Access Control

Pierangela Samarati
Dipartimento di Tecnologie dell'Informazione
Università degli Studi di Milano
e_mail: samarati@dti.unimi.it

FOSAD 2008

Access control

- It evaluates access requests to the resources by the authenticated users and, based on some access rules, it determines whether they must be granted or denied.
  - It may be limited to control only *direct* access.
  - It may be enriched with inference, information flow, and *non-interference* controls

Access Control vs other services

Correctness of access control rests on

- Proper user identification/authentication $\Rightarrow$ No one should be able to acquire the privileges of someone else
- Correctness of the authorizations against which access is evaluated (which must be protected from improper modifications)

Access Control and Authentication

- Authentication also necessary for accountability and establishing responsability.
- Each principal (logged subject) should correspond to a single user $\Longrightarrow$ no shared accounts
- In open systems it should rely on authenticity of the information, in contrast to authenticity of the identity (authentication)

Policies, Models, and Mechanisms

In studying access control, it is useful to separate

- Policy: it defines (high-level) guidelines and rules describing the accesses to be authorized by the system (e.g., closed vs open policies)
  - often the term policy is abused and used to refer to actual authorizations (e.g., Employees can read bullettin-board)
- Model: it formally defines the access control specification and enforcement.
- Mechanism: it implements the policies via low level (software and hardware) functions

Separation between policies and mechanisms

The separation between policies and mechanisms allows us to:

- Discuss access requirements independent of their implementation
- Compare different access control policies as well as different mechanisms that enforce the same policy
- Design mechanisms able to enforce multiple policies

Access control mechanisms – 1

Based on the definition of a reference monitor that must be

- tamper-proof: cannot be altered
- non-bypassable: mediates all accesses to the system and its resources
- security kernel confined in a limited part of the system (scattering security functions all over the system implies all the code must be verified)
- small enough to be susceptible of rigorous verification methods

Access control mechanisms – 2

The implementation of a correct mechanism is far from being trivial and is complicated by need to cope with

- storage channels (residue problem) Storage elements such as memory pages and disk sectors must be cleared before being released to a new subject, to prevent data scavenging
- covert channels Channels that are not intended for information transfer (e.g., program's effect on the system load) that can be exploited to infer information

Assurance How well does the mechanism do?

Saltzer and Schroeder design principles – 1

- Economy of mechanism
  - the protection mechanism should have a simple and small design.
- Fail-safe defaults
  - the protection mechanism should deny access by default, and grant access only when explicit permission exists.
- Complete mediation
  - the protection mechanism should check every access to every object.
- Open design
  - the protection mechanism should not depend on attackers being ignorant of its design to succeed; it may however be based on the attacker's ignorance of specific information such as passwords or cipher keys.

Saltzer and Schroeder design principles – 2

- Separation of privileges
  - the protection mechanism should grant access based on more than one piece of information.
- Least privilege
  - the protection mechanism should force every process to operate with the minimum privileges needed to perform its task.
- Least common mechanism
  - the protection mechanism should be shared as little as possible among users.
- Psychological acceptability
  - the protection mechanism should be easy to use (at least as easy as not using it).

Access control development process

Multi-phase approach from policies ..... to mechanism

Passes through the definition of an

- Access control model that formally defines the access control specification and enforcement. The model must be
  - complete: It should encompass all the security requirements that must be represented
  - consistent: Free of contradictions; e.g., it cannot both deny and grant an access at the same time

The definition of a formal model allows the proof of properties on the system. By proving that the model is "secure" and that the mechanism correctly implements the model we can argue that the system is "secure" (according to our definition of secure).

Security policies

Security policies can be distinguished in

Access control policies: define who can (or cannot) access the resources. Three main classes:

- Discretionary (DAC) policies
- Mandatory (MAC) policies
- Role-based (RBAC) policies

Administrative policies: define who can specify authorizations/rules governing access control

Coupled with DAC and RBAC

Discretionary (DAC) policies: basic approaches

Discretionary policies

Enforce access control on the basis of

- the identity of the requestors (or on properties they have)
- and explicit access rules that establish who can or cannot execute which actions on which resources

They are called discretionary as users can be given the ability of passing on their rights to other users (granting and revocation of rights regulated by an administrative policy)

Access Matrix Model

It provides a framework for describing protection systems.

Often reported as HRU model (from later formalization by Harrison, Ruzzo, and Ullmann)

Called access matrix since the authorization state (or protection system) is represented as a matrix

Abstract representation of protection system found in real systems (many subsequent systems may be classified as access matrix-based)

Access Matrix Model – protection state

State of the system defined by a triple (S,O,A) where

- S set of subjects (who can exercise privileges)
- O set of objects (on which privileges can be exercised) subjects may be considered as objects, in which case $S \subseteq O$
- A access matrix, where
  - rows correspond to subjects
  - columns correspond to objects
  - $A[s, o]$ reports the privileges of s on o

Changes of states via commands calling primitive operations:

enter r into $A[s, o]$, **delete** r from $A[s, o]$, **create subject** s', **destroy subject** s', **create object** o', **destroy object** o'

Access Matrix – Example

	File 1	File 2	File 3	Program 1
Ann	own read write	read write		execute
Bob	read		read write	
Carl		read		execute read

Commands

Changes to the state of a system modeled by a set of commands of the form

command $c(x_1, \ldots, x_k)$
if r_1 in $A[x_{s_1}, x_{o_1}]$ and
r_2 in $A[x_{s_2}, x_{o_2}]$ and
.....
$r_m$in $A[x_{s_m}, x_{o_m}]$
then op_1
op_2
.....
op_n
end

$r_1, ..., r_m$ are access modes; $s_1, ..., s_m$ and $o_1, ..., o_m$ are integers between 1 and k. If m=0, the command has no conditional part.

Commands – Examples

command CREATE(subj,file)
create object file
enter Own *into* $A[$subj,file$]$ **end.**

command CONFER$_{\text{read}}$(owner,friend,file)
if Own in $A[$owner,file$]$
then *enter* Read *into* $A[$friend,file$]$ **end.**

command REVOKE$_{\text{read}}$(subj,exfriend,file)
if Own in $A[$subj,file$]$
then *delete* Read *from* $A[$exfriend,file$]$ **end.**

Transfer of privileges

Delegation of authority can be accomplished by attaching flags to privileges (e.g., $*$ copy flag; $+$ transfer only flag)

- copy flag (*): the subject can transfer the privilege to others

  command TRANSFER$_{\text{read}}$(subj,friend,file)
  if Read* in $A[$subj,file$]$
  then *enter* Read *into* $A[$friend,file$]$ **end.**

- transfer-only flag (+): the subject can transfer to other the privilege (and the flag on it); but so doing it loses the authorization.

  command TRANSFER-ONLY$_{\text{read}}$(subj,friend,file)
  if Read+ in $A[$subj,file$]$
  then *delete* Read+ *from* $A[$subj,file$]$
  enter Read+ *from* $A[$friend,file$]$ **end.**

State transitions

The execution of a command $c(x_1, ..., x_k)$ on a system state $Q = (S, O, A)$ causes the transition to a state Q' such that:

$$Q = Q_0 \vdash_{op_1^*} Q_1 \vdash_{op_2^*} ... \vdash_{op_n^*} Q_n = Q'$$

where

- $op_1^* \ldots op_n^*$ are primitive operations in c
- the formal parameters $(x_1, ..., x_k)$ in the definition are replaced by the actual parameters supplied at the command call.

If the conditional part of the command is not verified, then the command has no effect and $Q = Q'$.

The safety problem

Concerned with the propagation of privileges. Problem of giving an answer to the question:

- Given a system with initial configuration Q_0 does there exist a sequence of requests that executed on Q_0 produces a state Q' where a appears in a cell $A[s, o]$ that did not have it in Q_0?

(Not all leakages of rights are bad $\Longrightarrow$ trustworthy subjects are ignored in the analysis).

Some results:

- undecidable in general (reduced to the halting problem of a Turing machine)
- decidable for mono-operational command (i.e., containing a single primitive operation)
- decidable when subjects and objects are finite

Access Matrix – implementation

Matrix is generally large and sparse. Storing the matrix $\Rightarrow$ waste of memory space

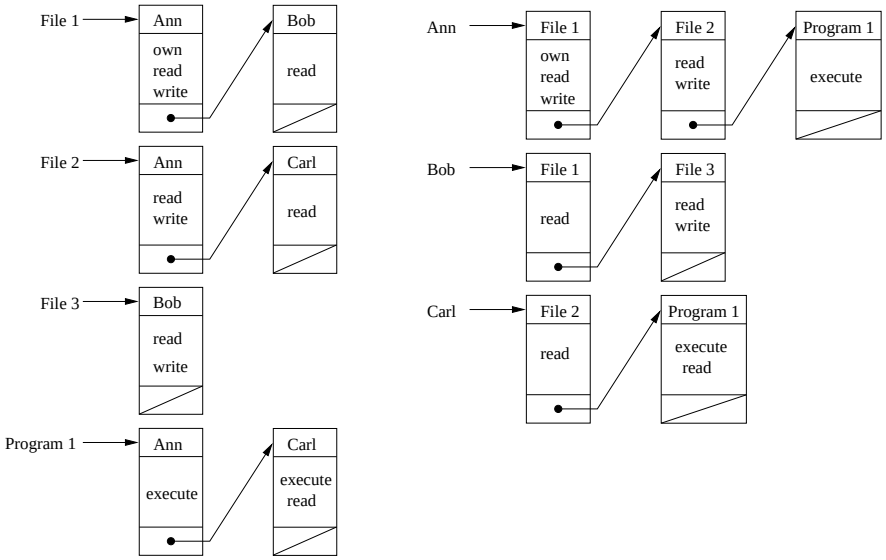
Alternative approaches

- Authorization table Store table of non-null triples (s,o,a). Generally used in DBMS.
- Access control lists (ACLs) Store by column.
- Capability lists (tickets) Store by row.

Authorization Tables

User	Access mode	Object
Ann	own	File 1
Ann	read	File 1
Ann	write	File 1
Ann	read	File 2
Ann	write	File 2
Ann	execute	Program 1
Bob	read	File 1
Bob	read	File 2
Bob	write	File 2
Carl	read	File 2
Carl	execute	Program 1
Carl	read	Program 1

Access control lists vs. Capability Lists



ACL vs Capabilities

- ACLs require authentication of subjects
- Capabilities do not require authentication of subjects, but require *unforgeability* and control of propagation of capabilities.
- ACLs provide superior for access control and revocation on a per-object basis.
- Capabilities provide superior for access control and revocation on a per-subject basis.
- The per-object basis usually wins out so most systems are based on ACLs.
- Some systems use abbreviated form of ACL (e.g., Unix 9 bits)

DAC weaknesses

Discretionary access controls constraint only direct access

No control on what happens to information once released

$\Longrightarrow$ DAC is vulnerable from Trojan horses exploting access privileges of calling subject

Trojan Horse: Rogue software. It contains a hidden code that performs (unlegitimate) functions not known to the caller.

Viruses and logic bombs are usually transmitted in the form of Trojan Horse

The Trojan Horse problem

File Market

Aug. 00; product X; price 7,000
Dec. 00; product Y; price 3,500
Jan. 01; product Z; price 1,200

owner Jane

The Trojan Horse problem

File Market

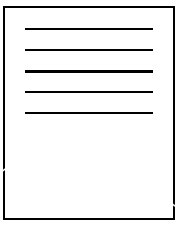
Aug. 00; product X; price 7,000
Dec. 00; product Y; price 3,500
Jan. 01; product Z; price 1,200

owner Jane

John

The Trojan Horse problem

Application



File Market

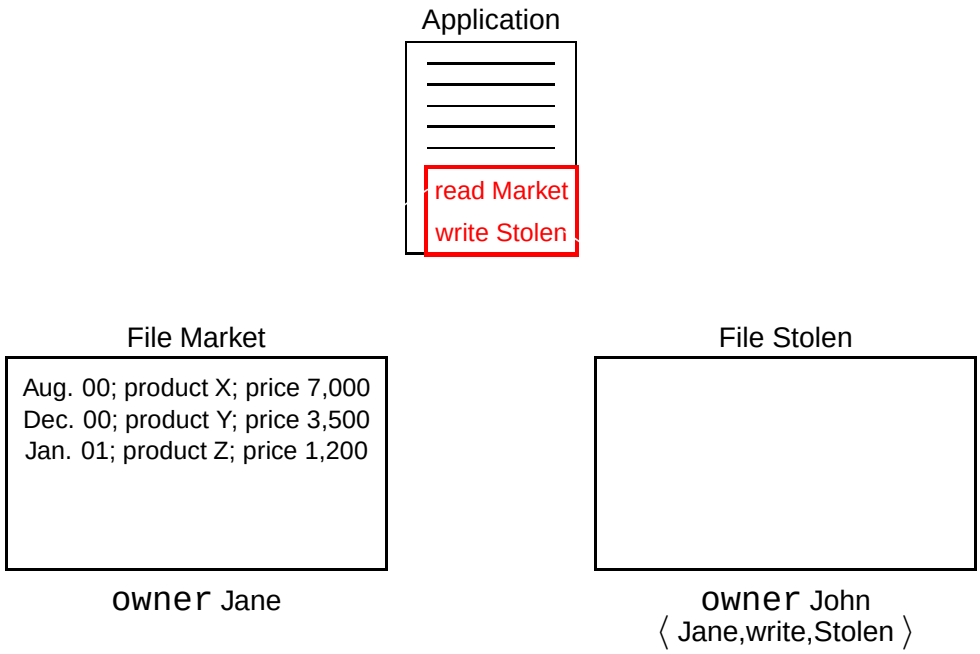
Aug. 00; product X; price 7,000
Dec. 00; product Y; price 3,500
Jan. 01; product Z; price 1,200

owner Jane

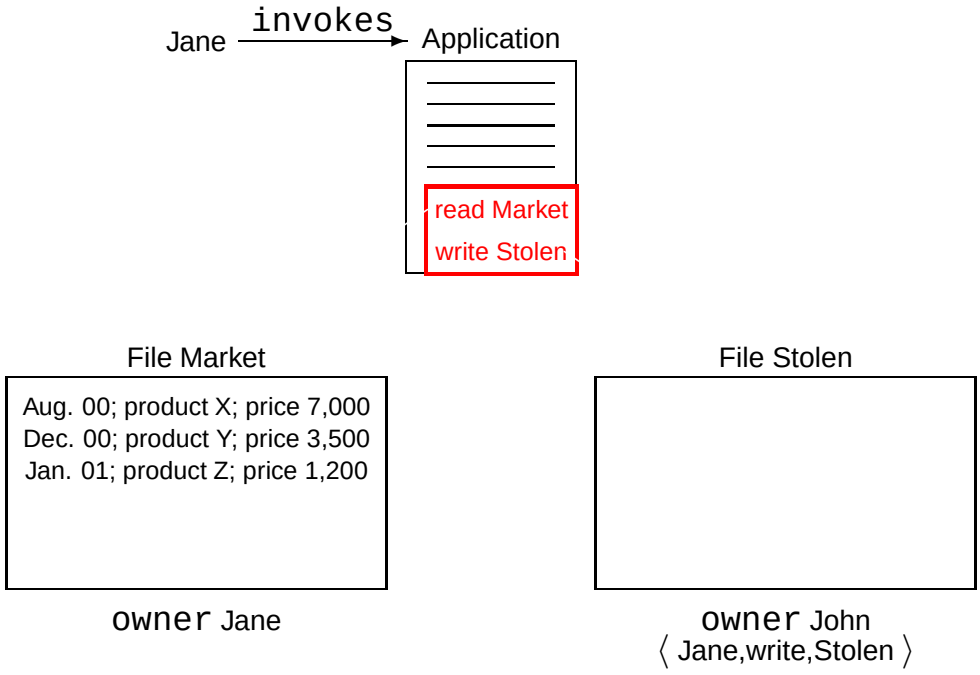
File Stolen

owner John
⟨ Jane,write,Stolen ⟩

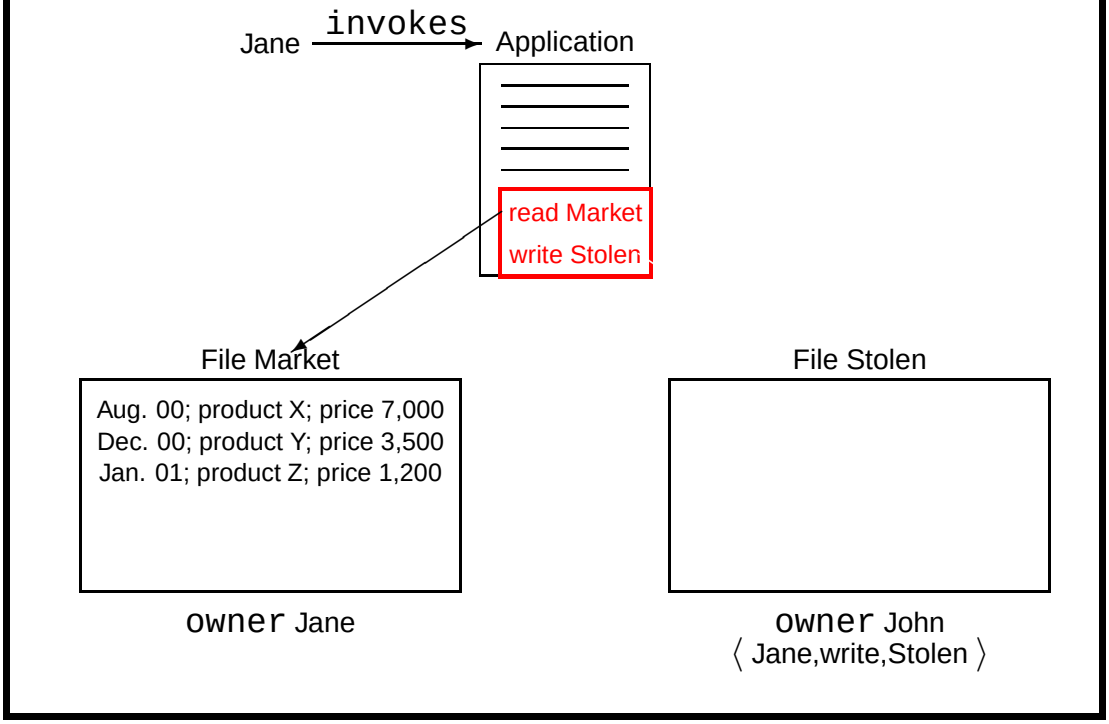
The Trojan Horse problem



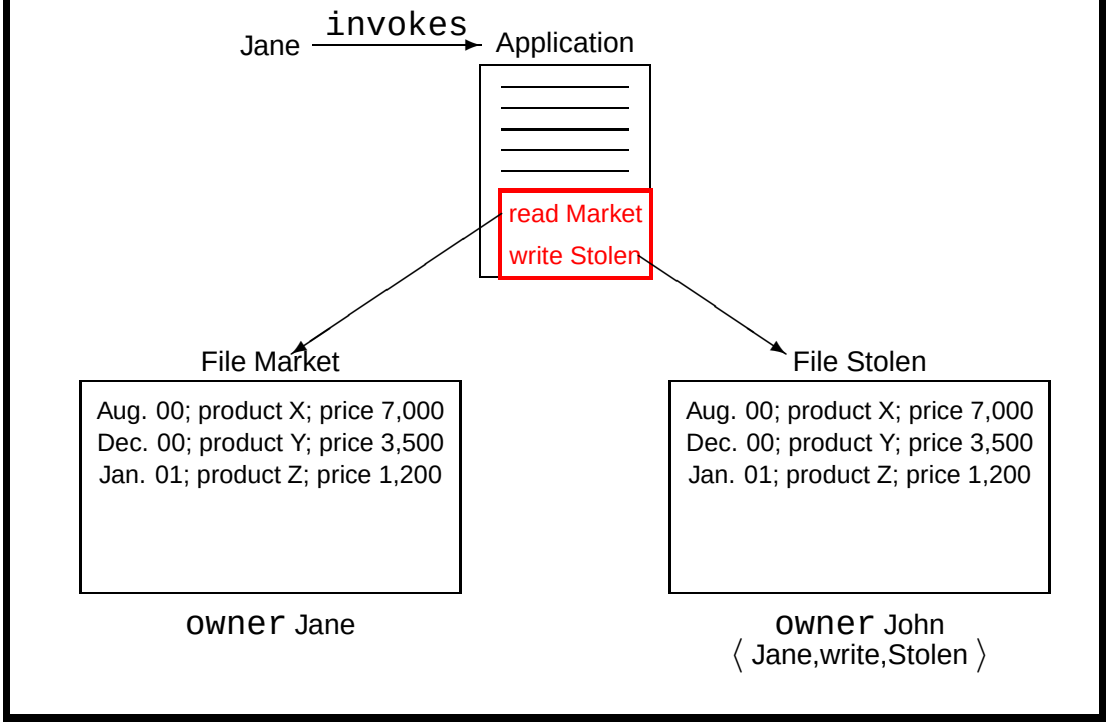
The Trojan Horse problem



The Trojan Horse problem



The Trojan Horse problem



Mandatory (MAC) policies

Mandatory policies

Mandatory access control: Impose restrictions on information flow which cannot be bypassed by Trojan Horses.

Makes a distinction between users and subjects operating on their behalf.

- User Human being
- Subject Process in the system (program in execution). It operates on behalf of a user.

While users may be trusted not to behave improperly, the programs they execute are not.

Mandatory policies

Most common form of mandatory policy is multilevel security policy.

- Based on classification of subjects and objects.
- Two classes of policies
  - Secrecy-based (e.g., Bell La Padula model)
  - Integrity-based (e.g., Biba model)

Security classification

Security class usually formed by two components

- Security level element of a hierarchical set of elements. E.g., TopSecret(TS), Secret(S), Confidential(C), Unclassified(U)

  $$TS > S > C > U$$

  Crucial (C), Very Important (VI), Important (I)

  $$C > VI > I$$

- Categories set of a non-hierarchical set of elements (e.g., Administrative, Financial). It may partion different area of competence within the system. It allows enforcement of "need-to-know" restrictions.

The combination of the two introduces a partial order on security classes, called dominates

$$(L_1, C_1) \succeq (L_2, C_2) \Longleftrightarrow L_1 \geq L_2 \wedge C_1 \supseteq C_2$$

Classification Lattice

Security classes together with $\succeq$ introduce a lattice $(SC, \succeq)$

Reflexivity of $\succeq$ $\forall x \in SC : x \succeq x$

Transitivity of $\succeq$ $\forall x, y, z \in SC : x \succeq y, y \succeq z \Longrightarrow x \succeq z$

Antisymmetry of $\succeq$ $\forall x, y \in SC : x \succeq y, y \succeq x \Longrightarrow x = y$

Least upper bound $\forall x, y \in SC : \exists\, !z \in SC$

- $z \succeq x$ and $z \succeq y$
- $\forall t \in SC : t \succeq x$ and $t \succeq y \Longrightarrow t \succeq z$.

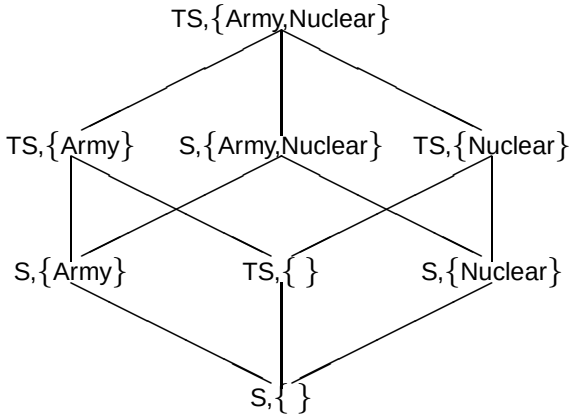
Greatest lower bound $\forall x, y \in SC : \exists\, !z \in SC$

- $x \succeq z$ and $y \succeq z$
- $\forall t \in SC : x \succeq t$ and $y \succeq t \Longrightarrow z \succeq t$.

Classification Lattice – example

Levels: Top Secret (TS), Secret (S)

Categories: Army, Nuclear



- lub(⟨TS,{Nuclear}⟩,⟨S,{Army,Nuclear}⟩) = ⟨TS,{Army,Nuclear}⟩
- glb(⟨TS,{Nuclear}⟩,⟨S,{Army,Nuclear}⟩) = ⟨S,{Nuclear}⟩

Semantics of security classifications

Each user is assigned a security class (clearance).

A user can connect to the system at any class dominated by his clearance.

Subjects activated in a session take on the security class with which the user has connected.

Secrecy classes

- assigned to users reflect user's trustworthiness not to disclose sensitive information to individuals who do not hold appropriate clearance.
- assigned to objects reflect the sensitivity of information contained in the objects and the potential damage that could result from their improper leakage

Categories define the area of competence of users and data.

Bell La Padula

Defines mandatory policy for secrecy

Different versions of the model have been proposed (with small differences or related to specific application environments); but the basic principles remain the same.

Goal: prevent information flow to lower or uncomparable security classes

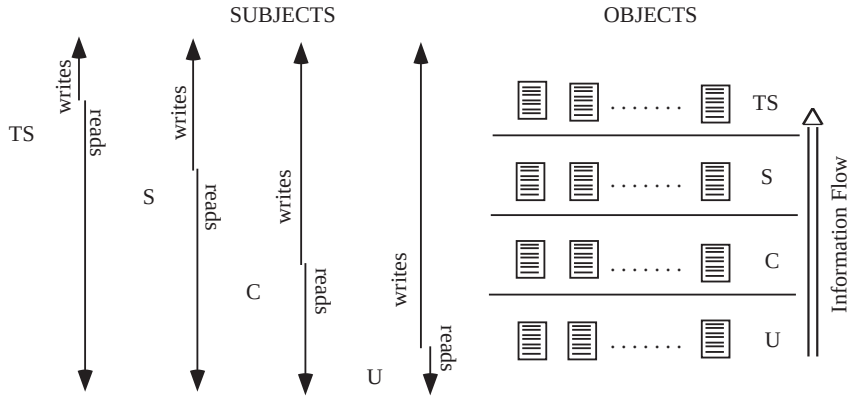
simple property A subject s can read object o only if $\lambda(s) \succeq \lambda(o)$

***-property** A subject s can write object o only if $\lambda(o) \succeq \lambda(s)$

$\Longrightarrow$ NO READ UP
NO WRITE DOWN

Easy to see that Trojan Horses leaking information through *legitimate* channels are blocked

Information flow for secrecy



Bell LaPadula security properties

System is modeled as state and transitions of states

State $v \in V$ ordered triple (b, M, λ)

- $b \in \wp(S \times O \times A)$: set of accesses (current) in state v
- M: Access matrix with S rows, O columns, A entries
- $\lambda : S \cup O \rightarrow L$: returns the classification of subjects and objects

simple security State (b, M, λ) is secure iff
$\forall (s, o, a) \in b, a = \texttt{read} \Rightarrow \lambda(s) \succeq \lambda(o)$

***-security** State (b, M, λ) is secure iff
$\forall (s, o, a) \in b, a = \texttt{write} \Rightarrow \lambda(o) \succeq \lambda(s)$

A state is secure iff it satisfies the simple security property and *-property.

State transition function $T : V \times R \rightarrow V$ transforms the state into another that satisfies the two properties.

BLP — Secure system

A system (v_0, R, T) is *secure* iff v_0 is secure and every state reachable from v_0 by executing a finite sequence of requests from R is secure.

Theo A system (v_o, R, T) is secure iff

- v_0 is a secure state
- T is such that $\forall v$ reachable from v_0 by executing one or more requests from R, if $T(v, c) = v'$, where $v = (b, M, \lambda)$, and $v' = (b', M', \lambda')$, then $\forall s \in S, o \in O$:
  - $(s, o, \texttt{read}) \in b' \wedge (s, o, \texttt{read}) \notin b \Rightarrow \lambda'(s) \succeq \lambda'(o)$
  - $(s, o, \texttt{read}) \in b \wedge \lambda'(s) \not\succeq \lambda'(o) \Rightarrow (s, o, \texttt{read}) \notin b'$
  - $(s, o, \texttt{write}) \in b' \wedge (s, o, \texttt{write}) \notin b \Rightarrow \lambda'(o) \succeq \lambda'(s)$
  - $(s, o, \texttt{write}) \in b \wedge \lambda'(o) \not\succeq \lambda'(s) \Rightarrow (s, o, \texttt{write}) \notin b'$

Problem: no restriction is placed on T, which can be exploited to leak information (Mc Lean's System Z)

BLP +tranquility

Security restrictions not enough. Need to control T.

Assume T as follows:

- when a subject requests any access on an object, the security level of all subjects and all objects is downgraded to the lowest level and the access is granted

Secure by BLP but .... not secure in a meaningful sense .....

BLP is made secure by addition of

Tranquility property The security level of subjects and objects cannot change

Loosening up tranquility restriction:

- Not all changes of levels leak information (e.g., upgrading can be ok)
- Trusted subjects can be allowed for downgrading (e.g., sanitization)

Exceptions to axioms

Real-word requirements may need mandatory restrictions to be bypassed

Data association: A set of values seen together is to be classified higher than the value singularly taken (e.g., *name* and *salary*)

Aggregation: An aggregate may have higher classification than its individual items. (e.g., the location of a *single* military ship is unclassified but the location of *all* the ships of a fleet is secret)

Sanitization and Downgrading: Data may need to be downgraded after some time (embargo). A process may produce data less sensitive than those it has read

$\Longrightarrow$ Trusted process

A trusted subject is allowed to bypass (in a controlled way) some restrictions imposed by the mandatory policy.

Coexistence of DAC and MAC

DAC and MAC not mutually exclusive

- E.g., BLP enforces DAC as well
  DAC property $b \subseteq \{(s, o, a)$ s.t. $a \in M[s, o]\}$

If both DAC and MAC are applied only accesses which satisfy both are permitted

DAC provides discretionality within the boundaries of MAC

Limitation of mandatory policies

Secrecy mandatory policy (even with tranquility) controls only overt channels of information (flow through legitimate channels).

Remain vulnerable to covert channels.

Covert channels are channels not intended for communicating information but that can, however, be exploited to leak information.

Every resource or observable of the system shared by processes of different levels can be exploited to create a covert channel.

Covert and timing channels – examples

- Low level subject asks to write a high level file. The system returns that the file does not exist (if the system creates the file the user may not be aware when necessary).
- Low level subject requires a resource (e.g., CPU or lock) that is busy by a high level subject. Can be exploited by high level subjects to leak information to subjects at lower levels.
- A high level process can lock shared resources and modify the response times of process at lower levels (timing channel). With timing channel the response returned to a low level process is the same, its the time to return it that changes.

Locking and concurrency mechanisms must be redefined for multilevel systems.
(Careful to not introduce denial-of-service.)

Covert channel analysis

Covert channel analysis usually done in the implementation phase (to assure that a system's implementation of the model primitive is not too weak).

Interface models attempt to rule out such channels in the modeling phase.

- Non interference: the activity of high level processes must not have any effect on processes at lower or incomparable levels.

Multilevel databases

The Bell-La Padula model was proposed for the protection at the OS level.

Subsequent approaches have investigated the application of multilevel policies to data models (DBMS, object-oriented systems, ...)

While in the OS context the security level is assigned to a file, DBMSs can afford a finer grained classification:

- relation
- attribute
- tuple
- element

Relational data model

Each relation is characterized by

- Scheme of the relation $R(A_1, \ldots, A_n)$. Indipendent from the state.
- Istance of the relation, dipendent on the state, composed of tuples $(a_1, \ldots, a_n)$

Name	Dept	Salary
Bob	Dept1	100K
Ann	Dept2	200K
Sam	Dept1	150K

  Key attributes uniquely identify tuples
  - No two tuples can have a same key
  - Key attributes cannot have null values

Multilevel DBMSs

In DBMSs that support element level classification, each relation is characterized by

- Scheme of the relation $R(A_1, C_1, \ldots, A_n, C_n)$, indipendent from the state
  - $C_i, i = 1, \ldots, n$ range of security classifications
- Set of istances of the relation R_c dependent on the state; one istance for each security class c. Each instance is composed of tuples $(a_1, c_1 \ldots, a_n, c_n)$.

  Instance at level c contains only elements whose classification is dominated by c.

Multilevel relational data model

Access control obeys the BLP principles

- no read up (the view of a subject at a given access class c contains only the elements whose classification is dominated by c)
- no write down further restricted

  $\Longrightarrow$ Every subject writes at only its level

  With classification at fine granularity write up is not needed

Multilevel relation – example

Name	$\lambda_{\mathbf{N}}$	**Dept**	$\lambda_{\mathbf{D}}$	**Salary**	$\lambda_{\mathbf{S}}$
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S

Instance U

Name	$\lambda_{\mathbf{N}}$	**Dept**	$\lambda_{\mathbf{D}}$	**Salary**	$\lambda_{\mathbf{S}}$
Bob	U	Dept1	U	100K	U
Sam	U	Dept1	U	-	U

S-instance is the whole relation

Multilevel relational model

For each tuple in a multilevel relation

- key attributes uniformly classified
  $\forall t \in T, \forall A_i, A_j \in AK : \lambda(t[A_i]) = \lambda(t[A_j])$
- the classifications of nonkey attributes must dominate that of key attributes
  $\forall t \in T, \forall A_i \in AK, A_j \notin AK : \lambda(t[A_j]) \geq \lambda(t[A_i])$

Polyinstantiation

Fine grained classification must take into consideration data semantics and possible information leakage

$\Longrightarrow$ Complications:

- Polyinstantiation: presence of multiple objects with the same name but different classifications
  $\Longrightarrow$ different tuples with same key but
  - different classification for the key (polyinstantiated tuples)
  - different values and classifications for one or more attributes (polyinstantiated elements)

Polyinstantiation

Polyinstantiated tuples

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S
Ann	U	Dept1	U	100K	U

Polyinstantiated elements

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S
Sam	U	Dept1	U	100K	U

Polyinstantiation

- Invisible A low level subject inserts data in a field that already contains data at higher or incomparable level
- Visible A high level subject inserts data in a field that contains data at a lower level

Invisible polyinstantiation

A low level subject requests insertion of a tuple

The relation already contains a tuple with the same primary key but with higher classification

- Tell the subject $\Longrightarrow$ information leakage
- Replace the old tuple with the new one $\Longrightarrow$ loss of integrity
- Insert a new tuple $\Longrightarrow$ polyinstantiated tuple

Polyinstantiated tuples – example

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S

Request by U-subject

INSERT INTO Employee VALUES Ann,Dept1,100K

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S
Ann	U	Dept1	U	100K	U

Polyinstantiated elements – example

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S

Request by U-subject

UPDATE Employee SET Salary="100K" WHERE Name="Sam"

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	150K	S
Sam	U	Dept1	U	100K	U

Visible polyinstantiation

A high level subject requests insertion of a new tuple.

The relation already contains a tuple with the same primary key but with a lower classification

- Tell the subject $\Longrightarrow$ denial of service
- Replace the old tuple with the new one $\Longrightarrow$ information leakage
- Insert a new tuple $\Longrightarrow$ polyinstantiated tuple

Polyinstantiated tuples – example

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	U	Dept1	U	100K	U
Sam	U	Dept1	U	150K	S

Request by S-subject

INSERT INTO Employee VALUES Ann,Dept2,200K

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	U	Dept1	U	100K	U
Sam	U	Dept1	U	150K	S
Ann	S	Dept2	S	200K	S

Polyinstantiated elements – example

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	100K	U

Request by S-subject

UPDATE Employee SET Salary="150K" WHERE Name="Sam"

Name	λ_N	Dept	λ_D	Salary	λ_S
Bob	U	Dept1	U	100K	U
Ann	S	Dept2	S	200K	S
Sam	U	Dept1	U	100K	U
Sam	U	Dept1	U	150K	S

Polyinstantiation

Possible semantics for polyinstantiated tuples and elements

- polyinstantiated tuples $\Longrightarrow$ different entities of the real world
- polyinstantiated elements $\Longrightarrow$ same entity of the real world

Polyinstantiation must be controlled (not all instances of the database may make sense)

- At most one tuple per entity should exist at each level

.... polyinstantiation quickly goes out of hand.....

Alternative: use of "restricted" values (instead of "null")

The community has longly debated wrt

- what is the correct classification granule
- polyinstantiation vs use of restricted values

Cover story

Commercial M-DBMSs (e.g., Oracle) support classification at the level of tuples.

Polyinstantiation is blaimed to be one of the main reasons why multilevel DBMSs have not succeeded.

Polyinstantiation is not always bad. It can be useful to support cover stories.

- cover story: incorrect values returned to low level subject to protect the real value (useful when returning restricted/null would leak information)

Support of fine-grained classification has also other problems

- support of integrity constraints becomes complex
- need to control inference channels

MDBMS – Architectures

- Trusted subject: data at different levels are stored in a single database.

  The DBMS must be trusted to ensure obedience of the mandatory policy.

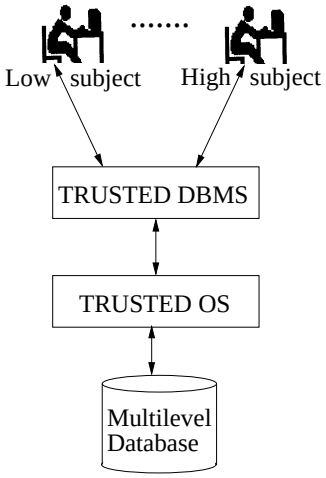
- Trusted computing base: data are partitioned in different databases, one for each level.

  Only the operating system needs to be trusted.

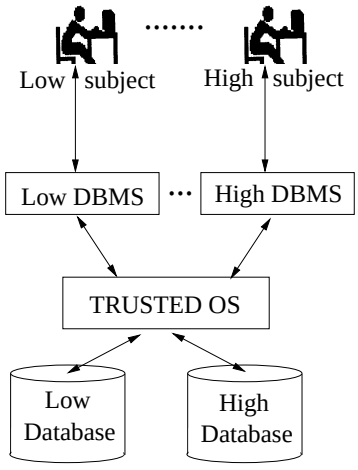
  Each DBMS is confined to access data that can be read at its level (no-read-up).

  (Decomposition and recovery algorithms must be carefully constructed to be correct and efficient.)

MDBMS – Architectures



(a) Trusted subject



(b) Trusted computing base

Integrity mandatory policy

Secrecy mandatory policies control only improper leakage of information.

Do not safeguard integrity $\Rightarrow$ information can be tampered

Dual policy can be applied for integrity, based on assignment of (integrity) classifications.

Integrity classes

- assigned to users reflect users' trustworthiness not to improperly modify information.
- assigned to objects reflect the degree of trust in information contained in the objects and the potential damage that could result from its improper modification/deletion

Categories define the area of competence of users and data.

Biba model for integrity

Defines mandatory policy for integrity

Goal: prevent information to flow to higher or uncomparable security classes

Strict integrity policy Based on principles dual to those of BLP

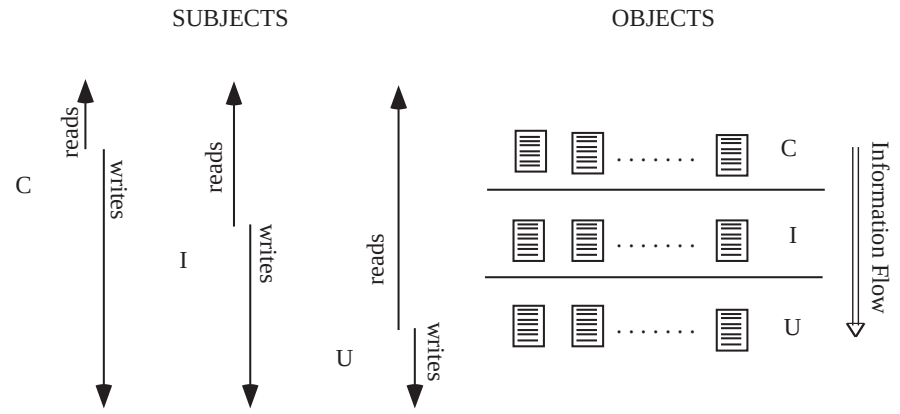
simple property A subject s can read object o only if $\lambda(o) \succeq \lambda(s)$

***-property** A subject s can write object o only if $\lambda(s) \succeq \lambda(o)$

$\Longrightarrow$ NO READ DOWN
NO WRITE UP

Secrecy and integrity policies can coexist ..... **but**
..... need "independent" labels

Information flow for integrity



Biba model for integrity – Alternative policies

Low-water mark for subjects (no write-up)

- A subject s can write object o only if $\lambda(s) \succeq \lambda(o)$
- A subject s can read any object o.
  After the access $\lambda(s) := \mathsf{glb}(\lambda(\mathsf{s}), \lambda(\mathsf{o}))$.

Drawback: order of operations affects subject's privileges

Low-water mark for objects (no read-down)

- A subject s can read object o only if $\lambda(o) \succeq \lambda(s)$
- A subject s can write any object o.
  After the access $\lambda(o) := \mathsf{glb}(\lambda(\mathsf{s}), \lambda(\mathsf{o}))$.

Drawback: it does not safeguard integrity but simply signals its compromise

Limitations of Biba policies

Biba's model for the protection of integrity has shortcomings

- flow restrictions may result too restrictive
- it enforces integrity only by preventing information flows from lower to higher access classifications $\Longrightarrow$ it captures only a very small part of the integrity problem

Role-Based (RBAC) policies

Role-based access control model – 1

Role named set of privileges related to execution of a particular activity

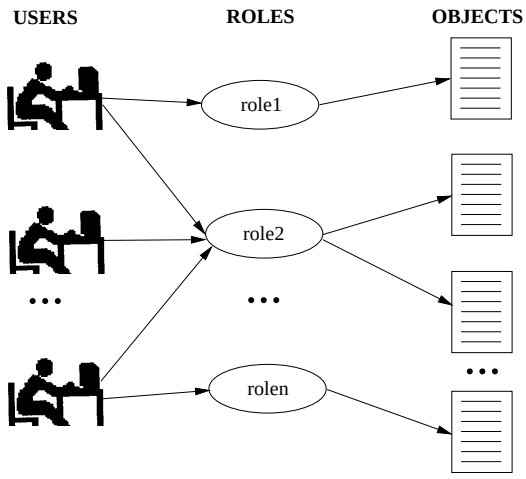
Access of users to objects mediated by roles

- Roles are granted authorizations to access objects
- Users granted authorizations to activate roles
- By activating a role r a user can executed all access granted to r
- The privileges associated with a role are not valid when the role is not active

Note difference between

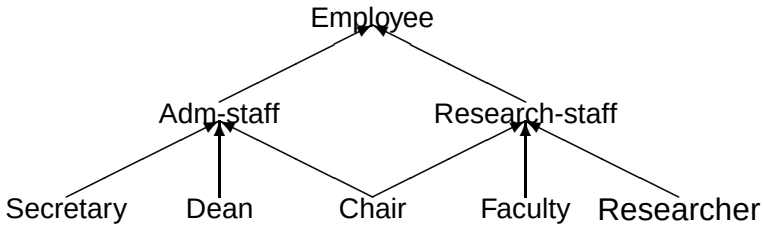
- *group*: set of users
- *role*: set of privileges

Role-based access control model – 2



Role-based access control model – 3

Role hierarchy defines specialization relationships



Hierarchical relationship $\Longrightarrow$ authorization propagation

- If a role r is granted authorization to execute (action, object) $\Longrightarrow$ all roles generalization of r can execute (action, object)
- If u is granted authorization to activate role $r \Longrightarrow u$ can activate all generalizations of r

RBAC – Advantages

Easy management easy to specify authorizations (e.g., it is sufficient to assign or remove a role for a user to enable the user to execute a whole set of tasks)

Role hierarchy can be exploited to support implication. Makes authorization management easier.

Restrictions Further restrictions can be associated with roles, such as cardinality or mutual exclusions.

Least privilege It allows associating with each subject the least set of privileges the subject needs to execute its work $\Longrightarrow$ Limits abuses and damages due to violations and errors.

Separation of duty Roles allow the enforcement of separation of duty (split privileges among different subjects).

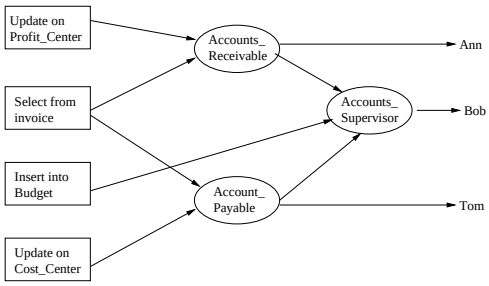
Role-based models

Work on role-based models has been addressing also:

- relationships beyond hierarchical (e.g., secretary can operate on behalf of his manager)
- hierarchy-based propagation not always wanted (some privileges may not propagate to subroles)
- enriched administrative policies (authority confinement)
- relationships with user identifiers (needed for individual relationships – e.g., “my secretary”)
- additional constraints E.g., dynamic separation of duty (e.g., completion of an activity requires participation of at least n individuals)

Roles in SQL

In SQL privileges can be grouped in roles that can be assigned to users or to other roles (nested)



By activating a role a user is enabled for all the privileges in a subset rooted at that role

- At all time, at most one role active per user
- roles can be granted to users with grant option $\Longrightarrow$ the user can grant it to others.

Administrative policies

Administrative policies

Define who can grant and revoke access authorizations.

- Centralized: a privileges authority (system security officer) is in charge of authorization specification.
- Ownership The creator of an object is its owner and as such can administer access authorization on the object.

  Ownership not always clear in:
  - hierarchical data models (e.g., object-oriented)
  - RBAC framework

Authority to specify authorizations can be delegated.

Delegation often associated with ownership: the owner of an object delegates administrative privileges to others.

Decentralized administration introduces flexibility, but complicates the scenario.

Decentralized administration

Different administrative policies can differ for how they respond to the following questions.

- At which granularity should administrative authorizations be supported?
- can further delegation be restricted?
- who can revoke authorizations?
- what happens to the authorizations granted by somebody whose administrative privileges are being revoked?

Decentralized administration

Different administrative policies can differ for how they respond to the following questions.

- At which granularity should administrative authorizations be supported?

  can go at the fine grain of each single access (action, object)
- can further delegation be restricted?

  usually not; but may be useful
- who can revoke authorizations?

  who granted them; the owner; every administrator
- what happens to the authorizations granted by somebody whose administrative privileges are being revoked?

  should we delete them? should we keep them?

Authorization administration in SQL

The user who creates a table is its owner and can grant authorizations on the table to others.

- authorizations can be granted with *grant-option*
- grant option on an access allows a user to further grant that access (and grant option) to others $\Longrightarrow$ chain of authorizations.
- users can revoke only authorizations they have granted

Examples of Grant

Ann is the owner of table DEPARTMENT

Ann

```
GRANT SELECT
ON TABLE Department
WITH GRANT OPTION
TO Bob
```

Ann

```
GRANT SELECT
ON TABLE Department
TO Chris
```

Bob

```
GRANT SELECT
ON TABLE Department
TO David
WITH GRANT OPTION
```

David

```
GRANT Select
ON TABLE Department
TO Frank
```

Revocation in administration in SQL

When a user is revoked the grant option for a privilege what should happen to the authorizations for the privilege she granted?

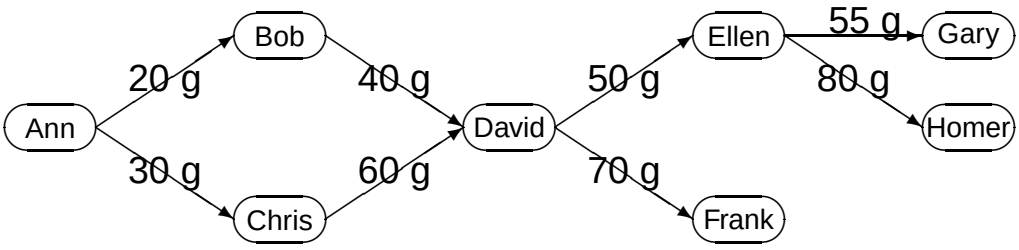
Revocation can be requested:

with cascade (recursive) If the revoker would not hold anymore the privilege with the grant option, the authorizations she granted are recursively deleted. Need to pay attention to cycles.

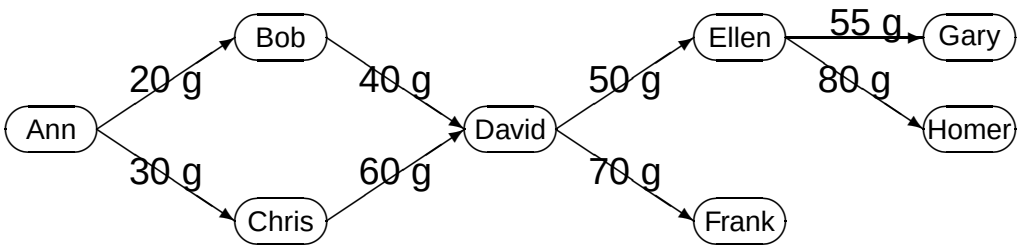
without cascade If the revocation of an authorization would imply recursive deletion, the revoke operation is not executed.

Note: The original (cascade) revocation policy was based on time: all authorizations granted in virtue of an authorization that was being revoked were deleted, regardless of other later authorizations that the user had received
Current SQL does not consider time.

Revocation in SQL – example

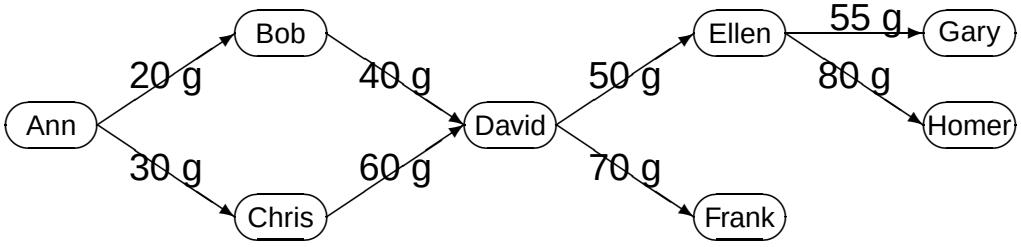


Revocation in SQL – example

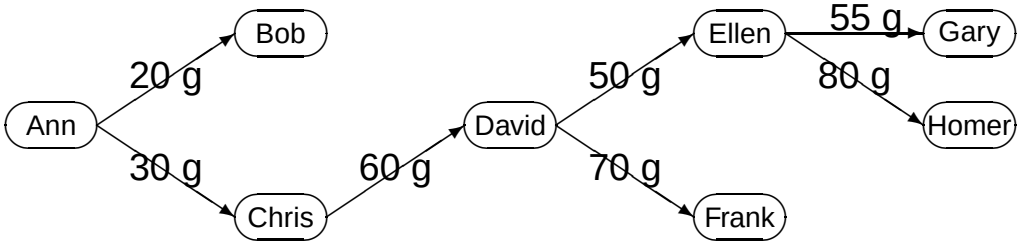


Bob revokes the authorization from David with cascade

Revocation in SQL – example



Bob revokes the authorization from David with cascade



Clark and Wilson model for Integrity

Limitations of Biba's integrity policies

Biba's model for the protection of integrity has shortcomings

- flow restrictions may result too restrictive
- it enforces integrity only by preventing information flows from lower to higher access classifications $\Longrightarrow$ it captures only a very small part of the integrity problem

Military vs commercial mechanisms

Military	Commercial
Data item associated with a particular level	Data item associated with a set of programs permitted to manipulate it
Users constrained by what they can read and write	Users constrained by which programs they are allowed to execute

Integrity (1)

Integrity is a more complex concept: ensuring that no resource has been modified in an unauthorized or improper way and that data stored in the system correctly reflect the real word they are intended to represent

$\Longrightarrow$ need to prevent flaws and errors

Any data management system has functionalities for ensuring integrity

- concurrency control and recovery techniques: to ensure that no concurrent access can lead to data loss or inconsistencies
- recovery techniques: to recover the state of the system in case of errors or violations
- integrity constraints: that enforce limitation on the values that can be given to data

Integrity (2)

- Atomicity: either all of the actions of a transaction are performed or none of them are.
- Consistency: a transaction must preserve the satisfaction of the integrity constraints on the data
- Isolation: the concurrent execution of a set of transactions must have the same effect as that of some serial execution of that set
- Durability: the results of committed transactions are permanent

Not enough! they do not take into consideration the subject that perform the action. They do not protect against abuses by legitimate users.

Clark and Wilson – 1

Clark and Wilson's model defines four basic criteria to safeguard integrity:

1. Autentication: the system must separately authenticate and identify every user, so that his actions can be controlled and audited.
2. Audit: the system must maintain an audit log that records every program executed and the name of the authorizing user.
3. Well-formed transactions: the system must ensure that specified data items can be manipulated only by a restricted set of programs, and the data center controls must ensure that these programs meet the well-formed transaction rule.
4. Separation of duty: the system must associate with each user a valid set of programs to be run, and the data center controls must ensure that these sets meet the separation of duty rule.

Clark and Wilson – 2

Based on the following concepts

- Constrained Data Items CDI: all those data items within the system to which the integrity model must be applied.
- Unconstrained Data Items UDI: items not subject to integrity constraints
- Integrity Verification Procedures. IVP procedures that allow to verify integrity, i.e., their goal is to confirm that all the CDIs conform to the integrity specification at the time IVP is executed.
- Transformation Procedures. TPs are all the procedures that can modify CDIs or take arbitrary input from users and creates CDIs. They corresponds to well-formed transactions: they change the set of CDIs from one valid state to another.

Clark and Wilson – 3

Integrity verification is a two-part process:

- Certification: done by the system security officer, system owner, and system custodian with respect to an integrity policy $\Longrightarrow$ Certification rules
- Enforcement: done by the system $\Longrightarrow$ Enforcement rules

CW rules – 1

Basic framework (internal consistency)

C1 All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run

C2 All TPs must be certified to be valid (preserve the validity of the state). For each TP and each set of CDIs, the security officer must specify a "relation", which defines that execution; relations are of the form (TPi, (CDIa, CDIb, ...))

E1 The system must maintain the list of relations specified in rule C2, and must ensure that the only manipulation of any CDI is by a TP, where TP is operating on the CDI as specified in some relation.

CW rule – 2

External consistency (separation of duty)

E2 The system must maintain a list of relations of the form (UserID, TPi, (CDIa,CDIb, ...)), which relates to a user, a TP, and the data objects that TP may reference on behalf of the user. It must ensure that only executions described in one of the relations are performed.

C3 The list of relations in E2 must be certified to meet the separation of duty requirement.

E3 The system must authenticate the identity of each user attempting to execute a TP.

CW rules – 3

Audit

C4 All TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstructed.

No enforcement rule is necessary

CW rules – 4

Uncostrained Data Items (EDIs), e.g., information typed by the user at the keyboard

C5 Any TP that takes a UDI as an input value must be certified to perform only valid transformation, or else no transformation, for any possible value of the UDI. The transformation should take the input from a UDI to a CDI, or the UDI is rejected. Typically this is an edit program.

E4 Only the agent permitted to certify entities may change the list of entities associated with other entities: specifically associated with a TP. An agent that can certify an entity may not have any execute rights with respect to that entity.

Clark and Wilson

Advantages:

- it addresses in a more complete way the integrity problem
- it models commercial environments

Shortcomings:

- not well formalized $\Longrightarrow$ it is difficult to reason about security properties

Chinese Wall

Chinese wall

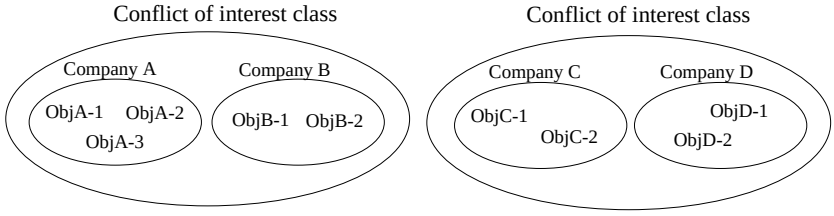
Special type of mandatory-style dynamic separation of duty for protecting secrecy

Goal prevent information flows which cause conflict of interest for individual consultants (e.g., an individual consultant should not have information about two banks or two oil companies)

Chinese wall

Objects organized hierarchically. Three levels

- basic objects (e.g., files)
- company datasets group objects referring to a same corporation
- conflict of interest classes groups all company datasets whose corporation are in competition



Chinese wall – Simple security rule

A subject s can access an object o only if:

- o is in the same company dataset as all the objects that s has already accessed, i.e., within the wall
- o belongs to an entirely different conflict of interest class

Chinese wall – sanitization

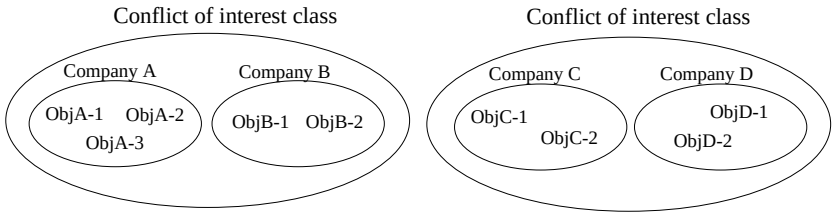
- Users may need to compare information of different corporations $\Longrightarrow$ Simple security rule might be too restrictive
- Allow information to be "sanitized". Sanitization takes the form of disguising a corporate information, in particular to prevent the discovery of a corporation's identity.

Chinese wall – *-property

A subject s can write an object o only if

- access is permitted by the simple security rule, and
- no object can be read by s (according to authorizations) which *i)* is in a different company dataset than o's, and *ii)* contains unsanitized information

Example:
Alice reads ObjA-1 e writes ObjC-1. Bob reads ObjC-1 e writes ObjB-1.



Chinese Wall – axioms

- The simple security rule prevents flow by a single user
- The *-property rule prevents indirect flows that can be enacted by collusions between users
- Sanitization provides more flexibility w.r.t. the application of the policy
- Discretionary access is assumed to be enforced

Chinese Wall

Chinese Wall not completely formalized and leaves open problems, such as:

- keep and manage history of access
- ensure accessibility (e.g., if all the users read the same dataset, the system becomes not usable)
- data sanitization is not addressed (and it is a complex problem)

But it remains an important contribution wrt introducing the concept of dynamic separation of duty and defining conflicts of interests.

Separation of duty

Separation of duty principle: no user (or restricted set of users) should have enough privileges to be able to abuse the system.

static who specifies the authorizations must make sure not to give "too much privileges" to a single user

dinamic the control on limiting privileges is enforced at runtime: a user cannot use "too many" privileges but he can choose which one to use. The system will consequently deny other accesses $\Longrightarrow$ more flexible

Example: order-goods, send-order, record-invoice, pay. Four employees. Protection requirements:
at least two people must be involved in the process

static the administrator assigns tasks to users so that none can execute all the four operations

dinamic each user can execute any operation, but cannot complete the process and execute all four.

Expanding authorizations

Access control in object-oriented systems – 1

Access control in object-oriented systems can exploit

- encapsulation: access to objects is always carried out via methods (procedures)

We can give users permission to execute procedures without necessarily requiring that they hold the authorizations for all the actions that the procedure will have to perform; requests for such actions can be controlled w.r.t.

- the owner of the method (as in CACL)
- the user who has granted the authorizations (as in OSQL)
- the code that requests the action (as in Java)

Access control in object-oriented systems – 2

The encapsulation property of OO systems has also been exploited to enforce flow controls and restrictions.

Starting point: adaptation of the no-write-down principle for discretionary policies: information can flow from an object o to an object o' only if the users authorized to read o' are a subset of the users authorized to read o

- too restrictive

In OO systems:

- we can control the flow of information mediating all message exchanges (method calls)
- we can specify exceptions to restrictions
  - information returned by a (trusted) method can be released to users not authorized to directly read the information the method accesses to compute its results

DAC – Expanding authorizations

Traditionally supported:

user groups Users collected in groups and authorizations specified for groups

conditional Validity of authorizations dependent on satisfaction of some conditions

- *system-dependent* evaluate satisfaction of system predicates
  - location
  - time
- *content-dependent* dependent on value of data (DBMS)
- *history dependent* dependent on history of requests

Relatively easy to implement in simple systems

Introduce complications in richer models

Expanding authorizations – 1

Specifications for single entities (users, files, ...) too heavy

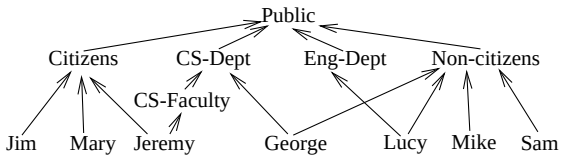
- support abstractions (grouping of them). Usually hierarchical relationships: users/groups; objects/classes; files/directories; .....

  Authorizations may propagate along the hierarchies

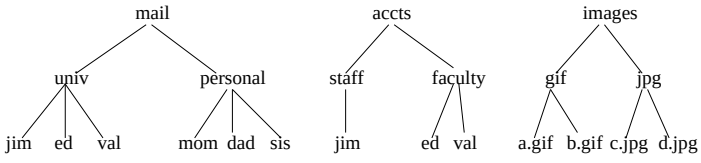
Hierarchical data systems

Support of hierarchies can be applied to all dimensions of authorizations.

Subjects (e.g., users vs groups)



Objects (e.g., files vs directories, objects vs classes)



Actions action grouping (e.g., write modes)

subsumption (e.g., write $\succeq$ read)

Expanding authorizations – 2

Usefulness of abstractions limited if exceptions are not possible. E.g., all Employees but Sam can read a file

- support negative authorizations

  (Employees, read, file, +) (Sam, read, file, -)

Presence of permissions and denials can bring inconsistencies

- how should the system deal with them?

Permissions and denials

Easy way to support exceptions via negative authorizations.

Negative authorizations first introduced by themselves as:

open policy: whatever is not explicitely denied can be executed; as opposed to

closed policy: only accesses explicitely authorized can be executed

Recent hybrid policies support both, but

- what if for an access we have both + and -? (inconsistency)
- what if for an access we have neither + nor -? (incompleteness)

Incompleteness may be solved by either

- assuming completeness: for every access either a negation or a permission must exist $\Longrightarrow$ too heavy
- assuming either closed or open as a basis default decision

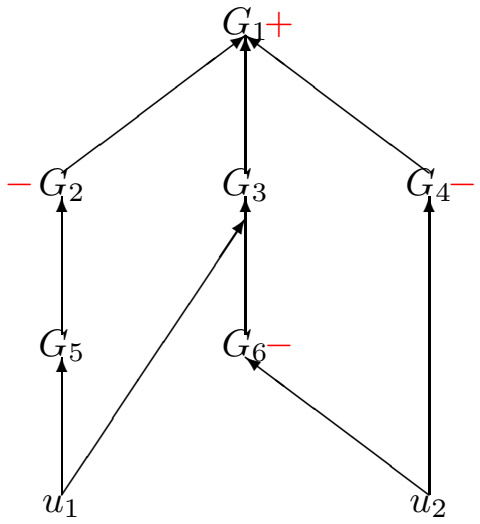
Permissions and denials – 2

Possible conflict resolution policies

- denials-take-precedence negative authorization wins (fail safe principle)
- most-specific-takes-precedence the authorization that is "more specific" wins
- most-specific-along-a-path-takes-precedence the authorization that is "more specific" wins only on the paths passing through it

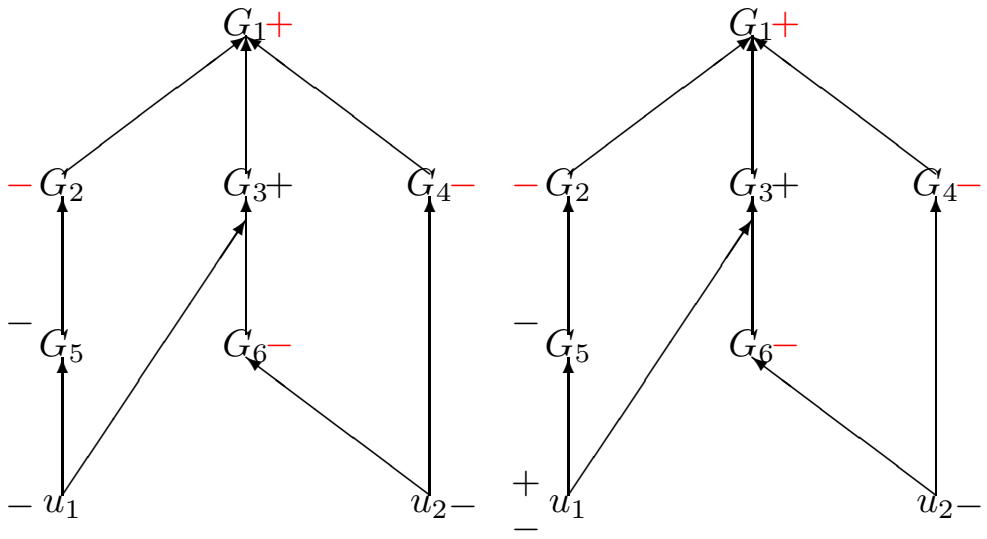
  $\Rightarrow$ authorizations propagate until overridden by more specific authorizations
- Other.....

Example of conflict resolution



explicit authorizations

Examples of conflict resolution



most specific

most specific along a path

Most specific takes precedence

Most specific intuitive and natural ..... but

- what is more specific if multiple hierarchies?

  `(Employees, read, file1, +)`
  `(Sam, read, directory1, -)`

- in some cases not wanted.

  E.g., authorizations that do not allow exceptions

  - `(Employees, read, bulletin-board, +)`
    I do not want anybody to be able to forbid
  - `(Employees, read, budget, +)`
    `(Temporary_employees, read, budget, -)`
    I do not want my restriction on temporary employees to be bypassed

Other conflict resolution policies

Strong vs weak (e.g., Orion) authorizations

Strong authorizations cannot be overridden

Weak authorizations can be overridden. Usually,

- Strong authorizations always override weak authorizations
- Weak authorizations override each other according to overriding (most specific) policy

Some limitations/complications:

- Supports only two levels. May not be enough
- Strong authorizations must be consistent

  Not easy when groupings are dynamic

  (e.g., content-based conditions)

Other conflict resolution policies – 3

Explicit priority authorizations have associated explicit priorities

- difficult to manage

Positional strength of authorizations depend on order in authorization list

- gives responsibility of explicitely resolving conflicts to security administrator
- controlled administration difficult to enforce

Grantor-dependent strength of authorizations depend on who granted them

- need to be coupled with others to support exceptions among authorizations stated by a single administrator

Time-dependent strength of authorizations depend on time they have been granted (e.g., more recent wins)

- limited applicability

Conflict resolution policies

Different conflict resolution policies are not in mutual exclusion. E.g., I can first apply “most specific” and then “denials-take-precedence” on the remaining conflicts

There is no policy better than the others:

- Different policies correspond to different choices that we can apply for solving conflicts.

Trying to support all the different semantics that negation can have (strong negation, exception,....) can lead to models not manageable.

$\Longrightarrow$ Often negative authorizations are not used.

However, they can be useful.

$\Longrightarrow$ Systems that support negative authorizations usually adopt one specific conflict resolution policy.

Positive and negative autorizations in Apache

Authorizations can be positive or negative.

Users can specify an order that defines how to interpret positive/negative authorizations. Two choices:

deny,allow negative authorizations are evaluated first and access is allowed by default. A requestor is granted access if it does not have any negative authorizations *or* it has a positive autorization.

allow,deny positive authorizations are evaluated first and access is denied by default. A requestor is denied access if it does not have any positive authorization *or* it has a negative authorization.

Example

Order Deny,Allow
Deny from all
Allow from .crema.unimi.it

Expanding DAC authorizations

Recent DAC models try to include at least

- positive as well as negative authorizations
- authorization propagation based on hierarchies
- conflict resolution and decision strategies
- additional implication relationships

Goal Be flexible and go towards support of multiple policies

- Different administrators may have different protection requirements
- Same administrator but objects to be protected differently
- Protection requirements may change over time

Logic-based authorization languages

Recent approaches based on use of some logic.

Good: increased expressiveness and flexibility

.... but we must be careful to

- balance flexibility/expressiveness vs performance
- not loose control over specifications
- guarantee behavior of the specifications (do not forget we are talking of security)

Some proposals allow multiple interpretations

$\Longrightarrow$ ambiguous semantics of security specifications

.... we will return on this when illustrating access controls in open systems

Example of a logic based authorization specification language

cando(o,s,⟨*sign*⟩a) : explicit authorizations.

dercando(o,s,⟨*sign*⟩a) : defines implied authorizations

do(o,s,⟨*sign*⟩a) states the accesses that must be allowed or denied.

done(o,s,r,a,t) access history.

error integrity constraints.

hie-predicates: hierarchical predicate

rel-predicates: application specific predicates (es., $\texttt{owner}(\texttt{user}, \texttt{object})$, $\texttt{supervisor}(\texttt{user1}, \texttt{user2})$).

FAF rule stratification

Format of rule is restricted to ensure stratification of rules

Level	Predicate	Rules defining predicate
0	`hie-`predicates `rel-`predicates `done`	base relations. base relations. base relation.
1	`cando`	body may contain `done`, `hie-` and `rel-`literals.
2	`dercando`	body may contain `cando`, `dercando`, `done`, `hie-`, and `rel-` literals. Occurrences of `dercando` literals must be positive.
3	`do`	in the case when head is of the form $\texttt{do}(\_, \_, +a)$ body may contain `cando`, `dercando`, `done`, `hie-` and `rel-` literals.
4	`do`	in the case when head is of the form $\texttt{do}(o, s, -a)$ body contains just one literal $\neg\texttt{do}(o, s, +a)$.
5	`error`	body may contain `do`, `cando`, `dercando`, `done`, `hie-`, and `rel-` literals.

Default rule: $\texttt{do}(o, s, -a) \leftarrow \neg\texttt{do}(o, s, +a)$

Examples of FAF rules

$$
\begin{aligned}
\texttt{cando}(\texttt{file1}, s, \texttt{+read}) &\leftarrow \texttt{in}(s, \texttt{Employees}) \;\&\; \\
&\quad \neg\texttt{in}(s, \texttt{Soft-Developers}). \\
\texttt{cando}(\texttt{file2}, s, \texttt{+read}) &\leftarrow \texttt{in}(s, \texttt{Employees}) \;\&\; \\
&\quad \texttt{in}(s, \texttt{Non-citizens}).
\end{aligned}
$$

$$
\begin{aligned}
\texttt{dercando}(\texttt{file1}, s, -\texttt{write}) &\leftarrow \texttt{dercando}(\texttt{file2}, s, +\texttt{read}). \\
\texttt{dercando}(o, s, -\texttt{grade}) &\leftarrow \texttt{done}(o, s, r, \texttt{write}, t)\& \\
&\quad \texttt{in}(o, \texttt{Exams}). \\
\texttt{dercando}(\texttt{file1}, s, -\texttt{read}) &\leftarrow \texttt{dercando}(\texttt{file2}, s', +\texttt{read})\& \\
&\quad \texttt{in}(s, g)\&\; \texttt{in}(s', g).
\end{aligned}
$$

Examples of FAF rules – 2

$$
\begin{aligned}
\texttt{do}(\texttt{file1}, s, +a) &\leftarrow \texttt{dercando}(\texttt{file1}, s, +a). \\
\texttt{do}(\texttt{file2}, s, +a) &\leftarrow \neg\texttt{dercando}(\texttt{file2}, s, -a). \\
\texttt{do}(o, s, +\texttt{read}) &\leftarrow \neg\texttt{dercando}(o, s, +\texttt{read}) \,\& \\
&\quad \neg\texttt{dercando}(o, s, -\texttt{read}) \,\& \\
&\quad \texttt{in}(o, \texttt{Pblc-docs}).
\end{aligned}
$$

$$
\begin{aligned}
\texttt{error} &\leftarrow \texttt{in}(s, \texttt{Citizens}) \,\&\, \texttt{in}(s, \texttt{Non} - \texttt{citizens}). \\
\texttt{error} &\leftarrow \texttt{do}(o, s, +\texttt{write}) \,\&\, \texttt{do}(o, s, +\texttt{evaluate}) \,\& \\
&\quad \texttt{in}(o, \texttt{Tech} - \texttt{reports}).
\end{aligned}
$$

Expanding authorization form

Observation Ability to execute activities requires several privileges. Granting and revoking such privileges may become a hassle
$\Longrightarrow$ (user,object) authorizations too complex to maintain by hand in large databases.

Look at applications and provide support for

- Application/task concepts
- Least privilege
- Separation of duty (static and dynamic)

Recent directions in access control

Access control in the global infrastructure .....

- need to interact with remote parties and access remote resources
- accesses as (action,object) limiting. E.g., service
- relationships with authentication may change
  - in some cases authentication not even wanted (anonymous transactions)
  - in an open system like Internet new users (not known at the server) can present requests
    - group and role administration may not be centralized
    - the system protecting resources may not know its users in advance

$\Longrightarrow$ access control based on the use of digital certificates (credentials)

A more general approach supporting certificates

Allow users to present digital certificates, signed by some authority trusted for making a statement, and can

- bind a public key to an identity (identity)
- bind a public key or identity to some properties (e.g., membership in groups)
- bind a public key or identity to the ability of enjoying some privileges (authorization)

The server can use certificates to enforce access control.

Certificate management relates to the context of:

- Certification Authorities
- Public Key Infrastructure
- Trust Management

Access control in open systems

The way access control is enforced may change

- What users can do depend on assertions (attributes) they can prove presenting certificates
- Access control does not return "yes/no" anymore, but responds with requirements that the requestor must satisfy to get access.

Not only the server needs to be protected .....

- Clients want guarantees too (e.g., privacy)

Can introduce some form of negotiation.

Basic scenario

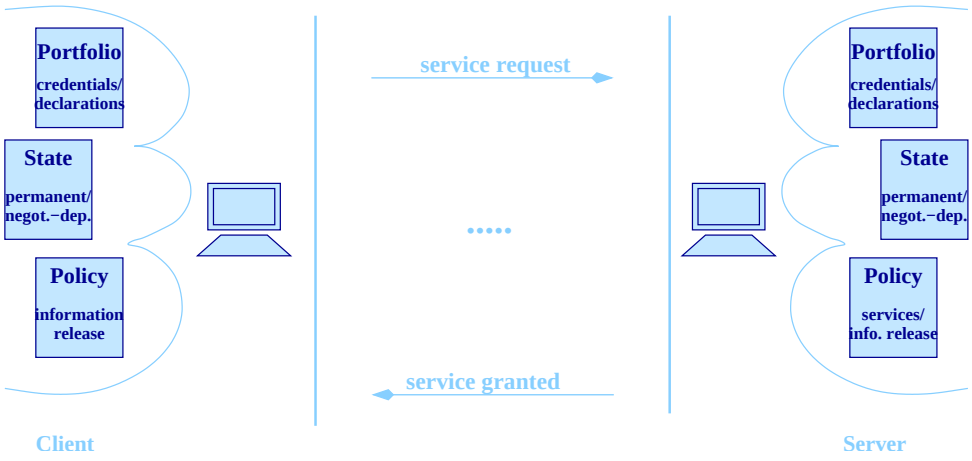
Network composed of different parties that interact with each other

- to offer services (servers)
- to require services (clients)
- a party can act both as a server and a client (servent in P2P)

Each party can have

- set of services it provides
- a portfolio of properties that the party enjoys
  - declarations: uttered by the party and not certified by any authority
  - credentials: digital certificates (c, K)
    - c: signed content
    - K: public digital signature verification key

Client-Server Interplay



Credential-based access control

Issues to be addressed:

- Expression of access control restrictions $\Longrightarrow$ language
- Communication of access control restrictions to be satisfied
  - Safeguard privacy of the involved parties
    - avoid unnecessary release of certificates and information
    - avoid leakage of access control policies and information

  $\Longrightarrow$ filtering and renaming of policies

Access control language – desiderata

- support reference to credentials and their signing authorities
- support fine-grained reference to properties/attributes within a credential
- support typical hierarchical relationships and abstractions on services and portfolio
  - refer to sets of services/properties with a single name
  - model gradual access to services
- usual expressiveness and flexibility
- provide metapolicies for protecting the policy when communication requisites
- ....

Communicating access control policies

- How should the server communicate to the client the requirements it needs to satisfy to get access?
  - Cannot send its rules
  - May need to perform a partial evaluation first (some conditions may query the server's private information)
  - May need to perform some renaming process (to not disclose the structure of its controls)
- What does the client do upon reception of the requirements?
  - different strategies with respect to which credentials/declarations submit
  - can make counter-requests
  - may want to be ensured of sufficient conditions

Need to ensure correctness of the overall process

Negotiating access

Policy and certificate interchange can be performed in

- whole policy communicated at once:
  - the server communicates the policy (possibly a disjunction of conditions)
  - the client can make a counter-request (for credentials/declarations)
  - if counter-request is satisfied and the client accepts the policy it decides credentials/declarations to submit
- policy communicated step by step
  - at each step each party can either satisfy the counter-part request or make additional requests

Whole policy communicated at once

Good:

- Allows establishment of sufficient conditions

Bad:

- Not always possible without compromising sensitive information in the server state (e.g., login/password control)
- may disclose too much of a policy (e.g., 1. registered, 2. EU citizens, 3. member of a partner association)

Possible solution a pre-requisite negotiation phase

Still lot of work to be done.....

- Strategies for selecting credentials/declarations to be released
- Ontologies for referring to attributes and certificates
- Regulating the use of certificates
- Evaluate privacy of server's policy
- Develop negotiation model and dialog
- Develop metapolicies for communicating requirements
- Determining retrieval of digital certificates not stored remotely
- Proving properties without releasing certificates.....

Augmenting expressiveness in the open world...

- Exploit semantic web to provide powerful and flexible context awareness
- Describe entities via context ontologies
- Provide access control rules with reasoning capability based on ontologies
- Protect privacy of context information which must be accessed only to authorized parties (e.g., personal identity)
- Integration of context information

Current directions in access control

- Policy composition
- Sticky policy support (and control of secondary-use, P3P)
- Certificate-based and attribute-based access control
- "Interactive" access control
- Support for dynamic conditions (purpose, payment, ...)
- Allow support for anonymous access and multiple identity management (e.g., peer-to-peer systems)
- Address, protect, and exploit semantically rich specifications

Looking for flexibility and expressiveness ... but also simplicity and manageability

eXtensible Access Control Markup Language

- XACML (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml)
  - A novel proposal by OASIS
  - Similar proposals: EPAL (IBM), WS-Policy (MS-IBM-BEA)
  - Important goals
    - Allow the central definition of access control policies, independent from the specific target and the integration between separate mechanisms
    - Offer all the features of a modern access control solution: roles, conditional authorizations, negative authorizations, flexible policy combination, compatibility with multiple user authentication mechanisms, etc.

Some recent/ongoing research projects

- PRIME (Privacy and Identity Management for Europe)
  - Integrated Project funded by the European Union under the VI Framework program
  - The main objective is the development of privacy-aware solutions for enforcing security.
- PrimeLife (Privacy and Identity Management in Europe for Life)
  - Large-scale Integrating Project funded by the European Union under the VII Framework program
  - The main objective is the development of solutions for protecting privacy in emerging Internet applications (e.g., collaborative scenarios and virtual communities) and for maintaining life-long privacy

References

- S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, and P. Samarati, "Access control," in *The Handbook of Computer Networks*, H. Bidgoli (ed.), Wiley, 2008.
- S. De Capitani di Vimercati, S. Foresti, and P. Samarati, "Recent Advances in Access Control," in *Handbook of Database Security: Applications and Trends*, M. Gertz, and S. Jajodia (eds.), Springer-Verlag, 2008.
- P. Samarati, and S. De Capitani di Vimercati, "Access control: Policies, models, and mechanisms," in *Foundations of Security Analysis and Design, R. Focardi, and R. Gorrieri (eds.)*, Springer-Verlag, 2001.