Informationssicherheit: 2023-W43

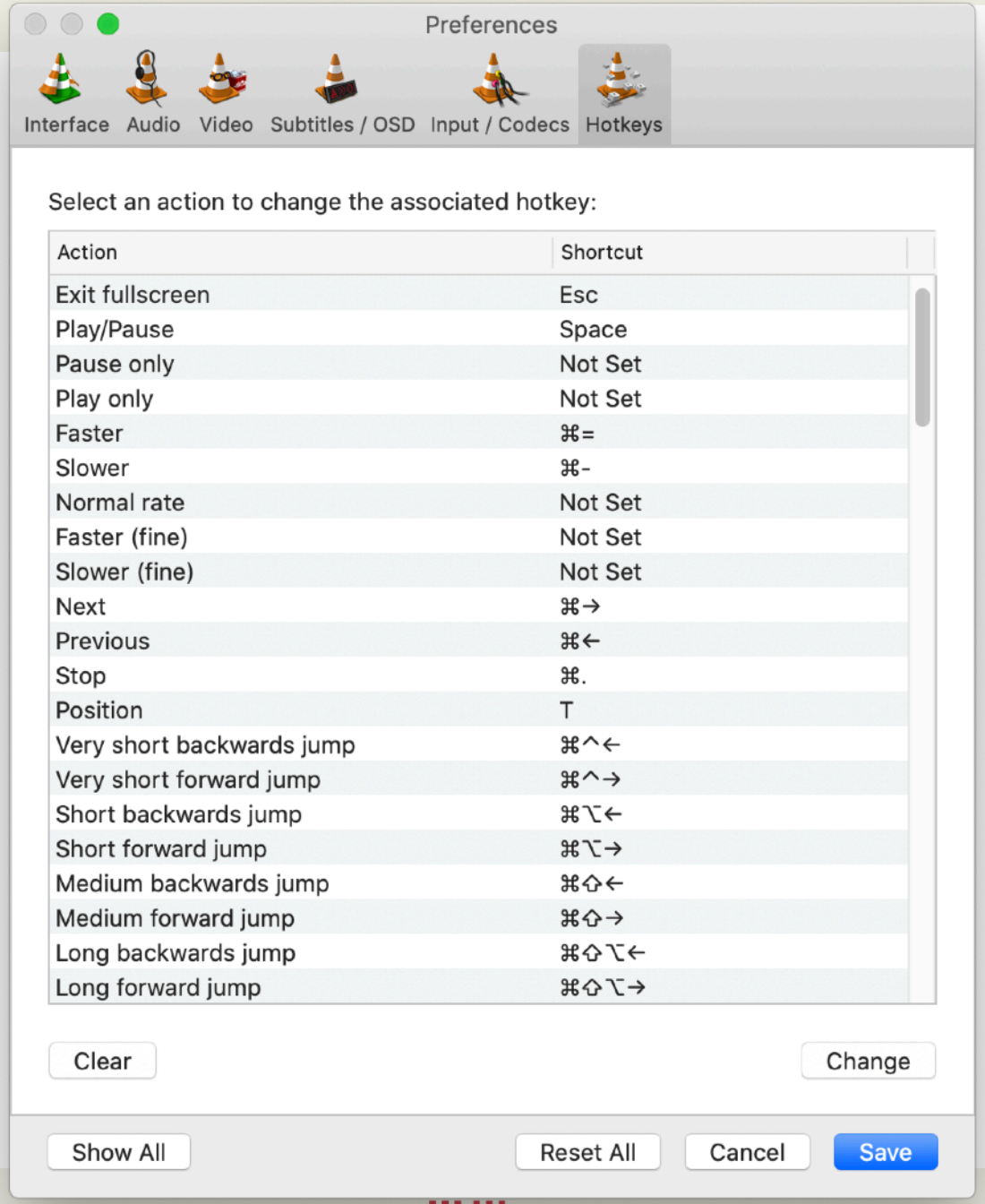
Einige **Designprinzipien**, und vor allem:

Zugriffskontrolle

Carsten Bormann

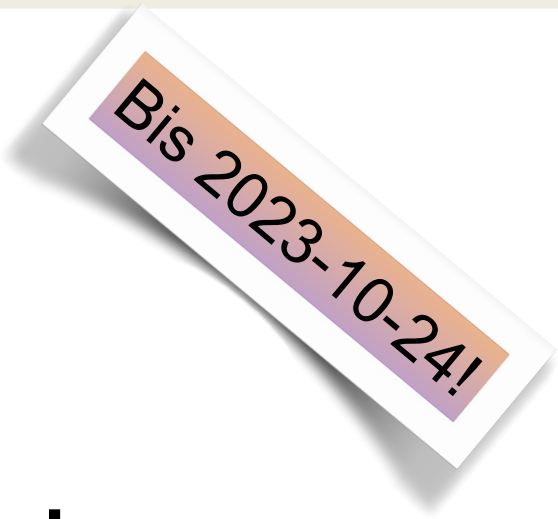
Empfehlung zum Durcharbeiten

- VLC Media Player (cross-platform)
- Tastaturkürzel für schneller/langsamer, vor zurück lernen
- Kann man sich auch selbst setzen (Preferences, Hotkeys)



Eselsbrücke: **CIA**

- **C**onfidentiality
- **I**ntegrity
- **A**vailability



Gruppenbildung

- Bitte Dreiergruppen bilden...
- ... und in Stud.IP eintragen

Gruppenbildung

Bis 2023-10-24!

- Sind alle in **Dreier**gruppen organisiert?
- Stud.IP...

Einige **Designprinzipien**

Carsten Bormann

Wo liegen die Schwachstellen?

- Schlechtes **Design**
  - (z.B. fehlende Kontrollen, zu grobe Rechtevergabe)
- Schlechte **Implementierung**
  - (z.B. Pufferüberläufe, schwache Mechanismen, Umgehungswege)
- Schlechte **Systemadministration**
  - (z.B. Account mit Standardpasswort, offene Ports in Firewall, Einsatz ungeeigneter Systeme und Werkzeuge)
- Schlechtes **Management**
  - (z.B. unklare Sicherheitspolitik, unklare Sicherheitsregeln, fehlendes Sicherheitsbewußtsein der Mitarbeiter, keine Mittel für Sicherheitsüberprüfungen)

Designprinzipien für sichere Systeme (1)

Saltzer, Schroeder (1975):

- **Principle of Economy of Mechanism**

  The protection mechanism should have a simple and small design.
- **Principle of Fail-safe Defaults**

  The protection mechanism should deny access by default, and grant access only when explicit permission exists.
- **Principle of Complete Mediation**

  The protection mechanism should check every access to every object.

Designprinzipien für sichere Systeme (2)

- **Principle of Open Design**

  The protection mechanism should not depend on attackers being ignorant of its design to succeed
  (no ***security by obscurity***).
  It may however be based on the attacker's ignorance of specific information such as passwords or cipher keys.

- **Principle of Separation of Privilege**

  The protection mechanism should grant access based on more than one piece of information.

Designprinzipien für sichere Systeme (3)

- **Principle of Least Privilege**

  The protection mechanism should force every process to operate with the minimum privileges needed to perform its task.
- **Principle of Least Common Mechanism**

  The protection mechanism should be shared as little as possible among users.
- **Principle of Psychological Acceptability**

  The protection mechanism should be easy to use (at least as easy as not using it).

Design principles for secure systems — Addendum (4)

- **Principle of Defense in Depth**
  There should be multiple layers of defense before a high-value target is compromised. (No Maginot lines.)

- **Principle of Securing the Weakest Link**
  The protection mechanism should not have weak spots that allow circumventing the well-secured parts. (Security often is a chain.)

- **Principle of Reluctance to Trust**
  The protection mechanism should not give unwarranted trust to any mechanism or entity. (Healthy skepticism.)

(Beyond the 8 principles listed by Saltzer/Schroeder)

IT-Sicherheit:
Zugriffskontrolle

AAA

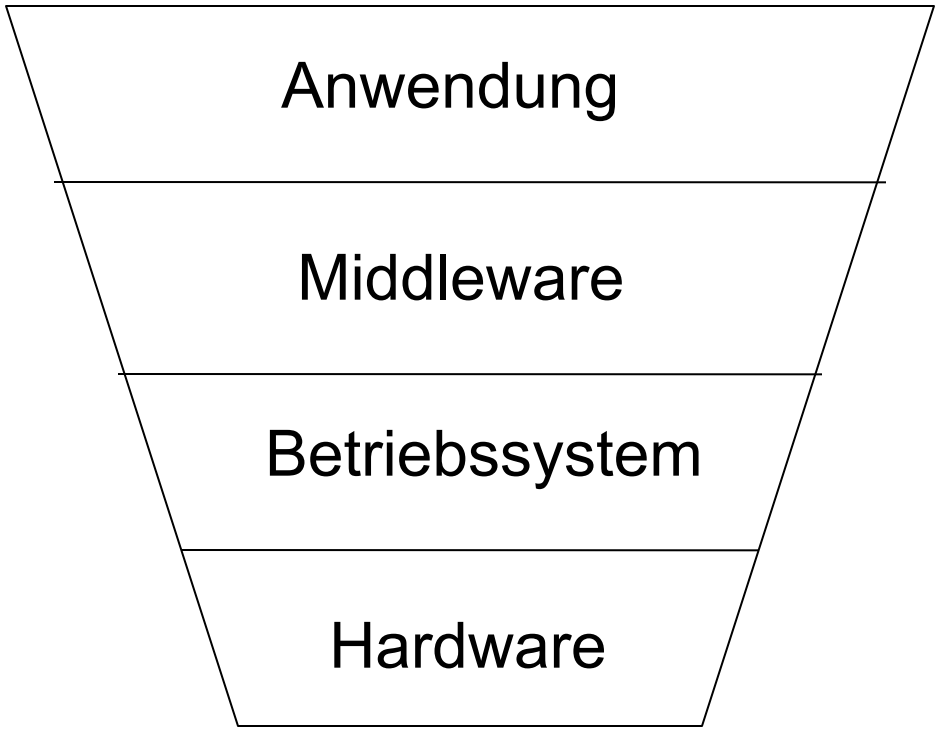
- Zugriffskontrolle (***access control***) regelt:
  „**Wer** darf in einem IT-System
  auf **welche Ressourcen**
  wie zugreifen“.

- Alternativer Begriff: Autorisierung (***authorization***)
  - AAA = Authentication, **Authorization**, Accounting
    (Authentisierung, **Autorisierung**, Abrechnung)

Grundbegriffe: Zugriffskontrolle

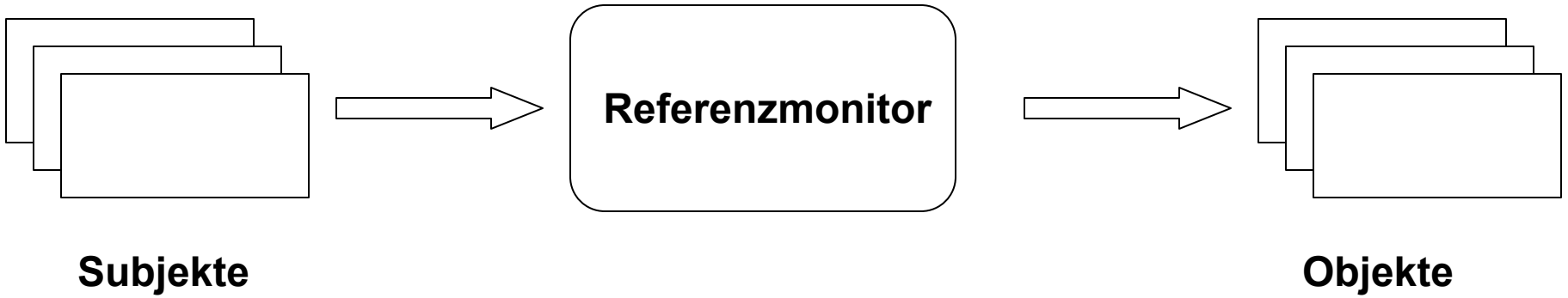
- Subjekt (***subject***): Initiator der Handlung
  - Benutzer (z.B. Alice, Bob), Gruppen, Prozesse, Rechner/Geräte
  - Genauer: ***principal*** als IT-Realisierung einer ***person*** oder Funktion
- Objekt (***object***): Gegenstand der Handlung, Ressource
  - Dateien, SQL-Tabellen, Konto, Prozesse
  - Achtung: Subjekte können auch Objekte sein.
- Berechtigung (***permission***)
  bzw. Zugriffsrecht (***access right***):
  - Was genau darf gemacht werden
  - Z.B.: lesen, schreiben, ausführen, löschen

Unterschiedliche Ebenen der Zugriffskontrolle



- Abhängigkeit der höheren Ebenen von den Zugriffskontrollmechanismen der niedrigeren Ebenen
- Zugriffskontrollmechanismen sind in höheren Ebenen komplexer
- Fehler in niedrigeren Ebenen können zum Umgehen der Sicherheitsmechanismen der höheren Ebenen führen

Referenzmonitor



Principle of Complete Mediation

The protection mechanism should check every access to every object.

Zugriffsmatrix (Lampson 1974)

- Zugriffskontrolle wird in ihrer grundlegenden Form dargestellt durch eine Zugriffskontrollmatrix (***access control matrix***) **M** mit:

  $M: S \times O \rightarrow 2^R$

  (**S** Menge der Subjekte, **O** Menge der Objekte,
  R Menge der Zugriffsrechte wie z.B. read, write, execute)

- Subjekt s darf Objekt o genau dann lesen, wenn

  $read \in M(s,o)$

Beispiel für eine Zugriffsmatrix

Subjekt/ Objekt	Datei 1	Datei 2	Datei 3	Prozess 1
Alice	{read, write}		{write}	
Bob		{execute}		{suspend}
Janet		{read}		
Frank	{read}			

M(Alice,Datei1) ={read,write}

Zugriffskontrolllisten (1)

- Zugriffsmatrix ist oft dünn besetzt, d.h. eine vollständige Speicherung ist zu speicherintensiv $O(|S|\times|O|)$
- **Zugriffskontrolllisten** (***access control lists, ACLs***): Realisierung der Matrix nach Spalten
- Subjekte und deren Zugriffsrechte werden **beim Objekt** gespeichert
- **Beispiel:**

  ACL(Datei1)=((Alice, {read,write}),(Frank,{read}))

Zugriffskontrolllisten (2)

Subjekt/ Objekt	Datei 1	Datei 2	Datei 3	Prozess 1
Alice	{read, write}		{write}	
Bob		{execute}		{suspend}
Janet		{read}		
Frank	{read}			

Zugriffskontrolllisten (3)

- Am häufigsten eingesetztes Konzept für die Zugriffskontrolle in gängigen Betriebssystemen (Windows NT (= 2000/XP), Unix/Linux)
- Vorteile:
  - Vergebene Zugriffsrechte sind effizient für Objekte bestimmbar
  - Rechterücknahme (pro Objekt) ist meist effizient realisierbar
  - Einfach zu implementieren

Zugriffskontrolllisten (4)

- Nachteile:
  - Bestimmen der Subjekt-Rechte i.d.R. sehr aufwendig (alle Objekte im System untersuchen)
  - Aufwendige ACL-Kontrollen bei jedem Zugriff
  - Schlechte Skalierbarkeit bei sehr dynamisch wechselnder Menge von Subjekten, falls ACLs für einzelne Subjekte/Nutzer vergeben werden

Beispiel: Zugriffskontrolle in Unix

- Subjekte:
  - Benutzer: User-ID, Group-ID, supplementary Group IDs (bis zu 16)
  - Prozess: User-ID/Group-ID (effective, real, saved), supplementary Group IDs
- Dateien:
  - Owner (User-ID), Group (Group-ID)
  - rwx-bits für: Owner, Group, Other
  - sUid, sGid: Rechteveränderung bei Dateiaufruf
- Verzeichnisse:
  - Ebenso, plus:
  - t-Bit (nur der Eigentümer darf Dateien in diesem Verzeichnis löschen)
  - sGid-Bit (steuert Gruppe bei Datei-Erzeugung)

Beispiel: Zugriffskontrolle in Unix

- Schutzbits sind einfache Form von ACL
  - Informationen für die Zugriffskontrolle den Dateien zugeordnet
- Schutzbit-ACL so nicht feingranular genug:
  - Alice kann nicht **Bob allein** nur Lese-Rechte auf eine bestimmte Datei geben
  - Uni Bremen: „grp“ zur Einrichtung von Gruppen durch Benutzer
- FreeBSD, Solaris haben zusätzliches ACL-Konzept
- POSIX-Kommandos: getfacl, setfacl
  - Und chmod-Erweiterungen: +a/–a/=a

FreeBSD-ACLs (POSIX .1e)

- Menge von Tripeln Typ:Name:Rechte
  - user::rwx (Dateieigner) 1
  - group::r-x (Gruppe der Datei) 3
  - other::r-x (Der Rest) 5
- Neu:
  - user:fritz:r-x (spezifischer Benutzer) 2
  - group:katzen:rwx (spezifische Gruppe) 4
- Einschränkung über **Maske** (chmod-Kompatibilität)
  - mask::r-x (schränkt alles außer 1 und 5 ein)

NFSv4-ACLs

- Allgemeinerer Nachfolger der POSIX-ACLs
- Liste von ACEs (Access Control Entry)
  - 17 permission bits, 4 inheritance flags
- Triviale ACL: Wie traditionelles UNIX

```
$ ls -v file.1
-r--r--r--   1 root     root      206663 May  4 11:52 file.1
     0:owner@:write_data/append_data/execute:deny
     1:owner@:read_data/write_xattr/write_attributes/write_acl/write_owner:allow
     2:group@:write_data/append_data/execute:deny
     3:group@:read_data:allow
     4:everyone@:write_data/append_data/write_xattr/execute/write_attributes
         /write_acl/write_owner:deny
     5:everyone@:read_data/read_xattr/read_attributes/read_acl/synchronize:allow
$
```

- Kommandos: ls –v, chmod A…

Windows-ACLs (1)

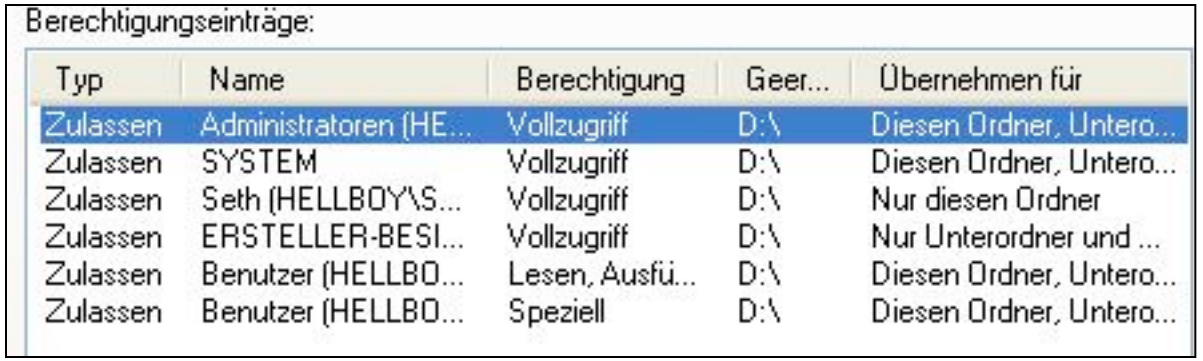
- Zugriffsrechte auf Dateien, Ordner oder Drucker werden mit **Access Control Lists** verwaltet.
- Pro **Objekt:**
  - Security identification number (SID) als Bezeichner für **Eigner** (*owner*)
  - **ACL**
- ACL besteht aus einzelnen Access Control Entries (ACEs):
  a) Typ des ACE (Erlauben, Verweigern; Überwachen)
  b) Subjekt (Benutzer oder Gruppe), für den/die der ACE gilt
     - SID als Bezeichner für Subjekt

     Berechtigungen, für die der ACE gilt (Bitmaske)

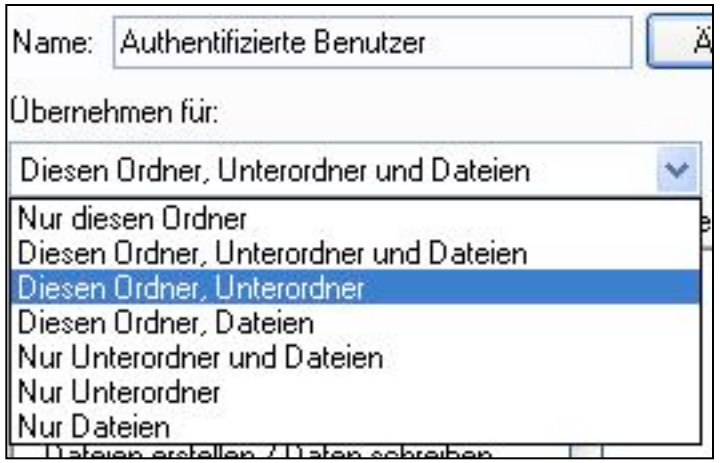
     Vererbbarkeit der ACE auf Unterobjekte

Windows-ACLs (2)

- ACL:

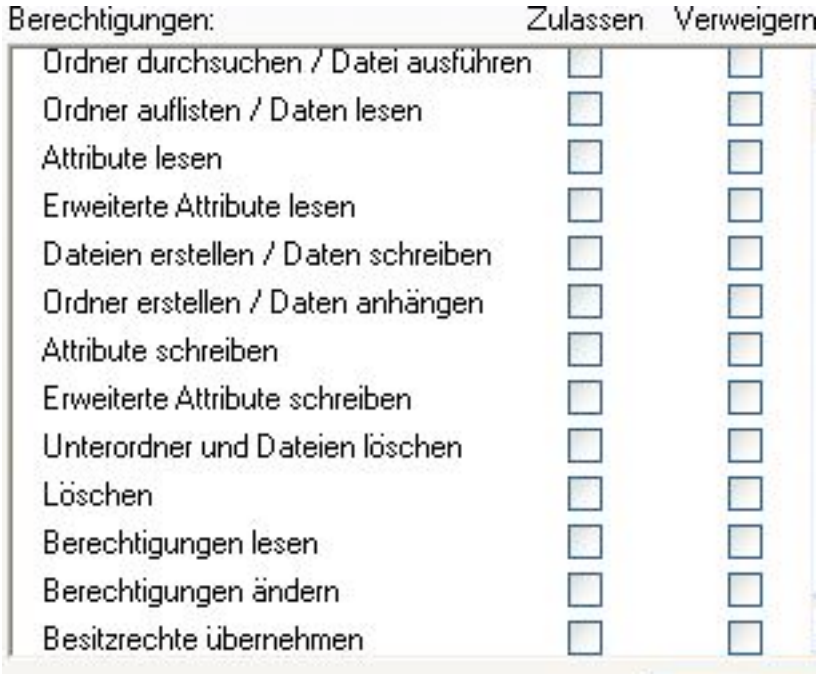


- Vererbbarkeit:



Windows-ACLs (3)

- Besitzer hat immer Vollzugriff
  - Besitzer kann auch eine Gruppe sein
- Alle ACEs werden in Reihenfolge ausgewertet
- GUI:
  Deny hat Vorrang vor Allow-Regeln
  - **Vorsicht**: Verbietet man einer Gruppe alles, gibt es keine Möglichkeit, Einzelnen Rechte zu geben!!



Capabilities (1)

- Zeilenweise Realisierung der Matrix
- Capability, Zugriffsticket mit Objekt-ID und Rechtebits
- Capability-Besitz berechtigt zur Wahrnehmung der Rechte
- Für jedes Subjekt **s** eine Capability-Liste (Clist)

  Beispiel Clist (Alice) = ((Datei1, {read,write}), ((Datei3, {write}))

Capabilities (2)

Subjekt/ Objekt	Datei 1	Datei 2	Datei 3	Prozess 1
Alice	{read, write}		{write}	
Bob		{execute}		{suspend}
Janet		{read}		
Frank	{read}			

Capabilities (3)

- Beispiele von Systemen mit Capability-Konzept:
  IBM System/38, Mach-µ-Kern (Ports), Amoeba
- UNIX: File Descriptor = flüchtige Capability
  - Deskriptor-Tabelle in User-Structure = C-List
  - Übergabe von Deskriptoren über UNIX-Domain-Sockets
- Achtung: **Keine** Capabilities in diesem Sinne:
  “POSIX Capabilities” (z.B. von Linux ab 2.2 unterstützt)
  - Besseres Wort: Privileges
- Renaissance des Capability-Konzeptes durch Zertifikate
  (X.509, später in VL)

Capabilities (4)

Vorteile :

- Einfache Bestimmung der Subjekt-Rechte
- Einfache Zugriffskontrolle: nur noch Ticketkontrolle!
- Einfache Delegation von Rechten andere Benutzer/Subjekte

Nachteile:

- Rechterücknahme schwierig (Ticket ist aus der Hand)
- Keine Subjekt-Ticket Kopplung,
  Besitz berechtigt automatisch zur Wahrnehmung der Rechte
- Objekt-Sicht auf Rechte schwierig: Wer darf was?

Bis hierher: DAC (Discretionary Access Control)

Bislang:

- **Subjekte** entscheiden über die Zugriffsberechtigungen
  (vgl. chmod in Unix/Linux oder Dialogfenster zur Rechtevergabe in Windows 2000/XP)
- Subjekt bestimmt weitgehend die Security Policy für „seine“ Objekte
  (**benutzerbestimmte Zugriffskontrolle** bzw. ***discretionary access control***)

Mandatory Access Control (MAC)

- In besonders sicherheitskritischen Bereichen mit z.B. hohen Anforderungen an die Vertraulichkeit ist benutzerbestimmte Zugriffskontrolle nicht angemessen
  - Typische Beispiele: Militär, Geheimdienste
- Einführung geeigneter Modelle, die **vom System** umgesetzt werden
- Systembestimmte Zugriffskontrolle (***mandatory access control***)
- Bekanntestes Beispiel für Mandatory Access Control: **Bell-LaPadula-Modell**

Das Bell-LaPadula-Modell (BLP)

- David Bell und Len LaPadula, 1973:
  auf Initiative der US Air Force

- Zweck:
  - Einfache Sicherheitsmechanismen für die Verifikation
  - Formales Sicherheitsmodell
  - Beschränkung des Informationsflusses, d.h. Sicherheitsziel ist die **Vertraulichkeit**

Bell-LaPadula-Modell: Konzepte (1)

Ein vereinfachtes BLP-Modell:

- Multilevel Security System (MLS)
  - Alle Subjekte und Objekte erhalten eine Sicherheitsmarkierung (***label***) gemäß der Stufe ihrer Vertraulichkeit wie z.B.
    top secret > secret > confidential > unclassified
  - Idee: Höher klassifizierte Daten dürfen nur von Mitarbeitern mit einer ausreichenden Sicherheitsstufe gelesen werden

Bell-LaPadula-Modell: Konzepte (2)

- Formalisierung der Idee:
  - **M** sei die Zugriffsmatrix.
  - **SC** sei eine Menge von Sicherheitsmarkierungen, und auf **SC** gelte eine partielle Ordnung ≤.
  - o ∈O: SC(o) ∈ SC sei die Sicherheitsmarkierung von Objekt o (***classification***)
  - s ∈S: SC(s) ∈ SC sei die Sicherheitsmarkierung von Subjekt s (***clearance***)

  - **Simple Security-Property (*no read-up*) :**
    ∀ s ∈ S. ∀ o ∈ O : *read* ∈ M(s,o) ⇒ SC(o) ≤ SC(s)

Bell-LaPadula-Modell: Konzepte (3)

- Die no-read-up-Regel reicht nicht: **Trojanische Pferde** könnten höher klassifizierte Daten einfach nach unten schreiben
- Die entscheidende Neuerung des BLP-Modelles ist mithin die zweite Regel:

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

  $\forall\, \mathbf{s} \in \mathbf{S}.\ \forall\, \mathbf{o} \in \mathbf{O} :\ \ \textit{write} \in \mathbf{M(s, o)} \Rightarrow \mathbf{SC(s)} \leq \mathbf{SC(o)}$
- Manchmal fordert man auch, dass sich die Sicherheitsmarkierungen weder für Objekte noch für Subjekte zur Systemlaufzeit ändern (Strong Tranquility Property)

Beispiele: BLP

- Viele Betriebssysteme bieten BLP-Erweiterungen: u.a.
  Sun Trusted Solaris 7, Trusted HP-Unix, Linux-Derivate
  - Erweiterter Referenz-Monitor zur Überprüfung der BLP-Regeln

- Daten-Diode (-„Pumpe“):
  - Daten können über ein Kommunikations-Device (Daten-Diode) **von Low nach High** transferiert (gepumpt) werden, aber nicht in der umgekehrten Richtung
  - Beispiel: NRL-Pump (US Naval Research Laboratory), 1993

Fazit: BLP

- Stärken:
  - Einfach zu implementieren (Anpassung des Referenzmonitors)
  - Formales Modell (man kann Sätze über Eigenschaften beweisen)
  - Gut geeignet zum Nachbilden hierarchischer Informationsflüsse:
    z.B. beim Militär, Geheimdienst
- Schwächen:
  - Beschränkte Ausdrucksfähigkeit:
    keine Integrität (blindes Schreiben)
  - Keine Modellierung von verdeckten Kanälen (***covert channels***),
    über die Informationen dann doch unberechtigt fließen können

Verdeckte Kanäle

- Zugriffskontrolle schützt den Datencontainer, nicht die Information
  - Möglichkeit von verdeckten Kanälen
- Verdeckte Kanäle (***covert channels***):

  Informationsfluss über einen Kanal, der nicht explizit zur Informationsübertragung vorgesehen ist

  Beispiele:
  - Ressourcen-Konflikt: High-Level-Prozess erzeugt eine Datei, so dass ein Low-Level-Prozess eine Fehlermeldung erhält, wenn eine Datei mit gleichem Namen erzeugt wird (1 Bit Information)
  - High-Level-Prozess kann gemeinsam genutzte Ressourcen (Position des Festplattenkopfes, Inhalt des Caches) in einem Zustand hinterlassen, so dass der Low-Level-Prozess anhand von Antwortzeiten zusätzliche Informationen gewinnen kann

Gruppenbildung

- Sind alle in **Dreier**gruppen organisiert?
- Stud.IP...

Dreiergruppen

- 1, 2, **3,** 4, 5