Data security

Pierangela Samarati

Dipartimento di Tecnologie dell'Informazione

Università di Milano

e_mail: `samarati@dti.unimi.it`

Security

Guaranteeing security means protecting information

- Secrecy (*Confidentiality*) Information can be released – directly or indirectly – only to users authorized to see it.

  Privacy: socially defined ability of an individual to determine whether, when, and to whom personal information is to be released

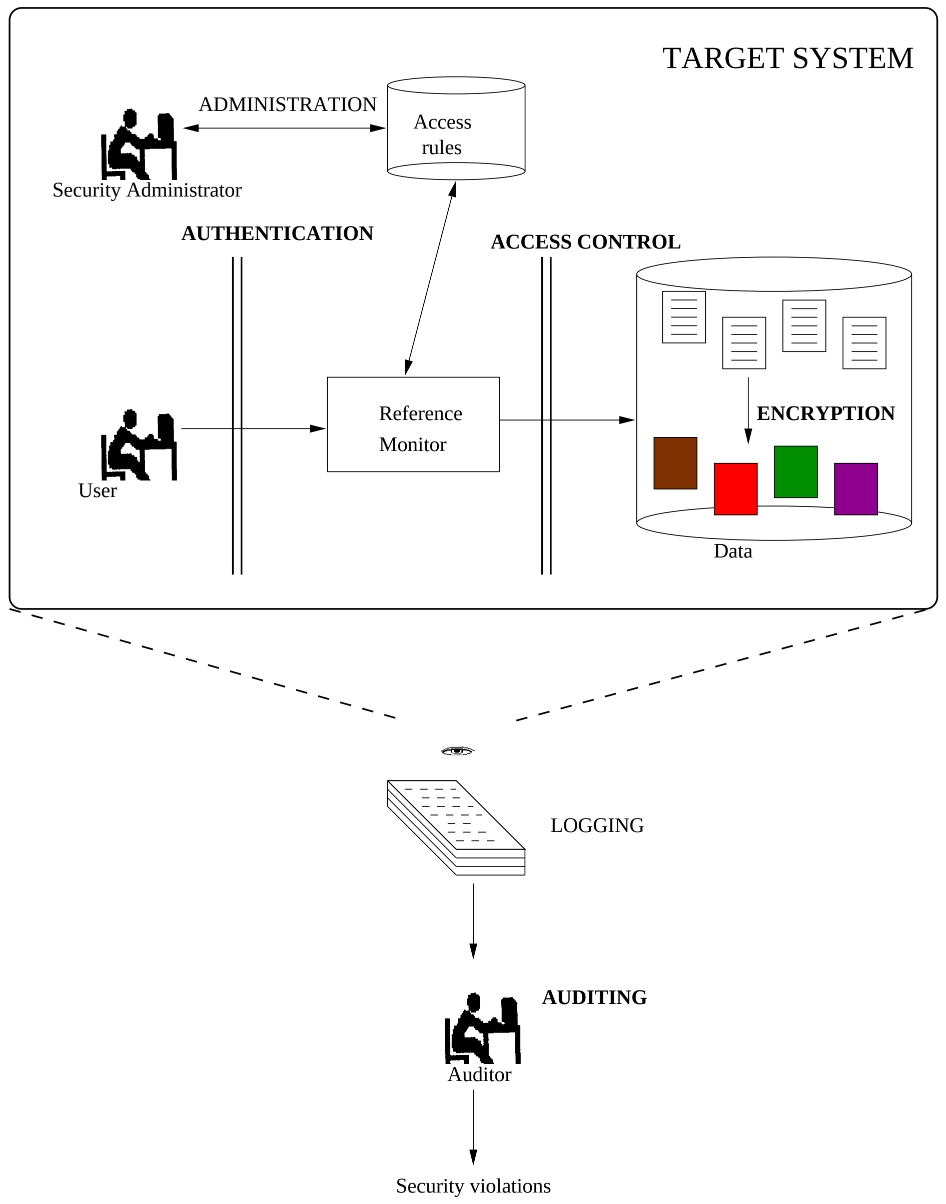
- Integrity Information must not be *improperly* modified, deleted, and tampered.

- Availability (no *denials-of-service*) Users must not be prevented from accessing data for which they have necessary permissions.

Security services

- Identification and Authentication provide the system with the ability of identifying its users and confirming their identity.
- Access Control 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.
  - Access control controls only *direct* access.
  - It may be enriched with inference, information flow, and *non-interference* controls
- Audit provides a post facto evaluation of the requests and the accesses occurred to determine whether violations have occurred or have been attempted.
- Encryption ensures that any data stored in the system or sent over the network can be deciphered only by the intended recipient.

Security services - The global picture



Authentication

- Establishes the identity of a “party” to another.
- Parties can be users or machines.
- Often bi-directional (mutual) authentication is requested. Authentication of a computer to a user can be needed to prevent spoofing attacks in which a computer masquerades as anotherone (e.g., to acquire the password of the users).
- Often combined authentication “user-to-computer” and “computer-to-computer” is needed.
- It is some sence the primary security service.
  - Correctness of the access control relies on a correct authentication.
  - Correctness of intrusion/violation control bases on correct authentication.

User to computer authentication

Can be based on:

- something the user knows (e.g., password)
- something the user has (e.g., magnetic card)
- something the user is, established on biometric characteristics (es., fingerprints, voice print, retinal prints).

or a combination of the above.

Password-based authentication

- Based on pairs:
  - login: the user identifies herself
  - password: the user gives the proofs of her identity
- Authentication method older and more widely used
  - simple
  - cheap
  - easily implementable

    but ....
  - also the weakest

Vulnerabilities of passwords

- Often passwords can be:
  - easily guessed (guessing)
  - read by people observing the legitimate users typing it in (snooping)
  - observed by third parties when travelling on the network (sniffed)
  - acquired by third parties impersonating the login interface (spoofing)
- Anybody that acquires the password of a user can impersonate the user (masquerading) in getting access to the system.

Vulnerabilities of passwords

One of the primary causes of password vulnerability is due to the users that do not choose or manage them properly.

MY KEYBOARD IS BROKEN. IT ONLY TYPES ASTERISKS FOR PASSWORDS.
DOGBERT'S TECH SUPPORT
TRY CHANGING YOUR PASSWORD TO FIVE ASTERISKS.
I HOPE I CAN REMEMBER IT.
www.dilbert.com
scottadams@aol.com
© 2001 United Feature Syndicate, Inc.

Causes of password vulnerability

The first step to limit password vulnerability is good password managament.
Often passwords are vulnerable because users do not put enough care:

- do not change password for a long time
- share passwords with colleagues and friends
- choose “weak” passwords because they are easy to remember (e.g., name or date of births of relatives or pets)
- use the same password on different computers
- write the passwords on paper to be sure to not forget it.

Choosing passwords

Good password management requires users to

- change their password often
- keep their passwords private
- choose passwords that cannot be easily guessed.
  A good password should
  - be of at least 8 characters and use a large set of characters (alphanumeric and special characters).
  - be not easily guessable and do not correspond to words in dictionaries (or slight variations of them).
  - easy to remember, otherwise users
    - would write it down
    - would forget it (denials-of-service)

...... Unfortunately, often these basic principles are not followed.

Control on passwords

Several systems use automatic controls to avoid weak password.

- restriction on the lenght and on the minimum number of characters, requiring mixing numerical and alphanumerical characters
- control against dictionaries and rejection of password belonging to the natural language (to prevent dictionary attacks).
- maximum time of validity for passwords (users are required to change their password when it expires).
  - keep history of recent passwords
  - keep minimum time of validity

Controls on passwords

In alternative

- the password can be chosen by the system.
  - non always well accepted (password can be difficult to remember)
  - problem of password distribution.
- use of sequences one-time-password (often users write them down to remember them).

Authentication based on possess

- Based on posses by users of tokens (of a size of a credit card).
- Each token has a cryptographic key (stored in the token) that can be used to prove the identity of the token to a computer.
- Tokens are safer than passwords
  - by keeping control on the tokens users maintain control on their identity

Vulnerabilities of tokens

- Token-based authentication proves only the identity of the token, not the identity of the user
  - tokens can be lost, stole, forged
  - everybody who acquires a token can impersonate the user
- often token-based authentication can be combined with authentication based on knowledge.
  - To masquerade as a user third parties need both, to have the token and to know the password.
  - Eg., ATM: ATM card + ATM code

Token-based authentication

Two kinds of token:

- memory card: have memory but do not have processing power.
  - cannot check the password of encrypt it for transmission.
  - the password is transmitted in the clear.
    - vulnerable to sniffing attacks
    - require the user to trust the authentication server
- smart token: have processing capabilities.

Authentication based on user characteristics

- Based on biometric characteristics of the user:
  - physical characteristics: fingerprints, shape of the hand, prints of the retin, or of the face, ...
  - behavioral characteristics: handwritten signature, voice,“keystroke dynamic”....
- Requires an initial enrollment phase that
  - performs several measures on the characteristic
  - defines a profile (template)

Authentication based on characteristics of the user

- Authentication: comparison between the characteristic measured for the user with the stored template.
- Authentication succeeds if they correspond (provided a tolerance interval)
- We cannot require exact match.
- It is important to verify the tolerance level so to maximize successes (correct authentication of legitimate users) and minimize failures.

Biometric authentication

- Even if less accurate is the strongest form of authentication
  - eliminates vulnerability due to impersonation.
- Limited use
  - expensive (need expensive hardware)
  - intrusive (not always well accepted)
    - Retinal scanners are one of the most accurate measures but there are debates on possible eye damages thy can cause/
  - political and social debates for potential threats to privacy.

Which authentication technique should be used?

- There are “stronger” techniques, not “better”, techniques
  - trade-off between costs and benefits: weaker methods can work well in several cases
  - Passwords are (and will be for a while) the authentication technique most used.
- From a purely technical point of view the best authentication method is given by the combination of:
  - biometric authentication between user and token
  - mutual cryptography-based authentication between token and system.

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)

Authentication also necessary for accountability and establishing responsability.

Each principal (logged subject) should correspond to a single user $\Longrightarrow$ no shared accounts

Policies, Models, and Mechanisms

In studying access control, it is useful to separate

- Policy 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)

- Mechanism Implements the policies via low level (software and hardware) functions

Such separation 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

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?

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

command REVOKE$_{read}$(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$_{read}$(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$_{read}$(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 $c(x_1, \ldots, x_k)$ on a system state $Q = (S, O, A)$ causes the transition to a state Q' such that:

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

where

- $op_1^* \ldots op_n^*$ are primitive operations in c
- the formal parameters $(x_1, \ldots, x_k)$ 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 helting 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

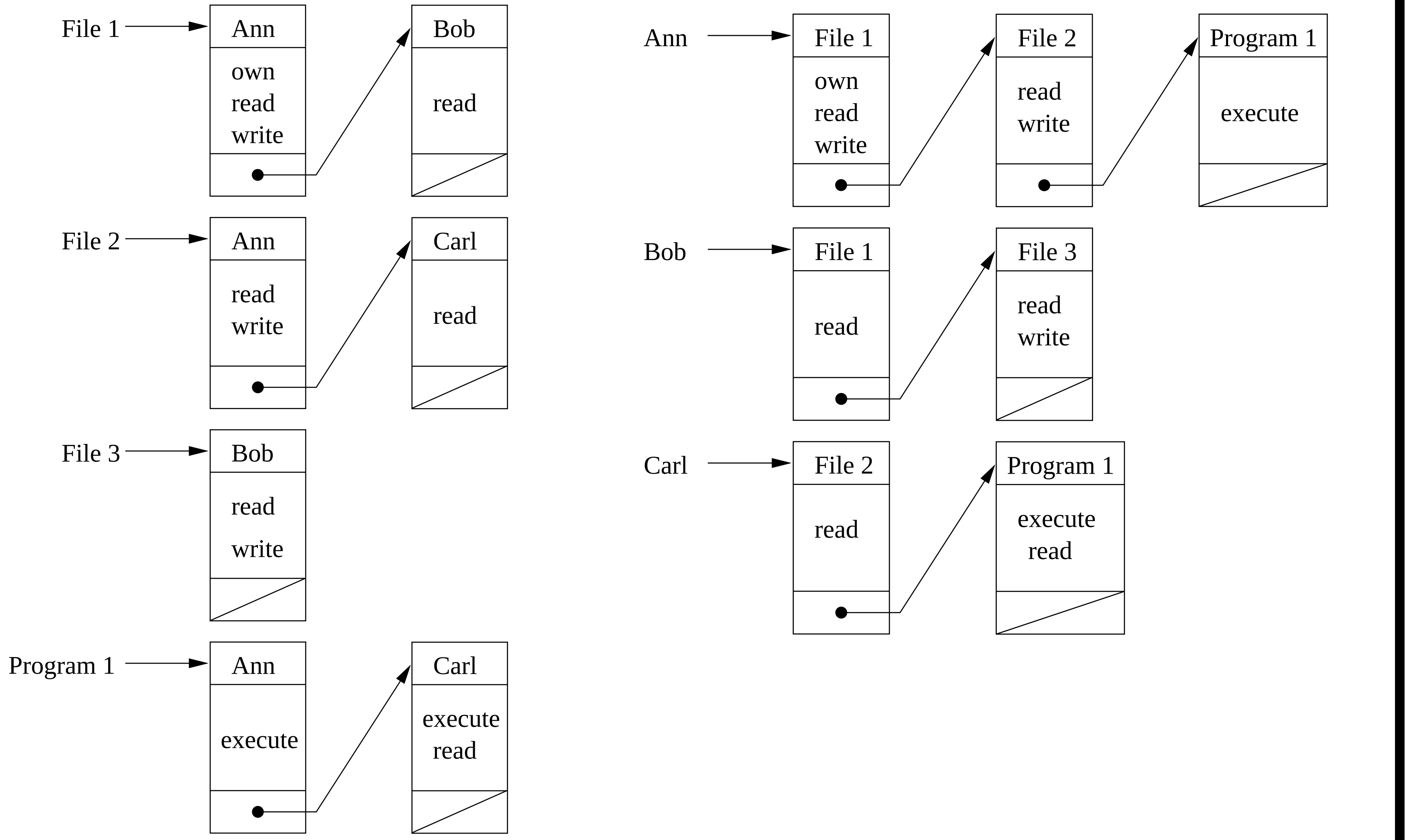
Alternative approaches

- 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

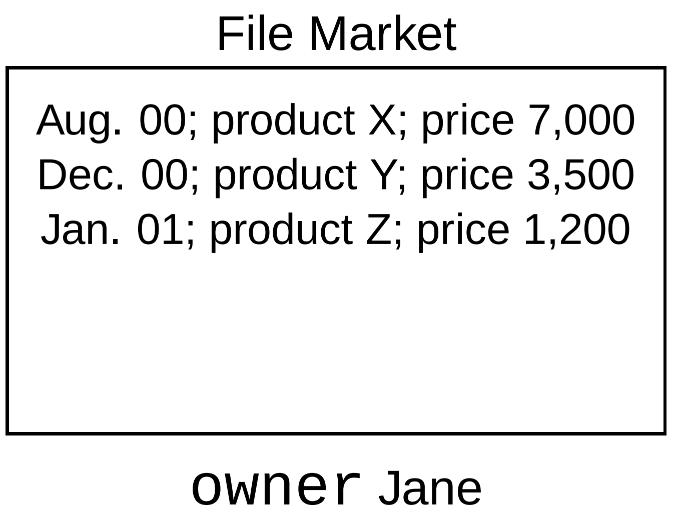
No control on what happens to information once released

$\Longrightarrow$ 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



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

The Trojan Horse problem

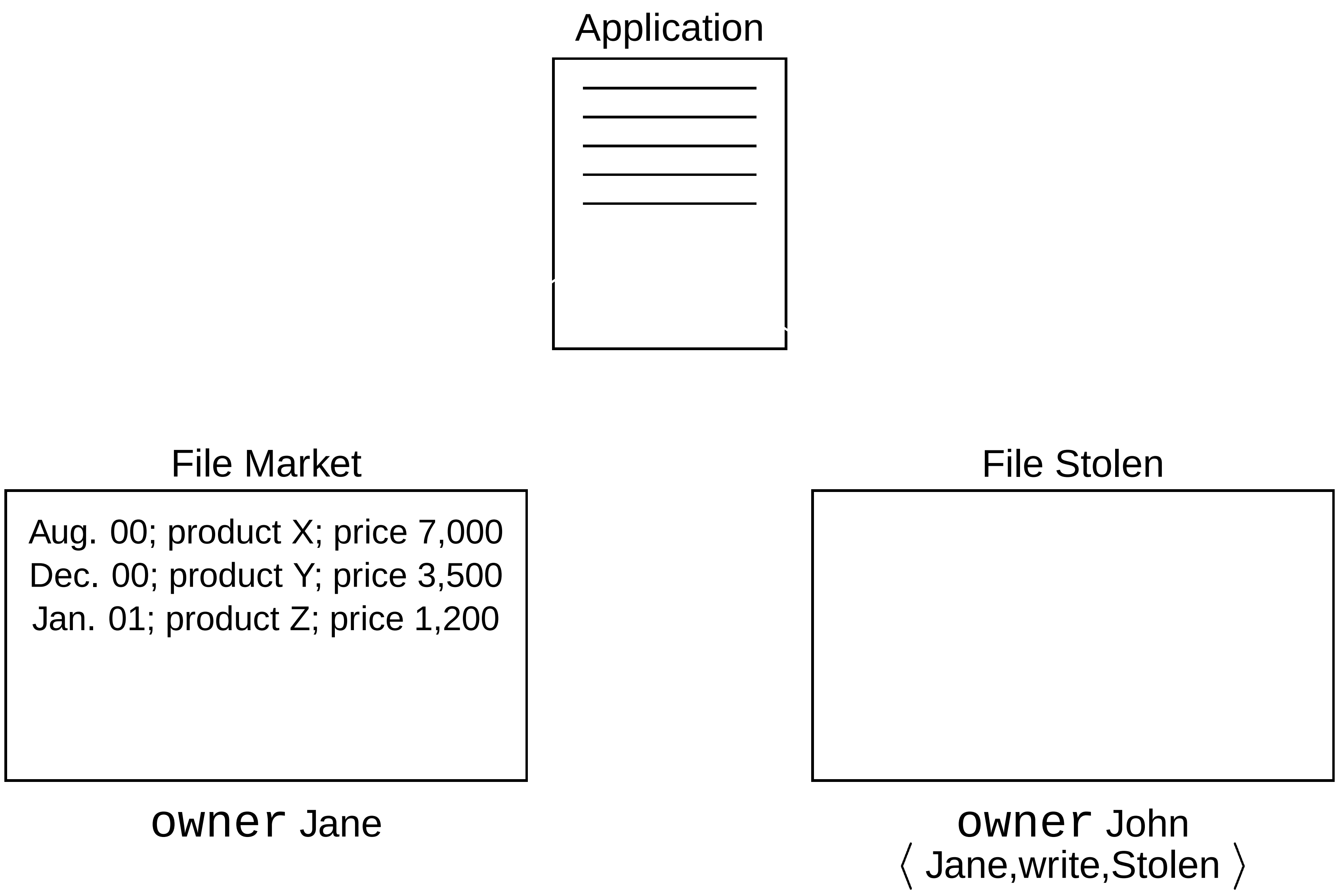
File Market

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

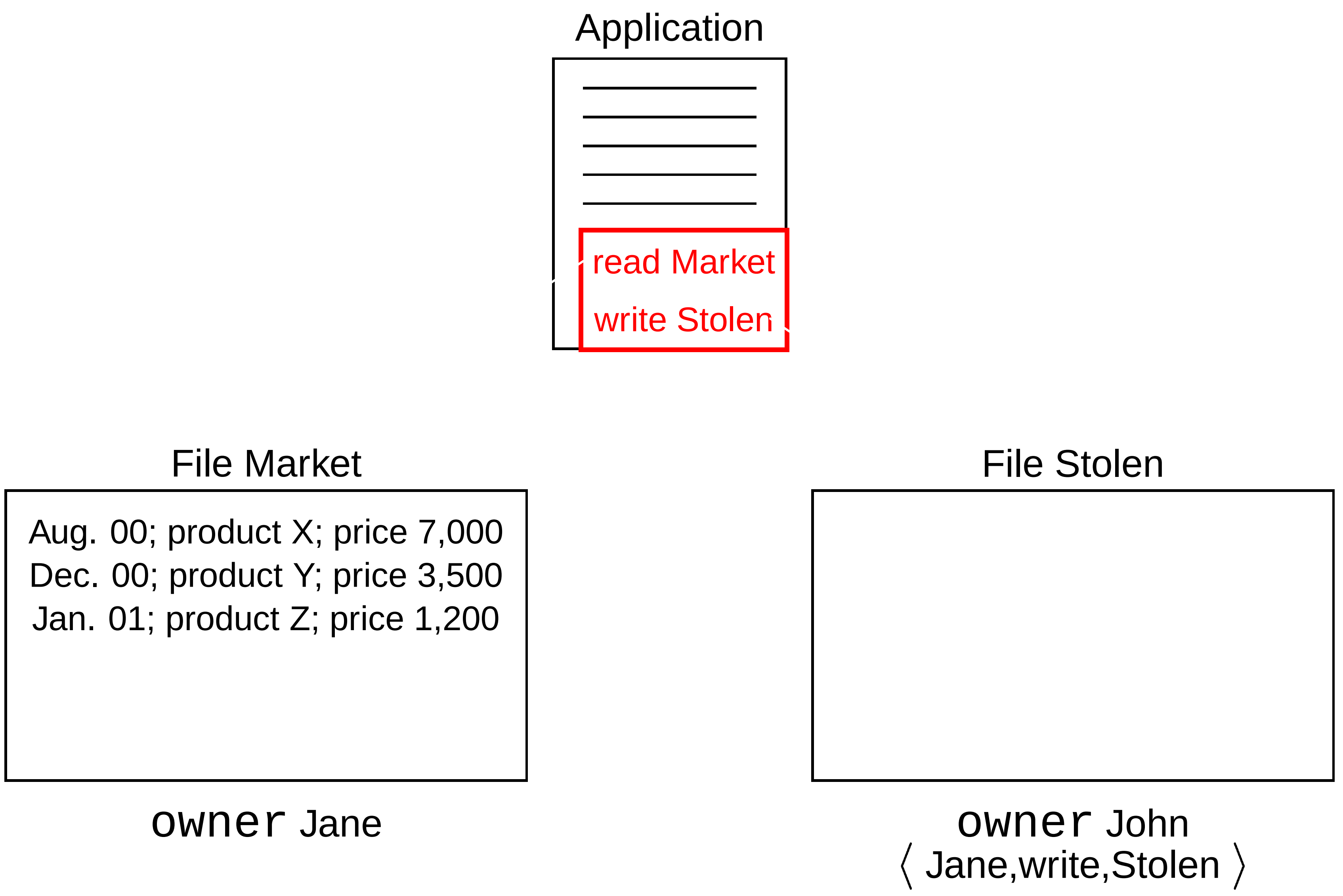
`owner` Jane

John

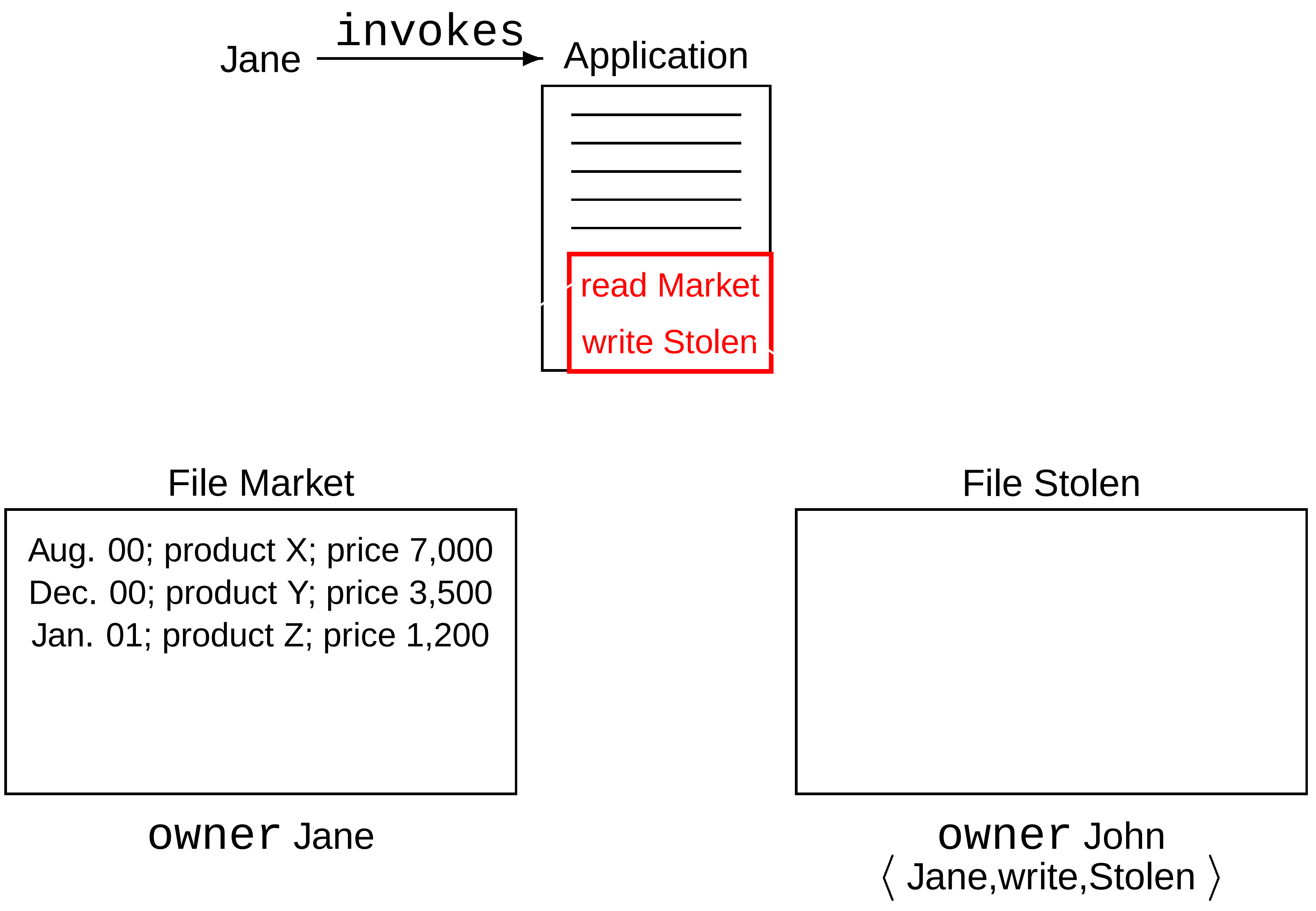
The Trojan Horse problem


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 ⟩

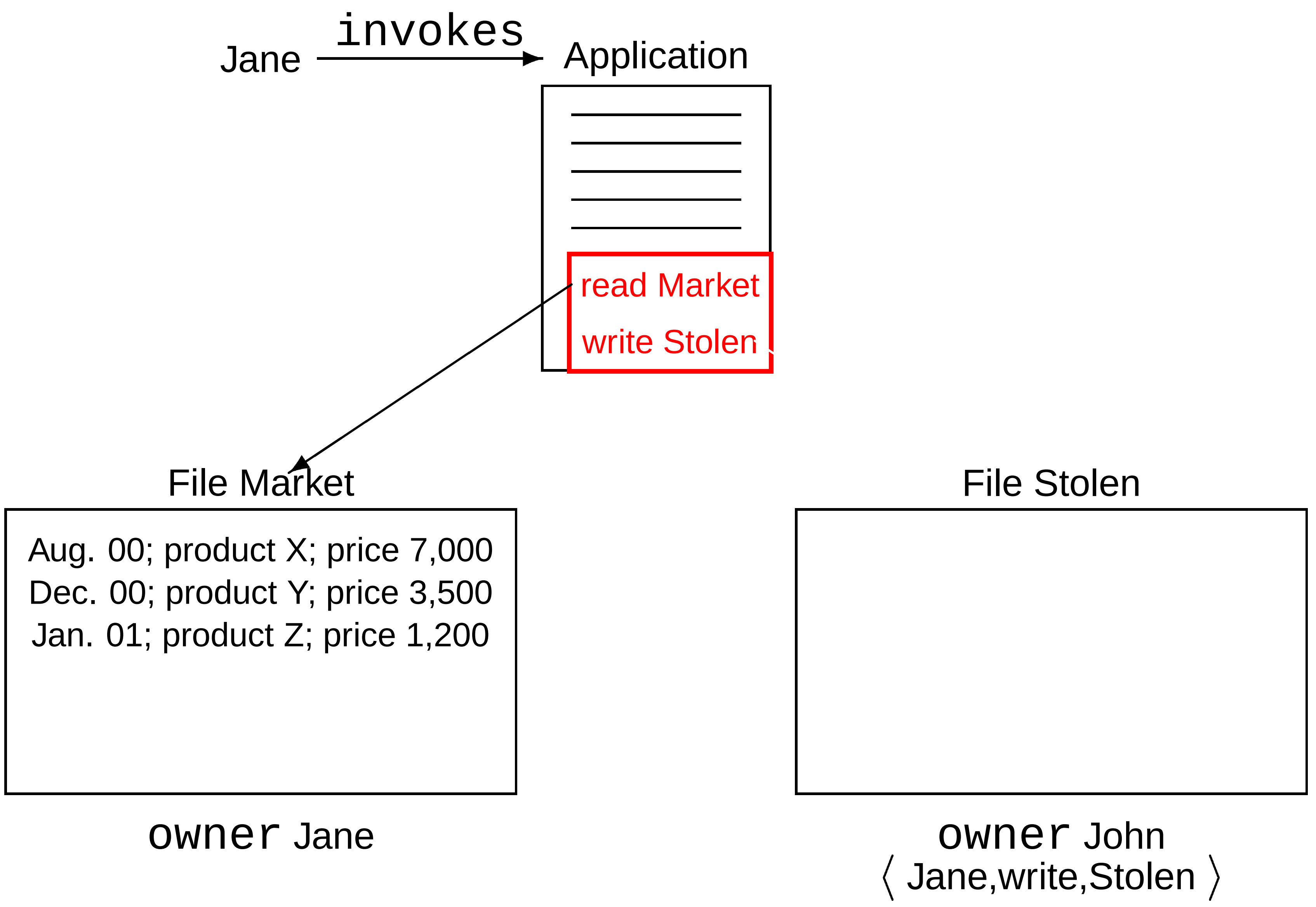
The Trojan Horse problem



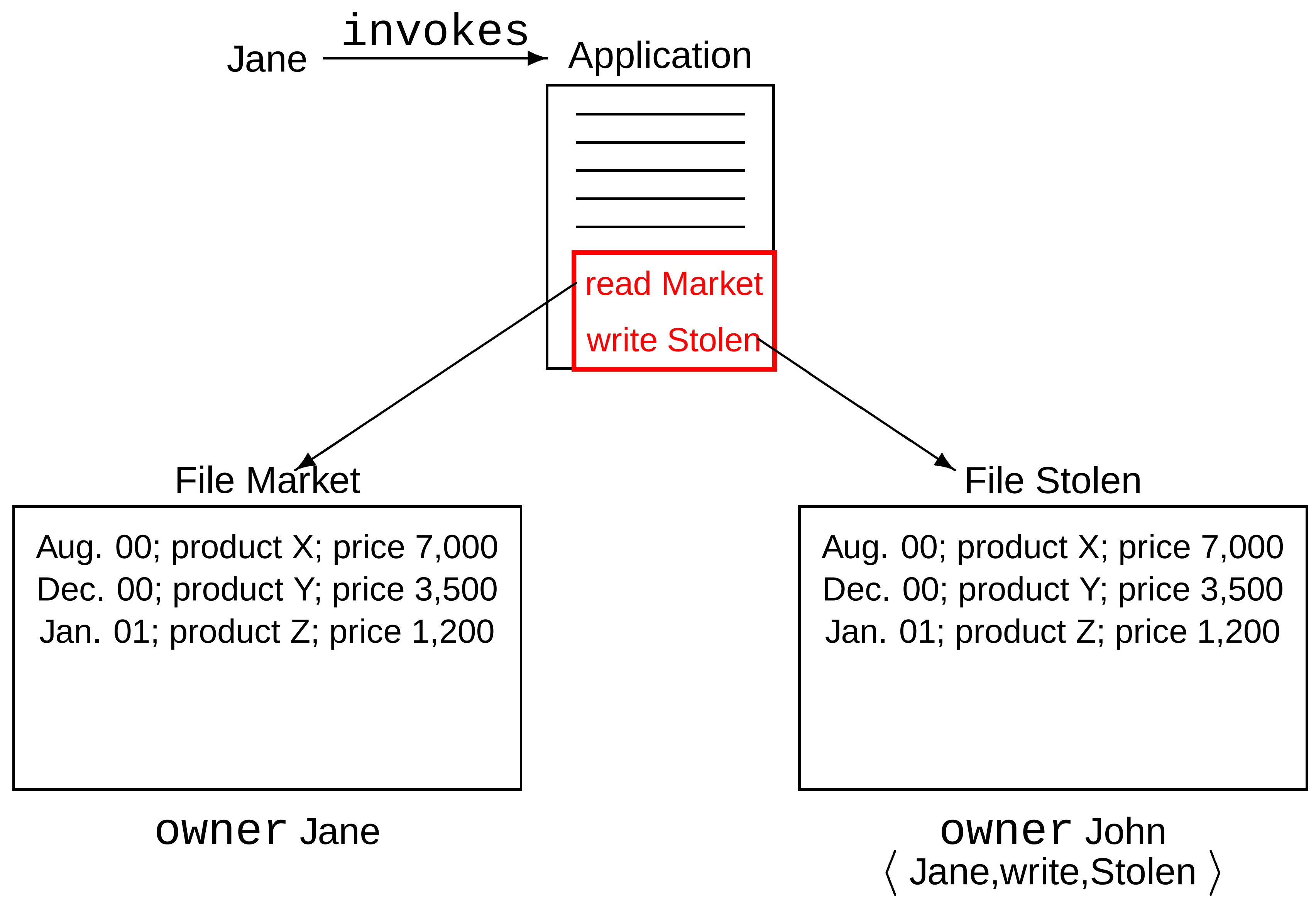
The Trojan Horse problem



The Trojan Horse problem



The Trojan Horse problem


Jane
invokes
Application
read Market
write Stolen
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
Aug. 00; product X; price 7,000
Dec. 00; product Y; price 3,500
Jan. 01; product Z; price 1,200
owner John
⟨ Jane,write,Stolen ⟩

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

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

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

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 $t \succeq y \Longrightarrow t \succeq z$.

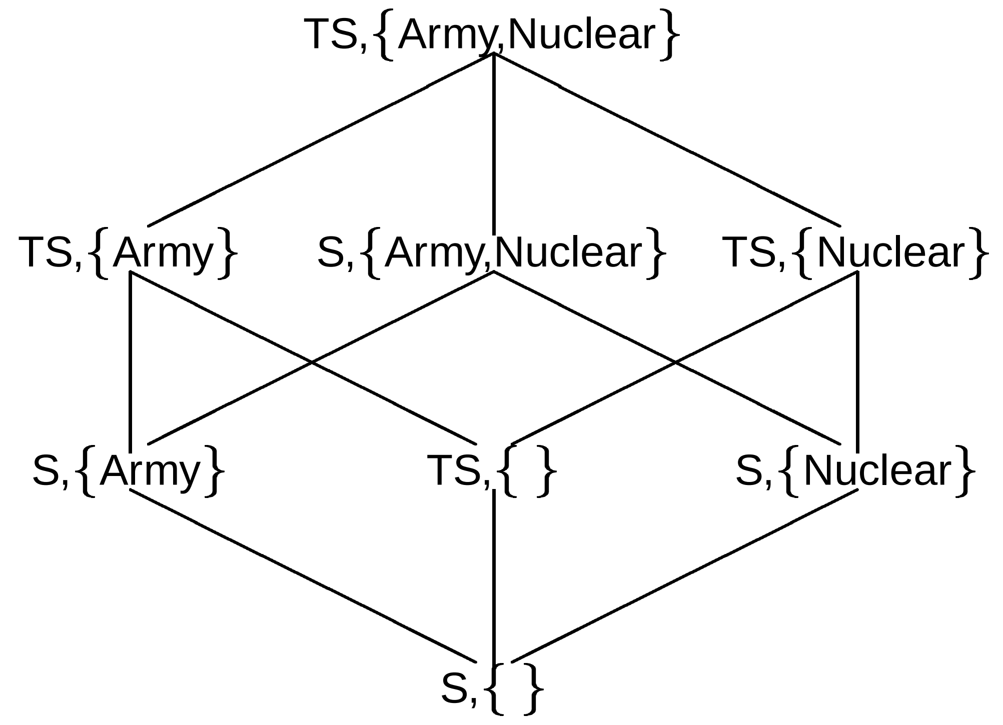
Greatest lower bound $\forall x, y \in SC : \exists\, !z \in SC$

- $x \succeq z$ 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 its 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 basic principles remain the same.

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

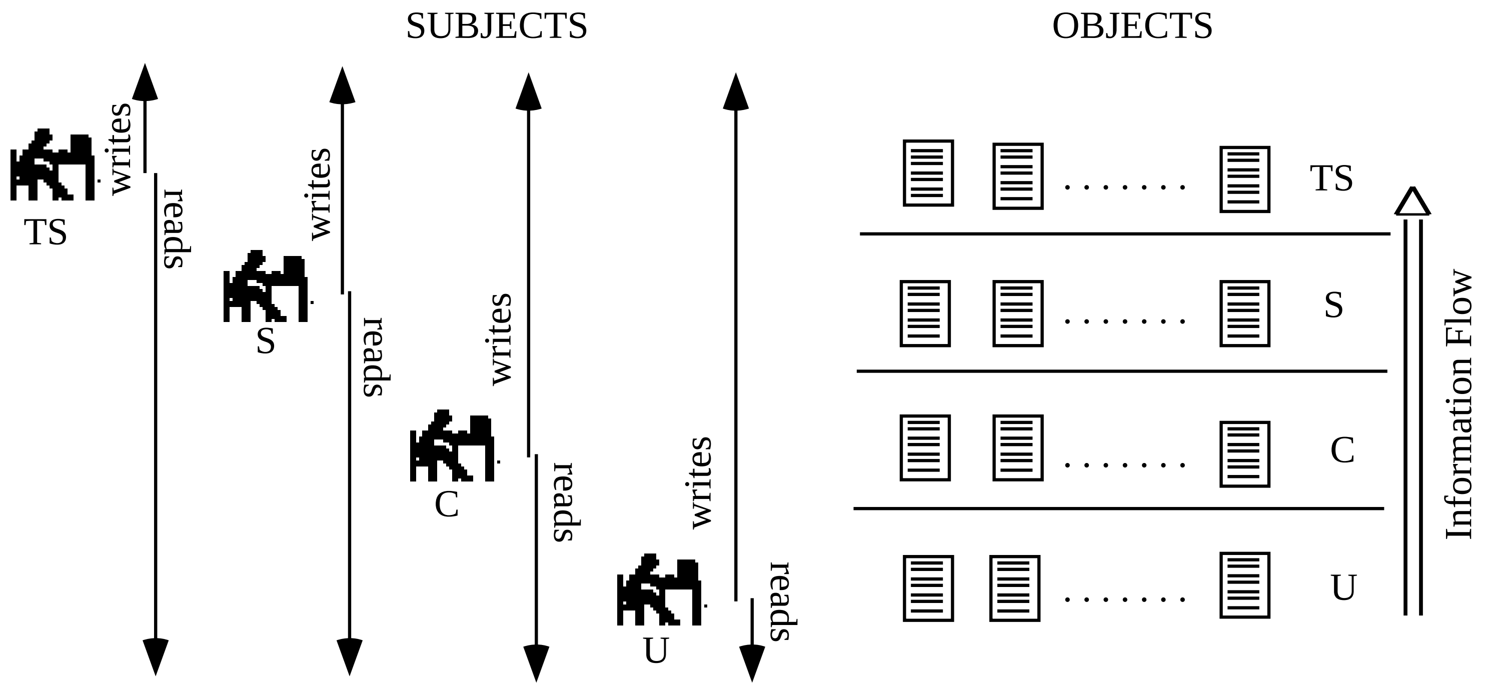
***-property** A subject s can write object o only if $\lambda(o) \succeq \lambda(s)$

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

$\Longrightarrow$ NO WRITE DOWN
NO READ UP

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
- $\lambda : S \cup O \to L$: 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 $T : V \times R \to V$ 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 place on T, which can be exploited to leak information

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)

Lattice models of information flow control

Extends Bell La Padula to include notion of information flow

Assume lattice $L = (SC, \succeq)$ of security classes and classification of objects, both logical (files) and physical (memory segments)

Information state of a system described by value and security class of objects

Access classes assigned to objects can be

- fixed – static binding
- variable – dynamic binding

A flow from object x to object y ($x \rightarrow y$) is authorized iff $\underline{y} \succeq \underline{x}$, where $\underline{y}$ and $\underline{x}$ are the classes of y and x after the flow

Definition of flow based on analysis of programs. E.g.,

$y := x$ (explicit flow) $z := x; y := z$ (indirect flow)
if $x = 0$ **then** $y = 0$ (implicit flow)

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 resourse 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 down information.
- A high level process can lock shared resourses and modify the <u>response times</u> 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 process 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, DBMS 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

Multilivel 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.

Multilivel 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

Multilivel relation – 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

Instance U

Name	λ_{N}	**Dept**	λ_{D}	**Salary**	λ_{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 object with the same name but different classification

  $\Longrightarrow$ 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 (es., Trusted 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.

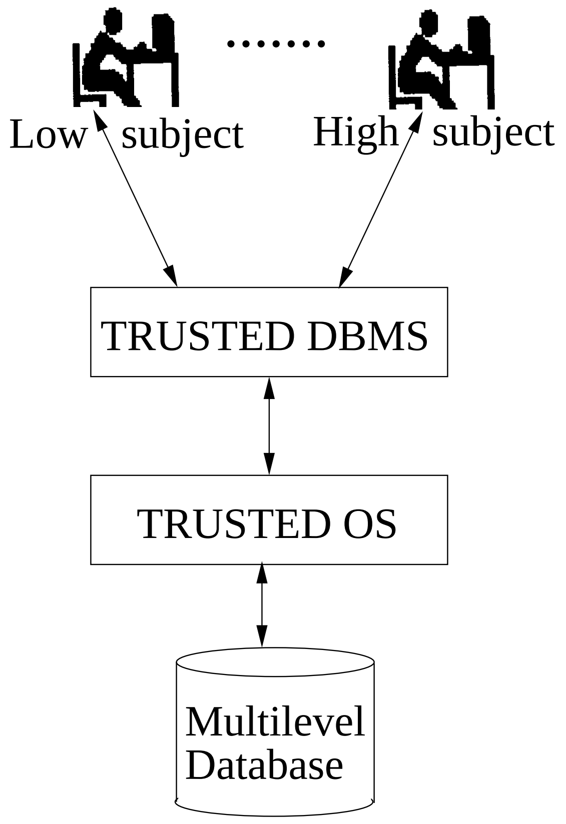
- Trusted computing base: data are partitioned in different databases, one for each level.

  Only the operating system needs to be trusted.

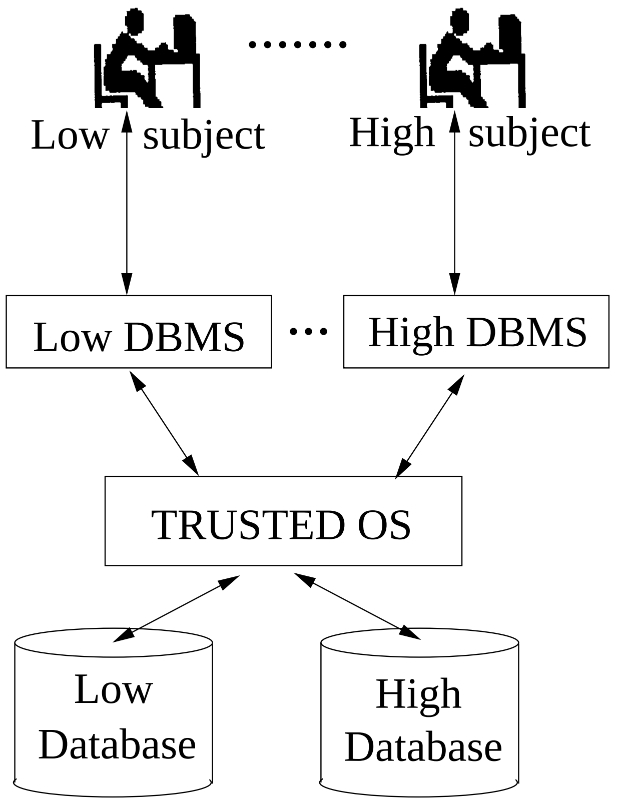
  Each DBMS is confined to access data that can be read at its level (no-read-up).

  (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

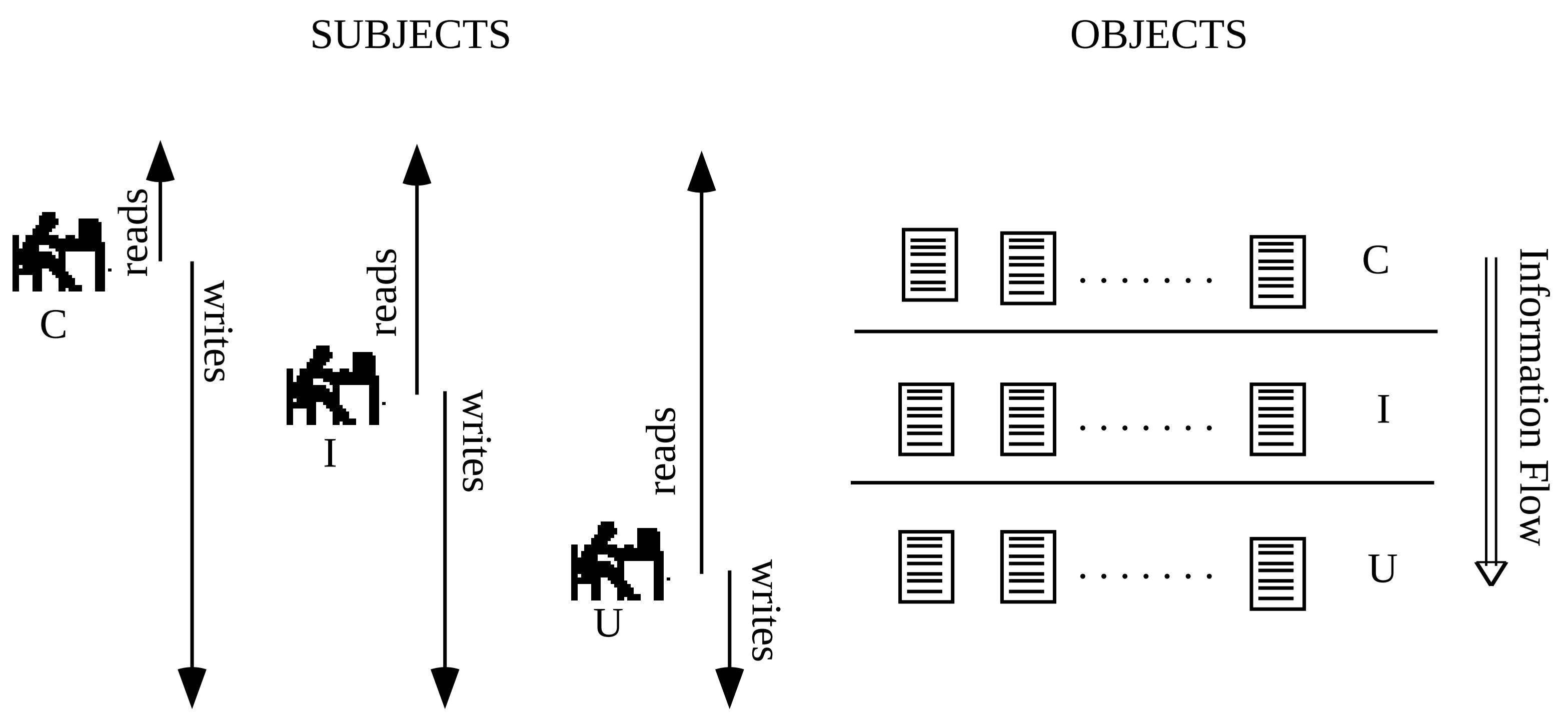
Strict integrity policy Based on principles dual to those of BLP

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

$\Longrightarrow$ NO WRITE UP
NO READ DOWN

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

- 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

- 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

Limitation of Biba policies

Biba principles control only compromises due to improper flows.

Integrity is a more complex concept.

A proposal by Clark e Wilson defines four basic criteris to safeguard integrity:

1. Authentication.
2. Audit.
3. Well-formed transactions Users cannot manipulate data arbitrarily but only through procedures preserve integrity (e.g., double entry bookeeping in accounting). Well-formed transation must provide serializability, recovery, and concurrency control.
4. Separation of duty When assigning users the programs he can execute, the system must enforce separation of duty.

Weakness of CW: not formalized $\Longrightarrow$ Difficult to reason about properties.

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)

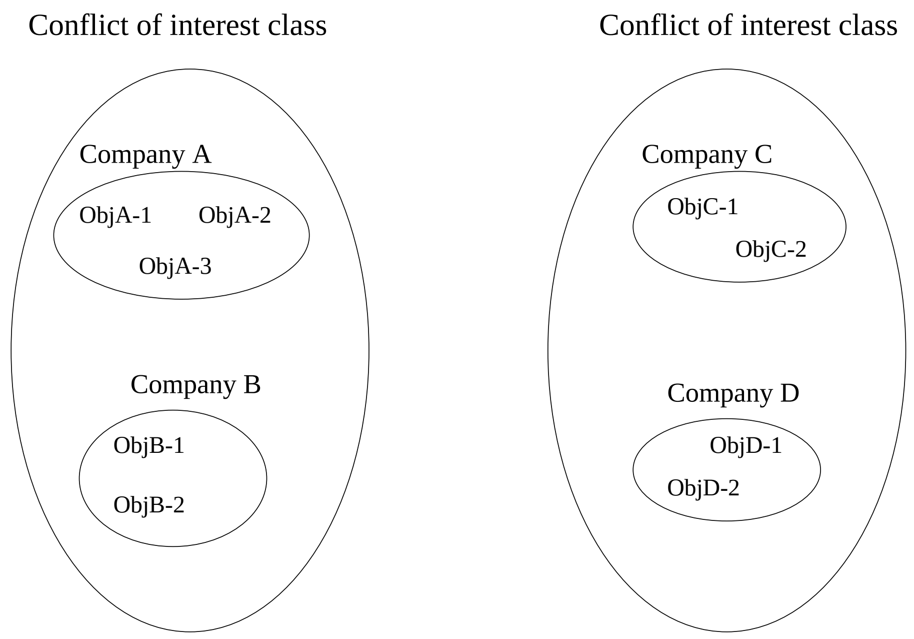
Objects organized hierarchically. Three levels

- 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

Restrict access with mandatory constraints

- If a <u>user</u> accesses an object o in a dataset d he will not be allowed access to objects belonging to datasets in the same conflict of interest class as d



Chinese wall

Other aspects:

- users vs subjects
- history keeping
- accessibility problems (e.g., all users read same dataset)
- sanitization

Chinese wall useful but model not well formalized

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.

- authorization 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

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?

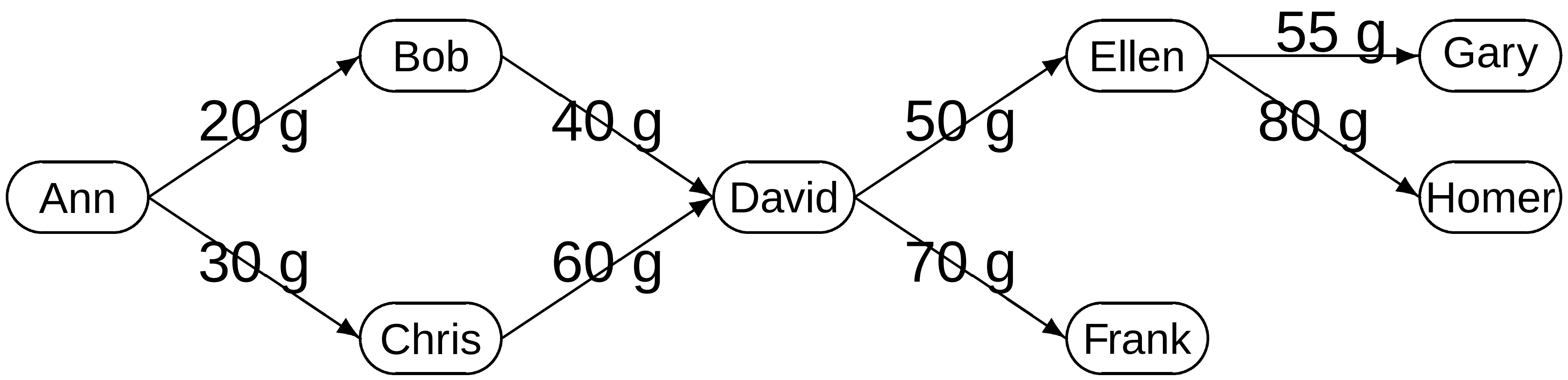
Revocation can be requested:

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.

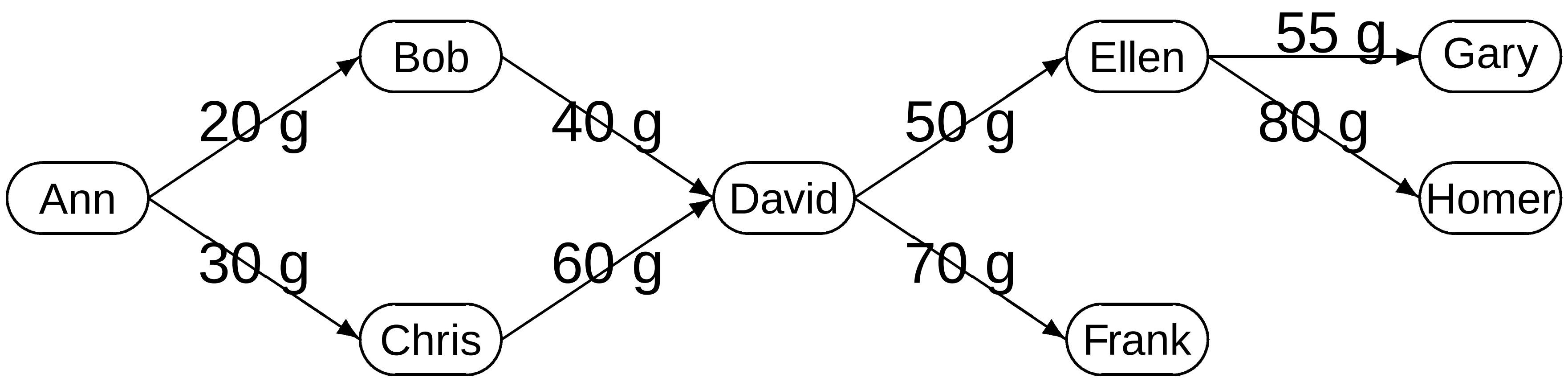
The original (cascade) revocation policy was based on time: all authorizations granted in virtue of an authorization that was being revoked were deleted, regardless of other later authorizations that the user had received.

Recursive revocation in SQL – example

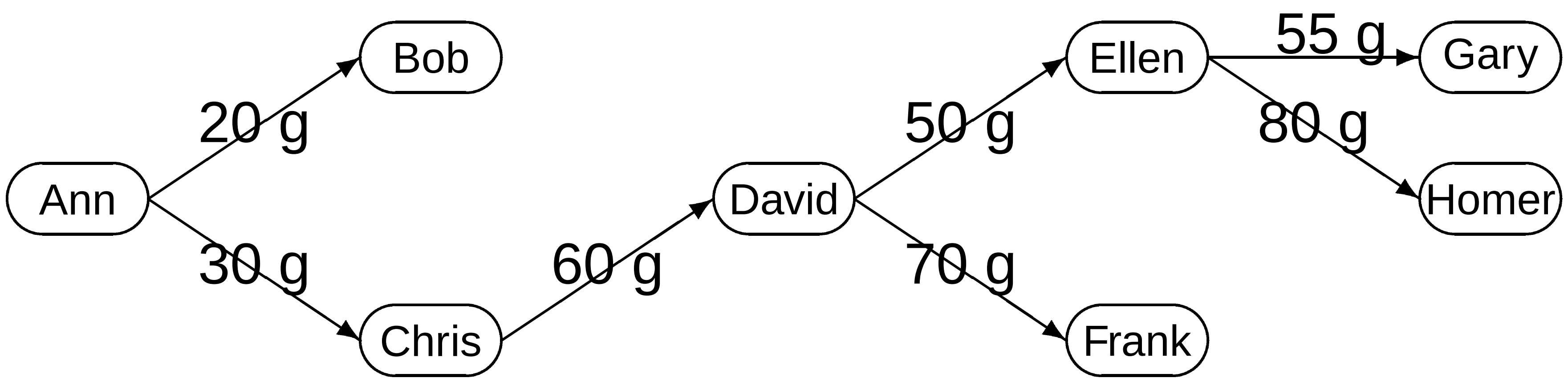


Bob revokes the autorization from David

Recursive revocation in SQL – example



Bob revokes the authorization from David



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

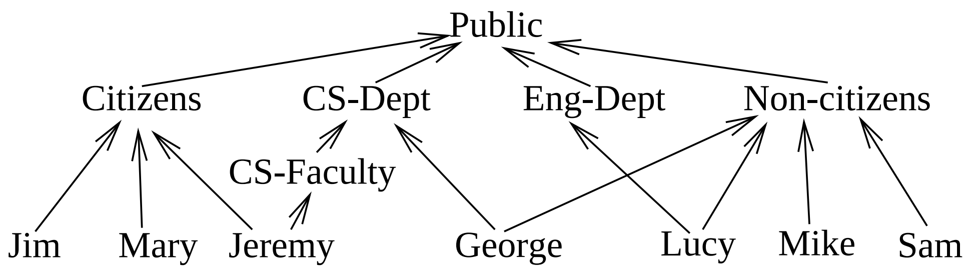
- support abstractions (grouping of them). Usually hierarchical relationships: users/groups; objects/classes; files/directories; .....

  Authorizations may propagate along the hierarchies

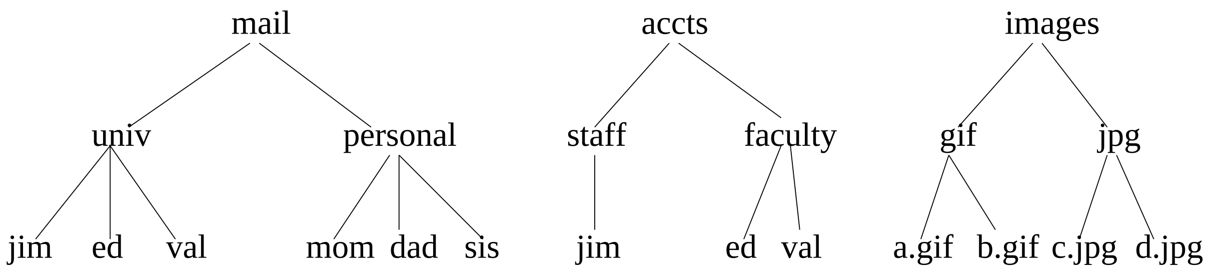
Hierarchical data systems – 2

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

$$\begin{tikzcd}
 & G_1 + & \\
- G_2 \arrow[ur] & G_3 \arrow[u] & G_4 - \arrow[ul] \\
G_5 \arrow[u] & G_6 - \arrow[u] & \\
u_1 \arrow[u] \arrow[uur] & & u_2 \arrow[ul] \arrow[uu]
\end{tikzcd}$$

explicit authorizations

Examples of conflict resolution

$$\begin{tikzcd}
 & G_1 + & \\
- G_2 \arrow[ur] & G_3 + \arrow[u] & G_4 - \arrow[ul] \\
- G_5 \arrow[u] & G_6 - \arrow[u] & \\
- u_1 \arrow[u] \arrow[uur] & & u_2 - \arrow[uul] \arrow[uu]
\end{tikzcd}$$

most specific

$$\begin{tikzcd}
 & G_1 + & \\
- G_2 \arrow[ur] & G_3 + \arrow[u] & G_4 - \arrow[ul] \\
- G_5 \arrow[u] & G_6 - \arrow[u] & \\
\substack{+ \\ -} u_1 \arrow[u] \arrow[uur] & & u_2 - \arrow[uul] \arrow[uu]
\end{tikzcd}$$

most specific along a path

Most specific take 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 <u>is granted access</u> 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 <u>is denied access</u> 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

Example of a logic based authorization specification language

cando(o,s,$\langle$*sign*$\rangle$a) : explicit authorizations.

dercando(o,s,$\langle$*sign*$\rangle$a) : defines implied authorizations

do(o,s,$\langle$*sign*$\rangle$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., $\mathtt{owner}(\mathtt{user},\mathtt{object}),\mathtt{supervisor}(\mathtt{user1},\mathtt{user2})$).

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 **`do`**$(\_, \_, +a)$ body may contain **`cando`**, **`dercando`**, **`done`**, `hie-` and `rel-` literals.
4	**`do`**	in the case when head is of the form **`do`**$(o, s, -a)$ body contains just one literal $\neg$**`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

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

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

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)

Role-based access control model

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

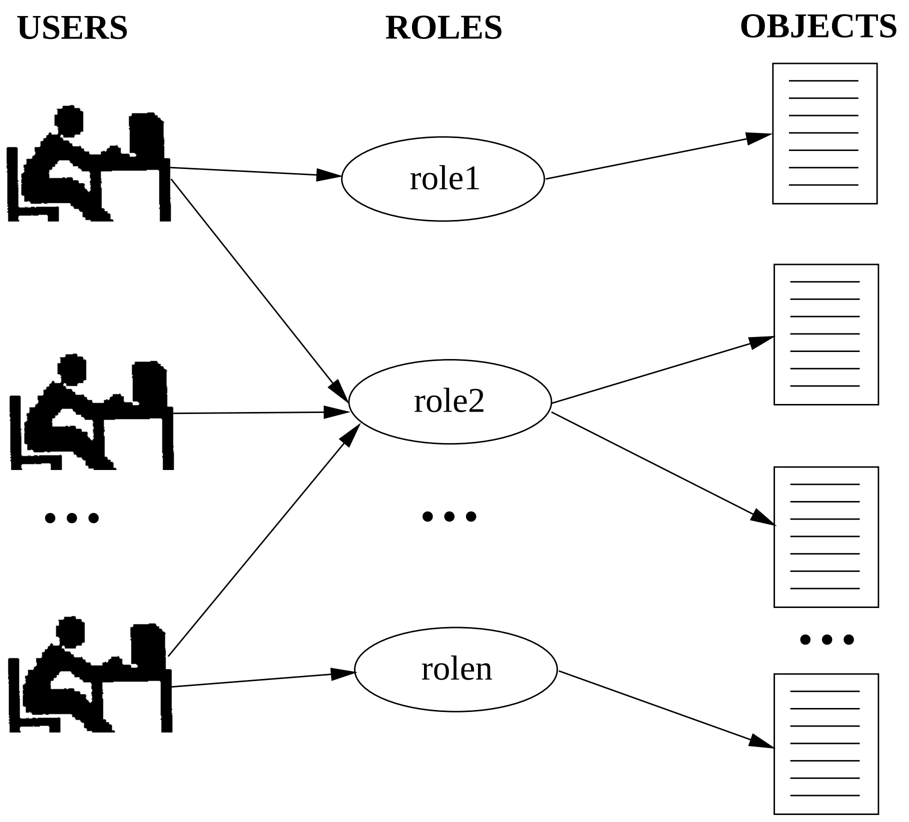
Access of users to objects mediated by roles

- 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

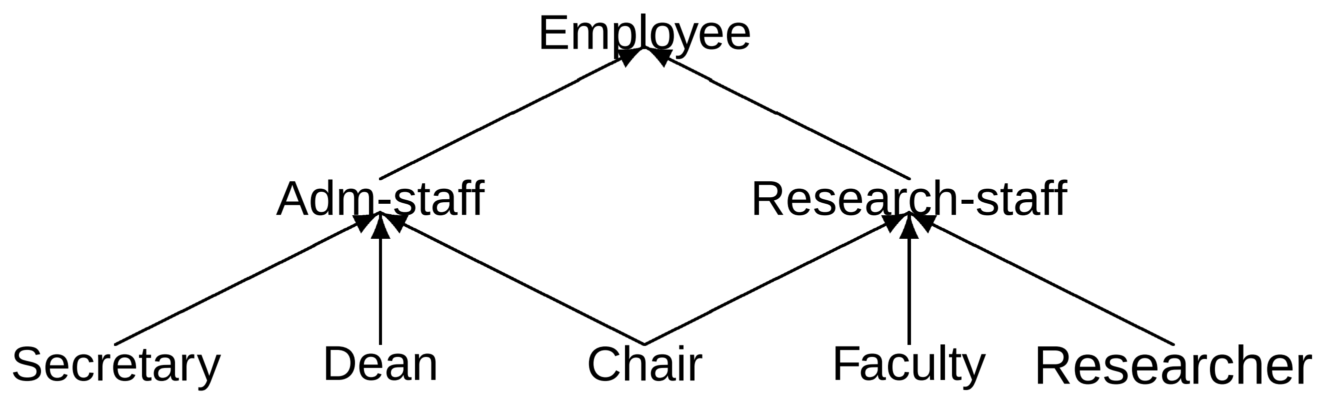
- *group*: set of users
- *role*: set of privileges

RBAC



Role-based access control model – 2

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 activat 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 to associated 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 to enforce separation of duty (split privileges among different subjects).

Role-based models – 4

Role-based models try to look at the real-world applications but there are still things to do .....

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

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

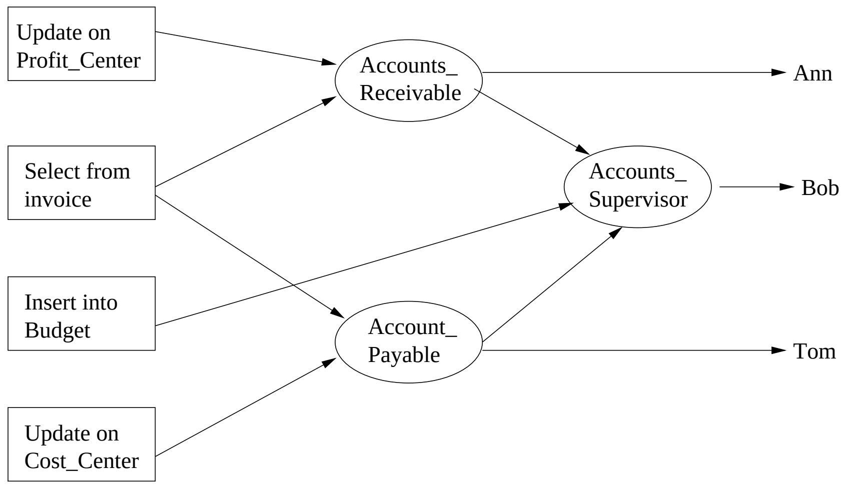
Example: order-goods, send-order, record-invoice, pay. Four employees.
Protection requirements:
<u>at least two people must be involved in the process</u>

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.

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 enable 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

Access control in open systems

Assumptions of traditional systems result limiting

- different independent access control policies may need to maintained and applied in combination

  $\Longrightarrow$ Need for policy composition

- authentication may not always be possible or wanted

  $\Longrightarrow$ Need for access control based on digital credentials

- need to augment expressiveness of access control (e.g., purpose-based restrictions) and support of dynamic conditions (e.g., payment)

  $\Longrightarrow$ Need for interactive access control systems

Policy composition

Security policies result from multiple input requirements

- Federated/Legacy Databases, Datawarehouses. The security policies of all individual data sources must be *harmonized*
- Statistical Repositories. Data owners' requirements must be combined with the repository's policy
- Large organizations. Each department may decide its own policy, to some extent
- Laws on Privacy Protection. Must be reconciled with the organization's policy
- Dynamic coalitions. Organizations come together for a limited time framework and need to establish coordinated policies.

Need to combine policies and maintain independence and control on the single components

Composition framework – desiderata (1)

Heterogeneous policy support

- Component policies may be of different kinds (e.g., open, closed)
- They may be specified in arbitrary languages and enforced by different mechanisms
- Possible scenarios: Federated databases, Legacy databases, Datawarehouses

Support of unknown policies

- Some policies may be partially/totally unknown at composition time (black boxes)
- Some are never available “as a whole”, and can only be queried at run-time
- Possible scenario: incorporating *central administration approval* — possibly implemented as an external module

Composition framework – desiderata (2)

Controlled interference

- Possible problem: merging rules for open and closed policies
  - (Closed) *grant access if* $(s, o, +a)$ *exists*
  - (Open) *grant access if* $(s, o, -a)$ *does not exist*

  The first rule has no effect in the combination – positive authorizations are ignored !
- *The internal behavior of component policies should not be affected*

Expressiveness

- A wide range of combinations should be covered

  Maximum/minimum privilege, priority levels, overriding, confinement, refinement, . . .
- With no changes to input specifications
- With no ad-hoc extensions to the structure of authorizations (such as labels for modeling priorities)

Composition framework – desiderata (3)

Support of different abstraction levels

- E.g., inter-department combination, single department policies, etc. (*zooming in/out*)
- Facilitates specification analysis and design
- Facilitates cooperative administration, and agreement on global policies
- Facilitates reasoning on specifications (e.g., correctness proofs)

Formal semantics

- The composition language should be
  - declarative
  - implementation independent
  - based on a solid formal framework to guarantee
    - non-ambiguous behavior
    - sound reasoning about specifications

Preliminary Concepts for the Algebra (1)

Basic domains

- S, set of subjects (e.g., users, groups, roles, applications)
- O, set of objects (e.g., files, relations, classes)
- A, set of actions (e.g., operations, methods)

Authorization term

- Triple of the form (*s*,*o*,*a*), where *s* is a constant in S or a variable over S, *o* is a constant in O or a variable over O, and *a* is a constant in A or a variable over A

Preliminary Concepts for the Algebra (2)

Policy

- A set of ground authorization terms

$$P = \{(s,o,a) \mid s \in \mathsf{S}, o \in \mathsf{O}, a \in \mathsf{A}\}$$

- $(s,o,a) \in P \Leftrightarrow P$ permits this access
- *P* represents the *outcome* or *semantics* of an authorization specification, where, for composition purposes, it is irrelevant how specifications have been stated and their outcome computed.

Parameters of the Algebra

Our algebra is parametric with respect to the following two languages.

1. An *authorization constraint language* $\mathcal{L}_{acon}$ and a semantic relation $\mathsf{satisfy} \subseteq (\mathsf{S} \times \mathsf{O} \times \mathsf{A}) \times \mathcal{L}_{acon}$
(e.g., $($*s op* *s*$_0)$, $($*o op* *o*$_0)$, $($*a op* *a*$_0)$, *p(s)*, *p(o)*, or *p(a)*)

2. A *rule language* $\mathcal{L}_{rule}$ and a semantic function $\mathsf{closure} : \wp(\mathcal{L}_{rule}) \times \wp(\mathsf{S} \times \mathsf{O} \times \mathsf{A}) \rightarrow \wp(\mathsf{S} \times \mathsf{O} \times \mathsf{A})$
(e.g., simple Horn clauses, built from authorization terms and base predicates, of the form $(s, o, a) \leftarrow L_1 \wedge \ldots \wedge L_n$, where $L_i =$ (*x op y*), *p(x)*, or auth. term)

Policy Expressions - Syntax

$$
\begin{aligned}
E &::= \mathbf{id} \mid E + E \mid E\&E \mid E - E \mid E\hat{}C \mid o(E, E, E) \mid \\
&\qquad E * R \mid T(E) \mid (E) \\
T &::= \tau\,\mathbf{id}.T \mid \tau\,\mathbf{id}.E \\
C &::= \mathbf{var\ op\ val} \mid \mathbf{pred}\ (\ arg\_list\ ) \\
R &::= A \leftarrow A\_list \\
A &::= (s, o, a) \\
&\ldots
\end{aligned}
$$

NB: C and R can be replaced by arbitrary languages

Policy Expressions - Semantics (1)

- $[\![P_1 + P_2]\!]_e \stackrel{\text{def}}{=} [\![P_1]\!]_e \cup [\![P_2]\!]_e$ e.g., maximum privilege
- $[\![P_1 \,\&\, P_2]\!]_e \stackrel{\text{def}}{=} [\![P_1]\!]_e \cap [\![P_2]\!]_e$ e.g., minimum privilege
- $[\![P_1 - P_2]\!]_e \stackrel{\text{def}}{=} [\![P_1]\!]_e \setminus [\![P_2]\!]_e$ e.g., exceptions, explicit prohibitions
- $[\![P * R]\!]_e \stackrel{\text{def}}{=} \mathsf{closure}(R, [\![P]\!]_e)$ e.g., for inheritance
- $[\![P\hat{\ }C]\!]_e \stackrel{\text{def}}{=} \{(s,o,a) \in [\![P]\!]_e \mid (s,o,a)\ \mathsf{satisfy}\ C\}$ authority conf.
- $[\![o(P_1,P_2,P_3)]\!]_e \stackrel{\text{def}}{=} [\![(P_1 - P_3) + (P_2 \& P_3)]\!]_e$ partial overriding
  $o(P_1, P_2, \hat{\ }C) = o(P_1, P_2, P_1\hat{\ }C)$
- $[\![\tau X.P]\!]_e(S) \stackrel{\text{def}}{=} [\![P]\!]_{e[X/S]}$ $S \subseteq \mathsf{S} \times \mathsf{O} \times \mathsf{A}$ a function over policies
  $[\![(\tau X.P)(P_1)]\!]_e \stackrel{\text{def}}{=} [\![\tau X.P]\!]_e([\![P_1]\!]_e) = [\![P]\!]_{e[X/[\![P_1]\!]_e]}$

Policy Expressions - Semantics (2)

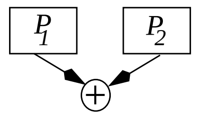
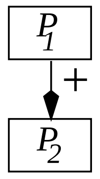
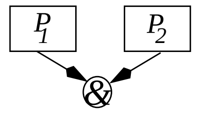
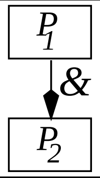
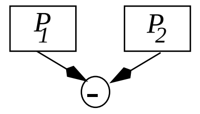
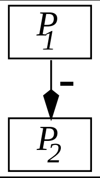
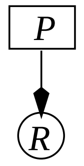
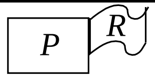
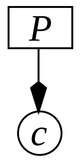
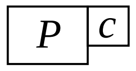
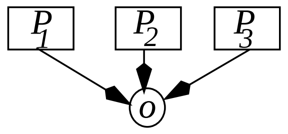
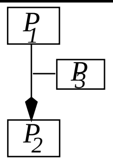
Environment

- Partial mapping from policy identifiers to sets of ground authorizations

$$e[X/S](Y) = \begin{cases} S & \text{if } Y = X \\ e(Y) & \text{otherwise} \end{cases}$$

- $[\![X]\!]_e \overset{\text{def}}{=} e(X)$

Graphical Representation

$P_1 + P_2$		
$P_1 \& P_2$		
$P_1 - P_2$		
$P * R$		
$P\hat{}c$		
$o(P_1,P_2,P_3)$		

An Example (1)

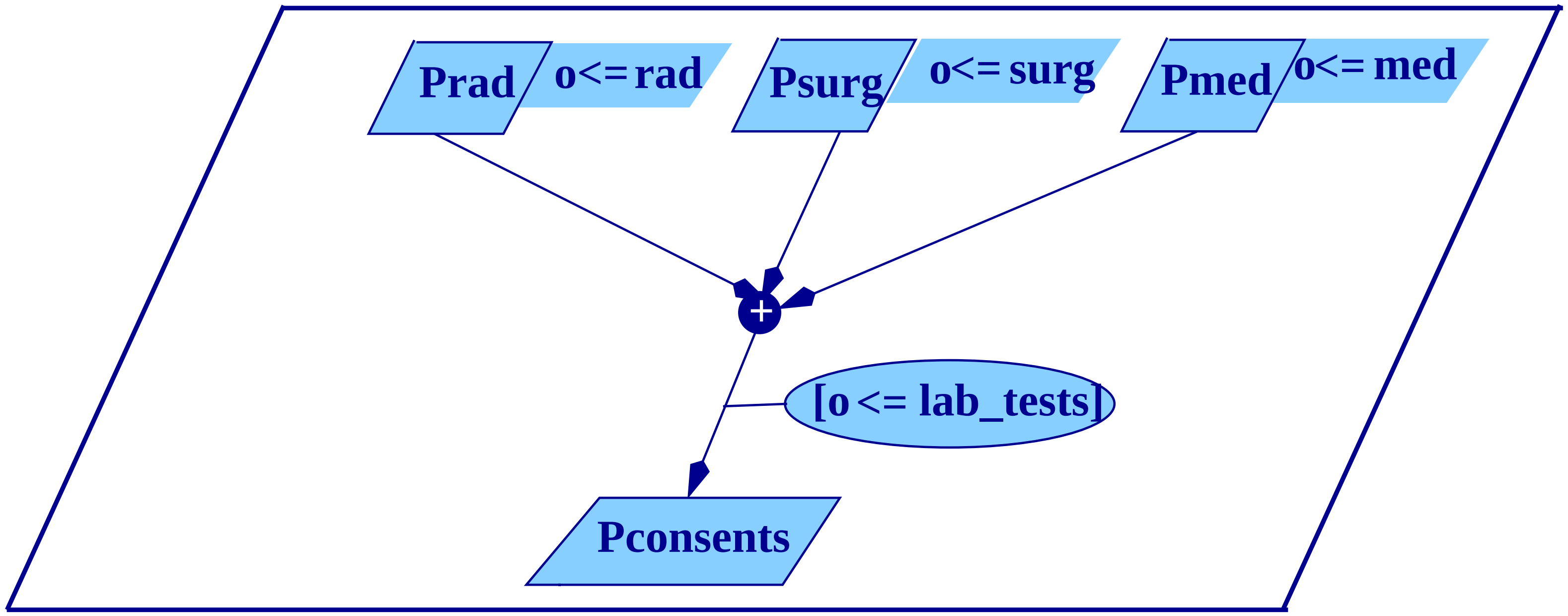
Hospital Domain

- H is composed of three depts: `Radiology`, `Surgery`, `Medicine`
- Each department can grant access to data under their domain
- H does not allow any access to `lab_tests` unless there is patient consent

An Example (1)

Hospital Domain

- H is composed of three depts: `Radiology`, `Surgery`, `Medicine`
- Each department can grant access to data under their domain
- H does not allow any access to `lab_tests` unless there is patient consent

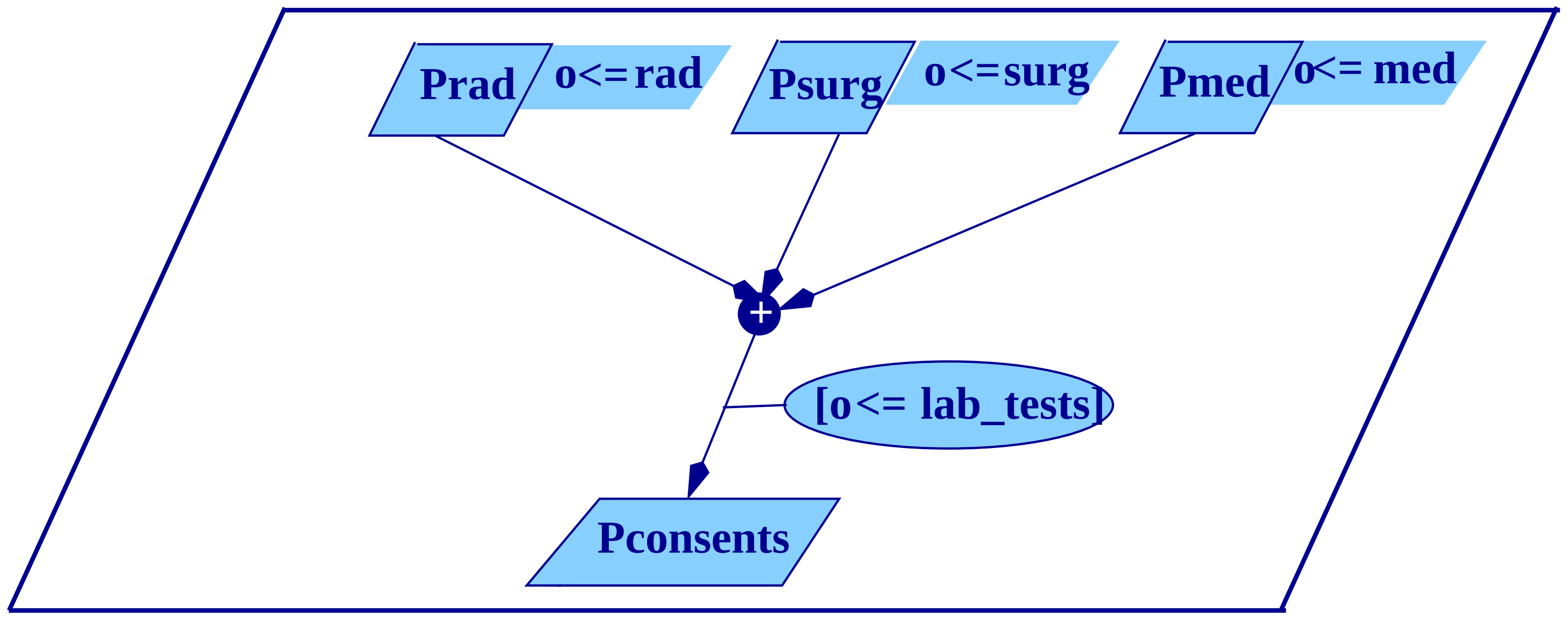


τPrad Psurg Pmed Pconsents. (o(Prad^[o<=rad]+Psurg^[o<=surg]+ Pmed^[o<=med],Pconsents,^[o<=lab_test]))

An Example (2)

Patient consent

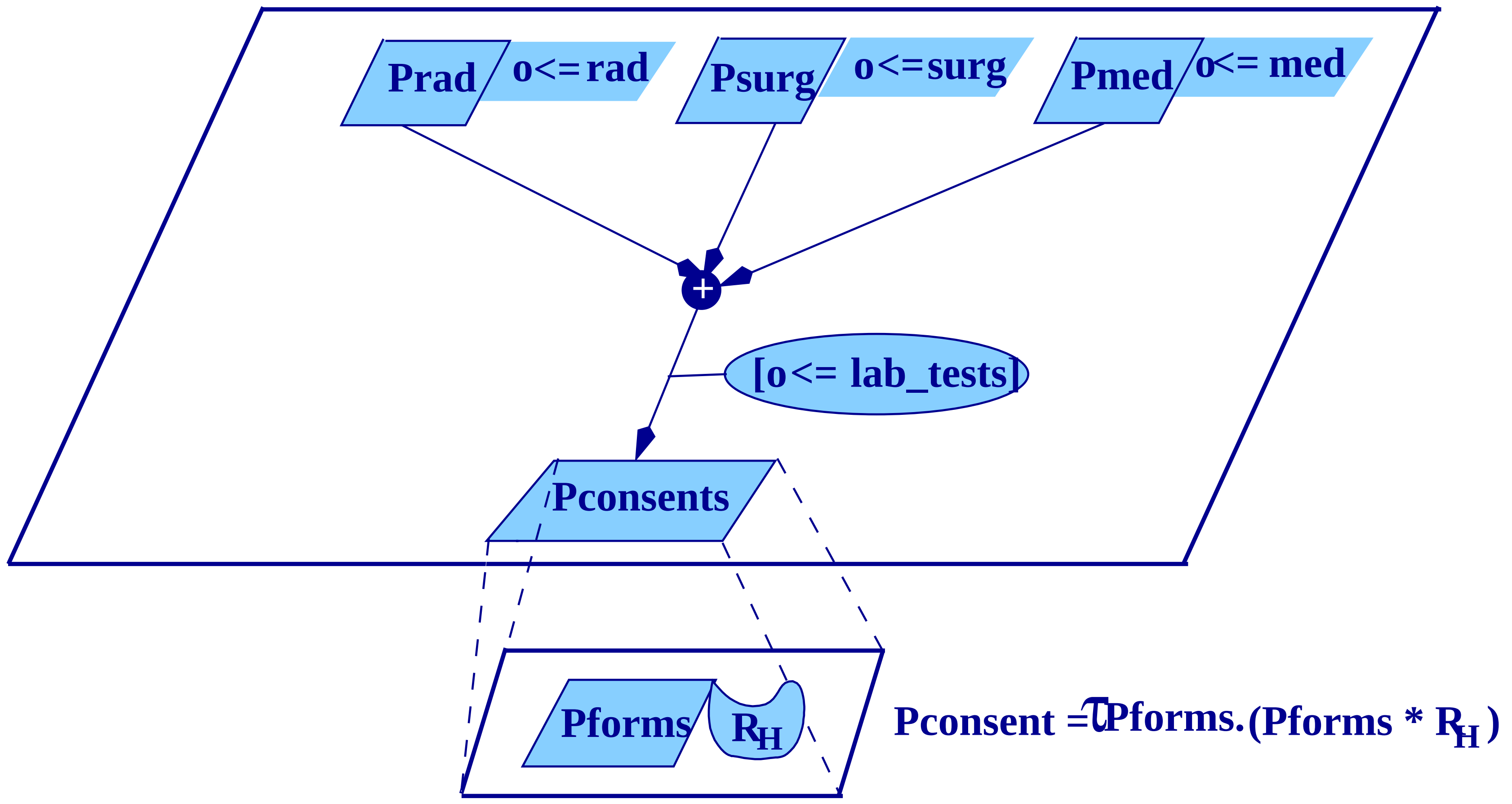
- Patient consent is collected by means of forms and propagated by R_H rules



An Example (2)

Patient consent

- Patient consent is collected by means of forms and propagated by R_H rules



Properties

The formal semantics on which the algebra is based allows us to reason about policy specifications and their properties.

Example

- *patient awareness and hospital authorization:* nobody can access `lab_tests` data if there are not both the patient consent and the hospital authorization for it.

Proof of the Property

Statement

Let $T = \tau(X\ Y).o(X, Y, \hat{}c)$. $\forall P_1, P_2, e$, and (s, o, a) satisfying c,
$(s, o, a) \in [\![T(P_1, P_2)]\!]_e \Leftrightarrow (s, o, a) \in [\![P_1]\!]_e$ and $(s, o, a) \in [\![P_2]\!]_e$

Proof

1. $[\![T(P_1, P_2)]\!]_e = ([\![P_1]\!]_e \setminus [\![P_1\hat{}c]\!]_e) \cup ([\![P_2]\!]_e \cap [\![P_1\hat{}c]\!]_e)$
2. (s, o, a) satisfies $c \Rightarrow (s, o, a) \notin [\![P_1]\!]_e \setminus [\![P_1\hat{}c]\!]_e$
3. $(s, o, a) \in [\![T(P_1, P_2)]\!]_e$ iff $(s, o, a) \in [\![P_2]\!]_e \cap [\![P_1\hat{}c]\!]_e$

   $$\equiv$$

   $(s, o, a) \in [\![T(P_1, P_2)]\!]_e$ iff $(s, o, a) \in [\![P_2]\!]_e$ and
   $(s, o, a) \in [\![P_1]\!]_e$

Translation into Logic Programs (1)

Policy expressions are enforced by translating them into logic programs

- *pe2lp* creates a distinct predicate symbol for each policy identifier and for each operator occurrence
- each operator occurrence is labeled with a distinct integer (*labeled policy expression*)
- *pe2lp* takes a labeled expression and an environment as input and returns a logic program
  - for each policy identifier *P* $\Rightarrow \mathtt{auth}_P$
  - for each operator $op_i \Rightarrow \mathtt{auth}_i$

Translation into Logic Programs (2)

E	*pe2lp*(E,e)
P	$\{\mathtt{auth}_P(s,o,a) \mid (s,o,a) \in e(P)\}$ if $e(P)$ is defined, $\emptyset$ otherwise
$F +_i G$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z), \mathtt{auth}_i(x,y,z) \leftarrow mainp_G(x,y,z)\}$ $\cup$ *pe2lp*$(F,e) \cup$ *pe2lp*(G,e)
$F \&_i G$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge mainp_G(x,y,z)\}$ $\cup$ *pe2lp*$(F,e) \cup$ *pe2lp*(G,e)
$F -_i G$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge \neg mainp_G(x,y,z)\}$ $\cup$ *pe2lp*$(F,e) \cup$ *pe2lp*(G,e)
$F\hat{}_iC$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge C\}$ $\cup$ *pe2lp*(F,e)
$o_i(F,G,R)$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge \neg mainp_R(x,y,z)$, $\mathtt{auth}_i(x,y,z) \leftarrow mainp_G(x,y,z) \wedge mainp_R(x,y,z)\}$ $\cup$ *pe2lp*$(F,e) \cup$ *pe2lp*$(G,e) \cup$ *pe2lp*(R,e)
$F *_i R$	$\{\mathtt{auth}_i(s,o,a) \leftarrow \mathtt{auth}_i(s_1,o_1,a_1) \wedge .. \wedge \mathtt{auth}_i(s_n,o_n,a_n) \mid$ $((s,o,a) \leftarrow (s_1,o_1,a_1) \wedge \ldots \wedge (s_n,o_n,a_n)) \in R\}$ $\cup \{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z)\} \cup$ *pe2lp*(F,e)
$\tau_i X.F(G)$	$\{\mathtt{auth}_X(x,y,z) \leftarrow mainp_G(x,y,z)\}$ $\cup$ *pe2lp*$(F,e) \cup$ *pe2lp*(G,e)

An Example of Translation

- Expression: $E = P + o(Q, S, R)$
- Mapping: e_0 maps P to $\{(\mathtt{s}', \mathtt{o}', \mathtt{a}'), (\mathtt{s}'', \mathtt{o}'', \mathtt{a}'')\}$
  Q, S, R are all undefined

Canonical Translation of E

$$\mathtt{auth}_P(\mathtt{s}', \mathtt{o}', \mathtt{a}')$$

$$\mathtt{auth}_P(\mathtt{s}'', \mathtt{o}'', \mathtt{a}'')$$

$$\mathtt{auth}_0(x, y, z) \leftarrow \mathtt{auth}_P(x, y, z)$$

$$\mathtt{auth}_0(x, y, z) \leftarrow \mathtt{auth}_1(x, y, z)$$

$$\mathtt{auth}_1(x, y, z) \leftarrow \mathtt{auth}_Q(x, y, z) \land \neg\mathtt{auth}_R(x, y, z)$$

$$\mathtt{auth}_1(x, y, z) \leftarrow \mathtt{auth}_S(x, y, z) \land \mathtt{auth}_R(x, y, z)$$

Access Control Enforcement

“Foreign” Policies

- $\mathtt{auth}_{\mathtt{P}}(s, o, a) \leftarrow \mathbf{in}((s, o, a), \mathtt{P} : \mathtt{grant}())$
- $\mathtt{auth}_{\mathtt{P}^-}(s, o, a) \leftarrow \mathbf{in}((s, o, a), \mathtt{P} : \mathtt{deny}())$

Complete/Partial Evaluation

- The evaluation is accomplished by applying standard techniques to the logic program

$$\left.\begin{array}{l} \mathbf{in}(\ldots, P : \ldots) \text{ s.t. } P \text{ is a black box} \\ \mathtt{auth}_P \text{ s.t. } P \text{ is a black box} \\ \text{base predicated undefined at materialization time} \end{array}\right\} \text{partial eval.}$$

Elementary Policy Specification

Closed Policy: P

Open Policy: $P_{all} - P_{open}$

Inheritance: $P * R_H$

$$\begin{aligned} R_H = \quad & \{(s,o,a) \leftarrow (s',o,a), s \leq s', (s,o,a) \leftarrow (s,o',a), o \leq o', \\ & (s,o,a) \leftarrow (s,o,a'), a \leq a'\} \end{aligned}$$

Denial Takes Precedence: $P^{+} - P^{-}$

Most Specific Takes Precedence:

$$\sum_{i \in H} (P_i^{+} * R_H - \sum_{j \leq i} P_j^{-} * R_H)$$

Strong and Weak Authorizations: $P^{+}_{weak} - P^{-}_{weak} + P^{+}_{strong} - P^{-}_{strong}$

Algebra for composing policies

Supports

- Heterogeneous policies: algebra constructs, or policy ids interpreted by wrappers
- Unknown policies: policy ids unbound in the environment
- Interference: controlled by the closure construct
- Expressiveness: operators of the algebra
- Different abstraction levels: component-based approach
- Formal semantics: exploited to reason about properties

Future work includes

- Administrative policies
- Incremental approaches to enforce changes to component policies
- Mechanized policy validation
- Assessment of different partial evaluation techniques

Access control in open systems

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
    - ∗ we may not know our users in advance

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

Policy maker & Co.

Some models (e.g., Policy Maker, Referee, Keynote) based on concept of trust management

- Use credentials that directly authorize actions $\Rightarrow$ combine authentication and access control
- noindent Assume requests and access restrictions associated with public keys (in contrast to principals)

Requests

$key_1, key_2, \ldots, key_n$ REQUESTS *actionstring*

- $key_1, key_2, \ldots, key_n$ public keys requesting action
- *actionstring* application-dependent description of the action

Policy maker – 2

Access restrictions (security policies) stated through

Assertions

source ASSERT *authoritystruct* WHERE *filter*

- *source* grantor of the authorization (either the local policy or the public key of a third party)
- *authoritystruct* public key to which privilege is granted
- *filter* condition that *actionstring* must satisfy for the assertion apply

authoritystruct and *filter* can be specified regular expression checkers, ad-hoc extensions, or programming languages

Policy maker – 3

Example of assertions

- `policy ASSERTS pgp:“0xf0012203a4b51677d8090aabb3cdd9e2f” WHERE PREDICATE=regexp:“(From:Alice) && (Organization: BobLabs)”`

  the specified (Alice’s) key can sign message originating from Alice within BobLabs

- `policy ASSERTS pgp:“0xf0012203a4b51677d8090aabb3cdd9e2f” WHERE PREDICATE=regexp:(Organization: BobLabs)”`

  the specified key (belonging to Security_CA) can create certificate in the name of the company

Policy maker – 4

Note The trusts in principal’s key is embedded in the policy

- Good: does not require external authorities
- Bad: requires public key management must be dealt with locally

Drawbacks:

- specification of authorizations for public keys not always meaningful and difficult to manage
- authorizations not always easy to understand
- specifications in programming languages not easy to control

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 based systems

The way access control is enforced may change

- What users can do depend on assertions 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)

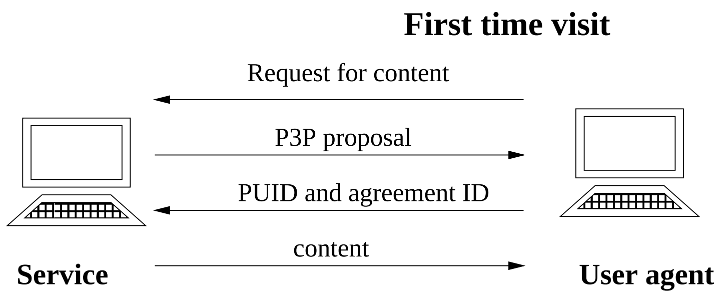
Can introduce some form of negotiation.

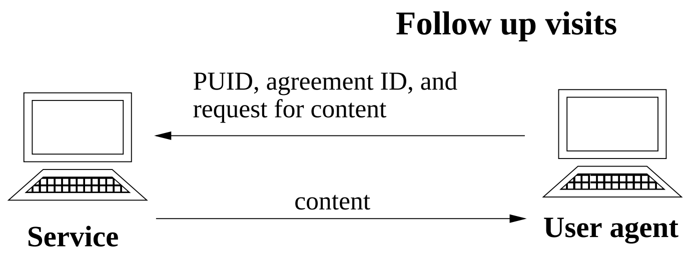
Platform for Privacy Preference (P3P)

Proposal of the W3C for establishing privacy preference

Goal provide a protocol to express privacy practices in a standard format that can be retrieved automatically and interpreted easily by user agents

- Policies expressed as XML documents.
- Each URL may have its own policy (at most one)





Platform for Privacy Preferences (P3P) – 2

P3P proposes a “dictionary” with which Web sites can communicate the privacy practice used for the data

It distinguishes

- categories of data (e.g., contact information, identifiers, demographical data, navigational data, ...)
- intented use (e.g., to complete the activity, site personalization, research, contact the user, other purposes, ...)
- recipient (e.g., the site and its partners, organizations that follow the same privacy practices, others)

Has encountered objections (users sendomly would change the defaults values....)

An approach to credential-based access control

Credential-based access control

- Parties can present digital certificates stating the identities or properties of the parties

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

Basic concepts

Network composed of different parties that interact with each other

- to offer services (servers)
- to require services (clients)

Each server has an associated set of services it provides.

Each party has an associated portfolio of properties that the party can submit in order to obtain (or offer) services.

- declarations: information uttered by the party and not certified by any authority (e.g., identity, address, hobbies)
- credentials: digital certificates (c, K)
  - c: signed content
  - K: public digital signature verification key

Credentials

We assume a semi-structured organization of credentials

Credential term: expression of the form credential_name(attribute_list)

- credential_name: name of the credential
- attribute_list: list of elements of the form “attribute_name=value_term”

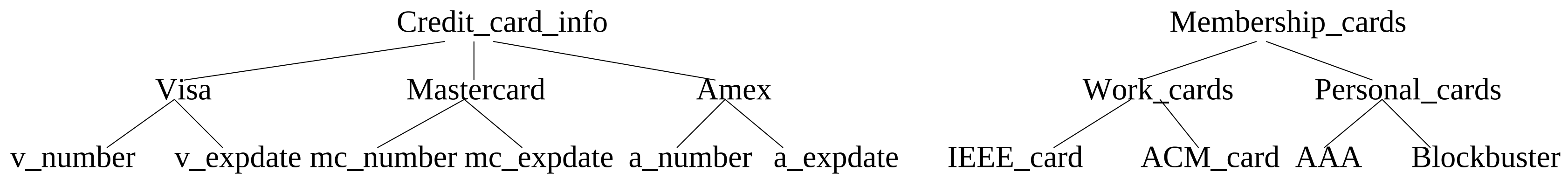
Example

- driver-license(name=“`John Doe`”)
- driver-license(name=X)
- enr_certificate(issuer=I,student_id=“`John Doe`”,university=U).

Portfolio abstraction

Partial order $\sqsubseteq_P$ defined on portfolio by collecting

- declarations into classes representing named sets of properties (e.g., “demographic_data” or “personal_data”)
- credentials into abstractions



$\Longrightarrow$ can refer to a set of declarations/credentials with a single name.

Services

Define the functionalities offered by a server.

Each service characterized by a name and a set of parameters (values).

Service terms have the form service_name(attribute_list)

- attribute_list: list of pairs "attribute_name=value_term".

  E.g.
  $\texttt{print}(\text{pub\_type}=\texttt{proceedings}, \text{conf}=\texttt{CCS}, \text{year}=\texttt{2000})$

Services may have associated facets that capture additional, or alternative (*polymorphic behavior*), functionalities

E.g.,

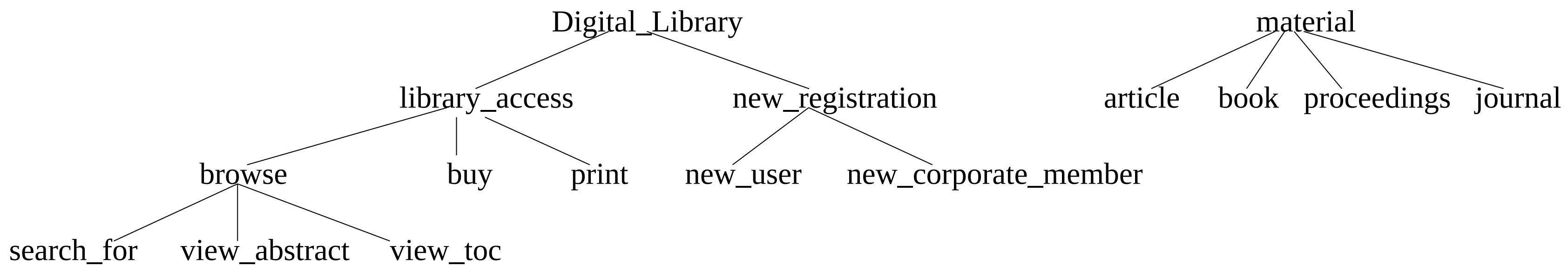
- service: flight-reservation
- facets: seat_assignments, lunch_choices, . . .

Service abstraction

Services can be grouped into classes $\Longrightarrow$ partial order $\sqsubseteq_{SN}$

- refer to groups of services with a single name
- model gradual access to services

Abstractions can be defined on values $\Longrightarrow$ partial order $\sqsubseteq_{V}$



Given *ground* service terms $s_1(L_1)$ and $s_2(L_2)$, $s_1(L_1) \sqsubseteq_{ST} s_2(L_2)$ iff

1) $s_1 \sqsubseteq_{SN} s_2$, and 2) $\forall$ "$a = V$" in L_2 : $\exists$ "$a = V'$" in L_1 s.t. $V' \sqsubseteq_{V} V$

Example

$\text{print}(\text{year}{=}1990, \text{pub\_type}{=}\texttt{proceedings}) \sqsubseteq_{ST}$
$\text{print}(\text{pub\_type}{=}\texttt{proceedings}) \sqsubseteq_{ST} \text{print}(\text{pub\_type}{=}\texttt{material})$.

Requirement specification language – 1

Logic language. Basic predicates:

- credential: credential(c, K) true iff the current state contains credential c verifiable with key K.
- declaration: declaration(d) true iff the current state contains declaration d, where d is of the form “att_name=att_value”.
- cert_authority: cert_authority(CA, K_{CA}) states that the party trusts certificates signed by authority CA whose public key is K_{CA}.
- A set of non predefined state inquiry predicates that evaluate information stored at the site
  - Persistent state (e.g., user profiles)
  - Negotiation state (e.g., current service)
- A set of non predefined abbreviation predicates
- A set of standard built-in math predicates, including $=, \neq, <$

Requirement specification language – 2

Three classes of rules:

- abbreviations: define “macros” that can be used in other rules
- service accessibility: define restrictions that the other party (client) has to satisfy to be granted access to services
- release: define restrictions on credentials and information that can be released to the other party

Abbreviation rules

Define “macros” that can be used as a shorthand for disjunctions or conjunctions of conditions (those in the body of abbreviation rules).

$$p(\vec{x}) \leftarrow q_1(\vec{x_1}), \ldots, q_n(\vec{x_n})$$

- p is an abbreviation predicate
- $q_i, i = 1, \ldots, n$, are basic predicates.

Example

1. $\textsf{info}(\mathrm{hobby}{=}X) \leftarrow \textsf{declaration}(\mathrm{name}{=}Y), \textsf{usr\_profile}(Y, \mathrm{hobby}{=}X)$
2. $\textsf{info}(\mathrm{hobby}{=}X) \leftarrow \textsf{declaration}(\mathrm{preferences.hobby}{=}X)$
3. $\textsf{principal}(P, K) \leftarrow \textsf{cert\_authority}(P, K)$
4. $\textsf{principal}(P, K) \leftarrow \textsf{principal}(I, K'),$
   $\textsf{credential}(\textsf{belongs\_to}(\mathrm{issuer}{=}I, \mathrm{principal}{=}P, \mathrm{key}{=}K), K')$

Service accessibility rules

- prerequisite rules state credentials and declarations that client must submit to have their access requests considered
- requisite rules state credentials and declarations that client must submit to have their access request granted
- facet rules state credentials and declarations that client must submit to enable specific features of a service

Distinction between prerequisites and requisites allow requests of preliminary credentials and declarations

Service requisite rules

State credentials and declarations that the client must submit to be granted the service.

$$\text{service\_reqs}(s(L)) \leftarrow q_1(\vec{x_1}), \ldots, q_n(\vec{x_n})$$

- $s(L)$ is a service term
- $q_i, i = 1, \ldots, n$ are basic predicates.

Example

1. $\text{service\_reqs}(\text{print}()) \leftarrow \text{declaration}(\mathrm{copyright}{=}\text{“}\texttt{accept}\text{”})$.
2. $\text{service\_reqs}(\text{print}(\mathrm{journal}{=}J, \mathrm{year}{=}X)) \leftarrow \text{customer\_affiliation}(A)$,
   $\text{subscription}(\mathrm{subscriber}{=}A, \mathrm{year}{=}X, \mathrm{journal}{=}J)$,
   $\text{curr\_year}(Y), X < Y$.

Requisite propagation

A service is subject to all prerequisite/requisite/facet requirements specified at more general levels

$\Longrightarrow$ prerequisite/requisite/facet requirements must propagate from more generic to more specific service terms.

Example

A request $\texttt{print}(\text{journal}=\texttt{"CACM"}, \text{year}=\texttt{"1999"})$ will be subject to both

1. $\textsf{service\_reqs}(\textsf{print}()) \leftarrow \textsf{declaration}(\text{copyright}=\text{“}\texttt{accept}\text{”})$.
2. $\textsf{service\_reqs}(\textsf{print}(\text{journal}=J, \text{year}=X)) \leftarrow \textsf{customer\_affiliation}(A)$,
   $\textsf{subscription}(\text{subscriber}=A, \text{year}=X, \text{journal}=J)$,
   $\textsf{curr\_year}(Y), X < Y$.

Requisite propagation rules

- APol be a set of requisite rules specified at a server
- $s(L)$ be a service term

The requisite propagation rules $\mathsf{Inh}(s(L), \mathsf{APol})$ for a $s(L)$ and APol:

$$\mathsf{service\_reqs*}(s(L)) \leftarrow$$
$$\mathsf{service\_reqs}(s_1(L_1)), \ldots, \mathsf{service\_reqs}(s_n(L_n))$$

- $s_1(L_1) \ldots, s_n(L_n)$ are all and only the ground instances of the service terms that occur in APol, such that the variables are bounded to leaves of the value hierarchy and $s(L) \sqsubseteq_{\mathsf{ST}} s_i(L_i)$.

Requisite satisfaction

- APol: server's policy
- Σ: server state
- $s(L)$: service

A set of credentials/declarations Info satisfies the requisites for service $s(L)$ w.r.t. server state Σ

iff

$$\mathsf{APol} \cup \mathsf{Inh}(s(L), \mathsf{APol}) \cup \Sigma \cup \mathsf{Info} \models \mathsf{service\_reqs}{*}(s(L))$$

Server's policy filtering mechanism

Given a service request $s(L)$

1. Select all the rules and abbreviations that apply to a given request
2. Simplify the rules by evaluating state enquiry predicates and built-in math predicates when possible
   - atoms that are satisfied are simplified away.
   - atoms that are not satisfied fail and make a rule not applicable.
3. Rename atoms in partially evaluated rules
4. Send the resulting filtered and renamed policy to the client

Server's policy filtering mechanism – Example

Policy

1. service_reqs(print()) $\leftarrow$ declaration(copyright=“accept”).
2. service_reqs(print(journal=J, year=X)) $\leftarrow$ customer_affiliation(A),
   subscription(subscriber=A, year=X, journal=J),
   curr_year(Y), $X < Y$.

Request

print(journal="CACM", year="1999")

Relevant rules

1. service_reqs$*$(print(journal="CACM", year="1999")) $\leftarrow$
   service_reqs(print(journal="CACM", year="1999")) ,
   service_reqs(print()) .
2. service_reqs(print(journal="CACM", year="1999")) $\leftarrow$
   customer_affiliation(A), subscription(subscriber=A, year=
   "1999", journal="CACM"), current_year(Y), 1999 $< Y$.
3. service_reqs(print()) $\leftarrow$ declaration(copyright="accept") .

Server's policy filtering mechanism – Example ('ed)

Assume Σ contains the facts:

- customer_affiliation("ACME")
- subscription(subscriber="ACME", year="1999", journal="CACM")
- current_year(2000).

Then

service_reqs(print(journal="CACM", year="1999")) $\leftarrow$
  customer_affiliation(A), subscription(subscriber=A, year=
  "1999", journal="CACM"), current_year(Y), 1999 $< Y$.

evaluates true and the policy is simplified as

service_reqs$*$(print(journal="CACM", year="1999")) $\leftarrow$
  service_reqs(print()) .

service_reqs(print()) $\leftarrow$ declaration(copyright="accept") .

Service prerequisite rules

Define preconditions that the client must satisfy to have its request considered.

$$\text{service\_prereqs}(s(L)) \leftarrow q_1(\vec{x_1}), \ldots, q_n(\vec{x_n}) \mid p_1(\vec{y_1}), \ldots, p_1(\vec{y_m})$$

- $s(L)$ is a service term
- $q_i, i = 1, \ldots, n$, are basic predicates
- $p_j, j = 1, \ldots, m$, are either state predicates or math built-in.

Example

1. $\text{service\_prereqs}(\text{library\_access}()) \leftarrow \text{declaration}(\text{login}{=}X, \text{passwd}{=}Y)$
   $\mid \text{usr\_profile}(\text{login}{=}X, \text{passwd}{=}Y).$
2. $\text{service\_prereqs}(\text{library\_access}()) \leftarrow \text{principal}(I, K_I),$
   $\text{credential}(\text{affiliation}(\text{issuer}{=}I, \text{user}{=}U, \text{user\_key}{=}K_U), K_I),$
   $\mid \text{registrations}(\text{subscriber}{=}I).$

Facet requisite rules

State credentials and declarations necessary to enjoy specific facets.

$$\text{facet\_reqs}(s(L), f) \leftarrow q_1(\vec{x_1}), \ldots, q_n(\vec{x_n})$$

- $s(L)$ is a service term
- f is the name of a facet associated with s
- $q_i, i = 1, \ldots, n$ are basic predicates.

Example

$\text{facet\_reqs}(\text{buy}(\text{material}{=}\texttt{proceedings}, \text{conf}{=}C, \text{ass}{=}A), \text{discount}) \leftarrow$
  $\text{credential}(\text{att\_certificate}(\text{issuer}{=}O, \text{attendant}{=}U, \text{conference}{=}C)K_O),$
  $\text{current\_customer}(U),$
  $\text{accredited\_organizer}(\text{company}{=}O, \text{association}{=}A),$
  $\text{principal}(O, K_O).$

Release rules

Regulate disclosure of declarations and credentials in the party's portfolio

$$\text{release\_reqs}(o) \leftarrow q_1(\vec{x_1}), \ldots, q_n(\vec{x_n})$$

- o is either a credential term $c(L)$ or a declaration d
- q_i, $i = 1, \ldots, n$, are basic predicates

Example

1. release_reqs$(\mathrm{credit_card_info}) \leftarrow$ current_service(buy()),
   declaration$(\mathrm{no_disclosure}{=}$`"accept"`) .
2. release_reqs(membership_card$(\mathrm{issuer}{=}X)) \leftarrow$
   credential(certified_server$(\mathrm{issuer}{=}X, \mathrm{server}{=}M), K)$,
   current_server(M), principal(X, K) .

Release requirements propagation

An object is subject to all release requirements specified at more general levels

$\Longrightarrow$ release requirements must propagate from more generic to more specific terms.

Let

- RPol be a set of release rules specified at a party
- o be an object (credential term or declaration)

The *release propagation* rules $\mathsf{Inh}(o, \mathsf{RPol})$ for o and RPol:

$$\mathsf{releasable*}(o) \leftarrow \mathsf{release\_reqs}(o_1), \ldots, \mathsf{release\_reqs}(o_n)$$

- $o_1, \ldots, o_n$ are all and only the ground instances of the terms that occur in RPol, such that the variables are bounded to leaves of the value hierarchy and $o \sqsubseteq_{\mathsf{P}} o_i$, for $i = 1, \ldots, n$.

Release requirements satisfaction

- RPol: party's release policy
- Σ: party's state
- Info: a set of credentials/declarations Info received

A credential/declaration o is releasable w.r.t. Σ and Info

iff

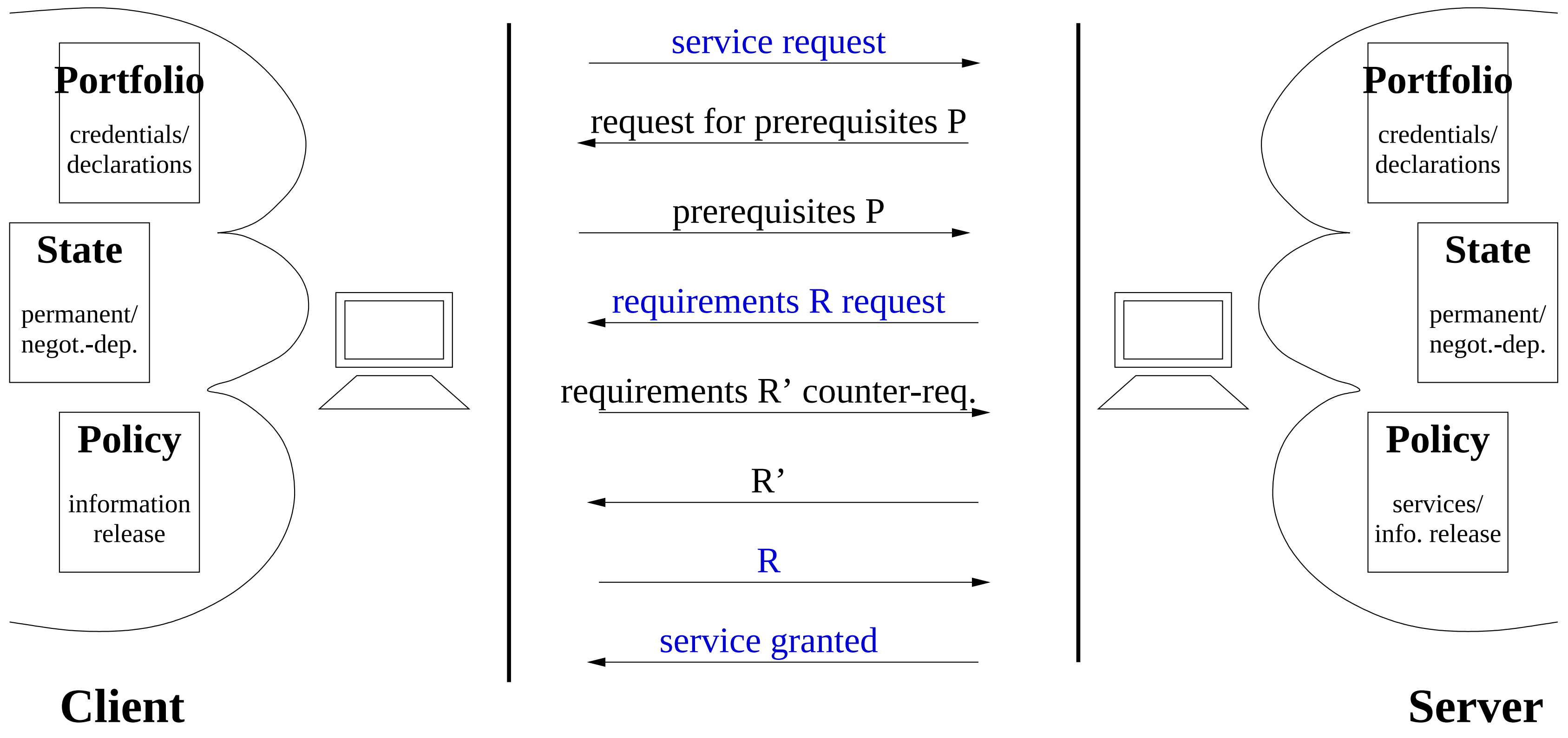
$$\mathsf{RPol} \cup \mathsf{Inh}(o, \mathsf{RPol}) \cup \Sigma \cup \mathsf{Info} \models \mathsf{releasable}{*}(o).$$

Client's policy evaluation

Upon reception of policy $\mathsf{RenFilter}(\mathsf{APol}, s(L), \Sigma, \rho)$ to be satisfied

1. search the portfolio for a set of credentials/declarations Info such that $\mathsf{RenFilter}(\mathsf{APol}, s(L), \Sigma, \rho) \cup \mathsf{Info} \models \mathsf{service\_reqs}{*}(s(L))$
2. Use release rules to prune the alternative *sets* $CS_1, \ldots, CS_n$ of credentials obtained
   - may send CS_i to the server only if $\mathsf{releasable}{*}(o)$ holds for all $o \in CS_i$.
   - possibly send counter-requests to the server

Client-Server Interplay



Correctness and complexity

Correctness: The filtering and renaming process preserves the original policy

Theo For all APol, $s(L)$, Σ, renaming $\mathsf{RenFilter}(\mathsf{APol}, s(L), \Sigma, \rho)$, and sets Info of credential and declaration atoms,

$$\begin{aligned}&\mathsf{APol} \cup \mathsf{Inh}(s(L), \mathsf{APol}) \cup \Sigma \cup \mathsf{Info} \models \mathsf{service\_reqs}{*}(s(L)) \iff \\ &\quad \mathsf{RenFilter}(\mathsf{APol}, s(L), \Sigma, \rho) \cup \mathsf{Info} \models \mathsf{service\_reqs}{*}(s(L))\end{aligned}$$

It ensures

- correctness of access enforcement
- ability to grant the service upon reception of credentials/declarations from the client

Complexity: $O(n \log n)$, where $n = |\mathsf{APol}| + |\Sigma|$.

Credential-based access control

Still considerable work to be done on credential-based, including

- Strategies for selecting credentials/declarations to be released
- Evaluate privacy of server’s policy
- Extend filtering/renaming features (e.g., support actions)
- Extend negotiation model and dialog
- Determining retrieval of digital certificates not stored remotely

Controlling access in open systems

Logic-based languages not always accepted (may be too complex for end-users)

Need to find a trade-off between

- complexity, and
- expressiveness
  - support for profile-based and assertion-based authorizations
  - support different kinds of rules (e.g., permissions and denials)
  - support for dynamic conditions and “interactive” access control

An simple but expressive access control language

Problem of protecting data published on the Web.

- producers: organizations that collect data, process them, and prepare them for distribution;
- distributors: data archives (or information brokers) that collect data from various producers and makes them available on the Web.
- user community: wants to access data

Problem: Develop an access control system that producers and distributors can use to state and enforce restrictions on the data they make available to different user populations on the Web.

Characterization of subjects

- User: human entities that can connect to the system and make requests. Each user has associated an identifier (usually the user's login), with which the user is referred to in the system.
- Purpose: reason for which data are being requested and will be used (e.g., Commercial, Teaching, Research consultancy)
- Project: named activity registered at the server, for which different users can be subscribed, and which may have one or more purposes (e.g., `Nesstar` and `Faster`).

Subjects – Examples

Each subject making a request to the Faster server with a triple

$\langle$ user, project, purpose $\rangle$

meaning user is making the access request for a given project and/or a given purpose.

Some elements within the triple may remain unspecified.

Examples

- $\langle$tom.smith,Faster,research$\rangle$

  user tom.smith for research purposes within the Faster project.

- $\langle$john.doe,_,commercial$\rangle$

  user john.doe for commercial purposes.

- $\langle$_,_,_$\rangle$ an anonymous user with undeclared project and purposes

Objects

- Datasets are the objects containing information whose access is being protected. With respect to statistical data archives they can be tables storing micro or macrodata.
- Metadata represent information associated with datasets. Metadata are not part of the dataset content. They provide additional contextual information explaining, for example, to which study a dataset is referred, how it has been obtained, by whom, and so on.

Actions

Kind of access (operation) that the requesting subject wishes to perform on the object.

Specific actions may vary depending on functionalities provided on specific kinds of datasets.

We can distinguish three main classes of actions:

- Browse: to visualize and query metadata associated with datasets. With the browse facility, users can walk through the metadata in order to choose the actual dataset they are interested in.
- Analyse-on-line: to query datasets. On-Line analysis includes a set of pre-defined operations that perform on-line calculations on selected data. Available operations may vary depending on kind of dataset under consideration.
- Download: to download data from the server. It allows users to save whole datasets on their local machine to perform off-line analysis.

Access requests

Access requests are triples of the form

$$\langle \textit{user,project,purpose} \rangle, \textit{action}, \textit{object}$$

Examples

- ⟨tom.smith,Faster,research⟩, download, dataset1

  user tom.smith requires to download dataset1 for research purposes within the Faster project.

- ⟨john.doe,Nesstar,_⟩, download, dataset1

  user john.doe requires to download dataset1 for use within the Nesstar project.

- ⟨_,_,_⟩, browse, meta_dataset5

  an anonymous user with undeclared project and purposes requires to browse metadata meta_dataset5

Subject information

The ACU server recognizes only users and projects registered at the server.

Each user and project are characterized by:

- identifier that allows the server to refer to the user (project, resp.).
- profile defines the name and value of some properties that characterize the user (e.g., name, address, occupation) or the project (e.g., title, abstract, or sponsor).

Intuitively, profiles are at users and projects what metadata are at datasets.

We view profiles as semi-structured documents (XML or RDF like).

Examples of users and projects profiles

User profile	**Project profile**
Name	ID
Login	Title
Title/Position/Job	Abstract
Address	Objectives
Email	Period
Telephone	Purposes
Validation information	Sponsor
Groups	Leaders
Purposes	Participants
Agreement	Responsible institution
On line agreement	
Registration Date	

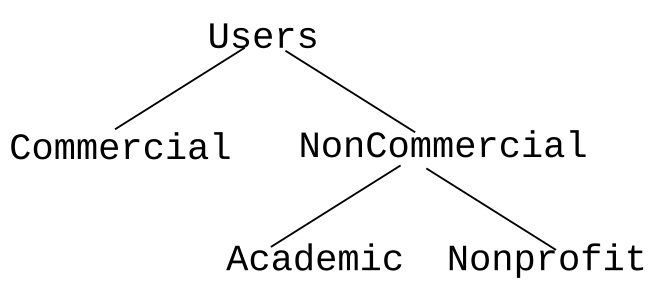
Users, projects, and purposes abstractions

Abstractions can be defined within the domains of users, projects, as well as purposes.

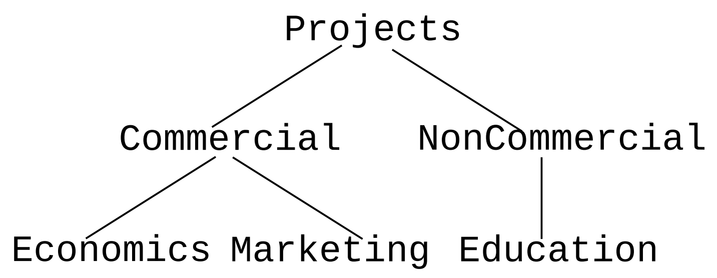
Abstractions allow to group together users (projects, and purposes, resp.) with common characteristics and to refer to the whole group with a name.

Groups need not be disjoint and can be nested.

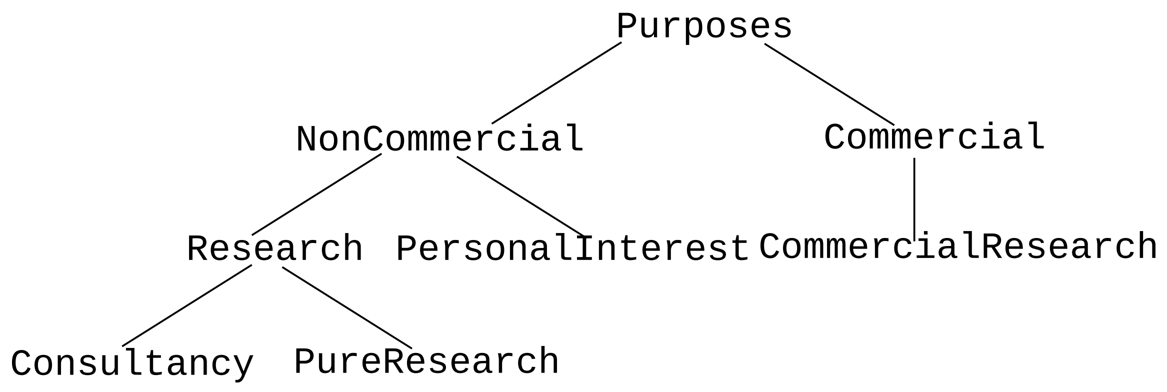
Subjects abstractions – example



User Hierarchy



Project Hierarchy



Purpose Hierarchy

Objects organization

- Datasets. For the time being, we consider access to whole datasets only (i.e., an access to a dataset is either allowed or denied).
- Metadata. They can be in the form of textual or semistructured documents (XML/RDF).

  Metadata are files that can be associated with each dataset.

A metadata document can then be referred to either by its identifier or, via function META, by the identifier of the dataset with which it is associated.

Bijective function META() makes the association between a dataset and its metadata.

E.g., META(*dataset1*) is the metadata file associated with dataset *dataset1*

Metadata querying

Properties within a metadata document are referred by means of path expressions, expressed, for instance, with the XPath language.

- Path expression: sequence of element names or predefined functions separated by character / (slash): $l_1/l_2/\ldots/l_n$.

  A path expression $l_1/l_2/\ldots/l_n$ on a document tree represents all the attributes or elements named l_n that can be reached by descending the document tree along the sequence of nodes named $l_1, l_2, \ldots, l_{n-1}$.

Example

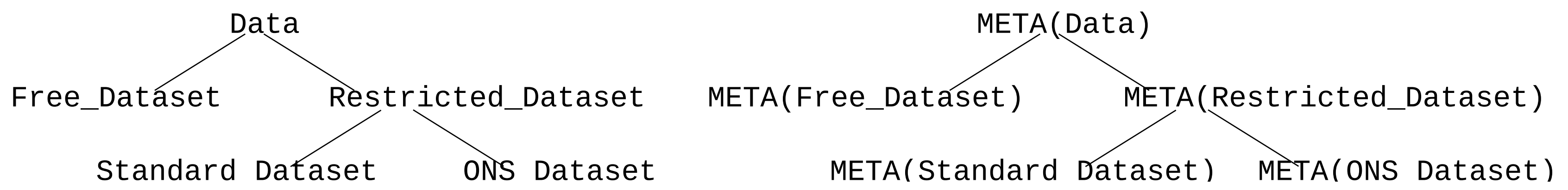
- META(*data1)/codeBook/stdyDscr/stdyInfo/subject*

  subject elements describing the data collection's intellectual content that are children of *stdyInfo* elements, that are children of the *stdyDscr* elements, and so on, in document META(*data1*).

Datasets and metadata abstractions

Datasets also can be organized in a hierarchical structure, defining sets of datasets that can be collectively referred together with a given name.

Metadata are associated only with specific datasets, not with abstractions on them, and no hierarchy is explicitly defined on metadata. However, the abstraction hierarchy defined on the data reflects in an abstraction hierarchy on the corresponding metadata.



ACU language desiderata

The ACU language should ...

- support access restrictions based on abstractions
- support access restrictions based on conditions on meta-data or on local user profile conditions.
- support access restrictions related to signed agreements, and other fulfillments to be accomplished via manual procedures.
- support both regulation in the form of authorizations and restrictions.
- have a declarative form.
- be simple and expressive.
- be easy to use to nonspecialists in the field.

ACU rules

Access control rules are usually based on triples stating which *subjects* can exercise which *action* on which *object*.

Two kind of access control rules:

- Authorizations specify permissions for subjects to access a (set of) datasets/metadata in a given mode.
- Restrictions specify conditions that must be satisfied for a given access to be granted.

Restrictions

Specify requirements that must all be satisfied for an access to be granted.

$$\langle \textit{subjects} \rangle \ \textsc{can} \ \langle \textit{actions} \rangle \ \langle \textit{objects} \rangle \ \textsc{only if} \ \langle \textit{conditions} \rangle$$

where:

- subjects identifies the set of subjects to which the restriction refers
- actions is the action (or class of actions) to which the restriction refers
- objects identifies the set of objects (either datasets or metadata) to which the restriction refers
- conditions is a boolean expression of conditions that every request to which the restriction applies must satisfy.

Lack to satisfy any of the restrictions that apply to a given request implies the request will be denied.

Authorizations

Specify permissions for accesses.

$$\langle \textit{subjects} \rangle \ \ \textsc{can} \ \langle \textit{actions} \rangle \ \langle \textit{objects} \rangle \ [\ \textsc{if} \ \langle \textit{conditions} \rangle]$$

where subjects, actions, and objects have the same syntax and semantics as in restrictions, and ⟨conditions⟩ is a boolean expression of conditions whose satisfaction authorizes the access.

An access is granted if there is satisfaction of at least one of the permissions that apply to the given request and no restriction is violated.

Subjects

Specified by means of a

subject expression: a boolean formula of terms that evaluate conditions on the user, project, and purpose of the request. Conditions can evaluate the value of the elements, their membership in defined groups/categorizations, or properties in their profiles.

Expressions can make use of the following keywords

- **user** indicates the identifier of the person making the request
- **purpose** indicates the purpose declared by the user for the request
- **project** indicates the project declared by the user for the request

Subjects – Example

- **user**/*citizenship=EC* AND (**project**/*sponsor=EC* OR **purpose** IN *research*)

  requests made by users who are European citizens and intend to use the data for research purposes or within an EC funded project

- **user** IN *NonCommercial-users* AND **purpose** IN *research*

  requests made by users belonging to group *NonCommercial-users* who intend to use the data for research purposes

- **user** IN *NonCommercial-users* AND **purpose** IN *research* AND **project**/*sponsor=EC*

  requests made by users belonging to group *NonCommercial-users* who intend to use the data for research purposes within an EC funded project

Simplified (indexed) subject expressions

For authorizations applicable to all users within a given group or that request access for a given project or purpose (or category thereof), the group, project, and/or purpose element can be explicitly factorized out, bringing a more readable and indexable subject expression.

A subject expression of the form

user IN group-id AND **project** IN project-id AND **purpose** IN purpose-id AND subject-expression

can be turned into an indexable expression of the form

group-id OF project-id PROJECTS FOR purpose-id PURPOSES WITH subject-expression

where the clauses " OF project-id PROJECTS", " FOR purpose-id PURPOSES", and "WITH subject expression" are optional and can be omitted.

Simplified (indexed) subject expressions – examples

- **user** IN *NonCommercial-users* AND **purpose** IN *research*

  $\Longrightarrow$ NonCommercial-users FOR research PURPOSES

- **user** IN *NonCommercial-users* AND **purpose** IN *research* AND **project**/*sponsor=EC*

  $\Longrightarrow$ NonCommercial-users FOR research PURPOSES WITH **project**/*sponsor=EC*

- **user** IN *NonCommercial-users* AND **project** IN *EC-sponsored*

  $\Longrightarrow$ NonCommercial-users OF EC-sponsored PROJECTS

Objects

object-id [WITH *conditional-object-expression*]   where:

- object-id is
  - the identifier of a dataset (or group of datasets)
  - the identifier of a metadata document (or group of them) with possibly associated an Xpath expression.
- conditional-object-expression boolean formula of conditions that can evaluate membership of the object in categories, values of properties on metadata and so on. Can make use of the following keywords:
  - **dataset** indicates the identifier of the dataset to which access is requested
  - **metadata** indicates the identifier of the metadata document to which access is requested or associated with the dataset to which access is being requested.

Objects – Examples

Some examples of objects are as follows:

- *Free_Datasets* WITH **metadata**/*producer=ACME*

  all datasets in the *Free_Datasets* class produced by ACME (where the produced in specified as property in the associated metadata)

- META(*Restricted_Datasets*)//*question_text*

  the text of the question_text element of the metadata documents associated with datasets in the *Restricted_Datasets* set.

Conditions

Element conditions defines conditions that must be satisfied for the request not to be rejected (as in the case of restrictions) or for the access to be granted (as in the case of authorizations).

Two kinds of conditions

- Static conditions evaluate membership of subjects and objects into classes or properties in their profiles and associated metadata.
- Dynamic conditions are conditions that can be brought to satisfactions at run-time processing of the request. They include: *Agreement acceptance*, *Payment*, *Registration*, *Form filling*.

For each of them we assume the existence of a procedure that perform the control and possibly trigger the necessary actions.

Examples of predicates

- **agreement**(*id*,a): checks if user *id* has accepted agreement a, and if not presents the user with the agreement. It returns true if the agreement has been accepted.
- **register**(*id*): checks if user/project *id* is registered, and if not starts the registration procedure. It returns true if the *id* was registered or the registration has been successfully completed.
- **fill_in_form**(*id*,*form*): checks if user *id* has filled in form *form*, and if not presents it to the user. It returns true if the user has filled the form.
- **payment**(id_1,id_2): checks if user id_1 has paid to access object id_2, and if not starts the payment procedure for the user. It returns true if the user had paid or the payment procedure completes successfully.

Keywords, predicates, and reserved identifiers

Keywords	CAN , WITH , IF , ONLY IF , IN , AND , OR , NOT , FOR , META, PURPOSES, PROJECTS	
Predicates	**agreement**(*id*,a)	checks if user *id* has accepted agreement a, and if not presents the user with the agreement. It returns true if the agreement has been accepted.
	register(*id*)	check if user/project *id* is registered, and if not starts the registration procedure for it. It returns true if the user/project was registered or the registration has been successfully completed.
	fill_in_form(*id*,*form*)	checks if user *id* has filled in form *form*, and if not presents it to the user. It returns true if the user has filled the form.
	payment(id_1,id_2)	checks if user id_1 has paid to access object id_2, and if not starts the payment procedure for the user. It returns true if the user had paid or the payment procedure completes successfully.
Reserved identifiers	**user**	bounded to the identity (if defined) of the user making a request
	project	bounded to the project (if defined) specified by the user making a request
	purpose	bounded to the purpose (if defined) specified by the user making a request
	dataset	bounded to the identifier of the dataset to which access is requested
	metadata	bounded to the identifier of the metadata document to which access is requested or associated with the dataset to which access is being requested.

Examples of ACU rules

- Users can access ADSN (anonymized data of Statistics Norway) only if they are registered and they have a registered project

  Users CAN download ADSN_data ONLY IF *register*(**user**) AND *register*(**project**)

- Question texts of market research should not be revealed to competitors

  Users CAN browse META(market-research)//questions ONLY IF NOT **user** IN Competitors

- Free access to Category1 (except for commercial use) to registered users

  Users WITH **purpose** $\neq$"Commercial" CAN access Category1 IF *register*(**user**)

Examples of ACU rules – 2

- Everybody can freely browse metadata

  Users CAN browse META(data)

- Users can download Data ONS_Dataset for a NonCommercial project or (for a Consultancy project if they are agree to Consultancy Condition)

  Users OF NonCommercial PROJECTS CAN download ONS_Datasets

  Users OF Consultancy PROJECTS CAN download ONS_Datasets IF *agreement*(**user**, CC.Id)

Current directions in access control

- Policy composition
- Assertion-based access control
- “Interactive” access control
- Support for dynamic conditions
- Looking for flexibility and expressiveness ... but also simplicity and manageability

OASIS (Organization for the Advancement of Structured Information Standards) currently working on a standard for the specification of access control policies.