Linux Security Modules - Sicherheit als Kernelmodul

Michael Pradel

17.06.2004

Inhaltsverzeichnis

1 Einleitung

Sicherheit in Computersystemen gehört zu den wichtigsten und meistdiskutierten Problemen in der Computerbranche überhaupt. In unzähligen Bereichen ist es heutzutage unabdingbar, festgesetzte Sicherheitsziele zu verwirklichen, um beispielsweise Firmengeheimnisse vor fremden Augen zu schützen, die Ergebnisse einer Wahl vor nicht berechtigten Änderungen zu bewahren oder die ständige Verfügbarkeit eines wichtigen Internetservers sicherzustellen. Sicherheit umfasst dabei viele Aspekte im und um das Computersystem herum. Grundsätzlich unterscheidet man drei große Ziele und damit verbundene Bedrohungen: Zum einen gilt es die Datenvertraulichkeit zu wahren - damit geheime Informationen auch geheim bleiben. Außerdem soll die Datenintegrität sichergestellt sein, d.h. unautorisierte Benutzer können Daten nicht modifizieren, löschen oder falsche Daten hinzufügen. Als drittes Ziel gilt die Systemverfügbarkeit - also Störungen zu verhindern,die zur Unbenutzbarkeit des Systems führen könnten.

Ich möchte mich hier mit der Sicherheit auf Betriebssystemebene und insbesondere dem Erreichen der ersten beiden Ziele Datenvertraulichkeit und Datenintegrität beschäftigen. Im Allgemeinen gilt Linux als relativ sicheres Betriebssystem. Besonders im Vergleich mit dem etwas weiter verbreiteten System eines großen amerikanischen Softwareherstellers wird häufig der Faktor Sicherheit als großer

Vorteil angepriesen. Grundsätzlich stimme ich mit dieser Aussage überein. Allerdings haben die standardmäßig verwendeten Mechanismen einige Schwächen und mangeln an Möglichkeiten zur flexiblen Umsetzung von Sicherheitskonzepten.

Einen in der 2.6er Reihe des Linux-Kernels fest integrierten Ansatz zur Lösung dieses Problems, die Linux Security Modules (LSM), möchte ich hier vorstellen. Die LSM ermöglichen die flexible Einbindung verschiedener Sicherheitsmodelle in Form von Modulen. Nach der allgemeinen Darstellung dieses Frameworks möchte ich SELinux als das wohl bekannteste Modul erläutern. Zuvor gehe ich jedoch auf einige gängige Sicherheitsmodelle und deren Verwendung unter Linux ein.

2 Sicherheitsmodelle

Aus der Sicht des Betriebssystems gibt es in einem Computer unzählige Objekte. Dies können Dateien, Prozesse, aber auch CPUs, Plattenlaufwerke oder andere Geräte sein. Außerdem existieren verschieden Rechte, z.B. das Lesen, Schreiben oder Ausführen. Eine Menge von (Objekt, Rechte)-Paaren nennt man Domäne. Ein Paar legt dabei für ein bestimmtes Objekt fest, welche Operationen auf diesem Objekt ausgeführt werden dürfen. In der Regel entspricht eine Domäne einem Benutzer, sie kann aber auch allgemeiner betrachtet werden. Eine sehr anschauliche Form der Darstellung der in einem Betriebssystem vorhandenen Domänen ist die Schutzmatrix.

2.1 Die Schutzmatrix

In einer Schutzmatrix werden in den Spalten alle Objekte und in den Zeilen alle Domänen dargestellt. In den dadurch entstehenden Feldern der Matrix werden nun die jeweils erlaubten Operationen der Domäne für das Objekt eingetragen. Versucht ein Benutzer beispielsweise auf eine bestimmte Datei lesend zuzugreifen, kann das Betriebssystem durch Nachschauen im richtigen Feld feststellen, ob diese Operation zu den gestatteten gehört.

2.2 Capabilities und Zugriffskontrolllisten

Man kann sich nun sicherlich gut vorstellen, dass die Schutzmatrix eines tatsächlich laufenden Systems sehr groß und äußerst dünn besetzt ist, da die meisten Domänen überhaupt keinen Zugriff auf viele Objekte haben. Das entsprechende Feld bleibt also leer. Daraus ergeben sich zwei Ansätze, um unnötige Redundanz zu sparen:

- Zum einen kann man nur die Spalten speichern und alle nicht leeren Zellen weglassen. Bei diesem Ansatz namens **Zugriffskontrolllisten (oder Access Controll List, ACL)** werden also für jedes Objekt die in irgendeiner Art zugriffsberechtigten Domänen und die jeweils erlaubten Operationen gespeichert. Dieses Prinzip wird standardmäßig bei Linux verwendet. Als Domänen sind dabei Benutzer, Gruppen (Zusammenfassungen von Benutzern) und „alle“gestattet. Außerdem sei bemerkt, dass jedes Objekt unter Linux einen Besitzer und eine Gruppe haben muss. Die angebotenen Operationen sind Lesen, Schreiben und Ausführen. Für ein Objekt ergibt sich daraus dann ein Neuner-Tupel, wobei die ersten drei Werte des Tupels die Rechte des Besitzers, die mittleren drei die Rechte der Gruppe und die letzten drei die Rechte der restlichen Welt an diesem Objekt darstellen. Zu beachten ist hierbei, dass die Einträge der Reihe nach gelesen werden und der erste passende vom Betriebssystem genommen wird. Zum Beispiel könnte sich für eine Datei folgendes Tupel ergeben:

  ```
  rwx--x--x
  ```

  Die so gekennzeichnete Datei darf von ihrem Besitzer gelesen, geschrieben und ausgeführt werden. Alle andere (auch Mitglieder der zur Datei gehörigen Gruppe) dürfen die Datei lediglich

ausführen.

Dieses von Linux verwendete Sicherheitsmodell ist so jedoch stark eingeschränkt. Beispielsweise sollte jeder Benutzer in der Lage sein, sein Passwort zu ändern. Da sich das für alle schreibbar machen der Passwort-Datei hier offensichtlich nicht anbietet, gibt es ein Programm um Passwörter zu ändern. Nun werden Programme normalerweise mit den Rechten dessen, der es gestartet hat, ausgeführt, so dass ein Benutzer immer noch nicht in der Lage ist, sein Passwort zu ändern. Aus diesem Grund wurde dem Neuner-Tupel noch ein zehntes Bit namens `SETUID` hinzugefügt. Ist es gesetzt, erhält das Programm bei der Ausführung die Rechte seines Besitzers. Nun gehört das Linux-Programm zum Ändern des Passwortes *passwd* dem Besitzer der Passwort-Datei *root* und hat das `SETUID` Bit gesetzt. Führt ein normaler Benutzer es aus, kann also problemslos das neue Passwort in die Datei eingefügt werden.

Aus dieser Idee ergeben sich leider weitere Probleme. Wird beispielsweise ein fehlerhaftes Programm (Und welches Programm ist schon erwiesenermaßen fehlerfrei?) mit dem `SETUID` Bit versehen, so könnten unter Ausnutzung dieses Fehlers nicht gewollte Operationen mit Hilfe der Dank des `SETUID` Bit verliehenen Rechte getätigt werden.
Neben `SETUID` gibt es auch noch `SETGID`, welches dem ausgeführten Programm analog zum eben besprochenen die Rechte der zugehörigen Gruppe verleiht.

- Die andere Variante zur Reduzierung der dünn besetzten Schutzmatrix sind **Capabilities**. Hierbei wird mit allen nicht-leeren Zellen jedem Prozess eine Liste mit allen Objekten auf die er zugreifen kann, inklusive der genau gestatteten Operationen, zugeordnet. Die Matrix wird also zeilenweise zerlegt. Dieses Modell wurde in der 2.2er-Reihe des Linux-Kernels eingeführt und bietet eine Möglichkeit *root* bestimmte Rechte zu entziehen. Im 2.6.2-Kernel sind 28 solcher entziehbaren Capabilities aufgeführt, nachzulesen in */usr/include/linux/capabilities.h*. Als Beispiel möchte ich `CAP_CHOWN` erwähnen - das Recht, den Besitzer einer Datei zu ändern, sowie `CAP_SYS_MODULE` - die Berechtigung, Kernel-Module zu laden. Mit Hilfe des Programmes *lcap* können diese Rechte ohne größeren Aufwand entzogen werden und sind bis zum nächsten Neustart des Systems nicht wieder erlangbar.

2.3 DAC und MAC

Beim Festlegen der Rechte und Regeln in einem System sind grundsätzlich zwei Wege unterscheidbar. Einerseits kann jeder Benutzer selbst bestimmen, wer für seine Dateien und andere Objekte welche Rechte hat. Diese Variante ist als **Discretionary Access Control (DAC)** oder auch benutzerbestimmte Zugriffskontrollstrategie bekannt und ist bei Linux der „übliche“Weg.
Ein Benutzer könnte allerdings aus verschiedenen Gründen, von Unwissenheit bis zu böser Absicht, seine Objekte mit mehr Rechten ausstatten, als der Allgemeinheit lieb ist, was die Notwendigkeit einer **Mandatory Access Control (MAC)** oder auch systembestimmten Zugriffskontrollstrategie begründet. Dabei ist der Besitzer einer Datei oder eines anderen Objektes nicht in der Lage dessen Rechte zu verändern, sondern diese werden vielmehr durch eine zentrale Stelle festgesetzt. MAC ist bei Linux eigentlich nicht vorgesehen, aber beispielsweise durch die SELinux Erweiterung möglich.

3 Linux Security Modules

3.1 Problemstellung und Lösungsidee

Für den Linux-Kernel existieren zahlreiche Implementierungen von erweiterten Zugriffskontrollmechanismen. Normalerweise bietet der Kernel Modulen aber leider keinen direkten Zugriff auf Kernel-Objekte, also den Strukturen, welche beispielsweise Prozesse, offene Dateien oder Netzwerk-Geräte darstellen, sodass die meisten dieser Projekte ihre Implementierung als Patch veröffentlichen. Um das

Prinzip der Modularisierung auch im Sicherheitsbereich nutzen zu können, strebte Linus Torvalds die Entwicklung eines generischen Frameworks zur Zugriffskontrolle an, welches die Einbindung von Sicherheitserweiterungen als Kernel-Modul ermöglichen soll.

3.1.1 Das generische Framework

Das generische Framework sollte drei Kriterien genügen: Zum ersten sollte das Framework allgemein genug gestaltet sein, um verschiedenste Sicherheitsmodelle durch das simple Einbinden des entsprechenden Moduls nutzen zu können. Gleichzeitig galt es dabei die notwendigen Veränderungen im Kernel-Code zu minimieren. Die dritte Forderung bestand in der Unterstützung der existierenden Posix.1e Capabilities in Form eines Moduls.

3.2 Implementierung

3.2.1 Sicherheitsfelder

Um Modulen das Speichern von sicherheitsrelevanten Daten direkt an Kernel-Objekten zu ermöglichen, wurde einigen Kernel-Objekten ein Sicherheitsfeld in Form eines `void*` Pointers hinzugefügt. Dieser ist *opaque*, wird also vom Rest des Kernels ignoriert. Die Nutzung und Verwaltung des Sicherheitsfeldes ist vollkommen dem Modul überlassen und kann auch vollständig weggelassen werden.

3.2.2 Hook Functions

Um oben erwähnte Sicherheitsfelder zu schreiben und lesen, und auf Kernel-Objekte zu bestimmten Zeitpunkten unmittelbar zugreifen zu können, bietet LSM sogenannte Hook Functions, also „Hakenfunktionen“an. Sie werden bei ganz bestimmten Ereignissen aufgerufen und können vom jeweiligen Modul implementiert werden. Gespeichert werden diese Funktionen in einer globalen Tabelle namens `security_ops`.

Als Beispiel möchte ich die Funktion `vfs_mkdir` anführen, welche zum Erstellen eines neuen Verzeichnisses auf der Ebene des virtuellen Dateisystems (*vfs*) dient. Sie enthält zwei Aufrufe von Hook-Functions. Der erste Aufruf (Zeile 10) dient, der Überprüfung, ob das Erstellen des Verzeichnisses gestattet ist. Gibt die Funktion `security_ops->inode_ops->mkdir` einen negativen Wert zurück, wird kein Verzeichnis erstellt. Außerdem wird beim Verlassen der `vfs_mkdir` Funktion eine Hook-Function aufgerufen (Zeile 17), um das Beschreiben des Sicherheitsfeldes der neu entstandenen Inode-Struktur zu ermöglichen.

```
01  int vfs_mkdir(struct inode *dir, struct dentry *dentry, int mode)
02  {
03    int error = may_create(dir, dentry, NULL);
04
05    if (error) return error;
06
07    if (!dir->i_op || !dir->i_op->mkdir) return -EPERM;
08
09    mode &= (S_IRWXUGO|S_ISVTX);
10    error = security_inode_mkdir(dir, dentry, mode);
11    if (error) return error;
12
13    DQUOT_INIT(dir);
14    error = dir->i_op->mkdir(dir, dentry, mode);
15    if (!error) {
```

```
16      inode_dir_notify(dir, DN_CREATE);
17      security_inode_post_mkdir(dir,dentry, mode);
18    }
19    return error;
20  }
```

3.2.3 Hinzufügen von Systemaufrufen

Desweiteren ermöglicht LSM einem Modul das Hinzufügen neuer Systemaufrufe. Diese können im Benutzeraddressraum laufenden Programmen dazu dienen, Informationen mit dem Kernel auszutauschen oder eine bestimmte Funktionalität aufzurufen.

Neben Sicherheitsfelder, Hook Functions und dem Hinzufügen bietet das LSM Framework Modullen noch einige andere Möglichkeiten. Auf diese möchte ich hier nicht weiter eingehen.

4 SELinux

Als Beispiel für ein Linux Security Module möchte ich auf SELinux eingehen. SELinux wurde von der National Security Agency (NSA) entwickelt, und 2001 unter der GPL veröffentlicht. Zunächst wurde es als Patch für den Linux-Kernel programmiert und später als LSM reimplementiert.

4.1 Konzepte

SELinux ermöglicht eine flexible Variante der oben beschriebenen systembestimmten Zugriffsstrategie names Flask. Sie bietet eine klare Trennung zwischen den Security Policies, also den durchzusetzenden Regeln, und deren Umsetzung. Diese findet im sogenannten Sicherheitsserver statt, der in den jeweiligen Kernel-Subsystemen (z.B. Prozessverwaltung, Dateisystem oder Socket- und Netzwerkimplementierung) zu finden ist. Um das Erlauben oder Verbieten einer Operation zu beschleunigen werden alle errechneten Zugriffsentscheidungen im Access Vector Cache (AVC) zwischengespeichert.

Jedem Kernelobjekt wird von einem Objektmanager eine Sicherheits ID (SID) zugeordnet. Diese ordnet es in einen Sicherheitskontext ein - einer Kombination aus Benutzernamen, Rolle, Typ und Sicherheitslevel. Wird ein neues Kernelobjekt erstellt, berechnet der Security Server aus der SID des Erzeugers, der SID eines verknüpften Objektes und der Sicherheitsklasse des neuen Objektes die neue SID, sodass es wiederum in einen Sicherheitskontext eingeordnet ist. Wird beispielsweise eine neue Datei erzeugt, kann deren Sicherheits ID mit folgendem Aufruf berechnet werden:

```
ret = security_transition_sid(
        current->sid,
        dir->sid,
        SECCLASS_FILE,
        &sid);
```

Dem Sicherheitsserver werden also die SID des erzeugenden Prozesses, des aktuellen Verzeichnisses und die Sicherheitsklasse `SECCLASS_FILE` übergeben. Die berechnete SID befindet sich anschließend im Parameter `sid`.

Einem Kernelobjekt werden vom zuständigen Sicherheitsserver eine Reihe von Attributen zugeordnet, mit deren Hilfe Zugriffsentscheidungen getroffen werden können. Beipielsweilse ordnet der Sicherheitsserver des Dateisystems jedem Verzeichnis folgende Attribute zu:

`add_name`	Datei hinzufügen
`remove_name`	Datei entfernen
`reparent`	Eltern-Verzeichnis verändern
`search`	Durchsuchen
`rmdir`	ganzes Verzeichnis entfernen
`mounton`	als Einhängepunkt (Mountpoint) verwenden
`mountassociate`	einhängbare Dateisysteme

Anderen Kernelobjekten (z.B. Prozessen, Dateien, Sockets oder Netzwerkschnittstellen) werden ebenso Attribute zugeordnet. Soll beispielsweise ein Dateisystem unter der Kontrolle von SELinux gemountet werden, wird überprüft, ob der Prozess die *mounton*-Berechtigung für das Zielverzeichnis und die *mount*-Rechte für das Dateisystem hat. Außerdem muss das *mountassociate*-Attribut das Einhängen des Dateisystems in das Verzeichnis gestatten.

4.2 Umsetzung als LSM

Bei der Implementierung von SELinux als LSM wird das bestimmten Kernelobjekten hinzugefügte Sicherheitsfeld in Form eines `void*` Pointers genutzt, um eine Sicherheitsstruktur entsprechend dem jeweiligen Typ des Kernelobjektes (z.B. `struct file_security_struct`) zu speichern. Diese enthält u.a. die SID des Objektes.

Die Umsetzung von Zugriffsentscheidungen findet mit Hilfe der Hook-Funktionen statt. Als Beispiel möchte ich die in 3.2.2 erwähnte Funktion zum Erstellen eines Verzeichnisses auf der Ebene des virtuellen Dateisystems wieder aufgreifen. Sie rief als Hook-Funktion `security_ops->inode_ops->mkdir` auf, welche im Kernel-Code in *security/selinux/hooks.c* zu finden ist:

```
static int selinux_inode_mkdir(struct inode *dir,
      struct dentry *dentry, int mask)
{
  return may_create(dir, dentry, SECCLASS_DIR);
}
```

Sie ruft wiederum die in derselben Datei zu findende Funktion `may_create()` auf:

```
static int may_create(struct inode *dir,
      struct dentry *dentry, u16 tclass)
{
  // [..]
  rc = avc_has_perm(tsec->sid, dsec->sid, SECCLASS_DIR,
DIR__ADD_NAME | DIR__SEARCH,
&dsec->avcr, &ad);
  // [..]
}
```

Hier wird nun `avc_has_perm()` aufgerufen, welche ich oben bereits erwähnte, um festzustellen ob das Erstellen des Verzeichnisses gestattet werden soll oder nicht.

Auf weitere Details der Implementierung den SELinux als LSM möchte ich hier nicht eingehen.

5 Zusammenfassung

Da die standardmäßig im Linux-Kernel integrierten Sicherheitsmodelle höheren Anforderungen nicht genügen, initierte Linus Torvalds die Schaffung eines Sicherheitsframeworks. Es sollte die Implementierung von unterschiedlichen Sicherheitsmodellen als Linux Kernelmodul ermöglichen, und somit eine

allgemeine, einfache und trotzdem ausreichend mächtige Schnittstelle zur Einbindung solcher bieten.

Mit den LSM wurde dieses Framework realisiert. Es bietet unter anderem vom Modul frei nutzbare Sicherheitsfelder für Kernelobjekte, um sicherheitsrelevante Informationen zu speichern. Außerdem ermöglichen Hook-Funktionen an wichtigen Stellen die Verwehrung von Rechten durch das Modul.

Ein das LSM Framework nutzende Modul ist SELinux. Es implemtiert die flexible Variante der systemgesteuerten Zugriffskontrolle Flask.

6 Quellen

- *Moderne Betriebssysteme* - Andrew Tanenbaum
- *Linux Security Modules: General Security Support for the Linux Kernel* - Chris Wright, Crispin Cowan
- *Integrating Flexible Support for Security Policies into the Linux Operating System* - Peter Loscocco, Stephen Smalley
- *Implementing SELinux as a Linux Security Module* - Stephen Smalley
- Linux Source 2.6.2