

# G Sicherheitsmodelle

---

- Ziel: Klassifikation und Einordnung bekannter und weiterer Sicherheitskonzepte
- Fokus in diesem Abschnitt: Zugriffskontrolle
- Beispiel für reale Umsetzung: SE-Linux
  
- Formalisierung von Sicherheitsmodellen:  
Grundlage für verifizierbare Aussagen
  - Problem: komplette Sicherheitsmodelle komplexer Anwendungen meist nicht handhabbar
  - aber: Grundlage für den Nachweis von Eigenschaften konkreter Konzepte

# G.1 Objekte und Subjekte

---

- ▲ **Objekte:** die zu schützenden Einheiten des Modells
- **Problem:** häufig sehr grob-granulare Festlegung der Objekte
  - z. B. Dateien, Directories
  - ↳ unvollständige Erfassung sicherheitsrelevanter Eigenschaften
    - z. B. Anlege-/Löschrecht für Directories
    - Folge: Sicherheitslücken
  - ↳ Zusammenfassung unterschiedlicher Einheiten zu einem Objekt
    - Verletzung des "need-to-know"-Prinzips

## G.1 Objekte und Subjekte (2)

---

- ▲ **Subjekte:** greifen auf Objekte zu
- Problem: grob-granulare Subjekte
  - Benutzer, Benutzergruppen
  - ↳ Rechte werden an ganze Gruppen vergeben ohne individuelle Bedürfnisse zu berücksichtigen
- Problem: Verbindung Subjekt - reale Person
  - bei Rechtevergabe erfolgt Bezug auf Vertrauenswürdigkeit einer Person
  - Rechteaübung erfolgt durch Software, die für die Person agiert
    - ? ist die Software so vertrauenswürdig wie die Person

## G.1 Objekte und Subjekte (3)

---

- Anwendungsspezifische Granularität
  - ◆ Zugriffsrechte für Objekte beliebiger Granularität festlegbar
    - Persistente Objekte einer Anwendung
    - Sätze einer Datenbank
  
- Anwendungsspezifische Rechte
  - ◆ nicht nur systemweite Standard-Rechte
    - z. B. Rechte zum Aufruf bestimmter Methoden (Kontrolle z. B. mit Hilfe des Typ-Systems)
  
- Feingranulare Subjekte
  - ◆ nicht nur Benutzer und Benutzergruppen
    - zeitlich- und Rollen-abhängige Rechte

## G.2 Zugriffsrechte

---

- universelle Rechte
  - erlauben allgemeine, nicht objektspezifische Operationen
  - z. B. read/write auf Dateien
    - unabhängig von der Semantik des Dateiinhalts
  - Realisierung: konventionelle Dateisysteme
  
- objektspezifische Rechte
  - Einschränkung von Zugriffsmöglichkeiten auf einen festgelegten funktionalen Kontext
  - Manipulation eines Objekts nur gemäß der festgelegten Semantik der erlaubten Operation (Typ-spezifische Rechte)
  - Realisierung:
    - objektorientierte Middleware
    - Datenbanksysteme

## G.3 Zugriffskontrollstrategien

---

### 1 Benutzer-bestimmbare Zugriffskontrollstrategien

---

#### *Discretionary Access Control (DAC)*

- Eigentümer eines Objekts ist für die Rechtevergabe verantwortlich
- Vergabe auf Basis einzelner Objekte
  - z. B. Datei-Zugriffsrechte unter UNIX (*chmod*)
- Festlegung Objekt-bezogener Sicherheitseigenschaften
- ★ keine System-globalen Sicherheitseigenschaften möglich / festlegbar
  - Abhängigkeiten zwischen Objekten werden bei Festlegung der Zugriffsrechte nicht zwangsläufig berücksichtigt
    - Benutzung, Kommunikation, Kooperation zwischen Objekten
      - ↳ Inkonsistenzen in der Gesamtstrategie (z. B. unbeabsichtigte Informationsflüsse)

## 2 Systembestimmte Zugriffskontrollstrategien

---

### *Mandatory Access Control (MAC)*

- Systemweite Regeln dominieren über Benutzer-bestimmte Regeln
  - Zugriff wird verweigert, auch wenn ihn benutzerspezifische Regeln erlauben würden
  - Benutzer kann systemweite Regeln weiter einschränken

## 3 Rollenbasierte Zugriffskontrollstrategien

---

### *Role-Based Access Control (RBAC)*

- Rechtemanagement nicht auf Basis von Subjekten (Benutzern) sondern an den durchzuführenden Aufgaben (Rollen) orientiert
- Subjekte erhalten Rechte aufgrund der Rolle, die sie gerade ausführen
  - ggf. zeitlich variabel

## G.4 Zugriffskontrollmodelle

### 1 Zugriffsmatrix-Modell (vgl. Abschnitt D.3)

- Schutzzustand des Systems wird durch Matrix beschrieben
  - Spalten = Objekte  $O$
  - Zeilen = Subjekte  $S$
  - Festlegung von Rechten  $R$  zu jedem Tupel  $(s, o)$
  
- statische Zugriffsmatrix
  - Modellierung von Anwendungsproblemen mit a priori bekanntem Rechtezustand
  - z. B. für Sicherheitsstrategien einfacher Router (zustandslose Paketfilter-Firewalls)

	Port 21 (ftp)	Port 25 (smtp)	Port 53 (DNS)	Port 514 (rsh)	> 1023
ftp-Server	receive		send		
Mail-Rechner		receive			send
externer Rechner		send	send		



# 1 Zugriffsmatrix-Modell (2)

## ■ dynamische Zugriffsmatrix

- Veränderungen der Matrix durch Operationen  
(z. B. `create/destroy` von Objekten oder Subjekten,  
`enter/delete` von Rechten)

## ■ Sicherheitseigenschaften

- Formalisierung von Schutzzuständen ermöglicht Untersuchung von Sicherheitseigenschaften der modellierten Systeme

## ◆ Safety-Problem

$r \in M_t(s,o)$  : In Schutzzustand  $M_t$  hat Subjekt  $s$  das Recht  $r$  an Objekt  $o$

- Objekt  $o$  hat das Recht  $r$  nicht.  
Ist sicher, dass es das Recht in Zukunft nicht erlangen kann?  
 $r \notin M_{t'}(s,o)$ .  $\nexists M_{t'}$  mit  $t' > t$  und  $r \in M_{t'}(s,o)$  ?
- allgemein nicht entscheidbar,  
für konkrete Fälle meist aber Entscheidung möglich

# 1 Zugriffsmatrix-Modell (3)

- ... Sicherheitseigenschaften
  - ◆ Soll-Ist-Vergleiche
    - erfüllt ein modelliertes System (Ist-Eigenschaften) die gestellten Anforderungen (Soll-Eigenschaften)
    - Verifikation bei formaler Spezifikation der Eigenschaften möglich
  - ◆ Modellierung von Zugriffsrechten
    - Modellierung konkreter Zugriffsrecht-Konzepten (z. B. UNIX-Rechte) durch Kommandos zur Modifikation einer Zugriffsmatrix
    - Basis für formale Überprüfung von Konzepten (vgl. Eckert, IT-Sicherheit 5. Auflage, Seite 241 ff)
  - ◆ Einsatz von Zugriffsmatrix-Modellen
    - erlauben feingranulare Spezifikation von Objekten und Subjekten
    - nur für kleine Modelle handhabbar
    - Überprüfung von Eigenschaften mit Hilfe von Werkzeugen möglich (Basis: Model-Checking-Ansatz)

## 2 Rollenbasierte Modelle

■ Definition:  $RBAC = (S, O, RL, P, sr, pr, session)$

$S$  endliche Menge der Benutzer des Systems

$O$  endliche Menge der zu schützenden Objekte

$RL$  endliche Menge von Rollen. Jede Rolle beschreibt eine Aufgabe und damit die Berechtigungen der Rollenmitglieder

$P$  endliche Menge der Zugriffsberechtigungen

$sr$  Relation der Rollenmitgliedschaft  $sr: S \rightarrow 2^{RL}$

$sr(s) = \{R_1, \dots, R_n\} \Rightarrow$  Subjekt  $s$  darf in den Rollen  $R_1 \dots R_n$  aktiv sein

$pr$  Berechtigungsrelation  $pr: RL \rightarrow 2^P$

ordnet jeder Rolle  $R \in RL$  die Zugriffsrechte zu, die ihre Mitglieder während ihrer Rollenaktivität wahrnehmen dürfen

$session$  Relation  $session \subseteq S \times 2^{RL}$  beschreibt Sitzungen  $(s, rl)$  mit  $rl \subseteq sr(s)$

Für ein Subjekt  $s$  und eine Sitzung  $(s, rl) \in session$  gilt:

$s$  besitzt alle Rechte der Rollen  $R \in rl$

Sitzung  $(s, rl)$ :  $s$  ist aktiv in den Rollen  $R \in rl$

## 2 Rollenbasierte Modelle (2)

### ■ Beispiel: Bank

- ▶ Rollen:  $RL = \{Zweigstellenleiter, Kassierer, Kundenbetreuer, Kunde\}$
- ▶ Objekte:  $O = \{Kundenkonten, Kreditdaten, Kundendaten\}$
- ▶ Zugriffsrechte:  $P = \{Konto sperren, Kreditrahmen festlegen, Einzahlung, \dots\}$
- ▶ Subjekte:  $S = \{Huber, Meier\}$
- ▶ Huber ist Zeigstellenleiter, Meier ist Kundenbetreuer, beide sind auch Kunden ihrer eigenen Bank  
 $sr(Huber) = \{Zweigstellenleiter, Kunde\}$   
 $sr(Meier) = \{Kundenbetreuer, Kunde\}$
- ▶ Beispiel für Rechtevergabe  
 $\{Konto sperren, Kreditrahmen festlegen\} \subseteq pr(Zweigstellenleiter)$   
 $\{Einzahlung, Auszahlung\} \subseteq pr(Kunde)$

## 2 Rollenbasierte Modelle (3)

### ■ Hierarchische RBAC

- ▶ Aufgaben und Zuständigkeiten sind in der Praxis häufig hierarchisch geordnet — z. B.
  - Zweigstellenleiter  $\geq$  Kassenprüfer
  - Kassenprüfer  $\geq$  Kundenbetreuer
  - Kassenprüfer  $\geq$  Kassierer
- ▶ Problem: durch Rollenhierarchie erbt z. B. Kassenprüfer alle Rechte von Kassierer, obwohl er diese (z. B. Ein- und Auszahlung auf Kundenkonten) für seine Aufgabe nicht benötigt

### ■ Beschränkte Rollenmitgliedschaft

- ◆ Regeln zur Aufgabentrennung (*separation of duty*) beschreiben den wechselseitigen Ausschluss von Rollenmitgliedschaften
  - ▶ statische Aufgabentrennung  $SSD \subseteq RL \times RL$  mit  $(R_i, R_j) \in SSD \Leftrightarrow$  gleichzeitige Mitgliedschaft in den Rollen  $R_i$  und  $R_j$  ist ausgeschlossen (z. B. Kassenprüfer darf nicht Kassierer sein)
  - ▶ dynamische Aufgabentrennung  $DSD \subseteq RL \times RL$  mit  $(R_i, R_j) \in DSD \Leftrightarrow$  gleichzeitige Aktivität in den Rollen  $R_i$  und  $R_j$  ist unzulässig

### 3 Chinese-Wall-Modell (Brewer-Nash-Modell)

---

- Hintergrund: unzulässige Ausnutzung von Insiderwissen bei Bank- und Börsentransaktionen verhindern
  - weiteres Einsatzfeld: Informationsfluss zwischen konkurrierenden Unternehmen in Unternehmensberatungen verhindern
  
- Idee: zukünftige Zugriffsmöglichkeiten eines Subjekts können durch seine Zugriffe in der Vergangenheit beschränkt werden
  - Basis: Zugriffsmatrix-Modell beschreibt die grundlegenden Rechte
  - Zugriffshistorie wird aufgezeichnet
  - Zugriffsentscheidungen auf Basis der Zugriffsmatrix + Zugriffshistorie
  
- Probleme:
  - Regeln meist zu restriktiv
  - Modell erfasst nur den Schutz von Vertraulichkeit, nicht aber Datenintegrität

## 4 Bell - La Padula - Modell

- erstes vollständig formalisiertes Sicherheitsmodell
  - 1973 im Auftrag der US Air Force entwickelt
- Basis: dynamisches Zugriffsmatrix-Modell
  - $R = \{read-only, append, execute, read-write, control\}$
  - (*control* erlaubt Rechteweitergabe und -rücknahme)
- **Multi-Level-Security (MLS)** - Modell
  - Subjekte und Objekte werden mit einer Sicherheitsstufe markiert
- regelt den Fluss von Informationen
  - ◆ Simple-Security-Regel (*no-read-up* Regel)
    - ein Subjekt auf Sicherheitsstufe  $k$  kann nur Objekte von Sicherheitsstufe  $\leq k$  lesen oder ausführen
  - ◆ \*-Regel (*no-write-down* Regel)
    - ein Subjekt auf Sicherheitsstufe  $k$  kann nur auf Objekte von Sicherheitsstufe  $\geq k$  schreiben

## 4 Bell - La Padula - Modell (2)

---

- sichert nur Datenvertraulichkeit, nicht aber Datenintegrität!

## 5 Biba - Modell

---

- umgekehrte Eigenschaften zum Schutz von Datenintegrität
  - ◆ Simple-Integrity-Regel und Integrity-\*-Regel
- ➔ einfache MLS-Modelle in der Praxis ungenügend
  - ◆ aber konzeptionelle Basis für heutige MAC-Systeme
  - ◆ Einsatz des Bell-LaPadula-Modells mit Erweiterungen in heutigen Betriebssystemen
    - SELinux
    - AppArmor
    - z/OS (IBM), Solaris 10 mit Trusted Extensions



# G.5 Beispiel SELinux

---

## 1 Überblick

---

### ■ Erweiterung von Linux um flexiblen MAC-Mechanismus

#### ► *Type Enforcement*

- jedem Subjekt und Objekt ist ein *Security Context = user : role : type* zugeordnet
- Objektzugriff erfordert Autorisierung des Subjekt-Typs (= *Domain*) für den Objekt-Typ – unabhängig von dem tatsächlichen Benutzer
- Autorisierung erfolgt auf Basis einer *SELinux Policy* (Datei, die alle Regeln des SELinux-Kerns enthält)
- Flexible Regeln durch rollenbasierte Zugriffskontrolle
- zusätzliche MLS durch Definition mehrere Sicherheitsebenen möglich (entsprechend Bell-LaPadula-Modell)

### ■ *Type Enforcement* erfolgt zusätzlich zu den herkömmlichen UNIX-Rechteprüfungen

## 2 Zugriffskontrolle durch Type Enforcement

- Zugriffe in SELinux müssen explizit erlaubt werden
  - Default: kein Zugriff (unabhängig von den Linux-Rechten!)
    - ↳ kein Superuser!
  - ◆ allow-Regeln spezifizieren den Zugriff eines Subjekt-Typs (Domain) auf einen Objekt-Typ
 

<i>source type(s)</i>	Domain eines Prozesses der auf Objekt zugreifen möchte
<i>target type(s)</i>	Typ eines Objekts auf das der Prozess zugreifen soll
<i>object class(es)</i>	Die Objektklasse(n), für die der Zugriff erlaubt ist
<i>permission(s)</i>	
  
- Beispiel 1:
 

```
allow user_t bin_t : file {read, execute, getattr};
```

  - Prozess der Domain `user_t` dürfen auf `file`-Objekte des Typs `bin_t` mit den Rechten `read`, `execute` oder `getattr` zugreifen

## 3 Zugriffskontrolle durch Type Enforcement

- Beispiel 2: Realisierung des Programms /bin/passwd
  - ◆ benötigt Schreibrechte auf /etc/shadow um verschlüsselte Passwörter zu ersetzen
  - ◆ normale Vorgehensweise:
    - /bin/passwd hat root-s-Bit
    - /etc/shadow ist nur für root les- und schreibbar
    - Problem: alle anderen Programme die mit root-uid ausgeführt werden könnten ebenfalls /etc/shadow modifizieren
  - ◆ SELinux
    - Typ passwd\_t als Domain für /bin/passwd
    - Typ shadow\_t als Typ für /etc/shadow
    - Regel:

```
allow passwd_t shadow_t : file {ioctl read write create getattr setattr lock relabelfrom relabelto append unlink link rename}
```