



11 Hintergrund

11.1 Geschichte

SELinux ist das Ergebnis einer langen Entwicklung, an der unter der Federführung der NSA viele unterschiedliche Unternehmen und Institute mitwirkten. Wie bereits in Teil I erwähnt wurde, entwickelten die beiden Wissenschaftler David Bell und Leonard LaPadula 1973 das nach ihnen benannte Modell eines Multi-Level-Security-Systems. Später waren Bell und LaPadula auch an der Entwicklung des Orange Books¹ beteiligt. Das *Orange Book* definiert sechs verschiedene Klassen: C1, C2, B1, B2, B3 und A1. Die Betriebssysteme in den Klassen C1 und C2 setzen für die Sicherheit der Daten lediglich DAC-Systeme ein, während ab der Stufe B1 Mandatory-Access-Systeme verlangt werden.

Die ersten MAC-Systeme waren Multi-Level-Security-Systeme und wurden für militärische und geheimdienstliche Zwecke eingesetzt. Es handelte sich häufig um Betriebssysteme, die außerhalb dieser Einsatzbereiche nicht genutzt wurden. Die MLS-Systeme waren aber sehr starr und konnten nicht flexibel angepasst werden. Sie hatten nur ein Ziel: Die Vertraulichkeit der Daten musste gewährleistet werden. Flexiblere Betriebssysteme der Stufen B1 und aufwärts wurden benötigt. Forscher der Information Assurance Research Group der NSA entwickelten daraufhin zusammen mit dem Unternehmen *Secure Computing Corporation* (SCC) eine neue flexible Mandatory-Access-Control-Architektur, die auf dem Modell des *Type-Enforcement* basierte. Dieser Mechanismus wurde zuerst für das Betriebssystem Logical Coprocessing Kernel (*LOCK*) entwickelt. *LOCK* wurde noch von Honeywells Secure Computing Technology Center (SCTC) entwickelt, aus dem später die Firma SCC hervorging. In Zusammenarbeit mit der NSA entstanden zwei Mach-Kernel-basierte Prototypen: *DTMach* und *DTOS*². An der Entwicklung von *DTOS* war auch die *Flux*-Forschungsgruppe der Universität in Utah beteiligt. Gemeinsam wurden die Techniken in das Betriebssystem *Fluke* implementiert. Dabei wurde die Architektur erweitert und ergänzt, sodass nun die Richtlinien dynamisch geladen werden konnten. Diese neue Architektur erhielt den Namen *Flask*³. *Fluke* war aber immer noch ein OS, das lediglich zu Forschungszwecken eingesetzt wurde. Um die Technologie einem breiteren Publikum zur Verfügung zu stellen und so mehr Erfahrung zu sammeln,

¹ Der eigentliche Titel lautet: Trusted Computer System Evaluation Criteria (TCSEC). Es wird aber allgemein als Orange Book bezeichnet.

² <http://www.cs.utah.edu/flux/dtos/>

³ <http://www.cs.utah.edu/flux/flask/>

implementierte die NSA die Flask-Architektur in Linux. Dabei wurde die NSA von *Network Associates* und *MITRE* unterstützt. Im Dezember 2000 wurde SELinux auf der Basis des Linux-Kernels 2.2 als Open-Source-Software veröffentlicht.

Auf dem Linux Kernel Summit 2001 in Ottawa wurde das Linux-Security-Module- (LSM-)Projekt gestartet. Hiermit sollte erstmals eine standardisierte Sicherheitschnittstelle im Linux-Kernel geschaffen werden. In den folgenden Jahren wurde SELinux auf die LSM-Schnittstelle portiert und ist seit dem Kernel 2.6 fest im Kernel enthalten.

Eine der ersten Distributionen, die SELinux aufgenommen hat, war *Fedora Core*. Ab Version 2 enthält diese Distribution die SELinux-Erweiterung. Während in der Version 2 die Unterstützung noch fehlerhaft war, konnte in den Versionen 3 bis 5 eine bessere Unterstützung erreicht werden. *Debian Etch* unterstützt ebenfalls SELinux.

Die erste kommerzielle Distribution mit SELinux-Unterstützung ist Red Hat Enterprise Linux. Die Version 4 unterstützt SELinux mit einer Targeted Policy (siehe Abschnitt 17.1).

Inzwischen wurde die Architektur auch in BSD⁴ und Darwin⁵ implementiert.

11.2 Architektur

In diesem Kapitel stelle ich Ihnen die Architektur des SELinux-Frameworks vor. Wahrscheinlich werden Sie nach dem ersten Lesen dieses Kapitels noch einige Fragen haben. Bitte lesen Sie dieses Kapitel erneut, wenn Sie sich ein wenig mit SELinux beschäftigt haben. Dann wird Ihnen einiges klarer werden.

SELinux basiert auf dem *Flask*-Kernel. Flask ist der Flux-Advanced-Security-Kernel. Flux ist eine Forschungsgruppe an der Universität von Utah, die sich allgemein mit Software-Systemen beschäftigt⁶.

Die Flask-Architektur ist zuerst für einen *Microkernel* entwickelt worden und wurde von dem *Fluke*-Microkernel-Betriebssystem abgeleitet. Die Flask-Architektur beschreibt die Interaktionen zwischen den beiden verschiedenen Subsystemen, die die Entscheidungen treffen und umsetzen. Dies sind in der Flask-Architektur zwei voneinander getrennte Subsysteme. Während die Umsetzung von dem *Object-Manager* durchgeführt wird, wird die Entscheidung im *Security-Server* getroffen (siehe Abbildung 11.1).

Der Object-Manager wird bei den zu überwachenden Objekten implementiert. Dieser Object-Manager fängt jede Anfrage auf das Objekt ab und stellt diese an den Security-Server. Dieser prüft auf beliebige Weise (bei SELinux anhand der SELinux-Richtlinie), ob der Zugriff erlaubt ist, und liefert die Entscheidung an den Object-

⁴ SEBSD: <http://www.trustedbsd.org/sebsd.html>

⁵ SEDarwin: <http://www.trustedbsd.org/sedarwin.html>

⁶ <http://www.cs.utah.edu/flux/>

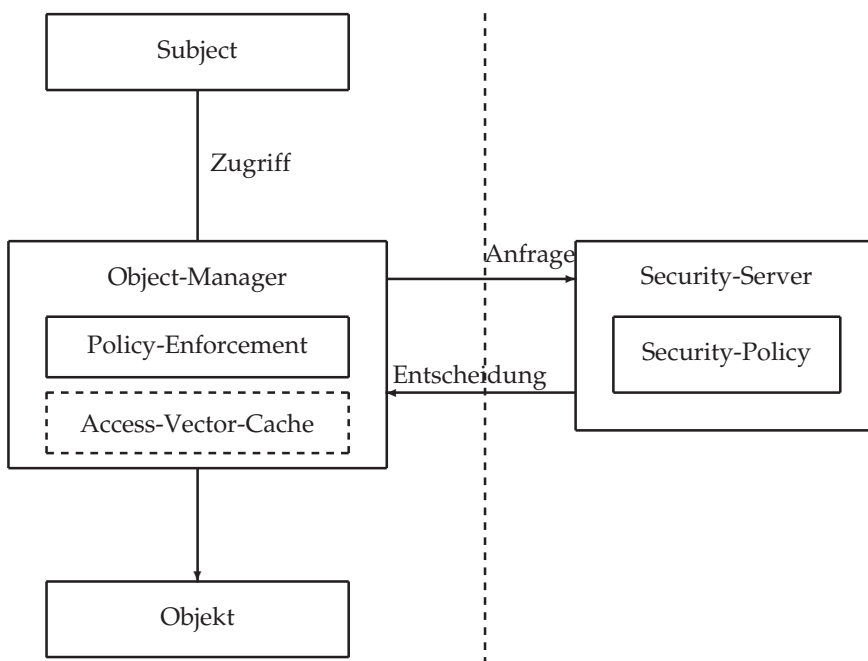


Abbildung 11.1: Flask stellt eine Trennung zwischen Policy Enforcement und Decision dar.

Manager zurück. Dieser setzt dann die Entscheidung um und gibt den Zugriff frei oder verweigert den Zugriff.

Der Object-Manager hat drei Aufgaben in der Flask-Architektur:

- Kommunikation mit dem Security-Server. Der Object-Manager muss mit dem Security-Server kommunizieren, um die Zugriffsanfragen weiterzuleiten und Labeling-Informationen zu erhalten.
- Speicherung der Entscheidungen in einem *Access-Vector-Cache* (AVC). Damit nicht jeder Zugriff erneut an den Security-Server weitergegeben werden muss, verfügt der Object-Manager über einen Access-Vector-Cache, in dem die Entscheidungen gespeichert werden.
- Registration bei dem Security-Server. Der Security-Server muss den Object-Manager benachrichtigen können, sobald sich die Policy ändert. Der Object-Manager muss dann die gespeicherten Entscheidungen im AVC löschen.

11.2.1 Access-Vector-Cache

Am einfachsten wäre es sicherlich, wenn der Object-Manager bei jedem Zugriff den Security-Server fragen würde, ob der Zugriff erlaubt ist. Dies wäre jedoch sehr lang-

sam und daher nicht sinnvoll. Die Flask-Architektur erlaubt daher das Caching der Entscheidungen in dem Object-Manager.

Das AVC-Modul wird von allen Object-Managern gemeinsam genutzt. So kommunizieren die Object-Manager nicht direkt mit dem Security-Server, sondern über den AVC.

Muss nun der Zugriff eines Subjekts auf ein Objekt evaluiert werden, sendet der Object-Manager einen Access-Vector, der aus dem *Subject-Security-Context*, dem *Object-Security-Context* und dem Zugriff besteht, an den AVC. Dieser prüft, ob die entsprechende Entscheidung bereits im AVC gespeichert ist. Ist das nicht der Fall, wird der Access-Vector an den Security-Server weitergesendet. Dieser prüft und entscheidet über den Zugriff und sendet die Entscheidung zurück an den AVC. Aus Geschwindigkeitsgründen kann der Security-Server mehrere (auch zusätzliche nicht angefragte) Entscheidungen gleichzeitig an den AVC senden. Dieser wird sie für zukünftige Zugriffe speichern. So wird zum Beispiel immer ein kompletter Access-Vector zurückgeliefert. Der Access-Vector enthält dann für das betroffene Objekt alle Zugriffe, die das Subjekt durchführen darf.

Hier erklärt sich dann auch, warum die Protokollmeldungen von SELinux nicht SELinux, sondern AVC im Namen tragen. Das Subsystem, das die Meldung auslöst, ist der AVC.