Information Security
CENG418
week-8

- Access Control

Access Control

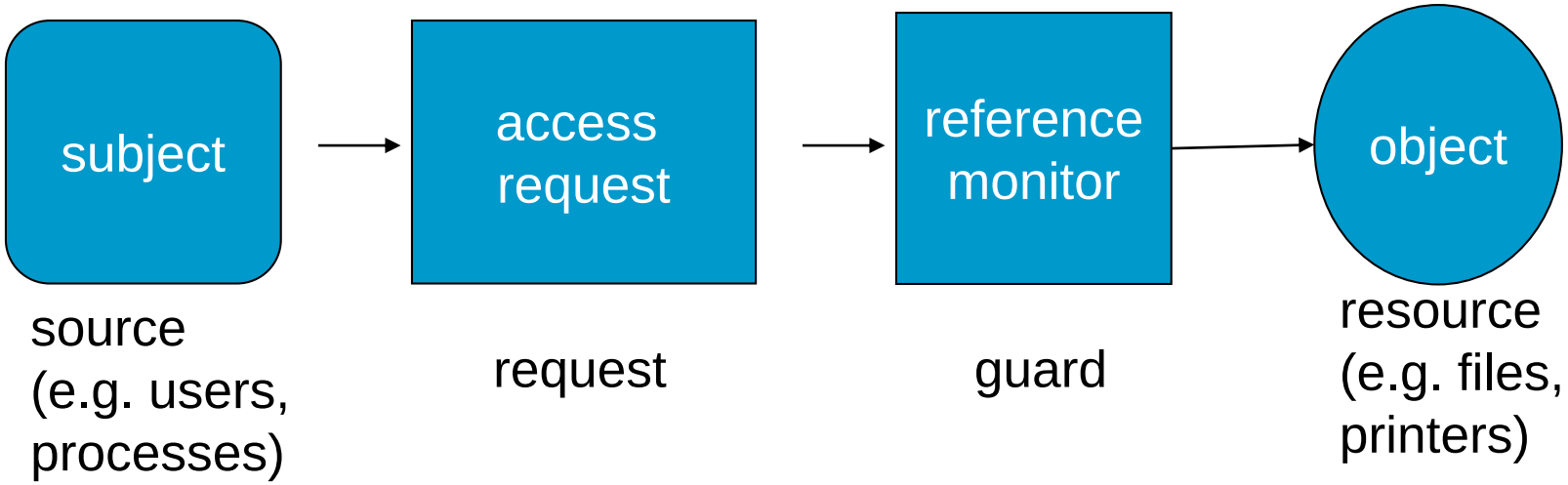
- You have now logged on to the system.
- You create new files and want to protect them.
- Some of them may be public, some only intended for a restricted audience, and some may be private.
- You need a language for expressing your intended access control policy and you need mechanisms that enforce access control.

Access Control

- Today we will start to cover *Access Control*
  - material is from Gollmann's Computer Security book (Chapter 5) (most slides are from his course too)
- A bit theoretic concept
  - because it is more than *read, write, execute*
- But still an operating system related concept
  - the resources are to be accessed but by whom?
  - access control paradigms center around this question

A Model for Access Control

There is an active entity, a *subject* or *principal*, accessing a passive *object* with some specific *access operation*, while a *reference monitor* grants or denies access.



Definition: A *principal* is an entity that can be granted access to objects or can make statements affecting access control decisions. A *subject* is an active entity within an IT system.

Basic Terminology

- Subject/Principal: active entity – user or process
- Object: passive entity – file or resource
- Access Modes:
  - Observe; look at the contents of an object
  - Alter; change the contents of an object
- Access operations: read, write, ...
  - Access operations vary from basic memory/file access to method calls in an object-oriented system.
  - Comparable systems may use different access operations.

Authorization

- Access control decision is actually an *authorization* decision
- if *o* is an object, authorization answers the question "**Who is trusted to access *o*?**"

Simple analogy

- Consider a paper-based office in which certain documents should only be read by certain individuals
- We could implement access control by
  - storing documents in filing cabinets
  - issuing keys to the relevant individuals for the appropriate cabinets

Simple analogy

- The reference monitor is the set of locked filing cabinets
  - An access request (an attempt to open a filing cabinet) is granted if the key fits the lock (and denied otherwise)

Options for Focusing Control

- Subjects and objects provide a different focus of control
  - What is the subject allowed to do?
  - What may be done with an object?
- Traditionally, multi-user operating systems manage files and resources, i.e. objects
  - Access control takes the second approach
- Application oriented IT systems, like DBMSs, offer services for the user and control the actions of subjects.

Elementary access operations

- On the most elementary level, a subject may
  - observe an object, or
  - alter an object.
- We refer to observe and alter as access modes.
- The four Bell-LaPadula (BLP) access rights:
  - execute
  - read
  - append, also called ***blind write***
  - write

Bell-LaPadula - BLP Access Rights and Modes

- Mapping between access rights and access modes.

	execute	append	read	write
observe			X	X
alter		X		X

- Write access usually includes read access. Hence, the write right includes observe and alter mode.
- Few systems implement append. Allowing users to alter an object without observing its content is rarely useful (exception: audit log).
- A file can be used without being opened and read. **Example:** use of a cryptographic key. This can be expressed by an execute right that includes neither observe nor alter mode.

Unix

- Access control expressed in terms of three operations:
  - read: from a file
  - write: to a file
  - execute: a file

- Applied to a directory, the access operations take different meanings:
  - read: list contents
  - write: create, delete or rename files in the directory
  - execute: search directory

- These operations differ from the **Bell-LaPadula model**. Unix write access does not imply read access
- Unix controls who can create and delete files by controlling the write access to the file's directory

Windows NT Family

- The *standard permissions* include:
  - delete;
  - write DACL (modify access control list);
  - write owner (modify owner of a resource).

- Terminology for access right manipulation
  - grant / revoke – if done by other party
  - assert / deny – if done by the owner itself

Ownership

- Ownership is an aspect often considered in access control rules.
- When a new object is created, in many operating systems the subject creating the object becomes its owner.

Who Sets the Policy?

Security policies specify how subjects access objects. There are two mechanisms for deciding who is in charge of setting the policy:

- The owner of a resource decides who is allowed access. Such policies are called discretionary as access control is at the owner's discretion.
- A system wide policy decides who is allowed access. Such policies are called mandatory.

Access Control Structures

- Requirements on access control structures:
  - ***The access control structure should help to express your desired access control policy.***
  - ***You should be able to check that your policy has been captured correctly.***
- Access rights can be defined individually for each combination of subject and object.
- For large numbers of subjects and objects, such structures are cumbersome to manage.
  - **Intermediate levels of control are preferable**.

Access Control Matrix

- S … set of subjects
- O … set of objects
- A … set of access operations
- Access control matrix: $M = (M_{so})_{s \in S, o \in O}$, $M_{so} \subseteq A$; M_{so} specifies the operations subject s may perform on object o.

	bill.doc	edit.exe	fun.com
Alice	{}	{exec}	{exec,read}
Bob	{read,write}	{exec}	{exec,read,write}

Access Control Matrix ctd.

- The access control matrix is
  - an abstract concept
  - not very suitable for direct implementation
    - Management of the matrix is likely to be extremely difficult if there are ten thousands of files and hundreds of users (resulting in millions of matrix entries)
    - The matrix is likely to be extremely sparse and therefore implementation is inefficient

Capabilities

- Focus on the subject
  - access rights are stored with the subject
  - capabilities ≡ rows of the access control matrix

Alice	edit.exe: {exec}	fun.com: {exec,read}

- Good match between *capabilities* and *distributed system security*
  - Security policies have to deal with roaming
- Problems of capabilities
  - How to check who may access to a specific object?
  - How to revoke a capability?

Protection and Authenticity of Capabilities

- If used in a single system
  - you may rely on the operating system's integrity and mechanisms employed by it
- If used over a network
  - authenticity and protection is mostly cryptographic

Access Control Lists (ACLs)

- Focus on the object
  - access rights are stored with the object
  - ACLs ≡ columns of the access control matrix

fun.com	Alice: {exec}	Bill: {exec,read,write}

- Access rights are often defined for groups of users
  - because individual subjects may create a huge list
- ACLs are typical for operating systems security
  - In UNIX, ACLs are attached to files

Aggregation Techniques

- ACLs and capability lists are of limited use (one focuses on subjects, the other focuses on objects)
- need to aggregate subjects and objects
  - Groups
  - Roles
  - Procedures
  - Data types

Role-based Access Control

Ownership

- Who is in charge of setting security policies.

1. We can define an owner for each resource. Policies defined by the owner are called discretionary because access control is at the discretion of the owner.
   - Access control based on user identities was called discretionary. We may adopting the term ***identity-based access control* (IBAC)** instead. IBAC incurs an identity management overhead and does not scale well.
2. A system-wide policy dictates who is allowed to have access. Policies imposed by the system are called mandatory.
   - Access control based on policies that refer to security labels for classified documents, e.g. confidential, top secret, was called ***mandatory access control***.

Intermediate Controls

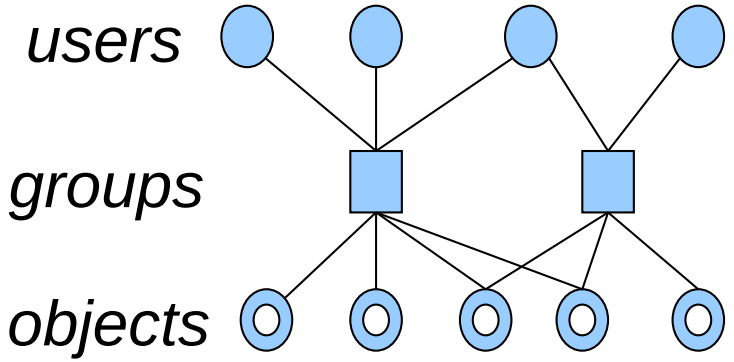
- In computer science, problems of complexity are solved by indirection (David Wheeler).
- This principle can also be applied to access control.
- We introduce intermediate layers between users and objects to represent policies in a more manageable fashion.

Group Permission

- Users with similar access rights are collected in groups and groups are given permission to access objects.
- An Example:
  - A teacher wants to give students access to course material.
  - Instead of putting all students individually into an ACL for each piece of course material, the teacher could put all students into a *group* and put this group into the respective ACLs.

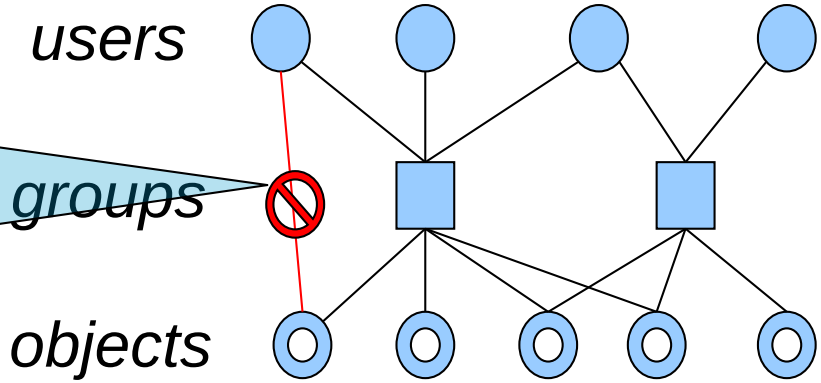
Groups & Negative Permissions

- Groups are an intermediate layer between users and objects.



- To deal with special cases, negative permissions withdraw rights

We have to know how conflicts will be resolved by the reference monitor.
ACLs process the list until the first entry matching.
Any conflicting entries later in the list are ignored.

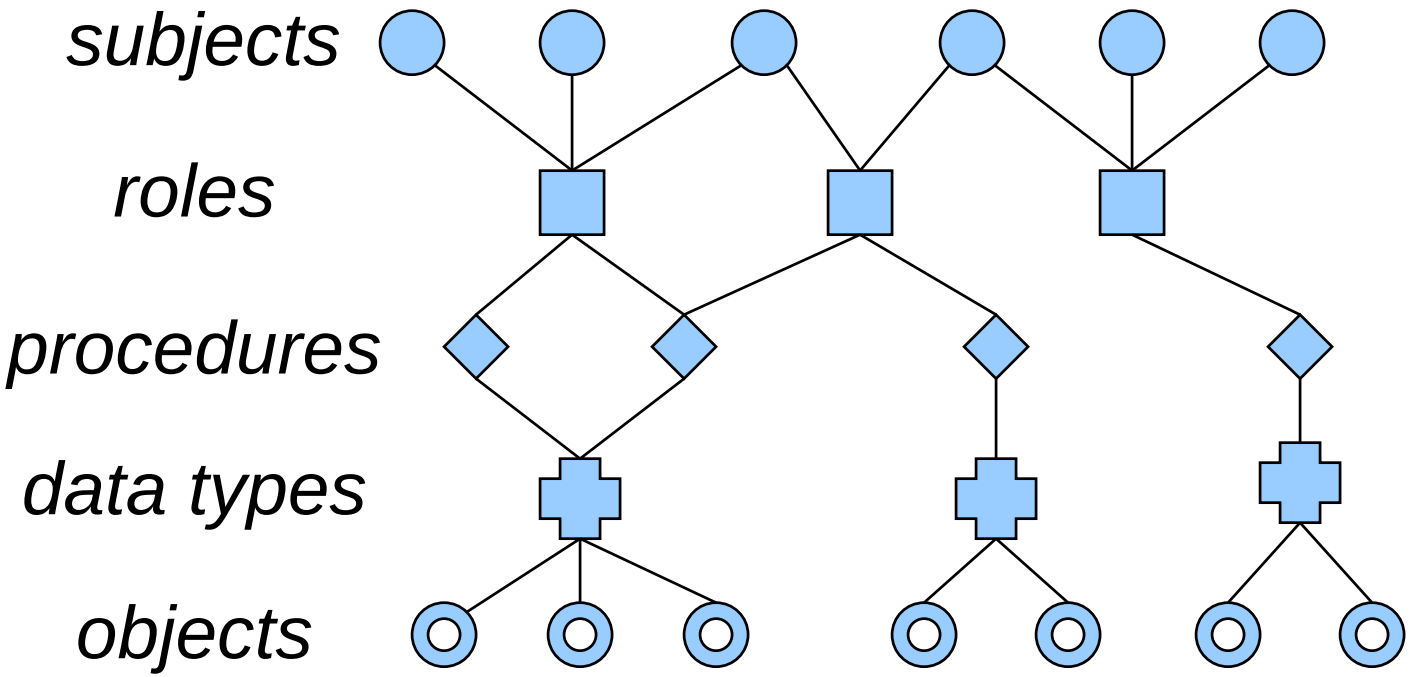


Role Based Access Control (RBAC)

- Data types: A *data type* is a set of objects with the same structure (e.g. bank accounts)
  - each object is of a certain data type and can be accessed only through procedures defined for this data type.
- Procedures: high level access control methods with more complex semantics than read or write
  - procedures can only be applied to objects of certain data types; example: funds transfer between bank accounts.
- Roles: collection of procedures assigned to roles; a user can have more than one role and more than one user can have the same role.
- When a user logs in, the process started derives its permissions from the roles assigned to that user.

Role-Based Access Control (RBAC)

- Several intermediate concepts can be inserted between subjects and objects



RBAC Example

- Objects are bank accounts
- Subjects are bank employees
- The set of bank accounts forms a data type
- We define roles
  - Teller
  - Clerk
  - Administrator
- We define procedures for
  - Crediting accounts (CA)
  - Debiting accounts (DA)
  - Transferring funds between accounts (TF)
  - Creating new accounts (NA)
- We assign procedure
  - CA and DA to the Teller role
  - TF to the Clerk role
  - NA to the Administrator role
  - The Administrator role can run all the procedures

Roles are a good match for typical access control requirements in businesses

RBAC Example and Role hierarchies

- Example:
  - The teacher could create a role *Student* for the students on his course and assign the permission to read course material to this role.
  - A role *Teacher* could be given the permission to edit the course material.
- *Role hierarchies* define relationships between roles.
  - A senior role can do anything the junior role can do. This helps policy administration.
  - In our example, a junior role *Teaching Assistant* could be given permission to edit material for exercise classes.

Protection Ring

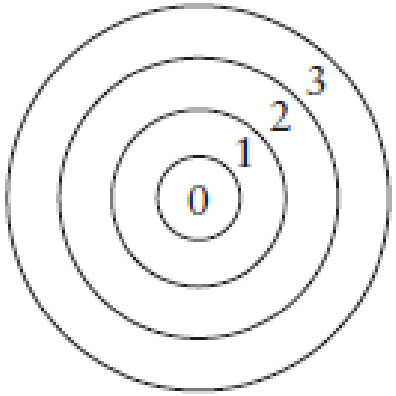
- This is simple example of an intermediate layer of hardware based access control for subjects and objects.
- Each subject (process) and each object is assigned a number, depending on its 'importance'.

0 – operating system kernel;

1 – operating system;

2 – utilities;

3 – user processes.



Ring 0 in the center giving the highest degree of protection. If a process is assigned the number *i*, we say the process 'runs in ring *i*'.

Unix uses two levels; with root and operating system running in ring 0 and user processes running in ring 3.

Policy Instantiation

- At this stage, security policies cannot refer to specific user identities, but can perhaps refer to generic *placeholder* principals such as *Teacher* and *Student* in the running example, or *owner*, *group*, *others* in Unix, or *friends* in a social network.
- The reference monitor has to resolve the placeholder principals to concrete user identities when processing an actual request.

Comparing Security Attributes

- When evaluating a security policy, the reference monitor compares the access rights granted to the subject with the access rights demanded by the policy.
- The most basic comparison is an equality check, e.g. between the user identity a process.

Security Labels and Partial orderings

- In several approaches to access control, functions are used to associate entities with a security label
  - a value that can be compared using an operator
- We can use a set *L* of security labels.
  - We need a way of comparing labels but we need not compare any pair of labels.
- A data structure with the property that some, but not all, elements can be compared is called a partial ordering.

Partial Ordering – An Example

- The department creates a group *Year_1* for first year students to manage access for resources specifically dedicated to them.
- There is also a group *Year_2* for second year students, *Year_3* for third year students, etc.
- The group of first year students would be contained in the group of all students, but there is no such relation between groups *Year_1* and *Year_2*.
- The best we can aim for is a *partial ordering.*

Partial orderings

- A partial ordering $\leq$ (read as 'less or equal' – but not necessarily numeric comparison) on a set L is relation on $L \times L$ that is
  - reflexive: for all $a \in L$, $a \leq a$
  - transitive: for all $a,b,c \in L$, if $a \leq b$ and $b \leq c$, then $a \leq c$
  - antisymmetric: for all $a,b \in L$, if $a \leq b$ and $b \leq a$, then $a = b$
- Examples for partial orderings

- $(\mathcal{P}(X), \subseteq)$, the power set of a set X with the subset relation as partial ordering.
- $(N, |)$, the natural numbers with the 'divides' relation as partial ordering.

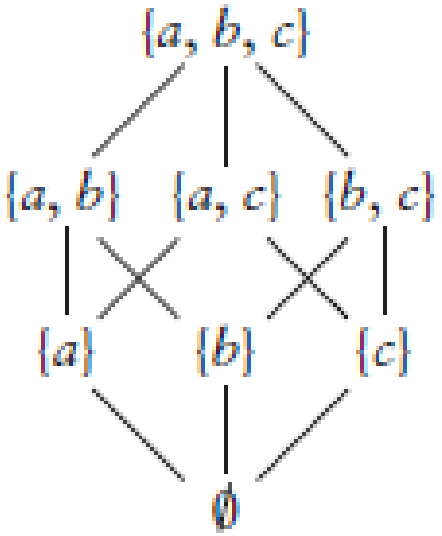
Accsess Control

- *Hasse diagrams* are a graphical representation of partially ordered sets (posets).
- A Hasse diagram is a directed graph in which the nodes are the elements of the set.
- The edges in the diagram give a ‘skeleton’ of the partial ordering.
- That is, for a, $b \in L$ we place an edge from a to b if and only if;
  - $a \leq b$ and $a \neq b$, and
  - there exists no $c \in L$ such that $a \leq c \leq b$ and $\boldsymbol{a \neq c, c \neq b}$.

Accsess Control

- With this definition, $a \leq b$ holds if and only if there is a path from a to b.
- Edges in the graph are drawn to point upwards.
- The Hasse diagram for the partially ordered set

$(P(\{a, b, c\}), \subseteq)$ is given in the following Figure:



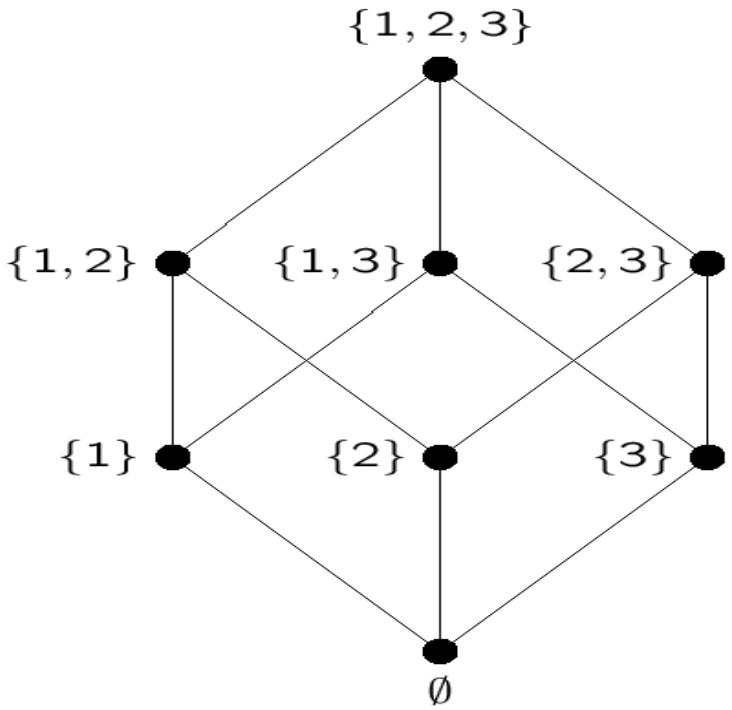
The Poset (Lattice) $(\mathcal{P}(\{a, b, c\}), \subseteq)$

Lesson

Access control algorithms compare attributes of subjects and objects. You always have to check what happens if an attribute is missing. Fail-safe behaviour would suggest that access should be denied. Often this is not the case and you could be in for an unpleasant surprise.

Example $-$ $\langle \mathcal{P}(\{1,2,3\}), \subseteq \rangle$

A *partial order*

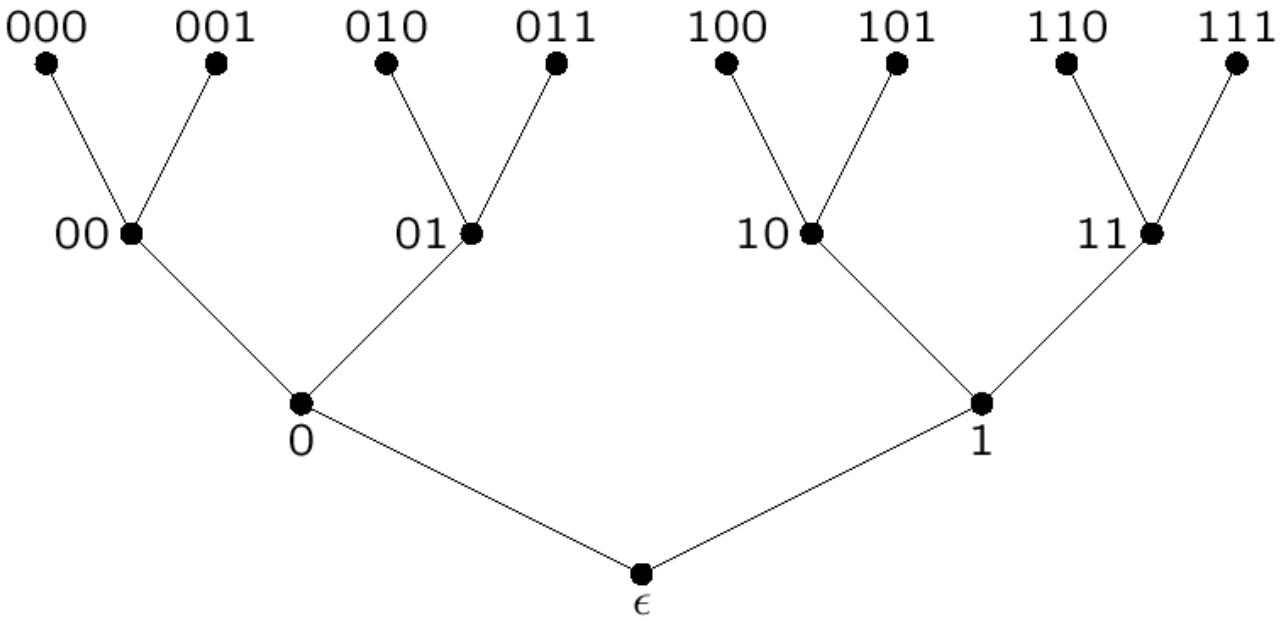


$\{1,2\} \not\subseteq \{1,3\}$ and $\{1,3\} \not\subseteq \{1,2\}$, for example

Example – $\langle \{0,1\}^*, \leqslant \rangle$

Let s and t be strings

$s \leqslant t$ if s is a *prefix* of t



Lattice of Security Levels

- If two groups of students should have access to a document, the department could use the longest common prefix of the abilities assigned to the two groups to label the document.

  For example,

  – if *Year_1* and *Year_2* should get access to a document, ability .3 could be used as a label.

- In general, given the standard confidentiality policy where a subject may observe an object only if the subject's security level is higher than the object's security level.

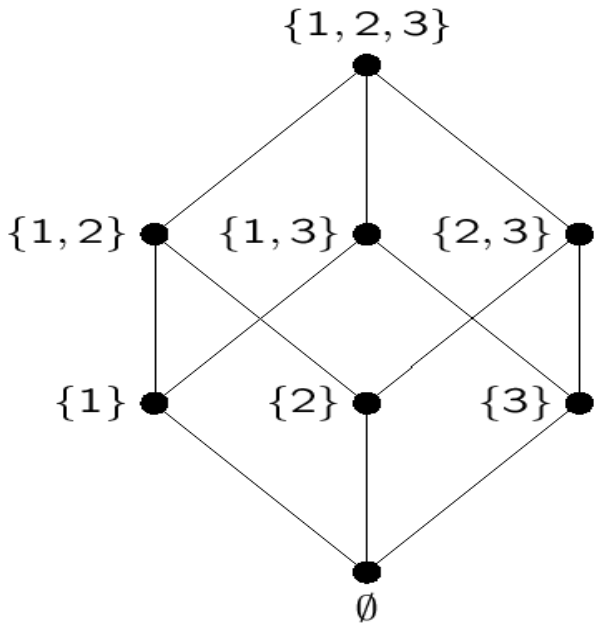
Lattice of Security Levels

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

- The mathematical structure that allows us to answer these two questions exists. It is called a *lattice*.

Upper bounds

An *upper bound* of x and y is an element that is greater than both x and y

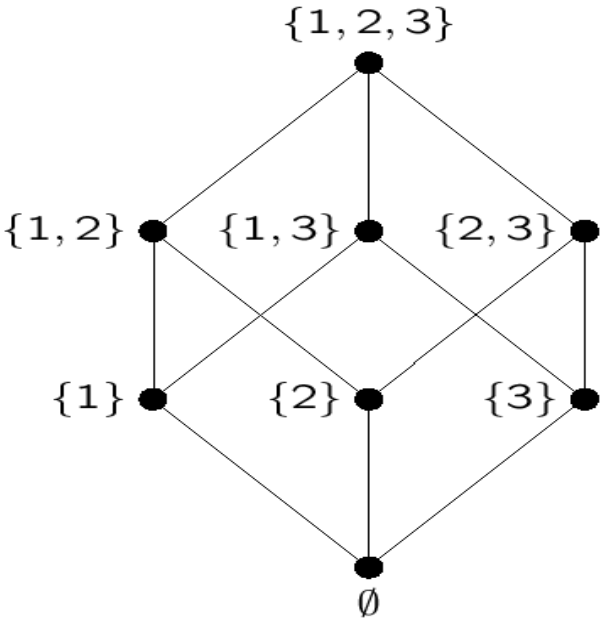


$\{1, 2\}$ and $\{1, 2, 3\}$ are upper bounds of $\{1\}$ and $\{2\}$

$\{1, 2\}$ is the *least upper bound* of $\{1\}$ and $\{2\}$

Lower bounds

A *lower bound* of x and y is an element that is less than both x and y

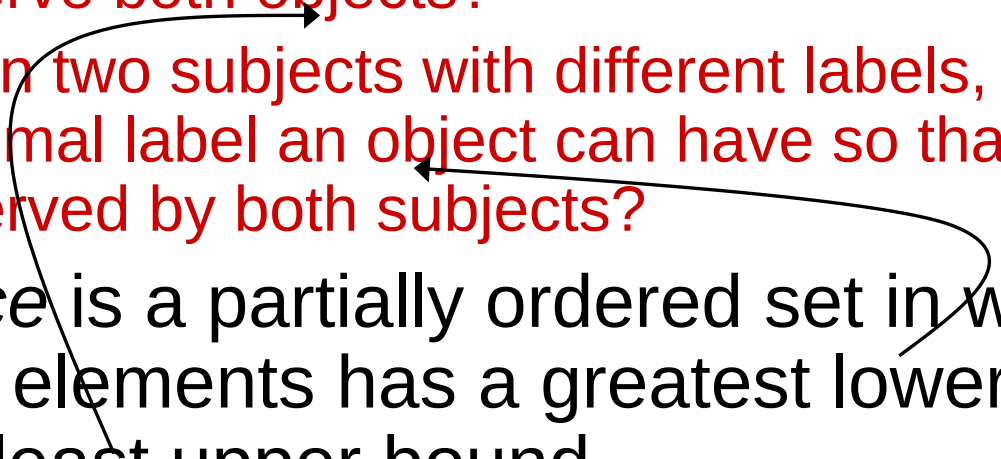


$\{1\}$ and $\emptyset$ are lower bounds of $\{1, 2\}$ and $\{1, 3\}$

$\{1\}$ is the *greatest lower bound* of $\{1, 2\}$ and $\{1, 3\}$

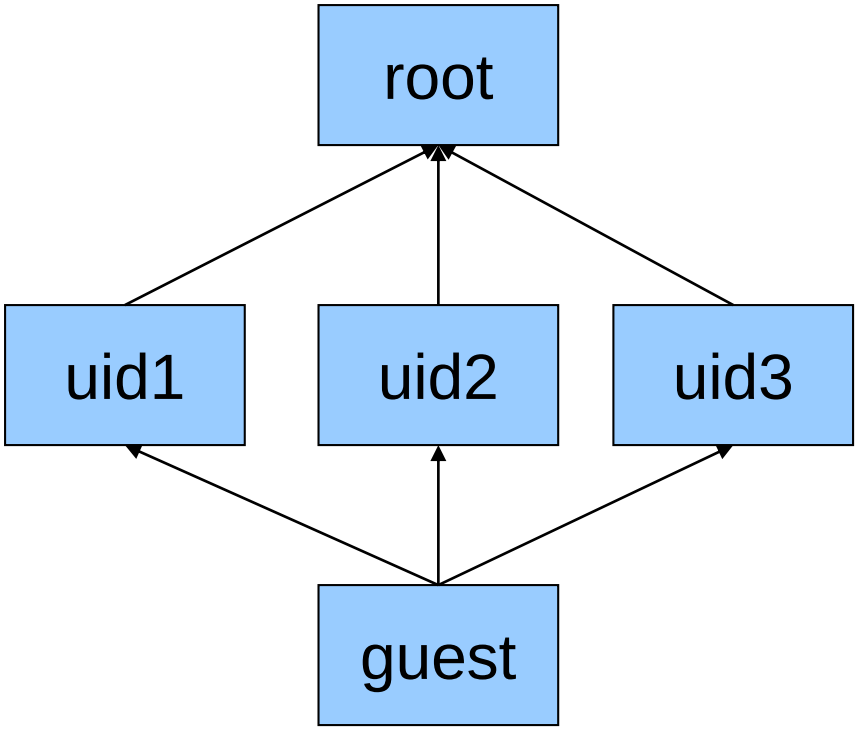
Lattices

- **General Rule**: a subject may observe an object only if the subject's label is higher than or equal to the object's label.
- Lattices are a mathematical structure where these questions have unique answers
  - Given two objects with different labels, what is the minimal label a subject must have to be able to observe both objects?
  - Given two subjects with different labels, what is the maximal label an object can have so that it can be observed by both subjects?
- A *lattice* is a partially ordered set in which every pair of elements has a greatest lower bound and a least upper bound



System Low and System High

- If $a \leq b$, we say '*a* is dominated by *b*' or '*b* dominates *a*'.
- If a label exists that is dominated by all other labels, it will be called System Low.
- If a label exists that dominates all other labels, it will be called System High.
- What are System Low and System High in the power set lattice example?



A ‘flat’ lattice

Models & Policies

- A security policy captures the security requirements of an enterprise or describes the steps that have to be taken to achieve security
- A security model is a formal description of a security policy
- Bell-LaPadula (BLP) model is the most famous one

Information flow policies

- To address confidentiality requirements
- We assume the existence of a lattice of security labels
- Every subject and object is assigned a security label using a security function λ
- **Information can flow from an entity *x* to an entity *y* if $\lambda(x) <= \lambda(y)$**
  - **information can flow from low security entity to high security one**
- Read and write access rights are defined in terms of information flow principles

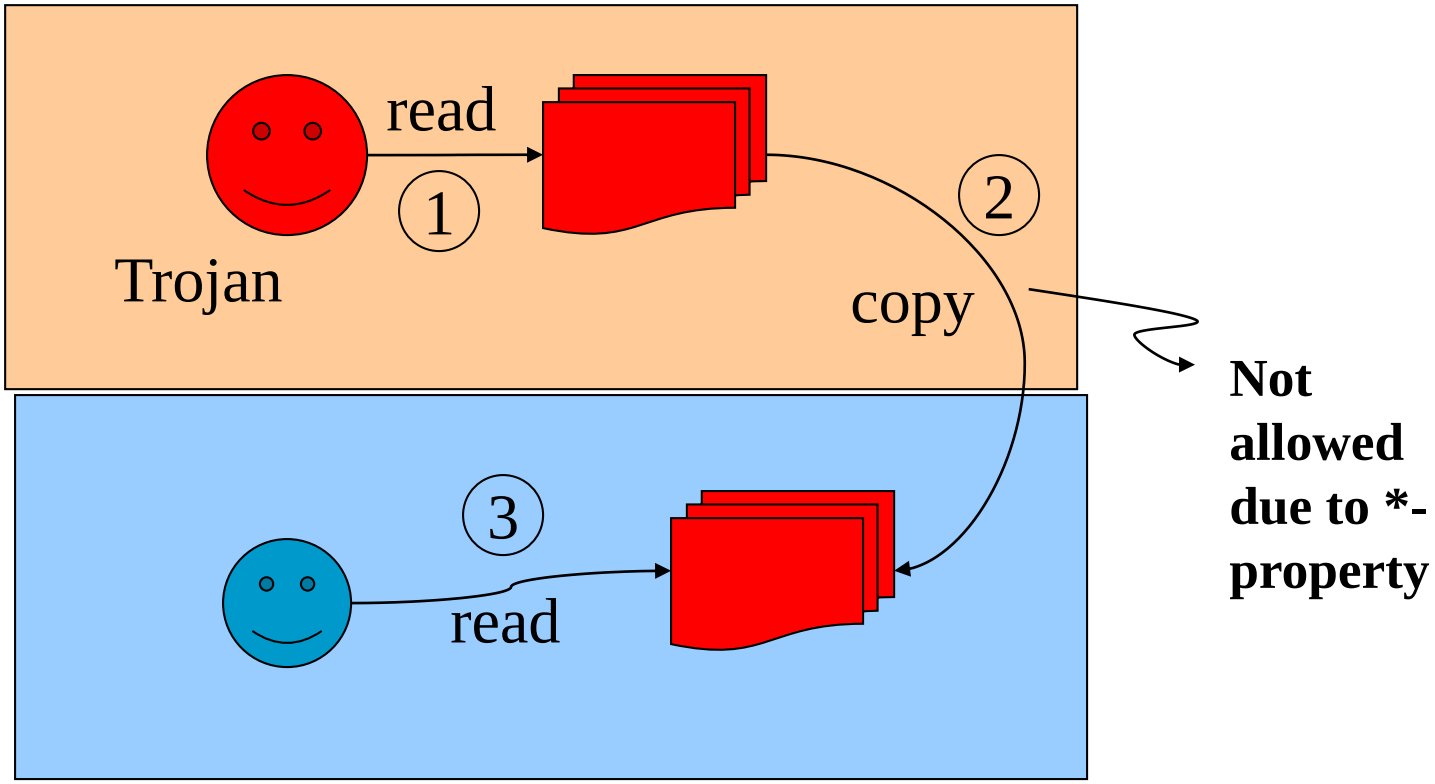
Read Access

- Information flow from an object *o* to a subject *s*
- Read access is granted if $\lambda(o) <= \lambda(s)$
  - you can read an object if your security label is larger than the object's
- This condition is known as "no read up" or the *simple security* (ss) *property* in BLP terms

Write Access

- Information flow from a subject *s* to an object *o*
- Write access is granted if $\lambda(s) <= \lambda(o)$
  - you can write to an object if your security label is smaller than object's
  - quite counter-intuitive, but necessary to prevent confidentiality violations such as
    - a top secret user writing to an insecure printer
- This condition is known as "no write down" or the $*$-*property* (*star property*) in BLP terms
- **No read-up and no write-down properties are "mandatory access control" policies of BLP**

Information flow blocked by *-property



A Trojan aims to read a high document and copy its contents to a low file.

No Write-Down

- The $*$ - property prevents high level entities from sending legitimate messages to low level entities
- Two ways to escape from this restriction:
  - Temporarily downgrade a high level subject; (downgrade current security level); BLP subjects should have no memory of their own! They have to forget what they knew when downgraded
    - Possible with processes, but not for human beings :)
  - Identify *trusted subjects* which are permitted to violate the $*$-property.
    - We redefine the $*$-property and demand it only for subjects, which are not trusted.

Discretionary Security Policy

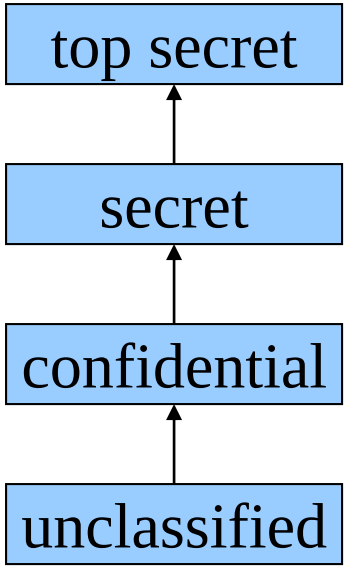
- Mandatory access control properties (ss and * properties) do not check whether a particular access is specifically permitted
- Discretionary Security Property (ds-property)
  - Defines the capability of a subject to operate on an object

In BLP, access must be permitted by the access control matrix M_{so}.

Multi level security (MLS)

- MLS: access control based on a partial ordering (actually a lattice) of security levels

- Traditional: hierarchical security levels (linear order):



Compartments

- In multi-level security, generally categories are used as well as the security *levels* in lattices
  - C is a set of all categories, e.g. project names, company divisions, academic departments, etc.
  - A *compartment* is a set of categories (c is a subset of C).
  - H is a set of security levels which are hierarchically ordered.
  - A security label (the function λ) is a pair (h,c), where $h \in H$ is a security level and $c \subseteq C$ is a compartment.
  - The partial ordering $\leq$ is defined by $(h_1,c_1) \leq (h_2,c_2)$ if and only if $h_1 \leq h_2$ and $c_1 \subseteq c_2$.

Compartments - Example

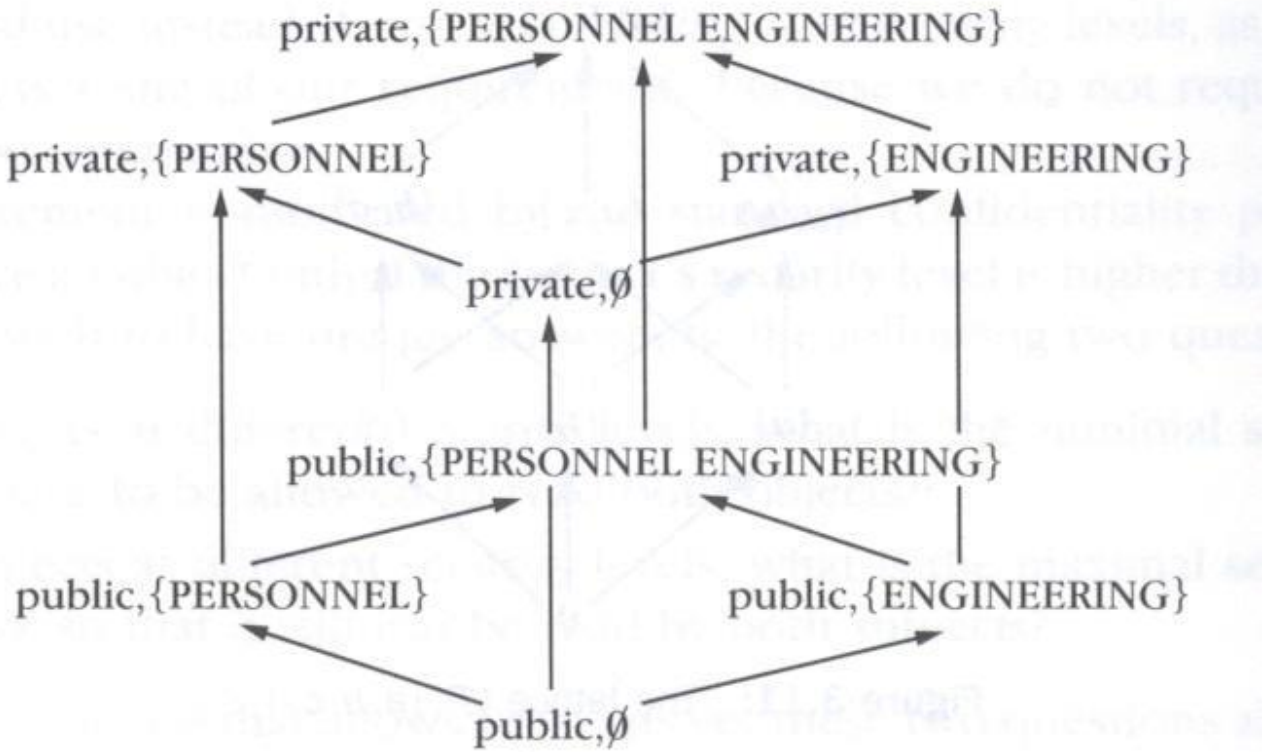
- Two hierarchical levels:
  - *public*, *private* (*public* ≤ *private*)
- Two categories: PERSONNEL, ENGINEERING
- For examples, the following relations hold:

(public, {PERSONNEL}) ≤ (private, {PERSONNEL})
(public, {PERSONNEL}) ≤ (public,{PERSONNEL,ENGINEERING})

- But the following one cannot be compared

(public, {PERSONNEL}) ≤ (private, {ENGINEERING})

Corresponding Lattice



A subject with security label (private, ENG) will not be able to read any object that has the category PER in the compartment of its label.

Thus, even an object labelled (public,{PER, ENG}) will be out of bounds.

The Bell-LaPadula Model

- Implements an information flow policy using a lattice with compartments and an access control matrix
- An example: evaluating a read access request in BLP
  - A read access request by subject *s* to object *o* is granted if
    - $\lambda(o) <= \lambda(s)$ (information flow policy) and
    - $r \in M [s, o]$ (appropriate entry in the access control matrix)
- BLP model actually a state machine

State Machine Models

- State machines (automata): popular tool for modelling many aspects of computing systems including security.
- The essential features of a state machine model are the concepts of a state and of state transitions.
  - A **state** is a representation of the system under investigation at one moment in time. **It should capture exactly those aspects of the system relevant to the problem**
  - The **state transition** (next state) function **defines the next state depending on the present state and the input**. An output may also be produced.

State Machine Models

- To design a secure system with the help of state machine models:
  - **define state set** so that it captures “security”
  - check that initial state of the system is ‘secure’
  - check that all state transitions starting in a “secure” state yield a “secure” state
- Security is then preserved by all state transitions. The system will always be ‘secure’.

States in BLP model

- A state in BLP model is
  - the current subjects, objects and access matrix among them and
  - the security levels of subjects and objects
  - current accesses by subjects to objects

Basic Security Theorem

- A state is secure, if all current access tuples *(s,o,a)* are permitted by the ss-, ∗-, and ds-properties.
- A state transition is secure if it goes from a secure state to a secure state.
- How would you define state transition in BLP?

Basic Security Theorem: If the initial state of a system is secure and if all state transitions are secure, then the system will always be secure.

Harrison-Ruzo-Ullman Model

- BLP has no policies for changing access rights or for the creation and deletion of subjects and objects.
- The Harrison-Ruzzo-Ullman (HRU) model defines authorization systems that address these issues.
- The components of the HRU model:
  - set of subjects S
  - set of objects O
  - set of access rights R
  - access matrix $M = (M_{so})_{s \in S, o \in O}$: entry M_{so} is a subset of R giving the rights subject s has on object o

Primitive Operations in Harrison-Ruzo-Ullman - HRU

- Six primitive operations for manipulating subjects, objects, and the access matrix:
  - enter $r \in R$ into M_{so}
  - delete $r \in R$ from M_{so}
  - create subject s
  - delete subject s
  - create object o
  - delete object o

Examples

- Subject *s* creates a file *f* so that *s* owns the file (access right *o*) and has read and write permission to the file (access rights *r* and *w*).

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

- The owner *s* of file *f* grants read access to another subject *p*

command grant_read(*s*,*p*,*f*)
    if *o* in $M_{s,f}$
        then enter *r* in $M_{p,f}$
end

Security vs. Complexity in HRU Model

- The access matrix describes the state of the system; commands change the access matrix.
- HRU can model policies for allocating access rights.
- To verify compliance with a given policy, you have to check that no undesirable access rights can be granted.
- HRU model has some definitions and theorems about the decidability of the safety of the system
  - Can be saying that HRU model does not help to verify safety in its full generality, but verification is possible with some restrictions.
- The result of those theorems is:
  - The more expressive and complex the security model, the more difficult to verify security

Access Control

- Dictates what types of access are permitted, under what circumstances, and by whom.
  - Discretionary access control (DAC); Controls access based on the identity of the requestor and on access rules stating what requestors are allowed to do.
  - Mandatory access control (MAC); based on comparing security labels with security clearances. (**Security labels** indicate how sensitive or critical system resources are. **Security clearances** indicate which system entities are eligible to access certain resources.)
  - Role-based access control (RBAC); controls that what acesses are allowed to users in given roles.

Not mutually exclusive

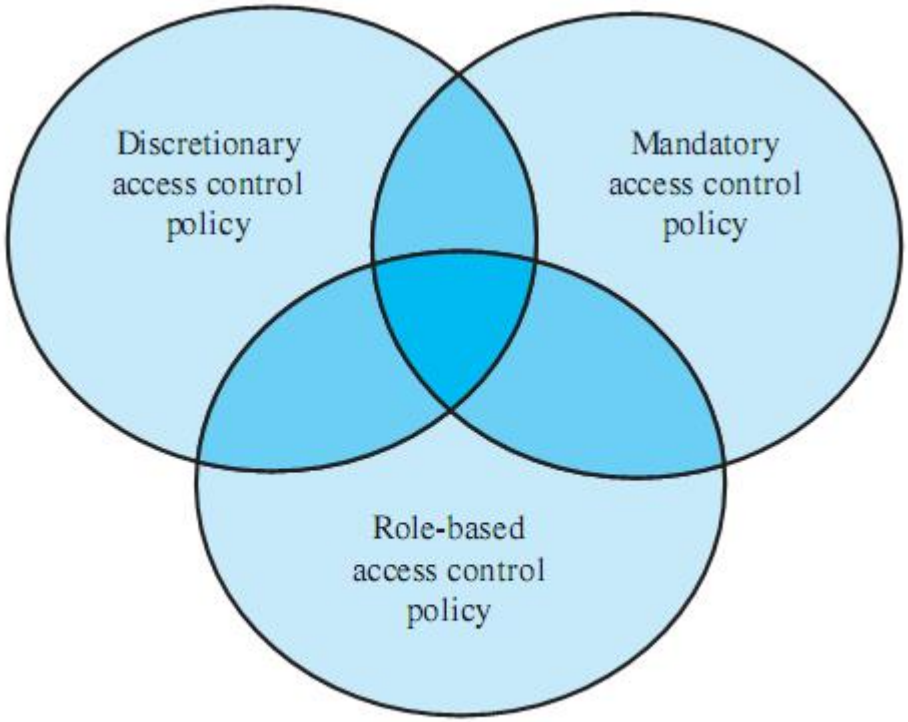


Figure 15.3 **Access Control Policies**