Access Control

Lecturer: Michael Clarkson

Lecture notes by Michael Clarkson
based in part on notes by Lynette I. Millett
from lectures by Professor Fred B. Schneider

This lecture is based in part on:

Pierangela Samarati and Sabrina De Capitani di Vimercati. Access Control: Policies, Models, and Mechanisms. In *Foundations of Security Analysis and Design: Tutorial Lectures*, Lecture Notes in Computer Science, vol. 2171, p. 137--193, 2001.

Note: If your browser does not support Unicode, you will not be able to view this page correctly. Your browser must be able to render the following symbols: ⊆ (subset or equal), ≤ (less than or equal), ≠ (not equal).

Recall that Lampson's *gold standard* identifies authorization, authentication, and audit as essential mechanisms for computer security. We begin studying authorization, which controls whether actions of principals are allowed, by considering access control. An *access control policy* specifies access rights, which regulate whether requests made by principals should be permitted or denied.

In access control, we refine the notion of a principal to be one of a:

- *user:* a human
- *subject:* a process executing on behalf of a user
- *object:* a piece of data or a resource.

Discretionary Access Control

A *discretionary access control* (DAC) policy is a means of assigning access rights based on rules specified by users. This class of policies includes the file permissions model implemented by nearly all operating systems. In Unix, for example, a directory listing might yield "... rwxr-xr-x ... file.txt", meaning that the owner of file.txt may read, write, or execute it, and that other users may read or execute the file but not write it. The set of access rights in this example is {read, write, execute}, and the operating system mediates all requests to perform any of these actions. Users may change the permissions on files they own, making this a discretionary policy.

A mechanism implementing a DAC policy must be able to answer the question: "Does subject S have right R for object O?" Abstractly, the information needed to answer this question can be represented as a mathematical relation D on subjects, objects, and rights: if (S,O,R) is in D, then S does have right R for object O; otherwise, S does not. More practically, the same information could also be represented as an *access control matrix*. Each row of the matrix corresponds to a subject and each column to an object. Each cell of the matrix contains a set of rights. For example:

	file1	file2
Alice	rwx	r-x
Bob	r--	rw-

Real systems typically store the information from this matrix either by columns or by rows. An implementation that stores by columns is commonly known as an *access control list* (ACL). File systems in Windows and Unix typically use such an implementation: each file is accompanied by a list containing subjects and their rights to that file. An implementation that stores by rows is commonly known as a *capability list*, by analogy with the use of capabilities in authentication. Each subject maintains an unforgeable list of the rights it has to objects. Both implementations make certain questions easier to answer than others. For example, it is easy in an ACL implementation to find the set of all subjects who may read a file, but it is difficult to find the set of all files that a subject may read.

The underlying philosophy in DAC is that subjects can determine who has access to their objects. There is a difference, though, between trusting a person and trusting a program. E.g., A gives B a program that A trusts, and since B trusts A, B trusts the program, while neither of them is aware that the program is buggy. Suppose a subject S has access to some highly secret object O. Moreover, suppose that another subject S' does not have access to O, but would like to. What can S' do to gain access? S' can write a program that does two things, the first of which is the following sequence of commands:

- Create a new object O'.
- Grant S write access to O'.
- Grant S' read access to O'.
- Copy O to O'.

The second thing the program does is to act like a video game (or some other application that S might be interested in running). If the program is run by S, then S' will get access to the contents of O (now in O'). This type of program is referred to as a *Trojan horse*. DAC mechanisms are typically insufficient to protect against Trojan horse attacks.

Mandatory Access Control

A *mandatory access control* (MAC) policy is a means of assigning access rights based on regulations by a central authority. This class of policies includes examples from both industry and government. The philosophy underlying these policies is that information belongs to an organization (rather than individual members of it), and it is that organization which should control the security policy. MAC policies strive to defend against Trojan horse attacks.

MAC Example 1. Multi-level security (MLS)

In national security and military environments, documents are labeled according to their *sensitivity* levels. In the US, these range from *Unclassified* (anyone can see this) to *Confidential* to *Secret* and finally (we believe) to *Top Secret*; other countries use similar classifications. These levels correspond to the risk associated with release of the information.

But it is not sufficient to use only sensitivity levels to classify objects if one wants to comply with the *need to know* principle: access to information should only be granted if it is necessary to perform one's duties. *Compartments* are used to handle this decomposition of information. Every object is associated with a set of compartments (e.g. crypto, nuclear, biological, reconnaissance, etc.). An object associated with {crypto, nuclear} may be accessed only by subjects who need to know about both cryptography and nuclear weapons.

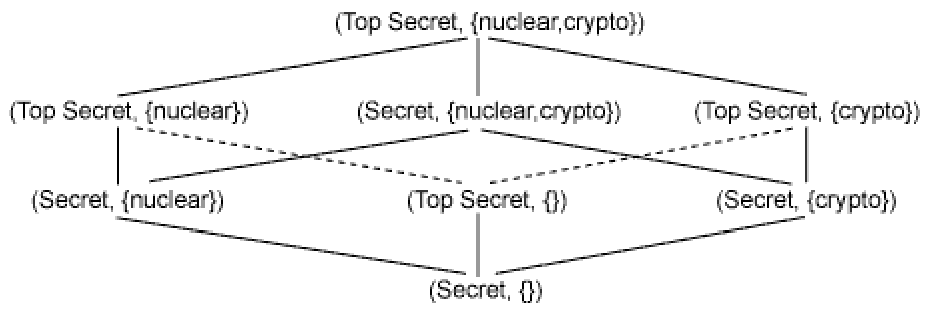
A *label* is a pair of a sensitivity level and a set of compartments. A document might have the label (Top Secret, {crypto,nuclear}) if it contained extremely sensitive information regarding cryptography and nuclear weapons. In practice, each paragraph in a document is assigned a set of compartments and a sensitivity. The classification of the entire document would then be the most restrictive classification given to a paragraph in that document.

Users are also labelled according to their security *clearance*. A user's clearance, just like a document's label, is a pair of a sensitivity level and a set of compartments.

Given two labels L1 = (S1, C1) and L2 = (S2, C2), we write that L1 $\leq$ L2---meaning that L1 is no more restrictive than L2---when

- S1 $\leq$ S2, where Unclassified $\leq$ Confidential $\leq$ Secret $\leq$ Top Secret, and
- C1 $\subseteq$ C2.

Notice that ≤ is a partial order: it is possible to have two labels that are incomparable (e.g. (secret, {crypto}) vs. (top secret, {nuclear})) according to ≤ . The following diagram depicts some of the ≤ relationships as a *lattice*, where a line from a label L1 lower in the lattice to a label L2 higher in the lattice denotes that L1 ≤ L2.



Bell and LaPadula (1973) gave a formal, mathematical model of multi-level security. This model enforces the *BLP policy*:

> Information cannot leak to subjects who are not cleared for the information.

Let L(X) denote the label of an *entity* X, where an entity is either a subject or an object. The *BLP security conditions* are:

- A subject S may read object O only if L(O) ≤ L(S). In other words, subjects are not allowed to "read up."
- A subject S may write object O only if L(S) ≤ L(O). In other words, a subject may not "write down."

Do the BLP security conditions enforce the BLP policy? First, note that a subject can never directly read an object for which it is not cleared. The first condition guarantees this. Second, a subject must never be able to learn information about some highly-labeled object O by reading another low-labeled object O'. Note that this is only possible if some other subject first reads O then writes O'. By the two conditions, a read then write by S entails L(O) ≤ L(S) ≤ L(O'). But then O actually has a lower label than O', so no information can have leaked.

The above was considered a significant result when it first was proved. But there are still some problems with the BLP formulation of MLS. These include:

- It is possible that the security level for an entity could be changed in mid-operation. This change could violate the information-flow constraints we wish to preserve.
- The model is concerned with only confidentiality, not integrity. For example, subjects can "write up". Thus, a subject that cannot read an object is permitted to make changes to that object; this is called a "blind write". Yet it makes little sense to trust a subject to modify the information contained in an object, if that subject is not trusted to read the information contained in the object.
- A subject S that wants to write object O is not allowed to do so if L(O) < L(S). The subject must login at a lower level than his or her clearance. It is annoying for principals to be forced to decide, at the time they login, what rights they will need. But it is also a nice application of the Principle of Least Privilege.
- Some processes must be allowed to violate the BLP security conditions. For example, an encryption program takes secret information and outputs encrypted but unclassified information. Similarly, to do accounting, programs that may access confidential data produce summary (billing) information that is not confidential. Such programs would seem to be violating the no "write down" condition. To handle these sorts of cases, we postulate the existence of *trusted subjects*, which are not restricted to the BLP security conditions. This introduces a potential vulnerability into the system, however. A program masquerading as a trusted subject could, in fact, be a Trojan horse, which is what we were trying to avoid in the first place.

Some real-world systems, including SELinux and TrustedBSD, combine MAC and DAC policies. In such cases, an operation is allowed only if both the MAC policy and the DAC policy both permit the operation.

MAC Example 2. Domain Type Enforcement

One way to address the problem of trusted subjects in MLS is to introduce a new kind of matrix. It looks like the access control matrix, but this is only a superficial resemblance. The rows in this matrix correspond to *domains* and the columns to *types*. Each entry contains a set of access rights. An entry [s,t] is the maximum permissions that domain s has with respect to an object of type t. In contrast to an access control matrix, this type enforcement matrix does not have commands associated with it. It cannot be manipulated by owners of objects; instead, it is controlled by system administrators. This makes it MAC, as opposed to DAC.

Note that a type enforcement matrix allows us to encode more than a lattice. For example, information flow is not necessarily transitive, and the matrix lets us express this, whereas the lattice does not. Consider the following example: a program sends data to an encryption routine that then sends encrypted data to the network. We would like an application program to be able to write to the encryption routine. The encryption routine should be able to read from the application program and write encrypted data to the network, and the network should be able to read from the encryption routine. The matrix is as follows:

	Application Program	Encryption	Network
Application Program		write	
Encryption	read		write
Network		read	

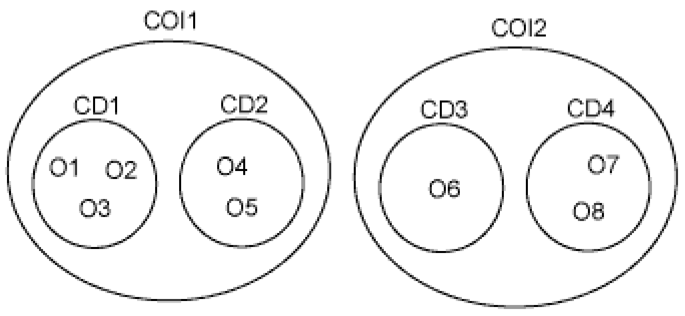
This matrix provides stronger constraints than simply making the encryption routine a trusted subject. A trusted subject can do what it wants, but here we make the encryption program's access rights more restrictive. Thus, if we still wish to do an analysis of the encryption program (e.g. to make sure any data that it writes is encrypted), we don't know need to worry about it writing anywhere other than to the network, so the scope of the analysis is narrowed (and therefore the analysis is easier.)

MAC Example 3. Chinese Wall

MLS is appropriate for national security confidentiality policies, and it is sometimes appropriate for business confidentiality policies. Consider a microprocessor company's plans for its next-generation chip. The company might consider these plans Top Secret and desire an access control mechanism that can prevent leakage of this sensitive information.

Other business confidentiality policies do not exhibit such close correspondence to MLS. Consider an investment bank. It employs consultants who both advise and analyze companies. When advising, such consultants learn secret information about a company's finances that should not be shared with the public. The consultant could exploit this *insider information* while performing analysis, to profit either himself or other clients. Such abuse is prohibited by law.

Brewer and Nash (1989) developed a MAC policy for this scenario, calling it *Chinese Wall* by analogy to the Great Wall of China. The intuition is that an unbreachable wall is erected between different parts of the same company; no information may pass over or through the wall. In the Chinese Wall policy, we (as usual) have have objects, subjects, and users. However, objects are now grouped into *company datasets* (CDs). For example, an object might be a file, and a company dataset would then be all of the files related to a single company. Company datasets are themselves grouped into *conflict of interest classes* (COIs). For example, one COI might be the set of all companies in the banking industry, and another COI might be all the companies in the oil industry.



The original security conditions for Chinese Wall given by Brewer and Nash were overly restrictive, and we omit them here. Sandhu (1992) later gave the following (less restrictive) conditions. Note that these conditions require the tracking the set of read objects for each user and subject.

1. A user U may read object O only if U has never read any object O' such that:
    a. COI(O) = COI(O'), and
    b. CD(O) ≠ CD(O').
2. A subject S associated with user U may read object O only if U may read O.
3. A subject S may write object O only if:
    a. S may read O, and
    b. S has never read an object O' such that CD(O) ≠ CD(O').

The first two conditions guarantee that a single user never breaches the wall by reading information from two different CDs within the same COI. The third condition guarantees that two or more users never cooperatively breach the wall by performing a series of read and write operations. Suppose that S1 has previously read from CD1, and S2 has previously read from CD2. Consider the following sequence of operations, based on the figure above.

- S1 reads information from an object in CD1.
- S1 writes that information to object O6 in CD3.
- S2 reads that information from O6.

At the end of this sequence, S2 would have read information pertaining to both CD1 and CD2, which would violate the Chinese Wall policy since both CDs are in the same COI. But Condition 3b prevents the write operation by restricting when a subject may write: once a subject reads two objects from different CDs, that subject may never write any object. So for read--write access, a user must create a distinct subject for each CD. For read-only access, a user can create a single subject to read from several COIs.

MAC Example 4. Clark-Wilson

The policies we have examined so far have been primarily concerned with maintaining secrecy of data. We are also interested in policies that ensure integrity of data. For instance, a bank is probably much more interested in ensuring that customers not be able to change account balances than it is in ensuring that account balances be kept secret. Although the latter is desirable, unauthorized data modification can usually cause much more harm to a business than inadvertent disclosure of information.

In 1987, Clark and Wilson examined how businesses manage the integrity of their data in the "real world" and how such management might be incorporated into a computer system. In the commercial world, we need security policies that maintain the integrity of data. In this environment, illegal data modification occurs due to fraud and error. And there are two classical techniques to prevent fraud and error.

The first technique is the principal of *well-formed transactions*. Users may not manipulate data arbitrarily, but only in constrained ways that preserve or establish its integrity. An example of this is double-entry bookkeeping. This involves making two sets of entries for everything that happens. Another example is the use of logs. Rather than erasing mistakes, the sequence of actions that reverses the mistake is performed and recorded on the log. A record of everything that has occurred is maintained. Using well-formed transactions makes it more difficult for someone to maliciously or inadvertently change data.

Another technique to prevent fraud is the principal of *separation of duty*. Here, transactions are separated into subparts that must be done by independent parties. This works to maintain integrity as long as there is no collusion between agents working on different subparts. A typical transaction might look as follows:

- A purchasing agent creates an order. The agent sends the order to both the supplier and the receiving agent.
- The supplier ships the goods to the receiving department. The receiving clerk checks the goods that were received against the original order and updates the inventory records.
- The supplier also sends an invoice to the accounting department. The accountant checks the invoice against both the original order and what the shipping clerk said was received. If they match, the accountant pays the bill.

In this scenario, no one person can cause a problem that will go unnoticed. The separation of duty rule being employed here is: A person who can create or certify a well-formed transaction may not execute it.

The question now becomes: How can we employ these commercial principles in computer systems? Clark and Wilson gave a set of rules for doing so. They postulate *trusted procedures* (TPs), which are programs that implement transactions. We summarize the Clark-Wilson rules as:

1. All subjects must be authenticated.
2. All TPs (and the operations they perform) must be logged---i.e., must be auditable.
3. All TPs must be approved by a central authority.
4. No data may be changed except by a TP.
5. All subjects must be cleared to perform particular TPs by a central authority.

Note how these rules exemplify the gold standard: they address authentication, audit, and (the final three rules) authorization.

Two other features of the Clark-Wilson model are:

- Data managed by a system may be either *constrained*, meaning it is governed by the Clark-Wilson rules, or *unconstrained*, meaning that it is not governed by the rules.
- A system may implement *integrity verification procedures* (IVPs), which test whether data is in a valid state. All TPs must guarantee the truth of IVPs.

Role-based Access Control

In the real world, security policies are dynamic. Access rights, whether discretionary or mandatory, need to change as the responsibilities of users change. This can make management of rights difficult. When a new user is authorized for a system, the appropriate rights for that user must be established. When a user changes job functions, some rights should be deleted, some maintained, and some added.

Role-based access control (RBAC) addresses this problem by changing the underlying subject--object model. A *role* is a job function or title---i.e., a set of actions and responsibilities associated with a particular working activity. Now, instead of an access control policy being a relation on subjects, objects, and rights, a policy is a relation on roles, objects, and rights; this is called a *right assignment*. For example, the role "513 TA" might be assigned the right to grade 513 homeworks. Further, subjects are now assigned to roles; this is called a *role assignment*. Each subject may be assigned to many roles, and each role may be assigned to many subjects. Finally, roles are hierarchical. For example, the role "513 Professor" should have all the rights that a "513 TA" does, and more.

Roles are similar to groups in Unix file system DAC, with two important distinctions. First, a group is a set of users, whereas a role is a set of rights. Second, a user is always a member of a group, whereas a subject may activate or deactivate the rights associated with any of the subject's roles. This enables finer-grained implementation of the Principle of Least Privilege. Subjects may login with most of their roles deactivated, and activate a role only when the rights associated with the role are necessary.

Nearly all real-world systems (including most operating systems and database systems) implement some form of RBAC. Either discretionary or mandatory policies can be built using RBAC as the underlying model.