Access Control: Policies, Models, and Mechanisms

Alessandro Armando

Computer Security Laboratory (CSec)
DIBRIS, University of Genova

Computer Security
Corso di Laurea Magistrale in Ingegneria Informatica

Motivation

Motivation

- Where are we? Confidentiality, Integrity, and Availability
- For C & I, we have seen approaches using cryptography:
  - Well suited for, e.g., network traffic.
  - Possibility for, e.g., protecting files on a computer.
  - Ill suited for, e.g., controlling processes, e.g., operations on data and other processes.

Basic concepts

Policies, models, and mechanisms

- A security policy defines what is allowed.
  It defines those executions of a system that are acceptable, or complementarily, those that are not acceptable.
  - It is analogous to a set of laws.
  - Typically defined in terms of high-level rules or requirements.

  They describe access restrictions, i.e. relationships between subjects and objects.

- A security model provides a formal representation of a security policy (or sometimes a family of policies).

- A security mechanism defines the low level sw/hw functions that implements the controls imposed by the policy and formally stated in the model.

Security Policies: Examples

- Security policy: A student has full access to information that he or she created. Students have no access to other students' information unless explicitly given. Students may access and execute a pre-defined selection of files and/or applications. ...
- E-Banking A bank customer may list his account balances and recent transactions. He may transfer funds from his accounts provided his total overdraws are under 10,000 Euro. Transfers resulting in larger overdraws must be approved by his account manager. ... Objects here include both data and processes.
- Privacy policies A subject may only have access to personal data if this access is necessary to perform its current task, and only if the subject is authorized to perform this task. In addition, the purpose of its current task must correspond to the purposes for which the personal data was obtained or consent must be given by the data subjects. Combines conditions with obligations on how data will be used.

Security Policies: Examples

- Security policy: A student has full access to information that he or she created. Students have no access to other students' information unless explicitly given. Students may access and execute a pre-defined selection of files and/or applications. ...
- E-Banking A bank customer may list his account balances and recent transactions. He may transfer funds from his accounts provided his total overdraws are under 10,000 Euro. Transfers resulting in larger overdraws must be approved by his account manager. ... Objects here include both data and processes.
- Privacy policies A subject may only have access to personal data if this access is necessary to perform its current task, and only if the subject is authorized to perform this task. In addition, the purpose of its current task must correspond to the purposes for which the personal data was obtained or consent must be given by the data subjects. Combines conditions with obligations on how data will be used.

Security Policies: Examples

- Security policy: A student has full access to information that he or she created. Students have no access to other students' information unless explicitly given. Students may access and execute a pre-defined selection of files and/or applications. ...
- E-Banking A bank customer may list his account balances and recent transactions. He may transfer funds from his accounts provided his total overdraws are under 10,000 Euro. Transfers resulting in larger overdraws must be approved by his account manager. ... Objects here include both data and processes.
- Privacy policies A subject may only have access to personal data if this access is necessary to perform its current task, and only if the subject is authorized to perform this task. In addition, the purpose of its current task must correspond to the purposes for which the personal data was obtained or consent must be given by the data subjects. Combines conditions with obligations on how data will be used.

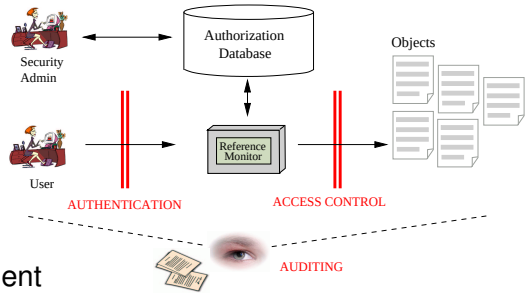
Security Policies

Access control policies can be grouped into three main classes:

- Discretionary (DAC) (authorization-based) policies control access based on the identity of the requestor and on access rules stating what requestors are (or are not) allowed to do.
- Mandatory (MAC) policies control access based on mandated regulations determined by a central authority.
- Role-based (RBAC) policies control access depending on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles.

DAC and RBAC policies are usually coupled with (or include) an administrative policy that defines who can specify authorizations/rules governing access control.

Access control – reference architecture



- System knows who the user is, i.e. authentication is done.
- Every request passes through a trusted component called reference monitor, that must enjoy the following properties:
  - tamper-proof: it should not be possible to alter it (or at least it should not be possible for alterations to go undetected);
  - non-bypassable: it must mediate all accesses to the system and its resources;
  - security kernel: it must be confined in a limited part of the system;
  - small: it must be of limited size to be susceptible of rigorous verification methods.

Protection state

- A state of a system is the collection of current values of all memory locations, storage, registers, and other components.
- The subset addressing protection is the system protection state.
- Examples of protection states

  File system: part of system state determining who is reading/writing files, access control information, etc.

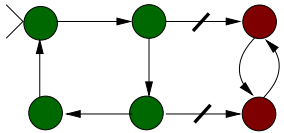
  Network: e.g., packet header information (identifying protocols) and packet locations, internal firewall states, etc.

  Program: e.g., part of run-time image corresponding to program counter, call stack, pointer addresses, etc.
- System changes correspond to (protection) state transitions.
  Abstraction: system as transition system.

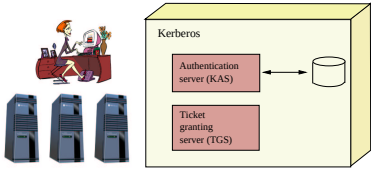
Protection state and security policy

- Let P be the system state space and $Q \subseteq P$ be the states in which system is authorized to reside in.
  - When the system is in a state $s \in Q$, the system is secure.
  - For $s \in P \setminus Q$, then system is not secure.
- A security policy characterizes Q. Hence a security policy partitions the states of the system into authorized or secure states, and unauthorized or nonsecure states.
- A security mechanism prevents a system from entering $P \setminus Q$.

A secure system is a system that starts in an authorized state and cannot enter an unauthorized state.



Example 1: Kerberos



Security policy: expresses which users can access what servers in a realm (or cross-realm).
The policy is formalized by the system administrator who registers users/servers in the database.

Security mechanism: Kerberos and kerberized application front-ends.

System state: determined by Kerberos server (e.g., system tables), state of protocol runs, and client/server states.

- Provides mechanism for CIA in a distributed setting.
- Partitions authentication, authorization, and access control.

Example 2: security policy for proprietary data

Security policy for company X

All information on product Y is confidential: it may only be read or modified by a subgroup Z and the system administrators.

Mechanism implications

- All printouts must be securely stored or shredded.
- All computer copies must be protected (AC, cryptography, ...)
- As company X stores its backup tapes in a vault in the town bank, X must ensure that only authorized employees have access to these tapes. Hence the bank's control on access to the vault and the procedures used to transport tapes to/from the bank are considered as security mechanisms.

System state

Includes not just the IT but also the entire operations. The security mechanisms are not only technical controls, but also procedural or operational controls.

Discretionary Access Control

Discretionary Access Control (DAC)

- Premise: users own resources and control their access.
- The owner of information or a resource is able to change its permission at his or her discretion. Owners can usually also transfer ownership of information to other users.
- Flexible, but open to mistakes, negligence, or abuse.
  - Requires that all system users understand and respect security policy and understand AC mechanisms.
  - Abuse, e.g., by Trojan horses: programs that trick users into transferring rights.

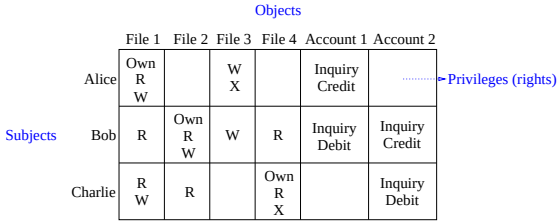
Discretionary Access Control

Access Control Matrix Model

- Simple framework for describing a protection system by describing the privileges of subjects on objects.

  Subjects: users, processes, agents, groups, ...
  Objects: data, memory banks, other processes, ...
  Privileges (or permissions/rights): read, write, modify, ...

- Used to represent a finite relation $AC \subseteq \text{Subjects} \times \text{Objects} \times \text{Privileges}$



given as a matrix.

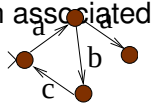
AC Matrix Model – formal definitions (I)

- A protection state (relative to a set Privileges) is a triple (S, O, M):
  - S: Set of subjects.
  - O: Set of objects.
  - M: a matrix defining the privileges for each $(s, o) \in S \times O$ where
    $$M(s, o) = \{p \in \text{Privileges} \mid (s, o, p) \in AC\}$$
- State transitions described by commands like
  - **enter** privilege/right p **into** $M(s, o)$
  - **create** subject s
  - **destroy** object o

  These transform one state into another by changing its parts.

AC Matrix Model – formal definitions (II)

- Write $(S, O, M) \vdash_c (S', O', M')$ to denote a transition associated with the command c.
- A starting state $s_0 = (S_0, O_0, M_0)$ and a set of commands C determines a state-transition system.



- So a model describes a set of (permitted) system traces, namely those traces of the form

$$s_0, s_1, s_2, s_3, \ldots$$

where $s_i \vdash_{c_i} s_{i+1}$ for $c_i \in C$.

Discretionary Access Control

The Harrison-Ruzzo-Ullman Model (1976)

The Harrison-Ruzzo-Ullman model defines authorization systems formalizing how to change access rights or how to create and delete subjects and objects.

- State (S, O, M), for subjects S, objects O, matrix M.
- State transitions described by commands of the form

command $c(x_1, \ldots, x_k)$
  if r_1 in $M(x_{s_1}, x_{o_1})$ and $\ldots$ r_m in $M(x_{s_m}, x_{o_m})$
  then op_1; $\ldots$ op_n
end

for rights r_i, integers s_i and o_i, primitive operations op_i, e.g.

- **enter** privilege/right p **into** $M(s, o)$
- **create** subject s

The Harrison-Ruzzo-Ullman Model (cont.)

Instances of

command $c(x_1, \ldots, x_k)$
  if r_1 *in* $M(x_{s_1}, x_{o_1})$ and $\ldots$ r_m *in* $M(x_{s_m}, x_{o_m})$
  then op_1; $\ldots$ op_n
end

are

command *create.file*(s, o)
  create o
  enter *Own* **into** $M(s, o)$
  enter R **into** $M(s, o)$
  enter W **into** $M(s, o)$
end

command *confer.write*$(s1, s2, o)$
  if $Own \in M(s1, o)$
  then enter W **into** $M(s2, o)$
end

The Harrison-Ruzzo-Ullman Model (cont.)

Six primitive operations causing a transition from state (S, O, M) to state (S', O', M'):

Operation	Conditions	New State
enter r **into** $M(s, o)$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $M'(s, o) = M(s, o) \cup \{r\}$ $M'(s_1, o_1) = M(s_1, o_1)$ for $(s_1, o_1) \neq (s, o)$
delete r **from** $M(s, o)$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $M'(s, o) = M(s, o) \setminus \{r\}$ $M'(s_1, o_1) = M(s_1, o_1)$ for $(s_1, o_1) \neq (s, o)$

Assumption: all subjects are objects, i.e. $S \subseteq O$.

Operation	Conditions	New State
create subject s'	$s' \notin S$	$S' = S \cup \{s'\}$ $O' = O \cup \{s'\}$ $M'(s,o) = M(s,o)$ for $s \in S$, $o \in O$ $M'(s',o) = \emptyset$ for $o \in O'$ $M'(s,s') = \emptyset$ for $s \in S'$
destroy subject s'	$s' \in S$	$S' = S \setminus \{s'\}$ $O' = O \setminus \{s'\}$ $M'(s,o) = M(s,o)$ for $s \in S'$, $o \in O'$
create object o'	$o' \notin O$	$S' = S$ $O' = O \cup \{o'\}$ $M'(s,o) = M(s,o)$ for $s \in S$, $o \in O$ $M'(s,o') = \emptyset$ for $s \in S'$
destroy object o'	$o' \in O$ $o' \notin S$	$S' = S$ $O' = O \setminus \{o'\}$ $M'(s,o) = M(s,o)$ for $s \in S'$, $o \in O'$

Exercise

1. Specify the commands *confer.execute* and *revoke.write*.
2. Compute the matrix that results from the following initial state

	File1	File2	File3	File4
Alice	Own R W		W X	
Bob	R	Own R W	W	R
Charlie	R W	R		Own R W

by executing the sequence of commands

create.file(Alice,File5)
confer.execute(Alice,Charlie,File5)
revoke.write(Bob,Alice,File1)
revoke.read(Charlie,Bob,File4)

Access matrix – policy example

Policy: No subject can acquire read access to a file unless that right R has been explicitly granted by the file's owner.

	File1	File2	File3	File4
Alice	Own R W		W X	
Bob	R	Own R W	W	R
Charlie	R W	R		Own R W

Formalization: Let $s = (S, O, M)$ be an authorized state such that $\mathit{Own} \in M(p, f)$ for subject p and file f but $R \notin M(q, f)$ for subject q. Let $s' = (S', O', M')$ be a state such that $s \vdash_c s'$ and $R \in M(q, f)$. Then s' is authorized iff $c = \mathit{confer.read}(p, q, f)$ where:

command $\mathit{confer.read}(s1, s2, o)$
  if $\mathit{Own} \in M(s1, o)$
  then enter R **into** $M(s2, o)$
end

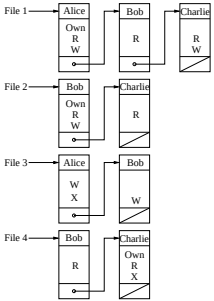
So if initial state is authorized, and this is the only command to insert read permissions into M, then all reachable states are authorized.

Implementing the Access Matrix: Data Structures

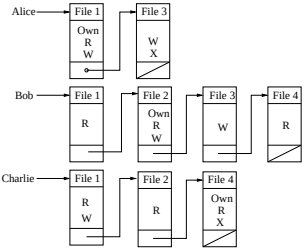
Access Matrix

	File 1	File 2	File 3	File 4
Alice	Own R W		W X	
Bob	R	Own R W	W	R
Charlie	R W	R		Own R X

AC List (ACL)

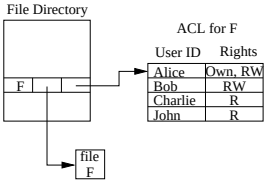


Capabilities List



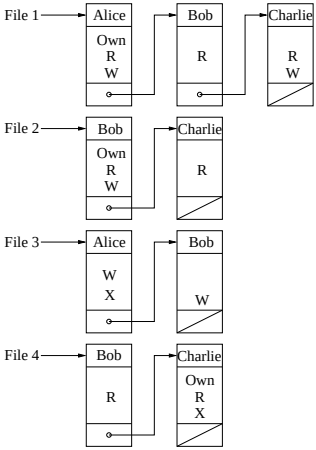
Access-control (authorization) list

- ACL: use lists to express view of each object o: i-th entry in the list gives the name of a subject s_i and the rights r_i in $M(s_i, o)$ of the access-matrix.
- Standard example: AC for files.

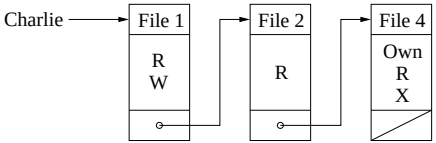


Owner has the sole authority to grant, revoke or decrease access rights to F to other users.

- ACLs are found, for example in the DEC VMS operating system and Linux.



Capability list



- Subject view of AC matrix.
- Less common than ACLs.
  - Not so compatible with object oriented view of the world.
  - Difficult to get an overview of who has permissions on an object.
  - Difficult to revoke a capability. E.g., `chmod 700 *`
- Application in distributed (e.g., mobile agent) setting.
  Users are endowed with credentials (e.g., from a credential server) that they present to network objects.

A simple DAC example: Unix

Unix provides a mechanism based on simplified ACLs suitable for a restricted class of DAC policies.

- Controls access per object (file, directory, ...) using permission scheme *owner/group/other*.
- Objects are assigned to a single user (owner) and a single group (normally the group of the directory containing it). They can be changed by using the `chown` and `chgrp` commands, respectively.
- Each user can be assigned to multiple groups (cf. `useradd`).
- Permission bits assigned to objects by their owners or by the administrator (root), (cf. `chmod`).

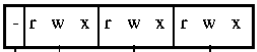
```
> ls -la
drwx------ 8 armando users 12288 May 26 22:34 .
drwx------ 9 armando users  4096 May 26 19:07 ..
-rw-r--r-- 1 armando users  6523 May 27 00:35 access.tex
drwxr-xr-x 2 armando users  2048 May 26 22:27 fig/
> groups
users admin libvirtd
```

A simple DAC example: Unix

Unix provides a mechanism based on simplified ACLs suitable for a restricted class of DAC policies.

- Controls access per object (file, directory, ...) using permission scheme *owner/group/other*.
- Objects are assigned to a single user (owner) and a single group (normally the group of the directory containing it). They can be changed by using the `chown` and `chgrp` commands, respectively.
- Each user can be assigned to multiple groups (cf. `useradd`).
- Permission bits assigned to objects by their owners or by the administrator (root), (cf. `chmod`).

```
> ls -la
drwx------ 8 armando
drwx------ 9 armando
-rw-r--r-- 1 armando
drwxr-xr-x 2 armando
> groups
users admin libvirtd
```



File Mode Bits

Other permissions
Group permissions
User permissions
Type of file: a dash "-" indicates a file, a "d" indicates a directory

Structure of File Mode Bits

The file mode bits have two parts:

- file permission bits, which control ordinary access to the file
- special mode bits, which affect only only executable files (programs) and, on most systems, directories

File Permission Bits

File permission bits control ordinary access to the file:

1. permission to read the file. For directories, this means permission to list the contents of the directory.
2. permission to write to (change) the file. For directories, this means permission to create and remove files in the directory.
3. permission to execute the file (run it as a program). For directories, this means permission to access files in the directory.

Special Mode Bits

Special mode bits affect only only executable files (programs) and directories:

1. Set the process's effective user ID to that of the file upon execution (called the "set-user-ID bit", or "setuid bit").
2. Set the process's effective group ID to that of the file upon execution (called the "set-group-ID bit", or "setgid bit").
3. When a user other than the owner executes the file, the process will run with user and/or group permissions set upon it by its owner. For example, if the file is owned by user root and group wheel, it will run as root:wheel no matter who executes the file.
4. Prevent unprivileged users from removing or renaming a file in a directory unless they own the file or the directory; this is called the "restricted deletion flag" for the directory, and is commonly found on world-writable directories like `/tmp`.

A simple DAC example: Unix

- Not all policies can be directly mapped onto this mechanism.
  How would we express that a patient can access his medical records at a hospital?
  Who owns the records?
- Supports limited delegation of rights using setuid [or setgid].
  - Executor takes on owner's user [group] identity during execution.
  - Example: normal users "upgraded" to root to change their passwords in the password file.
  - Open to abuse and the cause of many security holes.

A simple DAC example: Unix

- Not all policies can be directly mapped onto this mechanism.
  How would we express that a patient can access his medical records at a hospital? Who owns the records?
- Supports limited delegation of rights using setuid [or setgid].
  - Executor takes on owner's user [group] identity during execution.
  - Example: normal users "upgraded" to root to change their passwords in the password file.
  - Open to abuse and the cause of many security holes.

A simple DAC example: Unix

- Not all policies can be directly mapped onto this mechanism.
  How would we express that a patient can access his medical records at a hospital? Who owns the records?
- Supports limited delegation of rights using setuid [or setgid].
  - Executor takes on owner's user [group] identity during execution.
  - Example: normal users "upgraded" to root to change their passwords in the password file.
  - Open to abuse and the cause of many security holes.

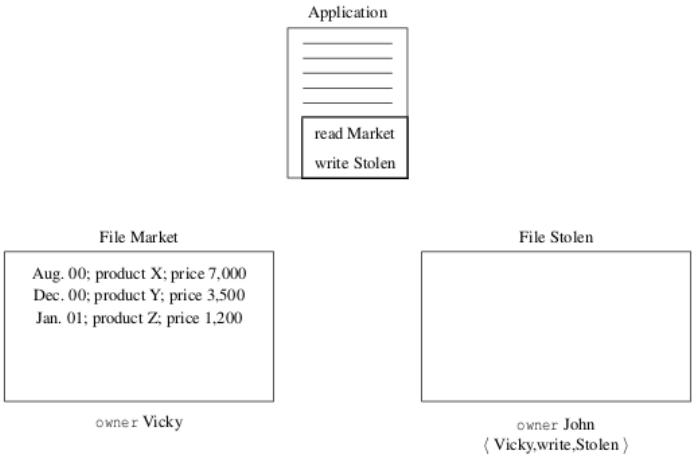
Problem with Discretionary Policies: Trojan Horses

- Discretionary policies do not distinguish users from subjects.
- Users are passive entities for whom authorizations can be specified and who can connect to the system.
- Once connected to the system, users originate processes (subjects) that execute on their behalf and, accordingly, submit requests to the system.
- Discretionary policies ignore this distinction and evaluate all requests submitted by a process running on behalf of some user against the authorizations of the user.
- This makes discretionary policies vulnerable from processes executing malicious programs exploiting the authorizations of the user on behalf of whom they are executing.
- Access control system can thus be bypassed by Trojan Horses embedded in programs.

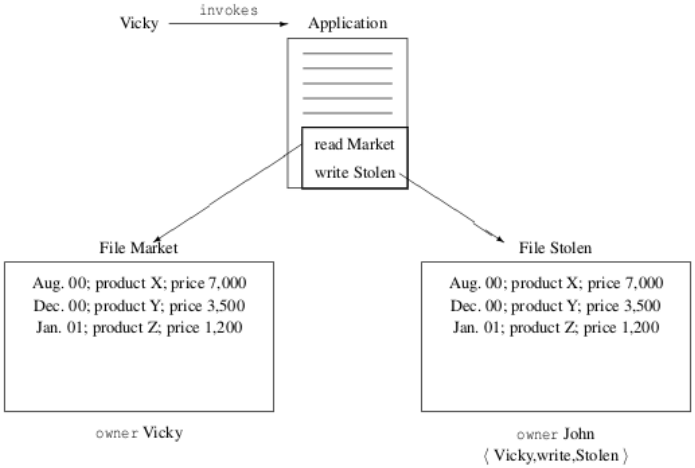
Problem with Discretionary Policies: Trojan Horses

- A Trojan Horse is a computer program which contains hidden functions that surreptitiously exploit the legitimate authorizations of the invoking process.
- A Trojan Horse can improperly use any authorizations of the invoking user, for example, it could even delete all files of the user.
- Since discretionary policies do not enforce any control on the flow of information once this information is acquired by a process,
  it is possible for processes to leak information to users not allowed to read it.
- All this can happen without the cognizance of the data administrator/owner, and despite the fact that each single access request is controlled against the authorizations.

Problem with Discretionary Policies: Trojan Horses



Problem with Discretionary Policies: Trojan Horses



Android Security Model

- Android is a privilege-separated operating system, in which each application runs with a distinct identity (Linux user ID and group ID).
- Parts of the system are also separated into distinct identities.
- Linux isolates applications from each other and from the system.

> No application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user.

- Additional finer-grained security features are provided through
  - a “permission” mechanism that enforces restrictions on the specific operations that a particular process can perform, and
  - per-URI permissions for granting ad-hoc access to specific pieces of data.

Mandatory Access Control

Mandatory Access Control (MAC)

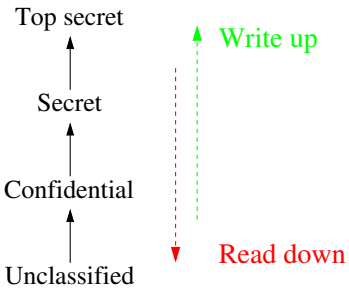
- AC decisions formalized (controlled) by comparing
  security labels indicating sensitivity/criticality of objects, with
  formal authorization, i.e. security clearances, of subjects.
- System wide access restriction to objects. Mandatory because subjects may not transfer their access rights.
- Example from military: users and objects assigned a clearance level like secret, top secret, etc. Users can only read [write] objects of equal or lower [higher] levels.
- More rigid than DAC, but also more secure.
- Concrete examples (e.g., Bell-LaPadula, Biba).

Access control models

Various types of security models:

- Models can capture policies for confidentiality (Bell-LaPadula) or for integrity (Biba, Clark-Wilson).
- Some models apply to environments with static policies (Bell-LaPadula), others consider dynamic changes of access rights (Chinese Wall).
- Security models can be informal (Clark-Wilson), semi-formal, or formal (Bell-LaPadula, Harrison-Ruzzo-Ullman).

MAC: Linear Ordering



Two principles are required to hold for confidentiality:

- Read down: a subject's clearance must dominate (i.e. $\geq$) the security level of the object being read.
- Write up: a subject's clearance must be dominated by (i.e. $\leq$) the security level of the object being written.
  - Often restricted only to same level (i.e. $=$) to avoid a subject overwriting data it cannot read;
  - Does not allow a subject to write "lower" data; to that end a subject should be enabled to dynamically decrease its level.

Dually for integrity: read up and write down.

MAC: The Lattice of Security Levels

To answer uniquely questions like

- Given 2 objects at different security levels, what is the minimal level a subject must have to be allowed to read both objects?
- Given 2 subjects at different security levels, what is the maximal level an object can have so that it still can be read by both subjects?

Def: a lattice $(L, \leq)$ consists of a set of security levels L and a partial ordering $\leq$, so that for every 2 elements $a, b \in L$ there exists a least upper bound $u \in L$ and a greatest lower bound $l \in L$, i.e.

$$\begin{array}{ll} a \leq u, b \leq u & \text{and } (a \leq v \ \& \ b \leq v) \rightarrow (u \leq v) \text{ for all } v \in L \\ l \leq a, l \leq b & \text{and } (k \leq a \ \& \ k \leq b) \rightarrow (k \leq l) \text{ for all } k \in L \end{array}$$

MAC: The Lattice of Security Levels

To answer uniquely questions like

- Given 2 objects at different security levels, what is the minimal level a subject must have to be allowed to read both objects?
- Given 2 subjects at different security levels, what is the maximal level an object can have so that it still can be read by both subjects?

Def: a lattice $(L, \leq)$ consists of a set of security levels L and a partial ordering $\leq$, so that for every 2 elements $a, b \in L$ there exists a least upper bound $u \in L$ and a greatest lower bound $l \in L$, i.e.

$$\begin{array}{ll} a \leq u, b \leq u & \text{and } (a \leq v \ \& \ b \leq v) \rightarrow (u \leq v) \text{ for all } v \in L \\ l \leq a, l \leq b & \text{and } (k \leq a \ \& \ k \leq b) \rightarrow (k \leq l) \text{ for all } k \in L \end{array}$$

MAC: The Lattice of Security Levels

To answer uniquely questions like

- Given 2 objects at different security levels, what is the minimal level a subject must have to be allowed to read both objects?
- Given 2 subjects at different security levels, what is the maximal level an object can have so that it still can be read by both subjects?

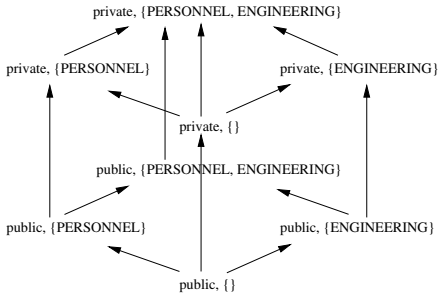
Def: a lattice $(L, \leq)$ consists of a set of security levels L and a partial ordering $\leq$, so that for every 2 elements $a, b \in L$ there exists a least upper bound $u \in L$ and a greatest lower bound $l \in L$, i.e.

$$a \leq u, b \leq u \quad \text{and } (a \leq v \ \& \ b \leq v) \rightarrow (u \leq v) \text{ for all } v \in L$$
$$l \leq a, l \leq b \quad \text{and } (k \leq a \ \& \ k \leq b) \rightarrow (k \leq l) \text{ for all } k \in L$$

Example (from DoD's Orange Book)

- A set H of classifications with a hierarchical (linear) ordering $\leq_H$.
- A set C of categories, e.g. project names, company divisions, etc.
- A security label is a pair (h, c) with $h \in H$ and $c \subseteq C$.
- Partial order of labels: $(h_1, c_1) \leq (h_2, c_2)$ iff $h_1 \leq_H h_2$ and $c_1 \subseteq c_2$.

For hierarchical levels *public* and *private*, and categories PERSONNEL and ENGINEERING, we have the lattice:
Note that (public,{PERSONNEL}) $\not\leq$ (private,{ENGINEERING}).



Mandatory Access Control

The Bell-LaPadula (BLP) Model (1975)

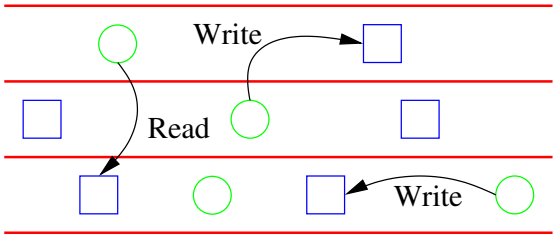
- Models confidentiality aspects of multi-user systems, e.g. in operating systems or database management systems.
- Probably most famous and influential security model:
  - Developed as part of U.S. government funded research at the MITRE corporation on security models and the prevention of disclosure threats in multi-user operating systems.
  - Basis of several standards, including DoD's Trusted Computer System Evaluation Criteria (TCSEC or "Orange Book").
  - It also raised some controversy (on suitable definition of security model).

The Bell-LaPadula (BLP) Model (cont.)

BLP models confidentiality by combining aspects of DAC and MAC:

- Access permissions are defined both through an AC matrix and through security levels.
- Multi-level security (MLS): mandatory policies prevent information flowing downwards from a high security level to a low one.
- BLP is a static model: security levels (labels) never change.

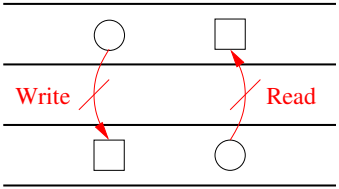
BLP: Level Diagrams

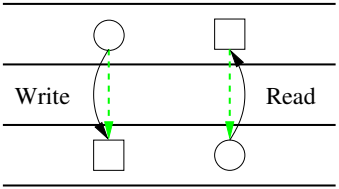


- Horizontal lines: boundaries between levels (with partial order $\leq$).
- Circles: subjects.
- Squares: objects.
- Directed arcs from subjects to objects: operations (e.g. read, write, execute).

BLP: Level Diagrams (cont.)

- Level diagrams also for disallowed operations and information flow:





- But level diagrams do not adequately represent the lattice properties of the security labels under the *dominates* ($\leq$) relation.
  E.g.: when a subject s tries to access an object o with an unrelated security label (i.e. where neither $label(s) \leq label(o)$ nor $label(s) \geq label(o)$).
- Formal BLP model for "real" security policies.

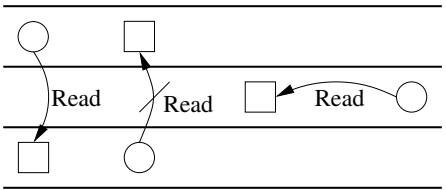
BLP: Security Properties

BLP defines different security properties for a state, e.g. the following two (slightly simplified) properties:

Simple security property (ss-property)

A subject s can read an object o only if the security level l_s of s dominates the security level l_o of o, i.e. $l_o \leq l_s$.

Also known as no-read-up (NRU):



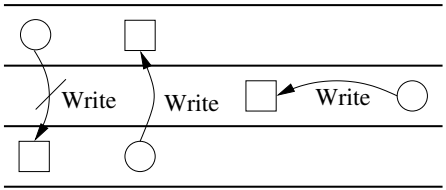
e.g.: an 'unclassified' s should not be able to read a 'confidential' o.

BLP: Security Properties (cont.)

$*$-property (star-property)

A subject s can write an object o only if the security level l_o of o dominates the security level l_s of s, i.e. $l_s \leq l_o$.

Also known as no-write-down (NWD):



e.g.: a 'confidential' s should not be able to write an 'unclassified' o.

BLP: Security Properties (cont.)

- NWD prevents a high-level subject from sending messages to a low-level one. Possible solutions:
  - Temporarily downgrade the level of s ("current security level").
  - Identify a set of subjects (called trusted subjects), which are permitted to violate the $*$-property.
- Rationale of ss-property and $*$-property: no information leakage.
  - No-read-up and no-write-down prevent untrusted subjects from simultaneously having read access to information at one level and write access to information at a lower level.
  - Read-down and write-up are fine: a 'confidential' s should be able to read an 'unclassified' o_1 in order to write a 'confidential' o_2.
- Other security properties can be formalized in BLP.

BLP: Features and Limitations

- BLP is well-suited for modeling confidentiality in operating systems or database management systems.
- BLP does suffer from a number of limitations, though:
  - It does not specify how to change access rights or how to create and delete subjects and objects.
  - It is a static model (this tranquility raised much controversy).
  - It contains covert channels, i.e. information-flows that are not controlled by security mechanisms.

  The first problem can be addressed by employing the Harrison-Ruzzo-Ullman model.

BLP: System Z and the Tranquility Property

- System Z has only one state transition, which
  - downgrades all subjects and objects to the lowest security level,
  - enters all access rights in all positions of the AC matrix M.
- System Z satisfies BLP's notion of security.

According to the basic security theorem of BLP, the state reached by this transition is secure, but is it really?

- The case against BLP (McLean): BLP needs to be improved, as a system that can be brought into a state where everyone is allowed to read everything is not secure.
- The case for BLP (Bell): this is not a problem of BLP but rather of correctly capturing the security requirements.
  If the user requirements call for such a transition, then it should be allowed in the model, else it should not be implemented.

BLP: System Z and the Tranquility Property (cont.)

- At the root of this disagreement is a state transition that changes security levels (and access rights).
- BLP is however a static model: security levels are fixed.

  > Strong tranquility property: the security levels of subjects and objects never change during system operation.

  > Weak tranquility property: the security levels never change in such a way as to violate a defined security policy.

  For example, it can be required that the level of an object never be changed while it is being used by some subject.
- This limitation is lifted in dynamic models that are based on BLP.

BLP: Covert Channels

Sometimes it is not sufficient to hide the contents of objects, but also their existence must be hidden.

- In BLP, the AC mechanism itself can be used to construct a covert channel, where information flows from a high security level to a low one:
  1. A low-level subject creates an object `dummy.obj` at its own level.
  2. Its high-level accomplice (e.g. a Trojan horse) either upgrades the security level of `dummy.obj` to high or leaves it unchanged.
  3. Later, the low-level subject tries to read `dummy.obj`.
     Success or failure of this request disclose the action of the high-level subject.
     Hence, one bit of information has flown from high to low.
- That is: telling a subject that a certain operation is not permitted already constitutes information-flow.
- Problem can be solved; e.g. in database security an object may have different values at different security levels (polyinstantiation).

BLP: Covert Channels

Sometimes it is not sufficient to hide the contents of objects, but also their existence must be hidden.

- In BLP, the AC mechanism itself can be used to construct a covert channel, where information flows from a high security level to a low one:
  1. A low-level subject creates an object `dummy.obj` at its own level.
  2. Its high-level accomplice (e.g. a Trojan horse) either upgrades the security level of `dummy.obj` to high or leaves it unchanged.
  3. Later, the low-level subject tries to read `dummy.obj`.
     Success or failure of this request disclose the action of the high-level subject.
     Hence, one bit of information has flown from high to low.
- That is: telling a subject that a certain operation is not permitted already constitutes information-flow.
- Problem can be solved; e.g. in database security an object may have different values at different security levels (polyinstantiation).

BLP: Covert Channels

Sometimes it is not sufficient to hide the contents of objects, but also their existence must be hidden.

- In BLP, the AC mechanism itself can be used to construct a covert channel, where information flows from a high security level to a low one:
  1. A low-level subject creates an object `dummy.obj` at its own level.
  2. Its high-level accomplice (e.g. a Trojan horse) either upgrades the security level of `dummy.obj` to high or leaves it unchanged.
  3. Later, the low-level subject tries to read `dummy.obj`.
     Success or failure of this request disclose the action of the high-level subject.
     Hence, one bit of information has flown from high to low.
- That is: telling a subject that a certain operation is not permitted already constitutes information-flow.
- Problem can be solved; e.g. in database security an object may have different values at different security levels (polyinstantiation).

Mandatory Access Control

The Biba Model (K.J. Biba, 1977)

- In BLP: no-read-up and no-write-down for confidentiality.
  - But: write-up and read-down can introduce integrity problems.
- Biba (also of MITRE) proposed a class of integrity models with the opposite rules:
  - Mandatory integrity model: no-read-down and no-write-up.
  - Relax no-read-down ("subject low watermark property"):
    Allow a subject to read down, but first lower its integrity level to that of the object being read.
  - Relax no-write-up ("object low watermark property"):
    Lower object level to that of subject doing the write.
- Biba and BLP can be combined (albeit not straightforwardly) to model both confidentiality and integrity.

The Biba Model (cont.)

- Addresses integrity in terms of access by subjects to objects using a model similar to that of BLP.
  - A lattice $(L, \leq)$ of security levels.
  - $f_S : S \rightarrow L$ and $f_O : O \rightarrow L$ assign integrity levels to subjects and objects.
  - Information may only flow downwards in the integrity lattice.
- Unlike BLP, there is no single high-level integrity policy but rather a variety of policies (some even mutually incompatible).
  - Static integrity levels.
  - Dynamic integrity levels.
  - Policies for invocation.

Biba: Static Integrity Levels

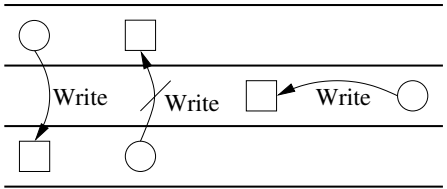
- Policies where integrity levels never change (mirroring BLP's tranquility).
- Two properties (dual of the mandatory BLP policies):

No-write-up

s can modify o if and only if $f_S(s) \geq f_O(o)$.

No-read-down

s can modify o if and only if $f_o(o) \geq f_s(s)$.



Biba: Dynamic Integrity Levels

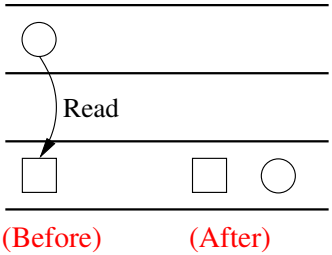
Low watermark properties (similar to Chinese Wall) automatically adjust the integrity level of an entity if it has come into contact with low-level information:

- Subject low watermark property: relax no-read-down.
  Allow a subject to read down, but first lower its integrity level to that of the object being read.
  Write operations are constrained according to the no-write-up principle.
- Object low watermark property: relax no-write-up.
  Lower object level to that of subject doing the write.
  Read operations are constrained according to the no-read-down principle.

Biba: Dynamic Integrity Levels (cont.)

Subject low watermark property

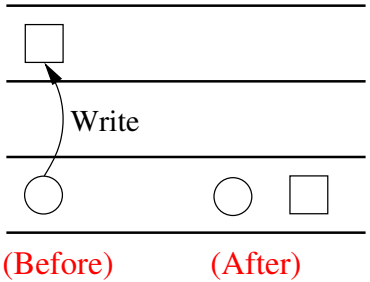
s can read an o at any integrity level.
The new integrity level of s is $\mathrm{glb}(f_S(s), f_O(o))$, where $f_S(s)$ and $f_O(o)$ are the integrity levels before the operation.



Biba: Dynamic Integrity Levels (cont.)

Object low watermark property

s can modify an *o* at any integrity level.
The new integrity level of *o* is $\mathrm{glb}(f_S(s), f_O(o))$, where $f_S(s)$ and $f_O(o)$ are the integrity levels before the operation.



Mandatory Access Control

The Chinese Wall Model (Brewer and Nash, 1989)

- A commercially inspired confidentiality model (whereas most commercial models focus on integrity).
- Models access rules in a consultancy business where analysts have to make sure that no conflicts of interest arise when they are dealing with different companies.
- Informally, conflicts arise
  - because clients are direct competitors in the same market, or
  - because of the ownership of companies.

Rule: There must be no information-flow that causes a conflict of interest.

The Chinese Wall Model (cont.)

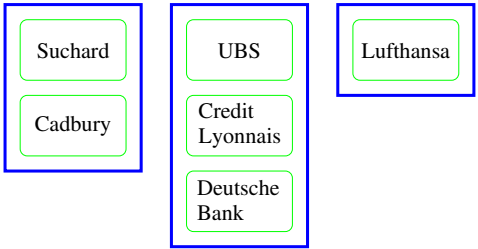
An adaptation of Bell-LaPadula, with three levels of abstraction:

1. Companies, subjects and objects:
   - A set C of companies, and a set S of subjects (the analysts).
   - A set O of objects, which are items of information (e.g. files) concerning a single company.
2. All objects concerning the same company are collected in a company dataset. $cd : O \to C$ gives the company dataset of each object.
3. Conflict of interest classes indicate which companies are in competition. $cic : O \to \mathcal{P}(C)$ gives the conflict of interest class for each object o, i.e. the set of companies that should not learn the contents of o.

The security label of an object o is the pair $(cic(o), cd(o))$.

The Chinese Wall Model (cont.)

- Each object belongs to a unique company dataset.
- Each company dataset is contained in a unique conflict class.
- A conflict class may contain one or more company datasets.
- For example, chocolate, banks and airlines:



- Six company datasets: one for each company.
- Three conflict classes: {Suchard, Cadbury}, {UBS, Credit Lyonnais, Deutsche Bank}, {Lufthansa}.

The Chinese Wall Model (cont.)

- Conflicts arise not only from objects currently addressed but also from objects that have been accessed in the past.
  A Boolean $S \times O$ matrix N records the subjects' actions:

  $$N(s, o) = \text{true, if subject } s \text{ has had access to object } o$$

  A secure initial state: set $N(s, o) = \text{false}$ for all $s \in S$ and $o \in O$.
- Access permissions change dynamically and must be reexamined at every state transition: as a subject accesses some objects, other objects that would previously have been accessible are now denied.

The Chinese Wall Model (cont.)

A simple policy to prevent conflict of interest:

A subject s can access any information as long as it has never accessed information from a different company in the same conflict class.

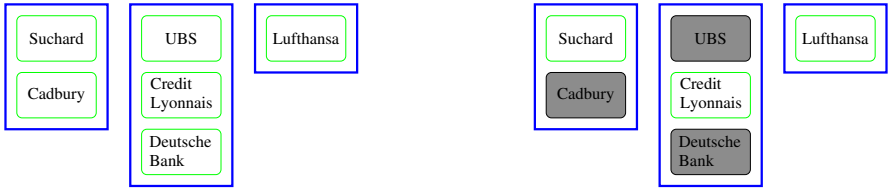
That is, access is granted if and only if requested object o belongs to

- either a company dataset already held by s (o is in the same company dataset as an object that has been previously accessed),
- or an entirely different conflict of interest class (i.e. the class has never before been accessed).

ss-property

s is permitted access to o only if for all o' with $N(s, o') = \text{true}$, it holds $cd(o) = cd(o')$ or $cd(o) \notin cic(o')$.

The Chinese Wall Model (cont.)



- Initially (figure on the left), each object can be accessed.
- If *s* reads from a file on Suchard, then a subsequent access request
  - to any bank or to Lufthansa would be granted,
  - to Cadbury files would be denied.
- A subsequent access
  - to Lufthansa does not affect future accesses,
  - to a file on Credit Lyonnais blocks future accesses to UBS or Deutsche Bank.
- From that point on (figure on the right, with grey datasets blocked), only objects on Suchard, Lufthansa or Credit Lyonnais (or in a new conflict class) can be accessed.

The Chinese Wall Model (cont.)

ss-property

s is permitted access to o only if for all o' with $N(s, o') = \text{true}$, it holds $cd(o) = cd(o')$ or $cd(o) \notin cic(o')$.

Indirect information-flow is still possible with this property, e.g.

- Two competitors, *Company1* and *Company2*, have their accounts with the same *Bank*.
- *Analyst1*, dealing with *Company1* and the *Bank*, updates the *Bank* portfolio with sensitive information about *Company1*.
- *Analyst2*, dealing with *Company2* and the *Bank*, now has access to information about a competitor's business.

The Chinese Wall Model (cont.)

- Information is sanitized if it has been purged of sensitive details and is not subject to access restrictions.

$$cic(o) = \emptyset \text{ for a sanitized object } o.$$

- Hence, grant write access to an object only if no other object can be read which is in a different company dataset and contains unsanitized information.

$*$-property

s is granted write access to o only if s has no read access to an object o' with $cd(o) \neq cd(o')$ and $cic(o') \neq \emptyset$.

Summarizing:

- BLP: access rights are (usually) assumed to be static.
- Chinese Wall: access rights are changed dynamically, and must thus be re-examined in every state transition.

Mandatory Access Control

The Clark-Wilson Integrity Model (1987)

- An informal integrity model
  - motivated by (and abstracted from) the way commercial organizations control the integrity of their data,
  - quite different from level-oriented ones (e.g. BLP and Biba).
- Objects (data items) are partitioned into
  - constrained data items (CDI), and
  - unconstrained data items (UDI).
- Nine rules, based on business "best practices", constraining
  - how integrity of CDIs must be validated,
  - how and by whom CDIs can be changed,
  - how UDIs can be changed to CDIs.

  For example (rule 2): application of a transformation procedure to any CDI must maintain the integrity of that CDI.

Role-Based Access Control

Role-Based Access Control: Motivations

- RBAC is an alternative to traditional DAC and MAC policies that is widely used in commercial applications.
- Allows to specify and enforce enterprise-specific security policies in a way that maps naturally to an organization's structure.
- For access control it is much more important to know what a user's organizational responsibilities are, rather than who the user is. (In most business activities user's identity is relevant only for accountability.)
- Conventional DAC approaches, in which individual user ownership of data is key, are not a good fit. Neither are the full mandatory access controls, in which users have security clearances and objects have security classifications.
- RBAC fills in this gap by merging the flexibility of explicit authorizations with additionally imposed organizational constraints.

Role-based Access Control

User	Permission
Alice	GrantTenure
Alice	AssignGrades
Alice	ReceiveHBenefits
Alice	UseGym
Bob	GrantTenure
Bob	AssignGrades
Bob	UseGym
Charlie	GrantTenure
Charlie	AssignGrades
Charlie	UseGym
David	AssignHWScores
David	Register4Courses
David	UseGym
Eve	ReceiveHBenefits
Eve	UseGym
Fred	Register4Courses
Fred	UseGym
Greg	UseGym

Role-based Access Control

Idea: decompose subject/object relationship using roles:

s has permission *p* on *o* iff there exists role *r* s.t. *s* has role *r* and *r* has permission *p* on object *o*.

User Assignment (UA)

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

Permission Assignment (PA)

Role	Permission
PCMember	GrantTenure
PCMember	AssignGrades
PCMember	ReceiveHBenefits
PCMember	UseGym
Faculty	AssignGrades
Faculty	ReceiveHBenefits
Faculty	UseGym
TA	AssignHWScores
TA	Register4Courses
TA	UseGym
UEmployee	ReceiveHBenefits
UEmployee	UseGym
Student	Register4Courses
Student	UseGym

Roles vs Groups

- Groups define sets of users while roles define sets of privileges.
- Roles can be “activated” and “deactivated” by users at their discretion.
- Group membership always applies, that is, users cannot enable and disable group memberships (and corresponding authorizations) at their will.
- However, since roles can be defined which correspond to organizational figures (e.g., secretary, chair, and faculty), a same “concept” can be seen both as a group and as a role.

RBAC: Administration Management

RBAC greatly simplifies the management of the security policy:

- when a new user joins the organization, the administrator only needs to grant her the roles corresponding to her job;
- when a user's job changes, the administrator simply has to change the roles associated with that user;
- when a new application or task is added to the system, the administrator needs only to decide which roles are permitted to execute it.

RBAC: Role Hierarchies

The use of role hierarchies leads to a simplification of the PA relation:

s has permission *p* on *o* iff there exist roles *r* and *r′* s.t. $r \geq r'$, *s* has role *r*, and *r′* has permission *p* on object *o*.

User Assignment (UA)

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

Permission Assignment (PA)

Role	Permission
PCMember	GrantTenure
PCMember	AssignGrades
PCMember	ReceiveHBenefits
PCMember	UseGym
Faculty	AssignGrades
Faculty	ReceiveHBenefits
Faculty	UseGym
TA	AssignHWScores
TA	Register4Courses
TA	UseGym
UEmployee	ReceiveHBenefits
UEmployee	UseGym
Student	Register4Courses
Student	UseGym
UMember	UseGym

RBAC: Role Hierarchies

The use of role hierarchies leads to a simplification of the PA relation:

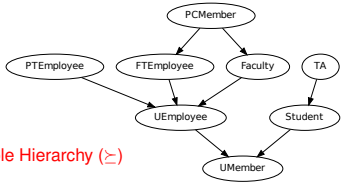
s has permission *p* on *o* iff there exist roles *r* and *r′* s.t. $r \geq r'$, *s* has role *r*, and *r′* has permission *p* on object *o*.

User Assignment (UA)

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

Permission Assignment (PA)

Role	Permission
PCMember	GrantTenure
Faculty	AssignGrades
TA	AssignHWScores
UEmployee	ReceiveHBenefits
Student	Register4Courses
UMember	UseGym



Role Hierarchy ($\succeq$)

Further Advantages of RBAC

- Least privilege. Roles allow a user to sign on with the least privilege required for the particular task she needs to perform. Users authorized to powerful roles do not need to exercise them until those privileges are actually needed. This minimizes the danger of damage due to inadvertent errors or Trojan Horses.
- Separation of duties. Separation of duties refer to the principle that no user should be given enough privileges to misuse the system on their own. For instance, the person authorizing a paycheck should not be the same person who can prepare them. Separation of duties can be enforced either statically or dynamically.
- Constraints enforcement. Roles provide a basis for the specification and enforcement of further protection requirements. For instance, cardinality constraints, that restrict the number of users allowed to activate a role or the number of roles allowed to exercise a given privilege. The constraints can be dynamic, i.e. be imposed on roles activation rather than on their assignment.

Beyond RBAC

- The simple hierarchical relationship is not sufficient to model the different kinds of relationships that can occur.
  For example, a secretary may need to be allowed to write specific documents on behalf of her manager, but neither role is a specialization of the other. Different ways of propagating privileges (delegation) should then be supported.
- In some cases the requestor's identity needs to be considered even when a role-based policy is adopted.
  For instance, a doctor may be allowed to specify treatments and access files but she may be restricted to treatments and files for her own patients, where the doctor-patient relationships is defined based on their identity.

Administrative RBAC

- Changes to RBAC policies subject to administrative policy.
- Several administrative models for RBAC: ARBAC97, SARBAC, Oracle DBMS, UARBAC, ...
- Key issue: definition of administrative domains, e.g.
  - ARBAC: admin. domain = role-based
  - UARBAC: admin. domain = attribute-based

ARBAC97: URA97 sub-model

In URA97, administrative actions can only modify the User Assignment (UA) relation.

- can_assign:
  $UEmployee : \{Student, \neg TA\} \Longrightarrow PTEmployee$
- can_revoke:
  $UEmployee : \{Student\} \Longrightarrow \neg Student$

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

ARBAC97: URA97 sub-model

In URA97, administrative actions can only modify the User Assignment (UA) relation.

- can_assign:
  $UEmployee : \{Student, \neg TA\} \Longrightarrow PTEmployee$
- can_revoke:
  $UEmployee : \{Student\} \Longrightarrow \neg Student$

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

ARBAC97: URA97 sub-model

In URA97, administrative actions can only modify the User Assignment (UA) relation.

- can_assign:
  $UEmployee : \{Student, \neg TA\} \Longrightarrow PTEmployee$
- can_revoke:
  $UEmployee : \{Student\} \Longrightarrow \neg Student$

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Fred	PTEmployee
Greg	UMember

ARBAC97: URA97 sub-model

In URA97, administrative actions can only modify the User Assignment (UA) relation.

- can_assign:
  $UEmployee : \{Student, \neg TA\} \Longrightarrow PTEmployee$
- can_revoke:
  $UEmployee : \{Student\} \Longrightarrow \neg Student$

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Fred	PTEmployee
Greg	UMember

ARBAC97: URA97 sub-model

In URA97, administrative actions can only modify the User Assignment (UA) relation.

- can_assign:
  $UEmployee : \{Student, \neg TA\} \Longrightarrow PTEmployee$
- can_revoke:
  $UEmployee : \{Student\} \Longrightarrow \neg Student$

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Fred	PTEmployee
Greg	UMember

ARBAC97: URA97 sub-model

In URA97, administrative actions can only modify the User Assignment (UA) relation.

- can_assign:
  $UEmployee : \{Student, \neg TA\} \Longrightarrow PTEmployee$
- can_revoke:
  $UEmployee : \{Student\} \Longrightarrow \neg Student$

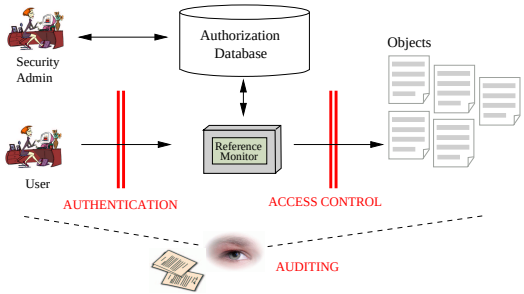
User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	PTEmployee
Greg	UMember

Summary

- We have seen several security models for confidentiality and integrity.
- Many other security models (and model types) have been proposed to address various security threats:
  - Non-interference models for deterministic systems.
  - Possibilistic models for non-deterministic systems.
  - Probabilistic models for non-deterministic systems.
  - ...
- Which model, or which combination of models, to choose depends on the particular security application and goals.

A few words about auditing

A few words about auditing



- Two components:
  - collection and organization of audit data,
  - analysis of data to discover or diagnose security violations.
- Intrusion detection:
  - Passive: offline analysis to detect possible intrusions or violation.
  - Active: real time analysis to take immediate protective response.

A few words about auditing (cont.)

- Questions:
  - How to determine what has to be audited?
  - What to look for in audit log data?
  - How to determine automatically if a violation has occurred or is occurring?
- Possible solutions:
  - Anomaly detection: the exploitation of the vulnerabilities involves abnormal use of the system.
  - Misuse detection: based on rules specifying sequences of events or observable properties of the system, symptomatic of violations.
- Auditing data needs protection from modification by an intruder.

Bibliography

- Dieter Gollmann. *Computer Security*. Wiley, 2000.
- Edward Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall, 1994.
- Matt Bishop. *Computer Security (Art and Science)*. Pearson, 2003.
- John McLean. *Security Models*, in J. Marciniak ed., *Encyclopedia of Software Engineering*, Wiley, 1994.
- Ravi Sandhu and Pierangela Samarati. *Access Control: Principles and Practice*, IEEE Communications 32(9):40–48, 1994. Various other papers by Sandhu & Samarati, and by David Ferraiolo et al.
- P. Samarati, and S. De Capitani di Vimercati, *Access Control: Policies, Models, and Mechanisms* in Foundations of Security Analysis and Design, R. Focardi, and R. Gorrieri (eds.), 2001. (Excellent survey available at `http://spdp.dti.unimi.it/papers/sam-fosad.pdf`)