Universität Leipzig
Fakultaet für Mathematik und Informatik
Institut für Informatik

Bachelorarbeit

Untersuchung von MAC-Implementationen

Abstract: Benutzerbestimmte Zugriffskontrolle ist an vielen Stellen schwer zu beschränken und zu administrieren. Der Ansatz der systembestimmten Zugriffskontrolle - Mandatory Access Control - gibt die Verantwortung an das System ab und gibt Benutzern deutlich weniger Rechte. Diese Arbeit vergleicht zwei Vertreter, welche Mandatory Access Control umsetzen, einerseits das Linux Security Module Framework und andererseits das FreeBSD MAC Framework, zudem werden die wichtigsten Policy Vertreter angegeben. Auf beiden Seiten finden sich ähnliche Ansätze wie die Umsetzung als Kernelmodul und vor allem generische Fähigkeiten, allerdings sind die implementierten Funktionalitäten unter FreeBSD im Detail oft besser durchdacht oder auch ausgereifter.

Leipzig, März 2010

vorgelegt von
Markus Nentwig
Studiengang Informatik

Betreuender Hochschullehrer:

Prof. Dr. Martin Bogdan
Fakultät für Mathematik und Informatik
Technische Informatik

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Arbeit unterstützt haben. Genannt seien dabei vor allem Herr Prof. Dr. Bogdan und mein Betreuer Dipl.-Inf. Jörn Hoffmann, die mir bei der Auswahl des Themas sowie bei der Bearbeitung immer zur Seite gestanden hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

2^{32} IP-Adressen sind nicht mehr ausreichend, um alle Computer der Welt zu adressieren - in den 80er Jahren undenkbar. Gelöst wird dieses Problem durch eine Vergrößerung des Adressraumes um den Faktor 2^{96}, auch IPv6 genannt. Nur ein Beispiel, dennoch spiegelt es die weltweite Tendenz einer vernetzten und durchaus komplexen Computerentwicklung wider, welche in der Vergangenheit zum Teil noch unzureichend bedacht wurde. Der vielschichtige Bereich der Computersicherheit muss sich dem genauso anpassen wie viele andere Abschnitte. Mit dem bisher größtenteils verwendeten Modell der *Discretionary Access Control* kann ein System vor bestimmten Bedrohungen nicht hinreichend geschützt werden, da alle Aktionen von Benutzern verantwortet werden müssen. Mit der Ergänzung durch die *Mandatory Access Control* fällt dieses Problem in weiten Teilen weg und gibt die Gewalt über die Zugriffskontrolle an eine Richtlinie ab, welche vom System umgesetzt und von einem Administrator zentral verwaltet wird.

Unter Linux wie auch unter FreeBSD gibt es zur Erweiterung um *Mandatory Access Control* jeweils ein großes Framework, welches die Grundlage für verschiedene Richtlinien zur Ausübung von Zugriffskontrolle bilden soll. Gewählt wurden die beiden Betriebssysteme auch, weil die Quelltexte derselbigen offen und frei für jedermann zur Verfügung stehen. Damit kann Sicherheit schon auf konzeptionell hohem Niveau gewährleistet werden, ohne das Risiko auf versteckte oder ungewollte Funktionalität. Die beiden Modelle Linux Security Module Framework und das FreeBSD MAC Framework werden in der folgenden Arbeit präsentiert, zudem werden für beide Implementationen die wichtigsten Richtlinien vorgestellt, welche die Zugriffskontrolle dann oftmals verschieden umsetzen. Untersucht wird vor allem, wie die beiden Frameworks ihre grundsätzlichen Konzepte realisieren, nebenher sind ebenso die Unterschiede beider Lösungen dargestellt. Es zeigt sich hierbei, dass beide Ansätze zwar ähnliche Ideen verwenden, diese aber im Detail von FreeBSD vielfach besser umgesetzt werden.

2 Grundlagen

Im folgenden Kapitel werden einige Begrifflichkeiten erklärt, welche im Hauptteil der Arbeit für das Verständnis vorausgesetzt werden. Zu Beginn wird im Abschnitt 2.1 der Terminus Computersicherheit in Bezug auf die Zugriffskontrolle definiert, im Anschluss daran sind im Abschnitt 2.2 historische sowie aktuelle Standards zur Computersicherheit aufgeführt. Danach sind verschiedene Zugriffskontrollmodelle detailliert unter 2.3 erläutert und abschließend noch der Begriff Audit im Absatz 2.4 erklärt.

2.1 Computersicherheit

Der Begriff *Computersicherheit* ist in der heutigen vernetzten Welt sehr weitreichend, da er im Endeffekt für viele verschiedene Bereiche gelten muss. Beispielsweise gilt es, Computer schon auf der physischen Ebene vor einem möglichen Zugriff zu schützen und nur autorisierten Personen Zugang zu Kühlanlagen oder zur Stromversorgung zu gewährleisten. Im weiteren Verlauf der Arbeit wird allerdings nicht weiter auf physischen Zugriffsschutz eingegangen, sondern speziell auf den Zugriffsschutz auf der Betriebssystemebene. Hier ist die Gewährleistung der Sicherheit weitaus schwieriger, was vor allem der Vielzahl der Nutzer an den Rechnern mit dazu unterschiedlichen Vertrauensstufen sowie der Anbindung an das Internet geschuldet ist. Auch moderne Betriebssysteme haben immer wieder Schwachstellen, über welche die internen Sicherheitsdienste übergangen oder irregeleitet werden bzw. Einstellungen ohne die erforderlichen Rechte geändert werden können. Diese gilt es schnell zu schließen - die bessere Lösung ist allerdings, diese schon von Beginn an mit einer sicheren Architektur vermeiden. Bei der Definition von Computersicherheit wir daraus folgend vor allem auf die Einhaltung der folgenden drei Punkte geachtet. [LRS06]

Confidentiality *Vertraulichkeit*, das heißt, Daten sollen nur den Personen zugänglich sein, welche dafür autorisiert sind.

Integrity *Integrität* in dem Sinne, dass Daten ihren Zustand zwischen zwei autorisierten Zugriffen nicht ändern.

Availability *Verfügbarkeit*, so dass autorisierte Benutzer jederzeit Zugriff auf die Daten haben.

Speziell für Zugriffskontrollen, auf welche in dieser Arbeit eingegangen wird, bilden folgende Begriffe eine wichtige Grundlage.

Identification Die *Identifikation* ist der erste Vorgang bei einem möglichen Zugriff. Der Zugreifende muss hierfür seinen Namen angeben.

Authentication Folgend wird bei der *Authentifizierung* ein Nachweis für die Identität gefordert, meist in Form eines Passwortes.

Authorization In diesem Schritt wird überprüft, welche *Berechtigungen* der Zugreifende besitzt.j

Accountability Die *Zurechenbarkeit* beschreibt die Forderung nach einer Kontrolle über durchgeführten Aufgaben oder Operationen.

2.2 Standards für Computersicherheit

Die Bewertung und Zertifizierung von IT-Systemen, Anwendungsprogrammen oder Hardware erfolgt aus verschiedenen Gründen heraus, der wichtigste ist aber, dass auf diese Weise einheitliche sowie verbindliche Sicherheitskriterien aufgeschlüsselt werden. Nur wenn die Vertrauenswürdigkeit der Systeme gewährleistet ist, können diese zum Einsatz kommen. Insgesamt orientieren sich die Standards von ihren Kriterien her mehr oder weniger direkt an den in Abschnitt 2.1 genannten Punkten. Eine Zertifizierung wird über ein Dokument bestätigt, welches angibt, gegen welche vor der Evaluierung festgelegten Bedrohungen das getestete Objekt wie geschützt ist. Zudem werden die angewandten Sicherheitsmechanismen auf Korrektheit der Funktionalität und das Vertrauenslevel überprüft. [Eck06]

Ein weiterer wichtiger Punkt für die Standardisierung von Zertifizierungen bei Computer- und IT-Systemen ist der immense Kostenaufwand, der alleine schon durch eine Zertifizierung nach einem Standard anfällt. Bei uneinheitlichen Sicherheitskriterien auf dem internationalen Markt müssten Programme dann unter Umständen mehrfach zertifiziert werden. Je nach Level der Zertifizierung müssen unterschiedliche Aspekte der Software validiert werden. Die höchsten Stufen der Zertifizierung können nur über komplett mathematisch formal nachgewiesene Korrektheit und der zugehörigen Implementation erreicht werden. Aufgrund der oben genannten Kostengründe wird dies in der Praxis nicht erreicht. Ausführlicher erklärt wird diese formale Verifizierung auch in [Kle08].

Die Entwicklung von Maßstäben für Computer- und IT-Sicherheit werden vor allem unter der Regie von staatlichen Institutionen geführt. Begonnen hat dies vor allem im militärischen Bereich im Jahr 1980 mit dem „Orange Book“ (vgl. 2.2.1. Diese Entwicklung erfolgte mit der Zielsetzung, in sensiblen technischen

Bereichen des Staates nur evaluierte Produkte zu verwenden. Die international wichtigsten Standards werde ich im folgenden vorstellen. [Ker95, LRS06]

2.2.1 „Orange / Red Book“

Erstmals 1980 wurden Sicherheitsrichtlinien für IT-Systeme aufgestellt. Diese wurden vom US-Institut National Computer Security Center entwickelt und unter dem Namen Trusted Computer System Evaluation Criteria - TCSEC veröffentlicht. Als Erweiterung kam 1985 die Trusted Network Interpretation - TNI als Bewertungskatalog für vernetzte IT-Systeme. Die Namensgebung erfolgte aufgrund der Umschlagfarbe des jeweiligen Werkes, bei TCSEC orange und bei der TNI rot. [Eck06, Cen83]

2.2.2 Deutsche und europäische Bestrebungen

Auch in Deutschland respektive Europa wurden vom „Orange Book“ ausgehend Kriterien für Zertifizierungen entwickelt. Kritikpunkte der Vorlage aus den USA wurden aufgegriffen und möglichst beseitigt. Daraus mündeten dann auf deutscher Seite IT-Kriterien, auch „Grünbuch“ genannt, und auf europäischer Ebene die Information Technology Security Evaluation Criteria - kurz ITSEC. Getrennt wurden hier erstmals funktionale Eigenschaften - welche den „Orange Book“ Regeln entsprechen - und Qualitätsaspekte der verwendeten Mechanismen. [Eck06]

2.2.3 Common Criteria

Die *Common Criteria* for Information Technology Security Evaluation (CC) geben einen internationalen Rahmen vor, der zur Bewertung sicherheitstechnischer Produkte herangezogen werden kann. Die CC hat sich aus europäischen und nordamerikanischen Bestrebungen zu einer international anerkannten Sicherheitsbewertung herausgebildet. Die Erstfassung 1.0 wurde im Januar 1996 veröffentlicht, mittlerweile ist Version 3.1 aktuell. [Eck06]

Eine Umsetzung dieser Kriterien erfolgte durch die Evaluation Assurance Level (EAL), welche auf einer Skala von eins bis sieben angeben, wie hoch die Vertrauenswürdigkeit des getesteten Computersystems ist. Zusätzlich werden verschiedene Schutzprofile angestrebt, welche ein Sicherheitskonzept für einen bestimmten Produktbereich, beispielsweise Smartcards, aufzeigen. Dadurch ist eine bestimmte Sicherheit und Funktionalität wie ein vollständigen Audit oder einen bestimmten Nutzerdatenschutz für das Produkt garantiert. Ziel der Common Criteria ist es, Zertifizierungen in mehreren Ländern zu ersparen. [CCE06]

2.3 Zugriffskontrolle

Allgemein ausgedrückt geht es bei der Zugriffskontrolle immer darum, auf welche Art und Weise der Zugriff auf schützenswerte Ressourcen kontrolliert wird. Dabei sind die unter Abschnitt 2.1 aufgezählten Kriterien die Maßgabe für die erreichte Sicherheit. Die drei bekanntesten Modelle zur Zugriffskontrolle werden im folgenden beschrieben.

2.3.1 Discretionary Access Control - DAC

Diskrete Zugriffskontrolle - auch benutzerbestimmbare Zugriffskontrolle - ermöglicht dem Besitzer, auf seinen Daten individuell Zugriffsrechte zu vergeben. Dies erfolgt meist über drei Grundtypen zur Steuerung der Rechte: Lesen, Schreiben und Ausführen. Zudem gibt es bei dieser Art der Kontrolle meist Gruppen, um anderen Nutzern leicht Rechte einzuräumen. Bei der DAC kann man auch relativ komplexe Fälle abbilden, allerdings ist dies eher umständlich. Als Erweiterung kann man andererseits Access Control Lists (ACLs) verwenden, um einen feinere Justierung der Rechte zu vergeben. Anwendung findet diese Methode in der Microsoft NT Produktlinie sowie bei den Linux / POSIX ACLs. [LRS06]

2.3.2 Mandatory Access Control - MAC

Die *systembestimmte* oder auch *obligatorische Zugriffskontrolle* ist in seiner gröbsten Definition einfach nur ein System, in welchem die Logik für die Zugriffskontrolle sowie die Erstellung der Regeln komplett von einem speziell dafür eingesetzten Administrator verwaltet und *allein* durch das Computersystem umgesetzt wird. Dies kann nun unterschiedliche Ausmaße haben. In vielen Fällen werden dafür alle Objekte - Dateien sowie Geräte - und Subjekte - also Nutzer - des Systems mit Labeln ausgestattet, die dann ein bestimmtes Vertrauenslevel widerspiegeln. Zudem sind oft sicherheitsrelevante Operationen nur möglich, wenn eine Überprüfung auf die nötigen Zugriffsrechte durch eine Kontrolle der angehafteten Labels erfolgreich war. Auf diese Art und Weise erfolgt unter Umständen auch allgemein ein Dateizugriff: Das Vertrauenslevel eines Subjektes wird dazu benutzt um dieses mit dem Vertrauenslevel des Objektes zu vergleichen und bei einer Autorisierung durch die MAC Policy kann dann der Zugriff erfolgen. Bestimmte Garantien über allgemeine Ziele der Computersicherheit wie Integrität oder Vertraulichkeit werden dann durch Anwendung spezieller Modelle erreicht. Mit dem Ansatz des Type Enforcement kann man beispielsweise die Integrität von Daten auf einem System sicherstellen, hierbei wird eine Entscheidung durch die verwendete MAC Poli-

cy im Zweifel immer höher gestellt als die der DAC Policy. Zudem besitzen alle Subjekte und Objekte eine Art Label, mit deren Hilfe bei jedem Objektzugriff unabhängig von der Identität des Nutzers geprüft wird, ob dem Nutzer der Zugriff gestattet ist. Als weitere Spezialisierung gibt es in etwa das Bell-LaPadula-Modell mit militärischem Ursprung, welches bei richtiger Umsetzung die Vertraulichkeit der Daten zusichert. Hier wird auf der Systemebene - also ohne jede Nutzerbeeinflußung - definiert, wie ein Informationsfluss kontrolliert ablaufen kann, selbst wenn ein Angreifer Zugriff zum System haben sollte. Die in diesem Zusammenhang oft genannte MLS (Multi Level Security) ist dabei nur die Umsetzung von Restriktionen nach dem Bell-LaPadula-Modell und nicht die ursprüngliche militärische Bedeutung eines kompletten Sicherheitsmodells. Eine potentielle Schwachstelle bei der systembestimmten Zugriffskontrolle ist in jedem Falle der verantwortliche Administrator, der die Rechte vergibt. Wenn dieser in einer komplexen Sicherheitsrichtlinie einen Fehler hat, kann der zusätzliche Schutz schnell nicht mehr gewährleistet werden. [HRJM07, DCN02, WWS07, PA08]

2.3.3 Role-based Access Control - RBAC

Bei diesem Modell wird eine *rollenbasierte Zugriffskontrolle* genutzt. Erst erfolgt eine Zuteilung von erlaubten Aktionen zu Rollen. Durch die Rolle(n), denen der Nutzer angehört, wird bestimmt, ob der Zugriff auf die angeforderten Daten erfolgen darf. Zu einer Person kann bei diesem Konzept immer auch mehr als nur eine Rolle zugewiesen werden, beispielsweise schließen sich Systemadministrator und Netzwerkadministrator nicht aus, sie decken nur andere Aufgabenbereiche ab. Benutzer können somit nicht darüber entscheiden, ob sie anderen Nutzern Zugriffsrechte geben oder nicht, dies bestimmt das System. [FK92] Dementsprechend wird bei RBAC das Prinzip des least privilege angewandt. Die manchmal geäußerte Aussage, RBAC wäre eine Form von MAC, ist nach der grundlegenden Definition von MAC falsch, denn MAC verwendet Labels zur Objekt- und Subjektidentifizierung sowie verschiedene Vertrauenslevel, wobei RBAC Rollen an Nutzer vergibt, welche damit die gewünschten Aufgaben erledigen können. [FKC03]

2.3.4 Zugriffskontrollmatrix

In einer Zugriffskontrollmatrix wird in einfacher Darstellung gezeigt, welcher Benutzer auf welche Daten wie zugreifen kann. Das folgende einfache Beispiel in der Tabelle 2.1 lässt erkennen, dass die fiktiven Benutzer „Foo“, „Bar“ und „Baz“ verschiedene Rechte auf den Objekten haben. Diese Form der Veranschaulichung ist allerdings oft ineffizient, da die Matrix bei vielen Subjekten

und Objekten nicht sehr dicht besetzt ist. In abgewandelter Form findet diese Matrix dennoch oft Anwendung. Die häufigste Form ist dabei die Zugriffskontrollliste, welche die Kontrolle aus der Objektsicht - also spaltenweise - darstellt. Zudem bilden sogenannte Capabilities Teile der Matrix aus der Sicht des Subjektes - also zeilenweise - ab. [Eck06]

		Objekt		
		Datei 1	Datei 2	...
Subjekt	Foo	owner,r,w,x	-	...
	Bar	r,w	owner,r,w,x	...
	Baz	-	-	...

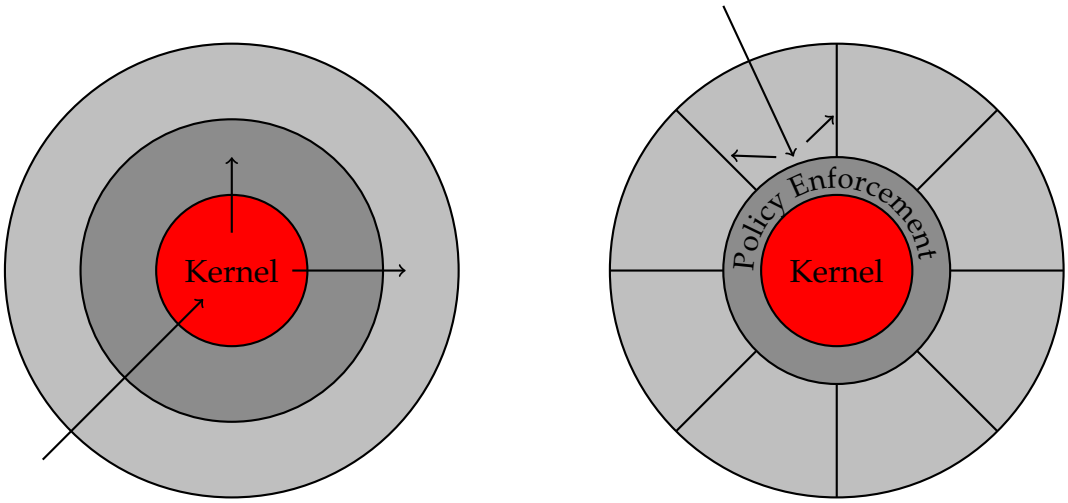
Tabelle 2.1: Einfache Zugriffskontrollmatrix, welche Rechte auf verschiedene Dateien anzeigt. Spalten- und Zeilenangaben können auch verwendet werden, um Zugriffskontrolllisten und Capabilities zu erklären.

2.3.5 Vergleich Zugriffskontrolle

Um eine bessere Übersicht über die Modelle des DAC und MAC zu erhalten, zeige ich zum einen einen schematischen Vergleich mit der Abbildung 2.1 und zum anderen stelle ich die beiden Modelle in der Tabelle 2.2 gegenüber.

2.4 Audit

Der Begriff *Audit* ist in dieser Arbeit vor allem von dem proaktiven Standpunkt aus gesehen. Bei einem Authentifizierungsversuch wird mittels Zugriffskontrollsystem (2.3) überprüft, ob der Benutzer für den Vorgang die nötigen Rechte besitzt. Danach muss er sich für jeden Programmzugriff erneut autorisieren, was durch Audits optimiert werden kann. Im Detail können mit einem Audit System zudem Sicherheitsereignisse und ausgeführte Befehle, aber auch Kernelprozesse kontrolliert und vor allem protokolliert werden. Dabei anfallende Daten, beispielsweise die Zugriffszeit (Timestamp), der zugreifende Benutzer und dessen Rechte sowie die zugegriffenen Daten können vom System protokolliert werden. Die gesammelten Informationen können dann ausgewertet werden, zum Beispiel in der Art und Weise, dass die mitgeschnittenen Entscheidungen bezüglich der Zugriffskontrolle an beteiligte Programme weitergeleitet werden können, wodurch unter Umständen Wartezeiten vermieden werden. [BTS09]



Wenn bei der *benutzerbestimmten Zugriffskontrolle (DAC)* über eine Schwachstelle Zugriff auf einen priviligierten Systembereich erlangt wird, ist das gesamte System kompromittiert.

Bei der *systembestimmten Zugriffskontrolle (MAC)* werden durch die Sicherheitspolicy Regeln für die Anwendungen aufgestellt, wodurch die Schwachstelle auf die Anwendung beschränkt wird.

Abbildung 2.1: Vergleich von DAC und MAC anhand einer schematischen Betriebssystemabbildung. Die Pfeile zeigen eine mögliche Ausbreitung einer Schwachstelle in den jeweiligen Zugriffskontrollmodellen. Als Ergebnis ist zu erkennen, dass für diese Art von Angriffen MAC einen besseren Schutz als die oft verwendete DAC bietet. [fli10]

Zugriffskontrolle	DAC	MAC
Wer legt Rechte fest?	Datenbesitzer	Administrator
Umsetzung komplexer Zugriffsregeln	schwieriger	einfacher
Nutzung bevorzugt	privat / geschäftlich	geschäftlich / militärisch
Anzahl Nutzer im System	wenige	viele
Zugriff durch Kontrolle von	Benutzer (Subjekt)	allgemeine Regeln zu Subjekten und Objekten
Verwendung von Labels für Objekte und Subjekte	nein	meist ja

Tabelle 2.2: Vergleich der Zugriffskontrollmodelle DAC und MAC nach ausgewählten Kriterien als allgemeine Übersicht. Der Vergleich fällt eher schwer, denn oft werden beide Modelle in Kombination verwendet. [Spe07]

Damit erfolgt eine Abgrenzung zu der reaktionären Lösung des Intrusion Detection Systems (IDS), welche sich vor allem mit der nachträglichen Auswertung von Logs beschäftigt. Dies wird hier nicht behandelt. Ein IDS reagiert beispielsweise beim Erkennen von bestimmten auffälligen Verhaltensmustern mit einer Verletzungsmeldung.

3 Stand der Technik

Im folgenden Kapitel werden nun je für Linux als auch FreeBSD die entwickelten MAC Frameworks vorgestellt. Seitens Linux wird das LSM Framework im Abschnitt 3.1 dargestellt, diesem folgend wird auf der FreeBSD Seite das TrustedBSD MAC Framework im Abschnitt 3.2 präsentiert. Die Ansätze ähneln sich dabei, allerdings finden sich im Detail durchaus verschiedene Konzepte, welche etwa bedingt durch die Betriebssystemphilosophie oder allein durch die verwendete Architektur entstanden sind.

3.1 Linux Security Module (LSM)

Das Linux Security Module stellt ein Sicherheitsframework dar, welches mit dem Gedanken entwickelt wurde, ein System zur systembestimmten Zugriffskontrolle in den Linux Kernel einzubauen, welches bisherige Kontrollmethoden von der Funktionalität her bei weitem übertrifft. Das Prinzip vom LSM ist dabei, eine Grundlage zu bilden, darauf aufbauend Kernelmodule mit Richtlinien zu laden, um damit den geforderten Sicherheitskontext zu leisten. Das LSM bietet demnach in seiner Grundausstattung keine zusätzliche Sicherheiten, diese werden erst durch die geladenen Sicherheitsmodule gewährleistet. [SVS06] Die Umsetzung des Linux Security Module Framework bildet dabei einen Kompromiss zwischen möglichst geringem Einfluss auf den Kernel und der Möglichkeit, verschiedenste Sicherheitsmodelle anzuwenden. Weitere Punkte, welche einen wichtigen Aspekt der Entwicklung darstellten, waren Effizienz im Einsatz, eine einfache konzeptuelle Umsetzung sowie eine allgemeingültige Schnittstelle, mit der man verschiedene Sicherheitsrichtlinien laden kann. [WCS+02] Eine vereinfachte Darstellung der Funktionsweise des LSM ist in der Abbildung 3.1 zu finden.
Ein mögliches Problem unter einem ungesicherten Linux stellt die Verwendung des SUID-Bit dar, welches für beliebige Programme gesetzt werden kann. Durch dieses Bit kann ein Benutzer auf Befehle zugreifen, die sonst nur Root zustehen. Dies ist für elementare Vorgänge - wie das Ändern von Dateirechten oder das Umstellen der Systemzeit - notwendig. Wenn in einem dieser freigegebenen Programme allerdings ein Fehler steckt, kann der Nutzer unter Umständen komplette Root-Privilegien erhalten [DCN02]. Umfassend mit dieser

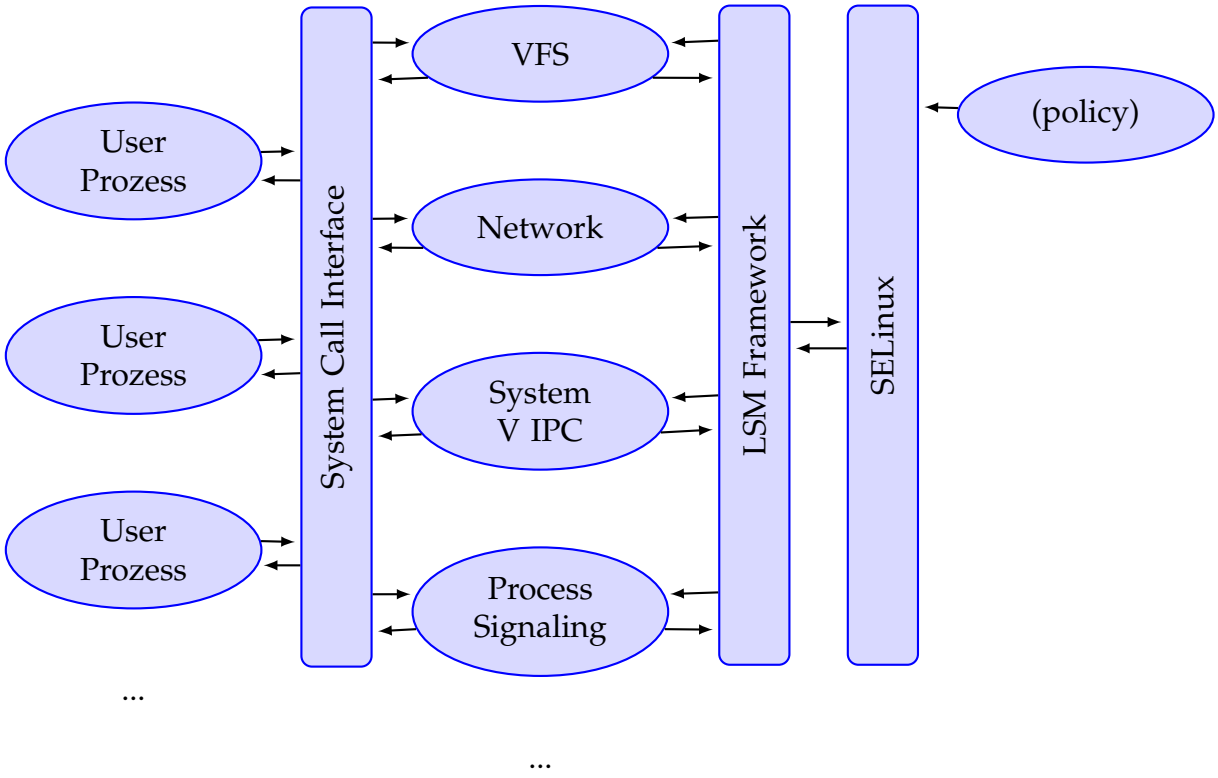


Abbildung 3.1: Veranschaulichung der Funktionsweise des LSM Frameworks. Jeder von einem Prozess abgegebenen System Call greift auf bestimmte Elemente im Kernel zu. Bevor dieser Zugriff gewährt wird, kann durch eine geladene Sicherheitspolicy geprüft werden, ob der Benutzer des entsprechenden Prozesses die nötigen Rechte besitzt. Zu einer Hauptsicherheitsrichtlinie (hier SELinux) können weitere Policies nur an der erstgeladenen angemeldet werden. (nach [VW03]

Thematik beschäftigt haben sich auch Wissenschaftler aus Berkeley, ihr Ansatz zur Lösung der bestehenden Probleme besteht darin, ein formales Modell zur Auswertung von User IDs mit Hilfe von endlichen Automaten zu erstellen. Damit können dann mögliche Schwachstellen aufgespürt werden, zudem kann man durch das Modell Ratschläge für eine ordnungsgemäße Benutzung der SUIDs geben [CWD02]. Um beispielsweise die Ausnutzung einer solchen SUID-Schwäche zu verhindern, bietet das LSM eine sichere Trennung von einfachen und privilegierten Ressourcen und somit auch eine Abschottung von Kernelressourcen einerseits und andererseits gegenüber Prozessen aus dem Benutzerbereich.

3.1.1 Grundansatz

Der Grundansatz der Sicherheitsarchitektur ist, dass der Zugriff auf alle Betriebssystemressourcen kontrolliert werden soll. Die erste mögliche Rechteprüfung kann beim Aufruf des jeweiligen System Calls stattfinden. Diese Methode wurde aber aus mehreren Gründen nicht gewählt. Einerseits müsste man dafür in allen System Calls einen Zugriff für LSM einbauen, dies wäre aufwändig und passt nicht zu dem generischen Anspruch, der erfüllt werden soll. Andererseits kann es durch einen Hook direkt beim System Call passieren, dass der Zugriff bei der Überprüfung, ob der System Call erlaubt ist, noch bestätigt wurde, beim wirklichen Aufruf des SysCalls kann die Rechtevergabe aber bereits geändert sein. Dann würde der Zugriff fälschlicherweise erlaubt werden, außerdem würde auf diese Art und Weise zweimal auf die Daten zugegriffen werden. Aus diesem Grund wurde auf die Lösung des Kernelmoduls gesetzt. Somit wird die Kommunikation nicht zwischen User Space und Kernel Space abgefasst, sondern erst im Kernel direkt vor dem Zugriff auf das geforderte Kernelobjekt. Hierfür wird im Kernelcode direkt vor dem Zugriff auf den Inode ein Hook gesetzt, der die Sicherheitsrichtlinie überprüft. Bei einer möglichen Ablehnung der Zugriffsanforderung wird eine Fehlermeldung zurückgegeben. Mit dieser kann dann der fehlerhafte Zugriff überprüft und ausgewertet werden, wodurch die Sicherheit des Systems weiter verbessert werden kann. Im Abschnitt 3.1.4 werden die Hooks im Detail erläutert. [WCS+02]

3.1.2 Einbindung ins System

Beim Systemstart wird das LSM Framework sehr zeitig initialisiert. Über die Funktion `security_init` wird schon während der Konfiguration von grundlegender Hardware wie den CPUs und dem Arbeitsspeicher und damit vor jedem Userprozess und dem Init Prozess das LSM initialisiert. Es bildet dann über Dummy Hooks, welche in über `default_security_ops` geladen wer-

den, die standardmäßige UNIX Zugriffskontrolle mit einem Superuser ab. Danach erfolgt eine Registrierung des primären Sicherheitsmoduls über die folgende Funktion.

```
int register_security(struct security_operations *ops)
{
    if (verify(ops)) {
        printk(KERN_DEBUG "%s could not verify "
        "security_operations structure.\n", __func__);
        return -EINVAL;
    }
    if (security_ops != &default_security_ops)
        return -EAGAIN;

    security_ops = ops;
    return 0;
}
```

[lin10]
Hierbei wird überprüft, ob die angegebenen Sicherheitsoperationen (`ops`) gültig sind. Eventuell vorher geladene Module werden mit dem neuen überschrieben und somit werden auch die Hooks des gerade registrierten Moduls genutzt. Im geladene Sicherheitsmodul kann festgelegt werden, ob das Modul, wenn einmal geladen, wieder per `unregister_security` entfernt werden kann oder nicht. Vor und nach der Benutzung eines solches Modul im Kernel werden durch das LSM selbst nur Funktionen geladen, die sich wie die traditionelle Unix Zugriffskontrolle verhalten.
Eine Art Stacking von mehreren Modulen ist prinzipiell erlaubt, funktioniert aber nicht mit beliebigen Sicherheitsmodulen und erzeugt potentiell Probleme. Das erstgeladene Modul ist gleichzeitig das primäre Sicherheitsmodul, dieses wird immer über die Kernelfunktion `register_security` geladen. Später geladene Module werden als Submodule dem primären untergeordnet, melden sich demnach auch nicht beim Kernel an, sondern bei dem erstgeladenen Modul. Dadurch ergibt sich zum Beispiel der Umstand, dass es immer nur ein Set von Sicherheitsoperationen gibt, welches sich am Kernel anmeldet. Dies ist einerseits dem Kernel an sich geschuldet, da er nur eine `security_ops` Tabelle zulässt, andererseits ist das LSM auch so konzipiert, sodass diese Lösung genutzt wird. Weitere Operationen der Submodule können nur kontrolliert von der primären Sicherheitskomponente arbeiten. Diese Tatsache ist problematisch, da Linus Torvalds das LSM unter der Bedingung in den Kernel aufgenommen hat, dass mehrere Sicherheitsmodule parallel arbeiten können

sollen. Kritisch ist weiterhin, dass jedes Modul festlegen kann, mit welchen anderen Modulen es zusammenarbeitet. [WCS+02, BTS09]

3.1.3 Label Management

Mit dem Linux Security Module Framework werden sicherheitsrelevante Kernelstrukturen um weitere Strukturen ergänzt, welche dann mit Labeling Informationen gefüllt werden können. Im Speziellen werden zu den Kerneldatenstrukturen, die in der Tabelle 3.1 aufgezeigt werden, `void` Pointer hinzugefügt. Über diese werden dann die Label Informationen mit den Kernelelementen verbunden. Durch diese Art von erweiterten Kernelsicherheitsfeldern (engl. Opaque Security Field) wird eine eindeutige Kennzeichnung der einzelnen Kernelobjekte sichergestellt und damit eine bessere Zugriffskontrolle gewährleistet, auch sind Manipulationen leichter erkennbar. Die Steuerung dieser erweiterten Kernelsicherheitsfelder übernimmt das geladene Sicherheitsmodul, im LSM selbst werden nur `void` Pointer wie hier im Beispiel zu einem Inode angegeben.

```
struct inode {
    #ifdef CONFIG_SECURITY
    void                    *i_security;
    #endif
};
```

Diese werden dann je nach Policy mit Funktionalität gefüllt. Das erforderliche Management erfolgt zum größten Teil über Hooks wie `alloc_security` und `free_security` direkt bei der Erstellung bzw. Löschung der entsprechenden Kernelstruktur. Zudem ist es wichtig zu erwähnen, dass die Sicherheitsfelder nicht automatisch durch das LSM vor mehreren gleichzeitigen Zugriffen geschützt werden - dies muss auch das geladene Sicherheitsmodul übernehmen. [WCS+02]

3.1.4 Hooks

Um das Prinzip der Linux Security Modules umzusetzen, wird vor dem Zugriff auf schützenswerte Kernelelemente der Sicherheitskontext geprüft. Dies geschieht, indem über Funktionspointer in der jeweiligen Kernelstruktur eine Sicherheitsfunktion aus der `security_ops` Tabelle des geladenen Policy Moduls gestartet wird. Als Ergebnis wird dann eine Fehlermeldung oder eine Zugriffserlaubnis zurückgegeben. Dieser Vorgang wird als (Kernel) Hook bezeichnet, eine schematische Darstellung findet sich in der Abbildung 3.2. Hooks

Struktur	Objekt
`task_struct`	Task (Prozess)
`linux_binprm`	Programm
`super_block`	Dateisystem
`inode`	Pipe, Datei, Socket
`file`	Offene Datei
`sk_buff`	Network Buffer (Packet)
`net_device`	Network Device
`kern_ipc_perm`	Semaphore, Shared Memory Segment oder Message Queue
`msg_msg`	Indviduelle Nachricht

Tabelle 3.1: Vom LSM Framework erweiterte Kernelstrukturen. Erweiterungen umschließen Funktionspointer zu Hooks sowie `void` Pointer für die erweiterten Kernelsicherheitsfelder [WCS+02]

ermöglichen zum einen den Zugriff auf die eben erläuterten erweiterten Kernelsicherheitsfelder, zum anderen erlauben sie Funktionsaufrufe zum Steuern der damit referenzierten Objekte. Im folgenden werden einige Kernel Hooks an der folgenden Funktion `vfs_mkdir` erläutert.

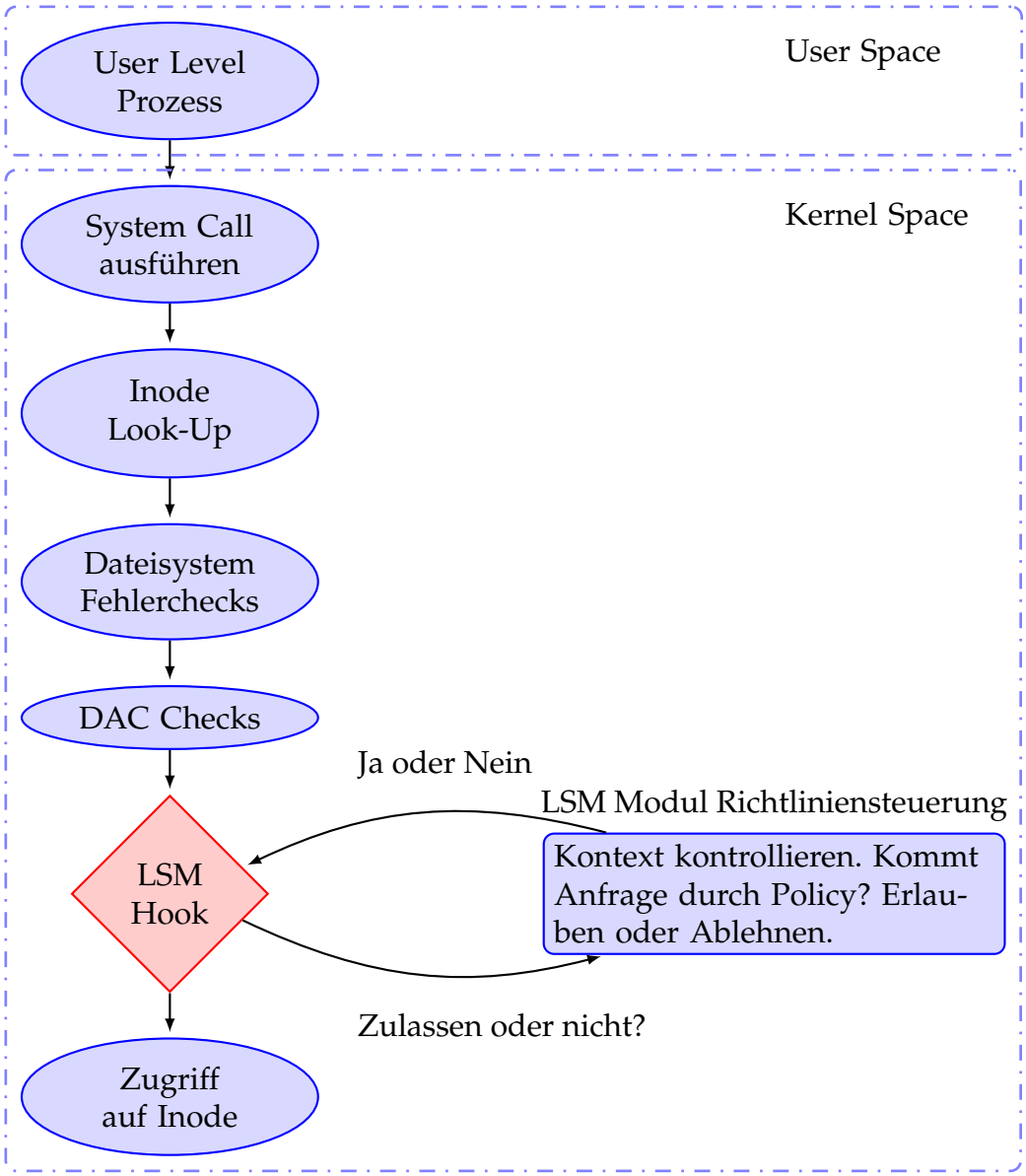


Abbildung 3.2: Veranschaulichung der Hook Architektur und dabei Einordnung des LSM-Hooks in die herkömmlichen Zugriffskontrollen basierend auf einem Inode. (nach [WCS+02])

```
int vfs_mkdir(struct inode *dir, struct dentry *dentry,
              int mode)
{

    int error = may_create(dir, dentry);

    if (error)
        return error;


    if (!dir->i_op->mkdir)

        return -EPERM;

    mode &= (S_IRWXUGO|S_ISVTX);

    error = security_inode_mkdir(dir, dentry, mode);

    if (error)
        return error;

    vfs_dq_init(dir);
    error = dir->i_op->mkdir(dir, dentry, mode);
    if (!error)
        fsnotify_mkdir(dir, dentry);
    return error;
}
```

Aus mehreren Gründen heraus wurden die Hooks direkt in der Kernelstruktur eingebaut. Beispielsweise kann durch die Funktion `may_create`, welche eine Standardzugriffskontrolle ist, nicht der genaue Zugriffsmodus und die geforderte Operation überprüft werden, sondern nur der Dateiname und das momentane Verzeichnis der Datei. Die Funktion `dir->i_op->mkdir` als zweite Standardüberprüfung fällt zudem als mögliche Einsprungstelle weg, weil zu internen Kernelfunktionen keine zusätzlichen Hooks hinzugefügt werden. Dies ist dem Fakt geschuldet, dass für Kernelmodule auch nicht-GPL-konforme Lizenzen verwendet werden können. Möglichen Problemen hierbei gehen Kernelentwickler somit aus dem Weg. Demnach wird mit dem Hook `security_inode_mkdir` die Erstellung des Verzeichnisses überwacht, zudem werden unter Umständen neue Informationen in das erweiterte Kernelsicherheitsfeld des Objektes geschrieben. [WCS+02]
Die soeben dargestellten Probleme bei der richtigen Positionierung und Anwendung von Hooks haben verschiedene Forschungsprojekte dazu veranlasst,

sich kritisch mit diesen Themen auseinanderzusetzen. Beispielsweise wurde die Konsistenz von gesetzten Hooks in dem gegebenen Kontext auf Korrektheit geprüft, indem die auszuführenden Hooks mit den entsprechenden sicherheitskritischen Operationen in Relation gesetzt werden [JEZ04]. Eine weitere Arbeit beschäftigt sich ausführlich damit, wie die Hooks an den richtigen Stellen automatisch gesetzt werden können [GJJ05]. Im folgenden gehe ich auf einige Bereiche ein, in denen Hooks verwendet werden.

3.1.4.1 Anwendungshooks

Zur Steuerung und Administration von Sicherheitsinformationen von Prozessen bietet das LSM mehrere Hooks, um beispielsweise bei einem `ls | grep "text"` die Kommunikation zwischen zwei oder mehr Prozessen abzusichern. [WCS+02]

3.1.4.2 Hooks beim Ausführen von Anwendungen

Das erweiterte Kernelsicherheitsfeld der Kernelstruktur `linux_binprm` verwaltet Sicherheitsinformationen beim Start von Prozessen. Bei der Ausführung von Programmen erfolgen teilweise Änderungen des Benutzerkontextes, welche auch angepasste Rechte nach sich ziehen. Dafür gibt es Hooks in der `execve` Funktion, um eine Ausführung mit den neuen Rechtevergaben zu prüfen, weiterhin werden durch einen weiteren Hook nach dem Starten der Anwendung auch die bestehenden Sicherheitsoptionen auf den neuen Kontext hin aktualisiert. [WCS+02]

3.1.4.3 Hooks rund um das Dateisystem

In Dateisystemen unter Unix, also auch unter Linux, werden Einträge jeglicher Art auf der Festplatte durch Inodes adressiert. Im Detail werden Inodes durch Inode-Nummern identifiziert, diese Nummer verweist dann erst auf eine Datei, wodurch der Datenzugriff möglich wird. Damit nehmen sie eine zentrale Rolle bei der Verwaltung des Dateisystems dar. Demnach ist es sinnvoll, an dieser Ebene anzusetzen, wenn eine zusätzliche Kontrollebene wie über das LSM eingeführt wird. Bei der Beschreibung der Hooks für Dateisysteme und der dazugehörigen Kernelstruktur `super_block` beziehe ich dem folgend die Hooks für Dateien und Inodes sowie deren Strukturen `file` und `inode` mit ein. Vom Dateisystem absteigend kann so herunter bis zum Inode durch Hooks eine feinere Zugriffskontrolle stattfinden, sei es beim Mounten von Dateisystemen oder bei der Prüfung, ob ein Inode an dieser Stelle erzeugt werden darf oder nicht. Als mögliche Alternativen könnten die Kernelstrukturen `dentry` oder `vfsmount` fungieren, allerdings referenzieren diese beiden Funktionen

auch auf `inode` respektive `super_block` Strukturen. Auf Grund dieses Alias Charakters ist derlei Möglichkeit nicht für den Einsatz geeignet. [WCS+02]

3.1.4.4 Netzwerk Hooks

Für die Kommunikation zwischen mehreren Rechnern oder ähnlichen Systemen in Netzwerken werden fast immer Sockets verwendet, um eine Verbindung zwischen den Endstellen herzustellen. Diese Sockets werden durch Hooks um Schnittstellen für den zusätzlichen Zugriffsschutz erweitert. In etwa handeln auf der Netzwerkebene mehrere Socket Hooks die Verbindung zwischen der Anwendungsschicht und dem Netzwerkbereich aus. Zum Beispiel findet etwa für ausgehende Pakete über IPv4 oder IPv6 eine Überprüfung statt, ob die geforderte Verbindung erlaubt ist oder nicht. Im Kernel werden in die Struktur `sk_buff`, welches Netzwerkpakete erstellt sowie in der `net_device` Struktur erweiterte Kernelsicherheitsfelder hinzugefügt, um einerseits auch über Netzwerkschichten hinweg Sicherheitsaspekte abdecken zu können und andererseits auch Netzwerkgeräte überprüfen zu können. Die bestehenden Hooks des zur Netzwerkanalyse genutzten Frameworks Netfilter, u. a. bekannt durch den IPTables Teil des Projektes, werden weiterhin genutzt und zudem durch die LSM-Funktionalität erweitert. [WCS+02, net10]

3.1.4.5 Weitere Hooks

Weitere Hooks existieren in etwa für die Interprozesskommunikation, wobei auch Schnittstellen für System V Derivate bereitgestellt werden. Ferner existieren auch Hooks zur Kontrolle und Änderung von Einstellungen wie dem Systemnamen oder den Zugriff auf eingehende oder ausgehende Ports. In herkömmlichen Linux Systemen wurden hierfür einfache POSIX Capability Checks verwendet, welche allerdings nur eine sehr grobe Kontrolle geben. [WCS+02]

3.1.5 Audit mit LSM

Durch das LSM wird ein Audit Interface bereitgestellt, welches Hooks bereitstellt mit denen Entscheidungen des LSM protokolliert werden können. Mit den Funktionen

- `audit_rule_init(field, op, rulestr, lsmrule)`
- `audit_rule_known(krule)`
- `audit_rule_match(secid, field, op, rule, actx)`
- `audit_rule_free(rule)`

können Vorgänge wie der Start neuer Prozesse, der Aufruf von Syscalls sowie Dateioperationen protokolliert werden. Diese Aufzeichnungen werden dann standardmäßig in der Datei `/var/log/messages` gespeichert. Sobald aber der Linux Auditdienst aktiv ist, werden die Meldungen (auch) nach `/var/log/audit/audit.log` geschrieben. In diesem Fall können die Logs dann durch Programme wie `ausearch`, `aureport` oder `autrace`, welche dem Auditdienst angehören, weiterarbeitet werden. Auditfunktionen sind eine maßgebliche Voraussetzung für bestimmte Zertifizierungen nach der Common Criteria. Allerdings hilft der LSM Audit hier nicht wirklich weiter, denn es werden nur Entscheidungen protokolliert, welche die MAC betreffen, nicht aber die bereits vorher stattfindenden Überprüfungen zur DAC. Für weitergehende Zertifizierungen ist jedoch ein kompletter Systemaudit unabdingbar. [Spe07]
Die Funktionen zum Aufnehmen und Auswerten von Daten im Betrieb eines Computersystems wird durch das LSM Framework allerdings prinzipiell erweitert. Die konstante Nutzung von Hooks zur Überprüfung von Zugriffsrechten macht es in vielen Situationen einfacher, Fehlermeldungen auszuwerten. Dies soll im folgenden anhand von Netzwerk Sockets kurz erläutert werden. In einem herkömmlichen Linux System werden Netzwerkpakete unter Umständen von softwarebasierten Firewalls wie IPTables geblockt. Allerdings werden darüber keine Fehlermeldungen generiert, daher sendet die Anwendung einfach weiter Pakete, bis ein Timeout erreicht wird. Mit dem LSM wird schon bei dem Versuch, einen Socket zu öffnen, erkannt, das der derzeitige Benutzer keine ausreichenden Rechte für diesen Bereich besitzt. Daraus folgend wird der Anwendung übermittelt, dass die angeforderte Aktion nicht erlaubt ist. Sinnvollerweise muss nun auch kein Timeout mehr abgewartet werden, zudem kann man durch die Fehlermeldung in vielen Fällen die Ursache besser eingrenzen als durch einen simplen Timeout. [BTS09]

3.1.6 Performance

Die Performance des LSM Frameworks ergibt für ein Computersystem nur eine minimale Mehrbelastung, allerdings ist zusätzlich immer mindestens ein Policy Modul notwendig, wodurch mit etwas größeren Einbußen zu rechnen ist. Im folgenden wurde mit einem Linux Kernel 2.5.15 und dem LMBench Microbenchmarks getestet, wobei ein Kernel ohne LSM gegen einen mit LSM gepatchten Kernel inklusive POSIX.1e Modul antritt. Verglichen wurden dann verschiedene Prozesstests wie zum Beispiel null call oder exec proc, aber auch Dateioperationen wie Anlegen und Löschen von Dateien. Ein weiterer Bereich war die Kommunikation zwischen Prozessen, beispielsweise über pipe oder mem read/write. Hier war dann vor allem die Bandbreite bei der Übertragung interessant. Bei den allermeisten Tests gab es im Vergleich zu dem ungepatch-

ten System einen Overhead von meist weniger als zwei Prozent, dies entspricht demnach üblichen Schwankungen bei Tests dieser Art. Bei einem weiteren Test, bei welchem ein Kernel kompiliert werden sollte, gab es keine Unterschiede zwischen beiden Varianten. Für den letzten vorgestellten Test wurde mit Hilfe des Webstone Benchmarks die Anzahl der möglichen Verbindungen zwischen 8-32 Clients pro Sekunde gemessen. Alle verwendeten Kernel verwenden den eingebauten Netfilter Support, um wiederum gleiche Voraussetzungen zu bilden. Der Overhead zwischen einem Standard-Kernel und dem LSM beträgt zwischen knapp 5-7 Prozent, zwischen dem Standard-Kernel und einem Kernel, welcher SELinux als Policy verwendet, allerdings schon 16-21 Prozent. Insgesamt kann man an den hier festgestellten Testergebnissen feststellen, dass durch das LSM Framework allein kaum Mehrleistung vonnöten ist - allerdings werden durch die Einbindung von entsprechenden Policy Modulen wie SELinux durchaus mehr Ressourcen für die gleichen Arbeiten benötigt. [WCS+02]

3.1.7 Anwendungen

SELinux (security-enhanced Linux) SELinux wurde unter einer speziell auf Sicherheit ausgelegten Betriebssystemarchitektur - der Flask Architektur - entwickelt, die flexibel Sicherheitspolicies unterstützen soll. Laut Flask wird zudem auf nicht benutzerbestimmbare Zugriffskontrolle - bei SELinux systembestimmt - gesetzt. [fla10] Diese Grundstruktur bietet eine Codetrennung von Richtliniendurchsetzung und der voranliegenden Entscheidungsfindung. Zudem beinhaltet SELinux einen Access Vector Cache (AVC), welcher zuvor getroffene Entscheidungen zwischenspeichert und somit den Leistungseinbruch durch die zusätzliche Zugriffskontrolle klein zu halten. Komplexe Regelsätze sind mit SELinux kein Problem und Beschränkungen jeder Art sind möglich. Allerdings bringen die feingranularen Möglichkeiten der Beschränkungen Probleme für unerfahrene Nutzer von SELinux mit sich, andere Anwendungen gestalten die Regelerstellung benutzerfreundlicher. Bei dem Webserver Benchmark Webstone schneidet SELinux eher schlecht ab, hier sind bei den Serververbindungen Einbußen von bis zu 21.6% zu verzeichnen. [WCS+02, SVS06]

TOMOYO Linux Ein weiteres Policy Modul zur Umsetzung von systembestimmter Zugriffskontrolle ist TOMOYO. Die Besonderheit liegt hier auf pfadbasierten Label Typisierung, des weiteren liegt der Fokus auf einer möglichst automatisierten Policygenerierung, welche im laufenden Betrieb stattfinden kann. Zur Zugriffsprüfung werden einfache Dateisystemattribute wie Lesen, Schreiben oder Zugriff für gewählte Pfade anstatt aufwendiger System Call Hooks genutzt. Im Vergleich zu SELinux benötigt TOMOYO zudem weniger Speicher oder sonstige Ressourcen. Allerdings kann ein Sicherheitskontext

nicht so detailliert geltend gemacht werden wie mit SELinux. Die Einstellungen zur Beschränkung von einzelnen Programmen kann gleichwohl sehr einfach getätigt werden. Hierdurch sind die entstehenden Regelsätze übersichtlich und einfach zu verstehen. [BBSS07, AB09, Mor09]

POSIX.1e Capabilities Die Möglichkeiten, welche die POSIX.1e Capabilities unter Linux schon vor dem LSM Patch boten, wurden aus dem Kernel herausgelöst und in einem Policy Modul gebündelt. Um durch das LSM eine ähnliche Funktionalität in Bezug auf POSIX.1e wie im Standard Linux Kernel zu erreichen, wurde das entsprechende Interface `capable` zu einem Hook umgebaut, wodurch jedwede gewünschte Logik in diesem Sicherheitsmodul untergebracht werden kann. Hierdurch wird dann der Einsatz von POSIX.1e Sicherheitsfunktionen durch das LSM gesteuert und kontrolliert. Die Capabilities eines einzelnen Prozesses werden nicht über die erweiterten Sicherheitsfelder der Kernelstruktur gespeichert, damit das Sicherheitsmodul leichter mit anderen Modulen zusammen geladen werden kann. Diese Lösung wurde angestrebt, weil die POSIX.1e Logik im Standard Kernel von vielen Anwendungen genutzt wird, die Sicherheitsfelder aber auch bei mehreren geladenen Sicherheitsmodulen nur für die erstgeladene vorhanden sind. [WCS+02]

AppArmor Eine weiteres System zur einfachen Umsetzung von Mandatory Access Control ist AppArmor. Sicherheitskritische Anwendungen können isoliert werden, ihre Rechte können damit einfach beschnitten werden. Für einen Anwender können so Programme, welche mit möglicherweise unsicheren Daten wie Mails arbeiten müssen, einfach vom Grundsystem getrennt werden. Im Vergleich zu SELinux sind Regeln für die Policy einfacher zu schreiben und zu verstehen - zu jeder Anwendung wird einfach der Pfad und die erlaubten Aktionen angegeben. Durch diese Angabe soll die Benutzbarkeit des Sicherheitsmoduls gestärkt werden. Geprüft werden können so Dateizugriffe über die Angabe des Dateipfades und zudem sind mit den Regeln Überprüfungen nach dem POSIX Capability Standard möglich. AppArmor besitzt zudem einen Lernmodus, welcher die Erstellung einer Policy unterstützen soll. [Mor09, MG08, Spe06]

SMACK (Simplified Mandatory Access Control Kernel) Der Ansatz von SMACK geht dahin, einzelne Benutzer und Ressourcen zu kontrollieren oder zu beschränken. Das System ist eher nicht auf einen allgemeinen Einsatz wie SELinux ausgelegt, da es beispielsweise Type Enforcement nicht umsetzt. Der Name SMACK beinhaltet passend das wichtige Grundprinzip Simplizität, welches bei der Labelgenerierung und auch bei der Policyerstellung umgesetzt wird. Bei der Erstellung einer Richtlinie in etwa wird durch die einfache An-

gabe von `Subjekt Objekt [-rwxa]` eine Regel erstellt. Der Administrator hat die Möglichkeit bis zu sieben Zeichen für ein Label zu verwenden, zudem kann durch die Angabe der reservierten Zeichen _ (floor), * oder ^ (hat) angegeben werden, ob ein niedriges oder hohes Sicherheitslevel besteht. [Mor09, Mor08, Mor10, Edg07]

LSM Portierung vom Openwall Kernel Patch Die Portierung des Openwall Projektes hin zu einem LSM Modul soll nur einen Teil der Funktionalität des Original Openwall Kernel Patch aufweisen. Dazu zählen Schutzmaßnahmen gegen Angriffe wie Race Conditions oder Buffer Overflows. Im Detail wird allen Benutzern außer Root verboten, Hard Links auf Dateien zu setzen, die ihnen nicht gehören. Darüber hinaus darf Root keine symbolischen Links benutzen. [WCS+02, Cow03]

DTE (Domain and Type Enforcement) Die Umsetzung von DTE mit dem LSM Framework gibt dem Administrator die Möglichkeit, einerseits allen Dateien - also Objekten - im System Typen zuzuordnen, andererseits werden Prozesse zu Domänen zugeordnet. Die Umsetzung der Mandatory Access Control durch die DTE Richtlinie erfolgt dann, indem bei Dateizugriffen von den Domänen auf Typen die Rechte überprüft werden. Weiterhin wird der Austausch von Mitteilungen über Systemereignisse zwischen Prozessen mit unterschiedlicher Domäne überprüft. In der Richtlinie kann zudem festlegt werden, welche Domänenwechsel erlaubt sind. [WCS+02, HK04]

TuxGuardian Mit TuxGuardian ist eine weitere Anwendung des LSM entstanden. Es setzt auf der Netzwerkebene an und kontrolliert dort die Erstellung sowie die weitere Nutzung von Sockets für Anwendungen. Zur schnellen Auswertung existiert eine grafische Benutzeroberfläche für den Administrator, um ungewünschte Netzwerkoperationen für Programme zu erkennen. Über diese Oberfläche kann dann auch eine zusätzliche Regel für die Policy erstellt werden, um für kommende Operationen dieser Anwendung eine Nutzung zu erlauben. [MG08] TuxGuardian scheint keine aktive Entwicklung zu besitzen, das letzte Update ist vom April 2006. [tux10]

grsecurity und Dazuko Beide Sicherheitserweiterungen entwickeln inzwischen aufgrund verschiedener Gründen nicht mehr mit dem LSM Framework, daher werde ich hier nicht weiter darauf eingehen. Weitere Informationen dazu gibt es im nächsten Abschnitt sowie unter [daz10, grs10].

3.1.8 Kritik

Trotz oder vielleicht gerade wegen dem vorrangigem Fokus des LSM Frameworks auf zusätzliche Sicherheit bei der Zugriffskontrolle gibt es einige Probleme, zum Beispiel mit den bereits vorhandenen Sicherheitserweiterungen grsecurity und RSBAC. Beide Erweiterungen bieten in bestimmten Bereichen eine breitere Funktionalität als durch das LSM bereitgestellt wird, auch abseits der Zugriffskontrolle. Bei RSBAC sind dies beispielsweise Funktionen zum sicheren Löschen oder zum partiellen Abschalten der Standard DAC. Dadurch müssten die Projekte ohnehin einen eigenen Kernelpatch bereitstellen. [rsb10]

Zudem sehen die beiden Erweiterungen im LSM einen grundsätzlich falschen Ansatz, der Linux auf lange Sicht mehr Schaden als Nutzen bringt. Ein Vorwurf von beiden Projekten ist, dass sich bei der LSM Entstehung zu wenig Gedanken um eine künftige Sicherheitsentwicklung und die Notwendigkeit von allgemein nötigen Hooks gemacht wurde. Auch haben die Entwickler von grsecurity Bedenken, dass die gegebenen Schnittstellen des LSM durch schadhaften Code wie etwa Rootkits verwendet werden können, um sich noch besser zu tarnen. Ein weiteres Problem betrifft das Stacking von verschiedenen Sicherheitsmodulen, vor allem von RSBAC genannt, dies wurde in der Arbeit bereits unter Abschnitt 3.1.2 erläutert. Die beiden Projekte äußern außerdem die Meinung, dass zum Umschreiben auf das LSM Framework viel Zeit in Anspruch genommen werden müsste, welche besser zum Lösen von neuen Problemen genutzt werden kann. Auch wird moniert, das nötige Hooks zum Teil nicht vorhanden sind oder von den Linux Kernel Entwicklern bereits abgelehnt worden sind. Ein weiteres negativ beeinflusstes Projekt ist Dazuko, hier wird durch das Laden von anderen Erweiterungen wie SELinux oder AppArmor eine sinnvolle Funktionsweise von Dazuko nahezu unmöglich. [rsb10, grs10, daz10]

Zudem existiert nach meinen Erkenntnissen keine Sicherheitserweiterung für das LSM, welche es erlaubt, nur kryptographisch gesicherte Kernelmodule oder ausführbare Dateien zu starten. Eine solche Erweiterung würde eine Kompromittierung des LSM weiter erschweren.

3.2 TrustedBSD MAC Framework

TrustedBSD ist ein Teilprojekt von FreeBSD, in welchem Sicherheitserweiterungen konzipiert und erstellt werden, die nach Fertigstellung wieder in das Mutterprojekt integriert werden. Der Weg des TrustedBSD MAC Frameworks ist es, als Kernelmodul direkt Einfluss auf Zugriffskontrollentscheidungen im System zu nehmen. Weiterhin erfolgt auch hier eine Abstraktion der allgemeinen Struktur des Frameworks von den Sicherheitsrichtlinien, wodurch zur Erstellung der Richtlinien weniger Komplexität vonnöten ist. Ohne geladene Sicher-

heitspolicy gibt das MAC Framework keine zusätzlichen Kontrollmöglichkeiten, es bietet nur eine Gerüst, welches von den Policies genutzt wird. Unter `sys/mac_policy.h` werden ladbare Policies sowie Schnittstellen zur Registrierung von Richtlinien angegeben.
Ein großer Vorteil bei der Konzeption des MAC Frameworks ist, dass es keine Einschränkungen bei der Tiefe des Eingriffs auf Kernelebene gibt, weil durch die Ausgliederung aus dem FreeBSD Projekt der Sicherheitsgedanke höher gestellt wird als im Mutterprojekt. Dadurch können einerseits bestehende Sicherheitsmechanismen genutzt werden, anderseits ist man darauf aber nicht festgelegt und kann an der nötigen Stelle vielversprechendere Ansätze verwirklichen. Das Framework soll es ermöglichen, beliebige Zugriffskontrollmodelle zu verwirklichen, hierfür steht der generische Anspruch des Ansatzes. Zudem gibt es die Möglichkeit, mehrere unabhängige Sicherheitsrichtlinien nebeneinander zu laden und zu einer durch das Framework administrierten deterministischen Arbeitsweise zu gelangen. Eine vereinfachte Darstellung des MAC-Frameworks ist in der Abbildung 3.3 zu sehen. [WFMV03]

3.2.1 Grundansatz

Das TrustedBSD MAC Framework möchte im Grundansatz zusätzliche Sicherheit bei der Zugriffskontrolle auf Betriebssystemressourcen bringen. Dieser Ansatz wird durch eine Modifizierung des bestehenden Kernels an den sicherheitsrelevanten Stellen erreicht. Die erweiterten Funktionen wie zum Beispiel die Labelverwaltung sowie die Zugriffskontrolle werden dabei eng mit den bestehenden Sperren für den exklusiven Zugriff auf bestimmte Ressourcen verbunden. Bei dem TrustedBSD MAC Framework erfolgt eine Dreiteilung der Interfaces zum Zugriff auf das Framework. Den ersten Teil betrifft den Kernel, hier werden wichtige Voraussetzungen für die Policies und das Labeling geregelt, weiterhin können wichtige Sicherheitsentscheidungen protokolliert werden. Das MAC-Framework bietet verschiedene Zugriffspunkte für die Labelverwaltung der Kernelobjekte und für Kernelfunktionen, damit diese mit dem Framework kommunizieren können. Im zweiten Part werden für die Sicherheitsrichtlinien Management- und Kontrollmethoden bereitgestellt. Das dritte Teilinterface gibt dem Nutzer die Möglichkeit, Änderungen der Richtlinien sowie die geladenen Policies und vergebenen Labels zu sichten. [WFMV03]

3.2.2 Einbindung ins System

Der Startvorgang des Kernel Frameworks findet während des Bootvorgangs bereits kurz nach der Initialisierung von grundlegenden Systemelementen wie dem Kernel Memory Allocator, der Konsole oder Locking Grundlagen - aller-

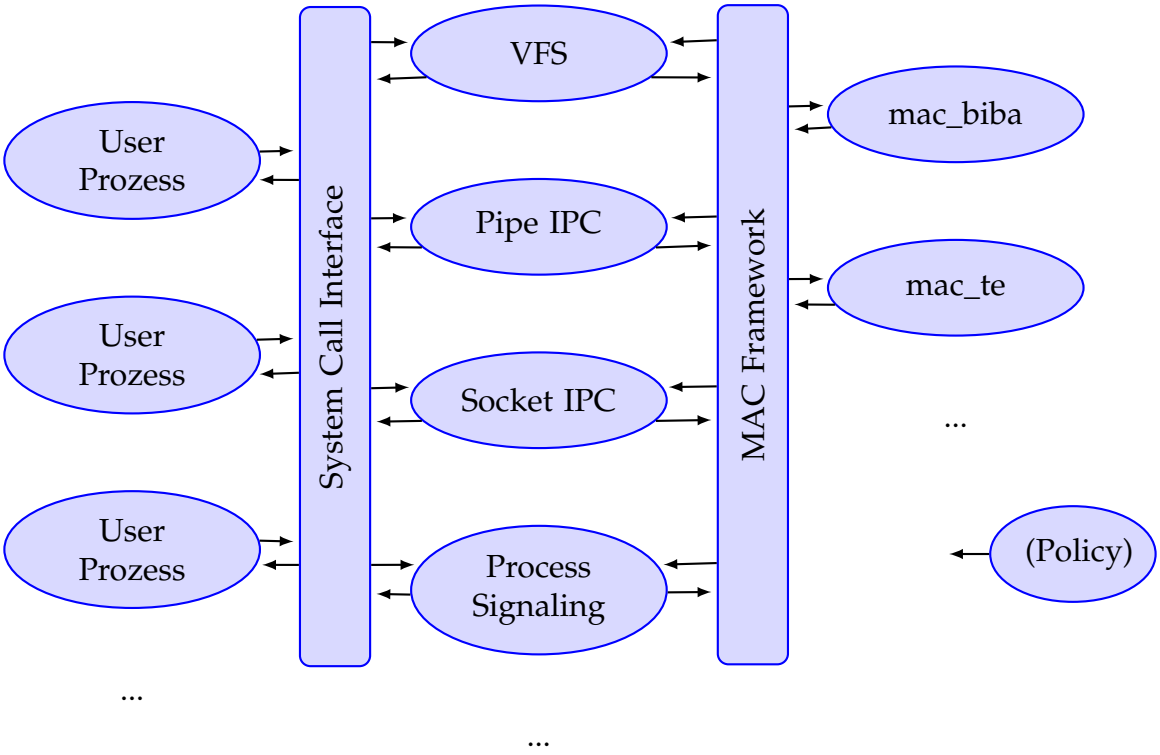


Abbildung 3.3: Veranschaulichung der Funktionsweise des FreeBSD MAC Frameworks. Ähnlich wie bei dem LSM Framework wird vor dem Zugriff auf Kernelelemente über System Calls durch einen Benutzer eine Rechteüberprüfung durchgeführt. Interessant ist hier, dass sich neue Policies direkt an dem MAC Framework anmelden, dieses kümmert sich um eine Einbindung und Nutzung an der richtigen Stelle. (nach [VW03])

dings vor jeglichem Kernel- oder Userprozess sowie der Geräteerkennung statt. Nach dem Aufruf werden die geforderten Zugriffskontrollrichtlinien geladen, zudem werden Verknüpfungen mit anderen Kerneldiensten ermöglicht, welche die Interaktion mit weiteren wichtigen Kernelobjekten zulassen. Eine zusätzliche Besonderheit sind die sogenannten früh registrierten Policies. Diese haben die Möglichkeit, bei der Geräteinitialisierung und bei Zuweisung von Kernelstrukturen allgegenwärtige Labels zu vergeben. Später geladenen Richtlinien ist diese Möglichkeit der Labelvergabe nicht möglich.
Die Registrierung von Policies erfolgt nach dem Laden eines Kernelmoduls, welches Beschreibungen zu den ladbaren Sicherheitsrichtlinien enthält. Allerdings kann ein Modul unter Umständen mehrere Policies enthalten, die nicht zwangsläufig zusammen geladen werden. Durch die Angabe von Eigenschaften für eine einzelne Policy kann bestimmt werden, ob die Registrierung des mit dem Framework bereits geladenen Kernelmoduls beim Bootvorgang stattfinden darf oder ob erst das System komplett hochgefahren werden muss. Zudem kann man auch festlegen, ob eine einmal registrierte Policy während des Betriebs wieder abgemeldet werden darf oder nicht. Die Registrierung von neuen Policies wird durch durch eine Prozedur von Sperren und Logs abgesichert. Das bedeutet, dass jede Änderung die erfolgreiche und vollständige Abarbeitung alle involvierten ausstehenden Hooks voraussetzt. Demnach kann einen neue Sicherheitsrichtlinie nur aktiviert werden, wenn alle bisher geladenen dem auch zustimmen.
Der Architektur des TrustedBSD MAC Frameworks nach können mehrere Sicherheitsrichtlinien gleichzeitig aktiv sein, in diesem Fall werden die Richtlinien dann von dem Framework an sich geordnet geladen und verwaltet - die Richtlinien müssen demnach nicht selbst für eine reibungslose Zusammenarbeit sorgen. Diese Vorgehensweise ist in Abbildung 3.4 nochmal näher erläutert. [WFMV03]

3.2.3 Label Management

Bei dem TrustedBSD MAC Framework erfolgt eine richtlinienunabhängige Label-Speicherung in Kernelobjekten sowie eine dauerhafte Speicherung von Labels über erweiterte Dateisystemattribute. Diese oft genutzten Sicherheitsfunktionen für das Labeling von Objekten werden direkt durch das MAC Framework bereitgestellt. Allerdings werden die Labels - je nach Policy - aus unterschiedlichen Quellen zusammengesetzt. Die Labels können beispielsweise direkt vom Systemadministrator oder auch durch die Laufzeitumgebung vergeben werden. Weiterhin können auch Dateisystemattribute oder Regelsätze in die Labelvergabe einbezogen werden, zudem gibt es Unterstützung im Labeling für verschiedenste Klassen sicherheitsrelevanter Kernelobjekte wie Datei-

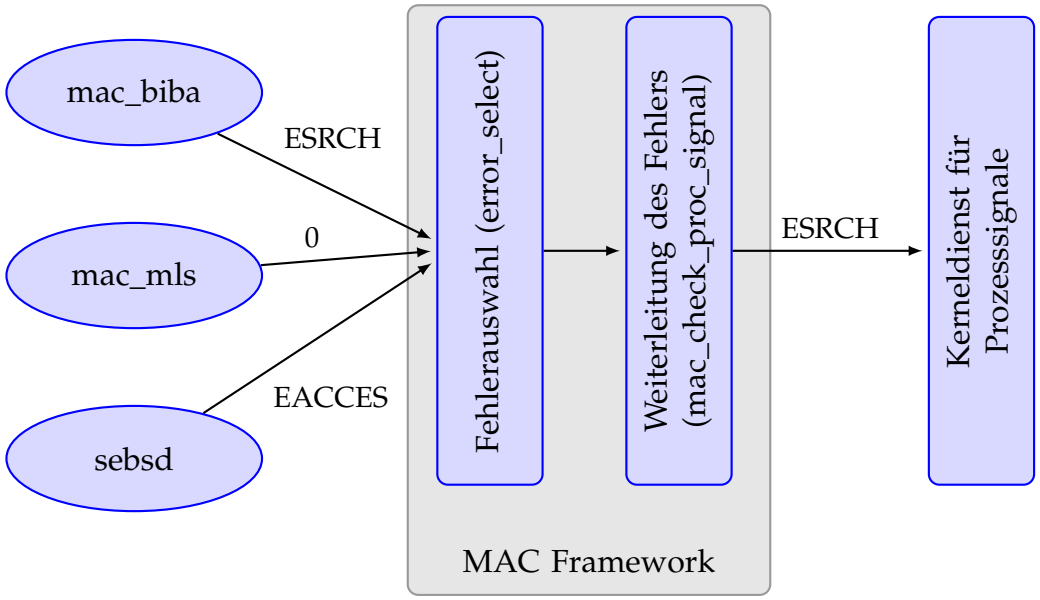


Abbildung 3.4: Mehrere Policies können kombiniert für eine Zugriffskontrolle genutzt werden. Jede der hier drei Policies beantwortet für sich, ob ein Zugriff genehmigt oder abgelehnt wird. Nur wenn alle Policies der Zugriffsanforderung zustimmen, wird diese auch zugelassen. Im MAC Framework wird dann, wenn nötig, ein Fehler ausgewählt, der über eine Kernelschnittstelle an die entsprechende Anwendung zurückgegeben wird. [WFMV03]

systeme, Netzwerk- und Prozesselemente. Alle durch das MAC Framework geänderten Kernelstrukturen sind in Tabelle 3.2 aufgelistet. Jede Policy hat die Möglichkeit, ein Label zu einem Objekt zu vergeben, allerdings werden die Labels immer nur mit ihrem zugehörigen Objekt zusammen initialisiert, belegt oder zerstört. Unter bestimmten Bedingungen wie zum Beispiel einem ungültigen Eintrag in einem dauerhaft gespeicherten Label oder ungenügende Ressourcen zur Erstellung des Labels ist es möglich, dass das Label nicht erstellt wird. Ein wichtiger Gedanke beim Labeling ist es, jeder Policy maximale Freiheit in der Frage, wie das erwünschte Verhalten umzusetzen ist, zu geben. Jede Policy kann in der Labelstruktur Platz reservieren, um Objekte zu referenzieren. Die Referenzen verweisen dann wiederum auf `void` Zeiger oder Integer Werte, womit ein Bezug zu Regelsätzen hergestellt werden kann. Probleme durch nahezu gleichzeitige parallele Zugriffe auf Labels jeder Art werden in der Regel durch Standardsperren des Kernels auf die jeweiligen Kernelobjekte verhindert. [WFMV03, VW03]

Struktur	Objekt
`struct ucred`	Prozessberechtigung
`struct vnode`	VFS Node
`struct socket`	BSD IPC Socket
`struct pipe`	IPC Pipe
`struct mbuf`	Netzwerk Paket
`struct mount`	eingehängtes Dateisystem
`struct ifnet`	Netzwerkgerät
`struct devfs_dirent`	Device Dateisystem Eintrag
`struct ipq`	IP Datenpuffer
`struct bpf_desc`	BPF Paket Sniffer

Tabelle 3.2: Kernelstrukturen, welche durch das TrustedBSD MAC Framework im Labeling Prozess eine eindeutige Kennung erhalten und zudem mit ergänzenden Sicherheitsinformationen klassifiziert werden können. [WFMV03]

3.2.4 Hooks

Die Hooks zum Zugriff auf kritische Systemfunktionen sind auch bei FreeBSD über den kompletten Kernel verteilt, unter anderem beim Prozessmanagement, dem Dateisystem, dem IPC und dem Netzwerk Stack. Dabei kann man allgemein Ähnlichkeiten im Aufbau feststellen. Zu Beginn einer Zugriffsprüfung wird ein Kontrollkontext mit Credentials, Zielobjekt und weiteren Argumenten übergeben. Darauf folgend wird der entsprechende Hook jeder geladenen Policy ausgeführt und auf das Ergebnis gewartet, wobei die Ergebnisse wiederum vom Framework angenommen werden und im Falle von Fehlern auch entschieden wird, welcher Fehler zurückgeliefert wird. Für Policy Hooks im FreeBSD MAC Framework gibt es drei allgemeine Grundtypen. `MAC_PERFORM` wird verwendet, um Ereignisse an interessierte Policies weiterzuleiten. Dies kann verschiedene Bereiche wie Richtlinienänderungen, die Labelverwaltung oder Objektlebenszyklus betreffen. Jedoch wird hierbei kein Rückgabewert angegeben. Die zweite Form des Hooks ist `MAC_CHECK`, worüber ein Hook zur Zugriffskontrolle aufgerufen wird. Dieser Modus kann als Rückgabewert Fehler übermitteln, welche dann durch das MAC Framework ausgewertet werden. Die letzte Art Hook `MAC_BOOLEAN` findet in Spezialfällen Anwendung. Implementiert wird dabei eine Entscheidungsfunktion, welche einen einfachen boolschen Rückgabewert hat. [WFMV03]

3.2.5 Audit

Wie auch das LSM Framework bietet das TrustedBSD MAC Framework einige Möglichkeiten, mit denen Systementscheidungen zur Zugriffskontrolle protokolliert werden können. Dabei werden allerdings wie auch beim LSM keine Informationen über standardmäßige DAC Überprüfungen aufgezeichnet. Am Ende ergibt sich durch die Audit Funktionen ebenfalls das Potential dazu, bestimmte Entscheidungen für Programme transparenter und schneller zugänglich zu machen. Zudem ist bei FreeBSD zu erkennen, dass das interne Auditsystem zum Aufnehmen von Kontrollentscheidungen bevorzugt wird. [WFMV03, Wat04]

3.2.6 Performance

Bisher gibt es keine weitreichenden Performanceanalysen für das MAC Framework. Allerdings gibt es eine prägnante Formulierung in der Arbeit von R. Watson und seinen Kollegen. Den Policies soll es in Eigenverantwortung überlassen sein, Zugeständnisse zwischen Sicherheit, Benutzbarkeit und Performance zu machen. Durch diese Definition können verschiedene Niveaus der Performance den gegebenen Ansprüchen der Anwendungsfälle gerecht wer-

den. Zudem ist in der Arbeit formuliert, dass ein zukünftiges Arbeitsfeld für das Framework auch eine ausführliche Performanceanalyse und -optimierung wird [WFMV03]. Auf eine ausführliche Analyse an dieser Stelle wird allerdings verzichtet, da diese den Rahmen der Arbeit sprengen würde.

3.2.7 Anwendungen

Im folgenden werden einerseits einige wichtige Policies näher erläutert, andererseits gibt Tabelle 3.3 eine Übersicht der gebräuchlichsten Policies des TrustedBSD MAC Frameworks.

SEBSD (security enhanced BSD) Security enhanced BSD ist eine Portierung des Sicherheitsmoduls SELinux nach FreeBSD unter Verwendung des MAC Frameworks. SEBSD setzt die grundlegenden Prinzipien des Flask Prinzips sowie des Access Vector Cache ein um Sicherheitsentscheidungen zu treffen und zwischenzuspeichern. Weiterhin kann der Kernel durch SEBSD Sicherheitsfelder administrieren, welche wiederum helfen, Rechtevergaben für Dateien und Prozesse festzulegen. Ein Großteil der zentralen Hilfsprogramme zur Steuerung von SELinux wurden direkt zu SEBSD übernommen, damit können beispielsweise SELinux APIs genutzt werden. Da FreeBSD mit erweiterten Dateisystemattributen arbeiten kann, konnten teilweise beim Labeling direkt die vorhandenen Möglichkeiten genutzt werden, ansonsten wurden die von SELinux gebotenen Optionen übernommen. Bei den Hooks wurden bis auf den Netzwerk- und System V IPC-Bereich alle von SELinux bereitgestellten Hooks umgesetzt. [VW03]

TrustedBSD Biba Policy Die Biba Policy richtet sich nach dem von K. Biba bereits 1977 entwickelten Sicherheitsmodell mit speziellem Fokus auf Datenintegrität. Dadurch werden Subjekte mit hohem Integritätslevel vor Änderungen durch Subjekte mit niedrigerem Sicherheitslevel geschützt. Der Schwerpunkt Datenintegrität ist vor allem gewählt worden, damit keine Änderungen an Daten zwischen zwei autorisierten Zugriffen erfolgen. Die Umsetzung des Prinzips erfolgt einerseits, indem Subjekte mit hoher Integrität auf niedere Objekte schreibend, aber nicht lesend zugreifen können. Andererseits können Subjekte mit niedrigem Integritätslevel schreibend, aber nicht lesend auf höhere Objekte zugreifen. Diese Policy wird in geschützten Systemen oft dazu verwendet, den sicherheitskritischen Codeanteil - beispielsweise einer Firma - ausreichend vor Änderungen zu schützen. [SC07, WFMV03]

Trusted Execution Mechanism Das TrustedBSD MAC Policy Modul `mac_chkexec` erweitert die MAC Komponente um eine wichtige Kompo-

Policy	Beschreibung
mac_biba	Hierarchische Integrität mit unveränderlichen Labeln
mac_bsdextended	„Dateisystemfirewall“, welche bestehende Rechte und Möglichkeiten nutzt
mac_ifoff	Abschalten von Interfaces
mac_lomac	Hierarchische Integrität mit fließenden Labeln
mac_mls	MLS mit Compartments
mac_none	Prototyp Policy
mac_partition	Sichtbarkeit zwischen Prozessen basierend von Prozess Partition Labels
mac_seeotheruids	Sichtbarkeit zwischen Prozessen basierend auf bestehenden Credentitals
mac_chkexec	kryptographische Überprüfung für Module, ausführbare Dateien und Bibliotheken
mac_test	wechselnde Tests
sebsd	Portierung von SELinux/Flask/TE

Tabelle 3.3: Kurzbeschreibung der wichtigsten Policies für das FreeBSD MAC Framework, diese werden zusammen mit dem Framework herausgegeben und stellen im Vergleich zum LSM eine komfortablere Ausgangslage dar.

nente. Für zu ladende Kernelmodule, ausführbare Dateien und Programmbibliotheken können mittels SHA1 und MD5 Checksummen gebildet werden, welche dann bei jedem Start auf Übereinstimmung geprüft werden. Bei einer Nichtübereinstimmung der gespeicherten und der beim Ausführen ermittelten kryptographischen Fingerabdrücke wird der Zugriff blockiert. [Per10]

4 Auswertung

Im folgenden Abschnitt werden einige Erkenntnisse im Vergleich der MAC Frameworks von Linux und FreeBSD vorgestellt.

4.1 Policies

Bei der Verwaltung von geladenen Sicherheitspolicies gehen die Frameworks unterschiedliche Wege. Der Grundansatz des LSM ist dabei, die erstgeladenene Policy für später geladene verantwortlich zu machen. Einerseits muss somit die erstgeladene Policy ein Verwaltungsinstrument für weitere Policies besitzen, außerdem gibt die Architektur des LSM nur die Möglichkeit, eine `security_ops`-Tabelle mit Sicherheitsoperationen anzulegen. Schon eine zweitgeladene Policy hat somit Probleme, ihre Funktionalität anzubieten. Das FreeBSD MAC Framework hingegen bietet eine generischere Lösung an, indem das Framework an sich verwaltet, in welcher Reihenfolge die geladenen Policies angewandt werden. Bei einer Zugriffskontrollentscheidung wird dann unter Umständen auf eine Erlaubnis jeder einzelnen durchzusetzenden Policy gewartet. Für ein wirklich allgemeines Framework ist das nach meinen Erkenntnissen die bessere Lösung. Seitens LSM Framework scheint damit als Policymodul eher einer der größeren Vertreter wie SELinux oder AppArmor in Frage zu kommen, damit umgeht man das Problem der primären Policy durch einen großen Alleskönner. Spezielle Policies haben es unter dem LSM somit aber deutlich schwerer als bei der Konkurrenz von FreeBSD, die durch wirklich parallele Richtlinien punktet. Unter FreeBSD werden mit der Installation des MAC Frameworks die gebräuchlichsten Policies ausgeliefert. Bei Linux werden je nach Distribution verschiedene Richtlinien präferiert, wobei man allerdings an der Festlegung auf beispielsweise AppArmor *oder* SELinux schon merkt, dass ein paralleles Nutzen mehrerer Policies nicht möglich ist.

4.2 Audit und Sicherheitsfunktionen

Für eine konsequente Durchsetzung von allgemeiner Computersicherheit ist es jedoch zusätzlich zur MAC nötig, Auditfunktionen für das komplette System anzubieten. Dies ist aber keine primäre Aufgabe der Frameworks, allerdings

werden die Entscheidungen zur Zugriffskontrolle innerhalb der Frameworks auch protokolliert und können weiterverwendet werden. Für höhere Zertifizierungen nach der Common Criteria wird aber beispielsweise ein kompletter Systemaudit gefordert. Dieser wird unter Linux mit dem Audit Dienst und unter FreeBSD mit dem OpenBSM Framework angestrebt. Umgesetzt wird dies im Zusammenhang mit höheren Zertifizierungen eher von Linux-Distributionen wie Red Hat Enterprise Linux oder SUSE Linux Enterprise Server sowie bei FreeBSD in verschiedenen Routern. [WWS07] Eine wichtige Schutzfunktion bei der Verwendung von MAC ist es auch, zu überprüfen, ob die zu ladenden Kernelmodule, aber auch ausführbare Dateien gegen Veränderungen beim Start des Systems beziehungsweise auch während des Betriebes gesichert sind. Beim MAC Framework von FreeBSD gibt es für diese Aufgabe eine Policy, hier können sicherheitskritische Objekte durch kryptographische Hashfunktionen gesichert werden. Im Gegensatz dazu gibt es für das LSM keine derartige Entwicklung, wobei dann ein mögliches Angriffsszenario zum Beispiel wäre, dass ein Sicherheitsmodul vor dem Laden ausgetauscht wird und somit das System kompromittiert.

4.3 Labelverwaltung

Ein allgemeines Problem der MAC Frameworks bringt die Label Typisierung mit ihren verschiedenen Sicherheitslevels mit sich. Auch durch die Verwendung von Mandatory Access Control hat man keine umfassende Sicherheit. Auch hier ist im Endeffekt ein Mensch - der Administrator - verantwortlich. Wenn dieser bei der Vergabe von Labels Fehler macht, bringt das System unter Umständen nicht mehr die gewünschte Leistung. Zudem gibt es ein systembedingtes Problem. Auf der einen Seite gibt es Programme, welche sich der genutzten Sicherheitspolicy bewusst sind. Auf der anderen Seite stehen die Anwendungen, die nichts von einer zusätzlichen Kontrolle wissen oder wissen sollen und sich somit auch einer Überprüfung auf einen bestimmten Sicherheitskontext hin entziehen können. Dennoch gilt für beide Seiten der gleiche Anspruch - die Kontrolle über Zugriffe auf Ressourcen soll kontrolliert und beschränkt werden. Um nun auch diese zweite „unwissende“ Gruppe von Anwendungen zu typisieren, werden diese Prozesse oftmals einfach als „zuverlässig“ klassifiziert. Ein mögliches Problem könnte dann sein, dass diese Anwendung mit erhaltenen Daten intern nicht richtig umgehen kann. Wissenschaftler der Pennsylvania State Universität haben sich mit dem Problem beschäftigt und eine Architektur entwickelt, mit welcher sicherheitsorientierte Programmiersprachen und Betriebssystem-MAC-Frameworks zusammengeführt werden können. Somit ist die Lücke zwischen Anwendungen, welche nichts von

einer Zugriffskontrolle wissen müssen einerseits und der Policy für die Kontrolle andererseits geschlossen. [HRJM07, LSMT01]

4.4 Allgemeine Entwicklung

Ein weiterer Unterschied zwischen beiden Frameworks ist die grundsätzliche Entwicklungsweise. Beim LSM werden Änderungen relativ zeitnah in den aktuellen Standard Kernel unter Linux eingebaut, damit verbunden sind unter Umständen auch Änderungen an Bereichen, die sonst Kernel Hackern vorbehalten bleiben, die nichts mit dem LSM zu tun haben. Unter FreeBSD wird das Problem anders gelöst. In dem Schwesterprojekt TrustedBSD ist es die Hauptaufgabe, sicherheitsrelevante Teilprojekte zu entwickeln und nach erfolgreicher Fertigstellung wieder in den Hauptzweig von FreeBSD zurück zu portieren. Beide Möglichkeiten haben Vor- und Nachteile, zum Beispiel könnten häufige Änderungen am LSM die Kernel Hacker des Standard Kernels stören oder neue Probleme erzeugen, konstruktive Zusammenarbeit hingegen kann sich auch positiv auswirken. Bei dem MAC Framework ist die schwierige Aufgabe zu meistern, eine Zusammenführung des FreeBSD- und TrustedBSD-Codeanteils durchzuführen. Mein Eindruck ist aber auch hier, als ob FreeBSD die bessere Variante gewählt hat, denn große Streitereien habe ich hier nicht über die Vorgehensweise gefunden, ganz im Gegenteil zum LSM.

5 Zusammenfassung und Ausblick

Mit der Entwicklung der Frameworks für erweiterte Zugriffskontrolle gehen FreeBSD sowie Linux einen wichtigen Schritt in Richtung mehr Sicherheit. Die Umsetzung erfolgt jeweils nach ähnlichen Grundsätzen und versucht, verschiedene Sicherheitsansätze zuzulassen und neue Entwicklungen einzubinden. Ein Beispiel dafür ist Linus Torvalds Aussage, dass nur ein generisches Framework zur Zugriffskontrolle Einzug in den Kernel finden kann, wonach dann eine entsprechende Implementierung erfolgte [WCS+02]. Ein Kernelmodul sorgt bei beiden Frameworks für die Funktionalität, im Einsatz selbst werden dann direkt vor den erlaubten Zugriffen per Kernel Hook die Berechtigungen überprüft. Mit dem speziellen Bezug auf Mandatory Access Control bieten sowohl das LSM als auch das FreeBSD MAC Framework viele Policies, die auf unterschiedliche Einsatzzwecke abzielen.

Im Detail habe ich dennoch den Eindruck gefunden, dass die von TrustedBSD initiierten Sicherheitsentwicklungen für FreeBSD besser durchdacht oder durch die grundsätzliche Systemarchitektur besser unterstützt werden. Hiermit ist im Detail die Möglichkeit gemeint, mehrere Sicherheitspolicies parallel zu laden und zu nutzen. Beim Labeln von Objekten sichert FreeBSD dem MAC Framework eine bessere Unterstützung durch das Dateisystem zu, als es bei Linux momentan der Fall ist. Zudem können unter FreeBSD Labels freier von jeder Policy vergeben werden als mit dem Modell unter Linux, bei welchem nur eine Policy effektiv die Labelvergabe nutzen kann. Verschiedene Entwickler von Policies für das LSM beschweren sich hingegen über mangelhafte Möglichkeiten zur parallele Nutzung von Richtlinien oder haben die Entwicklung ihrer LSM Policies eingestellt. Zusammenfassend scheint die Arbeit am LSM Framework mehr Probleme aufzuwerfen als die bei dem FreeBSD MAC Framework, was vielleicht aber auch an der prinzipiell kleineren Community bei FreeBSD liegt.

Ein Thema für eine zukünftige Arbeit könnte eine ausführliche Performance Analyse der beiden Frameworks sein, speziell auch bei FreeBSD. Hier wurden bisher keine Benchmarks erstellt, die eine Übersicht über Einbußen in der Leistung bringen. Allerdings ist dies nicht zu unterschätzen, da durch die Vielzahl der möglichen Richtlinien in Kombination nicht wenige Testfälle entstehen. Beim LSM ist dies einfacher zu handhaben, da meist nur eine Policy geladen wird. Ein weiteres mögliches Arbeitsfeld ist eine praktische Anwendung der

Frameworks, also die Erstellung von Policies. Dabei kann dann auf Probleme bei der Nutzung der Frameworks, der Einbindung ins System oder auch wieder auf Performancefragen eingegangen werden. Mit einem Fokus auf die generischen Audit Systeme der beiden Betriebssysteme Linux und FreeBSD und einer Verknüpfung hin zu den Zugriffskontrollframeworks könnte die bestehende Arbeit zudem erweitert werden und den Bereich der Validierung und Zertifizierung ob korrekter Funktionalität besser abdecken.

Glossar

Access Control List Vgl. Zugriffskontrollliste

Authentifizierung / Authentifikation Nachweis der Identität einem Kommunikationspartner gegenüber.

Authentisierung Bestätigung der Identität eines Benutzers, diese wird bei der Identitätsprüfung auf IT-Systemen verwendet.

Autorisierung Eine Authorisierung ist eine Erlaubnis für ein Subjekt, bestimmte Tätigkeiten auszuführen.

Bedrohungen Auch Gefährdungen (engl. threat), sind Ereignisse oder Umstände durch welche Schäden entstehen können. Auf die IT-Sicherheit übertragen also der Verlust der Grundwerte (Vertraulichkeit, Verfügbarkeit, Integrität, Zurechenbarkeit). Bedrohungen sind u.a. das Resultat einer oder mehrerer Schwachstellen bzw. durch Schwachstellen wird ein IT-System anfällig für Bedrohungen.

Capability Vgl. Zugriffsausweis

FreeBSD ist ein freies Betriebssystem aus der BSD Familie mit Ursprüngen aus dem Unixbereich. Die Stärken von FreeBSD liegen im Sicherheits- und Netzwerkbereich, daher wird es oft von Hardwareherstellern im Embedded Bereich eingesetzt.

Hook Ein Hook bezeichnet eine Technik, welche verwendet wird, um Funktionsaufrufe oder System Calls in einem bestehenden System abzufangen. In dem hier verwendeten Fall wird ein Hook zur Ergänzung um Sicherheitsfunktionen verwendet, im Detail zur Verbesserung der Zugriffskontrolle.

Least privilege Benutzer und Anwendungen besitzen nur die Rechte, welche gerade so zur Erfüllung ihrer Aufgaben erforderlich ist. [SS75]

Linux auch GNU/Linux genannt. Bezeichnet wird dadurch eine Gruppe von freien, unixartigen Betriebssystemen, welche den Linux Kernel und Software nach der GNU verwenden. Die GNU setzt sich zum Ziel, eine freie Alternative zu Unix zu bilden. Große Teile der Software für Linux, so auch der Kernel, stehen unter der freien Lizenz GPL. Die Entwicklung von Linux wurde durch Linus Torvalds begonnen, er beeinflusst auch heute noch die Fortentwicklung von Linux.

Multi Level Security bezeichnet im ursprünglichen Sinne die MAC, wie sie im militärischen Bereich entwickelt wurde. Dabei werden die Daten in mehrere Stufen unterteilt und sehr strikt nach Integritäts- oder auch Vertraulichkeitsanforderungen getrennt. Dieses exakte Modell wird heute kaum verwendet, verschiedene Weiterentwicklungen bauen aber darauf auf. Verwendung erfolgt heute eher für bestimmte Restriktionen bei Dateizugriffen.

Policy Enforcement Durchsetzung einer Policy, hier speziell für eine Policy, welche die MAC-Regelsätze anwendet.

POSIX Capabilities beschreiben einen Entwurf für Standard, welcher es ermöglicht, die bestehende Macht von root in mehrere Teilbereiche zu untergliedern. Die Möglichkeiten, die dadurch genutzt werden können, sind unter Linux teilweise implementiert. Allgemein ist eine Capability dabei eine Art Ausweis, mit welcher der Zugriff auf ein Objekt erlangt werden kann. Der Ansatz der Capabilities hat sich allerdings bisher nicht durchsetzen können. [Spe07]

Rootkit ist die Bezeichnung für ein Programm, welches in meist schädlicher Absicht verschleiert, dass das System kompromittiert wurde. Dabei können API Calls jeder Art abgefangen und verändert werden, beispielsweise wird der Angreifer verhindern, dass sein Login protokolliert wird. Allerdings gibt es auch gutartige Rootkits, welche das System durch die zusätzliche Abstraktion sicherer machen.

Schutzziele Auch Grundwerte der IT-Sicherheit. Zu ihnen gehört die Vertraulichkeit, Verfügbarkeit, Integrität und Zurechenbarkeit. (siehe 2.1)

Schwachstelle Auch Verwundbarkeit (engl. vulnerability). Sie ist als sicherheitsrelevanter Fehler eines IT-Systems zu verstehen und kann für einen Angriff ausgenutzt werden. Umgangssprachlich spricht man auch von einer Sicherheitslücke.

System Call Ein System Call ist ein Systemaufruf, welcher von der Benutzerebene des Computers Zugriff auf Funktionen des Betriebssystems ermöglicht - wird auch als SysCall bezeichnet.

Type Enforcement ist eine Umsetzung des MAC-Prinzips, dabei werden alle Objekte des Systems mit einem Label versehen. Der Prüfung und Durchsetzung der Regelsätze in diesem Computersystem erfolgt dann nicht über die eigentliche Ressource, sondern über die Typen, die den Objekten anhaften.

Verwundbarkeit Vgl. Schwachstelle.

Zugriffsausweis Möglichkeit der Zugriffskontrolle aus der Sicht des Subjektes, es werden beispielsweise zu einem Prozess die nutzbaren Capabilities zugeordnet.

Zugriffskontrollliste Stellen ein Mittel dar um Zugriffskontrollen durchzuführen. Zu jedem Objekt wird die Menge der nutzbaren Subjekte zugeordnet, es wird also aus der Objektsicht heraus gehandelt.

Abkürzungsverzeichnis

ACL	Access Control List
BSD	Berkeley Software Distribution
CC	Common Criteria
DAC	Discretionary Access Control
EAL	Evaluation Assurance Level
GNU	GNU's Not Unix
GPL	General Public License
IDS	Intrusion Detection System
IPC	Inter-process Communication
ITSEC	Information Technology Security Evaluation Criteria
LSM	Linux Security Module
MAC	Mandatory Access Control
MLS	Multi Level Security
RBAC	Rule-based Access Control
SUID	Set User ID
TCSEC	Trusted Computer System Evaluation Criteria
TE	Type Enforcement
TNI	Trusted Network Interpretation
VFS	Virtual File System

Literaturverzeichnis

[AB09] AGREITER, B.; BREU, R.: Model-Driven Configuration of SELinux Policies. In: *On the Move to Meaningful Internet Systems: OTM 2009*, Springer Berlin / Heidelberg, 2009

[BBSS07] BAEK, K.-H.; BRATUS, S.; SINCLAIR, S.; SMITH, S. W.: Attacking and Defending Networked Embedded Devices. (2007)

[BTS09] BLAICH, A.; THAIN, D.; STRIEGEL, A.: Reflections on the virtues of modularity: a case study in linux security modules. In: *Software: Practice and Experience* (2009). – ISSN 1097–024X

[CCE06] CCEVS: CC/CEM Documentation / CC. 2006. – Forschungsbericht

[Cen83] CENTER, National Computer S.: 5200.28-STD Trusted Computer System Evaluation Criteria / National Security Institute. 1983. – Forschungsbericht

[Cow03] COWAN, C.: Software Security for Open-Source Systems. In: *IEEE Security and Privacy* (2003)

[CWD02] CHEN, H.; WAGNER, D.; DEAN, D.: Setuid Demystified. In: *Proceedings of the 11th USENIX Security Symposium*. Berkeley, CA, USA : USENIX Association, 2002

[daz10] *Dazuko FAQ*. zuletzt gesichtet: 7.1.2010. – `http://dazuko.org/tgen.shtml#LSM`

[DCN02] DALTON, C. I.; CHOO, T. H.; NORMAN, A. P.: Design of Secure Unix / Hewlett-Packard Laboratories. 2002. – Forschungsbericht

[Eck06] ECKERT, C.: *IT-Sicherheit*. Oldenbourg Verlag München Wien, 2006. – ISBN 3–486–57851–0

[Edg07] EDGE, J.: Smack for simplified access control. In: *LWN.net* `http://lwn.net/Articles/244531/`, *zuletzt gesichtet 25.1.2010* (2007)

[FK92] FERRAIOLO, D.; KUHN, R.: Role-Based Access Control. In: *Proceedings of 15th National Computer Security Conference*. Gaithersburg, Maryland, US, 1992

[FKC03] FERRAIOLO, D.; KUHN, R.; CHANDRAMOULI, R.: *Role-Based Access Control*. Artech House Publishers, April 2003. – ISBN 1–580–53370–1

[fla10] *Flask: Flux Advanced Security Kernel*. zuletzt gesichtet: 20.1.2010. – http://www.cs.utah.edu/flux/fluke/html/flask.html

[fli10] *Fig.2 Discretionary and mandatory access control diagrams*. zuletzt gesichtet: 1.3.2010. – http://www.flickr.com/photos/redhatmagazine/481929076/

[GJJ05] GANAPATHY, V.; JAEGER, T.; JHA, S.: Automatic Placement of Authorization Hooks in the Linux Security Modules Framework. In: *Proceedings of the 12th ACM Conference on Computer and Communications Security*. Alexandria, Virginia, USA, 2005

[grs10] *grsecurity and LSM*. zuletzt gesichtet: 7.1.2010. – http://grsecurity.net/lsm.php

[HK04] HALYN, S.; KEARNS, P.: Modular Construction of DTE Policies. In: *Proceedings of the annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA : USENIX Association, 2004

[HRJM07] HICKS, B.; RUEDA, S.; JAEGER, T.; MCDANIEL, P.: From Trusted to Secure: Building and Executing Applications that Enforce System Security. In: *USENIX Annual Technical Conference*, 2007

[JEZ04] JAEGER, T.; EDWARDS, A.; ZHANG, X.: Consistency Analysis of Authorization Hook Placement in the Linux Security Modules Framework. In: *ACM Trans. Inf. Syst. Secur.* 7 (2004), Nr. 2, S. 175–205

[Ker95] KERSTEN, H.: *Sicherheit in der Informationstechnik*. Oldenbourg Verlag München Wien, 1995. – ISBN 3–486–23179–0

[Kle08] KLEIN, G.: Operating System Verification — An Overview / NICTA. Sydney, Australia, jun 2008 (NRL-955). – Forschungsbericht

[lin10] *Linux Kernel vfs_mkdir*. zuletzt gesichtet: 20.1.2010. – http://lxr.linux.no/linux+*/fs/namei.c#L2086

[LRS06] LEHTINEN, R.; RUSSELL, D.; SR., G. T. G.: *Computer Security Basics*. O'Reilly Media, 2006. – ISBN 0–596–00669–1

[LSMT01] LOSCOCCO, P. A.; SMALLEY, S. D.; MUCKELBAUER, P. A.; TAYLOR, R. C.: The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In: *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*. Berkeley, CA, USA, 2001. – ISBN 1–880446–10–3

[MG08] MURRAY, A. P.; GROVE, D. A.: PULSE: a Pluggable User-space Linux Security Environment. In: *Australasian Information Security Conference (AISC2008)*. Wollongong, Australia, 2008

[Mor08] MORRIS, J.: System Services and Smack. (2008). – http://www.schaufler-ca.com/data/SmackServices080616.pdf zuletzt gesichtet: 25.1.2010

[Mor09] MORRIS, J.: Linux Kernel Security Overview. In: *Kernel Conference Australia* (2009)

[Mor10] MORRIS, J.: The Simplified Mandatory Access Control Kernel. In: *linux.conf.au* (2010)

[net10] *Netfilter Project*. zuletzt gesichtet: 20.1.2010. – `http://www.netfilter.org/`

[PA08] PAPACHRISTODOULOU, L.; ANTAKIS, S.: Security-Enhanced Linux in a Health Information System / Eindhoven University of Technology. 2008. – Forschungsbericht

[Per10] PERON, C.: *Kernel module to deny execution of unsigned binaries?* zuletzt gesichtet: 23.2.2010. – `http://lists.freebsd.org/pipermail/trustedbsd-discuss/2006-August/000867.html`

[rsb10] *RSBAC and LSM*. zuletzt gesichtet: 7.1.2010. – `http://www.rsbac.org/documentation/why_rsbac_does_not_use_lsm`

[SC07] SIHAN, Q.; CHANGXIANG, S.: Design of secure operating systems with high security levels. In: *Science in China Press* (2007)

[Spe06] SPENNEBERG, R.: Allgemeine Schutzimpfung. (2006). – `http://www.linux-magazin.de/Heft-Abo/Ausgaben/2006/06/Allgemeine-Schutzimpfung`, zuletzt gesichtet 23.2.2010

[Spe07] SPENNEBERG, R.: *SELinux & AppArmor*. Addison-Wesley, München, 10. Oktober 2007. – ISBN 3–827–32363–0

[SS75] SALTZER, J. H.; SCHROEDER, M. D.: The protection of information in computer systems. In: *Proceedings of the IEEE* 63 (1975), June, Nr. 9, S. 1278–1308

[SVS06] SMALLEY, S.; VANCE, C.; SALAMON, W.: Implementing SELinux as a Linux Security Module / NSA. 2006. – Forschungsbericht

[tux10] *TuxGuardian Homepage*. zuletzt gesichtet: 25.1.2010. – `http://tuxguardian.sourceforge.net/`

[VW03] VANCE, C.; WATSON, R.: Security Enhanced BSD / Network Associates Laboratories. 2003. – Forschungsbericht

[Wat04] WATSON, R.: TrustedBSD: Trusted Operating System Features for BSD / Network Associates. 2004. – Forschungsbericht

[WCS+02] WRIGHT, C.; COWAN, C.; SMALLEY, S.; MORRIS, J.; KROAH-HARTMAN, G.: Linux Security Modules: General Security Support for the Linux Kernel. In: *Proceedings of the 11th USENIX Security Symposium*. Berkeley, CA, USA : USENIX Association, 2002. – ISBN 1–931971–00–5, S. 17–31

[WFMV03] WATSON, R.; FELDMAN, B.; MIGUS, A.; VANCE, C.: Design and Implementation of the TrustedBSD MAC Framework. In: *DARPA Information Survivability Conference and Exposition (DISCEX III)*. Washington, DC, USA : IEEE, 2003

[WWS07] WILSON, G.; WEIDNER, K.; SALEM, L.: Extending Linux for Multi-Level Security. (2007)

Erklärung

„Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann“.

Ort	Datum	Unterschrift