Security Policies

Sources:

- *Security Engineering* by Ross Anderson
- *Computer Security* by Matt Bishop
- *Security in Computing* by C.P. and S.L. Pfleeger

A *security policy* is a statement that partitions the states of a system into a set of *authorized* or *secure* states and a set of *unauthorized* or *nonsecure* states.

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

A *breach of security* occurs when a system enters an unauthorized state.

A security policy addresses confidentiality, integrity, and availability.

Definition. Let *X* be a set of entities and let *I* be some information. Then *I* has the property of *confidentiality* with respect to *X* if no member of *X* can obtain information about *I*.

Definition. Let *X* be a set of entities and let *I* be some information or a resource. Then *I* has the property of *integrity* with respect to *X* if all members of *X* trust *I*.

Definition. Let *X* be a set of entities and let *I* be a resource. Then *I* has the property of *availability* with respect to *X* if all members of *X* can access *I*.

A *confidentiality policy* or *information flow policy* indicates what entities are authorized to receive particular kinds of information.

An *integrity policy* indicates how information may be altered and what entities are authorized to alter it.

Traditionally, military security policies were intended primarily to ensure confidentiality.

Commercial security policies were intended primarily to address integrity.

A security policy also indicates what *services* must be available.

Definition. A *security mechanism* is an entity or procedure that enforces some part of the security policy.

Note that mechanism is distinct from policy.

Example: Copying another student's homework file.

Trust

Absolute security is difficult or impossible to verify.

Hence, security professionals often prefer to speak of *trusted* systems rather than *secure* ones.

Qualities of security and trust [Pfleeger]:

Secure	Trust
Either-or: something is or isn't secure	*Graded*: there are degrees of trustworthiness
Property of *presenter*	Property of *receiver*
Asserted based on product characteristics	*Judged* based on evidence and analysis (hence can grow over time)
Absolute: not qualified as to how, where, when, and by whom used	*Relative*: viewed in context of use
A *goal*	A *characteristic*

Trust often depends on a number of *assumptions*, both explicit and implicit.

Example: [Bishop] Trust in an OS security patch involves the following assumptions:

1. The patch came from the vendor and was not tampered with in transit.

2. The patch implementer is competent and exercised due diligence.

3. The vendor tested the patch thoroughly.

4. The vendor's test environment corresponds to the use environment.

5. The patch is installed correctly.

Policy Languages

A *policy language* is a language for precisely specifying a security policy.

A *high-level* policy language expresses policy constraints on entities using *abstractions*.

A *low-level* policy language is a set of inputs or arguments to commands that set or check constraints on a system.

Example: Java Policy Files [java.sun.com]

An example of two entries in a policy configuration file is:

```
// If the code is signed by "Duke", grant it read/write 'access to
//   all files in /tmp:
grant signedBy "Duke" {
  permission java.io.FilePermission "/tmp/*", "read,write";
};

// Grant everyone the following permission:
grant {
  permission java.util.PropertyPermission "java.vendor", “read”;
};
```

The contents of another policy configuration file:

```
grant signedBy "sysadmin", codeBase "file:/home/sysadmin/*" {
  permission java.security.SecurityPermission "Security.insertProvider.*";
  permission java.security.SecurityPermission "Security.removeProvider.*";
  permission java.security.SecurityPermission "Security.setProperty.*";
};
```

This specifies that only code that satisfies the following conditions can call methods in the Security class to add or remove providers or to set properties:

- The code was loaded from a signed JAR file that is in the "/home/sysadmin/" directory on the local file system.
- The signature can be verified using the public key referenced by the alias name "sysadmin" in the keystore.

Military Security Policy

Military security policy is based primarily on protecting the confidentiality of classified information.

Each piece of information is given a *sensitivity level*, e.g., unclassified, restricted, confidential, secret, and top-secret.

These levels form a *hierarchy*.

Information access is restricted by the *need-to-know rule*.

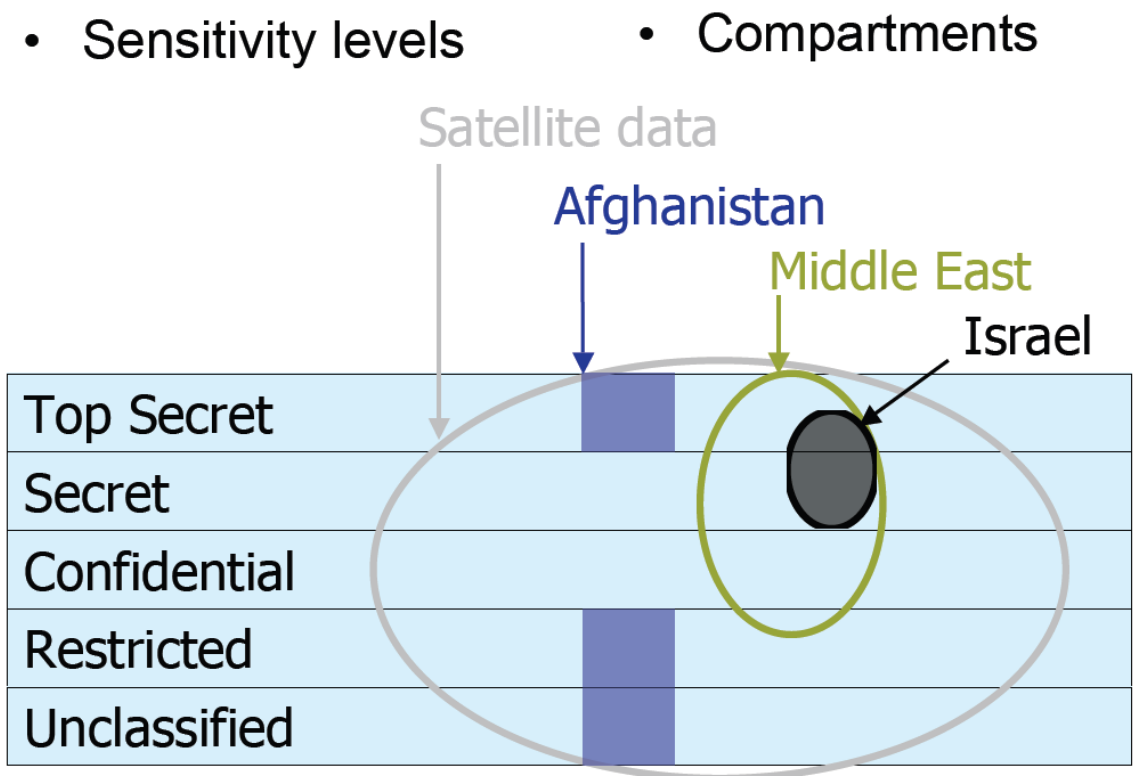
Access to sensitive information is granted only to subjects that require it to perform their jobs.

This model is also called *multilevel security* (*MLS*).

Each piece of classified information may be associated with one or more *compartments*.

A compartment corresponds to a project.

It may contain information at multiple sensitivity levels.



[www.cis.upenn.edu/~cse331/lectures/CSE331-35.pdf]

A single piece of information can be coded with multiple compartment names, e.g., *enigma*, *cabal*, or *bushmaster*.

The pair *<level, compartments>* is called the *class* or *classification* of a piece of information.

A *clearance* is an indication that a person

- is trusted to access information up to a certain sensitivity level
- needs to know certain categories of sensitive information

The clearance of a subject is also expressed as a pair *<level, compartments>*.

We say a subject *s* *dominates* an object *o* if

$$rank(s) \geq rank(o) \text{ and}$$
$$compartments(s) \supseteq compartments(o)$$

A subject can access an object only if it dominates it.

Information classified as <*secret*, {*Sweden*}> could be read by someone cleared for access to <*top secret*, {*Sweden*}> or <*secret*, {*Sweden*, *crypto*}>.

It could not be accessed by someone cleared for <*secret*, {*France*}>.

Sensitivity requirements are known as *hierarchical* requirements, because they reflect the hierarchy of sensitivity levels.

Need-to-know requirements are *nonhierarchical*, because compartments my cross sensitivity levels.

When sensitivity levels and compartments are used together, the result is a *lattice model*.

Types of Access Control

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*).

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*.

An *originator controlled access control* (*ORCON* or *ORGCON*) bases access on the creator of an object.

Bell and La Padula Confidentiality Model

The *Bell and La Padula* (*BLP*) *model* describes the permissible information flows in a secure system.

Its focus is on maintaining *secrecy*.

It is a formalization of the military security policy.

It combines mandatory and discretionary access control.

Each subject has a *security clearance* and each object has a *security classification*.

A set of *categories* or *codewords* is associated with each security classification.

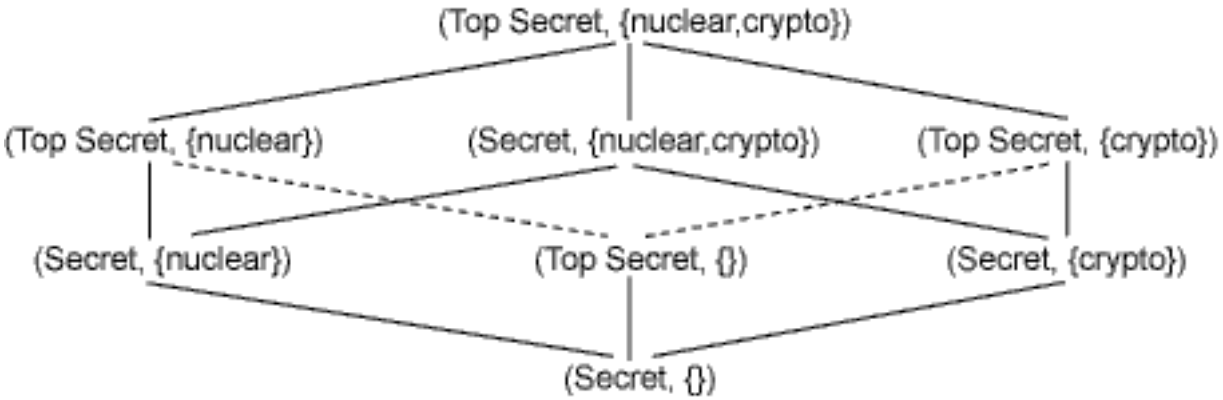
Each clearance/classification level and category form a *security level*.

Security levels change access.

Example: If George is cleared at the level (SECRET, {NUC, EUR}), he can access secret information in the categories NUC and EUR, but not in the category AIS.

The security level (L, C) *dominates* the security level (L', C') if an only if $L' \leq L$ and $C' \subseteq C$.

Example: If DocA is classified as (CONFIDENTIAL, { NUC }) and DocB is classified as (SECRET, {EUR, US}), then George dominates DocA but not DocB.



Two properties characterize the secure flow of information:

- Simple Security Property
- *-Property

Simple Security Property: *S* may have *read* access to *O* only if *S* dominates *O*.

Another condition is needed to prevent *write down*, which occurs when a subject with access to high-level data transfers the data by writing to a low level object.

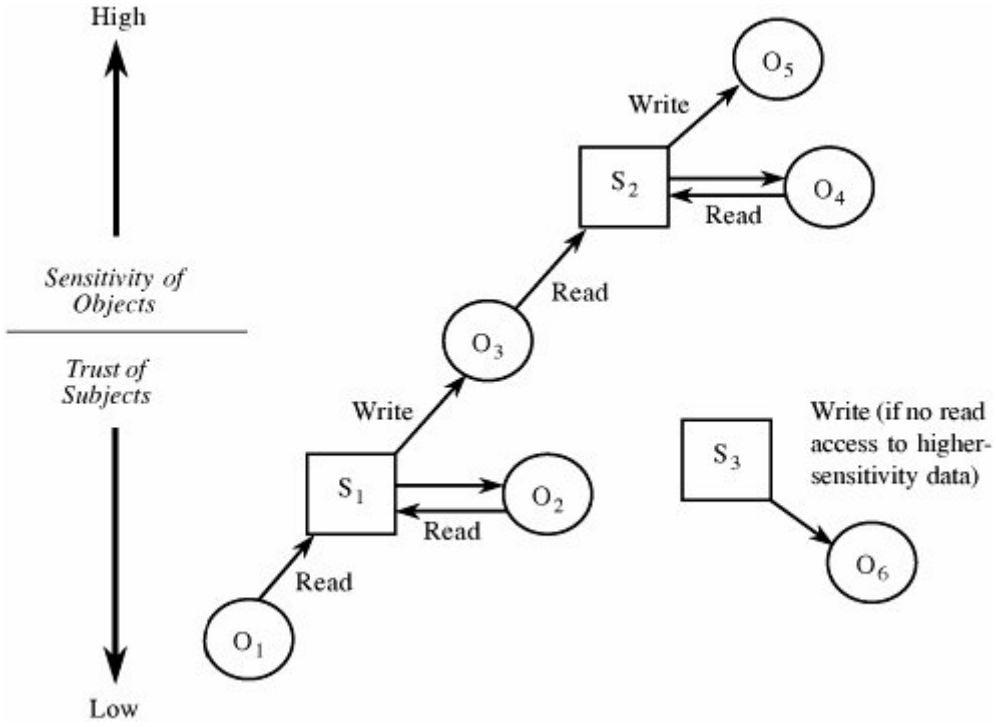
***-Property:** A subject *S* who has *read* access to an object *O* may have *write* access to an object *P* only if *P* dominates *O*.

The simple security condition is often described as "no reads up".

The *-Property is often described as "no writes down".

The Bell-La Padula model is extremely conservative.

Figure 5-7. Secure Flow of Information.



[Pfleeger & Pleeger, *Security in Computing*]

Basic Security Theorem. Let Σ by a (finite-state) system with a secure initial state σ_0, and let T be a set of state transformations. If every element of T preserves the simple security condition and the *-property, then every σ_i, $i \geq 0$, is secure.

To allow a high-level subject to communicate with a low-level subject, a *maximum security level* and a *current security level* are defined.

The maximum security level must dominate the current security level.

A *trusted subject* may decrease its security level from the maximum in order to communicate with entities at lower security levels.

Example: George could change his current security level to (SECRET, {EUR}) to send a document to someone with that clearance.

Implementing Multilevel Security

MLS systems must enforce information flow controls independently of user actions.

They therefore employ mandatory access control.

Several MLS operating systems have been implemented (e.g., *Multics*).

They've proven to be vulnerable to *covert channels*.

These channels used OS resources to transmit data between higher classified processes and lower classified processes, e.g.,

For example, the higher process might systematically modify a file name or attribute that is visible to the lower process in order to transmit data.

It is also tricky to safely *compose* MLS systems.

Other difficult issues are *combining* data from different compartments and *downgrading* it after sanitation.

Commercial Security Policies

Some of the concepts from military security policies carry over to commercial policies, e.g.,

- Businesses are typically divided into groups or departments, each responsible for a number of disjoint projects

- Data items may have different degrees of sensitivity, e.g., *public*, *proprietary*, *internal*.

However, the commercial world is less rigidly structured than the military world.

There is usually no formalized notion of a clearance.

Hence, rules for allowing access to information are less regularized.

For example, if a manager decides someone needs access to internal data from a project, he/she can simply grant that access.

Commercial security policies emphasize preservation of *data integrity* more than military policies do.

Lipner identified five requirements for commercial applications:

1. Users will use existing production programs and databases.

2. Programmers will develop and test programs on a non-production system; they may obtain production data for testing by a special process.

3. A special process must be followed to install a program from the development system onto the production system.

4. The process in (3) must be controlled and audited.

5. Managers and auditors must have access to the system state and the system logs that are generated.

These requirements suggest three principles of operation:

- *Separation of duty*: if two or more steps are required to perform a critical function, at least two different people should perform the function.

- *Separation of function*: Production resources are separated from development resources.

- *Auditing*: the process of analyzing a system to determine what actions took place and who performed them.

These principles guard against errors and subversion.

Biba Integrity Model

Biba proposed three integrity policies.

In his model, as system consists of a set S of subjects, a set O of objects, and a set I of integrity levels.

The integrity levels are ordered by the *dominance* relations $<$ and $\leq$.

The function $i: S \cup O \rightarrow I$ returns the integrity level of a subject or object.

The relation *reads* $\subseteq S \times O$ defines the ability of a subject to read an object.

The relation *writes* $\subseteq S \times O$ defines the ability of a subject to write an object.

The relation *executes* $\subseteq S \times O$ defines the ability of a subject to execute an object.

The higher the integrity level of a program, the more confidence one has that it will execute correctly.

Data at a higher integrity level is more accurate and/or reliable than data at a lower level.

Definition 6-1 [Bishop]: An *information transfer path* is a sequence of objects o_1, o_2, ..., o_{n+1} and a corresponding sequence of subjects s_1, s_2, ..., s_n such that s_i *reads* o_i and s_i *writes* o_{i+1} for all i, $1 \leq i \leq n$.

Low-Water-Mark Policy

Whenever a subject accesses an object, the policy changes the integrity level of the subject to the lower of the subject and the object:

1. s can write to o if and only if $i(o) \leq i(s)$.

2. If s reads o then $i'(s) = min(i(s), i(o))$, where $i'(s)$ is the subject's integrity level after the read.

3. s_1 can execute s_2 if an only if $i(s_2) \leq i(s_1)$.

The first rule prevents writing from one level to a higher level.

The second rule causes a subject's integrity level to drop whenever it reads an object at a lower integrity level.

The third rule allows a subject to execute another subject provided the second is not at a higher integrity level.

This policy prevents direct modifications that would lower integrity levels.

It also prevents indirect modification, by lowering the integrity level of a subject that reads from an object with a lower integrity level.

Theorem 6-1 [Bishop]: If there is an information transfer path from object o_1 to object o_{n+1}, then enforcement of the low-water-mark policy requires that $i(o_{n+1}) \leq i(o_1)$ for all $n > 1$.

Proof: Exercise (Use complete induction on n).

The problem with the low-water-mark policy is that the integrity levels of subjects decrease until they cannot access objects at a high integrity level.

Ring Policy

This policy ignores the issue of indirect modification.

1. Any subject may read any object, regardless of integrity levels.

2. s can write to o if and only if $i(o) \leq i(s)$.

3. s_1 can execute s_2 if an only if $i(s_2) \leq i(s_1)$.

Theorem 6-1 holds for this model also.

Biba's Model (Strict Integrity Policy)

This model is the "dual" of the Bell-LaPadula Model:

1. s can read o if and only if $i(s) \leq i(o)$.

2. s can write to o if and only if $i(o) \leq i(s)$.

3. s_1 can execute s_2 if an only if $i(s_2) \leq i(s_1)$.

Theorem 6-1 still holds, but its proof changes.

The Chinese Wall Model [Brewer and Nash]

This is a model of a *hybrid* security policy that addresses both confidentiality and integrity.

It describes policies that involve *conflict of interest* in business.

A stock exchange or investment house is a natural environment for this model.

In this context, the goal of the model is to *prevent a conflict of interest* when a trader represents multiple clients whose best interests differ.

Consider a database at an investment house.

The *objects* of the database are items of information related to a company.

A *company dataset* (*CD*) contains objects related to a single company.

A *conflict of interest* (COI) *class* contains the datasets of companies in competition.

Let $COI(O)$ represent the (unique) COI class of object O that contains the dataset $CD(O)$.

Suppose Anthony has access to the objects in the CD of Bank of America.

He cannot gain access to Citibank's CD.

Anthony should also not be assigned Citibank's portfolio after working on Bank of America's.

Let $PR(S)$ be the set of objects that S has read.

CW Simple Security Condition: S is allowed to read O if and only if any of the following is true:

1. There is an object O' such that S has already accessed O' and $CD(O') = CD(O)$.

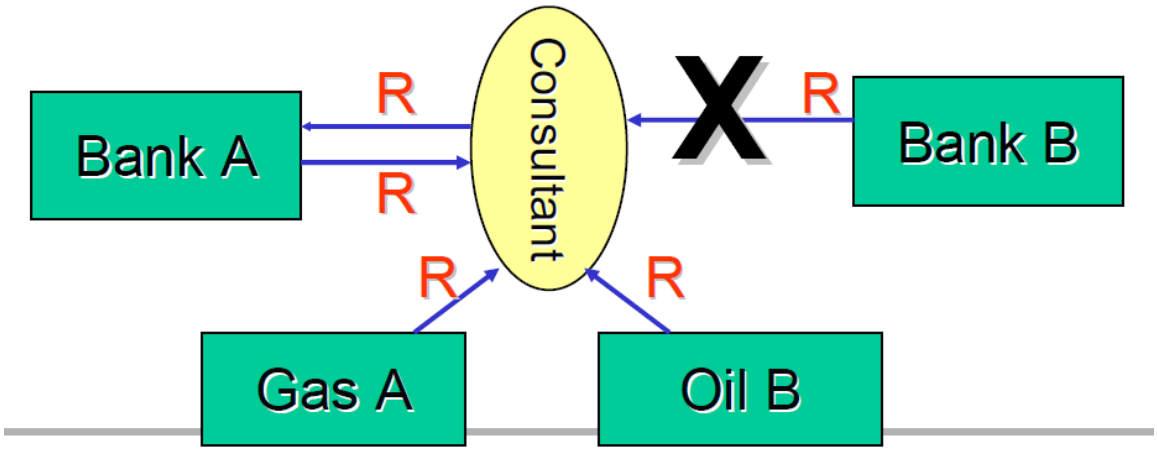
2. For all objects O', $O' \in PR(S)$ implies $COI(O') \neq COI(O)$.

3. O is a "sanitized" object.

Thus, a subject *S* can read an object *O* only if *O*:

- is in a dataset *S* has already accessed; or
- belongs to a COI from which *S* has not yet accessed any information

Assume that banks and energy companies have different COI sets.



[homes.cerias.purdue.edu/~bhargav/cs526/security-4.pdf]

Another condition is necessary to prevent improper indirect information flows.

CW-* Property: A subject *S* is permitted to write an object *O* if and only if both of the following conditions hold:

1. The CW simple security condition permits *S* to read *O*.

2. For all unsanitized objects *O*′, *S* can read *O*′ implies *CD*(*O*′) = *CD*(*O*).

Suppose Anthony can read objects in both Bank of America's CD and ARCO's CD.

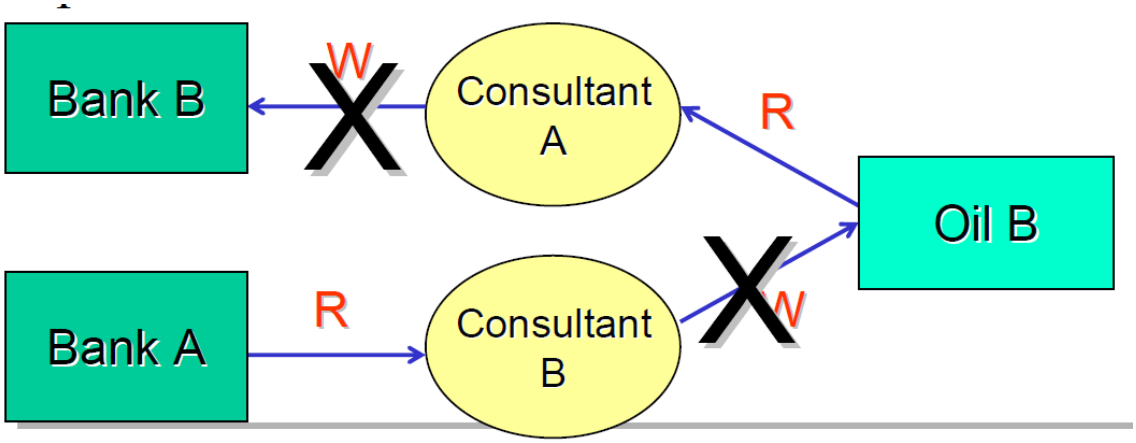
Then Condition 1 is met.

However, if Bank of America's CD contains unsanitized objects, then Condition 2 is false.

Hence Anthony cannot write objects in ARCOS's CD.

Thus, a subject *S* can write an object *O* if

- *S* can read *O* according to the CWSSC and
- No object has been read by *S* that is from a different CD from the one on which the write is to be performed



[homes.cerias.purdue.edu/~bhargav/cs526/security-4.pdf]