Grundseminar

Sicherheitsmechanismen in Linux

Seminararbeit

von

Barth, Michael
aus Schwäbisch Gmünd

26206

HTW Aalen

Hochschule für Technik und Wirtschaft

UNIVERSITY OF APPLIED SCIENCES

Prof. Hellmann
23.06.2008

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich meine Seminararbeit mit dem Thema

Sicherheitsmechanismen in Linux

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Aalen, 27.12.2008

Kurzfassung

In diesem Grundseminar werden die gängigsten Sicherheitsmodelle wie Mandatory Access Control, Access Control Lists, Linux Capabilities sowie andere behandelt. Desweiteren werden Kernelerweiterungen basierend auf dem LSM-Framework (welches zuerst vorgestellt wird) und basierend auf Kernelpatches für das Betriebssystem Linux vorgestellt, darunter SELinux, AppArmor und Ruleset Based Access Control. Der Begriff des „härtens“ wird ebenfalls erläutert.

Dieses Arbeit bezieht sich auf keine spezifische Linux Distribution und es wird nicht garantiert das jedes vorgestellte Sicherheitsmodell auf jeder Distribution vorhanden ist bzw. es dafür Module gibt.

Das Grundseminar befasst sich ausschließlich mit Sicherheit auf Applikationsebene exklusive dem Thema der Netzwerksicherheit. Gefahren und Bedrohungen der Betriebssystemsicherheit von Linux werden nur rudimentär vorgestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACL – Access Control List
DAC – Discretionary Access Control
DTE – Domain and Type Enforcement
FLASK – Flux Advanced Security Kernel
LSM – Linux Security Modules
MAC – Mandatory Access Control
RBAC – Role Based Access Control
RSBAC – Ruleset Based Access Control
SUID – Super User ID
TE – Type Enforcement

1 Einleitung

1.1 Motivation

Das Thema Sicherheit wird, Aufgrund der ständig zunehmenden Popularität, auch für Linux heutzutage immer wichtiger. Mit zunehmender Verbreitung wächst die Attraktivität für Angreifer die Schwachstellen des Systems zu erkunden und auszunutzen, Viren dafür zu schreiben oder einfach nur seinen Schabernack damit zu treiben, weil man es eben kann.

1.2 Problemstellung und -abgrenzung

Diese Arbeit konzentriert sich auf die Sicherheitsaspekte von einem Gebiet unter Linux: Sicherheit auf Applikationsebene. Dies umfasst Themen wie Zugriffsrechte, gehärtete Linux Distribution, SELinux, Access Control Lists und Linux Capabilities und Kernelerweiterungen.

Die Arbeit bezieht sich nicht auf eine Distribution im speziellen, sondern auf Linux im Allgemeinen.

1.3 Ziel der Arbeit

Ziel ist es, dem Leser ein Verständnis für die unter Linux eingesetzten Sicherheitsmechanismen zu vermitteln. Ihn darüber aufzuklären welche Gefahren bestehen, wie Linux davor schützt und welche zusätzlichen Schutzmaßnahmen man ergreifen kann.

2 Grundlagen

2.1 Was bedeutet Sicherheit?

An dieser Stelle ist es wichtig zwischen dem alltäglichen Gebrauch des Begriffs Sicherheit und dem Gebrauch von Sicherheit in dieser Arbeit zu unterscheiden. Laut Wikipedia bezeichnet Sicherheit „einen Zustand, der frei von unvertretbaren Risiken der Beeinträchtigung ist oder als gefahrenfrei angesehen wird.“ [1]

Diese Definition, auf die Sicherheit eines Betriebssystems angewendet, würde die Behandlung von sehr vielen Gefahren und Beeinträchtigungen beinhalten. Beispielsweise solche, die durch Programmfehler entstehen.

Dieser weite Begriff von Sicherheit wird hier nicht verwendet. Wenn in dieser Arbeit von Sicherheit die Rede ist bezieht sich dieser auf einen Zustand, der frei ist von Beeinträchtigungen durch Personen oder Programmen, die versuchen unerlaubten Zugriff auf ein System zu erlangen und dessen Betrieb zu stören.

2.2 Übersicht der Bedrohungen

Nachfolgend eine Übersicht, sowie kurze Beschreibung, der bekanntesten Bedrohungen für die Sicherheit eines Systems.

2.2.1 Buffer Overflows

Buffer Overflows gehören mit zu den häufigsten Sicherheitsproblemen bei Programmen. Dabei handelt es sich um „einen Programmfehler, bei dem ein Programmierer für bestimmte Daten weniger Speicher zur Verfügung stellt, als tatsächlich benötigt wird. Kommt die unerwartete Menge Daten an, so ist der zur Verfügung gestellte Puffer nicht ausreichend groß: Er läuft über.

Dieser Zustand kann für einen Einbruch ausgenutzt werden.“ [2] Man schreibt schädlichen Angriffscode in die übergelaufenen Speicherbereiche und hofft, dass diese irgendwann zur Ausführung kommen. Ist dies der Fall, so hat man sich eine Hintertür ins System geschaffen und kann diese für weitere Angriffe nutzen.

2.2.2 Viren, Würmer und Trojaner

„Ein Virus ist ein kleines Programm beziehungsweise eine Programmroutine, die selbstständig andere Programme »infiziert«. Ein Virus wird erst aktiv, wenn das entsprechende infizierte Programm gestartet wird.

Ein Wurm verbreitet sich dagegen selbständig in Netzwerken. [...]

Ein Trojaner ist dagegen ein angeblich harmloses Programm, das aber »bösen Code« zur Öffnung einer Hintertür für Angreifer oder andere Spitzfindigkeiten enthält“. [3]

2.2.3 Rootkits

Rootkits dienen dem Zweck, nach einem Einbruch in ein System die Spuren des Hackers wieder zu verwischen um zu verhindern, dass das Einbruchsopfer etwas vom Einbruch mitbekommt und entsprechende Gegenschritte einleiten kann. Heise unterscheidet hierbei zwei Arten von Rootkits:

„(...) Rootkits [lassen sich] grob betrachtet in datei- und kernelbasierte Arten unterteilen. Vereinfacht dargestellt tauschen dateibasierte Rootkits auf der Festplatte Systembefehle aus und ersetzen diese durch eigene, trojanisierte Versionen. Diese sind dann so modifiziert, dass zum Beispiel (...) [ein] neues ps-Kommando beziehungsweise der Windows-Task-Manager (...) bestimmte Prozesse und der Befehl netstat ausgewählte Netzwerkverbindungen nicht mehr [anzeigt]. Ähnliches gilt für ls, dir oder den Explorer, in deren Ausgabe verdächtige Dateien nicht mehr auftauchen. Das Verstecken von Prozessen und Dateien wird normalerweise so realisiert, dass der modifizierte Systembefehl keine Objekte anzeigt, die eine vorher definierte Zeichenkette enthalten.

Eine modernere und zugleich effektivere Unterart der dateibasierten Rootkits sind die Library Rootkits. Hier wird nicht jeder einzelne Systembefehl verändert, sondern direkt die Bibliothek, die eine Funktion zur Verfügung stellt und mit der die Systemprogramme dynamisch verlinkt sind. Das recht verbreitete t0rn Rootkit für Linux/Unix ist ein bekanntes Beispiel hierfür. Es tauscht unter anderem die Systembibliothek libproc.so aus, die Funktionen bereitstellt, über die Programme Prozessinformationen über das /proc-Dateisystem auslesen können. So zeigt selbst ein "sauberes" ps bestimmte Prozesse nicht mehr an, da es die Informationen erst gar nicht bekommt. Schaut man direkt im /proc-Dateisystem nach, findet man natürlich alle Angaben zu den versteckten Prozessen.“ [4]

3 Applikationssicherheit

3.1 Zugriffsrechte

Zugriffsrechte regeln im Allgemeinen die Rechte zwischen Subjekten (Benutzer, Prozesse, etc.) und Objekten (Dateien, Dienste, Prozesse, etc.), wenn diese in Interaktion miteinander treten. Sie legen fest, wer welche Zugriffe auf welche Objekte durchführen darf.

3.1.1 Discretionary Access Control

Linux beruht auf einem Zugriffsrechtemodell genannt Discretionary Access Control (kurz *DAC*), welches Benutzer grob in drei Arten einteilt: Besitzer, Gruppe und Welt (alle Anderen). Der Besitzer ist der Benutzer der die Datei angelegt hat, die Gruppe ist ein Zusammenschluss mehrerer Benutzer und Welt sind einfach alle anderen Benutzer. Die Zugriffsrechte sind fest verankert im Dateisystem und verdeutlichen Linux' Ausrichtung zum Mehrbenutzer-Betriebssystem.

Im folgenden soll kurz gezeigt werden, wie ein Dateisystemeintrag aufgebaut ist:

```
drwxr-xr-x   2 mbarth   users  4096 May  1 12:25 Verzeichnis
-rwxr--r--   1 mbarth   users   147 May  1 12:26 Datei
-rwxrwxr--   1 mbarth   users   643 May  1 12:27 Script
```

Das erste Bit `d` zeigt an ob es sich bei dem Eintrag um ein Verzeichnis handelt. Es folgen 9 Bits welche die Zugriffsrechte der drei Benutzerunterteilungen definieren. Die Zugriffsrechte pro Unterteilung umfassen 3 Bits: `rwx`. `r` steht für read (Leserechte), `w` für write (Schreibrechte) und `x` für execute (Ausführrechte). Der erste Dreierblock steht demnach für Besitzer, gefolgt von Gruppe und schließlich Welt.

Die Zahl nach den Zugriffsrechten ist ein Linkzähler, gefolgt vom Benutzernamen des Besitzers, seiner Gruppe, der Dateigröße, dem Datum der letzten Änderung und letztendlich dem Namen der Datei. Auf diese Weise ermittelt Linux, wer welche Rechte hat im Dateisystem.

Probleme mit DAC

Will man beliebige Rechtekombinationen bei einem großen System mit vielen hunderten oder tausenden Benutzern zuweisen, reichen die Möglichkeiten von DAC nicht aus.

Desweiteren ist das Ändern des eigenen Passworts problematisch. In Unix werden die Benutzerdaten inklusive Passwörtern in einer zentralen Datei gespeichert. Wenn man dem

Benutzer nun Schreibrechte für diese Datei gibt, könnte er ebenso auch die Passwörter anderer Benutzer ändern. Dies wäre ein schwerwiegendes Sicherheitsproblem.

Zur Lösung dieses Problems wurde ein neues Bit, das *Super User ID* Bit (kurz SUID) eingeführt welches das Ausführbit `x` bei den Besitzer-Bits ersetzt. Die Ausführrechte werden in diesem Fall über das Ausführbit der Gruppe oder Welt gewährt. Eine Datei welche mit diesem Bit ausgeführt wird erhält zur Laufzeit des Programms die Benutzer ID (User ID, UID) bzw. die Gruppen ID (Group ID, GID) des Besitzers. Im Falle der Passwortdatei wird man also zum Superuser (`root`). Diese Lösung an sich ist aber auch nicht optimal. Enthält das Programm zum Ändern des Passworts einen Fehler, lassen sich damit alle Benutzerkonten übernehmen und das System wäre kompromittiert.

3.1.2 Access Control Lists

Access Control Lists (ACLs) stellen eine Erweiterung des DAC-Konzepts dar, wie es in 3.1 vorgestellt wurde. Sie ermöglichen eine feinere Granularität bei der Steuerung der Zugriffsrechte für Dateisystemeinträge. Wo man bei DAC nur Rechte für *den* Besitzer, *eine* Gruppe und die Welt vergeben kann, erlauben ACLs hingegen eine Rechtesteuerung für *individuelle* Benutzer oder Gruppen für jede einzelne Datei.

Als Beispiel könnte eine Datei für die Gruppe `developer`, zu welcher auch der Besitzer der Datei gehört, die Rechte `rwx` besitzen. Nun gibt es noch eine Gruppe, `testteam`, das die Datei nur lesen und ausführen, jedoch nicht überschreiben darf. Mithilfe von ACLs kann man die Rechte für die individuelle Gruppe `testteam` auf `r-x` setzen, ohne die Rechte für Welt, welche `---` sind, ändern zu müssen. Dies lässt sich sogar auf der Ebene einzelner Benutzer realisieren: So könnte `Bob`, der Leiter des Testteams, als einzigster die Rechte `rwx` für die Datei besitzen.

ACLs sind standardmäßig nicht im Linux Kernel integriert, noch sind es die dafür notwendigen Änderungen damit bestimmte System Utilities die ACLs korrekt unterstützen. Hierfür muss der Kernel gepatched und die nötigen userland (user space) ACL Utilities müssen installiert werden. [5]

3.1.3 Mandatory Access Control

Mandatory Access Control (MAC) führt so genannte *Labels* ein um die Sicherheit im System zu erhöhen. Ein Label wird dem zu sichernden Objekt zugewiesen und repräsentiert wie sensibel die Daten des Objekts sind, also welcher Sicherheitsstufe es unterliegt. Jeder Benutzer besitzt eine Freigabe für gewisse Sicherheitsstufen und Einschränkungen für Andere. Die Sicherheitsbestimmungen, die über solche Labels festgelegt werden, können nicht durch den Benutzer selbst geändert werden, selbst nicht einmal durch den `root` Benutzer (das *Mandatory* im Namen steht für diese Tatsache). Eine Policy (Menge von Regeln) legt fest, wer Labels setzen und verändern kann, sowie welche Voreinstellung gilt. [6]

Dies hat den Vorteil das, selbst wenn ein Angreifer den `root` Benutzer übernommen hat, die Sicherheitsregeln der MAC immer noch aktiv sind und eine Hürde für den Angreifer darstellen.

Abstriche muss man hierfür allerdings bei der Performance und dem Bedienkomfort hinnehmen, denn die Objekte müssen manuell mit Labels versehen werden. Dies kann sich als sehr aufwändig erweisen, je nach Größe des Systems. Weist man keine Labels zu, erhöht MAC logischerweise auch nicht die Sicherheit des Systems.

Type Enforcement (TE)

Das *Type Enforcement model* ist nötig um ein Mandatory Access Control System zu realisieren. Type Enforcement bezieht sich auf Zugriffskontrolle und wird benutzt um der MAC Priorität über DAC zu gewähren. Eine Zugangsberechtigung wird hierbei dem Subjekt (z. B. einem Prozess der gerade ausgeführt wird) verliehen für den Zugriff auf ein Objekt (Dateien, Dienste, Prozesse, etc.) basierend auf einem Sicherheitskontext. Der Sicherheitskontext einer Domäne wird definiert durch eine Domain-Policy (Domänen-Sicherheitsrichtlinie). Bei SELinux, welches später noch behandelt wird (siehe 3.4), nutzt man ein erweitertes Attribut für den *Systemkontext* (siehe 3.4 SELinux Begriffe auf Seite 15 für eine Erklärung des Begriffs Systemkontext).

Prinzipiell wird bei TE folgendes gemacht: TE überprüft die Regeln vom Sicherheitskontext der Quelle des Subjekts und vergleicht diese mit den Regeln des Sicherheitskontextes des Zielobjekts. Darauf basierend wird dann über eine Freigabe des Zugriffes entschieden. Im Falle von MAC heißt das, es wird das Label (der Sicherheitskontext) der Quelle mit dem Label des Zielobjekts verglichen und basierend auf diesem Resultat ein Urteil gefällt ob der Zugriff rechtens ist. [7]

Dies bedeutet jedoch nicht, dass MAC Zugriffe die von DAC überschreibt, im Gegenteil: Fehlen dem Benutzer bei DAC die Rechte, so kommt MAC gar nicht erst zum Zuge, da diese die DAC-Rechte nur weiter einschränken, aber nicht überschreiben können.

3.2 Linux Capabilities

Mit Version 2.1 des Linux Kernels begann die Arbeit an den so genannten *capabilities*. Ziel war es, die Abhängigkeit vom `root` Account für bestimmte Aktionen zu lösen. Die Abhängigkeit von einem einzelnen Account welcher die Rechte für entsprechende Aktionen besitzt hat sich als gefährlich herausgestellt, da sich eben bei diesen Aktionen Schwachstellen einschleichen können. [5]

Capabilities versuchen diese Abhängigkeit aufzuheben, indem die Rechte des `root` Accounts in eine Reihe von Privilegien Sets aufgeteilt werden. Dies verhindert das man beim Erhalt der Rechte einen Port freizugeben damit einhergehend auch das Recht erhält, Dateien zu verändern und zu löschen. Dieser Ansatz basiert auf dem *least-privilege model*. [8]

3.2.1 Das Least-Privilege Model

Das least-privilege model verfolgt den Ansatz, dass jede Aufgabe nur die Rechte erhält die Sie direkt benötigt um erfüllt werden zu können. Es soll verzichtet werden auf jedes Recht das nicht für die Aufgabe benötigt wird, was letztendlich, so ist es gedacht, mehr Sicherheit gewährt. [8]

3.2.2 Implementierung

Es gibt eine Reihe von Capabilities welche für Linux implementiert wurden. Unter `/usr/include/linux/capability` kann man diese begutachten. Nachfolgend werden ein paar wenige Capabilities beispielhaft aufgeführt und kurz erklärt:

Capability Name	Bedeutung
CAP_CHOWN	Erlaubt das Ändern des Dateibesitzers
CAP_DAC_OVERRIDE	Überschreibe alle DAC Zugriffseinschränkungen
CAP_KILL	Erlaubt das Senden von Signalen an Prozesse die anderen Benutzern gehören
CAP_NET_RAW	Erlaube die Benutzung von raw sockets

Table 1: Übersicht über einige Capabilities

Capabilities sind vor allem für Programmierer von großem Nutzen, um die Sicherheit ihrer Programme zu erhöhen, speziell System Utilities geben erworbene `root`-Rechte wieder auf.[5]

3.3 Härten

Unter „härten“ versteht man das Sichern eines Systems durch entfernen nicht benötigter Komponenten in Form von optionalen oder nicht benötigten Betriebssystemfunktionen und Diensten sowie optionalen oder nicht benötigten Programmen, dem Entfernen aller nicht benötigten Benutzerkonten und dem schließen nicht benötigter Ports. Alle Prozesse und Dienste sollten mit den niedrigsten Privilegien die möglich sind ausgeführt werden, so dass die Aufgabe noch erfüllt werden kann (siehe 3.2 Das Least-Privilege Model). Dies gilt insbesondere auch für Benutzer-Rechte.

Ein System sollte gehärtet werden, bevor es mit einem Netzwerk verbunden wird. Updates oder neueste Treiber kann man über einen Wechseldatenträger von einem Zweitrechner mit Internetverbindung beziehen, sollte dies nötig sein.

IBM empfiehlt folgende Vorgehensweise um ein System zu härten [9]:

- Securing the boot process
- Securing services and daemons
- Securing local filesystems
- Enforcing quotas and limits
- Enabling Mandatory Access Control
- Updating and adding security patches

Häufig verwendet man auch Mandatory Access Control um ein System zu härten (siehe 3.1 Mandatory Access Control).

3.4 Kernelerweiterungen

Kernelerweiterungen ergänzen den normalen Linux Kernel um weitere Funktionen. Im Folgenden wird speziell auf Erweiterungen der Sicherheitsaspekte eingegangen.

3.4.1 LSM Framework

Das Linux Security Modules Framework (LSM Framework) ist ein allgemeines Kernel Framework welches mit der Kernel Version 2.6 offiziell eingeführt wurde. Es ermöglicht das Laden von Modulen in den Kernel zur Laufzeit. Dies hat den Vorteil, dass es den Kernel schlank, als auch universell hält (letzteres war ein besonderes anliegen von Linus Torvalds, welcher sich nicht auf eine bestimmte Kernelerweiterung zum Thema Sicherheit festlegen wollte). [10]

Das LSM Framework bietet verschiedene `hook`-Funktionen (Einhängepunkte) an, welche aufgerufen werden sobald über eine Zugriffsberechtigung entschieden werden muss. Über diese `hook`-Funktionen kann man nun eigenen Sicherheitscode einbinden und ausführen, welcher den Zugriff letztendlich regelt. Nachfolgend wird auf die Vor- und Nachteile von LSM eingegangen.

Vorteile:

- ✔ LSM ist eine einheitliche, stabile Schnittstelle.
- ✔ Module können zur Laufzeit ausgetauscht werden.

Nachteile:

- ✗ Eine Framework beinhaltet immer das Problem eingeschränkter Funktionalität: Wurde bei der Schnittstelle nicht an eine `hook`-Funktion für ein bestimmtes Feature gedacht, hat man als Entwickler eines LSM-Moduls keine Möglichkeit diesen Aspekt abzufangen und zu behandeln.
- ✗ Die Schnittstelle selbst stellt ein potentielles Angriffsziel/eine potentielle Sicherheitslücke dar.
- ✗ Es kann immer nur ein einziges Sicherheitsmodul geladen und aktiv sein.

3.4.2 SELinux

SELinux ist eine ursprünglich von der NSA entwickelte Kernelerweiterung, welche die FLASK Architektur (*Flux Advanced Security Kernel*) umsetzt die flexible Sicherheitsstrategien bietet. Eine Besonderheit von SELinux ist die Trennung der Regeln (*Policies*) von der Durchsetzung. Zuerst war SELinux ein Patch für Kernel 2.2, ab 2001 wurde es auf das LSM Framework portiert und mit Kernel 2.6 als integraler Bestandteil in den Kernel schließlich aufgenommen. Unterstützt werden folgende Sicherheitsmodelle: Mandatory Access Control, Type Enforcement (beide siehe 3.1) und *Role Based Access Control*.

Role Based Access Control (RBAC)

Wie der Name schon vermuten lässt, werden beim RBAC die Rechte den Rollen zugeteilt. Eine Rolle repräsentiert hierbei nicht eine Person bzw. einen Benutzer und seine Gruppe, sondern eine Aufgabe. Dementsprechend besitzt eine Rolle auch nur alle benötigen Rechte für die jeweilige Aufgabe.

Ein Benutzer kann mehrere Rollen besitzen und diese auch wechseln. Bei einem Wechsel verliert er die Rechte seiner vorherigen Rolle und erhält die Rechte der neuen Rolle die er annimmt. Dies gewährt, dass ein Benutzer immer mit den gerade nötigen Rechten, aber nicht mehr als nötig, seinen Tätigkeiten nachgehen kann ohne eine Einschränkung zu erleiden oder ein Sicherheitsrisiko darzustellen durch den Besitz unnötiger Rechte. Desweiteren legen die Systemregeln fest, welche Rollen ein Benutzer annehmen darf.

Das Verfahren ist nicht auf Lese-, Schreib- und Ausführrechte auf Dateien beschränkt. [11]

Begriffe

Folgende Begriffe werden im Kontext zu SELinux verwendet [12]:

- **identity** – feste Beschreibung der Identität
- **domain** – Sicherheitsrahmen für Subjekte, z.B. Prozesse
- **type** – Sicherheitsrahmen für Objekte, z.B. Dateien
- **role** – Regel, die Umgebungen beschreibt, z.B.:

  ```
  role  user_r  types  user_passwd_t
  ```

- **policy** – Menge von Rollen
- **security context:** `user:role:type`

Voraussetzungen

Das Dateisystem muss ein spezielles Attribut unterstützen für den Systemkontext, welcher für die Mandatory Access Control benötigt wird.

Neben der Kernelerweiterung sind ebenso noch einige Benutzermodule und Bibliotheken wie `policycoreutil, selinux-policy-default, checkpolicy, libselinux1, selinux-doc` und `selinux-utils` nötig.

3.4.3 AppArmor

AppArmor ist eine weitere Kernelerweiterung um die Sicherheit von Linux zu erhöhen. Ursprünglich entwickelt unter dem Namen *SubDomain* 1998, wurde es 2004 von Novell übernommen und in SUSE integriert. 2005 wurde SubDomain dann in *AppArmor* umbenannt und 2006 wurde der Sourcecode unter die GPL gestellt und veröffentlicht. Ebenso wie SELinux implementiert AppArmor ein Mandatory Access Control System und benutzt die LSM-Schnittstelle. AppArmor erlaubt es dem Administrator jedem Programm ein *Profil* zuzuweisen, welches die Rechte und Einschränkungen eines Programms definiert.

Zusätzlich dazu besitzt AppArmor einen *Learning Mode*, welcher Verletzungen der Profile loggt, aber nicht verhindert. Diese Logs basieren auf dem typischen Verhalten des Programms und können schließlich in ein Profil umgewandelt werden. Dies ermöglicht zu Beginn ein angenehmeres Arbeiten, da zu eng konfigurierte Profile, aufgrund von Unwissenheit über die Rechte die eine Applikation braucht, am Anfang sehr störend und ärgerlich sein können. Um die Sicherheit des Systems nicht zu kompromittieren sollte man die Logs aber vorher überprüfen um auszuschließen, dass das Programm unerlaubte Zugriffe durchführte während der Learning Mode aktiv war.

SELinux steht häufig in der Kritik sehr kompliziert zu sein, was den normalen Benutzer abschrecke. Daher war eine der Entwurfsrichtlinien, AppArmor als Alternative zu SELinux zu entwickeln. AppArmor implementiert MAC auf Basis von *Dateipfaden* (im Vergleich hierzu: SELinux benutzt Labels die einzelnen Dateien zugewiesen werden). Die Entwickler und Unterstützer von AppArmor sind der Meinung, Rechte basierend auf Dateipfaden seien leichter zu verstehen für den gewöhnlichen Benutzer als Labels. Desweiteren behaupten die Entwickler und Unterstützer, AppArmor benötige weniger Modifikationen um mit einem bestehenden System zu funktionieren. [13]

„Novell unternahm wiederholt Versuche, AppArmor in den von Linus Torvalds gepflegten offiziellen Linux-Kernel zu integrieren, stieß jedoch auf Vorbehalte in der Entwicklergemeinde. Die Bedenken richten sich primär gegen das Verfahren, Dateien über ihren Namen zu identifizieren.“ [14]

3.5 Ruleset Based Access Control

Ruleset Based Access Control (RSBAC) begann als Diplomarbeit 1996. Seit 2000 steht RSBAC als stabile Version zur Verfügung und ist unabhängig von der Regierung oder großen Unternehmen. RSBAC ist nach wie vor ein *Kernelpatch* und benutzt nicht die LSM-Schnittstelle. RSBAC selbst ist ein Framework für Sicherheitsmodule und bietet zusätzlich ein System zum Loggen von Sicherheitszugriffen um mögliche Angriffe zu erkennen.

RSBAC ist sehr flexibel und unterstützt verschiedene Sicherheitsmodule wie: Mandatory Access Control (MAC), Access Control Lists (ACL), DAZ (ein Antivirus Scanner Interface) und RC (Role Capacity). [15]

3.5.1 Role Capacity (RC)

Bei RC besitzt jeder User eine Standardrolle, welche er von all seinen Prozessen erbt. Die Rolle definiert welche Zugriffe auf welche Objekte dem User gestattet sind. Rollen können gewechselt werden wenn der Besitzer des Prozesses gewechselt wird durch einen *system call*, jedoch nur in andere „kompatible Rollen“. Will man in eine nicht kompatible Rolle wechseln muss man dies über eine ausführbare Datei machen (mit `initial_role` oder `force_role`). [15]

3.5.2 Die Logik des RSBAC Frameworks

Das RSBAC Framework basiert auf der Logik des *Generalized Framework for Access Control* von Abrams und LaPadula. [15] Dieses geht von drei fundamentalen Konzepten aus die nötig sind für ein Access Control System:

- „Authority: An authorized agent must define security policy, identify relevant security information, and assign values to certain attributes of `controlled resources`."

- „Attributes: Attributes describe characteristics or properties of `subjects` and `objects`. The computer system will base its decisions about access control on the attributes of the subjects and objects it controls. Examples of attributes are:

  security classification
  type of objectdomain of process
  date and time of last modification
  owner identification"

- „Rules: A set of formalized expressions defines the relationships among attributes and other security information for access control decisions in the computer system, reflecting the security policies defined by authority." (alle Listenpunkte nach [16])

„The generalized framework explicitly recognizes two parts of access control — adjudication and enforcement. We use the term *access control decision facility* (ADF) to denote the agent that adjudicates access control requests, and the term *access control enforcement facility* (AEF) for the agent that enforces the ADF's decisions. In a trusted computer system, the AEF corresponds to the system functions of the *trusted computing base* (TCB) and the ADF corresponds to the *access control rules* that embody the system's security policy, (...)". [16]

Alle relevanten system calls werde erweitert um einen s*ecurity enforcement code*. Dieser Code ruft die zentrale Entscheidungskomponente (*access control decision facility*, ADF), welche wiederum die Anfrage an alle aktiven Entscheidungsmodule weiterleitet (welche verschiedene Sicherheitsmodelle implementieren). Diese Treffen dann eine kombinierte, endgültige Entscheidung welche durch die *system call extensions* durchgesetzt werden.

Entscheidungen basieren auf dem Typ des Zugriffs (*request type*), dem Zugriffsziel und auf den Attributen des Subjekts, welches Zugriffsrechte anfordert, und den Attributen des Zieles auf das zugegriffen werden soll. Zusätzliche unabhängige Attribute können von individuellen Modulen genutzt werden, wie z.B. das *privacy module* (PM). Alle Attribute werden in voll geschützten Verzeichnissen gespeichert, wovon es eines für jedes *mounted device* gibt. Daher benötigen Änderungen an den Attributen spezielle *system calls*.

Da alle Arten von Entscheidungsanfragen auf einheitlichen Entscheidungsanfragen basieren, können viele unterschiedliche Sicherheitsmodelle als Entscheidungsmodul implementiert werden. Abgesehen von den eingebauten Modellen erlaubt die *optional Module Registration* (REG) die Registrierung von zusätzlichen, individuellen Entscheidungsmodulen zur Laufzeit. [15]

Illustration 3.1 gibt einen Überblick über die Logik dieses Frameworks.

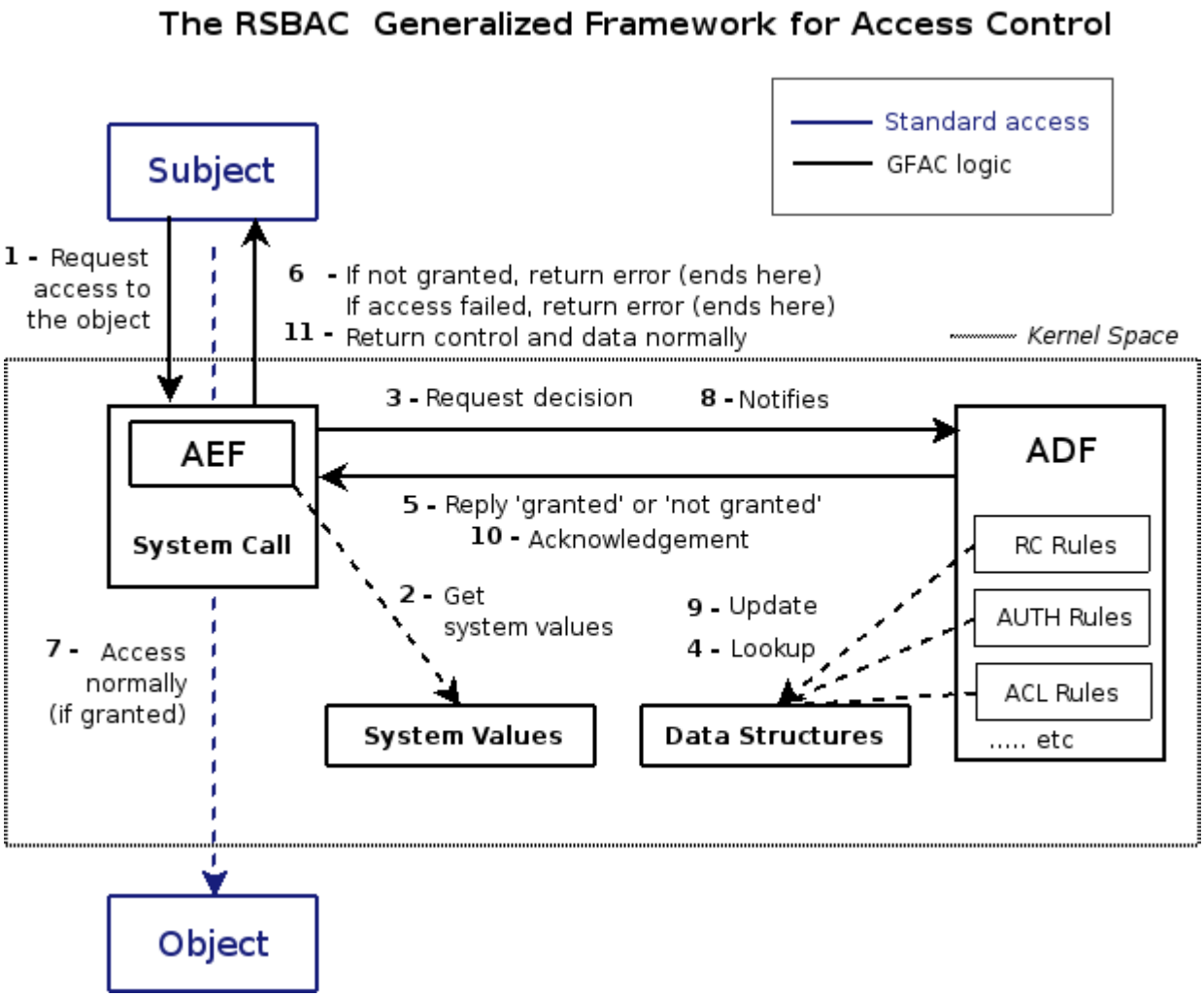

Illustration 3.1: Das RSBAC Generalized Framework for Access Control. Bild von [15]

4 Literaturverzeichnis

[1]: Wikipedia Security, http://de.wikipedia.org/wiki/Sicherheit

[2]: Basiswissen Buffer Overflow, http://tinyurl.com/6k96sf

[3]: Plötner, Johannes; Wendzel, Steffen, Praxisbuch Netzwerk-Sicherheit, 2007

[4]: Heimliche Hintertüren: Rootkits aufspüren und beseitigen, http://tinyurl.com/6chasb

[5]: Introduction to Linux Capabilities and ACL's, http://www.securityfocus.com/infocus/1400

[6]: Security Enhanced Linux: Einführung, Architektur, Anwendung, http://tinyurl.com/5frrkm

[7]: Wikipedia: Type Enforcement, http://en.wikipedia.org/wiki/Type_enforcement

[8]: A bid to resurrect Linux capabilities, http://lwn.net/Articles/199004/

[9]: Securing Linux, Part 3: Hardening the system, www.ibm.com/developerworks/linux/library/l-seclnx3

[10]: Linux Security Modules: General Security Hooks for Linux, http://tinyurl.com/6rdaj9

[11]: Wikipedia: Role Based Access Control, http://tinyurl.com/5e7xus

[12]: Mandatory Access Control durch SELinux, http://tinyurl.com/6lcjl5

[13]: Wikipedia: ArrArmor, http://en.wikipedia.org/wiki/AppArmor

[14]: Linux-Entwickler wechselt in Microsofts Windows Security Team, http://tinyurl.com/669zsh

[15]: What is RSBAC, http://www.rsbac.org/why

[16]: Rule-Set Modelingof a Trusted Computer System, http://www.acsa-admin.org/secshelf/book001/09.pdf

* Aufgrund einer maximalen Zeichenlänge von 50 Zeichen für **jedes** Feld der Literaturdatenbank von OpenOffice musste leider für einige URLs der Webservice *TinyURL.com* in Anspruch genommen worden, um die Quellen hier aufzuführen. Der Vollständigkeit halber führe ich die URLS nachfolgend nochmal von Hand auf.

[2] http://www.tecchannel.de/webtechnik/entwicklung/402308/

[4] http://www.heise.de/security/Heimliche-Hintertueren--/artikel/38057/1

[10] http://www.projectblackdog.com/3rdParty/kernel-doc-2.6.10.msw1/Documentation/DocBook/lsm/x31.html

[11] http://de.wikipedia.org/wiki/Role_Based_Access_Control

[12] http://www.luga.de/Angebote/Vortraege/MAC_SELinux/MAC_SELinux.pdf

[14] http://www.heise.de/newsticker/Linux-Entwickler-wechselt-in-Microsofts-Windows-Security-Team--/meldung/102115