Regelsatz-basierte Zugriffskontrolle nach dem "Generalized Framework for Access Control"-Ansatz am Beispiel Linux

Diplomarbeit

Betreuung:
Dr. Simone Fischer-Hübner
Dr. Hermann de Meer

Amon Ott
Email: 1Ott@Informatik.Uni-Hamburg.de
WWW: http://agn-www.informatik.uni-hamburg.de/people/1ott
Arbeitsbereich AGN
Fachbereich Informatik
Universität Hamburg

10. November 1997

Danksagung

Diese Diplomarbeit entstand in der Zeit vom November 1996 bis November 1997. Ich möchte mich bei meinen Betreuern Dr. Simone Fischer-Hübner und Dr. Hermann de Meer für ihre Betreuungsarbeit bedanken, bei Simone Fischer-Hübner insbesondere auch für die umfangreiche Unterstützung bei allen auftauchenden Fragen zu ihrem Datenschutzmodell und die vielen zusätzlichen Anregungen.

Außerdem bedanke ich mich bei meiner Verlobten Anne Brockmann für ihre Geduld und ihr Verständnis und bei meinem Kompagnon Karsten Brauch für die Arbeitsentlastung vor allem in der Endphase der Arbeit.

Amon Ott, Hamburg, am 10.11.1997.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Traditionelle Zugriffskontrollsysteme bilden stets eine Einheit im Systemkern. Die eigentliche Sicherheitspolitik ist dabei eng mit dem Gesamtentwurf verknüpft und wird im Sicherheitssystem fest kodiert, was eine Anpassung der Sicherheitspolitik an neue Anforderungen enorm erschwert.

Diese Arbeit verwendet einen neuen Ansatz von L. J. LaPadula[LaPadula95] auf der Grundlage des "Generalized Framework for Access Control", welches durch eine Projektgruppe unter Leitung von Marshall Abrams entwickelt wurde (siehe z. B. [Abrams+90]). In diesem wird durch geschickte Aufteilung der funktionalen Komponenten eine Konfiguration verschiedenster Sicherheitspolitiken anhand mehrerer, leicht erweiterbarer Sicherheitsmodelle ermöglicht.

Für die praktische Umsetzung dieses Ansatzes habe ich die Unix-Variante Linux in der Kernversion 2.0.30 gewählt, da dieses System im Quelltext frei zugänglich, sehr stabil und dem von La Padula gewählten Beispielsystem ähnlich ist. Außerdem hält es sich an die gängigen Unix-Standards, was eine Übertragung der Ergebnisse auf andere Unix-Systeme erleichtert. Da es üblich ist, allen Linux-Erweiterungen einen englischen Namen zu geben, habe ich das Quelltextpaket "Rule Set Based Access Control" (RSBAC) genannt.

Durch die Wahl eines Unix-Systems ergibt sich als Zielsetzung, eine schwache, diskrete Zugriffskontrolle durch Erweiterung im Kern um eine leistungsfähigere, flexiblere und mandatorische Kontrolle zu erweitern, die durch Konfiguration anstelle von umständlicher Kodierung mehrere anerkannte Sicherheitsmodelle durchsetzen kann. Der gewählte Ansatz erlaubt außerdem aufgrund der modularisierten Regelsätze ein einfaches Hinzufügen weiterer Sicherheitsmodelle.

Im Rahmen dieser Arbeit wurde der unvollständige Entwurf von La Padula überarbeitet, erweitert, für ein konkretes System vervollständigt und schließlich implementiert.

Als besonderes Beispiel für die Integrationsfähigkeit des entstandenen Systems wurde das komplexe Datenschutzmodell von Dr. Simone Fischer-Hübner[FiHue94, FiHue95, FiHue97] als fünftes Sicherheitsmodell ausgewählt, das hiermit zum ersten Mal in einem konkreten System implementiert ist. Die Umsetzung in die Systemgegebenheiten erfolgte in direkter Zusammenarbeit mit Frau Fischer-Hübner.

Da ich besonderen Wert auf den Grundbereich Privatheit gelegt habe, verwendet die umfangreiche Protokollierung Benutzerpseudonyme, die durch Sicherheits- bzw. Datenschutzbeauftragte jederzeit geändert werden können.

Kapitel 2 verschafft zunächst einen kurzen Überblick über allgemeine Sicherheitsbewertungen und die in dieser Arbeit integrierten Sicherheitsmodelle, um das Verständnis der weiteren Kapitel zu ermöglichen.

Kapitel 3 bietet dann einen Einblick in den Aufbau und eine grundlegende Sicherheitsbewertung des Linux-Systemkerns.

In Kapitel 4 werden das "Generalized Framework for Access Control" und der Ansatz von La Padula vorgestellt, der in Kapitel 5 zum fertigen Entwurf umgearbeitet und erweitert wird. Kapitel 6 betrachtet die Implementation in den Linux-Kern.

Installation und Administration des fertigen Systems sind Thema des Kapitels 7, gefolgt von der Gesamtbewertung des entstandenen Systems im Kapitel 8. Zum Abschluß bietet Kapitel 9 eine Zusammenfassung der gesamten Arbeit.

In den Anhängen befinden sich die von mir erstellten oder veränderten Quelltexte in ungekürzter Fassung.

Kapitel 2

Sicherheitsmodelle

Dieses Kapitel erläutert kurz die wichtigsten Beurteilungskriterien für die Sicherheit von Informationssystemen. Anschließend werden die in dieser Arbeit verwendeten Sicherheitspolitiken und -modelle so weit dargestellt, wie es für das Verständnis erforderlich ist. Eine umfassende Beschreibung der einzelnen Modelle kann hier natürlich nicht gegeben werden.

2.1 Allgemeine Sicherheitskriterien

2.1.1 Grundbereiche

Die Aspekte der Sicherheit der Informationstechnik werden klassisch in drei Bereiche aufgeteilt:

Vertraulichkeit: Schutz vor unbefugter Preisgabe von Informationen

Integrität: Schutz vor unbefugter Veränderung von Informationen und Gewährleistung ihrer inneren und äußeren Konsistenz. Dabei bezeichnet erstere die logische Konsistenz der Daten untereinander, letztere die korrekte Abbildung der Wirklichkeit.

Verfügbarkeit: Schutz vor unbefugter Vorenthaltung von Informationen oder Betriebsmitteln

Dazu kommt noch ein neuerer Bereich:

Privatheit: Schutz vor unbefugter Preisgabe oder Verwendung von personenbezogenen Informationen

Diese vier Kategorien dienen als Grundlage für die Einschätzung der Sicherheitseigenschaften eines Modells.

2.1.2 Ansatz der TCSEC ("Orange Book")

Die "Trusted Computer Security Evaluation Criteria" des US-amerikanischen Verteidigungsministeriums[TCSEC85] konzentrieren sich auf den Bereich Vertraulichkeit. Es werden hierarchisch aufsteigende Sicherheitsklassen von Systemen mit steigenden Anforderungen an Sicherheitsfunktionalität und Qualität definiert, die jeweils unter folgenden Aspekten beschrieben werden:

Sicherheitspolitik: Jedes System benötigt eine wohldefinierte Sicherheitspolitik.

Markierung: Jedem Objekt müssen Zugriffskontrollinformationen eindeutig zuzuordnen sein.

Identifizierung: Jedes Subjekt mit Zugriff zum System ist zu identifizieren und zu authentisieren.

Zuordbarkeit: Alle Aktionen im System müssen ihren jeweiligen Verursachern zuzuordnen sein.

Verläßlichkeit: Die Durchsetzung der obigen vier Kriterien ist durch unabhängig zu evaluierende Mechanismen sicherzustellen.

Fortlaufender Schutz: Die Sicherheit des Systems muß auch zukünftigen Anforderungen genügen können, weshalb unautorisierte Veränderungen an Geräten und Programmen zu verhindern sind.

Die definierten Sicherheitsklassen lassen sich folgendermaßen kurz charakterisieren:

D: Minimaler Schutz, die höheren Anforderungen sind nicht erreicht worden.

C: Diskreter Schutz

- **C1:** Diskreter Zugriffsschutz.
- **C2:** Kontrollierter Zugriffsschutz

B: Mandatorischer Schutz

- **B1:** Markierter Zugriffsschutz
- **B2:** Strukturierter Schutz
- **B3:** Sicherheitsdomänen

A: Verifizierter Schutz

- **A1:** Verifiziertes Design

Beyond A1: Verifizierte weitere Bereiche

Die mangelnde Behandlung der Grundbereiche Verfügbarkeit, Integrität und Privatheit läßt die Benutzung dieser Kriterien nur für eng begrenzte, meist militärische Anwendungen zu.

2.1.3 Ansatz der ITSEC

Die europäischen, harmonisierten Kriterien für die Bewertung der Sicherheit von Systemen der Informationstechnik ("Information Technology Security Evaluation Criteria") [ITSEC91], die in dieser Arbeit zugrundegelegt werden, unterteilen, anders als die TCSEC, die Sicherheit eines Systems in Funktionalität und Vertrauenswürdigkeit. Dabei werden die Grundbereiche Vertraulichkeit, Verfügbarkeit und Integrität abgedeckt.

Zur Bewertung der Funktionalität werden von den ITSEC folgende generische Oberbegriffe ("Generic Headings") vorgeschlagen, die sich leicht den oben genannten Grundbereichen zuordnen lassen:

- Identifizierung und Authentisierung
- Zugriffskontrolle
- Beweissicherung
- Protokollauswertung
- Wiederaufbereitung
- Unverfälschtheit
- Zuverlässigkeit der Dienstleistung
- Übertragungssicherung

Für die Sicherheitsbeurteilungen in dieser Arbeit werden zwei Oberbegriffe ergänzt, um den Grundbereich Privatheit mit abzudecken:

- Schutz von personenbezogenen Benutzerdaten
- Schutz von personenbezogenen Daten anderer Personen

Die Vertrauenswürdigkeit schließlich umfaßt die Wirksamkeit und die Korrektheit der Sicherheits-Funktionalität eines Systems.

Die ITSEC erlauben bereits recht differenzierte Sicherheitsuntersuchungen für Systeme, die gar nicht oder nur in geringem Umfang personenbezogenene Daten speichern und verarbeiten. Leider ist der Grundbereich Privatheit nicht berücksichtigt worden.[1]

[1] Für eine ausführlichere kritische Auseinandersetzung mit diesem Kriterienkatalog, wie auch mit den TCSEC, siehe z.B. [Pfitzmann+93].

2.1.4 Privatheit in den Common Criteria

Die Common Criteria sind als Harmonisierung der Sicherheitskriterien der USA, Kanadas und der Europäischen Union entstanden. Das Ziel war die Zusammenfassung der jeweiligen nationalen oder regionalen Sicherheitsbeurteilungskriterien in einem einheitlichen, international verwendbaren Standarddokument. Auch hier wird zwischen Funktionalität und Vertrauenswürdigkeit der Sicherheitsmaßnahmen unterschieden.

Die Vorab-Version 1.0 vom Januar 1996[CC96] unterscheidet ca. 180 funktionale Sicherheitskomponenten in 76 Familien, die wiederum in neun Klassen gruppiert sind. Zwischen funktionalen Sicherheitskomponenten können zusätzlich Abhängigkeiten definiert werden.

In der Klasse Privatheit (FPR) wird der Schutz von personenbezogenen Benutzerdaten thematisiert. Es werden vier Komponenten betrachtet:

Anonymität: Die Identität bestimmter Benutzer ist für das System nicht feststellbar.

Pseudonymität: Die Identität eines Benutzers ist durch ein Pseudonym verborgen, das die Beweissicherung ermöglicht. Nur unter mandatorisch festgelegten Bedingungen erlaubt das System die Identifizierung des Benutzers durch andere Personen.

Unverknüpfbarkeit: Der gleichzeitige Gebrauch mehrerer Ressourcen durch einen Benutzer kann nicht zu einem für andere Benutzer sichtbaren Gesamtprofil verknüpft werden.

Unbeobachtbarkeit: Die Verwendung einer Ressource durch einen Benutzer ist für andere Benutzer nicht feststellbar.

Die beschriebenen Aspekte stehen zum Teil im Widerspruch zu anderen Sicherheitsaspekten wie Beweissicherung und Zugriffskontrolle. Bei der Entwicklung einer Sicherheitspolitik sind deshalb alle Aspekte gegeneinander abzuwägen und Kompromisse zu schließen.

Der Schutz von personenbezogenen Daten anderer Personen, deren Speicherung und Verarbeitung Aufgabe einer Datenverarbeitungsanlage sein kann, wäre auch hier noch dringend zu ergänzen.

2.1.5 Pseudonyme Beweissicherung

Einen Kompromiß zwischen den Anforderungen der Privatheit einerseits und der Beweissicherung und Protokollauswertung andererseits bildet die pseudonyme Beweissicherung. In [Sobirey+97] werden die Verstärkung dieser Problematik durch einbrucherkennende Systeme und mögliche Lösungsansätze für pseudonyme Beweissicherung und datenschutzorientierte Einbruchserkennung dargestellt.

Die Vergabe von Pseudonymen und die pseudonyme Beweissicherung sind Bestandteil des Systementwurfs in Kapitel 5.

2.2 Diskrete Zugriffskontrolle in Unix-Systemen (Discretionary Access Control, DAC)

Diskrete Zugriffskontrolle wird bereits in den TCSEC definiert als eine Methode der Zugriffskontrolle auf Basis von Benutzeridentitäten und Gruppen, denen Objekte zugeordnet werden. Da der Besitzer eines Objektes den Zugriff für alle andere Benutzer verwaltet, ist die Durchsetzung der jeweiligen Sicherheitspolitik vollständig von der Diskretion der Objekt-Besitzer abhängig.

Die in jedem Unix-System vorhandene diskrete Zugriffskontrolle unterscheidet zwischen zwei Typen von Benutzern: Der Systemverwalter "root" hat unbeschränkten Zugriff auf alle Ressourcen, während allen anderen Benutzern nur kontrollierter Zugang erlaubt wird.

Alle Objekte, also Dateien, Verzeichnisse und Kommunikationskanäle, haben einen Besitzer und eine zugeordnete Gruppe. Jeder Benutzer befindet sich in mindestens einer Gruppe, von denen stets eine die Hauptgruppe ist, die bei der Neuerzeugung von Prozessen und Objekten dort zusammen mit dem Benutzer eingetragen wird.

Nur der Besitzer und der Systemverwalter dürfen eine neue Besitz- oder Gruppenzuordnung eines Objektes festlegen und Zugriffsrechte auf dieses verwalten. Der Besitzer darf nur solche Gruppen eintragen, in denen er selbst Mitglied ist.

Zugriffsrechte werden für den Besitzer, die zugeordnete Gruppe und alle anderen Benutzer jeweils pauschal festgelegt. Die Gruppenrechte gelten für alle Benutzer, die dieser Gruppe zugeordnet sind, unabhängig von ihrer aktiven Gruppe. Mögliche Rechte sind Lesen (r), Schreiben (w) und Ausführen (x). Das Ausführrecht auf ein Verzeichnis ermöglicht, dieses zum aktuellen Verzeichnis zu machen.

Üblicherweise erfolgt die Darstellung als Kette von neun Zeichen in der Form `rwxrwxrwx`. Je eine Dreiergruppe steht für Benutzer-, Gruppen- und Fremdzugriffe. Nicht gesetzte Rechte erhalten anstatt des Buchstabens einen Strich. Die Kette `rw-r--r--` steht also für Lese- und Schreibrecht für den Besitzer, Leserecht für die zugeordnete Gruppe, sowie Leserecht für alle anderen Benutzer.

Die diskrete Zugriffskontrolle hat einige prinzipbedingte Nachteile in der Sicherheit. Das Hauptproblem ergibt sich aus dem Besitzer-Prinzip, das jedem Benutzer die volle Verfügungsgewalt über die von ihm erstellten oder ihm anvertrauten Objekte gibt. Selbst unter Annahme der vollen Vertrauenswürdigkeit aller Benutzer im Rahmen der jeweils verwalteten Objekte kann durch Bedienungsfehler, fehlerhafte oder bösartige Programme, z. B. ein Trojanisches Pferd, enormer Schaden, wie die Vernichtung oder Verfälschung wichtiger Daten, angerichtet werden.

Wegen der Reduktion der Rechteverteilung auf die drei Stufen Besitzer, systemweite Gruppe und alle Benutzer ist eine sinnvolle Sicherheitsadministration nur mit Hilfe von zusätzlichen, vom Systemverwalter einzurichtenden Gruppen möglich.

Die Sonderrolle des Systemverwalter-Zugangs ohne Zugriffsbeschränkung, der viele wichtige Aufgaben als einziger wahrnehmen darf und deswegen oft verwendet werden muß, erhöht das beschriebene Risiko noch einmal beträchtlich.

2.3 Klassische Mandatorische Zugriffskontrolle (Mandatory Access Control, MAC)

Als mandatorisch wird ein Sicherheitsmodell bezeichnet, in dem eine Sicherheitspolitik ohne Einfluß der einzelnen Benutzern verbindlich durchgesetzt wird. Alle Subjekte und Objekte bekommen ihre Zugriffsattribute vom System oder einer zentralen Instanz zugewiesen, die durch spezielle Benutzerkonten repräsentiert wird.

Entsprechend [LaPadula95] wird in dieser Arbeit die Umsetzung des Bell-LaPadula-Modells als das mandatorische Modell bezeichnet. Selbstverständlich sind die anderen, nachfolgend vorgestellten Modelle ebenfalls mandatorisch.

2.3.1 Bell-LaPadula

Definitionen

Das Modell von Bell und LaPadula[Bell+76] beschreibt den Zugriff aktiver Entitäten, genannt Subjekte, auf passive Entitäten, genannt Objekte. Eine Entität kann, je nach Zugriff, beide Rollen einnehmen.

Aus der Unterscheidung zwischen lesendem und schreibendem Zugriff ergeben sich bei Bell und LaPadula vier mögliche Zugriffsmodi: Weder Lesen noch Schreiben (execute, e), nur Lesen (read, r), nur Schreiben (append, a), sowie Lesen und Schreiben (write, w). Die Menge alle Zugriffsmodi wird als $\mathcal{A}$ bezeichnet.

Systemzustände

Jeder aktuelle Zugriff eines Subjektes S_i auf ein Objekt O_j im Modus $\underline{\mathrm{x}}$ ergibt ein Triple (S_i, O_j, $\underline{\mathrm{x}}$). Über alle Subjekte und Objekte ergibt sich die Menge aktueller Zugriffe b.

Objekte sind nach dem Vater-Sohn-Prinzip strukturiert und bilden somit eine Hierarchie $\mathcal{H}$ aus einem oder mehreren hierarchisch geordneten, voneinander unabhängigen Bäumen.

Die erlaubten Zugriffe aller Subjekte auf alle Objekte befinden sich in der Matrix $\mathcal{M}$. Jede Zelle $\mathcal{M}_{i,j}$ enthält also eine Teilmenge von $\mathcal{A}$ mit den erlaubten

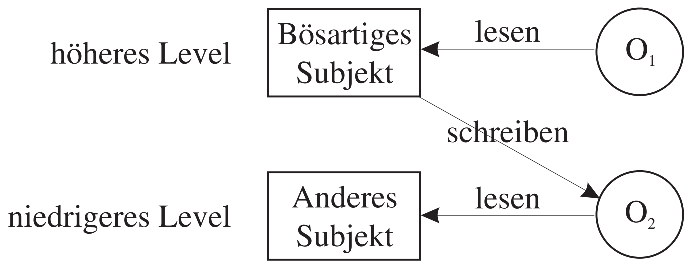

Abbildung 2.1: Unerlaubter Informationsfluß nach Bell-La Padula

Zugriffen von S_i auf O_j.

Ein Sicherheitslevel ist ein Paar (Sicherheitsklassifikation, Menge von Kategorien). Eine Sicherheitsklassifikation ist dabei ein Wert aus einer hierarchischen Ordnung, z. B. öffentlich, vertraulich, geheim und streng geheim, eine Kategorie ist eine formale Zuordnung zu einem Anwendungsbereich. Eine Entität mit dem Sicherheitslevel (S_1,K_1) dominiert eine zweite mit (S_2,K_2), wenn $S_1 \geq S_2$ und $K_1 \supseteq K_2$. Die Eigenschaft "dominiert" bildet über alle Entitäten eine partielle Ordnung $\propto$.

Die Zuordnung von Sicherheitslevels zu Subjekten und Objekten, die Klassifizierungsfunktion $\mathcal{F}$, ist ein Tripel (f_S, f_O, f_C) von Sicherheitslevel-Zuordnungsfunktionen. $f_S(S_i)$ bezeichnet das maximale Sicherheitslevel des Subjektes S_i, $f_O(O_j)$ das Sicherheitslevel des Objektes O_j und $f_C(S_i)$ das aktuelle Sicherheitslevel des Subjektes S_i. Es wird bei Subjekten also zwischen maximalem und aktuellem Level unterschieden. Für alle S_i muß stets gelten: $f_S(S_i)$ dominiert $f_C(S_i)$.

Ein Zustand z des Modells ist ein Tupel (b, $\mathcal{M}$, $\mathcal{F}$, $\mathcal{H}$). Ein System ist die Folge aller Tupel (Anfrage, Entscheidung, Folgezustand) mit dem Anfangszustand z_0.

Sicherheitseigenschaften

Nach Bell und La Padula muß zunächst das simple-security-property (no read-up) erfüllt sein. Dieses sagt aus, daß ein Subjekt S_i ein Objekt O_j nur lesen darf, d. h., (S_i,O_j,r) oder (S_i,O_j,w) ist ein aktueller Zugriff, wenn S_i O_j dominiert.

Um das künstliche Herabstufen eines Objektes durch ein bösartiges Subjekt in einen niedrigeren Level mit Hilfe des Kopierens zu verhindern (siehe Abbildung 2.1), ist außerdem das *-property (no write-down) zu erfüllen. Dazu muß gelten: Hat ein Subjekt S_i gleichzeitig aktuellen Lesezugriff auf ein Objekt O_1 und aktuellen Schreibzugriff auf ein Objekt O_2, dann wird O_1 von O_2 dominiert, d. h., O_1 hat einen niedrigeren Sicherheitslevel als O_2. Informationsfluß kann somit nur von unten nach oben erfolgen.

Da eine strikte Durchsetzung des *-property die Benutzbarkeit des Systems stark reduzieren würde, können als vertrauenswürdig gekennzeichnete Subjekte

("Trusted Subjects") von der *-Eigenschaft ausgenommen werden.

Die Zugriffskontrolle durch die Matrix $\mathcal{M}$ der erlaubten Zugriffe von Subjekten auf Objekte wird als diskrete Zugriffskontrolle, die zugehörige Sicherheitseigenschaft als ds-property bezeichnet. Der aktuelle Zugriffsmodus muß stets in den erlaubten Modi enthalten sein.

Alle genannten "properties" sind wie die Sicherheitslevel vom System mandatorisch durchzusetzen. Jede Sicherheitseigenschaft wird zusätzlich zu den anderen eingesetzt und kann die Sicherheit des Systems deshalb nicht reduzieren. Ein Zustand, der alle Sicherheitseigenschaften erfüllt, wird als "sicher" bezeichnet.

Entscheidungsregeln

Aus den genannten drei "properties" ergibt sich als Regel für die Entscheidung über eine Zugriffserlaubnis:

> Ein aktueller Zugriff (S_i,O_j,$\underline{x}$) wird nur gestattet, wenn die folgenden Sicherheitseigenschaften erfüllt sind:
>
> 1. ss-property: S_i dominiert O_j, falls $\underline{x}$ = r oder $\underline{x}$ = w ($\underline{x}$ enthält Lesezugriff).
> 2. *-property: S_i ist vertrauenswürdig ("Trusted Subject") *oder*
>   (a) O_j dominiert den aktuellen Level von S_i, falls Modus = a
>   (b) Der Level von O_j ist gleich dem aktuellen Level von S_i, falls Modus = w
>   (c) Der aktuelle Level von S_i dominiert den von O_j, falls Modus = r
> 3. ds-property: $\underline{x}$ ist in der Zelle $\mathcal{M}_{i,j}$ der Matrix $\mathcal{M}$ der erlaubten Zugriffe enthalten.

Funktionen

Zum Umgang mit dem so definierten Sicherheitssystem sind für jede Änderung des Systemzustandes Zustandsübergangsfunktionen nötig, die durch Beachtung der Entscheidungsregeln jeden sicheren Zustand beweisbar in einen neuen sicheren Zustand überführen. Durch Induktion kann dann die Sicherheit jedes erreichbaren Systemzustandes nachgewiesen werden.[2]

Ein vollständiger Satz solcher Funktionen kann folgendermaßen aussehen:

- Änderung der Menge b der aktuellen Zugriffe:
  - get-access(): Triple zur Menge b hinzufügen.

[2] Für die formale Definition der Funktionen und den Beweis der Sicherheitserhaltung siehe [Bell+76].

- release-access(): Triple aus b entfernen.

- Änderung der Matrix $\mathcal{M}$ der erlaubten Zugriffe

  - give-access-permission(): Zugriffsmodus zu einer Zelle der Matrix $\mathcal{M}$ hinzufügen.
  - rescind-access-permission(): Zugriffsmodus aus einer Zelle von $\mathcal{M}$ entfernen

- Änderung der Klassifizierungsfunktion $\mathcal{F}$:

  - change-object-level(): Sicherheitslevel eines Objektes ändern (Funktion f_O).
  - change-current-level(): aktuelles Sicherheitslevel eines Subjektes ändern (Funktion f_C).

- Änderung der Objekt-Hierarchie $\mathcal{H}$:

  - create-object(): Objekt als Blatt in einen Objektbaum einhängen.
  - delete-object-group(): Objekt mit allen untergeordneten Objekten aus einem Objektbaum entfernen.

Bewertung

Das Bell-LaPadula-Modell behandelt ausschließlich den Grundbereich Vertraulichkeit. Integrität, Verfügbarkeit und Privatheit der Daten werden nicht gewährleistet. So ist es zum Beispiel möglich, bei niedrigster Sicherheitsklassifikation sämtliche Daten in allen zugeordneten Kategorien willkürlich zu überschreiben, falls diese nicht über die diskrete Kontrolle, also nicht-mandatorisch, geschützt wurden. Derartige Angriffe können auch ohne Absicht des Benutzers, z. B. durch bösartige oder fehlerhafte Programme (Viren etc.) oder Bedienungsfehler, stattfinden. Insbesondere kann auch der diskrete Zugriffsschutz unbemerkt außer Kraft gesetzt werden.

Das Konzept des vertrauenswürdigen Subjektes, das letztendlich nur über Benutzer abgebildet werden kann, erlaubt durch die Umgehung des *-property zusätzliche Angriffsmöglichkeiten über Benutzerkonten mit höherer Sicherheitsklassifizierung.

Insgesamt ist der alleinige Einsatz dieses Modells nur dann zu empfehlen, wenn ausschließlich Vertraulichkeit von Daten benötigt wird, z. B. bei Sammlungen vertraulicher, nicht personenbezogener Dokumente, die leicht wiederherzustellen sind.

2.3.2 Unix System V/MLS

Das in dieser Arbeit verwendete Modell mandatorischer Zugriffskontrolle entspricht weitgehend demjenigen des Unix System V/MLS, Version 1.2.1. Dieses Betriebssystem wurde 1989 vom National Computer Security Center der USA mit dem Sicherheitslevel B1 nach TCSEC[TCSEC85] zertifiziert[Flink+88].

Unix System V/MLS implementiert das oben beschriebene Bell-La Padula-Modell mit leichten Änderungen. So wird z.B. statt des ds-property die Unix-eigene diskrete Zugriffskontrolle verwendet. Sicherheitslevel stehen mit Klassifikation und Kategorien zur Verfügung, simple-security-property und *-property werden durchgesetzt. Im Gegensatz zum Bell-La Padula-Modell ist das Schreiben hier nur bei gleichem Level gestattet.

Im Bell-La Padula-Modell sind bereits vier Zugriffsmodi definiert:

- Ausführen (execute, E)
- Lesen (read, R)
- Schreiben (write, W)
- Anfügen (Append, A)

In diesem Unix-System werden zehn Modi hinzugefügt, die einen Teil der Bedeutungen der ursprünglichen Modi übernehmen:

- Suchen im Verzeichnis (search, S)
- Überschreiben (overwrite, O)
- Erzeugen (create, C)
- Linken (link, U)
- Löschen (unlink, U)
- Status Lesen (read file/i-node status, St)
- Status Ändern (change status, Ch)
- Signal senden (send signal/kill, K)
- Interprozeßkommunikation (IPC) lesen (read IPC, Ripc)
- IPC schreiben (write IPC, Wipc)

Als Subjekte fungieren Prozesse, die jeweils den Sicherheitslevel ihrer Besitzer erben. Die Objekte werden in vier Typen unterschieden:

- Datei

Anforderung	Bedingung für Zugriff
R/S/E	S $\geq$ O
W(O/A)	S = O
C/L/U	S = Od
St	S $\geq$ O
Ch	S = O
Ripc	S $\geq$ O
Wipc	S = O
K	S = O

Tabelle 2.1: Zugriffsbedingungen in System V/MLS

- Verzeichnis
- Interprozeßkommunikationskanal
- Systemkontrollinformation

Die Tabelle 2.1 bietet eine Übersicht über die erlaubten Zugriffe von Subjekten auf Objekte. Dabei bezeichnet S das Subjekt, O das Objekt, Od das betroffene Verzeichnisobjekt und $\geq$ bzw. = stehen für "dominiert" bzw. "hat gleichen Level". Lesen und Schreiben auf Verzeichnisse stehen für Zugriffe auf Einträge, ein Öffnen ist nicht möglich.

2.4 Integritätsmodell von Clark-Wilson (CWI)

Das Integritätsmodell von David D. Clark und David R. Wilson setzt den Schwerpunkt weniger auf Vertraulichkeit der Daten, als auf ihre Integrität.[Clark+87] Dabei wird zwischen interner und externer Integrität unterschieden.

Das Modell definiert die kontrollierten Objekte als "Constrained Data Items" (CDI). Auf diese darf nur mit geprüften, wohldefinierten Transaktionsprozeduren (TP) und Integritäts-Verifikations-Prozeduren (IVP) zugegriffen werden. Transaktionsprozeduren überführen CDI von einem integeren Zustand in einen anderen, während die IVP die Integrität der CDI im System überprüfen und sicherstellen. Der alleinige Zugriff durch diese Prozeduren ist vom System durchzusetzen. Mit Hilfe der Induktion läßt sich so stets ein integrer Zustand nachweisen.

Die Sicherheitseigenschaften dieses Modells sind untergliedert in die Zertifizierungsregeln C1 bis C5, die von Administratoren, und die Durchsetzungsregeln E1 bis E4, die vom System durchzusetzen sind.

Da das System die TP und IVP nicht auf ihre Funktionalität überprüfen kann, müssen diese von außen durch einen Sicherheitsbeauftragten zertifiziert werden.

Die interne Integrität läßt sich durch drei Regeln beschreiben:

C1: (Zertifizierung:) Alle IVP müssen sicherstellen, daß sich zum Zeitpunkt ihrer Ausführung alle überprüften CDIs in einem gültigen, also integren, Zustand befinden.

C2: Alle TP müssen zertifiziert werden, daß sie jedes integere CDI in einen integeren Folgezustand überführen. Für jede TP und jede Menge von CDIs, auf die sie angewendet werden darf, muß der Sicherheitsbeauftragte eine Relation festlegen, die diese Anwendung beschreibt. Eine Relation hat also die Form (TP_i,(CDI_a,CDI_b,CDI_c,...), wobei die CDI-Liste die Argumente spezifiziert, für die die TP zertifiziert wurde.

E1: (Durchsetzung:) Das System muß die in C2 definierten Relationen verwalten und durchsetzen, daß nur in einer Relation befindliche Manipulationen durchgeführt werden.

Die externe Integrität kann nur mit Hilfe der Trennung der Aufgaben ("Separation of Duty") sichergestellt werden, da eine inhaltlich falsche Eingabe für das System nicht erkennbar ist. Es wird also eine Kontrolle von Benutzern durch andere Benutzer erzwungen.

Dafür werden weitere Regeln benötigt, um festzulegen, welche Benutzer welche TP auf welche CDIs anwenden dürfen. Regel E2 schließt zwar Regel E1 mit ein, wird aber aufgrund der verschiedenen Zielsetzungen beibehalten:

E2: Das System muß eine Liste von Relationen verwalten, die Benutzer, TP und die CDIs, auf die diese TP von diesem Benutzer angewendet werden darf, in Beziehung setzen. Das System muß durchsetzen, daß nur in solcher Relation befindliche Manipulationen durchgeführt werden. Die Relationen entsprechen der Form (Benutzer-ID,TP_i,(CDI_a,CDI_b,CDI_c,...)).

C3: Die entsprechend E2 definierten Relationen sind zu zertifizieren, daß sie die Anforderung der Trennung der Aufgaben erfüllen.

Da die Regel E2 auf Benutzeridentitäten zurückgreift, müssen diese vom System sichergestellt werden:

E3: Das System muß die Identität jedes Benutzers, der eine TP ausführen möchte, authentisieren.

Um ein CDI auf einen früheren integeren Zustand zurückführen zu können, ist jede Änderung ausreichend zu protokollieren. Außerdem muß die Umwandlung eines nicht-kontrollierten Datenelements ("Unconstrained Data Item", UDI) in ein CDI ermöglicht werden:

C4: Jede TP muß zertifiziert werden, daß sie alle Informationen, die zur Umkehr der vorgenommenen Veränderungen notwendig sind, in einer nur erweiterbaren Protokolldatei ablegt.

C5: Jede TP, die ein UDI zur Eingabe hat, muß entweder gültige oder gar keine Veränderungen daran vornehmen. Die TP soll das UDI entweder in ein CDI umwandeln oder zurückweisen.

Schließlich ist zur korrekten Funktion des Systems eine Umgehung der Zertifizierung durch Aufgabentrennung zu vermeiden:

E4: Das System muß sicherstellen, daß nur ein zur Zertifizierung von Systemelementen berechtigter Benutzer Relationen zwischen Elementen verändern darf. Ein Benutzer, der ein Element zu zertifizieren berechtigt ist, darf dieses nicht ausführen.

Das Zusammenspiel dieser neun Regeln beruht auf der Trennung von Zertifizierung und Durchsetzung. Zur Vermeidung von Fehlerquellen sollte ein möglichst großer Anteil von der fehlerträchtigen Zertifizierung zur Durchsetzung verlagert werden.

Obwohl im Clark-Wilson-Modell der Schwerpunkt auf der Integrität liegt, erlauben die Zugriffsrelationen auch eine benutzerabhängige, mandatorische Verwaltung der Vertraulichkeit. Wie die Verfügbarkeit und die Privatheit wird diese aber nicht vom System aktiv unterstützt oder gar erzwungen.

Durch die Notwendigkeit, TP und IVP durch einen Sicherheitsbeauftragten zertifizieren zu lassen, wird diesem aufgrund der Komplexität dieser Aufgabe eine große Verantwortung aufgezwungen. Die Umsetzung der Regeln C1 bis C5 sollte unbedingt mit einer Aufgabentrennung verbunden sein, da der Sicherheitserfolg sonst fraglich ist.

Ein alleiniger Einsatz dieses Modells ist nur dort zu empfehlen, wo die Integrität von Daten höheres Gewicht hat als Vertraulichkeit und Verfügbarkeit, z. B. in einigen kommerziellen Bereichen. Für die Verarbeitung personenbezogener Daten ist es nicht ausreichend.

2.5 Funktionale Kontrolle (FC)

Das rollenbasierte Modell der funktionalen Kontrolle ist direkt aus dem Text von La Padula[LaPadula95] übernommen. Es wird an jeden Benutzer genau eine Rolle, z. B. allgemeiner Benutzer, Sicherheitsbeauftragter oder Systemadministrator, vergeben. Gleichzeitig ist jedes Objekt genau einer Kategorie, z. B. allgemeines, Sicherheits- oder Systemobjekt, zugeordnet.

Der Sicherheitsbeauftragte legt fest, welche Rollen mit welchen Kategorien "kompatibel" sind, d. h., mit welcher Rolle ein Benutzer auf welche Objektkategorien zugreifen darf. Das Sicherheitssystem sorgt für die Durchsetzung dieser Festlegungen.

Die funktionale Kontrolle ist in dieser sehr einfachen Form nur zum Schutz von systemeigenen Verwaltungsinformationen geeignet, erzwingt hier aber bereits eine wirksame Aufgabentrennung. Eine Ausweitung auf weitere Rollen könnte Vertraulichkeit, Integrität und Verfügbarkeit der Nutzdaten deutlich stärken. Ohne Unterscheidung wenigstens rudimentärer Zugriffsmodi ist das Modell aber nur als Unterstützung für andere Modelle einsetzbar.

2.6 Änderung der Sicherheitsinformationen (SIM)

Auch dieses rollenbasierte Modell stammt aus dem Text von La Padula[LaPadula95]. Geschützt werden ausschließlich Daten, denen der Typ Sicherheitsinformation zugewiesen wurde. Nur Benutzer mit der Rolle Sicherheitsbeauftragter erhalten auf Sicherheitsinformationen Schreibzugriff.

Wie die funktionale Kontrolle ist auch dieses Modell eher als Ergänzung für Schwachstellen anderer Modelle zu sehen. Bei ansonsten rein diskreter Zugriffskontrolle können hiermit zumindest die wichtigsten Systemdateien vor dem Systemverwalter in Integrität und Verfügbarkeit geschützt werden.

2.7 Datenschutzmodell von Simone Fischer-Hübner (Privacy Model, PM)

Auf der 17. Nationalen Computer-Sicherheitskonferenz in Baltimore, USA, wurde 1994 von Simone Fischer-Hübner das von ihr entwickelte Sicherheitsmodell vorgestellt,[3] das den Anforderungen des bundesdeutschen Datenschutzgesetzes[BDSG91] und der EU-Datenschutzrichtlinie[EU95] Rechnung trägt.

Grundlage dieser Gesetze sind drei im "Volkszählungsurteil" des Bundesverfassungsgerichtes zum Schutz des Grundrechtes auf informationelle Selbstbestimmung festgelegte und im Bundesdatenschutzgesetz angewandte Prinzipien:

Zweckbindung: Jede Erhebung von personenbezogenen Daten erfordert einen wohldefinierten und berechtigten Zweck. Eine Verwendung dieser Daten zu einem anderen Zweck ist nur mit Einwilligung aller Betroffenen gestattet. (§ 14 BDSG)

Erforderlichkeit: Die Erhebung und Verarbeitung personenbezogener Daten sind nur in der Art und dem Umfang gestattet, wie sie für die Aufgaben der erhebenden und verarbeitenden Instanz erforderlich sind. (§§ 13 I, 14 I BDSG)

[3] Siehe [FiHue94].

Organisatorischer und technischer Datenschutz: Durch geeignete organisatorische und technische Maßnahmen sind Vertraulichkeit, Integrität und Verfügbarkeit der erhobenen Daten zu gewährleisten. (§ 9 BDSG)

Im Rahmen des Modells soll als weiteres Ziel, im Gegensatz zu den klassischen Modellen, die modellinhärente Notwendigkeit der Sammlung weiterer persönlicher Daten der Benutzer zu Kontrollzwecken minimiert werden.

2.7.1 Definitionen

Zur Beschreibung des Modelles werden einige Definitionen benötigt:

Subjekte: Subjekte sind die aktiven Elemente eines Systems, z. B. Benutzer und Prozesse. $\mathcal{S}$ ist die Menge aller Subjekte. Das *aktive* Subjekt ist dasjenige, das eine Zustandsänderung verursacht.

Objekte: Objekte sind die passiven Elemente eines Systems, die personenbezogene Daten enthalten. $\mathcal{O}$ ist die Menge aller Objekte.

Objektklassen: Jedem Objekt ist genau eine Klasse zugeordnet, z. B. Untersuchungsbefund.

Aufgaben: Aufgaben sind die Grundlage für alle Objektzugriffe. Jedes Subjekt kann zur selben Zeit nur eine Aufgabe haben. Diese wird als *aktuelle Aufgabe* bezeichnet. $\mathcal{T}$ ist die Menge aller Aufgaben T_i. Diejenigen Aufgaben, die ein Subjekt wahrnehmen darf, sind dessen *autorisierte Aufgaben*.

Verantwortliche Benutzer: Jeder Aufgabe können mehrere Benutzer als verantwortlich zugeordnet werden. Diese sind berechtigt, die Autorisierung anderer Benutzer für diese Aufgabe zu autorisieren (Ausstellen von Tickets, siehe Abschnitt 2.7.4), was eine Delegation der Aufgabe ermöglicht.

Benutzerrolle: Jedem Benutzer ist genau eine der Rollen normaler Benutzer, Sicherheitsbeauftragter, Datenschutzbeauftragter, TP-Verwalter oder Systemadministrator zugeordnet.

Zwecke: Jeder Aufgabe ist genau ein Zweck zugeordnet. Jeder Objektklasse ist mindestens ein Zweck zugeordnet. $\mathcal{P}$ ist die Menge aller Zwecke.

Transformationsprozeduren (TP): Zugriffe von Subjekten auf Objekte sind nur mit Transformationsprozeduren gestattet. $\mathcal{TRANS}$ ist die Menge aller TP. Die TP, die ein Subjekt zu einem Zeitpunkt ausführt, ist dessen *aktuelle TP*. Wird keine TP ausgeführt, ist die aktuelle TP das leere Element NIL.

Autorisierte TP: Für jede Aufgabe müssen die zulässigen TP autorisiert werden.

Zugriffsmodi: Alle Zugriffe von Subjekten auf Objekte erfolgen in mindestens einem der Modi Lesen, Schreiben, Löschen, Erzeugen und Anfügen.

Notwendige Zugriffe: Für jede Aufgabe ist im Voraus zu definieren, auf welche Objektklasse mit welcher TP in welchen Modi zuzugreifen notwendig ist. Einer Aufgabe können mehrere notwendige Zugriffe zugewiesen werden. Notwendige Zugriffe bilden Tupel (T_i, Klasse_j, TP_k, x) mit x = Menge von Zugriffsmodi.

Aktuelle Zugriffe: Ein Tupel (S_i, O_j, x) mit x $\in$ {Lesen, Schreiben, Anfügen} ist ein aktueller Zugriff. $\mathcal{CA}$ ist die Menge aller aktuellen Zugriffe.

Einwilligungen: Die Betroffenen eines Datenobjekts O_j können in dessen Benutzung für einen Zweck p_i einwilligen. $\mathcal{C}$ ist die Menge aller Einwilligungen (p_i, O_j).

2.7.2 Datenschutz-Invarianten

Im Modell gelten einige Datenschutz-Invarianten. Ein Zustand, der alle diese Bedingungen erfüllt, heißt *Datenschutz-orientierter Zustand*.

Autorisierung der aktuellen Aufgabe: Die aktuelle Aufgabe jedes Subjekts S_i muß für dieses Subjekt autorisiert sein, d. h., aktuelle Aufgabe(S_i) $\in$ autorisierte Aufgaben(S_i).

Autorisierung der TP: Die aktuelle TP jedes Subjektes S_i muß für dessen aktuelle Aufgabe autorisiert sein, d. h., aktuelle TP(S_i) $\in$ autorisierte TP(aktuelle Aufgabe(S_i)).

Notwendigkeit des Zugriffs: Jeder aktuelle Zugriff (S_i, O_j, x) muß in der Menge der notwendigen Zugriffe enthalten sein, d. h., (aktuelle Aufgabe(S_i), Objektklasse(O_j), aktuelle TP(S_i), x) ist ein notwendiger Zugriff.

Zweckbindung: Für jeden aktuellen Zugriff (S_i, O_j, x) muß

- der Zweck der aktuellen Aufgabe des Subjektes in den Zwecken der Klasse des Objektes enthalten sein, d. h., Zweck(aktuelle Aufgabe(S_i)) $\in$ Zweckmenge(Klasse(O_j)) *oder*
- eine Einwilligung der Betroffenen für den Zweck der aktuellen Aufgabe und das Objekt vorliegen, d. h., (Zweck(aktuelle Aufgabe(S_i)), O_j) $\in$ $\mathcal{C}$.

2.7.3 Datenschutz-Beschränkungen

Da das Erzeugen und das Löschen von Objekten Zustandsübergänge des Systems beinhalten, sind diesbezüglich Beschränkungen notwendig, um wieder zu einem Datenschutz-orientierten Zustand zu gelangen.

Notwendigkeit der Erzeugung: Ein Subjekt mit der aktuellen Aufgabe T_i, das eine TP TP_k ausführt, darf nur dann ein Objekt der Klasse K_j erzeugen, wenn (T_i, K_j, TP_k, Erzeugen) ein notwendiger Zugriff ist.

Notwendigkeit des Löschens: Ein Subjekt mit der aktuellen Aufgabe T_i, das eine TP TP_k ausführt, darf nur dann ein Objekt der Klasse K_j löschen, wenn (T_i, K_j, TP_k, Löschen) ein notwendiger Zugriff ist.

Zweckbindung der Erzeugung: Ein Subjekt darf nur dann ein Objekt einer Klasse erzeugen, wenn der Zweck seiner aktuellen Aufgabe in den Zwecken der Klasse enthalten ist.

Zweckbindung des Löschens: Ein Subjekt darf nur dann ein Objekt löschen, wenn der Zweck seiner aktuellen Aufgabe in den Zwecken der Klasse des Objektes enthalten ist oder eine Einwilligung für den Zweck der aktuellen Aufgabe und das Objekt vorliegt.

Eine Folge von Zuständen heißt *Datenschutz-orientierte Folge von Zuständen*, wenn alle Zustände Datenschutz-orientiert sind und alle Zustandsübergänge die Beschränkungen erfüllen. Ein System mit Anfangszustand z_0 heißt *Datenschutz-orientiertes System*, wenn alle in ihm möglichen Folgen mit Anfangszustand z_0 Datenschutz-orientiert sind.

2.7.4 Zustandsübergangsfunktionen

Alle Übergänge von einem Zustand zu einem Folgezustand erfolgen mit Zustandsübergangsfunktionen. Diese unterteilen sich in nicht-privilegierte Funktionen zum direkten Umgang mit Objekten und TP, sowie privilegierte Kontrollfunktionen zur Änderung von Kontrollinformationen.

Es ist bewiesen worden, daß diese Zustandsübergangsfunktionen die Datenschutz-Invarianten erhalten und die Datenschutz-Beschränkungen nicht verletzen.[4]

Nicht-privilegierte Funktionen

Diese Funktionen erlauben das Erwerben und Freigeben von Zugriffen, das Starten und Beenden von TP, das Setzen der aktuellen Aufgabe, sowie das Erzeugen und Löschen von Objekten. Sie sind nicht-privilegiert in dem Sinne, daß sie unabhängig von der Benutzerrolle des aufrufenden Subjektes benutzt werden können.

[4] Der Beweis von Simone Fischer-Hübner ist fertiggestellt, aber noch nicht veröffentlicht.

get-access: Diese Funktion dient dem Erwerben von Zugriffen. Sie unterliegt der Notwendigkeit des Zugriffes und der Zweckbindung.

release-access: Die Freigabe eines Zugriffes ist ohne Beschränkung möglich.

execute-TP: Hiermit kann die Ausführung einer TP veranlaßt werden. Voraussetzungen sind die Autorisierung der TP für die aktuelle Aufgabe des Subjektes und eine aktuelle TP NIL.

exit-TP: Das Beenden einer TP unterliegt keinen Beschränkungen. Es werden alle aktuellen Zugriffe dieses Subjektes freigegeben.

change-current-task: Mit dieser Funktion kann ein Subjekt seine aktuelle Aufgabe setzen, falls es für die gewünschte Aufgabe autorisiert ist und die aktuelle TP NIL hat.

create-object: Diese Funktion zum Erzeugen von Objekten unterliegt den in Abschnitt 2.7.3 genannten Beschränkungen.

delete-object: Das Löschen von Objekten unterliegt ebenfalls den in Abschnitt 2.7.3 genannten Beschränkungen.

Privilegierte Kontrollfunktionen

Die Kontrollfunktionen sind, bis auf eine Ausnahme, grundsätzlich von der Benutzerrolle des aufrufenden Subjektes abhängig. Die meisten dieser Funktionen gehorchen dem "Vier-Augen-Prinzip", wonach niemals eine Person alleine eine sicherheitsrelevante Systemänderung vornehmen darf.

Die Realisierung des Prinzips erfolgt mit sogenannten Tickets. Diese sind vom System verwaltete Autorisierungen, eine bestimmte Funktion einmalig mit vorgegebenen Parametern auszuführen. Tickets werden von einem Benutzer mit der Rolle Datenschutzbeauftragter erzeugt, für die Funktionen zur Verwaltung der autorisierten Aufgaben auch von den jeweils für die Aufgabe verantwortlichen Benutzern. Tickets enthalten außerdem eine zeitliche Gültigkeitsbeschränkung.

Der Aufruf der mit Tickets geschützten Funktionen ist nur Benutzern mit der Rolle Sicherheitsbeauftragter unter Angabe des jeweiligen Tickets gestattet. Werden alle Vorgaben des Tickets und die funktionsabhängigen Zusatzbedingungen erfüllt, wird die Funktion ausgeführt und das Ticket gelöscht. Diese Funktion erzeugt ein solches Ticket:

create-ticket(Subjekt, Funktion, Parameter-Liste, Gültigkeitsdauer): Erzeugen eines Tickets.

Nur mit Tickets benutzbar, die von einem Datenschutzbeauftragten ausgestellt wurden, sind folgende Funktionen. Gegebenenfalls gelten die angebenen Zusatzbedingungen:

add-NA($\mathbf{T}_i$, $\mathbf{K}_j$, $\mathbf{TP}_k$, x): Hinzufügen eines notwendigen Zugriffes.

delete-NA($\mathbf{T}_i$, $\mathbf{K}_j$, $\mathbf{TP}_k$, x): Löschen eines notwendigen Zugriffes. Es darf kein aktueller Zugriff aufgrund dieses Tupels vorhanden sein.

add-task($\mathbf{T}_i$, $\mathbf{P}_j$): Hinzufügen der Aufgabe T_i mit Zweck P_j.

delete-task($\mathbf{T}_i$): Löschen der Aufgabe T_i. Es darf kein Subjekt die aktuelle Aufgabe T_i haben.

add-object-class($\mathbf{K}_i$, $\mathcal{P}_j$): Hinzufügen der Objekt-Klasse K_i mit der Zweckmenge $\mathcal{P}_j$.

delete-object-class($\mathbf{K}_i$): Löschen der Objekt-Klasse K_i. Es darf kein Objekt dieser Klasse existieren.

add-authorized-TP($\mathbf{T}_i$, $\mathbf{TP}_j$): Hinzufügen der autorisierten Transformationsprozedur TP_j für die Aufgabe T_i.

delete-authorized-TP($\mathbf{T}_i$, $\mathbf{TP}_j$): Löschen der autorisierten Transformationsprozedur TP_j für die Aufgabe T_i. Es darf keine aktuelle Transformationsprozedur aufgrund dieser Autorisierung vorhanden sein.

add-consent($\mathbf{O}_i$, $\mathbf{P}_j$): Hinzufügen der Einwilligung für O_i mit Zweck P_j.

delete-consent($\mathbf{O}_i$, $\mathbf{P}_j$): Löschen der Einwilligung für O_i mit Zweck P_j. Es darf kein aktueller Zugriff aufgrund dieser Einwilligung vorhanden sein.

set-role($\mathbf{S}_i$, $\mathbf{Rolle}_j$): Setzen der Benutzerrolle Rolle_j für Subjekt S_i.

add-responsible-user($\mathbf{S}_i$, $\mathbf{T}_j$): Hinzufügen des verantwortlichen Benutzers S_i für die Aufgabe T_j.

delete-responsible-user($\mathbf{S}_i$, $\mathbf{T}_j$): Löschen des verantwortlichen Benutzers S_i für die Aufgabe T_j.

Für diese beiden Funktionen können Tickets auch von verantwortlichen Benutzern ausgestellt werden:

add-authorized-task($\mathbf{S}_i$, $\mathbf{T}_j$): Hinzufügen der autorisierten Aufgabe T_j für den Benutzer S_i.

delete-authorized-task($\mathbf{S}_i$, $\mathbf{T}_j$): Löschen der autorisierten Aufgabe T_j für den Benutzer S_i. T_j darf nicht die aktuelle Aufgabe von S_i sein.

Schließlich gibt es noch zwei Funktionen, die ohne Ticket von allen Subjekten mit der Benutzerrolle TP-Verwalter ausführbar sind:

create-TP($\mathbf{TP}_i$): Erzeugen der Transformationsprozedur TP_i.

delete-TP($\mathbf{TP}_i$): Löschen der Transformationsprozedur TP_i. Kein Subjekt darf TP_i als aktuelle Transformationsprozedur haben.

Zur Erleichterung der Systemverwaltung sind zusätzlich Kontrollfunktionen zum Auslesen von Kontrolldaten möglich. Deren Verwendung sollte jeweils auf die Benutzerrollen beschränkt werden, für die diese Informationen notwendig sind.

2.7.5 Bewertung

Der Fokus des Datenschutzmodells liegt auf der Privatheit. Vertraulichkeit, Integrität und Verfügbarkeit werden für personenbezogene Daten und Transformationsprozeduren als Teilaspekte der Privatheit über die Definition notwendiger Zugriffe ebenfalls sichergestellt.

Verwaltungsinformationen des Betriebssystems wie Verzeichnisse, Authentisierungsdaten etc. können nur durch Aufnahme in den Modellbereich, d. h., Deklaration als personenbezogen, geschützt werden. Ist dieses für ein Objekt nicht möglich, kann darauf beliebig zugegriffen werden.

Die Anwendung dieses Modells empfiehlt sich grundsätzlich bei der Speicherung und Verarbeitung personenbezogener Daten. Es sollte jedoch ein weiteres Sicherheitsmodell zum Schutz der systemeigenen Daten hinzugezogen werden, da dieser sonst nur auf Umwegen und mit großem Administrationsaufwand zu erreichen ist.

Kapitel 3

Aufbau des Linux-Kerns

Voraussetzung für das Verständnis dieser Arbeit sind Kenntnisse im Umgang mit Unix-Systemen, die hier nicht vermittelt werden können.[1] Dieses Kapitel beschränkt sich auf die Funktionsweise und Organisation des Systemkerns in Linux.[2]

3.1 Grundlagen

Unix-Systeme sind klassisch in drei Schichten aufgeteilt, Geräte (Hardware), Systemkern und Prozesse (siehe Abbildung 3.1). Sämtliche Zugriffe auf Geräte erfolgen ausschließlich durch den Kern, der dafür auch die Gerätetreiber enthält.

Der Linux-Kern ist, soweit möglich, architekturunabhängig, d. h., Gerätetreiber werden durch standardisierte Schnittstellen, sogenannte virtuelle Treiber, angesprochen. Dieses betrifft auch Speicherverwaltung und Prozessoreigenschaften. Dabei ist der Kern monolithisch und läuft dementsprechend mit allen Teilen im unbeschränkten Systemmodus des Prozessors, während alle Prozesse im Benutzermodus ausgeführt werden.

Die in der verwendeten Kern-Version 2.0.30 unterstützten Architekturen sind Alpha, i386, 68000er (Macintosh etc.), Mips, Power-PC und Sparc. Die von mir im Rahmen dieser Arbeit vorgenommenen architekturabhängigen Änderungen beschränken sich zwar auf i386-PC, sind aber mit geringem Aufwand auf die anderen Architekturen übertragbar.

Eine Besonderheit von Linux sind die Kernmodule, die zur Laufzeit durch den Systemverwalter dem Kern hinzugefügt werden können, um dessen Funktionalität temporär zu erweitern. Fast alle Gerätetreiber sind als Modul konfigurierbar.

Der gesamte Kern und die meisten System- und Anwendungsprogramme sind im Quellcode frei verfügbar. Die verwendete Programmiersprache ist fast aussch-

[1] Für einen allgemeinen Einstieg in Unix siehe z. B. [Todino+96], für den Einstieg in Linux empfehle ich [SuSE97].

[2] Siehe [Beck+95] für eine ausführliche Beschreibung des Linux-Kerns.

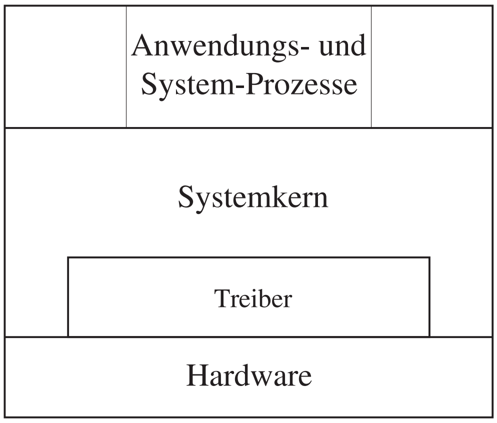

Abbildung 3.1: Klassische Schichtenaufteilung eines Unix-Systems

ließlich C, nur die gerätenahen Routinen sind in Assembler geschrieben.

3.2 Systemaufrufe

Alle Aktivitäten eines Prozesses außerhalb seines zugewiesenen Speicherbereiches erfordern Aufrufe von Kernroutinen, sogenannte Systemaufrufe. Dazu ist jeweils eine überwachte Umschaltung des Prozessormodus erforderlich. Die nachfolgende Tabelle 3.1 gibt einen Überblick über die vorhandenen Systemaufrufe.

Systemaufruf	Beschreibung
Prozeßverwaltung	
adjtimex()	Systemzeitwerte setzen und lesen
alarm()	Systemzeitgeber auf Alarmzeit setzen
brk()	Größe des nicht verwendeten Datensegmentes setzen
exit()	Beenden eines Prozesses
fork(), clone()	Erzeugen eines Kindprozesses als Kopie des aktuellen
getuid(), getgid(), geteuid(), getegid()	Reale (normale) und effektive Benutzer- bzw. Gruppen-Id des Prozesses auslesen
Fortsetzung auf der nächsten Seite...	

... Fortsetzung Linux-Systemaufrufe	
Systemaufruf	Beschreibung
getpid(), getppid, getpgid(), getpgrp()	Prozeß-Id des Prozesses und des Elternprozesses, dessen Gruppen-Id und Prozeßgruppe auslesen
setreuid(), setregid()	Reale (normale) und effektive Benutzer- bzw. Gruppen-Id des Prozesses ändern
setfsuid(), setfsgid()	Benutzer- bzw. Gruppen-Id des Prozesses für Dateisystemzugriffe ändern
setuid(), setgid()	Effektive und Dateisystem-Benutzer- bzw. -Gruppen-Id des Prozesses ändern, bei Benutzer root alle
getpriority(), setpriority()	Ausführungsprioritäten von Prozessen ändern
ioperm(), iopl()	Zugriffsrechte auf Hardwareports ändern
kill()	Senden eines Signals an einen Prozeß
modify_ldt()	lokale Deskriptortabelle ändern, nur für Windows-Emulation
create_module()	Hinzufügen eines Kernelmoduls vorbereiten
delete_module()	Entfernen eines Kernelmoduls
init_module()	Hinzufügen eines Kernelmoduls, erfordert vorheriges create_module()
get_kernel_syms()	Symboltabelle des Kerns auslesen
nice()	Ausführungspriorität des aufrufenden Prozesses ändern
pause()	Ausführung bis zum nächsten Signal unterbrechen
personality()	Ausführungsumgebung des aufrufenden Prozesses ändern
ptrace()	Ablaufkontrolle eines anderen Prozesses
reboot()	System herunterfahren und neu starten
setdomainname()	Domänennamen des Systems setzen
getgroups(), setgroups()	Gruppenrechte eines Prozesses auslesen oder setzen
sethostname()	Hostnamen des Systems setzen
Fortsetzung auf der nächsten Seite...	

... Fortsetzung Linux-Systemaufrufe	
Systemaufruf	Beschreibung
getitimer(), setitimer()	Systemzeitgeber für Prozeßüberwachung setzen
getrlimit(), setrlimit(), getrusage	Ressourcenschranken des aktuellen Prozesses auslesen und setzen
signal(), sigaction(), sigpending(), sigsuspend(), sgetmask(), ssetmask(), sigprocmask(), sigreturn()	Signalverhalten des Prozesses auslesen und setzen
sysinfo()	Systemauslastung auslesen
syslog()	System-Protokoll auslesen und verwalten
time(), stime(), gettimeofday(), settimeofday()	Systemzeit auslesen und setzen
times()	verbrauchte Systemzeit auslesen
uname()	Systeminformationen auslesen
vm86()	virtuellen 8086-Modus setzen (nur für Emulatoren)
wait4(), waitpid()	auf Ende eines Prozesses warten
Dateisystem	
access()	DAC-Rechte für ein Objekt auslesen
bdflush()	Ausschreiben eines Puffercache
chdir(),fchdir()	Arbeitsverzeichnis setzen
chmod(),fchmod()	DAC-Rechte für Objekt setzen
chown(), fchown()	Besitzer eines Objektes für DAC ändern
chroot()	Wurzelverzeichnis für Prozeß setzen
dup(), dup2()	Kopie des Dateideskriptors erzeugen
exec_ve()	Ausführen eines Folge-Prozesses mit der selben Prozeß-ID
fcntl()	Eigenschaften einer geöffneten Datei ändern
Fortsetzung auf der nächsten Seite...	

... Fortsetzung Linux-Systemaufrufe	
Systemaufruf	Beschreibung
ioctl()	Eigenschaften eines Gerätetreibers ändern
link(), symlink()	(symbolischen) Link im Dateisystem erzeugen
unlink()	Löschen von Dateien und Links
rename()	Datei umbenennen
rmdir()	Verzeichnis löschen
lseek(), llseek()	Position in der Datei ändern
mount()	Dateisystem einhängen
umount()	Dateisystem aushängen
creat()	Neue Datei erzeugen
open()	Vorhandene Datei öffnen, dabei evt. neu erzeugen oder leeren
mkdir()	Verzeichnis anlegen
mknod()	Pseudodatei erzeugen, z.B. für Geräte in /dev
close()	Datei schließen (Hilfsnachricht, keine Entscheidung)
pipe()	Pseudodatei als Übertragungskanal erzeugen
read(), write()	Lesen und Schreiben einer geöffneten Datei
readdir()	nächsten Verzeichniseintrag lesen
getdents()	Verzeichniseinträge aus angegebenem Verzeichnis lesen
readlink()	symbolischen Link verfolgen
select()	Multiplexen von Ein- und Ausgabeoperationen über Wartemechanismen
stat(), fstat(), lstat(), new_stat(), new_fstat(), new_lstat()	Dateiverwaltungsinformationen auslesen
statfs(), fstatfs()	Verwaltungsinformationen für das Dateisystem der angegebenen Datei auslesen
sync(), fsync()	Ausschreiben aller Puffercaches
sysfs()	Auslesen aller bekannten Dateisysteme
Fortsetzung auf der nächsten Seite...	

... Fortsetzung Linux-Systemaufrufe	
Systemaufruf	Beschreibung
truncate(), ftruncate()	Dateilänge ändern
uselib()	Laufzeitbibliothek laden
umask()	Standard-DAC-Rechte für neue Dateien und Verzeichnisse setzen
utime()	Zeitstempel einer Datei setzen
utimes()	Zeitstempel einer Datei setzen
vhangup()	Terminal zurücksetzen
Kommunikation	
ipc()	Komplette Verwaltung der Interprozeßkommunikation nach System V
socketcall()	Verwaltung der Interprozeßkommunikation über Sockets
Speicherverwaltung	
mmap(), munmap()	Dateieinblendung in Speicher (de)aktivieren
mprotect()	Speicher-Zugriffsschutz ändern
swapon(), swapoff()	Auslagerungsspeicher (de)aktivieren
Initialisierung	
idle()	Leerlauf, nur für Idle-Prozeß
setup()	Dateisysteme initialisieren und Dateisystem für Hauptverzeichnis einbinden

Tabelle 3.1: Linux-Systemaufrufe

3.3 Verzeichnisse der Kernquellen

Wie in Abbildung 3.2 ansatzweise zu sehen, sind die funktionellen Gruppen des Systemkerns in den Quellen ziemlich fein gegliedert. Wichtig ist vor allem die Aufteilung zwischen den architekturabhängigen Quellen unter dem Verzeichnis `arch` und den architekturunabhängigen Quellen in den übrigen Verzeichnissen.

Im Verzeichnis `include` befinden sich die C-Header-Dateien für den gesamten architekturunabhängigen Teil des Kern. Die architekturabhängigen Header-Dateien sind im Unterverzeichnis `asm`, das einen Unix-Link auf das jeweilige Architekturunterverzeichnis darstellt, z. B. `asm-i386`.

`fs` enthält das Linux-eigene virtuelle Dateisystem und zur Zeit 17 Unterverzeichnisse mit implementierten Dateisystemen, `kernel` zentrale Bestandteile wie das Prozeßmanagement und `mm` die Speicherverwaltung. Die Bedeutung der übri-

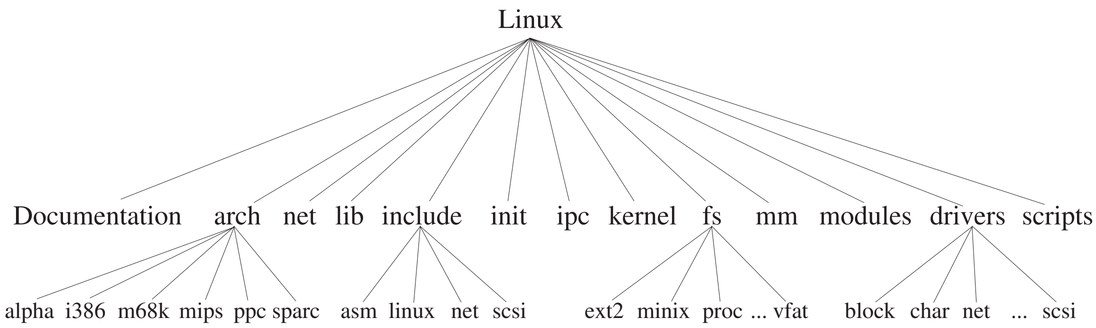

Abbildung 3.2: Verzeichnisstruktur der Kernquellen

gen Verzeichnisse ergibt sich aus den jeweiligen Namen.

Bei der Bezeichnung von Code-Teilen ist es unter Linux üblich, relative Pfade ab dem Quell-Hauptverzeichnis anzugeben, bei Header-Dateien ab dem `include`-Verzeichnis. In dieser Arbeit wird diese Konvention ebenfalls verwendet.

3.4 Ablauf des Systemstarts

Beim Start eines PC wird vom BIOS der erste Sektor der Festplatte geladen und ausgeführt. Dieser lädt den ersten Sektor einer ausgewählten Partition, der ebenfalls direkt gestartet wird und das jeweilige Betriebssystem hochfährt.

Linux wird in der Regel durch den Linux-Loader (LILO) gestartet, der sich in beiden erwähnten Sektoren befinden kann. Dieses Programm bekommt bei der Einrichtung die Position des Linux-Kerns auf der Festplatte mitgeteilt und kann ihn deshalb direkt laden und ausführen. Außerdem ist es möglich, mehrere verschiedene Kerne und sogar DOS und andere Systeme über ein einfaches Menü zu starten.

Der Systemkern ermittelt und initialisiert zunächst die wichtigsten Gerätekomponenten, um dann in den "Protected Mode" des Prozessors umzuschalten. Erst dieser ermöglicht direkt virtuelle Speicherverwaltung, Kern- und Benutzer-Modus, lineare Speicheradressierung, 32-Bit-Code und einiges mehr. Nach einer weiteren Stufe der Geräteerkennung kann schließlich die Ausführungsumgebung für C-Code aktiviert und die erste C-Funktion gestartet werden. Aus diesem Grund sind alle bisher beschriebenen Funktionen direkt in Assembler programmiert.

Die erste C-Funktion start_kernel() aus der Datei `init/main.c` sichert die Erkennungsdaten der Assembler-Routinen und initialisiert die restlichen Kernkomponenten. Der laufende Kern wird zum ersten Prozeß mit der Nummer 0 und startet den Prozeß Nummer 1, den Init-Prozeß. Danach geht der Kern-Prozeß in eine Endlos-Schleife (Idle-Prozeß), um künftig mit niedrigster Priorität nicht benötigte Prozessorzeit aufzubrauchen.

Der Init-Prozeß verwendet den Systemaufruf setup(), um die restlichen Gerätetreiber zu initialisieren und das Dateisystem des Hauptverzeichnisses einzubinden (root mount). Dann wird eine Verbindung zur Konsole hergestellt und eines der Programme `/etc/init`, `/bin/init` oder `/sbin/init` ausgeführt, das alle anderen benötigten Prozesse des Systems startet. Ist dieses nicht möglich, wird stattdessen versucht, die Shell `/bin/sh` mit dem Shell-Skript `/etc/rc` zu starten, um das System vom Systemverwalter reparieren zu lassen. Der Init-Prozeß ist, wie der Kern-Prozeß, während der gesamten Laufzeit des Systems aktiv.

Ab dem Start des Init-Prozesses können bei Bedarf zusätzliche Kernmodule für weitere Funktionalitäten nachgeladen werden. Dieses kann später durch den Kernel-Daemon, einen weiteren Prozeß, automatisiert werden.

3.5 Kernkonfiguration

Da die Kernquellen jedem Linux-Administrator zur Verfügung stehen, ist es möglich, den Kern gezielt an die eigenen Bedürfnisse anzupassen. Dafür gibt es drei verschiedene Konfigurationsprogramme, eines mit einfachen Abfragen, eines menügeführt im Textmodus und eines mit grafischer X-Oberfläche. Ich werde mich hier auf das Menü im Textmodus beschränken.

Zur Konfiguration ist im Hauptverzeichnis der Kernquellen der Befehl `make menu_config` einzugeben, der das Menüprogramm erzeugt und aufruft. Dort können viele veränderbare Parameter des Kern, insbesondere die einzubindenden Gerätetreiber, eingestellt werden. Viele Kernbereiche lassen sich sowohl fest, als auch in Modul-Form hinzufügen.

Nach Speicherung der Einstellungen kann mit `make dep && make clean && make zImage` ein neuer Systemkern erzeugt werden. Die gewählten Module lassen sich mit `make modules && make modules_install` übersetzen und installieren. Anschließend ist der neue Kern aus dem Verzeichnis `arch/i386/boot` in das Hauptverzeichnis zu kopieren und dem Linux-Loader bekanntzugeben. Beim nächsten Systemstart steht der neue Kern bereits zur Verfügung.

3.6 Sicherheitsbewertung

In diesem Unterkapitel wird eine informelle Darstellung und Bewertung der Sicherheitseigenschaften eines Standard-Linuxsystems nach den Funktionalitäts-Oberbegriffen der ITSEC vorgenommen.[3] Die Vertrauenswürdigkeit kann hier nur grob eingeschätzt werden.

---

[3] Siehe Abschnitt 2.1.3.

3.6.1 Funktionalität

Identifizierung und Authentisierung

Identifizierung und Authentisierung erfolgen durch Benutzernamen und Paßworte. Die zulässigen Benutzer sind in der Datei `/etc/passwd`, die Paßworte in verschlüsselter Form in `/etc/shadow` abgelegt. Da viele Programme auf die erste Datei zugreifen, muß diese für alle Benutzer lesbar sein, die zweite wird dagegen nur vom Systemverwalter verwendet. Die Sicherung geschieht aber durch Administration, nicht durch das System.

Die Authentisierung erfolgt durch Prozesse im Besitz des Systemverwalters. Durch dieses Verhalten kann jeder Prozeß, der Systemverwalterrechte bekommt, anderen Benutzern Zugang gewähren und sogar die Identifizierungs- und Authentisierungsdaten ändern. Dieses ist eine deutliche Sicherheitslücke.

Zugriffskontrolle

Die Zugriffskontrolle für die meisten Ressourcen erfolgt nach dem diskreten Modell, das in Unterkapitel 2.2 beschrieben wurde. Die Rechte des Systemverwalters sind nicht einschränkbar, aber Voraussetzung für die meisten administrativen Tätigkeiten. Deshalb müssen viele Aufgaben mit uneingeschränkten Rechten erledigt werden.

Außerdem ist durch die rein diskrete Rechtevergabe keine organisatorische Kontrolle von Datenströmen, aber Angriffe durch bösartige Programme möglich. Diese Zugriffskontrolle halte ich in vielen Fällen für unzureichend.

Der Zugriff auf den Adreßbereich eines anderen Prozesses ist jedoch ohne ausdrückliche Freigabe als gemeinsamer Speicher nicht möglich.

Beweissicherung

Die Beweissicherung erfolgt durch die Prozesse, die eine Aufgabe ausführen, durch sogenanntes "logging" in Dateien auf Anwendungsebene. Nur für wenige Bereiche, die vor allem Treiber- und Netzwerkleistungen betreffen, kann eine Protokollierung durch den Kern vom Systemverwalter aktiviert werden.

Protokolliert wird entweder über Kernmechanismen mit Hilfe eines Hintergrundprozesses oder direkt in programmspezifische Dateien.

Protokollauswertung

Die Auswertung erfolgt programmabhängig, da kein einheitliches Format existiert. Es gibt für einige Protokolltypen Hilfsprogramme zur Auswertung.

Wiederaufbereitung

Von Prozessen angeforderter Speicher wird stets zuerst gelöscht. Der Lesezugriff auf gespeicherte Daten ist erst nach entsprechendem Schreiben möglich. Nur bei Freigabe des direkten Zugriffes auf Speichermedien können diese ohne vorheriges Schreiben gelesen werden.

Unverfälschtheit

Es gibt keine Systemmechanismen zur Gewährleistung der Integrität von Daten außer der korrekten Speicherung auf Datenträger. Die Unverfälschtheit ist somit ausschließlich von der Korrektheit der verwendeten Programme und der diskreten Zugriffskontrolle abhängig.

Zuverlässigkeit der Dienstleistung

Linux enthält als Echtzeitmechanismus lediglich die garantiert zügige Abarbeitung von Unterbrechungsanforderungen durch Geräte. Die unnötige Blockade von Betriebsmitteln durch Prozesse ist nicht beschränkt, diese können aber jederzeit beendet und die Betriebsmittel wieder freigegeben werden. Außerdem können für Prozesse obere Schranken für den Verbrauch der meisten Ressourcen festgelegt werden, um das Gesamtverhalten zu verbessern.

Sicherung und Wiederherstellung von Daten sind Aufgabe des Systemverwalters bzw. der die Daten verarbeitenden Programme.

Übertragungssicherung

Linux unterstützt verschiedene Übertragungsprotokolle mit sehr unterschiedlichen Eigenschaften. Die mit Abstand häufigste Verwendung findet zur Zeit noch die TCP/IP-Protokollfamilie, die bekanntermaßen recht unsicher ist. Für die gesicherte Übertragung sind etliche Zusatzprodukte auf Prozeßebene erhältlich, die den gesamten Sicherheitsbereich abdecken.[4]

Schutz von personenbezogenen Benutzerdaten

Die Benutzerdaten werden nur über die diskrete Zugriffskontrolle geschützt. Anonyme und pseudonyme Zugänge sind nicht vorgesehen, könnten aber durch Administration eingerichtet werden. Die verwendeten Ressourcen sind für andere Benutzer eingeschränkt beobachtbar und verknüpfbar.

[4] Beispiele sind die Secure Shell (SSH), die eine sichere Verschlüsselung der Übertragung mit Sitzungsschlüsseln und RSA-Authentisierung von Benutzern und Rechnern bietet, oder das verschlüsselnde Netzwerkdateisystem crypt.

Schutz von personenbezogenen Daten anderer Personen

Dieser Aspekt der Privatheit wird nicht unterstützt. Das Besitzer-Prinzip der diskreten Zugriffskontrolle ist für personenbezogene Daten ungeeignet, da der Besitzer der Daten der Betroffene sein muß, der aber in der Regel nicht als Benutzer eingetragen ist.

3.6.2 Vertrauenswürdigkeit

Die Wirksamkeit der Maßnahmen kann aufgrund fehlender Sicherheitsanforderungen leider nicht eingeschätzt werden.

Die Korrektheit der Implementation der genannten Funktionalität erscheint mir im Kernbereich aufgrund der hohen Verbreitung, der Verfügbarkeit der Quellen und der umfangreichen Untersuchungen und Tests subjektiv recht hoch, ein informeller oder gar formeller Nachweis der Korrektheit ist mir jedoch nicht bekannt. Vor allem der formelle Nachweis dürfte wegen der monolithischen Kernstruktur nicht mit vertretbarem Aufwand durchführbar sein.

Kapitel 4

Generalized Framework for Access Control (GFAC)

4.1 Allgemeines Modell

4.1.1 Motivation

Das Generalized Framework for Access Control wurde zum ersten Mal 1990 auf der 13. Nationalen Computer-Sicherheits-Konferenz vorgestellt[Abrams+90]. Der Anlaß für seine Entwicklung war die zunehmende Menge an inflexiblen Sicherheitsmodellen und Implementationen, die zwar viele, aber bei weitem nicht alle Anforderungen erfüllen konnten. Außerdem stiegen mit zunehmender Komplexität der Betriebssysteme der Aufwand, Änderungen der Sicherheitspolitik in die Systeme hineinzuprogrammieren, und die Wahrscheinlichkeit von sicherheitsrelevanten Fehlern.

Benötigt wurde deshalb eine Möglichkeit, Sicherheitsmechanismen mit wenig Aufwand flexibel an die jeweiligen Bedürfnisse anpassen zu können. Anstelle von Kodierung sollte Konfiguration die Anpassung ermöglichen. Durch gezielte Kombination von für Teilbereiche anerkannt sicheren Modellen war sogar ein Zuwachs an Sicherheit ereichbar.

Durch sinnvolle Gliederung sollten außerdem mögliche Fehlerquellen besser lokalisierbar und vermeidbar sein.

4.1.2 Aufbau

Das Generalized Framework for Access Control teilt Zugriffskontrolle allgemein in die vier Komponenten Zugriffskontrollinformationen, Kontext, Autoritäten und Regeln. Zugriffskontrolle ist dabei die Anwendung der Regeln auf Basis der von den Autoritäten festgelegten Zugriffskontrollinformationen und des jeweiligen Kontextes.

Die Zugriffskontrolle ist bei jedem Zugriff eines Subjektes auf ein Objekt, Zugriffskontrollinformationen oder den Kontext durchzuführen.

4.1.3 Zugriffskontrollinformationen (Access Control Information, ACI)

Zugriffskontrollinformationen sind stets an Subjekte oder Objekte gebunden. Sie umfassen alle diejenigen ihrer Daten, die von den Regeln zur Entscheidungsfindung verwendet werden.

Subjekt-spezifische ACI beinhalten z. B. Identifizierungs- und Authentisierungsdaten, Sicherheitsklassifikation, Aufgaben und Rollen im System und vieles mehr.

Beispiele für Objekt-spezifische ACI sind Identifizierung, Sicherheitslevel, Erzeuger, zugelassene Verarbeitungs-Programme und individuelle Zugriffskontrollisten (Access Control Lists, ACL), die eine subjektbezogene Zugriffsverwaltung erlauben.

4.1.4 Kontext (Access Control Context, ACC)

Als Kontext werden alle Informationen bezeichnet, die zwar von den Regeln verwendet werden, aber weder Subjekten noch Objekten zuzuordnen sind.

Zum Kontext gehören Systemdaten, wie Zeit, Status etc., aber auch Objekt-Definitionen, die nur im Rahmen des Sicherheitssystems benötigt werden, wie Benutzergruppen, Aufgaben oder Rollen.

4.1.5 Regeln (Access Control Rules, ACR)

Die Regeln definieren die Politik der Zugriffskontrolle. Im Gegensatz zur gängigen Verwendung des Begriffs umfassen die Regeln im GFAC nicht die Systemverwaltungsfunktionen des vertrauenswürdigen Systemkerns (Trusted Computing Base, TCB), sondern lediglich die durchzusetzenden Entscheidungsvorschriften.

Daraus ergibt sich eine Zweiteilung der Systemfunktionen in einen politikabhängigen Bereich, der Zugriffentscheidungen anhand der Regeln durchführt, und einen politikunabhängigen Bereich, der für die Verwaltung des Systems zuständig ist. Diese klare Trennung der Aufgaben ermöglicht, durch einfache Regelanpassung die Sicherheitspolitik zu ändern, ohne den politikunabhängigen Bereich zu modifizieren.

Der Zugriff eines Subjektes auf ein Objekt im Modus M wird nur dann gestattet, wenn die Zugriffskontrollinformationen von Subjekt und Objekt sowie der aktuelle Kontext die Regeln erfüllen.

Mehrere zusammengehörige Regeln können zu einem Regelsatz zusammengefaßt werden. Beispiele für einfache Regeln sind Zeitbeschränkungen oder das simple-security-property des Bell-La Padula-Modells, Beispiele für Regelsätze sind Umsetzungen des Bell-La Padula-Modells oder des Clark-Wilson-Integritätsmodell.

Bei Erstellung und Anwendung der Regeln sind folgende Punkte zu beachten:

- Kombination von Regeln: Sobald mehrere Regeln anwendbar sind, muß eine eindeutige logische Verknüpfung oder eine Präzedenz dieser Regeln definiert sein.
- Erzeugung von Objekten und Vererbung: Für die Erzeugung neuer Subjekte und Objekte sind Regeln notwendig, die die zu vergebenden ACI durch Standardwerte oder Vererbung festlegen.
- Kombination und Konfiguration von Regelsätzen: Obwohl beliebige Regeln durch die Autoritäten definiert werden können, sind in der Praxis zertifizierte Regelsätze nur unter Verlust der Zertifizierung veränderbar. Im GFAC können solche Regelsätze durch bedarfsabhängige Kombination und Konfiguration ohne Sicherheitsbeschränkung an individuelle Anforderungen angepaßt werden.

4.1.6 Autoritäten (Access Control Authorities, ACA)

Autoritäten sind spezielle Regeln, die gezielt den Zugriff auf alle vier Komponenten des GFAC festlegen. Diese Regeln bilden den zentralen Punkt für die Wirksamkeit eines Zugriffskontrollsystems.

Durch die Autorisierung von Autoritäten ergeben sich mehrstufige Autoritätsbäume, die als Blätter entweder extern oder mit politikabhängigen Verfahren festgelegte oberste Instanzen der Autorität enthalten.[1]

4.1.7 Andere Ansätze

Parallel zum Generalized Framework for Access Control wurde die Problematik, mehrere Sicherheitsmodelle miteinander zu verbinden und Übergänge zu schaffen, von einer Arbeitsgruppe um Hilary Hosmer untersucht[Hosmer92-1, Hosmer92-2].

Der gewählte Lösungsansatz ist ähnlich. Auch Hosmer trennt die Durchsetzung der Zugriffskontrolle von der Entscheidung. Alle aktiven Politiken entscheiden unabhängig voneinander. Die Einzelergebnisse werden anschließend mit Hilfe einer sogenannten Metapolitik zu einer Gesamtentscheidung zusammengesetzt.

Da auch diese Metapolitik konfigurierbar ist, ergibt sich die Möglichkeit, die Einzelpolitiken attributabhängig zu gewichten. Dieses ermöglicht z.B. die Weitergabe von Daten *inklusive* der zugehörigen Sicherheitspolitik oder den Einsatz von einander widersprechenden Sicherheitspolitiken innerhalb eines Systems.

[1] Siehe Beispiel in [Abrams+90]

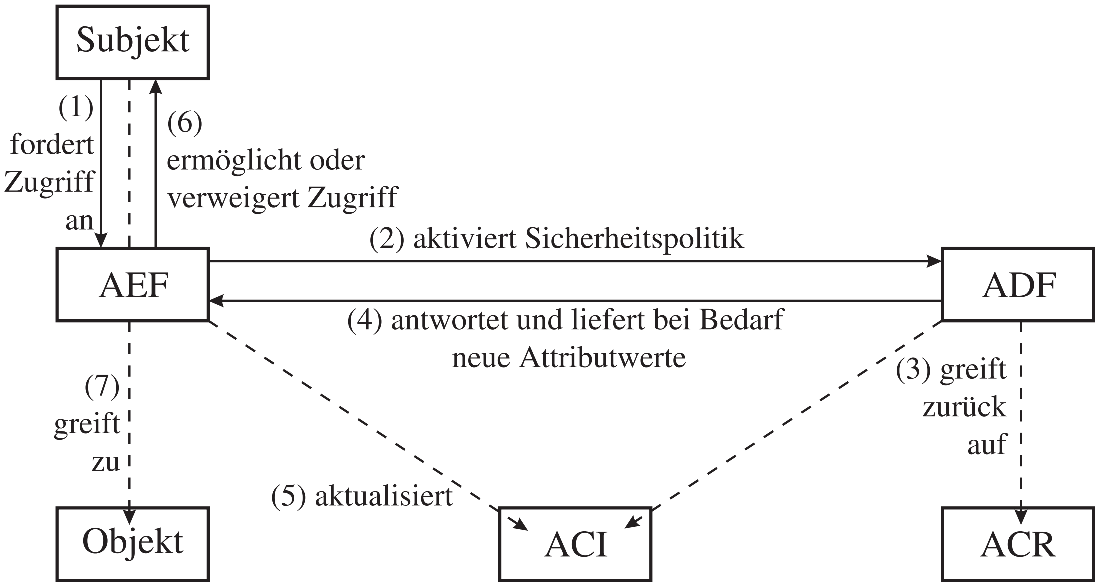

Abbildung 4.1: Funktionale Komponenten des "Generalized Framework for Access Control" nach La Padula

Selbstverständlich gestattet auch das Generalized Framework for Access Control die Festlegung von Regeln zur Konfliktbewältigung und damit alle eben genannten Möglichkeiten.

4.2 Ansatz von La Padula

La Padula spezifizierte ansatzweise die Umsetzung des Generalized Framework for Access Control in einem Unix-System[LaPadula95]. Dazu verwendete er die Sicherheitsmodelle klassische mandatorische Zugriffskontrolle (Bell-La Padula) nach System V/MLS (MAC), Clark-Wilson-Integritäts-Modell (CWI), funktionale Kontrolle (FC) und Änderung der Sicherheitsinformationen (SIM) in teilweise vereinfachter Form.

4.2.1 Komponenten

La Padula bezeichnet den politikabhängigen, die Regelsätze enthaltenden Bereich des vertrauenswürdigen Systemkerns als *Entscheidungskomponente* (Access Control Decision Facility, ADF). Den Teil des politik*un*abhängigen Bereichs, der bei jeder sicherheitsrelevanten Systemanfrage die Entscheidung veranlaßt und durchsetzt, nennt er entsprechend die *Durchsetzungskomponente* (Access Control Enforcement Facility, AEF).

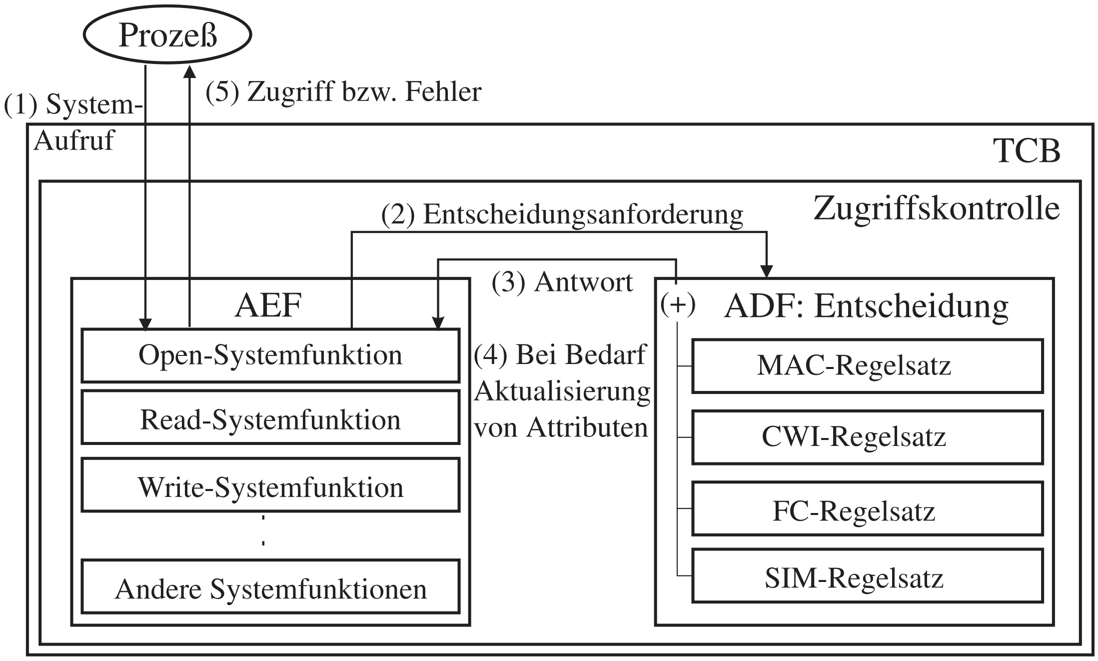

Abbildung 4.2: Ablauf der Zugriffsentscheidung im Unix-Kern nach La Padula

Das logische Zusammenspiel der funktionalen Systemkomponenten findet sich in Abbildung 4.1. Die Abbildung 4.2 zeigt zum Vergleich den angepaßten Ablauf einer Zugriffsentscheidung in einem Unix-System.

Diese beginnt mit einem Systemaufruf an die Trusted Computing Base (TCB), den Systemkern. Dort wird innerhalb des Durchsetzungscodes der zugehörigen Bearbeitungsroutine eine Anforderung an die Entscheidungskomponente generiert. Diese Anfrage durchläuft alle aktiven Regelsätze, die Gesamtentscheidung wird zusammengesetzt und mit neuen Attributwerten an die Durchsetzung zurückgegeben. Je nach Ergebnis veranlaßt diese entweder die entsprechende Attributsetzung und ermöglicht den Zugriff oder liefert eine Fehlermeldung an den aufrufenden Prozeß.

4.2.2 Anpassungen der Sicherheitsmodelle für die Entscheidungskomponente

Klassische mandatorische Zugriffskontrolle

In diesem Regelsatz werden die Kategorien des Bell-La Padula-Modelles nicht unterstützt. Zugriffsentscheidungen sind lediglich von der Sicherheitsklassifikation abhängig, die hier somit direkt als Sicherheitslevel fungiert.

Abweichend von System V/MLS werden dem Bell-La Padula-Modell andere

Anforderung	Bedingung für Zugriff
create	O bekommt Level von S
delete	P = O
delete-data	P = O
execute	P $\geq$ O
read	keine Bedingung
read-open	P $\geq$ O
read&write-open	P = O
write	keine Bedingung
write-open	P = O

Tabelle 4.1: MAC-Zugriffsbedingungen auf Dateien

Anforderung	Bedingung für Zugriff
create	O bekommt Level von S
delete	P = O
read	P $\geq$ O
search	P $\geq$ O
write	P = O

Tabelle 4.2: MAC-Zugriffsbedingungen auf Verzeichnisse

Zugriffsmodi zugefügt:

- Erzeugen (create)
- Löschen (delete)
- Daten Löschen (delete-data)
- Öffnen zum Lesen (read-open)
- Öffnen zum Schreiben (write-open)
- Öffnen zum Lesen und Schreiben (read&write-open)
- sowie einige Zugriffe auf die Benutzer- und Rechteverwaltung

Die Tabellen 4.1 bis 4.5 spezifizieren die Bedingungen für positive Zugriffsentscheidungen des MAC-Regelsatzes, unterschieden nach dem jeweiligen Typ des Objektes. Dabei bezeichnet P den Prozeß, O das Objekt und $\geq$ bzw. = stehen für "dominiert" bzw. "hat gleichen Level". Da Lesen und Schreiben auf Dateien

Anforderung	Bedingung für Zugriff
alter	P = O
create	O bekommt Level von S
delete	P = O
read	keine Bedingung
read&write-open	P = O
write	keine Bedingung

Tabelle 4.3: MAC-Zugriffsbedingungen auf Interprozeßkommunikationskanäle

Anforderung	Bedingung für Zugriff
change-owner	P = O
create	O bekommt Level von S
delete	P = O
get-permissions-data	P >= O
get-status-data	P >= O
modify-access-data	P = O
modify-permissions-data	P = O

Tabelle 4.4: MAC-Zugriffsbedingungen auf Systemkontrollinformationen

Anforderung	Bedingung für Zugriff
clone	P2 bekommt Level von P1
send-signal	P1 = P2

Tabelle 4.5: MAC-Zugriffsbedingungen beim Prozeßmanagement

Rolle	Programm	Erzeugen oder Löschen	Ändern
TP-Anwender	TP		CDI
TP-Manager	TPICD	TP, TPICD, CDIIC	CDIIC
IVP-Anwender	IVP		
IVP-Manager		IVP, CDI	

Tabelle 4.6: Aufgaben der Clark-Wilson-Rollen

und Kommunikationskanäle nur nach entsprechendem Öffnen möglich sind, wird hier keine Einschränkung vorgenommen. Lesen und Schreiben auf Verzeichnisse stehen für Zugriffe auf Einträge, ein Öffnen ist nicht möglich.

Clark-Wilson-Modell

Benutzer, Programme und Objekte werden durch Attribute jeweils in mehrere Kategorien eingeteilt. Ein Benutzer ist TP-Anwender, TP-Manager, IVP-Anwender, IVP-Nutzer oder ohne Modellrelevanz, während ein Programm TP, IVP, Zugriffskontrolldaten-TP (TPICD) oder unkontrolliert sein kann. Objekte werden als CDI, Zugriffskontroll-CDI und UDI unterschieden. Tabelle 4.6 zeigt die Zuordnung von Integritätsrollen zu Programmtypen und Datentypen.

Festgelegt werden außerdem Regeln, welche Benutzer welche TPs auf welche CDIs und welche Benutzer welche IVPs auf welche CDIs anwenden dürfen. Um illegale Zugriffskombinationen zu verhindern, wird für jede Regel protokolliert, welche Prozesse aufgrund dieser Regel ihren ersten Zugriff auf ein CDI haben. Jeder weitere Zugriff auf andere CDI wird nur gestattet, wenn der Prozeß in der Tabelle bei einer passenden Regel eingetragen ist. Gleichzeitig werden die Prozeßeinträge um nicht mehr zutreffende Kombinationen reduziert.

Ein Prozeß, der ein Programm ausführt, bekommt dessen Programmtyp zugewiesen. Prozesse der Typen TP, IVP und TPICD dürfen keine Prozeßkopien von sich erzeugen (Systemaufrufe clone und fork) und auch nicht durch andere Prozesse überwacht werden. Das empfangen von Signalen ist zulässig.

Dateien der Typen TP, IVP, TPICD und CDI dürfen nicht den Besitzer wechseln und nur von berechtigten Benutzern umbenannt, mit weiteren Namen versehen oder im Status eingesehen werden.

Natürlich können hier lediglich die Durchsetzungsregeln von Clark und Wilson umgesetzt werden, die Zertifizierungsregeln sind alleinige Aufgabe der Organisation und Administration.

A	B	Ergebnis
Ja	Ja	Ja
Ja	Nein	Nein
Ja	Egal	Ja
Ja	Undefiniert	Undefiniert
Nein	Nein	Nein
Nein	Egal	Nein
Nein	Undefiniert	Undefiniert
Egal	Egal	Egal
Egal	Undefiniert	Undefiniert
Undefiniert	Undefiniert	Undefiniert

Tabelle 4.7: Symmetrische logische Verknüpfung “und-plus” der Einzelentscheidungen nach La Padula

Funktionale Kontrolle

Die funktionale Kontrolle wird bei La Padula auf die Rollen allgemeiner Benutzer, Sicherheitsbeauftragter und Systemadministrator, sowie auf die Objektkategorien allgemeines, Sicherheits- und Systemobjekt beschränkt. Zugriffe werden nicht nach ihrer Art unterschieden. Allgemeiner Benutzer ist kompatibel mit allgemeines Objekt, Sicherheitsbeauftragter und Systemadministrator zusätzlich mit Sicherheits- bzw. Systemobjekt.

Veränderung von Sicherheitsinformationen

Es werden hier nur die Rolle Sicherheitsbeauftragter und der Datentyp Sicherheitsinformation (si) verwendet. Auf Daten dieses Typs darf nur ein Sicherheitsbeauftragter schreibend zugreifen.

4.2.3 Durchsetzung der Zugriffskontrolle

Die Durchsetzung der Zugriffskontrolle erfolgt durch die Erweiterung aller sicherheitsrelevanten Systemaufrufe um einen Aufruf der Entscheidungskomponente und entsprechende Ablehnung oder Gewährung des Zugriffs.

4.2.4 Entscheidungsverfahren

Jeder Regelsatz für eines der Modelle wird einzeln aufgerufen und liefert als Ergebnis einen der Werte “Ja”, “Nein”, “Egal” oder “Nicht definiert”. Die Einzelentscheidungen werden anschließend mit der in Tabelle 4.7 angegebenen symmetrischen logischen Verknüpfung “und-plus” zu einem Gesamtergebnis zusammengesetzt, das der Durchsetzungskomponente zurückgegeben wird. Zusätzlich können von jeder Regel noch Anpassungen der Sicherheitsattribute vorgenommen werden. Der Ablauf der Entscheidung wurde bereits in Abbildung 4.2 auf Seite 38 dargestellt.

Kapitel 5

Umsetzung des “Generalized Framework for Access Control” in Linux

5.1 Entwurfsziele

Zusätzlich zum Hauptziel des Entwurfes, das “Generalized Framework for Access Control” in Linux umzusetzen, wurden noch einige weitere Ziele berücksichtigt:

- Unabhängigkeit aller verwendeten Sicherheitsmodelle, d. h., jedes Sicherheitsmodell sorgt für die Sicherheit der eigenen Administration
- Wahlweise An- und Abschaltbarkeit von Modulen zur Laufzeit unter deren eigener Kontrolle
- Vergabe von Benutzerpseudonymen und umfangreiche, aber pseudonyme Beweissicherung in einheitlichem Format
- Unterstützung möglichst vieler verschiedener Dateisysteme durch ausschließliche Verwendung des virtuellen Dateisystems und Speicherung der Zugriffskontroll- und Kontextdaten als Dateien in einem modellübergreifend besonders geschützten Verzeichnis
- Weitestgehende Kapselung aller funktional trennbaren Teile des Systems, insbesondere der funktionalen Komponenten, mit ausschließlichem Zugriff über wohldefinierte Funktionen
- Bei höchstmöglicher Sicherheit möglichst geringer Verlust an Effizienz, Stabilität und Flexibilität des Systems
- Leichte Erweiterbarkeit um zusätzliche Sicherheitsmodelle

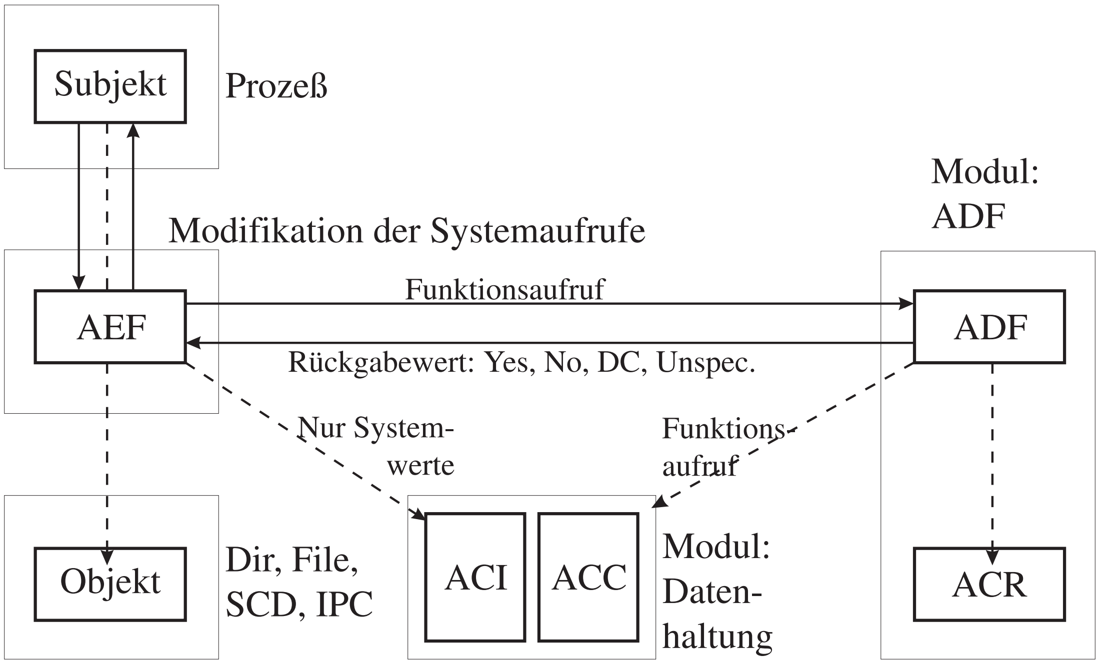

Abbildung 5.1: Umsetzung des “Generalized Framework for Access Control” in Module

5.2 Gliederung

Wie in Abbildung 5.1 zu sehen, wird das Gesamtmodell in drei Teile gegliedert: Durchsetzung (AEF), Entscheidung (ADF) und Datenhaltung.

Die Struktur des Linuxkerns erzwingt dabei, die Durchsetzung durch Anpassung aller Systemaufrufe zu implementieren, während Entscheidung und Datenhaltung in eigenständige Module gekapselt werden können. Aus Gründen des Aufwandes werde ich mich auf die Anpassung der für die implementierten Modelle sicherheitsrelevanten Systemaufrufe beschränken.

5.3 Ablauf der Zugriffskontrolle

Jeglicher Zugriff auf Ressourcen wird durch die Durchsetzungskomponente abgefangen. Diese läßt die Berechtigung durch eine Anforderung an die Entscheidungskomponente überprüfen, welche als Entscheidungsgrundlage wiederum die zu Subjekt und Objekt gehörigen Attribute und die Kontextinformationen von der Datenhaltung anfordert.

Die diskrete Zugriffskontrolle bleibt erhalten und wird stets zuerst durchgesetzt. Erst, wenn diese den Zugriff genehmigt, erfolgt die Überprüfung durch die anderen Sicherheitsmodelle. Ihre Administration unterliegt ebenfalls der erwei-

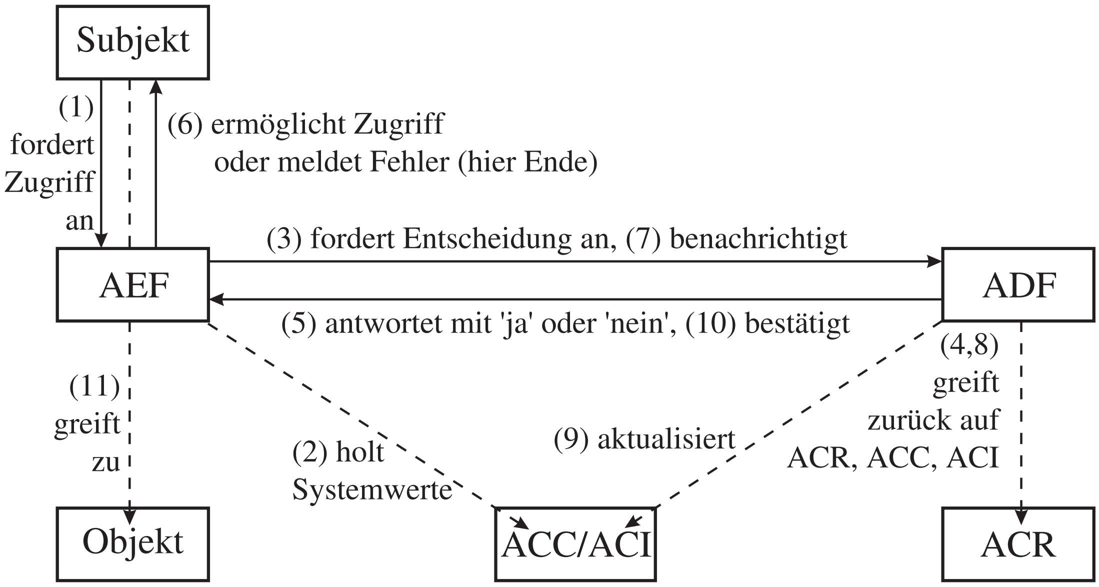

Abbildung 5.2: Angepaßter Ablauf der Zugriffskontrolle

terten Zugriffskontrolle.

Da trotz einer positiven Entscheidung der Zugriff auf eine Ressource fehlschlagen kann, dürfen die Zugriffskontrollinformationen und der Kontext erst nach erfolgreicher Freigabe angepaßt werden. Dafür ist ein weiterer Aufruf der Entscheidungskomponente notwendig. Es ergibt sich der in Abbildung 5.2 dargestellte Ablauf.

5.4 Datenhaltung

Das Datenhaltungsmodul ist für Speicherung, Konsistenzerhaltung und Verfügbarkeit sämtlicher Subjekt- und Objektattribute, d.h., der Zugriffskontrollinformationen (Access Control Information, ACI), und für den Kontext, speziell modellspezifische Zusatzlisten, zuständig. Zugriffe auf diese Daten sind nur über definierte Schnittstellen möglich. Das Modul ermöglicht parallele, reentrante Aufrufe durch interne Synchronisation. Daten werden stets parallel im Hauptspeicher und auf der Festplatte gehalten.

Aufgrund der Unix-Eigenschaft, zur Laufzeit Dateisysteme hinzuzufügen oder zu entfernen, muß die Attributverwaltung für Verzeichnisse und Dateien für jedes Dateisystem unabhängig erfolgen.

Attribut	Politik	Mögliche Werte
Benutzer-Bezeichner	Alle	Ganze Zahl (Systemabhängig)
Benutzer-Pseudonym	Alle	Ganze Zahl (Systemabhängig)
Sicherheitslevel	MAC	unclassified, confidential, secret, top secret
Systemrolle	MAC, FC, SIM	user, security officer, administrator
Integritätsrolle	CWI	NIL, TP-user, TP-manager, IVP-user
CWI-Relationen	CWI	Listen-Bezeichner
Datenschutzrolle	PM	user, sec admin, data protection officer, TP manager, system admin
Autorisierte Aufgaben	PM	Aufgabenlisten-Bezeichner

Tabelle 5.1: Benutzerattribute

5.4.1 Datenumfang

Die Datenhaltung umfaßt stets alle Attribute aller Resourcen, die von den implementierten Sicherheitspolitiken benötigt oder kontrolliert werden. Außerdem werden hier diejenigen Relationen verwaltet, die durch Konfigurationen veränderbar sind, z. B. die Zugriffsrelationen des Clark-Wilson-Modelles. Unveränderliche Relationen werden in der Entscheidungskomponente fest einkodiert.

Allgemeine Datenhaltung

In der allgemeinen Datenhaltung werden diejenigen Attribute für alle Sicherheitspolitiken gemeinsam gespeichert, die direkt und eindeutig Benutzern, Prozessen oder Objekten zuzuordnen sind. Die Tabellen 5.1, 5.2 und 5.3 geben eine Übersicht über alle benötigten Attribute für die bereits implementierten Politiken, Unterkapitel 5.6 erläutert die Verwendung der aufgeführten Attribute. Die Speicherung der Attribute erfolgt in dynamischer Verwaltung als Listen.

Clark-Wilson (CWI)

Für das Clark-Wilson-Modell sind zusätzlich beliebig viele Tupel (Benutzer-Identifier, TP-Identifier, Liste von CDIs, Liste der diese Regel momentan anwendenden Prozesse) zu verwalten. Da diese keine eindeutige Zuordnung erlauben, sind die zugehörigen Datenstrukturen abgetrennt.

Attribut	Politik	Mögliche Werte
Prozeß-Identifier	Alle	(Systemabhängig)
Besitzer	Alle	Benutzer-Bezeichner
Maximales Sicherheitslevel	MAC	unclassified, confidential, secret, top secret
Aktuelles Sicherheitslevel	MAC	unclassified, confidential, secret, top secret
Mac-Auto	MAC	False, True
Min-write-open	MAC	unclassified, confidential, secret, top secret
Max-read-open	MAC	unclassified, confidential, secret, top secret
Mac-Trusted	MAC	False, True (Trusted Subject)
Prozeß-Typ	CWI	NIL, TP, IVP, TPICD
PM-Prozeß-Typ	PM	none, TP
Enthaltene PM-TP	PM	TP-Bezeichner
Aktuelle Aufgabe	PM	Aufgaben-Bezeichner

Tabelle 5.2: Prozeßattribute

Attribut	Politik	Mögliche Werte
Objekt-Bezeichner	Alle	(Systemabhängig)
Sicherheitslevel	MAC	unclassified, confidential, secret, top secret
Objekt-Kategorie	FC, SIM	general, security, system
Objekt-Typ	Alle	file, dir, ipc
Programm-Typ	CWI	NIL, TP, IVP, TPICD
PM-Objekt-Typ	PM	none, TP, personal data, non personal data, ipc, dir
PM-Objekt-Klasse	PM	Klassen-Bezeichner (nur für personal data)
Enthaltene PM-TP	PM	TP-Bezeichner (nur für Dateien)
PM-ipc-Zweck	PM	Zweck-Bezeichner (nur für Interprozeßkommunikation)

Tabelle 5.3: Objektattribute

Das Clark-Wilson-Modell wurde im Rahmen dieses Projektes bisher nicht vollständig implementiert.

Datenschutzmodell (PM)

Das Datenschutzmodell erfordert eine Reihe von unabhängigen Objektdefinitionen und Relationen, die getrennt von der allgemeinen Datenhaltung zur Verfügung gestellt werden.

Die Tabellen 5.4 und 5.5 enthalten einen Überblick über alle modellspezifischen Listen. Die Hilfslisten sind dabei Listen von Mengen, die über Bezeichner in den Hauptlisten zugegriffen werden.

5.4.2 Kontrollfunktionen

Zur korrekten Funktion der Datenhaltung sind Prozeduren zur Initialisierung, zur Einbindung und Entfernung von Dateisystemen notwendig.

Die Initialisierungsfunktion setzt Grundwerte und liest von allen momentan eingebundenen Dateisystemen die Zugriffskontrollinformationen ein. Entsprechend lesen die beiden anderen Funktionen die ACI für ein einzelnes Dateisystem nachträglich ein bzw. entfernen diese aus dem Hauptspeicher.

5.4.3 Zugriffsfunktionen

Diese Funktionen bieten Zugriff auf einzelne Attribute. Mögliche Anforderungen sind das Setzen und das Auslesen eines Attributwertes für ein Ziel, d. h., einen Benutzer, einen Prozeß oder ein Objekt, sowie das Entfernen der gesamten ACI eines Zieles bei dessen Löschung.

Das Auslesen eines nicht zuvor gesetzten Wertes liefert einen Standardwert zurück, der zur Kompilationszeit festgelegt wird. Das Setzen eines Attributwertes außerhalb des definierten Bereiches oder für ein falsches Objekt ist nicht möglich und führt zu einer Fehlermeldung.

Die spezifischen Zugriffsfunktionen für die PM-Daten enthalten auch die benötigten Mengenoperationen für die Mengen-Elemente der Hilfslisten und liefern grundsätzlich einen Fehler, falls versucht wird, auf ein nicht existierendes Objekt oder Element zuzugreifen.

5.5 Erzwingungskomponente (AEF)

5.5.1 Vorgehen

In jedem sicherheitsrelevanten Systemaufruf ist nach der Abfrage der diskreten Zugriffsbedingungen ein Aufruf der Entscheidungsfunktion der Entscheidungs-

Liste	Daten	Bedeutung
Aufgaben	Bezeichner	Identifizierung
	Zweck	Zweck der Aufgabe
	TP-Mengen-Bezeichner	Autorisierte TP (Hilfsliste)
	Benutzer-Mengen-Bezeichner	Verantwortliche Benutzer (Hilfsliste)
Objekt-Klassen	Bezeichner	Identifizierung
	Zweck-Mengen-Bezeichner	Erhebungszwecke der Daten in dieser Klasse (Hilfsliste)
Notwendige Zugriffe	Aufgabe	Aktuelle Aufgabe
	Klasse	Klasse der zu nutzenden Daten
	TP	Zugreifende TP
	Zugriffe	Erlaubte Zugriffe (Lesen, Schreiben, Erzeugen, Löschen, Anfügen)
Betroffenen-Einwilligungen	Objekt-Bezeichner	Zugegriffenes Objekt
	Zweck-Bezeichner	Eingewilligter Zweck
TP	Bezeichner	Identifizierung
Zwecke	Bezeichner	Identifizierung
Freigaben (Tickets)	Bezeichner	Identifikator der Freigabe durch einen Datenschutzbeauftragten oder verantwortlichen Benutzer
	Aussteller	Benutzer, der Freigabe ausgestellt hat
	Funktion	Freigegebene Funktion
	Parameter	Freigegebene Parameter
	Zeitstempel	Gültigkeitsbeschränkung

Tabelle 5.4: PM-Objektlisten

Liste	Bedeutung
Aufgaben-Mengen	Autorisierte Aufgaben eines Benutzers
TP-Mengen	Autorisierte TP einer Aufgabe
Benutzer-Mengen	Verantwortliche Benutzer für eine Aufgabe
Zweck-Mengen	Erhebungszwecke der Daten einer Objektklasse

Tabelle 5.5: PM-Hilfslisten

komponente durchzuführen und der Zugriff bei negativer Entscheidung abzulehnen.

Nach Freigabe des Zugriffs, aber vor der Rückgabe der Kontrolle an den Prozeß, ist die Datenanpassungsfunktion der Entscheidungskomponente aufzurufen, um den Zustand des Zugriffskontrollsystems anpassen zu können. Dieser Aufruf kann entfallen, wenn keiner der Entscheidungsregelsätze ihn benötigt.

5.5.2 Übersicht der sicherheitsrelevanten Systemaufrufe

Diese Tabelle zeigt alle für die implementierten Modelle sicherheitsrelevanten Systemaufrufe eines Standard-Systemkerns mit einer kurzen Beschreibung ihrer Funktion.[1] Bei der Integration weiterer Modelle wären eventuell weitere Funktionen zu berücksichtigen.

Systemaufruf	Beschreibung
access()	DAC-Rechte für ein Objekt auslesen
adjtimex()	Systemzeitwerte setzen und lesen
chdir(),fchdir()	Arbeitsverzeichnis setzen
chmod(),fchmod()	DAC-Rechte für Objekt setzen
chown(), fchown()	Besitzer eines Objektes für DAC ändern
close()	Datei schließen (Hilfsnachricht, keine Entscheidung)
creat()	Neue Datei erzeugen
create_module()	Hinzufügen eines Kernelmoduls vorbereiten
delete_module()	Entfernen eines Kernelmoduls
exec_ve()	Ausführen eines Folge-Prozesses mit der selben Prozeß-ID
exit()	Beenden (Hilfsnachricht, keine Entscheidung)
fork(), clone()	Erzeugen eines Kindprozesses als Kopie des aktuellen
getdents()	Verzeichniseinträge aus angegebenem Verzeichnis lesen
init_module()	Hinzufügen eines Kernelmoduls, erfordert vorheriges create_module()
Fortsetzung auf der nächsten Seite...	

[1] Siehe Tabelle 3.1 für die Liste aller Aufrufe.

... Fortsetzung Sicherheitsrelevante Systemaufrufe	
Systemaufruf	Beschreibung
ioperm(), iopl()	Zugriffsrechte auf Hardwareports ändern
ipc()	Komplette Verwaltung der Interprozeßkommunikation nach System V
kill()	Senden eines Signals an einen Prozeß
link(), symlink()	(symbolischen) Link im Dateisystem erzeugen
mkdir()	Verzeichnis anlegen
mknod()	Pseudodatei erzeugen, z.B. für Geräte in /dev
mount()	Dateisystem einhängen
msgget()	IPC-Nachrichten-Warteschlange erzeugen
msgrcv()	IPC-Nachrichten-Warteschlange auslesen
msgsnd()	IPC-Nachrichten-Warteschlange beschreiben
open()	Vorhandene Datei öffnen, dabei evt. neu erzeugen oder leeren
ptrace()	Ablaufkontrolle eines anderen Prozesses
readdir()	nächsten Verzeichniseintrag lesen
reboot()	System herunterfahren und neu starten
rename()	Datei umbenennen
rmdir()	Verzeichnis löschen
setdomainname()	Domänennamen des Systems setzen
setfsuid(), setfsgid()	Benutzer- bzw. Gruppen-Id des Prozesses für Dateisystemzugriffe ändern
setgroups()	Gruppenrechte eines Prozesses setzen
sethostname()	Hostnamen des Systems setzen
setreuid(), setregid	Reale (normale) und effektive Benutzer- bzw. Gruppen-Id des Prozesses ändern
setrlimit()	Ressourcenschranken des aktuellen Prozesses setzen
setuid(), setgid	Effektive und Dateisystem-Benutzer- bzw. -Gruppen-Id des Prozesses ändern, bei Benutzer root alle
Fortsetzung auf der nächsten Seite...	

... Fortsetzung Sicherheitsrelevante Systemaufrufe	
Systemaufruf	Beschreibung
socketcall()	Verwaltung der Interprozeßkommunikation über Sockets
stat(), fstat(), lstat(), new_stat(), new_fstat(), new_lstat()	Dateiverwaltungsinformationen auslesen
statfs(), fstatfs()	Verwaltungsinformationen für das Dateisystem der angegebenen Datei auslesen
stime(), settimeofday()	Systemzeit setzen
swapon(), swapoff()	Auslagerungsspeicher (de)aktivieren
syslog()	System-Protokoll auslesen und verwalten
truncate(), ftruncate()	Dateilänge ändern
umount()	Dateisystem aushängen
unlink()	Löschen von Dateien und Links
utime()	Zeitstempel einer Datei setzen
utimes()	Zeitstempel einer Datei setzen

Tabelle 5.6: Sicherheitsrelevante Systemaufrufe

5.6 Entscheidungskomponente (ADF)

5.6.1 Aufbau und Entscheidungsfindung

Wie in Unterkapitel 5.3, Ablauf der Zugriffskontrolle, beschrieben, werden ein Entscheidungs- und ein Datenanpassungsaufruf benötigt. Beide Aufrufe verwenden den Vorschlag von La Padula,[2] jeden Regelsatz unabhängig aufzurufen und das Ergebnis logisch zusammenzusetzen. Die Entscheidungswerte "Ja", "Nein", "Egal" und "Nicht definiert", sowie deren logische Verknüpfung "und-plus"[3] sind direkt übernommen.

Eine Gesamtentscheidung von "Nicht definiert" erzeugt dabei eine Fehlermeldung und wird in "Nein" umgesetzt. Die Datenanpassung liefert einen Fehler, wenn mindestens ein Modell einen Fehler geliefert hat. Dieser Fehler ist in der Durchsetzungskomponente entsprechend zu behandeln.

[2] Siehe Abschnitt 4.2.4.
[3] Siehe Tabelle 4.7.

5.6.2 Anfragen der Durchsetzungskomponente(AEF) an die Entscheidungskomponente(ADF)

Die nachfolgende Tabelle 5.7 beschreibt die Anfragen der Durchsetzung an die Entscheidung. Die mit “AO” gekennzeichneten Anforderungen sind von mir definiert, die mit “LP” gekennzeichneten aus [LaPadula95] übernommen und angepaßt worden.

Jede Anfrage enthält zwei Parameter, den aufrufenden Prozeß und das Ziel, auf das zugegriffen werden soll. Die Tabelle gibt als zweiten Parameter für jede Anfrage die möglichen Zieltypen an, für die diese Anfrage definiert ist. Die rechte Spalte enthält jeweils diejenigen Systemaufrufe aus Abschnitt 5.5.2, die eine Anfrage dieses Typs durchführen.

Anfrage	Beschreibung	Linux-Systemaufruf
ADD-TO-KERNEL(process, file) [AO]	Hinzufügen eines Kernel-Moduls	create_module(), init_module()
ALTER(process, ipc) [LP]	Ändern der Kontrollinformation für IPC-Objekte	ipc(), msgctl(), shmctl()
APPEND-OPEN(process, file/ipc) [LP]	Öffnen zum Anhängen	open(), msgsnd()
CHANGE-GROUP(process, process/file/dir) [LP]	(aktuelle) Gruppe der Datei/des Verzeichnisses/des Prozesses ändern	chgrp(), fchgrp(), setgid(), setfsgid(), setregid(), setgroups()
CHANGE-OWNER(process, process/file/dir) [LP]	Besitzer oder (aktuelle) Gruppe der Datei/des Verzeichnisses/des Prozesses ändern (beinhaltet Änderung des Attributs owner bei Prozessen!)	chown(), fchown(), setuid(), setfsuid(), setreuid()
CHDIR(process, dir) [AO]	Neues Arbeitsverzeichnis setzen	chdir(), fchdir()
Fortsetzung auf der nächsten Seite...		

...Fortsetzung Anfragen AEF an ADF		
Anfrage	Beschreibung	Linux-Systemaufruf
CLONE(process, process) [LP]	Aus aktuellem Prozeß Kopie erzeugen (entspricht CREATE für process)	fork(), clone()
CLOSE(process, file/dir) [LP]	Datei/Verzeichnis wurde geschlossen, Hilfsnachricht der AEF an die ADF	close()
CREATE(process, file/directory/scd/ipc) [LP]	Neues Objekt erzeugen	creat(), ipc(), socketcall(), mkdir(), mknod(), symlink(), open(), msgget(), shmget()
DELETE(process, file/directory/scd/ipc) [LP]	Objekt löschen	ipc(), socketcall(), rmdir(), unlink(), msgctl()
EXECUTE(process, file) [LP]	Programm aus Datei ausführen	exec_ve()
GET-PERMISSIONS-DATA(process, file/directory/scd/ipc) [LP]	Diskrete Zugriffsrechte auslesen	access(),
GET-STATUS-DATA(process, file/dir/ipc/scd) [LP]	Statusdaten auslesen, wie Typ, Besitzer, Dateigröße	stat(), fstat(), lstat(), new_stat(), new_fstat(), new_lstat(), statfs(), fstatfs(), msgctl(), shmctl()
LINK-HARD(process,file) [LP]	Harten Link, d.h. weiteren Namen, erzeugen - verhindert möglicherweise echtes Löschen	link()
Fortsetzung auf der nächsten Seite...		

... Fortsetzung Anfragen AEF an ADF		
Anfrage	Beschreibung	Linux-Systemaufruf
MODIFY-ACCESS-DATA(process, file/dir) [LP]	Zugriffsinformationen ändern, z.B. letzte Zugriffszeit	utime()
MODIFY-ATTRIBUTE(process, user/process/file/dir/ipc, attribute, value) [LP]	Attribut auf neuen Wert setzen (Verwaltungsinformation des Regelsystems), Sonderfall: Attribut none heißt Rücksetzen aller Attribute des Zieles auf Standardwerte	rsbac_set_attr(), rsbac_remove_target()
MODIFY-PERMISSIONS-DATA(process, file/dir/ipc/scd) [LP]	Diskrete Zugriffsrechte setzen	chmod(), fchmod(), ioperm(), iopl()
MODIFY-SYSTEM-DATA(process, scd) [AO]	Systemdaten ändern, z.B. Uhrzeit	adjtimex(), stime(), settimeofday(), sethostname(), setdomainname(), setrlimit(), swapon(), swapoff(), syslog()
MOUNT(process, dir) [AO]	Dateisystem in angegebenes Verzeichnis einbinden	mount()
READ(process, directory) [LP]	Verzeichnisdaten lesen	readdir(), readlink(), getdent()
READ-ATTRIBUTE(process, user/process/file/dir/ipc, attribute) [LP]	Attribut auslesen (Verwaltungsinformationen des Regelsystems)	rsbac_get_attr()
READ-WRITE-OPEN(process, file/ipc) [LP]	Öffnen zum Lesen und Schreiben	ipc(), socketcall(), open(), shmat()
Fortsetzung auf der nächsten Seite...		

... Fortsetzung Anfragen AEF an ADF		
Anfrage	Beschreibung	Linux-Systemaufruf
READ-OPEN(process, file/dir/ipc) [LP]	Öffnen zum Lesen	ipc(), open(), msgrcv(), shmat()
REMOVE-FROM-KERNEL(process) [AO]	Modul entladen	delete_module()
RENAME(process, file/dir) [LP]	Umbenennen einer Datei oder eines Verzeichnisses	rename()
SEARCH(process, directory) [LP]	Lesen eines Verzeichnisses von der AEF aus	Kernel-intern
SEND-SIGNAL(process, process) [LP]	Signal senden an Ziel-Prozeß	kill()
SHUTDOWN(process) [AO]	System herunterfahren	reboot()
SWITCH-LOG(process) [AO]	Protokollierung für ADF-Module an- oder abschalten	rsbac_adf_log_switch()
SWITCH-MODULE(process) [AO]	ADF-Module an- oder abschalten	rsbac_switch()
TERMINATE(process) [LP]	Prozeß wurde beendet, Hilfsnachricht der AEF an die ADF	sys_exit()
TRACE(process, process) [LP]	Tracen des Zielprozesses durch aufrufenden Prozeß, d.h., Lesen und Schreiben des Speichers.	ptrace()
TRUNCATE(process, file) [LP]	Alle Daten in der Datei löschen (truncate)	open(), truncate(), ftruncate()
UMOUNT(process, dir) [AO]	Dateisystem ausbinden	umount()
WRITE(process, dir) [LP]	Schreiben auf Verzeichnis	rename()
WRITE-OPEN(process, file) [LP]	Öffnen zum Schreiben	open()
Fortsetzung auf der nächsten Seite...		

... *Fortsetzung Anfragen AEF an ADF*		
Anfrage	Beschreibung	Linux-Systemaufruf

Tabelle 5.7: Anfragen der AEF an die ADF

5.6.3 Klassische mandatorische Zugriffskontrolle (MAC)

Das Bell-La Padula-Modell unterscheidet zwischen dem maximalen und dem aktuellen Sicherheitslevel eines Subjektes, ohne dabei ein Subjekt näher zu spezifizieren. In einem Unix-System sind als Subjekte sowohl Benutzer, als auch Prozesse zu sehen, wobei ein aktuelles Sicherheitslevel für einen Benutzer höchstens als Vorgabe für künftige Prozesse zu verstehen ist und hier nicht verwendet wird.

Das Setzen des aktuellen Sicherheitslevels wird im Ansatz von La Padula nicht näher erläutert und war deshalb von mir zu spezifizieren.

Automatische Anpassung des aktuellen Sicherheitslevels

Da das simple-security-property ein möglichst hohes aktuelles Sicherheitslevel zum Lesen wünschenswert macht und Prozesse häufiger Lese- als Schreibzugriffe benötigen, wird das aktuelle Sicherheitslevel jedes Prozesses auf das maximale Sicherheitslevel seines Besitzers vorinitialisiert. Danach kann der Prozeß durch einen Systemaufruf sein aktuelles Sicherheitslevel im Rahmen der Sicherheitseigenschaften jederzeit neu setzen.

Das *-property erfordert, daß Schreibzugriffe nur auf Daten erfolgen dürfen, deren Sicherheitslevel mindestens so hoch ist wie das aktuelle Sicherheitslevel des Prozesses. Damit also ein Prozess auf Daten mit einem niedrigeren Sicherheitslevel schreiben kann, ist vorher sein aktuelles Sicherheitslevel anzupassen. Dieses erfordert entweder, *jedes* Programm um den zugehörigen Systemaufruf set_current_security_level zu erweitern, das für den Betrieb des Systemes Schreibzugriffe benötigt, oder ein Verfahren zu entwickeln, das die Anpassung automatisch vornimmt, da sonst bereits der Systemstart wegen der notwendigen Schreibzugriffe unmöglich wird.

Das von mir entwickelte Verfahren auto-write setzt bei jeder Schreibanforderung, die sonst abzuweisen wäre, das aktuelle Sicherheitslevel des Prozesses auf das des Zielobjektes. Dabei ist allerdings zu gewährleisten, daß kein Lesezugriff auf ein Objekt mit höherem Level besteht oder, wegen des frei verwendbaren Adreßraums im Hauptspeicher, vorher bestand. Deshalb enthält das Prozeß-Attribut max-read-open das Maximum der Sicherheitslevel aller bisher zum Lesen geöffneten Objekte und darf niemals unterschritten werden.

Für Lesezugriffe existiert ein entsprechender Mechanismus namens auto-read, der zusätzlich zur Schranke min-write-open natürlich das maximale Sicherheitslevel berücksichtigt. Für Lese- und Schreibzugriffe auf das selbe Objekt werden beide Verfahren zum auto-read-write kombiniert.

Setzt ein Prozeß sein aktuelles Sicherheitslevel per Systemaufruf, so wird der Automatismus offensichtlich nicht benötigt und deshalb über das Attribut mac-auto abgeschaltet. Selbstverständlich werden auch hier die Schranken max-read-open und min-write-open weiterverwendet.

Ist ein Prozeß über das Attribut mac-trusted als vertrauenswürdig gekennzeichnet ("Trusted Subject"), so werden alle Bedingungen für das Schreiben und die Schranke min-write-open außer Kraft gesetzt.

Das Kopieren eines Prozesses übernimmt alle beteiligten Attribute. Beim Ausführen eines anderen Programmes im selben Prozeß werden vom Kern sämtliche Dateien geschlossen und der Speicher gelöscht, weshalb alle Werte gefahrlos zurückgesetzt werden können.

Es gelten folgende Definitionen:

Auto-write: Zugriff wird zugelassen, wenn

1. Aktuelles Sicherheitslevel(P) $\leq$ Sicherheitslevel(O) *oder*
2. Prozeß ist vertrauenswürdig (Mac-Trusted ist gesetzt) *oder*
3. Mac-Auto ist aktiviert, aktuelles Sicherheitslevel(P) $>$ Sicherheitslevel(O), Sicherheitslevel(O) $\geq$ Max. Read-Open

Bei der Benachrichtigung wird durchgeführt:

- Min. Write-Open := min(Sicherheitslevel(O), Min. Write-Open))
- im Fall 3 außerdem: aktuelles Sicherheitslevel(P) := Sicherheitslevel(O)

Auto-read: Zugriff wird zugelassen, wenn

1. Aktuelles Sicherheitslevel(P) $\geq$ Sicherheitslevel(O) *oder*
2. Mac-Auto ist aktiviert, aktuelles Sicherheitslevel(P) $<$ Sicherheitslevel(O), Sicherheitslevel(O) $\leq$ maximales Sicherheitslevel(Prozeßbesitzer), Sicherheitslevel(O) $\leq$ Min. Write-Open

Bei der Benachrichtigung wird durchgeführt:

- Max. Read-Open := max(Sicherheitslevel(O), Max. Read-Open)
- im Fall 3 außerdem: aktuelles Sicherheitslevel(P) := Sicherheitslevel(O)

Auto-read-write: Zugriff wird zugelassen, wenn

1. Prozeß ist vertrauenswürdig (Mac-Trusted ist gesetzt) und Auto-read erlaubt den Zugriff *oder*
2. Aktuelles Sicherheitslevel(P) = Sicherheitslevel(O) *oder*

3. Mac-Auto ist aktiviert, Sicherheitslevel(O) $\leq$ maximales Sicherheitslevel(Prozeßbesitzer), Sicherheitslevel(O) $\geq$ Max. Read-Open, Sicherheitslevel(O) $\leq$ Min. Write-Open

Bei der Benachrichtigung wird durchgeführt:

- Max. Read-Open := max(Sicherheitslevel(O), Max. Read-Open)
- Min. Write-Open := min(Sicherheitslevel(O), Min. Write-Open)
- im Fall 3 außerdem: aktuelles Sicherheitslevel(P) := Sicherheitslevel(O)

Im weiteren Text werden ausschließlich diese Abkürzungen verwendet.

Zugriffsbedingungen

Die nachfolgenden Tabellen 5.8 und 5.9 beschreiben die Bedingungen für eine Zugriffsgewährung in Abhängigkeit von Anfrage und Ziel.

Eine Angabe "P>O" bedeutet dabei, der Wert des Attributes "Sicherheitslevel" des Prozesses ist größer als der des Zieles. ":=" bezeichnet das Setzen von neuen Attributwerten in der Benachrichtigungsfunktion. Ein Strich steht immer dann, wenn dieser Fall nicht definiert oder von dieser Politik nicht betroffen ist.

Zum Schutz der Zugriffskontrollinformationen wird zusätzlich auf Benutzer-Rollen zurückgegriffen. So ist z. B. das Ändern von Sicherheitslevels nur Sicherheitsbeauftragten gestattet. Einzelne Kontrollmechanismen müssen aus systeminternen Gründen zusätzlich für Administratoren freigegeben werden.

Anfrage	Zugriffsziel		
	Benutzer	Prozess	Systemdaten
ADD-TO-KERNEL(process, file) [AO]	-	-	-
ALTER(process, ipc)	-	-	-
APPEND-OPEN(process, file/ipc)	-	-	-
CHANGE-GROUP(process, process/file/dir/ipc)	-	-	-
CHANGE-OWNER(process, process/file/dir/ipc)	-	(Alter Besitzer) $\geq$ (Neuer Besitzer) oder system-role(alter Besitzer) = Administrator; owner + owner-sec-level := (Neuer Besitzer)	-

Fortsetzung auf der nächsten Seite...

...Fortsetzung MAC - Benutzer, Prozesse, Systemkontrolldaten			
Anfrage	Benutzer	Prozess	Systemdaten
CHDIR(process, directory)	-	-	-
CLONE(process, process)	-	P_2:=P_1 (Übernahme aller Attribute)	-
CLOSE(process, file/dir)	-	-	-
CREATE(process, directory/ipc)	-	-	-
DELETE(process, file/directory/ipc)	-	-	-
EXECUTE(process, file)	-	-	-
GET-PERMISSIONS-DATA(process, file/directory/scd/ipc)	-	-	-
GET-STATUS-DATA(process, file/dir/ipc/scd)	-	-	-
LINK-HARD(process,file)	-	-	-
MODIFY-ACCESS-DATA(process, file/dir)	-	-	-
MODIFY-ATTRIBUTE(process, user/process/file/dir/ipc, attribute, value)	Alles, außer (sowohl Attribut ist MAC-relevant, als auch Besitzerrolle ist nicht Sicherheitsbeauftragter) und owner (aus Konsistenzgründen)		
MODIFY-PERMISSIONS-DATA(process, file/dir/ipc/scd)	-	-	Besitzerrolle = Sicherheitsbeauftragter
MODIFY-SYSTEM-DATA(process, scd) [AO]	-	-	Besitzerrolle = Systemadministrator
Fortsetzung auf der nächsten Seite...			

... Fortsetzung MAC - Benutzer, Prozesse, Systemkontrolldaten			
Anfrage	Benutzer	Prozess	Systemdaten
MOUNT(process, dir) [AO]	-	-	-
READ(process, directory)	-	-	-
READ-ATTRIBUTE(process, user/process/file/dir/ipc, attribute)	Alles, außer (sowohl Attribut ist MAC-relevant, als auch Besitzerrolle ist nicht Sicherheitsbeauftragter)		
READ-OPEN(process, file/dir/ipc)	-	-	-
READ-WRITE-OPEN(process, file/ipc)	-	-	-
REMOVE-FROM-KERNEL(process) [AO]	Besitzerrolle = Systemadministrator		
RENAME(process, file/dir)	-	-	-
SEARCH(process, directory)	-	-	-
SEND-SIGNAL(process, process)	-	auto-write	-
SHUTDOWN(process) [AO]	Besitzerrolle = Systemadministrator		
SWITCH-LOG(process) [AO]	Besitzerrolle = Sicherheitsbeauftragter		
SWITCH-MODULE(process) [AO]	Besitzerrolle = Sicherheitsbeauftragter		
TERMINATE(process)	-	-	-
TRACE(process, process)	-	auto-read-write	-
TRUNCATE(process, file)	-	-	-
Fortsetzung auf der nächsten Seite...			

... Fortsetzung MAC - Benutzer, Prozesse, Systemkontrolldaten			
Anfrage	Benutzer	Prozess	Systemdaten
UMOUNT(process, dir) [AO]	Besitzerrolle = Systemadministrator		
WRITE(process, dir)	-	-	-
WRITE-OPEN(process, file)	-	-	-

Tabelle 5.8: Entscheidungsgrundlagen MAC - Benutzer, Prozesse, Systemkontrolldaten

Anfrage	Zugriffsziel		
	Datei	Verzeichnis	Prozeßkomm.
ADD-TO-KERNEL(process, file) [AO]	auto-read, Besitzerrolle = Systemadministrator	-	-
ALTER(process, ipc)	-	-	auto-write
APPEND-OPEN(process, file/ipc)	auto-write	-	auto-write
CHANGE-GROUP(process, process/file/dir/ipc)	auto-write	auto-write	auto-write
CHANGE-OWNER(process, process/file/dir/ipc)	auto-write	auto-write	auto-write
CHDIR(process, directory)	-	auto-read	-
CLONE(process, process)	-	-	-
CLOSE(process, file/dir)	-	-	-
CREATE(process, directory/ipc)	-	auto-write, O_2 :=P	O:=P
Fortsetzung auf der nächsten Seite...			

... Fortsetzung MAC - Datei, Verzeichnis, Interprozeßkommunikation			
Anfrage	Datei	Verzeichnis	Prozeßkomm.
DELETE(process, file/directory/ipc)	auto-write	auto-write	auto-write
EXECUTE(process, file)	auto-read, Rücksetzung der Max- und Min-Werte	-	-
GET-PERMISSIONS-DATA(process, file/directory/scd/ipc)	-	-	-
GET-STATUS-DATA(process, file/dir/ipc/scd)	-	-	-
LINK-HARD(process,file)	auto-write	-	-
MODIFY-ACCESS-DATA(process, file/dir)	auto-write	auto-write	-
MODIFY-ATTRIBUTE(process, user/process/file/dir/ipc, attribute, value)	Alles, außer (sowohl Attribut ist MAC-relevant, als auch Besitzerrolle ist nicht Sicherheitsbeauftragter) und owner		
MODIFY-PERMISSIONS-DATA(process, file/dir/ipc/scd)	auto-write	auto-write	auto-write (= ALTER)
MODIFY-SYSTEM-DATA(process, scd) [AO]	-	-	-
MOUNT(process, dir) [AO]	-	auto-write, Besitzerrolle = Systemadministrator	-
READ(process, directory)	-	auto-read	-
Fortsetzung auf der nächsten Seite...			

...Fortsetzung MAC - Datei, Verzeichnis, Interprozeßkommunikation			
Anfrage	Datei	Verzeichnis	Prozeßkomm.
READ-ATTRIBUTE(process, user/process/file/dir/ipc, attribute)	Alles, außer (sowohl Attribut ist MAC-relevant, als auch Besitzerrolle ist nicht Sicherheitsbeauftragter)		
READ-OPEN(process, file/dir/ipc)	auto-read	auto-read	auto-read
READ-WRITE-OPEN(process, file/ipc)	auto-read-write	-	auto-read-write
REMOVE-FROM-KERNEL(process) [AO]	Besitzerrolle = Systemadministrator		
RENAME(process, file/dir)	auto-write	auto-write	-
SEARCH(process, directory)	-	auto-read	-
SEND-SIGNAL(process, process)	-	-	-
SHUTDOWN(process) [AO]	Besitzerrolle = Systemadministrator		
SWITCH-LOG(process) [AO]	Besitzerrolle = Sicherheitsbeauftragter		
SWITCH-MODULE(process) [AO]	Besitzerrolle = Sicherheitsbeauftragter		
TERMINATE(process)	-	-	-
TRACE(process, process)	-	-	-
TRUNCATE(process, file)	auto-write	-	-
UMOUNT(process, dir) [AO]	Besitzerrolle = Systemadministrator		
WRITE(process, dir)	-	auto-write	-
Fortsetzung auf der nächsten Seite...			

...Fortsetzung MAC - Datei, Verzeichnis, Interprozeßkommunikation			
Anfrage	Datei	Verzeichnis	Prozeßkomm.
WRITE-OPEN(process, file)	auto-write	-	-

Tabelle 5.9: Entscheidungsgrundlagen MAC - Datei, Verzeichnis, Interprozeßkommunikation

5.6.4 Clark-Wilson-Modell (CWI)

Das Clark-Wilson-Modell wurde im Rahmen dieses Projektes bisher nicht weiter umgesetzt.

5.6.5 Funktionale Kontrolle (FC)

Die Entscheidungsgrundlage der funktionale Kontrolle als rollenbasiertes Modell sind fast ausschließlich die System-Rolle des Besitzers des aufrufenden Prozesses und die Objekt-Kategorie des Objektes, auf das zugegriffen werden soll. Ist das Zugriffsziel keiner der Objekttypen Datei, Verzeichnis, Interprozeßkommunikation und Systemkontrollinformation, wird der Zugriff grundsätzlich gestattet.

Da für Systemkontrollinformationen keine Attribute verwaltet werden, gilt für diese stets die Objekt-Kategorie "system". Einzige Ausnahme bilden deren Zugriffskontrollinformationen, die als "security" eingestuft werden.

Eine Ausnahme ist, entsprechend der mandatorischen Kontrolle, der Zugriff auf die Zugriffskontrollinformationen der funktionalen Kontrolle und auf interne Verwaltungsattribute wie Besitzer eines Prozesses. Dieser wird nur Benutzern mit der Rolle Sicherheitsbeauftragter gestattet.

Außerdem muß, wie bei der mandatorischen Kontrolle, bei einigen Systemkontrollinformationen der Zugriff sowohl für Administratoren, als auch für Sicherheitsbeauftragte gestattet werden, um die Lauffähigkeit des Betriebssystems gewährleisten zu können.

Schließlich gibt es noch Systemverwaltungsaufgaben, die nur von Administratoren ausgeführt werden dürfen. Dazu zählen das Laden und Entfernen von Kernmodulen (ADD-TO-KERNEL, REMOVE FROM KERNEL), das Einbinden und Entfernen von Dateisystemen (MOUNT, UMOUNT) sowie das Herunterfahren des Systems (SHUTDOWN).

Es ergeben sich somit folgende Entscheidungsgrundlagen, die nacheinander abgearbeitet werden:

- Ist die Anforderung das Lesen oder das Setzen von für die funktionale Kontrolle relevanten Attributen, wird der Zugriff gestattet, falls die Rolle des Besitzers des Prozesses Sicherheitsbeauftragter ist.

- Sind das Ziel ein Objekt vom Typ Datei, Verzeichnis, Interprozeßkommunikation oder Systemkontrollinformation, die Rolle des Besitzers des Prozesses R und die Objektkategorie O, wird der Zugriff gestattet, falls R kompatibel zu O oder O = "generell".

- Ist die Anforderung ein Zugriff auf die Zugriffskontrollinformationen von Systemkontrollinformationen, wird der Zugriff gestattet, falls die Rolle des Besitzers des Prozesses B Sicherheitsbeauftragter ist.

Nach positiver Entscheidung und erfolgter Durchführung sind auch in dieser Politik Attribute anzupassen, allerdings lediglich bei der Anfrage CREATE zum Erzeugen neuer Objekte. Dabei erben Dateien und Verzeichnisse die Objektkategorie des übergeordneten Verzeichnisses, Interprozeßkommunikationsobjekte vom Besitzer des erzeugenden Prozesses.

Zur Wahrung der Konsistenz wird auch hier beim Wechsel des Prozeßbesitzers das Attribut Besitzer des Prozesses angepaßt.

5.6.6 Änderung der Sicherheitsinformationen (SIM)

Die Politik der Veränderung von Sicherheitsinformationen entscheidet, ähnlich der funktionalen Kontrolle, als rollenbasiertes Modell ebenfalls aufgrund der System-Rolle des Besitzers des aufrufenden Prozesses und des Datentyps des Objektes, auf das zugegriffen werden soll. Ist das Zugriffsziel keiner der Objekttypen Datei, Verzeichnis, Interprozeßkommunikation und Systemkontrollinformation, oder ist nur Lesezugriff gewünscht, wird der Zugriff grundsätzlich gestattet.

Da für Systemkontrollinformationen keine Attribute verwaltet werden, gilt für diese stets der Datentyp "unklassifiziert". Einzige Ausnahme bilden deren Zugriffskontrollinformationen, die als "Sicherheitsinformation" eingestuft werden.

Für diese Politik relevante Attribute werden ebenfalls als Sicherheitsinformationen eingestuft und dürfen nur von Sicherheitsbeauftragten modifiziert werden.

Daraus ergeben sich folgende Entscheidungsgrundlagen, die nacheinander abgearbeitet werden:

- Ist die Anforderung das Setzen von für diese Politik relevanten Attributen, wird der Zugriff gestattet, falls die Rolle des Besitzers des Prozesses Sicherheitsbeauftragter ist.

- Sind das Ziel ein Objekt vom Typ Datei, Verzeichnis, Interprozeßkommunikation oder Systemkontrollinformation, die Anforderung eine Schreibanforderung, die Rolle des Besitzers des Prozesses R und der Objektdatentyp O, wird der Zugriff gestattet, falls R = Sicherheitsbeauftragter oder O $\neq$ "Sicherheitsinformation".

Die Attributsetzung erfolgt analog zur funktionalen Kontrolle nur bei der Anforderung CREATE. Dateien und Verzeichnisse erben das Attribut Datentyp vom übergeordneten Verzeichnis, Interprozeßkommunikationsobjekte erhalten den Datentyp Sicherheitsinformation gesetzt, falls der Besitzer des aufrufenden Prozesses Sicherheitsbeauftragter ist.

5.6.7 Datenschutzmodell (PM)

Im Datenschutzmodell wird auf die Prädikate "Notwendiger Zugriff", "Zweckbindung" und "Einwilligung" zurückgegriffen, die in Unterkapitel 2.7 definiert wurden. In den Zugriffs-Tabellen werden mehrere Kurzformen verwendet, die sich auf die veränderlichen Parameter beschränken:

Objekt-Typ	Kurzform	Vollform
Datei	Notwendig (*Modus*)	Notwendig (Aktuelle Aufgabe (Prozeß), Klasse (Datei), TP (Prozeß), *Modus*)
	Notwendig (*Klasse*, *Modus*)	Notwendig (Aktuelle Aufgabe (Prozeß), *Klasse*, TP (Prozeß), *Modus*)
	Zweckbindung	Zweckbindung (Aktuelle Aufgabe (Prozeß), Klasse (Datei))
	Einwilligung	Einwilligung (Zweck (Aktuelle Aufgabe (Prozeß)), Datei))
IPC	Notwendig (*Modus*)	Notwendig (Aktuelle Aufgabe (Prozeß), IPC, TP (Prozeß), *Modus*)
	Zweckbindung	Zweckbindung (Aktuelle Aufgabe (Prozeß), IPC-Zweck (IPC-Objekt)) oder (Zugriffs-Modus ist nur lesend und IPC-Zweck (IPC-Objekt) ist NIL)

Tabelle 5.10: Kurzformen in den PM-Entscheidungsgrundlagen

Entsprechend der mandatorischen Kontrolle werden im Datenschutz-Modell für die System- und Sicherheitsadministration die PM-Rollen der Besitzer der aufrufenden Prozesse verwendet.

Die Zugriffstabellen wurden nach einer bisher unveröffentlichten Systemspezifikation von Simone Fischer-Hübner erstellt.[4] Wie in den MAC-Entscheidungsgrundlagen stehen Striche immer dann, wenn der Fall nicht definiert ist oder von diesem Modell nicht kontrolliert wird, und ":=", wenn Attribute bei der Benachrichtigung gesetzt werden. Aus Platzgründen wurden die sechs möglichen Zugriffsziele wieder auf zwei Tabellen verteilt.

[4] Siehe ansatzweise in [FiHue97].

Anfrage	Zugriffsziel		
	Benutzer	Prozess	Systemdaten
ADD-TO-KERNEL(process, file) [AO]	-	-	-
ALTER(process, ipc)	-	-	-
APPEND-OPEN(process, file/ipc)	-	-	-
CHANGE-GROUP(process, process/file/dir/ipc)	-	-	-
CHANGE-OWNER(process, process/file/dir/ipc)	-	Aktuelle Aufgabe(Prozeß) $\in$ Autorisierte Aufgaben(Neuer Besitzer)	-
CHDIR(process, directory)	-	-	-
CLONE(process, process)	-	PM-Prozeß-Typ(Prozeß) = NIL, Übernahme aller PM-Attribute	-
CLOSE(process, file/dir)	-	-	-
CREATE(process, directory/ipc)	-	-	-
DELETE(process, file/directory/ipc)	-	-	-
EXECUTE(process, file)	-	-	-
GET-PERMISSIONS-DATA(process, file/directory/scd/ipc)	-	-	-
GET-STATUS-DATA(process, file/dir/ipc/scd)	-	-	-
Fortsetzung auf der nächsten Seite...			

...Fortsetzung PM - Benutzer, Prozesse, Systemkontrolldaten			
Anfrage	Benutzer	Prozess	Systemdaten
LINK-HARD(process,file)	-	-	-
MODIFY-ACCESS-DATA(process, file/dir)	-	-	-
MODIFY-ATTRIBUTE(process, user/process/file/dir/ipc, attribute, value)	Alles, außer Attribut ist PM-relevant		-
MODIFY-PERMISSIONS-DATA(process, file/dir/ipc/scd)	-	-	-
MODIFY-SYSTEM-DATA(process, scd) [AO]	-	-	Prozeß-Besitzer-PM-Rolle ist Systemadministrator
MOUNT(process, dir) [AO]	-	-	-
READ(process, directory)	-	-	-
READ-ATTRIBUTE(process, user/process/file/dir/ipc, attribute)	Wenn PM-relevant: Prozeß-Besitzer-PM-Rolle ist Sicherheitsbeauftragter oder Datenschutzbeauftragter		-
READ-OPEN(process, file/dir/ipc)	-	-	-
READ-WRITE-OPEN(process, file/ipc)	-	-	-
REMOVE-FROM-KERNEL(process) [AO]	Prozeß-Besitzer-PM-Rolle ist Systemadministrator		
Fortsetzung auf der nächsten Seite...			

... Fortsetzung PM - Benutzer, Prozesse, Systemkontrolldaten			
Anfrage	Benutzer	Prozess	Systemdaten
RENAME(process, file/dir)	-	-	-
SEARCH(process, directory)	-	-	-
SEND-SIGNAL(process, process)	-	PM-Prozeß-Typ des aufrufenden Prozesses ist NIL und (wenn Signal ist hartes Beenden und PM-Prozeß-Typ des Zielprozesses ist TP: Prozeß-Besitzer-PM-Rolle ist TP-Manager)	-
SHUTDOWN(process) [AO]	Prozeß-Besitzer-PM-Rolle ist Systemadministrator		
SWITCH-LOG(process) [AO]	Prozeß-Besitzer-PM-Rolle ist Sicherheitsbeauftragter		
SWITCH-MODULE(process) [AO]	Prozeß-Besitzer-PM-Rolle ist Sicherheitsbeauftragter		
TERMINATE(process)	-	-	-
TRACE(process, process)	-	PM-Prozeß-Typ des aufrufenden Prozesses und des Zielprozesses ist NIL	-
TRUNCATE(process, file)	-	-	-
UMOUNT(process, dir) [AO]	Prozeß-Besitzer-PM-Rolle ist Systemadministrator		
WRITE(process, dir)	-	-	-
Fortsetzung auf der nächsten Seite...			

... Fortsetzung PM - Benutzer, Prozesse, Systemkontrolldaten			
Anfrage	Benutzer	Prozess	Systemdaten
WRITE-OPEN(process, file)	-	-	-

Tabelle 5.11: Entscheidungsgrundlagen PM - Benutzer, Prozesse, Systemkontrolldaten

Anfrage	Zugriffsziel		
	Datei	Verz.	Prozeßkomm. (IPC)
ADD-TO-KERNEL(process, file) [AO]	Prozeß-Besitzer-PM-Rolle ist Systemadministrator und PM-Objekt-Typ(Datei) ist NIL oder nicht-personenbezogen	-	-
ALTER(process, ipc)	-	-	-
APPEND-OPEN(process, file/ipc)	PM-Objekttyp(Datei) $\neq$ TP und [wenn PM-Objekttyp(Datei) $\neq$ personenbezogen: PM-Prozeßtyp(Prozeß) $\neq$ TP] und [wenn PM-Objekttyp(Datei) $=$ personenbezogen: Notwendig(Anfügen) und (Zweckbindung oder Einwilligung)]	-	[wenn IPC-Zweck(IPC-Objekt) $=$ NIL: PM-Prozeßtyp(Prozeß) $\neq$ TP] oder [wenn IPC-Zweck(IPC-Objekt) $\neq$ NIL: aktuelle Aufgabe(Prozeß) $\neq$ NIL, Notwendig(Anfügen) und Zweckbindung
CHANGE-GROUP(process, process/file/dir/ipc)	-	-	-
CHANGE-OWNER(process, process/file/dir/ipc)	PM-Objekt-Typ ist weder TP noch personenbezogen	-	-
CHDIR(process, directory)	-	-	-
Fortsetzung auf der nächsten Seite...			

... Fortsetzung PM - Datei, Verzeichnis, Interprozeßkommunikation			
Anfrage	Datei	Verz.	Prozeßkomm. (IPC)
CLONE(process, process)	-	-	-
CLOSE(process, file/dir)	-	-	-
CREATE(process, directory/ipc)	-	[wenn PM-Prozeß-Typ = TP: Notwendig (NIL-Klasse, Erzeugen)], Klasse := NIL, PM-Objekt-Typ := (wenn Verz.: DIR, wenn PM-Prozeß-Typ = TP: personenbezogen, sonst: none), Enthaltene PM-TP(Neues Objekt) := NIL	[wenn aktuelle Aufgabe(Prozeß) = NIL: IPC-Zweck(Objekt) := NIL], [sonst: Notwendig(Erzeugen), IPC-Zweck(Objekt) := Zweck(aktuelle Aufgabe(Prozeß))]
DELETE(process, file/directory/ipc)	[wenn PM-Objekttyp(Datei) = TP: Prozeß-Besitzer-PM-Rolle ist TP-Manager] oder [wenn PM-Objekttyp(Datei) = personenbezogene: Notwendig(Löschen) und (Zweckbindung oder Einwilligung)]	-	[wenn aktuelle Aufgabe(Prozeß) $\neq$ NIL: Notwendig(Löschen) und Zweckbindung] oder [wenn aktuelle Aufgabe(Prozeß) = NIL: IPC-Zweck(IPC-Objekt) = NIL
Fortsetzung auf der nächsten Seite...			

... Fortsetzung PM - Datei, Verzeichnis, Interprozeßkommunikation			
Anfrage	Datei	Verz.	Prozeßkomm. (IPC)
EXECUTE(process, file)	wenn PM-Objekttyp(Datei) = TP: [TP(Datei) ∈ autorisierte TP(aktuelle Aufgabe(Prozeß)) und PM-Prozeß-Typ(Prozeß) = NIL)], TP(Prozeß) := TP(Datei), PM-Prozeß-Typ(Prozeß) := TP	-	-
GET-PERMISSIONS-DATA(process, file/directory/scd/ipc)	-	-	-
GET-STATUS-DATA(process, file/dir/ipc/scd)	-	-	-
LINK-HARD(process,file)	PM-Objekt-Typ ist weder TP noch personenbezogen	-	-
MODIFY-ACCESS-DATA(process, file/dir)	wenn PM-Objekttyp(Datei) = TP: Prozeß-Besitzer-PM-Rolle ist TP-Manager	-	-
MODIFY-ATTRIBUTE(process, user/process/file/dir/ipc, attribute, value)	[Attribut ist nicht pm-relevant] oder [Attribut ist none (d.h., alle löschen), PM-Objekttyp(Datei) ist none oder nicht-personenbezogen]	Attribut ist none oder nicht pm-relevant	[Attribut ist nicht pm-relevant] oder [Attribut ist none, IPC-Zweck ist NIL]
Fortsetzung auf der nächsten Seite...			

...Fortsetzung PM - Datei, Verzeichnis, Interprozeßkommunikation			
Anfrage	Datei	Verz.	Prozeßkomm. (IPC)
MODIFY-PERMISSIONS-DATA(process, file/dir/ipc/scd)	wenn PM-Objekttyp(Datei) = TP: Prozeß-Besitzer-PM-Rolle ist TP-Manager	-	-
MODIFY-SYSTEM-DATA(process, scd) [AO]	-	-	-
MOUNT(process, dir) [AO]	-	Prozeß-Besitzer-PM-Rolle ist Systemadministrator	-
READ(process, directory)	-	-	-
READ-ATTRIBUTE(process, user/process/file/dir/ipc, attribute)	wenn PM-relevant: Prozeß-Besitzer-PM-Rolle ist Sicherheits- oder Datenschutzbeauftragter		
READ-OPEN(process, file/dir/ipc)	PM-Objekttyp(Datei) ≠ TP und [wenn PM-Objekttyp(Datei) = personenbezogen: Notwendig(Lesen) und (Zweckbindung oder Einwilligung)]	-	[wenn aktuelle Aufgabe(Prozeß) ≠ NIL: Notwendig(Lesen) und Zweckbindung] oder [wenn aktuelle Aufgabe(Prozeß) = NIL: IPC-Zweck(IPC-Objekt) = NIL
READ-WRITE-OPEN(process, file/ipc)	PM-Objekttyp(Datei) ≠ TP und [wenn PM-Objekttyp(Datei) ≠ personenbezogen: PM-Prozeßtyp(Prozeß) ≠ TP] und [wenn PM-Objekttyp(Datei) = personenbezogen: Notwendig(Lesen und Schreiben) und (Zweckbindung oder Einwilligung)]	-	[wenn IPC-Zweck(IPC-Objekt) = NIL: PM-Prozeßtyp(Prozeß) ≠ TP] oder [wenn IPC-Zweck(IPC-Objekt) ≠ NIL: aktuelle Aufgabe(Prozeß) ≠ NIL, Notwendig(Lesen und Schreiben) und Zweckbindung
Fortsetzung auf der nächsten Seite...			

... Fortsetzung PM - Datei, Verzeichnis, Interprozeßkommunikation			
Anfrage	Datei	Verz.	Prozeßkomm. (IPC)
REMOVE-FROM-KERNEL(process) [AO]	Prozeß-Besitzer-PM-Rolle ist Systemadministrator		
RENAME(process, file/dir)	wenn PM-Objekttyp(Datei) = TP: Prozeß-Besitzer-PM-Rolle ist TP-Manager	-	-
SEARCH(process, directory)	-	-	-
SEND-SIGNAL(process, process)	-	-	-
SHUTDOWN(process) [AO]	Prozeß-Besitzer-PM-Rolle ist Systemadministrator		
SWITCH-LOG(process) [AO]	Prozeß-Besitzer-PM-Rolle ist Sicherheitsbeauftragter		
SWITCH-MODULE(process) [AO]	Modul ≠ PM		
TERMINATE(process)	-	-	-
TRACE(process, process)	-	-	-
TRUNCATE(process, file)	PM-Objekttyp(Datei) ≠ TP und [wenn PM-Objekttyp(Datei) = personenbezogen: Notwendig(Schreiben) und (Zweckbindung oder Einwilligung)]	-	-
UMOUNT(process, dir) [AO]	Prozeß-Besitzer-PM-Rolle ist Systemadministrator		
Fortsetzung auf der nächsten Seite...			

...Fortsetzung PM - Datei, Verzeichnis, Interprozeßkommunikation			
Anfrage	Datei	Verz.	Prozeßkomm. (IPC)
WRITE(process, dir)	-	-	-
WRITE-OPEN(process, file)	PM-Objekttyp(Datei) $\neq$ TP und [wenn PM-Objekttyp(Datei) $\neq$ personenbezogen: PM-Prozeßtyp(Prozeß) $\neq$ TP] und [wenn PM-Objekttyp(Datei) $=$ personenbezogen: Notwendig(Schreiben) und (Zweckbindung oder Einwilligung)]	-	-

Tabelle 5.12: Entscheidungsgrundlagen PM - Datei, Verzeichnis, Interprozeßkommunikation

5.7 Zusätzliche Systemaufrufe

Zur Verwaltung der Zugriffskontrolle werden in dieser Arbeit weitere sicherheitsrelevante Systemaufrufe definiert. Dabei wird zwischen allgemeinen und modellspezifischen Aufrufen unterschieden.

5.7.1 Allgemeine Aufrufe

Die nachfolgende Tabelle zeigt die für alle Modelle verwendeten Systemaufrufe:

Systemaufruf	Beschreibung
rsbac_stats()	Statusinformationen ausgeben
rsbac_get_attr()	Attributwert auslesen (Angabe der Inode)
rsbac_get_attr_n()	Attributwert für Zugriffskontrollsystem auslesen (Angabe des Pfadnamens)
Fortsetzung auf der nächsten Seite...	

...Fortsetzung Neue allgemeine Systemaufrufe	
Systemaufruf	Beschreibung
rsbac_set_attr()	Attributwert setzen (Angabe der Inode)
rsbac_set_attr_n()	Attributwert für Zugriffskontrollsystem setzen (Angabe des Pfadnamens)
rsbac_remove_target()	Alle Attributwerte eines Zieles zurücksetzen (Angabe der Inode)
rsbac_remove_target_n()	Alle Attributwerte eines Zieles zurücksetzen (Angabe des Pfadnamens)
rsbac_switch()	Entscheidungsmodul an- und abschalten (Angabe der Modulnummer und des Wertes 0 oder 1)
rsbac_adf_log_switch()	Protokoll für ADF-Anfrage an- und abschalten (Angabe der Anfragennummer und des Wertes 0 oder 1)

Tabelle 5.13: Neue allgemeine Systemaufrufe

5.7.2 Modellspezifische Aufrufe

MAC

Für die mandatorische Kontrolle werden die zwei Aufrufe sys_rsbac_mac_set_curr_seclevel und sys_rsbac_mac_get_curr_seclevel zur Verfügung gestellt, die einem Prozeß erlauben, sein aktuelles Sicherheitslevel zu setzen und auszulesen. Das zu setzende Level muß innerhalb der auch für die automatische Anpassung Grenzen liegen. Durch explizites Ändern wird die automatische Anpassung des aktuellen Sicherheitslevels deaktiviert.

PM

Das Datenschutzmodul benötigt, wie in Abschnitt 2.7.4 beschrieben, von Benutzern bzw. Prozessen verwendbare nicht-privilegierte und privilegierte Kontroll-Funktionen. Auch diese sind, sofern nicht implizit vorhanden, durch Systemaufrufe zu realisieren.

Die PM-Systemaufrufe zeigt die folgende Tabelle:

Systemaufruf	Beschreibung
Fortsetzung auf der nächsten Seite...	

... Fortsetzung PM-Systemaufrufe	
Systemaufruf	Beschreibung
rsbac_stats_pm()	PM: Statusinformationen ausgeben
rsbac_pm_change_current-_task()	PM: Aktuelle Aufgabe dieses Prozesses ändern
rsbac_pm_create_file()	PM: Datei mit angegebener Klasse erzeugen
rsbac_pm	PM: Tickets erzeugen und Kontrollfunktionen aufrufen

Tabelle 5.14: PM-Systemaufrufe

Sämtliche privilegierte Kontrollfunktionen werden durch den Systemaufruf sys_rsbac_pm abgedeckt. Dieser bekommt die gewünschte Funktion, die Parameter und gegebenenfalls den Ticketbezeichner übergeben. Zur besseren Verwaltung, aber auch aufgrund von Besonderheiten des Gesamtsystems, mußten einige Kontrollfunktionen hinzugefügt werden. Tabelle 5.15 gibt eine Übersicht über die neuen Funktionen:

Systemaufruf	Beschreibung
set-object-class	Nachträgliches Setzen oder Ändern der Objektklasse für anderweitig erzeugte Objekte mit automatischer Kennzeichnung als persönliche oder nicht-persönliche Daten. Erfordert Ticket.
switch-pm	An- oder Abschalten des gesamten Datenschutz-Moduls, nur bei Aktivierung in der Kernkonfiguration. Erfordert Ticket.
write-list	Liste von Kontrollinformationen ausgeben. Nur Sicherheitsbeauftragte, Datenschutzbeauftragte und eingeschränkt TP-Verwalter
set-tp	Programm mit TP-Bezeichner als TP kennzeichnen. Nur TP-Verwalter.

Tabelle 5.15: Neue PM-Kontrollfunktionen

Kapitel 6

Entstandener Programmcode

Sämtlicher hier vorgestellter Programmcode ist von mir in der Programmiersprache C geschrieben und wird unverändert zitiert. Headerdateien werden im Verzeichnis `include/rsbac`, Codedateien ab dem Verzeichnis `rsbac`, jeweils unter dem Hauptverzeichnis der Kernquellen, abgelegt. Wie im Kapitel 3 erläutert, verwendet man für beide Dateitypen nur den relativen Pfad ab ihrem jeweiligen Hauptverzeichnis.

Entsprechend den Linuxkonventionen sind Bezeichner und Kommentare grundsätzlich englisch. Sämtliche Erweiterungen sind im Konfigurationssystem des Kerns gruppenweise zu aktivieren. Dadurch sind insbesondere die unterstützten Sicherheitsmodelle einzeln zu- oder abschaltbar. Bei Deaktivierung aller Erweiterungen entsteht ein unveränderter Linux-Kern der Version 2.0.30. Die Kernkonfiguration wird im Kapitel 7 näher erklärt.

Alle von mir geschriebenen und geänderten Quelltexte finden sich außerdem in ungekürzter Fassung im Anhang. Dort befinden sich auch die Quellen der im nächsten Kapitel vorgestellten Administrationswerkzeuge, auf die hier nicht weiter eingegangen wird.

6.1 Allgemeine Datenhaltung

Alle Zugriffsfunktionen und Instanzen der vorgestellten Datentypen sind in das Modul `data_structures/aci_data_structures.c` gekapselt. Von außen sichtbar sind ausschließlich die unten vorgestellten Kontroll- und Zugriffsfunktionen, jedoch keine Datenstrukturen. Verwendete Datentypen werden über die Datei `types.h` nur insoweit sichtbar, wie sie für die Funktionsaufrufe notwendig sind.

Diese Maßnahmen bieten einen gewissen Schutz vor unkontrolliertem Zugriff auf die Daten aus anderen Teilen des Kernes heraus, geben aber keine Sicherheit vor Zerstörung durch willkürliches Überschreiben.

Die allgemeine Datenhaltung ist außerdem zuständig für die Synchronisation sämtlicher Zugriffe, sowie für die Sicherung auf Datenträger und die Wiederherstellung beim Systemneustart. Diese Mechanismen sind für alle anderen Komponenten vollständig transparent.

6.1.1 Datentypen

In den Header-Dateien `types.h` und `pm_types.h` für die PM-Datentypen werden zunächst die verwendeten Attribute mit ihren Wertebereichen definiert:

```
typedef uid_t rsbac_uid_t;                    /* Same as user in Linux */
typedef pid_t rsbac_pid_t;                    /* Same as pid in Linux  */
enum    rsbac_security_level_t {SL_unclassified, SL_confidential, SL_secret,
                                SL_top_secret, SL_rsbac_internal};
                                             /* MAC security levels   */
enum    rsbac_system_role_t {SR_user, SR_security_officer, SR_administrator};
                                             /* For FC and SIM        */
enum    rsbac_integrity_role_t {IR_none, IR_TP_user, IR_TP_manager,
                                IR_IVP_user};
                                             /* For CWI               */
enum    rsbac_process_type_t {PT_none, PT_TP, PT_IVP, PT_TPICD};
                                             /* For CWI               */
enum    rsbac_object_category_t {OC_general, OC_security, OC_system};
                                             /* For FC                */
enum    rsbac_object_type_t {OT_file, OT_dir, OT_ipc, OT_scd};
                                             /* For MAC               */
typedef enum rsbac_process_type_t rsbac_program_type_t;
                                             /* Also for CWI          */
enum    rsbac_data_type_t {DT_none, DT_CDI, DT_CDIIC, DT_SI};
                                             /* Also for CWI          */

enum    rsbac_pm_role_t {PR_user,PR_security_officer,
                         PR_data_protection_officer,
                         PR_tp_manager,PR_system_admin,
                         PR_none};

enum    rsbac_pm_process_type_t {PP_none, PP_TP};

enum    rsbac_pm_object_type_t {PO_none, PO_TP, PO_personal_data,
                                PO_non_personal_data, PO_ipc, PO_dir};

typedef enum rsbac_pm_process_type_t rsbac_pm_program_type_t;
```

Das Security-Level rsbac-internal kann nicht von außen gesetzt werden, sondern ist für die interne Sicherung der Zugriffskontrollinformationen auf der Festplatte bestimmt. Dazu wird beim Laden der Zugriffskontrollinformationen eines Dateisystems automatisch für das Verzeichnis `rsbac` dieses Attribut gesetzt. Da ein Benutzer nie das Security-Level rsbac-internal erreichen kann, verhindert die mandatorische Zugriffskontrolle jeglichen Benutzerzugriff auf diese Daten. Zusätzlich sind die Objekt-Kategorie "system", der Datentyp "SI" und entsprechende diskrete Rechte gesetzt, so daß funktionale Kontrolle, Sicherheitsdaten-Modifikations-Modell und diskrete Kontrolle einen zusätzlichen Schutz bieten können.

Um die feste Attributstruktur nicht in Abhängigkeit einzelner Sicherheitspolitiken durch Unterlisten zu durchbrechen, sind in den Hauptdatenstrukturen ausschließlich Attribute zugelassen. Politikspezifische Listen werden getrennt verwaltet und über Identifikations-Attribute zugegriffen.

```
typedef u_int rsbac_cwi_relation_id_t;
typedef u_int rsbac_pm_task_id_t;
typedef u_long rsbac_pm_task_set_id_t;
typedef u_int rsbac_pm_tp_id_t;       /* transformation procedure id */
typedef u_int rsbac_pm_tp_set_id_t;   /* transformation procedure set id */
typedef u_int rsbac_pm_ru_set_id_t;   /* responsible user set id */
typedef u_int rsbac_pm_purpose_id_t;
typedef long rsbac_pm_pp_set_id_t;   /* purpose set id */
typedef u_int rsbac_pm_object_class_id_t;
typedef u_int rsbac_pm_tkt_id_t;      /* ticket id */
typedef long rsbac_pm_time_stamp_t;   /* for ticket time stamps */
```

Um einzelne Dateien systemweit eindeutig bezeichnen zu können, werden Dateisystem und Nummer der Verwaltungsstruktur in diesem Dateisystem (Inode) verwendet.

```
struct rsbac_fs_file_t
    {
      kdev_t               device;
      u_long               inode;
    };
```

Die Behandlung der Attribute erfolgt getrennt nach Benutzer, Prozeß, Datei, Verzeichnis und Interprozeßkommunikation. Systemkontrolldaten sind keine Attribute zugeordnet. In den Datenstrukturen werden Datei- und Verzeichnisattribute gemeinsam gespeichert. Es ergeben sich folgende Strukturen in der Datei `aci_data_structures.h`:

```
struct rsbac_user_aci_t
    {       rsbac_uid_t                  id;
            rsbac_pseudo_t               pseudo;
      enum  rsbac_security_level_t       access_appr;
```

```
      enum   rsbac_system_role_t          system_role;
      enum   rsbac_integrity_role_t       integrity_role;
             rsbac_cwi_relation_id_t      cwi_rel_id;
             rsbac_pm_task_set_id_t       pm_task_set;
      enum   rsbac_pm_role_t              pm_role;
    };

struct rsbac_process_aci_t
    {
             rsbac_pid_t               id;
             rsbac_uid_t               owner;
      enum   rsbac_security_level_t    owner_sec_level;
      enum   rsbac_security_level_t    current_sec_level;
      enum   rsbac_security_level_t    min_write_open;   /* for *-property */
      enum   rsbac_security_level_t    max_read_open;    /* for *-property */
             boolean                   mac_auto;   /* auto-curr-sec-level? */
             boolean                   mac_trusted; /* trusted process?    */
      enum   rsbac_process_type_t      process_type;
             rsbac_pm_tp_id_t          pm_tp;
             rsbac_pm_task_id_t        pm_current_task;
      enum   rsbac_pm_process_type_t   pm_process_type;
    };

struct rsbac_file_dir_aci_t
    {
      u_long                            id;     /* file system inode number */
      enum   rsbac_security_level_t     sec_level;       /* MAC */
      enum   rsbac_object_category_t    object_category; /* FC */
      enum   rsbac_object_type_t        object_type;     /* MAC, only OT_file, OT_dir */
      enum   rsbac_data_type_t          data_type;       /* CWI */
      rsbac_program_type_t              program_type;    /* CWI */
      rsbac_pm_object_class_id_t        pm_object_class; /* PM  */
      rsbac_pm_tp_id_t                  pm_tp;           /* PM (for FILE only) */
      enum   rsbac_pm_object_type_t     pm_object_type;  /* PM */
    };

struct rsbac_ipc_aci_t
    {
      struct rsbac_ipc_t                id;
      enum   rsbac_security_level_t     sec_level;
      enum   rsbac_object_category_t    object_category;
      enum   rsbac_object_type_t        object_type;     /* OT_ipc only */
      enum   rsbac_data_type_t          data_type;
```

```
        rsbac_pm_object_class_id_t pm_object_class; /* ipc only */
        rsbac_pm_purpose_id_t      pm_ipc_purpose;
  enum  rsbac_pm_object_type_t     pm_object_type;
  };
```

6.1.2 Verwaltung im Hauptspeicher

Verwaltet werden die Attributstrukturen jeweils in doppelt verketteten Listen mit den üblichen Zugriffsfunktionen Hinzufügen, Entfernen und Suchen. Um parallelen Zugriff auf die Listen zu ermöglichen, werden sie jeweils im Listenkopf durch Semaphore mit gegenseitigem Ausschluß geschützt. Alle in diesem Abschnitt aufgeführten Definitionen befinden sich in der Datei `aci_data_structures.h`.

Als Beispiel einer Listenstruktur seien hier Kopf- und Elementtyp der Benutzerliste aufgeführt:

```
struct rsbac_user_list_item_t
    {
      struct rsbac_user_aci_t         aci;
      struct rsbac_user_list_item_t * prev;
      struct rsbac_user_list_item_t * next;
    };

struct rsbac_user_list_head_t
    {
      struct rsbac_user_list_item_t * head;
      struct rsbac_user_list_item_t * tail;
      struct rsbac_user_list_item_t * curr;
      struct semaphore                sem;
    };
```

Während Benutzer-, Prozeß- und Interprozeßkommunikationsliste systemweit zentral gehalten werden können, müssen Datei- und Verzeichnisattribute für jedes Dateisystem getrennt vorliegen. Außerdem macht ihre große Anzahl eine lineare Suche in einer einzigen Liste sehr ineffektiv.

Deshalb wird hier erst einmal eine Liste von Dateisystemen, hier Devices genannt, geführt (siehe Abbildung 6.1). Innerhalb eines Devices gibt es ein Feld von Listenköpfen für Unterlisten, deren Elemente wiederum die Attributstrukturen der Dateien und Verzeichnisse enthalten. Die Auswahl der Feldelemente erfolgt über eine einfache Modulo-Hashfunktion nach der Inode-Nummer.

Schutz vor Inkonsistenz der Device-Liste bieten ein Lesezugriffs- und ein Schreibanforderungszähler, sowie ein darüber sitzendes Semaphor. Schreibzugriffe werden grundsätzlich bevorzugt behandelt.

Jeder Aufruf einer Zugriffsfunktion auf Attribute hält den Lesezugriffszähler solange um eins erhöht, bis keinerlei Zugriff auf interne Strukturen mehr nötig ist.

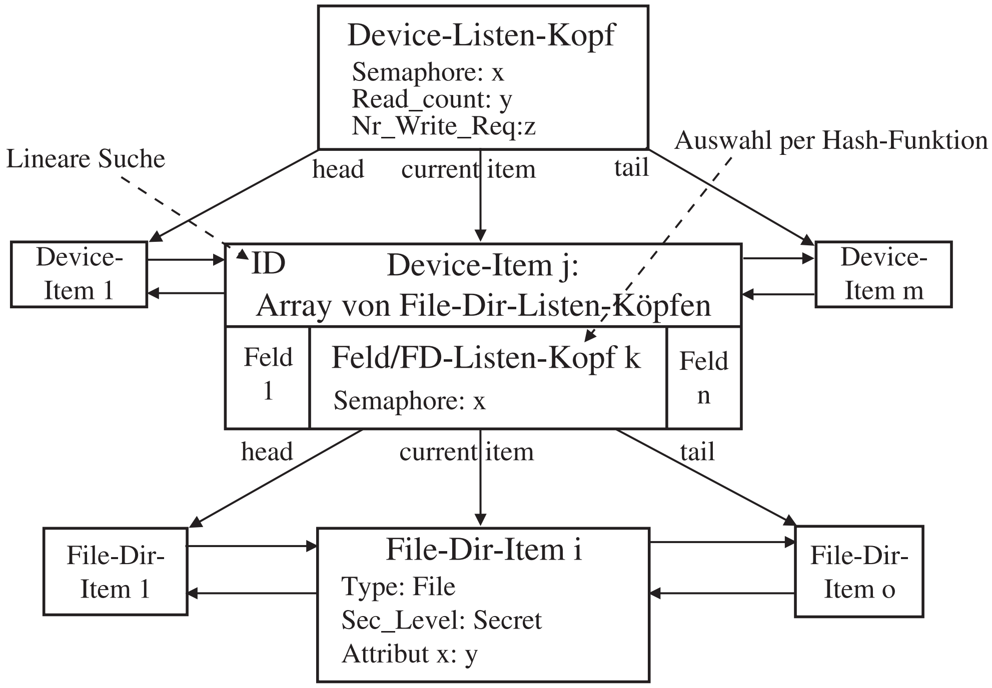

Abbildung 6.1: Datenstruktur für Datei- und Verzeichnisattribute

Auch Schreibzugriffe auf Unterlisten erfordern in der Device-Liste lediglich Lesezugriffe, so daß nur das Hinzufügen oder Entfernen von Dateisystemen Schreibzugriff und somit exklusive Nutzung der Device-Liste benötigen.

Unterlisten werden wie die anderen Objektlisten durch einfache Semaphore bei jedem Zugriff gesichert.

6.1.3 Verwaltung auf der Festplatte

Die Speicherung auf Festplatte erfolgt für jedes Dateisystem getrennt, und zwar unter dessen Hauptverzeichnis im Verzeichnis `rsbac`. Bei jeder Änderung eines Attributes wird, falls möglich, die zugehörige Liste gespeichert, um stets ein aktuelles Abbild der Zugriffsrechte auf der Festplatte zu erhalten.

Der Verzeichnisname und die Dateinamen werden in der Datei `aci_data_structures.h` festgelegt. Da die Zugriffskontrolle für beliebige Dateisysteme, auch FAT, funktionieren soll und ein Zeichen für die Namen von Sicherheitskopien verwendet wird,[1] darf der Dateiname höchstens 7 Zeichen lang sein. Prozeß- und Interprozeßkommunikations-Liste müssen nicht gespeichert werden und sind deshalb nicht mit aufgeführt.

[1] Die speziellen Dateisysteme proc, nfs und iso9660 werden als nur lesbar behandelt, d.h., die Attribute werden nicht auf die Festplatte geschrieben.

```
#define RSBAC_ACI_PATH          "rsbac"
#define RSBAC_ACI_FD_NAME       "fd_aci."
#define RSBAC_ACI_USER_NAME     "useraci"
#define RSBAC_ACI_CWI_NAME      "cwi_aci"
/* dir creation mode for discretionary access control: rwx for user only*/
#define RSBAC_ACI_DIR_MODE       0700
/* file creation mode for discretionary access control: rw for user only*/
#define RSBAC_ACI_FILE_MODE      0600
```

Einen Sonderfall stellen die Datei- und Verzeichnisattribute dar, die, wie weiter oben dargestellt, auf mehrere Teillisten verteilt im Speicher vorliegen. Hier wird jede Teilliste getrennt auf Festplatte gespeichert, wofür der Dateiname um die Listennummer ergänzt wird. Dieses reduziert den Speicherungsaufwand deutlich.

6.1.4 Kontrollfunktionen

Als Kontrollfunktionen werden diejenigen Aufrufe bezeichnet, die sich auf den gesamten Datenbestand auswirken. Sie werden in der Datei `aci.h` deklariert und in der Datei `aci_data_structures.c` definiert. Die erste ist die Initialisierungs-Funktion, vor deren Aufruf keinerlei Zugriff auf Attribute möglich ist. Diese initialisiert die Listenstrukturen und liest die Attributinformationen aller zur Aufrufzeit eingebunden Dateisysteme. Die Initialisierungsfunktion wird vom Init-Prozeß direkt nach der Einbindung des Haupt-Dateisystems aufgerufen, um von Anfang an eine Zugriffskontrolle zu ermöglichen.

Die nächsten beiden Funktionen werden beim Einbinden bzw. Entfernen weiterer Dateisysteme aufgerufen. Erstere liest die abgelegten Attributinformationen, letztere entfernt das Dateisystem aus der Device-Liste und gibt den dafür belegten Speicher frei. Der anzugebende Parameter vom Typ kdev_t ist ein kerninterner Verwaltungsbezeichner für das Dateisystem.

Die letzte Funktion liefert schließlich eine Statusausgabe über den Standard-Kernprotokollmechanismus.

Die Deklarationen sehen folgendermaßen aus:

```
extern int rsbac_init(void);
extern int rsbac_mount(kdev_t);
extern int rsbac_umount(kdev_t);
extern int rsbac_stats(void);
```

6.1.5 Zugriffsfunktionen

Als Zugriffsfunktionen werden diejenigen Aufrufe bezeichnet, die sich nur auf einzelne Attribute auswirken. Die allgemeine Datenhaltung bietet insgesamt drei Funktionen, um auf Attributdaten zuzugreifen.

Die ersten beiden lesen bzw. setzen einen Wert für genau ein Ziel und genau ein Attribut. Unter Ziel ist dabei ein Benutzer, ein Prozeß oder ein Objekt zu verstehen. Das Auslesen eines Attributes für ein noch nicht registriertes Ziel liefert keine Fehlermeldung, sondern einen Standardwert zurück, der über die Variablen default_target_type_aci in der Datei `data_structures/aci_data_structures.c` festgelegt ist. Das Setzen eines Attributes für ein noch nicht registriertes Ziel führt zur automatischen Registrierung. Da das Ziel mit seinem Typ anzugeben ist, läßt sich die Zulässigkeit des angegebenen Attributs leicht überprüfen und der Zugriff gegebenenfalls abweisen.

Die dritte Funktion dient zur kompletten Entfernung aller Attributwerte eines Zieles, genauer, der Rücksetzung auf die Standardwerte durch Deregistrierung.

Zur Parameterübergabe sind zunächst eigene Datentypen in der Datei `types.h` definiert, um nicht-sinnvolle Werte von vornherein ausschließen zu können:

```
enum   rsbac_target_t {T_FILE, T_DIR, T_IPC, T_SCD, T_USER, T_PROCESS};

union  rsbac_target_id_t
       {
          struct rsbac_fs_file_t   file;
          struct rsbac_fs_file_t   dir;
          void                   * ipc;
          struct inode           * scd;
          rsbac_uid_t              user;
          rsbac_pid_t              process;
       };

enum   rsbac_attribute_t {A_security_level, A_object_category, A_data_type,
                          A_system_role, A_integrity_role, A_pm_role,
                          A_process_type, A_pm_process_type, A_pm_current_task,
                          A_object_type, A_pm_object_class, A_pm_ipc_purpose,
                          A_pm_object_type, A_program_type, A_pm_program_type,
                          A_pm_tp, A_pm_task_set, A_owner, A_pseudo,
                          A_current_sec_level, A_min_write_open,
                          A_max_read_open, A_mac_auto, A_mac_trusted,
                          A_cwi_rel_id,
                          A_none};

union  rsbac_attribute_value_t
       {
          enum rsbac_security_level_t   security_level;
          enum rsbac_object_category_t  object_category;
          enum rsbac_data_type_t        data_type;
          enum rsbac_system_role_t      system_role;
```

```
    enum rsbac_integrity_role_t   integrity_role;
    enum rsbac_pm_role_t          pm_role;
    enum rsbac_process_type_t     process_type;
    enum rsbac_pm_process_type_t  pm_process_type;
    rsbac_pm_task_id_t            pm_current_task;
    enum rsbac_object_type_t      object_type;
    rsbac_pm_object_class_id_t    pm_object_class;
    rsbac_pm_purpose_id_t         pm_ipc_purpose;
    enum rsbac_pm_object_type_t   pm_object_type;
    rsbac_program_type_t          program_type;
    rsbac_pm_program_type_t       pm_program_type;
    rsbac_pm_tp_id_t              pm_tp;
    rsbac_pm_task_set_id_t        pm_task_set;
    rsbac_uid_t                   owner;           /* process owner */
    rsbac_pseudo_t                pseudo;
    enum rsbac_security_level_t   current_sec_level;
    enum rsbac_security_level_t   min_write_open;
    enum rsbac_security_level_t   max_read_open;
    boolean                       mac_auto;
    boolean                       mac_trusted;
    rsbac_cwi_relation_id_t       cwi_rel_id;
    int                           dummy;
  };
```

Die Zugriffsfunktionen sind ebenfalls in der Datei `aci.h` deklariert und in der Datei `aci_data_structures.c` definiert.

```
extern int rsbac_get_attr(enum rsbac_target_t,
                          union rsbac_target_id_t,
                          enum rsbac_attribute_t,
                          union rsbac_attribute_value_t *);

extern int rsbac_set_attr(enum rsbac_target_t,
                          union rsbac_target_id_t,
                          enum rsbac_attribute_t,
                          union  rsbac_attribute_value_t);

extern int rsbac_remove_target(enum rsbac_target_t,
                               union rsbac_target_id_t);
```

Als Beispiel für den Ablauf in einer Zugriffsfunktion sei hier das Setzen eines Datei- oder Verzeichnisattributes aufgeführt:

```
case T_FILE:
```

```
case T_DIR:
  if (rsbac_debug_ds)
    printk(KERN_DEBUG
           "rsbac_set_attr(): Setting file/dir attribute for device %u, inode %lu\n",
           (u_int) tid.file.device, tid.file.inode);
  /* get list number */
  head_nr = (tid.file.inode % RSBAC_NR_FD_LISTS);
  /* wait for access to device_list_head */
  down(&(device_list_head.sem));
  /* If there are write requests, wait */
  while (device_list_head.write_request > 0)
    {
      up(&(device_list_head.sem));
      schedule();
      down(&(device_list_head.sem));
    };
  /* add this process to access count */
  device_list_head.count++;
  /* free access to device_list_head */
  up(&(device_list_head.sem));
  /* lookup device */
  device_p = lookup_device(tid.file.device);
  if (!device_p)
    {
      printk(KERN_WARNING "%s\n",
             "rsbac_set_attr(): Could not lookup device!\n");
      /* wait for access to device_list_head */
      down(&(device_list_head.sem));
      /* remove this process from access count */
      device_list_head.count--;
      /* free access to device_list_head */
      up(&(device_list_head.sem));
      return(-RSBAC_EINVALIDDEV);
    }
  /* protect this list */
  down(&(device_p->head_table[head_nr].sem));
  /* look up the item */
  fd_item_p = lookup_file_dir(tid.file.inode, device_p);

  if (!fd_item_p)
    { /* Not found -> add a new item with default values */
      if (rsbac_debug_ds)
        printk(KERN_DEBUG
```

```
      "rsbac_set_attr(): adding file/dir item!\n");
    /* fd_aci was initialized with default_fd_aci, now set id */
    fd_aci.id = tid.file.inode;
    /* if Dir: set object_type dir */
    if (target == T_DIR)
      fd_aci.object_type = OT_dir;
    /* and add item */
    fd_item_p = add_file_dir_item(fd_aci, device_p);
    /* if could not add -> return error */
    if (!fd_item_p)
      {
        printk(KERN_CRIT "%s\n",
               "rsbac_set_attr(): Could not add file/dir item!\n");
        /* unprotect the list */
        up(&(device_p->head_table[head_nr].sem));
        /* wait for access to device_list_head */
        down(&(device_list_head.sem));
        /* remove this process from access count */
        device_list_head.count--;
        /* free access to device_list_head */
        up(&(device_list_head.sem));
        return(-RSBAC_ECOULDNOTADDITEM);
      }
  }
/* OK, now we can do the attribute setting itself */
switch (attr)
  {
    case A_security_level:
      /* security_level SL_rsbac_internal is only set internally */
      if (   (value.security_level == SL_rsbac_internal)
          || (fd_item_p->aci.sec_level == SL_rsbac_internal) )
        {
          err = -RSBAC_EINTERNONLY;
          printk(KERN_WARNING "%s\n",
          "rsbac_set_attr(): SL_rsbac_internal for file/dir, value not set.");
        }
      else
        fd_item_p->aci.sec_level = value.security_level;
      break;
    case A_object_category:
      fd_item_p->aci.object_category = value.object_category;
      break;
    case A_object_type:
```

```
      fd_item_p->aci.object_type = value.object_type;
      break;
    case A_data_type:
      fd_item_p->aci.data_type = value.data_type;
      break;
    case A_program_type:
      fd_item_p->aci.program_type = value.program_type;
      break;
    case A_pm_object_class:
      fd_item_p->aci.pm_object_class = value.pm_object_class;
      break;
    case A_pm_tp:
      fd_item_p->aci.pm_tp = value.pm_tp;
      break;
    case A_pm_object_type:
      fd_item_p->aci.pm_object_type = value.pm_object_type;
      break;
    default:
      err = -RSBAC_EINVALIDATTR;
  }
/* if no error, write to disk */
if (!err)
  write_fd_list(head_nr, &(device_p->head_table[head_nr]), device_p);
/* we are ready, so unprotect the list */
up(&(device_p->head_table[head_nr].sem));
/* wait for access to device_list_head */
down(&(device_list_head.sem));
/* remove this process from access count */
device_list_head.count--;
/* free access to device_list_head */
up(&(device_list_head.sem));
/* and return                          */
return(err);
break;
```

6.2 Politikspezifische Datenhaltung

Die Sicherheitspolitiken CWI und PM benötigen zusätzlich eigenständige Listen und Regeln. Diese werden in den Dateien `cwi_data_structures.h` und `pm_data_structures.h` definiert.

Im Rahmen des RSBAC-Projektes sind die Datenstrukturen für das Clark-Wilson-Modell noch nicht implementiert, weshalb sich dieses Unterkapitel ausschließlich mit dem Datenschutz-Modul beschäftigt.

6.2.1 Datentypen und Verwaltung

Die Tabellen 5.4 und 5.5 in Abschnitt 5.4.1 beschreiben die zur Verwaltung der PM-spezifischen Kontext-Daten notwendigen Listen und Hilfslisten. Deren grundlegende Datenstrukturen befinden sich in `pm_types.h`, insbesondere auch die verwendeten Identifikator-Typen:

```
enum    rsbac_pm_list_t {PL_task,PL_class,PL_na,PL_cs,PL_tp,PL_pp,PL_tkt,PL_none};

enum    rsbac_pm_all_list_t {PA_task,PA_class,PA_na,PA_cs,PA_tp,PA_pp,PA_tkt,
                             PA_task_set,PA_tp_set,PA_ru_set,PA_pp_set,PA_none};

enum    rsbac_pm_set_t  {PS_TASK,PS_TP,PS_RU,PS_PP,PS_NONE};

/* unions */

union rsbac_pm_set_id_t
  {
    rsbac_pm_task_set_id_t  task_set;
    rsbac_pm_tp_set_id_t    tp_set;
    rsbac_pm_ru_set_id_t    ru_set;
    rsbac_pm_pp_set_id_t    pp_set;
  };

union rsbac_pm_set_member_t
  {
    rsbac_pm_task_id_t      task;
    rsbac_pm_tp_id_t        tp;
    rsbac_uid_t             ru;
    rsbac_pm_purpose_id_t   pp;
  };

/* IDs */

struct  rsbac_pm_na_id_t
  {
      rsbac_pm_task_id_t                task;
      rsbac_pm_object_class_id_t        class;
      rsbac_pm_tp_id_t                  tp;
  };
```

```
struct  rsbac_pm_cs_id_t
  {
      rsbac_pm_purpose_id_t              purpose;
      struct rsbac_fs_file_t             file;
  };
```

Die Verwaltung der Hauptlisten erfolgt wie bei den Attributlisten über doppelt verkettete Listen mit Kopfstruktur. Die Speicherung auf der Festplatte erfolgt identisch zur Benutzerliste. Die Nutzdaten der Listen befinden sich wie bei der allgemeinen Datenhaltung gekapselt in einer eigenen Struktur:

```
struct rsbac_pm_task_data_t
    {
      rsbac_pm_task_id_t                 id;
      rsbac_pm_purpose_id_t              purpose;
      rsbac_pm_tp_set_id_t               tp_set;
      rsbac_pm_ru_set_id_t               ru_set;
    };

struct rsbac_pm_class_data_t
    {
      rsbac_pm_object_class_id_t            id;
      rsbac_pm_pp_set_id_t                  pp_set;
    };

struct rsbac_pm_na_data_t
    {
      rsbac_pm_task_id_t                 task;
      rsbac_pm_object_class_id_t         class;
      rsbac_pm_tp_id_t                   tp;
      rsbac_pm_accesses_t                accesses;
    };

struct rsbac_pm_cs_data_t
    {
      rsbac_pm_purpose_id_t              purpose;
      struct rsbac_fs_file_t             file;
    };

struct rsbac_pm_tp_data_t
    {
      rsbac_pm_tp_id_t                   id;
    };
```

```
struct rsbac_pm_pp_data_t
    {
      rsbac_pm_purpose_id_t                 id;
    };

struct rsbac_pm_tkt_data_t
    {
             rsbac_pm_tkt_id_t                        id;
             rsbac_uid_t                              issuer;
      enum   rsbac_pm_tkt_function_type_t             function_type;
      union  rsbac_pm_tkt_internal_function_param_t  function_param;
             rsbac_pm_time_stamp_t                    valid_until;
    };
```

Einen Sonderfall bildet die Ticket-Liste. Da Tickets für verschiedene Verwaltungsfunktionen mit jeweils unterschiedlichen Parametern definiert werden müssen, sind die zugehörigen Definitionen in der Datei `pm_ticket.h` abgelegt.

Die Hilfslisten sind, wie bei der Umsetzung beschrieben, als Listen von Mengen aufzufassen. Es gibt Mengenlisten für autorisierte Aufgaben, autorisierte TP, verantwortliche Benutzer und Zwecke. Implementiert sind sie als Listen, deren Elemente Köpfe von weiteren, einfacheren Listen sind. Zur besseren Verwaltung werden die Längen der Unterlisten in deren Listenkopf abgelegt. Für das Beispiel der Aufgaben-Mengen-Liste sieht das in der Datei `pm_data_structures.h` so aus:

```
/* This list represents sets of task-ids, using a set-id and a sublist each */
struct rsbac_pm_task_set_sublist_item_t
    {
      rsbac_pm_task_id_t                       id;
      struct rsbac_pm_task_set_sublist_item_t * prev;
      struct rsbac_pm_task_set_sublist_item_t * next;
    };

struct rsbac_pm_task_set_list_item_t
    {
      rsbac_pm_task_set_id_t                   id;
      int                                      sublist_length;
      struct rsbac_pm_task_set_sublist_item_t * sublist_head;
      struct rsbac_pm_task_set_sublist_item_t * sublist_tail;
      struct rsbac_pm_task_set_list_item_t    * prev;
      struct rsbac_pm_task_set_list_item_t    * next;
    };

struct rsbac_pm_task_set_list_head_t
    {
```

```
    struct rsbac_pm_task_set_list_item_t * head;
    struct rsbac_pm_task_set_list_item_t * tail;
    struct rsbac_pm_task_set_list_item_t * curr;
    struct semaphore                       sem;
  };
```

Bei der Sicherung auf die Festplatte werden für jedes Listenelement Bezeichner, Länge der Unterliste und Elemente der Unterliste ohne weitere Aufteilung hintereinandergeschrieben.

6.2.2 Kontrollfunktionen

Die PM-Initialisierungsfunktion wird in der Datei `pm.h` deklariert und in der Datei `pm_data_structures.c` definiert. Die Deklaration sieht folgendermaßen aus:

```
extern int rsbac_init_pm(void);
```

6.2.3 Zugriffsfunktionen

Die Zugriffsfunktionen der PM-Datenstrukturen finden sich in den selben Dateien wie die Kontrollfunktionen. Außerdem sind in `pm_types.h` allgemeine Datentypen für Zugriffe auf die Hauptlisten definiert:

```
enum   rsbac_pm_target_t {PMT_TASK,
                          PMT_CLASS,
                          PMT_NA,
                          PMT_CS,
                          PMT_TP,
                          PMT_PP,
                          PMT_TKT,
                          PMT_NONE};

union  rsbac_pm_target_id_t
       {
          rsbac_pm_task_id_t          task;
          rsbac_pm_object_class_id_t  class;
          struct rsbac_pm_na_id_t     na;
          struct rsbac_pm_cs_id_t     cs;
          rsbac_pm_tp_id_t            tp;
          rsbac_pm_purpose_id_t       pp;
          rsbac_pm_tkt_id_t           tkt;
          int                         dummy;
       };
```

```
enum   rsbac_pm_data_t
       {                  PD_purpose,
                          PD_tp_set,
                          PD_ru_set,
                          PD_pp_set,
                          PD_task,
                          PD_class,
                          PD_tp,
                          PD_accesses,
                          PD_file,
                          PD_issuer,
                          PD_function_type,
                          PD_function_param,
                          PD_valid_until,
                          PD_none
       };

union  rsbac_pm_data_value_t
       {
          rsbac_pm_purpose_id_t         purpose;
          rsbac_pm_tp_set_id_t          tp_set;
          rsbac_pm_ru_set_id_t          ru_set;
          rsbac_pm_pp_set_id_t          pp_set;
          rsbac_pm_task_id_t            task;
          rsbac_pm_object_class_id_t    class;
          rsbac_pm_tp_id_t              tp;
          rsbac_pm_accesses_t           accesses;
          struct rsbac_fs_file_t        file;
          rsbac_uid_t                   issuer;
          enum   rsbac_pm_tkt_function_type_t   function_type;
          union  rsbac_pm_tkt_internal_function_param_t  function_param;
          rsbac_pm_time_stamp_t         valid_until;
          int                           dummy;
       };


union  rsbac_pm_all_data_value_t
       {
          struct rsbac_pm_task_data_t   task;
          struct rsbac_pm_class_data_t  class;
          struct rsbac_pm_na_data_t     na;
          struct rsbac_pm_cs_data_t     cs;
          struct rsbac_pm_tp_data_t     tp;
```

```
        struct rsbac_pm_pp_data_t     pp;
        struct rsbac_pm_tkt_data_t    tkt;
        int                           dummy;
     };
```

Zugriffe auf die Hilfslisten erfolgen ausschließlich über die Mengenoperationen Menge Erzeugen, Menge Löschen, Menge Leeren, Element Hinzufügen, Element Löschen, sowie das Prädikat Mengenelement zur Abfrage. Die Benennung der jeweiligen Menge erfolgt durch Angabe des Typs und des Mengenidentifikators, die des Elements durch dessen Bezeichner.

```
int rsbac_pm_create_set     (enum  rsbac_pm_set_t,          /* set type */
                             union rsbac_pm_set_id_t);      /* set id   */


int rsbac_pm_remove_set     (enum  rsbac_pm_set_t,          /* set type */
                             union rsbac_pm_set_id_t);      /* set id   */

int rsbac_pm_add_to_set(enum  rsbac_pm_set_t,          /* set type          */
                        union rsbac_pm_set_id_t,       /* set id            */
                        union rsbac_pm_set_member_t);  /* set member to add */

int rsbac_pm_remove_from_set(enum  rsbac_pm_set_t,          /* see above */
                             union rsbac_pm_set_id_t,
                             union rsbac_pm_set_member_t);

int rsbac_pm_clear_set(enum  rsbac_pm_set_t,          /* set type     */
                       union rsbac_pm_set_id_t);      /* set id       */


boolean  rsbac_pm_set_member(enum  rsbac_pm_set_t,          /* set type */
                             union rsbac_pm_set_id_t,       /* set id   */
                             union rsbac_pm_set_member_t);  /* member   */
```

Auf die Hauptlisten wird vergleichbar der allgemeinen Datenhaltung zugegriffen. Allerdings gibt es hier keinerlei Standardwerte, weshalb zu den Funktionen zum Setzen und Abfragen einzelner Werte auch eine Erzeugungs-, eine Lösch- und eine Komplettabfragefunktion benötigt werden.

Da Anfragen nach nicht existierenden Zielen zu einer Fehlermeldung führen müssen, gibt es zusätzlich ein Existenz-Prädikat und eine Ausgabefunktion.

```
int rsbac_pm_set_data(enum rsbac_pm_target_t,          /* list type */
                      union rsbac_pm_target_id_t,      /* item id in list */
                      enum rsbac_pm_data_t,            /* data item */
```

```
                      union rsbac_pm_data_value_t);    /* data value */

int rsbac_pm_get_data(enum rsbac_pm_target_t,          /* list type */
                      union rsbac_pm_target_id_t,      /* item id in list */
                      enum rsbac_pm_data_t,            /* data item */
                      union rsbac_pm_data_value_t *);  /* for return value */

int rsbac_pm_add_target(enum rsbac_pm_target_t,            /* list type */
                        union rsbac_pm_all_data_value_t);  /* values for all */
                                                           /* data items,    */
                                                           /* incl. item id  */

int rsbac_pm_remove_target(enum rsbac_pm_target_t,        /* list type */
                           union rsbac_pm_target_id_t);   /* item id in list */

int rsbac_pm_get_all_data(enum rsbac_pm_target_t,          /* list type */
                          union rsbac_pm_target_id_t,      /* item id in list */
                          union rsbac_pm_all_data_value_t *);  /* for return value */

boolean rsbac_pm_exists(enum rsbac_pm_target_t,          /* list type */
                        union rsbac_pm_target_id_t);     /* item id in list */

int rsbac_pm_write_list(char *,                          /* filename */
                        enum rsbac_pm_list_t);           /* list type (incl. sets) */
```

Schließlich gibt es noch eine Statusfunktion, die die Summen der Listenelemente aller Listen ausgibt:

```
extern int rsbac_stats_pm(void);
```

6.3 Erzwingungskomponente (AEF)

6.3.1 Anpassung der vorhandenen Systemaufrufe

Dieser Abschnitt beschreibt die Anpassung der in Abschnitt 5.5.2 aufgeführten Systemaufrufe. Es werden der Name der angepaßten Prozedur, der Name der Datei, die jeweilige Anforderung an die Entscheidungsprozedur `rsbac_adf_request()`, gegebenenfalls auch an die Benachrichtigungsprozedur `rsbac_adf_set_attr()` für Vollzugsmeldungen, und die Art des Operationszieles angegeben. Ein Stern bei der Anforderung markiert diejenigen Systemaufrufe, die die ADF vom Vollzug benachrichtigen. Bei zukünftigen Sicherheitsmodulen wäre diese Liste gegebenenfalls zu erweitern.[2]

[2] Siehe auch Abschnitt 5.5.1.

Mit einem Stern versehene Ziele werden im Systemaufruf neu erstellt und bei der Benachrichtigung in den new-target-Parametern eingetragen, um die benötigten Attribute erzeugen zu können. Die in der selben Spalte angegebenen Pseudoattribute enthalten Hilfsinformationen für die Entscheidungsfindung.

Einzelne Systemaufrufe erfüllen mehrere Funktionen und benötigen entsprechend mehrere Entscheidungen. Außerdem ist die Unterstützungs-Prozedur `lookup()` zum kerninternen Lesen von Verzeichnisinhalten mit der Anforderung SEARCH auf den Zieltyp `dir` versehen worden, so daß einige Aufrufe indirekt auch diese Anforderung enthalten.

Systemaufruf	angepaßte Prozedur	Datei	ADF-Anforderung	Ziel
access()	sys_access()	fs/open.c	MODIFY-ACCESS-DATA	file/dir
adjtimex()	sys_adjtimex()	kernel/time.c	MODIFY-SYSTEM-DATA	scd: time_strucs
chdir()	sys_chdir()	fs/open.c	CHDIR	dir
fchdir()	sys_fchdir()	fs/open.c	CHDIR	dir
chmod()	sys_chmod()	fs/open.c	MODIFY-PERMISSIONS-DATA	file/dir
fchmod()	sys_fchmod()	fs/open.c	MODIFY-PERMISSIONS-DATA	file/dir
chown()	sys_chown()	fs/open.c	CHANGE-OWNER	file/dir
fchown()	sys_fchown()	fs/open.c	CHANGE-OWNER	file/dir
close()	close_fp()	fs/open.c	CLOSE	file/dir
	sys_close()	fs/open.c	CLOSE	ipc: socket
creat()	siehe open()			
create_module()	sys_create_module()	kernel/module.c	ADD-TO-KERNEL	-
Fortsetzung auf der nächsten Seite…				

...*Fortsetzung Anpassung der Systemaufrufe*				
Systemaufruf	angepaßte Prozedur	Datei	ADF-Anforderung	Ziel
delete-_module()	sys_delete-_module()	kernel/module.c	REMOVE-FROM-KERNEL	-
exec_ve()	do_execve()	fs/exec.c	EXECUTE*	file
exit()	do_exit()	kernel/exit.c	TERMINATE	process
fork(), clone()	do_fork()	kernel/fork.c	CLONE*	process+process*
getdents()	sys-_getdents()	fs/readdir.c	READ	dir
init-_module()	sys_init-_module()	kernel/module.c	ADD-TO-KERNEL	-
ioperm()	sys_ioperm()	arch/i386/kernel/ioport.c	MODIFY-PERMISSIONS-DATA	scd:ioports
iopl()	sys_iopl()	arch/i386/kernel/ioport.c	MODIFY-PERMISSIONS-DATA	scd:ioports
ipc()	siehe msgget() etc.			
kill()	send_sig()	kernel/exit.c	SEND-SIGNAL (nicht für PID 0)	process, attr.signal = signal
link()	do_link()	fs/namei.c	LINK-HARD	file
mkdir()	do_mkdir()	fs/namei.c	CREATE*	dir+dir*, attr.mode = mode
mknod()	do_mknod()	fs/namei.c	CREATE*	dir+file*, attr.mode = mode
Fortsetzung auf der nächsten Seite...				

... Fortsetzung Anpassung der Systemaufrufe				
Systemaufruf	angepaßte Prozedur	Datei	ADF-Anforderung	Ziel
mount()	do_mount()	fs/super.c	MOUNT (direkter Aufruf von rsbac_mount()*)	dir
msgctl()	sys_msgctl()	ipc/msg.c	CHANGE-GROUP	ipc: msg
			CHANGE-OWNER	ipc: msg
			DELETE*	ipc: msg
			ALTER	ipc: msg
msgget()	new_que()	ipc/msg.c	CREATE*	ipc: msg
msgrcv()	real_msgrcv()	ipc/msg.c	READ-OPEN*	ipc: msg
			CLOSE	ipc: msg
msgsnd()	real_msgsnd()	ipc/msg.c	APPEND-OPEN*	ipc: msg
			CLOSE	ipc: msg
open()	open_namei()	fs/namei.c	READ	dir
			CREATE*	dir+file*, attr.mode = mode
			TRUNCATE	file
			APPEND-OPEN	file
			READ-OPEN	file
			READ-WRITE-OPEN	file
			WRITE-OPEN	file
	do_open() (nur Benachrichtigung)	fs/open.c	APPEND-OPEN*	file
Fortsetzung auf der nächsten Seite...				

... Fortsetzung Anpassung der Systemaufrufe				
Systemaufruf	angepaßte Prozedur	Datei	ADF-Anforderung	Ziel
			READ-OPEN*	file
			READ-WRITE-OPEN*	file
			WRITE-OPEN*	file
ptrace()	sys_ptrace()	arch/i386/-kernel/-ptrace.c	TRACE	process
readdir()	old_readdir()	fs/readdir.c	READ	dir
reboot()	sys_reboot()	kernel/sys.c	SHUTDOWN	-
rename()	do_rename()	fs/namei.c	RENAME	file/dir
			WRITE (bei Verschieben)	dir
rmdir()	do_rmdir()	fs/namei.c	DELETE*	dir
setdomain-name()	sys_setdo-mainname()	kernel/sys.c	MODIFY-SYSTEM-DATA	scd: net_id
setfsuid()	sys_setfsuid()	kernel/sys.c	CHANGE-OWNER	-, attr.owner = ruid
setfsgid()	sys_setfsgid()	kernel/sys.c	CHANGE-GROUP	-, attr.group = gid
setgroups()	sys-_setgroups()	kernel/sys.c	CHANGE-GROUP	process
sethostna-me()	sys-_sethostname()	kernel/sys.c	MODIFY-SYSTEM-DATA	scd: host-_id
setreuid()	sys-_setreuid()	kernel/sys.c	CHANGE-OWNER*	process, attr.owner = ruid
Fortsetzung auf der nächsten Seite...				

... *Fortsetzung Anpassung der Systemaufrufe*				
Systemaufruf	angepaßte Prozedur	Datei	ADF-Anforderung	Ziel
setregid()	sys-_setregid()	kernel/sys.c	CHANGE-GROUP	process, attr.group = gid
setrlimit()	sys-_setrlimit()	kernel/sys.c	MODIFY-SYSTEM-DATA	scd: rlimit
setuid()	sys_setuid()	kernel/sys.c	CHANGE-OWNER*	process, at-tr.owner = uid
setgid()	sys_setgid()	kernel/sys.c	CHANGE-GROUP	process, attr.group = gid
shmctl()	sys_shmctl()	ipc/shm.c	CHANGE-GROUP	ipc: shm
			CHANGE-OWNER	ipc: shm
			DELETE	ipc: shm
			ALTER	ipc: shm
	killseg()	ipc/shm.c	DELETE* (nur Benach-richtigung)	ipc: shm
shmget()	newseg()	ipc/shm.c	CREATE*	ipc: shm
shmat()	sys_shmat()	ipc/shm.c	READ-OPEN*	ipc: shm
			READ-WRITE-OPEN*	ipc: shm
shmdt()	shm_close()	ipc/shm.c	CLOSE	ipc: shm
socketcall()	sys_socket()	net/socket.c	CREATE*	ipc: socket
	sys_connect()	net/socket.c	READ-WRITE-OPEN*	ipc: socket, attr-.sockadr = Zieladr.
Fortsetzung auf der nächsten Seite...				

... Fortsetzung Anpassung der Systemaufrufe				
Systemaufruf	angepaßte Prozedur	Datei	ADF-Anforderung	Ziel
	sys_listen()	net/socket.c	READ-WRITE-OPEN*	ipc: socket
	sys_shutdown()	net/socket.c	DELETE*	ipc: socket
stat()	sys_stat()	fs/stat.c	GET-STATUS-DATA	file/dir
fstat()	sys_fstat()	fs/stat.c	GET-STATUS-DATA	file/dir
lstat()	sys_lstat()	fs/stat.c	GET-STATUS-DATA	file/dir
new_stat()	sys_new_stat()	fs/stat.c	GET-STATUS-DATA	file/dir
new_fstat()	sys_new_fstat()	fs/stat.c	GET-STATUS-DATA	file/dir
new_lstat()	sys_new_lstat()	fs/stat.c	GET-STATUS-DATA	file/dir
statfs()	sys_statfs()	fs/open.c	GET-STATUS-DATA	file/dir
fstatfs()	sys_fstatfs()	fs/open.c	GET-STATUS-DATA	file/dir
stime()	sys_stime()	kernel/time.c	MODIFY-SYSTEM-DATA	scd: clock
Fortsetzung auf der nächsten Seite...				

... Fortsetzung Anpassung der Systemaufrufe				
Systemaufruf	angepaßte Prozedur	Datei	ADF-Anforderung	Ziel
settimeofday()	sys_settime-ofday()	kernel/time.c	MODIFY-SYSTEM-DATA	scd: clock
swapon()	sys_swapon()	mm/swapfi-le.c	MODIFY-SYSTEM-DATA	scd: swap
swapoff()	sys_swapoff()	mm/swapfi-le.c	MODIFY-SYSTEM-DATA	scd: swap
symlink()	do_symlink()	fs/open.c	CREATE*	dir+file*, attr.mode = mode
syslog()	sys_syslog()	kernel/-printk.c	MODIFY-SYSTEM-DATA	scd: syslog
truncate()	sys_trunca-te()	fs/open.c	TRUNCATE	file
ftruncate()	sys_ftrunca-te()	fs/open.c	TRUNCATE	file
umount()	sys-_umount()	fs/super.c	UMOUNT (direkter Aufruf von rsbac-_umount()*)	dir
unlink()	sys_unlink()	fs/namei.c	DELETE*	file
utime()	sys_utime()	fs/open.c	MODIFY-ACCESS-DATA	file/dir
utimes()	sys_utimes()	fs/open.c	MODIFY-ACCESS-DATA	file/dir

Tabelle 6.1: Anpassung der Systemaufrufe

Fortsetzung auf der nächsten Seite...

... Fortsetzung Anpassung der Systemaufrufe				
Systemaufruf	angepaßte Prozedur	Datei	ADF-Anforderung	Ziel

6.3.2 Beispiele

Entscheidungsanfrage

```
/* RSBAC */
#ifdef CONFIG_RSBAC
union rsbac_target_id_t       rsbac_target_id;
union rsbac_target_id_t       rsbac_new_target_id;
union rsbac_attribute_value_t rsbac_attribute_value;
#endif

/* RSBAC */
#ifdef CONFIG_RSBAC
printk(KERN_DEBUG "do_fork() [sys_fork(),sys_clone()]: calling ADF\n");
rsbac_target_id.process = current->pid;
rsbac_attribute_value.dummy = 0;
if (!rsbac_adf_request(R_CLONE,
                       current->pid,
                       T_PROCESS,
                       rsbac_target_id,
                       A_none,
                       rsbac_attribute_value))
  {
    return -EPERM;
  }
#endif
```

Benachrichtigung

```
/* RSBAC: notify ADF of forked process */
#ifdef CONFIG_RSBAC
printk(KERN_DEBUG "do_fork() [sys_fork(),sys_clone()]: calling ADF_set_attr\n");
rsbac_new_target_id.process = p->pid;
if (rsbac_adf_set_attr(R_CLONE,
                       current->pid,
                       T_PROCESS,
                       rsbac_target_id,
                       T_PROCESS,
                       rsbac_new_target_id,
                       A_none,
```

```
                        rsbac_attribute_value))
 {
   printk(KERN_WARNING
          "do_fork() [sys_fork(), sys_clone()]: rsbac_adf_set_attr() returned error");
 }
#endif
```

6.4 Entscheidungskomponente (ADF)

6.4.1 Gliederung

Die Entscheidungskomponente gliedert sich in den Hauptteil und mehrere, voneinander unabhängige Entscheidungsmodule. Jedes Entscheidungsmodul kann getrennt in der Kernkonfiguration ein- oder ausgeschlossen werden. Außerdem kann dort festgelegt werden, ob sich die Module zur Laufzeit einzeln an- oder abschalten lassen. Ein Modul gilt dann als aktiv, wenn es eingeschlossen und angeschaltet ist.

Der Hauptteil der Entscheidungskomponente befindet sich in der Datei `adf/main.c`. Er besteht hauptsächlich aus den drei Funktionen rsbac_admin, rsbac_adf_request und rsbac_adf_set_attr. Die Deklaration erfolgt in `adf.h`:

```
extern  int  rsbac_admin(void);

extern  enum rsbac_adf_req_ret_t  rsbac_adf_request(
                                     enum  rsbac_adf_request_t,
                                           rsbac_pid_t,
                                     enum  rsbac_target_t,
                                     union rsbac_target_id_t,
                                     enum  rsbac_attribute_t,
                                     union rsbac_attribute_value_t);

extern  int  rsbac_adf_set_attr(     enum  rsbac_adf_request_t,
                                           rsbac_pid_t,
                                     enum  rsbac_target_t,
                                     union rsbac_target_id_t,
                                     enum  rsbac_target_t,
                                     union rsbac_target_id_t,
                                     enum  rsbac_attribute_t,
                                     union rsbac_attribute_value_t);
```

Die Funktion rsbac_admin ersetzt, falls sie in der Kernkonfiguration aktiviert wurde, die interne Linux-Funktion suser. Diese überprüft in vielen Systemaufrufen, ob der Besitzer des aufrufenden Prozesses der Benutzer root, also der Systemverwalter ist. Falls eines der Module MAC, FC oder SIM aktiv ist, überprüft rsbac_admin stattdessen, ob die System-Rolle des Benutzers Administrator ist. Ansonsten verhält sich die Funktion exakt wie suser.

rsbac_adf_request ist der zentrale Entscheidungsaufruf in der Durchsetzungskomponente. Hier werden die Grundinformationen über den aufrufenden Prozeß, wie der Prozeß-Besitzer, zusammengestellt und allgemeine Aufgaben wie das Löschen von Objekten wahrgenommen. Nach dem Aufruf aller aktiven Entscheidungsmodule wird das Gesamtergebnis ermittelt, abhängig von der Protokolleinstellung der Anfrage pseudonym protokolliert und zurückgegeben.

Die in Tabelle 4.7 definierte logische Verknüpfung der Einzelergebnisse "undplus" ist in der Funktion adf_and_plus umgesetzt:

```
static enum rsbac_adf_req_ret_t
    adf_and_plus(enum rsbac_adf_req_ret_t res1,
                 enum rsbac_adf_req_ret_t res2)
  {
    switch (res1)
      {
        case GRANTED:     if (res2 == DO_NOT_CARE)
                            return (GRANTED);
                          else
                            return (res2);
        case NOT_GRANTED: if (res2 == UNDEFINED)
                            return (UNDEFINED);
                          else
                            return (NOT_GRANTED);
        case DO_NOT_CARE: return (res2);
        default:          return (UNDEFINED);
      }
  };
```

Als Beispiel für einen Modulaufruf sei derjenige des FC-Moduls wiedergegeben. Die Funktionsparameter der Hauptfunktion werden unverändert weitergegeben und durch den Besitzer des aufrufenden Prozesses ergänzt. Da ein Vorwert "undefined" nicht mehr veränderbar ist, wird in diesem Fall zur Optimierung auf den Aufruf verzichtet.

```
/******* FC ********/
#ifdef CONFIG_RSBAC_FC
#ifdef CONFIG_RSBAC_SWITCH
if (rsbac_switch_fc)
#endif
  /* no need to call module, if already undefined */
  if(result != UNDEFINED)
    result = adf_and_plus(result, rsbac_adf_request_fc(request,
```

```
                                                  caller_pid,
                                                  target,
                                                  tid,
                                                  attr,
                                                  attr_val,
                                                  owner) );
#endif  /* FC */
```

Schließlich dient rsbac_adf_set_attr zur Benachrichtigung der Entscheidungskomponente. Entsprechend dem vorigen Aufruf werden auch hier die Prozeßinformationen zusammengestellt, allgemeine Aufgaben erledigt, die aktiven Module aufgerufen, protokolliert und ein Gesamtfehlercode zurückgegeben. Die zusätzlichen Parameter dienen gegebenenfalls zur Angabe eines neu erzeugten Objektes.

Jedes Entscheidungsmodul muß somit zumindest je eine Entscheidungs- und eine Benachrichtigungsfunktion enthalten.

In beiden Verteilerfunktionen ist die Protokollierung abhängig von der Protokollstufe, die der Anforderung zugeordnet wurde. Es gibt drei Stufen: kein Protokoll, nur bei Ablehnung oder undefiniert, sowie volles Protokoll. Die Voreinstellungen können mit Kernparametern beim Systemstart festgelegt werden, Änderungen sind dann für Sicherheitsbeauftragte mit dem Systemaufruf sys_rsbac_switch_adf_log zur Laufzeit möglich. Zur Protokollierung wird das Pseudonym des Besitzers des aufrufenden Prozesses verwendet.

Gesichtert werden Anforderung, Prozeß-ID, Programmname, Benutzer (mit Pseudonym, falls definiert), Zugriffsziel, Attribut mit Wert und die Gesamtentscheidung. Der Protokollteil der Funktion `rsbac_adf_request` sieht folgendermaßen aus:

```
/* logging request on debug level, if enabled for this request type */
/* loglevel 2: log everything */
/* loglevel 1: log, if denied */
/* loglevel 0: log nothing */

if (   rsbac_debug_adf[request] == 2
    || (   rsbac_debug_adf[request] == 1
        && (result == NOT_GRANTED
            || result == UNDEFINED)) )
  {
    char  request_name[80] = "";
    char  res_name[80] = "";
    char  target_type_name[80] = "";
    char  target_id_name[80] = "";
    char  attr_name[80] = "";
    char  command[17] ="";
    rsbac_pseudo_t  pseudo = 0;
```

```
    /* Get owner's logging pseudo */
    if (rsbac_get_attr(T_USER,target_id,A_pseudo,&i_attr_val))
      {
        printk(KERN_WARNING
               "rsbac_adf_request(): rsbac_get_attr() for pseudo returned error %i",
               error);
        return(NOT_GRANTED);  /* something weird happened */
      }
    /* if pseudo is not registered, return attribute value is 0 (see later) */
    pseudo = i_attr_val.pseudo;

    get_request_name(request_name, request);
    get_target_name(target_type_name, target, target_id_name, tid);
    get_attribute_name(attr_name, attr);
    get_result_name(res_name, result);
    if ((current) && (current->comm))
      {
        strncpy(command,current->comm,16);
        command[16] = (char) 0;
      }
    /* if pseudo is set, its value is != 0, else -> use id */
    if (pseudo)
      printk(KERN_DEBUG
             "rsbac_adf_request(): request %s, caller_pid %u, caller_proc_name %s,
               caller_pseudo %lu, target-type %s, tid %s, attr %s, value %lu, result %s\n",
             request_name, (u_int) caller_pid, command, (u_long) i_attr_val.pseudo,
             target_type_name, target_id_name, attr_name, attr_val.pseudo, res_name);
    else
      printk(KERN_DEBUG
             "rsbac_adf_request(): request %s, caller_pid %u, caller_proc_name %s,
               caller_uid %u, target-type %s, tid %s, attr %s, value %lu, result %s\n",
             request_name, (u_int) caller_pid, command, (u_int) target_id.user,
             target_type_name, target_id_name, attr_name, attr_val.pseudo, res_name);
  }
```

6.4.2 MAC

Das MAC-Modul definiert zunächst die in Abschnitt 5.6.3 genannten Funktionen auto-write, auto-read und auto-read-write. Da diese sowohl für die Entscheidung, als auch für die Attributsetzung verwendet werden, gibt der Parameter set_level an, ob ein automatisch gesetztes aktuelles Sicherheitslevel auch abzuspeichern ist:

```
static enum rsbac_adf_req_ret_t
           auto_write(     rsbac_pid_t             pid,
                      enum rsbac_security_level_t  target_sec_level,
                           boolean                 set_level);

static enum rsbac_adf_req_ret_t
           auto_read (     rsbac_pid_t             pid,
                      enum rsbac_security_level_t  target_sec_level,
                           boolean                 set_level)

static enum rsbac_adf_req_ret_t
     auto_read_write (     rsbac_pid_t             pid,
                      enum rsbac_security_level_t  target_sec_level,
                           boolean                 set_level)
```

Die Entscheidung erfolgt abhängig von der jeweiligen Anforderung als direkte Umsetzung der Entscheidungsgrundlagen in den Tabellen 5.8 und 5.9. Als Beispiel sei die Anforderung CREATE aufgeführt, bei der ein auto-write auf das Zielverzeichnis durchzuführen ist:

```
case R_CREATE:
  switch(target)
    { /* Creating dir or (pseudo) file IN target dir! */
      case T_DIR:
        /* Mode of created item is ignored! */
        /* test write access to target: get its sec_level */
        if (rsbac_get_attr(T_DIR,
                           tid,
                           A_security_level,
                           &i_attr_val1))
          {
            printk(KERN_WARNING
                   "rsbac_adf_request_mac(): rsbac_get_attr() returned error!\n");
            return(NOT_GRANTED);
          }
        /* and perform auto-write without setting attributes */
        return(auto_write(caller_pid,
                          i_attr_val1.security_level,
                          FALSE));
        break;

      /* IPC is always granted */
      case T_IPC:
        return(GRANTED);
```

```
    /* all other cases are undefined */
    default: return(UNDEFINED);
   }
```

Ist das Erzeugen eines neuen Objektes erfolgreich verlaufen, so wird die Benachrichtigungsfunktion mit Angabe des neuen Objektes aufgerufen. Sollte auto-write hier zu einem ablehnenden Ergebnis kommen, wird der Fehler DECISION-MISMATCH zurückgeliefert.

Für das neue Objekt sind anschließend die für dieses Modul relevanten Attribute korrekt zu initialisieren. Dazu wird auch auf andere Attribute zurückgegriffen. Der entsprechende Abschnitt sieht folgendermaßen aus:

```
case R_CREATE:
  switch(target)
    {
      /* Creating dir or (pseudo) file IN target dir! */
      case T_DIR:
        /* Mode of created item is ignored! */
        /* test write access to target: get its sec_level */
        if (rsbac_get_attr(T_DIR,
                           tid,
                           A_security_level,
                           &i_attr_val1))
          {
            printk(KERN_WARNING
                   "rsbac_adf_set_attr_mac(): rsbac_get_attr() returned error!\n");
            return(NOT_GRANTED);
          }
        /* and perform auto-write with setting of attributes */
        result = auto_write(caller_pid,
                            i_attr_val1.security_level,
                            TRUE);
        if ((result != GRANTED) && (result != DO_NOT_CARE))
          return(-RSBAC_EDECISIONMISMATCH);

        /* Get current-sec-level from process... */
        i_tid.process = caller_pid;
        if (rsbac_get_attr(T_PROCESS,
                           i_tid,
                           A_current_sec_level,
                           &i_attr_val1))
          {
            printk(KERN_WARNING
                   "rsbac_adf_set_attr_mac(): rsbac_get_attr() returned error!\n");
```

```
      return(-RSBAC_EREADFAILED);
    }
  /* Set security-level for new item */
  if (rsbac_set_attr(new_target,
                     new_tid,
                     A_security_level,
                     i_attr_val1))
    {
      printk(KERN_WARNING
             "rsbac_adf_set_attr_mac(): rsbac_set_attr() returned error!\n");
      return(-RSBAC_EWRITEFAILED);
    }
  /* Set object-type for new item */
  if (new_target == T_FILE)
    i_attr_val1.object_type = OT_file;
  else
    i_attr_val1.object_type = OT_dir;
  if (rsbac_set_attr(new_target,
                     new_tid,
                     A_object_type,
                     i_attr_val1))
    {
      printk(KERN_WARNING
             "rsbac_adf_set_attr_mac(): rsbac_set_attr() returned error!\n");
      return(-RSBAC_EWRITEFAILED);
    }
  return(0);
  break;

case T_IPC:
  i_tid.process = caller_pid;
  /* Get current-sec-level from process... */
  if (rsbac_get_attr(T_PROCESS,
                     i_tid,
                     A_current_sec_level,
                     &i_attr_val1))
    {
      printk(KERN_WARNING
             "rsbac_adf_set_attr_mac(): rsbac_get_attr() returned error!\n");
      return(-RSBAC_EREADFAILED);
    }
  /* Set security-level for this ipc item */
  if (rsbac_set_attr(T_IPC,
```

```
                        tid,
                        A_security_level,
                        i_attr_val1))
          {
            printk(KERN_WARNING
                   "rsbac_adf_set_attr_mac(): rsbac_set_attr() returned error!\n");
            return(-RSBAC_EWRITEFAILED);
          }
        /* Set object-type for this ipc item */
        i_attr_val1.object_type = OT_ipc;
        if (rsbac_set_attr(T_IPC,
                        tid,
                        A_object_type,
                        i_attr_val1))
          {
            printk(KERN_WARNING
                   "rsbac_adf_set_attr_mac(): rsbac_set_attr() returned error!\n");
            return(-RSBAC_EWRITEFAILED);
          }
        return(0);
        break;

      /* all other cases are undefined */
      default:
        return(-RSBAC_EINVALIDTARGET);
    }
```

Sollte eine Auto-Funktion bei dem Benachrichtigungsaufruf zu einem ablehnenden Ergebnis kommen, wird ein entsprechender Fehler zurückgeliefert.

6.4.3 CWI

Dieses Modul ist im Rahmen dieses Projektes noch nicht implementiert.

6.4.4 FC

Die funktionale Kontrolle definiert zunächst die Hilfsfunktion check_role_fc, die für ein Zugriffsziel und einen Benutzer die Kompatibilität von Kategorie und Rolle überprüft und eine entsprechende Entscheidung liefert:

```
static enum rsbac_adf_req_ret_t
         check_role_fc(   enum  rsbac_target_t          target,
                          union rsbac_target_id_t       tid,
                                rsbac_uid_t             owner);
```

Die eigentliche Entscheidungsfunktion verwendet, wie im folgenden Beispiel, für die meisten Anforderungen diese Rollenprüfung:

```
case R_READ_WRITE_OPEN:
  switch(target)
    {
      case T_FILE:
      case T_IPC:
        return(check_role_fc(target,tid,owner));

      /* all other cases are undefined */
      default: return(UNDEFINED);
    }
```

Die Benachrichtigungsfunktion beschränkt sich auf das Initialisieren der relevanten Attribute Kategorie, Besitzer und Typ für neu erzeugte Objekte.

6.4.5 SIM

Wie die funktionale Kontrolle verwendet auch dieses Modul eine Hilfsfunktion, check_role_sim. Diese überprüft beim Objekt-Datentyp "sicherheitsrelevant" die Rolle des Prozeßbesitzers. Ist diese Sicherheitsbeauftragter oder der Datentyp ein anderer, wird der Zugriff erlaubt, sonst verboten:

```
static enum rsbac_adf_req_ret_t
         check_role_sim(  enum  rsbac_target_t          target,
                          union rsbac_target_id_t       tid,
                                rsbac_uid_t             owner);
```

Die Rollenprüfung wird von der Entscheidungsfunktion für alle Anforderungen aufgerufen, die einen Schreibzugriff implizieren. Ansonsten wird der Wert "Egal" zurückgegeben.

Auch hier setzt die Benachrichtigungsfunktion lediglich die relevanten Attribute Objekttyp, Objekt-Datentyp und Besitzer für neue Objekte.

6.4.6 PM

Auch das Datenschutz-Modul verwendet diverse Hilfsfunktionen. Die erste heißt get_ipc_purpose und erleichtert den Zugriff auf den Zweck einer Interprozeßkommunikation. Die zweite, tp_check, verweigert den Zugriff, falls der aufrufende Prozeß eine TP ist, und erlaubt ihn andernfalls. Diese Funktion wird benutzt, um Schreibzugriffe von TP auf nicht-persönliche Daten und damit einen unerwünschten Informationsfluß zu verhindern.

Die nachfolgenden Funktionen bilden die häufigsten Entscheidungsgrundlagen ab, wie sie in der Tabelle 5.12 zu finden sind. Dabei sind die Funktionsnamen als

Abkürzungen zu betrachten. na_and_pp_or_cs überprüft für Dateien die Bedingung “Notwendig (*Zugriff*) und (Zweckbindung oder Einwilligung)”, na_and_pp_ipc für Interprozeßkommunikation “Notwendig (*Zugriff*) und Zweckbindung”, sowie na_ipc für Interprozeßkommunikation “Notwendig (*Zugriff*)”. Die Funktionsköpfe sehen folgendermaßen aus:

```
static rsbac_pm_purpose_id_t
  get_ipc_purpose(struct rsbac_ipc_t ipc_id);

static enum rsbac_adf_req_ret_t
  tp_check(rsbac_pid_t caller_pid);

static enum rsbac_adf_req_ret_t
  na_and_pp_or_cs(       rsbac_pid_t          caller_pid,
                  struct rsbac_fs_file_t      file,
                         rsbac_pm_accesses_t  acc);

static enum rsbac_adf_req_ret_t
  na_and_pp_ipc(       rsbac_pm_task_id_t   task,
                       rsbac_pid_t          caller_pid,
                       rsbac_pm_accesses_t  acc,
                struct rsbac_ipc_t          ipc_id);

static enum rsbac_adf_req_ret_t
  na_ipc(rsbac_pm_task_id_t   task,
         rsbac_pid_t          caller_pid,
         rsbac_pm_accesses_t  acc);
```

In der Entscheidungsfunktion rsbac_adf_request_pm werden diese Hilfsfunktionen dann, soweit möglich, für die Entscheidungsfindungen entsprechend den Entscheidungsgrundlagen der Tabellen 5.11 und 5.12 benutzt.

Am Beispiel der Anforderung APPEND-OPEN läßt sich das Zusammenspiel nachvollziehen:

```
case R_APPEND_OPEN:
    switch(target)
      {
        case T_FILE:
          /* get pm_object_type of target */
          if (rsbac_get_attr(T_FILE,
                             tid,
                             A_pm_object_type,
                             &i_attr_val1))
            {
```

```
      printk(KERN_WARNING
             "rsbac_adf_request_pm(): rsbac_get_attr() returned error!\n");
      return(NOT_GRANTED);
    }
  /* no append_open on TPs */
  if(i_attr_val1.pm_object_type == PO_TP)
    return(NOT_GRANTED);
  /* TPs must not write on other than personal_data */
  if(i_attr_val1.pm_object_type != PO_personal_data)
    return(tp_check(caller_pid));

  /* check necessary && (purpose_bind || consent) */
  return(na_and_pp_or_cs(caller_pid,
                         tid.file,
                         RSBAC_PM_A_APPEND));
  break;

case T_IPC:
  /* get IPC-purpose */
  i_pm_pp = get_ipc_purpose(tid.ipc);
  /* if IPC-pp is NIL -> process type must be NIL */
  if(!i_pm_pp)
    {
      /* get process-type of caller-process */
      i_tid.process = caller_pid;
      if (rsbac_get_attr(T_PROCESS,
                         i_tid,
                         A_pm_process_type,
                         &i_attr_val1))
        {
          printk(KERN_WARNING
                 "rsbac_adf_request_pm(): rsbac_get_attr() returned error!\n");
          return(NOT_GRANTED);
        }
      if(i_attr_val1.pm_process_type == PP_TP)
        return(NOT_GRANTED);
      else
        return(GRANTED);
    }
  /* OK, we do have an IPC-purpose */
  /* get current_task of caller-process */
  i_tid.process = caller_pid;
  if (rsbac_get_attr(T_PROCESS,
```

```
                         i_tid,
                         A_pm_current_task,
                         &i_attr_val1))
              {
                printk(KERN_WARNING
                       "rsbac_adf_request_pm(): rsbac_get_attr() returned error!\n");
                return(NOT_GRANTED);
              }
            /* if current_task = NIL -> do not grant */
            if(!i_attr_val1.pm_current_task)
              {
                  return(NOT_GRANTED);
              }
            /* check necessary && purpose_bind */
            return(na_and_pp_ipc(i_attr_val1.pm_current_task,
                                 caller_pid,
                                 RSBAC_PM_A_APPEND,
                                 tid.ipc));
            break;

          /* all other cases are undefined */
          default: return(UNDEFINED);
        }
```

Wie schon in mehreren anderen Modulen initialisiert auch in diesem die Benachrichtigungsfunktion lediglich PM-relevante Attribute für neue Objekte. Die Benachrichtigung EXECUTE setzt außerdem bei der Ausführung von als TP gekennzeichneten Programmen die entsprechenden Prozeßattribute:

```
case R_EXECUTE:
  switch(target)
    {
      case T_FILE:
        /* get pm_object_type of target */
        if (rsbac_get_attr(T_FILE,
                           tid,
                           A_pm_object_type,
                           &i_attr_val1))
          {
            printk(KERN_WARNING
                   "rsbac_adf_request_pm(): rsbac_get_attr() returned error!\n");
            return(-RSBAC_EREADFAILED);
          }
        /* if not TP: do nothing */
```

```
  if(i_attr_val1.pm_object_type != PO_TP)
    return(0);
  /* get pm_tp of target */
  if (rsbac_get_attr(T_FILE,
                     tid,
                     A_pm_tp,
                     &i_attr_val1))
    {
      printk(KERN_WARNING
             "rsbac_adf_request_pm(): rsbac_get_attr() returned error!\n");
      return(-RSBAC_EREADFAILED);
    }
  /* if no tp: error! */
  if(!i_attr_val1.pm_tp)
    {
      printk(KERN_WARNING
             "rsbac_adf_request_pm(): file with object_type TP has no tp_id!\n");
      return(-RSBAC_EINVALIDVALUE);
    }
  /* Set pm_tp for this process */
  i_tid.process = caller_pid;
  if (rsbac_set_attr(T_PROCESS,
                     i_tid,
                     A_pm_tp,
                     i_attr_val1))
    {
      printk(KERN_WARNING
             "rsbac_adf_set_attr_pm(): rsbac_set_attr() returned error!\n");
      return(-RSBAC_EWRITEFAILED);
    }
  /* Set pm_process_type for this process */
  i_attr_val1.pm_process_type = PP_TP;
  if (rsbac_set_attr(T_PROCESS,
                     i_tid,
                     A_pm_process_type,
                     i_attr_val1))
    {
      printk(KERN_WARNING
             "rsbac_adf_set_attr_pm(): rsbac_set_attr() returned error!\n");
      return(-RSBAC_EWRITEFAILED);
    }
  return(0);
/* all other cases are undefined */
```

```
    default:
      return(-RSBAC_EINVALIDTARGET);
  }
```

6.5 Zusätzliche Systemaufrufe

Im Rahmen der Systemerweiterungen wurden zusätzlich zu den vorhandenen sowohl allgemeine, also modellunabhängige, als auch modellspezifische Systemaufrufe notwendig. Ich habe mich bemüht, letztere auf ein Minimum zu beschränken.

6.5.1 Allgemeine Aufrufe

Um aus der Prozeßebene Attributwerte setzen und auslesen zu können, sind entsprechende Systemaufrufe verfügbar, deren Ausführung aber ebenfalls der vollen Zugriffskontrolle unterliegt. Die jeweiligen Anfragen finden sich weiter unten in Abschnitt 6.5.3.

Deklariert und definiert werden die Systemaufrufe in den Dateien `syscalls.h` und `syscalls.c`. Bei den Zugriffssystemaufrufen gibt es jeweils eine Funktion, die als Zielbezeichnung die interne Identifikationsnummer verlangt, und eine, die stattdessen die Angabe von Namen erlaubt. Die Namensumsetzung ist zur Zeit nur für Dateien und Verzeichnisse implementiert. Der eigentliche Zugriff auf die Daten erfolgt mit den Zugriffsfunktionen der allgemeinen Datenhaltung aus Abschnitt 6.1.5.

Die Deklaration wird mit Hilfsmakros vorgenommen und sieht vereinfacht folgendermaßen aus:

```
int rsbac_get_attr (enum  rsbac_target_t,
                    union rsbac_target_id_t *,
                    enum  rsbac_attribute_t,
                    union rsbac_attribute_value_t *);

int rsbac_get_attr_n (enum  rsbac_target_t,
                            char *,
                      enum  rsbac_attribute_t,
                      union rsbac_attribute_value_t *);

int rsbac_set_attr (enum  rsbac_target_t,
                    union rsbac_target_id_t *,
                    enum  rsbac_attribute_t,
                    union rsbac_attribute_value_t *);

int rsbac_set_attr_n (enum  rsbac_target_t,
                            char *,
```

```
                     enum  rsbac_attribute_t,
                     union rsbac_attribute_value_t *);

int rsbac_remove_target (enum  rsbac_target_t,
                        union rsbac_target_id_t *);

int rsbac_remove_target_n (enum  rsbac_target_t,
                                char *);
int rsbac_stats ();
```

6.5.2 Modellspezifische Aufrufe

Die modellspezifischen Systemaufrufe sind ebenfalls in `syscalls.h` deklariert und in `help/syscalls.c` definiert. Für diejenigen Funktionen, für die jeweils nur vom betroffenen Modul Zugriffsentscheidungen zu treffen sind, erfolgt keine ADF-Anfrage, sondern jeweils der direkte Aufruf einer Hilfsfunktion aus dem ADF-Modul. Ein Beispiel ist der Aufruf sys_rsbac_pm:

```
asmlinkage int sys_rsbac_pm(enum  rsbac_pm_function_type_t    function,
                            union rsbac_pm_function_param_t * param_p,
                                  rsbac_pm_tkt_id_t           ticket)
  {
#ifndef CONFIG_RSBAC_PM
    return (0);
#else
    union  rsbac_pm_function_param_t k_param;

    if (rsbac_debug_aef_pm)
      printk(KERN_DEBUG "sys_rsbac_pm(): called for function %i!\n",
             function);
    /* get parameters from user space */
    rsbac_get_user((u_char *) &k_param, (u_char *) param_p, sizeof(k_param) );
    /* call pm function and return its result */
    return(rsbac_pm(function, k_param,ticket));
#endif
  }
```

MAC

Der Einfachheit und Trennung halber habe ich für die Politik MAC zwei Systemaufrufe hinzugefügt, mit denen ein Prozeß sein aktuelles Sicherheitslevel setzen und auslesen kann. Diese beiden in Abschnitt 5.7.2 erläuterten Systemaufrufe werden in den Funktionen rsbac_mac_set_curr_seclevel und rsbac_mac_get_curr_seclevel in der Datei `adf/mac/syscalls.c` direkt umgesetzt.

Wie in Abschnitt 5.6.3 gefordert, wird beim Setzen des aktuellen Sicherheitslevels das Prozeß-Attribut "mac-auto" auf "false" gesetzt und damit ab sofort keine automatische Anpassung mehr vorgenommen. Außerdem werden die Schranken max-read-open und min-write-open berücksichtigt. Die beiden Aufrufe sind wie folgt deklariert:

```
int  rsbac_mac_set_curr_seclevel(enum rsbac_security_level_t);

enum rsbac_security_level_t  rsbac_mac_get_curr_seclevel(void);
```

PM

Die Verwaltung des Datenschutzmoduls erfolgt ausschließlich über vier eigene Systemaufrufe, die in der Datei `adf/pm/syscalls.c` implementiert sind. Der erste Aufruf verursacht die Ausgabe des Modulstatus über die oben genannte Zugriffsfunktion rsbac_stats_pm(). Zwei weitere Aufrufe erlauben einem Prozeß, seine aktuelle Aufgabe zu setzen und ein Objekt einer bestimmten Klasse zu erzeugen.

Die Deklarationen lauten wie folgt:

```
int sys_rsbac_stats_pm(void);

int sys_rsbac_pm_change_current_task(rsbac_pm_task_id_t);

int sys_rsbac_pm_create_file(const char *,                /* name  */
                             int,                         /* mode  */
                             rsbac_pm_object_class_id_t); /* class */

int sys_rsbac_pm(enum  rsbac_pm_function_type_t,
                 union rsbac_pm_function_param_t *,
                       rsbac_pm_tkt_id_t);
```

Der vierte Aufruf ist für die eigentliche Verwaltung zuständig. Er dient als Verteilfunktion für die privilegierten Kontrollfunktionen, für die die benötigten Parameter zu übergeben sind. Für die meisten Unterfunktionen werden Tickets vergeben und bei einem weiteren Aufruf zur eigentlichen Durchführung zur Verfügung gestellt.

Die für diesen Systemaufruf verwendeten Datentypen finden sich in `pm_ticket.h`:

```
enum    rsbac_pm_function_type_t     {/* tkt issued by data_prot_officer, */
                                      /* called by security_officer */
                                      PF_add_na, PF_delete_na, PF_add_task,
                                      PF_delete_task, PF_add_object_class,
```

```
                                    PF_delete_object_class,
                                    PF_add_authorized_tp,
                                    PF_delete_authorized_tp,
                                    PF_add_consent, PF_delete_consent,
                                    PF_add_purpose, PF_delete_purpose,
                                    PF_add_responsible_user,
                                    PF_delete_responsible_user,
                                    PF_delete_user_aci,
                                    PF_set_role,
                                    PF_set_object_class,
                                    PF_switch_pm,
                                    /* tkt issued by data_prot_officer and */
                                    /* resp. user, called by security_officer */
                                    PF_add_authorized_task,
                                    PF_delete_authorized_task,
                                    /* called by sec_off or data_prot, no ticket */
                                    PF_write_list,
                                    /* called by tp_manager, no ticket */
                                    PF_create_tp, PF_delete_tp, PF_set_tp,
                                    /* called by data_prot_officer and */
                                    /* responsible user */
                                    PF_create_ticket,
                                    /* never to be called, internal */
                                    PF_none};

union   rsbac_pm_function_param_t
         {
           struct rsbac_pm_add_na_t                  add_na;
           struct rsbac_pm_delete_na_t               delete_na;
           struct rsbac_pm_add_task_t                add_task;
           struct rsbac_pm_delete_task_t             delete_task;
           struct rsbac_pm_add_object_class_t        add_object_class;
           struct rsbac_pm_delete_object_class_t     delete_object_class;
           struct rsbac_pm_add_authorized_tp_t       add_authorized_tp;
           struct rsbac_pm_delete_authorized_tp_t    delete_authorized_tp;
           struct rsbac_pm_add_consent_t             add_consent;
           struct rsbac_pm_delete_consent_t          delete_consent;
           struct rsbac_pm_add_purpose_t             add_purpose;
           struct rsbac_pm_delete_purpose_t          delete_purpose;
           struct rsbac_pm_add_responsible_user_t    add_responsible_user;
           struct rsbac_pm_delete_responsible_user_t delete_responsible_user;
           struct rsbac_pm_delete_user_aci_t         delete_user_aci;
           struct rsbac_pm_set_role_t                set_role;
```

```
    struct rsbac_pm_set_object_class_t         set_object_class;
    struct rsbac_pm_switch_pm_t                switch_pm;
    struct rsbac_pm_add_authorized_task_t      add_authorized_task;
    struct rsbac_pm_delete_authorized_task_t   delete_authorized_task;
    struct rsbac_pm_write_list_t               write_list;
    struct rsbac_pm_create_tp_t                create_tp;
    struct rsbac_pm_delete_tp_t                delete_tp;
    struct rsbac_pm_set_tp_t                   set_tp;
    struct rsbac_pm_create_ticket_t            create_ticket;
    int                                        dummy;
  };
```

Die Funktionsparameter sind jeweils in einer eigenen Struktur abgelegt. Als Beispiel sei diejenige für die Funktion add_na zitiert:

```
struct rsbac_pm_add_na_t
  {
    rsbac_pm_task_id_t            task;
    rsbac_pm_object_class_id_t    class;
    rsbac_pm_tp_id_t              tp;
    rsbac_pm_accesses_t           accesses;
  };
```

6.5.3 Entscheidungsanfragen in den neuen Systemaufrufen

Auch die hinzugefügten Systemaufrufe unterliegen der Zugriffsentscheidung und enthalten folgende Anfragen an die Entscheidungskomponente:

Systemaufruf	angepaßte Prozedur	Datei	ADF-Anforderung	Ziel
rsbac_stats()	sys_rsbac_stats()	rsbac/help/syscalls.c	GET-STATUS-DATA	scd: rsbac
rsbac_get_attr()	sys_rsbac_get_attr()	rsbac/help/syscalls.c	READ-ATTRIBUTE	Alle, attr: Alle
rsbac_get_attr_n()	sys_rsbac_get_attr_n()	rsbac/help/syscalls.c	READ-ATTRIBUTE	Alle, attr: Alle
Fortsetzung auf der nächsten Seite…				

... Fortsetzung ADF-Anfragen der neuen Systemaufrufe				
Systemaufruf	angepaßte Prozedur	Datei	ADF-Anforderung	Ziel
rsbac_set-_attr()	sys_rsbac-_set_attr()	rsbac/help/-syscalls.c	MODIFY-ATTRIBUTE	Alle, attr: Alle
rsbac_set-_attr_n()	sys_rsbac-_set_attr_n()	rsbac/help/-syscalls.c	MODIFY-ATTRIBUTE	Alle, attr: Alle
rsbac-_remove-_target()	sys_rsbac-_remove-_target()	rsbac/help/-syscalls.c	MODIFY-ATTRIBUTE	Alle, attr: none
rsbac-_remove-_target_n()	sys_rsbac-_remove-_target_n()	rsbac/help/-syscalls.c	MODIFY-ATTRIBUTE	Alle, attr: none
rsbac-_switch()	sys_rsbac-_switch()	rsbac/help/-syscalls.c	SWITCH-MODULE	-, attr-.switch-_target = module
rsbac_adf-_log_switch()	sys_rsbac-_adf_log-_switch()	rsbac/help/-syscalls.c	SWITCH-LOG	-
rsbac_stats-_pm()	sys_rsbac-_stats_pm()	rsbac/help/-syscalls.c	GET-STATUS-DATA	scd: rsbac

Tabelle 6.2: ADF-Anfragen der neuen Systemaufrufe

Kapitel 7

Installation und Administration der “Rule Set Based Access Control”

Die Quellen des RSBAC-Paketes sind in zwei Teile unterteilt, die Kernerweiterungen in der Datei `rsbac-patch.gz` und die Administrationswerkzeuge in der Datei `rsbac-admin.tar.gz`, die getrennt zu installieren sind. Beide Dateien sind, wie diese Arbeit, über das Internet unter

http://agn-www.informatik.uni-hamburg.de/people/1ott/rsbac

zu beziehen.

Grundkenntnisse im Umgang mit Unix-Systemen werden in diesem Kapitel vorausgesetzt.

7.1 Systemkern

7.1.1 Installation der Quellen

Zwingende Grundlage der Installation sind die Linux-Kernquellen in der Version 2.0.30. Diese sind üblicherweise in einer Datei namens `linux-2.0.30.tar.gz` erhältlich, die z. B. im Verzeichnis `/usr/src` mit dem Befehl `tar xvzf linux-2.0.30-.tar.gz` zu entpacken ist. Dabei wird ein Unterverzeichnis `linux` erzeugt, das in `linux-2.0.30-rsbac` umbenannt werden sollte, um Verwechslungen zu vermeiden. Existieren bereits ein Verzeichnis oder eine Datei mit dem Namen `linux`, sind diese natürlich vorher zu löschen oder umzubenennen. Für den gesamten Quellbaum inklusive RSBAC sind mindestens 30 MB zu veranschlagen.

Anschließend ist die Datei `rsbac-patch.gz` mit `gzip -d rsbac-patch.gz` zu dekomprimieren. Ich empfehle, die entstandene Datei `rsbac-patch` ebenfalls in das Verzeichnis `/usr/src` zu kopieren. Ist das geschehen, können im Kernquell-Hauptverzeichnis `/usr/src/linux-2.0.30-rsbac` mit dem Befehl `patch -p1`

</usr/src/rsbac-patch die Kernerweiterungen eingespielt werden. Da das Protokoll auf die Standard-Fehlerausgabe geschrieben wird, kann durch Anhängen von 2>/usr/src/err an den Befehl eine Protokolldatei erzeugt und anschließend überprüft werden. Schlägt die Erweiterung fehl, sollte der gesamte Quellbaum neu erzeugt werden, da eine Wiederholung der Anpassung problematisch ist.

Bei der Erweiterung eines bereits vorhandenen und anderweitig veränderten Quellbaumes kann es passieren, daß Fehler entstehen oder Teile zurückgewiesen werden. Dann ist es sinnvoll, eine Protokolldatei erzeugen zu lassen und gründlich zu prüfen, ob alle Änderungen wie vorgesehen durchgeführt wurden. Selbstverständlich sollte vor jeder Änderung eine Sicherheitskopie des Urzustandes angelegt werden.

7.1.2 Kernkonfiguration und Übersetzung

Die Konfiguration des Kernes erfolgt, wie in Unterkapitel 3.5 beschrieben, durch Eingabe von `make menuconfig` im Quellhauptverzeichnis. Das Konfigurationsmenü enthält jetzt ein weiteres Untermenü "Rule Set Based Access Control", in dem die Einstellungen der folgenden Tabelle aktiviert werden können:

Menüpunkt	Bedeutung
Rule Set Based Access Control	Hauptschalter für sämtliche Erweiterungen. Ist dieser ausgeschaltet, verhält sich der Kern exakt wie die Originalversion. Alle weiteren Punkte erscheinen nur, wenn der Hauptschalter aktiviert ist.
RSBAC support for MAC policy	Aktivierung der mandatorischen Kontrolle
RSBAC support for CWI policy	Clark-Wilson-Modul, leider noch ohne Funktion.
RSBAC support for FC policy	Funktionale Kontrolle
RSBAC support for SIM policy	Kontrolle der Modifikation von Sicherheitsinformationen
RSBAC support for PM policy	Datenschutzmodul
RSBAC policies switchable	An- und Abschalten einzelner Entscheidungsmodule zur Laufzeit, je nach Modul verschiedene Berechtigungen notwendig.
Fortsetzung auf der nächsten Seite...	

... Fortsetzung Menüpunkte zur Rule Set Based Access Control	
Menüpunkt	Bedeutung
RSBAC net support	Kontrolle von Netzwerk-Verbindungen
RSBAC check sys_syslog	Überprüfung des Systemaufrufes sys_syslog, verursacht spürbare Zusatzbelastung des Systems ohne großen Sicherheitsnutzen und sollte deaktiviert bleiben
RSBAC enhanced admin check	Erscheint nur, wenn eine der Politiken MAC, FC oder SIM aktiviert wurde. Ändert die Überprüfung des Benutzers auf Kennung "root" (Unix-Systemverwalter) in eine Überprüfung auf die Systemrolle "Systemadministrator", falls eine der genannten Politiken aktiv ist.[1] Kann die Stabilität des Kerns reduzieren und ist nur dann sinnvoll, wenn mehrere Administratorkennungen verwendet werden sollen.

Tabelle 7.1: Menüpunkte zur Rule Set Based Access Control

Nach der Konfiguration kann der Kern wie gewohnt übersetzt und installiert werden. Vor dem Neustart mit diesem Kern ist unbedingt eine der `user_aci`-Dateien aus dem Administrationspaket in das Verzeichnis `/rsbac` zu kopieren, da der Systemstart ohne den eingetragenen Administrator "root" nicht möglich ist! Außerdem sollten ein Sicherheitsbeauftragter mit der Benutzernummer 400 und, falls das Datenschutzmodell verwendet werden soll, ein Datenschutzbeauftragter mit der Nummer 401 eingerichtet werden. Die verwendeten Kennungen können dann später geändert werden.

Erscheint beim Systemstart eine Meldung "rsbac_init(): User ACI could not be read" und fährt das System nicht vollständig hoch, so ist die kopierte Datei für das verwendete System unbrauchbar. In diesem Fall sind mit einem Warmstart ein unveränderter Kern hochzufahren, in der Kernkonfiguration alle Politiken zu deaktivieren sowie ein neuer Wartungskern zu erzeugen und ebenfalls in das LILO-Startmenü einzutragen. Mit dem Wartungskern können die Administrationswerkzeuge verwendet werden, um die benötigten Rollen neu zu setzen.

7.1.3 Kernparameter

Beim Start des Systems ermöglicht der Linux-Loader (LILO), dem Kern Parameter zu übergeben, die sein Verhalten beeinflussen. Die von den RSBAC-Erweiterungen akzeptierten Parameter zeigt die folgende Tabelle:

Parameter	Bedeutung
rsbac_debug_ds	Erweiterte Ablauf- und Fehlermeldungen der Datenhaltung
rsbac_debug_aef	Erweiterte Ablauf- und Fehlermeldungen der Durchsetzung
rsbac_debug_adf	Voreinstellung: Protokollierung abgelehnter Anforderungen an die Entscheidungskomponente (für den Testbetrieb dringend, später bedarfsabhängig zu empfehlen)
rsbac_debug_adf_all	Voreinstellung: Protokollierung aller Anforderungen an die Entscheidungskomponente (erzeugt sehr viele Daten!)
rsbac_debug_ds_pm	Erweiterte Ablauf- und Fehlermeldungen der Datenhaltung für das Datenschutzmodul
rsbac_debug_aef_pm	Erweiterte Ablauf- und Fehlermeldungen der PM-Systemaufrufe
rsbac_debug_adf_pm	Erweiterte Ablauf- und Fehlermeldungen der Zugriffskontrolle in PM-Systemaufrufe
rsbac_debug_pm	Setzt rsbac_debug_ds_pm, rsbac_debug_aef_pm, rsbac_debug_adf_pm (für den PM-Testbetrieb ebenfalls zu empfehlen)
rsbac_debug_all	Setzt rsbac_debug_ds, rsbac_debug_aef, rsbac_debug_adf_all, rsbac_debug_pm

Tabelle 7.2: Zusätzliche Kern-Start-Parameter

7.2 Administrationswerkzeuge

7.2.1 Installation

Die Administrationswerkzeuge in der Datei `rsbac-admin.tar.gz` können in einem beliebigen Verzeichnis, z. B. `/root/rsbac`, mit dem Befehl `tar xvzf rsbac-admin-.tar.gz` ausgepackt werden. Danach sind in `Makefile` die Pfade zum Verzeichnis der Kernquellen, für die Programminstallation und für die Hilfe-Seiten (man-Seiten) anzupassen.

Mit `make` werden die Hilfsprogramme erzeugt, mit `make install` installiert und mit `make allclean` das Verzeichnis wieder aufgeräumt. Falls sie nicht existiert, erzeugt `make install` außerdem die Datei `/rsbac/useraci`, die die benötigten Benutzerattribute für den Systemstart und die Administration enthält.

7.2.2 Allgemeine Administration

Die allgemeine Administration erfolgt ausschließlich über Kommandozeilen. Dazu existieren mehrere Hilfsprogramme, die sich der in Abschnitt 6.5.1 vorgestellten Systemaufrufe bedienen. Jedes dieser Programm gibt beim Aufruf ohne Parameter eine kurze Hilfe aus. Die Überprüfung der Berechtigung des jeweiligen Aufrufes erfolgt wie beschrieben in den Systemaufrufen. Die folgende Tabelle bietet eine kurze Beschreibung der Befehle:

Programm	Bedeutung
attr_get_fd	Auslesen von Attributwerten von FILE- und DIR-Objekten aus der allgemeinen Datenhaltung. Die Parameter sind Zieltyp (FILE oder DIR), Attributname und die Namen der Dateien oder Verzeichnisse, deren Attribute auszulesen sind. Die Ausgabe erfolgt als Zahlenwert des jeweiligen Wertes.
attr_set_fd	Setzen entsprechender Attributwerte. Die Parameter sind Zieltyp (FILE oder DIR), Attributname, Zahlenwert und die Namen der Dateien oder Verzeichnisse, deren Attribute zu setzen sind.
attr_get_up	Auslesen von Attributwerten von USER- und PROCESS-Objekten aus der allgemeinen Datenhaltung. Die Parameter sind Zieltyp (USER oder PROCESS), Attributname und die Nummer des Objektes, dessen Attribut auszulesen ist. Die Ausgabe erfolgt als Zahlenwert des jeweiligen Wertes.
attr_set_up	Setzen entsprechender Attributwerte. Die Parameter sind Zieltyp (USER oder PROCESS), Attributname, Zahlenwert und die Nummer des Objektes, dessen Attribut zu setzen ist.
attr_rm_user	Löschen aller Attribute der angegebenen Benutzer
rsbac_stats	Ausgabe des Status der allgemeinen Datenhaltung über das System-Log.
switch_adf_log	Festlegen der Protokollstufe eines Systemaufrufes.[2] Die Parameter sind Name der Anforderung und Stufe (0-2).
switch_module	An- und Abschalten von Modulen, falls aktiviert. Die Parameter sind Modulname und 0 oder 1.

Tabelle 7.3: Werkzeuge zur allgemeinen Administration

Bis auf das Datenschutzmodell erfolgt die gesamte Administration aller implementierten Sicherheitsmodelle durch das Setzen entsprechender Attribute. Dieses ist aber nur Benutzern mit der System-Rolle Sicherheitsbeauftragter gestattet, die bei der Installation für Benutzer Nr. 400 eingetragen wird. Ist kein Sicherheitsbeauftragter vorhanden, muß, wie in Abschnitt 7.1.2 beschrieben, ein Wartungskern ohne Sicherheitsmodule erzeugt und mit diesem die Neuverteilung der Rollen vorgenommen werden.

Die zu setzenden Attribute sind in Abschnitt 6.1.1 bereits beschrieben worden. In Abschnitt 6.1.5 wurden außerdem die den verwendeten Systemaufrufen zugrundeliegenden Datentypen vorgestellt, in denen sich alle verwendbaren Attribute wiederfinden. Die exakte Schreibweise ist der jeweiligen Kurzhilfe der Befehle zu entnehmen. Eine eventuell benötigte Ordnungsnummer ergibt sich aus dem Datentyp des Attributs und ist ebenfalls der Kurzhilfe zu entnehmen.

7.2.3 Administration des PM-Moduls

Sobald das Datenschutzmodul aktiv ist, können keine für dieses Modul relevanten Attribute mit den bisher genannten Befehlen mehr gesetzt werden. Stattdessen erfolgt die gesamte Verwaltung mit dem Programm `rsbac_pm`, das hauptsächlich eine Eingabeschnittstelle zu dem gleichlautenden, Ticket-basierten Systemaufruf darstellt, der in Abschnitt 6.5.2 vorgestellt wurde. Allerdings können hier an Stelle von Ordnungsnummern auch die jeweiligen Bezeichnungen verwendet werden, beispielsweise `security_officer` bei der Funktion `set_role`.

Da die Vergabe von PM-Rollen nur mit Tickets möglich ist, muß jeweils mindestens ein Benutzer mit der Rolle Sicherheitsbeauftragter und der Rolle Datenschutzbeauftragter existieren, was bei den Installation sichergestellt sein sollte. Ansonsten sind der schon beschriebene Wartungskern zu installieren und die Rollen mit `attr_set_up USER pm_role Ordnungsnummer Benutzernummer` zu setzen.

7.2.4 Weitere Programme

Bei den Administrationswerkzeugen befinden sich noch weitere Programme, deren Verwendung hier kurz erläutert werden soll. Selbstverständlich unterliegen diese Programme der Zugriffskontrolle.

Programm	Bedeutung
rsbac_pm	Komplette Administration des PM-Moduls, siehe Abschnitte 7.2.3 und 6.5.2
rsbac_stats-_pm	Ausgabe des Status der Datenhaltung des PM-Moduls über das System-Log.
pm_create	Erzeugen von Dateien mit angegebener PM-Objektklasse. Die Parameter sind Objektklasse, diskreter Modus und Namen der Dateien.

Fortsetzung auf der nächsten Seite…

... Fortsetzung Weitere Werkzeuge	
Menüpunkt	Bedeutung
pm_ct_exec	Ausführen eines Programmes mit angegebener aktueller Aufgabe, das diese nicht selbsttätig setzen kann. Die Parameter sind aktuelle Aufgabe, Programmname und dessen Parameter.

Tabelle 7.4: Weitere Werkzeuge

7.3 Anwendungsbeispiel

Zur Demonstration des Systems ist in Zusammenarbeit mit Simone Fischer-Hübner ein vereinfachtes Anwendungsbeispiel entstanden, das zwar mehrere der implementierten Sicherheitsmodelle verwendet, sich aber auf das umfangreichste, das Datenschutzmodell, konzentriert. Weitere Module können natürlich jederzeit dazugeschaltet werden.

7.3.1 Aufgabenstellung

Für eine kleinere Operationsklinik ist elektronische Datenverarbeitung zu betreiben. Dabei soll der Datenschutz für sämtliche Patientendaten gewährleistet, aber auch Forschung in Form von statistischer Auswertung des Operationsverlaufes betrieben werden. Die Grundsätze minimalen Wissens und größtmöglicher Aufgabentrennung sind außerdem zu beachten.

Die Vorhaltung und Verarbeitung der Daten soll innerhalb eines geschützten Systems ohne Fremdzugriffe und Weitergabe betrieben werden. Die einzigen Ausnahmen sind die Weitergabe der Abrechnungdaten eines Patienten an die zuständige Krankenkasse und die Übermittlung der Diagnose bei einer Überweisung zu einer anderen Behandlungsstelle. Beides erfolgt über eine bestehende, angemessen gesicherte Netzwerkverbindung.

Der Gesamtaufenthalt eines Patienten soll folgende Schritte durchlaufen:

1. Aufnahme durch die Verwaltung
2. Diagnose und Behandlungsauftrag durch einen Internisten
3. Operation durch einen Chirurgen
4. Betreuende Therapie
5. Entlassung
6. Abrechnung mit der zuständigen Krankenkasse

7.3.2 Umsetzung in das Datenschutzmodell

Zunächst sind die Zwecke der Datenspeicherung mit ihren zugeordneten Aufgaben festzulegen:

Zweck	Behandlung	Verwaltung	Forschung
Aufgaben	Diagnose	Aufnahme	Statistik
	Operation	Entlassung	
	Therapie	Abrechnung	
	Überweisung	Datenweitergabe	

Tabelle 7.5: Klinikbeispiel: Zwecke und Aufgaben

Zur Speicherung der Daten sind Objektklassen mit zugeordneten Zwecken zu definieren:

Objektklasse	Zwecke	Inhalte
Aufnahmedaten	Verwaltung	Grunddaten des Patienten
Abrechnungsdaten	Verwaltung	Abrechnung der erbrachten Leistungen
Befund	Behandlung	Diagnosedaten
Behandlungsauftrag	Behandlung	Anweisungen an Operateure und Therapeuten
OP-Daten	Behandlung	Verlauf der Operation
Leistungsdaten	Verwaltung, Behandlung	Erbrachte Leistungen
Statistiken	Forschung	Auswertungen der Operationsverläufe

Tabelle 7.6: Klinikbeispiel: Objektklassen

Als nächstes werden Benutzer definiert und für ihre Aufgaben autorisiert:

Benutzer	Autorisierte Aufgaben
Internist	Diagnose, Therapie, Überweisung
Chirurg	Operation, Überweisung
Fortsetzung auf der nächsten Seite...	

... Fortsetzung Klinikbeispiel: Benutzer und ihre autorisierten Aufgaben	
Benutzer	Autorisierte Aufgaben
Therapeut	Therapie
Verwalter	Aufnahme, Entlassung
Buchhalter	Abrechnung, Datenweitergabe
Forscher	Statistik

Tabelle 7.7: Klinikbeispiel: Benutzer und ihre autorisierten Aufgaben

Zur Verarbeitung der Daten werden folgende Transformationsprozeduren definiert:

TP	Einsatz
pm_create	Erzeugen von Dateien einer Klasse
Anfüge-Editor	Erfassen eines Textes und Anhängen an eine Datei
Editor	Ändern einer Textdatei
Anzeigeprogramm	Bildschirmanzeige einer Textdatei
Löschprogramm	Löschen einer Datei
Weitergabeprogramm	Verschlüsselte Weitergabe von Dateiinhalten über Interprozeßkommunikation
Statistikprogramm	Lesen einer Datei, Erstellen einer Statistik und Schreiben in eine andere Datei

Tabelle 7.8: Klinikbeispiel: Transformationsprozeduren

Der nächste Schritt ist die Bestimmung der autorisierten Transformationsprozeduren aller Aufgaben:

Aufgabe	Autorisierte TP
Diagnose	pm_create, Anfüge-Editor, Editor, Anzeigeprogramm
Operation	pm_create, Anfüge-Editor, Editor, Anzeigeprogramm
Fortsetzung auf der nächsten Seite...	

... Fortsetzung Klinikbeispiel: Autorisierte TP der Aufgaben	
Aufgabe	Autorisierte TP
Therapie	Anfüge-Editor, Anzeigeprogramm
Überweisung	Weitergabeprogramm
Aufnahme	pm_create, Editor
Entlassung	Anfüge-Editor
Abrechnung	Editor, Anzeigeprogramm
Datenweitergabe	Weitergabeprogramm
Statistik	pm_create, Editor, Statistikprogramm

Tabelle 7.9: Klinikbeispiel: Autorisierte TP der Aufgaben

Schließlich ist als letztes die Menge der notwendigen Zugriffe zu bestimmen. Die möglichen Zugriffsarten sind Lesen, Schreiben, Löschen, Erzeugen und Anfügen.[3]

Aufgabe	Objektklasse	TP	Zugriffe
Diagnose	Befund	pm_create	Erzeugen
		Editor	Lesen, Schreiben, Anfügen
		Anzeigeprogramm	Lesen
	Leistungsdaten	Anfüge-Editor	Anfügen
	Behandlungsauftrag	pm_create	Erzeugen
		Editor	Lesen, Schreiben, Anfügen
Operation	Behandlungsauftrag	Anzeigeprogramm	Lesen
	OP-Daten	pm_create	Erzeugen
		Editor	Lesen, Schreiben, Anfügen
	Leistungsdaten	Anfüge-Editor	Anfügen
Therapie	Behandlungsauftrag	Anzeigeprogramm	Lesen
	Leistungsdaten	Anfüge-Editor	Anfügen
Überweisung	Befund	Weitergabeprogramm	Lesen
Fortsetzung auf der nächsten Seite...			

[3]Siehe Abschnitt 2.7.1.

... Fortsetzung Klinikbeispiel: Notwendige Zugriffe			
Aufgabe	Objektklasse	TP	Zugriffe
	Behandlungsauftrag	Weitergabeprogramm	Lesen
	Interprozeßkommunikation	Weitergabeprogramm	Erzeugen, Schreiben, Anfügen
Aufnahme	Aufnahmedaten	pm_create	Erzeugen
		Editor	Lesen, Schreiben, Anfügen
	Leistungsdaten	pm_create	Erzeugen
		Anfüge-Editor	Anfügen
Entlassung	Aufnahmedaten	Anfüge-Editor	Anfügen
	Leistungsdaten	Anfüge-Editor	Anfügen
Abrechnung	Leistungsdaten	Anzeigeprogramm	Lesen
	Abrechnungsdaten	pm_create	Erzeugen
		Editor	Lesen, Schreiben, Anfügen
Datenweitergabe	Abrechnungsdaten	Weitergabeprogramm	Lesen
	Interprozeßkommunikation	Weitergabeprogramm	Erzeugen, Schreiben, Anfügen
Statistik	Statistikdaten	pm_create	Erzeugen
		Editor	Lesen, Schreiben, Anfügen
		Löschprogramm	Löschen
		Statistikprogramm	Schreiben, Anfügen
	Befund	Statistikprogramm	Lesen
	Behandlungsauftrag	Statistikprogramm	Lesen
	OP-Daten	Statistikprogramm	Lesen

Tabelle 7.10: Klinikbeispiel: Notwendige Zugriffe

Für das PM-Modul sind in der aktuellen Fassung alle genannten Bezeichnungen als Zahlenwert zu kodieren und über Tickets mit rsbac_pm einzugeben.

7.3.3 Andere Modelle

Da das PM-Modell nur personenbezogene Daten und Administrations-Systemaufrufe vor Unbefugten schützt, bleiben Dateien und Verzeichnisse von der diskreten Rechteverwaltung abhängig. Es liegt nahe, diese durch mindestens ein anderes Modell vor Zugriffen zu schützen. Aufgrund der höheren Sicherheit sollte aber zumindest die Authentisierungsdatei `/etc/shadow` als personenbezogen mit eigener Objektklasse definiert und das Öffnen über notwendige Zugriffe beschränkt werden.

Für dieses Beispiel bietet sich die Verwendung der funktionalen Kontrolle an, um den Zugriff auf alle anderen sicherheitsrelevanten Dateien und Verzeichnisse über die Objektkategorien Sicherheits- und Systemobjekt einzuschränken. Für die sicherheitsrelevanten Objekte, die, wie die Identifizierungsdatei `/etc/passwd`, nur nicht unbefugt verändert werden sollen, kann das Sicherheits-Informations-Modifikations-Modell (SIM) mit dem Attribut Datentyp eingesetzt werden.

Das mandatorische Modell ist dann zusätzlich sinnvoll, wenn auch vertrauliche, nicht-personenbezogene Daten abzulegen sind, z. B. Geschäftsdaten.

7.3.4 Ablauf aus Systemsicht

Der Durchgang eines Patienten durch die Klinik wird durch folgende Schritte abgebildet:

1. Der Verwalter nimmt den Patienten in die Klinik auf. Dabei erzeugt er mit pm_create eine Datei für die Leistungsdaten und mit dem Editor eine Datei für die Aufnahmedaten, die er dort auch gleich vermerkt. Die Aufnahme wird anschließend mit dem Anfügeeditor als Leistung eingetragen.

2. Der Internist erzeugt eine Diagnosedatei und verwendet den Editor, um die Diagnose zu schreiben und bei Bedarf anzupassen. Anschließend erzeugt er eine Datei mit dem Behandlungsauftrag für diesen Patienten, den er per Editor einträgt und bei Bedarf ebenfalls anpaßt. Zum Schluß wird die erbrachte Leistung an die Leistungsdaten angefügt.

   Bei Bedarf kann der Patient außerdem an einen anderen Arzt oder eine andere Klinik überwiesen werden. Dazu können bisherige Diagnose und Behandlungsauftrag verschlüsselt per Netzwerk übermittelt werden.

3. Der Chirurg liest den Behandlungsauftrag mit dem Anzeigeprogramm und operiert den Patienten. Anschließend erzeugt und editiert er die OP-Daten, um den Verlauf der Operation zu dokumentieren. Auch der Chirurg fügt seine Leistung den Leistungsdaten hinzu.

   Wie der Internist kann der Chirurg den Patienten bei Bedarf an einen anderen Arzt oder eine andere Klinik überweisen und Diagnose- und Behandlungsauftragsdaten dorthin übertragen.

4. Der Therapeut liest ebenfalls den Behandlungsauftrag, therapiert und pflegt den Patienten und fügt seine erbrachte Leistung an die Leistungsdaten an.

5. Nach Ende der Behandlung wird der Patient durch den Verwalter entlassen, der dazu Aufnahme- und Leistungsdaten mit dem Anfügeeditor abschließt.

6. Der Buchhalter liest die Leistungdaten und erzeugt und editiert die Abrechnungsdaten, die er anschließend mit dem Weitergabeprogramm an die Krankenkasse übermittelt.

7. Befund, Behandlungsauftrag und OP-Daten können vom Forscher mit dem Statistikprogramm ausgewertet und Statistikdaten erzeugt, geändert sowie gelöscht werden. Die Verwendung weiterer Daten des Patienten erfordert dessen Einwilligung zum Zweck Forschung.

Kapitel 8

Bewertung des Systems

8.1 Sicherheit

Wie in Unterkapitel 3.6 für das Standard-Linuxsystem kann auch hier nur eine informelle Darstellung und Bewertung der Sicherheitseigenschaften des entstandenen Systems nach den Funktionalitäts-Oberbegriffen der ITSEC vorgenommen werden. Die Vertrauenswürdigkeit kann ebenfalls nur grob eingeschätzt werden.

8.1.1 Funktionalität

Identifizierung und Authentisierung

Identifizierung und Authentisierung erfolgen weiterhin durch Benutzernamen und Paßworte. Die zulässigen Benutzer sind in der Datei `/etc/passwd`, die Paßworte in verschlüsselter Form in `/etc/shadow` abgelegt. Da viele Programme auf die erste Datei zugreifen, muß diese für alle Benutzer lesbar sein, die zweite wird dagegen nur von Prozessen im Besitz des Systemverwalters gelesen.

Die Identifizierungs- und die Authentisierungsdatei können durch Kennzeichnung als Sicherheitsinformation über das SIM-Modul zusätzlich gesichert werden, nach der Implementierung des Clark-Wilson-Modells außerdem als "Constrained Data Item" mit ausschließlichem Zugriff durch festgelegte Benutzer über Transformationsprozeduren. Das Datenschutzmodul ermöglicht zusätzlich die Unterscheidung nach der Art des Zugriffes, bietet aber keine Integritätskontrolle durch IVP.

Für die Authentisierungsdatei kann außerdem durch das Sicherheitslevel "streng geheim" das unbefugte Lesen der kodierten Paßwörter verhindert werden, was aber die Klassifizierung des Benutzers root auf der selben Ebene erfordert und neue Lücken erzeugen kann.

Solange Identifizierung und Authentisierung nicht durch den Kern erfolgen, bleibt eine unangenehme Sicherheitslücke bestehen.

Zugriffskontrolle

Die Erweiterung der Zugriffskontrolle ist das Hauptziel der zusätzlichen Sicherheitsmechanismen und hinreichend beschrieben.

Durch den Verzicht auf die Kontrolle der eigentlichen Lese- und Schreiboperationen wirken sich Rechteänderungen erst beim erneuten Öffnen aus, wodurch sich möglicherweise kleinere Lücken ergeben können.

Beweissicherung

Die Beweissicherung erfolgt jetzt zusätzlich bei jeder Anfrage an die Entscheidungskomponente über den Kern-Protokoll-Mechanismus. Ausgegeben werden Datum, Uhrzeit, Anforderung, Prozeß-ID, Programmname, Benutzer (mit Pseudonym, falls definiert), Zugriffsziel, Attribut mit Wert und die Gesamtentscheidung. Die Protokollierung ist durch den Sicherheitsbeauftragten für jeden Anfragetyp getrennt auf einen der Werte "aus", "bei Ablehnung" und "an" einstellbar.

Protokollauswertung

Es existiert bisher kein Auswertungsprogramm für die zusätzliche Beweissicherung. Die vielen Parameter und das einheitliche Format erleichtern aber eine gezielte Extraktion von Werten.

Wiederaufbereitung

Die Gewinnung fremder Daten durch Wiederverwendung von Ressourcen ist bereits im Standard-Linux nur bei direktem Zugriff auf Geräte möglich. Ein gezieltes Überschreiben der Datenbereiche beim Löschen von Dateien ist bisher nicht vorgesehen.

Unverfälschtheit

Die Integrität der überwachten Daten wird durch das Clark-Wilson-Modul geschützt. Die Wiederherstellung eines älteren, gültigen Datenzustandes muß aber noch durch die Transformationsprozeduren geleistet werden, da sie nicht in den Systemkern integriert ist.

Das PM-Modul umfaßt ebenfalls weite Teile des Clark-Wilson-Moduls, wie Aufgabentrennung und ausschließliche Ausführung von zertifizierten Transformationsprozeduren auf dafür freigegebenen Daten. Es kann deshalb auch anstelle des Clark-Wilson-Moduls eingesetzt werden, es sollten dann aber die administrativen Aufgaben des Clark-Wilson-Modells zusätzlich durchgeführt werden.

Zuverlässigkeit der Dienstleistung

Die Zuverlässigkeit der Dienstleistung ist durch die RSBAC-Erweiterungen nicht erhöht worden. Die zusätzliche Komplexität des Systemkerns kann die Zuverlässigkeit sogar vermindern.

Übertragungssicherung

Durch die zusätzlichen Mechanismen kann eine Übertragung nur eingeschränkt, aber nicht gesichert werden. Die Übertragungssicherung liegt auch nicht im Zielbereich dieser Arbeit.

Schutz von personenbezogenen Benutzerdaten

Das implementierte Datenschutzmodell ist direkt auf die Gewährleistung der Privatheit personenbezogener Daten zugeschnitten. Durch geeignete Administration kann die Auswertung der Beweissicherung, und damit die Privatheit der Benutzer, wirksam gesichert werden.

Anonyme und pseudonyme Zugänge und weiterhin nur durch Administration möglich, Beobachtbarkeit und Verknüpfbarkeit sogar eher erhöht worden. Die Verwendung von Pseudonymen bietet aber einen gewissen Schutz vor dem Mißbrauch der Protokolldaten. Trotzdem sind die in [Sobirey+97] beschriebenen Verfahren zur Gewinnung von Benutzerprofilen und indirekten Identifizierung weiterhin möglich.

Schutz von personenbezogenen Daten anderer Personen

Der Schutz dieser Daten wird durch das Datenschutzmodul sichergestellt.

Gesamtbeurteilung

Die bisher erhalten gebliebene Arbeitsweise und Struktur des Unix-Systems birgt noch einige prinzipbedingte Schwächen, die durch reine Umsetzung des "Generalized Framework for Access Control" nicht behebbar sind. Doch auch diese funktionalen Schwächen könnten mit vertretbarem Aufwand beseitigt werden.

8.1.2 Vertrauenswürdigkeit

Die Wirksamkeit der Maßnahmen ist über die zugrundeliegenden, bekannten Sicherheitsmodelle abgesichert.

Die Korrektheit der Implementation der zusätzlichen Funktionalität kann im Rahmen dieser Arbeit nur durch einfache Tests nahegelegt werden.

Ein informeller Nachweis wäre für die Datenhaltung und die Entscheidungskomponente wegen ihrer Eigenständigkeit und Kapselung möglich. Für die Durchsetzung kann ich dieses nicht abschätzen, da viele Systemaufrufe stark mit anderen Kernfunktionen zusammenwirken und die Nicht-Umgehbarkeit der Erweiterungen daher schwierig nachzuweisen wäre.

Da im Wesentlichen der vorhandene Linuxkern erweitert wurde, ist ein zukünftiger formaler Beweis aufgrund der Komplexität grundsätzlich nicht möglich. Außerdem ist die verwendete Programmiersprache C für Beweise problematisch.

Die Vertrauenswürdigkeit der Korrektheit des erweiterten Linux-Kerns halte ich für die schwächste Stelle des Systems, die nur durch umfangreiche Tests abzusichern wäre. Derartige Tests könnten beispielsweise in der Linux-üblichen Weise durch Verteilung über das Internet an alle interessierten Administratoren geschehen, die jeweils in einer eigenen Testumgebung Teilbereiche untersuchen und zur Koordinierung alle Fehlfunktionen an mich zurückmelden. Die Entwicklung einer zentralen Testumgebung, die alle Aspekte der verwendeten Sicherheitsmodelle abdeckt, halte ich für sehr aufwendig.

8.2 Performanz

Die Performanzeinbußen des Systems sind stark von den aktiven Entscheidungsmodulen, der Protokollierung und der verfügbaren Hauptspeicher- und Prozessorleistung abhängig. Eine deutliche Lastreduzierung ergibt sich durch den Verzicht auf die Kontrolle der eigentlichen Lese- und Schreiboperationen.

Subjektiv ist eine Verlangsamung bei geringem Protokollaufkommen kaum festzustellen. Ein hohes Aufkommen, insbesondere bei Überprüfung und Protokollierung des sys_syslog-Systemaufrufes, über den das Protokoll ausgelesen wird, beeinträchtigt das Systemverhalten aber deutlich.

Für eine genaue Ermittlung des Performanzverlustes könnte bei praxisnahen Protokolleinstellungen ein Anwendungsbenchmark verwendet werden, wie es zum Vergleich von Computersystemleistungen üblich ist. Leider steht mir kein derartiges Produkt zur Verfügung.

8.3 Erfahrungen im Umgang

Trotz der sehr einfach gehaltenen Bedienung über Kommandozeilen führt der Umgang mit dem System bei vertretbarem Aufwand schnell zu nachvollziehbaren Sicherheitseffekten. Wegen des komplexen Verfahrens beim Systemstart ist das Bereithalten eines Wartungskerns für die ersten Versuche aber dringend zu empfehlen, da hier leicht Administrationsfehler das System unbenutzbar machen können.

In der derzeitigen Kernversion 2.0.30 gibt es leider eine kaum durchschaubare Wechselwirkung in den von der RSBAC-Datenhaltung stark beanspruchten Synchronisationsmechanismen, die insbesondere auf leistungsschwächeren PC Systemabstürze hervorrufen können.

8.4 Ausblick

Ich werde auch nach Ende dieser Arbeit das Projekt RSBAC weiterführen. Für die erste Zeit habe ich auch schon einige Vorstellungen, was ich verwirklichen möchte:

- Identifizierung und Authentisierung durch das RSBAC-System im Kern durch Umleitung über die Bibliotheksfunktionen in libc
- Feinentwurf und Implementierung des Clark-Wilson-Moduls
- Explizites Überschreiben alter Daten bei Truncate und Delete durch Anpassung der Systemaufrufe
- Koordinierte Verbreitung des Systems zum Testen und für neue Anregungen
- Übertragung auf andere Linux-Plattformen mit Hilfe interessierter Administratoren
- Möglicherweise Übertragung auf andere Betriebssysteme
- Benutzeroberfläche zur Grundadministration der bisherigen Module
- Ausführliche englische Dokumentation
- Verbesserte Backupmöglichkeit für die Datenhaltung

Die Übertragung des RSBAC-Systems sollte auf jedes Betriebssystem möglich sein, das Ressourcenzugriffe ausschließlich über Aufrufe an seinen Kern gestattet. Während die Entscheidungskomponente keine wesentlichen und die Datenhaltungskomponente nur Änderungen beim Dateizugriff benötigten, wäre die Durchsetzung wieder durch Anpassung aller sicherheitsrelevanten Systemaufrufe zu implementieren.

Kapitel 9

Zusammenfassung

Das Hauptziel dieser Arbeit war die Umsetzung des “Generalized Framework for Access Control” (GFAC) mit Hilfe eines Ansatzes von L. J. La Padula in ein konkretes Unix-System. Dieses Rahmenwerk verteilt die Funktionalitätsbereiche von Zugriffskontrollsystemen auf voneinander unabhängige Komponenten, um dadurch die Flexibilität zu erhöhen und die Fehleranfälligkeit zu erniedrigen.

Ausdruck der erreichbaren Flexibilität ist die Möglichkeit, mehrere Sicherheitsmodelle als sogenannte Regelsätze gleichzeitig, erweiterbar und konfigurierbar zu integrieren. Möglich ist dieses durch eine Trennung der durchsetzenden, modellunabhängigen von den zugriffsentscheidenden, modellabhängigen Funktionen. Die gesamten Entscheidungsregeln jeweils eines Sicherheitsmodells bilden dabei einen unabhängigen Regelsatz.

Als Grundlage aller Sicherheitsbewertungen dienen die Grundbereiche Vertraulichkeit, Integrität, Verfügbarkeit und Privatheit. Während der ältere, US-amerikanische Kriterienkatalog TCSEC (“Orange Book”) nur den Grundbereich Vertraulichkeit umfaßt, enthält der in der europäischen Union und in dieser Arbeit verwendete Kriterienkatalog ITSEC acht Oberbegriffe aus den Grundbereichen Vertraulichkeit, Integrität und Verfügbarkeit, um die Funktionalität von Sicherheitsmechanismen zu beurteilen. Außerdem beinhalten die ITSEC Kriterien zur Vertrauenswürdigkeit bezüglich der Wirksamkeit eines Funktionalitätsprofils und der Korrektheit seiner Implementation.

Zur Abdeckung des Grundbereiches Privatheit habe ich zwei weitere Oberbegriffe hinzugefügt, Schutz der personenbezogenen Daten von Benutzern und Schutz der personenbezogenen Daten von anderen Betroffenen. Der erste dieser beiden Oberbegriffe findet sich sinngemäß auch in dem in Arbeit befindlichen, international vereinheitlichenden Kriterienkatalog Common Criteria.

Von den fünf ausgewählten Sicherheitsmodellen sind zwei, die funktionale Kontrolle und die Modifikation von Sicherheitsinformationen, nur einfache, rollenbasierte Beispielmodelle. Die klassische mandatorische Zugriffskontrolle nach Bell-La Padula konzentriert sich, konform zu den TCSEC, auf den Grundbereich Vertraulichkeit, während das Clark-Wilson-Modell hauptsächlich die Integrität von Daten zum Ziel hat.

Das fünfte, nicht von La Padula vorgeschlagene Sicherheitsmodell ist das Datenschutzmodell von Simone Fischer-Hübner, dessen Umsetzung und Implementation einen weiteren Schwerpunkt dieser Arbeit bilden. Das Modell wurde gezielt zum Schutz personenbezogener Daten, also für den Grundbereich Privatheit, entwickelt, deckt aber Vertraulichkeit und Integrität als Teilziele der Privatheit mit ab.

Da eine ausführliche Protokollierung aller Benutzeraktivitäten im Sinne aller Sicherheitsmodelle, gleichzeitig aber problematisch für den Schutz der personenbezogenen Benutzerdaten ist, habe ich mich für die Verwendung von Pseudonymen entschieden. Trotzdem kann eine indirekte Zuordnung über Benutzerprofile prinzipiell weiterhin vorgenommen werden.

Die Umsetzung in das gewählte Betriebssystem Linux folgte nicht nur den genannten allgemeinen, sondern auch noch weiteren, zum Teil systemabhängigen Zielen. Dazu gehören die selbständige, unabhängige Sicherung jedes einzelnen, in ein Entscheidungsmodul zu kapselnden Regelsatzes, die eigenkontrollierte An- und Abschaltbarkeit solcher Module, die Unabhängigkeit vom verwendeten Dateisystem und eine bei höchstmöglicher Sicherheit größtmögliche Flexibilität, Erweiterbarkeit, Stabilität und Effizienz, um die Verfügbarkeit zu maximieren und den späteren Änderungsaufwand zu minimieren.

Die im GFAC definierten Zugriffskontroll-, Kontext- und Autorisierungsinformationen wurden in einer eigenen Komponente, der Datenhaltung, zusammengefaßt. Zugriffskontrollinformationen sind an Subjekte, also Benutzer und Prozesse, sowie Objekte, also Dateien, Verzeichnisse und Interprozeßkommunikationskanäle, gebundene Attribute. Den Kontext bilden alle anderen, überwiegend modellabhängigen Informationen, die von den Regeln benötigt werden, wie definierte Aufgaben oder Benutzergruppen. Zugriffe auf die Datenhaltung erfolgen ausschließlich über definierte Zugriffsfunktionen.

Zugriffe auf Massenspeicher oder andere Systemkomponenten aus der Prozeßebene sind in einem Unix-System nur über den Systemkern mit Hilfe von Systemaufrufen möglich. Die Durchsetzung einer Zugriffskontrolle erfordert deshalb eine Erweiterung aller relevanten Systemaufrufe um eine entsprechende Entscheidung, die hier über den Aufruf einer Funktion aus der Entscheidungskomponente erfolgt. Anschließend kann, abhängig von deren Ergebnis, die eigentliche Funktionalität des Systemaufrufes durchgeführt oder eine Fehlermeldung an den Prozeß zurückgegeben werden. War der Systemaufruf erfolgreich, wird die Entscheidungskomponente davon benachrichtigt, damit alle Sicherheitsmodule die von ihnen verwalteten Attribute an den neuen Systemzustand anpassen können.

Parameter der Entscheidungs- und der Benachrichtigungsfunktion sind die Anforderung, die in abstrahierter Form die gewünschte Funktionalität beschreibt, der Bezeichner des aufrufenden Prozesses und gegebenenfalls das Ziel des Zugriffes, ein Subjekt oder Objekt. Auf dieser Basis fordern die Entscheidungsmodule die benötigten Attribute von der Datenhaltung an und bilden eine Entscheidung. Die zentrale Verteilerfunktion verknüpft die Einzelentscheidungen schließlich zu einer Gesamtentscheidung, die der Durchsetzung zurückgegeben wird.

Aus der Erweiterung eines bestehenden Systems mit gegebenen Sicherheitsmodellen entstehen verschiedene praktische Probleme, da z.B. Programme sich nicht modellkonform verhalten oder Objekte sich innerhalb und außerhalb des Modellanwendungsbereiches befinden können.

Das notwendige, aber in den vorhandenen Programmen nicht vorgesehene Setzen eines aktuellen Sicherheitslevels für das Bell-LaPadula-Modell erforderte z.B. eine entsprechende Automatisierung. Da der einem Prozeß zugewiesene Hauptspeicherbereich, anders als in dem Referenzsystem Multics, keiner Zugriffskontrolle unterliegt, war außerdem die Einhaltung des *-property bei Lese- und Schreibzugriffen mit Hilfe von Schranken-Attributen sicherzustellen, um den möglichen Umweg über eine Zwischenspeicherung zu kompensieren.

Insgesamt habe ich 15 Systemaufrufe für modellübergreifende und modellabhängige Zustandsänderungen hinzugefügt, die natürlich ebenfalls der Zugriffskontrolle unterliegen. Aufgerufen werden sie mit Hilfe von Administrationswerkzeugen, die per Kommandozeile zu bedienen sind.

Die Kodierung erfolgte in der Programmiersprache des Linuxkerns, C. Sämtliche von mir vorgenommenen Erweiterungen des Kernes sind über die üblichen Konfigurationsprogramme an die eigenen Bedürfnisse hinsichtlich Modulauswahl, Schaltbarkeit, Netzwerkunterstützung etc. anpaßbar und durch Neuübersetzung und Installation zu aktivieren. Sind alle Erweiterungen abgeschaltet, entsteht ein unveränderter Linuxkern der Version 2.0.30.

Das Clark-Wilson-Integritätsmodell ist im Rahmen dieses Projektes noch nicht implementiert worden.

Eine informelle Sicherheitsbeurteilung nach den erweiterten Oberbegriffen der ITSEC zeigt bei Einsatz aller Module eine merkliche Verbesserung in den Bereichen Identifizierung, Authentisierung und Protokollauswertung sowie starke Verbesserungen in den Bereichen Zugriffskontrolle, Beweissicherung, Unverfälschtheit und Schutz der personenbezogenen Daten von Benutzern und anderen Betroffenen.

Aufgrund der erhalten gebliebenen Struktur des Unix-Systems zeigen sich aber trotzdem noch leichte Schwächen in den Bereichen Identifizierung und Authentisierung, Wiederaufbereitung, Zuverlässigkeit der Dienstleistung, Übertragungssicherung und Schutz von personenbezogenen Benutzerdaten. So sind z.B. der Schutz der Authentisierungsdaten und die Übertragungssicherung weiterhin administrationsabhängig, Benutzerprofile mit indirekter Identifizierung möglich und gelöschte Daten nicht automatisch auch physikalisch gelöscht. In einem nächsten Schritt der Sicherheitsoptimierung wären auch diese Punkte zu verbessern.

Die Hauptschwäche der vorgenommenen Implementation ist allerdings die wegen der Größe und Struktur des monolithischen Linux-Kernes mangelnde Nachweisbarkeit ihrer Korrektheit, die nur durch umfangreiche Tests abgemildert werden könnte.

Der Performanzverlust ist bei angemessenem Protokollaufwand subjektiv kaum feststellbar, eine Messung war mangels eines aussagekräftigen Verfahrens leider nicht möglich.

Insgesamt sind die gesteckten Ziele voll erreicht worden und ein benutzbares Linux-System mit hoher Sicherheitsfunktionalität entstanden. Trotzdem sind im Laufe der Arbeit konzeptionelle Schwächen an den Sicherheitsmodellen und am verwendeten System deutlich geworden, die vor allem die Vertrauenswürdigkeit beeinträchtigen. In der Fortführung des Projektes werde ich gezielt die gefundenen Schwächen abmildern.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Abrams+90] Abrams, M.D., Eggers, K.W., La Padula, L. J., Olson, I.M., A Generalized Framework for Access Control: An Informal Description, Proceedings of the 13th National Computer Security Conference, Oktober 1990

[Beck+95] Beck, M., Böhme, H., Dziadzka, M., Kunitz, U., Magnus, R., Verworner, D., Linux-Kernel-Programmierung, Algorithmen und Strukturen der Version 1.2, Addison-Wesley, Bonn, 1995

[Bell+76] Bell, D.E., La Padula, L. J., Secure Computer System: Unified Exposition and Multics Interpretation, Mitre Report Nr. ESD-TR-75-306, The MITRE Corporation, Bedford, MA, USA, 1976

[BDSG91] Einwag, A., Bundesbeauftragter für den Datenschutz (Hrsg.), Bundesdatenschutzgesetz - Text und Erläuterung, Bonn, Dezember 1991

[CC96] Bundesamt für Sicherheit in der Informationstechnik, Common Criteria for Information Technology Security Evaluation, Version 1.0, Januar 1996

[Clark+87] Clark, D.D., Wilson, D.R., A Comparison of Commercial and Military Computer Security Policies, CH2416-6/87/0000/0184$01.00, IEEE, 1987

[EU95] Direktive 95/46/EC des Europäischen Parlamentes und des EU-Ministerrates, Schutz von Individuen bezüglich der Verarbeitung und freien Weitergabe ihrer personenbezogenen Daten, 1995

[FiHue94] Fischer-Hübner, S., Towards a Privacy-Friendly Design and Use of IT-Security Mechanisms, Proceedings of the 17th National Computer Security Conference, Baltimore MD, Oktober 1994

[FiHue95] Fischer-Hübner, S., Considering Privacy as a Security Aspect: A Formal Privacy Model, Dasy Papers 5/95, Institute of Computer and System Sciences, Copenhagen Business School

[FiHue97] Fischer-Hübner, S., A Formal Task-based Privacy Model and its Implementation: An updated Report, Second Nordic Workshop on Secure Computer Systems (NORDSEC'97), Helsinki, 1997

[Flink+88] Flink, C.W.II, Weiss, J.D., System V/MLS Labeling and Mandatory Policy Alternatives, AT&T Technical Journal, Mai/Juni 1988

[Hosmer92-1] Hosmer, H., The Multipolicy Paradigm, Proceedings of the 15th National Computer Security Conference, Oktober 1992

[Hosmer92-2] Hosmer, H., Metapolicies II, Proceedings of the 15th National Computer Security Conference, Oktober 1992

[ITSEC91] Kriterien für die Bewertung der Sicherheit von Systemen der Informationstechnik (ITSEC), Vorläufige Form der harmonisierten Kriterien, Version 1.2; Amt für amtliche Veröffentlichungen der Europäischen Gemeinschaften, Luxemburg, 1991

[LaPadula95] La Padula, L.J., Rule Set Modeling of a Trusted Computer System, Essay, in: Information Security: An Integrated Collection of Essays, Hrsg.: Abrams, M.D., Jajodia, S., Podell, H.J., IEEE Computer Society Press, 1995

[Pfitzmann+93] Pfitzmann, A., Rannenberg, K., Staatliche Initiativen und Dokumente zur IT-Sicherheit, Eine kritische Würdigung, Computer und Recht, Ausg. 3/1993

[Sobirey+97] Sobirey, M., Fischer-Hübner, S., Rannenberg, K., Pseudonymous Audit for Privacy Enhanced Intrusion Detection, Proceedings of the IFIP TC11 13th international conference on Information Security (SEC '97), Verlag Chapman & Hall, Copenhagen, Denmark, 1997

[SuSE97] S.u.S.E. GmbH, S.u.S.E. Linux 5.0, Unix für PCs, Quellcode und Anwendungen, Fürth, 1997

[TCSEC85] Trusted Computer Security Evaluation Criteria ("Orange Book"), Department of Defense, Nr. DOD 5200.28-STD, USA, 1985

[Todino+96] Todino, G., Strang, J., Peek, J., UNIX - ein praktischer Einstieg, O'Reilly International Thomson Verlag, Bonn, 1996

Anhang A

Erklärung

Hiermit erkläre ich, daß ich diese Arbeit eigenhändig und selbständig erstellt und dabei nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Amon Ott

Anhang B

Quelltexte

Dieser Anhang enthält die von mir im Rahmen dieser Arbeit erstellten oder veränderten Quelltexte. Wie im Kapitel 7 erwähnt, können sämtliche Quelltexte auch über Internet unter http://agn-www.informatik.uni-hamburg.de/people/1ott/rsbac bezogen werden.

Aus Platzgründen befindet sich der Inhalt dieses Anhangs in einem Extra-Band.