Access Control Models
Part I

Murat Kantarcioglu
UT Dallas
Two main categories:

- **Discretionary Access Control Models (DAC)**
  - **Definition:** If an individual user can set an access control mechanism to allow or deny access to an object, that mechanism is a *discretionary access control (DAC)*, also called an *identity-based access control (IBAC)*.

- **Mandatory Access Control Models (MAC)**
  - **Definition:** When a system mechanism controls access to an object and an individual user cannot alter that access, the control is a *mandatory access control (MAC)* [, occasionally called a *rule-based access control.*]
Other models:

- The Chinese Wall Model – it combines elements of DAC and MAC
- RBAC Model – it is a DAC model; however, it is sometimes considered a policy-neutral model
- The Biba Model – relevant for integrity
- The Information-Flow model – generalizes the ideas underlying MAC
• DAC policies govern the access of subjects to objects on the basis of subjects' identity, objects’ identity and permissions

• When an access request is submitted to the system, the access control mechanism verifies whether there is a permission authorizing the access

• Such mechanisms are discretionary in that they allow subjects to grant other subjects authorization to access their objects at their discretion
• **Advantages:**
  – Flexibility in terms of policy specification
  – Supported by all OS and DBMS

• **Drawbacks:**
  – No information flow control (Trojan Horses attacks)
The Harrison-Ruzzo-Ullman (HRU) has introduced some important concepts:

- The notion of *authorization systems*
  - This is why we include it among the DAC models, even though the distinction between DAC and MAC was introduced much later
- The notion of *safety*

The HRU Model

To describe the HRU model we need:

- $S$ be a set of subjects
- $O$ be a set of objects
- $R$ be a set of access rights
- an access matrix $M = (M_{so})_{s \in S, o \in O}$
- the entry $M_{so}$ is the subset $R$ specifying the rights subject $s$ has on object $o$
The HRU Model – Primitive Operations

The model includes six *primitive operations* for manipulating the set of subjects, the set of objects, and the access matrix:

- **enter** $r$ into $M_{so}$
- **delete** $r$ from $M_{so}$
- **create subject** $s$
- **delete subject** $s$
- **create object** $o$
- **delete object** $o$
Commands in the HRU model have the format:

\[
\text{command } c(x_1, \ldots, x_k) \\
\text{if } r_1 \text{ in } M_{s_1, o_1} \text{ and} \\
\text{if } r_2 \text{ in } M_{s_2, o_2} \text{ and} \\
\quad \vdots \\
\text{if } r_m \text{ in } M_{s_m, o_m} \\
\text{then } op_1, \ldots, op_n \\
\text{end}
\]
The HRU Model - Commands

- The indices $s_1, \ldots, s_m$ and $o_1, \ldots, o_m$ are subjects and objects that appear in the parameter list $c(x_1, \ldots, x_k)$.
- The condition part of the command checks whether particular access rights are present; the list of conditions can be empty.
- If all conditions hold, then the sequence of basic operations is executed.
- Each command contains at least one operation.
- Commands containing exactly one operation are said *mono-operational* commands.
command create_file (s,f)
create f
enter o into M_{s,f}
enter r into M_{s,f}
enter w into M_{s,f}
end

command grant_read (s,p,f)
if o in M_{s,f}
then enter r into M_{p,f}
end
• A protection system is defined as
  – A finite set of rights
  – A finite set of commands
• A protection system is a state-transition system
The effects of a command are recorded as a change to the access matrix (usually the modified access control matrix is denoted by $M'$)

Hence the access matrix describes the state of the protection system

What do we mean by the state of the protection system?

- The state of a system is the collection of the current values of all memory locations, all secondary storage, and all registers and other components of the system
- The state of the protection system is the subset of such a collection that deals with allocation of access permissions; it is thus presented by the access control matrix
The HRU Model – States

**Definition.** A state, i.e. an access matrix $M$, is said to *leak* the right $r$ if there exists a command $c$ that adds the right $r$ into an entry in the access matrix that previously did not contain $r$. More formally, there exist $s$ and $o$ such that $r \not\in M_{so}$ and, after the execution of $c$, $r \in M'_{so}$.

*Note:* The fact that an right is leaked is not necessarily bad; many systems allow subjects to give other subjects access rights
The HRU Model – Safety of States

What do we mean by saying that a state is “safe”? 

**Definition 1**: “access to resources without the concurrence of the owner is impossible” [HRU76]

**Definition 2**: “the user should be able to tell whether what he is about to do (give away a right, presumably) can lead to the further leakage of that right to truly unauthorized subjects” [HRU76]
The problem motivating the introduction of safety can be described as follows:

“Suppose a subject $s$ plans to give subjects $s’$ right $r$ to object $o$. The natural question is whether the current access matrix, with $r$ entered into $(s’,o)$, is such that right $r$ could subsequently be entered somewhere new.”
Assume to have a protection system with the following two commands:

```plaintext
command grant_execute (s,p,f)
    if o in Ms,f
    then enter x into Mp,f
end

command modify_own_right (s,f)
    if x in Ms,f
    then enter w into Ms,f
end
```
The HRU Model – An example of “unsafe” protection system

• Suppose user Bob has developed an application program; he wants this program to be run by other users but not modified by them.

• The previous protection system is not safe with respect to this policy; consider the following sequence of commands:
  - Bob: grant_execute (Bob, Tom, P1)
  - Tom: modify_own_right (Tom, P1)

It results in access matrix where the entry $M_{Tom,P1}$ contains the w access right.
**Definition.** Given a protection system and a right \( r \), we say that the initial configuration \( Q_0 \) is *unsafe* for \( r \) (or leaks \( r \)) if there is a configuration \( Q \) and a command \( \alpha \) such that
- \( Q \) is reachable from \( Q_0 \)
- \( \alpha \) leaks \( r \) from \( Q \)

We say \( Q_0 \) is *safe* for \( r \) if \( Q_0 \) is not unsafe for \( r \).

**Alternative (more intuitive) definition.** A state of a protection system, that is, its matrix \( M \), is said to be *safe* with respect to the right \( r \) if no sequence of commands can transform \( M \) into a state that leaks \( r \).

**Theorem.** Given an access matrix \( M \) and a right \( r \), verifying the safety of \( M \) with respect to \( r \) is an undecidable problem.
The safety question is

- decidable for mono-operational protection systems
- undecidable for biconditional monotonic protection systems
  - Monotonic protections system means deletion of access rights are not allowed once it is entered in the protection system.
  - Biconditional means there is exactly two conditions in the precondition part of the commands.
- decidable for monoconditional monotonic protection systems
  - Monoconditional means there is exactly one condition in the precondition part of the commands.
The results on the decidability of the safety problem illustrate an important security principle, the *principle of economy of mechanisms*

- if one designs complex systems that can only be described by complex models, it becomes difficult to find proofs of security
- in the worst case (undecidability), there does not exist a universal algorithm that verifies security for all problem instances
Other Theoretical Models

- The take-grant model  
  (by A. Jones, R. Lipton, and L. Snyder)
- The schematic protection model  
  (by R. Sandhu)
- The typed access matrix model  
  (by R. Sandhu)
Other Models

- DAC models have been widely investigated in the area of DBMS
- The first DAC model for relational databases has been developed by Griffiths and Wide
- Several extensions to such model have been developed
DAC – additional features and recent trends

• Flexibility is enhanced by supporting different kinds of permissions
  – Positive vs. negative
  – Strong vs. weak
  – Implicit vs. explicit
  – Content-based
Positive and Negative Permissions

- Positive permissions $\rightarrow$ Give access
- Negative permissions $\rightarrow$ Deny access
- Useful to specify exceptions to a given policy and to enforce stricter control on particular crucial data items
Positive and Negative Permissions

Main Issue: Conflicts
Authorization Conflicts

- Main solutions:
  - No conflicts
  - Negative permissions take precedence
  - Positive permissions take precedence
  - Nothing take precedence
  - Most specific permissions take precedence
Weak and Strong Permissions

- Strong permissions cannot be overwritten
- Weak permissions can be overwritten by strong and weak permissions
Implicit and Explicit Permissions

• Some models support implicit permissions
• Implicit permissions can be derived:
  – by a set of *propagation rules* exploiting the subject, object, and privilege hierarchies
  – by a set of user-defined *derivation rules*
Derivation Rules: Example

• Ann can read file F1 from a table if Bob has an explicit denial for this access
• Tom has on file F2 all the permissions that Bob has
• Derivation rules are a way to concisely express a set of security requirements
• Derivation rules are often expressed according to logic programming
• Several research efforts have been carried out to compare the expressive power of such languages
• We need languages based on SQL and/or XML
Content-based Permissions

• Content-based access control conditions the access to a given object based on its content
• This type of permissions are mainly relevant for database systems
• As an example, in a RDBMS supporting content-based access control it is possible to authorize a subject to access information only of those employees whose salary is not greater than 30K
Content-based Permissions

Two are the most common approaches to enforce content-based access control in a DBMS:

- by associating a predicate (or a Boolean combination of predicates) with the permission
- by defining a view which selects the objects whose content satisfies a given condition, and then granting the permission on the view instead of on the basic objects
DAC models - DBMS vs OS

- Increased number of objects to be protected
- Different granularity levels (relations, tuples, single attributes)
- Protection of logical structures (relations, views) instead of real resources (files)
- Different architectural levels with different protection requirements
- Relevance not only of data physical representation, but also of their semantics
The Trojan Horse

Process P

read O1
write O2

O1
(ada,r,O1)

O2
(ada,r,O2), (ada,w,O2), (bob,r,O2)
The Trojan Horse

- DAC models are unable to protect data against Trojan Horses embedded in application programs
- MAC models were developed to prevent this type of illegal access
MAC

- MAC specifies the access that subjects have to objects based on subjects and objects classification
- This type of security has also been referred to as *multilevel security*
- Database systems that satisfy multilevel security properties are called multilevel secure database management systems (MLS/DBMSs)
- Many of the MLS/DBMSs have been designed based on the Bell and LaPadula (BLP) model
Bell and LaPadula Model

Elements of the model:

- **objects** - passive entities containing information to be protected
- **subjects**: active entities requiring accesses to objects (*users, processes*)
- **access modes**: types of operations performed by subjects on objects
  - read: reading operation
  - append: modification operation
  - write: both reading and modification
Bell and LaPadula Model

- Subjects are assigned clearance levels and they can operate at a level up to and including their clearance levels
- Objects are assigned sensitivity levels
- The clearance levels as well as the sensitivity levels are called access classes
BLP Model - access classes

- An access class consists of two components:
  - a security level
  - a category set

- The security level is an element from a totally ordered set - example
  - \{Top Secret (TS), Secret (S), Confidential (C), Unclassified (U)\} where \(TS > S > C > U\)

- The category set is a set of elements, dependent from the application area in which data are to be used - example
  - \{Army, Navy, Air Force, Nuclear\}
Access class \( c_i = (L_i, SC_i) \) dominates access class \( c_k = (L_k, SC_k) \), denoted as \( c_i \geq c_k \), if both the following conditions hold:

- \( L_i \geq L_k \)  
  The security level of \( c_i \) is greater or equal to the security level of \( c_k \)
- \( SC_i \supseteq SC_k \)  
  The category set of \( c_i \) includes the category set of \( c_k \)
• If \( L_i > L_k \) and \( SC_i \subseteq SC_k \), we say that \( c_i \) strictly dominates \( c_k \)
• \( c_i \) and \( c_k \) are said to be incomparable (denoted as \( c_i < > c_k \)) if neither \( c_i \geq c_k \) nor \( c_k \geq c_i \) holds
Access classes

\[ c_1 = (TS, \{\text{Nuclear, Army}\}) \]
\[ c_2 = (TS, \{\text{Nuclear}\}) \]
\[ c_3 = (C, \{\text{Army}\}) \]

- \( c_1 \geq c_2 \)
- \( c_1 > c_3 \) \( (TS > C \text{ and } \{\text{Army}\} \subset \{\text{Nuclear, Army}\}) \)
- \( c_2 <> c_3 \)
The state of the system is described by the pair \((A, L)\), where:

- \(A\) is the set of current accesses: triples of the form \((s, o, m)\) denoting that subject \(s\) is exercising access \(m\) on object \(o\) - example (Bob, \(o_1\), read)

- \(L\) is the level function: it associates with each element in the system its access class

Let \(O\) be the set of objects, \(S\) the set of subjects, and \(C\) the set of access classes

\[ L : O \cup S \rightarrow C \]
BLP Model - Axioms

- Simple security property (*no-read-up*)
  a given state \((A, L)\) satisfies the simple security property if for each element \(a = (s,o,m) \in A\) one of the following condition holds:
  1. \(m = \text{write}\)
  2. \(m = \text{read or } m = \text{read\&write and } L(s) \geq L(o)\)

- Example: a subject with access class \((C, \{\text{Army}\})\) is *not allowed* to read objects with access classes
  \((C, \{\text{Navy, Air Force}\})\) or \((U, \{\text{Air Force}\})\)
BLP Model - Axioms

- The simple security property prevents subjects from reading data with access classes dominating or incomparable with respect with the subject access class.
- It therefore ensures that subjects have access only to information for which they have the necessary access class.
BLP Model - Axioms

• Star (*) property (*no-write-down*)
  a given state \((A, L)\) satisfies the *-property if for each element \(a = (s, o, m) \in A\) one of the following condition holds
  1. \(m = \text{read}\)
  2. \(m = \text{write}\) and \(L(o) \geq L(s)\)
  3. \(m = \text{read\&write}\) and \(L(o) = L(s)\)

• Example: a subject with access class \((C,\{\text{Army, Nuclear}\})\) is not allowed to write data into objects with access class \((U, \{\text{Army, Nuclear}\})\)
BLP Model - Axioms

- The *-property has been defined to prevent information flow into objects with lower-level access classes or incomparable classes.
- For a system to be secure both properties must be verified by any system state.
Summary of access rules:

- **Simple security property**: A subject has read access to an object if its access class dominates the access class of the object;
- ***-Property**: A subject has append access to an object if the subject's access class is dominated by that of the object
Problem

• Colonel has (Secret, {Nuclear, Army}) clearance
• Major has (Secret, {Army}) clearance
• The Colonel needs to send a message to the Major. The Colonel cannot write a document that has access class (Secret, {Army}) because such a document would violate the \*-property
• To address this problem the model provides a mechanism; each subject has a maximum access class and a current access class
• A subject may change its access class; the current access class must however be dominated by the maximum access class
An Example of Application
The DG/Unix B2 System

- B2 is an evaluation class for secure systems defined as part of the Trusted Computer System Evaluation Criteria (TCSEC), known also as the Orange Book
- DG/Unix Provides mandatory access controls
  - MAC label identifies security level
  - Default labels, but can define others
- Initially
  - Processes (users) assigned MAC label of parent
    - Initial label assigned to user, kept in Authorization and Authentication database
  - Object assigned label at creation
    - Explicit labels stored as part of attributes
    - Implicit labels determined from parent directory
Directory Problem

- Process $p$ at access class MAC_A tries to create file `/tmp/x`
- `/tmp/x` exists but has access class MAC_B
  - Assume MAC_B $> MAC_A$ (MAC_B dominates MAC_A)
- Create fails
  - Now $p$ knows a file named $x$ with a higher label exists
- Fix: only programs with same MAC label as directory can create files in the directory
  - This solution is too restrictive
Multilevel Directory

- Directory with a set of subdirectories, one per label
  - Not normally visible to user
  - $p$ creating /tmp/x actually creates /tmp/d/x where $d$ is directory corresponding to MAC_A
  - All $p$’s references to /tmp go to /tmp/d
- The directory problem illustrates an important point:
  \[\text{Sometimes it is not sufficient to hide the contents of objects. Also their existence must be hidden.}\]
Bell and LaPadula Model

• It is a significant model and it has been used in both OS and DBMS

• Some criticisms:
  – Only dealing with confidentiality, not with integrity
  – Containing covert channels (see the textbook for more discussions on this)