Part III

Access Control Fundamentals
3.1 Authentication and Access Control

3.2 Discretionary Access Control (DAC)
- Examples for DAC
- Access Control Matrix

3.3 Mandatory Access Control (MAC)
- Introduction to Mandatory Access Control
- The Bell-La Padula Model
- The Biba Model
Access to the System

- **A system can protect itself in two ways:**
  1. **It can limit who can access the system**
     - This requires the system to implement a two-step process of
       - **identification** (asking you who you are) and
       - **authentication** (asking you to prove it)
  2. **It can limit what user can do** after they have accessed the system
     - This requires the system to implement **access control** mechanisms
3.1 Authentication and Access Control
3.2 Discretionary Access Control (DAC)
3.3 Mandatory Access Control (MAC)

Authentication Techniques

- **Weak authentication**
  - Passwords
    - Despite of drawbacks, passwords are the most common authentication technique
    - If properly used, passwords provide very effective user authentication

- **Strong authentication**
  - Challenge-response protocols
    - E.g., user should demonstrate that he is in possession in some key

- **Biometric authentication**
  - Techniques based on biometric data
    - Fingerprint, iris, face recognition, voice
Exercise for Authentication

- Authentication protocols has been discussed very extensively in other lectures (e.g., System Security I)
  - We will not discuss them extensively in this course

**Exercise 4 in Lab**

- How and which authentication techniques are implemented in the common operating systems (Windows, Unix)
- Possible attacks on authentication
Access Control: Notion

- **The primary purpose of security mechanisms in a system is to control access to resources**
  - Resources: Files, memory areas, processor time, devices, database records

- **Some history**
  - Early systems had no internal access control
    - Any user could access any file simply by knowing its name
  - Access control became a more serious issue with the emergence of disk storage, on which files of many users could be stored (before the days of network and interactive computing)
  - Controlling access to disk files was probably the first widespread computer security concern
    - For the first time the system, rather than the operator, was required to enforce access control
Access Control: Definition

- **Access control (AC) is the process of**
  - mediating every request to resources and data maintained by a system, and
  - determining whether the request should be granted or denied

- **The development of an AC system requires**
  - definition of regulations according to which access is to be controlled, and
  - their implementation as functions executable by a computer system

- **Important question: Who defines the regulations?**
  - E.g., the system, some user, administrator, owner of some resource
The formalization phase allows the definition of a formal model making it possible to **define and prove security properties** that systems enforcing the models will enjoy.

Therefore, by proving that
- **the model is "secure"**, and
- that the mechanism **correctly implements the model**
we can argue that **the system is "secure"**
Access control policies can be grouped into three main classes:

- **Discretionary (or authorization-based) policies**
  - Users, on their discretion, can specify, to the system who can access their objects (e.g., files).

- **Mandatory policies**
  - The system controls access based on mandated regulations determined by a central authority.

- **Role-based policies**
  - Access to some resource depends on the role that a user have within the system and on rules stating what access are allowed to users in given roles.

- **Administrative policies**
  - Define **who** can specify authorization/rules governing access control.
  - Usually coupled with (or included in) discretionary and role-based policies.
3.1 Authentication and Access Control

3.2 Discretionary Access Control (DAC)
- Examples for DAC
- Access Control Matrix

3.3 Mandatory Access Control (MAC)
- Introduction to Mandatory Access Control
- The Bell-La Padula Model
- The Biba Model
Discretionary Access Control (DAC)

- **Discretionary Access Control (DAC) implements discretionary policies**
  - DAC base access on the identity of the subject and identity of the object involved
    - Access control is left to the discretion of the owner
    - The owner of the object constrains *who* and *how* (read, write, execute) can access it
  - DAC centers around the concept of users having control over system resources
  - Each system resource is assigned ownership by one or more entities
  - DAC is also called *identity-based access control* (IBAC) or *authorization-based access control*
Discretionary Access Control (cntd.)

- **Advantages**
  - Simplicity, flexibility and ease of implementation

- **Drawback**
  - There is no formal assurance concerning the flow of information
  - Trojan horse threads

- **Examples for DAC**
  - Passwords for file access
  - Capability list
  - Owner/Group/Other
  - Access control lists (ACL)
Example for DAC: Passwords for File Access

- A password-based access scheme is used to protect files by assigning to each file a password by its owner.
  - Only users who know the password are able to access the file.
    - Note: This password has nothing to do with any password the user might need to log into the system.
    - Usually there must exist two passwords for each file: one for controlling reading, and one for controlling writing.
  - In a system with thousand of files password-based access schemes are unsuitable.
  - This was one of the primary protection mechanisms in the early systems.
Problems with the Passwords for File Access

- **Management problem**
  - In a large organization where users come and go daily, a password-based protection scheme for all files becomes impossible to manage.

- **Revocation problem**
  - There is no way to revoke one user's access to the file (by changing the password) without revoking everyone's access.

- **Tracking problem**
  - There is no way for the system to keep track of who has access to the file, since passwords are distributed manually without system knowledge.

- **Possibility of unauthorized use**
  - Passwords for file access tend to be embedded as character strings within programs that need to use the files; so one user's program can be run by another person who does not necessarily know the passwords for all the files the program needs.

- **Remembering problem**
  - Requiring the user to remember a separate password for each file is an unreasonable burden.
3.1 Authentication and Access Control

3.2 Discretionary Access Control (DAC)

3.3 Mandatory Access Control (MAC)

Example for DAC: Capability List

- **Notion of capability**
  - A capability list (or a capability, for short) is like a access-key to a specific object, along with the mode of access (read, write, execute)
  - A subject possessing a capability may access the object in the specified mode

- **Access control with capabilities**
  - At the highest level in the system, where we are concerned with users and files, the system holds a capability list for every subject (e.g., user)
    - User cannot add capabilities to this list
    - Exception: If a user creates a new file, the capability for this file will be added in the list
    - User might be allowed to give access to files by passing copies of their own capabilities to other users
    - Users might be able to revoke access to their own files by taking away capabilities from others (revocation can be difficult to implement)
Disadvantages and Advantages of Capabilities

- **Disadvantage: Management problem**
  - The system must maintain a list for each user that may contain hundreds or thousands of entries.
  - When a file is deleted, then the system has to update every capability list (for each user).
  - Answering a simple question as a "who has access to this file?" requires the system to undergo a long search through every users capability list.

- **Successful use of capabilities**
  - At lower levels in the system, where capabilities provide the underlying protection mechanism and not the user-visible access control scheme.
    - Low-level use of capabilities used by hardware (will be discussed later).
    - Low-level use of capabilities used by software (will be discussed later).
Example for DAC: Owner/Group/Other Technique

- Few bits of access control information are attached to each file
- These bits specify the access modes for different classes of users

  - Usually there are no more than four classes of users (owner of the file, users belonging to the owners group or project, special system users, and the rest of the world)
  - In a large system, where users are grouped by project or department, most access control needs are satisfied by this technique

```
filename.txt  1  1  1  1  1  0  1  0  0
R = read
W = write
E = execute
```
Advantages and Disadvantages of Owner/Group/Other Technique

- **Advantages**
  - Effective, simple and very common discretionary access control scheme
    - Implemented in Unix, DECs RSX and VMS, and many other systems

- **Disadvantages/Problems**
  - The technique fails apart when access across specific groups are required
  - Inability of the technique to specify access rights for an individual user
    - There is no way for user X to specify that ONLY user Y, and nobody else, should have access to the file, unless there is a group defined in the system to which only X and Y belong
An access control list (ACL) is placed on each file.
The ACL identifies the individual users or groups who may access the file.

**ACL for file F1**

- Stefan.Crypto: read
- * . Crypto: read
- Marcel: no access
- * .: read

**ACL for file F2**

- Sadeghi: read
- Palacios: no access
- * .: no access
- /*: read

**ACL for file F3**

- Hans.Trust: read
- Marcel. Trust: read
- * . Trust: no access
- * . Crypto: read

n = no access
Marcel has no access to F1 unless he is in the crypto group.
Advantages and Disadvantages of ACLs

**Advantages**
- Because all access control information for a file is stored in one place and is clearly associated with the file,
  - identifying who has access to a file can be done very efficiently
  - adding or deleting names to the list can be done very efficiently

**Disadvantages/Problems**
- Performance
  - The ACL has to be scanned each time any user wants to access the file (impact on systems where large number of files are opened in relatively short time)
- Storage management
  - Maintaining a variable-length list for each file results in either a complex directory structure or wasted space for unused entries
  - Problem only for systems having huge number of very small files (e.g., Unix)
Different DAC policies and models have been proposed in the literature.

The most popular discretionary model is the **Access Matrix Model**.

Core of the Access Matrix Model is the **Access Control Matrix (ACM)**

- **Rows**: Set of subjects $S$ in the system
- **Columns**: Set of objects $O$ in the system

Matrix entry $ACM[i,j]$: The rights that a subject $s_i$ has on object $o_j$

- This is a subset of the set of rights $R$ that is defined in the system
- E.g., $R = \{\text{read, write, execute}\}$

The Access Control Matrix can be

- **static** (entries do not change in the course of time)
- **dynamic** (entries change in the course of time)
Example for an Access Control Matrix

\[ S = \{\text{Process1, Process2}\} \]
\[ O = \{\text{File1, File2, Process1, Process2}\} \]
\[ R = \{\text{read, write, execute, append, own}\} \]

The subject that has this right can grant or take rights to the other subjects (and to itself)

Subject is allowed to attach some data (or a file) to the object (integrity concern!)

Subject is allowed to execute the object

Subject is allowed to write into the object i.e., alter the object (integrity concern!)

Subject is allowed to read the object (confidentiality concern!)

Note: \( S \subseteq O \)

The Access Control Matrix could be:

<table>
<thead>
<tr>
<th></th>
<th>File1</th>
<th>File2</th>
<th>Process1</th>
<th>Process2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process1</td>
<td>\textit{read, write, own}</td>
<td>\textit{read}</td>
<td>\textit{read, write, own, execute}</td>
<td>\textit{write}</td>
</tr>
<tr>
<td>Process2</td>
<td>\textit{append}</td>
<td>\textit{read, write, own}</td>
<td>\textit{read}</td>
<td>\textit{read, write, own, execute}</td>
</tr>
</tbody>
</table>
ACM Example: Description

- File entitlements
  - Process1 can read or write to File1 and can only read File2
  - Process2 can append data to File1 and can read or write to File2

- Process entitlements
  - Process1 can communicate with Process2 by writing to it
  - Process2 can read from Process1

- Note: The "own" right gives the creator of an object special rights (i.e., grant new rights to another subject or remove existing rights and delete rights)
  - Process 1 owns File1 and itself ⇒ can change the rights on them!
  - Process 2 owns File2 and itself ⇒ can change the rights on them!
3.1 Authentication and Access Control

3.2 Discretionary Access Control (DAC)
- Examples for DAC
- Access Control Matrix

3.3 Mandatory Access Control (MAC)
- Introduction to Mandatory Access Control
- The Bell-La Padula Model
- The Biba Model
Mandatory Access Control (MAC)

- **Idea of MAC:** A system mechanism (e.g., the operating system) controls access to an object and an individual user cannot alter that access

  - Neither the subject nor the owner of the object can determine whether access is granted
  - The system mechanism will check information associated with both the subject and the object to determine whether the subject should access the object
  - Rules describe the conditions under which access is allowed
  - Mandatory policies are also called rule-based policies
  - German: "Systembestimmte Zugriffsregeln"

- **Example**
  - Sometimes, the law allows the fiscal authorities to have access on your bank account record
Mandatory Access Control (MAC) (cntd.)

- **Goal of MAC:**
  - Preserve confidentiality and integrity of information
  - Prevent some types of Trojan horse attacks
  - Prevent that a user can change security attributes

- **Types of mandatory policies**
  - **Secrecy policies:** controls the direct and indirect flows of information to the purpose of preventing leakages to unauthorized subjects
  - **Integrity policies:** controls the direct and indirect flows of information to the purpose of preventing unauthorized altering of objects

- **Mandatory control can be used in conjunction with discretionary controls**
  - Serve as an additional and stronger restriction on access
MAC and Multilevel Security (MLS)

- Many different MAC schemes have been defined
  - Nearly all are variants of the U.S. Department of Defences multilevel security policy
  - It is difficult to discuss mandatory controls apart from multilevel security

- Idea of multilevel security (MLS)
  - Each object in the system (e.g., a file) possesses a classification
  - Each subject in the system (e.g., a user) possesses a clearance
  - In order to determine whether a subject is allowed to access an object, the subjects clearance is compared to the objects classification
  - Both classification and clearance are sometimes denoted as access classes
Both classification and clearance are made up of two components:

- A *security level*, that is an element of linearly ordered set
  - E.g., `Unclassified < Confidential < Secret < TopSecret`

- A set of one or more *categories* (also called compartments), consisting of names of the thematic areas to which an object may belong
  - E.g., `\{Nato, Nuclear, Navy\}` or `\{Administration, Laboratory, Surgery\}`
  - Categories are independent of each other and are not ordered

**Example**

- An object $o_i$ could have a classification $(Confidential, \{Nato\})$
- A subject $s_j$ could have a clearance $(Secret, \{Nato, Navy\})$
Mathematical Relationships

When categories and levels are combined, four relationships are possible between two access classes

1. **The first access class dominates (dom) the second**
   - That is, the level of the first is greater than the level of the second, and the category set of the first contains all the categories of the second
   - Example: \( AC_1 = (\text{Secret}, \{\text{Nato}, \text{Navy}\}) \), \( AC_2 = (\text{Confidential}, \{\text{Nato}\}) \)
     \[ \Rightarrow AC_1 \text{ dom } AC_2 \]
     because \( \text{Secret} \geq \text{Confidential} \) and \( \{\text{Nato}\} \subset \{\text{Nato}, \text{Navy}\} \)

2. **The second access class dominates the first**

3. **The access classes are equal**
   - Special case where both 1. and 2. are true
   - Example: \( AC_1 = (\text{Secret}, \{\text{Nato}, \text{Navy}\}) \), \( AC_2 = (\text{Secret}, \{\text{Nato}, \text{Navy}\}) \)
     \[ \Rightarrow AC_1 = AC_2 \]
     because \( AC_1 \text{ dom } AC_2 \) and \( AC_2 \text{ dom } AC_1 \)

4. **The access classes are disjoint and cannot be compared**
   - The first contains a category not in the second, and the second contains a category not in the first
   - Example: \( AC_1 = (\text{Secret}, \{\text{Nato}, \text{Navy}\}) \) and \( AC_1 = (\text{Secret}, \{\text{Nuclear}, \text{Navy}\}) \) are disjoint classes
Example: "Dominance" Relation

- Set of security levels
  \[ L = \{ TS, S \} \text{ with } TS > S > C > U \]
- Set of categories
  \[ C = \{ Nuclear, Army \} \]
- Possible access classes (compartments)
  \[
  \begin{align*}
  AC_1 &= (S, \{\emptyset\}) \\
  AC_2 &= (S, \{Army\}) \\
  AC_3 &= (S, \{Nuclear\}) \\
  AC_4 &= (S, \{Army, Nuclear\}) \\
  AC_5 &= (TS, \{\emptyset\}) \\
  AC_6 &= (TS, \{Army\}) \\
  AC_7 &= (TS, \{Nuclear\}) \\
  AC_8 &= (TS, \{Army, Nuclear\})
  \end{align*}
  \]
- Lattice of access classes (see figure right)
  - The "Dominance" relation is represented as →
  - Dominance is transitive
Mathematical Models for Multilevel Security

- **Protection of confidentiality of information: Bell La Padula Model (BLP)**
  - Addresses the confidentiality problem by mathematically describing the "read" and "write" restrictions based on confidentiality (secrecy) access classes

- **Protection of integrity of information: Biba Model**
  - Addresses the modification problem by mathematically describing read and write restrictions based on integrity access classes
The Bell-La Padula (BLP) Model

- First mathematical model of a multilevel secure computer system
  - Developed and formalized by David Bell and Leonard La Padula; Published 1973
  - Has influenced the development of many other models
  - Has strongly influenced the development of computer security technologies (e.g., UNIX, Multics)

- Controls the information flow
  - The flow model in BLP is motivated by the confidentiality of information
  - A confidentiality policy prevents the unauthorized disclosure of information; unauthorized alternation of information is secondary

- Suited for modeling strong hierarchical systems
  - It was designed for the military in USA
**Basic Security Theorem of the BLP Model**

- **BLP Basic Security Theorem (rules of the model)**

  1. **Simple security property** ("no-read-up"-principle):
     
     A subject $s$ is allowed a "read" access to an object $o$ only if the access class of the subject dominates the access class of the object, i.e., $AC(s) \supset AC(o)$, and $s$ has discretionary read access to $o$.

  2. **Confinement property (or *-property)** ("no-write-down"-principle):
     
     A subject $s$ is allowed a "write" access to an object $o$ only if the access class of the subject is dominated by the access class of the object, i.e., $AC(o) \supset ACs$, and $s$ has discretionary write access to $o$. 
3.1 Authentication and Access Control
3.2 Discretionary Access Control (DAC)
3.3 Mandatory Access Control (MAC)

Introduction to Mandatory Access Control
The Bell-La Padula Model
The Biba Model

Bell-LaPadula Model: An Example

- **Simple example:** In a system two security levels has been defined \{Unclassified, Secret\} with Secret > Unclassified. No categories has been defined.
  - Subjects and objects in the system: two processes (subjects) and two files (objects)
    - One file and one process are Unclassified
    - Other file and other process are SECRET

- The information flow is in the picture above is represented as an arrow →
Contamination in the Case of Absence of the *-Rule

- **System**
  - Simple system from the last slide
  - Access control model: Just the first rule ("no-read-up") of BLP

- **Situation**
  - The *Unclassified* process cannot read the *Secret* file (Rule 1)
  - Both processes can read the *Unclassified* file
  - But, the *Secret* process may read information out of the *Secret* file and writes it into the *Unclassified* file (write-down)
    - Information flow from secret file to unclassified file
    - A violation of the security policy! ⇒ Need of Rule 2 ("no-write-down")
  - The write-down problem is a continual source of frustration, because even the best technical solutions to the problem adversely affect the usability of the system
3.1 Authentication and Access Control
3.2 Discretionary Access Control (DAC)
3.3 Mandatory Access Control (MAC)

Introduction to Mandatory Access Control

The Bell-La Padula Model

The Biba Model

Summary: Allowed Information Flow for Secrecy

\[
SL = \{L_1, L_2, L_3, L_4\} \quad \text{Security levels for secrecy}
\]

\[
L_1 > L_2 > L_3 > L_4
\]

\[PID1, PID2, ..., \text{ - Process ID’s}\]

Rules: ”No-read-up” and”no-write-down”
Biba Model for Integrity

- **Goal of the model:** Protection of integrity of information
  - Developed 1977 by Kenneth Biba
  - Addresses the modification problem by mathematically describing "read" and "write" restrictions based on integrity access classes

- **All concepts of multilevel security (MLS) are used**
  - An integrity access class is a pair consisting of integrity level and set of categories
    - Integrity levels are defined as an ordered set, e.g., `Unclassified < Integr_Sensitive < High_Integrity`
    - Categories are unordered set of thematic areas
  - The "Dominance" relation `dom` is the same as in other MLS models
  - Remind:
    - For subjects, the access class is called "classification"
    - For objects, the access class is called "clearance"
Basic Security Theorem of the Biba Model

- **Biba Basic Security Theorem (rules of the model)**

  1. **Simple integrity** ("no-write-up"-principle):

     A subject $s$ is allowed a "write" access to an object $o$ only if the integrity access class of the subject dominates the integrity access class of the object, i.e., $AC(s) \text{ dom } AC(o)$, and $s$ has discretionary read access to $o$.

  2. **Integrity confinement (or *-property)** ("no-read-down"-principle):

     A subject $s$ is allowed a "read" access to an object $o$ only if the integrity access class of the subject is dominated by the integrity access class of the object, i.e., $AC(o) \text{ dom } ACs$, and $s$ has discretionary write access to $o$. 
Biba Model: An Example

- **Simple example:**
  - In a system two integrity levels has been defined \{High\_integrity, Low\_integrity\} with High\_integrity > Low\_integrity. No categories has been defined.
  - Subjects and objects in the system: two processes (subjects) and two files (objects)
    - One file and one process are High\_integrity
    - Other file and other process are Low\_integrity
  - The information flow is in the picture above is represented as an arrow →
Biba Model: Some Notes

- **Note**: Subjects and objects in the system can have different secrecy access class and integrity access class
  - Example:
    - A file can have low secrecy access class (e.g., unclassified). So, everybody can read that file.
    - The same file can have the highest integrity class (e.g., top_integrity). So, only subjects having the highest integrity clearance can modify this file.

- **Importance of the both rules**
  - Rule 1 is the logical integrity write-up restriction that prevents contamination of high-integrity data.
  - What is the role of Rule 2 (integrity confinement property)?
Contamination in the Case of Absence of the *-Rule

- **System**
  - Simple system from the last slide
  - Access control model for integrity: Just the first rule ("no-write-up") of Biba

- **Situation**
  - The \textit{Low\_integrity} process is not allowed to write into and contaminate a \textit{High\_integrity} file (Rule 1)
  - But, the \textit{High\_integrity} process may receive \textit{Low\_integrity} data and may write that data into the \textit{High\_integrity} file
    - Information flow from \textit{Low\_integrity} file to \textit{High\_integrity} file
    - A violation of the integrity policy! \Rightarrow Need of Rule 2 ("no-read-down")
Summary: Allowed Information Flow for Integrity

\[ SL = \{ L_1, L_2, L_3 \} \]

Security levels for integrity \( L_1 > L_2 > L_3 \)

\( PID_1, PID_2, \ldots \) - Process ID's

Rules: "No-write-up" and "no-read-down"
Some Remarks on Integrity Control

- **Mandatory integrity control mechanisms are very important in operating systems**
  - The primary application has been to avoid modification of some certain system programs and system databases that are important to the operation of the system and yet do not involve information with any secrecy content
  - For example, the list of user allowed to access the system might not be secret, but it must be protected from modification by untrusted software
  - This protection must be stronger that the discretionary protection provided for user files (mandatory integrity mechanisms provide that type of protection!)

- **Mandatory integrity control mechanisms are also very important in commercial systems**
  - E.g., who is allowed to change records in an Aircompany-database with flights?
3.1 Authentication and Access Control
3.2 Discretionary Access Control (DAC)
3.3 Mandatory Access Control (MAC)

Introduction to Mandatory Access Control

The Bell-La Padula Model

The Biba Model

Literature


  Technical Report MTR-2547 v2, MITRE, 1973

- [Biba] K. J. Biba: "Integrity Considerations for Secure Computer Systems"