Zugriffsrechte unter Linux und UNIX

Referat von
Cathrin Firchau
Hendrik Huch

1. Grundlagen

Das Problem der Zugriffskontrolle:

Welche Objekte sind einem Subjekt zugänglich und in welcher Form?

- Objekte: Ressourcen wie etwa Dateien, Geräte, Netzwerke, Pipes, andere Prozesse,...
- Subjekte: Programme bzw. Benutzer
- Zugang: Art der Operation die auf die Objekte angewendet werden kann
- Problem: Nicht alles lässt sich auf Dateizugriffe abbilden
- Beispiele:
  - Gerätetreiber oder Kernmodule
  - Direkter Hardware-Zugriff
  - TCP/UDP-Ports rohe IP-Pakete senden
  - Netzwerkkonfiguration
  - Scheduling
  - Signale an fremde Prozesse schicken
  - Zugriffe auf Pipes, Shared Memory, ...

Die konventionelle Lösung: Superuser

Diese Lösung nutzt den klassischen Unix-Ansatz, also Zugriffsrechte im Dateisystem für User, Group und Other, und alle restlichen Zugriffe lauten „root-sein-oder-nicht-sein". Doch die Einteilung des Dateisystems ist grundsätzlich ziemlich grob und viele Programme müssen mit Root-Rechten gestartet werden (setuid / System Privilege Table).

- Setuid: das Programm wird mit Root-Rechten ausgeführt
  => Problem: Sicherheitslücke ermöglicht Ausführung von beliebigem Code mit Root-Rechten
- System Privilege Table: Liste mit Programmen, die spezielle Rechte haben müssen
  => Problem: Sicherheitslücke oder Austausch der Datei ermöglichen ebenfalls Ausführung mit Root-Rechten

Die Allmächtigkeit von root geht soweit, dass es alles darf, also Verzeichnisse ein-/aushängen, Dateien ändern, löschen, ..., Programme ausführen, Dienste an-/ausschalten, Benutzer, anlegen/löschen, usw.

Das Problem der Superuser-Rechte ist, dass Angreifer, die Superuser-Rechte erlangen, das System nach belieben ändern können, also Angriffe mittels Root-Kits, ..., Missbrauch des „gehackten" Systems (DOS, ...), usw.

Die Lösung wäre die Zugriffseinschränkung für root. Doch wie kann man root einschränken?
Lösung: Durch den Kernel! Der Kernel ist der Kern des Betriebssystems. Zu seinen Aufgaben gehören:

- Zugriffskontrolle über Prozesse
- Zugriff auf Register und Speicherbereiche
- Bereitstellung von System-Calls
- Zugriffskontrolle über Peripherie
- usw.

1.1 Arten von Zugriffskontrollmodellen (Allgemeine Sicherheitskonzepte)

Bevor man sich für eine konkrete, vernünftige und stabile bzw. solide Systemlösung entscheidet (im Weiteren „Gehärtetes System" genannt), muss man die verschiedenen Sicherheitskonzepte kennen, um zu beurteilen, welches Modell die gewünschten

Anforderungen erfüllt.

- Keine Zugriffsbeschränkung
  Viele Systeme haben keine Zugriffsbeschränkungen, weil entweder alle Benutzer alles dürfen oder die Applikation implementiert selbst ein Sicherheitskonzept. Dieses Konzept findet man häufig in Embedded Systemen oder Stand Alone Geräten.

- UNIX Sicherheit
  Unix ist ein Multiusersystem und implementiert ein einfaches und rudimentäres Sicherheitskonzept. Es existieren unter Unix zwei Arten von Benutzer, entweder man ist "root" und darf alles oder man ist ein normaler Benutzer und darf nur das, was explizit von root erlaubt wird. Jeder Benutzer kann unter Unix selbst festlegen, ob seine Dateien privat oder für seine Gruppe verfügbar sind oder für alle öffentlich sind. Dieses Konzept für das Dateisystem ist ein eingeschränktes Discrete Access Control.

- Linux Capabilities
  Das Konzept, dass root alles darf, ist für ein modernes Sicherheitskonzept nicht geeignet, weil es zu wenig die verschiedenen Rechte differenziert. Deshalb gibt es bei POSIX (Standard) eine Aufteilung der root Rechte in verschiedene Capabilities. root besitzt normalerweise immer alle Capabilities. Jeder Prozess mit root-Rechten kann aber einzelne Rechte mit dem Funktionsaufruf cap_set_proc, wenn er diese Rechte nicht benötigt, abgeben. Zusätzlich zu den POSIX Capabilities definiert Linux weitere Linux-spezifische. Unter Linux werden alle Capabilities in der Datei /usr/include/linux/capability.h definiert und erklärt.

- Discrete Access Control
  Discrete Access Control ist ein Sicherheitskonzept, in der Zugang zu allen Objekten (Prozesse, Dateien und Ressourcen) mit einer Access Control List kontrolliert werden kann. Zugangsrechte über die Objekte entscheiden nur der Ersteller oder Besitzer dieser Objekte. Dieses Sicherheitskonzept ermöglicht eine sehr gute Kontrolle über alle Objekte. Der Nachteil ist, dass jeder Benutzer die Rechte seiner Dateien selber setzen muss. Der Administrator kann die allgemeine Systemsicherheit nur dadurch erhöhen, in dem er den Benutzer nur minimale Rechte gibt, so dass der maximale Schaden durch falsch gesetzte Rechte der Benutzer minimal gehalten werden kann.

- Mandatory Access Control
  Der Unterschied zwischen Discrete Access Control und Mandatory Access Control ist die Möglichkeit des Administrators, die vollständige Kontrolle des Administrators über alle Objekte zu haben. Diese Kontrolle erhält der Administrator durch die Erstellung einer Sicherheitspolicy. Diese Policy bestimmt dann drch ein Programm unter anderem die Rechte aller neu erzeugten Objekte. Des Weiteren werden alle Aktionen anhand der Policy überprüft.

- Type Enforcement
  Im Type Enforcement Modell (TE-Modell) gibt es Domänen und Typen. Jedem Prozess wird einer Domäne zugeordnet und jedem Objekt ein Typ. Die Verknüpfung zwischen Typen und Domänen ergibt die Access Matrix, in der die Rechte für jede Domäne zu einem Typ beschrieben wird. Jedes Programm ist ein Typ und falls dieser Typ in der aktuellen Domäne enthalten ist, dann ist die Ausführung dieses Programms in der Domäne erlaubt. Für den Wechsel, auch Transition genannt, werden Eintrittspunkte für jede Domäne als zusätzlicher Typ definiert.

- Role-based Access Control
  Das Role-based Access Control Model (RBAC-Modell) erlaubt den Anwendern in verschieden Rollen zu arbeiten. Mit jeder Rolle ist eine Menge von Rechten verknüpft, so dass eine einfache Benutzerverwaltung möglich ist. Die Rollen entsprechen häufig dem Aufgabenbereich des Benutzers. Wie die Rechte aussehen wird im RBAC-Modell

nicht erklärt/nicht genau definiert.

- Flask Security Architecture
  Das Information Assurance Research Office der NSA und die Firma Secure Computing Corporation (SCC) entwickelten zusammen ein Mandatory Access Control auf Basis von Type Enforcement für das LOCK System. Später ist dies in zwei Prototypen für das Mach-System dtMach und dtOS umgesetzt worden. Dieses System ist dann mit der Flux Research Group der Universität von Utah zur Flask Architektur weiterentwickelt worden. Diese Architektur ist von der NSA nach Linux unter dem Namen SELinux portiert worden, um es einer größeren Zielgruppe zur Verfügung zu stellen.
  Auch das Flask Konzept unterscheidet zwischen Subjekten (hier Prozesse) und Objekten (hier Ressourcen), ähnlich wie das Type Enforcement Modell. Jedes Subjekt und jedes Objekt besitzt einen eigenen Security Context, in der die jeweilige Security Policy definiert ist. Für bessere Effizienz und höhere Abstraktion wird aber jedem Subjekt und jedem Objekt nur ein SID (Security Identifier) zugewiesen. Diese SID ist nur ein Integer, der durch den Security Server zu einem Security Context übersetzt wird. Die SID ist nicht-persistent und nur lokal definiert. Für eine Sicherheitsentscheidung bekommt der Security Server zwei SIDs. Diese SIDs werden zu den jeweiligen Security Contexts aufgelöst. Der Security Server bietet zwei Arten von Entscheidungen an:

  - Labeling Decision
    Die Etikettierung (Labeling Decision) oder auch Transitionsentscheidungen definiert den Security Context für neue Subjekte oder neue Objekte. Die Prozess Transition erfolgt, wenn ein Prozess ein anderen Prozess startet. Der Security Context des Vaterprozesses bestimmt den neuen Security Context des Kindprozess mit. Die Objekt Transition ist das Erstellen von neuen Objekten, wie z.B. neuen Dateien. Der Security Context des neuen Objekts ist abhängig von dem erzeugenden Prozess und verwandte Objekte, wie z.B. das Verzeichnis in der eine Datei erzeugt wird. Jeder dieser Transitionen muss durch eine Zugangsentscheidung entschieden werden.

  - Access Decision
    Die Zugangsentscheidung (Access Decision) liefert aus zwei SIDs und einer Klasse ein Zugangsvektor (Access Vector). Jede Objektklasse besitzt eine Menge von zugehörigen Rechten, um die Operationen auf diesem Objekt zu kontrollieren. Die Menge der Rechte wird in einem Zugangsvektor bitweise dargestellt. Der zurückgelieferte Zugangsvektor beschreibt alle erlaubten Rechte. Alle Zugangsentscheidungen werden in einem Cache (Access Vector Cache, kurz AVC) für bessere Performance abgelegt. Zusätzlich zu dem Zugangsvektor mit erlaubten Rechten bietet der Security Server zwei Zugangsvektoren für das Audit an. Jeweils einen Zugangsvektor für das Audit, wenn eine Aktion erlaubt wird, und eine, wenn eine verboten wird.
    Das Flask Konzept kann dazu verwendet werden, um z.B. das Type Enforcement und das Role-based Access Control in einem gehärteten System umzusetzen.

- Sonstige Konzepte

  - Verstecken von Informationen
    Viele gehärtete Systeme bieten Optionen an, um diverse Informationen zu verstecken. Dadurch kann ein Angriff auf das System erschwert werden, weil notwendige Informationen nicht sichtbar sind. Manche Informationen werden dadurch versteckt, in dem sie nicht vorhersehbar, sondern zufällig generiert werden, so dass ein Angreifer dies nicht im Voraus wissen kann. Beispielsweise kann man die PID Erzeugung zufällig gestalten.

  - Abschalten von Fähigkeiten
    Ein modernes Betriebssystem bietet vielfältige Möglichkeiten, die bei einem

gehärteten System oft nicht gebraucht werden. Nicht gebrauchte Fähigkeiten kann man deshalb oft deaktivieren, so dass bestimmte Arten von Angriffen, die diese Fähigkeit benötigen, nicht funktionieren. Beispielsweise erfolgen viele bekannte Angriffe durch einen Stack-Overflow und um solche Angriffe zu erschweren, kann man das Ausführen von Programmen auf dem Stack allgemein verbieten. So kann der Angreifer keinen eigenen Code im Stack ausführen. Dies ist aber nicht auf allen Prozessorarchitekturen möglich und ist somit nicht portabel. Auch führen manche Programme eigenen Code auf dem Stack aus, so dass solche Programme nicht mehr funktionieren.

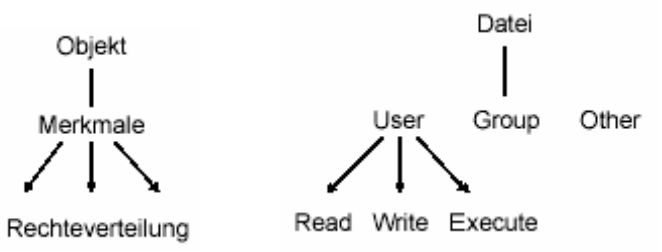
- Virtuelle Systeme

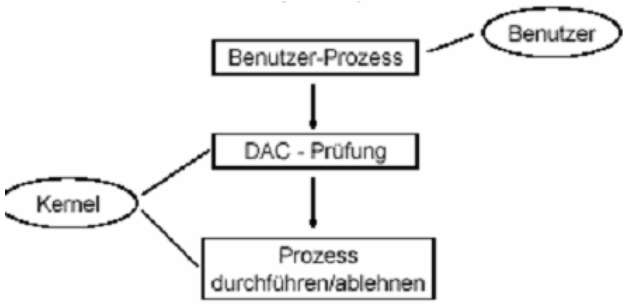
  Um das Betriebssystem noch sicherer zu machen, kann man jeden Dienst in einem virtuellen System starten, so dass sich die Dienste nicht gegenseitig stören. Das Ziel ist es, viele Dienste jeweils im eigenen virtuellen System auf einem Server laufen zu lassen. Beispiele dafür sind chroot, jail oder virtuelle Betriebssysteme. Virtuelles Betriebssystem bedeutet, dass man innerhalb eines laufenden Betriebssystems (dem so genannten Host-System) ein anderes Betriebssystem (oft als Gast-System bezeichnet) startet. Ein bekanntes Beispiel für virtuelle Betriebssysteme ist VMWare.

1.2 Discretionary access control (DAC)

Bei der klassischen Zugriffskontrolle in Linux hat der Besitzer einer Datei die volle Verfügungsgewalt über sein Objekt. Damit hängt die Sicherheit der enthaltenen Daten nur von der Diskretion (dem Ermessen) des Besitzers ab, man spricht deshalb von diskreter Zugriffskontrolle.
D.h. DAC bedient sich der normalen Dateizugriffsrechte bzw. auf Wunsch auch ACLs, um den Zugriff auf Daten bzw. Informationen in Abhängigkeit der Benutzeridentität bzw. Gruppenmitgliedschaft zu beschränken. Der Root ist nicht von diesen Beschränkungen ausgenommen. DAC wird zusammen mit MAC für die Kontrolle des Dateisystems verwendet.

-> Zugriffseinschränkung über Benutzer und Gruppen:



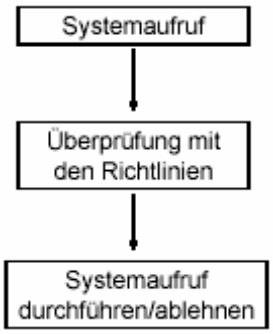


1.3 Mandatory access control (MAC)

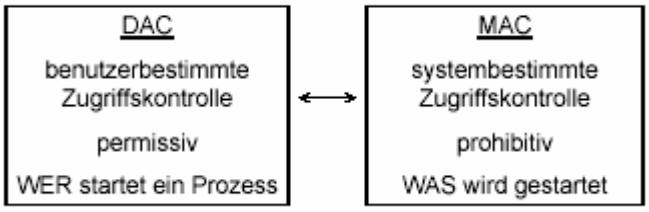
Im Gegensatz zum DAC verlangt eine mandatorische Zugriffskontrolle, dass die Erzeuger und Bearbeiter eines Objekts keinen direkten Einfluss auf dessen Zugriffsrechte haben. Stattdessen entscheiden das verwendete Modell und der Sicherheitsadministrator über alle Sicherheitseinstellungen.

Hierbei handelt es sich um ein System mit dessen Hilfe Daten mit „Labels“ in verschiedene hierarchische Sicherheitsstufen wie „public“, „private“, „private-engeneering“ eingeteilt werden können. Diese Einteilungen können von normalen Benutzern nicht geändert werden und somit wird dafür gesorgt, dass Informationen nicht „unabsichtig“ veröffentlicht oder ausgetauscht werden können.

-> Systembestimmte Zugriffskontrolle durch Richtlinien (so genannte Policy), die während der Laufzeit nicht änderbar sind:



1.4 Vergleich DAC - MAC



permissiv: Freizügigkeit gewährend prohibitiv: verbietend, verhindernd

2. Unix Rechte

Für das Funktionieren eines Mehrbenutzer Systems ist das Vorhandensein verschiedener "Benutzer" mit verschieden Privilegien notwendig. Ein normaler Benutzer kann ein Linux System z.B. nicht einfach herunterfahren oder die gesamte Festplatte löschen. Abgesehen vom Recht, etwas tun zu dürfen, gibt es auch Eigentumsrechte. Jeder Benutzer kann mehreren Gruppen angehören. Gleichzeitig hat jede Datei einen Benutzer und eine Gruppe als Eigentümer. Man muss zudem Unterscheiden zwischen Subjekten (Menschen, Prozessen ....), Objekten (Verzeichnissen, Dateien,....) und Rechten(Schreiben, Lesen,....).
Diese werden in einer sogenannten Zugriffsmatrix gespeichert, die angibt welche Rechte ein Subjekt an einem Objekt hat.
z.B.

	Objekte				
Subjekte	p3	s1	O2	D3	M4	B3
	S1	r, m	y, d	s	p, g	w
	S2		y, r	s, r, m	p	

2.1. root

root ist der wohl wichtigste Benutzer des Systems. Er darf buchstäblich alles. Deswegen ist er auch der Administrator des Systems, auch wenn für bestimmte Bereiche vielleicht andere Benutzer zuständig sind (z.B. Webmaster). Root kann z.B.: Alle Dateien anschauen, auch solche die ihm nicht gehören. Ihre Emails lesen, sofern sie noch nicht abgeholt wurden. Überprüfen, an wen Sie Emails geschickt haben. Ihr Passwort ändern. Ihren Login sperren und noch viele andere unheimliche Sachen.

2.2. Zugriffsrechte und Eigentumsrechte

Die Zugriffsrechte und Eigentümer einer Datei können mit dem Befehl ls -ls angezeigt werden. Dies soll an folgendem Beispiel erläutert werden (Ausschnitt):

```
drwx------   2 root     root         1024 Dec 27 13:16 conf/
drwxr-x---   5 niels    weber         1024 Dec 27 17:29 data/
drwxr-x---   2 niels    weber         1024 Dec 21 21:38 images/
-rw-r-----   3 niels    weber         1024 Dec 23 00:34 interna
drwxr-xr-x   2 root     root          1024 Nov 25 18:35 mnt/
drwxr-x---  66 wwwrun   weber         6144 Jan  5 04:02 proxy_cache/
drwxr-x---   5 root     root         2048 Jun  6  1997 sbin/
drwxr-x---   2 root     root         1024 Jun  6  1997 support/
```

Die erste Spalte stellt die Zugriffsrechte und bestimmte Statusinformationen dar. d Steht für Verzeichnis, r für lesen, w für schreiben und x für ausführbar. (Verzeichnisse müssen als ausführbar deklariert werden, damit der Benutzer darauf zugreifen kann.)
Von der ersten Stelle abgesehen ist diese Darstellung in Dreiergruppen gegliedert:
d rwx r-x --- 5 niels weber 1024 Dec 27 17:29 data/
Das erste Segment, rwx, zeigt die Zugriffsrechte für den Eigentümer einer Datei bzw. hier den des Verzeichnisses an. Das nächste Segment, r-x, stellt die Rechte für die Gruppe dar. Die übrigbleibende Zeichenkette --- bestimmt die Rechte für "den Rest der Welt", also Benutzer, die keine Eigentümer der Datei sind. (Aus Gründen der Übersichtlichkeit wurden zusätzliche Leerzeichen eingefügt)
In der zweiten und dritten Spalte sind die Eigentümer der Datei aufgeführt:
- rw- r-- --- 3 niels weber 1024 Dec 23 00:34 interna
Diese Datei gehört dem Benutzer niels und der Gruppe weber. Niels darf (rw) lesend und schreibend darauf zugreifen während die Gruppe weber nur lesenden Zugriff darauf hat. Der "Rest der Welt" darf auf diese Datei in keiner Art und Weise zugreifen.

2.3. Zugriffsrechte ändern

Mit chmod (Change Mode) ändern Sie die Zugriffsrechte auf eine oder mehrere Dateien. Sie können Rechte entweder wegnehmen (-) oder hinzufügen (+). Zusätzlich müssen Sie angeben, ob dies für die Gruppe (g) den Eigentümer (u) oder den Rest der Welt (o) geschehen soll. Mehrere Angaben können Sie durch Komata abtrennen.
Beispiele:
Die Datei geheim soll nur noch für Sie zugänglich sein. Also nehmen Sie der Gruppe und allen anderen die Rechte.
mueller@mein-pc:/home/mueller > chmod g-rwx,o-rwx geheim
Sie wollen die Datei projekt mit all Ihren Kollegen gemeinsam bearbeiten. Also geben Sie der Gruppe Schreibrecht.
mueller@mein-pc:/home/mueller > chmod g+w projekt

2.4. Eigentümer ändern

chown (Change Owner) teilt einer Datei oder mehreren Dateien einen neuen Eigentümer und eine neue Gruppe zu. Dies funktioniert allerdings nur, wenn Sie als root eingeloggt sind. Die Option -R (Recursive) schließt evtl. vorhandene Unterverzeichnisse und die darin enthaltenen Dateien ein.
Beispiel:
Dieser Befehl weist das Verzeichnis /home/mueller und alle darin enthaltenen Dateien dem Benutzer mueller zu.
mein-pc:/root # chown mueller.users /home/mueller –R

3. ACL

3.1. Was sind ACLs?

Um ACLs zu nutzen braucht man ein erweitertes Dateisystem. Hierbei beziehe ich mich auf AFS.

3.2. AFS - Andrew/Advanced File System

AFS ist ein verteiltes Dateisystem, bei dessen Entwurf besondere Aufmerksamkeit auf effizienten Datenzugriff sowohl in lokalen als auch weitverteilten Netzen gelegt wurde.
AFS wurde ursprünglich im Information Technology Center an der Carnegie-Mellon University entwickelt und später von der Firma Transarc vermarktet und weiterentwickelt. Nach dem Kauf von Transarc durch IBM wurde es unter der Bezeichnung OpenAFS als OpenSource freigegeben. AFS ist als Dateisystem in vielen Großforschungsinstitutionen im Einsatz (Argonne National Laboratory, CERN, Jet Propulsion Laboratory, MIT, ...)

Im Gegensatz zu UNIX werden bei AFS die Zugriffsrechte mit ACLs auf Verzeichnis-, nicht auf Dateiebene vergeben. Das heißt jedes Verzeichnis im AFS hat eine ACL, welche die Zugriffsrechte für dieses Verzeichnis und alle darin enthaltenen Dateien bestimmt. Eine ACL kann bis zu zwanzig Einträge enthalten. Beim Erstellen von Unterverzeichnissen wird die ACL des übergeordneten Verzeichnis vererbt.
Es gibt sieben ACL Attribute, die man in zwei Gruppen einteilen kann:
Attribute, die für das Verzeichnis selbst gelten:

l, : Der Benutzer darf das Verzeichnis lesen, zum Beispiel den ls Befehl ausführen oder sich die ACL des Verzeichnisses anzeigen lassen. Das Lesen von Dateien im Verzeichnis bzw. das Lesen von Unterverzeichnissen wird dadurch nicht erlaubt. Zum Lesen von Unterverzeichnissen werden für jedes Unterverzeichnis jeweils wieder lookup-Rechte benötigt.

i, --insert: Der Benutzer darf neue Dateien in das Verzeichnis schreiben bzw. kopieren. Verändern darf er diese ohne das w-Recht nicht mehr. Außerdem darf er Unterverzeichnisse anlegen.

d, --delete: Der Benutzer darf Dateien aus dem Verzeichnis löschen bzw. verschieben.

a, --administer: Der Benutzer darf die ACL dieses Verzeichnisses verändern.

Attribute die für Dateien innerhalb des Verzeichnisses gelten:

r, --read: Der Benutzer darf den Inhalt der Datei lesen bzw. kopieren und sich mit 'ls -l' die Dateiattribute anzeigen lassen.

w, --write: Der Benutzer darf Dateien verändern und die Attribute der Datei mit chmod verändern. Die Datei muss dabei bereits existieren.

k, --lock: Der Benutzer darf ein lock auf Dateien in diesem Verzeichnis erlangen.

3.3. Wie kann ich ACLs auslesen bzw. verändern?

Zur allgemeinen Verwirrung werden mit dem 'ls' Kommando nur die Unix-Rechte angezeigt, obwohl diese wie oben beschrieben nur mehr beschränkte Gültigkeit haben. Zur Anzeige der ACL eines Verzeichnisses dient das Kommando:

```
fs listacl <dirname>
```

Eine typische Ausgabe ist z.B:

```
Access list for /home/csaa/csaa5698 is
Normal rights:
system:administrators rlidwka
system:anyuser rl
csaa5698 rlidwka
```

An diesem Verzeichnis haben die Gruppe system:administrators und der Benutzer csaa5698 alle Rechte, während die Gruppe system:anyuser (vergleichbar mit der Gruppe 'others' unter Unix) die Rechte read und lookup hat. Hier wird ersichtlich, dass im AFS ACL's für Benutzergruppen und für einzelne Benutzer vergeben werden können. Da jede ACL nur maximal 20 Einträge enthalten kann, gibt es für jeden Benutzer die Möglichkeit, eigene Gruppen zu definieren, die er für ACLs verwenden kann.
Zum verändern der ACL eines Verzeichnisses in AFS dient das Kommando:

```
fs setacl <dirname> <user bzw. group> <acl>
```

Dabei ist <dirname> der Name des Verzeichnisses in AFS, dessen ACLs verändert werden soll, <user> bzw. <group> ist der Benutzername bzw. Gruppenname für den die Rechte vergeben werden und <acl> ist eine Buchstabenkombination für die ACLs. Dabei verwendet man 'rlidwka' oder 'all' für alle Rechte, 'rl' oder 'read' für Leserechte auf das Verzeichnis, 'rlidwk' oder 'write' für die Schreibrechte auf das Verzeichnis (es fehlt dabei das Recht, die ACL zu verändern) usw. Mit 'none' für <acl> wird der entsprechende Benutzer bzw. Gruppe aus der ACL für das Verzeichnis gelöscht.
Achten Sie darauf, dass Sie sich nicht selbst die Rechte nehmen dürfen, da Sie in diesem Fall wirklich keinen Zugriff mehr haben, sich die Rechte auch danach nicht mehr geben können!
Hilfe zum Befehl fs erhält man mit fs help bzw. mit fs help <command>
Wird ein Unterverzeichnis im AFS erstellt, erbt das Unterverzeichnis die ACL des übergeordeten Verzeichnisses.
Der Befehl fs enthält keine Option für rekursives Verändern von ACL's. Dies kann mit einem einfachen 'find' Befehl durchgeführt werden:

```
find <dirname> -type d -exec fs setacl {} <user> <acl> \;
```

3.4. Wie kann ich eigene Gruppen für ACL's erstellen?

Wie bereits erwähnt kann eine ACL maximal 20 Einträge umfassen. Deshalb ist es unumgänglich, mit Gruppen zu arbeiten, wenn vielen Benutzern Rechte am betreffenden Verzeichnis eingeräumt werden sollen. Mit dem Befehl:
pts creategroup <groupname>
kann jeder Benutzer eigene Gruppen anlegen. Mit dem Befehl:
pts adduser -user <username> -group <groupname>
können dieser Gruppe die Benutzer hinzugefügt werden. Dabei muss der Gruppenname dem Format groupowner:groupname genügen (z.B.) csaa5698:Testgruppe.

Mit dem Befehl:
pts membership <groupname>
werden die Mitglieder einer Gruppe ausgegeben. Eigene Gruppen löscht man mit:
pts delete <groupname>
Hilfe zum Befehl pts erhält man mit pts help bzw. mit pts help <command>

4. Capabilities

Unter Capabilities versteht man zeilenweise gespeicherte Rechte die man auch Zugriffstickets nennt. Zu sehen wie eine dünnbesetzte Zugrissmatrix. Sie sind unverfälschbar. Der Besitz eines Objektes (Verzeichnis, Datei) berechtigt zur Wahrnehmung der Rechte.

ObjektUID	Rechte Bits

Jedes Objekt hat seine eigene Capabilitie Liste in der alle Rechte eingetragen sind.
Beispiel:
CList(Joe)= ((Datei 1, { r,x g), (Datei 2, { w g))
CList(Hans)=((Datei2, {w,g),(Datei 1, (w,g))
Capabilities werden zum Beispiel benutzt von:
- IBM System/38
- INTEL iAPX 432
- Mach (Ports = Caps)
- Amoeba
- Handles im Linux-Kern
-

Vorteile von Capabilities sind in Einem die einfach Rechte-Bestimmung und zum anderen die einfach Zugriffskontrolle nur über die Ticketkontrolle. Daraus ergeben sich aber die Nachteile, dass die Rücknahme von Rechten schwierig ist.

Capabilities spalten die Rechte von root auf. Posix definiert unter anderem folgende Capabilities:
- CAP_CHOWN erlaubt einem Prozess, owner und group einer beliebigen Datei zu verändern
- CAP_DAC_OVERRIDE erlaubt einem Prozess sich über Zugriffsrechte auf Dateien hinwegzusetzen
- CAP_SETUID erlaubt einem Prozess sein uid zu ändern
- CAP_KILL erlaubt einem beliebigen Prozess ein Signal zu senden

Linux definiert weitere Capabilities:
- CAP_NET_BIND_SERVICE erlaubt einem Prozess das Binden an Ports 1-1024
- CAP_NET_RAW erlaubt einem Prozess "rohe" Pakete zu senden und zu empfangen
- CAP_SYS_NICE erlaubt einem Prozess die Priorität eines beliebigen Prozesses zu erhöhen und/oder Real-Time Scheduling

5. Role-Based Access Control (RBAC)

Eines der meisten Wettbewerbsprobleme in der Verwaltung großer Netzwerke liegt in der Komplexität der Sicherheitsadministration. Role-Based Access Control (RBAC), 1992 eingeführt durch Ferraiolo und Kuhn, wurde das Hauptmodell für die fortgeschrittene Zugriffskontrolle, weil es die Komplexität und die Kosten der Sicherheitsadministration in großen Netzwerkapplikationen reduziert. Die meisten IT-Vertriebe haben RBAC in ihre Produktlinie hineingenommen, und die Technologie findet Applikationen von der Gesundheitsversorgung bis zur Verteidigung, als Erweiterung zu allgemein akzeptierten

Handelssystemen, für die es ursprünglich gemacht wurde. RBAC ist seit dem 19. Februar 2004 ein Amerikanischer Standard – ANSI INCITS 359-2004.

Dabei werden die administrativen Teile auf mehrere Administratoren aufgeteilt, d.h. es gibt keinen allmächtigen Administrator mehr, sondern mehrere sich ergänzende Administratoren. Bevor die Administratoren mit der Arbeit beginnen können, müssen sie sich zunächst wie normale Benutzer im System einloggen und anschließend eine administrative Rolle übernehmen. Alle administrativen Aktivitäten können überwacht, protokolliert und zurückverfolgt werden, da sich die Administratoren vor der Übernahme authentifizieren müssen. Die administrativen Teile werden standardmäßig in folgende Rollen gesplittet: Security administrator (secadmin), System administrator (admin), Operator (oper).

6. LSM Architektur

Linux Security Modules wird die neue offizielle und fest integrierte Linux Sicherheits-API sein. Diese API ist schon integriert in dem noch in Entwicklung befindlichen Linuxkernel Version 2.6. Die Entstehung dieser API beginnt mit dem 2.5 Linux Kernel Summit im März 2001, wo die NSA einen Vortrag über ihr Security-Enhanced Linux hielt. Durch diesen Vortrag inspiriert beschrieb Linus Torvalds wie ein Security-Framework nach seiner Meinung auszusehen hat, so dass er es in den Standard Linux-Kernel integrieren würde. Dieses Framework soll einem Kernelmodul erlauben transparent in den Entscheidungsprozess des Kernels einzugreifen. Jedoch sollen nur vom Linuxkernel erlaubte Operationen durch das Modul abgelehnt werden können, d.h. die Sicherheit eines Linuxsystems kann nur erhöht, aber nicht erniedrigt werden. Des weiteren soll auch die Migration der Linux Capabilities in dieses Framework möglich sein.
Das Linux Security Modules Projekt begann, ermutigt durch die Aussage von Linus Torvalds, damit ein solches Framework zu entwickeln. Die Entwicklung findet durch Immunix, SELinux, SGI, Janus und einzelne Personen wie Greg Kroah-Hartman, James Morris und andere statt. Der erste LSM-Patch ging dann in den 2.5.29 Linuxkernel ein und wird seither innerhalb des 2.5 Kernels und als Patch für den 2.4 Kernel weiterentwickelt. Dieses Projekt ist aber umstritten bei einigen Kernelentwicklern, denen die LSM-API aus verschiedenen technischen Gründen nicht gefällt. Deshalb ist selbst beim aktuellen 2.5.66 Kernel ein Teil des LSM-Projekts als externer Patch realisiert.

7. Vorstellung des NSA SELinux / LSM-Architektur

7.1. SELinux

Die National Security Agency/Central Security Service ist Amerikas Organisation im Bereich Kryptologie. Es verwaltet hoch spezialisierte Anstrengungen, um amerikanische Informationssysteme zu schützen und nach „fremder Intelligenz“ zu forschen. Da die NSA

eine Organisation im Bereich Hochtechnologie ist, bewegt sie sich an den Grenzen der Kommunikation und Datenverarbeitung. Außerdem ist es eine der bedeutendsten Zentren der Fremdsprachenanalyse und der Forschung innerhalb der Regierung.

Von ihr stammt auch das Security-Enhanced Linux (kurz: SELinux). Dieses stellt eine Portierung der Flask Security Architektur nach Linux dar. Es war auch der Anstoß für das LSM-Projekt, wie bereits erwähnt. Der mitgelieferte Security Server baut auf der Flask Security Architektur auf und implementiert eine Kombination aus Type Enforcement und Role-based Access Control. Die Implementierung des SELinux Security Servers hat aber folgende Besonderheiten:

- Das Type Enforcement kennt bei SELinux keinen Unterschied zwischen Objekten und Subjekten. Eine Domäne ist einfach ein anderer Name für einen Type eines Prozesses.
- Da jedes Objekt und Subjekt bei SELinux noch zusätzlich eine Security Class besitzt, wird bei einer Transition oder Access Decision zusätzlich diese Class berücksichtigt.
- Des Weiteren werden Domänen und Benutzer getrennt behandelt. RBAC stellt somit eine weitere Abstraktion zwischen Benutzer und Domänen dar.

Diese Security Architektur benötigt aber gepatchte Versionen aller Programme, die einen Benutzer authentifizieren. Diese Patches und Programme sind für Red Hat 7.1 entwickelt und getestet worden. Die Konfiguration der Security-Policy findet über Textdateien statt.

Für die Portierung auf verschiedene Distributionen schaue man auf die Homepage „SELinux Distribution Integration News“, wo unter anderem eine Portierung zur aktuellen SuSE Distribution existiert. Dies ist z.B. für die SuSE Distribution als Source-RPMS gelöst, die sehr einfach zu installieren sind. Zusätzlich zu den Programmen müssen auch die Regeln angepasst werden. Da die Pakete nicht direkt von SuSE stammen, kann man aber nicht den automatischen SuSE-Update Mechanismus für diese Pakete verwenden. Dies bedeutet einen höheren Wartungsaufwand für den Systemadministrator.

7.2. Übersicht

Eigenschaften	Linux Kernel Patches				Linux Security Modules	
	RSBAC	grsecurity	Systrace	Openwall	SELinux	LSM Openwall
Homepage	rsbac.org	grsecurity.org	citi.umich.edu	openwall.com	nsa.gov	
unterstützte Linux Kernel	2.2, 2.4	2.4	2.4, 2.5.52	2.2	2.4, 2.5	2.4, 2.5
Discrete Access Control	Ja	Ja	Nein	Nein	Nein	Nein
Mandatory Access Control	Ja	Ja	Ja	Nein	Ja	Nein
Autom. Policy-Erstellung	Nein	Nein	Ja	Nein	Nein	Nein
Type Enforcement	Nein	Nein	Nein	Nein	Ja	Nein
Role-based Access Control	Ja	Nein	Nein	Nein	Ja	Nein
Non-Exec Stack	Nein	Ja	Nein	Ja	Nein	Nein

8. Beurteilung/Fazit

Wenn Capabilities wesentlich besser wie herkömmliche Superuser-Systeme sind, warum werden sie dann nur recht selten verwendet? Nun zum einen gibt es neben den vielen Vorteilen auch Nachteile bei Dingen wie etwa der Rechterücknahme oder der Rechtekonsistenzhaltung beim Systemneustart, auf der anderen Seite ist das Problem historisch bedingt. Die ersten Capability-Systeme wurden in Hardware gebaut und sollten den Zugriff auf den Hauptspeicher regeln. Das machte sie extrem langsam und komplex und führte zu einem äußerst schlechten Ruf. Auch in den 80ern wurde nur recht schlechte Arbeit an Microkernels (die den Capability-Systemen ähnlich sind) abgeliefert und ebenso schlechte Analysen ergaben, dass das ein Problem der Microkernels im Allgemeinen sei (obwohl es an schlechten Implementationen lag). Des Weiteren lohnen sich derartige Systeme auch erst ab einer gewissen Größe, d.h. wenn ein kleines System mit nur ein oder zwei Administratoren und einer überschaubaren Anzahl an Benutzern zu administrieren ist und dabei die Sicherheit des Systems nicht die entscheidende Rolle spielt, so kann man dies auch mit einem normalen Superuser-System problemlos angehen. Übrigens gibt es auch UNIX-kompatible Umgebungen, die auf ein Capability-System aufsetzen, wenngleich dies andersherum nicht möglich ist. Existierender Code kann also trotzdem weiterverwendet werden.

9. Quellen

http://www.inf.fu-berlin.de/lehre/SS02/sysi/
http://www.nsa.gov/selinux/
http://www.nm.informatik.uni-muenchen.de/Literatur/MNMPub/Fopras/jaeg03/HTML-Version/jaeg03.html
http://selinux.sourceforge.net/ <- SELinux Distribution Integration News