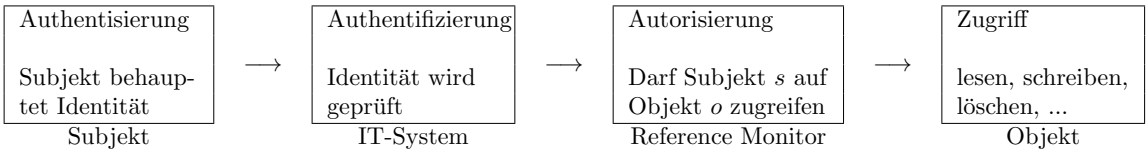
13 Zugriffskontrolle: Einleitung



Zugriffskontrolle: Wer darf im System wann auf welche Ressourcen zugreifen

- **Subjekt**: Initiator der Handlung
  - Benutzer (Alice, Bob), Gruppen, Prozesse, Rechner/Geräte
  - Genauer: Principals sind verschiedene Subjekte zugeordnet
- **Objekt**: Gegenstand der Handlung, Ressource
  - Dateien, Konto, Prozesse
    (Subjekte können auch Objekte sein)
- **Zugriffsrecht** (Access Right):
  - lesen, schreiben, löschen, hinzufügen
  - ausführen (z.B. Nutzung von Schlüsseln, ohne diese zu lesen)
  - zeit- oder ereignisabhängig

Reference Monitor: Setzt das Sicherheitsmodell durch

Umsetzungen: In Anwendung, Middleware, Betriebssystem, Hardware
(siehe übernächste Vorlesung)

Zugriffsmatrix Einfachste Form eines Sicherheitsmodells

- S: Menge der Subjekte
- O: Menge der Objekte
- R: Menge der Zugriffsrechte

- Zugriffsmatrix: $M = (m_{s,o})_{\substack{s \in S \\ o \in O}}$ mit $m_{s,o} \in \mathcal{P}(R)$

Subjekt s hat auf Objekt o Zugriffsrecht $r \in R$, wenn $r \in m_{so}$.

	Datei 1	Datei 2	Datei 3	Prozess 1
Nutzer 1	{read, write}		{read}	
Nutzer 2				{suspend}
Nutzer 3		{execute}		
Nutzer 4	{read}			

Matrix häufig dünn besetzt (vollständige Speicherung nicht notwendig)

- Access Control List: Speicherung Rechte der Subjekte beim Objekt
  Spalten von M: ACL(Datei1) = ((Nutzer1, {read, write}), (Nutzer2, {read}))
- Capabilities: Speicherung der Rechte an Objekte beim Subjekt
  Zeilen von M: CList(Nutzer3) = (Datei2, {execute})

Vorteile: Transparent, leicht zu verifizieren
Nachteile: Schlecht skalierbar, Änderungen schwer umsetzbar

Mögliche Abstraktionen

- Gruppen, Zuständigkeitsbereiche, Rollen, Sicherheitslevel
- negative Zugriffsrechte
- Protection Ring: Level von Zugriffsrechten

  0 Operating System Kernel
  1 Operating System
  2 Services
  3 User Processes

Wir unterscheiden:

- statische Zugriffsmatrix: Matrix ändert sich nicht
  z.B. in Firewalls (später)
- dynamische Zugriffsmatrix: Matrix ändert sich
  Klassische Betriebssysteme

Harrison-Ruzzo-Ullman (HRU)

Notwendige Verallgemeinerung: Änderung der Zugriffsmatrix bei

- neuen Subjekten und Objekten
- Änderung von Zugriffsrechten

Änderungen basieren auf sechs Basisoperatoren

- enter $r \in R$ to or delete $r \in R$ from m_{so}
- create or delete subject $s \in S$
- create or delete object $o \in O$

Typischer Aufbau von Kommandos:

- Abhängig von erlaubten Zugriffen (if ... then)
- werden Basisoperationen ausgeführt

$M \xrightarrow{c} M'$: Kommando c überführt M in M'

Beispiel.

command createFile (s,f)
  create f
  enter owner into m_{sf}
  enter read into m_{sf}
  enter write into m_{sf}
end

command grantReadWrite (s,p,f)
  if owner in m_{sf} then
    enter read into m_{pf}
    enter write into m_{pf}
  end if
end

Frage: Welche Konsequenzen hat dies auf die Sicherheit?
Formalisierung des Modells zum Nachweis der Sicherheitseigenschaften

Definition 13.1. Sei $r \in R$ ein Zugriffsrecht. Eine Matrix M ist unsicher in Bezug auf r, wenn es es ein Kommando c mit $r \notin m_{so} \xrightarrow{c} m'_{so} \ni r$.

Hinweis: Bedeutet nicht, dass das Modell unsicher ist
Inhaber der Datei könnte Zugriffsrecht hinzufügen (siehe Beispiel oben)

Aber: Kann Recht r auch von einem anderen Subjekt hinzugefügt werden?

Beispiel. Inhaber 1 ändert ownerchip auf Inhaber 2. Dieser fügt Zugriffsrecht hinzu und gibt ownerchip zurück.

Formalisierung: Sicherheitsmodell als Transitionssystem

- Mengen von Rechten R (endlich), Subjekten S und Objekten O
- endliche Menge von Kommandos C
- Wir interpretieren Matrizen $M = (m_{so})_{\substack{s \in S' \\ o \in O'}}, S' \subseteq S, O' \subseteq O$ als Zustände
- Kommandos c definieren die Übergangsfunktion $(M \xrightarrow{c} M')$
- Anfangszustand: Zugriffsmatrix bei Initialisierung des Systems

Definition 13.2. Eine Zugriffsmatrix ist sicher in Bezug auf ein Zugriffsrecht r, wenn keine Folge von Kommandos r hinzufügen kann.

Harrison, Ruzzo, Ullman (1976):

Satz 13.3 (Safety-Problem)**.** *Gegeben ein Sicherheitsmodell, eine (initiale) Zugriffsmatrix M und ein Zugriffsrecht r. Die Entscheidung, ob M sicher in Bezug auf r ist, ist unentscheidbar.*

Theorem bedeutet nicht, dass Sicherheitsmodelle nicht verifizierbar sind.
Es gibt aber kein allgemeines Verfahren dafür.

Folgerungen: Sicherheitsmodelle müssen einfach sein, um sie zu verifiziert.

Satz 13.4. *Das Safety-Problem ist für Sicherheitsmodelle entscheidbar, in denen jedes Kommando genau eine Basisoperation ausführt (mono-operational).*

Proof. Idee: Für jede Regel r und jede Zugriffsmatrix M_0 gilt:
Die Länge einer minimale Folge $M_0 \xrightarrow{c_1} M_1 \xrightarrow{c_2} \cdots \xrightarrow{c_t} M_t$, die das Recht r hinzufügt, ist beschränkt in Anzahl Rechte, Subjekte und Objekte von M_0.

Dazu zeigt man zunächst (liegt an der geforderten Minimalität):

1. $c_1, \ldots, c_t$ sind keine delete-Kommandos
2. Höchstens c_1 kann ein create-Kommando sein

Wir müssen also nur zwei Fälle unterscheiden:

- c_1 ist create-Kommando und $c_2, \ldots, c_t$ sind enter-Kommandos:
  - c_1 kann Anzahl der Objekte oder Subjekte um 1 erhöhen.
  - Für M_1 gilt also $|S_1| \cdot |O_1| \leq (|S_0| + 1)(|O_0| + 1)$.
  - In jedes $m_{so} \in M_1$ können nur $|R|$ Rechte hinzugefügt werden.
  - Also: $t - 1 \leq |R| \cdot (|S_0| + 1)(|O_0| + 1)$.
- c_1 ist enter-Kommando und $c_2, \ldots, c_t$ sind enter-Kommandos:
  - In jedes $m_{so} \in M_0$ können nur $|R|$ Rechte hinzugefügt werden.
  - Also: $t \leq |R| \cdot |S_0| \cdot |O_0|$ Schritten hinzugefügt.

Im schlimmsten Fall: $t \leq |R| \cdot (|S_0| + 1)(|O_0| + 1) + 1$

Beweis zu 1. und 2. in Übungsaufgabe. □

Mono-operational Sicherheitsmodelle sind uninteressant:
Anlegen eines Objektes nur ohne Festlegung von Zugriffsrechten möglich

Weitere mögliche Einschränkungen:

- Monotonic: Kein Löschen von Zugriffsrechten aus der Matrix
- $n-$Conditional: Höchstens n Bedingungen in if ... then

Entscheidbar: $1-$conditional monotonic
Schon unentscheidbar: $2-$conditional monotonic Sicherheitsmodelle