Chapter 6

Access control

Read Chapter 4 (Access control) of the text book.

6.1 Introduction

We have understood identification and authentication as a way to ensure a user's identity. The next problem is to make sure that this (authenticated) user can only access the resources that he is allowed to access: "Who is allowed to do what?"?

In overview, we have

- identification
  announcing one's identity
- authentication
  proving one's identity
- authorization
  granting access to system resources
- audit
  review the system records to ensure compliance with a security policy

(see figure at page 112 of the text book)

6.2 Access control policies

An *access control policy* dictates what types of access are permitted, under what circumstances, and by whom.
Three classes of access control policies:

- Discretionary access control (DAC)
  Owner of a resource can determine who has access.
  - Each resource has an owner who determines its access policy.
  - Access rights express this policy and define which user has which rights w.r.t. the resource.
- Mandatory access control (MAC)
  System-wide policy determines who has access.
  - Each resource and user has a sensitivity label; only if they match, the user can access the resource.

  - Make sure the sensitivity label of a resource does not change when importing/exporting data.

- Role-based access control (RBAC)
  Based on the roles that users have in a system and on rules stating what accesses are allowed to users in given roles.
  - Users are assigned (multiple) roles.
  - The user's current role determines if he can access a resource.

Note: MAC, DAC and RBAC are separate concepts. A system can use only MAC, only DAC or only RBAC. Alternatively, a system may use a combination of MAC and DAC, or a combination of MAC+RBAC.

Some aspects of access control:

- Principle of least privilege
  A user (or program) should have the access privileges required for executing its task, and nothing more.
- Separation of duty
  Divide the steps of a function among different individuals. Reason: don't provide a single entity with too much power (Nick Leeson)
- Policy combinations
  Multiple policies may apply to a given resource. How to solve conflicts?
- Administrative privileges
  Access control to access control system (policies on who can modify access control rules).

6.3 Terminology

- Subject (user/program/entity)
  Three classes:
  - Owner (creator of a resource)
  - Group (users treated similarly, groups may intersect)
  - World (everybody else who can access the system)
- Object (resource)
- Access right
  - read (including copy, print – really?)
  - write (add, modify, delete)
  - execute (a program)
  - delete
  - create
  - search (list files in directory)

6.4 Discretionary Access Control

General access control model:

- Set S of subjects
- Set O of objects
- Set A of access rights
- Function $M\colon S \times O \to \mathcal{P}(A)$, the access control matrix.

Example 6.4.1 (access control matrix) *Consider S ={Alice, Bob, Carol}, O ={f1, f2, f3}. Suppose Alice can read and write file1, and Bob can read all files. Then the access control matrix looks as follows:*

	f1	*f2*	*f3*
Alice	*RW*		
Bob	*R*	*R*	*R*
Carol			

Access control matrix is in general large and sparse (many empty fields).
Decomposed into rows or into columns.
(see figure at page 117 of the text book)

- Storing the ACM per object yields an Access Control List (ACL): for each object list users and their permitted access rights. Users that are not listed have default (empty) access rights.

  Not convenient for determining the access rights of a specific user, but useful for determining which subjects have access rights to a particular object.

  Example 6.4.2 (access control list) *Continuing the above examples, we have the following access control lists:*

  f1:

subject	*rights*
Alice	*RW*
Bob	*R.*

  f2:

subject	*rights*
Bob	*R.*

  f3:

subject	*rights*
Bob	*R.*

- Storing the ACM per subject yields Capability Tickets. Each user has a number of tickets specifying authorized objects and operations for him.

  A ticket may be lent or given to others.

  Tickets can be stored

  - by the operating system (at a safe place): centralized access verification by OS
  - by the user (in form of a cryptographic token): distributed access verification by the requested resource.

  Easy to determine access rights for a user, but difficult to determine the list of users with specific access rights for a specific resource.

  Example 6.4.3 (Capability tickets) *Continuing the example above, the users can store the following capability tickets:*

  - *Alice: (f1: RW),*
  - *Bob: (f1: R, f2: R, f3: R),*
  - *Carol: ().*

Alternative: store as list of triplets (s, o, a). (see figure at page 118 of the text book).

6.5 A more refined access control model

We consider again S, O, and A. But now we divide O into four categories:

- Processes (delete, stop, wake up)
- Devices (read, write, block, unblock)
- Memory (read, write)
- Subjects (grant, delete)
  This category has to do with management of the access control matrix.

If we flag $a \in A$, e.g. (s, o, a^*), then we mean that subject s has the right to transfer access right a on object o to another subject. Moreover, he can transfer a^*.

If s has access right *owner*, e.g. (s, o, owner), then s has full capabilities over o, including transfer.

Access right *control* is similar, but s is not allowed to destroy o.

Subjects can create other subjects whom they "own". This could give a hierarchy of subjects. (see figure p119 of the text book)

6.6 The Military Lattice

An example of MAC.

Assume a set of *security levels* L, e.g. {*unclassified, confidential, secret, top-secret*}. Assume a (partial or total) order on L, e.g. unclassified < confidential < secret < top-secret

Assume a security level assignment to subjects and to objects. $l\colon O \to L$, $l\colon S \to L$.

Now we can define for a given access right a the access control matrix by: $M = \{(s, o, a) \mid l(s) \geq l(o)\}$

6.7 Role-based access control

MAC is too rigid for most organizations. DAC assumes that resources have owners while in many organizations the resources are not owned by users.

RBAC is based on the roles that users assume in a system (e.g. job function). Users can have different roles. The subject/object relation of DAC is decomposed by introducing roles. Compared to DAC, RBAC introduces an indirection:

- DAC: users $\xrightarrow{have\_access}$ objects.
- RBAC: users $\xrightarrow{member\_of}$ roles $\xrightarrow{have\_access}$ objects.

6.7.1 Basic RBAC (RBAC$_0$)

- Access decisions are based on the role the users have in the organization.
- Process of security management is more streamlined.

(See the figure on p129 of the text book.)

Consider a set of users U, a set of roles R, a set of permissions P, and a set of sessions Se. Example:

- $P = \{$ John, Pim, Lily $\}$,
- $R = \{$ manager, employee, secretary $\}$,

- $P = \{$ signcontract, printcontract $\}$,
- $S = \{s_0, s_1, s_2\}$.

User-to-role assignment: $UA \subseteq U \times R$. A user can be member of many roles, a role can have many users as member.

Example: $UA = \{$ (John, manager), (John, employee), (Dave, employee) $\}$

Permission-to-role assignment: $PA \subseteq P \times R$. A permission can be assigned to many roles, a role can have many permissions.

Example: $PA = \{$ (signcontract, manager), (printcontract, manager), (printcontract, secretary) $\}$

6.7.2 Extension: hierarchies (RBAC$_1$)

Extending this model (RBAC$_0$) to RBAC$_1$ by adding a role hierarchy. To reflect the organizational structure in RBAC. Inheritance of permission from less powerful (junior) to more powerful (senior) roles.

Hierarchy: RH $\subseteq$ R $\times$ R. Satisfying the properties of a partial order.

Example 6.7.1 (Role hierarchy) *Suppose that for a project, everyone is a* project-member*. Some of the project participants are* test-engineers*, and some are* programmers*. Both these roles have access to specific parts (tests, source code) regular project members do not have access to. The project is led by a* project-manager*. I.e., we have:*

$$\begin{tikzcd} \textit{project-manager} \arrow[d] \\ \begin{array}{c}\textit{test-engineer}\\ \textit{programmer}\end{array} \arrow[d] \\ \textit{project-member.} \end{tikzcd}$$

This gives the following role hierarchy: RH = $\{$ (test-engineer, project-member), (programmer, project-member), (project-manager, test-engineer), (project-manager, programmer) $\}$.

6.7.3 Extension: constraints (RBAC$_2$)

Extending RBAC$_0$ to RBAC$_2$ by adding constraints.
Examples of constraints:

- A user may never belong to both role r_1 and role r_2 (static).
- A user may never activate role r_1 and role r_2 in the same session (dynamic).
- Every user that is assigned to role r_1 must also be assigned to role r_2.
- At most k users may belong to role r.
- At least k users must belong to role r.
- At most k roles may get permission p.

6.7.4 Extension: constraints + hierarchy (RBAC$_3$)

Joining models RBAC$_1$ and RBAC$_2$ gives the *consolidated* model RBAC$_3$.

Chapter 7

Multi-level security

Read Chapter 10 (Multilevel security) of the text book.

Recall the access control model: we have

- subjects S,
- objects O,
- rights/permissions P.

We now extend this to include *security levels* L, with a (partial) order $\leq$ on L (which levels are supersede other levels). Now, subjects are assigned *security clearance* via $l\colon S \to L$, and objects are assigned *security classes* via $l\colon O \to L$. A subject s has permission p on object o if and only if $l(o) \leq l(s)$.

Example 7.0.2 (Security levels) *Suppose we have a company with Alice, Bob and Carol, where Carol is working on a patent draft (top secret). This could be expressed as:*

- $S = \{Alice, Bob, Carol\}$,
- $O = \{$ *thesis_alice, patent_draft* $\}$,
- $P = \{$ *read, write* $\}$,
- $L = \{high, low\}$, $low \leq high$,
- $l(Alice) = l(Bob) = l(thesis\_alice) = low$, $l(Carol) = l(patent\_draft) = high$.

Note that the requirement "s has permission p on o if and only if $l(o) \leq l(s)$" ensures that (for example) a given user s cannot read some file o if his security level is too low. But is this enough to maintain confidentiality of o?

No. Counterexample:

Example 7.0.3 (leaking information) *Suppose we have two users: s, t. t has high level clearance, s does not. User t has access to o, so t can create a copy of o at a lower security level at which s has read access and therefore confidentiality of o is not enforced.*

More precisely, let $L = topsecret > secret > confidential > restricted > unclassified$ and let $l(o) = topsecret$, $l(John) = topsecret$, $l(Sarah) = secret$. Then John reads file o, creates file o' with $l(o') = secret$, which contains a copy of o. Now Sarah *can read o', so she learns o.*

7.1 The Bell-LaPadula model

The Bell-LaPadula model (BLP, 1970's, USA-Department of Defense) is designed to prevent this kind of unwanted information disclosure.

BLP is an example of a multi-level security (MLS) model. MLS = security control to manage information with different sensitivities in an information system with users having different security clearances.

BLP is only concerned with confidentiality. Not with integrity.

The two main requirements of BLP are:

- **No read up** (or *Simple Security Property*): A subject can only read an object of less or equal security level,
- **No write down** (or **-Property*): A subject can only write an object of greater or equal security level.

Note that both the simple security property and the $\star$-property *forbid* access. Neither requires that a specific access right is granted. Both only require that specific access rights are *not* granted.

These two rules are "system wide" rules, hence they refer to a *Mandatory Access Control* policy (MAC) (see Section 6.2).

As an example, consider log files: Every user action can be logged, but only administrators may read the log file.

In addition we can define a Discretionary Access Control to fine-tune access to objects based on the wishes of the owner of the object. However, the MAC rules stated above always have precedence – if they forbid access, then access is forbidden.

- **ds-property**: A subject may grant to another subject access to an object based on the owner's discretion, but constrained by the MAC rules.

Formal description of the BLP model. The *state* of a system is fully described by a three-tuple (b, M, l), where

- **Current access set** b. $(s, o, a) \in b$ means that subject s currently has access to object o in access mode a. We consider a dynamic system, so b may change over time.
  remark: here, $s \notin O$ (subjects are not objects)
- **Access matrix** $M = s \times o$. $M_{s,o} = a$ means that s is permitted a access to o.
- **Level function** l. $l(s)$ and $l(o)$ define the classification level of s and o. $l_c(s)$ defines the (dynamic) current classification level of s. We require $l_c(s) \leq l(s)$ to allow s to log in to the system with a lower security clearance.

The three requirements of BLP formally translate to requirements on the current state b:

- **ss-property** $\forall_{(s,o,read)\in b}\ l_c(s) \geq l(o)$
- ***-property** $\forall_{(s,o,write)\in b}\ l_c(s) \leq l(o)$
- **ds-property** $\forall_{(s,o,a)\in b}\ a \in M_{s,o}$

The (dynamic) system is now formally defined by the collection of states and how the system transforms (or transits from) one state into the next state. Possible transitions are:

- **Get access**: Add a triple (s, o, a) to b.
- **Release access**: Delete a triple (s, o, a) from b.
- **Change object level**: Change $l(o)$.

- **Change current subject level**: Change $l_c(s)$.
- **Give access permissions**: Add a to $M_{s,o}$.
- **Rescind access permissions**: Delete a from $M_{s,o}$.

The BLP model allows us to formally prove that every possible state of a given system is secure (i.e. satisfies the three requirements). We will not go into details of how to prove this.

Disadvantages of the BLP model:

- Confidentiality only, no integrity.
- No management of access control (such as transfer of permissions).
- No management for downgrading objects.
- Classification creep: if you combine information from files at different security levels, the resulting file is at the highest of these levels.
- Covert channels (= transfer of information not encompassed in the model).

Example of covert channels: Suppose John wants to transfer information $b_1 b_2 \dots b_n$ (n bits) to Sarah whose security clearance is too low to read the information. Then John can use e.g. system resources to make a covert channel. He first fills the file system completely. Then (at a certain time t_1) he deletes one file if $b_1 = 1$ or does nothing if $b_1 = 0$. Next, he fills the file system again. At time t_2, he again deletes a file or does nothing, according to bit b_2. He repeats this for all bits.

7.2 The BiBa model

The BLP model only deals with confidentiality. The BiBa model only with integrity: there is data that must be visible to many users, but should only be modified by users with the correct authentication level.

The two main requirements of BiBa are:

- **Simple integrity** A subject can only modify (write) an object of less or equal integrity level, s can modify o only if $l(s) \geq l(o)$.
- **Integrity confinement** A subject can only read an object of greater or equal integrity level, s can read o only if $l(s) \leq l(o)$.

Note that the Biba model reverses the restrictions of read and write in comparison with the BLP model.

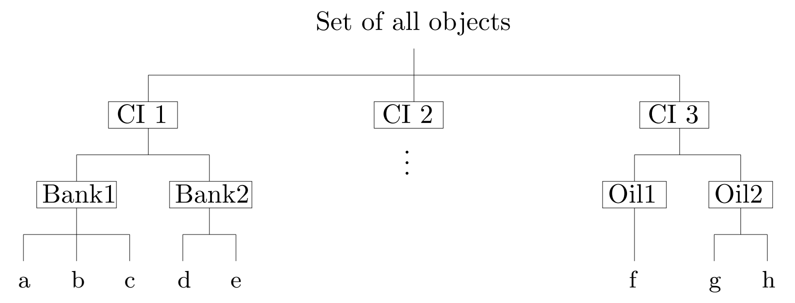
7.3 The Chinese Wall model

Setting: A subject may not have access to two classes of objects.

For instance: Financial analyst may not advice two competitors since he may learn confidential information from one that has value for the other.

Elements of the model:

- **Subjects**
- **Objects** Concerning a single corporation.
- **Dataset** (DS) All objects that concern the same corporation.
- **Conflict of Interest class** (CI) All datasets whose corporations are in competition.
- **Access rules** Rules for read and write access.



See Figure 10.6a (p318) of the text book, incompletely reprinted below.
Once a subject accesses data from one dataset, a wall is set up to protect all other datasets in the same CI class.

Two rules:

- **simple security rule**: s can read o only if
  - o is in the same DS as an object already accessed by s, **OR**
  - o belongs to a CI from which s has not yet accessed any information.
- ***-property rule**: s can write an object o only if
  - s can read o according to the simple security rule, **AND**
  - all objects that s can read are in the same DS as o.

The second rule is to disallow the following: s_1 writes confidential information from o_1 into o_2 from another CI class. s_2 who has no permission to read o_1, but who does have permission to read o_2 can now read the copy.

The second rule is quite restrictive: s cannot write at all, or s's access (both read and write) is limited to a single dataset.