer@supergrobi ~]$ id -Z
user_u:user_r:user_t
[benutzer@supergrobi ~]$ newrole -r sysadm_r
"user_u:sysadm_r:sysadm_t" ist kein gueltiger Kontext
[benutzer@supergrobi ~]$ su -
Passwort:
[root@supergrobi ~]# id -Z
user_u:user_r:user_t
[root@supergrobi ~]# newrole -r sysadm_r
"user_u:sysadm_r:sysadm_t" ist kein gueltiger Kontext
```

Obwohl der Benutzer mit `su` den Linux-Benutzer gewechselt hat, kann er nicht die Rolle wechseln. Vielmehr kennt SELinux immer noch den originalen Benutzer. So schützt SELinux bei Verwendung der *Strict-Policy* das System auch vor den angemeldeten Benutzern. Dies ist bei der *Targeted-Policy* nicht der Fall.

[1] Wenn bei Ihnen dieser Unterschied nicht zu erkennen ist, prüfen Sie, ob SELinux sich im Enforcing-Modus befindet.

17.2.2 Die booleschen Variablen der Strict-Policy

Die Strict-Policy weist einige Variablen auf, die in der Targeted- Policy nicht vorhanden sind. Eigentlich weist die Strict-Policy aber wesentlich weniger boolesche Variablen als die Targeted-Policy auf. Die Strict-Policy erlaubt nicht die Deaktivierung der Überwachung für einzelne Dienste, wie die Targeted-Policy es tut. Die wenigen zusätzlichen booleschen Variablen sollen kurz erläutert werden.

- `allow_httpd_staff_script_anon_write`: Diese und die folgenden beiden booleschen Variablen definieren, ob die entsprechenden ausgeführten *CGI*-Scripts öffentliche Daten (*public_content_t*) schreiben dürfen.
- `allow_httpd_sysadm_script_anon_write`
- `allow_httpd_user_script_anon_write`
- `secure_mode`: Diese boolesche Variable hat mehrere Funktionen:
  - Ein Wechsel mit `newrole` ist nur für unprivilegierte Rollen möglich. Aus der Rolle *staff_r* ist kein Wechsel mehr in die Rolle *sysadm_r* möglich. Damit kann man bei Anmeldung über die SSH *root* keine administrativen Änderungen mehr vornehmen.
  - Ein Wechsel mit `su` ist ebenfalls nur für nicht- privilegierte Benutzer möglich.
  - Ein Wechsel über `userhelper` ist auch nur dann möglich, wenn *secure_mode* abgeschaltet ist.

17.2.3 Schutz zusätzlicher Applikationen mit SELinux

Der Schutz zusätzlicher Applikationen mit SELinux ist auch bei der Strict-Policy sehr einfach. Eigentlich erfolgt die Erzeugung der Richtlinien identisch. Sie müssen hier nur darauf achten, dass Sie die richtigen Domänen und Rollen in der Richtlinie angeben. Dies soll kurz am Beispiel des Kommandos `date` in Kapitel 28 gezeigt werden.

17.2.4 Neue Rollen zur Delegation der Root-Fähigkeiten

Es ist sehr leicht möglich, die vorhandenen Rollen der Strict-Policy um weitere Rollen zu erweitern, die dann den Zugriff auf bestimmte Funktionen erlauben. So können administrative Aufgaben an Benutzer vergeben werden, die dann diesen Teilaspekt der Systemadministration übernehmen. In Kapitel 28.3 wird das erläutert.

17.3 Die MLS-Policy

Multi Level Security (*MLS*) war bereits Bestandteil von vielen Trusted-Operating-Systemen. Auch SELinux enthielt MLS lange als experimentellen Bestandteil. Dies war aber für viele Jahre nicht wirklich verwendbar. Mit dem Linux-Kernel 2.6.12 wurde ein neues MLS-Konzept in dem Kernel implementiert und erstmals eine funktionstüchtige MLS-Policy ausgeliefert. Die wesentliche Arbeit wurde dabei von der Firma Trusted Computer Solutions, Inc.[2] geleistet. Diese Firma vertreibt nun Produkte, die auf der MLS-Technologie basieren.

[2] *http://www.trustedcs.com*

Das *Type-Enforcement*-(TE-)Sicherheitsmodell in SELinux ist sehr mächtig und kann an viele Anforderungen angepassst werden. Dabei ist TE sehr flexibel und erlaubt die Konfiguration sehr fein granulierter Mandatory-Access-Control-Mechanismen. TE bietet dabei ein Modell für den kontrollierten Zugriff auf Dateien und die sichere Ausführung von Applikationen. So stellt es die Integrität des Systems sicher und kann optimal nicht-hierarchische Strukturen überwachen.

MLS hat dagegen als wesentliches Ziel die *Vertraulichkeit* der Daten. Dabei werden die Daten meist in hierarchischen Strukturen verwaltet. Diese Funktion kann mit reinen TE-Methoden nicht oder nur sehr schlecht umgesetzt werden. Optimal ist eine Mischung, wenn die Vertraulichkeit der Daten garantiert werden muss.

Die vorhandene Unterstützung im Kernel war sehr unflexibel. So musste bis Kernel 2.6.11 zum Zeitpunkt der Übersetzung der Benutzer bereits entscheiden, ob SELinux MLS unterstützen sollte. Dies wurde mit dem Kernel 2.6.12 modifiziert, sodass lediglich die Policy diese Entscheidung trifft. Dazu wurde die Policy-Sprache erweitert. MLS nutzt nun Constraints: `mlsconstrain`. Hiermit werden die Zugriffe modelliert:

```
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 ));
```

17.3.1 Labeling der Objekte

Jedes Objekt auf dem System benötigt einen MLS *Label*. Dieser kann entweder explizit von der Policy vergeben werden oder durch den Kernel impliziert werden. Die meisten Objekte eines Systems erhalten den Label *SystemLow*. Hierbei handelt es sich um alle Binärdateien, Bibliotheken etc. Lediglich Dateien, deren direkter Zugriff eine Umgehung der granulären Sicherheitsbestimmungen erlauben würde, werden mit *SystemHigh* gelabelt.

So erhält die Datei `/etc/shadow` den Label *SystemLow* während die Datei `/dev/mem` den Label *SystemHigh* erhält.

Subjekte erhalten üblicherweise einen Bereich zugewiesen, z.B. *SystemLow-SystemHigh*. Dieser Bereich besteht immer aus zwei Komponenten: der niedrigen und der hohen (Freigabe-)Stufe. Die hohe Stufe beschreibt, auf welche Daten zugegriffen werden darf.

17.3.2 MLS in der Praxis

Zunächst unterscheidet sich die MLS-Policy kaum von der Strict-Policy. Die Unterschiede treten nur bei dem Zugriff auf Dateien mit dem Label *SystemHigh* auf.

Mit der MLS-Policy gelang es Red Hat, die Zertifizierung des RHEL 5 nach dem *Common-Criteria*Standard zu erhalten. Nachdem die Vorgänger-Version Red Hat Enterprise Linux 4 (RHEL4) Anfang 2006 die international anerkannte Sicherheitszertifizierung *EAL4+* erhielt, erfüllt nun auch das im März 2007 erschienene RHEL5 die Kriterien für EAL4+ sowie für Labeled Security Protection Profile (*LSPP*) nach dem Common-Criteria- Standard. Ebenfalls dazu gehören die rollenbasierte Zugangskontrolle (role based access control), *RBAC*), die die Zugriffsrechte des Super-Users *root* einschränkt. Die Zertifizierung gilt für den Einsatz von RHEL5 auf Server-Systemen von IBM (System x, System p5, System z sowie eServer). Sie ist zudem um *ALC_FLR.3* (Flaw Remediation) erweitert worden.

18 SELinux-Kommandos

In diesem Kapitel finden Sie eine komplette Aufstellung der Kommandos von Debian Etch, Fedora Core 3-6 und RHEL 4 und 5. Einige Kommandos stehen allerdings nur unter den neueren Versionen zur Verfügung. Mir ist bewusst, dass es sicherlich einigen Lesern wie eine Übersetzung der Manpages erscheint. Dennoch halte ich es für wichtig, die Befehle aufzuführen und zu erläutern, da aufgrund des neuartigen Rechtekonzepts ansonsten häufig Missverständnisse bei dem Lesen der Manpages auftreten.

18.1 apol

apol ist ein grafisches Werkzeug, mit dem Sie die SELinux-Policy analysieren können. Mit diesem Werkzeug können Sie die Policy- Komponenten (Type, Role, User etc.), Regeln (TE, *RBAC*, *MLS*) und Kontexte betrachten und Domänentransitionen,

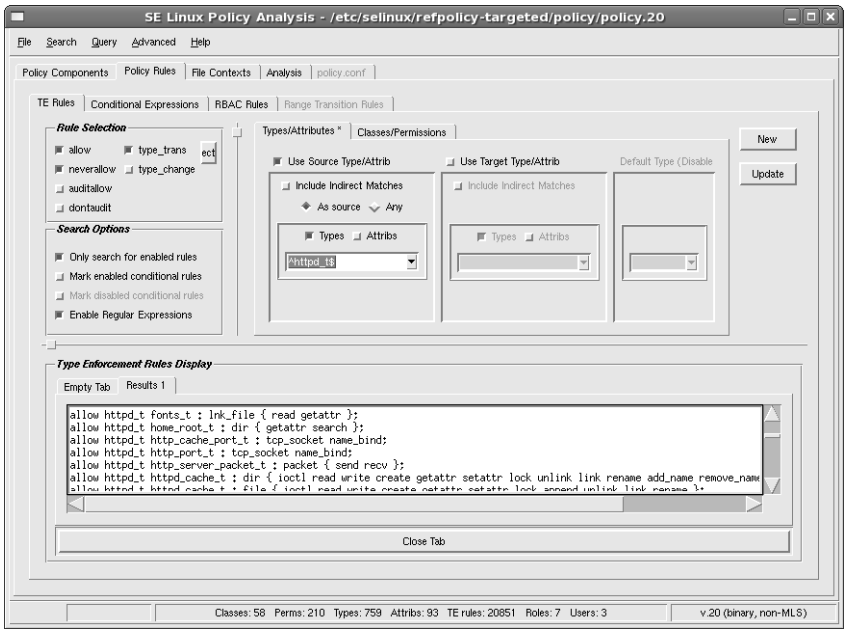

Abbildung 18.1: Mit apol können Sie die SELinux-Richtlinie analysieren.

den *Informationsfluss* und Relabel-Berechtigungen prüfen. Abbildung 18.1 zeigt das Werkzeug in seiner Funktion. Um apol verwenden zu können, müssen Sie als Erstes eine Policy laden. Die weitere Verwendung wird in Kapitel 20 genauer betrachtet.

18.2 avcstat

Das Kommando `avcstat` gibt Ihnen statistische Informationen über den *Access-Vector-Cache* aus (siehe Kapitel 11.2). Der Access-Vector-Cache speichert die Ergebnisse der Evaluation des Regelwerks ab. Mit diesem Befehl können Sie prüfen, wie viele Anfragen bereits an den AVC gestellt wurden und wie viele Anfragen der AVC aus seinem Cache beantworten konnte.

```
[root@supergrobi ~]# avcstat
   lookups       hits     misses     allocs   reclaims      frees
 267881146  267874309       6837       6837       6304       6328
```

Bei Angabe eines Sekundenintervalls zeigt der Befehl ähnlich den Befehlen `vmstat` oder `iostat` zunächst die Werte seit dem Bootzeitpunkt und dann in dem entsprechenden Intervall die Werte des letzten Intervalls an:

```
[root@supergrobi ~]# avcstat 5
   lookups       hits     misses     allocs   reclaims      frees
  90042834   90034745       8089       8089       7056       7586
     22346      22346          0          0          0          0
     19716      19716          0          0          0          0
```

Mit der Option `-c` erhalten Sie immer die Addition der Werte.

18.3 audit2allow

Dieser Befehl wird eines Ihrer wichtigsten Hilfsmittel sein. Mit diesem Befehl können Sie SELinux-Protokollmeldungen direkt in die entsprechenden *Type-Enforcement*-Regeln umwandeln. Dieser Befehl unterstützt Sie also bei der Entwicklung und Anpassung von SELinux-Richtlinien. Dieser Befehl unterstützt je nach eingesetzter SELinux-Version unterschiedliche Optionen. Einige Optionen stehen nur bei Verwendung der Reference-Policy zur Verfügung. Dies habe ich an den entsprechenden Stellen vermerkt.

Zunächst können Sie mit diesem Werkzeug die Protokollmeldungen aus einer Datei einlesen und in *allow*-Regeln umwandeln. Hierzu verwenden Sie:

```
[root@supergrobi ~]# audit2allow -i /var/log/messages
allow dhcpc_t etc_t:file write;
```

Achtung

Auf einem *Debian*System befinden sich die Meldungen in der Datei `/var/log/syslog`!

Allerdings bietet das Werkzeug auch noch weitere Optionen an:

- `-a,--all`: Diese Option liest die Meldungen sowohl aus der Datei `/var/log/messages` als auch aus der Datei `/var/log/audit/audit.log` ein. Diese Option steht erst ab Version FC 5 zur Verfügung.
- `-d,--dmesg`: Liest die Meldungen über `dmesg` ein.
- `-f,--fcfile <datei>`: Hiermit können Sie eine File-Context-Datei angeben. Gleichzeitig muss die Option `-M` angegeben werden (Reference-Policy).
- `-h,--help` Gibt eine kurze Hilfe aus.
- `-i,--input <datei>`: Liest die Meldungen aus der angegebenen Datei.
- `-l,--lastreload`: Liest lediglich die Meldungen seit dem letzten *Reload* der Policy ein.
- `-m,--module <modulname>`: Diese Option generiert die Syntax für ein Policy-*Modul* (Reference-Policy).
- `-M <modul>`: Dies erzeugt ein fertiges Modul. Dann darf nicht die Option `-o` angegeben werden (Reference-Policy).
- `-o,--output <datei>`: Schreibt die Regeln in die angegebene Datei.
- `-r, --requires`: Erzeugt die *Require*-Syntax für Module. Dies ist bei der Reference-Policy erforderlich, um sicherzustellen, dass die in dem Modul genutzten und in anderen Modulen definierten Typen zur Verfügung stehen.
- `-R, --reference`: Dies erzeugt ein Modul für die Reference-Policy, bei der der `audit2allow`-Befehl vor der Erzeugung der *allow*-Regeln die Interfaces der Reference-Policy durchsucht. Wird hier ein passendes *Interface* gefunden, wird dieses anstelle der *allow*-Regel genutzt.
- `-t, --tefile <datei>`: Hiermit können Sie eine TE-Datei angeben. Dies kann genutzt werden, um alte TE-Dateien in das modulare Format der Reference-Policy zu übertragen.
- `-v,--verbose`: Gibt zusätzliche Kommentare aus, die die Regeln erläutern.

18.4 audit2why

Dieses Kommando liest wie `audit2allow` die SELinux-Protokollmeldungen ein und versucht. diese näher zu erklären. Leider handelt es sich dabei jedoch immer nur um generische Erläuterungen. Daher ist dieses Programm eigentlich überflüssig:

```
[root@bibo ~]# audit2why < /var/log/messages
Sep  5 08:28:43 bibo kernel: audit(1157437723.096:2): avc:  denied ↵
          write  for  pid=2228 comm="dhclient-script" name= ↵
         "resolv.conf" dev=dm-0 ino=5180835 scontext=system_u: ↵
         system_r:dhcpc_t tcontext=root:object_r:etc_t tclass=file
    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 audit2- ↵
         allow with this audit message as input.
```

18.5 chcat

Dieser Befehl erlaubt das Ändern der *MCS-Kategorie* einer Datei oder eines Benutzers. Sie können auch eine Kategorie hinzufügen oder entfernen. Hierzu unterstützt der Befehl die folgenden Optionen:

- `-d`: Löscht eine Kategorie.
- `-l`: Bearbeitet anstelle von Dateien einen Benutzer.
- `-L`: Zeigt die verfügbaren Kategorien an.

Mit diesem Befehl können Sie sehr einfach Kategorien zu einem Benutzer oder einer Datei hinzufügen oder Kategorien entfernen:

```
[root@supergrobi ~]# chcat -- -CompanyConfidential /docs/ ↵
         businessplan.odt
[root@supergrobi ~]# chcat -l +CompanyConfidential juser
```

Die Kategorie kann mit `+` hinzugefügt und mit `-` entfernt werden. Wenn Sie `-Kategorie` verwenden möchten, müssen Sie durch Angabe von `--` dem Befehl mitteilen, dass es sich nicht um eine Option handelt.

Die Kategorien werden in der Datei `/etc/selinux/<policy>/setrans.conf` mit Namen versehen. Diese Datei können Sie auch selbst mit dem Befehl `semanage` anpassen.

18.6 chcon

Hiermit ändern Sie den *Security-Context* einer Datei. Dabei können Sie entweder den gesamten Kontext angeben oder bei Angabe einer Option nur einen Teil des Kontextes ändern.

Um nur einen Teil des Kontextes zu ändern, stehen die folgenden Optionen zur Verfügung:

- `-r, --role`: Ändert nur die Rolle.
- `-t, --type`: Ändert nur den Typ.
- `-u, --user`: Ändert nur den SELinux-Benutzer.
- `-l, --range`: Ändert nur die *MLS*-Range.

Um ganze Verzeichnisse einschließlich ihres Inhalts zu bearbeiten, geben Sie die Option `-R, --recursive` an, wie Sie es auch von Befehlen wie `chown` gewohnt sind.

Um die Ausgabe zu beeinflussen, können Sie mit `-f, --silent, --quiet` diese unterdrücken, mit `-v, --verbose` eine diagnostische Ausgabe erzwingen, und mit `-c,--changes` werden die diagnostischen Mitteilungen nur angezeigt, wenn tatsächlich eine Änderung vorgenommen wurde.

Mit der Option `--reference=<file>` geben Sie nicht den zu setzenden *Security-Context* an, sondern `chcon` übernimmt den Kontext der angegebenen Referenzdatei. Die Option `-h, --no-dereference` erlaubt es, den Kontext einer symbolischen Verknüpfung (Symlink) anstelle der referenzierten Datei zu setzen.

Um für ein Verzeichnis einen neuen Kontext zu setzen und die veränderten Dateien angezeigt zu bekommen, verwenden Sie:

```
[root@supergrobi~]# chcon -c -r -t httpd_sys_content_t /var/www-new/
```

18.7 checkmodule

Der Befehl `checkmodule` prüft die Syntax und kompiliert ein SELinux-Policy-Modul. Hierbei kann der Befehl sowohl ein Base-Modul als auch ein nachladbares *Modul* erzeugen. Jede Policy besteht aus genau einem *Base-Modul* und mehreren weiteren Modulen. Das entstandene Policy-Modul kann dann mit dem Befehl `semodule_package` in ein Policy-Paket (Endung `.pp`) umgewandelt werden und mit dem Befehl `semodule` installiert werden.

Der Befehl unterstützt die folgenden Optionen:

- `-b`: Hiermit liest der Befehl zu Debugging-Zwecken anstelle einer TE-Datei ein binäres Policy-Modul ein.
- `-d`: Dies aktiviert den Debug-Modus.
- `-m`: Hiermit generieren Sie ein Nicht-Base-Modul.

- `-M`: Dies schaltet *MLS/MCS*-Unterstützung bei der Erzeugung des Moduls ein.
- `-o <datei>`: Das Modul wird in diese Datei geschrieben. Wenn Sie keine Datei angeben, überprüft der Befehl lediglich die Syntax und erzeugt kein Modul.

Um ein Modul aus einer *Type-Enforcement*-Datei zu bauen, verwenden Sie:

```
[root@supergrobi ~]# checkmodule -m httpd-add.te -o httpd-add.mod
```

18.8 checkpolicy

Dies ist der SELinux-*Policy!-Compiler*. Hiermit wird die SELinux-Policy in ihre binäre Form übersetzt. Bei den modernen modularen Richtlinien mit der Reference-Policy benutzen Sie diesen Befehl allerdings nicht mehr direkt.

18.9 fixfiles

Mit diesem Script können Sie einfach den *Security-Context* vieler Dateien auf Ihrem System korrigieren. Hierzu lädt das Script den Default-Kontext der Dateien aus der Policy und setzt den Kontext der analysierten Dateien entsprechend. Speziell RPM-basierte Distributionen profitieren von der Option `-R`, mit der nur die Dateien bestimmter RPM-Pakete mit dem korrekten Security-Context versehen werden.

Das Script kennt vier verschiedene Befehle:

- `check`: Hiermit zeigt das Script nur alle abweichenden Kontexte an, aber ändert diese nicht.
- `restore`: Hiermit werden alle fehlerhaften Kontexte repariert.
- `relabel`: Dieser Befehl führt ein *Relabeling* durch und löscht vorher das Verzeichnis `/tmp`.
- `verify`: Hiermit zeigt das Script alle Dateien mit fehlerhaftem Kontext an.

Einige Optionen verändern das Verhalten des Befehls:

- `-l <logdatei>`: Die Ausgaben werden in dieser Protokolldatei gespeichert.
- `-o <datei>`: Alle Dateien, deren Kontext sich unterscheidet, werden in dieser Datei gespeichert.
- `-F`: Dateien mit einem *customizable_type* werden auch zurückgesetzt.
- `-f`: Vor dem Löschen des Verzeichnisses `tmp` erfolgt keine Nachfrage.
- `-R <rpmpaket>`: Nur die Dateien des RPM-Pakets werden geprüft. Leider funktioniert dies bisher noch nicht für Debian-Pakete.
- `-C <fc-datei>`: Das Script führt einen Diff der angegebenen Datei mit der aktuellen Datei durch und ändert nur entsprechende Dateien.

Um alle Dateien in dem Pfad `/usr/bin` zu überprüfen, verwenden Sie:

```
[root@supergrobi ~]# fixfiles verify /usr/bin
```

18.10 genhomedircon

Dieser Befehl wird eigentlich nicht von Ihnen manuell, sondern bei der Übersetzung der Policy automatisch aufgerufen. Er erzeugt die Konfigurationseinträge in der Policy, die für die Benutzer und ihre Heimatverzeichnisse benötigt werden. Hierzu verwendet der Befehl das Präfix, das bei dem Befehl `semanage` für den Benutzer angegeben wurde. Daher wird dieser Befehl auch jedes Mal aufgerufen, wenn mit `semanage` die SELinux-Benutzer verändert werden.

Hierzu verwendet der Befehl als Template die Datei `contexts/files/homedir_template`. Da Sie diesen Befehl nie benötigen werden, verweise ich für alle weiteren Informationen auf die Manpage.

18.11 getenforce

Mit dem Kommando `getenforce` können Sie prüfen, in welchem Zustand sich das SELinux-System in diesem Moment befindet. Es liefert als Ergebnis entweder

- *Permissive*: SELinux ist aktiv, aber erzwingt nicht die Durchsetzung der Richtlinien,
- *Enforcing*: SELinux erzwingt das Einhalten der Richtlinien oder
- *Disabled*: SELinux ist abgeschaltet.

Für den Einsatz in Scripts eignet es sich weniger. Hier ist das Kommando `selinuxenabled` besser geeignet (siehe Abschnitt 18.24).

18.12 getsebool

Mit dem Befehl `getsebool` lesen Sie die aktuellen Werte der booleschen Variablen aus. Entweder geben Sie die Option `-a` an, um alle Werte zu erfahren, oder Sie fragen spezifisch nach dem Wert einer Variablen:

```
# getsebool allow_kerberos
allow_kerberos --> on
```

In einigen Fällen wird `getsebool` einen *Pending*-Wert ausgeben:

```
# getsebool allow_kerberos
allow_kerberos --> on pending: off
```

Dies erfolgt immer dann, wenn für eine Variable bereits ein neuer Wert bestimmt worden ist, dieser aber noch nicht aktiviert wurde. Um mehrere Änderungen der booleschen Variablen gleichzeitig zu ermöglichen, können Sie die Werte zunächst laden und dann gleichzeitig aktivieren (siehe Kapitel 15).

18.13 load_policy

Mit diesem Befehl wird eine neue Policy in den Kernel geladen. Normalerweise können Sie das auch mit dem Befehl `semodule -R` erreichen. Jedoch haben Sie dann nicht die Möglichkeit, die folgenden Entscheidungen zu fällen:

- `-b`: Mit dieser Option werden die booleschen Variablen auf den in der Policy gesetzten Wert initialisiert. Wird diese Option nicht angegeben, bleiben die aktuell gesetzten Werte erhalten.
- `-q`: Hiermit unterdrücken Sie Warnmeldungen bei dem Laden der Policy.

18.14 matchpathcon

Mit diesem Befehl können Sie den Default-*Security-Context* einer Datei ermitteln. Dies können Sie gut zur Fehlersuche einsetzen. Hierzu rufen Sie einfach den folgenden Befehl auf:

```
[root@supergrobi ~]# matchpathcon /usr/sbin/sealert
/usr/sbin/sealert       system_u:object_r:sbin_t
```

Der Befehl sucht nach der angegebenen Datei in der Datei `file_contexts` der aktuell verwendeten SELinux-Policy. Wenn Sie eine abweichende Datei `file_contexts` zur Ermittlung verwenden möchten, können Sie diese mit der Option `-f <datei>` angeben. Mit der Angabe des Präfix der Datei kann die Suche beschleunigt werden (`-p <prefix>`). Mit der Option `-n` unterdrücken Sie die Ausgabe des Dateinamens. Die Option `-N` unterdrückt die Übersetzung der *MLS*-Security-*Level* und zeigt diese numerisch an:

```
[root@supergrobi ~]# matchpathcon -n -N /usr/sbin/sealert
system_u:object_r:sbin_t:s0
```

Schließlich können Sie mit der Option `-V` auch prüfen, ob die Datei auch den Default-Kontext aufweist (Verify).

```
[root@supergrobi ~]# matchpathcon -V /usr/sbin/sealert
/usr/sbin/sealert verified.
```

18.15 newrole

Mit diesem Befehl starten Sie eine neue Shell in einem neuen Kontext. Der neue Kontext wird dabei von dem alten Kontext abgeleitet. Mit der Option `-r, --role` können Sie eine andere Rolle und mit `-t, --type` einen anderen Typ einnehmen. Wird nur eine Rolle angegeben, so wird der Default-Typ der Rolle verwendet. Mit der Option `-l, --level` kann auch die Sicherheitsstufe (Security-Level) ausgewählt werden.

Üblicherweise verwenden Sie diesen Befehl zum Beispiel als *root*, um beim Einsatz der Strict-Policy aus der Rolle *staff_r* in die Rolle *sysadm_r* zu wechseln. Hierbei überprüft dann der Befehl, ob der Wechsel erlaubt ist, und fordert bei Bedarf ein Kenn-

wort des Benutzers an. Der *Rollenwechsel* wird durch das Beenden der Shell rückgängig gemacht. Insofern ähnelt der Befehl `newrole` dem Befehl `su`.

Wenn Sie weitere Optionen an die aufgerufene Shell übergeben wollen, können Sie das nach `--` machen. Um zum Beispiel nur einen Befehl einzugeben, können Sie die folgenden Zeilen nutzen:

```
[root@supergrobi ~]# newrole -r sysadm_r -- -c setenforce 0
```

18.16 restorecon

restorecon Mit diesem Befehl können Sie den *Security-Context* von einer oder mehreren Dateien wiederherstellen oder prüfen. Dieses Werkzeug wird auch von dem Script `fixfiles` (siehe Abschnitt 18.9) verwendet.

Achtung

Der Befehl `restorecon` verfolgt keine symbolischen Verknüpfungen (Symlinks).

Der Befehl kennt einige Optionen, die sein Verhalten beeinflussen:

- `-e <verzeichnis>`: Dieses Verzeichnis wird von der Verarbeitung ausgenommen.
- `-F`: Dies erzwingt auch Änderungen an Dateien, deren Kontext als *customizable_type* normalerweise geschützt ist.
- `-f <datei>`: Der Befehl liest die zu verarbeitenden Dateien aus der Datei ein. Mit `-` liest er sie von der Standardeingabe. Dies kann gut in einer Pipe verwendet werden.
- `-i`: Mit dieser Option ignoriert der Befehl nicht-existierende Dateien.
- `-n`: Hiermit führt der Befehl lediglich eine Überprüfung durch und verändert keinen Kontext.
- `-o <datei>`: Alle Dateinamen, die nicht den richtigen Kontext aufweisen, werden in dieser Datei gespeichert.
- `-R, -r`: Rekursive Verarbeitung.
- `-v`: Der Befehl zeigt die Veränderungen des Typs oder der Rolle an.
- `-vv`: Hiermit zeigt der Befehl auch Veränderungen des SELinux-Benutzers an.

18.17 run_init

Die meisten Dienste auf dem Linux-System werden von dem SysVInit-Daemon gestartet. Die SELinux-Policys berücksichtigen dies und erlauben der Domäne *initrc_t* den Start der Prozesse und den Wechsel in die neuen prozessspezifischen Domänen. Wenn *root* ebenfalls einen derartigen Dienst starten möchte, muss sichergestellt sein, dass der Dienst ebenfalls über die Domäne *initrc_t* gestartet wird. Dies erledigt dieser Befehl.

Wenn Sie einen Dienst starten oder stoppen möchten, so verwenden Sie folgenden Befehl:

```
[root@supergrobi ~]# run_init /etc/init.d/named start
```

Achtung

Bei den meisten von der Distribution mitgelieferten *Targeted-Policy*-Richtlinien ist das für die hier enthaltenen Dienste nicht erforderlich. Wenn Sie aber selbst einen Dienst hinzufügen oder eine *Strict-Policy* einsetzen, benötigen Sie diesen Befehl.

18.18 sealert

Dieser Befehl steht auf Fedora Core 6, RHEL 5 und neueren Distributionen zur Verfügung. Er stellt das Frontend für den `setroubleshoot`-Daemon dar. Hiermit kön-

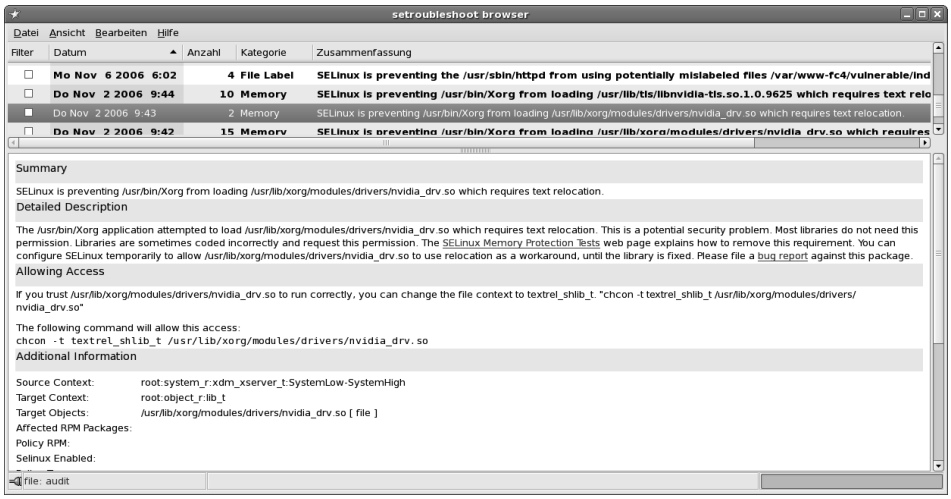

Abbildung 18.2: sealert weist auf ein Problem mit dem Nvidia-Grafikkartentreiber hin.

nen Sie die Meldungen des seTroubleShoot-Daemons analysieren. Damit hilft dieses Werkzeug Ihnen bei der Anpassung Ihrer Policy und der Diagnose der Fehlermeldungen. Abbildung 18.2 zeigt eine typische Analyse durch `sealert`.

Wenn Sie Sealert auf der Kommandozeile aufrufen möchten, sollten Sie die Option `-b` verwenden, um den grafischen Browser zu starten. Die weitere Optionen sind:

- `-b, --browser`: Start des grafischen Browsers
- `-H, --html_output`: HTML-Ausgabe
- `-s, --service`: Start als DBUS-Dienst
- `-S, --noservice`: Start als Standalone-Applikation
- `-l, --lookup <id>`: Sucht nach Alarmmeldungen über ihre ID-Nummer.
- `-a, --analyze <file>`: Lädt die angegebene Protokolldatei.

18.19 seaudit

Das Kommando `seaudit` dient der einfacheren Analyse der Protokolldateien. Das Programm liest die Protokolldateien `/var/log/audit/audit.log` und `/var/log/messages` und stellt ihre Meldungen übersichtlich dar. Alternative Protokolldateien[1] können Sie mit dem Parameter `--log` auf der Kommandozeile angeben. Für die richtige Funktion müssen Sie auch die Policy laden. Auch diese können Sie auf der Kommandozeile angeben (`--pol`).

Sie können in diesen Meldungen suchen und Filter definieren (siehe Abbildung 18.3). Kapitel 14 erläutert die Anwendung des Programms.

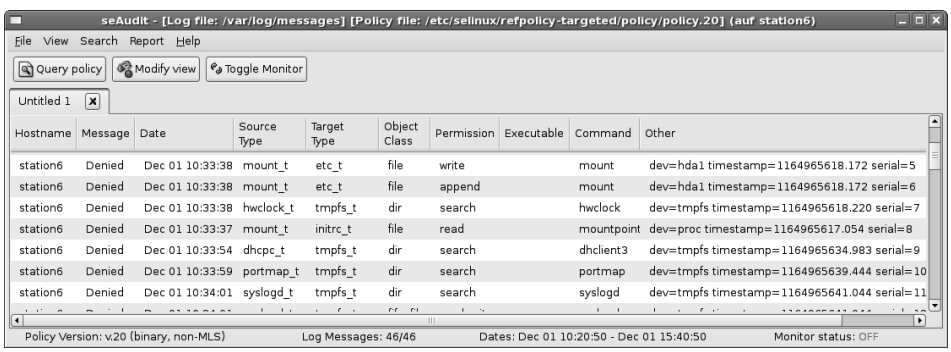

Abbildung 18.3: Mit seaudit können Sie die Protokollmeldungen komfortabel browsen.

[1] Für Debian müssen Sie die Datei `/var/log/syslog` angeben.

18.20 seaudit-report

Der Befehl `seaudit-report` dient Ihnen zur Erzeugung eines Berichtes. Dieser Bericht fasst die SELinux-Protokollmeldungen zusammen. Die Anwendung dieses Befehls wird in Kapitel 14 genau beschrieben.

Für seine Funktion unterstützt der Befehl die folgenden Optionen:

- `-s, --stdin`: Einlesen der Protokollmeldungen von der Standardeingabe
- `-m, --malformed`: Aufnahme ungültiger Meldungen in den Bericht
- `-o, --output <datei>`: Ausgabe des Berichtes in diese Datei
- `-c, --config <datei>`: Konfigurationsdatei
- `--html` Ausgabe in HTML
- `--stylesheet`: CSS-Stylesheet. Dieses wird für die Formatierung der HTML-Ausgabe verwendet. Sie können sich nach dem mitgelieferten Stylesheet[2] richten.
- `-v, --version`: Ausgabe der Programmversion

18.21 sediff

Mit dem Kommando `sediff` können Sie zwei SELinux-Policys semantisch miteinander vergleichen. Der Befehl zeigt dabei die Unterschiede der beiden Policys an. Der Befehl wird genauer bei dem Upgrade der Policys beschrieben (siehe Kapitel 21). Die folgenden Parameter verändern die Ausgabe des Befehls:

- `-c, --classes`: Objektklassen
- `-t, --types`: Typen
- `-a, --attributes`: Attribute
- `-r, --roles`: Rollen
- `-u, --users`: SELinux-Benutzer
- `-b, --booleans`: Boolesche Variablen
- `-T, --terules`: Type-Enforcement-Regeln
- `-R, --roletrans`: Rollen-Transitionen
- `-A, --roleallows`: Rollenwechsel
- `-C, --conds`: Bedingungen
- `-q, --quiet`: Nur Unterschiede werden angezeigt
- `-s, --stats`: Statistische Anzeige

18.22 sediffx

Dieser Befehl ist eine grafische Variante des Befehls `sediff`. Auch dieser Befehl wird in Kapitel 21 genauer betrachtet.

[2] `/usr/share/setools/seaudit-report.css`

18.23 seinfo

Dieser Befehl ist auf älteren Fedora Core 3- und 4- und RHEL 4-Systemen nicht vorhanden. Mit diesem Befehl können Sie sich die Informationen zu bestimmten Komponenten einer Policy anzeigen lassen. Dieser Befehl kann sowohl eine binäre Policy als auch eine Policy im Quelltext verarbeiten und schnell die gewünschten Informationen liefern.

Die Manpage führt die vielen verschiedenen Optionen auf. Um zum Beispiel die in der Policy definierten Benutzer aufzuführen, verwenden Sie:

```
# seinfo -u

Users: 3
   system_u
   root
   user_u
```

Um anschließend die Rollen des Benutzers angezeigt zu bekommen, können Sie die Option `-x` hinzufügen:

```
# seinfo -uroot -x
   root
      roles:
         system_r
         sysadm_r
         user_r
```

18.24 selinuxenabled

Dies ist ein einfaches kleines Kommando, das lediglich prüft, ob SELinux im Kernel aktiviert ist oder nicht. Dieses Werkzeug ist für den Einsatz in Scripts gedacht, die den Zustand von SELinux im Kernel ermitteln müssen. Das Kommando hat zwei verschiedene Rückgabewerte:

- `0`: SELinux ist aktiv. Dies kann sowohl im *Enforcing* als auch im *Permissive*Modus sein.
- `1`: SELinux ist nicht aktiv (*Disabled*).

18.25 semanage

Dies ist der zentrale Befehl zur Verwaltung der SELinux-*Reference-Policy*. Mit ihm und mit dem Befehl `semodule` (siehe Abschnitt 18.26) führen Sie alle administrativen Tätigkeiten durch. Dabei können wesentliche Bestandteile der Policy mit diesem Befehl erweitert werden, ohne dass, wie bei der *Example-Policy*, eine erneute Übersetzung der gesamten Policy erforderlich ist.

Sie können mit diesem Befehl SELinux-Benutzer anlegen, die Zuordnung der Linux- und SELinux-Benutzer definieren und Security-Contexts für Dateien, Ports etc. erweitern.

Hierzu unterstützt der Befehl eine Vielzahl von Optionen, die zunächst vorgestellt werden sollen. Im Anschluss werden einige Beispiele der Anwendung gezeigt.

- `-a, --add`: Hiermit können Sie ein Objekt zur Policy hinzufügen. Ein Objekt ist entweder ein Linux-Benutzer, ein SELinux-Benutzer, eine Datei oder ein Port.
- `-d, --delete`: Hiermit löschen Sie ein Objekt.
- `-f, --ftype`: Diese Option gibt den File-Type an.
- `-l, --list`: Hiermit zeigen Sie die Objekte an.
- `-L, --level`: Hiermit geben Sie einen Security-Level an.
- `-m, --modify`: Dies ändert ein Objekt.
- `-n, --noheading`: Dies unterdrückt die Ausgabe der Titelzeile im Listing.
- `-p, --proto`: Hiermit gegeben Sie das Protokoll und den Port an.
- `-r, --range`: Hiermit geben Sie einen MLS-Bereich an.
- `-R, --role`: Dies erlaubt die Angabe der Rolle. Wenn es mehrere Rollen gibt, müssen Sie diese in Anführungszeichen setzen.
- `-P, --prefix`: Hiermit können Sie bei Benutzern ein Präfix angeben, das den Typen *home_dir_t* und *home_t* vorangestellt wird.
- `-s, --seuser`: Dies ist der SELinux-Benutzername.
- `-t, --type`: Dies ist der Typ des Objekts.
- `-T, --trans`: Dies ist eine SELinux-Translation.

Zusätzlich zu diesen Optionen verlangt der Befehl `semanage` auch die Angabe der Objektart, die verwaltet werden soll:

- `login` Diese Objekte weisen einem Linux-Benutzer einen *SELinux!-User* und seinen *MLS/MCS*-Bereich zu.

```
# semanage login -l

Login Name           SELinux User         MLS/MCS Range

__default__          user_u               s0
root                 root                 SystemLow-SystemHigh
```

- `user` Diese Objekte weisen einem SELinux-Benutzer seine möglichen Rollen, das Labeling-Präfix und MLS/MCS-Level und Bereiche zu.

```
# semanage user -l

                Labeling   MLS/       MLS/
SELinux User    Prefix     MCS Level  MCS Range                ➥
          SELinux Roles

root            user       s0         SystemLow-SystemHigh     ➥
          system_r sysadm_r user_r
system_u        user       s0         SystemLow-SystemHigh     ➥
          system_r
user_u          user       s0         SystemLow-SystemHigh     ➥
          system_r sysadm_r user_r
```

- `port`: Hiermit werden einzelnen TCP- und UDP-Ports SELinux-Typen zugewiesen.
```
# semanage port -l
SELinux Port Type              Proto    Port Number

afs_bos_port_t                 udp      7007
afs_fs_port_t                  tcp      2040
...
```
- `interface`: Hiermit können einzelnen Netzwerkkarten Contexts zugewiesen werden. Dies wird aktuell noch nicht von den Richtlinien genutzt.
- `fcontext`: Hiermit können Sie den *Security-Context* einer Datei anzeigen und modifizieren. Dies ist die häufigste Verwendung des Befehls.
```
# semanage fcontext -l | head
SELinux fcontext    type        Context
/.*                 all files   system_u:object_r:default_t:s0
/xen(/.*)?          all files   system_u:object_r:xen_image_t:s0
...
```
- `translation`: Dies zeigt und modifiziert die *SELinux!-Translation*. Eine Translation ist die Übersetzung eines numerischen SELinux-Levels in einen sinnvollen Namen.
```
# semanage translation -l
Level                     Translation

s0
s0-s0:c0.c1023            SystemLow-SystemHigh
s0:c0.c1023               SystemHigh
```

Im Folgenden werde ich einige typische Verwendungen des Befehls `semanage` zeigen. Diese sollen Ihnen einen Eindruck davon vermitteln, wann und zu welchem Zweck Sie den Befehl anwenden können.

Achtung

Verfügt Ihre Distribution nicht über diesen Befehl, da sie die Example-Policy und nicht die Reference-Policy einsetzt, so müssen Sie für die folgenden Verwaltungschritte die Policy editieren und neu übersetzen. Eigentlich macht dieser Befehl auch nichts anderes.

Ein Nachteil des Befehls soll nicht verschwiegen werden. Leider können Sie mit dem Befehl nicht durch die Policy festgelegte Einstellungen verändern. Sie können lediglich Benutzer, Dateien und Ports hinzufügen. Ein Löschen dieser Informationen ist nur dann möglich, wenn Sie diese auch mit dem Befehl hinzugefügt haben. Durch die Policy definierte Daten können Sie nicht löschen. Der Befehl quittiert das mit einem Fehler:

```
# semanage port -d  -p tcp 22
/usr/sbin/semanage: Port tcp/22 ist in der Richtlinie festgelegt ➥
         und kann nicht entfernt werden
```

18.25.1 SELinux-User-Verwaltung

Normalerweise darf in der Strict-Policy nur *root* die SELinux-Policy verwalten. Um einem weiteren Benutzer auch den Zugriff auf die Rolle *sysadm_r* zu erlauben, sind drei Schritte erforderlich. Zunächst müssen Sie den Benutzer als Benutzer des Linux-Systems anlegen. Anschließend müssen Sie einen SELinux-Benutzer anlegen. Schließlich müssen Sie SELinux anweisen, bei der Anmeldung des Benutzers ihm den neuen SELinux-User zuzuweisen.

1. Anlegen des Linux-Benutzers. Hierzu können Sie einfach mit `useradd` einen neuen Benutzer anlegen:

   ```
   # useradd steve
   ```

2. Anlegen des SELinux-Benutzers. Der SELinux-Benutzer muss nicht den gleichen Namen tragen. Ein beliebiger Name ist möglich. Dennoch erleichert es die Administration, wenn die Namen ähnlich sind. Auch ist es nicht erforderlich, dass *_u* angehängt wird. Dies ist lediglich eine Konvention.

   ```
   # semanage user -a -P user -R "sysadm_r user_r staff_r" steve_u
   ```

   Der SELinux-Benutzer *steve_u* hat nun Zugriff auf die Rollen *sysadm_r*, *user_r* und *staff_r*. Dateien, die er in seinem Heimatverzeichnis erzeugt, erhalten den Typ *user_home_t*. Dies wird durch das angegebene Präfix bestimmt.

3. Zuordnung der Linux- und *SELinux!-User*. Um nun bei einer Anmeldung des Benutzers *steve* diesem den entsprechenden SELinux-Benutzer zuzuweisen, müssen Sie noch ein entsprechendes Login-Objekt anlegen:

```
# semanage login -a -s steve_u steve
```

18.25.2 Hinzufügen neuer Dateien

Viele Benutzer schalten SELinux bei dem ihrem ersten Kontakt ab, da sich das Linux-System nicht so verhält, wie es sich normalerweise verhalten sollte. Ein typisches Problem ist der Einsatz des Systems als Webserver und die Erzeugung zusätzlicher Verzeichnisse, in denen die Webseiten verwaltet werden (*DocumentRoot*).

Häufig vergessen dann SELinux-Erstanwender, dass die von ihnen angelegten Verzeichnisse nicht den korrekten Context besitzen. Damit der *Apache* Webserver diese Verzeichnisse und die in ihnen befindlichen Daten lesen darf, benötigen diese Verzeichnisse einen entsprechenden Typ. Üblicherweise ist dies *httpd_sys_content_t*. Wenn Sie die Dateien manuell mit diesem Typ versehen, besteht die Gefahr, dass bei einem späteren *Relabeling* des gesamten Systems wieder die Default-Typen angewendet werden. (Wie Sie dies auf modernen Systemen ausschließen können, ist in Abschnitt 16.4 beschrieben.) Da die Dateien in diesen Verzeichnissen aber immer den entsprechenden Typ aufweisen sollen, ist es sinnvoller, statt einer manuellen Zuweisung SELinux anzuweisen, die entsprechenden Typen zu vergeben.

Hierzu fügen Sie die neuen Verzeichnisse einfach der Liste der File-Contexts hinzu. Handelt es sich um das Verzeichnis `/www`, das die Webseiten enthält, geht das ganz einfach mit folgendem Befehl:

```
# semanage fcontext -a -t httpd_sys_content_t -f "" '/www(/.*)?'
```

Bei der Angabe der Dateien kann, wie hier gezeigt, ein regulärer Ausdruck für die Beschreibung verwendet werden. Hier bezieht sich der Ausdruck auf das Verzeichnis `/www` und alle darin befindlichen Dateien. Wenn Sie nun das Verzeichnis neu erzeugen, wird es zunächst seinen *Security-Context* von dem Elternverzeichnis erben. Mit `restorecon` weisen Sie ihm den richtigen Kontext zu.

18.25.3 Verwaltung der Netzwerk-Ports

Ein weiteres Einsatzgebiet des Befehls `semanage` ist die Zuordnung von SELinux-Typen zu Netzwerk-Ports. Vielen Standard-Ports ist bereits durch die Policy ein Typ zugewiesen worden, der den Einsatz des entsprechenden Dienstes ermöglicht. Häufig möchten jedoch Anwender einen Dienst auf einem anderen *Port* starten. Dies hat vollkommen unterschiedliche Gründe. So mag ein Anwender glauben, durch den Betrieb des *Secure-Shell*-Servers auf dem Port 222 statt 22 sei dieser besser vor Angriffen geschützt, oder ein anderer Anwender möchte den *Squid*-Webproxy auf dem Port 8888 betreiben.

In den meisten Fällen wird SELinux diese Abweichungen nicht erlauben. Die Dienste dürfen nur wenige Ports mit bestimmten Typen nutzen. Der Anwender muss dann die von ihm gewünschten Ports zur Liste der erlaubten Ports hinzufügen. Die beiden folgenden Befehle ermöglichen dies für die oben aufgeführten Szenarien:

```
# semanage port -a -t http_cache_port_t -p tcp 8888
# semanage port -a -t ssh_port_t -p tcp 222
```

18.26 semodule

Mit diesem Werkzeug verwalten Sie die Module der modernen SELinux-Reference-Policy. Wenn Ihre Distribution noch die ältere Example-Policy einsetzt (Fedora Core 2, 3 oder 4 und RHEL 4), existiert dieses Werkzeug bei Ihnen nicht. Auf den modernen Systemen stellt dieses Werkzeug zusammen mit dem Befehl `semanage` (siehe Abschnitt 18.25) die zentrale Administrationskomponente dar.

Mit diesem Werkzeug verwalten Sie die Module. Hierzu gehören die folgenden Aufgaben:

- Installation neuer Module
- Aktualisierung (Upgrade) der Module
- Anzeigen der geladenen Module
- Entfernen einzelner Module
- Erneuter Aufbau der Policy aus dem Modulspeicher
- Erneutes Laden der Policy

Die Module, die Sie mit dem Befehl verwalten können, werden von dem Befehl `semodule_package` erzeugt. Dabei erhalten die Module üblicherweise die Endung `.pp`. Bei den Modulen müssen Sie zwischen Base- und Nicht-Base-Modulen unterscheiden. Während immer nur ein *Base-Modul* geladen sein kann, können gleichzeitig viele verschiedene Nicht-Base-Module geladen sein. Üblicherweise existieren je Policy nur zwei verschiedene Base-Module:

- `base.pp`: Dieses *Modul* stellt im Normalfall die Grundfunktionalitäten der SELinux-Policy zur Verfügung.
- `audit.pp`: Dieses Modul unterscheidet sich von dem Modul `base.pp` lediglich in der Protokollierung. Während das normale Base-Modul viele Ereignisse mit *dont_audit*-Regeln nicht protokolliert, sind diese bei dem `audit.pp`-Modul deaktiviert. Dieses Modul eignet sich also zur Fehlersuche und wird nur in diesen Fällen bei Bedarf geladen.

Achtung

Leider deaktiviert das Modul `audit.pp` lediglich die *dont_audit*-Regeln im Base-Modul. Die entsprechenden Regeln in anderen Modulen können noch nicht abgeschaltet werden.

Mit den folgenden Parametern können Sie den Befehl verwenden:

- `-R, --reload`: Diese Option führt einen *Reload* der Policy durch. Dies ist interessant, da der Reload auch protokolliert wird und dieser Protokolleintrag von dem Befehl `audit2allow` (siehe Abschnitt 18.3) als Markierung genutzt werden kann.
- `-B, --build`: Hiermit wird die binäre Policy (`policy/policy.XX`) neu aus den verwendeten Modulen zusammengestellt. Der Kernel lädt immer nur die binäre Komplettdatei. Die Module dienen lediglich als Abstraktionsschicht.
- `-i, --install <modul.pp>`: Hiermit installieren oder ersetzen Sie ein geladenes Modul. Dabei wird das Modul auch in das Verzeichnis `modules/active/modules/` kopiert und die Policy neu gebaut.
- `-u, --upgrade <modul.pp>`: Hiermit wird ein geladenes Modul aktualisiert und durch eine neue Version ausgetauscht.
- `-b, --base <modul.pp>`: Dies lädt oder ersetzt ein Base-Modul.
- `-r, --remove <modul>`: Hiermit löschen Sie ein Modul aus der Policy. Dabei wird die Policy neu gebaut und das Modul aus dem Verzeichnis `modules/active/modules/` entfernt.
- `-l, --list-modules`: Diese Option zeigt die geladenen Module an.
- `-s, --store <speicher>`: Hiermit können Sie einen anderen Policy-Speicher auswählen. Im Moment ist diese Option noch experimentell. Dies erlaubt die Anbindung des Policy-Management-Servers (siehe Kapitel 32).
- `-n, --noreload`: Hiermit unterdrücken Sie nach einem Neubau der Policy deren Reload.

Wenn die Policy aufgrund von Moduländerungen neu gebaut wird, wird das alte Verzeichnis `/etc/selinux/<policy>/modules/active` umbenannt in `previous` und entsprechend der neuen Policy ein Verzeichnis `active/` neu angelegt.

Um ein Base-Modul zu installieren, verwenden Sie:

```
[root@supergrobi ~]# semodule -b audit.pp
```

18.27 semodule_expand

Dieser Befehl wird normalerweise nicht benötigt. Hiermit kann ein Entwickler manuell ein Base-Policy-Modul in eine binäre Policy umwandeln. Dieser Vorgang wird normalerweise von dem Befehl `semodule` bei Bedarf automatisch ausgeführt. Daher können Sie diesen Befehl bei dem täglichen Umgang ignorieren.

18.28 semodule_link

Dieser Befehl wird, wie der Befehl `semodule_expand` (siehe Abschnitt 18.27), normalerweise nicht benötigt. Hiermit kann manuell eine Gruppe von Modulen mit einem Base-Modul zu einem neuen Modul zusammengefasst werden. Dieses Linking erfolgt üblicherweise automatisch bei den entsprechenden `semodule`Aufrufen.

18.29 semodule_package

Dieser Befehl erzeugt ein neues Modul-Paket. Hierzu lädt dieser Befehl ein binäres Policy-*Modul* (`*.mod`) und optional weitere Dateien und baut hieraus ein Modul-Paket mit der Endung `.pp`. Dieses Paket kann dann mit dem Befehl `semodule` installiert werden (siehe Abschnitt 18.26). Das binäre Modul mit der Endung `.mod` wird von dem Compiler `checkmodule` (siehe Abschnitt 18.7) gebaut.

Um ein einfaches Modul für den Webserver zu bauen, können Sie den folgenden Befehl verwenden:

```
[root@supergrobi ~]# semodule_package -o httpd.pp -m httpd.mod ➥
          -f httpd.fc
```

Der Befehl unterstützt die folgenden Optionen:

- `-o, --outfile <datei.pp>`: Das fertige Paket wird in dieser Datei gespeichert. Diese Datei trägt üblicherweise die Endung `.pp`.
- `-s, --seuser <seuser-datei>`: Hiermit wird zusätzlich eine *SELinux!-User*-Datei mit in das Paket eingebunden. Dies erlaubt es, bei dem Laden des Moduls automatisch einen Benutzer hinzuzufügen.
- `-u, --user_extra <datei>`: Dies bindet eine weitere zusätzliche Datei in das Paket ein.
- `-m, --module <datei.mod>`: Dies ist das von dem Befehl `checkmodule` kompilierte binäre Policy-Modul.
- `-f, --fc <datei.fc>`: Hiermit kann eine File-Context-Datei angegeben werden.
- `-n, --nc <datei.nc>`: Hiermit können Sie eine *Netfilter*-Context-Datei angeben. Diese Option steht nur in ganz neuen Versionen zur Verfügung.

18.30 sesearch

Mit dem Kommando `sesearch` können Sie auch in einer binären Policy nach *Type-Enforcement*-Regeln suchen. Handelt es sich um eine binäre Policy, so werden die Regeln dennoch im Klartext dargestellt. Lediglich die Namen der verwendeten Attribute stehen dann nicht zur Verfügung.

Die Funktion des Befehls kann über viele verschiedene Optionen und Parameter angepasst werden. Um zum Beispiel nach allen Allow-Regeln zu suchen, die die Domäne *httpd_t* als Source-Type verwenden, können Sie den folgenden Befehl verwenden:

```
# sesearch -s httpd_t --allow

247 Rules match your search criteria
allow httpd_tmpfs_t tmpfs_t : filesystem associate;
allow httpd_tmpfs_t httpd_tmpfs_t : filesystem associate;
allow httpd_tmp_t httpd_tmp_t : filesystem associate;
...
```

Bei jedem Aufruf muss mindestens einer der folgenden Parameter angeben werden:

- `-a, --all`: Diese Option zeigt alle Regeln an.
- `--allow`: Diese Option zeigt nur die `allow`-Regeln an.
- `--neverallow`: Diese Option zeigt nur die *neverallow*-Regeln an.
- `--audit`: Diese Option zeigt alle `auditallow` und *dontaudit*Regeln an.
- `--rangetrans`: Diese Option zeigt die *Range-Transition*-Regeln an.
- `--type`: Diese Option sucht lediglich die *type_trans* und *type_change*-Regeln.
- `--role_allow`: Hiermit zeigen Sie die `role`-Regeln an.
- `--role_trans`: Diese Option filtert die `role_transition`-Regeln.

Des Weiteren können Sie die Ausgabe mit den folgenden Optionen einschränken:

- `-s, --source`: Hiermit geben Sie den Source-Typ an.
- `-t, --target`: Dies erlaubt die Angabe des Target-Typs der anzuzeigenden Regeln.
- `--role_source`: Hiermit können Sie die Source-Rolle bestimmen.
- `--role_target`: Analog bestimmt diese Option die Target-Rolle.
- `-c, --class`: Hiermit können Sie die Regeln auf bestimmte Target-Klassen einschränken.
- `-p, --perms`: Diese Option erlaubt die Einschränkung auf bestimmte Berechtigungen.
- `-b, --boolean`: Diese Option sucht Bedingungen mit dem angegebenen Namen in der Bedingung.

- `-i, --indirect`: Hiermit wird die Suche um die Attribute des Typs erweitert.
- `-n, --noregex`: Hiermit schalten Sie die Suche mit regulären Ausdrücken ab.
- `-l, --lineno`: Hiermit werden bei einer Policy im Quelltext die Zeilennummern der Regeln in der `policy.conf`Datei angezeigt.
- `-C, --show_cond`: Dies zeigt die Bedingungen bei bedingten Regeln mit an.

Um zum Beispiel alle Regeln anzuzeigen, die von der booleschen Variablen *httpd_can_network_connect_db* verwaltet werden und sich auf einen *TCP-Socket* beziehen, können Sie folgenden Befehl verwenden:

```
[root@supergrobi ~]# sesearch --allow --boolean ↵
        httpd_can_network_connect_db  --class tcp_socket
Found 6 av rules:
   allow httpd_sys_script_t @ttr0110 : tcp_socket  recv_msg send_msg;
   allow httpd_sys_script_t httpd_sys_script_t : tcp_socket ↵
         { ioctl read write create getattr setattr append bind ↵
         connect listen accept getopt setopt shutdown };
   allow httpd_sys_script_t postgresql_port_t:tcp_socket name_connect;
   allow httpd_sys_script_t mysqld_port_t : tcp_socket name_connect ;
   allow httpd_t postgresql_port_t : tcp_socket name_connect ;
   allow httpd_t mysqld_port_t : tcp_socket name_connect ;
```

18.31 sestatus

Dieser Befehl bietet Ihnen die Möglichkeit, einfach und schnell den Status des SELinux-Systems und wichtiger Dateien und Prozesse zu bestimmen. Hierzu rufen Sie einfach den Befehl auf:

```
[root@supergrobi ~]# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   permissive
Mode from config file:          enforcing
Policy version:                 21
Policy from config file:        targeted
```

Die Ausgabe kann mit `-b` um die booleschen Variablen erweitert werden. Einige Versionen dieses Befehls zeigen die booleschen Variablen automatisch an. Mit der Option `-v` wird die Ausgabe um den Kontext einiger Dateien und Prozesse erweitert, die Sie in der Datei `/etc/sestatus.conf` editieren können:

```
[files]
/etc/passwd
/etc/shadow
/bin/bash
/bin/login
/bin/sh
/sbin/agetty
/sbin/init
/sbin/mingetty
/usr/sbin/sshd
/lib/libc.so.6
/lib/ld-linux.so.2
/lib/ld.so.1

[process]
/sbin/mingetty
/sbin/agetty
/usr/sbin/sshd
```

So können Sie sehr einfach den Kontext wichtiger Dateien und Prozesse überwachen. Diese Datei können Sie mit einem beliebigen Editor erweitern.

18.32 setenforce

Mit diesem Befehl kann der Systemadministrator, wenn er über die ausreichenden SELinux-Privilegien verfügt, SELinux in den *Permissive* oder *Enforcing*Modus schalten. Ein Abschalten von SELinux ist mit diesem Werkzeug nicht möglich. Dies kann nur bei einem Reboot erfolgen. Hierzu müssen Sie entweder die Datei `/boot/grub/menu.lst`[3] oder `/etc/selinux/config` editieren. Um SELinux in den Enforcing-Modus zu versetzen, verwenden Sie `setenforce 1|Enforcing`. Den Permissive-Modus erreichen Sie mit `setenforce 0|Permissive`.

Achtung

Bei der *Strict-Policy* kann *root* in der Rolle *staff_r* zwar das System in den Enforcing-Modus versetzen. Um das System aber wieder in den Permissive-Modus zu versetzen ist die Rolle *sysadm_r* erforderlich.

[3] Hier genügt die Angabe von `selinux=0` auf der Kernel-Zeile.

18.33 setfiles

Dieses Kommando kann ebenfalls wie `restorecon` und `fixfiles` (siehe Abschnitt 18.16 und 18.9) den *Security-Context* einer Datei anpassen. Hierbei ist es jedoch besser für die Verarbeitung kompletter Dateisysteme geeignet. Dabei kann es sich auch um fremde oder temporär unter einem alternativen Pfad gemountete Dateisysteme handeln.

Bei dem Aufruf müssen Sie sowohl die Textdatei angeben, in der die Dateikontexte gespeichert sind (normalerweise `file_contexts`), als auch die zu verarbeitenden Dateien.

Im Einzelnen unterstützt der Befehl die folgenden Optionen:

- `-c`: Hiermit prüfen Sie ob, die binäre Policy mit den Angaben in `file_contexts` übereinstimmt.
- `-d`: Dies gibt zusätzlich jede Datei und den entsprechenden Kontext aus.
- `-l`: Hiermit werden Veränderungen des Kontextes über Syslog protokolliert.
- `-n`: Diese Option führt lediglich eine Überprüfung durch.
- `-q`: Hiermit unterdrücken Sie Fehlermeldungen.
- `-r <root>`: Dies erlaubt die Angabe eines alternativen Root-Verzeichnisses.
- `-e <verzeichnis>`: Hiermit schließen Sie das Verzeichnis aus.
- `-F`: Diese Option erzwingt auch die Modifikation der Dateien, die über einen *customizable_type* verfügen.
- `-o <datei>`: Diese Option speichert die Namen der Dateien mit falschem Kontext in der Datei ab.
- `-s`: Hiermit liest der Befehl die zu verarbeitenden Dateien über die Standardeingabe.
- `-v`: Diese Option zeigt Änderungen des Typs und der Rolle an.
- `-vv`: Diese Option zeigt zusätzlich auch Änderungen des SELinux-Benutzers an.
- `-W`: Hiermit erhalten Sie für alle Dateien, die keinen Kontext besitzen, eine Warnmeldung.

Natürlich müssen Sie nicht unbedingt die Datei `file_contexts` aus Ihrem SELinux-Verzeichnis verwenden, sondern können auch eine eigene Datei erzeugen. Dies kann für die verschiedensten Zwecke sinnvoll sein. Dann müssen Sie nur darauf achten, dass diese Datei die richtige Syntax aufweist. Diese Syntax wird in Abschnitt 16.3 besprochen.

18.34 setsebool

Mit dem Befehl `setsebool` setzen Sie die booleschen Variablen (siehe Kapitel 15). Entweder Sie setzen eine oder auch gleichzeitig mehrere boolesche Variablen:

```
superdebian:~# setsebool use_samba_home_dirs on
superdebian:~# setsebool use_samba_home_dirs=on use_nfs_home_dirs=on
```

Diese Veränderungen gehen jedoch mit dem nächsten Reboot verloren. Um die Veränderungen dauerhaft einzutragen, geben Sie zusätzlich die Option `-P` an. Wartende Änderungen (*Pending* Changes) werden auch bei dem Aufruf aktiviert und werden bei Angabe der Option `-P` persistent.

18.35 setroubleshootd

Der Setroubleshootd-Daemon gehört zu einem Framework, das die Meldungen des Audit-Subsystems analysiert und dem Benutzer leichter verständlich präsentiert. Dieses Framework verwendet eine Client-Server-Architektur. Der Server ist `setroubleshootd`. Dieser Server ist in Python programmiert worden und verfügt über die notwendigen Rechte, die Meldungen zu überwachen. Bei seinem Start lädt der Dienst einige Plugins, die es ihm erlauben, die verschiedenen *Access-Vector-Cache*-Meldungen zu verstehen. Der Benutzer greift auf diese Informationen über den Befehl `sealert` zu.

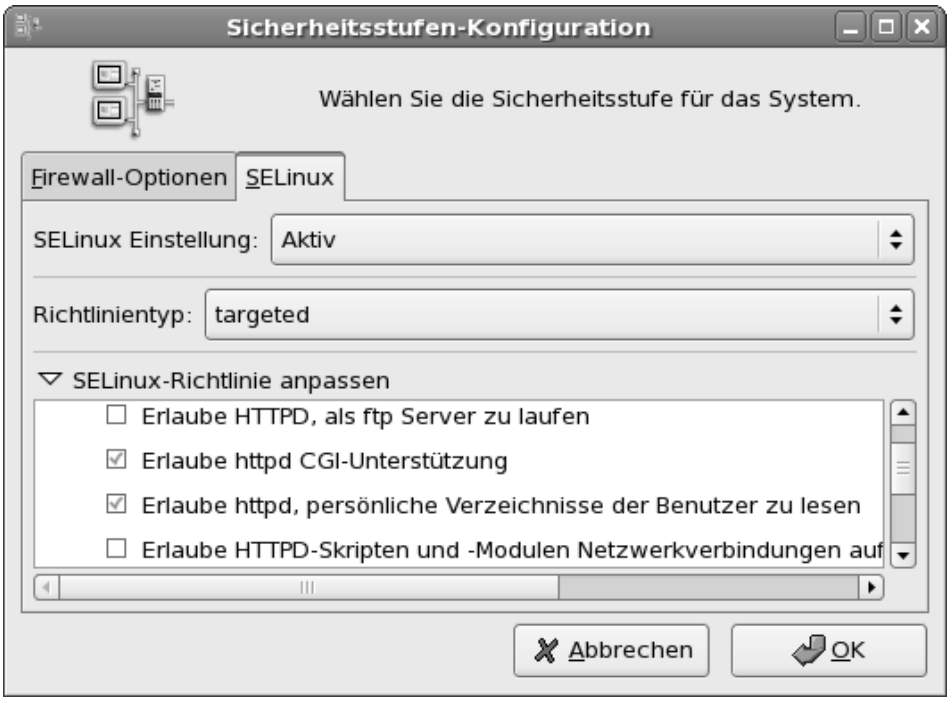

Abbildung 18.4: RedHat-basierte Distributionen erlauben eine grafische Verwaltung der SELinux-Einstellungen.

18.36 system-config-securitylevel

Fedora Core und *RedHat Enterprise Linux*-Distributionen verfügen über das Werkzeug `system-config-securitylevel`. Hiermit können Sie grafisch (Abb. 18.4) die SELinux-Einstellungen anpassen. Die Curses-basierte Textoberfläche des Werkzeugs unterstützt leider nicht die Verwaltung der SELinux-Einstellungen, sondern nur die Verwaltung der Firewallregeln. Bei dem Einsatz dieses Werkzeugs sollten Sie jedoch beachten, dass dieses Werkzeug gleichzeitig neben den SELinux-Einstellungen auch die Firewalleinstellungen überschreibt. Falls Sie also eigene Einstellungen bezüglich der Firewall vorgenommen haben, sollten Sie zunächst prüfen, ob dieses Werkzeug Ihre Einstellungen modifiziert oder unwirksam macht.

18.37 togglesebool

Mit diesem Befehl können Sie einfach den Wert einer booleschen Variable ändern. War der Wert vorher 1, so ist er anschließend 0 und umgekehrt. Dabei sollten Sie jedoch beachten, dass dieser Befehl nur die aktuellen Werte im Speicher modifiziert und nicht den Wert der booleschen Variable zum Bootzeitpunkt modifiziert.

19 Typische SELinux-Administrationsaufgaben

Viele Administratoren schalten als Erstes bei dem Einsatz der entsprechenden Distributionen SELinux ab. Das ist schade und meist unnötig, wenn grundsätzliche Kenntnisse des Systems existieren.

Dieses Kapitel versucht die typischen Probleme beim Einsatz von SELinux zu erklären und Lösungen aufzuzeigen.

19.1 Abschalten von SELinux

Vielleicht wollen Sie SELinux einfach nur abschalten. Ich kann dieses Vorgehen nicht empfehlen, sondern Ihnen nur raten, sich mit dem System zu beschäftigen und es zu nutzen. Es wird die Sicherheit Ihres Systems enorm erhöhen.

Falls Sie sich dennoch dazu entscheiden, möchte ich Ihnen zeigen, wie Sie dies richtig bewerkstelligen. Eigentlich ist es ganz einfach. Es gibt drei Möglichkeiten:

- Sie können SELinux komplett abschalten.
- Sie können SELinux in einen Warnmodus versetzen.
- Sie können SELinux für einen einzelnen Dienst abschalten, wenn Sie die Targeted Policy verwenden.

Während die dritte Alternative im nächsten Abschnitt betrachtet wird, werden wir hier die ersten beiden Möglichkeiten besprechen. Die Konfiguration von SELinux erfolgt in der Datei `/etc/selinux/config`. Diese Datei enthält unter anderem die folgenden Zeilen:

```
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - SELinux is fully disabled.
SELINUX=permissive
```

Der Wert der Variable `SELINUX` bestimmt das Verhalten bei dem Boot:

- *Enforcing*: SELinux ist aktiv und setzt die Policy um. SELinux kann zur Laufzeit in den Modus Permissive versetzt werden.

- *Permissive*: SELinux ist aktiv. Verletzungen der Policy werden erlaubt, aber protokolliert. SELinux kann zur Laufzeit in den Modus Enforcing versetzt werden.
- *Disabled*: Dies schaltet SELinux komplett ab. Es kann zur Laufzeit nicht aktiviert werden. Ein Reboot ist hierzu erforderlich.

Um SELinux komplett abzuschalten, genügt daher die folgende Zeile in der Datei `/etc/selinux/config`:

```
SELINUX=disabled
```

Achtung

Es ist wichtig, dass Sie in dieser Zeile keine Leerzeichen einfügen.

Sie müssen aber berücksichtigen, dass nun ohne einen Neustart des Systems keine Aktivierung von SELinux möglich ist. SELinux kann nicht, wie zum Beispiel AppArmor, im laufenden Betrieb geladen werden. Außerdem verlangt ein späterer Start mit aktiviertem SELinux immer ein komplettes *Relabeling* (siehe Abschnitt 19.3) des Systems, so als ob SELinux nie auf dem System aktiv gewesen sei (siehe Kapitel 22).

19.2 Abschalten von SELinux für einen Dienst

In vielen Fällen entsteht der Wunsch, SELinux abzuschalten, wenn ein Dienst sich nicht wie erwartet verhält. Anstatt SELinux komplett abzuschalten, sollten Sie in diesem Fall abwägen, ob Sie zunächst SELinux nur für diesen Dienst oder das betroffene Programm abschalten. Dann können Sie später, nach der Lektüre dieses Buchs, mit mehr Verständnis, Wissen und Zeit, SELinux entsprechend anpassen und wieder für den Dienst aktivieren.

Wenn Sie die *Targeted-Policy* einsetzen, können Sie für jeden überwachten Dienst SELinux einzeln an- bzw. wieder abschalten.

Hierzu existiert für jeden Dienst eine boolesche Variable: `<dienst>_disable_trans`. Wenn Sie diese Variable setzen und den Dienst neu starten, wird der Dienst anschließend nicht von SELinux in seiner eigenen Domäne überwacht, sondern wird in der Domäne *unconfined_t* kontrolliert. Damit ist der Dienst faktisch ohne besondere Überwachung.

Für den Webserver lautet die Variable zum Beispiel: `httpd_disable_trans`.

19.3 Erneutes Labeln des Betriebssystems

Ein *Relabeling* des Systems ist nur in seltenen Fällen nötig. Hierbei handelt es sich um den Wechsel der Policy oder um den Neustart des Systems, nachdem es zwischendurch ohne SELinux-Unterstützung betrieben wurde.

Jede Datei auf einem SELinux-System besitzt einen *Security-Context*. Dieser Security-Context bestimmt, wer diese Datei nutzen darf. Falls eine Datei keinen Security-Context besitzt, erhält sie automatisch einen Default-Context. Dieser hängt von der Policy ab. Mit dem Security-Context bestimmt SELinux, welche Prozesse auf die Datei zugreifen dürfen. In vielen Fällen führt diese Tatsache dazu, dass der gewünschte Zugriff auf die Datei nicht möglich ist. Handelt es sich nur um eine Datei, kann der Security-Context sehr einfach mit dem Befehl `restorecon` wiederhergestellt werden.

Wird das System mit einem Kernel gebootet, der kein SELinux unterstützt, oder wurde SELinux vorübergehend abgeschaltet, so sind während des Betriebs alle Dateien ohne einen Security-Context angelegt worden. Hierbei handelt es sich nun um zahlreiche über das gesamte System verstreute Dateien. Diese alle mit `restorecon` einzeln zu labeln würde recht lange dauern. Mit `find` können Sie zwar diese Dateien finden, jedoch ist das recht aufwendig.

Meist erkennt das Betriebssystem selbst, dass es vorübergehend ohne SELinux-Unterstützung betrieben wurde, und erzwingt das Relabeling. Wenn Sie es manuell anstoßen möchten, gibt es zwei Möglichkeiten:

- Sie können die Datei `/.autorelabel` anlegen. Findet der SysV-Init diese Datei, so führt er automatisch ein Relabeling bei dem Boot durch.
- Sie rufen den Befehl `fixfiles relabel` auf.

Wenn Sie mit dem Werkzeug `system-config-securitylevel` einer Fedora oder RHEL-Distribution die Policy wechseln, wird ebenfalls das *Relabeling* des gesamten Systems erzwungen. Wenn Sie manuell die Policy wechseln, indem Sie die Policy in der Datei `/etc/selinux/config` ändern, dann müssen Sie auch die Datei `/.autorelabel` erzeugen.

19.4 Programme in unconfined_t funktionieren nicht

SELinux beschränkt unter Fedora und RHEL in den neueren Versionen auch Programme in der Domäne *unconfined_t*. Dabei überwacht SELinux einige Speicheroperationen:

- `execmod`: Hierbei überwacht das System, ob ein Programm eine Speicherseite, die zuvor geschrieben (modifiziert) wurde, anschließend ausführen möchte. Da diese Funktionalität bei vielen Angriffen (z.B. Buffer-Overflows) benötigt wird, verhindert SELinux dies. Ein häufiger Grund für derartige Fehler sind jedoch Text-Relocations. Diese können Sie für eine Bibliothek erlauben. Allerdings sollten Sie anschließend einen Bug-Report verfassen, da es sich eigentlich um einen

Fehler handelt. Ulrich Drepper hat weitere Informationen auf seiner Homepage[1]. Sie erkennen den Fehler an der folgenden Meldung:

```
error while loading shared libraries: /usr/lib/<bibliothek>.so
cannot restore segment prot after reloc: Permission denied
```

Um Text-Relocations zu erlauben, stellen Sie sicher, dass die betroffene Bibliothek den Typ *textrel_shlib_t* besitzt:

```
# /usr/sbin/semanage fcontext -a -t textrel_shlib_t ➥
        '/usr/lib/<bibliothek>.so'
# /sbin/restorecon -v /usr/lib/<bibliothek>.so
```

Zusätzlich kann die boolesche Variable `allow_execmod` aktiv sein. Dann dürfen in der *Targeted-Policy* alle Programme und Bibliotheken vom Typ *unconfined_t* im Vorfeld modifizierten Speicher ausführen.

- `execmem`: Dieser Fehler tritt auf, wenn eine Applikation ein Anonymous Mapping[2]als ausführbar kennzeichnet. Um dies zu erlauben, existiert die boolesche Variable `allow_execmem`, die dies für alle Applikationen erlaubt.
- `execstack`: Dieser Fehler tritt auf, wenn eine Applikation versucht, ihren Stack oder Teile davon ausführbar zu machen. Dies kann nur über boolesche Variablen erlaubt werden. Hierzu stehen unter Fedora 6 zwei Variablen zur Verfügung:

  ```
  allow_execstack --> on
  allow_java_execstack --> off
  ```

- `execheap`: Dieser Fehler tritt auf, wenn ein Programm Daten auf dem Heap ausführen möchte. Dies sollte bei sauberer Programmierung nie erforderlich sein. Dennoch können Sie es mit einer booleschen Variable erlauben: `allow_execheap`.

19.5 KDE-Programme

Nach der Installation und Aktivierung von SELinux funktioniert die grafische Oberfläche *KDE* möglicherweise nicht mehr wie erwartet. Wenn Sie auf Ihrem System zu einem späteren Zeitpunkt SELinux installieren und aktivieren und vorher bereits KDE benutzt haben, kann es zu Problemen kommen. KDE legt viele temporäre Dateien in `/tmp` und `/var/tmp` an. Diese können bei dem automatischen *Relabeling* nicht richtig erkannt werden und erhalten daher nicht den korrekten *Security-Context*. Bei einem erneuten Start von KDE kann es daher nicht auf die Dateien zugreifen und startet nicht korrekt. Löschen Sie daher einfach diese Dateien von KDE:

```
rm -rf /var/tmp/kdecache-<user>
```

[1] *http://people.redhat.com/drepper/textrelocs.html*

[2] Programme können Dateien in den Speicher »mappen«. Hierzu dient der Funktionsaufruf `mmap`. Wenn allgemein Speicher benötigt wird, ohne dass eine spezielle Datei genutzt wird, kann als Flag bei dem Aufruf `MAP_ANONYMOUS` angegeben werden. Dann wird lediglich eine bestimmte Menge Speicher zur Verfügung gestellt.

19.6 Start eines Dienstes mit ungewöhnlichem Port

Einige Administratoren möchten einen Netzwerkdienst auf einem Nicht-Standard-*Port* starten. SELinux verhindert dies für die überwachten Dienste in der *Targeted-Policy*. In der *Strict-Policy* wird das für jeden Dienst überwacht. Um einen weiteren Port zu erlauben, muss dieser SELinux bekannt gemacht werden. Um zum Beispiel den Webserver auf dem Port 8088 zu starten, ist es erforderlich, dass der Port über den entsprechenden Typ verfügt. Zunächst sollte geprüft werden, welchen Typ der Port benötigt. Normalerweise startet der *Apache* Webserver auf dem Port 80. Mit dem folgenden Befehl ermitteln Sie den Typ des Ports:

```
[root@supergrobi ~]# semanage port -l | grep 80
amanda_port_t                  tcp      10080, 10081, 10082, 10083
amanda_port_t                  udp      10080, 10081
hplip_port_t                   tcp      1782, 2207, 2208, 8290, ➥
         50000, 50002, 8292, 9100, 9101, 9102, 9220, ➥
         9221, 9222, 9280, 9281, 9282, 9290, 9291, 9292
http_cache_port_t              tcp      8888, 3128, 8080, 8118
http_port_t                    tcp      81, 80, 443, 488, 8008, ➥
         8009, 8443
ocsp_port_t                    tcp      9080
soundd_port_t                  tcp      8000, 9433
transproxy_port_t              tcp      8081
xen_port_t                     tcp      8002
zope_port_t                    tcp      8021
```

Da Sie nun wissen, dass SELinux dem Webserver den Zugriff auf Ports vom Typ *http_port_t* erlaubt, können Sie mit `semanage` Ihren Port mit dem entsprechenden Typ versehen:

```
[root@supergrobi ~]# semanage port -a -p tcp -t http_port_t 8088
```

Leider kann ein Port nicht zwei Typen gleichzeitig erhalten. Ports, die durch die Policy definiert werden, können auch nicht modifiziert werden. Wenn Sie einen Port verwenden möchten, der bereits in der Policy anders definiert wurde, müssen Sie dem Webserver entweder erlauben, auf Ports mit diesem Typ zuzugreifen, oder die Policy modifizieren, indem Sie das entsprechende Policy-Modul entfernen oder die gesamte Policy neu bauen. Hinweise hierzu finden Sie in Abschnitt 16.5.

19.7 Verwendung einer SWAP-Datei

Wenn der *SWAP*-Speicher in Form von SWAP-Partitionen nicht reicht, kann auch eine Datei als SWAP genutzt werden. Hierzu muss die Datei lediglich erzeugt und entsprechend formatiert werden:

```
[root@supergrobi ~]# dd if=/dev/zero of=/swap bs=1M count=100
100+0 Datensaetze ein
100+0 Datensaeze aus
104857600 Bytes (105 MB) kopiert, 0,262746 Sekunden, 399 MB/s
[root@supergrobi ~]# mkswap /swap
Setting up swapspace version 1, size = 104853 kB
```

Damit SELinux die Verwendung als SWAP zulässt, muss die Datei nun auch noch den richtigen Typ erhalten. Hierzu benötigt sie den Typ *swapfile_t*. Dies erreichen Sie mit:

```
[root@supergrobi ~]# chcon -t swapfile_t /swap
```

Da der Typ *swapfile_t* zu den *Customizable Types* gehört, ist die Registrierung in der Policy mit `semanage` nicht zwingend erforderlich.

19.8 Apache Webserver

Viele Administratoren setzen ein exponiertes Linux-System als Betriebssystem für einen *Apache* Webserver ein. Hier ist ein hohes Maß an Sicherheit erforderlich. Dies trifft besonders bei Webservern mit dynamisch generierten Webseiten zu. Häufig gab es in der Vergangenheit Probleme und Sicherheitslücken, die einen Angriff ermöglichten. Dies wird in der Zukunft sicherlich weiterhin so sein.

Daher ist es besonders wichtig, den Webserver mit SELinux zu überwachen. Hier tauchen aber auch die meisten Probleme auf. Daher will ich Ihnen in den folgenden Abschnitten die wesentlichen Probleme und ihre Lösungen aufzeigen.

19.8.1 Neues DocumentRoot-Verzeichnis

Viele Benutzer legen für ihren Webserver ein eigenes *DocumentRoot*-Verzeichnis an. Dies trifft besonders in vielen *Shared-Hosting*-Umgebungen zu. Die SELinux-Policy berücksichtigt aber nur die Default-Verzeichnisse der Linux-Distributionen:

- `/var/www`
- `/srv/www`
- `/www`

Falls Sie Ihr DocumentRoot-Verzeichnis an einer anderen Stelle, z.B. `/home/www` ablegen möchten, so berücksichtigt die Policy nicht das Verzeichnis und erlaubt dem Apache Webserver nicht den Zugriff auf die dort gespeicherten Dateien.

Laut seiner Policy darf der Apache nur auf Dateien mit folgendem Typ zugreifen:

- *httpd_sys_content_t*: Alle Dateien mit diesem Typ dürfen von dem Webserver ausgeliefert werden. Auch *PHP*-Scripts benötigen diesen Typ, wenn sie mit `mod_php` ausgeliefert werden. Dies ist ein *Customizable Types*. Es ist daher nicht zwingend erforderlich, die Dateien in der Policy zu registrieren. Allerdings kann es auch nicht schaden. Um sicherzustellen, dass alle Dateien in dem Verzeichnis `/web` diesen Typ erhalten, verwenden Sie:

```
[root@supergrobi ~]# semanage fcontext -a -t ➥
         httpd_sys_content_t '/web(/.*)?'
```

- *httpd_sys_script_exec_t*: *CGI*-Scripts benötigen diesen Typ. Sie dürfen dann auf alle Dateien mit dem Typ *httpd_sys_** zugreifen.
- *httpd_sys_script_ro_t*: Dateien mit diesem Typ dürfen nur von CGI-Scripts mit Typ *httpd_sys_script_t* gelesen werden.
- *httpd_sys_script_rw_t*: Dateien mit diesem Typ dürfen nur von CGI-Scripts mit dem Typ *httpd_sys_script_t* gelesen und geschrieben werden.
- *httpd_sys_script_ra_t*: Diese Dateien dürfen gelesen und im Append-Modus von CGI-Scripts mit dem Typ *httpd_sys_script_t* geschrieben werden.
- *httpd_unconfined_script_exec_t*: Dies ist die letzte Rettung für Scripts, die so kompliziert sind, dass eine Anpassung der Policy zu aufwendig ist. Bevor Sie die SELinux-Überwachung für den gesamten Webserver abschalten, können Sie einem Script diesen Typ zuweisen und damit die Überwachung nur für das Script abschalten.

19.8.2 Gleichzeitiger Zugriff per FTP, Samba etc.

Häufig werden die Webseiten, die der *Apache* Server ausliefern soll, von den Anwendern über eine Windows-Freigabe per *Samba* oder mithilfe eines *FTP*-Servers verwaltet. Sobald die Dateien aber mit Samba oder mithilfe eines FTP-Servers geschrieben werden, besitzen sie nicht den richtigen Typ für die Auslieferung durch den Webserver. Viele Benutzer schalten daher die Überwachung für den Webserver ab. Dies muss jedoch nicht sein, da die SELinux Policy diese gemeinsame Nutzung erlaubt.

Hierzu wurden in der Policy die Typen *public_content_t* und *public_content_rw_t* geschaffen. Die Dienste Apache, FTP, Samba und *Rsync* dürfen alle Dateien vom Typ *public_content_t* lesen. Wenn einer dieser Dienste auch Dateien schreiben soll, so müssen Sie zuvor das Verzeichnis, in dem der Dienst Schreibrechte erhalten soll, mit dem Typ *public_content_rw_t* versehen und für den Dienst die boolesche Variable `allow_<domain>_anon_write` setzen. Um CGI-Scripts Schreibrechte an diesen Dateien zu geben, verwenden Sie die boolesche Variable `allow_httpd_sys_script_anon_write`. Die so erzeugten Dateien können dann auch per FTP oder Samba zur Verfügung gestellt werden.

19.8.3 Zugriff auf eine MySQL-Datenbank

Häufig werden bei dem Betrieb eines Webservers PHP- oder Perl-Scripts eingesetzt, die Zugriff auf eine Datenbank oder andere Netzdienste benötigen. Normalerweise unterbindet die SELinux-Policy diese Zugriffe. Um diese Zugriffe ohne großen Aufwand zu erlauben, können Sie zwei boolesche Variablen verwenden. Hierbei handelt es sich um:

- `httpd_can_network_connect`: Diese Variable erlaubt dem Webserver den Aufbau von Netzwerkverbindungen zu beliebigen entfernten TCP-Diensten.
- `httpd_can_network_connect_db`: Diese Variable erlaubt dem Webserver den Aufbau von TCP-Verbindungen zu Ports, die als Typ *mysqld_port_t* oder *postgresql_port_t* aufweisen. Dies sind normalerweise die Ports 3306 und 5432. Wenn Ihre Datenbank einen anderen Port verwendet, können Sie den entsprechenden Port natürlich mit einem der beiden Typen versehen. Dann wird die Policy den Zugriff erlauben.

19.9 Sicherung (Backup)

Wenn Sie ein SELinux-System sichern, ist es wichtig, dass Sie auch die *Security-Contexts* der Dateien sichern. Kommandos wie `tar` können dies nicht.

Wenn Sie ein Ext3-Dateisystem einsetzen, können Sie die Kommandos `dump` und `restore` verwenden. Diese sichern die Dateien auf Dateisystemebene und berücksichtigen in neueren Versionen die erweiterten Attribute, in denen die SELinux-Contexts gespeichert werden.

Wenn Sie bisher das Kommando `tar` eingesetzt haben, können Sie dieses durch `star` ersetzen:

```
star --xattr -H=exustar -c -f sicherung.star /verz
```

20 Analyse mit apol

Eine formale Analyse einer SELinux-Richtlinie hat viele verschiedene Aspekte. Wenn Sie aber die Zusammenhänge und Beziehungen zwischen den verschiedenen Benutzern, Rollen und Typen analysieren möchten, ist `apol` das Werkzeug Ihrer Wahl. Dieses Kapitel zeigt Ihnen, wie Sie mit `apol` Ihre Policy analysieren.

Das Programm `apol` ist in dem Paket `setools-gui` enthalten. Nach dem Start des Kommandos müssen Sie zunächst eine Policy laden. Hierzu wählen Sie FILE/OPEN... Dort navigieren Sie zu dem Verzeichnis, in dem die Policy enthalten ist, und wählen diese aus. Wenn Sie die modulare *Reference-Policy* verwenden, gibt es keine ASCII-Version der Policy, sondern nur die binäre Variante, z.B.: `/etc/selinux/targeted/policy/policy.21`. Während die Policy geladen wird, zeigt Ihnen `apol` den Fortschritt an (siehe Abbildung 20.1).

Abbildung 20.1: Das Laden der Policy kann einige Minuten dauern.

Leider haben die binären Policys den Nachteil, dass die Namen der Attribute nicht mehr in der Datei enthalten sind. Das Programm erzeugt dann neue Namen (siehe Abbildung 20.2).

Abbildung 20.2: Bei binären Policys muss apol neue Namen für die Attribute erzeugen, da die originalen Namen nicht in der Policy gespeichert sind.

Anschließend präsentiert sich die `apol`-Oberfläche (Abbildung 20.3). Die `apol`-Oberfläche ist in mehreren Registerkarten angeordnet, die einzeln in den nächsten Abschnitten betrachtet werden. Im unteren Bereich der Darstellung zeigt Ihnen `apol` Informationen über die aktuell geladene Policy an.

20.1 Policy-Components

Auf dieser Registerkarte können Sie die einzelnen Komponenten der Policy betrachten und nach ihnen suchen. Interessieren Sie sich zum Beispiel für alle Benutzer, die die Rolle *sysadm_r* verwenden dürfen, wechseln Sie zunächst auf die Registerkarte POLICY COMPONENTS und dort auf die Registerkarte USERS. Hier wählen Sie dann bei den Suchoptionen die Rolle (ROLE) aus und in dem nun erscheinenden Pulldown-Menü die Rolle *sysadm_r*. Nach Bestätigung mit OK zeigt Ihnen `apol` alle Benutzer an, die diese Rolle verwenden können (Abbildung 20.3).Auch die Security-Contexts der einzelnen Netzwerkobjekte (NET CONTEXTS) und Dateien (FS CONTEXTS) lassen sich anzeigen.

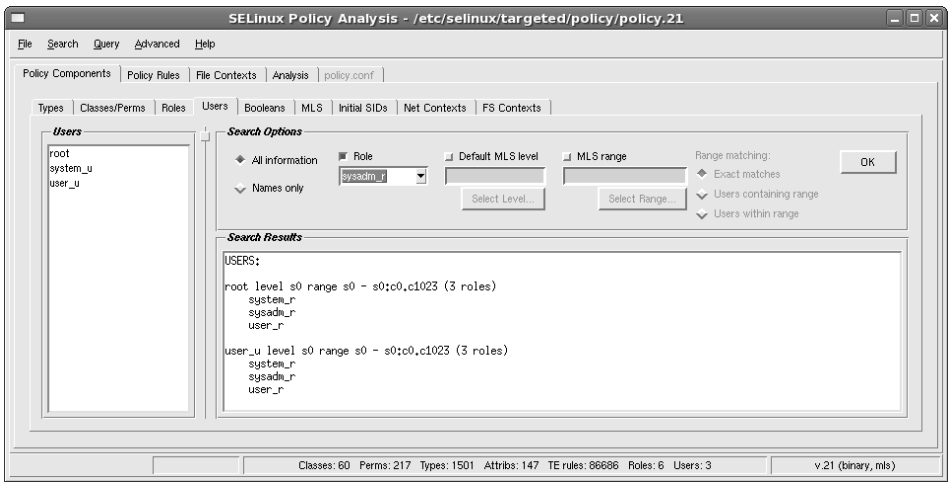

Abbildung 20.3: apol unterstützt Sie bei der Suche nach einzelnen Policy-Komponenten wie Rollen und Benutzern.

20.2 Policy-Rules

Möchten Sie wissen, ob eine bestimmte Regel in der Policy vorhanden ist, können Sie dies über die Registerkarte POLICY RULES erfahren. Hierzu wählen Sie auf dieser Registerkarte die Informationen aus, die Sie kennen, zum Beispiel den Source-Typ und den Target-Typ. Sie können auch nur einen angeben und erhalten dann entsprechend mehr Informationen.

Möchten Sie zum Beispiel wissen, ob es eine Regel gibt, die der Domäne *dhcpd_t* Zugriffe auf Objekte vom Typ *etc_t* gewährt, wählen Sie diese Informationen aus und bestätigen diese mit NEW SEARCH. `apol` liefert Ihnen das Ergebnis (siehe Abbildung 20.4).

Neben einfachen TE-Regeln können Sie auch nach *RBAC*-Regeln und *Range-Transition*-Regeln in *MLS*-Policys suchen.

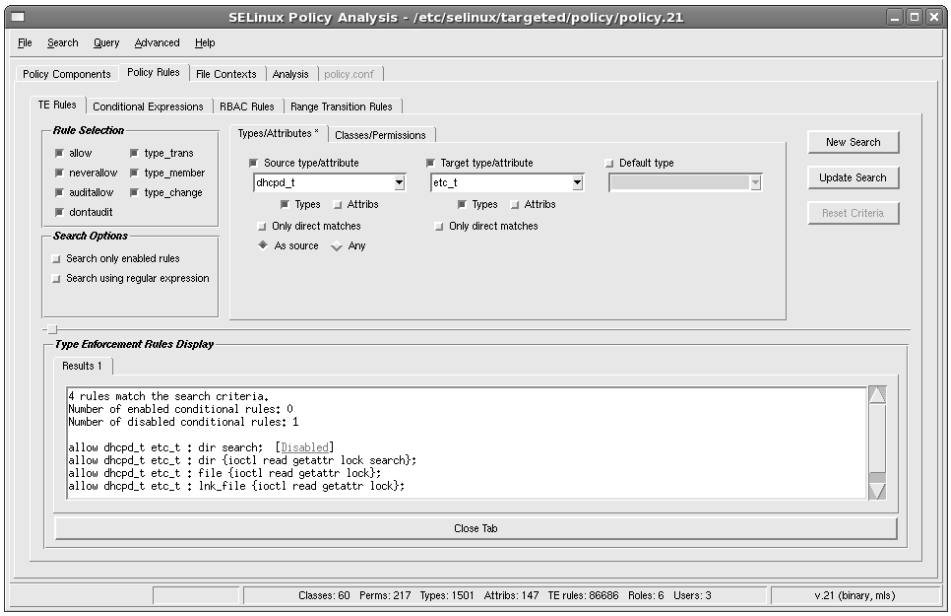

Abbildung 20.4: apol unterstützt Sie auch bei der Suche nach Regeln, die den Zugriff auf bestimmte Objekte zulassen.

20.3 File Contexts

Auf der Registerkarte FILE CONTEXTS können Sie die Security-Contexts der einzelnen Dateien analysieren. Hierzu benötigt `apol` eine Index-Datei, die Sie zunächst erzeugen müssen. Hierzu können Sie einzelne Verzeichnisse angeben, die durchsucht werden sollen. Die Erzeugung dauert eine Weile.

Anschließend können Sie in der Datenbank nach Dateien mit bestimmten Security-Contexts suchen. In dem Beispiel aus Abbildung 20.5 wird zum Beispiel nach allen Dateien des SELinux-Benutzers *user_u* mit dem Typ *etc_t* in der Datenbank gesucht.

20.4 Analysis

Die Registerkarte ANALYSIS ist die mächtigste und gleichzeitig auch komplizierteste Funktion von `apol`. Hier stehen Ihnen insgesamt fünf verschiedene Analysen zur Verfügung:

- Domain Transition: Sie können feststellen, welche Domänentransitionen einer bestimmten Domäne zur Verfügung stehen.
- Direct Information Flow: Nach der Wahl eines Start- oder Ziel-Typs können Sie prüfen, welche Domäne Informationen dieses Typs lesen oder schreiben darf.
- Transitive Information Flow

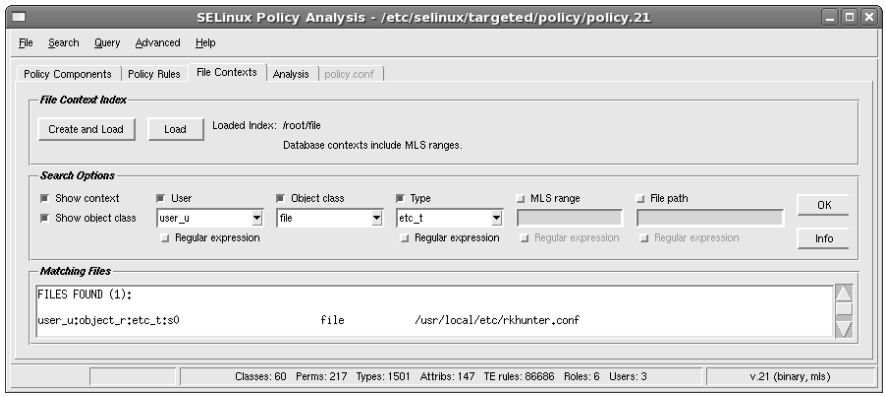

Abbildung 20.5: apol sucht nach bestimmten Dateien im Verzeichnisbaum. Aus Geschwindigkeitsgründen verwendet es hierbei eine Datenbank.

- Direct Relabel
- Types Relationship Summary

Hiermit können Sie nachvollziehen, welche Domänen alle Daten eines bestimmten Typs lesen dürfen. Um nachzuvollziehen, welche Prozesse auf Dateien vom Typ *shadow_t* zugreifen dürfen, benötigen Sie diesen Modus (Abbildung 20.6).

Alle Funktionen aufzuzählen würde hier zu weit führen. Sinnvollerweise experimentieren Sie mit dem Werkzeug ein wenig, wenn Sie an derartigen Fragen interessiert sind.

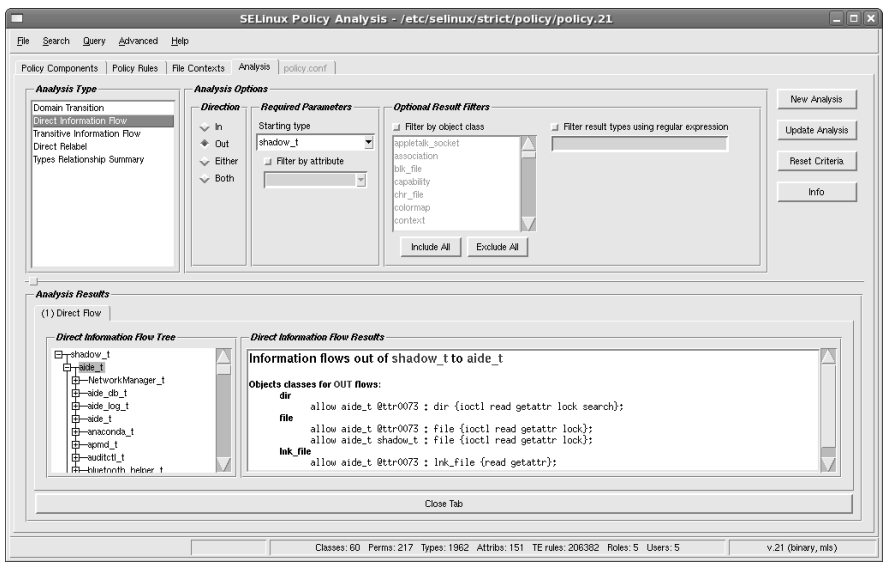

Abbildung 20.6: apol zeigt Ihnen, welche Domänen auf Dateien mit einem bestimmten Typ zugreifen dürfen. Hier erkennen Sie auch den Nachteil der fehlenden Attributnamen.

21 SELinux-Update

Ein Update der SELinux-Richtlinien wird meist durch den Distributor ausgelöst. Sowohl *Debian* als auch *Fedora Core* liefern im Laufe der Lebenszeit einer Distribution neue *Update*-Pakete aus. Auf vielen einfachen Desktop-Systemen können diese Updates sicherlich unproblematisch installiert werden. Handelt es sich jedoch um einen Server im Produktionsbetrieb, der mit SELinux überwacht wird, sollte man sich zuvor einige Gedanken über das Update machen.

Was kann passieren?

- Die neue Policy erzwingt ein *Relabeling* des Systems. Hierzu ist ein Reboot erforderlich.
- Das System bootet anschließend nicht mehr, da die neue Policy fehlerhaft ist.
- Die eigenen Anpassungen funktionieren anschließend nicht mehr, da sich einige Typen geändert haben.
- Die Applikationen werden nun mit neuen Richtlinien überwacht. Bisher erlaubte und von dem Server genutzte Funktionen werden nun verboten. Der Server kann seine Aufgaben nicht mehr wahrnehmen.

Um grob einen Überblick über die Modifikationen zu erhalten, sollte das Update-Paket zunächst auf einem Testsystem installiert werden. Anschließend können Sie sich mit `rpm` die Dokumentation des Paketes ansehen.

```
[root@supergrobi ~]# rpm -q --changelog selinux-policy
* Fr Jun 01 2007 Dan Walsh <dwalsh@redhat.com> 2.4.6-75
- Allow samba to remove log files

* Mi Mai 30 2007 Dan Walsh <dwalsh@redhat.com> 2.4.6-74
- Fixes for nagios, postfix, procmail, saslauthd, arpwatch, ➥
         avahi, dovecot

* Mi Mai 23 2007 Dan Walsh <dwalsh@redhat.com> 2.4.6-73
- Allow aixexec to use /tmp
- Allow amanda to read var_lib files
...
```

Ähnliche Informationen finden Sie bei Debian in der Datei `/usr/share/doc/selinux-policy-refpolicy-targeted/changelog.Debian.gz`. Diese geben Ihnen einen ersten Hinweis, ob die Änderungen in der Policy für Sie relevant sein können.

Wenn Sie die alte Policy sichern, bevor Sie die neue Policy auf Ihrem Testsystem installieren, können Sie auch die beiden Versionen miteinander vergleichen. Hierzu dient der Befehl `sediff` aus dem `setools`-Paket. Dieser Befehl zeigt Ihnen die Unterschiede zwischen zwei Policy-Dateien an:

```
[root@supergrobi etc]# sediff /etc/selinux/targeted/policy/ ➥
          policy.21 /etc/selinux-old/targeted/policy/policy.21
Classes (Added 0, Removed 0, Modified 0)
   Added Classes: 0
   Removed Classes: 0
   Modified Classes: 0

Commons (Added 0, Removed 0, Modified 0)
   Added Commons: 0
   Removed Commons: 0
   Modified Commons: 0

Types (Added 0, Removed 18, Modified 0)
   Added Types: 0
   Removed Types: 18
   - aide_db_t
   - aide_exec_t
   - aide_log_t
   - aide_t
...
Roles (Added 0, Removed 0, Modified 5)
   Added Roles: 0
   Removed Roles: 0
   Modified Roles: 5
   * secadm_r (5 Removed Types)
      - aide_t
      - dovecot_deliver_t
...
Booleans (Added 0, Removed 31, Modified 0)
   Added Booleans: 0
   Removed Booleans: 31
   - allow_console_login
   - allow_gpg_execstack
   - allow_mplayer_execstack
   - allow_ptrace
...
TE Rules (Added 235, Added New Type 0, Removed 17550, ➥
          Removed Missing Type 18472
, Modified 526)
   Added TE Rules: 235
```

```
      + allow NetworkManager_t xend_t : dir  getattr ioctl ➥
         lock read search ;
      + allow NetworkManager_t xend_t : file  getattr ioctl ➥
         lock read ;
      + allow NetworkManager_t xend_t : lnk_file  getattr ioctl ➥
         lock read ;
      + allow cardmgr_t xend_t : dir  getattr ioctl lock read ➥
         search ;
      + allow cardmgr_t xend_t : file  getattr ioctl lock read ;
...
```

Dieser Vergleich zweier Policys kann auch grafisch erfolgen. Hierzu dient der Befehl `sediffx` aus dem Paket `setools-gui`. Nach dem Start des Werkzeugs müssen Sie zunächst die beiden Policys laden. Dann können Sie den Diff starten (siehe Abbildung 21.1). Sobald die Analyse vollendet ist, können Sie sich grafisch einen Überblick über die Unterschiede der Policy verschaffen (siehe Abbildung 21.2).

Abbildung 21.1: Während der Diff läuft, zeigt Ihnen sediffx den aktuellen Stand an.

Nach der Analyse der Unterschiede können Sie abwägen, ob Ihr Produktivsystem davon betroffen ist. Dann sollten Sie erst recht diese Funktionen zunächst auf dem Testsystem prüfen.

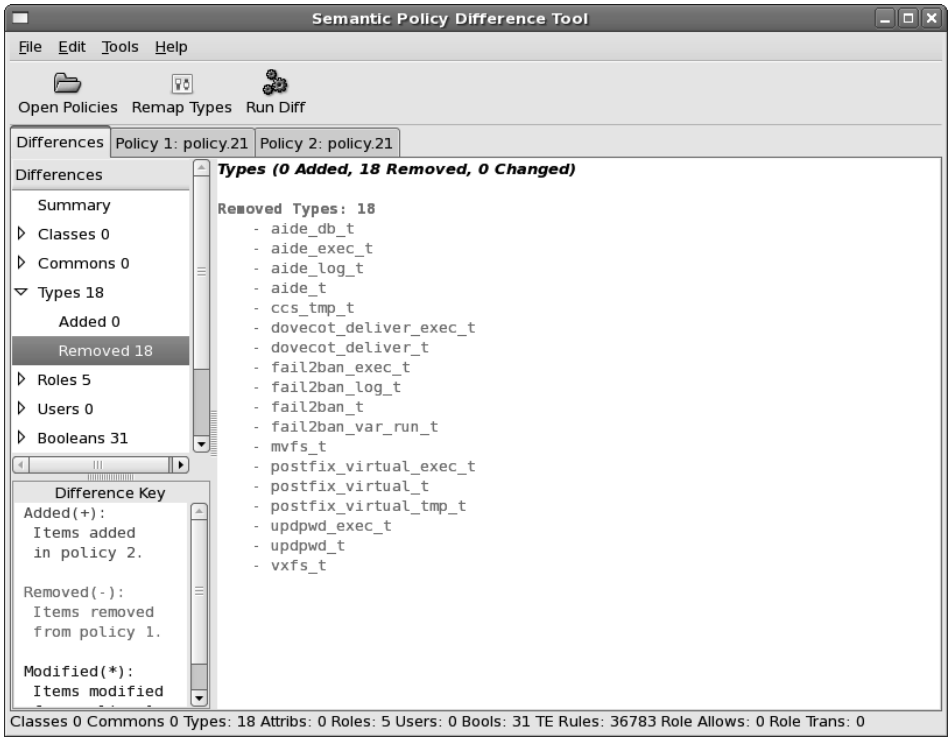

Abbildung 21.2: Der Befehl sediffx zeigt die Unterschiede der Policys grafisch und in Farbe an.

22 SELinux-Installation

In diesem Kapitel werden wir die notwendigen Maßnahmen betrachten, die erforderlich sind, wenn Sie ein Linux-System mit SELinux schützen möchten. Wir werden uns nicht damit beschäftigen können, wie ein *SUSE*-System mit SELinux versehen wird. Die grundsätzliche Unterstützung durch die Distribution muss vorhanden sein. Ansonsten würde die detaillierte Beschreibung der Vorgehensweise den Rahmen dieses Buches sprengen. In diesem Fall müssten auch Befehle wie `ls`, `ps`, `id`, `login` und viele weitere gepatcht, neu übersetzt und ausgetauscht werden. Mir geht es in diesem Kapitel darum, Ihnen zu zeigen, wie Sie zum Beispiel ein *Fedora Core* oder *Debian* Etch-System, auf dem Sie bisher kein SELinux nutzen, mit SELinux sichern.

Da aktuell nur die Fedora Core-, Red Hat Enterprise Linux- und Debian Etch-Distributionen SELinux unterstützen, werde ich mich im Folgenden auf diese Distributionen beschränken. Da die Vorgehensweise bei der Fedora Core- und der Red Hat Enterprise Linux-Distribution identisch ist, fasse ich diese unter Fedora Core zusammen.

22.1 Fedora Core

Die *Fedora Core*Distributionen unterstützen SELinux seit der Version 2. Die Versionen 2 bis 4 verwendeten hierbei die *Example-Policy*. Red Hat Enterprise Linux verwendete diese Policy in der Version RHEL 4. Diese soll hier nicht besprochen werden. Wenn Sie diese Versionen einsetzen und diesbezüglich Hilfe suchen, lesen Sie Kapitel 31.

Hier betrachte ich die Versionen Fedora Core $\geq$ 5 und RHEL $\geq$ 5. Diese Distributionen setzen die aktuelle *Reference-Policy* ein, die von der Firma Tresys[1] entwickelt wurde.

Die Aktivierung von SELinux auf diesen Distributionen ist recht einfach. Zunächst sollten Sie sicherstellen, dass Sie die notwendigen Pakete installiert haben. Dies ist jedoch meist bereits der Fall. Die folgenden Pakete sollten Sie auf Ihrem System wiederfinden:

- `libselinux`
- `selinux-policy`
- `libselinux-python`

[1] *http://www.tresys.com*

- selinux-policy-targeted
- checkpolicy
- policycoreutils
- setools

Nun müssen Sie die Datei `/etc/selinux/config` editieren. In dieser Datei stellen Sie sicher, dass die folgenden Zeilen enthalten sind:

```
SELINUX=permissive
SELINUXTYPE=targeted
```

Erzeugen Sie nun die Datei `/.autorelabel` zum Beispiel mit dem Befehl `touch`. Bevor Sie booten, sollten Sie auch noch Ihre Kernel-Konfiguration prüfen. Wenn Sie einen eigenen Kernel verwenden, ist es wichtig, dass dieser Kernel auch SELinux unterstützt. Bei dem Fedora Core Kernel ist das immer der Fall. Jedoch kann SELinux beim Booten abgeschaltet sein. Prüfen Sie hierzu, ob in der Datei `/boot/grub/menu.lst` auf der entsprechenden Kernel-Zeile die Option `selinux=0` angegeben ist, und entfernen Sie diese Option.

Führen Sie nun einen Reboot durch. Dieser Reboot wird einige Zeit benötigen, da das Dateisystem nun neu gelabelt wird. Diesen Vorgang sollten Sie einmal wiederholen[2]. Hierzu rufen Sie erneut die beiden folgenden Befehle auf:

```
[root@supergrobi ~]# touch /.autorelabel
[root@supergrobi ~]# reboot
```

Nun sollte sich das System in einem definierten Zustand befinden, und SELinux ist im *Permissive*-Modus aktiv. Sie können nun die Protokolle analysieren und Ihre Richtlinien entsprechend modifizieren. Um das System dauerhaft in den Enforcing-Modus zu schalten, editieren Sie die Datei `/etc/selinux/config` und ändern die folgende Zeile:

```
SELINUX=enforcing
```

22.2 Debian Etch

Die *Debian*-Distribution unterstützt ab der Version Etch (4.0) auch SELinux mit der *Reference-Policy*. Erste Unstable-Versionen setzten noch die *Example-Policy* ein. Kurz vor der Veröffentlichung erfolgte jedoch der Wechsel auf die Reference-Policy. Leider ist nach der Installation SELinux zunächst deaktiviert. Es gibt noch einzelne Probleme mit einigen Paketen, die vor einer Aktivierung per Default noch ausgeräumt werden müssen. Dieses Kapitel zeigt Ihnen, welche Änderungen Sie auf einem Debian-System durchspielen müssen, damit es mit SELinux funktioniert. Einige dieser Änderungen werden mit der Zeit sicherlich nicht mehr notwendig sein. Sie können sich

[2] Bevor das *Relabeling* beginnt, werden bereits einige Prozesse gestartet. Die entsprechenden Binärdateien wiesen noch nicht den richtigen Security-Context auf. Daher verfügen diese Prozesse erst nach dem nächsten Reboot über die richtige Domäne.

immer auf *http://wiki.debian.org/SELinux/Setup* ein Bild von dem aktuellen Stand machen. Dieses Kapitel basiert ebenfalls auf den dort hinterlegten Informationen.

Die folgenden Schritte führen auf einem Debian Etch-System zu einer funktionierenden Überwachung durch SELinux.

1. Prüfen Sie zunächst, ob Sie einen Kernel verwenden, der SELinux unterstützt. Wenn Sie einen Debian-Kernel aus den Serien *Etch*, *Testing* oder *Unstable* verwenden, ist dies der Fall. Wenn Sie den Kernel selbst bauen, müssen Sie darauf achten, dass die SELinux-Optionen und `CONFIG_AUDIT`-Optionen aktiviert sind.
2. Prüfen Sie, ob Ihr eingesetztes Dateisystem SELinux unterstützt. Unter Debian sind dies im Moment: `ext2`, `ext3`, `xfs` und `jfs`. Das Dateisystem `reiserfs` unterstützt SELinux nicht!
3. Installieren Sie die benötigten Pakete:

   ```
   apt-get install selinux-basics selinux-policy-refpolicy-targeted
   ```

4. Editieren Sie die Datei `/boot/grub/menu.lst`. SELinux ist zwar im Debian-Kernel enthalten, aber per Default nicht aktiviert. Um SELinux beim Booten zu aktivieren, ergänzen Sie die Kernel-Zeile um die Option `selinux=1`.

   

   Achtung

   Wenn Sie die originale Debian-`menu.lst` verwenden, können Sie die folgende Zeile entsprechend anpassen:

   ```
   # kopt=root=/dev/hda1 ro selinux=1
   ```

   Nun genügt es, den Befehl `update-grub` aufzurufen. Dieser wird alle Kernel entsprechend aktualisieren. Vorsicht: Diese Zeile beginnt mit einem Kommentarzeichen!

5. Nach dem Reboot wird das SELinux-Dateisystem unter `/selinux` gemountet. Erzeugen Sie hierzu den entsprechenden Mountpoint:

   ```
   superdebian:~# mkdir /selinux
   ```

6. Nun müssen Sie noch einige Pakete modifizieren, damit SELinux reibungslos arbeitet.

   (a) PAM
   In der Datei `/etc/pam.d/login` müssen Sie das Kommentarzeichen in der folgenden Zeile entfernen:

   ```
   session required pam_selinux.so multiple
   ```

   Wenn Sie die SSH verwenden, führen Sie diese Änderung auch in der Datei `/etc/pam.d/ssh` durch. Dort müssen Sie aber auch das Schlüsselwort `multiple` entfernen. Die Option `multiple` weist PAM an, den Benutzer zu

fragen, wenn er mehr als eine potenzielle Rolle verwenden kann. Dies ist bei der Anmeldung aus der Ferne nicht erwünscht.

Falls Sie sich grafisch über einen Display-Manager anmelden, müssen Sie auch in dessen PAM-Konfiguration die entsprechende Zeile hinzufügen.

(b) Initscripts
Editieren Sie die Datei `/etc/default/rcS` und setzen Sie die Variable `FSCKFIX=yes`. Falls Sie später auch die Strict-Policy einsetzen möchten, müssen Sie auch die Datei `/etc/init.d/bootmisc.sh` editieren. Suchen Sie dort nach `Update motd` und kommentieren Sie die beiden folgenden Zeilen. Löschen Sie die Dateien `/var/run/motd` und `/etc/motd`. Erzeugen Sie selbst eine neue statische Datei `/etc/motd` mit der Nachricht des aktuellen Tages (Message of the Day). In zukünftigen Versionen wird es auch genügen, die Variable `EDITMOTD=no` in der Datei `/etc/default/rcS` zu setzen.

(c) Postfix
Aktuell existiert keine SELinux-Policy für Exim. Daher wird Postfix als Mailserver beim Einsatz von SELinux empfohlen. Die Postfix-Policy unterstützt jedoch bisher nicht die Verwendung des Chroot. Daher müssen Sie in der Datei `/etc/postfix/master.cf` in der *Chroot*-Spalte (fünfte Spalte) alle `-` durch `n` ersetzen. Zusätzlich sollten Sie die Datei `/etc/default/postfix` erzeugen und die folgende Zeile eintragen:

```
SYNC_CHROOT="n"
```

Anschließend starten Sie Postfix neu.

(d) Statische `tty` und `pty`
Der Betrieb von SELinux ist problematisch, wenn statische Terminal-Dateien verwendet werden. Linux unterstützt dynamische `pty` seit geraumer Zeit, und Debian Etch nutzt diese Funktion. Die Terminals werden dann bei der Erzeugung automatisch richtig gelabelt. Prüfen Sie, ob bei Ihnen das `devpts` Dateisystem verwendet wird. Ist das der Fall, können Sie die statischen Geräte löschen:

```
rm -f /dev/[tp]ty[abcdepqrstuvwxyz][0-9a-f]
```

(e) Udev
Damit auch der `udevd` statische Geräte anlegt, editieren Sie die Datei `/etc/udev/udev.conf`:

```
no_static_dev="1"
```

(f) *Backup*-Cronjob
Dies ist leider unter Debian Etch noch einer der wesentlichen Fehler. In dem Paket `cron` sind auch die auszuführenden Cronjobs enthalten. In der Datei `/etc/cron.daily/standard` werden unter anderem auch wichtige Systemdateien gesichert. Hierzu gehören auch die Dateien `/etc/shadow` und `/etc/gshadow`. Sobald SELinux den *Cron*-Daemon überwacht, ist der Zugriff auf

diese Dateien verboten. Daher sollten Sie diese Datei editieren und folgendermaßen anpassen:

```
# Backup key system files
#

if cd $bak ; then
  cmp -s passwd.bak /etc/passwd || ➥
        (cp -p /etc/passwd passwd.bak && chmod 600 passwd.bak)
  cmp -s group.bak /etc/group || ➥
        (cp -p /etc/group group.bak && chmod 600 group.bak)

  if test -x /usr/sbin/selinuxenabled &&! /usr/sbin/ ➥
        selinuxenabled ; then
    # run only if SELinux disabled
    if [ -f /etc/shadow ] ; then
      cmp -s shadow.bak /etc/shadow || ➥
        (cp -p /etc/shadow shadow.bak && chmod 600 shadow.bak)
    fi
    if [ -f /etc/gshadow ] ; then
      cmp -s gshadow.bak /etc/gshadow || ➥
        (cp -p /etc/gshadow gshadow.bak && chmod 600 ➥
        gshadow.bak)
    fi
  fi
fi
```

(g) Locate-Cronjob
Der Befehl `locate` ist Teil des Pakets `fileutils`. Es ist ein sehr nützliches Werkzeug, um Dateien schnell zu finden. Hierzu verwendet es eine Datenbank, die zuvor angelegt werden muss. Dazu würde der Befehl aber sehr weitreichende SELinux-Privilegien benötigen. Dies wird aktuell nicht empfohlen. Daher sollten Sie die Datei `/etc/cron.daily/find` editieren und entsprechend abändern:

```
#! /bin/sh
exit 0
# Funktioniert nicht mit SELinux
#
#
# cron script to update the 'locatedb' database.
#
...
```

7. Rufen Sie die folgenden Befehle auf:

```
superdebian:~# touch /.autorelabel
superdebian:~# reboot
```

8. Wiederholen Sie den Schritt 7.
9. Prüfen Sie nun den Status Ihrer SELinux-Installation:

```
superdebian:~# check-selinux-installation
A dynamic MOTD is present in /var/run/motd.
```

   Hier wurde noch nicht die Message-of-the-Day-Datei angepasst. Ist alles in Ordnung, sollte der Befehl keinerlei Warnung mehr ausgeben.

23 Sicherer Betrieb eines Webservers mit FastCGI und SELinux

Das Ziel dieses Kapitels ist es, Ihnen zu zeigen, wie Sie mit ein wenig Aufwand dynamische Webseiten mit unterschiedlichen SELinux-Richtlinien überwachen. Hierzu stelle ich im Folgenden zunächst die Schwierigkeiten vor. Anschließend zeige ich mit `mod_fcgid` eine performante Lösung des Problems.

Achtung

Dieses Kapitel setzt teilweise Kenntnisse des Teils IV voraus. Bitte lesen Sie zunächst dort die einleitenden Kapitel.

Der Betrieb eines *Apache* Webservers ist sicherlich einer der häufigsten Gründe für den Einsatz des Betriebssystems Linux. Da ein Webserver seine Anfragen von anonymen Benutzer entgegennimmt, sollten seine Aktivitäten besonders überwacht werden. Hierzu ist ein Mandatory- Access-Control-System wie SELinux besonders geeignet. Alle Distributionen, die SELinux unterstützen, enthalten daher auch eine Policy für den Webserver Apache. Der Apache ist der am häufigsten eingesetzte Webserver auf der Plattform Linux. Die Policy stellt sicher, dass der Webserver-Prozess nur auf die Daten zugreifen kann, die er für seine Ausführung auch benötigt.

Ein Webserver stellt häufig viele verschiedene Funktionen zur Verfügung. So werden häufig mehrere Virtual Hosts durch einen Webserver bedient. Meist werden die Webseiten nicht statisch hinterlegt, sondern durch Programme erzeugt, die von dem Webserver ausgeführt werden. Dabei benötigen die unterschiedlichen Programme häufig unterschiedliche Privilegien.

Wie der Apache, die Scripts und die SELinux-Richtlinien für eine Webanwendung angepasst werden können, wird in Abschnitt 16.5 an einem Beispiel beschrieben.

SELinux erlaubt zwar, ähnlich den AppArmor Hats, den Wechsel einer Domäne durch einen Prozess[1], jedoch unterstützt der Apache Webserver dies bisher nicht. Das bedeutet, dass SELinux nicht die verschiedenen Virtual Hosts eines Apache Webservers unterscheiden kann.

Bei dem Einsatz von dynamischen Webseiten könnte der Eindruck entstehen, dass hier SELinux in der Lage sein sollte, die Scripts zu unterscheiden. Die meisten Scriptsprachen, die für die Entwicklung von dynamischen Webseiten eingesetzt werden (*PHP*, *Perl*, *Ruby*, *Python* etc.), stehen aber als Module für den Apache zur Verfügung. Wenn die Script-Interpreter als Modul geladen werden, werden die Scripts im Kontext des Webservers ausgeführt. Eine Unterscheidung der Scripts und damit die Überwachung der von den Scripts benötigten Privilegien ist nur in der Richtlinie des Webservers möglich. Benötigt ein Script erweiterte Rechte, so erhalten alle Scripts, die im Kontext des Webservers ausgeführt werden, diese Privilegien.

Um die Scripts und die Virtual Hosts doch untereinander zu trennen, müssen die Scripts als CGI-Scripts ausgeführt werden. CGI-Scripts werden von dem Webserver als externe Prozesse aufgerufen und kommunizieren mit dem Webserver über das Common-Gateway-Interface. Die Programmiersprache dieser CGI-Programme ist beliebig. Sie können in jeder Sprache geschrieben werden, die auf dem System ausgeführt werden kann.

Die Verwendung von CGI-Scripts hat jedoch einen entscheidenden Nachteil: geringe Geschwindigkeit. Bei dem Aufruf eines Perl- oder PHP-Scripts als CGI-Programm wird zunächst der Interpreter für die Scriptsprache gestartet, dieser liest das Script ein, parst und kompiliert es möglicherweise vor, bevor er das Script ausführt. Nach der Ausführung werden das Script und der Interpreter beendet. Bei jeder neuen Ausführung muss wieder zunächst der Interpreter geladen werden, der anschließend das Script lädt und ausführt. Werden die Script-Interpreter als Modul in den Apache geladen (`mod_perl`, `mod_python` und `mod_php`), entfällt das Laden und Starten des Interpreters in jedem Fall. Zusätzlich können die Module die Scripts auch zwischen den Aufrufen speichern. Hier können dann auch die vorkompilierten Versionen gecacht werden. Dadurch kann die Ausführung um das bis zu 100-fache gesteigert werden. Zusätzlich können die Scripts bestimmte Werte über mehrere Scriptaufrufe hinweg speichern. Dies erlaubt sehr einfach die Überwachung des Zustands der Sitzungen.

Heute werden daher meist diese Scripts über die als Modul in den Apache eingebauten Interpreter aufgerufen. Dann kann aber leider keine Unterscheidung seitens SELinux erfolgen. SELinux bietet diese Unterscheidung nur für CGI-Scripts. Einen Ausweg aus der Misere bietet die *FastCGI*-Schnittstelle mit dem Apache-Modul `mod_fcgid`, die im Folgenden dargestellt wird.

[1] Das Interface `domain_dyntrans_type` erlaubt einer Domäne, die Domäne zu wechseln. Der Bibliotheksaufruf `setcon(3)` erlaubt, den Context des Prozesses zu setzen.

23.1 FastCGI mit mod_fcgid

FastCGI ist ein Standard, um Programme zur Erzeugung dynamischer Webseiten in einen Webserver einzubinden. FastCGI ist damit vergleichbar zum Common Gateway Interface (CGI). Es umgeht jedoch dessen Performance-Probleme, indem FastCGI das auszuführende Programm (inklusive Interpreter) nur einmal lädt. Dieses steht dann für mehrere Requests zur Verfügung. Hierbei ist es egal, ob die weiteren Anfragen von demselben Client oder von unterschiedlichen Clients erfolgen. Zur Kommunikation mit dem Webserver nutzt FastCGI Unix-Domain-Sockets oder TCP-Netzwerkverbindungen und nicht wie CGI Umgebungsvariablen und die Standard-Ein- und -Ausgabe. Das bedeutet, dass das FastCGI-Programm sogar auf einem anderen Rechner laufen kann.

Mit FastCGI sind die folgenden Funktionen möglich:

- Einfacher Betrieb mehrerer Versionen eines Interpreters auf derselben Maschine (z.B. PHP4 und PHP5)
- Gleichzeitiger Betrieb unterschiedlich kompilierter Interpreter derselben Version (z.B. PHP5) mit unterschiedlichen `php.ini`-Konfigurationsdateien
- Start eines Interpreters über *SuExec* mit unterschiedlichen Benutzern und Gruppen
- Betrieb des Apache Webservers mit dem *Worker-MPM*. Dieses Multi-Processing-Modul erlaubt eine wesentlich bessere Skalierung durch den Apache und beschleunigt den Zugriff auf die Webseiten enorm. Leider kann PHP als `mod_php` noch nicht grundsätzlich mit dem Worker-MPM gemeinsam betrieben werden. Dies muss von Fall zu Fall getestet werden, da die PHP-Erweiterungen nicht alle threadsafe sind. Bei dem Betrieb über FastCGI tritt dieses Problem nicht auf.
- Überwachung der unterschiedlichen Interpreter oder auch eines Interpreters mit verschiedenen SELinux-Richtlinien[2]

Die Homepage von FastCGI (*http://www.fastcgi.com/*) hält Implementierungen für alle bedeutenden Webserver vor. Dennoch sollten Sie nicht das originale FastCGI-Modul für den Apache Webserver nutzen. In fast allen Distributionen ist inzwischen das Modul `mod_fcgid` des Chinesen Pan Qingfeng enthalten. Dies ist ein stabiles, zum `mod_fastcgi`-Modul binär-kompatibles Modul. Es wird unter *http://fastcgi.coremail.cn/* gepflegt. Dieses Modul ist auch in der Fedora-Distribution enthalten (teilweise in Extras).

Für die folgenden Schritte benötigen Sie:

- Apache 2.x Webserver
- `mod_fcgid`

[2] Das wollen wir am Ende dieses Kapitels erreichen!

- PHP mit `--enable-fastcgi` übersetzt[3]
- *SuExec*[4]

Zunächst werden wir in diesem und den folgenden Abschnitten das System so konfigurieren, dass Sie eine funktionstüchtige FastCGI-Installation besitzen. Dabei werden wir die Konfiguration am Beispiel eines Virtual Hosts durchspielen. Dies ist aber nicht unbedingt erforderlich. Im letzten Schritt werden wir dann die SELinux-Richtlinien entsprechend anpassen.

23.1.1 Konfiguration des Apache und mod_fcgid

Da die Konfiguration nicht einfach ist, sollten Sie die Schritte einzeln nachvollziehen und testen. Für diese Zeit sollte SELinux deaktiviert sein (*Permissive* Mode)!

Um `mod_fcgid` zu verwenden, sollten Sie zunächst das Paket installieren. Ich werde es hier am Beispiel der Debian Etch- und Fedora Core 6-Distribution vorführen. Installieren Sie zunächst die folgenden Pakete:

- Fedora Core 6
  - `httpd`
  - `php-cli`
  - `mod_fcgid`
  - `mod_fcgid-selinux`
- Debian Etch
  - `apache2`
  - `apache2-mpm-worker`
  - `php5-cgi`
  - `libapache2-mod-fcgid`

Auf einem Fedora Core-System müssen Sie nun den Apache so konfigurieren, dass er auch das *Worker-MPM* verwendet. Dieses Multi-Processing-Modul ist wesentlich performanter. Stoppen Sie hierzu den Apache, falls er laufen sollte, und editieren Sie die Datei `/etc/sysconfig/httpd`. Entfernen Sie das Kommentarzeichen in der entsprechenden Zeile:

```
# The default processing model (MPM) is the process-based
# 'prefork' model.  A thread-based model, 'worker', is also
# available, but does not work with some modules (such as PHP).
# The service must be stopped before changing this variable.
#
HTTPD=/usr/sbin/httpd.worker
```

[3] Dies ist bei den Paketen von Fedora Core 6 (`php-cli`) und Debian Etch (`php5-cgi`) der Fall. Sie können das aber auch testen, indem Sie den folgenden Befehl ausführen: `/usr/bin/php-cgi -i | grep Fast`. Wenn Ihnen hier `CGI/FastCGI` angezeigt wird, können Sie das PHP-Paket verwenden.

[4] *SuEcex* ist bei den meisten Distributionen im Apache-Paket enthalten.

Wenn Sie nun den Apache neu starten, sollte er das Worker-MPM verwenden. Falls dabei die folgende Fehlermeldung auftritt, müssen Sie das PHP-Modul in der Apache-Konfiguration noch deaktivieren:

```
[root@supergrobi ~]# /etc/init.d/httpd start
httpd starten: [Sun Apr 22 04:08:22 2007] [crit] Apache is running ↵
         a threaded MPM, but your PHP Module is not compiled to be ↵
         threadsafe. You need to recompile PHP.
Pre-configuration failed                [FEHLGESCHLAGEN]
```

Hierzu wechseln Sie in das Verzeichnis `/etc/httpd/conf.d` und löschen die Datei `php.conf` oder benennen diese so um, dass die Endung nicht mehr `.conf` lautet. Der Apache wird die Datei und damit auch das in dieser Datei angegebene Modul nicht mehr laden. Bei dieser Gelegenheit deaktivieren Sie am besten auch SSL, da dies bei der im Folgenden gewählten Virtual-Host-Konfiguration zu Konflikten führen kann.

Auf einem Debian Etch-System ist dies nicht erforderlich. Die Auswahl des richtigen MPMs erfolgt durch unsere Paketwahl.

Nun müssen wir das Modul `mod_fcgid` einbinden. Dies ist bei Debian wieder sehr einfach. Hier ist das Modul direkt nach der Installation eingebunden. Sie können das überprüfen, indem Sie die Links in dem Verzeichnis `/etc/apache2/mods-enabled` kontrollieren. Sollte hier keine Verknüpfung zu den entsprechenden Dateien in dem Verzeichnis `/etc/apache2/mods-available` existieren, können Sie dies mit den Befehlen `a2enmod` und `a2dismod` unter Debian korrigieren. Zusätzlich müssen Sie noch die Datei `/etc/apache2/mods-available/fcgid.conf` anpassen und die Dateiendung für den `fcgid`-Handler korrigieren:

```
<IfModule mod_fcgid.c>
  AddHandler fcgid-script .php
  SocketPath /var/lib/apache2/fcgid/sock
  IPCConnectTimeout 20
</IfModule>
```

Auf einem Fedora Core-System sollten Sie in dem Verzeichnis `/etc/httpd/conf.d` eine Datei `fcgid.conf` finden. Passen Sie diese Datei so an, dass nur noch die folgenden Zeilen enthalten sind:

```
LoadModule fcgid_module modules/mod_fcgid.so

<IfModule !mod_fastcgi.c>
  SocketPath run/mod_fcgid
  SharememPath run/fcgid_shm

  <IfModule mod_fcgid.c>
     AddHandler fcgid-script .php
  </IfModule>
</IfModule>
```

Jetzt benötigen wir zunächst eine PHP-Datei, die wir anschließend auch ausführen können. Zunächst soll hier einfach eine kleine Datei `index.php` genutzt werden, die folgenden Inhalt hat:

```
<? phpinfo(); ?>
```

Diese Datei sollte für einen Virtual Host in einer Shared-Hosting-Umgebung angezeigt werden. Hierzu muss nun ein Virtual Host angelegt werden. Dafür erzeugen wir zunächst ein Verzeichnis in `/var/www`. Bitte verwenden Sie zunächst keinen anderen Pfad, da ansonsten die Verwendung des *SuExec* im nächsten Schritt fehlschlägt. Ich werde dort genauer erläutern, warum das so ist.

```
[root@supergrobi ~]# cd /var/www
[root@supergrobi www]# mkdir myvhost
[root@supergrobi www]# echo '<? phpinfo(); ?>' > myvhost/index.php
```

Nun benötigen wir die Konfigurationsdatei für den Virtual Host. Legen Sie diese bei Debian in dem Verzeichnis `/etc/apache2/sites-available` an.

Bei Fedora Core könnten Sie die Datei mit Einschränkungen in dem Verzeichnis `/etc/httpd/conf.d` anlegen. Ich selbst bevorzuge bei Fedora Core, ähnlich der Debian-Konfiguration, jedoch ein eigenes Verzeichnis `/etc/httpd/vhosts.d` für alle Virtual Hosts. Hierzu legen Sie bitte das Verzeichnis an und fügen eine entsprechende Include-Direktive an die Konfigurationsdatei `httpd.conf` an:

```
[root@supergrobi www]# cd /etc/httpd/
[root@supergrobi httpd]# mkdir vhosts.d
[root@supergrobi httpd]# echo 'NameVirtualHost *' >> /etc/httpd/ ➥
         conf/httpd.conf
[root@supergrobi httpd]# echo 'Include vhosts.d/*.conf' >> /etc/ ➥
         httpd/conf/httpd.conf
```

Legen Sie hier die Konfigurationsdatei mit dem Namen `myvhost` (Debian) bzw. `myvhost.conf` (Fedora) und dem folgenden Inhalt an:

```
<Virtualhost *>
  DocumentRoot /var/www/myvhost
  ServerName www.myvhost.com
  DirectoryIndex index.php
  <Directory /var/www/myvhost>
     FCGIWrapper /var/www/myvhost/php-fcgi .php
     Options +ExecCGI
     order allow,deny
     allow from all
  </Directory>
</Virtualhost>
```

Die wichtigen Parameter in dieser Datei sind:

- `FCGIWrapper` Dieser Parameter definiert ein Script, das für jede `.php`-Datei aufgerufen werden soll. Dieser Parameter kann natürlich auch für unterschiedliche Verzeichnisse unterschiedlich gesetzt werden.
- `Options +ExecCGI` Hiermit erlauben wir in diesem Verzeichnis den Aufruf von CGI-Scripts.

Auf einem Fedora Core-System muss auch der Parameter `NameVirtualHost` aktiviert werden. Dies ist bereits weiter oben passiert. Bei Debian ist der Parameter bereits in der Datei `/etc/apache2/sites-available/default` eingetragen. Bei der Debian-Distribution müssen Sie nun nur noch die angelegte Website aktivieren:

```
# a2ensite myvhost
Site myvhost installed; run /etc/init.d/apache2 reload to enable.
```

Jetzt benötigen wir noch den *FCGIWrapper*, der in der Konfiguration angegeben wurde. Diesen legen Sie mit dem angegebenen Namen (`/var/www/myvhost/php-fcgi`) an:

```
#!/bin/sh
PHPRC='/etc/'
export PHPRC
exec /usr/bin/php-cgi
```

Mit dem Parameter `PHPRC` geben Sie das Verzeichnis an, in dem der PHP-Interpreter seine Konfigurationsdatei `php.ini` sucht. Hiermit können Sie also unterschiedliche Konfigurationsdateien für jeden Virtual Host und für jedes Verzeichnis angeben.

Achtung

Geben Sie hier nicht den kompletten Pfad zur Datei `php.ini` an, sondern lediglich das Verzeichnis, in dem sich die Datei befindet.

Auf einem Debian Etch-System müssen Sie die Datei anpassen:

```
#!/bin/sh
PHPRC='/etc/php5/cgi'
export PHPRC
exec /usr/bin/php5-cgi
```

Diese Datei muss nun auch noch ausführbar sein:

```
[root@supergrobi ~]# chmod 755 /var/www/myvhost/php-fcgi
```

Damit der Zugriff auf den Virtual Host nun auch noch funktioniert, tragen Sie testweise den Namen in der Datei `/etc/hosts` ein:

```
127.0.0.1 localhost www.myvhost.com
```

Nun sollten Sie mit einem Browser (z.B. Firefox) auf den Virtual Host zugreifen können. Tragen Sie dazu als Adresse *http://www.myvhost.com* ein. Anschließend sollten Sie die PHP-Info- Webseite sehen (siehe Abbildung 23.1).

Abbildung 23.1: Die PHP-Info-Webseite wird nun über die FastCGI-Schnittstelle aufgerufen. Das exakte Aussehen hängt von der Distribution ab (hier: Debian Etch).

23.2 SuExec und mod_fcgid

Mit der FastCGI-Schnittstelle kann der PHP-Interpreter auch für unterschiedliche Scripts mit unterschiedlichen Benutzern aufgerufen werden. So können Sie bereits mit den Rechten des UNIX-Dateisystems die Möglichkeiten der Scripts einschränken. Werden die Scripts über modulare Interpreter in dem Apache aufgerufen, so

müssen sie alle denselben Benutzerkontext verwenden. Als CGI-Script können sie über *SuExec* mit einem jeweils anderen Benutzer aufgerufen werden.

Die Konfiguration ist jetzt sehr einfach. Rufen Sie als Erstes den Befehl `suexec` auf. Während unter Fedora Core der Befehl im Pfad gefunden wird, müssen Sie auf Debian Etch den korrekten Pfad angeben:

```
superdebian:~# /usr/lib/apache2/suexec -V
 -D AP_DOC_ROOT="/var/www"
 -D AP_GID_MIN=100
 -D AP_HTTPD_USER="www-data"
 -D AP_LOG_EXEC="/var/log/apache2/suexec.log"
 -D AP_SAFE_PATH="/usr/local/bin:/usr/bin:/bin"
 -D AP_UID_MIN=100
 -D AP_USERDIR_SUFFIX="public_html"
```

Wichtig in der Ausgabe sind die folgenden Angaben:

- `AP_DOC_ROOT`: *SuExec* erlaubt nur den Aufruf von Befehlen in diesem Verzeichnisbaum. Daher haben wir unser `DocumentRoot` in diesem Verzeichnisbaum angelegt.
- `AP_(UID,GID)_MIN`: *SuExec* wechselt für den Aufruf des Programms den Benutzer und die Gruppe. Die UID und GID müssen mindestens den Wert 100 haben.
- `AP_LOG_EXEC`: In diesem Protokoll werden die Aufrufe und Fehler protokolliert.

Erzeugen Sie nun eine Gruppe und einen Benutzer für den *SuExec*-Aufruf (bei Debian verzichten Sie auf die Option `-M` bei dem `useradd`-Aufruf):

```
[root@supergrobi ~]# groupadd -g 10000 phpinfo
[root@supergrobi ~]# useradd -g phpinfo -s /bin/false -u 10000 ➥
          -M -d /var/www phpinfo
```

Nun müssen Sie die Konfiguration des Virtual Hosts anpassen und *SuExec* aktivieren:

```
<Virtualhost *>
  DocumentRoot /var/www/myvhost
  ServerName www.myvhost.com
  SuexecUserGroup phpinfo phpinfo
  DirectoryIndex index.php
....
```

Achtung

Wenn sich der Debian Etch-Webserver bei einem Neustart sich über den neuen Parameter beschwert, müssen Sie noch das *SuExec*-Modul aktivieren: `a2enmod suexec`.

SuExec verlangt, dass auch das aufzurufende Script und das Verzeichnis diesem Benutzer und der Gruppe gehört. Wenn das nicht der Fall ist, kommt es zu einem Internal Server Error, und *SuExec* protokolliert in seinem Log:

```
[2007-04-22 05:41:22]: uid: (10000/phpinfo) gid: (10000/10000) ⬅
        cmd: php-fcgi
[2007-04-22 05:41:22]: target uid/gid (10000/10000) mismatch with ⬅
        directory (0/0) or program (0/0)
```

Korrigieren Sie dies mit zwei Zeilen:

```
[root@supergrobi ~]# chown phpinfo.phpinfo /var/www/myvhost/
[root@supergrobi ~]# chown phpinfo.phpinfo /var/www/myvhost/php-fcgi
```

Wenn Sie nun auf die Webseite zugreifen und anschließend die gestarteten Prozesse betrachten, sollten die PHP-Prozesse dem Benutzer *phpinfo* gehören:

```
# ps -ef | grep php
phpinfo   6939  6873  0 17:48 ?        00:00:00 /usr/bin/php5-cgi
```

Weitere Hinweise zu *SuExec* finden Sie in der Apache-Online-Dokumentation[5].

23.3 SELinux und mod_fcgid

Da wir nun für jeden Virtual Host oder sogar für jedes PHP-Script[6] einen eigenen `FCGIWrapper` definieren können, können diese Prozesse dann auch über eigene Richtlinien überwacht werden.

Um SELinux zu aktivieren, benötigen wir zunächst die Richtlinien für den Webserver Apache und für `mod_fcgid`. Bei Fedora Core sind letztere Richtlinien in einem eigenen Paket verfügbar. Dieses Paket kann verwendet werden. Im Folgenden werden wir aber die Richtlinien auch für `mod_fcgid` selbst bauen, da wir sie unter Debian sowieso brauchen.

[5] *http://httpd.apache.org/docs/2.2/suexec.html#model*

[6] Um für jedes PHP-Script einen eigenen *FCGIWrapper* zu definieren, können Sie die `<files>`-Container verwenden.

Auf dem Debian Etch müssen Sie nun sicherstellen, dass das SELinux-Paket für den Webserver geladen ist. Falls es vorher noch nicht geladen war, müssen Sie anschließend die Dateien neu labeln. Hierzu legen Sie die Datei `/.autorelabel` an und booten das System. Auf einem Fedora-System ist die Apache-Richtlinie im Base-Modul der Targeted-Policy enthalten.

```
superdebian:~# semodule -i /usr/share/selinux/refpolicy-targeted/ ↵
         apache.pp
superdebian:~# touch /.autorelabel
superdebian:~# reboot
```

Zunächst benötigen wir noch eine SELinux-Richtlinie für `mod_fcgid`. Diese Richtlinie muss den FastCGI-Socket mit dem richtigen Kontext versehen und dem Webserver den Zugriff auf den Socket erlauben. Hierzu benötigen wir eine TE-Datei:

```
policy_module(fcgid, 1.0.0)

type httpd_fastcgi_sock_t;
files_type(httpd_fastcgi_sock_t)

require
        type httpd_t;
;

# Allow httpd to create and use sockets for communicating with ↵
         mod_fcgid
allow httpd_t httpd_fastcgi_sock_t:dir  rw_dir_perms setattr ;
allow httpd_t httpd_fastcgi_sock_t:sock_file  create_file_perms ;
```

Zusätzlich benötigen wir eine FC-Datei:

```
/var/run/mod_fcgid(/.*)?  gen_context(system_u:object_r: ↵
         httpd_fastcgi_sock_t,s0)
```

Achtung

Bei der Debian Etch-Distribution befindet sich dieses Verzeichnis an anderer Stelle, und zwar in `/var/lib/apache2/fcgid`. Dementsprechend müssen Sie die FC-Datei anpassen!

Aus diesen beiden Dateien bauen Sie ein SELinux-Package. Wie das bei Ihrer Distribution funktioniert, lesen Sie in Abschnitt 24.4.

Nun möchten wir, dass die Scripts, die über unseren FastCGI-Wrapper aufgerufen werden, durch eine eigene Richtlinie überwacht werden. Hierzu benötigen wir eine *Domänentransition*. Bei dem Aufruf des Wrappers muss eine Transition aus der Domäne *httpd_t* in eine neue Domäne, die wir definieren, erfolgen. Für diesen Zweck gibt es vorbereitete Schnittstellen in der Apache-Richtlinie. Die Schnittstelle `apache_content_template` erzeugt die entsprechenden Typen und Regeln, um automatisch beim Aufruf der Scripts eine Domänentransition durchzuführen. Hierzu definiert die Schnittstelle bei Angabe von *name* die Typen *httpd_name_script_t* und *httpd_name_script_exec_t*. Zusätzlich definiert diese Schnittstelle auch alle erforderlichen Typen für normale Webseiten, Konfigurationsdateien und veränderliche Dateien. Lesen Sie sich die Schnittstellenbeschreibung und die Regeln in der Datei `apache.if` durch, um einen Eindruck von der Funktion zu erhalten.

Um nun für unseren Virtual Host eine eigene Domäne zu erzeugen, die mit einer eigenen Richtlinie überwacht werden kann, nutzen wir die folgende TE-Datei (`phpinfo.te`):

```
policy_module(phpinfo,1.0.0)
apache_content_template(phpinfo)
```

Damit die Dateien auch den richtigen Kontext erhalten, nutzen wir die folgende FC-Datei (`phpinfo.fc`):

```
/var/www/myvhost(/.*)?  gen_context(system_u:object_r: ➥
         httpd_phpinfo_content_t,s0)
/var/www/myvhost/php-fcgi -- gen_context(system_u:object_r: ➥
         httpd_phpinfo_script_exec_t,s0)
```

Nachdem Sie auch dieses Modul gebaut und geladen haben, können Sie die Dateien Ihres Virtual Hosts entsprechend relabeln:

```
[root@supergrobi ~]# restorecon -FR /var/www/myvhost/
[root@supergrobi ~]# restorecon -FR /var/run/mod_fcgid
```

Die Option `-F` erzwingt dabei auch das Relabeln von Dateien, die einen *Customizable Types* (siehe Abschnitt 16.4) aufweisen.

Nun müssen Sie den Webserver neu starten. Bei einem erneuten Zugriff auf das PHP-Script sollte nun der PHP-Interpreter in unserer selbst definierten Domäne *httpd_phpinfo_script_t* arbeiten. Denken Sie daran, dass sich SELinux immer noch im *Permissive*-Modus befinden sollte. Dies sollten Sie mit `ps` überprüfen:

```
[root@supergrobi fastcgi]# ps -efZ | grep php
root:system_r:httpd_phpinfo_script_t phpinfo 19886 19820 ➥
         1 06:24 ?    00:00:00 /usr/bin/php-cgi
```

Natürlich müssen Sie die Richtlinien nun noch für Ihre Applikation entsprechend erweitern, aber mit diesen grundsätzlichen Werkzeugen sollten Sie in der Lage sein, die unterschiedlichen Scripts zur Erzeugung von dynamischen Webseiten bei Bedarf in unterschiedlichen Domänen zu überwachen.

23.4 Geschwindigkeit von mod_fcgid und SELinux

Sicherlich hat sich der eine oder andere Leser bereits gefragt, ob das Ganze denn überhaupt Sinn macht. Wie schnell kann dieses Konstrukt noch sein? Dies können Sie recht einfach selbst herausfinden. Hierzu führen Sie zwei Tests durch.

Sie müssen die Geschwindigkeit des Apache-*Prefork-MPM* mit `mod_php` mit der Geschwindigkeit des Apache-*Worker-MPM* mit FastCGI-PHP vergleichen. Dabei sollten Sie den Zugriff sowohl auf statische Elemente als auch auf dynamisch generierte Elemente messen. Auch wenn Sie eine dynamisch generierte Webseite nutzen, sind viele Elemente (wie z.B. Bilder) häufig immer noch statisch.

Ich habe hierzu ein Bild in das *DocumentRoot*-Verzeichnis des Apache kopiert und mit dem Werkzeug *ApacheBench* die Geschwindigkeit des Apache gemessen. Es handelte sich bei diesem Versuch nicht um einen wissenschaftlichen und reproduzierbaren Benchmark. Auch habe ich im Vorfeld den Apache nicht besonders konfiguriert, sondern die Default-Werte der Debian-Distribution beibehalten. Lediglich das FastCGI-Wrapper-Script wurde modifiziert:

```
#!/bin/sh
PHPRC='/etc/php5/cgi'
export PHPRC
PHP_FCGI_CHILDREN=8
export PHP_FCGI_CHILDREN
PHP_FCGI_MAX_REQUESTS=5000
export PHP_FCGI_MAX_REQUESTS
exec /usr/bin/php5-cgi
```

Die Messungen erfolgten mit folgenden Befehlen:

```
superdebian:~# ab -n4000 http://www.myvhost.com/python.gif
superdebian:~# ab -n4000 http://www.myvhost.com/index.php
```

Es wurden die beantworteten Anfragen je Sekunde gemessen. Das lässt den Schluss

Anfrage	Prefork	Worker
`python.gif`	3500	5000
`index.php`	530	500

zu, dass es hier bei diesem einfachen lokalen Test kaum Unterschiede gibt. Bei dem Betrieb auf produktiven Systemen stellt man fest, dass der Apache-MPM-Worker mit FastCGI-PHP durchaus auch Vorteile in der Geschwindigkeit entwickeln kann.

Die Sicherheit ist speziell mit SELinux auf jeden Fall stark verbessert. SELinux kann nun ähnlich AppArmor die unterschiedlichen Virtual Hosts auf einem Webserver mit getrennten Richtlinien überwachen.

24 Die erste eigene Policy

In diesem Kapitel werden wir unsere erste eigene Policy für eine neue Applikation erzeugen. Nein, wir werden nicht eine komplette Policy für ein ganzes System aus der Taufe heben. Dies würde den Rahmen des Buches sprengen und macht auch für die meisten Anwender keinen Sinn (siehe jedoch Kapitel 29).

Basierend auf der mitgelieferten *Targeted-Policy* unserer Distribution werden wir nun für einen Befehl, der im Moment noch nicht überwacht wird, eine eigene Policy entwickeln und hierbei alle einzelnen Schritte durchlaufen. Diese Policy sollte aber anschließend auch bei Verwendung der *Strict-Policy* fast unverändert funktionieren (siehe Abschnitt 28.1).

Hierzu habe ich für dieses Kapitel eine ganz einfache Applikation ausgewählt: `date`. Der Befehl `date` hat zwei ganz einfache Aufgaben:

- Anzeige des Datums und der Uhrzeit
- Stellen des Datums und der Uhrzeit

Wir werden nun eine Policy erzeugen, die den Befehl `date` gesondert überwacht und seine Aktionen prüft.

Einige Leser mögen nun enttäuscht sein, dass ich nicht mit einer Policy für ein Oracle-Database-Management-System beginne. Die hier vorgestellten Schritte lassen sich aber so auf alle weiteren Programme übertragen. Im weiteren Verlauf werde ich Ihnen auch zeigen, wie Sie eine Policy für Netzwerkdienste und kompliziertere Applikationen erzeugen.

Die folgenden Schritte sollten unabhängig von der eingesetzten Distribution funktionieren. Wichtig ist jedoch, dass diese Distribution die modulare *Reference-Policy* verwendet. Für Anwender der Example-Policy (Fedora Core 3, 4 und Red Hat Enterprise Linux 4) werden Hinweise in dem Kapitel 31 gegeben. Dennoch ist es sinnvoll, dass Sie dieses Kapitel zunächst lesen.

Achtung

Ich werde hier die *Fedora Core*-Distribution voraussetzen. Die Beispiele wurden unter Fedora Core umgesetzt. Falls Sie dies unter Debian durchführen möchten, sollten Sie zunächst den folgenden Punkt beachten.

Sie benötigen das Paket `selinux-policy-refpolicy-dev`. Dieses Paket enthält die Scripts und Befehle für die Erzeugung der Richtlinien. Wenn es noch nicht installiert sein sollte, benötigen Sie auch noch das Paket für den Makroprozessor *M4* (`m4`). Im Weiteren können sich einige Ausgaben der Befehle und einige Pfade unterscheiden. Wundern Sie sich bitte nicht.

Bei Einsatz der Debian-Distribution sollten Sie beachten, dass *Debian* Etch aktuell noch nicht den Audit-Daemon enthält. Daher erfolgt die SELinux-Protokollierung in der Datei `/var/log/syslog`.

Bevor Sie beginnen, stellen Sie bitte sicher, dass das Paket `selinux-policy-devel` installiert ist. Wir benötigen die in diesem Paket vorhandenen Dateien, um weitere Module für die geladene Policy zu entwickeln.

24.1 Start

Um nun mit der Entwicklung zu beginnen, benötigen wir zunächst drei Dateien, deren Namen sich lediglich in der Endung unterscheiden:

- `date.te`
- `date.fc`
- `date.if`

Die Datei `date.te` enthält später die *Type-Enforcement*- Regeln. In der Datei `date.fc` speichern wir die File-Context-Informationen, und die Datei `date.if` definiert eine Schnittstelle, sodass andere SELinux-Module die hier definierten Funktionen nutzen können. Zunächst beschäftigen wir uns nur mit den ersten beiden Dateien.

Erzeugen Sie sich hierzu an geeigneter Stelle ein Verzeichnis, und legen Sie zunächst die drei Dateien leer an:

```
[root@supergrobi ~]# mkdir selinux-date
[root@supergrobi ~]# cd selinux-date/
[root@supergrobi selinux-date]# touch date.te
[root@supergrobi selinux-date]# touch date.fc
[root@supergrobi selinux-date]# touch date.if
```

24.2 Domänen und Typen

Wir erzeugen zunächst nur ein Policy-Gerüst, das wir anschließend schrittweise mit den Richtlinien füllen. Damit der Prozess `date` von SELinux überwacht werden kann, müssen wir für diesen Prozess eine eigene Domäne definieren. Erinnern Sie sich: Eine Domäne ist das Gleiche wie ein Typ. Der Unterschied ist zunächst nur sprachlicher Natur.

Die Definition des Typs *date_t* erfolgt in der Datei `date.te`. Wie gelingt es uns nun, dafür zu sorgen, dass bei dem Aufruf des Befehls `date` der entstehende Prozess in der Domäne `date_t` gestartet wird? Wir benötigen eine *Domänentransition*. Ruft ein Benutzer den Befehl `date` auf, so würde dieser per Default in der Domäne des Benutzers gestartet werden. Nun soll automatisch ein Wechsel in die Domäne *date_t* erfolgen. Um diese Domänentransition nur bei dem Aufruf des Befehls `/bin/date` auszulösen, benötigen wir noch einen weiteren Typ für die Binärdatei `/bin/date`. In Anlehnung an die restliche Policy verwenden wir hier *date_exec_t*.

Wir benötigen also zwei Typen in der Datei `date.te`:

```
policy_module(date,1.0.0)

type date_t;
type date_exec_t;
```

Die erste Zeile definiert den Namen und die Version des Moduls. Diese werden später von dem Befehl `semodule -l` angezeigt. Die beiden nächsten Zeilen definieren die Typen *date_t* und *date_exec_t*.

Nun müssen wir noch die Policy davon in Kenntnis setzen, dass der erste Typ eine Domäne ist. Der zweite Typ wird verwendet, um einen Wechsel der Domäne zu erzwingen. Dies könnten wir in expliziten SELinux-Regeln beschreiben. Schöner ist es jedoch, wenn wir Schnittstellen der geladenen Policy verwenden. Um auf diese Schnittstellen zuzugreifen, musste das Paket `selinux-policy-devel` installiert werden. In diesem Paket finden Sie unter `/usr/share/selinux/devel/include/*/` die Schnittstellen der SELinux Policy. Für den Umgang mit Domänen befindet sich die Schnittstelle in `/usr/share/selinux/devel/include/kernel/domain.if`[1]. Wenn Sie diese Datei öffnen und lesen, finden Sie dort reichlich Kommentare und Schnittstellen-Abschnitte (*Interface*). Zunächst benötigen wir eine Möglichkeit, um einen Typ als Domäne einsetzen zu können. Wir finden die Schnittstelle mit dem Namen `domain_type`:

```
########################################
## <summary>
##      Make the specified type usable as a domain.
## </summary>
## <param name="type">
```

[1] Wie Sie weitere Schnittstellen finden und zuordnen, zeige ich Ihnen weiter unten.

```
##      <summary>
##      Type to be used as a domain type.
##      </summary>
## </param>
#
interface(‘domain_type’,‘
        # start with basic domain
        domain_base_type($1)

...
```

Wir benötigen eine zweite Schnittstelle, um den Typ *date_exec_t* als Eintrittspunkt in die Domäne *date_t* zu definieren. Hier benutzen wir die Schnittstelle `domain_entry_file`. Eine dritte Schnittstelle (`domain_auto_trans`) erlaubt der Domäne *unconfined_t* tatsächlich den Wechsel in die Domäne *date_t* bei dem Aufruf einer Datei vom Typ *date_exec_t*.

Unsere Datei `date.te` sieht nun folgendermaßen aus:

```
policy_module(date,1.0.0)

type date_t;
type date_exec_t;

domain_type(date_t)
domain_entry_file(date_t, date_exec_t)
domain_auto_trans(unconfined_t, date_exec_t, date_t)
```

Achtung

Achten Sie auf die Syntax in der TE-Datei. Echte SELinux-Policy-Anweisungen werden mit einem Semikolon abgeschlossen, während Schnittstellenaufrufe ohne Semikolon geschrieben werden.

Hinweis

Sie können sich diese Schnittstellen merken. Diese benötigen Sie bei jeder neuen Policy.

24.3 File-Contexts

Nun müssen wir noch dafür sorgen, dass die Binärdatei `/bin/date` den richtigen *Security-Context* erhält. Hierfür ist die Datei `date.fc` verantwortlich. In dieser Datei werden alle Dateikontexte definiert, die für die Funktion der Applikation erforderlich sind. Da der Befehl `date` ohne Protokoll-, Konfigurations- und weitere temporäre Dateien auskommt, genügt hier ein Eintrag in der Datei `date.fc`:

```
# date executable will have:
# label: system_u:object_r:date_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/bin/date   --   gen_context(system_u:object_r:date_exec_t,s0)
```

Diese Datei ist aus drei Spalten aufgebaut. In der ersten Spalte befindet sich der Name der Datei. Hierbei kann es sich auch um einen regulären Ausdruck handeln. In der zweiten Spalte wird der Dateityp angegeben:

- `--`: Eine einfache Datei
- `-d`: Ein Verzeichnis
- `-l`: Eine Verknüpfung

Bleibt die zweite Spalte leer, so wird der Dateityp nicht eingeschränkt, sondern die Datei darf einen beliebigen Typ besitzen. In der dritten rechten Spalte wird der Context angegeben, den die Datei aufweisen soll. Auf einem *MLS/MCS*System wird hierzu das Makro `gen_context` verwendet. Unterstützt das SELinux-System kein MLS/MCS, kann der Context auch direkt eingetragen werden.

24.4 Übersetzung

Nun müssen die von uns erzeugten Textdateien in ein binäres SELinux-Policy-Modul übersetzt werden. Hierzu ist in dem Entwicklungspaket ein `Makefile` enthalten, das wir hierzu verwenden.

```
[root@supergrobi selinux-date]# make -f /usr/share/selinux/devel/Make
file
Compiling targeted date module
/usr/bin/checkmodule:  loading policy configuration ⬅
        from tmp/date.tmp
/usr/bin/checkmodule:  policy configuration loaded
/usr/bin/checkmodule:  writing binary representation ⬅
        (version 6) to tmp/date.mod
Creating targeted date.pp policy package
rm tmp/date.mod tmp/date.mod.fc
```

Achtung

Dieses Makefile ist bei der Debian-Distribution nicht ganz vollständig. Daher erzeugen Sie sich in Ihrem aktuellen Verzeichnis selbst einen kleinen Zweizeiler, der dann die Aufgabe übernimmt. Erzeugen Sie die Datei `Makefile` mit folgendem Inhalt:

```
HEADERDIR:=/usr/share/selinux/refpolicy-targeted/include
include $(HEADERDIR)/Makefile$
```

Achten Sie bitte auf die Groß- und Kleinschreibung. Anschließend können Sie dann den Befehl `make` ohne den Verweis mit der Option `-f ...` verwenden.

Achten Sie darauf, dass Sie sich bei dem Aufruf weiterhin in dem Ordner befinden, in dem Sie Ihre Dateien abgespeichert haben. Der Verweis auf das Makefile erfolgt mit der Option `-f /usr/share/selinux/devel/Makefile`. Bei dem Aufruf wird der Befehl `make` entsprechend dem `Makefile` versuchen, sämtliche `*.te`-Dateien in dem aktuellen Verzeichnis zu übersetzen und für jede `*.te`-Datei ein Policy-Package zu bauen. Daher ist es sinnvoll, in der Zukunft für jedes Package ein eigenes Verzeichnis zu verwenden.

Nun haben Sie Ihr erstes *Policy!-Package* erzeugt: `date.pp`. Bei dem Übersetzen erkennt das Kommando `make`, welche Policy Sie geladen haben. In meinem Fall handelt es sich um die *Targeted-Policy*. Falls Sie gerade die *Strict-Policy* verwenden, werden Sie an den entsprechenden Stellen in der Ausgabe des Befehls diese referenziert finden. Bitte wechseln Sie aber für diesen Test in die Targeted-Policy. Hiermit ist die Erstellung zunächst wesentlich einfacher. Wir werden später das Modul auch für die Strict-Policy erzeugen (siehe Kapitel 28).

24.5 Laden der Policy und Labeln der Dateien

Nun müssen wir diese Policy erstmals laden. Hierfür verwenden Sie den Befehl `semodule`. Nach dem Laden können Sie mit dem gleichen Befehl überprüfen, ob das Modul erfolgreich geladen wurde:

```
[root@supergrobi selinux-date]# semodule -i date.pp
[root@supergrobi selinux-date]# semodule -l | grep date
date    1.0.0
```

Nachdem das Modul geladen worden ist, kann mit dem Befehl `restorecon` der Kontext der binären Datei `/bin/date` entsprechend eingestellt werden:

```
[root@supergrobi selinux-date]# ls -Z /bin/date
-rwxr-xr-x  root root system_u:object_r:bin_t          /bin/date
[root@supergrobi selinux-date]# restorecon /bin/date
[root@supergrobi selinux-date]# ls -Z /bin/date
-rwxr-xr-x  root root system_u:object_r:date_exec_t    /bin/date
```

Nun besitzt die Datei den richtigen Kontext. In Kombination mit dem TE-Richtliniengerüst wird beim Aufruf dieses Befehls SELinux den Prozess in der Domäne *date_t* überwachen.

24.6 Test und Analyse der Fehlermeldungen

Nun ist es an der Zeit, unser Policy-Package zu testen. Ob der Befehl `date` weiterhin funktioniert, hängt vom Zustand des SELinux-Systems (Permissive oder Enforcing) ab. Im Enforcing-Modus sollte der Befehl nicht funktionieren, da wesentliche TE-Regeln noch fehlen. Im Permissive-Modus sollte der Befehl funktionieren. Alle noch nicht erlaubten Zugriffe sollten jedoch protokolliert werden.

Am einfachsten erfolgt die Analyse tatsächlich im *Permissive*-Modus. Sie erhalten dann auf einen Schlag sämtliche Fehlermeldungen in den Protokollen für die weitere Analyse. Diesen Weg wollen wir hier auch gehen. Aktivieren Sie daher den Permissive-Modus für diesen Test. Hierfür müssten Sie natürlich beim Einsatz der *Strict-Policy* vorher in die Rolle *sysadm_r* wechseln[2].

```
[root@supergrobi ~]# setenforce 0
[root@supergrobi ~]# getenforce
Permissive
```

Führen Sie nun ein *Reload* der SELinux-Policy durch. Dies erleichtert später die Analyse der Fehlermeldungen, da der Reload protokolliert wird. So müssen wir uns nur auf die Meldungen nach dem letzten Reload konzentrieren.

```
[root@supergrobi ~]# semodule -R
```

Nun rufen wir den Befehl `date` auf. Im Permissive-Modus sollte der Befehl uns auch die aktuelle Uhrzeit und das Datum anzeigen.

```
[root@supergrobi ~]# date
Mo 6. Nov 19:16:26 CET 2006
```

Wenn Sie nun die in der Datei `/var/log/audit/audit.log` protokollierten Fehlermeldungen analysieren, stellen Sie fest, dass der Befehl `date` tatsächlich in der Domäne *date_t* ausgeführt wird. Der Source-Context bei den meisten Fehlermeldungen ist *root*:*system_r*:*date_t*:

```
type=AVC msg=audit(1168283757.301:91): avc:  denied  { read write } ➥
         for  pid=2890 comm="date" name="2" dev=devpts ino=4 ➥
         scontext=root:system_r:date_t:s0-s0:c0.c1023 tcontext= ➥
         root:object_r:devpts_t:s0 tclass=chr_file
```

Eine Vielzahl von Fehlermeldungen wird protokolliert. Auf meinem Fedora Core 6-System handelt es sich um die folgenden Meldungen des *Access-Vector-Cache* (AVC):

[2] `newrole -r sysadm_r`

```
type=AVC msg=audit(1168283757.303:93): avc:  denied  { read } for ➥
         pid=2890 comm="date" name="ld.so.cache" dev=hda2 ➥
         ino=352549 scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:ld_so_cache_t:s0 tclass=file
type=AVC msg=audit(1168283757.303:94): avc:  denied  { getattr } ➥
         for pid=2890 comm="date" name="ld.so.cache" dev=hda2 ➥
         ino=352549 scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:ld_so_cache_t:s0 tclass=file
type=AVC msg=audit(1168283757.303:95): avc:  denied  { search } ➥
         for pid=2890 comm="date" name="lib" dev=hda2 ino=160161 ➥
         scontext=root:system_r:date_t:s0-s0:c0.c1023 tcontext= ➥
         system_u:object_r:lib_t:s0 tclass=dir
type=AVC msg=audit(1168283757.303:95): avc:  denied  { read } for ➥
         pid=2890 comm="date" name="librt.so.1" dev=hda2 ino= ➥
         160214 scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:lib_t:s0 tclass=lnk_file
type=AVC msg=audit(1168283757.303:95): avc:  denied  { read } for ➥
         pid=2890 comm="date" name="librt-2.5.so" dev=hda2 ➥
         ino=162740 scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:lib_t:s0 tclass=file
type=AVC msg=audit(1168283757.304:96): avc:  denied  { getattr } ➥
         for pid=2890 comm="date" name="librt-2.5.so" dev=hda2 ➥
         ino=162740 scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:lib_t:s0 tclass=file
type=AVC msg=audit(1168283757.304:97): avc:  denied  { execute } ➥
         for pid=2890 comm="date" name="librt-2.5.so" dev=hda2 ➥
         ino=162740 scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:lib_t:s0 tclass=file
type=AVC msg=audit(1168283757.304:98): avc:  denied  { read } for ➥
         pid=2890 comm="date" name="ld-2.5.so" dev=hda2 ➥
         ino=162736 scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:ld_so_t:s0 tclass=file
type=AVC msg=audit(1168283757.305:99): avc:  denied  { search } ➥
         for pid=2890 comm="date" name="/" dev=dm-2 ino=2 ➥
         scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:usr_t:s0 tclass=dir
type=AVC msg=audit(1168283757.305:99): avc:  denied  { search } ➥
         for pid=2890 comm="date" name="locale" dev=dm-2 ➥
         ino=259845 scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:locale_t:s0 tclass=dir
type=AVC msg=audit(1168283757.305:99): avc:  denied  { read } for ➥
         pid=2890 comm="date" name="locale-archive" dev=dm-2 ➥
         ino=262709 scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:locale_t:s0 tclass=file
```

```
type=AVC msg=audit(1168283757.305:100): avc:  denied  { getattr } ➥
         for pid=2890 comm="date" name="locale-archive" dev=dm-2 ➥
         ino=262709 scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=system_u:object_r:locale_t:s0 tclass=file
type=AVC msg=audit(1168283757.307:101): avc:  denied  { getattr } ➥
         for pid=2890 comm="date" name="2" dev=devpts ino=4 ➥
         scontext=root:system_r:date_t:s0-s0:c0.c1023 ➥
         tcontext=root:object_r:devpts_t:s0 tclass=chr_file
```

Diese Fehlermeldungen können nun mit dem Befehl `audit2allow` (siehe Abschnitt 18.3) analysiert werden. Rufen Sie hierzu den Befehl wie folgt auf:

```
[root@supergrobi ~]# audit2allow  -l -i /var/log/audit/audit.log
allow date_t devpts_t:chr_file  getattr read write ;
allow date_t etc_t:dir search;
allow date_t ld_so_cache_t:file  getattr read ;
allow date_t ld_so_t:file read;
allow date_t lib_t:dir search;
allow date_t lib_t:file  execute getattr read ;
allow date_t lib_t:lnk_file read;
allow date_t locale_t:dir search;
allow date_t locale_t:file  getattr read ;
allow date_t usr_t:dir search;
```

So zeigt der Befehl alle benötigten *allow*-Regeln für unser Modul an. Leider berücksichtigt der Befehl bei diesem Aufruf nicht die modulare Struktur der Reference-Policy. Würden Sie diese Regeln so ohne weitere Anpassung in Ihre TE-Datei übernehmen und versuchen, das Modul neu zu übersetzen, würden Sie den folgenden Fehler erhalten:

```
[root@supergrobi selinux-date]# make -f /usr/share/selinux/devel/ ➥
         Makefile
Compiling targeted date module
/usr/bin/checkmodule:  loading policy configuration from ➥
         tmp/date.tmp
date.te:9:ERROR 'unknown type devpts_t' at token ';' on line 76741:
allow date_t devpts_t:chr_file  getattr read write ;

/usr/bin/checkmodule:  error(s) encountered while parsing ➥
         configuration
make: *** [tmp/date.mod] Fehler 1
```

Der Typ *devpts_t* wird zwar durch die *Reference-Policy* definiert, ist aber in deren Modulen verborgen. Ein direkter Zugriff auf diesen Typ ist nicht erlaubt.

Es existieren zwei Auswege aus diesem Dilemma:

1. Sie verlangen spezifisch, dass die entsprechenden Typen bei der Übersetzung und beim Laden Ihres Moduls vorhanden sind. Falls sich jedoch später die Policy in einem Update ändert, besteht die Gefahr, dass auch die von Ihnen verwendeten Typen sich ändern.
2. Oder Sie benutzen die von der Policy zur Verfügung gestellten Schnittstellen. Diese sollten sich auch bei einem späteren Upgrade der Policy nicht ändern.

24.7 Policy mit Require-Block

Die erste Variante ist die Quick-and-dirty-Methode. Um diese einzusetzen, genügt es, den Befehl `audit2allow` mit der Option `-r` aufzurufen und das Ergebnis in der TE-Datei zu verwenden:

```
[root@supergrobi selinux-date]# audit2allow -r  -l -i /var/log/ ↵
         audit/audit.log
require
        class chr_file  getattr read write ;
        class dir search;
        class file  execute getattr read ;
        class lnk_file read;
        type date_t;
        type devpts_t;
        type etc_t;
        type ld_so_cache_t;
        type ld_so_t;
        type lib_t;
        type locale_t;
        type usr_t;
        role system_r;
;

allow date_t devpts_t:chr_file  getattr read write ;
allow date_t etc_t:dir search;
allow date_t ld_so_cache_t:file  getattr read ;
allow date_t ld_so_t:file read;
allow date_t lib_t:dir search;
allow date_t lib_t:file  execute getattr read ;
allow date_t lib_t:lnk_file read;
allow date_t locale_t:dir search;
allow date_t locale_t:file  getattr read ;
allow date_t usr_t:dir search;
```

Nun erzeugt der Befehl zusätzlich einen `require`-Block, in dem die Verfügbarkeit der entsprechenden Typen und Klassen erzwungen wird.

Wenn Sie diesen Block in Ihre Datei `date.te` eintragen, hat diese den folgenden Inhalt:

```
policy_module(date,1.0.1)

type date_t;
type date_exec_t;
domain_type(date_t)
domain_entry_file(date_t, date_exec_t)
domain_auto_trans(unconfined_t, date_exec_t, date_t)

require {
        class chr_file { getattr read write };
        class dir search;
        class file { execute getattr read };
        class lnk_file read;
        type date_t;
        type devpts_t;
        type etc_t;
        type ld_so_cache_t;
        type ld_so_t;
        type lib_t;
        type locale_t;
        type usr_t;
        role system_r;
};

allow date_t devpts_t:chr_file { getattr read write };
allow date_t etc_t:dir search;
allow date_t ld_so_cache_t:file { getattr read };
allow date_t ld_so_t:file read;
allow date_t lib_t:dir search;
allow date_t lib_t:file { execute getattr read };
allow date_t lib_t:lnk_file read;
allow date_t locale_t:dir search;
allow date_t locale_t:file { getattr read };
allow date_t usr_t:dir search;
```

Nun gelingt auch die Übersetzung des Policy-Package. Achten Sie darauf, die *Versionsnummer* Ihres Paketes anzupassen. Dies wird leider nicht automatisch vorgenommen.

Nun können Sie Ihre neue Version des Policy-Package `date.pp` laden:

```
[root@supergrobi selinux-date]# semodule -u date.pp
[root@supergrobi selinux-date]# semodule -l | grep date
date    1.0.1
```

Die Kontrolle der Versionsnummer zeigt Ihnen auch, dass der Austausch Ihres Moduls erfolgreich geglückt ist.

Testen Sie nun den Befehl `date` erneut. Finden Sie in den Protokolldateien AVC-Meldungen, die auf weitere Regelverstöße hinweisen? Falls dies der Fall ist, wiederholen Sie bitte die letzten Schritte.

Versetzen Sie SELinux wieder in den *Enforcing*-Modus. Funktioniert der Befehl immer noch? Melden Sie sich zum Test als einfacher Benutzer an. Kann auch ein unprivilegierter Benutzer den Befehl verwenden?

Herzlichen Glückwunsch. Sie haben Ihr erstes Modul erfolgreich geschrieben!

24.8 Policy unter Zugriff auf die Schnittstellen

Wie ich bereits beschrieben habe, handelt es sich bei der letzteren Vorgehensweise häufig um die schnellere, aber nicht so zukunftsicherere Variante. Um von Änderungen der Policy unabhängig zu sein, sollten Sie die von der Policy zur Verfügung gestellten Schnittstellen nutzen. Auch hierbei unterstützt Sie der Befehl `audit2allow`.

Entfernen Sie die in dem letzten Abschnitt hinzugefügten Zeilen wieder aus Ihrer Datei `date.te`, falls Sie die Schritte nachvollzogen haben, und übersetzen Sie erneut das Paket. Denken Sie daran, wieder die Versionsnummer zu erhöhen. Führen Sie wieder ein Upgrade des Moduls durch, und testen Sie den Befehl im *Permissive*-Modus:

```
[root@supergrobi selinux-date]# setenforce 0
[root@supergrobi selinux-date]# semodule -u date.pp
[root@supergrobi selinux-date]# date
Mo 8. Jan 21:36:15 CET 2007
```

Wenn Sie nun den Befehl `audit2allow` mit der Option `-R` aufrufen, wird der Befehl versuchen, die notwendigen Schnittstellen in der *Reference-Policy* selbst zu ermitteln und zu verwenden. Leider gelingt dies nicht immer in allen Fällen. Diese Option setzt natürlich auch die Installation des Pakets `selinux-policy-devel` voraus. In dem folgenden Listing habe ich die überflüssigen kommentierten Zeilen gelöscht. Ab Fedora Core 6 und bei Debian Etch sieht die Ausgabe auch ein wenig anders aus[3]. Wundern Sie sich daher nicht, falls bei Ihnen die Ausgabe auch etwas anders aussehen sollte.

[3] Die `optional_policy`-Container sind dort verschwunden.

Achtung

Falls Sie die Fehlermeldung `could not open interface info [/var/lib/sepolgen/interface_info]` bekommen, müssen Sie vorher noch den Befehl `sepolgen-ifgen` aufrufen. Dieser Befehl erzeugt die benötigte Datei.

```
[root@supergrobi selinux-date]# audit2allow -R -l -i /var/log/ ⬅
         audit/audit.log
allow date_t devpts_t:chr_file { getattr read write };

#allow date_t etc_t:dir search;
optional_policy(‘files’, ‘
        files_search_etc(date_t)
’);
allow date_t ld_so_cache_t:file { getattr read };

#allow date_t ld_so_t:file read;
optional_policy(‘libraries’, ‘
        libs_use_ld_so(date_t)
’);

#allow date_t lib_t:dir search;
optional_policy(‘libraries’, ‘
        libs_search_lib(date_t)
’);
allow date_t lib_t:file { execute getattr read };

#allow date_t lib_t:lnk_file read;
optional_policy(‘libraries’, ‘
        libs_read_lib_files(date_t)
’);

#allow date_t locale_t:dir search;
optional_policy(‘miscfiles’, ‘
        miscfiles_read_localization(date_t)
’);
allow date_t locale_t:file { getattr read };

#allow date_t usr_t:dir search;
optional_policy(‘files’, ‘
        files_search_usr(date_t)
’);
```

Wenn wir nun diese Zeilen an unsere rudimentäre Type-Enforcement-Datei `date.te` anhängen und versuchen, das Paket neu zu übersetzen, erscheinen auf Fedora Core 5 und 6 (ohne Updates) mehrere Fehlermeldungen:

```
[root@supergrobi selinux-date]# make -f /usr/share/selinux/devel/ ↵
         Makefile Compiling targeted date module
date.te:15: Warning: deprecated use of module name (files) as ↵
         first parameter of optional_policy() block.
date.te:21: Warning: deprecated use of module name (libraries) as ↵
         first parameter of optional_policy() block.
date.te:26: Warning: deprecated use of module name (libraries) as ↵
         first parameter of optional_policy() block.
date.te:32: Warning: deprecated use of module name (libraries) as ↵
         first parameter of optional_policy() block.
date.te:37: Warning: deprecated use of module name (miscfiles) as ↵
         first parameter of optional_policy() block.
date.te:43: Warning: deprecated use of module name (files) as ↵
         first parameter of optional_policy() block.
/usr/bin/checkmodule:  loading policy configuration from tmp/date.tmp
date.te:21:ERROR 'syntax error' at token '}' on line 76837:
#line 21
                } else {
/usr/bin/checkmodule:  error(s) encountered while parsing ↵
         configuration
make: *** [tmp/date.mod] Fehler 1
```

Diese Meldungen weisen uns zunächst darauf hin, dass hier Probleme zwischen dem Kommando `audit2allow` und dem Compiler bestehen. Die einfachste Vorgehensweise ist zunächst das Löschen des Modulnamens in jedem `optional_policy`-Konstrukt.

Achtung

Wenn Sie sämtliche Updates unter Fedora Core 6 installieren, taucht diese Fehlermeldung inzwischen nicht mehr auf. Auch Fedora 7 weist diesen Fehler nicht mehr auf.

Hier sehen Sie ein Beispiel, bei dem der Modulname `files` gelöscht wurde. Achten Sie darauf, dass weiterhin die richtigen Anführungszeichen gesetzt sind:

```
#allow date_t etc_t:dir search;
optional_policy(‘
        files_search_etc(date_t)
’);
```

Dennoch treten weiterhin Fehler auf:

```
[root@supergrobi selinux-date]# make -f /usr/share/selinux/devel/ ↵
         Makefile Compiling targeted date module
/usr/bin/checkmodule:  loading policy configuration from tmp/date.tmp
date.te:10:ERROR ’unknown type devpts_t’ at token ’;’ on line 76742:
allow date_t devpts_t:chr_file { getattr read write };

/usr/bin/checkmodule:  error(s) encountered while parsing ↵
         configuration
make: *** [tmp/date.mod] Fehler 1
```

Der Compiler beschwert sich, dass der Typ *devpts_t* unbekannt sei. Dieser Typ steht nicht frei zur Verfügung, und eine Schnittstelle muss genutzt werden. Leider hat der Befehl `audit2allow` nicht selbst diese Schnittstelle gefunden. Um die richtige Schnittstelle zu finden, müssen Sie alle verfügbaren durchsuchen. Hier bietet sich der Befehl `grep` an:

```
# grep -R devpts_t /usr/share/selinux/devel/include/
/usr/share/selinux/devel/include/admin/su.if:   dontaudit $1_su_t ↵
         initrc_devpts_t:chr_file  getattr ioctl ;
/usr/share/selinux/devel/include/admin/portage.if:      allow $1 ↵
         portage_devpts_t:chr_file  rw_file_perms setattr ;
/usr/share/selinux/devel/include/admin/portage.if:      ↵
         term_create_pty($1,portage_devpts_t)
/usr/share/selinux/devel/include/kernel/terminal.if:            ↵
         type devpts_t;
/usr/share/selinux/devel/include/kernel/terminal.if:    allow $1 ↵
         devpts_t:filesystem associate;
...
```

Die Ausgabe habe ich gekürzt, um Platz zu sparen. Ansonsten hätte ich hier mehr als zwei Seiten benötigt. Wir müssen die Suche einschränken, um schneller zum Erfolg zu kommen. Eine genaue Analyse der Fehlermeldung zeigt, dass wir eine Schnittstelle benötigen, die folgende Funktion enthält:

```
allow date_t devpts_t:chr_file
```

Deshalb können wir unser Suchmuster konkretisieren: `allow $1 devpts_t:chr_file`. Falls wir hiermit keinen Erfolg haben, können wir immer noch den einen oder anderen Parameter weglassen. Der Parameter `$1` wird in der Schnittstelle als Platzhalter verwendet. Wenn wir die Schnittstelle benutzen, wird er durch unseren Typ *date_t* ersetzt.

Wenn wir nun suchen, erhalten wir:

```
[root@supergrobi selinux-date]# grep -R 'allow $1 devpts_t: ➥
         chr_file' /usr/share/selinux/devel/include/
/usr/share/selinux/devel/include/kernel/terminal.if:    allow $1 ➥
         devpts_t:chr_file ioctl;
/usr/share/selinux/devel/include/kernel/terminal.if:    allow $1 ➥
         devpts_t:chr_file setattr;
/usr/share/selinux/devel/include/kernel/terminal.if:    allow $1 ➥
         devpts_t:chr_file  rw_term_perms lock append ;
```

Dies sind nur noch drei Fundstellen. Nun müssen wir in der Datei `/usr/share/selinux/devel/include/kernel/terminal.if` die richtige Schnittstelle finden. Da wir Lese- und Schreibberechtigungen benötigen, handelt es sich um die Schnittstelle, in der *rw_term_perms* zugewiesen werden. Dies ist das `term_use_generic_ptys`-Interface:

```
########################################
## <summary>
##      Read and write the generic pty
##      type.  This is generally only used in
##      the targeted policy.
## </summary>
## <param name="domain">
##      <summary>
##      Domain allowed access.
##      </summary>
## </param>
#
interface(`term_use_generic_ptys',`
        gen_require(`
                type devpts_t;
        ')

        dev_list_all_dev_nodes($1)
        allow $1 devpts_t:dir list_dir_perms;
        allow $1 devpts_t:chr_file { rw_term_perms lock append };
')
```

Wir können nun unsere Datei `date.te` editieren und die Zeile

```
allow date_t devpts_t:chr_file { getattr read write };
```

durch die Zeilen

```
#allow date_t devpts_t:chr_file { getattr read write };
optional_policy(‘
        term_use_generic_ptys(date_t)
’);
```

ersetzen. Bei einer erneuten Übersetzung stellen wir fest, dass auch die Zeile

```
allow date_t ld_so_cache_t:file { getattr read };
```

Probleme erzeugt. Auch hier ist die Anwendung einer Schnittstelle erforderlich. Wenn wir genauso wie vorhin suchen, erhalten wir:

```
[root@supergrobi selinux-date]# grep -R ’allow $1 ld_so_cache_t: ➥
         file’ /usr/share/selinux/devel/include/
/usr/share/selinux/devel/include/system/libraries.if:     ➥
         allow $1 ld_so_cache_t:file r_file_perms;
```

Eine Suche nach dem Interface-Namen in der Datei `libraries.if` ergibt: `libs_use_ld_so`. Bei Anwendung dieser Schnittstelle darf der Prozess auf Bibliotheken zugreifen. Eine Analyse der Datei `date.te` zeigt, dass diese Schnittstelle bereits verwendet wird. Es genügt daher, die entsprechende Zeile einfach auszukommentieren:

```
...
#allow date_t ld_so_cache_t:file { getattr read };
#allow date_t ld_so_t:file read;
optional_policy(‘
        libs_use_ld_so(date_t)
’);
...
```

So nähern wir uns iterativ unserer Policy. Ein erneuter Übersetzungsversuch schlägt immer noch fehl. Diesmal stört sich der Compiler an der folgenden Zeile:

```
allow date_t lib_t:file { execute getattr read };
```

Scheinbar versucht der Befehl `date` Bibliotheken zu lesen und auszuführen. Ein Blick in die Schnittstellendatei `libraries.if` zeigt, dass wir noch die Schnittstelle `libs_use_lib_files` benötigen. Also editieren wir die Datei `date.te` wieder entsprechend:

```
#allow date_t lib_t:file { execute getattr read };
optional_policy(‘
        libs_use_lib_files(date_t)
’);
```

Die letzte Zeile, die von dem Compiler noch mit einer Fehlermeldung quittiert wird, ist:

```
allow date_t locale_t:file { getattr read };
```

Eine Analyse der Schnittstelle `miscfiles_read_localization` in der Datei `miscfiles.if` zeigt jedoch, dass diese Zeile auch in der schon geladenen Schnittstelle enthalten ist. Diese Zeile kann daher auskommentiert werden.

```
#allow date_t locale_t:dir search;
#allow date_t locale_t:file { getattr read };
optional_policy(‘
        miscfiles_read_localization(date_t)
’);
```

Wenn wir nun das Policy-Package wieder bauen, gelingt dies auch:

```
[root@supergrobi selinux-date]# make -f /usr/share/selinux/devel/ ↵
         Makefile
Compiling targeted date module
/usr/bin/checkmodule:  loading policy configuration from tmp/date.tmp
/usr/bin/checkmodule:  policy configuration loaded
/usr/bin/checkmodule:  writing binary representation (version 6) ↵
         to tmp/date.mod
Creating targeted date.pp policy package
rm tmp/date.mod tmp/date.mod.fc
```

Nun können Sie das Modul wieder laden.

```
[root@supergrobi selinux-date]# semodule -u date.pp
libsemanage.semanage_direct_upgrade: Previous module date is same ↵
         or newer.
semodule:  Failed on date.pp!
```

Uups. Hier haben wir wohl vergessen, die Versionsnummer im Modul anzupassen. Dies lässt sich aber schnell nachholen. Dann ist das Laden des Moduls erfolgreich.

Wenn Sie nun das SELinux-System wieder in den *Enforcing*-Modus versetzen und den Befehl `date` aufrufen, sollte dieser funktionieren. Im Protokoll dürfen keine Fehlermeldungen auftauchen.

Herzlichen Glückwunsch. Sie haben erneut erfolgreich ein SELinux- Policy-Package gebaut, das lediglich die Schnittstellen der Reference-Policy verwendet.

24.9 Setzen der Uhrzeit erlauben

Bisher erlaubt die Policy lediglich das Auslesen der Uhrzeit. Ein Setzen der Uhrzeit mit dem `date`-Kommando wird nicht erlaubt:

```
[root@supergrobi selinux-date]# date -s 07:00
date: das Datum kann nicht gesetzt werden: Die Operation ist ↵
         nicht erlaubt
Mi 10. Jan 07:00:00 CET 2007
```

Im Audit-Log taucht folgende Meldung auf:

```
type=AVC msg=audit(1168436786.621:432): avc:  denied  { sys_time } ↵
         for  pid=20569 comm="date" capability=25 scontext=root: ↵
         system_r:date_t:s0-s0:c0.c1023 tcontext=root:system_r: ↵
         date_t:s0-s0:c0.c1023 tclass=capability
```

Der Prozess mit dem Source-Context *root:system_r:date_t* versucht, auf die Capability (*tclass=capability*) *CAP_SYS_TIME* zuzugreifen. SELinux verhindert diesen Zugriff bisher.

Um das Setzen der Uhrzeit zu erlauben, müssen wir unsere Policy entsprechend erweitern. Die entsprechende Regel lautet:

```
allow date_t self:capability sys_time;
```

Hierfür benötigen wir keinen Schnittstellenzugriff. Wenn wir die Regel hinzufügen und unser Policy-Package neu übersetzen, müssen wir daran denken, dass wir auch die Versionsnummer inkrementieren. Nach dem Laden des neuen Package ist auch ein Setzen der Zeit möglich.

```
[root@supergrobi selinux-date]# semodule -u date.pp
[root@supergrobi selinux-date]# date -s 07:00
Mi 10. Jan 07:00:00 CET 2007
[root@supergrobi selinux-date]# date
Mi 10. Jan 07:00:02 CET 2007
```

24.10 Ist die Policy nun fertig?

Ist unsere Policy nun fertig? Nun, wir haben die beiden wesentlichen Anwendungen des Befehls `date` getestet. Er funktioniert in dieser Form sowohl für den Systemadministrator als auch den unprivilegierten Benutzer. Dennoch können wir nicht ausschließen, dass auch noch andere Programme auf dem System den Befehl `date` für ihre Funktionen nutzen. Häufig wird dieser Befehl in Scripts verwendet. Um sicherzustellen, dass das System trotz unserer Policy-Erweiterung so funktioniert, wie wir uns das vorstellen, sollten wir das System neu booten. So können wir prüfen, ob während des Bootvorgangs ein Script oder ein Dienst auf diesen Befehl zugreift und dieser Zugriff aktuell von unserer Policy unterbunden wird.

Auf einem Fedora Core 6-System ist das nicht der Fall. Daher scheinen wir tatsächlich mit der Policy fertig zu sein.

Falls wir später aber weitere Policys erzeugen, die genau dieses Problem aufweisen, müssen wir jetzt vorsorgen. Wir müssen eine Schnittstelle schaffen, sodass wir später in einem weiteren Policy-Package diese Schnittstelle zum `date.pp` nutzen können.

24.11 Interface

Jedes vollständige Policy-Package besteht normalerweise aus drei Dateien:

- `date.te`
- `date.fc`
- `date.if`

Während wir die ersten beiden Dateien bereits mit Inhalt gefüllt haben, haben wir bisher die letzte Datei ignoriert. Hierbei handelt es sich um die Interface-Datei. In der *Interface*-Datei werden Zugänge zu den in dem Policy-Package definierten Typen offengelegt. Hierbei handelt es sich meist um Zugänge zu folgenden Typen:

- Die Domäne, in der der Prozess ausgeführt wird. Hierfür wird eine Domänentransition in dem Interface definiert.
- Logdateien. Um die Protokolldateien von Diensten lesen zu dürfen, wird eine Schnittstelle definiert, die dies anderen Domänen erlaubt.
- Lesen und Schreiben von Daten. Wenn ein Dienst Daten erzeugt oder anbietet, dann existieren häufig Schnittstellen, sodass auch andere Programme diese Daten lesen oder verändern dürfen.

Da wir in unserem Fall abgesehen von dem Hilfstyp (*date_exec_t*) nur einen neuen Typ (*date_t*) definiert haben, benötigen wir auch nur eine Schnittstelle.

Die Schnittstelle hat folgendes Aussehen und folgende Syntax:

```
########################################
## <summary>
##      Execute a domain transition to run date.
## </summary>
## <param name="domain">
##      Domain allowed to transition.
## </param>
#
interface(‘date_domtrans’,‘
        gen_require(‘
                type date_t, date_exec_t;
        ’)

        domain_auto_trans($1,date_exec_t,date_t)

        allow $1 date_t:fd use;
        allow date_t $1:fd use;
        allow $1 date_t:fifo_file rw_file_perms;
        allow $1 date_t:process sigchld;
’)
```

Hier wird zunächst die Dokumentation in XML angegeben. Damit der Compiler nicht über die Dokumentation stolpert, wird diese mit Kommentarzeichen versehen. Entsprechende Dokumentationsbrowser können diese jedoch extrahieren und anzeigen. Dann wird die Schnittstelle `date_domtrans` definiert. Diese Schnittstelle erwartet genau einen Parameter. Dieser Parameter ist der Name der Domäne, von der aus der Wechsel in die Domäne *date_t* beim Aufruf eines Binärprogramms vom Typ *date_exec_t* (`/bin/date`) erfolgen soll. Die Variable `$1` in der Schnittstelle wird durch den Parameter ersetzt. In der Schnittstelle wird mit dem `gen_require`-Makro zunächst sichergestellt, dass die Typen existieren. Dann wird die automatische Domänentransition definiert und die Kommunikation zwischen den beiden Domänen über File-Descriptors (fd) und Signale erlaubt.

In unserem Fall ist die Schnittstelle von geringer Bedeutung, da wir festgestellt haben, dass keine weitere Policy diese Schnittstelle benötigt.

24.12 Fazit

Ich hoffe, Sie haben mit diesem Kapitel einen kleinen ersten Einblick in die Erzeugung einer Policy für eine neue Applikation gewonnen. Die Erstellung der Policy ist nicht so einfach und so automatisiert wie bei AppArmor. Dafür bietet SELinux aber auch eine wesentlich weiter gefasste Überwachung und ist auf Fedora-, Red Hat Enterprise Linux- und Debian Etch-Systemen die erste Wahl.

Lassen Sie sich nicht verunsichern, wenn Sie meine Vorgehensweise in diesem Kapitel nicht direkt vollkommen verstanden haben. Wir werden in den folgenden Kapiteln auf einige Punkte genauer eingehen. Bei der täglichen Anwendung werden Ihnen viele Dinge in Fleisch und Blut übergehen, und Sie werden sich über viele jetzt unverständliche Aspekte bald keine Gedanken mehr machen.

Fassen wir noch einmal zusammen. Ein *Policy!-Package* besteht aus drei Dateien: einer Type-Enforcement-Datei mit der Endung `.te`, einer File-Context-Datei mit der Endung `.fc` und einer *Interface*-Datei mit der Endung `.if`. Diese Dateien haben auch immer einen typischen Aufbau. Im Folgenden zeige ich noch einmal den Inhalt der Dateien bei einer Targeted-Policy. Typisch für die Type-Enforcement-Datei ist der folgende Inhalt:

```
policy_module(myapp,1.0.0)

type myapp_t;
type myapp_exec_t;
domain_type(myapp_t)
domain_entry_file(myapp_t, myapp_exec_t)
domain_auto_trans(unconfined_t, myapp_exec_t, myapp_t)

term_use_generic_ptys(myapp_t)
files_search_etc(myapp_t)
libs_use_ld_so(myapp_t)
```

```
libs_search_lib(myapp_t)
libs_use_lib_files(myapp_t)
miscfiles_read_localization(myapp_t)
```

Die FC-Datei hat meist folgenden Inhalt:

```
# myapp executable will have:
# label: system_u:object_r:myapp_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/bin/myapp               --      gen_context(system_u:object_r: ➥
         myapp_exec_t,s0)
```

Hier können aber auch weitere Typen definiert werden, wenn die Applikation auch temporäre Dateien oder Protokolle benötigt.

Schließlich gibt es noch die Interface-Datei. Hier werden die Schnittstellen definiert. In vielen Fällen existiert hier nur eine Schnittstelle:

```
########################################
## <summary>
##      Execute a domain transition to run myapp.
## </summary>
## <param name="domain">
##      Domain allowed to transition.
## </param>
#
interface(‘myapp_domtrans’,‘
        gen_require(‘
                type myapp_t, myapp_exec_t;
        ’)

        domain_auto_trans($1,myapp_exec_t,myapp_t)

        allow $1 myapp_t:fd use;
        allow myapp_t $1:fd use;
        allow $1 myapp_t:fifo_file rw_file_perms;
        allow $1 myapp_t:process sigchld;
’)
```

Dies sind die Bausteine, aus denen alle Policy-Packages gebaut werden. Im nächsten Kapitel werden wir sehen, wie boolesche Variablen in der Policy definiert werden. Dann werden wir kompliziertere Packages für Netzwerkdienste bauen.

25 Boolesche Variablen

In diesem Kapitel werden Sie den Einsatz von booleschen Variablen kennenlernen. Eine boolesche Variable erlaubt uns die Definition von Bedingungen in der Policy. Wir können in Abhängigkeit von der booleschen Variablen SELinux-Regeln aktivieren oder deaktivieren. Dies ermöglicht die Definition sehr mächtiger Regelwerke, die von dem Benutzer leicht online angepasst werden können. Jede Änderung der booleschen Variable wird sofort von der Policy durch veränderte Regeln honoriert.

25.1 Überblick

Die Struktur der bedingten Regeln ist eigentlich recht einfach. Zunächst benötigen wir eine boolesche Variable. Die wird mit dem Schlüsselwort `bool` definiert. Etwa so:

```
bool date_adjust_time false;
```

Anschließend können Sie dann in dem Regelwerk `if ... then ... else` Konstrukte aufbauen. Dies sieht dann so aus:

```
if (date_adjust_time) {
   # Diese Regeln werden ausgefuehrt, wenn die Variable wahr ist
} else {
   # Diese Regeln werden ausgefuehrt, wenn die Variable falsch ist
}
```

Die boolesche Variable wird hier definiert, und ihr wird der Default-Wert *false* zugewiesen. Bei der Übersetzung der Policy werden alle Regeln übersetzt. Später kann einfach durch das entsprechende Setzen der Variablen ausgewählt werden, ob der erste Regelblock (*true*) oder der zweite Regelblock (*false*) verwendet werden soll. Dies erfolgt am einfachsten mit dem Befehl `setsebool` (siehe auch Abschnitt 18.34).

25.2 Operatoren

Maximal zwei boolesche Variablen können mit Operatoren verknüpft werden. Hierzu stehen die folgenden Operatoren zur Verfügung:

`&&`	Logisches UND
`||`	Logisches ODER
`^`	Logisches EXLUSIV-ODER
`!`	Negation
`==`	Gleichheit
`!=`	Ungleichheit

So können Sie auch in Abhängigkeit von zwei Variablen bestimmte Regeln aktivieren oder deaktivieren. Mehr als zwei Variablen können nicht miteinander kombiniert werden. Leider können die bedingten Abfragen auch noch nicht geschachtelt werden, um diese Einschränkung zu umgehen. Schachtelungen sind nicht erlaubt.

25.3 Booleans und Interfaces

Beim Einsatz der Reference-Policy kann die boolesche Variable so definiert werden, wie oben beschrieben wurde. Allerdings gibt es auch noch ein Makro (`gen_tunable`) für die Definition der booleschen Variablen:

```
gen_tunable(date_adjust_time,false)
```

Hiermit wird der Variablen gleichzeitig ihr Default-Wert *false* zugewiesen. Auch für die `if`-Konstrukte existiert ein Makro (`tunable_policy`), das die Entwicklung vereinfacht:

```
tunable_policy('date_adjust_time','falls_wahr','falls_falsch');
```

Hier wird aber noch nicht die Verwendung der Makros erzwungen. Dennoch verwendet die Reference-Policy durchgehend bereits diese Makros anstatt direkt die booleschen Variablen zu definieren oder direkt die `if`-Konstrukte zu nutzen. Dies wollen wir daher auch tun.

25.4 Praktische Anwendung bei unserer date-Policy

Um einen praktischen Einsatz zu testen, wollen wir unsere `date`-Policy-Package um eine boolesche Variable erweitern. Die boolesche Variable soll definieren, ob das Setzen der Zeit mit dem Befehl `date` erlaubt ist oder ob dies verboten ist. Für das Setzen der Zeit muss der Befehl die Capability *CAP_SYS_TIME* nutzen. Erlauben wir in der Policy die Nutzung dieser Capability, darf der Befehl die Zeit setzen. Verbieten wir die Nutzung, so schlägt das Setzen der Zeit fehl.

Wir benötigen also zunächst eine boolesche Variable, die wir `date_adjust_time` nennen. In Abhängigkeit von dieser Variablen müssen wir nun die *Type-Enforcement*-Regel, die die Nutzung der Capability erlaubt, aktivieren und deaktivieren. Die folgenden Zeilen erreichen dieses Ziel:

```
gen_tunable(date_adjust_time,false)

tunable_policy(‘date_adjust_time’,‘
        allow date_t self:capability sys_time;
’);
```

Natürlich darf in dem Policy-Package keine weitere Zeile enthalten sein, die bedingungslos den Zugriff auf die Capability *CAP_SYS_TIME* erlaubt. Wenn Sie nun Ihr Policy-Package neu übersetzen und laden[1], können Sie über die Variable das Verhalten der Policy beeinflussen.

Prüfen Sie zunächst, ob die Policy tatsächlich unsere boolesche Variable kennt:

```
[root@supergrobi selinux-date]# semodule -u date.pp
[root@supergrobi selinux-date]# getenforce
Enforcing
[root@supergrobi selinux-date]# getsebool date_adjust_time
date_adjust_time --> off
[root@supergrobi selinux-date]# date -s 7:00
date: das Datum kann nicht gesetzt werden: Die Operation ist nicht ↵
         erlaubt
Mo 15. Jan 07:00:00 CET 2007
[root@supergrobi selinux-date]# setsebool date_adjust_time=1
[root@supergrobi selinux-date]# date -s 7:00
Mo 15. Jan 07:00:00 CET 2007
```

Wie Sie sehen und selbst nachvollziehen können, genügt lediglich das Setzen der Variablen, um ein komplett neues Verhalten der Policy zu erreichen.

[1] Denken Sie daran, die Versionsnummer zu inkrementieren!

26 Policy für einen Netzwerkdienst

In diesem Kapitel wollen wir ein *Policy!-Package* für den *HAVP* erzeugen. Der HTTP-AntiVirus-Proxy (HAVP, *http://www.server-side.de*) ist ein Proxy für Linux-Installationen. Dabei analysiert er die Downloads mit einem Virenscanner. Für dieses Programm existiert weder ein RPM-Paket innerhalb der Fedora-Distribution noch eine Richtlinie für SELinux. Debian-Anwender können den HAVP als Paket installieren. Hier werden wir eine Richtlinie für den HAVP unter Fedora erzeugen.

26.1 Installation von HAVP

Hierzu ist zunächst die Installation erforderlich. In diesem Kapitel verwende ich die Version 0.86 des HAVP. Nach dem Download und dem Auspacken des Quelltextes wird der HAVP hierzu mit folgenden Befehlen installiert:

```
[root@supergrobi havp-0.86]# ./configure
[root@supergrobi havp-0.86]# make
[root@supergrobi havp-0.86]# make install
```

Das Startscript in dem Verzeichnis `etc/init.d/` müssen Sie in dieser Version noch manuell installieren:

```
[root@supergrobi havp-0.86]# cp etc/init.d/havp /etc/init.d
```

Nun müssen Sie noch einen Benutzer und eine Gruppe *havp/havp* anlegen und die Konfigurationsdatei `/usr/local/etc/havp/havp.config` editieren. Anschließend müssen Sie dafür sorgen, dass der Benutzer *havp* die folgenden Verzeichnisse schreiben darf: `/var/log/havp`, `/var/tmp/havp` und `/var/run/havp`.

Für das Verzeichnis `/var/tmp/havp` verlangt der HAVP den Einsatz eines Dateisystems mit mandatory Locks. Dies lässt sich leicht unter Einsatz des `tmpfs`-Filesystems erreichen. Tragen Sie dazu die folgende Zeile in der Datei `/etc/fstab` ein:

```
tmpfs   /var/tmp/havp  tmpfs  defaults,mand 0 0
```

Dieses Dateisystem ähnelt einer *Ramdisk*. Jedoch bezieht es seinen Speicherplatz aus dem virtuellen Speicher und belegt ihn auch nur bei tatsächlicher Verwendung. Steht nicht genug RAM zur Verfügung, weicht dieses Dateisystem auch auf den Swap-Speicher aus.

Achtung

Leider gibt es mit dem `tmpfs` in Abhängigkeit von der Linux-Kernel-Version Probleme. Bitte testen Sie, ob es bei Ihnen funktioniert. Ansonsten müssen Sie hier eine normale Ramdisk oder eine normale Partition verwenden.

26.2 Erzeugung der Policy

Um nun ein Policy-Modul für den HAVP-Proxy zu erzeugen, benötigen wir wieder drei Dateien:

- `havp.te`
- `havp.fc`
- `havp.if`

Anstatt dieselbe Methode anzuwenden wie in Kapitel 24, nutzen wir hier ein Script, das von den SELinux-Entwicklern speziell für diesen Zweck erzeugt wurde: `policygentool`. Dieses Script befindet sich auf einem Fedora-System nach Installation des `selinux-policy-devel`-Paketes in dem Verzeichnis `/usr/share/selinux/devel`. Auf Debian-Systemen befindet sich das Werkzeug nach Installation des entsprechenden Paketes im Pfad. Bei dem Aufruf müssen Sie angeben, wie das zu erzeugende *Modul* heißen soll und welches Binärprogramm Sie überwachen möchten:

```
[root@supergrobi ~]# /usr/share/selinux/devel/policygentool havp ↵
         /usr/local/sbin/havp
This tool generate three files for policy development, A Type ↵
         Enforcement (te) file, a File Context (fc), and ↵
         a Interface File(if). Most of the policy rules will ↵
         be written in the te file. Use the File Context file ↵
         to associate file paths with security context. ↵
         Use the interface rules to allow other protected ↵
         domains to interact with the newly defined domains.

After generating these files use the /usr/share/selinux/devel/ ↵
         Makefile to compile your policy package. Then use the ↵
         semodule tool to load it.

# /usr/share/selinux/devel/policygentool myapp /usr/bin/myapp
# make -f /usr/share/selinux/devel/Makefile
# semodule -l myapp.pp
# restorecon -R -v /usr/bin/myapp "all files defined in myapp.fc"
```

```
Now you can turn on permissive mode, start your application and ➥
         avc messages will be generated. You can use audit2allow ➥
         to help translate the avc messages into policy.

# setenforce 0
# service myapp start
# audit2allow -R -i /var/log/audit/audit.log
Return to continue:

If the module uses pidfiles, what is the pidfile called?
/var/run/havp/havp.pid
If the module uses logfiles, where are they stored?
/var/log/havp
If the module has var/lib files, where are they stored?

Does the module have a init script? [yN]
y
Does the module use the network? [yN]
y
[root@supergrobi ~]# ls
havp.te havp.if havp.fc
[root@supergrobi ~]# make -f /usr/share/selinux/devel/Makefile
Compiling targeted havp module
/usr/bin/checkmodule:  loading policy configuration from tmp/havp.tmp
/usr/bin/checkmodule:  policy configuration loaded
/usr/bin/checkmodule:  writing binary representation (version 6) ➥
         to tmp/havp.mod
Creating targeted havp.pp policy package
rm tmp/havp.mod tmp/havp.mod.fc
[root@supergrobi ~]# semodule -i havp.pp
```

Nun müssen Sie die in der FC-Datei definierten Dateien mit ihrem neuen Security-Context versehen und den Dienst starten.

```
[root@supergrobi ~]# restorecon -v -R /usr/local/sbin/havp
/var/run/havp /var/log/havp/
[root@supergrobi ~]# run_init /etc/init.d/havp start
```

Mit dem Befehl `run_init` wird der Dienst in dem richtigen Kontext gestartet. Es handelt sich hierbei um denselben Kontext, in dem auch der Befehl `init` diese Dienste startet.

Vorher sollten Sie SELinux in den Permissive-Modus versetzt haben. Dann können Sie nun mit `audit2allow -R` die protokollierten Verletzungen der rudimentären HAVP-Policy analysieren:

```
allow havp_t clamd_var_lib_t:dir  getattr read search ;
allow havp_t clamd_var_lib_t:file  getattr read ;
allow havp_t self:capability  setgid setuid ;

#allow havp_t http_cache_port_t:tcp_socket name_bind;
optional_policy(‘
        corenet_tcp_bind_http_cache_port(havp_t)
’);
#allow havp_t inaddr_any_node_t:tcp_socket node_bind;
optional_policy(‘
        corenet_tcp_bind_inaddr_any_node(havp_t)
’);
allow havp_t tmp_t:file  create getattr read unlink write ;
allow havp_t tmpfs_t:dir  add_name remove_name search write ;
allow havp_t tmpfs_t:file  create getattr lock read setattr ➥
         unlink write ;

#allow havp_t var_lib_t:dir search;
optional_policy(‘
        files_search_var_lib(havp_t)
#       files_list_var_lib(havp_t)
#       files_var_lib_filetrans(havp_t)
#       files_read_var_lib_files(havp_t)
#       files_read_var_lib_symlinks(havp_t)
#       files_manage_urandom_seed(havp_t)
#       files_manage_mounttab(havp_t)
’);

#allow havp_t var_log_t:file append;
optional_policy(‘
        logging_rw_generic_logs(havp_t)
#       logging_manage_generic_logs(havp_t)
’);
```

Diese von `audit2allow` generierten Regeln sind noch nicht optimal. Einige Aspekte erlauben eine Optimierung:

- Der Zugriff auf den Typ *clamd_var_lib_t* muss über ein Interface der Clamav-Richtlinie erfolgen.
- Scheinbar werden noch Regeln benötigt, um auf allgemeine Protokolldateien zuzugreifen. Dies lässt den Schluss zu, dass die erzeugten Protokolldateien den falschen Typ aufweisen.

Um den Punkt Eins zu lösen, müssen wir zunächst herausfinden, wie die Schnittstelle der Clamav-Richtlinie heißt. Hierzu lesen wir die Datei `clamav.if`, die sich auf einem Fedora-System in `/usr/share/selinux/devel/include/services/clamav.if` befindet. Hierin finden wir die Schnittstelle `clamav_search_lib`, die die von uns gewünschten Regeln definiert:

```
########################################
## <summary>
##      Search clamav libraries directories.
## </summary>
## <param name="domain">
##      <summary>
##      Domain allowed access.
##      </summary>
## </param>
#
interface(‘clamav_search_lib’,‘
        gen_require(‘
                type clamd_var_lib_t;
        ’)

        files_search_var_lib($1)
        allow $1 clamd_var_lib_t:dir search_dir_perms;
’)
```

So können die ersten beiden Regeln, die von `audit2allow` erzeugt wurden, durch die folgende Zeile ersetzt werden:

```
# allow havp_t clamd_var_lib_t:dir { getattr read search };
# allow havp_t clamd_var_lib_t:file { getattr read };
clamav_search_lib(havp)
```

Um den Punkt Zwei zu lösen, ist es zunächst erforderlich, die bereits definierten File-Contexts zu analysieren. Aktuell enthält die Datei `havp.fc` die folgenden Einträge:

```
# havp executable will have:
# label: system_u:object_r:havp_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/usr/local/sbin/havp -- gen_context(system_u: object_r:havp_exec_t,s0)
/var/run/havp/havp.pid gen_context(system_u: ➥
         object_r:havp_var_run_t,s0)
/var/log/havp gen_context(system_u:object_r: havp_var_log_t,s0)
```

Leider werden die Dateien in dem Verzeichnis `/var/log/havp` noch nicht mit dem richtigen Typ versehen. In dieser Datei ist lediglich das Verzeichnis selbst aufgeführt.

Am einfachsten geben Sie diese Dateien zusätzlich an:

```
/var/log/havp/access.log gen_context(system_u:object_r: ➥
        havp_var_log_t,s0)
/var/log/havp/error.log gen_context(system_u:object_r: ➥
        havp_var_log_t,s0)
```

Die zweite Methode ist eine Modifikation der Verzeichnisangabe:

```
/var/log/havp(/.*)? gen_context(system_u:object_r: havp_var_log_t,s0)
```

Nun können Sie die Zeilen entfernen, die der Domäne *havp_t* den Zugriff auf den Typ *var_log_t* erlauben. Der Zugriff auf die Protokolldateien sollte weiterhin erlaubt sein. Aktualisieren Sie die Versionsnummer in der Datei `havp.te`, editieren Sie die Datei `havp.fc` wie besprochen, und übersetzen Sie das SELinux-Modul neu:

```
[root@supergrobi ~]# make -f /usr/share/selinux/devel/Makefile
```

Nun können Sie das Modul mit `semodule -u havp.pp` upgraden und mit `restorecon` die Typen der Protokolldateien entsprechendrestorecon einstellen. Anschließend sollte der HAVP-Proxy ohne Fehlermeldungen starten und funktionieren.

Zum Abschluss stelle ich noch einmal die fertige Richtlinie vor. Ich habe, um die Lesbarkeit zu erhöhen, einige Kommentare entfernt und einige neu hinzugefügt. Zunächst die Datei `havp.te`:

```
policy_module(havp,1.0.6)

########################################
#
# Declarations
#

type havp_t;
type havp_exec_t;
domain_type(havp_t)
init_daemon_domain(havp_t, havp_exec_t)

# log files
type havp_var_log_t;
logging_log_file(havp_var_log_t)

# pid files
type havp_var_run_t;
files_pid_file(havp_var_run_t)
```

```
##############################################
#
# havp local policy
#
# Check in /etc/selinux/refpolicy/include for macros to use ➥
          instead of allow rules.

# Some common macros (you might be able to remove some)
files_read_etc_files(havp_t)
libs_use_ld_so(havp_t)
libs_use_shared_libs(havp_t)
miscfiles_read_localization(havp_t)
## internal communication is often done using fifo and unix sockets.
allow havp_t self:fifo_file { read write };
allow havp_t self:unix_stream_socket create_stream_socket_perms;

# log files
allow havp_t havp_var_log_t:file create_file_perms;
allow havp_t havp_var_log_t:sock_file create_file_perms;
allow havp_t havp_var_log_t:dir { rw_dir_perms setattr };
logging_log_filetrans(havp_t,havp_var_log_t,{ sock_file file dir })

# pid file
allow havp_t havp_var_run_t:file manage_file_perms;
allow havp_t havp_var_run_t:sock_file manage_file_perms;
allow havp_t havp_var_run_t:dir rw_dir_perms;
files_pid_filetrans(havp_t,havp_var_run_t, { file sock_file })

## Networking basics (adjust to your needs!)
sysnet_dns_name_resolve(havp_t)
corenet_tcp_sendrecv_all_if(havp_t)
corenet_tcp_sendrecv_all_nodes(havp_t)
corenet_tcp_sendrecv_all_ports(havp_t)
corenet_non_ipsec_sendrecv(havp_t)
corenet_tcp_connect_http_port(havp_t)
allow havp_t self:tcp_socket { listen accept };

# Init script handling
init_use_fds(havp_t)
init_use_script_ptys(havp_t)
domain_use_interactive_fds(havp_t)


clamav_search_lib(havp_t)
```

```
allow havp_t self:capability { setgid setuid };

#allow havp_t http_cache_port_t:tcp_socket name_bind;
optional_policy(‘
        corenet_tcp_bind_http_cache_port(havp_t)
’);

#allow havp_t inaddr_any_node_t:tcp_socket node_bind;
optional_policy(‘
        corenet_tcp_bind_inaddr_any_node(havp_t)
’);
allow havp_t tmpfs_t:dir { add_name remove_name search write };
allow havp_t tmpfs_t:file { create getattr lock read setattr ➥
         unlink write };

#allow havp_t var_lib_t:dir search;
optional_policy(‘
        files_search_var_lib(havp_t)
’);
```

Die Datei `havp.fc` hat den folgenden Inhalt:

```
# havp executable will have:
# label: system_u:object_r:havp_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/usr/local/sbin/havp--gen_context(system_u: ➥
         object_r:havp_exec_t,s0)
/var/run/havp/havp.pid gen_context(system_u: ➥
         object_r:havp_var_run_t,s0)
/var/log/havp gen_context(system_u: object_r:havp_var_log_t,s0)
/var/log/havp/access.log gen_context(system_u: ➥
         object_r:havp_var_log_t,s0)
/var/log/havp/error.log gen_context(system_u: ➥
         object_r:havp_var_log_t,s0)
```

Durch Experimente mit den Richtlinien können Sie versuchen herauszufinden, ob der HAVP einige der von dem Werkzeug `policygentool` automatisch hinzugefügten Regeln nicht benötigt. Hierzu löschen Sie diese Regeln oder versehen diese mit einem Kommentarzeichen, aktualisieren die Version der Richtlinie, übersetzen sie neu und testen diese. Funktioniert der Proxy weiterhin und erkennen Sie keine Meldungen in den Protokollen, so wurden die Regeln nicht benötigt.

Im letztem Test aktivieren Sie SELinux wieder im Enforcing-Modus und testen den HAVP erneut. Auch sollten Sie die korrekte Funktion bei einem Reboot und einem automatischen Start prüfen.

26.3 Verzicht auf run_init

Warum müssen wir den Dienst mit `run_init` starten? Es wäre doch schön, wenn dieser Dienst genauso wie auch die anderen Dienste mit `/etc/init.d/havp` direkt gestartet werden könnte. Das geht auch, jedoch benötigen wir dazu noch einige zusätzliche Regeln in unserer Richtlinie.

Bisher wird in der Richtlinie nur eine einzige Domänentransition zugelassen:

```
init_daemon_domain(havp_t, havp_exec_t)
```

Diese Zeile verlangt und erlaubt die Domänentransition aus der Domäne *initrc_t* in die Domäne *havp_t* beim Aufruf einer Datei mit dem Typ *havp_exec_t*. Wenn wir aber als *root* das Startscript aufrufen, arbeiten wir nicht in der Domäne *initrc_t*, sondern in der Domäne *unconfined_t*. Daher müssen wir auch noch die Transition aus dieser Domäne verlangen und erlauben. Die erforderlichen Regeln haben wir bereits bei dem Beispiel aus Abschnitt 24.2 besprochen:

```
domain_auto_trans(unconfined_t, havp_exec_t, havp_t)
domain_entry_file(havp_t, havp_exec_t)
```

Werden diese Regeln hinzugefügt, ist ein Start mit `run_init` nicht mehr erforderlich. Der Dienst kann direkt über das Startscript gestartet werden.

27 Labeln von Objekten

Damit SELinux funktioniert, muss jedes Objekt gelabelt werden. Bisher sind wir hierauf weniger eingegangen, da die Dateien in den meisten Fällen bereits gelabelt sind. Dieses Kapitel soll Ihnen zeigen, wie Sie die verschiedenen Objekte labeln können und wie Sie diese Labels auch anschließend wieder verändern.

Jedes Objekt auf einem SELinux-System besitzt einen Security-Context. Dieser *Security-Context* besteht aus dem SELinux-Benutzer, der Rolle, dem Typ und – bei Einsatz der entsprechenden Policy – aus der MLS-Sensitivität und Kategorie:

```
[spenneb@supergrobi ~]$ ls -lZ /bin/ls
-rwxr-xr-x  root root system_u:object_r:ls_exec_t      /bin/ls
```

Das Zuweisen des Security-Contexts bezeichnet man als *Labeling* (oder *Labeln*). Diese Zuweisung erfolgt meist automatisch: entweder bei der Installation von SELinux und dem ersten Labeling des Systems oder durch Funktionen wie die `type_transition`, die nicht nur Domänentransitionen von Prozessen, sondern auch Typentransitionen von Dateien auslösen kann.

Wenn ein neues Objekt erzeugt wird, muss es einen Label (Security Context) erhalten. Welcher Label dies ist, hängt von den SELinux-Regeln und dem *Object-Manager* ab. Existieren keine SELinux-Regeln bezüglich des Labels, so werden die hartkodierten Default-Einstellungen des Object-Managers genutzt. In den meisten Fällen bedeutet das, dass der Object-Manager den Security-Context des erzeugenden Prozesses oder des Objekts, in dem sich das neue Objekt befindet, vererbt. Letzteres trifft zum Beispiel auf Dateien zu: Erzeugen Sie eine neue Datei in einem Verzeichnis, so erbt diese Datei den Typ des übergeordneten Verzeichnisses, eine hartkodierte Rolle *object_r* und den SELinux-Benutzer des erzeugenden Prozesses.

27.1 Labeling von Dateien

In den meisten Fällen müssen auf einem SELinux-System Dateien gelabelt werden, die in dem Verzeichnisbaum referenziert werden. Auch wenn Sie denken, dass dies doch kein Problem darstellen sollte, werde ich mich kurz damit beschäftigen. Diese Dateien können auf einer ganzen Reihe von Dateisystemen gespeichert werden: auf `ext3`, `xfs`, `iso9660`, `vfat` und `ntfs`, auf Netzwerkdateisystemen wie `nfs` und `smbfs` oder auf Pseudo-Dateisystemen wie `proc` und `sysfs`.

Dateien, die in einem ext3-Dateisystem gespeichert werden, können auch den Security-Context in den erweiterten Attributen des Dateisystems dauerhaft speichern. Dieser steht dann auch nach einem Reboot wieder zur Verfügung. Netzwerkdateisysteme bieten bisher diese Funktion nicht[1]. Dateien in einem proc-Dateisystem existieren nur zur Laufzeit und überleben einen Reboot nicht. Die Label müssen daher jedes Mal neu erzeugt werden.

Wie bestimmt und speichert der Kernel den Security-Context einer Datei? Das Verhalten ist in der Type-Enforcement-Richtlinie policy/modules/kernel/filesystem.te festgelegt. Hier wird bestimmt, dass Dateisysteme, die den Security-Context persistent speichern können, die erweiterten Attribute benutzen. Weitere Varianten der Funktion sind task-orientiert oder transitionsorientiert. Einige Dateisysteme versehen auch alle Dateien mit einem einheitlichen (generellen) Security-Context. Fedora Core 6 verwaltet die Dateisysteme wie folgt:

Mechanismus	Dateisysteme
Erweiterte Attribute[2] fs_use_xattr	encfs, ext2, ext3, gfs2, gfs, jffs2, jfs, xfs
Einheitlicher Security-Context genfscon	bdev, binfmt_misc, capifs, configfs, futexfs, ibmasmfs, inotifyfs, nfsd, oprofilesystemfs, ramfs, romfs, rpc_pipefs, autofs, automount, cifs, smbfs, fat, msdos, ntfs-3g, ntfs, vfat, iso9660, udf, nfs, nfs4, afs, hfs, hfsplus, reiserfs
Task-orientiert fs_use_task	eventpollfs, pipefs, sockfs
Transitionsorientiert fs_use_trans	mqueue, shm, tmpfs

Auf Dateisystemen, die keine Speicherung des Security-Contexts unterstützen, und auf Pseudo-Dateisystemen, die keine Unterscheidung benötigen, erhalten das Dateisystem und alle gespeicherten Dateien denselben Security-Context. Bei der task-orientierten Ermittlung des Security-Contexts ist lediglich der Security-Context des Prozesses ausschlaggebend, der das Objekt erzeugt. Das Dateisystem erhält einen eigenen Security-Context. Dies ist für Pipes und Sockets sinnvoll.

Wenn der Security-Context transitionsbasiert ermittelt wird, bedeutet das, dass die Dateien in dem Dateisystem abhängig von dem Label des erzeugenden Prozesses und dem Label des Dateisystems berechnet werden. Dies wird für Dateisysteme wie tmpfs oder shm genutzt.

Für jedes Dateisystem kann es in der Policy nur eine Einstellung geben. Das bedeutet, dass zum Beispiel alle ext3-Dateisysteme auf einem SELinux-System gleich behandelt werden und den Security-Context der gespeicherten Dateien in den erweiterten Attributen abspeichern. Wenn Sie dies verhindern möchten, können Sie einzelne

[1] Aktuell arbeitet man an der Unterstützung von SELinux durch nfs.

[2] Dies setzt voraus, dass die Dateisysteme auch tatsächlich über erweiterte Attribute verfügen.

Dateisysteme anders mounten. Hierzu unterstützt der Befehl `mount` auf einem SELinux-System die folgenden Optionen:

- `context=`
  Diese Option wird normalerweise nur bei Dateisystemen eingesetzt, die keine erweiterten Attribute unterstützen. Sie können diese Option aber auch zum Beispiel bei `xfs` einsetzen. Die Option ist auch sinnvoll, wenn Sie auf einem SELinux-System das Dateisystem eines Linux-Systems ohne SELinux mounten. Mit der Option geben Sie den Security-Context für alle Dateien auf dem Dateisystem an. Dieser wird nicht tatsächlich dort gespeichert, sondern nur für die Dauer des Mounts auf alle existierenden und neu erzeugten Dateien in diesem Dateisystem angewendet. Eine typische Anwendung ist: `context=system_u:object_r:removable_t`.
- `fscontext=`
  Anstelle der Option `context=` können Sie auch `fscontext=` und die folgende Option verwenden. Mit dieser Option setzen Sie lediglich den Security-Context des Dateisystems. Die in diesem Dateisystem gespeicherten Dateien behalten ihren eigenen Security-Context, der durch erweiterte Attribute definiert wird. Diese Funktion kann bei dem Mountvorgang andere Funktionen und Überprüfungen auslösen.
- `defcontext=`
  Mit dieser Option setzen Sie auf einem Dateisystem, das erweiterte Attribute unterstützt, den Default- Security-Context für Dateien, die keinen Security-Context besitzen. Üblicherweise erhalten diese einen Typ wie *unlabeled_t*.

27.1.1 Labeling mit xattrs

In diesem Abschnitt wollen wir genauer untersuchen, wie eine Datei auf einem Dateisystem, das erweiterte Attribute unterstützt, ihren Security-Context erhält. Definiert wird dieses Verhalten in der SELinux-Policy durch folgende Zeile:

```
fs_use_xattr ext3 gen_context(system_u:object_r:fs_t,s0);
```

Wie schon weiter oben erwähnt wurde, erbt die Datei zunächst den Typ des Verzeichnisses, in dem sie erzeugt wird, sowie den Benutzer des aufrufenden Prozesses und erhält die Rolle *object_r*.

Dies kann jedoch durch eine `type_transition`Regel modifiziert werden. Dies wird am einfachsten an einem Beispiel deutlich:

```
type_transition smbd_t samba_etc_t:file samba_secrets_t;
```

Erzeugt ein Prozess in der Domäne *smbd_t* eine Datei (*file*) in einem Verzeichnis mit dem Typ *samba_etc_t*, so erhält die erzeugte Datei den Typ *samba_secrets_t*.

Diese Funktion wird allerdings nur selten gebraucht. Die gesamte *Targeted-Policy* enthält gerade 30 Typen-Transitionen für Dateien in den TE-Dateien. In den Schnittstellen werden weitere 51 definiert. Dies wird sinnvoll eingesetzt, wenn Sie bestimmte

Dateien, die von bestimmten Prozessen erzeugt werden, mit besonderem Schutz versehen möchten.

Außerdem kann auch eine Applikation bewusst ein *Relabeling* einer Datei durchführen. Hierzu muss diese Applikation aber spezifisch für SELinux entwickelt worden sein und die SELinux-Bibliotheken benutzen. Bei allen normalen Applikationen werden die oben beschriebenen Verfahren genutzt. Wenn Sie eine SELinux-fähige Applikation entwickeln möchten, die diese Funktionen nutzt, sollten Sie sich die folgenden Funktionsaufrufe ansehen:

- `setfilecon(3)`[2]
- `lsetfilecon(3)`
- `fsetfilecon(3)`

Dies muss natürlich durch die entsprechenden Privilegien auch der Applikation erlaubt werden. Hier sind die Privilegien `relabelfrom` und `relabelto` erforderlich.

Der initiale Security-Context wird üblicherweise bei der Aktivierung von SELinux gesetzt (siehe auch Kapitel 22). Diesen Vorgang nennt man *Labeling* (*Labeln*) oder auch *Relabeling* (*Relabeln*) des Systems[3]. Dieses Labeling wird von File-Context-Definitionen gesteuert. Diese werden in den FC-Dateien der SELinux-Module angegeben. Das initiale Labeling ist erforderlich, da ansonsten SELinux jeden Zugriff verweigern würde. SELinux benötigt für den Betrieb diese Security-Contexts. Das Labeling versetzt das System in einen wohldefinierten sicheren Zustand.

Die File-Context-Definitionen weisen den Dateien, basierend auf ihrem Pfadnamen, einen Security-Context zu. Die SELinux-Policy wertet anschließend dann nur noch den Security-Context aus und muss sich um die Dateinamen nicht weiter kümmern. Dies verhindert Probleme bei der Auswertung von Hard-Links, *Chroot*-Umgebungen, etc.

Die File-Context-Definitionen der Policy werden in den Dateien `/etc/selinux/<policy>/contexts/files/*` gespeichert und aus den geladenen Modulen hier zusammengefasst. Diese Definitionen werden nur ausgewertet, wenn eine Datei spezifisch neu gelabelt wird. Dies kann durch ein komplettes Relabeling des Systems oder durch die Befehle `restorecon` oder `setfiles` erfolgen. Beachten Sie, dass *Customizable Types* nicht bei einem Relabeling verändert werden (siehe Abschnitt 16.4). Ansonsten werden diese Dateien nicht genutzt! Neue Dateien werden also nicht mit den hier definierten File-Contexts angelegt!

Im Folgenden ist ein Auszug aus der Datei `files_contexts` dargestellt:

```
/.*     system_u:object_r:default_t:s0
/xen(/.*)?      system_u:object_r:xen_image_t:s0
/mnt(/[^/]*)    -l      system_u:object_r:mnt_t:s0
/mnt(/[^/]*)?   -d      system_u:object_r:mnt_t:s0
```

[2] Die (3) deutet an, dass die Manpage des Befehls im Kapitel 3 der Manpages zu finden ist.

[3] Sie relabeln das gesamte System, indem Sie die Datei `/.autorelabel` erzeugen und das System neu booten.

```
/bin/.* system_u:object_r:bin_t:s0
/dev/.* system_u:object_r:device_t:s0
/lib/.* system_u:object_r:lib_t:s0
/var/.* system_u:object_r:var_t:s0
/srv/.* system_u:object_r:var_t:s0
/tmp/.* <<none>>
/usr/.* system_u:object_r:usr_t:s0
/sys/.* <<none>>
/opt/.* system_u:object_r:usr_t:s0
/etc/.* system_u:object_r:etc_t:s0
/mnt/[^/]*/.*   <<none>>
/dev/.*mouse.*  -c      system_u:object_r:mouse_device_t:s0
/dev/.*tty[^/]* -c      system_u:object_r:tty_device_t:s0
```

Der Aufbau der Datei ist einfach. Sie besteht aus drei Spalten. Die erste Spalte gibt den Pfad der zu labelnden Dateien an. Hierbei handelt es sich immer um einen regulären Ausdruck. Die mittlere Spalte ist optional und definiert den Dateitypen. Hier sind die folgenden Angaben möglich[4]:

- Nichts: Jede beliebige Datei
- `--`: Nur einfache Dateien
- `-d`: Verzeichnisse
- `-l`: Symlinks
- `-c`: Character Devices
- `-b`: Block Devices
- `-s`: Sockets
- `-p`: Pipes

Die dritte Spalte enthält den Security-Context der Datei. Steht hier ein `<<none>>`, so bedeutet das, dass diese Datei zur Laufzeit gelabelt wird. Der richtige Security-Context für eine Datei wird mit dem `matchpathcon(3)`-Bibliotheksaufruf[5]gesucht. Hierbei wird immer der »closest match« bevorzugt. Wenn Sie sich dafür interessieren, welchen Security-Context eine Datei erhält, können Sie auch den Befehl `matchpathcon` aufrufen.

Wie Sie diese File-Contexts selbst anpassen können, wird in Abschnitt 16.3 und Abschnitt 24.3 genauer erläutert.

27.1.2 Task-basiertes Labeling

Bei dem task-basierten Dateisystem-Labeling mit `fs_use_task` erbt eine neue Datei ihren Security-Context von dem erzeugenden Prozess. Der Prozess kann nicht den Security-Context anschließend ändern. Diese Funktion wird für Dateisysteme

[4] Diese Angabe entspricht auch der Angabe bei dem Befehl `find`.

[5] Die 3 in Klammern zeigt an, dass die Funktion im Kapitel 3 der Manpages erläutert wird. Da es hier auch einen Befehl in Kapitel 8 gibt, müssen Sie die Manpage wie folgt anzeigen: `man 3 matchpathcon`.

genutzt, die nicht zur Speicherung von Daten dienen, sondern spezielle Ressourcen zur Kommunikation des Prozesses enthalten. Typisch ist diese Funktion zum Beispiel für Sockets und Pipes. Diese Strukturen werden von dem Linux-Kernel in einem eigenen Pseudo-Dateisystem verwaltet.

Um dieses Labeling zu definieren, verwendet die SELinux-Policy die folgende Zeile:

```
fs_use_task sockfs gen_context(system_u:object_r:fs_t,s0);
```

Diese Zeile definiert, dass das Dateisystem den Typ *fs_t* erhält und alle in ihm angelegten Dateien den Security-Context des aufrufenden Prozesses erben. Hierdurch kann in der Policy dann auf den Typ des Sockets mit *self* zugegriffen werden:

```
allow calamaris_t self:unix_stream_socket create_stream_socket_perms;
```

27.1.3 Transitionsbasiertes Labeling

Das transitionsbasierte Labeling mit `fs_use_trans` ähnelt dem task-basierten Labeling. Jedoch wird hier nicht der Security-Context des erzeugenden Prozesses vererbt, sondern die neue Datei erhält ihren Security-Context abhängig von Typen-Transitionen (`type_transition`).

Jedoch unterscheidet sich diese Typen-Transition leicht von der bereits weiter oben (siehe Kapitel 27.1.1) erläuterten Typen-Transition bei Dateisystemen mit erweiterten Attributen. Während dort der neue Security-Context basierend auf dem Security-Context des erzeugenden Prozesses und des Verzeichnisses ermittelt wird, wird bei dem transitionsbasierten Labeling der Security-Context aus dem Security-Context des erzeugenden Prozesses und des Dateisystems berechnet. Diese Dateisysteme unterstützen keine Ordner. Existiert keine `type_transition`-Regel, so erbt das Objekt den Security-Context des Dateisystems.

Ein Beispiel verdeutlicht das:

```
fs_use_trans tmpfs gen_context(system_u:object_r:tmpfs_t,s0);
```

Diese Zeile in der Datei `filesystem.te` definiert, dass das `tmpfs`Dateisystem transitionsbasiert neuen Objekten ihren Security-Context zuweist. Das Dateisystem selbst erhält den Security-Context *system_u:object_r:tmpfs_t*. Alle Objekte, für die keine Typen-Transition definiert ist, erhalten ebenfalls diesen Security-Context.

```
type_transition havp_t tmpfs_t:file havp_data_t;
```

Diese Zeile würde alle Dateien, die ein Prozess mit der Domäne *havp_t* in dem `tmpfs` erzeugen würde, mit dem Typ *havp_data_t* ausstatten.

27.1.4 Generelles Labeling

Einige Dateisysteme, die eine dauerhafte Speicherung von Dateien erlauben, unterstützen nicht die Speicherung von Security-Contexts. Hier können daher nicht spezifische Security-Contexts für die unterschiedlichen Dateien verwaltet werden. Diese

Dateisysteme und die in ihnen gespeicherten Dateien erhalten alle denselben Security-Context. In der SELinux-Richtlinie wird dies im Quelltext in der Datei `filesystem.te` definiert:

```
genfscon fat / gen_context(system_u:object_r:dosfs_t,s0)
```

Diese Zeile definiert, dass alle `fat`-Dateisysteme und alle dort gespeicherten Dateien den angegebenen Security-Context erhalten. Hierbei ist der Pfad in der Spalte drei ein relativer Pfad zur Wurzel des Dateisystems. Während dies bei klassischen Dateisystemen keinen Sinn macht, wird diese Struktur bei dem `proc`Dateisystem intensiv genutzt:

```
genfscon proc /sys gen_context(system_u:object_r:sysctl_t,s0)
genfscon proc /irq gen_context(system_u:object_r:sysctl_irq_t,s0)
genfscon proc /net/rpc gen_context(system_u:object_r: ↵
         sysctl_rpc_t,s0)
genfscon proc /sys/fs gen_context(system_u:object_r: ↵
         sysctl_fs_t,s0)
genfscon proc /sys/kernel gen_context(system_u:object_r: ↵
         sysctl_kernel_t,s0)
genfscon proc /sys/kernel/modprobe gen_context(system_u:object_r: ↵
         sysctl_modprobe_t,s0)
```

Hier handelt es sich bei der Spalte drei um die verschiedenen Unterverzeichnisse in dem `proc`Dateisystem. Die dort befindlichen Dateien erhalten so unterschiedliche Security-Contexts.

Eine Ausnahme in dem `proc`-Dateisystem stellen die Process-ID-Verzeichnisse für jeden einzelnen Prozess dar. Diese erhalten den Security-Context des dazugehörigen Prozesses.

Dieses pfadbasierte Labeling kann auch für `fat` und andere Dateisysteme genutzt werden. Dort ist es jedoch meist nicht sinnvoll, da es keine feste Vorgabe gibt, wie die Dateien und Verzeichnisse dort heißen.

27.2 Labeling von Netzwerkobjekten

Auch Netzwerkobjekte wie Netzwerkkarten, IP-Adressen und Ports erhalten einen Security-Context. Zusätzlich können auf modernen Kerneln sogar IP-Pakete und IPsec-Security-Associations einen Security-Context erhalten. Diese Funktionen werden im Folgenden erläutert.

27.2.1 Netzwerkkarten

Netzwerkkarten können auch einen eigenen Security-Context erhalten. Aktuell wird diese Möglichkeit jedoch nicht genutzt. Das Labeling erfolgt mit dem SELinux-Schlüsselwort `netifcon`:

```
netifcon eth0 gen_context(system_u:object_r:netif_t,s0) ↵
         gen_context(system_u:object_r:packet_t,s0)
```

Hierbei gibt die zweite Spalte die Netzwerkkarte an, die gelabelt werden soll. In der dritten Spalte befindet sich der Security-Context der Netzwerkkarte. In der vierten Spalte wird der Security-Context der Pakete angegeben, die über diese Netzwerkkarte empfangen wurden. Diese letzte Funktion wird in den Policys auch bei Fedora Core 7 noch nicht genutzt, obwohl ein Paket-Labeling mit *Netfilter* bereits möglich ist.

Netzwerkkarten, die nicht mit einem spezifischen Security-Context versehen werden, erhalten den initialen Security-Context: `gen_context(system_u:object_r:netif_t,s0 - mls_systemhigh)`

Es ist möglich, eine Netzwerkkarte mit einem eigenen Security-Context zu versehen. Dann müsste aber im Wesentlichen das Modul `corenetwork` kopiert werden, da ansonsten keine Applikation über diese Netzwerkkarte senden oder empfangen dürfte. Wenn Sie eine Applikation verwenden, die als einzige über eine bestimmte Netzwerkkarte versenden darf, können Sie das über diese Funktion realisieren.

27.2.2 IP-Adressen

Auch IP-Adressen können mit dem Schlüsselwort `nodecon` mit einem Security-Context versehen werden. Aber auch diese Funktion wird in den aktuellen Policys kaum genutzt. Alle IP-Adressen erhalten den folgenden initialen Security-Context: `gen_context(system_u:object_r:node_t,s0 - mls_systemhigh)`. Die Policy sieht vor, dass anschließend die IP-Adressen in unterschiedliche Gruppen eingeteilt werden und entsprechende Namen erhalten. Dies erfolgt in der Reference-Policy-Quelltextdatei `kernel/corenetwork.te.in`:

compat_ipv4	compat_ipv4_node_t
inaddr_any	inaddr_any_node_t
link_local	link_local_node_t
lo	lo_node_t
mapped_ipv4	mapped_ipv4_node_t
multicast	multicast_node_t
site_local	site_local_node_t
unspec	unspec_node_t
node_name	node_name_node_t

Dies wird aber aktuell noch nicht genutzt.

27.2.3 Ports

Auch Netzwerk-Ports können von SELinux mit einem Security-Context versehen werden. Dies wird exzessiv von der aktuellen Policy genutzt. Jeder Port erhält entweder einen eigenen Security-Context mit dem Schlüsselwort `portcon` oder den initialen Security-Context (SID) `gen_context(system_u:object_r:port_t,s0)`.

Ein spezifischer Security-Context wird mit dem Schlüsselwort `portcon` zugewiesen:

```
portcon tcp 1-511 gen_context(system_u:object_r:reserved_port_t, s0)
portcon tcp 80 gen_context(system_u:object_r:httpd_port_t, s0)
```

Hierbei werden in der Spalte zwei das Protokoll und in der Spalte drei die Portnummer angegeben. Als Port kann auch ein Portbereich angegeben werden. Auch hier gilt, dass der »closest match« Vorrang hat. Der Port 80 erhält also den Typ *httpd_port_t*.

Für die Definition des Security-Contexts eines Ports bietet das Modul `corenetwork` zwar eine Schnittstelle an, die aber in den aktuellen Versionen der Policys nicht genutzt werden kann.

Hier besteht im Moment nur die Möglichkeit, selbst das Schlüsselwort zu verwenden oder, wie in Kapitel 16.2 beschrieben, den Security-Context mit `semanage` zu setzen.

27.2.4 IP-Pakete

Auch IP-Pakete können gelabelt werden. Dies erfolgt aber aktuell nicht durch die Policy, sondern über *Netfilter* mit dem Kommando `iptables`. Hierzu können Sie die folgende Syntax verwenden:

```
iptables -t mangle -A OUTPUT -p udp --dport 64005 -j SECMARK ➥
         --selctx system_u:object_r:traceroute_client_packet_t:s0
```

Neue Entwicklungen in diesem Bereich sind auch die Netlabel-Funktionalitäten des Kernel 2.6.19. Weitere Informationen erhalten Sie unter *http://netlabel.sourceforge.net/*.

27.2.5 Security-Associations

Auch IPsec-Security-Associations (*IPsec-SA*) können ab Kernel 2.6.16 mit einem Security-Context versehen werden. Diese werden dann auf die übertragenen Pakete kopiert. Bisher wird dies jedoch nicht genutzt. Die IPsec-SA erhalten aufgrund der aktuellen Policy keinen Security-Context und gelten daher als *unlabeled_t*.

Sie können dafür sorgen, dass diese IPsec-SAs einen Security-Context erhalten, indem Sie den Befehl `setkey` verwenden.

Aus der `setkey`-Manpage:

```
-ctx doi algorithm context-name
     Specify an access control label. The access control
     label is interpreted by the LSM (e.g., SELinux).
     Ultimately, it enables MAC on network communications.
     doi         The domain of interpretation, which is
                 used by the IKE daemon to identify the
                 domain in which negotiation takes place.
     algorithm   Indicates the LSM for which the label is
```

```
                generated (e.g., SELinux).
    context-name
                The string representation of the label
                that is interpreted by the LSM.
```

Eine sinnvolle Anwendung in den von den Distributionen zur Verfügung gestellten Policys gibt es aber noch nicht. Sie müssten hierzu die Policys selbst anpassen. Weitere Informationen erhalten Sie unter *http://www.cse.psu.edu/~tjaeger/research/netmac.html*.

27.3 Labeling weiterer Objekte

Weitere Objekte verfügen über eigene Object-Manager und benötigen ebenfalls einen Security-Context. Allerdings müssen Sie sich über diese Objekte meist keine Gedanken machen. Das Labeling erfolgt automatisch und kann von Ihnen nicht beeinflusst werden. Der Vollständigkeit halber wird es hier aufgeführt.

27.3.1 Socket

Jeder Socket eines Prozesses erbt automatisch den Security-Context des erzeugenden Prozesses. Die Vergabe des Security-Contexts ist task-orientiert. Für einen Zugriff auf den Socket wird also eine Regel nach dem folgenden Muster benötigt:

```
allow calamaris_t self:unix_stream_socket create_stream_socket_perms;
```

Sockets, die von dem Kernel erzeugt wurden, erhalten den Security-Context `gen_context(system_u:object_r:unlabeled_t,mls_systemhigh)`.

27.3.2 System V IPC

System V-InterProcessCommunication-Objekte werden entweder task-orientiert oder transitionsorientiert gelabelt. Während *msg*-Objekte transitionsorientiert gelabelt werden, werden *shm*-, *sem*- und *msgq*-Objekte task-orientiert gelabelt.

27.3.3 Capability

Capabilitys, die von einem Prozess ausgeübt werden, haben immer den Security-Context des Prozesses. Daher wird der Zugriff über Regeln erlaubt, die der folgenden ähneln:

```
allow loadkeys_t self:capability { setuid sys_tty_config };
```

Hier kann das Schlüsselwort *self* verwendet werden.

27.3.4 Prozess

Auch Prozesse besitzen einen Security-Context. Dies ist eines der wesentlichen Prinzipien, auf denen die Sicherheit von SELinux aufbaut. Dieser Security-Context wird

zunächst von dem Elternprozess geerbt. In bestimmten Fällen können Domänentransitionen erfolgen. Dies ist bereits ausführlich in Kapitel 12 erläutert worden. In seltenen Fällen ist auch eine dynamische Domänentransition möglich. Diese kann von dem Prozess durch einen `setcon(3)`-Funktionsaufruf ausgelöst werden und muss von SELinux mit dem Schlüsselwort `dyntransition` erlaubt werden. Aktuell wird dies aber kaum genutzt. Die Entwickler von SELinux sehen diese Funktion auch berechtigterweise als gefährlich an.

27.3.5 System und Security

Diese beiden Objekte existieren jeweils nur einmal auf einem SELinux-System. Sie erhalten automatisch von dem Kernel ihren Security-Context, der im Weiteren auch nicht geändert werden kann. Daher ist deren Betrachtung nicht weiter relevant.

28 Entwicklung unter der Strict-Policy

In diesem Kapitel werde ich das Beispiel aus Kapitel 24 bei Anwendung der Strict-Policy wiederholen. Anschließend (siehe Abschnitt 28.3) werde ich die Erzeugung von zusätzlichen Rollen und ihre Zuweisung an Benutzer beschreiben. Hiermit können Sie einzelne Aufgaben, die normalerweise nur *root* durchführen dürfte, delegieren.

28.1 Der Befehl date

In diesem Abschnitt werden wir das Beispiel aus Kapitel 24 wiederholen. Ich werde mich dabei auf die wesentlichen Unterschiede bei Anwendung der Strict-Policy beschränken. Bitte lesen Sie daher vorher Kapitel 24, und vielleicht versuchen Sie auch, die dort erwähnten Schritte nachzuvollziehen, bevor Sie mit diesem Kapitel beginnen. Unterschiede treten hier insbesondere in den verwendeten Ausgangsdomänen bei der Domänentransition auf.

Im Grunde starten wir bei der Erzeugung von Richtlinien für die Strict-Policy genauso wie bei der Targeted-Policy. Wir erzeugen die drei Dateien:

- `date.te`
- `date.fc`
- `date.if`

Während die Datei `date.if` zunächst leer sein kann, enthält die Datei `date.fc` denselben Inhalt wie bei der Targeted-Policy:

```
/bin/date -- gen_context(system_u:object_r:date_exec_t,s0)
```

Die Datei `date.te` sieht aber ein wenig anders aus:

```
policy_module(date,1.0.0)

type date_t;
type date_exec_t;

domain_type(date_t)
domain_auto_trans(domain, date_exec_t, date_t)
domain_entry_file(date_t, date_exec_t)
```

Die Domäne *unconfined_t*, die wir in der Targeted-Policy als Ausgangsdomäne für die Domänen-Transition angegeben haben, existiert nicht in der Strict-Policy. Die unterschiedlichen Benutzer arbeiten nun in unterschiedlichen Domänen. Einfache Benutzer arbeiten in der Domäne *user_t*. Der Benutzer *root* arbeitet in der Domäne *staff_t* oder *sysadm_t*. Anstatt jede dieser Domänen einzeln anzugeben, können wir auch ein *Attribut* benutzen, das diese Domänen gruppiert: *userdomain*. Dann wäre es aber nur den Benutzern erlaubt, den Befehl `date` aufzurufen. Da es aber wahrscheinlich sehr viele Scripts gibt, die ebenfalls den Befehl benötigen, ist es hier sinnvoll, allen Domänen den Wechsel in die Domäne *date_t* zu erlauben. Wir schränken dann weiterhin die Möglichkeiten des Befehls `date` ein. Alle Domänen können wir sehr einfach mit dem Attribut *domain* beschreiben.

Nachdem wir das *Policy!-Package* übersetzt haben (siehe Abschnitt 24.4), können wir das Policy-Package laden und den Befehl `date` neu labeln:

```
[root@supergrobi date_mod]# semodule -i date.pp
[root@supergrobi date_mod]# restorecon /bin/date
[root@supergrobi date_mod]# ls -Z /bin/date
-rwxr-xr-x  root root system_u:object_r:date_exec_t    /bin/date
```

Nun sollte sich das System im Permissive-Modus befinden, um den Befehl zu testen und die Richtlinienverletzungen zu protokollieren. Anschließend verwenden wir wieder den Befehl `audit2allow`, um aus den protokollierten Richtlinienverletzungen Regeln zu erzeugen.

```
[root@supergrobi date_mod]# date
So 22. Apr 14:10:03 CEST 2007
[root@supergrobi date_mod]# audit2allow -R -l -i ⏎
         /var/log/audit/audit.log >> date.te
```

Der Befehl `audit2allow` hat die neuen Regeln an die vorhandene Datei `date.te` angehängt. Diese müssen wir jetzt analysieren und möglicherweise durch weitere Schnittstellen aus der Reference-Policy ergänzen. Ich habe im Weiteren zur Erhöhung der Lesbarkeit auf die `optional_policy`-Blöcke verzichtet. Des Weiteren habe ich alle Regeln gelöscht, die nicht die Domäne *date_t* als Source-Domäne verwenden. Zusätzlich habe ich die Regeln sortiert, sodass die bereits erkannten Schnittstellen vor den `allow`-Regeln aufgeführt werden:

```
policy_module(date,1.0.0)

type date_t;
type date_exec_t;

domain_type(date_t)
domain_auto_trans(domain, date_exec_t, date_t)
domain_entry_file(date_t, date_exec_t)
```

```
### Neu durch audit2allow
role staff_r types date_t;

files_search_etc(date_t)
libs_use_ld_so(date_t)
libs_search_lib(date_t)
libs_read_lib_files(date_t)
files_search_usr(date_t)
miscfiles_read_localization(date_t)

allow date_t shlib_t:file { execute getattr read };
allow date_t staff_devpts_t:chr_file { getattr read write };
allow date_t staff_t:fd use;
allow date_t staff_t:process sigchld;
```

Als neue Komponente fällt die Zeile *role* auf. Diese Zeile gibt nun an, welche Rollen auf eine Domäne zugreifen dürfen.

Zunächst müssen wir die restlichen `allow`-Zeilen durch Schnittstellen ersetzen. Wir gehen genauso vor wie bei der Targeted-Policy. Nach einigen Versuchen und Anpassungen könnte die TE-Datei wie folgt aussehen:

```
policy_module(date,1.0.3)

type date_t;
type date_exec_t;

domain_type(date_t)
domain_auto_trans(domain, date_exec_t, date_t)
domain_entry_file(date_t, date_exec_t)

### Neu durch audit2allow
role staff_r types date_t;

files_search_etc(date_t)
libs_use_ld_so(date_t)
libs_search_lib(date_t)
libs_read_lib_files(date_t)
files_search_usr(date_t)
miscfiles_read_localization(date_t)
libs_use_shared_libs(date_t)
locallogin_use_fds(date_t)
userdom_use_user_terminals(staff,date_t)
userdom_use_all_users_fds(date_t)
userdom_sigchld_all_users(date_t)
```

Einige Anpassungen sind noch erforderlich, wenn alle Benutzer den Befehl `date` verwenden sollen. Unter anderem sind das die Rollenberechtigungen. Die folgenden Zeilen bestimmen, dass die entsprechenden Rollen Prozesse in der Domäne *date_t* ausführen dürfen.

```
role user_r types date_t;
role sysadm_r types date_t;
role system_r types date_t;
```

Sicherlich sollten Sie auch noch den Rechner einmal durchstarten und anschließend nach weiteren Meldungen suchen, die auf Fehler in der Domäne *date_t* hinweisen. Möglicherweise existieren noch weitere Konstellationen, in denen dieser Befehl gebraucht wird.

28.2 Ein Modul für die Targeted- und Strict-Policy

In diesem Abschnitt wollen wir die TE-Dateien des `date`-SELinux-Policy-Package zusammenführen. Mit Hilfe von `if`-Anweisungen können wir eine TE-Datei bauen, die in beiden Policys funktioniert.

Hierzu verwenden wir folgendes Konstrukt:

```
ifdef(‘targeted_policy’,‘
    Targeted-Policy
’,‘
    Strict-Policy (und auch MLS)
’)
```

Indem wir die Zeilen, die jeweils nur in der Strict- oder Targeted-Policy benötigt werden, isolieren, können wir sehr leicht ein Modul bauen, das für beide Policys übersetzt und dann in beiden geladen werden kann.

Die fertige Policy sieht bei mir folgendermaßen aus. Sie kann bei Ihnen aufgrund Ihrer Vorgehensweise oder der Version Ihrer Policy leicht anders aussehen:

```
policy_module(date,1.1.0)

type date_t;
type date_exec_t;

domain_type(date_t)

ifdef(‘targeted_policy’,‘
    domain_auto_trans(unconfined_t, date_exec_t, date_t)
’,‘
    domain_auto_trans(domain, date_exec_t, date_t)
    role staff_r types date_t;
    role user_r types date_t;
    role sysadm_r types date_t;
```

```
')
domain_entry_file(date_t, date_exec_t)

files_search_etc(date_t)
libs_use_ld_so(date_t)
miscfiles_read_localization(date_t)
files_search_usr(date_t)
libs_search_lib(date_t)
libs_read_lib_files(date_t)
libs_use_shared_libs(date_t)

ifdef(`targeted_policy',`
   term_use_generic_ptys(date_t)
',`
   userdom_use_user_terminals(staff,date_t)
   userdom_use_user_terminals(user,date_t)
   locallogin_use_fds(date_t)
   userdom_use_sysadm_ptys(date_t)
   userdom_use_all_users_fds(date_t)
   userdom_sigchld_all_users(date_t)
')

gen_tunable(date_adjust_time,false)
tunable_policy(`date_adjust_time',`allow date_t self:capability ➥
         sys_time;');
```

28.3 Weitere Rollen zur Delegation

Eine besondere Eigenschaft der Strict-Policy ist die Aufteilung des Systems in unterschiedliche Rollen. Da das System bereits eine saubere Trennung der Benutzer durchführt, kann dies recht einfach genutzt werden, um weitere Rollen für bestimmte Aufgaben hinzuzufügen. Möchten wir zum Beispiel eine Rolle hinzufügen, die es erlaubt, den Webserver zu verwalten, ist das recht einfach möglich.

Zunächst brauchen wir einen Linux-Benutzer, der den Webserver neu starten darf. Hierzu sind unter Linux zwingend *root*-Rechte erforderlich. Wenn Sie das nicht möchten, können Sie den Webserver auch auf einem hohen Port starten. Dann kann er auch mit den Rechten eines normalen Benutzers gestartet werden, wenn Sie zusätzlich alle Dateirechte anpassen. Hier gehen wir aber davon aus, dass er weiter auf Port 80 betrieben werden soll. Wir legen also einen Benutzer *webmaster* an, der ein eigenes Heimatverzeichnis erhält und unter Linux mit der ID 0 arbeitet. Wir wollen diesen Benutzer anschließend so mit SELinux einschränken, dass er nur den Webserver administrieren darf.

```
[root@supergrobi ~]# useradd -o -u 0 -g 0 webmaster
[root@supergrobi ~]# passwd webmaster
```

```
Changing password for user webmaster2.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

Nun erzeugen wir ein SELinux-Policy-Modul:

```
 1 policy_module(webmaster,1.1.0)
 2
 3 require {
 4    type httpd_t;
 5    type consoletype_t;
 6    type httpd_log_t;
 7 }
 8
 9 userdom_base_user_template(webmaster)
10 role webmaster_r types { httpd_t consoletype_t initrc_t};
11 init_domtrans_script(webmaster_t)
12 apache_manage_all_content(webmaster_t)
13 apache_manage_config(webmaster_t)
14 apache_manage_log(webmaster_t)
15 apache_manage_sys_content(webmaster_t)
16 kernel_dontaudit_read_system_state(webmaster_t)
17 corecmd_exec_all_executables(webmaster_t)
18
19 allow consoletype_t webmaster_tty_device_t:chr_file ↵
    rw_term_perms ;
20 allow httpd_t webmaster_tty_device_t:chr_file ↵
    rw_term_perms ;
21
22 gen_user(webmaster_u,webmaster,webmaster_r,s0,s0)
```

Ich habe diesmal das Listing mit Zeilennummern versehen, da ich mich im Folgenden auf die Zeilennummern beziehen werde.

In der Zeile 9 erzeuge ich mit der Schnittstelle `userdom_base_user_template` sämtliche Typen, die für den Benutzer *webmaster* erforderlich sind. Zusätzlich erhält die Domäne *webmaster_t* die wichtigsten Privilegien, die für einen unprivilegierten Benutzer erforderlich sind. Sie können die Regeln im Einzelnen in der Schnittstellendefinition `userdomain.if` nachlesen.

In der Zeile 10 definiere ich, dass die Rolle *webmaster_r* erzeugt werden soll und Zugriff auf die Domänen *httpd_t*, *consoletype_t* und *initrc_t* erhält. Bei dem Start des Webservers über das SysV-Init-Script (`/etc/init.d/httpd`) erfolgt zunächst ein Wechsel in die Domäne *initrc_t*. Für die Ausgabe erfolgt ein Wechsel in die Domäne *consoletype_t*. Für den Start des Webservers erfolgt anschließend ein Wechsel in die Domäne *httpd_t*. Weitere Domänen darf der Benutzer mit dieser Rolle später nicht erreichen.

In der Zeile 11 bestimme ich, dass bei dem Aufruf des SysV-Init-Scripts ein Wechsel in die Domäne *initrc_t* erfolgen soll. Dies ist für den erfolgreichen Aufruf des Scripts erforderlich.

Die vier Zeilen 12 bis 15 versehen die Domäne *webmaster_t* mit den notwendigen Rechten, um die Konfigurationsdateien des Webservers und die Webinhalte zu administrieren.

Die Zeile 16 unterdrückt unnötige Protokollmeldungen, und die Zeile 17 erlaubt der Domäne *webmaster_t* den Aufruf der wichtigsten Befehle (`vi`, `less` etc.).

Achtung

Der Compiler erzeugte bei mir immer einen Fehler, wenn sich die Zeile 17 mit dem Makro `gen_user` weiter oben in der TE-Datei befand. Es dauerte eine Weile, bis ich beim ersten Mal diesen Fehler gefunden hatte. Achten Sie darauf, dass sich dieses Makro als letzte Zeile in der TE-Datei wiederfindet.

Die folgenden `allow`-Zeilen ergänzen die Schnittstellenaufrufe.

Die letzte Zeile schließlich erzeugt den SELinux-Benutzer *webmaster_u*. Dieser erhält als einzige Rolle *webmaster_r*.

Nun müssen Sie nur noch einem Linux-Benutzer diesen SELinux-Benutzer zuweisen. Dies erfolgt elegant mit `semanage`:

```
[root@supergrobi ~]# semanage login -a -s webmaster_u webmaster
[root@supergrobi ~]# semanage login -l
Login Name                SELinux User              MLS/MCS Range
...
webmaster                 webmaster_u               s0
```

Wenn sich nun der Benutzer *webmaster* anmeldet, verfügt er über die erforderlichen Privilegien, um den Webserver neu zu starten und dessen Konfigurationsdateien zu editieren. Weitere Funktionen (selbst das Ändern des Kennworts) sind nicht möglich. Sicherlich ist dieses Beispiel noch ausbaufähig, aber es sollte Ihnen auch nur als Grundgerüst für eigene Experimente dienen.

29 Die komplett eigene Policy

Ich kann einem Anfänger nicht empfehlen, eine komplette eigene Policy aufzustellen. Dies würde sicherlich nicht zum Erfolg führen. Auch für fortgeschrittene Anwender wird das eine sehr mächtige Aufgabe sein. Falls Sie dies vorhaben, sollten Sie sich sicherlich an der Reference-Policy orientieren und zunächst mit einem sehr kleinen System beginnen.

Weitere Ansätze und Anregungen bieten alternative Policys, so wie die in Abschnitt 30.3 besprochene Variante, die gemeinsam mit dem Editor `seedit` eingesetzt wird.

Am wahrscheinlichsten kommt der Wunsch nach einer eigenen Policy auf, wenn Sie SELinux auf einem Linux-System einsetzen möchten, für das es bisher noch keine Unterstützung seitens der Reference-Policy gibt. Die Reference-Policy unterstützt im Moment (mehr oder weniger) die folgenden Distributionen:

- Red Hat (Fedora, RHEL)
- Debian
- Gentoo
- SUSE

Falls Sie nun versuchen, die Reference-Policy für eine weitere, möglicherweise eigene Distribution anzupassen, müssen Sie wissen, wie die Reference-Policy installiert wird.

Hierzu laden Sie zunächst die Reference-Policy von ihrer Homepage (*http://serefpolicy.sf.net* oder *http://oss.tresys.com/projects/refpolicy/wiki/WikiStart*). Entpacken Sie die Reference-Policy an einem geeigneten Ort (z.B. `/tmp`). Anschließend rufen Sie den folgenden Befehl auf, der die Reference-Policy automatisch in `/etc/selinux/refpolicy/src/policy/` installiert:

```
#  make install-src
```

Nun müssen Sie die Datei `build.conf` in dem Source-Verzeichnis editieren. Hier befinden sich zu Beginn einige Einstellungen, die Sie anpassen können.

```
########################################
#
# Policy build options
#
```

```
# Policy version
# By default, checkpolicy will create the highest
# version policy it supports.  Setting this will
# override the version.  This only has an
# effect for monolithic policies.
#OUTPUT_POLICY = 18

# Policy Type
# strict, targeted,
# strict-mls, targeted-mls,
# strict-mcs, targeted-mcs
TYPE = strict

# Policy Name
# If set, this will be used as the policy
# name.  Otherwise the policy type will be
# used for the name.
NAME = refpolicy

# Distribution
# Some distributions have portions of policy
# for programs or configurations specific to the
# distribution.  Setting this will enable options
# for the distribution.
# redhat, gentoo, debian, suse, and rhel4 are current options.
# Fedora users should enable redhat.
#DISTRO = redhat

# Direct admin init
# Setting this will allow sysadm to directly
# run init scripts, instead of requring run_init.
# This is a build option, as role transitions do
# not work in conditional policy.
DIRECT_INITRC=n

# Build monolithic policy.  Putting n here
# will build a loadable module policy.
MONOLITHIC=y

# Number of MLS Sensitivities
# The sensitivities will be s0 to s(MLS_SENS-1).
# Dominance will be in increasing numerical order
# with s0 being lowest.
MLS_SENS=16
```

```
# Number of MLS Categories
# The categories will be c0 to c(MLS_CATS-1).
MLS_CATS=256

# Number of MCS Categories
# The categories will be c0 to c(MLS_CATS-1).
MCS_CATS=256

# Set this to y to only display status messages
# during build.
QUIET=n
```

Anschließend genügt es, die Policy mit `make install` zu installieren. Nun müssen Sie nur noch die Datei `/etc/selinux/config` editieren:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of these two values:
#       targeted - Only targeted network daemons are protected.
#       strict - Full SELinux protection.
SELINUXTYPE=refpolicy
```

Nach einem zweimaligen Relabeling des Systems sollte die Reference-Policy arbeiten.

```
# touch /.autorelabel
# reboot
```

Ich will hier natürlich nicht verschweigen, dass Sie, wenn Ihr System bisher keine SELinux-Unterstützung besitzt, noch wesentlich mehr Anpassungen durchführen müssen. Mindestens folgende Komponenten müssen angepasst und neu übersetzt werden:

1. Kernel
2. PAM
3. `ls`, `find`, `ps`, `id`
4. `init`
5. `glibc`

Die Patches für diese Applikationen sind, solange sie noch nicht upstream wieder in die Applikationen zurückgeflossen sind, unter *http://selinux.sourceforge.net/devel/userland.php3* zu finden.

30 SELinux-Policy-Editoren und -IDEs

Bei der Entwicklung von SELinux-Policys wünscht man sich als Administrator schnell eine Oberfläche, die einem bei der Erzeugung und Fehlersuche behilflich ist. Es gibt eine ganze Reihe von Programmen, die versuchen, dies zu leisten. Im Folgenden will ich eine Auswahl kurz vorstellen.

Es handelt sich im Einzelnen um:

- Vim
- SELinux Policy IDE (SLIDE)
- SELinux Policy Editor
- Cross Domain Solutions Framework (CDS Framework)

30.1 Vim als SELinux-Editor

Der Editor Vi oder Vim ist sicherlich für viele Administratoren das Werkzeug der Wahl zum Editieren von Textdateien. Zumindest auf mich trifft das zu. Vim kennt Syntax-Highlighting, das bei der Erstellung von Type-Enforcement-Dateien helfen kann. Sie erkennen dann auf den ersten Blick, ob die Syntax Ihrer Datei stimmt oder nicht. Hierzu benötigen Sie jedoch eine Syntax-Beschreibungsdatei. Eine derartige Datei wurde von Thomas Bleher erzeugt und unter der folgenden URL veröffentlicht: *http://www.cip.informatik.uni-muenchen.de/~bleher/selinux/te.vim*.

Diese Datei müssen Sie in Ihrem Verzeichnis `~/.vim/syntax` ablegen. Dann müssen Sie noch die folgende Zeile zu Ihrer Datei `~/.vimrc` hinzufügen:

```
autocmd BufRead,BufNewFile          *.te set filetype=te
```

Anschließend wird der Vim Type-Enforcement-Dateien mit Syntax-Highlighting in verschiedenen Farben darstellen. Einen Screenshot spare ich mir hier, da dieser in Schwarz-Weiß keinen Sinn machen würde.

30.2 SELinux Policy IDE (SLIDE)

Die SELinux Policy IDE ist ein Plugin für die Eclipse Integrated Development Environment (IDE). Eclipse ist ein Open-Source- Framework. Dabei unterstützt die gra-

fische Oberfläche Eclipse die Entwicklung von Software jeder Art. Die häufigste Anwendung ist sicherlich der Einsatz als Java- IDE. Dies ist auch der ursprüngliche Zweck der Software.

Die Firma Tresys hat für Eclipse ein SELinux-Plugin geschaffen, das Sie bei der Entwicklung von SELinux-Modulen unterstützt.

30.2.1 SLIDE-Installation

Bevor Sie SLIDE installieren, müssen Sie einige Voraussetzungen schaffen. Sie benötigen:

- Eclipse Version $\geq$ 3.1
- Setools
- Reference Policy
- Checkpolicy Version $\geq$ 1.28

Diese Pakete sind jedoch in den meisten modernen SELinux-Distributionen enthalten.

Auf der SLIDE-Homepage[1] stehen fertige Pakete für die Fedora Core-Distributionen zur Verfügung. Falls Sie diese Distribution nutzen, ist die Verwendung des RPM-Pakets sicherlich die einfachste Variante.

Falls Sie die Debian-Distribution einsetzen, können Sie SLIDE entweder über die Eclipse-Update-Site oder aus den Quellen installieren. Wenn Sie den Eclipse-Update wählen, nutzen Sie im Menü HELP den Punkt SOFTWARE UPDATES und dort FIND AND INSTALL. Nach Anwahl von SEARCH FOR NEW FEATURES TO INSTALL legen Sie eine neue Site mit den folgenden Informationen an:

- Name: Tresys Technology
- URL: *http://oss.tresys.com/eclipse-update/*

Nun können Sie direkt SLIDE aus dem Internet installieren. Ansonsten können Sie aber auch den Subversion-Zugang unter *svn co http://oss.tresys.com/repos/slide /trunk/slide-plugin* benutzen. Dieser letzte Weg wird jedoch nicht empfohlen.

30.2.2 SLIDE-Funktionen

SLIDE unterstützt Sie nun bei der Anlage und Verwaltung Ihrer Module. Es bietet Syntax-Highlighting und automatische Vervollständigung.

Um ein neues Projekt anzulegen, wählen Sie FILE NEW und dann PROJECT. In dem nun erschienenen Dialog (Abbildung 30.1) wählen Sie das SLIDE Project aus.

In dem nächsten Dialog wählen Sie, ob Sie nur ein Modul oder eine komplette neue Policy erzeugen möchten (Abbildung 30.2). Im zweiten Fall wird die vorhandene Reference-Policy für die Anpassung kopiert.

[1] *http://oss.tresys.com/projects/slide*

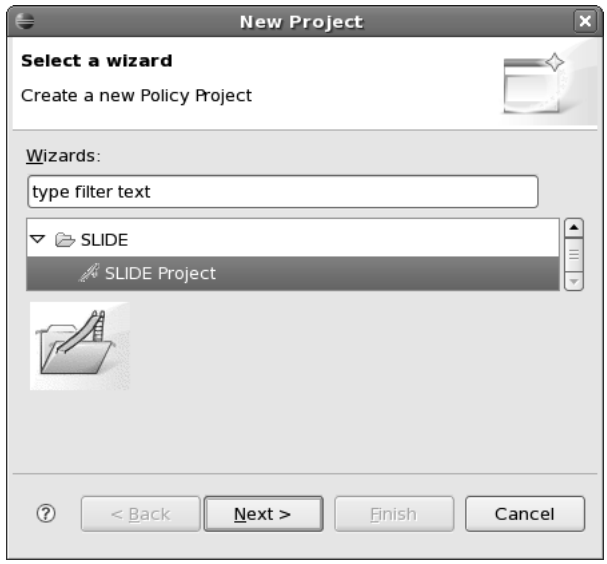

Abbildung 30.1: Eclipse bietet die Erzeugung von SLIDE-Projekten.

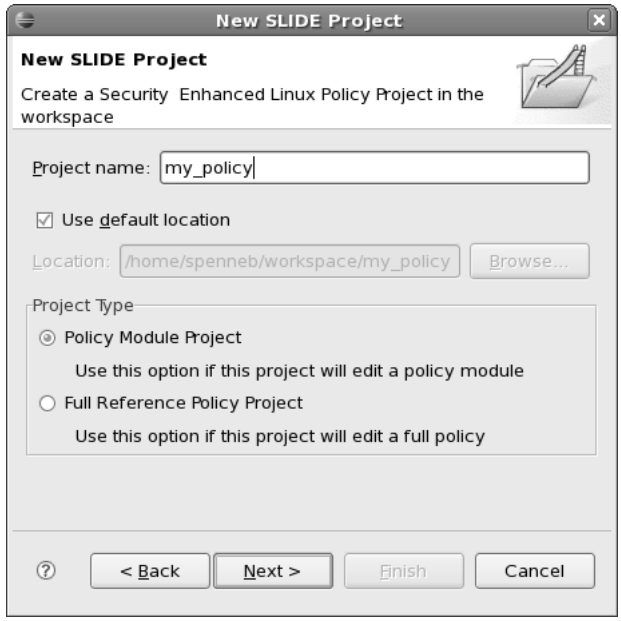

Abbildung 30.2: SLIDE kann Sie sowohl beim Erstellen von Policy-Modulen als auch beim Erstellen von ganz neuen Reference-Policys unterstützen.

Auf der folgenden Seite müssen Sie den Ort der Reference-Policy angeben (Abbildung 30.3). Möchten Sie nur ein Modul erzeugen, müssen Sie hier mindestens den

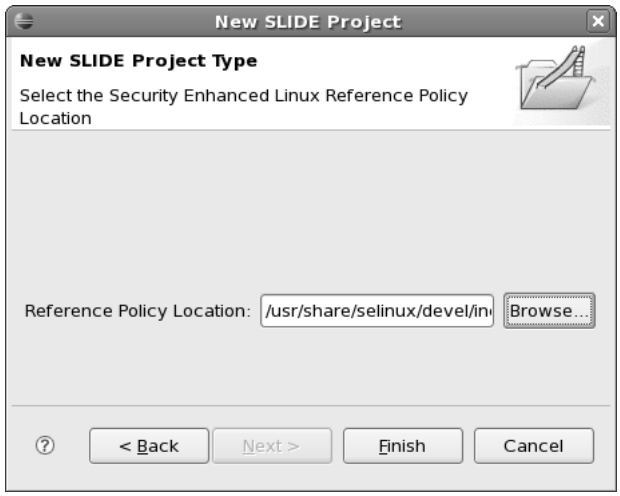

Abbildung 30.3: SLIDE benötigt den Pfad zur Reference-Policy.

Pfad zu den Include-Dateien der aktuellen Policy angeben. Möchten Sie eine neue Policy bauen, müssen sich hier auch alle Quellen befinden, denn SLIDE wird diese zunächst für Ihr Projekt kopieren, damit Sie diese anpassen können.

Nun können Sie ein neues Modul erzeugen. Hierzu wählen Sie FILE/NEW und dann SLIDE MODULE. In dem folgenden Fenster (Abbildung 30.4) geben Sie einen Namen für das Modul an.

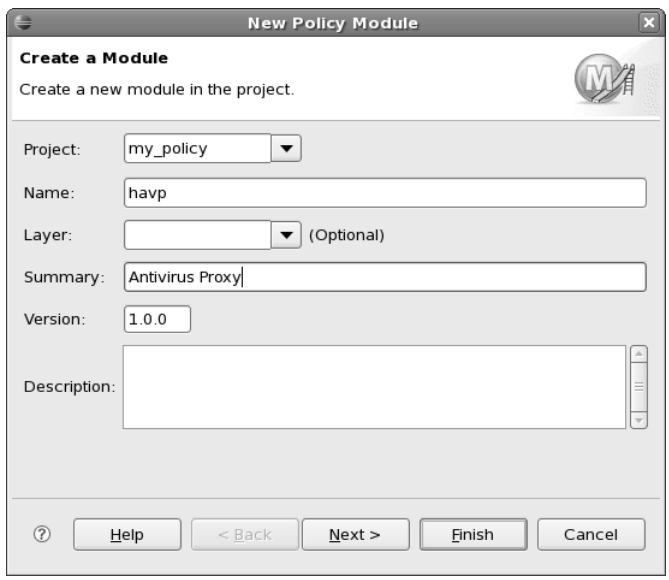

Abbildung 30.4: SLIDE fragt Sie nach den wesentlichen Informationen des Moduls.

Auf der nächsten Seite (Abbildung 30.5) stellt SLIDE Ihnen Fragen, die denen des Befehls `policygentool` ähneln. Auch aus diesen Fragen erzeugt SLIDE ein Grundgerüst für das Modul.

Anschließend können Sie in der IDE auf Ihre *Type-Enforcement*-Datei zugreifen. Bei dem Editieren unterstützt SLIDE Sie mit der Vervollständigung Ihrer Eingabe bei Bedarf. Hierzu drücken Sie einfach STRG+Leertaste (Abbildung 30.6). Die Auswahl erfolgt dann mit der Maus.

Über die rechte Maustaste steht auch noch ein Kontext-Menü mit Autotexten zur Verfügung, die größere Block-Templates automatisch hinzufügen können. Hier stehen zur Verfügung:

- ADD TUNABLE POLICY BLOCK
- ADD OPTIONAL POLICY BLOCK
- ADD IFDEF BLOCK

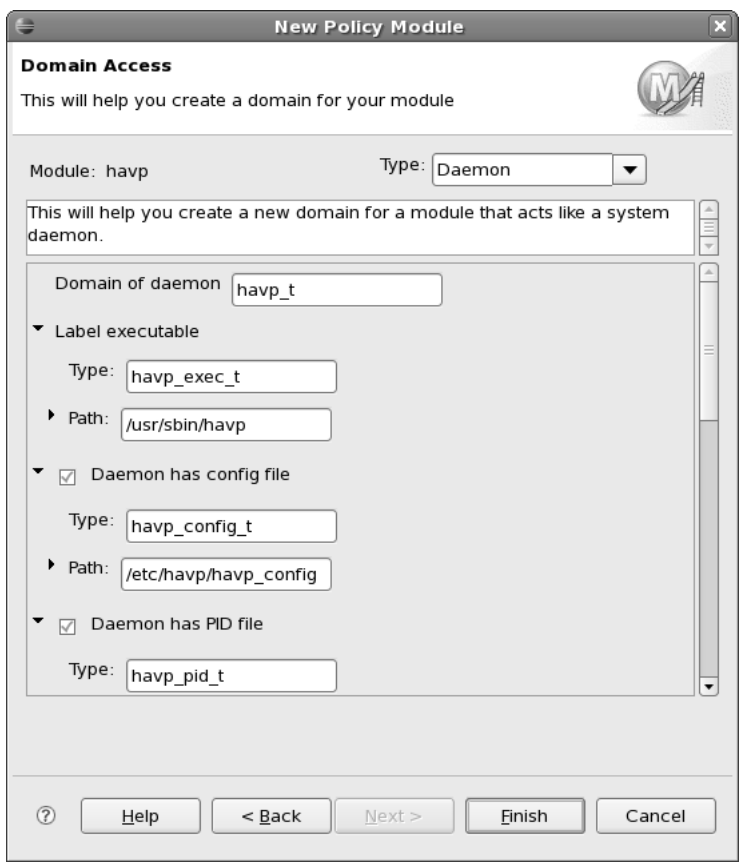

Abbildung 30.5: SLIDE erstellt aus den Antworten ein Grundgerüst für die Policy.

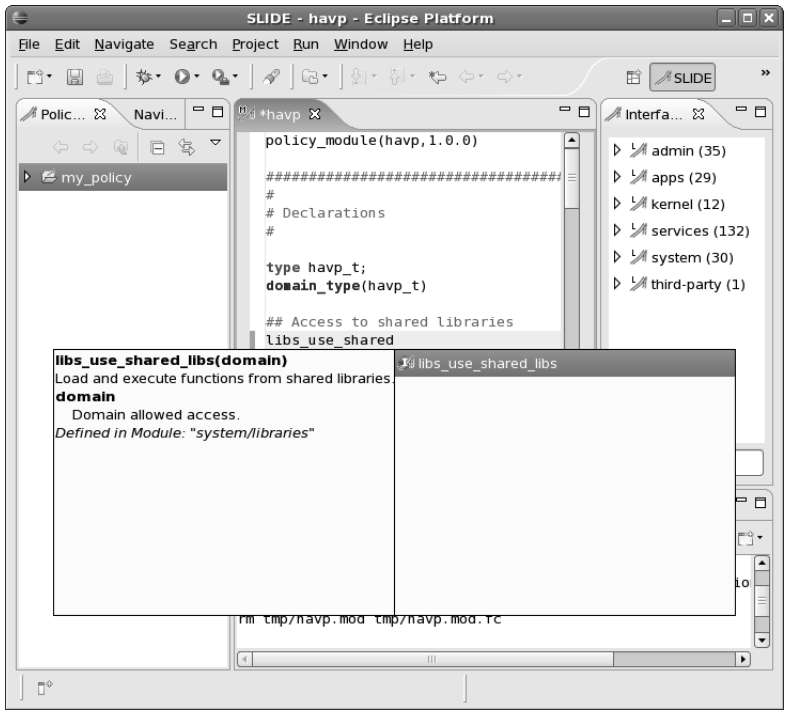

Abbildung 30.6: SLIDE kennt alle Interfaces und kann mit STRG+Leertaste eine Autovervollständigung anbieten.

- MOVE OUT OF BLOCK
- REMOVE BLOCK
- TOGGLE COMMENT

Ein Test des Moduls kann über die rechte Maustaste und das Menü RUN AS und RUN ausgelöst werden. Zunächst müssen Sie hier aber noch die Umgebung konfigurieren. Hierzu existiert nach der Auswahl POLICY TEST ein Button NEW[2]. Wählen Sie dann in der Registerkarte MAIN zunächst Ihr Projekt und dann in der Registerkarte POLICY Ihr Modul (Abbildung 30.7). Geben Sie der Konfiguration einen sinnvollen Namen, und Sie sind mit der Konfiguration fast fertig. Optional können Sie auch noch ein Test-Script angeben. Dies wird gestartet, sobald die Policy geladen wurde.

Damit der Test aber auch tatsächlich durchgeführt werden kann, benötigen Sie noch ein Testsystem, und SLIDE benötigt eine Netzwerkverbindung mit dem Testsystem. Hierbei kann es sich auch um das lokale System handeln. Auf dem Testsystem muss aber zuvor das Paket *SLIDE!-Remote* installiert sein. Dieser Dienst wird von SLIDE für die Kommunikation mit dem Testsystem genutzt. SLIDE erhält über diesen Dienst

[2] Dieser ist bei Fedora als Icon mit einem kleinen Plus oben versteckt.

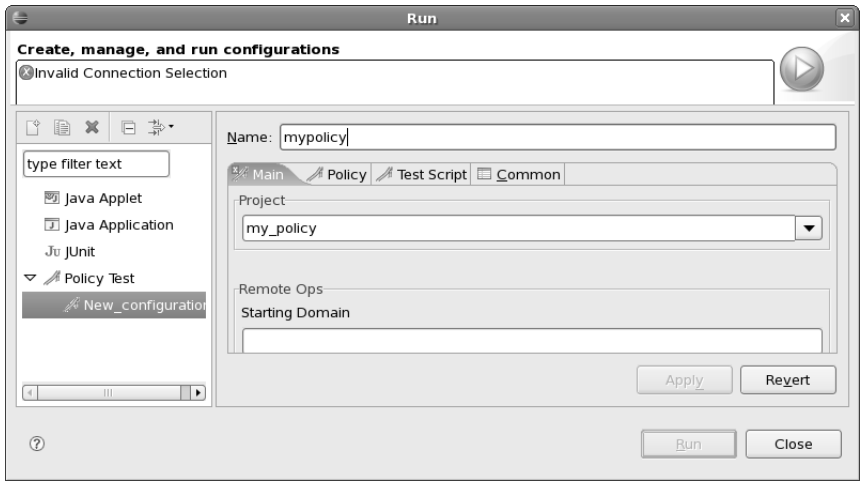

Abbildung 30.7: SLIDE erlaubt auch den Test der Module.

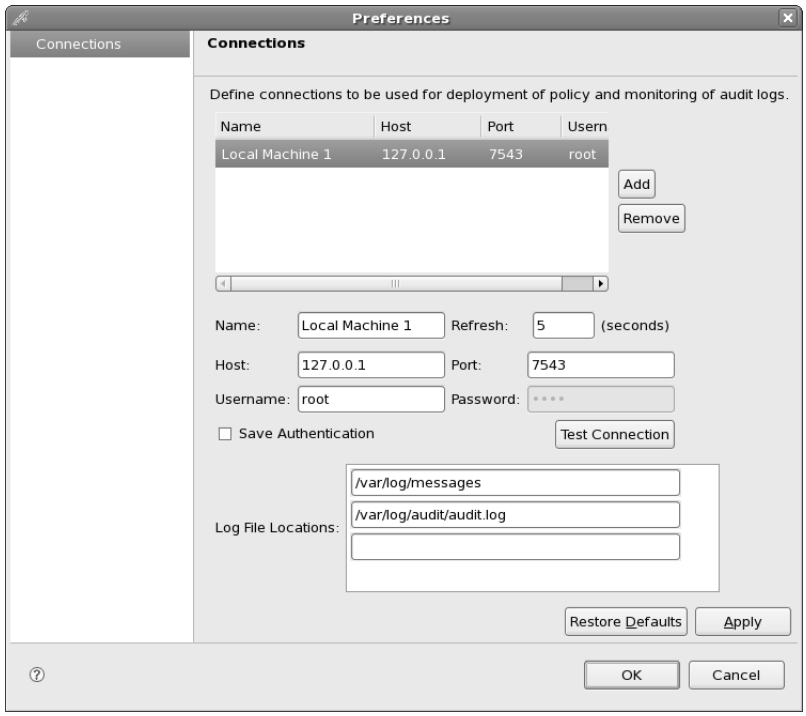

Abbildung 30.8: Für den Test benötigt SLIDE eine Verbindung zu SLIDE-Remote auf dem Testsystem.

die Audit-Meldungen des Testsystems und überträgt das Modul auf das Testsystem (Abbildung 30.8).

SLIDE-Remote ist auch über die SLIDE-Homepage verfügbar. Es sollte im Moment lediglich für die Entwicklung von SELinux-Richtlinien eingesetzt und nicht auf Produktionsystemen verwendet werden.

30.3 SELinux Policy Editor (SEedit)

Das Programm `seedit` wurde zuerst von Hitachi Software entwickelt[3]. Seitdem hat Yuichi Nakamura es überarbeitet und verbessert. Die aktuelle Version steht unter *http://seedit.sourceforge.net* zur Verfügung.

Bei SEedit handelt es sich im Wesentlichen um drei Komponenten:

- Eine neue SELinux-Policy (*Simplified-Policy*)
- Einen SELinux-Policy-Compiler für die Simplified Policy Description Language
- Einen grafischen Editor für die Verwaltung

Die neue Simplified Policy wurde in der *Simplified Policy Description Language* (SPDL)[4] geschrieben. Für die Übersetzung in die binäre SELinux-Policy enthält das Projekt den SPDL-Compiler `seedit-converter`. Dieser erzeugt die binäre Policy. Die Administration erfolgt sehr bequem über eine grafische Oberfläche.

30.3.1 Simplified Policy Description Language (SPDL)

Die Simplified Policy Description Language verwendet eine Syntax, die der Syntax ähnelt, die von AppArmor verwendet wird. Dadurch kapselt sie die komplizierten internen Funktionen wirkungsvoll, sodass der Anwender sich keine Gedanken um Typen, abstrakte Berechtigungen etc. machen muss. Am deutlichsten wird die Funktion der SPDL an einer Beispiel-Policy für den Dienst `ntpd`:

```
{
domain ntpd_t;
program /usr/sbin/ntpd;
include common-relaxed.sp;
include daemon.sp;
include nameservice.sp;

allowpriv cap_ipc_lock;
allowpriv cap_sys_time;
allowpriv cap_sys_resource;
allowpriv netlink;

allow /etc/ntp.conf r,s;
```

[3] *http://hitachisoft.jp/Products/secure-linux/*
[4] *http://seedit.sourceforge.net/doc/2.1/spdl_spec.pdf*

```
allow /etc/ntp/** r,s;
allow /var/lib/ntp/** r,w,s;

allownet -protocol udp,tcp -port 123 client,server;
allownet -protocol  udp -port 53 client;

allow etc_runtime_t r,s;
}
```

Eine File-Context-Datei ist nicht erforderlich. Diese wird aus der SPDL-Datei automatisch erzeugt. Schlüsselwörter definieren nun das Verhalten:

- `domain`: Dies definiert den Namen der Domäne.
- `program`: Dies definiert das Binary, das überwacht werden soll.
- `include`: Hiermit werden die Regeln aus weiteren Dateien eingelesen.
- `allowpriv`: Hiermit erlauben Sie den Zugriff auf eine Capability.
- `allow`: Hiermit erlauben Sie den Zugriff auf Dateien. Wildcards (*) sind erlaubt. Die Angabe von zwei Sternen erlaubt den Zugriff auf den gesamten Verzeichnisbaum.
- `allownet`: Hiermit erlauben Sie den Zugriff auf bestimmte Protokolle und Ports.

Die Policy in der SPDL-Sprache ähnelt einem AppArmor-Profil. Sie ist sofort von jedem UNIX-Administrator nachzuvollziehen und zu verstehen.

Für die automatische Erzeugung der benötigten Richtlinien können Sie das Werkzeug `audit2spdl` verwenden. Dieses arbeitet analog dem Befehl `audit2allow`.

Während die aktuellen Richtlinien noch keine Role-Based-Access-Control nutzen, kann die aktuelle Version 2.1 des Kommandos `seedit` bereits damit umgehen. Nach dem Aktivieren der RBAC-Unterstützung mit `seedit-rbac on` stehen drei Rollen zur Verfügung:

- *sysadm_r*
- *staff_r*
- *user_r*

Diese Rollen können auch einfach um weitere Rollen erweitert werden. Um zum Beispiel eine Rolle *webmaster_r* in der SPDL anzulegen, verwenden Sie den folgenden Befehl, der Ihnen zunächst die Grundregeln liefert:

```
[root@supergrobi ~]# seedit-template -r webmaster_r -u webmaster

role webmaster_r;
user webmaster;
include user_common.sp;
include common-relaxed.sp;
allow ~/** r,w,s;
```

```
allowpriv part_relabel;
allowpriv dac_override;
allowpriv dac_read_search;
```

Diese können Sie nun anpassen und zum Beispiel die folgenden Zeilen hinzufügen:

```
allow /etc/httpd/** r,w,s;
allow /var/www/* r,w,s;
```

30.3.2 SEedit-Installation

Die Installation von `seedit` ist unkritisch. Für Fedora existieren ab der Version 6 fertige Pakete. Auch für CentOS 5 sind Pakete über die Homepage verfügbar.

Achtung

Leider lassen sich die mit `seedit` erzeugten Richtlinien nicht mit den auf klassischem Wege erzeugten Richtlinien mischen. Sie müssen sich dann schon entscheiden, ob Sie die klassische Methode oder die SEedit-Methode wählen möchten.

30.4 Cross Domain Solutions Framework (CDS Framework)

Das *CDS Framework* Toolkit ist ein neues Werkzeug, das die konzeptionelle Erzeugung von Richtlinien erlaubt. Es stellt ähnlich SEedit eine Hochsprache zur Verfügung, in der die Eigenschaften einer Applikation oder Gruppe von Applikationen beschrieben werden. Diese Hochsprache kann anschließend in eine SELinux-Richtlinie übertragen werden. Somit erleichtert dieses Werkzeug die Erzeugung von SELinux-Richtlinien.

Das CDS Framework Toolkit besteht aus den folgenden Komponenten:

- Konzeptionelles Modell: Dieses Modell bietet dem Richtlinienautor nur wenige wichtige Sicherheitseigenschaften. Hierbei handelt es sich in erster Linie um den Informationsfluss.
- Hochsprache
- Integrated Design Environment(IDE): Eine Oberfläche, in der die Richtlinie entsprechend dem Modell beschrieben werden kann.
- Compiler: Dieser Compiler erzeugt aus der Hochsprache direkt die SELinux-Richtlinie.

Das CDS Framework benötigt das Eclipse Graphical Editing Framework. Dieses lässt sich bei den meisten Distributionen über ein Paket nachinstallieren.

30.4.1 CDS-Konzept

Das CDS verfolgt mit seinem Konzept zwei Ziele: Überwachung des Informationsflusses und Trennung der Informationsdomänen. Hierzu definiert es die folgenden Begriffe:

- `domain`
  Eine CDS-Domäne ist eine Informationsdomäne. Eine Domäne kann verschiedene Objekte, wie Prozesse, Dateien und Sockets, enthalten. Alle Prozesse innerhalb einer Domäne besitzen dieselben Privilegien, und alle Ressourcen genießen denselben Schutz. Prozesse innerhalb einer Domäne haben normalerweise unbeschränkten Zugriff auf die Ressourcen in dieser Domäne[5]. Im grafischen Editor werden Domänen als Kasten dargestellt.
- `shared resource`
  Eine gemeinsam genutzte Ressource ist ein Objekt, über das zwei Domänen Informationen austauschen können. Typisch ist dies für Dateien, Sockets und Named Pipes. Im grafischen Editor werden diese als Kreise dargestellt.
- `access`
  Der Zugriff definiert, wie eine Domäne auf eine Ressource zugreifen darf. Normalerweise hat jeder Prozess vollen Zugriff auf die Ressourcen in der eigenen Domäne. Das bedeutet, dass nur noch die Zugriffe auf gemeinsam mit anderen Domänen genutzte Ressourcen (`shared resource`) definiert werden müssen. CDS kennt drei Zugriffe, die mit Pfeilen symbolisiert werden: lesen (read), schreiben (write) und lesen-und-schreiben (readwrite).
- `decomposition`
  Dieses Modell würde nicht den Aufbau von komplexen Rechtesystemen erlauben. Daher ermöglicht es das CDS, eine Domäne in weitere Domänen (`children`) zu unterteilen und die Zugriffe innerhalb der Domäne zu verwalten. Auch die Subdomänen dürfen nur über Shared Resources kommunizieren. Auch Subdomänen können weiter unterteilt werden.

Das CDS Toolkit enthält einige grundsätzlich definierte Ressourcen und Domänen. Diese werden im grafischen Editor als Kreise und Kästen mit gestrichelter Linie dargestellt. Diese Domänen können nicht unterteilt werden.

Auch der Domänenwechsel kann grafisch dargestellt werden. Das CDS verwendet hier sowohl den `entrypoint` als auch `enter`. Ein Entrypoint ist eine Datei, deren Aufruf den Domänenwechsel auslöst. Dieser Entrypoint wird grafisch als Fünfeck dargestellt. Zusätzlich zeigt ein Pfeil von der ausgehenden Domäne zur Zieldomäne (siehe Abbildung 30.9).

Schließlich erlaubt das CDS noch die Definition von Steuerungseinrichtungen wie Inter-Process-Calls und Message-Queues. Diese werden als kleine Kreise dargestellt.

[5] Dies ist bei SELinux normalerweise nicht so!

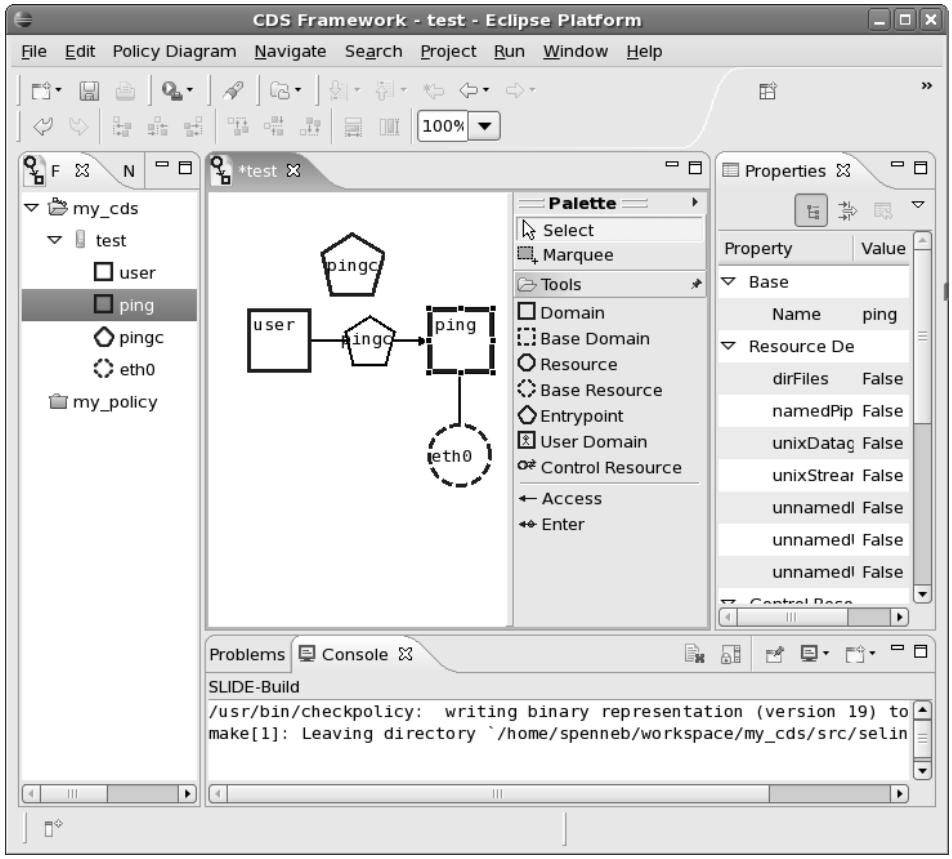

Abbildung 30.9: Eclipse erlaubt mit dem CDS Toolkit die Erzeugung abstrakter Richtlinien.

31 Die Example-Policy

Die Example-Policy basiert auf der originalen Policy, die von der National Security Agency (NSA) mit der originalen SELinux-Version veröffentlicht wurde. Heute wird diese Policy kaum noch eingesetzt und wurde weitestgehend durch die Reference-Policy ersetzt. Einer der wenigen Fälle, in denen man diese Policy noch auf Produktivsystemen finden wird, ist eine Red Hat Linux Enterprise 4-Installation. Aber auch hier ist sicherlich ein Wechsel auf eine moderne Variante mit Reference-Policy zu empfehlen. Bei Fedora Core wurde die Example-Policy nur in den Versionen 3 und 4 eingesetzt. Diese haben aber bereits das Ende ihres Lebenszyklus erreicht, sodass diese Versionen hoffentlich kaum noch auf Produktivsystemen zu finden sind.

Die Example-Policy ist über mehrere Jahre entstanden und verbessert worden. Eine der wesentlichen Verbesserungen war die Einführung der Trennung in Targeted- und Strict-Policy. Während die Strict-Policy tatsächlich direkt von der originalen NSA-Policy abgeleitet wurde, hat die Targeted-Policy nur noch gewisse Ähnlichkeiten.

31.1 Struktur der Example-Policy

Die *Example-Policy* liegt als Quelltext in einem Verzeichnis unterhalb des `/etc/selinux`Verzeichnisbaums. So befindet sich der Quelltext der Targeted-Policy in `/etc/selinux/targeted/src`. Hier befinden sich viele einzelne Dateien, die bei dem Bau der Policy zunächst alle konkateniert und anschließend übersetzt werden. Die Policy liegt also sowohl in einer ASCII- (`policy.conf`) als auch in einer binären Form (`policy.19`) vor. Für den Bau und die Verwaltung der Policy gibt es in dem Quelltextverzeichnis auch eine `Makefile`-Datei. Dieses Makefile kennt verschieden Make-Ziele:

- `make policy.conf`: Dies baut die binäre Policy als monolithische Datei.
- `make install`: Dies installiert die Policy und alle benötigten zusätzlichen Dateien.
- `make load`: Dies lädt die aktuelle binäre Policy-Datei.
- `make relabel`: Hiermit labeln Sie sämtliche Dateien auf dem System neu.

Die einzelnen Quelltextdateien sind in unterschiedlichen Verzeichnissen angeordnet. Die wichtigsten Verzeichnisse sind:

- `flask`: Hier befinden sich die Konfigurationsdateien, die die Objektklassen beschreiben. Diese Dateien müssen von Ihnen eigentlich nie modifiziert werden.

- `domains`: Hier befinden sich die TE-Dateien für die einzelnen Domänen. Für die Übersichtlichkeit wird jede Domäne in einer eigenen Datei verwaltet. Zusätzlich existieren die Unterverzeichnisse `program/` und `misc/` in denen weitere Quelltext-Dateien enthalten sind. Diese Unterverzeichnisse weisen eine Besonderheit auf. Beide enthalten zusätzlich ein weiteres Unterverzeichnis `unused/`. Sie können einzelne TE-Dateien in diese Unterverzeichnisse verschieben. Diese Dateien werden dann bei dem Bau der Policy nicht berücksichtigt. Sie können also auch bei der Example-Policy recht einfach einzelne Applikationen aus der Überwachung entfernen oder hinzufügen, in dem Sie deren TE-Dateien verschieben.
- `types`: Hier befinden sich alle weiteren Typ-Definitionen, die nicht in den TE-Dateien in `domains/` angegeben wurden. Hierbei handelt es sich in erster Linie um Typen für Netzwerkobjekte, Geräte etc.
- `file_contexts`: Für jede Domäne, deren TE-Datei in `domains/` auftaucht, werden auch File-Context-Definitionen benötigt. Diese werden in diesem Verzeichnis gespeichert. Auch hier gibt es wieder die Konstellation mit den Unterverzeichnissen `program/` und `misc/`.

31.2 Anpassung und Entwicklung von eigenen Regeln

Die wesentlichen Schritte, die in diesem Buch erwähnt worden sind, gelten auch für die Example-Policy. Auch beim Einsatz dieser Policy gibt es bereits den Befehl `audit2allow` der aus den Protokollmeldungen Regeln erzeugt. Diese müssen nun nicht an die Reference-Policy mit ihren Schnittstellen angepasst werden, da eh die gesamte Policy in einem Rutsch übersetzt wird. Hier sind daher keine Abhängigkeiten zu berücksichtigen.

Im Grunde erzeugen Sie ebenfalls eine TE-Datei und eine FC-Datei. Diese werden dann entsprechend in den Verzeichnissen `domains/program/` und `file_contexts/program/` gespeichert. Anschließend übersetzen und laden Sie die neue Policy mit `make load`.

32 SELinux Policy Management Server

Die Umstellung der SELinux-Policy auf die Reference-Policy hat auch die Policy modularisiert. Dabei wurde die Möglichkeit geschaffen, die Module aus unterschiedlichen Quellen zur Verfügung zu stellen. Der Befehl `semanage` kann verschiedene Modulspeicher verwalten. Die Konfiguration erfolgt in der Datei `/etc/selinux/semanage.conf`.

Eine Variante in dieser Datei ist der SELinux *Policy Management Server*. Dieser wurde von Tresys entwickelt und existiert in einer Alpha-Version. Leider war zu dem Zeitpunkt, als dieses Buch geschrieben wurde, die Entwicklung des Servers etwas eingeschlafen. Dennoch sollen der Server, seine Möglichkeiten und die damit verbundene zukünftige Vision für SELinux vorgestellt werden. Inwieweit diese Vision allerdings Wirklichkeit wird, vermag ich im Moment nicht zu sagen.

Die Entwicklung von SELinux hat in den vergangenen Jahren gezeigt, dass eine einfache, aber auch flexible Verwaltung der Policy erforderlich ist. Administratoren müssen dynamisch die Policy aktualisieren, erweitern und modifizieren können. Diese Vorgänge müssen sicher, unproblematisch und schnell erfolgen. Die modulare Reference-Policy versucht, dies zu ermöglichen. Die Module sind komplett eigenständige Policy-Bausteine, die recht einfach miteinander zu einer einzigen Policy für ein System verbunden werden können. Dazu ist keinerlei Modifikation der Policy-Quellen, keine explizite Übersetzung und damit auch keine Policy-Entwicklungsumgebung erforderlich. Dies erlaubt ein einfaches Deployment von neuen Modulen, Anpassungen und Aktualisierungen. Auf den Zielsystemen ist lediglich ein Linking der Module erforderlich. Die Erleichterung ist am ehesten mit der Einführung von Paketverwaltungssystemen wie RPM vergleichbar.

Dennoch fehlen für den umfassenden Einsatz von SELinux noch drei Funktionen:

- Bisher kann die Verwaltung der Policy nicht granular delegiert werden. Ein Administrator darf entweder die gesamte Policy modifizieren oder hat keinerlei Zugriff. Das Recht, nur ein Modul zu aktualisieren, kann nicht verwaltet werden.
- User-Space-Applikationen können noch nicht von SELinux profitieren. Bisher kann SELinux nur Aktionen entscheiden, die in dem Kernel stattfinden. Den Zugriff auf eine Spalte in einer MySQL-Datenbank kann SELinux noch nicht überwachen.

- Die Policy-Module sollten netzwerkweit identisch verwaltet werden. Es ist meist nicht sinnvoll, dass auf gleichartigen Systemen Module mit unterschiedlichen Versionen eingesetzt werden.

Der Policy Management Server soll diese Probleme lösen, indem er in seiner endgültigen Version die folgenden Ziele verfolgt:

- Granulare Zugriffsrechte bei der Verwaltung der Policy
- Unterstützung für User-Space-SELinux-Object-Manager
- Netzwerkunterstützung

Die aktuell zur Verfügung stehende Version des SELinux Policy Management Servers ist ein erster Versuch der Implementierung des ersten Ziels. Leider ist es nicht möglich, den Policy-Server auf modernen Implementierungen der Reference-Policy zu installieren. Bitte benutzen Sie für Ihren Test eine Fedora Core 3-Distribution, wenn Sie die Funktionen nachvollziehen wollen. Natürlich müssen Sie dieses System vorher auf die Reference-Policy umgestellt haben. Viele Befehle und Bibliotheken wurden inzwischen stark weiterentwickelt und sind nicht mehr kompatibel.

32.1 Installation

Die Installation des Policy Management Servers sollte noch nicht auf Produktionssystemen erfolgen. Der Policy Management Server ist noch im Alpha-Stadium und wird sich in zukünftigen Versionen wahrscheinlich (auch inkompatibel) verändern. Installieren Sie den Policy Management Server nur, wenn Sie sich mit den neuen Funktionen vertraut machen möchten. Verwenden Sie hierzu bitte ein Testsystem. Sie benötigen neue SELinux-Kommandos, Bibliotheken und einen gepatchten Kernel.

32.1.1 Anpassung der Policy

Der Policy Management Server benötigt eine neue angepasste Policy. Sie müssen einige Klassen hinzufügen. Diese neue Policy wird ohne Policy Management Server nicht mehr funktionieren. Daher sollten Sie zunächst eine Kopie Ihrer aktuellen Policy erzeugen. Dies ist erfreulicherweise bei SELinux sehr einfach:

```
[root@supergrobi ~]# cd /etc/selinux/
[root@supergrobi selinux]# cp -a targeted server
```

Nun sollte die neue Policy zur Default-Policy werden und der Rechner neu gebootet werden. Hierzu editieren Sie die Datei `/etc/selinux/config` und ändern die folgende Zeile:

```
SELINUXTYPE=server
```

Nun müssen Sie den Policy Management Server herunterladen. Die zum Zeitpunkt der Veröffentlichung dieses Buches aktuelle Version befindet sich auch auf der CD.

Die weitere Installation ist recht aufwendig und kann auf modernen SELinux-Varianten nicht mehr durchgeführt werden, da die modernen Policy-Compiler die verwendete Syntax nicht mehr unterstützen.

Wenn Sie den Policy-Server auf einer älteren Distribution installieren möchten, z.B. Fedora Core 3, dann finden Sie auf der Policy Management Server-Homepage ein Howto (*http://sepolicy-server.sourceforge.net*).

32.1.2 Anwendung des Policy Management Servers

Nach der Installation können Sie den Policy Management Server starten. Für den ersten Start empfiehlt es sich, dass Sie die Konfigurationsdatei `/etc/selinux/server/policy-server.conf` editieren und die folgenden Parameter anpassen:

```
fork-daemon = false
log-level = 4
```

Nun können Sie ihn mit dem folgenden Kommando starten:

```
[root@supergrobi ~]# policy-server -m
No configuration file found for libsemod.
Could not open /etc/selinux/targeted/modules/active/base.pp for ➥
         caching; metapolicy enforcement disabled.
Skipping network socket connection.
Policy server ready and waiting for connections.
```

Anschließend müssen Sie die Konfigurationsdatei `semod.conf` erzeugen und in `/etc/selinux/server/modules/` ablegen. Bei dieser Version des Policy-Servers lautet der Name noch `semod.conf`. Moderne Systeme nennen diese Datei heute `semanage.conf`. Dort editieren Sie die folgende Zeile:

```
module-store = /var/run/policy-server
```

Nun können Sie bereits die Verbindung zum Policy-Server testen:

```
[root@supergrobi ~]# semodule -l
No modules.
```

Dann können Sie die einzelnen Bestandteile der Policy labeln. Hierzu müssen Sie die Datei `genfs_contexts` im Policy-Source-Verzeichnis editieren und die folgenden Zeilen hinzufügen:

```
policycon type . system_u:object_r:policy_t
policycon role . system_u:object_r:policy_t
```

Dies labelt die Verwaltung aller Typen und aller Rollen mit dem Typ *policy_t*. Um zum Beispiel sämtliche Typen eines imaginären Servers auszunehmen und mit dem Typ *policy_server_t* zu versehen, würden Sie auch die folgende Zeile voranstellen:

```
policycon type server.  system_u:object_r:policy_apache_t
policycon role server.  system_u:object_r:policy_apache_t
```

Nun müssen Sie für den Policy Management Server ein binäres Base-Modul erzeugen. Hierzu wechseln Sie in das Policy-Source-Verzeichnis. Dort erzeugen Sie eine Datei `types/metapolicy.te` mit folgendem Inhalt:

```
type policy_t;
type policy_server_t;
```

Dann rufen Sie den Befehl `make policy.conf` auf und bauen das Modul:

```
[root@supergrobi policy]# make policy.conf
mkdir -p tmp
m4 -Imacros -s flask/security_classes flask/initial_sids flask/ ↵
         access_vectors tunables/distro.tun tunables/tunable.tun ↵
         attrib.te tmp/program_used_flags.te macros/program/ ↵
         apache_macros.te
...
mv policy.conf.tmp policy.conf
[root@supergrobi policy]# checkmodule policy.conf -o policy.bmod
checkmodule: loading policy configuration from policy.conf
checkmodule: policy configuration loaded
checkmodule: writing binary representation (version 3) to policy.bmod
[root@supergrobi policy]# make file_contexts
make: Nothing to be done for 'file_contexts'.
[root@supergrobi policy]# sepackagemodule policy.bpp policy.bmod ↵
         file_contexts/file_contexts
```

Dieses Modul können Sie nun in dem Policy Management Server installieren:

```
[root@supergrobi policy]# semodule -v -b policy.bpp
Attempting to install base module 'policy.bpp':
Ok.
```

32.1.3 Erlauben der Policy-Verwaltung

Der Policy Management Server überwacht nun, wer die Policy verändern darf. Wenn Sie jetzt ein Test-Modul schreiben und übersetzen, werden Sie dieses Modul im Enforcing-Modus nicht installieren dürfen. Erzeugen Sie das folgende Modul:

```
module server 1.0.0;

require {
   class file { read write};
}

type server_t;
type server_test_t;
role server_r types { server_t server_test_t };

allow server_t server_test_t : file { read write };
```

Der Versuch, das Modul zu installieren, wird fehlschlagen:

```
[root@supergrobi tmp]# setenforce 1
[root@supergrobi tmp]# semodule -v -i server.pp Attempting to ➥
         install module 'server.pp':
Failed.
```

Nun werden wir ein zweites Modul bauen, das es uns erlaubt, das erste Modul zu installieren.

```
module allow_server 1.0.0;

require {
  type policy_server_t, unconfined_t;
  class policy.role { add remove use.src };
  class policy.type { add remove use.src.allow use.tgt.allow
                      associate.role};
}

allow unconfined_t policy_server_t : policy.role { add remove ➥
         use.src };
allow unconfined_t policy_server_t : policy.type { add use.src. ➥
         allow
                                              use.tgt.allow
                                              associate.role };
```

Dieses Modul muss zuvor im Permissive-Modus installiert werden:

```
[root@supergrobi tmp]# setenforce 0
[root@supergrobi tmp]# semodule -v -i allow_server.pp
Attempting to install module 'allow_server.pp':
Ok.
```

Anschließend kann das System wieder in den Enforcing-Modus geschaltet werden. Nun überwacht der SELinux Policy Management Server wieder die Verwaltung der Module. Dennoch kann jetzt auch das Modul `server.pp` installiert werden:

```
[root@supergrobi tmp]# setenforce 1
[root@supergrobi tmp]# semodule -v -i server.pp
Attempting to install module 'server.pp':
Ok.
[root@foo tmp]# semodule -l
allow_server 1.0.0
server 1.0.0
```

32.2 Fazit

Wenn der Policy Management Server weiterentwickelt und von modernen Distributionen unterstützt wird, ist die zentrale Verwaltung der SELinux-Richtlinien für mehrere Systeme gleichzeitig in greifbarer Nähe. Damit wird der Einsatz auch in größeren Netzwerken interessant. Speziell die Möglichkeit, die Verwaltung einzelner Module an unterschiedliche Administratoren zu delegieren, eröffnet neue Perspektiven in der Verwaltung von Systemen. Auch wird es mit dem Policy Management Server möglich sein, bei der Installation eines RPM-Pakets zu entscheiden, welche Policy-Module das RPM-Paket austauschen darf. So kann ein nicht vertrauenswürdiges RPM-Paket nicht einfach die Richtlinie ändern und den eigenen Applikationen Schreibrechte an der Datei `/etc/shadow` einräumen.

Die Unterstützung von User-Space-Object-Managern wird dann vielleicht auch die Zugriffsverwaltung für Dienste ermöglichen, die im Moment noch nicht über ein Mandatory-Access-Control-System verfügen, wie zum Beispiel SQL-Datenbanken.

Noch ist das jedoch Zukunftsmusik. Leider hat sich seit der Vorstellung des SELinux Policy Management Daemons im April 2005 nichts mehr an der Entwicklung getan.

33 SELinux-erweitertes XWindow

SELinux bietet über Flask auch die Möglichkeit, externe Object-Manager anzubinden, die für die Entscheidung, ob der Zugriff erlaubt ist, den Security-Server im Linux-Kernel fragen (siehe Kapitel 11.2).

Eine Applikation, die das nutzen könnte, ist der X-Server. Es gab in der Vergangenheit bereits mehrere Versuche, den X-Server in die Lage zu versetzen, diese Anfragen an den Security-Server weiterzuleiten. Dies hat mehrere Gründe. Im Moment bietet der X-Server wenig Möglichkeiten, die Sicherheit der verschiedenen X-Clients zu garantieren.

- Vertraulichkeit
  Viele Applikationen nutzen als X-Client gleichzeitig einen X-Server. Es ist für eine Applikation sehr leicht, die Vertraulichkeit der weiteren gleichzeitig laufenden Applikationen zu brechen. Ein Screenshot ist sicherlich die am einfachsten nachzuvollziehende Methode, um die Ausgaben eines weiteren X-Clients zu stehlen. Auch die Zwischenablage hat auf die Daten aller Applikationen Zugriff und kann genutzt werden, um Daten einer Applikation einer weiteren Applikation zur Verfügung zu stellen. Bösartige Applikationen können diese Funktionen für sich nutzen.
- Integrität
  Der X-Server schützt nicht die Datenintegrität. Ein X-Client kann die Ausgabe eines weiteren X-Clients direkt verändern. Bösartige Clients können auch die Eingaben weiterer X-Clients verändern und zusätzliche Daten injizieren.
- Verfügbarkeit
  Der X-Server schützt auch seine eigene Verfügbarkeit unzureichend. Einzelne X-Clients können die Fenster weiterer X-Clients schließen und die Zeichensätze und die Zugriffskontrolllisten des X-Servers manipulieren (`xhost(1x)`).

Bereits 2003 wurden diese Probleme von Doug Kilpatrick et al. [17] erkannt und wurden erste Modifikationen an dem X-Server und SELinux vorgenommen. Diese Modifikationen waren aber lange Zeit nur als Patch verfügbar und wurden von den Distributionen nicht genutzt.

Mit dem X.org X-Server X11R7 wurde im Februar 2007 erstmalig das XACE-Framework (X Access Control Extension) aufgenommen. Dies wurde von Eamon Walsh auf dem SELinux Symposium (siehe Kapitel 34) vorgestellt [18].

XACE ist ein Satz von »hooks«, der von anderen X-Erweiterungen genutzt werden kann, um über den Zugriff zu entscheiden. Das Ziel von XACE ist der geordnete Zugriff auf die Funktionen und ähnelt damit dem LSM-Ansatz des Linux-Kernels.

XACE stellt damit eine Verallgemeinerung der bereits existenten Security-Erweiterung dar, die bisher nur ein sehr einfaches Modell unterstützte: X-Clients konnten vertrauenswürdig sein oder nicht. Nicht vertrauenswürdige X-Clients wurden in bestimmten Bereichen beschränkt. Die einzige Applikation, die diese bisher vorhandene Funktion tatsächlich genutzt hat, ist die Secure-Shell, die zwei verschiedene Arten des X-Forwarding kennt (`-X` und `-Y`). XACE ersetzt zum größten Teil die vorhandenen sicherheitsspezifischen Prüfungen mit Callback-Funktionen. Diese können dann von zusätzlichen Modulen genutzt werden. Hierzu gehört dann auch SELinux.

Auf der Roadmap für die Version X11R7.3 des X.org X-Servers ist die Unterstützung von SELinux aufgeführt. Diese Version wurde im August 2007 erwartet und wurde bisher noch nicht veröffentlicht.

34 SELinux-Symposium

SELinux ist ein Konzept, das sich schnell entwickelt befindet. Das ist und war auch ein Problem bei der Erstellung dieses Buches. Wenn Sie bezüglich der Entwicklung auf dem neuesten Stand bleiben möchten, sollten Sie sich die jährlichen SELinux Symposien ansehen. Diese finden seit 2004 statt.

Natürlich ist es nicht für jeden möglich, tatsächlich an einer dieser Konferenzen teilzunehmen. Auch ich habe es bisher nicht geschafft. Aber die Webseite des SELinux Symposium (*http://selinux-symposium.org*) hält die Vortragsfolien und, wenn verfügbar, auch die dazugehörigen Artikel der Sprecher zum Download vor. Hier können Sie sich über die neuesten Entwicklungen informieren.

Im März 2007 fand das Symposium in Baltimore statt. Im Folgenden finden Sie das Programm der zwei Tage:

- *An Example of a Better Path to Information Assurance through Partnership* Richard Schaeffer, Director, NSA
- *What's New with SELinux* Stephen D. Smalley, NSA
- *Security-Enhanced Darwin: Porting SELinux to Mac OS X* Chris Vance, Todd Miller, and Rob Dekelbaum, SPARTA, Inc., USA
- *Extending SELinux policy management to networked policy domains* Joshua Brindle, Karen Vance, and Chad Sellers, Tresys
- *Using the Flask Security Architecture to Facilitate Risk Adaptable Access Controls* Machon Gregory and Peter Losocco, NSA
- *Using GConf as an Example of How to Create an Userspace Object Manager* James Carter, NSA
- *Integrating X.Org with Security-Enhanced Linux* Eamon Walsh, NSA
- *FCGlob: A new SELinux file context syntax* Donald Miner, UMBC and James Athey, Tresys
- *Reference Policy* Chris PeBenito, Tresys
- *SELinux Upstream Future Directions* Karl MacMillan, Red Hat
- *Targeted Policy* Dan Walsh, Red Hat
- *SLIDE: The SELinux Policy IDE* Brian Williams, Tresys
- *Extending SELinux Policy Model and Enforcement towards Trusted Computing Paradigms* Xinwen Zhang, Samsung
- *SETools* Chris PeBenito, Tresys

- *Security Enhanced PostgreSQL* KaiGai Kohel, NEC
- *CLIP - a Certifiable Linux Integration Platform* Art Wilson, Tresys
- *A Lot Can Happen in a Year: CIPSO, NetLabel, and Linux* Paul Moore, HP
- *Trusted Cups* Matt Anderson, HP
- *Open and Secure: Linux Today and Tommorrow* Dr. Daniel Frye, IBM Open Systems Development
- *Implementation of SELinux for Embedded Linux Environments* Hadi Nahari, Montavista
- *SELinux activities for embedded area, in Japan* KaiGai Kohel, NEC
- *CDSFramework* Brian Williams, Tresys
- *Connecting SCADA and corporate IT networks using SELinux* Ryan Bradetich, University of Idaho
- *Enhancing the Security of Enterprise Products with SELinux* Spencer Shimko, Tresys
- *Towards Intuitive Tools for Managing SELinux: Hiding the Details but Retaining the Power* James Athey, Chris Ashworth, Donald Miner, and Frank Mayer, Tresys
- *Madison: A New Approach to Automated Policy Generation* Karl MacMillan, Red Hat
- *Setroubleshoot: A User Friendly Tool to Diagnose AVC Denials* John Dennis, Red Hat
- *The Design and Implementation of a Guard Installation and Administration Framework* Boyd Fetcher and Christopher Roberts, United States Joint Forces Command
- *Securing Inter-process Communications in SELinux* Spencer Shimko and Joshua Brindle, Tresys
- *Integrating SELinux with Security-typed Languages* Boniface Hicks, Sandra Rueda, Trent Jaeger, and Patrick McDaniel, Pennsylvania State University
- *Case Study: Trusted Computer Solution development and deployment of SELinux based Solutions* George Kamis, Trusted Computer Solutions
- *Multilevel Secure Applications to Security Enhanced Linux* Andy Suchoski and Rick Supplee, HP
- *Extending Linux for Multi-Level Security* Klaus Weidner, atsec, George Wilson, IBM, and Loulwa Salem, IBM

Diese Aufstellung sollte Ihnen zeigen, dass SELinux auch Bereiche erreicht, die bisher nicht mit SELinux in Zusammenhang gebracht werden, wie PostgreSQL, und dass auch immer neue Ansätze für die Pflege und die Entwicklung von SELinux-Richtlinien gesucht werden. SELinux ist sicherlich ein Konzept, das sich in der Zukunft noch weiterentwickeln wird.

A Auditd-Daemon

Das Audit-System wurde von Rik Faith als Low-overhead-System-Call-Audit-Framework erschaffen. Es ist fester Bestandteil des Linux-Kernels 2.6 geworden und wird in allen modernen Distributionen eingesetzt. Neben den Funktionen im Linux-Kernel werden einige Userspace-Werkzeuge benötigt, die von den entsprechenden Distributionen als Paket mitgeliefert werden. Die Homepage des Audit-Frameworks ist: *http://people.redhat.com/sgrubb/audit/*.

Ältere Audit-Subsysteme wie *Laus* von Olaf Kirch und Thomas Biege wurden durch dieses System abgelöst. Laus ist noch Bestandteil älterer Distributionen wie SLES 9 und RHEL 3. Audit-Subsysteme werden für die Zertifizierung der Betriebssysteme nach den *Common-Criteria* benötigt und bieten auch für den normalen Anwender die Möglichkeit, viele Ereignisse eines Betriebssystems genau zu protokollieren.

Das Audit-Subsystem erlaubt die Protokollierung der folgenden Ereignisse auf einem Linux-System:

- Aufruf und Beendigung von System-Calls (Entry und Exit)
- Erzeugung neuer Prozesse (Tasks)
- Dateioperationen (Watches)

Dabei übernimmt der Auditd-Daemon die Protokollierung der Ereignisse, während die Konfiguration mit dem Befehl `auditctl` erfolgt. Die Befehle `aureport ausearch` und `autrace` erlauben die Analyse und weitere Verarbeitung der Protokolle.

A.1 Zertifizierungen nach Common Criteria

Für den Einsatz in bestimmten Umgebungen ist die Zertifizierung nach den Common Criteria for Information Technology Security Evaluation erforderlich. Die *Common-Criteria* (CC) sind ein internationaler Standard (ISO/IEC 15408) für Computersicherheit. Die CC definieren sieben Stufen der Vertrauenswürdigkeit (Evaluation Assurance Level, EAL 1-7). Diese beschreiben die Korrektheit der Implementierung des betrachteten Systems. Höhere EAL-Stufen implizieren nicht automatisch eine bessere Sicherheit, sondern lediglich die Tatsache, dass das betrachtete Produkt (Target of Evaluation, TOE) genauer analysiert wurde. Die Analyse erfolgt entsprechend einem Schutzprofil (Protection Profile, PP), aus dem ein Sicherheitsziel (Security Target) für das entsprechende Produkt abgeleitet wird.

Typische Schutzprofile sind:

- CAPP
  Das Controlled Access Protection Profile (*CAPP*) verlangt die Implementierung eines Discretionary-Access-Control-(*DAC*-)Systems in dem analysierten Betriebssystem.
- LSPP
  Das Labeled Security Protection Profile (*LSPP*) verlangt den Einsatz eines Mandatory-Access-Control-(*MAC*-)Systems wie SELinux. AppArmor kann den Ansprüchen aktuell nicht genügen.

Bisherige Zertifizierungen gelangen den folgenden Distributionen:

Produkt	Kernel	PP	EAL
SLES 8	2.4	ST	2+
SLES 8 SP3	2.4	CAPP	3+
RHEL3 UP2	2.4	CAPP	3+
SLES 9	2.6	CAPP	4+
RHEL4 UP1	2.6	CAPP	4+
RHEL5	2.6	LSPP	4+

Für die einfachere Konfiguration werden zwei Konfigurationsdateien genutzt:

- `/etc/audit/auditd.conf`
  Diese Datei enthält Konfigurationsparameter für den Auditd-Daemon. Diese Datei wird daher in dem entsprechenden Abschnitt A.2 genauer betrachtet.
- `/etc/audit/audit.rules`
  Diese Datei enthält Regeln, die automatisch bei dem Start des Auditd-Daemons mit dem Befehl `auditctl` eingelesen werden. Daher wird diese Datei gemeinsam mit dem Befehl behandelt (siehe Abschnitt A.3).

A.2 auditd

Der `auditd` nimmt die Meldungen des Audit-Subsystems des Kernels entgegen und protokolliert sie in einer Protokolldatei. Hierbei können die folgenden Optionen angegeben werden:

- `-f`: Der Auditd-Daemon verbleibt im Vordergrund, und sämtliche Meldungen erfolgen auf der Standardfehlerausgabe anstatt in der Protokolldatei.
- `-l`: Der Auditd-Daemon erlaubt die Verwendung von symbolischen Verknüpfungen beim Öffnen der Konfigurationsdateien.
- `-n`: Der Auditd-Daemon forkt bei seinem Start nicht in den Hintergrund. Dadurch ist es möglich, den Auditd-Daemon direkt über die Datei `/etc/inittab` zu starten. Dies nutzt jedoch keine Distribution.

Während seiner Funktion kann der Auditd-Daemon über Signale gesteuert werden. Drei Signale können genutzt werden:

- `SIGHUP`
  Wie die meisten Dienste liest auch der Auditd-Daemon bei diesem Signal seine Konfigurationsdatei neu ein.
- `SIGTERM`
  Der Auditd-Daemon beendet sich.
- `SIGUSR1`
  Der Auditd-Daemon rotiert seine Protokolldateien. Die Rotation der Protokolle sollte immer direkt über den Auditd-Daemon und nicht über externe Protokolle erfolgen, um einen Verlust von Nachrichten zu verhindern. Externe Protokolle beenden häufig zunächst einen Dienst, rotieren die Protokolle und starten den Dienst dann neu.

Die Konfigurationsdatei `/etc/audit/auditd.conf` enthält weitere Parameter, über die Sie das Verhalten des Dienstes kontrollieren können. Während einige davon das unmittelbare Verhalten beeinflussen, definieren etwa die Hälfte der Parameter die Reaktionen in einem Fehlerfall.

- `log_file`: Dieser Parameter bestimmt den absoluten Pfad der Protokolldatei.
- `log_format`: Hier stehen zwei Werte zur Verfügung: `RAW` protokolliert die Daten, während `NOLOG` keine Protokollierung in der Protokolldatei durchführt. Die Daten werden aber dennoch an den Dispatcher weitergegeben.
- `priority_boost`: Dies ist der negative Nice-Wert für den Start des Auditd-Daemons. Bei einem Wert von 3 (Default) wird der `auditd` mit einem Nice-Wert von -3 gestartet.
- `flush`: Dieser Parameter beschreibt, wie und wann die Daten auf die Festplatte geschrieben werden. Mit `none` erfolgt keine spezielle Kontrolle. Bei `incremental` wird entsprechend dem `freq`-Parameter ein Flush ausgelöst. Eine Synchronisation erfolgt bei `data` und `sync`. Im ersten Fall werden lediglich die Nutzdaten synchronisiert, während im letzteren Fall auch die Meta-Daten synchronisiert werden. Diese letzten beiden Varianten stellen die sinnvollen Einstellungen dar.
- `freq`: Dieser Parameter beschreibt, nach wie vielen Einträgen ein Flush auf die Festplatte erfolgen muss. Dieser Parameter ist nur gültig, wenn gleichzeitig der Parameter `flush` den Wert `incremental` hat.
- `num_logs`: Dieser Parameter bestimmt, wie viele Versionen der Protokolldatei bei einer Rotation maximal entstehen dürfen. Die maximale Anzahl ist 99. Da die Rotation mit zunehmender Zahl der Protokolldateien länger benötigt, ist eine Erhöhung des Backlogs im Kernel ratsam (siehe Abschnitt A.3).
- `dispatcher`: Der Auditd startet dieses Programm und übergibt eine Kopie sämtlicher Meldungen. Hiermit können auch weitere Programme die Meldungen in Echtzeit erhalten.
- `disp_qos`: Hiermit kann die Kommunikation zwischen dem Auditd und dem Dispatcher konfiguriert werden. Für diese Kommunikation existiert ein 128-k-Puffer,

der für die meisten Zwecke ausreicht. Ist dieser Puffer jedoch voll, bestimmt dieser Parameter, wie der Auditd verfahren soll. Bei `lossy` (Default) werden neue Meldungen in das Protokoll geschrieben und nicht an den Dispatcher übermittelt. Mit `lossless` wartet der Auditd, bis der Dispatcher die Meldung entgegennehmen kann. Dadurch besteht die Gefahr, dass weitere Meldungen nicht auf die Festplatte geschrieben werden.

- `max_log_file`: Dieser Parameter bestimmt die maximale Größe der Protokolldatei in Megabytes.
- `max_log_file_action`: Hiermit bestimmen Sie, wie der Auditd bei Erreichen der maximalen Protokollgröße reagieren soll:
  - `ignore`: Keinerlei Maßnahme
  - `syslog`: Warnung über den Syslogd
  - `suspend`: Einstellen der Protokollierung
  - `rotate`: Rotation der Protokolle und automatisches Löschen alter Protokolle entsprechend dem Parameter `num_logs`
  - `keep_logs`: Rotation der Protokolle unter Bewahrung alter Protokolldateien
- `action_mail_acct`: Eine gültige E-Mail-Adresse für die Benachrichtigung
- `space_left`: Mindestgröße des freien Speicherplatzes in Megabytes. Diese Angabe bezieht sich auf die Partition, auf der sich auch die Auditd-Protokolldateien befinden. Dieser Wert sollte so eingestellt werden, dass Sie ausreichend Zeit haben, die Protokolle zu analysieren, alte Protokolle zu sichern und zu löschen.
- `space_left_action`: Hiermit bestimmen Sie die Reaktion beim Unterschreiten des freien Speicherplatzes:
  - `ignore`: siehe oben
  - `syslog`: siehe oben
  - `email`: Versand einer E-Mail und Protokollierung per Syslogd. Dies ist die sinnvollste Einstellung bei mehreren Systemen.
  - `exec`: Aufruf des angegebenen Scripts
  - `suspend`: siehe oben
  - `single`: Wechsel des Systems in den Single-User-Modus
  - `halt`: Anhalten des Systems
- `admin_space_left`: Absolute Mindestgröße des freien Speicherplatzes. Dieser Wert sollte kleiner sein als `space_left`. Auf einem System, bei dem Sie zwingend einen Audit-Trail benötigen, sollte dieser Wert mindestens so groß gewählt werden, dass sämtliche für die Wartung erforderlichen Aktionen protokolliert werden können. Die Aktion sollte dann auf `single` eingestellt werden, damit auch nur noch diese Aktionen protokolliert werden.
- `admin_space_left_action`: siehe oben
- `disk_full_action`: Hiermit definieren Sie die Reaktion des Auditd, wenn der Speicherplatz auf der Festplatte erschöpft ist. Sinnvoll sind hier nur `single` oder `halt`.

- `disk_error_action`: Diese Aktion wird bei einem Fehler während der Protokollierung auf die Festplatte ausgelöst.

A.3 auditctl

Mit dem Befehl `auditctl` konfigurieren Sie das Audit-Subsystem im Linux-Kernel. Dabei können Sie mit dem Befehl sowohl allgemeine Einstellungen definieren und ändern als auch in Regeln die Ressourcen definieren, die von dem Audit-Subsystem überwacht werden sollen.

Zu den allgemeinen Einstellungen gehören die folgenden:

- Das Backlog im Kernel nimmt die Meldungen auf, die von dem Auditd noch nicht gelesen worden sind. Der Default-Wert beträgt 64. Sie können mit dem Befehl `auditctl -b <backlog>` diesen Wert anpassen.
- Auch die Aktivierung und Deaktivierung des Audit-Systems ist möglich. Mit `auditctl -e 0` deaktivieren Sie jegliches Auditing. Mit `auditctl -e 1` schalten Sie es wieder ein. Eine Sonderstellung nimmt `auditctl -e 2` ein. Hiermit versiegeln Sie das Audit-Subsystem so, dass keinerlei Änderungen bis zu einem Reboot möglich sind. Jeder Versuch wird protokolliert und abgelehnt.
- Schließlich können Sie noch die Reaktion des Kernels beim Auftreten eines Fehlers einstellen. Hierbei kann es sich zum Beispiel um ein volles Backlog handeln. Es gibt drei verschiedene Werte, die Sie bei dem Befehl `auditctl -f [0..2]` übergeben können. Bei Angabe von 2 wird eine Kernel-Panic ausgelöst. Mit 1 erfolgt eine Protokollierung über `printk` während eine 0 keinerlei Reaktion auslöst.
- Die Rate der Meldungen kann mit der Option `-r` beschränkt werden. Sie geben hier die Anzahl der Meldungen je Sekunde an. Eine Überschreitung führt zu einem Fehler (siehe `-f`). Bei Angabe von 0 (Default) ist diese Funktion deaktiviert.
- Um selbst Meldungen zu erzeugen, die von dem Audit-System protokolliert werden, können Sie den Befehl `auditctl -m text` verwenden.
- Mit dem Befehl `auditctl -s` können Sie den aktuellen Status des Subsystems anzeigen:

```
[root@supergrobi ~]# auditctl -s
AUDIT_STATUS: enabled=1 flag=1 pid=2578 rate_limit=0 ➥
          backlog_limit=320 lost=0 backlog=0
```

Die weiteren Parameter dienen zur Konfiguration der zu überwachenden Ressourcen. Hierzu können Sie mit dem Befehl `auditctl` Regeln definieren. Mit `auditctl -l` zeigen Sie sämtliche Regeln an.

Um lediglich den Zugriff auf eine Datei zu überwachen, können Sie eine *Watch* verwenden. Sie erzeugen eine Watch mit der Option `-w` und löschen sie wieder mit `-W`. Um zum Beispiel den Zugriff auf die Datei `/etc/passwd` zu überwachen, nutzen Sie:

```
[root@supergrobi ~]# auditctl -w /etc/passwd
[root@supergrobi ~]# auditctl -l
LIST_RULES: exit,always watch=/etc/passwd perm=rwxa
```

Erfolgt nun ein Zugriff, wird er vom Audit-Daemon protokolliert:

```
type=SYSCALL msg=audit(1177929808.093:81): arch=40000003 syscall=5 ⬅
          success=yes exit=3 a0=bffe0872 a1=8000 a2=0 a3=8000 ⬅
          items=1 ppid=3969 pid=15352 auid=500 uid=500 gid=500 ⬅
          euid=500 suid=500 fsuid=500 egid=500 sgid=500 fsgid=500 ⬅
          tty=pts3 comm="cat" exe="/bin/cat" subj=user_u:system_r: ⬅
          unconfined_t:s0 key=(null)
type=CWD msg=audit(1177929808.093:81):  cwd="/buch/selinux"
type=PATH msg=audit(1177929808.093:81): item=0 name="/etc/passwd" ⬅
          inode=2158841 dev=fd:00 mode=0100644 ouid=0 ogid=0 ⬅
          rdev=00:00 obj=system_u:object_r:etc_t:s0
```

Alle drei Meldungen gehören zusammen. Dies können Sie an dem Parameter `msg` erkennen. Dieser enthält neben dem Zeitpunkt, wann die Meldung ausgelöst wurde, auch die Seriennummer (Event-ID) des Ereignisses. Diese ist in allen drei Meldungen gleich (81). Daher gehören diese Meldungen alle zu demselben Ereignis[1]. Der Zeitpunkt wird in Millisekunden seit dem 1.1.1970 angegeben (UNIX-Epoche). Um diese Zahl in ein für Menschen lesbares Datum umzuwandeln, können Sie das folgende Perl-Script verwenden:

```
$ perl -e "print scalar(localtime(1177929808))"
Mon Apr 30 12:43:28 2007
```

Eine weitere einfache Variante ist:

```
$ date -d @1177929808
```

Die Angabe der Millisekunden kann unterbleiben, da die Ausgabe in Sekunden erfolgt.

Jeder Zugriff auf die Datei `/etc/passwd` wird nun von dem Auditd protokolliert. Um diese Überwachung wieder abzuschalten, verwenden Sie `auditctl -W /etc/passwd`.

Zusätzlich können Sie auch noch komplexere Regeln in fünf verschiedenen Listen definieren:

- `task`: Diese Regeln werden jedes Mal geprüft, wenn ein neuer Prozess mit `fork` oder `clone` erzeugt wird.
- `entry`: Diese Regeln werden bei jedem Aufruf eines System-Calls geprüft.
- `exit`: Diese Regeln werden bei jeder Beendigung eines System-Calls geprüft.
- `user`: Hiermit können Sie Meldungen aus dem User-Space filtern.
- `exclude`: Diese Liste erlaubt Ihnen die systemweite Definition von Ausnahmen, die nicht protokolliert werden sollen.

Bei jeder Regel müssen Sie auch definieren, was mit den durch die Regel definierten Meldungen geschehen soll:

[1] Achtung: Der Auditd beginnt bei jedem Neustart wieder mit eins.

- `never`: Dieser Parameter erzeugt keine Protokollierung.
- `always`: Dieser Parameter löst eine Protokollierung aus.

Die Reihenfolge der Regeln in den Listen ist wichtig, da diese Listen entsprechend abgearbeitet und nach der ersten zutreffenden Regel alle weiteren ignoriert werden.

Ein Beispiel:

```
auditctl -a exit,always -S open -F auid>500 euid=0
```

Mit diesem Befehl fügen Sie eine Regel am Ende der Liste `exit` ein. Diese Regel überwacht alle `open`-System-Calls. Protokolliert werden die System-Calls, bei denen der Benutzer sich nicht als *root* angemeldet hat (`auid>500`) und der Zugriff jetzt doch als *root* erfolgt (`euid=0`). Dies kann zum Beispiel nach einem `su`-Kommando der Fall sein.

Die Auflistung der einzelnen Regelfelder würde hier jedoch zu weit führen und lediglich die Manpage wiedergeben. Das Audit-Paket enthält auch eine Reihe von Beispieldateien, wie die `sample.rules`-, `capp.rules`- und `lspp.rules`-Dateien, die Ihnen weitere Anregungen geben werden.

In der Realität werden Sie auch Ihre Regeln immer in derartigen Dateien anlegen und diese Dateien mit dem Befehl `auditctl -R` einlesen. Syntaxfehler können Sie beim Einlesen dieser Dateien übrigens mit der Option `-i` ignorieren.

A.4 aureport

Dieses Werkzeug erzeugt aus den Protokollen fertige Berichte. Geben Sie nur den Befehl `aureport` an, so erhalten Sie eine komplette Zusammenfassung:

```
[root@supergrobi ~]# aureport

Summary Report
======================
Range of time in logs: 27.10.2006 16:37:04.549 - ⏎
         30.01.2007 13:21:17.605
Selected time for report: 27.10.2006 16:37:04 - ⏎
         30.01.2007 13:21:17.605
Number of changes in configuration: 507
Number of changes to accounts, groups, or roles: 102
Number of logins: 310
Number of failed logins: 41
Number of authentications: 944
Number of failed authentications: 80
Number of users: 3
Number of terminals: 48
Number of host names: 16
Number of executables: 50
```

```
Number of files: 33
Number of AVC denials: 987
Number of MAC events: 511
Number of failed syscalls: 584
Number of anomaly events: 356
Number of responses to anomaly events: 0
Number of crypto events: 0
Number of process IDs: 4802
Number of events: 25232
```

Einzelne Informationen können Sie in detaillierteren Berichten zusammenfassen lassen:

```
[root@supergrobi ~]# aureport -f --summary

File Summary Report
===========================
total  file
===========================
278  /sbin/ldconfig
56  /home/spenneb/.mozilla/plugins/libflashplayer.so
43  /etc/auto.misc
40  /usr/lib/tls/libnvidia-tls.so.1.0.9625
...
```

Auch die Ausgabe eines detaillierten Berichtes ist möglich. Hierzu unterlassen Sie die Angabe von `-summary`:

```
[root@supergrobi ~]# aureport -f

File Report
===============================================
# date time file syscall success exe auid event
===============================================
1. 29.10.2006 12:35:37 /home/spenneb/.mozilla/plugins/libflash- ↵
         player.so 125 no /usr/lib/firefox-1.5.0.7/firefox-bin 500 64
2. 29.10.2006 12:35:37 /home/spenneb/.mozilla/plugins/libflash- ↵
         player.so 125 no /usr/lib/firefox-1.5.0.7/firefox-bin 500 65
3. 29.10.2006 12:35:41 /home/spenneb/.mozilla/plugins/libflash- ↵
         player.so 125 no /usr/lib/firefox-1.5.0.7/firefox-bin 500 66
4. 29.10.2006 12:35:41 /home/spenneb/.mozilla/plugins/libflash- ↵
         player.so 125 no /usr/lib/firefox-1.5.0.7/firefox-bin 500 67
...
```

Um weitere Informationen über das Ereignis zu erhalten, benötigen Sie nun den Befehl `ausearch`.

Achtung

Leider funktionieren diese Befehle nicht mit den Meldungen des AppArmor-Systems. AppArmor verwendet eine Syntax, die von `aureport`, `ausearch` und `autrace` nicht verstanden wird.

A.5 ausearch

Mit dem Befehl `ausearch` suchen Sie in den Audit-Protokollen nach bestimmten Ereignissen. Ist Ihnen zum Beispiel in den Berichten von `aureport` ein Ereignis aufgefallen, zu dem Sie weitere Details benötigen, können Sie diese mit `ausearch` suchen:

```
[root@supergrobi ~]#  ausearch -a 262 --end 08.11.2006 ↵
         --start 08.11.2006
----
time->Wed Nov  8 08:37:03 2006
type=SYSCALL msg=audit(1162971423.157:262): arch=40000003 ↵
         syscall=102 success=no exit=-13 a0=3 a1=bfdcb5b0 ↵
         a2=bd31e8 a3=91d8ae8 items=0 ppid=1 pid=4096 auid=500 ↵
         uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 ↵
         tty=(none) comm="httpd" exe="/usr/sbin/httpd" ↵
         subj=user_u:system_r:httpd_t:s0 key=(null)
type=AVC msg=audit(1162971423.157:262): avc:  denied ↵
         { name_connect } for  pid=4096 comm="httpd" dest=80 ↵
         scontext=user_u:system_r:httpd_t:s0 tcontext=system_u: ↵
         object_r:http_port_t:s0 tclass=tcp_socket
```

Die Nummer des Ereignisses (*Event-ID*) geben Sie mit der Option `-a`, `--event` an. Da der Audit-Daemon bei jedem neuen Start neu mit der Nummerierung beginnt, findet der Befehl `ausearch` häufig viele Meldungen mit identischer Nummer. Hier können Sie dann mit den Optionen `--start` und `--end` den zu durchsuchenden Zeitraum einschränken. Bei einem deutschen System müssen Sie hierbei den Zeitraum auch in deutscher Notation angeben. Auf einem englischen System wählen Sie: `--end 11/08/06 --start 11/08/06`

Natürlich können Sie auch nach jedem anderen Aspekt suchen, der in einer Audit-Meldung protokolliert wurde. Besonders interessant sind hierbei die Optionen `-k` und `-w`. Wurde beim Setzen der Audit-Regeln ein Schlüssel mit der Option `-k` definiert, kann `ausearch` hiernach suchen. Mit der Option `-w` suchen Sie sogar nach beliebigen Zeichenketten. Alle weiteren Optionen sind in der Manpage erläutert.

A.6 autrace

Das Kommando `autrace` ähnelt dem Kommando `strace`. Es nutzt für die Erkennung der aufgerufenen System-Calls aber das Audit-Subsystem des Kernels. Hierzu erzeugt es vor dem Aufruf des zu analysierenden Programms Audit-Regeln und startet dann das Programm. Die aufgerufenen System-Calls werden dann von dem Audit-Subsystem über den Auditd protokolliert. Nach seiner Beendigung teilt der Befehl mit, wie diese Protokollmeldungen angezeigt werden können.

```
[root@supergrobi ~]# autrace /bin/ls /tmp
Waiting to execute: /bin/ls
...
Cleaning up...
Trace complete. You can locate the records with 'ausearch -i ↵
         -p 13014'
[root@supergrobi ~]# ausearch -i -p 13014
----
type=SYSCALL msg=audit(03.05.2007 20:30:11.085:895) : arch=i386 ↵
         syscall=close success=yes exit=0 a0=1 a1=0 a2=82aff4 ↵
         a3=82b4c0 items=0 ppid=13012 pid=13014 auid=spenneb ↵
          uid=root gid=root euid=root suid=root fsuid=root ↵
         egid=root sgid=root fsgid=root tty=pts2 comm=ls ↵
         exe=/bin/ls subj=user_u:system_r:unconfined_t:s0 key=(null)
----
...
```

Vor und nach dem Aufruf des zu analysierenden Programmes löscht der Befehl `autrace` sämtliche geladenen Audit-Regeln.

B Profile für den Benchmark

Für den Benchmark mit LMBench und UNIXBench habe ich Profile erzeugt, die die Aktionen dieser Programme überwachen. Hierzu wurden die Programme der Einfachheit halber im Heimatverzeichnis des Benutzers *root* installiert.

B.1 AppArmor-Profile für den Benchmark

Das folgende AppArmor-Profil wurde für den Benchmark UNIXBench verwendet:

```
# vim:syntax=apparmor
# Last Modified: Sat Jul  7 13:37:04 2007
#include <tunables/global>

/root/unixbench-4.1.0/Run {
  #include <abstractions/base>
  #include <abstractions/bash>
  #include <abstractions/nameservice>

  capability dac_override,

  /*.s rw,
  /bin/bash ixr,
  /bin/cat ixr,
  /bin/chmod ixr,
  /bin/date ixr,
  /bin/df ixr,
  /bin/gawk ixr,
  /bin/grep ixr,
  /bin/ls ixmr,
  /bin/rm ixr,
  /bin/sed ixr,
  /bin/sleep ixr,
  /bin/sort ixr,
  /bin/sync ixr,
  /bin/uname ixr,
  /dev/tty rw,
  /etc/magic r,
```

```
  /proc/loadavg r,
  /proc/uptime r,
  /root/unixbench-4.1.0 r,
  /root/unixbench-4.1.0/Makefile r,
  /root/unixbench-4.1.0/Run mr,
  /root/unixbench-4.1.0/pgms r,
  /root/unixbench-4.1.0/pgms/* ixr,
  /root/unixbench-4.1.0/results/* rw,
  /root/unixbench-4.1.0/src r,
  /root/unixbench-4.1.0/testdir r,
  /root/unixbench-4.1.0/testdir/* rw,
  /root/unixbench-4.1.0/tmp r,
  /root/unixbench-4.1.0/tmp/* rw,
  /tmp r,
  /tmp/* rw,
  /usr/bin/as ixr,
  /usr/bin/dc ixr,
  /usr/bin/expr ixr,
  /usr/bin/file ixr,
  /usr/bin/gcc-4.1 ixr,
  /usr/bin/join ixr,
  /usr/bin/ld ixr,
  /usr/bin/make ixr,
  /usr/bin/nm ixr,
  /usr/bin/od ixr,
  /usr/bin/strip ixr,
  /usr/bin/tail ixr,
  /usr/bin/tee ixr,
  /usr/bin/time ixr,
  /usr/bin/uptime ixr,
  /usr/bin/wc ixr,
  /usr/bin/who ixr,
  /usr/include/* r,
  /usr/include/bits/* r,
  /usr/include/gnu/* r,
  /usr/include/sys/* r,
  /usr/lib/* r,
  /usr/lib/gcc/** r,
  /usr/lib/gcc/i586-suse-linux/4.1.2/cc1 ixr,
  /usr/lib/gcc/i586-suse-linux/4.1.2/collect2 ixr,
  /usr/share/misc/magic r,
  /usr/share/misc/magic.mgc r,
  /var/run/utmp rw,
  /var/tmp r,
}
```

Für den Benchmark LMBench wurde das folgende Profil verwendet:

```
# vim:syntax=apparmor
# Last Modified: Sat Jul  7 16:53:15 2007
#include <tunables/global>

/root/lmbench-3.0-a8/src/rerun flags=(complain) {
  #include <abstractions/base>
  #include <abstractions/bash>
  #include <abstractions/nameservice>
  #include <abstractions/user-tmp>
  #include <abstractions/wutmp>

  capability net_bind_service,

  /bin/bash ixr,
  /bin/cp ixr,
  /bin/date ixr,
  /bin/gawk ixr,
  /bin/grep ixr,
  /bin/hostname ixr,
  /bin/mkdir ixr,
  /bin/mktemp ixr,
  /bin/mount Ux,
  /bin/netstat ixr,
  /bin/rm ixr,
  /bin/rmdir ixr,
  /bin/sed ixr,
  /bin/sleep ixr,
  /bin/touch ixr,
  /bin/true ixr,
  /bin/uname ixr,
  /dev/tty rw,
  /proc/* r,
  /proc/net/* r,
  /root/lmbench-3.0-a8/** rw,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/bw_file_rd ixr,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/bw_mem ixr,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/bw_mmap_rd ixr,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/bw_pipe ixr,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/bw_tcp ixr,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/bw_unix ixr,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_connect ixr,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_ctx ixr,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_fs ixr,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_http ixr,
  /root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_mem_rd ixr,
```

```
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_mmap ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_ops ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_pagefault ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_pipe ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_proc ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_rpc ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_select ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_sig ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_syscall ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_tcp ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_udp ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lat_unix ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lmbench ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lmdd ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/lmhttp ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/msleep ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/par_mem ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/par_ops ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/stream ixr,
/root/lmbench-3.0-a8/bin/i686-pc-linux-gnu/tlb ixr,
/root/lmbench-3.0-a8/scripts/build ixr,
/root/lmbench-3.0-a8/scripts/compiler ixr,
/root/lmbench-3.0-a8/scripts/config ixr,
/root/lmbench-3.0-a8/scripts/gnu-os ixr,
/root/lmbench-3.0-a8/scripts/make ixr,
/root/lmbench-3.0-a8/scripts/os ixr,
/root/lmbench-3.0-a8/scripts/results ixr,
/root/lmbench-3.0-a8/scripts/target ixr,
/root/lmbench-3.0-a8/src/rerun mr,
/sbin/ifconfig ixr,
/tmp/hello ixr,
/usr/bin/as ixr,
/usr/bin/env ixr,
/usr/bin/expr ixr,
/usr/bin/gcc-4.1 ixr,
/usr/bin/ld ixr,
/usr/bin/make ixr,
/usr/bin/nm ixr,
/usr/bin/strip ixr,
/usr/bin/uptime ixr,
/usr/include/** r,
/usr/lib/* r,
/usr/lib/gcc/i586-suse-linux/** r,
/usr/lib/gcc/i586-suse-linux/4.1.2/cc1 ixr,
/usr/lib/gcc/i586-suse-linux/4.1.2/collect2 ixr,}
```

B.2 SELinux-Richtlinie für den Benchmark

SElinux benötigte auch Richtlinien für die beiden Benchmarks. Zunächst die Type-Enforcement-Datei für den UNIXBench-Benchmark:

```
policy_module(unixbench,1.0.2)

########################################
#
# Declarations
#
require {
        type initrc_var_run_t;
        type user_home_t;
}

type unixbench_t;
type unixbench_exec_t;
domain_type(unixbench_t)
domain_entry_file(unixbench_t, unixbench_exec_t)
domain_auto_trans(unconfined_t, unixbench_exec_t, unixbench_t)


# Some common macros (you might be able to remove some)
files_read_etc_files(unixbench_t)
libs_use_ld_so(unixbench_t)
libs_use_shared_libs(unixbench_t)
miscfiles_read_localization(unixbench_t)
## internal communication is often done using fifo and unix sockets.
allow unixbench_t self:fifo_file { read write };
allow unixbench_t self:unix_stream_socket create_stream_socket_perms;
allow unixbench_t initrc_var_run_t:file { read write lock };
allow unixbench_t self:capability dac_override;
allow unixbench_t self:fifo_file { getattr ioctl };
allow unixbench_t user_home_t:file { execute setattr read create ↵
         getattr execute_no_trans write ioctl unlink append };
corecmd_exec_bin(unixbench_t)
corecmd_exec_ls(unixbench_t)
corecmd_exec_shell(unixbench_t)
corecmd_read_bin_symlinks(unixbench_t)
corecmd_search_bin(unixbench_t)
corecmd_search_sbin(unixbench_t)
files_manage_generic_tmp_files(unixbench_t)
files_read_etc_runtime_files(unixbench_t)
files_read_usr_files(unixbench_t)
```

```
fs_getattr_xattr_fs(unixbench_t)
kernel_read_system_state(unixbench_t)
nscd_read_pid(unixbench_t)
term_search_ptys(unixbench_t)
term_use_generic_ptys(unixbench_t)
unconfined_signull(unixbench_t)
userdom_manage_generic_user_home_content_dirs(unixbench_t)
userdom_search_generic_user_home_dirs(unixbench_t)
```

Für das Modul wird natürlich auch eine File-Context-Datei benötigt:

```
# unixbench executable will have:
# label: system_u:object_r:unixbench_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/root/unixbench-4.1.0/Run -- gen_context ➥
        (system_u:object_r:unixbench_exec_t,s0)
```

Ähnliche Dateien wurden für den LMBench-Benchmark verwendet:

```
policy_module(lmbench,1.0.2)

################################################
#
# Declarations
#

require {
        type portmap_port_t;
        type port_t;
        type lo_node_t;
        type user_home_t;
        type netif_t;
        type etc_runtime_t;
        type tmp_t;
        type hi_reserved_port_t;
        type initrc_var_run_t;
        type http_port_t;
}

type lmbench_t;
type lmbench_exec_t;
domain_type(lmbench_t)
domain_entry_file(lmbench_t, lmbench_exec_t)
domain_auto_trans(unconfined_t, lmbench_exec_t, lmbench_t)
```

```
# Some common macros (you might be able to remove some)
files_read_etc_files(lmbench_t)
libs_use_ld_so(lmbench_t)
libs_use_shared_libs(lmbench_t)
miscfiles_read_localization(lmbench_t)
## internal communication is often done using fifo and unix sockets.
allow lmbench_t self:fifo_file { read write };
allow lmbench_t self:unix_stream_socket create_stream_socket_perms;

allow lmbench_t etc_runtime_t:dir search;
allow lmbench_t hi_reserved_port_t:tcp_socket { name_bind ↵
        name_connect send_msg recv_msg };
allow lmbench_t hi_reserved_port_t:udp_socket { name_bind ↵
        send_msg recv_msg };
allow lmbench_t http_port_t:tcp_socket { name_bind ↵
        name_connect send_msg recv_msg };
allow lmbench_t initrc_var_run_t:file { read write lock };
allow lmbench_t lo_node_t:node { tcp_recv tcp_send udp_recv ↵
        udp_send };
allow lmbench_t netif_t:netif { tcp_recv tcp_send udp_recv ↵
        udp_send };
allow lmbench_t port_t:tcp_socket { name_bind name_connect ↵
        send_msg recv_msg };
allow lmbench_t portmap_port_t:tcp_socket { name_connect ↵
        send_msg recv_msg };
allow lmbench_t self:capability net_bind_service;
allow lmbench_t self:fifo_file { getattr ioctl };
allow lmbench_t self:netlink_route_socket { write getattr read ↵
        bind create nlmsg_read };
allow lmbench_t self:process { signal signull sigkill };
allow lmbench_t self:tcp_socket { setopt read bind create accept ↵
        write getattr connect listen };
allow lmbench_t self:udp_socket { setopt read bind create ioctl ↵
        write getattr connect };
allow lmbench_t self:unix_dgram_socket create;
allow lmbench_t tmp_t:dir { search read create write getattr rmdir ↵
        remove_name add_name };
allow lmbench_t tmp_t:file { execute setattr read create ↵
        execute_no_trans write getattr unlink append };
allow lmbench_t user_home_t:file { execute read create getattr ↵
        execute_no_trans write ioctl unlink };
corecmd_exec_bin(lmbench_t)
corecmd_exec_shell(lmbench_t)
corecmd_read_bin_symlinks(lmbench_t)
```

```
corecmd_search_bin(lmbench_t)
corecmd_search_sbin(lmbench_t)
corenet_tcp_bind_inaddr_any_node(lmbench_t)
corenet_udp_bind_inaddr_any_node(lmbench_t)
corenet_udp_sendrecv_generic_port(lmbench_t)
corenet_udp_sendrecv_portmap_port(lmbench_t)
dev_read_urand(lmbench_t)
files_read_etc_runtime_files(lmbench_t)
files_read_usr_files(lmbench_t)
files_read_usr_symlinks(lmbench_t)
fs_getattr_xattr_fs(lmbench_t)
hostname_exec(lmbench_t)
kernel_read_network_state(lmbench_t)
kernel_read_network_state_symlinks(lmbench_t)
kernel_read_sysctl(lmbench_t)
kernel_read_system_state(lmbench_t)
kernel_search_network_sysctl(lmbench_t)
mount_exec(lmbench_t)
nscd_read_pid(lmbench_t)
sysnet_exec_ifconfig(lmbench_t)
sysnet_read_config(lmbench_t)
term_search_ptys(lmbench_t)
term_use_generic_ptys(lmbench_t)
userdom_manage_generic_user_home_content_dirs(lmbench_t)
userdom_search_generic_user_home_dirs(lmbench_t)
corenet_tcp_bind_dhcpd_port(lmbench_t)
corenet_udp_bind_ipp_port(lmbench_t)
```

Die FC-Datei enthielt den Eintrag für das Kommando selbst:

```
# lmbench executable will have:
# label: system_u:object_r:lmbench_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/root/lmbench-3.0-a8/src/rerun          --      gen_context ➥
         (system_u:object_r:lmbench_exec_t,s0)
```

C Ergebnisse des LMBench

C.1 AppArmor

Hier finden Sie die ausführlichen Ergebnisse des LMBench-Benchmarks für AppArmor.

```
                L M B E N C H  3 . 0   S U M M A R Y
                ------------------------------------
                (Alpha software, do not distribute)

Basic system parameters
------------------------------------------------------------------------------
Host                 OS Description              Mhz  tlb  cache  mem   scal
                                                     pages line   par   load
                                                           bytes
--------- ------------- ----------------------- ---- ----- ------ ------ ----
station2  Linux 2.6.18. noapparmor              1852                      1
station2  Linux 2.6.18. apparmor                1852                      1

Processor, Processes - times in microseconds - smaller is better
------------------------------------------------------------------------------
Host                 OS  Mhz null null      open slct sig  sig  fork exec sh
                             call  I/O stat clos TCP  inst hndl proc proc proc
--------- ------------- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
station2  noapparmor    1852 0.33 0.48 3.52 4.46 6.22 0.77 2.16 135. 468. 2420
station2  apparmor      1852 0.33 0.57 3.94 9.67 6.22 0.77 2.16 136. 635. 2994

Basic integer operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  intgr intgr  intgr  intgr  intgr
                         bit   add    mul    div    mod
--------- ------------- ------ ------ ------ ------ ------
station2  noapparmor    0.3300 0.3400 3.3100   36.1   27.1
station2  apparmor      0.3300 0.3300 3.3100   36.1   27.1

Basic uint64 operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS int64  int64  int64  int64  int64
                         bit    add    mul    div    mod
--------- ------------- ------ ------ ------ ------ ------
station2  noapparmor           1.3600
station2  apparmor             1.3600

Basic float operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  float  float  float  float
                         add    mul    div    bogo
--------- ------------- ------ ------ ------ ------
station2  noapparmor    1.9600 2.7200   15.1   15.1
station2  apparmor      1.9600 2.7200   15.1   15.1
```

```
Basic double operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  double double double double
                         add    mul    div    bogo
--------- ------------- ------ ------ ------ ------
station2  noapparmor    1.9600 2.7200   15.1   15.1
station2  apparmor      1.9600 2.7200   15.1   15.1

Context switching - times in microseconds - smaller is better
-------------------------------------------------------------------------
Host                 OS  2p/0K 2p/16K 2p/64K 8p/16K 8p/64K 16p/16K 16p/64K
                         ctxsw  ctxsw  ctxsw ctxsw  ctxsw   ctxsw   ctxsw
--------- ------------- ------ ------ ------ ------ ------ ------- -------
station2  noapparmor    2.6400 3.0600 3.6000 3.5100 5.8500 4.16000    12.4
station2  apparmor      2.7200 3.1100 3.6500 3.5400 6.1500 3.97000    13.5

*Local* Communication latencies in microseconds - smaller is better
---------------------------------------------------------------------
Host                 OS 2p/0K  Pipe AF     UDP  RPC/   TCP  RPC/ TCP
                        ctxsw       UNIX         UDP         TCP conn
--------- ------------- ----- ----- ---- ----- ----- ----- ----- ----
station2  noapparmor    2.640 8.456 16.5  16.2        20.8        49.
station2  apparmor      2.720 8.694 16.5  16.2        21.1        49.


File & VM system latencies in microseconds - smaller is better
-------------------------------------------------------------------------------
Host                 OS   0K File      10K File     Mmap    Prot   Page   100fd
                        Create Delete Create Delete Latency Fault  Fault  selct
--------- ------------- ------ ------ ------ ------ ------- ----- ------- -----
station2  noapparmor      37.9   52.5   86.9   88.8  2607.0 0.614 1.64880 2.789
station2  apparmor        42.5   55.9   93.2   89.7  2614.0 0.671 1.64120 2.786

*Local* Communication bandwidths in MB/s - bigger is better
-----------------------------------------------------------------------------
Host                OS  Pipe AF    TCP  File   Mmap  Bcopy  Bcopy  Mem   Mem
                             UNIX      reread reread (libc) (hand) read write
--------- ------------- ---- ---- ---- ------ ------ ------ ------ ---- -----
station2  noapparmor    1556 2476 1152 2363.6 4065.1 1085.4 1132.7 4076 1716.
station2  apparmor      1502 2461 1157 2389.7 4061.9 1086.5 1134.3 4078 1712.

Memory latencies in nanoseconds - smaller is better
    (WARNING - may not be correct, check graphs)
------------------------------------------------------------------------------
Host                 OS   Mhz   L1 $   L2 $    Main mem    Rand mem    Guesses
--------- -------------   ---   ----   ----    --------    --------    -------
station2  noapparmor     1852 1.3280 9.4720   43.7       162.5
station2  apparmor       1852 1.3270 9.3540   43.7       162.6
```

C.2 SELinux

Hier finden Sie die ausführlichen Ergebnisse des LMBench-Benchmarks für SELinux.

```
                L M B E N C H  3 . 0   S U M M A R Y
                ------------------------------------
                (Alpha software, do not distribute)

Basic system parameters
------------------------------------------------------------------------------
Host                 OS Description              Mhz  tlb  cache  mem   scal
                                                     pages line   par   load
                                                           bytes
--------- ------------- ----------------------- ---- ----- ------ ------ ----
station1. Linux 2.6.18-       i686-pc-linux-gnu 2991                       1
station1. Linux 2.6.18-            i686-selinux 2991                       1
```

```
Processor, Processes - times in microseconds - smaller is better
------------------------------------------------------------------------------
Host                 OS  Mhz null null      open slct sig  sig  fork exec sh
                             call  I/O stat clos TCP  inst hndl proc proc proc
--------- ------------- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
station1. noselinux     2991 0.47 0.64 4.57 5.96      0.93 2.54 148. 480. 1970
station1. selinux       2991 0.47 0.74 5.66 7.23      0.92 2.67 149. 508. 2130

Basic integer operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS  intgr intgr  intgr  intgr  intgr
                          bit   add    mul    div    mod
--------- ------------- ------ ------ ------ ------ ------
station1. noselinux     0.3300 0.3400 0.3500   36.0   27.1
station1. selinux       0.3300 0.3400 0.3500   36.3   27.1

Basic uint64 operations - times in nanoseconds - smaller is better
-------------------------------------------------------------------
Host                 OS int64  int64  int64  int64  int64
                          bit    add    mul    div    mod
--------- ------------- ------ ------ ------ ------ ------
station1. noselinux             0.3900
station1. selinux               0.3900

Basic float operations - times in nanoseconds - smaller is better
-----------------------------------------------------------------
Host                 OS  float  float  float  float
                         add    mul    div    bogo
--------- ------------- ------ ------ ------ ------
station1. noselinux     2.0100 2.7400   15.1   15.1
station1. selinux       2.0100 2.7400   15.1   15.1

Basic double operations - times in nanoseconds - smaller is better
------------------------------------------------------------------
Host                 OS  double double double double
                         add    mul    div    bogo
--------- ------------- ------  ------ ------ ------
station1. noselinux     2.0100 2.7400   15.1   15.1
station1. selinux       2.0100 2.7400   15.1   15.1

Context switching - times in microseconds - smaller is better
-------------------------------------------------------------------------
Host                 OS  2p/0K 2p/16K 2p/64K 8p/16K 8p/64K 16p/16K 16p/64K
                         ctxsw  ctxsw  ctxsw ctxsw  ctxsw   ctxsw   ctxsw
--------- ------------- ------ ------ ------ ------ ------ ------- -------
station1. noselinux     3.7800 3.9100 4.4100 4.2700 6.5600 5.01000    13.5
station1. selinux       4.1200 4.2100 4.5800 4.6300 8.2700 4.90000    14.8

*Local* Communication latencies in microseconds - smaller is better
---------------------------------------------------------------------
Host                 OS 2p/0K  Pipe AF     UDP  RPC/   TCP  RPC/ TCP
                        ctxsw       UNIX         UDP         TCP conn
--------- ------------- ----- ----- ---- ----- ----- ----- ----- ----
station1. noselinux     3.780  11.3 69.2
station1. selinux       4.120  12.3 53.0

File & VM system latencies in microseconds - smaller is better
------------------------------------------------------------------------------
Host                 OS   0K File      10K File     Mmap    Prot   Page   100fd
                        Create Delete Create Delete Latency Fault  Fault  selct
--------- ------------- ------ ------ ------ ------ ------- ----- ------- -----
station1. noselinux       19.2   15.7   58.5   32.3  2959.0 0.464 1.62070 2.955
station1. selinux         40.1   23.8   78.0   37.2  2966.0 0.374 1.61960 2.946

*Local* Communication bandwidths in MB/s - bigger is better
-----------------------------------------------------------------------------
Host                OS  Pipe AF    TCP  File   Mmap  Bcopy  Bcopy  Mem   Mem
                             UNIX      reread reread (libc) (hand) read write
```

```
--------- ------------- ---- ---- ---- ------ ------ ------ ------ ---- -----
station1. noselinux     1440 2517      2381.8 4121.7 1072.0 1133.9 4120 1694.
station1. selinux       1460 2059      2354.9 4112.4 1068.7 1134.2 4073 1702.

Memory latencies in nanoseconds - smaller is better
    (WARNING - may not be correct, check graphs)
------------------------------------------------------------------------------
Host                 OS   Mhz   L1 $   L2 $    Main mem    Rand mem    Guesses
--------- -------------   ---   ----   ----    --------    --------    -------
station1. noselinux      2991 1.3380 9.3630   44.3        154.3
station1. selinux        2991 1.3380 9.3760   44.2        154.9
```

Stichwortverzeichnis