Informationssicherheit:
Zugriffskontrolle

Teil 2: Mandatory Access Control

Wiederholung: MAC & BLP

- MAC: Systembestimmte Zugriffskontrolle (*mandatory access control*)
- Bekanntestes Beispiel für MAC: **Bell-LaPadula**-Modell (BLP)
- Idee:
  - Klassifizierung von Subjekten und Objekten mit **Labels**
  - **Simple Security-Property** (No read-up-Regel)
  - ***-Property** (No write-down-Regel)

Weiteres MAC-Beispiel: Chinese Wall-Modell

- Brewer und Nash, 1989
- Modell aus dem Finanzbereich
- Ausgangssituation:
  - Berater sind für verschiedene Unternehmen/Organisationen (z.B. Banken, Ölfirmen) tätig
  - Insiderwissen darf nicht genutzt werden

Chinese Wall-Modell: Interessenkonflikte

- Vermeidung von **Interessenkonflikten** (***conflict of interest, COI)*** :

  Ein Berater darf nicht mehrere Firmen/Organisationen aus derselben Konfliktklasse **(*COI class*)** beraten

- Beispiele für Konfliktklassen:
  - Beispiele: Banken, Ölfirmen

Chinese Wall: Begrifflichkeiten

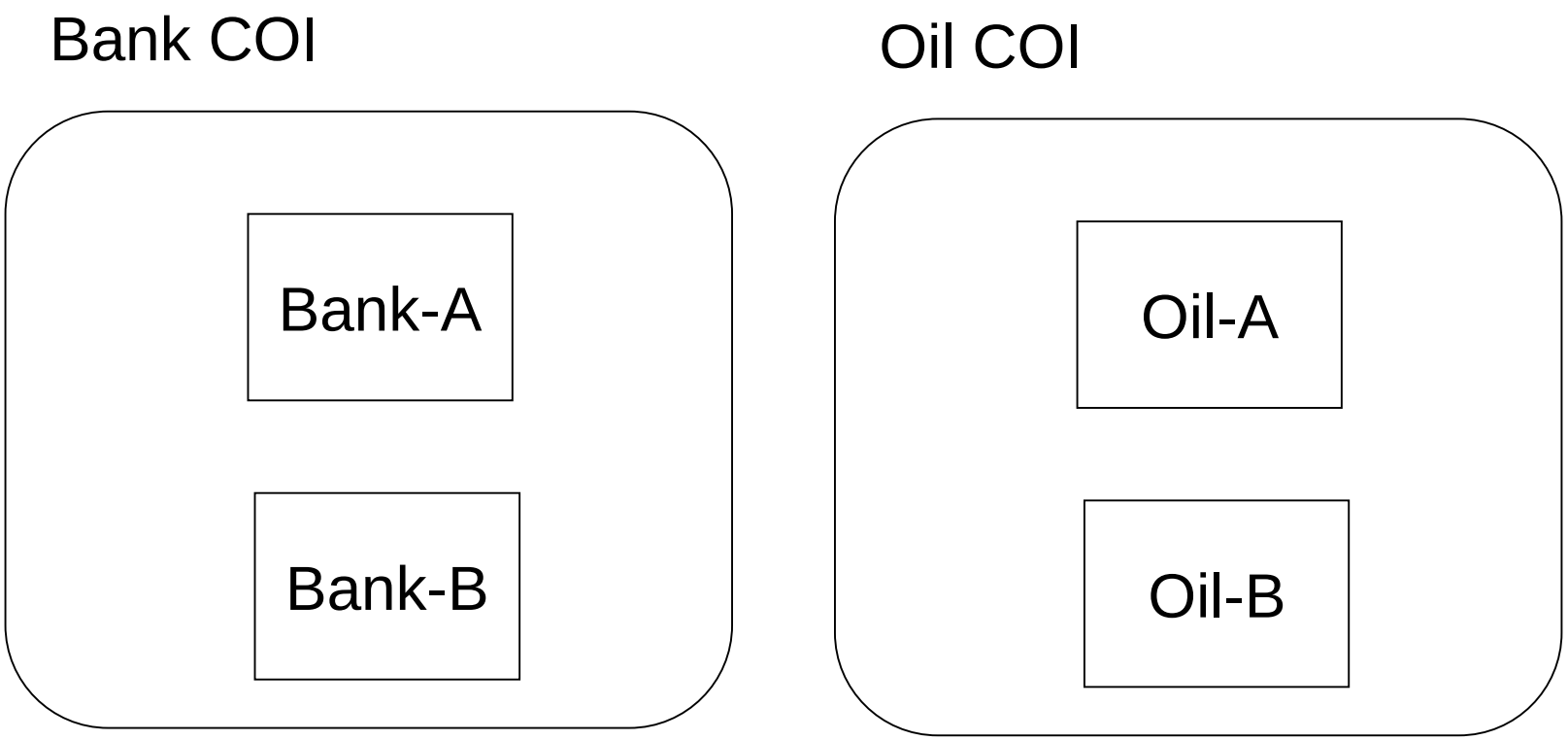
- Drei Arten von Begrifflichkeiten (hierarchische Strukturierung der Daten):

1. Objekte wie in BLP, Zugriffsrechte read, write

2. Objekte gehören eindeutig zu Firmen/Organisationen
   - Menge der Firmendaten: Company Data Set (**CD**)
   - **CD(o)** gibt das eindeutige Company Data Set von Objekt **o** an

3. Firmen/Organisationen gehören eindeutig Konfliktklassen (COIs) an
   - COIs enthalten also die CDs der dazugehörigen Firmen/Organisationen

Beispiel: Aufteilung in Konfliktklassen



Simple Security Property

- **Simple Security Property:**

  Ein Berater darf auf ein Objekt o lesend zugreifen, nur falls eine der folgenden Bedingungen erfüllt ist:

  1. Der Berater hat schon auf andere Objekte dieser Firma/Organisation zugegriffen oder
  2. Das Objekt gehört zu einer Konfliktklasse, für die der Berater noch kein Objekt gelesen hat.

- Berücksichtigung der Zugriffshistorie; Aufbau einer Chinese Wall

Beispiel

- Ausgangssituation:
  - Bank-A und Bank-B
  - Oil-A und Oil-B
  - Alice und Bob sind Berater
  - Annahme: Bob berät Bank-B und hat noch nie eine Ölfirma beraten
- Auf die folgenden Objekte darf Bob dann noch lesend zugreifen:
  - Alle Objekte von Oil-A oder Oil-B
  - Alle Objekte von Bank-B (aber nicht von Bank-A)

Reicht die Simple Security-Regel aus?

- **Beispiel**:
  - Bob berät Bank-B und Oil-A;
  - Alice berät Bank-A und Oil-A.
  - Alice hat insbesondere auch Schreibrechte auf Objekte aus Oil-A.
  - Simple Security-Regel gilt offenbar.
- Ein Trojanisches Pferd könnte Informationen über Bank-A von Alice nach Oil-A kopieren.
  - Bob hat nun mittels Oil-A Zugriff auf diese Informationen.
  - Bob hat also auf Objekte aus Bank-B und (indirekt) Bank-A-Zugriff.
- Vergleich mit der Problematik „Verdeckte Kanäle“
- Einführung einer neuen Regel erforderlich

Chinese Wall-Modell: *-Property

- ***-Property**:

  Ein Berater hat schreibenden Zugriff auf ein Objekt **o**, wenn er bislang nur Objekte aus der Firma von Objekt **o**, aber keine anderen gelesen hat.

- Für unser Beispiel bedeutet dies:

  Alice darf **nicht** Objekte in Klasse Oil-A schreiben, weil sie lesenden Zugriff auch auf Objekte aus Bank-A besitzt.

Bewertung/Einordnung

- Chinese Wall-Modell zugeschnitten auf bestimmte Problematik aus dem Finanzbereich
- Nur Vertraulichkeit, keine Integrität
- Umsetzung durch organisatorische Maßnahmen und weniger durch IT-Systeme

Andere Modelle für die Zugriffskontrolle

- Biba-Modell (Ken Biba, 1977), Integrität (aber kein Informationsfluss)
  - Ab MS-Vista enthält Windows Biba-Konzepte (z.B. Beschränkung von Änderungen von Registry-Einträgen)
- Clark-Wilson (David Clark, David Wilson, 1987): Sicherheitsmodell für Banken- und Buchhaltungsprogramme
- Non-Interference-Modelle (z.B. Meseguer, Goguen, 1982):

  $\rightarrow$ Verdeckte Kanäle
- Domain Type Enforcement (s. folgende Folie)

Domain-Type-Enforcement (DTE)

- Boebert und Kain, 1985
- **Idee**:
  - Zuordnung von Objekten (z.B. Dateien, Sockets, Prozesse) zu Typen
  - Zuordnung von Subjekten (z.B. Nutzer, Prozesse) zu Domains
- Zusätzliche Zugriffsmatrix
  - nicht für einzelne Objekte und Subjekte, sondern für Typen und Domains

  Beispiel:

  `M(admin_d, writable_t)={create, write, read, search}`
- Vertraulichkeit und Integrität
- Anwendung: SELinux

Rollenbasierte Zugriffskontrolle

- **Idee der rollenbasierten Zugriffskontrolle (*role-based access control, RBAC*)**:

  Knüpfung der Rechte nicht mehr direkt an Benutzer, sondern an Rollen (***roles***)
- Rollen entsprechen häufig Funktionen/Aufgaben, die in einer Organisation (Krankenhaus, Bank, Behörden, Unternehmen) auftreten
- Geeignet für hierarchisch aufgebaute Organisationen
- Andere Form von MAC
- Beispiel: Bank
  - Kassierer, Kassenprüfer, Zweigstellenleiter
- State-of-the-Art!

RBAC gemäß ANSI-Standard

- ANSI-Standard für RBAC (ANSI-INCITS 359-2004)
- Entitäten des ANSI-Standards für RBAC:
  - **Benutzer** (***users***) sind Personen (und keine Prozesse, vgl. Subjekt-Begriff)
  - **Zugriffsberechtigungen** (***permissions***) sind Paare *(Operation,Objekt)*.
    Beispiele: (read, File1), (read, File2), (debit, account1), (credit, account2)
  - Eine **Rolle** ist eine Sammlung von Zugriffsberechtigungen.
- Rollen werden Benutzern zugewiesen. Eine Rolle ist also Mittler zwischen Benutzern und Berechtigungen.
- Rollen werden von Benutzern in **Sitzungen** (***sessions***) aktiviert.
  - Unterscheidung von Rollenzuweisung und -aktivierung. Welches Sicherheitsprinzip?
  - Least Privilege
- Formalisierung?

Formalisierung von RBAC gemäß ANSI-Standard

Mengen:

- Users, Roles, Permissions, Sessions

Relationen:

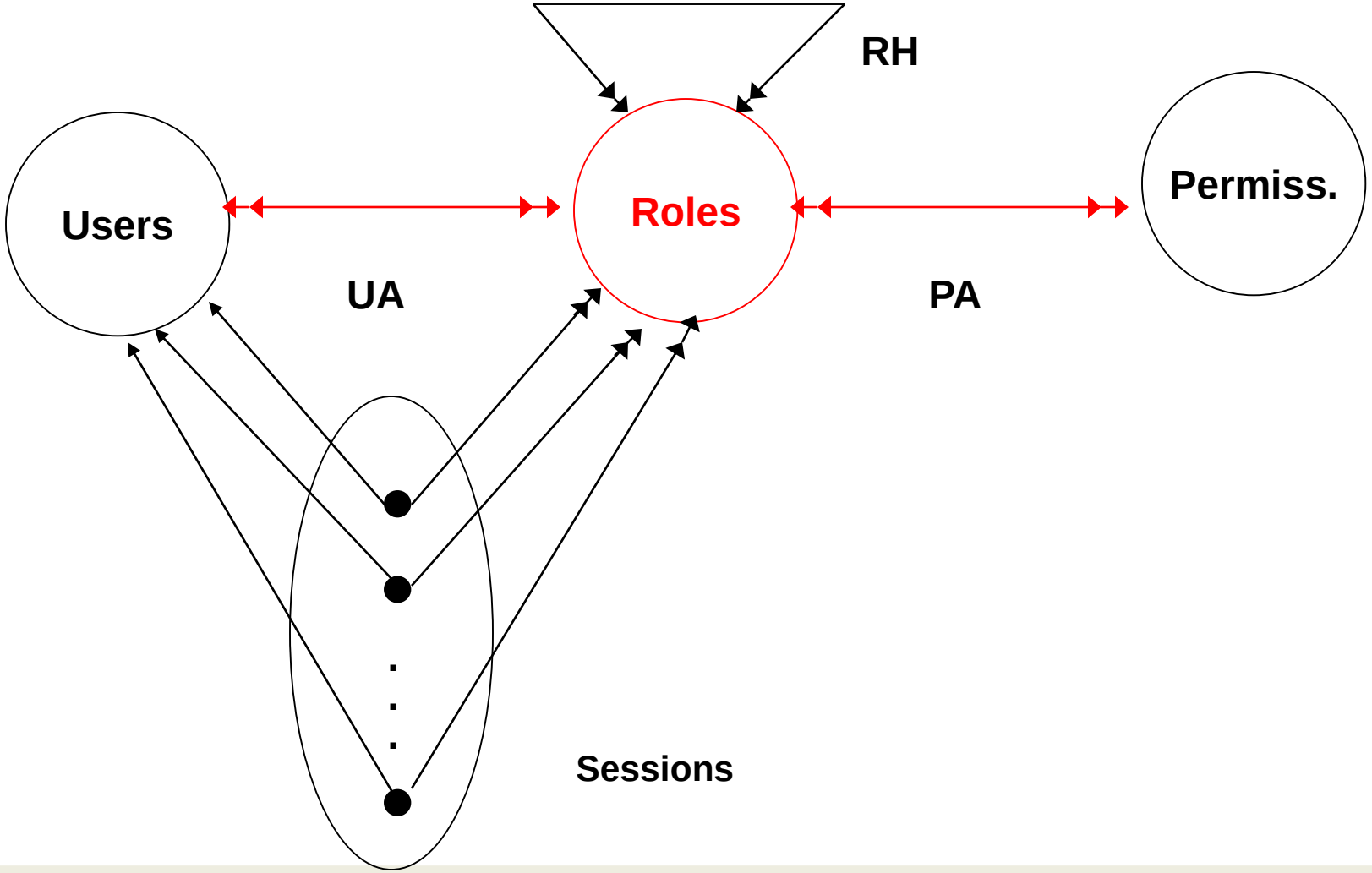
- UA $\subseteq$ Users $\times$ Roles (***user assignment***)
  Zuordnung von Benutzern zu Rollen
- PA $\subseteq$ Permissions $\times$ Roles (***permission assignment***)
  Zuordnung von Zugriffsberechtigungen zu Rollen
- RBAC soll Organisationsstrukturen nachbilden, also Einführung von Rollenhierarchien (***role hierarchies***):
  RH $\subseteq$ Roles $\times$ Roles, RH ist eine partielle Ordnung auf Roles $\times$ Roles

Funktionen:

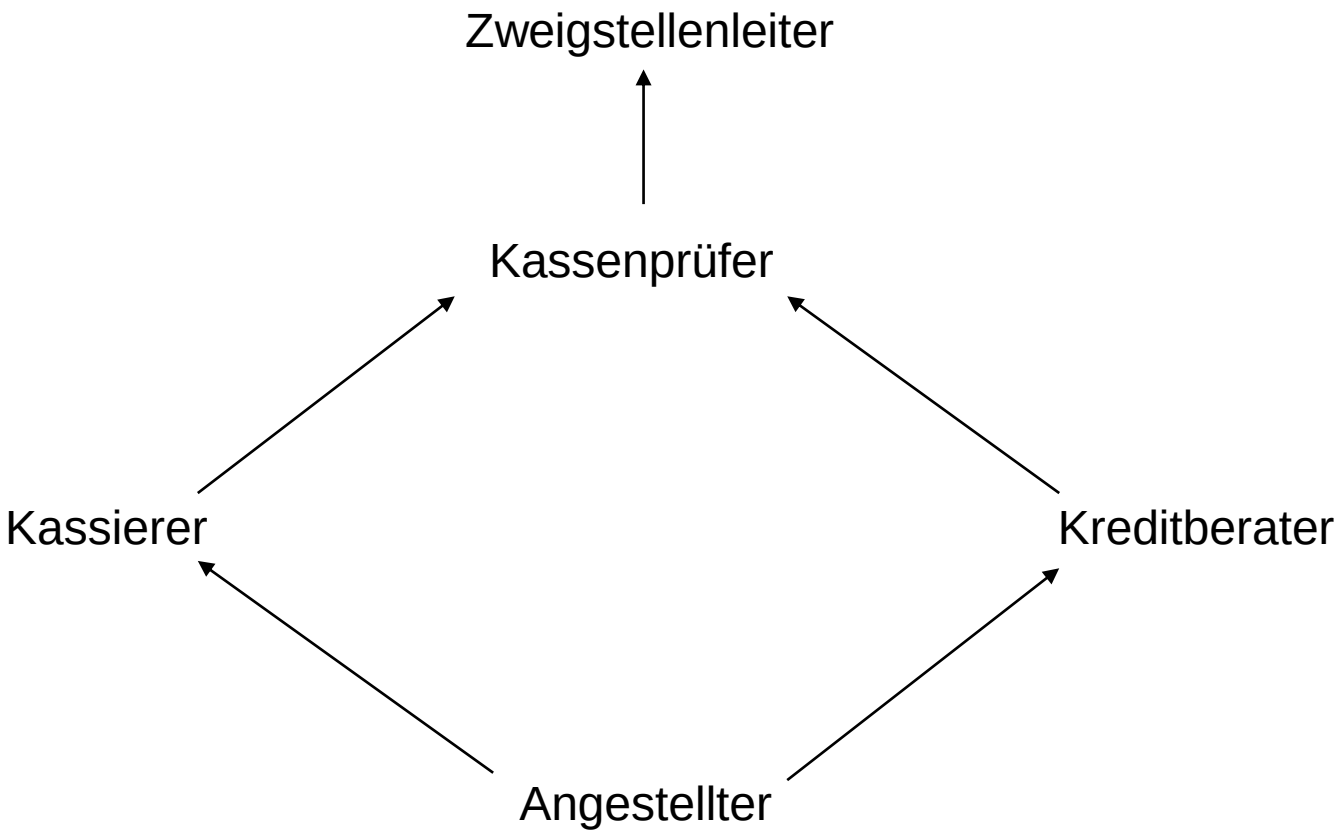
session_user : Sessions $\rightarrow$ User – eindeutiger Besitzer einer Session

session_roles: Sessions $\rightarrow 2^{Roles}$ – die in einer Session aktivierten Rollen

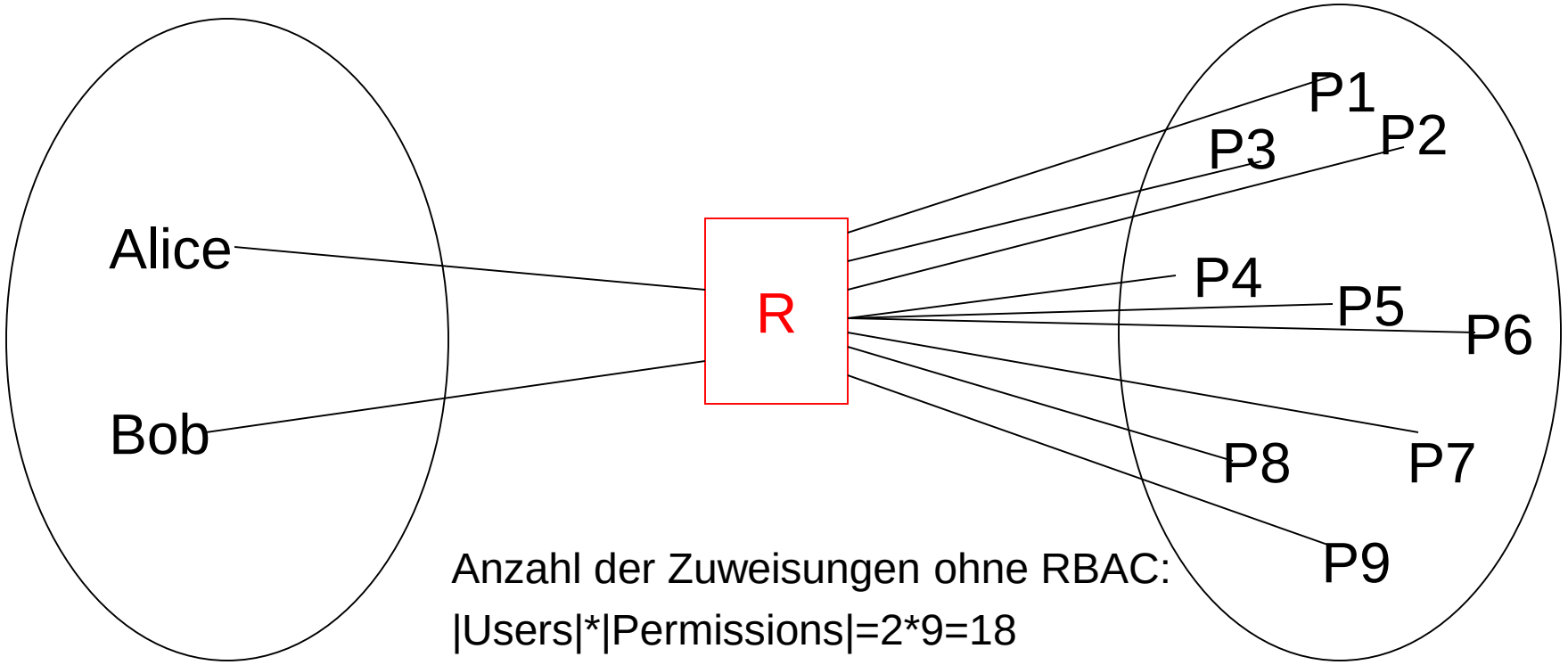
Komponenten von RBAC



Rollenhierarchie: Beispiel



Warum vereinfacht RBAC das Berechtigungsmanagement ?



Anzahl der Zuweisungen ohne RBAC:
|Users|*|Permissions|=2*9=18
Anzahl der Zuweisungen mit RBAC:
|Users|+|Permissions|= 2+9=11

Erweiterung von RBAC: Separation of Duty – Motivation

- Niedergang der Barings-Bank, 1995

„In a fatal mistake, the bank allowed Leeson to remain Chief Trader while being responsible for settling his trades, a job that is usually split. This had made it much simpler for him to hide his losses.“

„Der Starhändler musste keine Rechenschaft über seine Transaktionen geben, eine Compliancestelle oder eine eigene Abwicklungsabteilung gab es nicht. Im Grunde kontrollierte sich Leeson weitgehend selbst.“ ARD, 27.02.2020, 25 Jahre später

- Ähnlicher Fall, aber mit etwas geringeren Folgen:
  Betrug bei der Société Générale, Jérôme Kerviel, 2008

Erweiterung von RBAC: Separation of Duty (SoD)

- Separation of Duty (SoD) – Aufgabentrennung
  - Manchmal auch „Vieraugen-Prinzip“ genannt
- Natürliche Abbildung von organisationsinternen Kontrollregeln mittels RBAC
- Beispiele für die Aufgabentrennung:
  - Die Rollen **Kassenprüfer** und **Kassierer** dürfen nicht von ein und derselben Person angenommen werden.
  - Trennung der Rollen **Kunde** und **Kreditberater**

Statische *Separation of Duty*

- Im RBAC-Formalismus:

  CR (Menge von Konfliktrollen) – z.B. CR = {Kassierer, Kassenprüfer}

  roles(u) = {r ∈R| (u,r) ∈ UA} – Rollen von u

  Statische Separation of Duty:
  ∀ u ∈ User: |roles(u) ∩ CR| ≤ 1
  „User u darf höchstens eine Konfliktrolle annehmen“***.***

- Darf ein **Kreditberater** auch **Kunde** in der Bank sein, in der er arbeitet?
  - Einführung von dynamischer SoD

Dynamische *Separation of Duty*

- Trennung von Rollen nicht beim UA, sondern beim Aktivieren von Rollen in Sitzungen (flexibler als statische SoD)

- Im RBAC-Formalismus:

  active_roles(u) = {r ∈ R | (u,r) ∈ UA ∧
  ∃ s ∈ S.(user(s)=u ∧ r ∈ session_roles(s))}

  „Die **u** zugewiesenen Rollen, die in mindestens einer Session **s** von **u** aktiviert sind“

  Dynamic Separation of Duty:

  ∀ u ∈ User: | active_roles(u) ∩ CR| ≤ 1

  „User u darf höchstens eine Konfliktrolle **aktivieren**“**.**

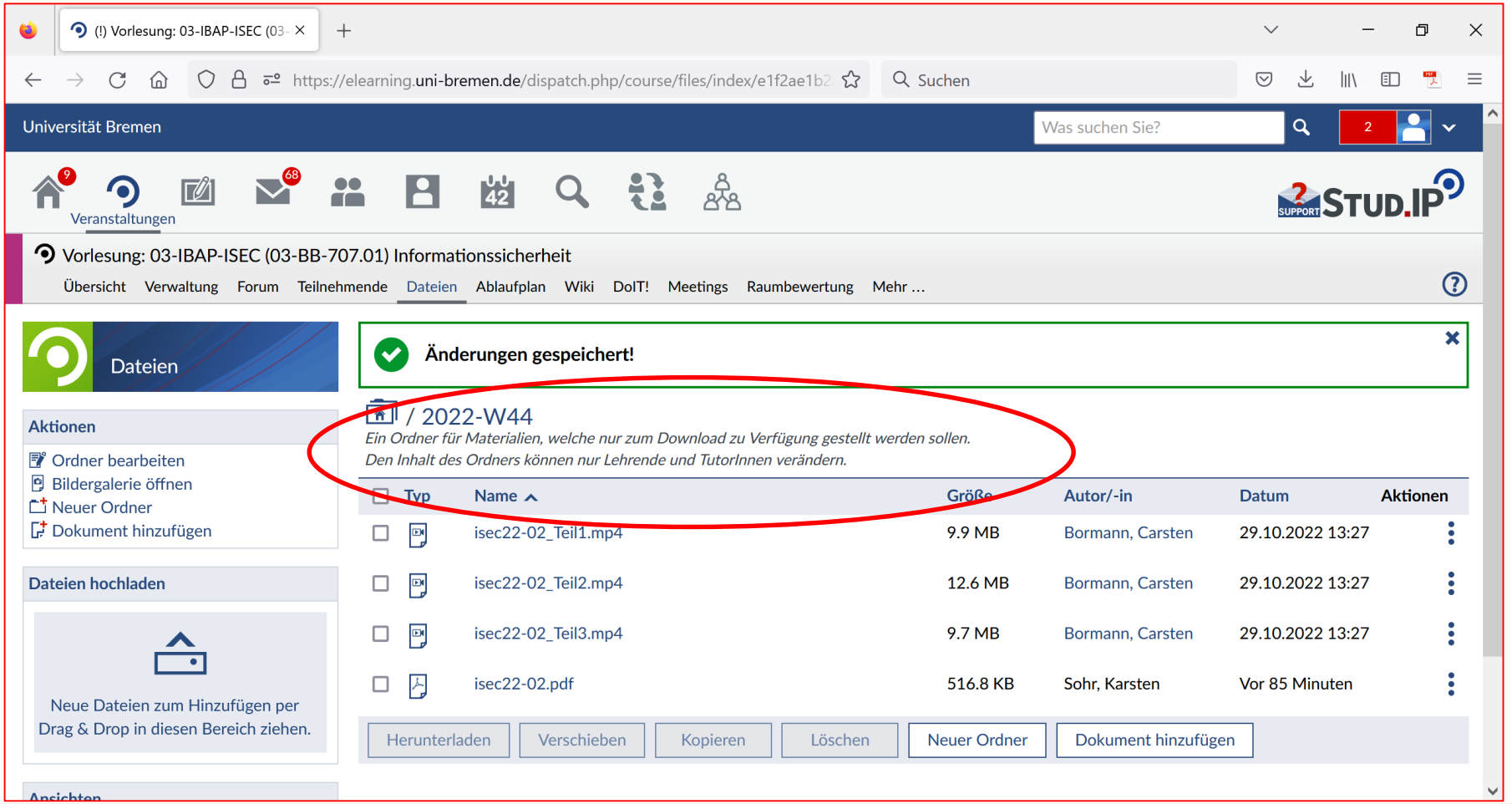
Authorization Constraints

- Statische und dynamische SoD sind Ausprägungen des allgemeineren Konzeptes der Authorization Constraints.
- Rollenbasierte Authorization Constraints sind Einschränkungen der Rollenrelationen und drücken organisatorische Regeln aus
- Andere Formen von rollenbasierten Authorization Constraints:
  - Verschiedenste (teilweise applikationsabhängige) SoD-Beschänkungen:
    - Object-based Dynamic SoD: Ein Mitarbeiter aus der Finanzabteilung eines Unternehmens darf nicht gleichzeitig eine Bestellung vorbereiten **und dann** genehmigen.
    - Werden oft im Workflow-Kontext verwendet
  - Kardinalitätsbeschränkungen
  - Prerequisite Roles
  - Temporale Beschränkungen

Einsatzgebiete von RBAC

- Betriebssysteme (RACF von IBM, Windows NT/2000/7/10 zumindest als Gruppenkonzept, SELinux - später)
- Middleware (z.B. Web Services)
- Rollenbasierte Administrationswerkzeuge wie z.B. DirXMetaRole von Evidian (früher Siemens), IBM Tivoli, Google IAM, AWS IAM (Amazon)
- Datenbanken (z.B. Oracle, Sybase, Informix)
- Enterprise Resource Planning Systeme wie z.B. SAP
- Krankenhausinformationssysteme, Bankenanwendungen (weite Verbreitung im Bankenbereich), E-Government, ...

RBAC in Stud.IP: Beispiel



Stärken

- Rollenkonzepte sind flexibel verwendbar
- Vereinfachtes Berechtigungsmanagement
  (wenn bspw. Benutzer häufig wechseln, dann braucht i.d.R. nur die Relation UA angepasst zu werden)
- Abbildung von hierarchischen Organisationsstrukturen
- Intuitive Abbildung auf Geschäftprozesse
- Möglichkeit, organisationsinterne Sicherheits- und Kontrollregeln abzubilden (Authorization Constraints)
- Sowohl Vertraulichkeit als auch Datenintegrität
  (BLP kann durch RBAC simuliert werden)

Schwächen

- Woher bekommt man überhaupt die Rollen, die für eine Organisation relevant sind? (Problem des Role Engineerings)
- Automatische Zuweisung von Berechtigungen zu Rollen wird nicht ausreichend unterstützt (manuelle Zuweisung in großen Organisationen nicht praktikabel)
- Rollenproliferation (Beispiele: Krankenhaus mit 300 Rollen, Verwaltung der Universität: 800 SAP-Rollen)
  - Faustregel: |Roles|/|User|= 0.04 (20.000 Mitarbeiter in der Verwaltung!?)
- Viele Projekte zur Einführung von RBAC scheitern aus o.g. Gründen
- Kein Schutz gegenüber Covert Channels (wie bei allen Modellen für die Zugriffskontrolle)

SELinux – Security Enhanced Linux

- Von der NSA (National Security Agency) entwickelt, 2001
- Erweiterung von Linux um MAC-Konzepte; frei verfügbar
- Unterstützung von SUSE-, Debian-, Ubuntu-Linux, auch von Android
- Mittlerweile Einsatz in britischen Behörden, um BLP-artige Zugriffskontrolle umzusetzen

Ziele

- Konfigurierbare Sicherheitsrichtlinien (***security policies***) für die Zugriffskontrolle, u.a.:
  - RBAC
  - BLP
  - Chinese-Wall
  - Biba
  - Clark-Wilson
  - Eigene, organisationsabhängige Security Policies
- Trennung von Definition und Durchsetzung (***enforcement***) der Sicherheitsrichtlinien
- Trennung von Prozessen

SELinux – Konzepte

Drei Zugriffskontrollkonzepte/-modelle:

1. DTE als Basis:
   - Der Einfachheit halber aber keine Domains, sondern nur Typen

2. RBAC:
   - Kein PA, sondern Zuordnung von DTE-Typen zu Rollen
   - UA wie immer
   - Auch Unterstützung von Authorization Constraints

3. IBAC (identity-based access control):
   - Jeder Prozess behält seine ID:
     im Gegensatz zum s-Bit in Unix/Linux

Sicherheitsarchitektur

- Sicherheitsrichtlinien werden in verschiedenen Textdateien definiert
- Integration eines Security Servers in den Linux-Kernel
- Laden der Sicherheitsrichtlinien beim Start in den Kernel
  - Dynamisches Nachladen von Sicherheitsrichtlinien möglich
- Überprüfung von Zugriffen durch den Security Server:
  - Werden beim Zugriff die definierten Sicherheitsrichtlinien verletzt?
  - Security Server als API, der Rest des Kernels ruft diesen auf (enforcement)
  - Subjekte und Objekte haben Security Identifier (SID), die bei der Überprüfung verwendet werden
  - SID: Handler auf Security Kontext, der die eigentlichen Sicherheitsinformationen (Rolle, UserID, Typen…) enthält (Indirektion)

Wie sehen Sicherheitsrichtlinien in SELinux aus?

- DTE-Zugriffsmatrix

  `allow type_1 type_2:class {perm_1 … perm_n}`

  `allow user_t tmp_t:dir {read search add_name remove_name}`

  „Jeder Nutzer kann im /tmp-Verzeichnis Dateien erzeugen und löschen“
- User Assignments

  `user username roles {role_1, …, role_n}`
- Type Assignments

  `role rolename types {type_1, …, type_n}`
- Zusätzlich: Datei für Authorization Constraints

Abschließende Anmerkungen

- Erhöhte Sicherheit durch:
  MAC-Unterstützung, Trennung von Prozessen, kein s-Bit
- Kompatibilität zu Linux-Distributionen/-Derivaten, z.B. SUSE, Debian, Android (Ubuntu setzt eher auf AppArmor)
- Flexible Konfigurationsmöglichkeiten von Sicherheitsrichtlinien (aber Dateien mit Sicherheitsregeln gut schützen!)

- Nachteil: Komplexität der Policy-Definition
  - Werkzeuge zur Policy-Analyse wünschenswert, Usability-Aspekt
  - Vorgegebene Policy-Definitionen

- Mehr über SELinux:
  https://github.com/SELinuxProject

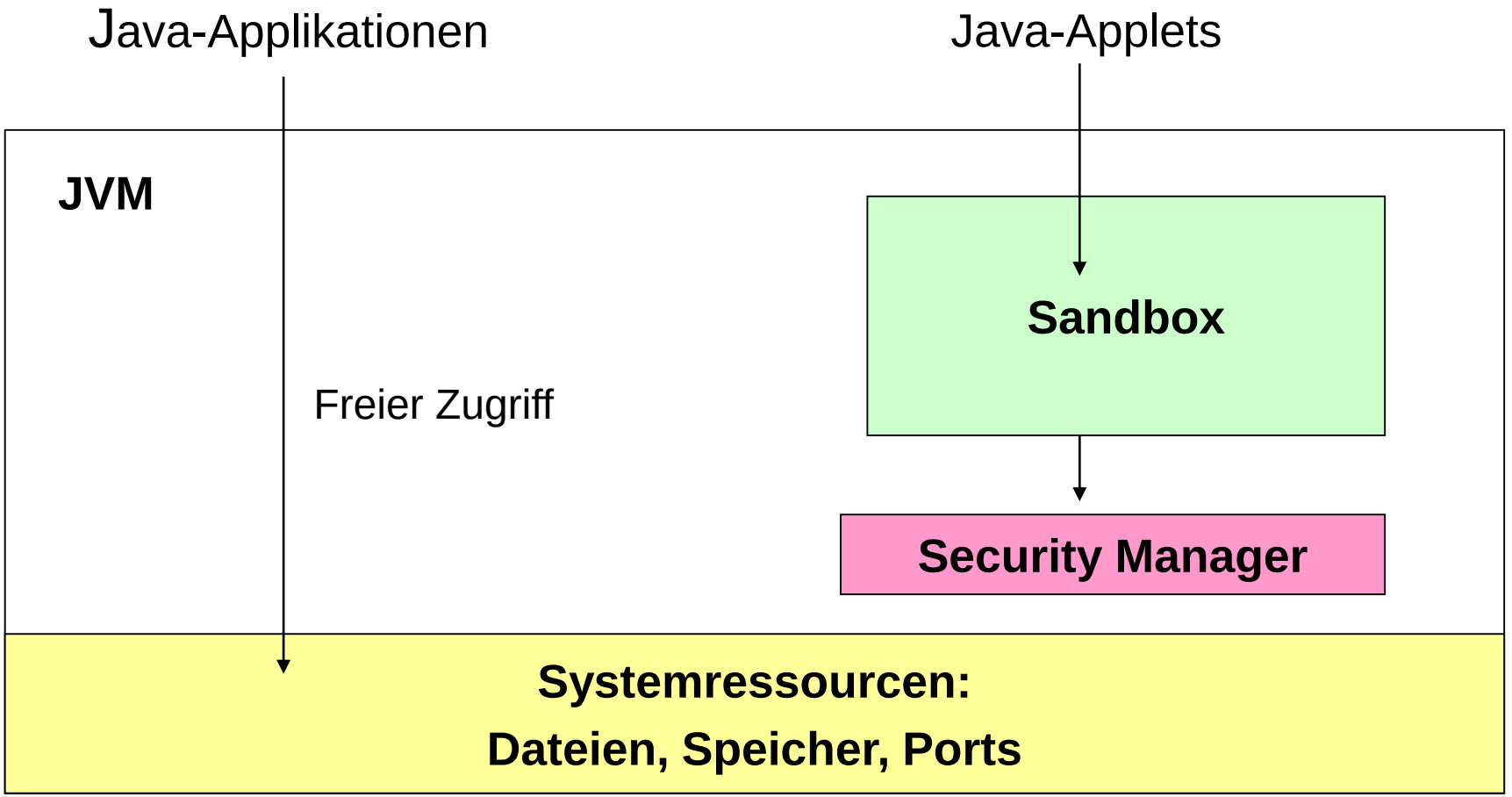
Sandboxing

- Einschränkung der Funktionalität von Programmen/Prozessen (z.B. durch eine virtuelle Maschine)
- „Programm kann nur in einem Sandkasten spielen.“

- Prominentes Beispiel: Java-Applets
  - Heute Renaissance im App-Konzept für Smartphones (App-Konzept aber deutlich umfassender)
  - Beispielsweise kein schreibender und lesender Zugriff auf Dateien, keine beliebigen Netzverbindungen, kein Zugriff auf nativen Code
  - Umsetzung dieser Sicherheitsrichtlinien durch den Java-Security Manager
    - Java-Systemklasse
    - Referenzmonitor

- Nicht einfach umzusetzen, wie die Java-Sicherheitsprobleme zeig(t)en

Java-Sandboxing im JDK 1.0 (veraltet)

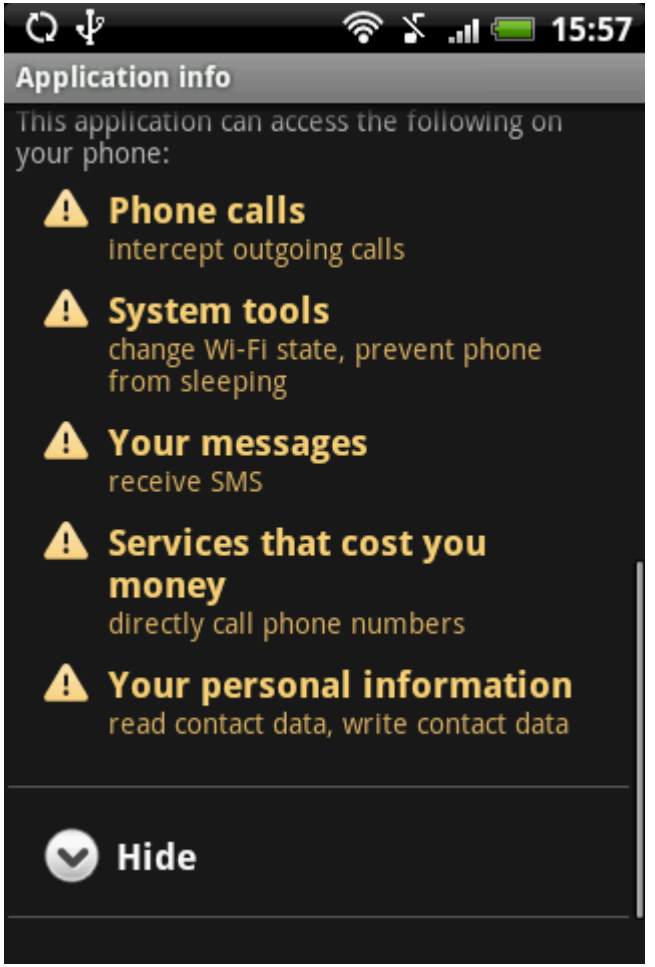


Sandboxing in Android

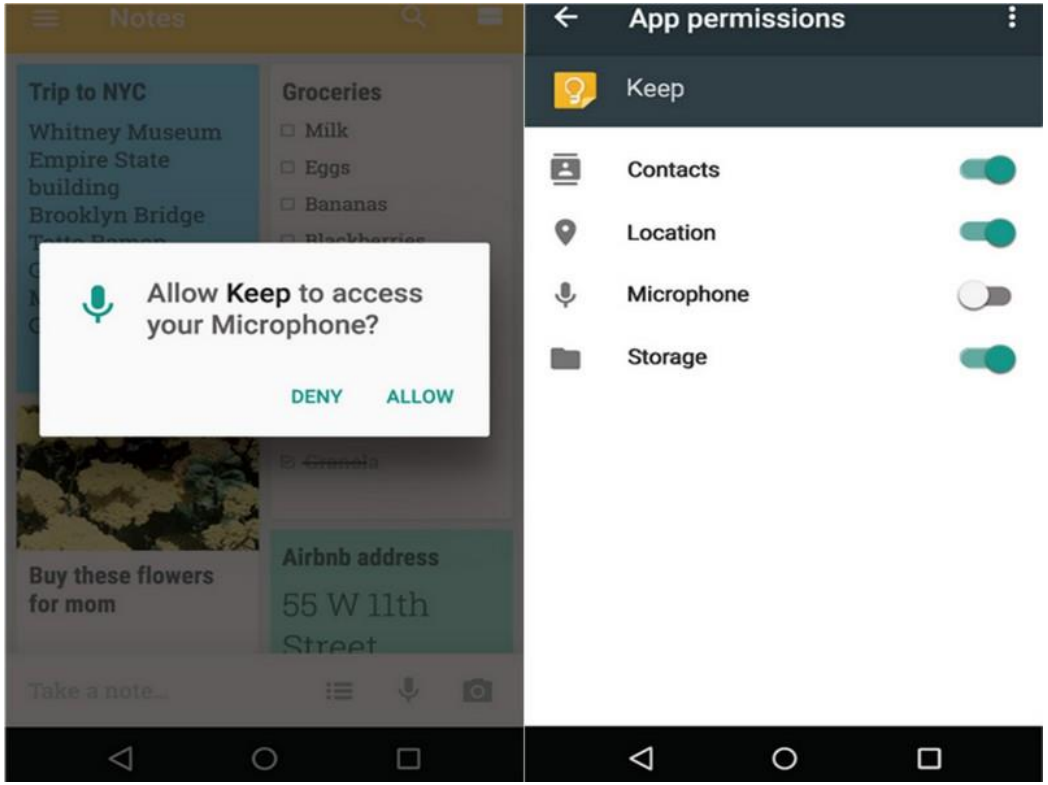
- Android eine der wichtigsten Smartphone-Plattformen (neben iOS), Marktanteil: > 80%
- Linux- und Java-basierte Plattform, mit einigen Änderungen
- Apps können auf das Gerät geladen werden
  - Aus Google Play
  - Aus anderen Quellen
- Erweitertes Applet-Konzept, mit den bekannten Risiken
- **Android-Sandbox:** u.a. Apps werden jeweils unter eigener Unix-User ID installiert

Android Permissions

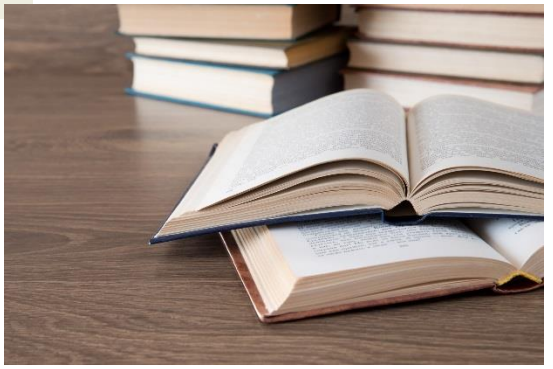
- Benutzer muss Berechtigungen bei der Installation freigeben (jedoch ab Android 6.0 zur Laufzeit!)
- Typische vordefinierte Berechtigungen:
  INTERNET, LOCATION, RECEIVE_SMS
- Problem:
  - Benutzer muss Entscheidung fällen
  - Deshalb: Berechtigung grob granular (z.B. INTERNET)



Runtime-Bestätigung ab Android 6.0



Literatur



- D. Elliott Bell, Leonard J. LaPadula. Secure Computer Systems: Mathematical Foundations. MITRE Corporation, 1973
- David F.C. Brewer and Dr. Michael J. Nash: *The Chinese Wall Security Policy*. In: IEEE (Hrsg.): *Proceedings of IEEE Symposium on Security and Privacy*. 1989, S. 206–214.
- Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. 1996. Role-Based Access Control Models. IEEE Computer 29, 2 (February 1996), 38-47.
- Peter Loscocco, Stephen Smalley. Meeting critical security objectives with Security-Enhanced Linux. *Proceedings of the 2001 Ottawa Linux Symposium*. 2001.
- Gary McGraw, Edward W. Felten. 1999. Securing Java: Getting Down to Business with Mobile Code. John Wiley & Sons, Inc., New York, NY, USA.