



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 *Integritätssicherung*. 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 herabgesetzt, 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

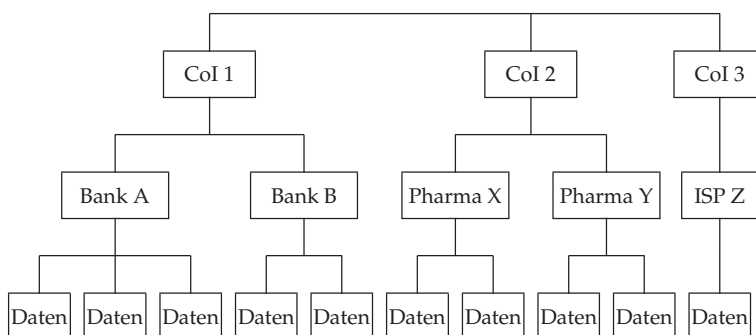


Abbildung 2.1: Chinese-Wall-Klassifizierung

- 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 Rollenzuordnung 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 verzichtet¹. 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. Februar 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 zu 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 *Capability*s standardisiert werden. Leider hat sich dieser Draft nie durchgesetzt.

Jeder Prozess besitzt nun drei Bitmaps, in denen die *Capability*s gespeichert werden:

- Vererbte *Capability*s (Inheritable, I)
- Erlaubte *Capability*s (Permitted, P)
- Effektive *Capability*s (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 *Capability*s 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 Capability der Prozess aktiv nutzen darf. Ein Prozess kann eine Capability vorübergehend in dem Satz der effektiven Capabilitys deaktivieren und später wieder aktivieren. Mehr Capability, als sich im Satz der erlaubten Capabilitys befinden, kann er aber nicht benutzen.

Die vererbaren Capabilitys geben an, welche Capability 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 Capability nur für Prozesse. Die POSIX- Capability 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 Capability existieren?

Insgesamt existieren in Abhängigkeit der Kernel-Version etwa 30 verschiedene Capability. 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 Capabilities 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 Capabilities. 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 Capabilities. 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 Domännennamens
 - 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 Capabilities ein?

Die Capabilities 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 Capabilities. Diese Verwaltung ist meist auch sehr flexibel und komfortabel. Aber Linux-Systeme, die diese Funktionen nicht besitzen, können auch von den Capabilities profitieren. Die meisten Administratoren kennen nur nicht die Möglichkeiten.

Die Capabilities können über `/proc/sys/kernel/cap-bound` gelesen und geschrieben werden. Dies ist das Capability-Bounding-Set. Dieses gibt die maximal ausübaren Capabilities 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 Capabilities 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 Capabilities 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	

* = Capabilities 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. Capabilities, 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 Capabilities.

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-Capabilities. Diese benötigen einen Kernel-Patch, damit sie funktionieren. Dieser wurde von Serge E. Hallyn² auf der *linux-security-module*-Mailingliste veröffentlicht³ 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-Capabilities 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-Capabilities 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-Capabilities 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
        / rwcmlxi

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 Virenschanner.
- *CAP*: Erlaubt die Definition von maximalen Capabilities 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.

