

Kapitel 5 Zugriffskontrolle, Rechteverwaltung

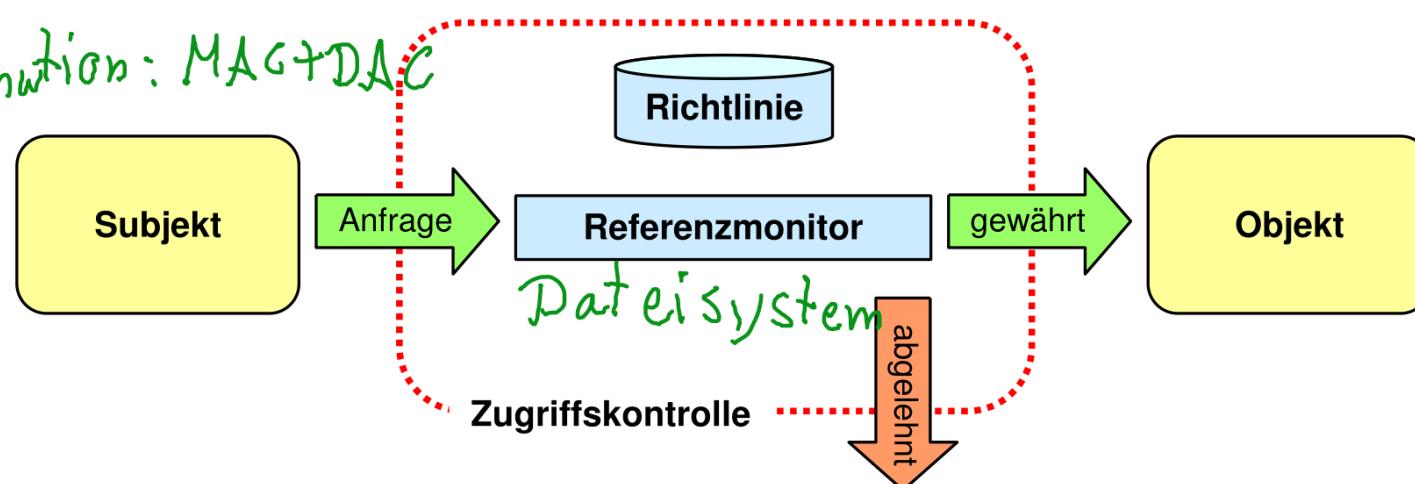
5.1 Sicherheitsmodelle

- Abstraktion von realen Gegebenheiten, Vereinfachung

Zugriffskontroll-Ansätze

- Discretionary Access Control (DAC), Benutzerspezifisch als Variante: Role Based Access Control (RBAC)
- Mandatory Access Control (MAC) oder systembestimmt
- Rule based Access Control

↳ Kombination: MAC+DAC
(XACML)



5.1.1 Zugriffsmatrix-Modell (ZM) (Access-Matrix-Model)

Einfaches Modell zur Beschreibung der Zugriffsrechte

- (Dynamische) Menge von Objekten O_t , z.B. Datei, Port, Record
- (Dynamische) Menge von Subjekten S_t mit: $S_t \subseteq O_t$ z.B. Nutzer
- Menge von Rechten R , z.B. r,w,x, send, receive,...
- Zugriffsmatrix $M_t : S_t \times O_t \rightarrow 2^R$, Schutz-Zustand zur Zeit t

Objekte

M_t	Datei1	Datei2	Datei3	Prozess1	Prozess2
Prozess1	{ read, write }		{ read, write }		{ send, receive }
Prozess2				{ send, receive }	
Prozess3		{ owner, execute }		{ signal }	

Subjekte

Fazit: Zugriffsmatrix-Modell

Positiv:

- Sehr einfaches, intuitives Modell
- Einfach zu implementieren

Negativ:

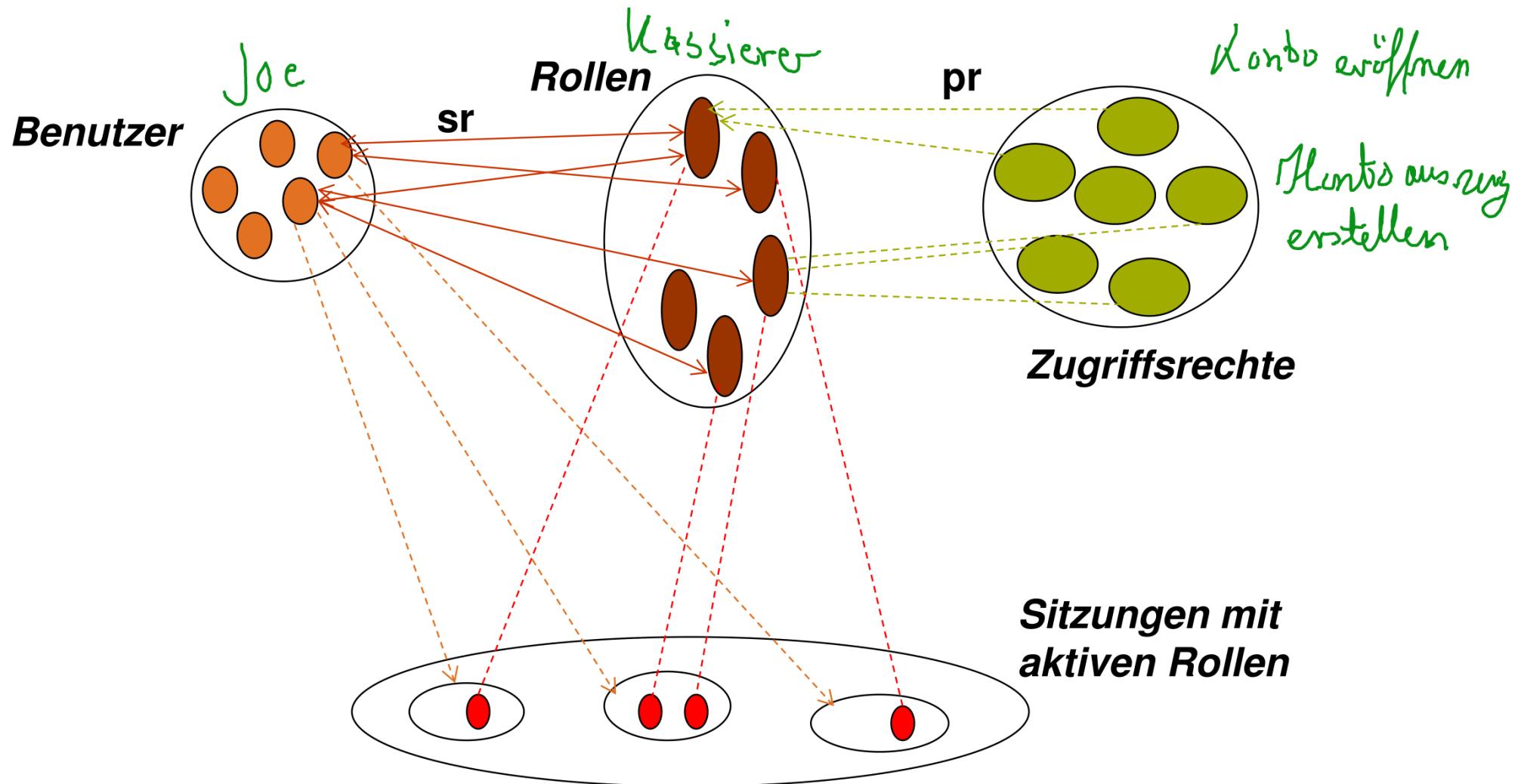
- Skaliert schlecht bei dynamischen Mengen von Subjekten, z.B. Web-Umfeld
- Rechtevergabe ist an Subjekte, nicht an deren Aufgaben orientiert: Problem: Szenarien mit gleichen Subjekten, die über die Zeit wechselnde Aufgaben erfüllen, sind kaum modellierbar
- Ungeeignet für Unternehmens- oder Verwaltungsumgebungen

5.1.2 Role-based Access-Control (RBAC)

- 1992 Ferraiolo u. Kuhn, NIST RBAC Standard zusammen mit R. Sandhu <http://csrc.nist.gov/rbac/sandhu-ferraiolo-kuhn-00.pdf>
- **Ziel:** effektive, skalierende, effiziente Rechteverwaltung
- **Lösung:** **Aufgabenorientierte Rechtevergabe** durch Rollen
- **Rolle:** beschreibt eine Aufgabe bzw. die damit verbundenen
 - Verantwortlichkeiten, Pflichten und Berechtigungen
- Nachbilden von **Organisationsstrukturen in Unternehmen**
 - Rechte und Verantwortlichkeiten sind häufig direkt aus den Organigrammen der HR Abteilungen ableitbar
- Erfüllen der Prinzipien: **need-to-know**, **separation-of-duty** *Benutzer*
- Weit verbreitet: u.a. Betriebssysteme, ERP, CMS, ... *nie 2 Rollen gleichzeitig ausfüllen*

Komponenten eines (einfachen) RBAC-Modells

- Menge von Subjekten = **Benutzer** z.B. Krankenhaus:
 - Oberarzt
 - Zivi
 - Krankenschwester
 - Verwaltungsangestellte
- Menge von Rollen **Role**, Rolle $r \in Role$
- Menge von **Zugriffsrechten** P (permission) für Objekte
- Zwei Abbildungen:
 - (1) Benutzer-Rollenzuordnung $sr: S \rightarrow 2^{Role}$ *hohe Dynamik*
 - (2) Rechte-Rollenzuordnung $pr: Role \rightarrow 2^P$ *sehr statisch*
- **Sitzung** $session \subseteq S \times 2^{Role}$, $(s, RL) \in session$, dann ist RL die Menge der **aktiven Rollen** des Benutzers s , $RL \subseteq sr(s)$
 d.h. falls $R_i \in session(s)$ gilt, dann ist s in der Rolle R_i eingeloggt und ist aktiv in dieser Rolle



Beispiel: **Rollen** und deren Berechtigungen im Krankenhauszenario:

- **Behandelnde Ärzte** (auch AiP, PJ, zeitweise zugeordnete Ärzte):
 - ganze Patientenakte im Behandlungszusammenhang (außer besonders sensible Daten), (lesend, schreibend)
 - abteilungsinterne Daten aller Aufenthalte
- **Pflegekräfte**:
 - Zugriff auf Krankenakte; Umfang durch Abteilungsleiter festgelegt
- **Famulanten**, Studenten, Auszubildende:
Auszubildende Ärzte
 - erforderlicher Umfang durch verantwortlich Lehrenden festgelegt (im Rahmen seiner eigenen Befugnisse).
- **Verwaltungsmitarbeiter**:
 - Stammdaten, (lesend, schreibend)
 - abrechnungsrelevante Daten (u. U. auch besonders sensible!)

Rollenhierarchien

Ziel: Nachbilden hierarchischer Organisationsstrukturen

Definition der **Rechte-Vererbung**:

Gegeben sei die partielle Ordnung \leq auf der Menge der Rollen. Dann gilt: R_i erbt alle Rechte von R_j falls gilt:

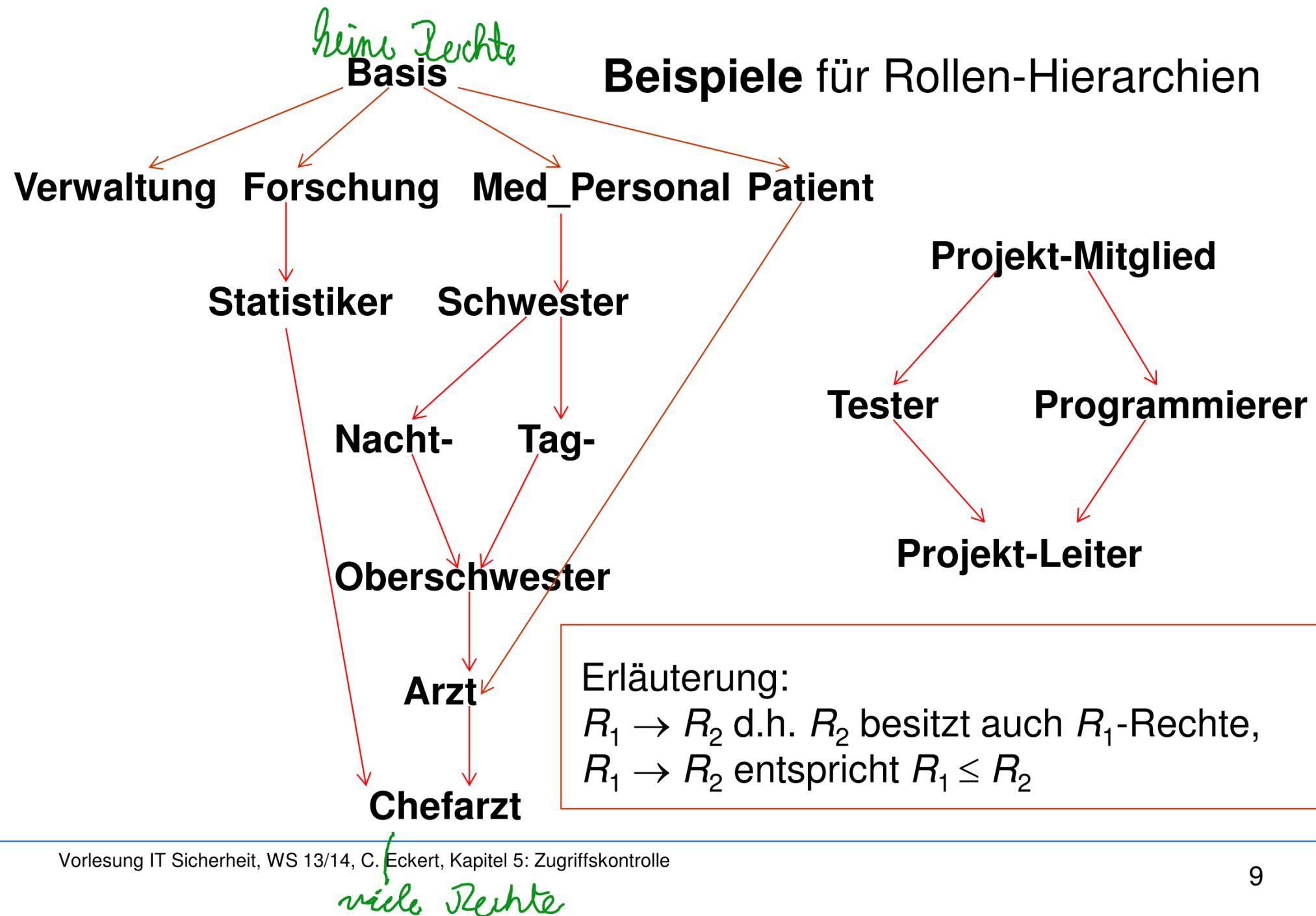
$$R_j \leq R_i, \quad R_i, R_j \in \text{Role}$$

Vererbung der Rollenmitgliedschaft für Subjekte s :

falls $R_j \leq R_i$, dann gilt: $\forall s \in S : R_i \in sr(s) \Rightarrow R_j \in sr(s)$

Rechtevererbung ist aber nicht unproblematisch! **Wieso?**

*Zu viele Rechte?
Nichtung need-to-know
separation of duty*



Umsetzung der RBAC-Eigenschaften:

(1) Erweitertes Log-in: Prüfen auf Rollenmitgliedschaft

Ein Subjekt darf nur in solchen Rollen **aktiv** sein, in denen es **Mitglied** ist: $\forall s \in S : R_j \in \text{session}(s) \Rightarrow R_j \in sr(s)$

(2) Erweiterte Zugriffskontrolle:

Sei $\text{exec} : S \times P \rightarrow \text{Boolean}$ definiert, mit

$\text{exec}(s,p) = \text{true} \Leftrightarrow s \text{ ist zu } p \text{ berechtigt}$

Dann muss bei Zugriffsversuchen von $s \in S$ geprüft werden:

$\text{exec}(s,p) = \text{true, genau dann, wenn } \exists R_i \in \text{Role} \text{ mit}$

$R_i \in \text{session}(s) \wedge p \in pr(R_i) \vee \exists R_j \wedge R_j \leq R_i \wedge p \in pr(R_j)$

aktive Rolle

Vererbung von Rechten

Beschränkungen (Constraint RBAC)

Ziel: Regeln, die die Rollenmitgliedschaft beschränken

(1) **Statische Aufgabentrennung** (separation of duty):

- Wechselseitiger Ausschluss von Rollenmitgliedschaften:
Festlegen einer Relation $SSD \subseteq Role \times Role$, mit
 $(R_i, R_j) \in SSD \Leftrightarrow R_i$ und R_j sind **wechselseitig ausgeschlossen**
- Sei $Member(R_i) = \{s \mid s \in S \wedge R_i \in sr(s)\}$
Beschränkungsregel: $\forall R_i, R_j \in Role, \forall s \in S :$
 $(s \in Member(R_i) \wedge s \in Member(R_j)) \Rightarrow (R_i, R_j) \notin SSD$

Beispiel: Bank-Szenario, Rollen = {Kassierer, Kassenprüfer}

Die Aufgaben müssen wechselseitig ausgeschlossen sein:

$(Kassierer, Kassenprüfer) \in SSD$

(2) **Dynamische Aufgabentrennung:**

→ mitgliedschaft prinzipiell erlaubt

Wechselseitiger Ausschluss von Rollenaktivitäten

- Festlegen einer Relation $DSD \subseteq Role \times Role$, mit $(R_i, R_j) \in DSD \Leftrightarrow R_i$ und R_j sind dynamisch w.A.

- Beschränkungsregel: Für jedes Subjekt s sei:

$$Active(s) = \{ R_i \mid \exists RL \subseteq Role \wedge (s, RL) \in session \wedge R_i \in RL \}, \quad \left| \begin{array}{l} \text{aktive} \\ \text{Rollen} \end{array} \right.$$

dann muss gelten:

$$(s \in Member(R_i) \wedge s \in Member(R_j) \wedge \{R_i, R_j\} \subseteq Active(s)) \Rightarrow (R_i, R_j) \notin DSD$$

Beispiel: R1 = Kundenbetreuer; R2 = Konto_Besitzer;

Beschränkung: $(R1, R2) \in DSD$

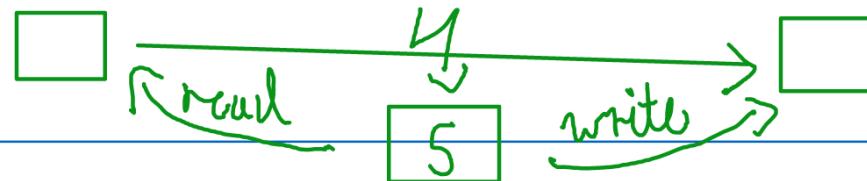
Fazit RBAC

Positiv?

- + Skalierbarkeit, Dynamiz.
- + Nachbildung von (Unternehmens-) Strukturen
- + mehrere Rollen möglich
- + Verbindung am Workflows: intuitiv, need-to-know

Negativ?

- widersprüchliche Rechtevergabe (Bsp Bank / Konto)
- Rückzug von Rollen mitgliedschaften
- Tendenz zu vielen Rollen



5.1.3 Bell-LaPadula-Modell (Bell, LaPadula 1973) (BLP)

Ziel: einfache Regeln zur Beschränkung von Informationsflüssen
↳ erstes vollständig formalisiertes Modell

Lösung: Multi-level Security (MLS), Labeling und MAC-Regeln

BLP-Modell:



- erstes formalisiertes Modell: Zertifizierung: Orange Book, CC
- Integriert in viele Systeme: u.a. SE-Linux, Free BSD Varianten

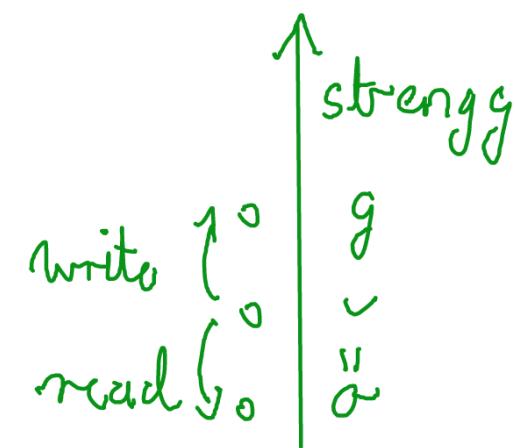
Komponenten des BLP- Modell (nur Auszug)

- Zugriffsrechte (wie beim ZM): \rightarrow DAC
 $R = \{ \text{read-only}, \text{append}, \text{execute}, \text{read-write}, \text{control} \}$
erweitern aber nicht lesen \rightarrow Rechtenweitergabe
- Menge von Sicherheitsklassen SC , $X \in SC$ mit $X = (A, B)$,
 A ist Sicherheitsmarke (label), z.B. vertraulich, geheim
 B Menge von Kategorien (compartments), z.B. Arzt

- $\forall s \in S$: Clearance: $SC(s) \in SC$; Maximale $SC(s)_{MAX}$ und aktuelle $SC(s)_{AKT}$
- $\forall o \in O$: Classification $SC(o) \in SC$
- Partielle Ordnung auf SC : (SC, \leq) , für $X, Y \in SC$ gilt:
 $X = (A, B)$, $Y = (A', B')$ $X \leq Y \Leftrightarrow \begin{array}{l} A \leq A' \wedge B \subseteq B' \\ \text{„öffentliche < vertrauliche < geheim < streng geheim“} \end{array}$ *Labels* *Kategorie*

BLP-MAC-Policy (mandatory access control)

- **Regeln**: no-read-up, no-write-down, tranquility
- Intuitive Interpretation der Regeln:
 - Festlegen einer part. Ordnung über Sicherheitsmarken
 - Informationsflüsse sind nur entlang der Ordnung zulässig, von unten nach oben



geheim + read \rightarrow File 1 + $sc(s) < sc(o) = h$

(S)

öffentlich + write \rightarrow File 2 + $sc(s) > sc(o) = l$

Simple-security-Property (no-read-up-Regel):

Für $z \in \{\text{read - only, execute}\} : z \in M_t(s, o) \wedge SC(s) \geq SC(o)$

Informationsfluss nur von oben nach unten

*-Property (no-write-down-Regel):

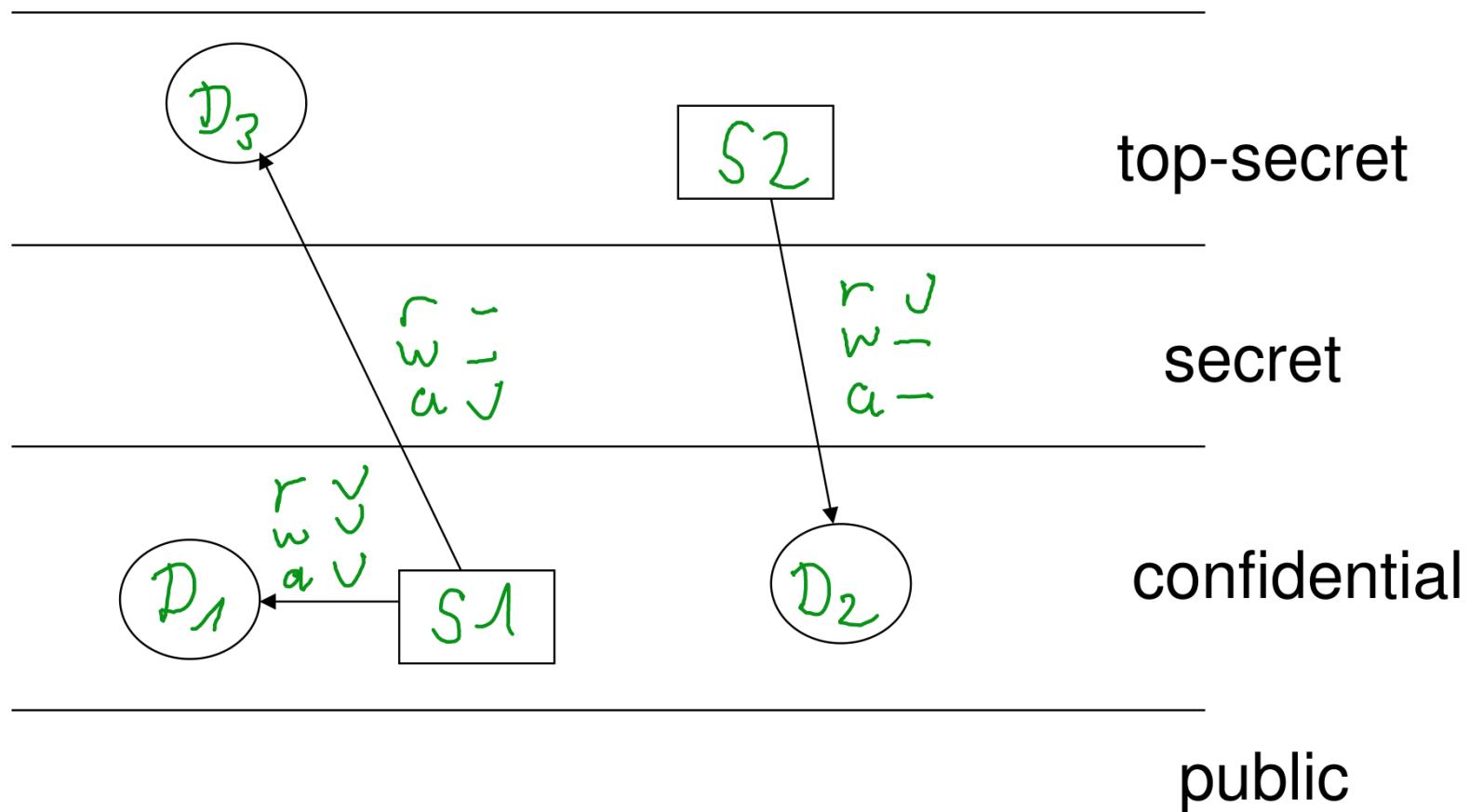
- append $\in M_t(s, o) \wedge SC(s) \leq SC(o)$; und
- read-write $\in M_t(s, o) \wedge SC(s) = SC(o)$

Wicht Informationen
von „oben“ nach „unten“
schreiben

Ggf. Strong-Tranquility-Regel: während der Systemlaufzeit
keine Änderung der Subjekt-Clearance oder der
Objekt-Classification

Beispiel: BLP-Bedingungen: erlaubte Zugriffe?

$sel(s) \leq s_{ub}(b)$
 $r \geq$
 $w \leq$
 $a \leq$
 $x \geq$
 Plausw.?



Zulässige Informationsflüsse

○ Objekt

□ Subjekt

Grenzen des BLP-Modells: u.a.

- (1) Problem: Information/Objekte werden **sukzessive immer höher eingestuft**. **Warum tritt das Phänomen auf?**

Subjektbetrüter \rightarrow Gruppenleiter \rightarrow Manager
Klim-Datenanpassend \rightarrow Kontrolle \rightarrow Kontrolle

- (2) Einführung von **Trusted Subjects** (u.a. Systemprozesse), Sie unterliegen nicht den BLP-Regeln, sie dürfen Sicherheitseinstufungen ändern (zurückstufen), **Problem?**

Dotklassifizieren der Objekte schwierig
 \Rightarrow Umplementation schwierig

- (3) Problem des **Blinden Schreibens**: s darf o modifizieren, aber anschließend (wegen no-read-up) nicht lesen!

Problem?

Unterschrift nicht überprüfbar

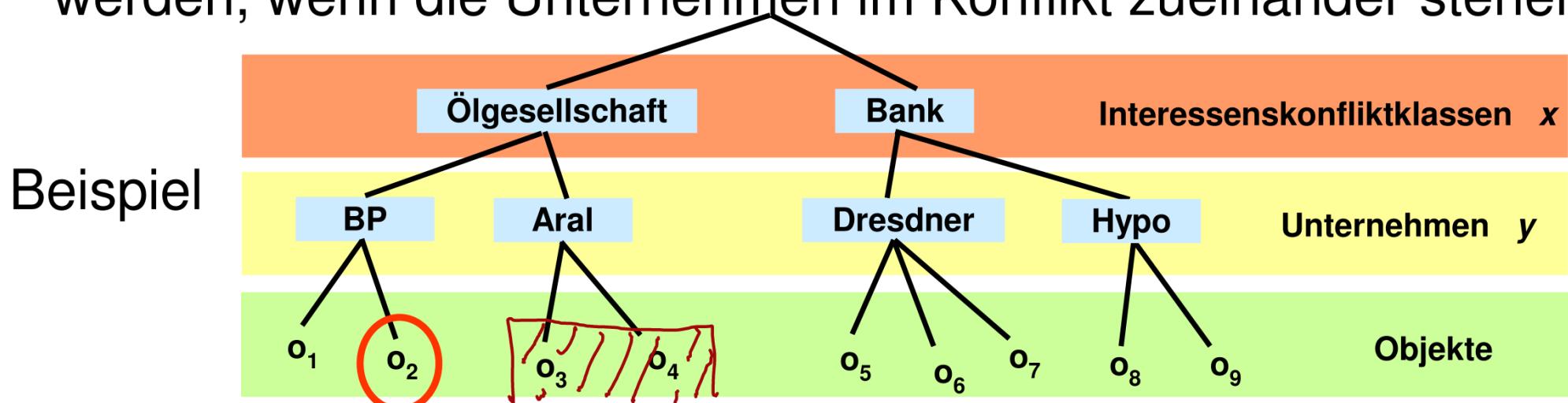
5.1.4 Chinese Wall Modell, Brewer-Nash, 1989

- **Interessenskonflikt-Klassen**
- Zugriffe sind abhängig von der **Zugriffshistorie**



COI (Conflict of Interest)-Regel:

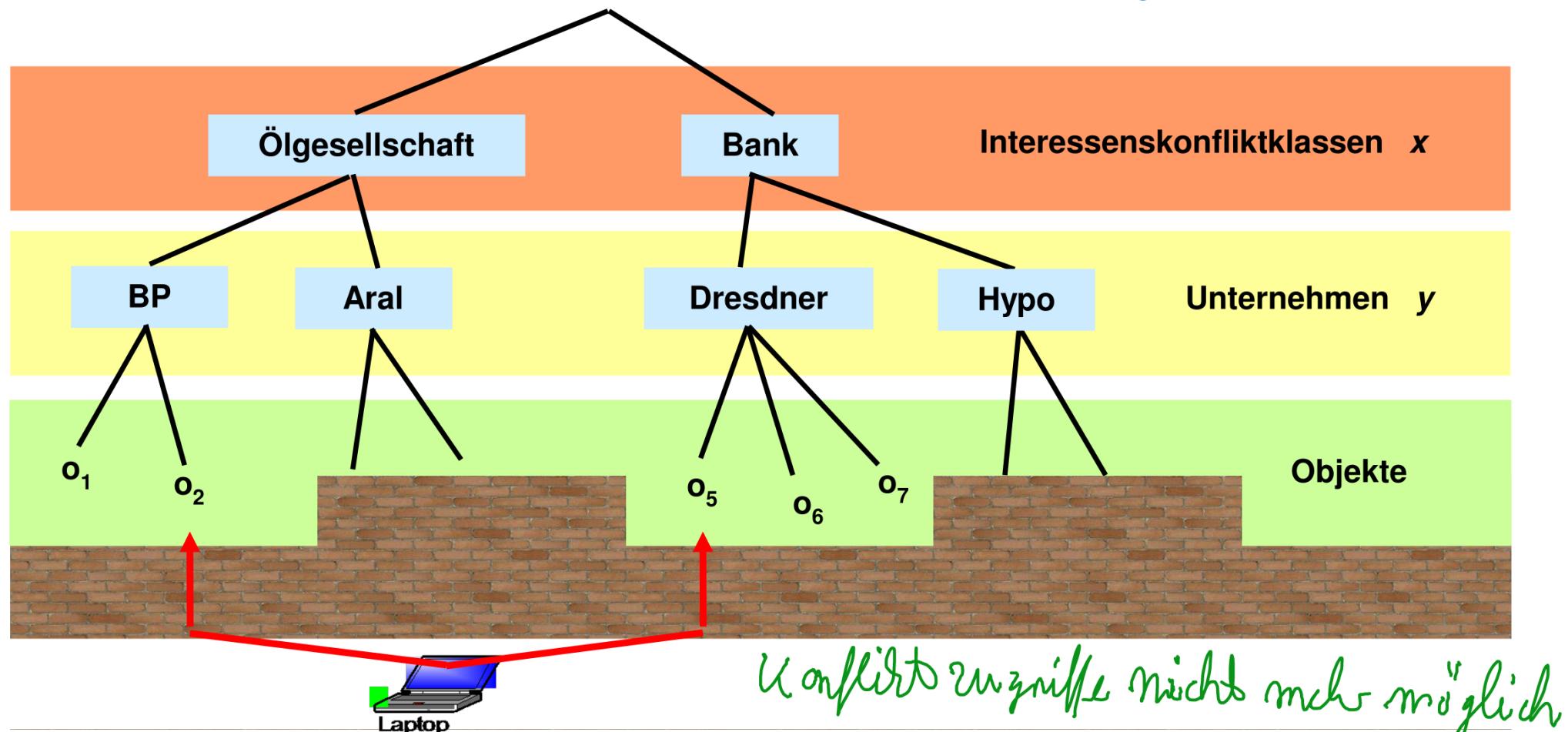
- Nach einem Zugriff auf Objekte eines Unternehmens U darf nicht mehr auf Objekte eines Unternehmens $U \neq U$ zugegriffen werden, wenn die Unternehmen im Konflikt zueinander stehen.



Zugriff: Subjekt s_1 auf das Objekt o_2 (zum Zeitpunkt t')

Nach Zugriff (Zugriffskontext-abhängig) wird eine ‚Zugriffsmauer‘ aufgebaut:

Beispiel: Mauerbau nach Zugriff auf o_2 und o_5



Konzepte des Chinese-Wall-Modells

Objekte

- Zweistufige Objekthierarchie
 - Einteilung in unterschiedliche Konfliktklassen K (Branchen)
z.B.: Banken und Ölgesellschaften
 - Unternehmen werden Konfliktklassen zugeordnet
z.B.: Dresdner und HypoVereins-Bank der Klasse Bank

$o \in$ Objekte und deren Klassifikation

$y(o)$: Unternehmen, zu dem das Objekt gehört

$x(o)$: Interessenkonfliktklasse von o

y_0 : Öffentliches Objekt/Information, sanitized

x_0 : Interessensfreie Information *frei von kritischen
Informationen*

Schutzzustand: Besteht aus zwei Teilen

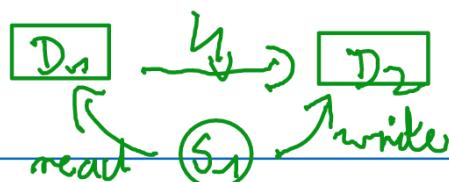
- M_t Zugriffsmatrix \rightarrow DAC
 - Benutzerbestimmbare, generische Rechte
 $R = \{ \text{Read}, \text{Write}, \text{Execute} \}$
- N_t Zugriffshistorie

Zugriffshistorie

- $N_t: S \times O \rightarrow 2^R$ mit $N_t(s, o) = \{r_1, \dots, r_n\}$, falls
 Subjekt s mit den Rechten $\{r_1, \dots, r_n\}$ auf Objekt o vor dem
 Zeitpunkt t zugegriffen hat

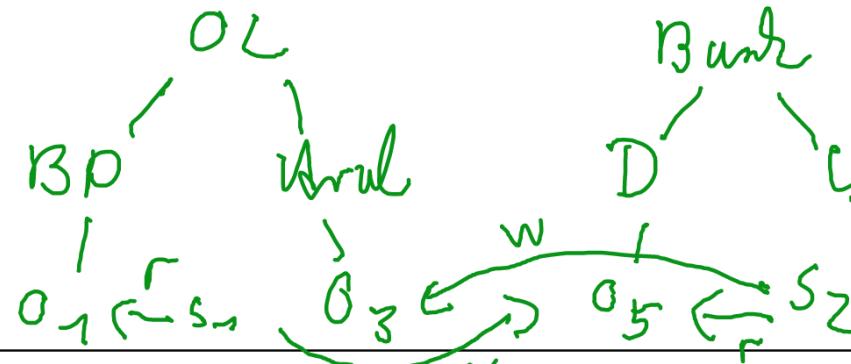
(1) Leseregel: Lesezugriff (Read-Access): Simple Property

- Leserecht, falls: $r \in M_t(s, o) \wedge \forall o' \in O, N_t(s, o') \neq \emptyset$,
 dann muss gelten: $(y(o') = y(o) \vee x(o') \neq x(o)) \vee$
 $y(o') = y_0 \vee y(o) = y_0$



Leseregel allein ist **nicht** ausreichend!

Beispiel:



(2) Schreibregel: Ein Subjekt $s \in S$ erlangt Schreibzugriff auf ein Objekt o zur Zeit t genau dann, wenn

$\text{write} \in M_t(s, o) \wedge \forall o' \in O \quad (1)$ und

$\text{read} \in N_t(s, o') \rightarrow (y(o') = y(o) \vee y(o') = y_0) \quad (2)$

(1) Unternehmenseigene Info
(2) öffentliche Information

Problem bei Chinese-Wall Modell

- Aufhebung von Interessenskonflikten nicht vorgesehen
- Historie wächst im Prinzip unbegrenzt
↳ sehr restriktiv

Sicherheitsmodelle für zukünftige IT Systeme?

Zur Erinnerung: Charakteristika der zukünftigen Systeme

- Heterogen, dynamisch, mobil, vertrauenswürdige und nicht-vertrauenswürdige Komponenten kooperieren, ...

Anforderung: u.a.

- **Vertrauensmodelle:**
 - Attribut-basierte Modelle, z.B. Identitätsbasiert,
 - Verhaltensbasiert (z.B. Reputation)
- **Kontext-abhängige Modelle**
 - Kontext-bewusste Zugriffskontrollen
 - Definition von Kontexten, Erfassen des Kontextes über Sensoren, erweiterte Kontrollen
 - **Beispiele?**

Attribut-basierte Beschreibung
↳ Kontext
↳ Attestation

5.2. Konzepte zur Umsetzung der Rechteverwaltung

Basis: Varianten einer Zugriffsmatrix

- idR. dünn besetzt (sparse)
- Implementierung als Tabelle ist ineffizient

Subjekt-Sicht: **Capability-Listen** →

Gängige Konzepte

- Zugriffskontrolllisten
- Capability-Listen
- Domain-Type-Enforcement

ACL

Subjekte	Objekte	
	Datei 1	Datei 2
Bill	owner, r,w	w
Joe	r,x	
Alice		owner, r,x

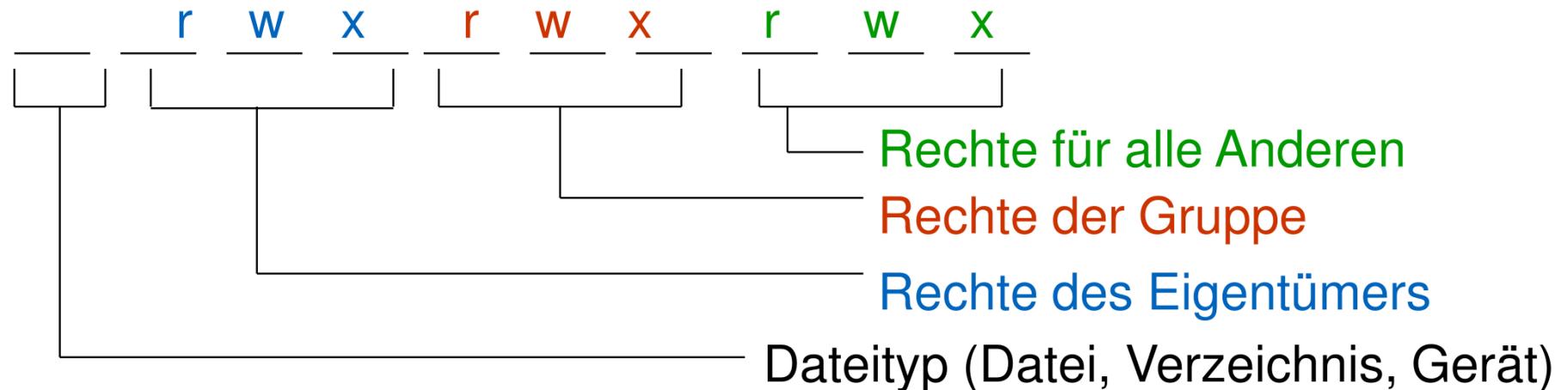
↑
Objekt-Sicht: **Zugriffskontrollliste**

5.2.1 Zugriffskontrollliste: Access-Control-List (ACL)

- **Spaltenweise** Realisierung der Matrix
- ACLs: am häufigsten eingesetztes Konzept in klassischen BS u.a. in UNIX/Linux, Windows-Varianten, MAC-OS, Android
- Eine ACL ist eine geschützte Datenstruktur des BS

Beispiel: ACL unter Unix/Linux (Android): vereinfachte ACL

- Rechtevergabe nur an: **Eigentümer**, **Gruppe**, **Rest der Welt**



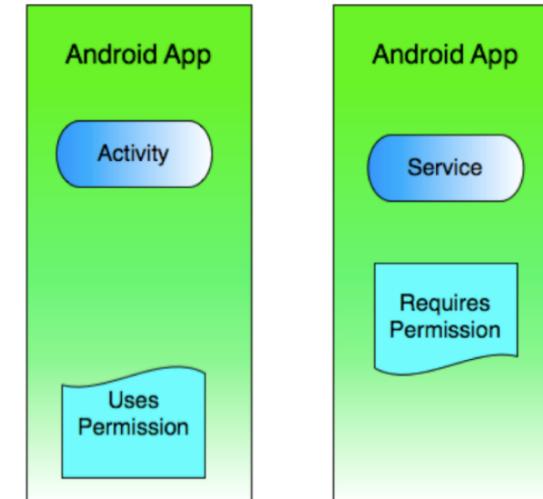
Beispiel: Android

- Verwendung der **Linux-Zugriffskontrolle** (UID, GID, rwx-Bits)
 - Jede App erhält bei der Installation eigene UIDs \rightarrow *dynamisch*
 - Seit Version 4.2 Multiuser Support (ux_ayy)

$$\text{UID} = \langle \text{USER} \rangle^* 100000 + (\langle \text{App} \rangle + 10000)$$
 - Dateien sind standardmäßig nur für den User les- und schreibbar (Zugriffsbits für "world" nicht gesetzt)

 \hookrightarrow Zugriffsbits nur bei Geräten
- **Feingranulare Privilegien** für Apps (und ihrer Komponenten)
 - > 194 verschiedene Zugriffsberechtigungen!

Flomylea

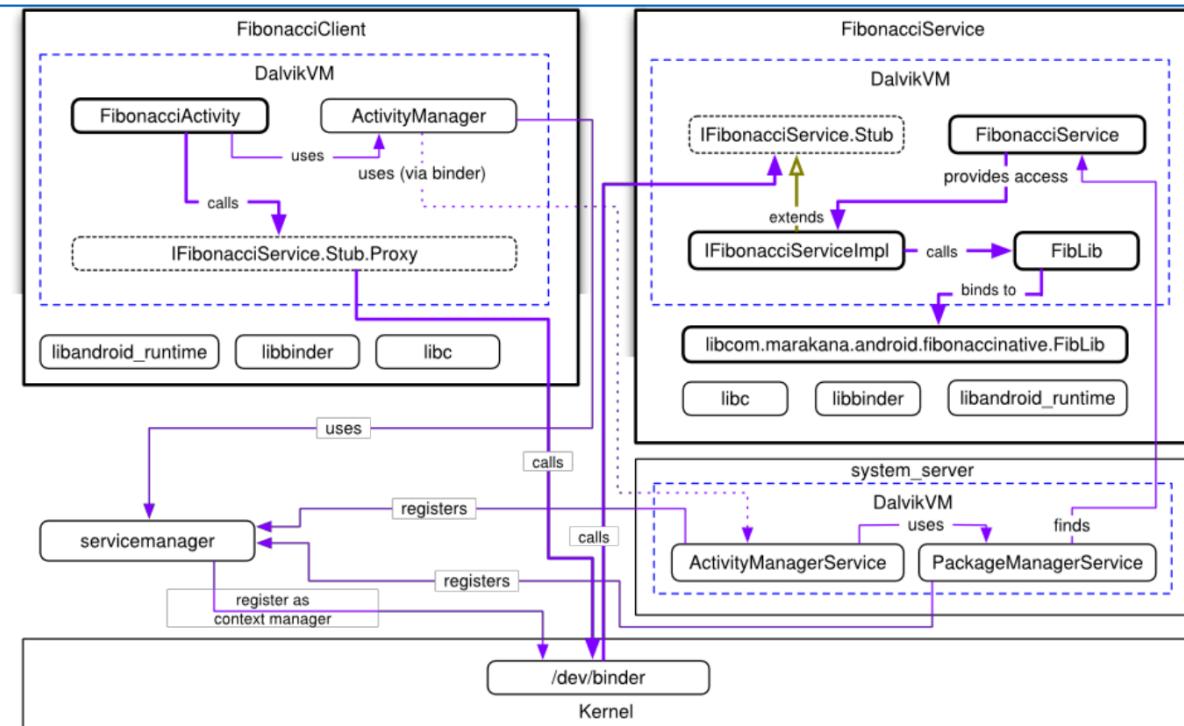


- Apps müssen global deklarieren, auf welche Komponenten sie **welche Zugriffe benötigen**:
 - z.B. Internet, GPS, Telefonbuch, ..., read, write, ..
 - Bei App-Installation sieht Anwender die angeforderten Rechte (oder Signatur)
 - Nutzer kann sie nur **komplett gewähren oder ablehnen**
 - Eine App erhält keine Rechte, die nicht explizit angefordert wurden.
- Apps können **eigene Rechte** definieren, um anderen Apps Zugriff auf ihre Daten und/oder Dienste zu gewähren.
- **Zugriffsrechte:** 
 - Permission-Labels, Eigene Labels möglich

INTERNET
↓
INET
↓
3d03
↓
char
inet 'create' 28

Vier Stufen

- **Normal:** Keine Rückfrage an den Benutzer
- **Dangerous:** Rückfrage
- **Signature:** mit gleichem Schlüssel signierte Anwendungen



SignatureOrSystem: Signature und für Systemanwendungen

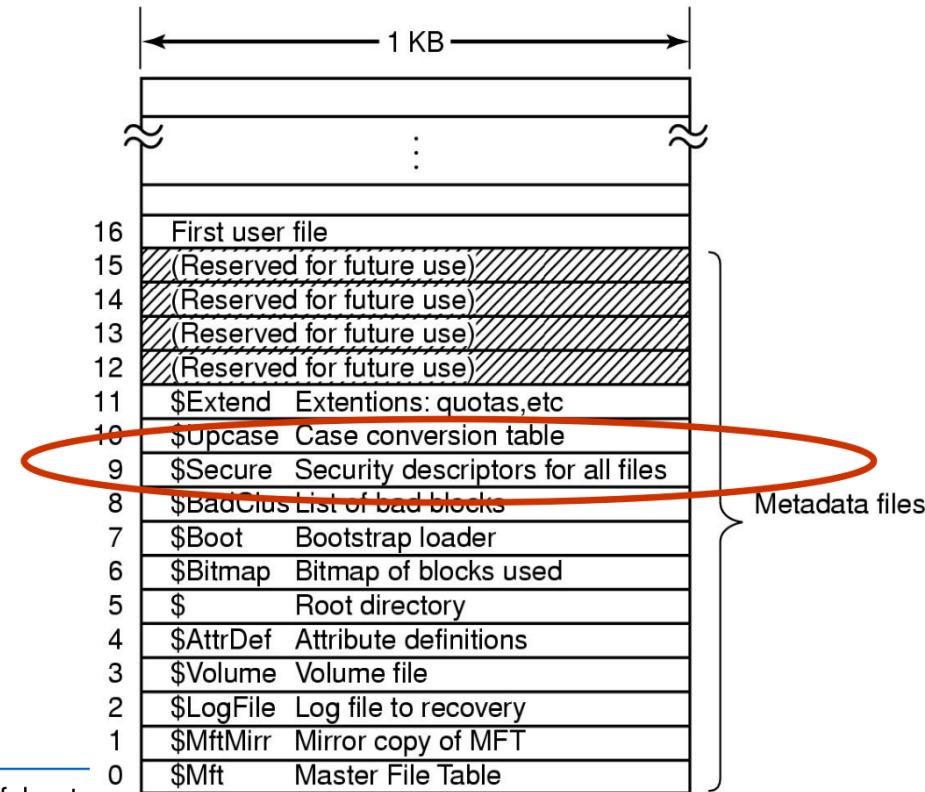
Nutzerinteraktion:

- Bestätigung der Rechte von Apps **bei der Installation**.
- Ein späteres Entziehen von Rechten ist **nicht möglich**.
- Spezielle Rechte sind **temporär global deaktivierbar**.
 - z.B. Ortsinformationen Zugriff auf Funkschnittstellen

Weiteres Beispiel: Windows Vista/Windows 7

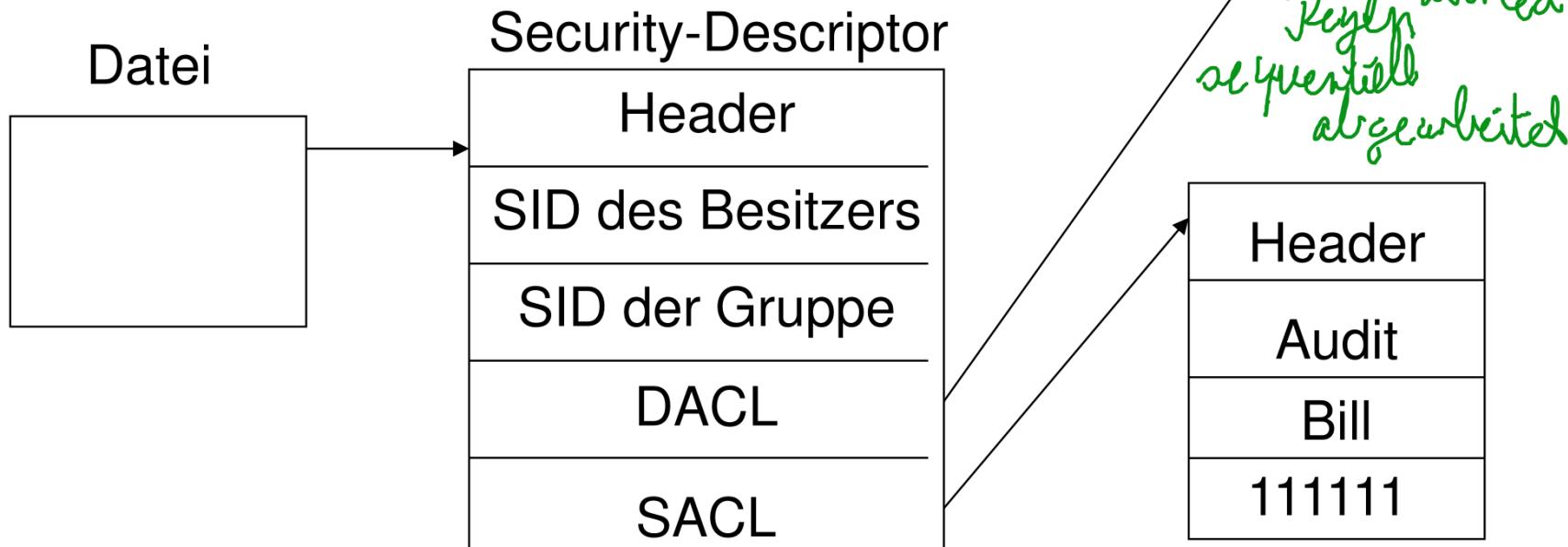
- differenzierte ACL mit **Berechtigungen bzw. Verboten** für einzelne Benutzer, aber auch für Gruppen
- Berechtigungen/Verbote in **Security-Descriptor** gespeichert
- Security-Deskriptoren in **Master-File-Table (MFT)** verwaltet

- Eintrag 0:
Adressen der MFT-Blöcke
- Eintrag 2: Log-Datei,
z.B. neue Verzeichnisse.
Strukturelle Änderungen
- Eintrag 9: **Security-
Deskriptoren**



Security-Descriptor enthält:

- SID = systemweit eindeutige Security-ID des Besitzers des Objekts und der Gruppe
- **DACL** = Discretionary ACL: Liste von ACEs
- ACE = Access-Control-Element mit allow/deny
- **SACL** = System-ACL, spezifiziert die Operationen, deren Nutzungen zu protokollieren sind



5.2.2 Capability-Konzept

Zeilenweise Realisierung der Matrix

→ Zugriffsticket

- Capability: **Zugriffsticket** mit Objekt-UID und Rechte-Bits
- Capability-Besitz berechtigt zur Wahrnehmung der Rechte
- Für jedes Subjekt s wird eine **Capability-Liste** verwaltet

Beispiel Clist (Joe) = ((Datei1, {r,x}), (Datei2, {w}))

Tickets kann wiedergenutzt werden, gerichtet an Subjekt

Vor- und Nachteile von ACLs und Capabilities?

ACLs

- + Rechte an Objekt effizient bestimmbar
- + einfache Rechtenübernahme
- + dezentrale Kontrolle möglich
- Bestimmen der Subjektrechte schwierig
- schlechte Skalierbarkeit (Subjekte)

Capabilities

- + Rechte für Subjekte einfach bestimmbar
- + Festlegen von Protection Domains möglich
- keine Subjekt-Ticket-Hopplung
Besitz \Rightarrow Wahrnehmung der Rechte
- Rechtenübernahme schwierig für Objekte

In heutigen Systemen häufig: Kombination aus beiden Ansätzen:

- ACL für ersten Zugriff, danach
- Ausstellen einer **Capability**:

File-Handle (Unix) bzw. Object-Descriptor (Windows)

$fd = \text{open}(<\text{name}>, <\text{rechte}>)$
 $= \text{real}(fd, \dots)$

$\Rightarrow \text{mmap}(fd)$ benötigt keine **Capability**

5.2.3 Domain-Type Enforcement, DTE

1995 vorgestellt, u.a. in Linux/Unix-Varianten umgesetzt

Lee Badger et. al *A Domain and Type Enforcement UNIX Prototype*,
Fifth USENIX UNIX Security Symposium Proceedings, 1995

Konzept: *Trennung von Subjekten in Gruppen: Domänen*
Trennung von Objekten in Gruppen: Typen

- vereinfachtes Capability und Rollen-Konzept, spezielle Mandatory Access Policies
- jedem Subjekt wird ~~zu einem Zeitpunkt~~ genau eine Domäne als Attribut zugeordnet (Domänenwechsel), z.B. *domain project*
- jedem zu schützenden Objekt wird ein Typ zugeordnet, z.B. *type budget*
- Die DTE Tabelle spezifiziert, auf welche Typen eine Domäne read und/oder write Zugriff besitzt

datei *Prozess*

- die **Domain-Transitions Tabelle** (DTT) spezifiziert, welche Domänen von einer gegebenen Domain aufgerufen werden dürfen:
root domain
 - Domänenwechsel über die Ausführung einer speziellen Operation: **enter**
*beim Ausführen eines Programms (fork)
→ separation of duty*
 - Domänen-Attribut für ein Subjekt ist eine Art Capability, das Subjekt gehört zur Domäne (vereinfachte Rollenmitgliedschaft) und besitzt die Rechte der Domäne
 - DTE definiert **mandatorische Zugriffsregeln**
Implementierung: *↳ systembestimmt*
 - Domänen-Attribut als Bestandteil des **Prozess-Deskriptors**
 - Typ-Attribut Bestandteil des **Objekt-Deskriptors** (z.B. inode)

Beispiel - DTE Linux Implementierung

- Prozesse werden in Domänen aufgeteilt
 - Ausführung des „Init“ Prozess erfolgt in einer Default Domäne
 - Domänenwechsel durch Ausführen eines Prozesses
- Dateien werden verschiedenen Typen zugeordnet
 - 3 Typen pro Datei (Inode):
 - etype: Typ der Datei / des Verzeichnis
 - rtype: Typ des Verzeichnis und seiner Kinder
 - utype: Typ der Kinder des Verzeichnisses

Init
↳ Appache
↳ log

for

Beispiel

- Domänen: `common_d`, `log_d`
 - Default Domäne: `common_d` (`rwx` → `root_t`, `r` → `log_t`)
 - `log_d` (`r` → `root_t`, `rw` → `log_t`, `x` → `log_xt`)
-  Transition von `common_d` → `log_d` durch Aufruf von `/usr/bin/rsyslogd`
- Typen: `root_t`, `log_t`, `log_xt`
 - Default Typ: `root_t`
 - utype von `/var/log` → `log_t`
 - etype von `/usr/bin/rsyslogd` → `log_xt`
- Auswirkungen:
 - Syslog Prozess darf nur Logdateien schreiben.
 - Syslog darf keine Datei ausführen, die er selbst schreiben darf
 - Syslog darf keine Programme mit Typ `root_t` ausführen
 - Insbesondere kein `/bin/bash` für eine mögliche Reverse Shell

Fallbeispiel: SE Linux (Security Enhanced)

- Ende 2000 als NSA Forschungsprojekt gestartet
- Nutzt Linux Security Module im Kernel (LSM)
- Ist in Version 2.6 des Linux Kernels integriert

Ziele:

- Kernel soll die Ausführung von Applikationen anhand von **Regeln** überwachen können
- Unterstützung von MAC, RBAC und **DTE**
- Subjekte (process) und Objekte (file) besitzen einen eigenen **Sicherheitskontext** zur Laufzeit
 - Für Objekte über Attribut beschrieben: ls -Z <filename>
 - Für Prozesse: ps axZ
 - Sicherheitskontext: user_r:role_r:type_t:mls-component

Policy Decision

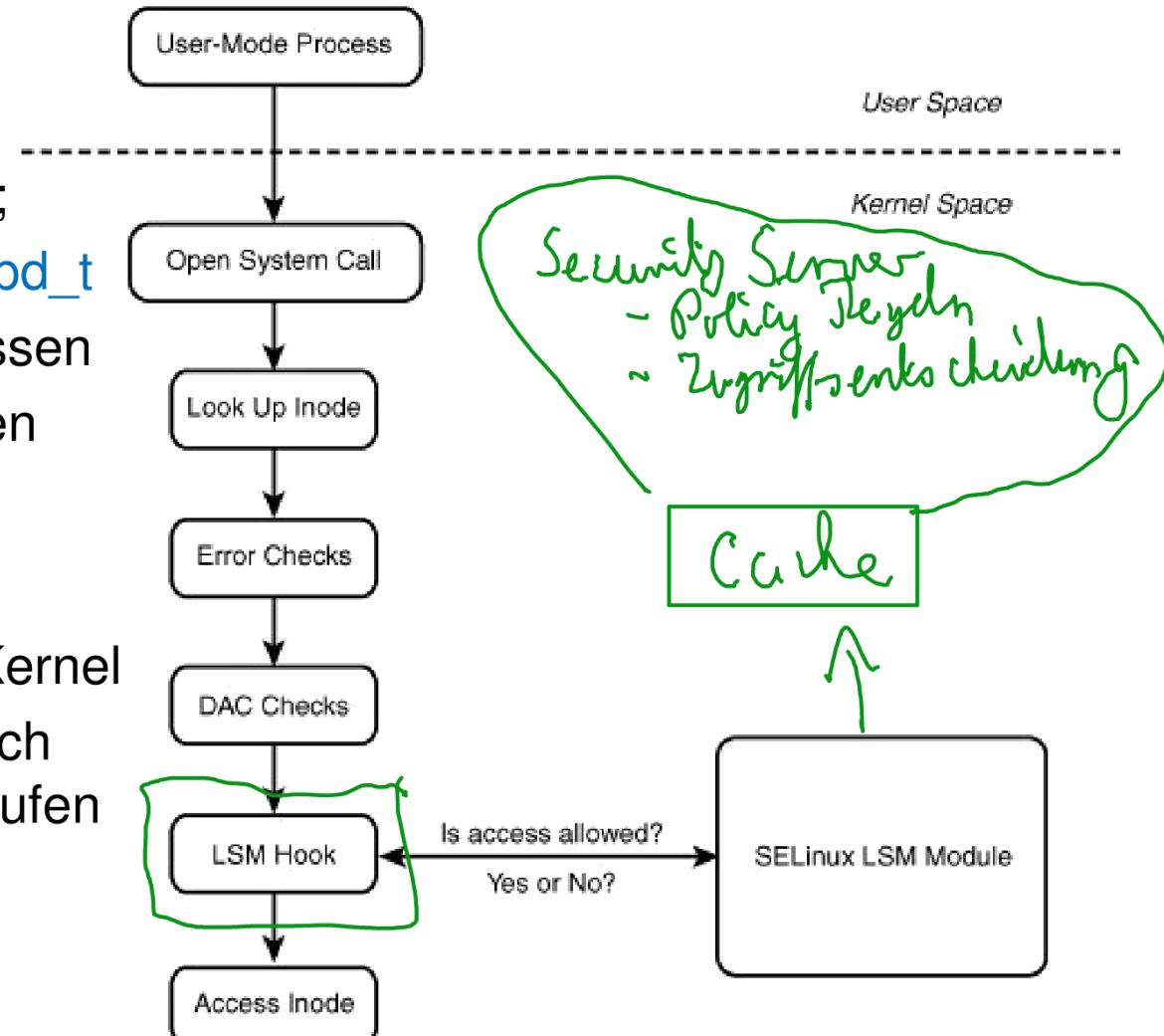
Struktur einer **Regel**: `allow type1_t type2_t:class { operation };`

Beispiel:

- `allow httpd_t httpd_log_t:dir create;`
D.h. Prozesse mit Kontext-Typ `httpd_t` dürfen neue Dateien in Verzeichnissen mit Kontext-Typ `httpd_log` erzeugen

Policy Enforcement

- Sicherheitsmodule erweitern den Kernel
- LSM Sicherheitschecks werden nach den Standard Linux Checks aufgerufen



Erläuterung: Linux Security Modules (LSM)

- Schnittstelle im Linux Kernel
- Jeder Syscall im Kernel ruft eine Callback Funktion aus dem Sicherheitsframework auf
 - Prüfung und evtl. Einschränkungen der Berechtigungen für den aktuellen Aufruf
 - Aktivierung des Frameworks durch Laden eines entsprechenden Moduls
- Aber: Schnittstelle könnte auch durch Angreifer genutzt werden
 - Erleichterter Einstieg in Kernel-interne Datenstrukturen
 - SELinux als umfangreichste, komplexeste Lösung
 - Alternativ: Apparmor, GRSecurity, Tomoyo Linux, Smack

5.3 Umsetzen der Zugriffskontrolle

Allgemeines Prinzip

- idR. Aufteilung in **Berechtigungs-** und **Zulässigkeitskontrolle**

Berechtigungskontrolle: PDP: **Policy Decision Point**

- Prüfung beim **erstmaligem** Zugriff auf ein Objekt
- von vertrauenswürdigen Systemdiensten (z.B. Dateisystem)
- Ausstellung einer **Bescheinigung**, z.B. File-Handle, Ticket

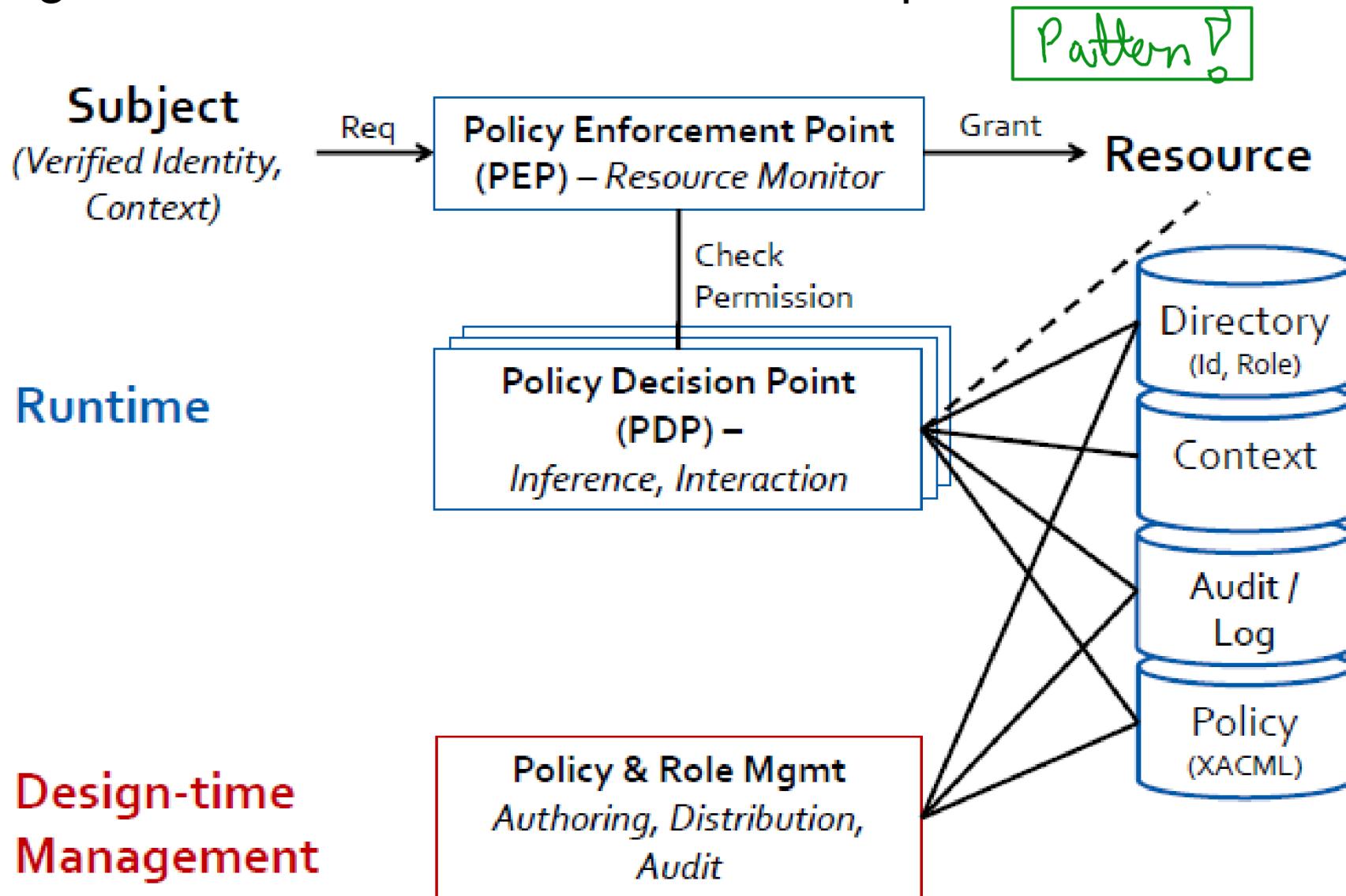
Zulässigkeitskontrolle: PEP: **Policy Enforcement Point**

- durch Objektverwalter (z.B. user-level Server)
- bei Objektzugriff: Prüfen der **Gültigkeit der Bescheinigung**
- **kein Zugriff** auf die Rechteinformation notwendig

Konsequenzen?

*- Weitergabe von Tickets
- Zürgung von Berechtigungen schwierig*

Allgemeines Schema für Zusammenspiel: PDP und PEP

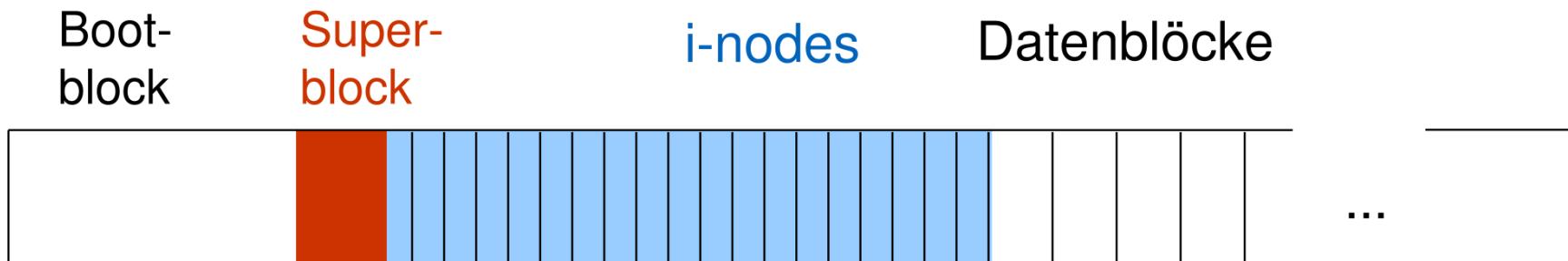


Beispiel Unix/Linux

- Subjekte/Prozesse identifiziert über **UID, GUID**
- Zu schützende Objekte: **Dateien, Verzeichnisse**
- Dateien/Verzeichnisse werden BS-intern über einen Datei-Deskriptor, die **i-node** (index-node), beschrieben

Datei-Deskriptor

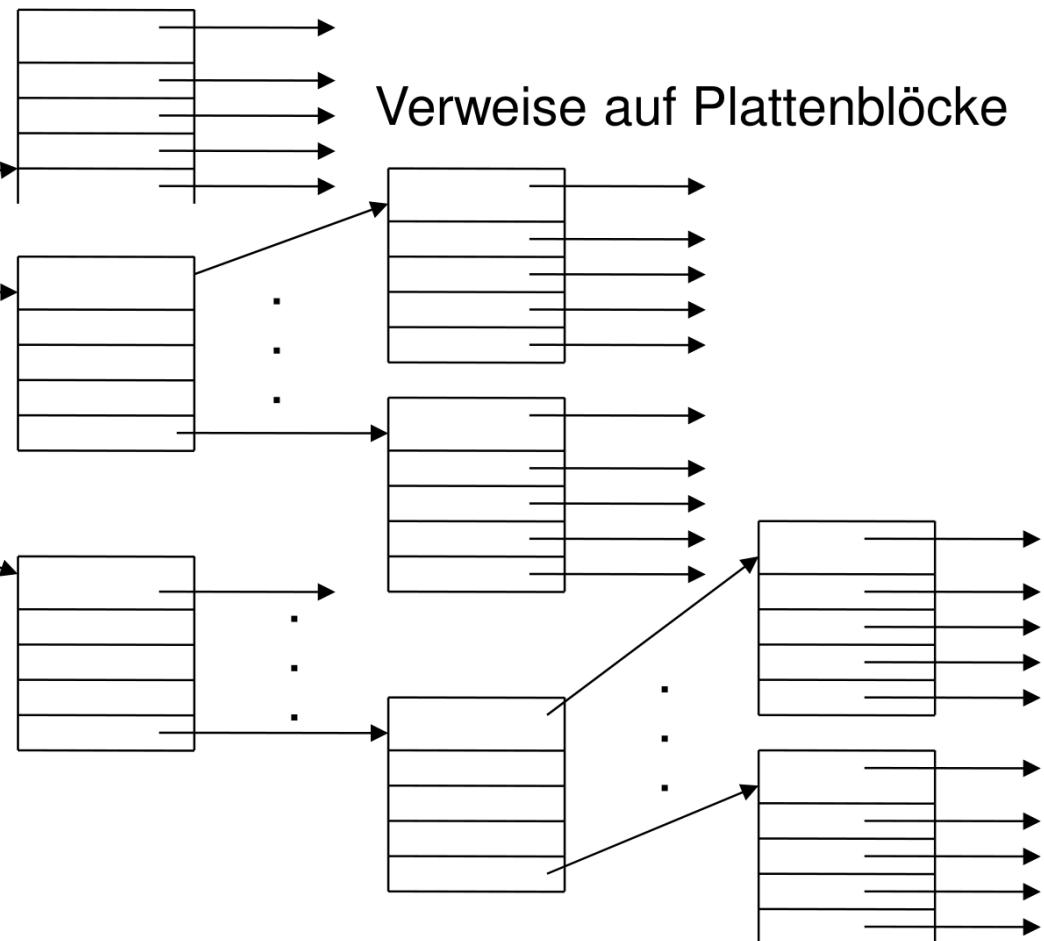
- die i-node enthält u.a. **Name des Datei-Owners** und die **ACL**
- die i-nodes werden **auf der Festplatte** verwaltet,
- beim Öffnen der Datei (open-call) wird i-node eingelagert



owner joe, uid
 group student, guid
 type regular file
perms rwxr-xr-x
 accessed Feb 12 1999 3:00 P.M.
 modified Feb 11 1999 10:16 A.M.
 Adressen der 10 ersten Plattenblöcke
 einfach indirekt
 zweifach indirekt
 dreifach indirekt

Unix-i-node

ACL als Bestandteil der i-node unter Unix



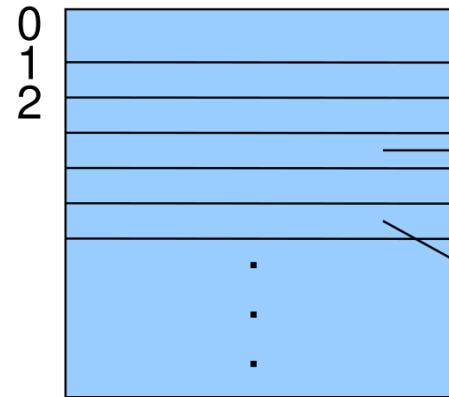
Ablauf bei der Zugriffskontrolle unter Unix/Linux

Open-System-Call: Angabe des Zwecks **r, w, x**

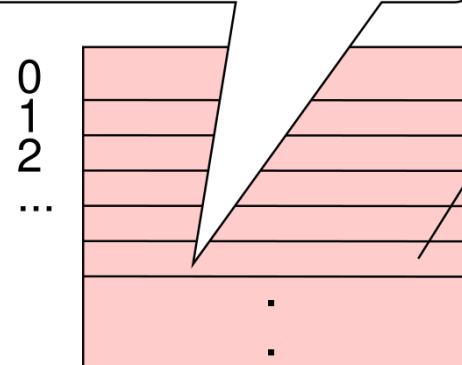
Aktionen des Unix Kerns (siehe nächste Folie)

- (1) Laden der i-node der zu öffnenden Datei <datei_i> in i-node Tabelle des Kernels
- (2) Prüfen, ob zugreifender Prozess gemäß der ACL der Datei zum gewünschten Zugriff **r, w, x** berechtigt ist
- (3) Falls o.k., return **File-Handle**: enthält Information über zulässige Zugriffsrecht **r, w, x**
 - Eintrag mit Rechten in Open File Tabelle des Kernels
 - Verweis in File-Descriptor-Tabelle des Prozesses auf Recht
- (4) **Zugriffe** auf geöffnete Datei <datei_i> mit File-Handle Dateisystem führt Zulässigkeitskontrolle durch.

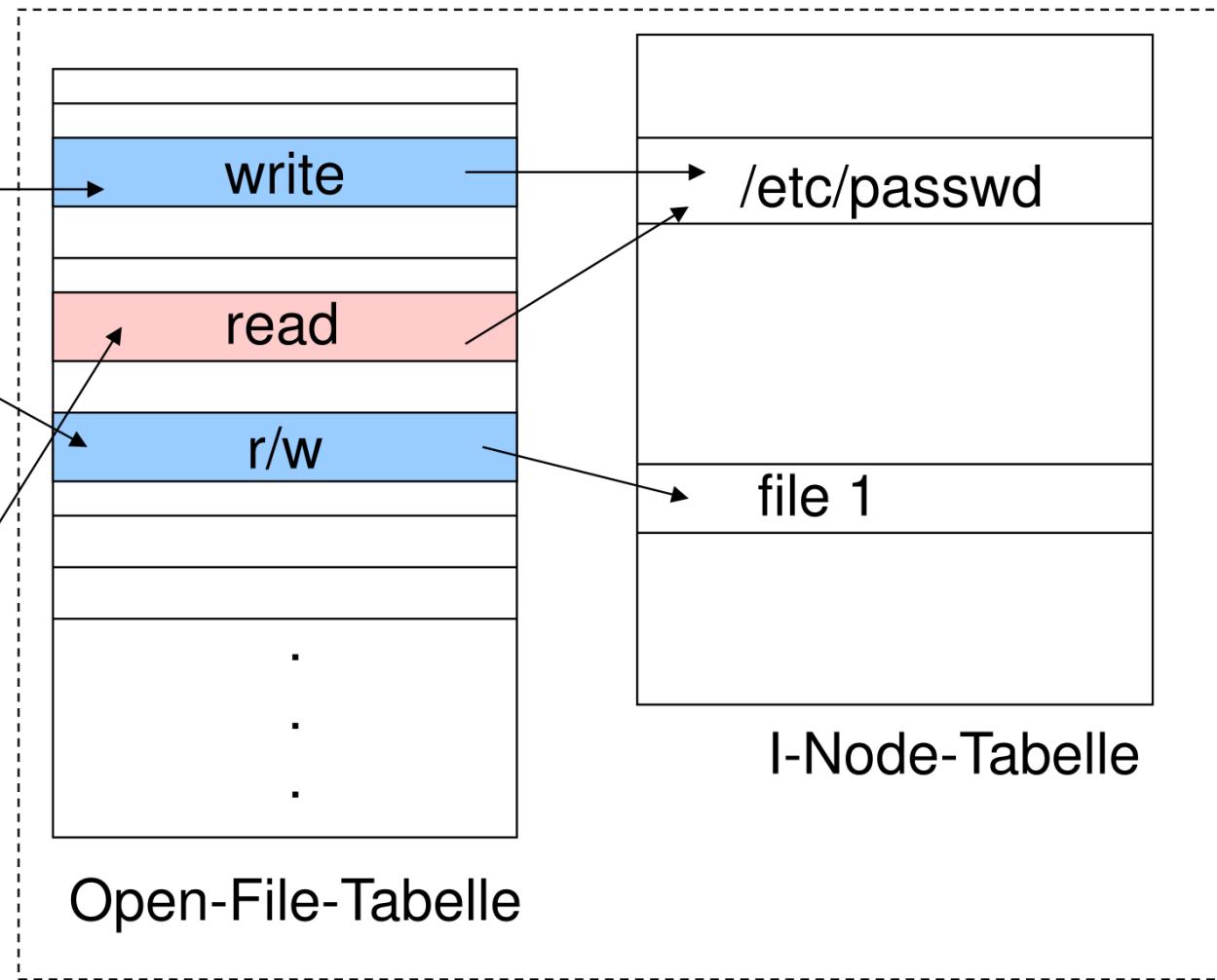
Prozess A
File-Descriptor-Tabelle für A



File-Handle zur lesenden Nutzung von /etc/passwd durch Prozess B



BS-Kerndatenstrukturen für Unix-Zugriffskontrolle:



File-Descriptor-Tabelle für B

(BS)-systemglobale Tabellen

5.4 Zugriffskontrolle in offenen Dienste-Plattformen

- **SAML**: XML-basiertes Framework zum Austausch von Sicherheitsinformationen in verteilten Umgebungen
- **XACML** zur Spezifikation von Policies

5.4.1 SAML V2.0 (Security-Assertion-Markup-Language)

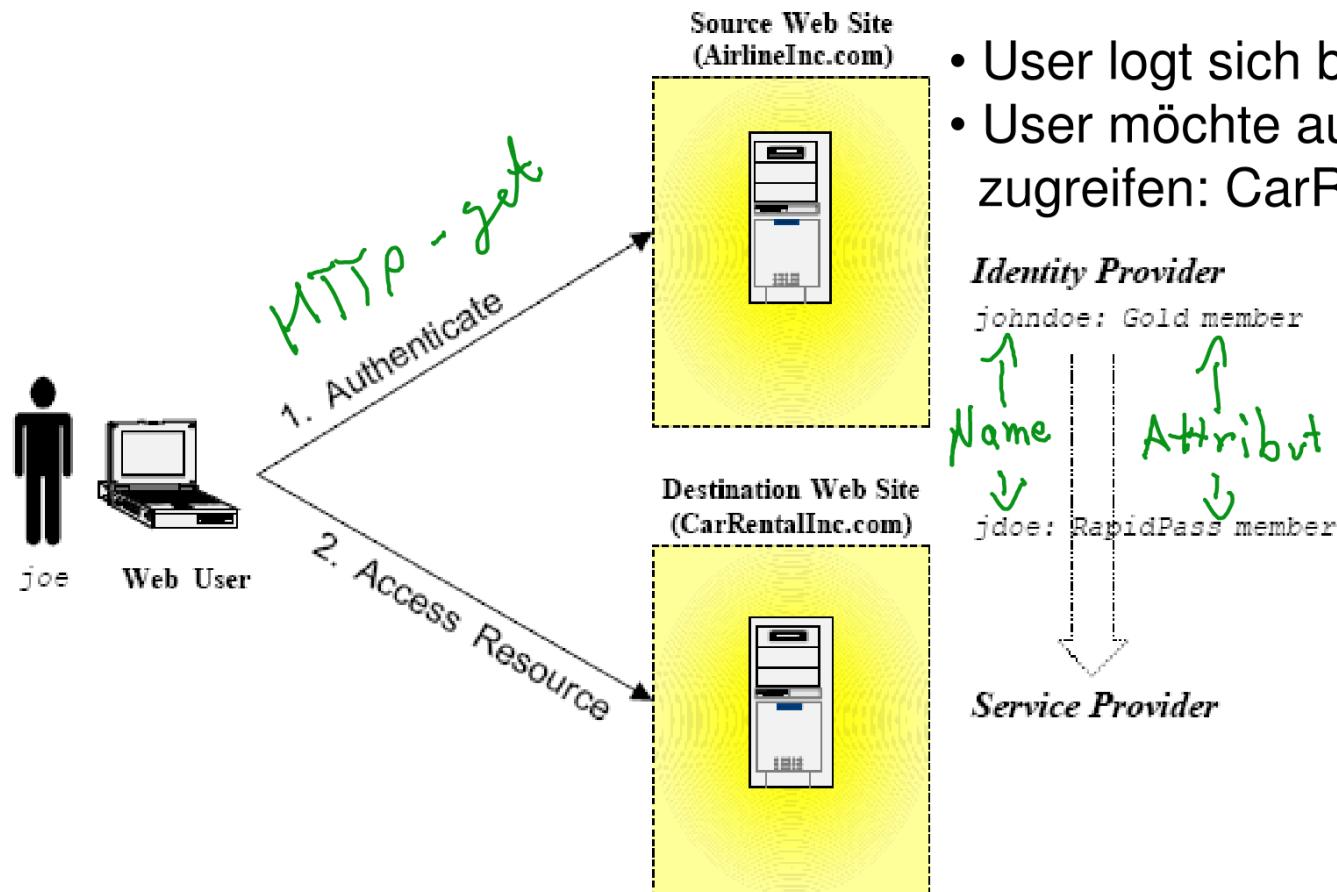
2001 OASIS (Organization for the Advancement of Structured Information Standards)

- **assertions** = Aussagen **eines Ausstellers** über **ein Subjekt**
 - **authentication** assertion: Subjekt ist authentisiert
 - **authorization** decision assertion: Subjekt ist autorisiert
 - **attribute** assertion: Subjekt hat bestimmte Attribute
- **Single-Sign-on** für unterschiedliche administrative Domänen

Analogien: PKI? Kerberos-Tickets?

SAML-Protokoll

- zwischen Policy-Enforcement-Points u. Policy-Decision-Points
- Frage-/Antwort-Nachrichtenformate: SAML-request/reply



- User logt sich bei [AirlineInc.com](#) ein
- User möchte auf anderen Dienst zugreifen: [CarRentalInc](#)
- [AirlineInc.com](#) sichert [CarRental](#) zu, dass der Benutzer bekannt ist
- [CarRental](#) vertraut den Aussagen von [AirlineInc](#)
- Benutzer muss sich [nicht](#) bei [CarRental](#) erneut anmelden

Use-case: Quelle: SAML-Spezifikation

Authentication-Assertion

- Die ausstellende Autorität bestätigt, dass:
 - Subjekt S sich zum Zeitpunkt T mittels Ls *gültigk* authentifiziert hat, z.B. über Passwort, nPA, RSA SecureID

Bewertung:

- schwache Bindung zwischen tatsächlich durchgeföhrter Authentifikation und dem Versand von Assertions
- Konsequenz:
 - ein Nachweis, dass die Assertion tatsächlich für den anfragenden Nutzer ausgestellt wurde, müsste erfolgen
 - z.B. über das SubjectConfirmation-Element, das vom Nutzer den Nachweis eines Credentials fordert

Beispiel: Authentication-Assertion

```

1: <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
2:   Version="2.0"
3:   IssueInstant="2005-01-31T12:00:00Z">
4:   <saml:Issuer>
5:     www.acompany.com
6:   </saml:Issuer>
7:   <saml:Subject>
8:     <saml:NameID
9:       Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
10:      j.doe@company.com
11:    </saml:NameID>
12:  </saml:Subject>
13:  <saml:Conditions>
14:    NotBefore="2005-01-31T12:00:00Z"
15:    NotOnOrAfter="2005-01-31T12:00:00Z">
16:  </saml:Conditions>
17:  <saml:AuthnStatement
18:    AuthnInstant="2005-01-31T12:00:00Z" SessionIndex="67775277772">
19:    <saml:AuthnContext>
20:      <saml:AuthnContextClassRef>
21:        urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
22:      </saml:AuthnContextClassRef>
23:    </saml:AuthnContext>
24:  </saml:AuthnStatement>
25: </saml:Assertion>

```

Subjekt S

Zeitpunkt T

Methode M

+ XML Signature

Attribute-Assertion, gut geeignet für attribut-basierte Kontrolle

Die ausstellende Autorität bestätigt, dass:

- Kreditrahmen

- Subjekt **S** ist assoziiert mit den **Attributen A, B, ...**
- mit den Werten “a”, “b”, “c”...
- Attribute sind über **Attribut-Statements** spezifiziert

```

1: <saml:AttributeStatement xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
2:   <saml:Attribute
3:     NameFormat="http://smithco.com"
4:     Name="PaidStatus"
5:     <saml:AttributeValue>
6:       PaidUp
7:     </saml:AttributeValue>
8:   </saml:Attribute>
9:   <saml:Attribute
10:    NameFormat="http://smithco.com"
11:    Name="CreditLimit"
12:    <saml:AttributeValue xsi:type="smithco:type">
13:      <smithco:amount currency="USD">
14:        500.00
15:      </smithco:amount>
16:    </saml:AttributeValue>
17:  </saml:Attribute>
18: </saml:AttributeStatement>

```

Subjekt **S** = <http://smithco.com>
Attribut **A** = **PaidStatus**
Attributwert **a**= **PaidUp**

Attribut **B** = **CreditLimit**
Attributwert **b**= **500**

Authorization-Decision-Assertion

- Ausstellende Autorität entscheidet, ob eine Anfrage genehmigt werden soll:
 - Anfrage von **Subjekt S** für einen Zugriff **A**
 - auf **Ressource R**
 - unter der **Bedingung E**, z.B. zur *Büro-Arbeitszeit*

```
1: <saml:Assertion ...>
2:   <saml:Conditions .../>
3:   <saml:AuthorizationStatement
4:     Decision="Permit"
5:     Resource="http://jonesco.com/rpt_12345.htm">
6:     <saml:Subject>
7:       <saml:NameIdentifier
8:         SecurityDomain="smithco.com"
9:         Name="joeuser" />
10:      </saml:Subject>
11:    </saml:AuthorizationStatement>
12:  </saml:Assertion>
```

Bedingung E

Ressource R

Subjekt S

Fazit

- SAML bietet Mechanismen, um sicherheitsbezogene Infos zu verteilen: Authentifizierungs-, Autorisierungs-Assertions
- SAML definiert aber *nicht*, wie die Informationen der Assertions (z.B. die Attribute, die ein Subjekt hat), für eine konkrete Policy genutzt werden sollen
- Policies und deren Umsetzung können flexibel mit anderen Konzepten definiert werden
- Zur Spezifikation von **konkreten Access-Control-Policies** wird ein anderer Standard verwendet:
XACML (eXtensible Access-Control-Markup-Language)

SAML und XACML kommen häufig kombiniert zum Einsatz

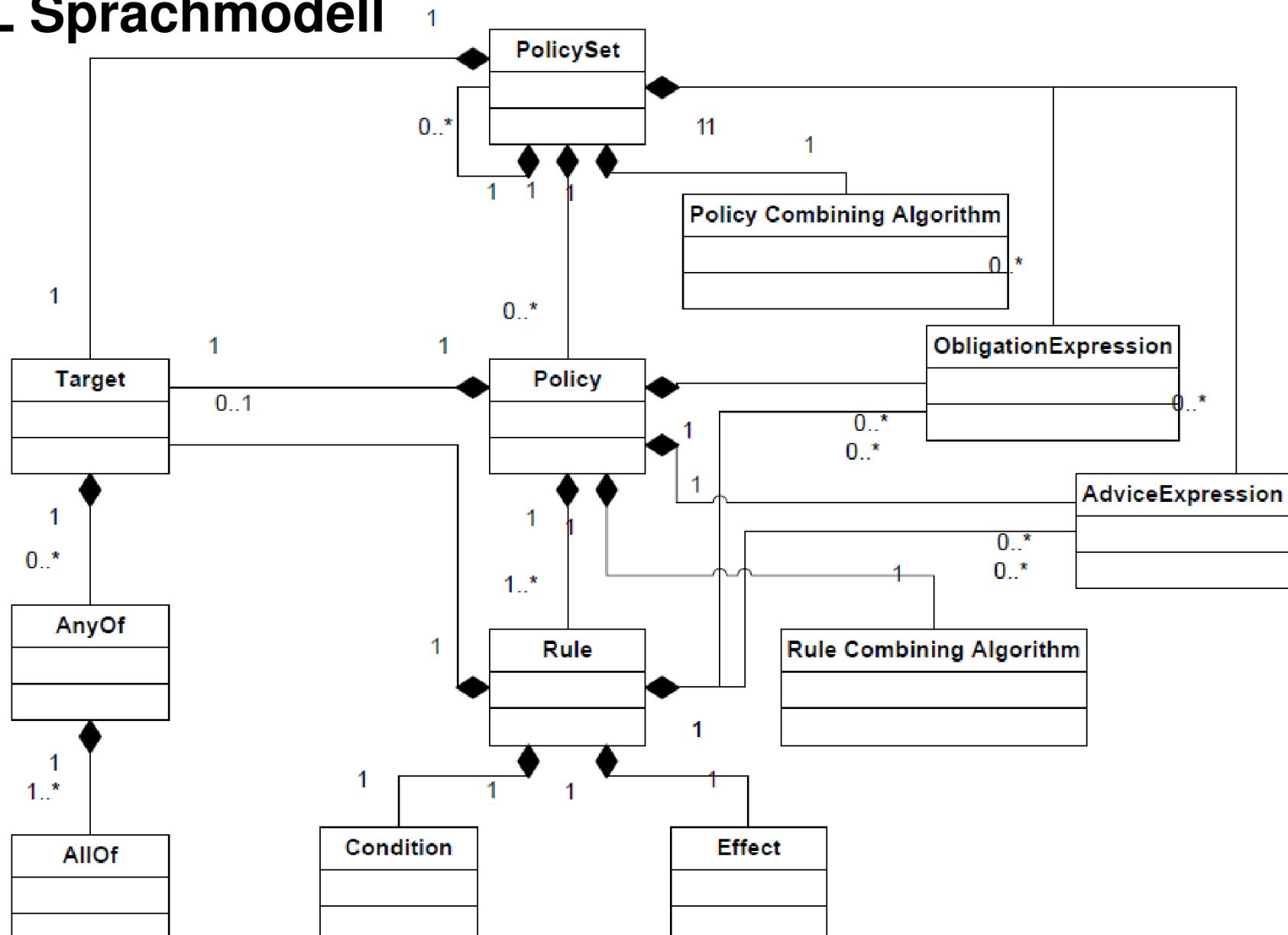
5.4.2 XACML (eXtensible Access-Control-Markup-Language)

- OASIS-Spezifikation, siehe <http://www.oasis-open.org>
- seit August 2010: XACML Version 3.0 veröffentlicht
- XML-basierte Sprache, um **Zugriffsregeln** zu spezifizieren

Sprachkonzepte:

- Regel, Policy, Policy-Set
 - Eine **Regel** umfasst die Elemente
 - **Target**: spezifiziert das Ziel, für das die Regel gilt, mit Attributen u. logischen Ausdrücken
 - **Effect**: permit oder deny (Zugriffserlaubnis bzw. -verbot)
 - **Condition**: Boolescher Ausdruck, optional
 - **Obligation** Expression und **Advise**-Expression
 - Eine **Policy** fasst eine **Menge von Regeln** zusammen
- Menge von Regeln*
- Menge von Policies*
- ist die Regel anwendbar (indirekt)*
- Problem: widersprüchliche Regeln*
- Mit dem Zugriff*

XACML Sprachmodell



Beispiel- Policy mit nur einer Regel

```
1: <Policy
2:   xmlns="..." xmlns:xsi="..." xsi:schemaLocation="..."
3:   PolicyId="urn:oasis:names:tc:xacml:3.0:example:SimplePolicy1"
4:   Version="1.0"
5:   RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
6:   <Description>
7:     Medi Corp access control policy
8:   </Description>
9:   <Target/>
10:  <Rule
11:    RuleId= "urn:oasis:names:tc:xacml:3.0:example:SimpleRule1"
12:    Effect="Permit">
13:    <Description>
14:      Any subject with an e-mail name in the med.example.com domain
15:      can perform any action on any resource.
16:    </Description>
17:    <Target>
18:      <AnyOf>
19:        <AllOf>
20:          <Match
21:            MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
22:            <AttributeValue
23:              DataType="http://www.w3.org/2001/XMLSchema#string"
24:              >med.example.com</AttributeValue>
25:            <AttributeDesignator
26:              MustBePresent="false"
27:              Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
28:              AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
29:              DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
30:            </Match>
31:          </AllOf>
32:        </AnyOf>
33:      </Target>
34:    </Rule>
35:  </Policy>
```

Deny overwrites permit

Informelle Beschreibung

Spezifikation der Regel

Keine Conditions

Keine Obligationen

Beispiel Forts.

- Anfrage an den PDP, eine Zugriffsentscheidung zu treffen
- Folgende Anfrage sei gegeben:
Bart Simpson mit der E-Mail Adresse `bs@simpsons.com`
möchte seine medizinischen Daten von Medi Corp lesen.
- Die XML Repräsentation obiger Anfrage findet sich auf der nachfolgenden Folie

Erläuterung:

- **1. Attribute-Element:** beschreibt zugreifendes Subjekt
in Form von Attributen: hier E-Mail-Adresse `bs@simpsons.com`
- **2. Attribute-Element:** beschreibt gewünschte Ressource:
hier über URI spezifiziert: `file://medico/record/patient/BartSimpson`

1. Attribut: Subjekt

```
1: <Request
2:   xmlns="..." xmlns:xsi="..." xsi:schemaLocation="..."
3:   ReturnPolicyIdList="false">
4:   <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
5:     <Attribute IncludeInResult="false"
6:       AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id">
7:       <AttributeValue
8:         DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
9:         >bs@simpsons.com</AttributeValue>
10:      </Attribute>
11:    </Attributes>
12:    <Attributes
13:      Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
14:      <Attribute IncludeInResult="false"
15:        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
16:        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
17:          >file:///medico/record/patient/BartSimpson</AttributeValue>
18:      </Attribute>
19:    </Attributes>
20:    <Attributes
21:      Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
22:      <Attribute IncludeInResult="false"
23:        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
24:        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
25:          >read</AttributeValue>
26:      </Attribute>
27:    </Attributes>
28:  </Request>
```

Mein
Regeln
nur für anwendbar

2. Attribut: Objekt/ Ressource

```
14:      <Attribute IncludeInResult="false"
15:        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
16:        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
17:          >file:///medico/record/patient/BartSimpson</AttributeValue>
18:      </Attribute>
19:    </Attributes>
20:    <Attributes
21:      Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
22:      <Attribute IncludeInResult="false"
23:        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
24:        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
25:          >read</AttributeValue>
26:      </Attribute>
27:    </Attributes>
28:  </Request>
```



3. Attribut: Zugriffs- recht

```
22:      <Attribute IncludeInResult="false"
23:        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
24:        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
25:          >read</AttributeValue>
26:      </Attribute>
27:    </Attributes>
28:  </Request>
```



Erläuterung: Forts.

- 3. Attribute-Element: beschreibt die gewünschte Operation:
hier **read**

Aktionen des PDP:

- Suchen nach Policy im Policy-Repository
- Vergleich der Attribute in der Anfrage mit denen der Policy
- Für das Beispiel gilt:
 - angeforderte Ressource **stimmt** mit Target-Element **überein**
 - angefragte Aktion/Operation **stimmt** mit Target **überein**
 - Aber die Attribute des anfragenden Subjekts **stimmen nicht** mit **med.example.com** überein

Als Antwort liefert der PDP die Aussage, dass die Policy-Regeln nicht anwendbar ist: **not applicable**

Der Antwort Kontext des PDP ist wie folgt:

```
1: <Response xmlns="..." xmlns:xsi="..." xsi:schemaLocation="...">
2:   <Result>
3:     <Decision>NotApplicable</Decision>
4:   </Result>
5: </Response>
```

nicht auflösbare conditions
zu wenig Information

Der PEP muss somit den Zugriff auf die Ressource verweigern

Auszug aus einer Policy mit einer **Obligation** Regel

- **PEP** muss Mail senden, wenn Zugriff erlaubt wurde

Identifikation
der Mail
Adresse

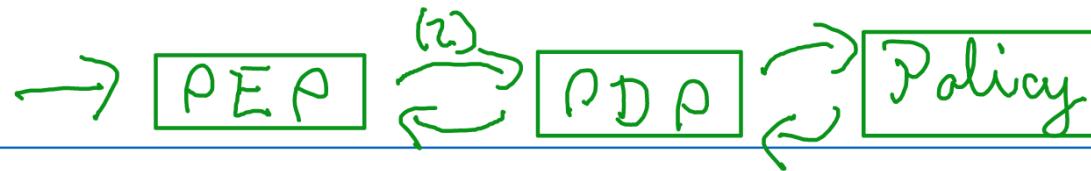
Text der
Mail

Verweis auf
weitere
Infos für
Mail-Body

```

1.  <Policy ... >
2.    <ObligationExpressions>
3.      <ObligationExpression
4.        ObligationId="urn:oasis:names:tc:xacml:example:obligation:email" FulfillOn="Permit">
5.        <AttributeAssignmentExpression
6.          AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:mailto">
7.            <AttributeSelector
8.              MustBePresent="true"
9.              Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
10.             Path="md:record/md:patient/md:patientContact/md:email"
11.             DataType="http://www.w3.org/2001/XMLSchema#string"/>
12.           </AttributeAssignmentExpression>
13.           <AttributeAssignmentExpression
14.             AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:text">
15.               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
16.                 >Your medical record has been accessed by:</AttributeValue>
17.             </AttributeAssignmentExpression>
18.             <AttributeAssignmentExpression
19.               AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:text">
20.                 <AttributeDesignator
21.                   MustBePresent="false"
22.                   Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
23.                   AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
24.                   DataType="http://www.w3.org/2001/XMLSchema#string"/>
25.                 </AttributeAssignmentExpression>
26.               </ObligationExpression>
27.             </ObligationExpressions>

```



Zusammenfassung: Zusammenspiel PDP u. PEP:

- ein XACML-Policy-Enforcement-Point (PEP) erhält eine **Anfrage** zur Nutzung einer Ressource
- der PEP erhält **SAML-Assertions** mit Aussagen über den Anfrager, Zeit/Datum, Ort, Eigenschaften der Ressource, ...
- der PEP übergibt die Informationen an einen Policy-Decision-Point (PDP)
- der PDP sucht zutreffende Policies aus Datenbank, evaluiert die Anfrage, bereinigt ggf. auftretende Konflikte
- PDP sendet **Antwort** an PEP: permit, deny, not applicable, indeterminate (z.B. Eingabewerte fehlen)
- der **PEP** setzt die Entscheidung durch, d.h. er **gewährt oder verweigert** den Zugriff auf die Ressource

Sicherheitsprobleme:

- die XACML-Spezifikation enthält **keine** Vorgaben in Bezug auf die Absicherung der Regeln, Policies, Nachrichten
- Ebenfalls **keine** Vorgaben bezüglich Maßnahmen zum Vertrauensaufbau (Trust)
- **Angriffsmöglichkeiten:** klassisches ‚Repertoire‘:
 - **keine vertrauliche** Datenübertragung, Privacy-Verletzung Beispiel?
 - **kein Manipulationsschutz:** Beispiel?
 - **kein Schutz vor Replay, Löschen** von Nachrichten, **Einspielen** Beispiel?
 - **keine Vorgaben zum Schutz der Policy-Datenbank** Beispiel?

Fazit:

- SAML/XACML: **Attribut-basiert** statt nur identitätsbasiert
 - gut geeignet für **dezentrales** Web-Service-Umfeld,
 - einfache **Policy-Sprache**, um auch komplexe Regelwerke zu formulieren, die verteilt verarbeitet werden
 - **ad-hoc etablierbare Vertrauensbeziehungen** über Attribute und Assertions (keine PKI vorab notwendig)
- **Aber fehlende Sicherheit:** **zusätzliche** Maßnahmen erforderl.
 - vertrauliche, integre, authentische Kommunikation
 - Vertrauenswürdigkeit der PDPs (wie?)
 - Schutz der Policy-Datenbanken: Zugriffe, Inhalte
 - Privacy: Schutz von Attributinformationen
 - Frische-Nachweise, um DoS abzuwehren