h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

IT-Sicherheit (SoSe 2016)

Kapitel 7: Zugriffskontrolle

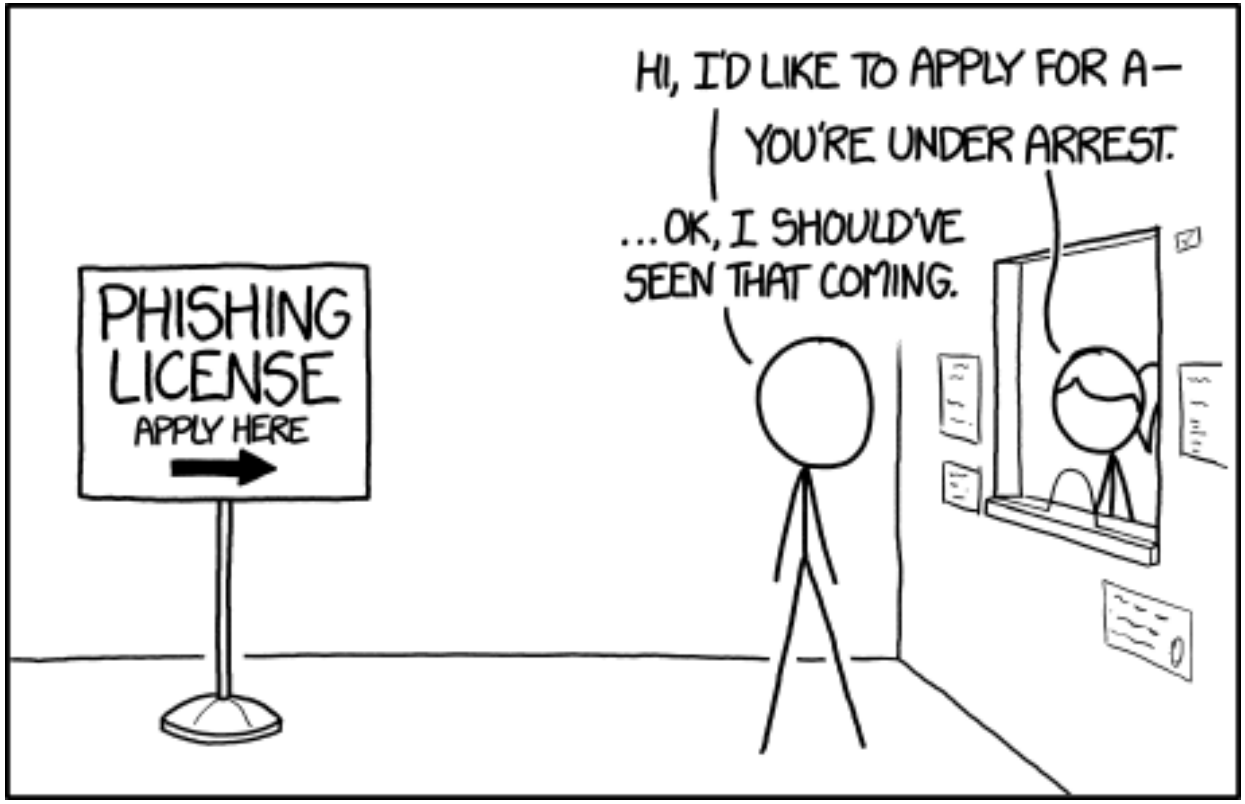
Stefan Edelkamp

edelkamp@tzi.de

(Based on slides provided by Andreas Heinemann)

http://imgs.xkcd.com/comics/phishing_license.png

Teaser of the day

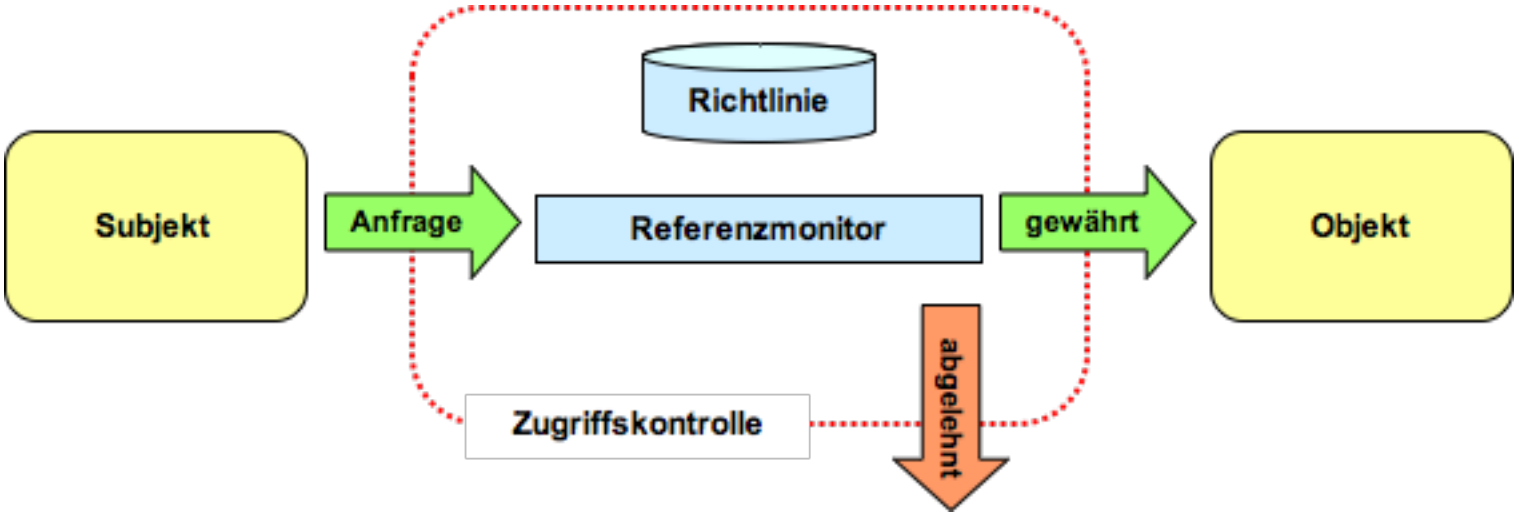


Agenda

- Einführung
- Modelle und Konzept
- Umsetzungsbeispiele und Implementierungen
- Zusammenfassung

Einleitung

- Ziele der Zugriffskontrolle
  - Überwachung und Steuerung des Zugriffs auf Ressourcen
  - Sicherstellung von Integrität, Vertraulichkeit und Verfügbarkeit von Informationen
  - Rechteverwaltung
- Realisierung auf verschiedenen Ebenen möglich, z.B.
  - Datenbankanwendung kontrolliert die Zugriffe auf Datenbanktabellen
  - Betriebssystem kontrolliert Zugriffe auf Dateien



Sicherheitsmodelle und Mechanismen

- Ausgangspunkt
  1. Sicherheitsrichtlinien innerhalb einer Organisation (z.B. Abläufe der Verwaltung, Schutz geistigen Eigentums)
  2. Kontrollprinzipien
     - Kontrolle der Einhaltung der Sicherheitsrichtlinien (z.B. Vier-Augen-Prinzip, Klassifikation vertraulicher Information)
- Sicherheitsmodelle
  - Abstraktion von realen Gegebenheiten bzw. Vereinfachung
  - Festlegung von Kontrollprinzipien zur Durchsetzung einer Sicherheitsrichtlinie
- Mechanismen
  - Konkrete Implementierung der Modelle

Arten der Zugriffskontrolle

- Discretionary Access Control (DAC)
  - Eigentümer ist für den Schutz eines Objekts selbst verantwortlich
  - Rechte werden für einzelne Objekte vergeben
  - Objektbezogene Sicherheitseigenschaften, aber keine systemweiten
  - Fast alle Betriebssysteme (z.B. UNIX, Windows) unterstützen DAC

- Mandatory Access Control (MAC)
  - Systembestimmte (regelbasierte) Festlegung von Sicherheitseigenschaften
  - Benutzerdefinierte Rechte werden durch systembestimmte überschrieben (dominiert)

- Role Based Access Control (RBAC)
  - Definition von Rollen mit bestimmten Rechten und Zuordnung von Subjekten zu Rollen
  - Kann genutzt werden, um sowohl MAC als auch DAC umzusetzen

Modellierung: Zugriffsmatrix-Modell (Access-Matrix-Model)

- Matrix beschreibt Rechte von Subjekten an Objekten
  - (Dynamische) Menge von Objekten O, z.B. Datei, Port, Prozess
  - (Dynamische) Menge von Subjekten S, z.B. Prozess, Nutzer
    (Subjekte können auch Objekte sein, d.h. $S \subseteq O$)
  - Menge von Rechten R, z.B. **`read`**, **`write`**, **`execute`**, **`send`**, **`receive`**

- Zugriffsmatrix $M_t : S \times O \rightarrow 2^R$, beschreibt Zugriffsrechte zum Zeitpunkt t

Subjekte

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

Objekte

Zugriffsmatrix-Modell

- Vorteile
  - Sehr einfach und intuitiv nutzbar, flexibel einsetzbar
  - Feingranulare Subjekte / Objekte und Rechte
  - Einfach zu implementieren
  - Konzeptionelle Grundlage der Zugriffskontrolle fast aller Betriebssysteme

- Nachteile
  - Skaliert sehr schlecht (riesige Tabellen!)
  - Kann dynamische Rechte nicht gut abbilden
  - Rechtevergabe an Subjekte und nicht an Aufgaben
    - Szenarien mit gleichen Subjekten, die wechselnde Aufgaben erfüllen kaum modellierbar
  - Rechtevergabe bzw. -rücknahme relativ komplex

  - Ungeeignet für Unternehmens- oder Verwaltungsumgebungen

Role-Based Access Control (RBAC)

- NIST-Standard, entwickelt 1992 von Sandhu, Ferraiolo und Kuhn
  - http://csrc.nist.gov/rbac/sandhu-ferraiolo-kuhn-00.pdf
- Ziel: Effektive, skalierende und effiziente Rechteverwaltung
- Ansatz: Aufgabenorientierte (dynamische) Rechtevergabe durch Rollen
- Rolle: Beschreibt Aufgabe und die damit verbundenen Berechtigungen
- Nachbilden von Organisationsstrukturen in Unternehmen
  - Rechte und Verantwortlichkeiten sind häufig direkt aus den Organigrammen der Personalabteilungen ableitbar
  - Prinzipien wie *need-to-know*, *separation-of-duty* lassen sich gut abbilden
- Einsatz weit verbreitet z.B.: Microsoft Active Directory, Microsoft SQL Server, SELinux, FreeBSD, Solaris, Oracle RDBMS, PostgreSQL 8.1, SAP R/3

Role-Based Access Control (RBAC) – Komponenten

- Komponenten eines RBAC-Modells
  - Menge von Subjekten (Benutzer) S
  - Menge von Rollen R
  - Menge von Zugriffsrechten P (permissions) für Objekte

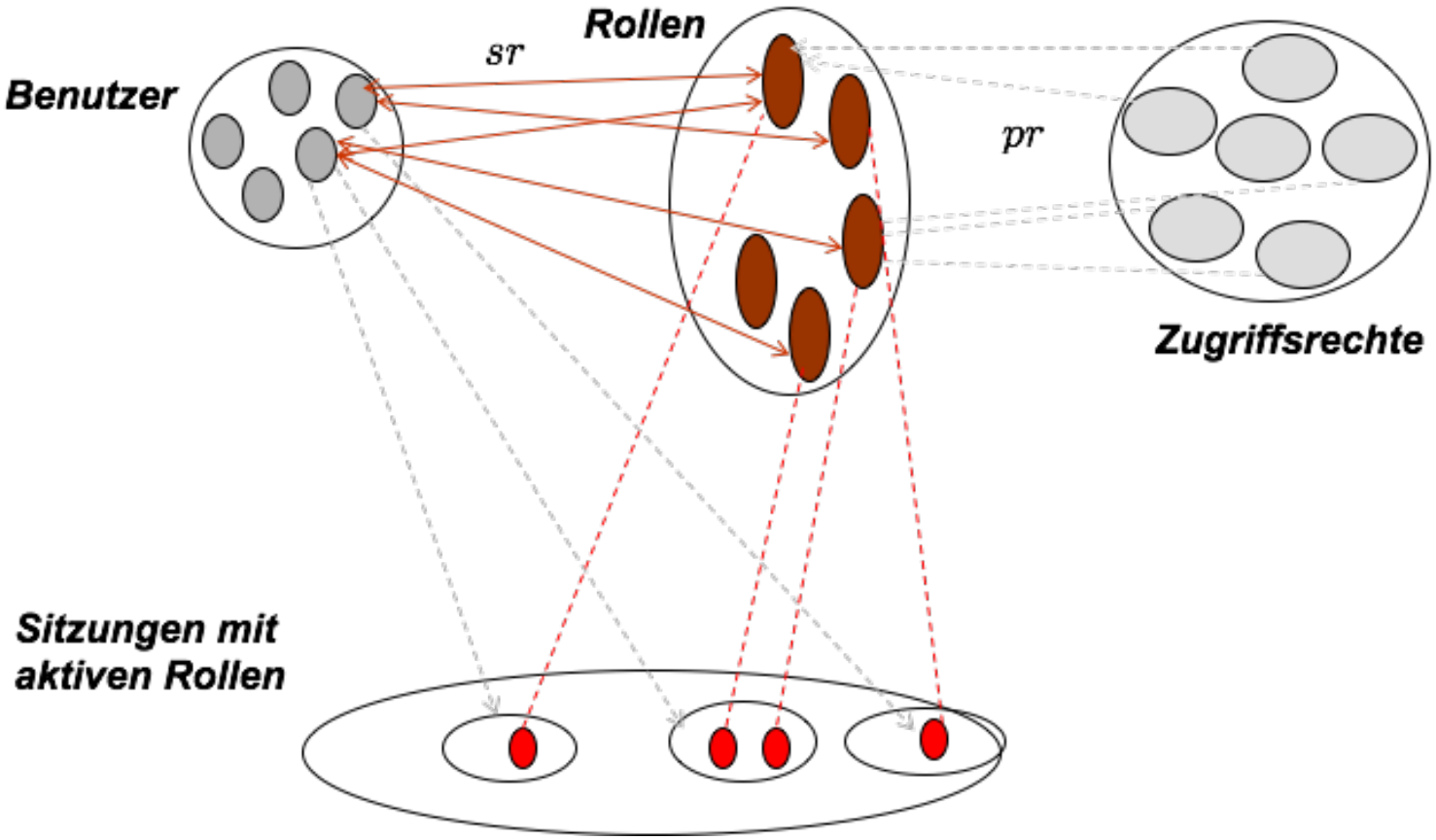
- Zwei Abbildungen:
  - Zuordnung Benutzer - Rollen $sr: S \rightarrow 2^R$
  - Zuordnung Rolle - Rechte $pr: R \rightarrow 2^P$

- Sitzung ist eine Relation $session \subseteq S \times 2^R$
  - Modelliert die „aktive“ Zuordnung von Subjekten auf Rollen
  - D.h. die Rolle die jetzt gerade vom Subjekt angenommen wird

- Für alle Paare $(s,rl) \in session$ gilt immer $rl \subseteq sr(s)$
  - rl sind die „aktiven“ Rollen eines Subjekts s; $(rl \in 2^R)$
  - D.h. ein Subjekt kann nur eine Rolle annehmen die ihm zugeordnet ist

- Rollen bestimmen Zugriffsberechtigung (Abbildung pr)

Role-Based Access Control (RBAC) – Komponenten



Role-Based Access Control (RBAC) – Beispiel

Rollen und deren Berechtigungen im Krankenhaus

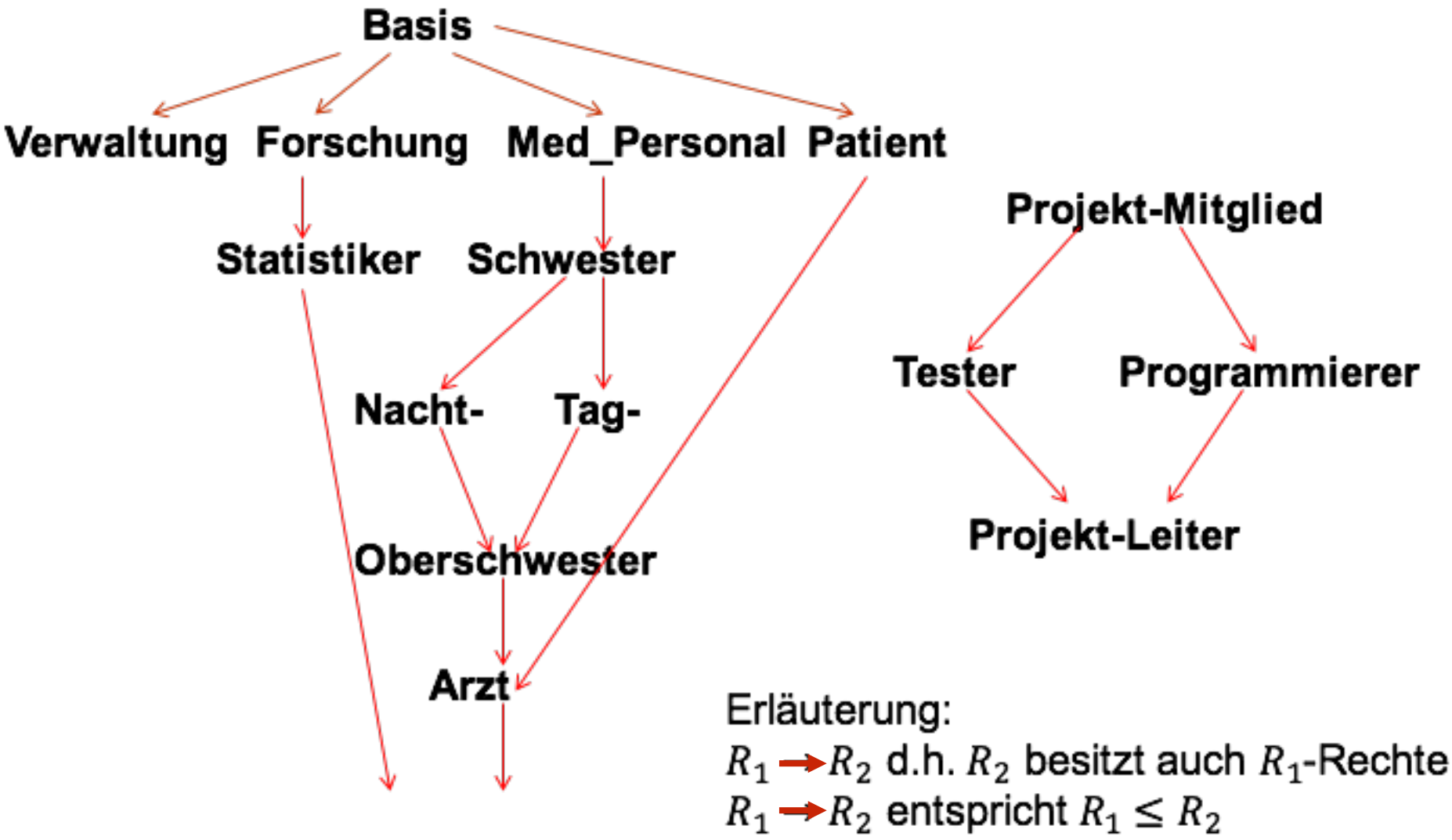
- Behandelnde Ärzte (auch zeitweise zugeordnete Ärzte, AiP, PJ)
  - ganze Patientenakte im Behandlungszusammenhang (außer besonders sensible Daten): lesend und schreibend
  - abteilungsinterne Daten aller Aufenthalte
- Pflegekräfte
  - Zugriff auf Krankenakte; Umfang durch Abteilungsleiter festgelegt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

Role-Based Access Control (RBAC) – Hierarchien

- Rollenhierarchien
  - Ziel: Abbildung von Hierarchien in Organisationen
    - Rechte werden hierarchisch „vererbt“
    - Aber keine Vererbung von Rollen

- Definition einer partiellen Ordnung ≤ auf Rollen
  - falls $R_i \leq R_j$ dann besitzt R_j alle Rechte von R_i und ggf. noch zusätzliche Rechte

RBAC Rollen-Hierarchien am Beispiel



Erläuterung:

$R_1 \rightarrow R_2$ d.h. R_2 besitzt auch R_1-Rechte

$R_1 \rightarrow R_2$ entspricht $R_1 \leq R_2$

Role-Based Access Control (RBAC) – Rollenausschluss

- Statischer Ausschluss von Rollen („separation of duty“)
  - $SSD \subseteq R \times R$ definiert Rollenpaare, die „statisch“ ausgeschlossen sind
    - $\forall R_i, R_j \in R, \ \forall s \in S$:
      $(s \in member(R_i) \wedge s \in member(R_j)) \Rightarrow (R_i, R_j) \notin SSD$
      wobei $member(R_i) = \{ s \in S \mid R_i \in sr(s) \}$
  - Beispiel Bank: Kassierer darf nicht gleichzeitig Kassenprüfer sein, d.h. (Kassierer, Kassenprüfer) $\in SSD$

- Dynamischer Ausschluss von Rollen
  - $DSD \subseteq R \times R$ definiert Rollen, die sich „dynamisch“ ausschließen
  - $\forall R_i,R_j \in R, \ \forall s \in S$:
    $(s \in member(R_i) \wedge s \in member(R_j) \wedge \{R_i,R_j\} \subseteq active(s)) \Rightarrow (R_i,R_j) \notin DSD$
    wobei $active(s) = \{R_i \mid \exists R_s \in 2^R \wedge (s,R_s) \in session \wedge R_i \in R_s\}$
  - Beispiel Bank: Kundenbetreuer darf nicht sein eigenes Konto betreuen,d.h. (Kundenbetreuer, Kontoinhaber) $\in DSD$

Role-Based Access Control (RBAC) – Fazit

- Rollenkonzepte sind sehr flexibel verwendbar, skalieren gut
- Statische und dynamische Ausschlüsse von Rollen können modelliert werden
- Direktes Nachbilden bekannter Organisations- und Rechtestrukturen in Unternehmen
- Rollenbasierte Zugriffskontrolle ist de-facto-Standard in Unternehmen und Unternehmenssoftware wie SAP/R3
- Intuitive und relativ einfache Abbildung der Rollen auf Geschäftsprozesse (Workflows) im Prinzip möglich
- **Änderungen der Abbildung zwischen Rollen und Rechten sind selten**
- **Änderung der Rollenmitgliedschaften sind häufig**

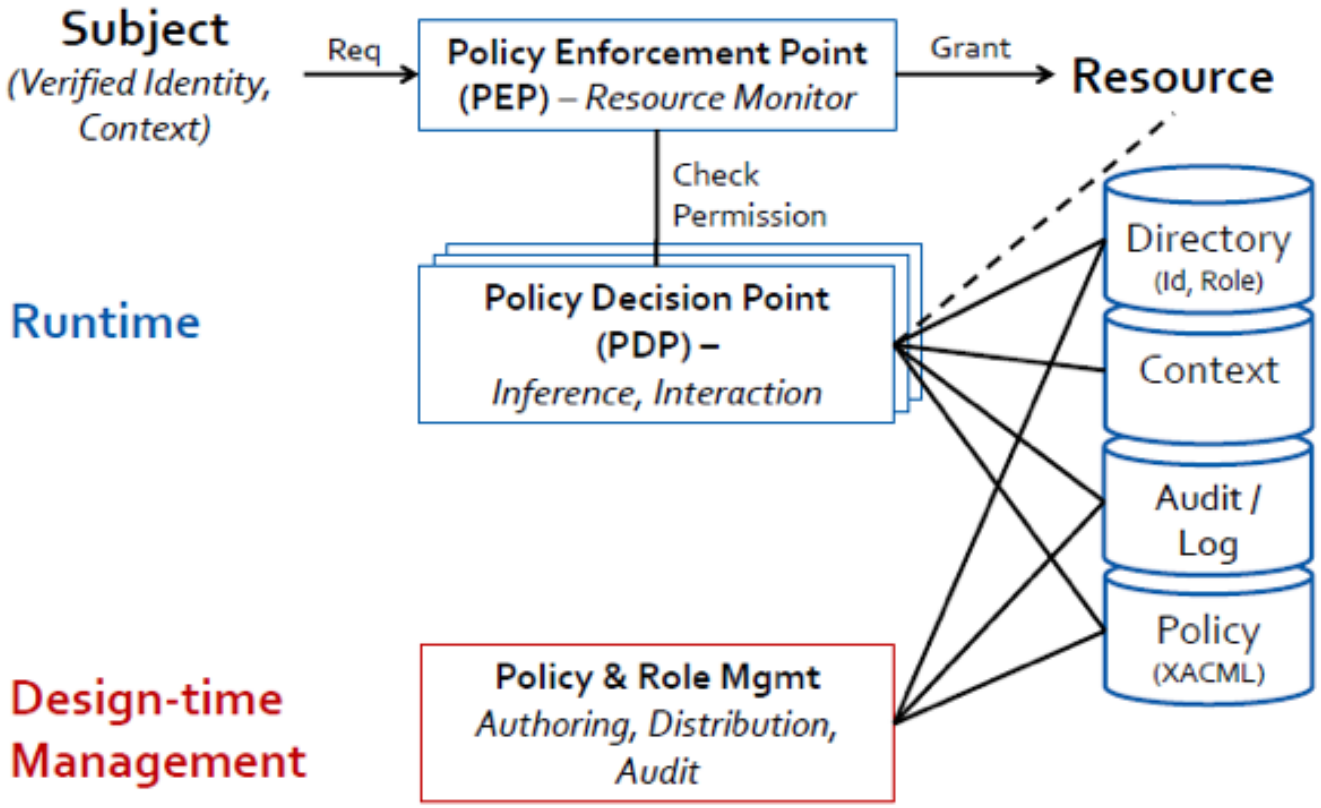
Architekturmodell zur Umsetzung der Zugriffskontrolle

- Design: Aufteilung in Berechtigungs- und Zulässigkeitskontrolle

- Berechtigungskontrolle
  - PDP: *Policy Decision Point*
  - Prüfung beim erstmaligem Zugriff auf ein Objekt (von vertrauenswürdigen Systemdiensten (Kernel-level;z.B. Dateisystem))
  - Ausstellung einer Bescheinigung, z.B. File-Handle, Ticket

- Zulässigkeitskontrolle
  - PEP: *Policy Enforcement Point*
  - durch Objektverwalter (z.B. User-level Filessystem)
  - bei Objektzugriff: Prüfen der Gültigkeit der Bescheinigung
  - kein Zugriff auf die Rechteinformation notwendig

Zusammenspiel von PDP und PEP



Zur Implementierung von Zugriffskontrolle

- Ausgangspunkt: Access Control Matrix
- Üblicherweise dünn besetzt
- Implementierung als Tabelle ineffizient

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

Zwei Realisierungen:

- Access Control Lists (ACL): spaltenweise Speicherung der Matrix (Objekt-Sicht)
- Capabilities: zeilenweise Speicherung der Matrix (Subjekt-Sicht)

Access Control Lists

- Spaltenweise Speicherung der Zugriffskontrollmatrix
- Weit verbreitetes Konzept in Betriebssystemen (z.B. UNIX/Linux, Windows, iOS, Android)
- ACL ist eine geschützte Datenstruktur des Betriebssystems
- Vorteil: Rechte an einem Objekt sind effizient bestimmbar
- Nachteil: Schlechte Skalierbarkeit bei dynamischen Subjektmengen

$$ACL(Datei1) = \{(Prozess1, \{read, write\})\}$$

$$ACL(Datei2) = \{(Prozess3, \{owner, execute\})\}$$

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

Capabilities

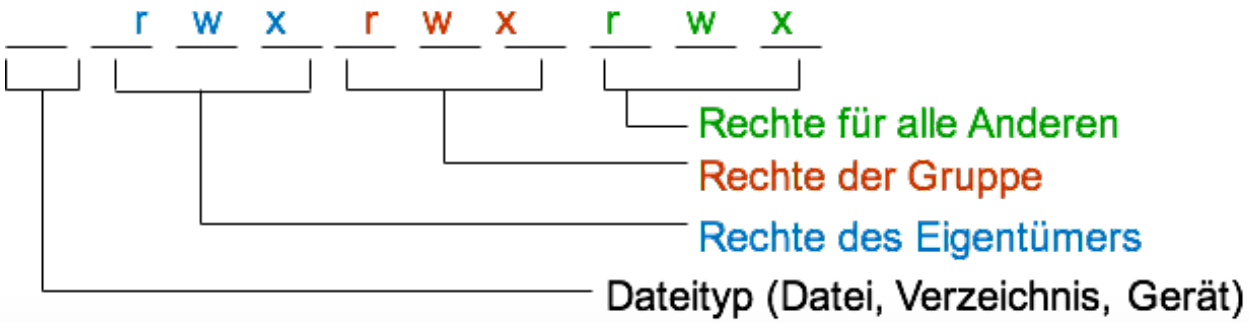
- Zeilenweise Speicherung der Zugriffskontrollmatrix
- Speichert für alle Subjekte eine Liste von Objekten samt Zugriffsrechten
  - Capability ist eine Art Zugriffsticket auf Objekte
  - Capability-Besitz berechtigt zur Wahrnehmung der Rechte
- Umgesetzt in IBM System/38, Mach-µ-Kern (Ports), POSIX
- Vorteil: Alle Rechte eines Subjekts sind effizient bestimmbar
- Nachteil: „Wer darf was“ ist schwierig zu bestimmen

$$CL(Prozess3) = \{(Datei2, \{owner, execute\}), (Prozess1, \{signal\})\}$$

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

Access Control Lists – Beispiel Unix/Linux

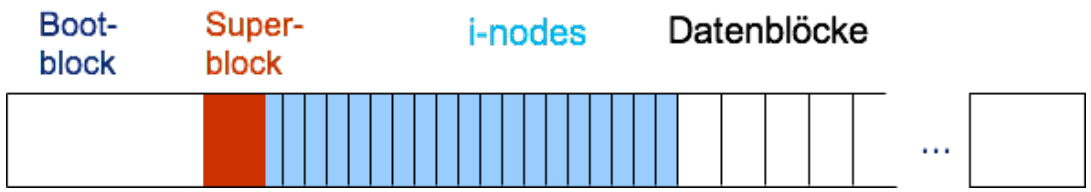
- Beispiel: ACL unter Unix / Linux: vereinfachte ACL
- Rechtevergabe an: Eigentümer (owner), Gruppe (group), Rest der Welt (other)
- Eindeutige Identifikation durch User-IDs in /etc/passwd
- Zuordnung zu Gruppen in /etc/groups
- Rechte `r: read w: write x: execute`




```
heinemann — andreas@Ubuntu-1004-lucid-32-minimal: ~ — ssh andreas@andre...
andreas@Ubuntu-1004-lucid-32-minimal:~$ ls -ld Documents/
drwxr-xr-x 2 andreas andreas 4096 2016-06-08 10:37 Documents/
andreas@Ubuntu-1004-lucid-32-minimal:~$ id
uid=1000(andreas) gid=1000(andreas) groups=109(admin),1000(andreas),1002(svn)
andreas@Ubuntu-1004-lucid-32-minimal:~$
```

Zugriffskontrolle unter Unix/Linux

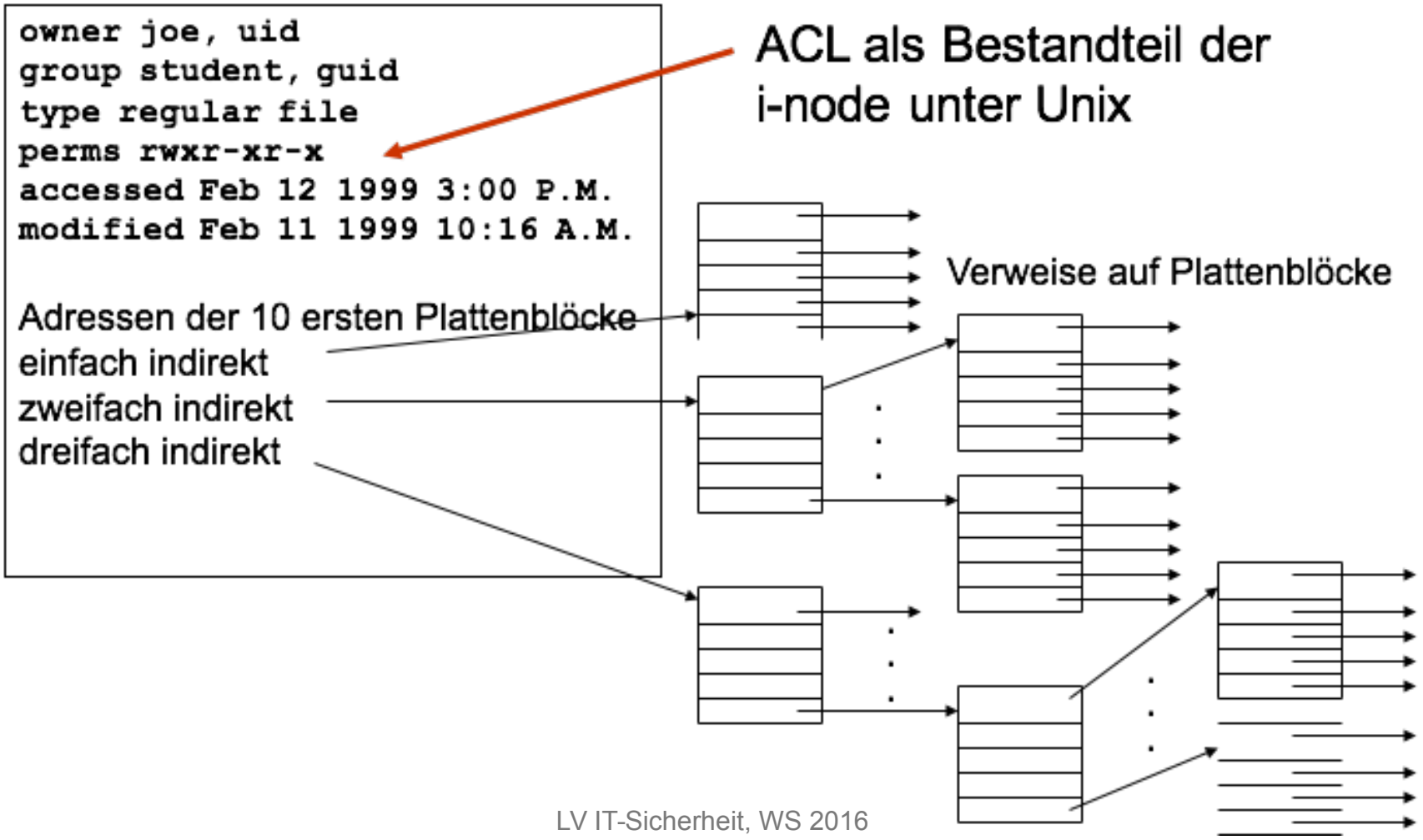
- Subjekte / Prozesse identifiziert über UID, GID
- Zu schützende Objekte: Dateien, Verzeichnisse
- Dateien / Verzeichnisse werden BS-intern über einen Datei-Deskriptor, die sog. i-node (index-node), beschrieben
  - ⋊ Datei-Deskriptor / i-node: Enthält u.a. Name des Datei-Owners und die ACL
- i-nodes werden auf der Festplatte verwaltet
- beim Öffnen einer Datei (System-Call `open()`) wird i-node im Hauptspeicher eingelagert



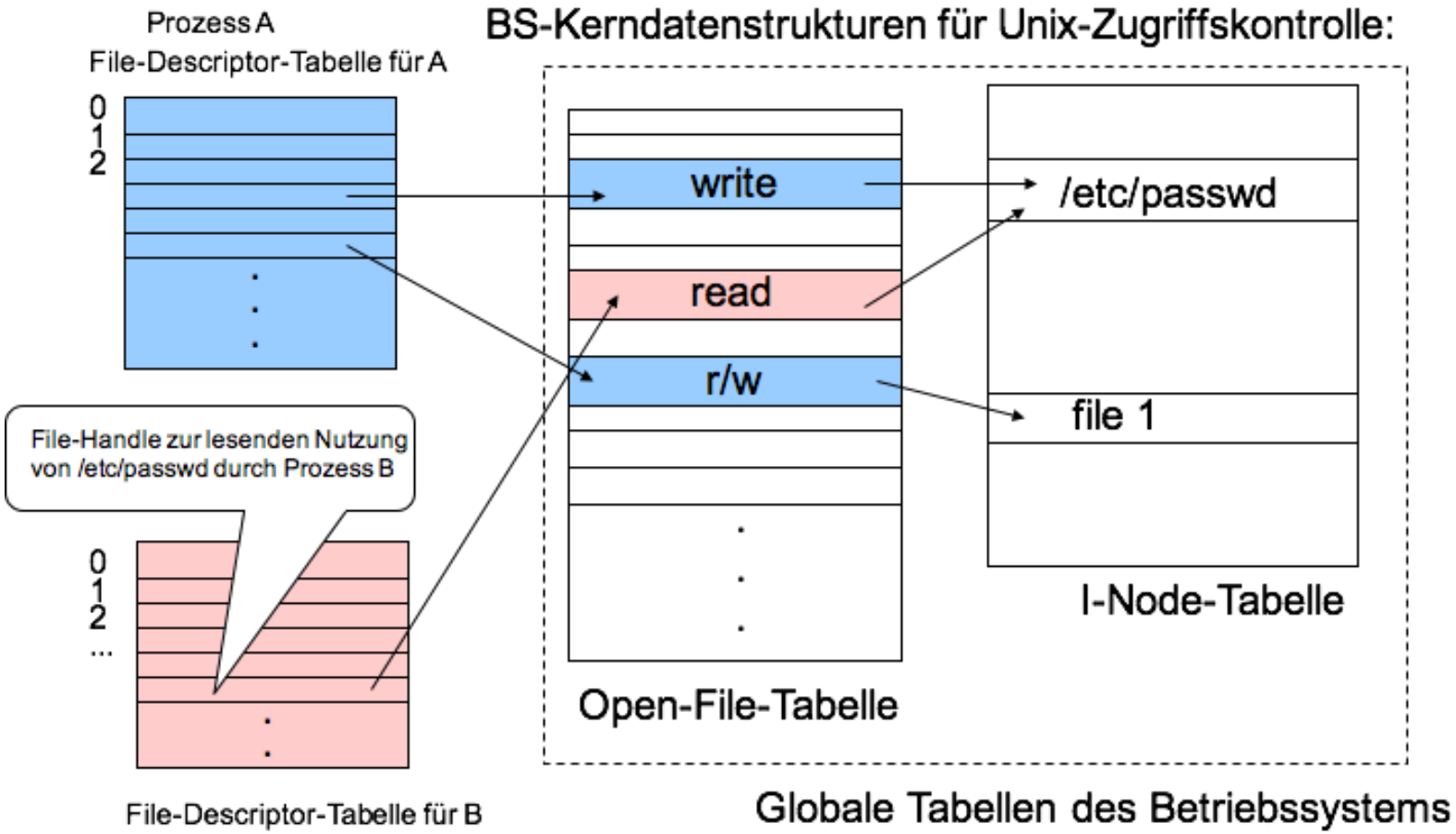
Ablauf bei der Zugriffskontrolle unter Unix/Linux

- Aktion des Prozesses
  - System-Call: `open("/temp/test.txt", O_RDWR)`;
- Aktionen des Unix Kerns
  1. Laden der i-node der zu öffnenden Datei in i-node Tabelle des Kernels
  2. Prüfen, ob zugreifender Prozess gemäß der ACL der Datei zum gewünschten Zugriff **`r`**, **`w`**, **`x`** berechtigt ist (*Policy Decision Point*)
  3. Falls OK, return File-Handle *fd*: enthält Information über zulässige Zugriffsrechte **`r`**, **`w`**, **`x`**
     - Eintrag mit Rechten in *Open File* Tabelle des Kernels
     - Verweis in *File Descriptor* Tabelle des Prozesses auf Rechte
  4. Zugriffe auf geöffnete Datei mit File-Handle *fd*
     - Dateisystem führt Zulässigkeitskontrolle durch (*Policy Enforcement Point*)

Unix/Linux i-node



Ablauf bei der Zugriffskontrolle unter Unix/Linux (2)



ACL und Capabilities

- Häufig Kombination aus beiden Ansätzen
  1. ACL für ersten Zugriff, danach
  2. Ausstellen einer Capability
     - z.B. File-Handle (Unix)
     - z.B. Object-Descriptor (Windows)

Beispiel: Zugriffskontrolle bei relationalen Datenbanken

- Relationale Datenbanken speichern die Daten in Form von Relationen (Tabellen)

Name	Day	Flight	Status
Alice	Mon	GR123	private
Bob	Mon	YL011	business
Bob	Wed	BX201	
Carol	Tue	BX201	business
Alice	Thu	FL9700	business

Flight	Destination	Departs	Days
GR123	THU	7:55	1-4--
YL011	ATL	8:10	12345-7
BX201	SLA	9:20	1-3-5-
FL9700	SLA	14:00	-2-4-6-
GR127	THU	14:55	-2-5-

- Zugriff/Manipulation von Tabellen mittels SQL
  `select`, **`update`**, **`delete`**, **`insert`** etc.
- Beispiel: **`select Name, Status from Diary where Day = 'Mon'`**

Name	Status
Alice	private
Bob	business

Beispiel: Zugriffskontrolle bei relationalen Datenbanken (2)

- Implementiert Discretionary Access Control (DAC)
- Entitäten
  - *Nutzer* (u.U. einige DBMS Nutzerverwaltung)
  - *Aktionen* (SQL Befehle, z.B. **`delete`**)
  - *Objekte* (z.B. Tabellen, Tabellenspalten, Views)
- Nutzer führen Aktionen auf Objekten aus. Ein DBMS muss diese Aktion erlauben oder abweisen
- Bei Erzeugung eines Objektes wird ein Nutzer als Besitzer zugewiesen.
- Besitzer kann Aktionen auf Objekten durchführen.
- Besitzer kann anderen Nutzern Rechte einräumen.

Beispiel: Zugriffskontrolle bei relationalen Datenbanken (3)

- Rechte werden als Privilegien modelliert (*granter*, *grantee*, *object*, *action*, *grantable*)

```
GRANT SELECT, UPDATE (Day, Flight)
      ON TABLE Diary
      TO Art, Zoe
```

```
REVOKE UPDATE (Day, Flight)
    ON TABLE Diary
    FROM Art
```

- Option *grantable* erlaubt es, Rechter weiter zu geben)

User Art gibt Rechte weiter an User Joe

```
GRANT SELECT, UPDATE (Day, Flight)
    ON TABLE Diary
    TO Art
    WITH GRANT OPTION
```

```
GRANT SELECT, UPDATE (Day, Flight)
      ON TABLE Diary
      TO Zoe
      WITH GRANT OPTION
```

Zusammenfassung

- Zugriffskontrolle
  - überwacht und steuert den Zugriff auf Ressourcen (Dateien, Prozesse, Hardware etc.) eines Systems
  - kümmert sich um die Verwaltung von Rechten (und Rollen)

- ACL und Capabilites als wesentliche Bausteine für die Implementierung

- Role-Based Access Control (RBAC) erlaubt ein direktes Nachbilden bekannter Organisations- und Rechtestrukturen in Unternehmen und ist de-facto-Standard für Unternehmenssoftware