Chapter 2
Basic Notions on Access Control

Abstract Security certifications deal with security-related properties and features of software products. In this chapter we introduce some basic concepts on software systems features related to security and dependability. Such features are often designated with the acronym *CIA*: they include *Confidentiality* features, ensuring that only authorized users can read data managed by the software system, *Integrity* features, ensuring that only authorized users can can modify the software system's resources, and *Availability* features, ensuring that only authorized users can use the software system. In this chapter we shall introduce some concepts underlying some software security features which are an important target for the software certification process. Specifically, we shall focus on access control subsystems, which are among the most widespread security features added to operating systems, middleware and application software platforms. Such subsystems will be among the targets of certification discussed in this book.

2.1 Introduction

The software security field is a relatively recent one;[1] but today software security is so wide notion that has come to mean different things to different people. For our purposes, we shall loosely define it as the set of techniques aimed at designing, implementing and configuring software so that it will function as intended, even under attack.

Security features, such as access control systems [6], are added to software products in order to enforce some desired security properties. Although the notion of specific security-related features and subsystems is an important one, it is much easier for such features to protect a software product that

[1] According to Gary McGraw [4], academic degrees on software security appeared as recently as 2001. Our own online degree course on Systems and Network Security at the University of Milan (`http://cdlonline.unimi.it`) was started in 2003.

is fault-free than one internally riddled with faults which may cause failures which prevent security features from doing their job.

Our notion of software security is therefore twofold: firstly, software security is about protecting "plain" software systems by adding to them a number of specific security features; secondly, software security is about programming techniques for developing secure software systems, designing and coding them to withstand (rather than prevent) attacks. The former notion of software security follows naturally from a system-centric approach, where access to software systems "from outside" must be controlled and regulated. The second notion, instead, relies on the developers' insight and focuses on methods for identifying and correcting dangerous faults which may be present in the software itself.

Most modern software systems do include some security features, but adding such features does not guarantee security *per se*. Indeed, software security is a system-wide issue that includes both adding security features (such as an access control facility) and achieving security-by-coding (e.g., via robust coding techniques that make attacks more difficult).

We can further clarify this distinction via a simple example. Let us assume that a web server has a buffer overflow fault [2], and that we want to prevent a remote attacker from overflowing the buffer by sending to the server an oversize HTTP `GET` request. A way to prevent this buffer overflow attack could be adding a software feature to the web server, a `monitor` function that observes HTTP requests as they arrive over port 80, and drops them if they are bigger than a pre-set threshold. Another way to achieve the same result consists in fixing the web server source code to eliminate the buffer overflow fault altogether. Clearly, the latter approach can only be adopted when the server's source code is available and its operation is well understood.

In most organizations, software security is managed by system administrators who set up and maintain security features such as access control and intrusion detection systems, firewalls and perimeter defenses, as well as antivirus engines. Usually, these system administrators are not programmers, and their approach is more oriented to adding security features to protect a faulty software than to correcting the faults in the software that allow attackers to take advantage of it.

Furthermore, software development projects (like any other type of project) have schedules and deadlines to respect for each step of the development process. The pressure put on development teams make many developers care for little else than making the software work [2], neglecting verification of security properties.

As we shall see, security certifications try to deal simultaneously with both aspects of software security: they certify the presence of some security features as well as the outcome of testing their functionality. From this point of view security is an emergent property of the entire software system rather than

[2] The reader who is unaware of what a buffer overflow fault is can skip this example and come back to it after reading Chapter 4.

a set of individual requirements. This is an important reason why software security must be part of a full lifecycle-based approach to software development. In this chapter we shall introduce some concepts underlying software security features which are an important target for the security certification process. Specifically, we shall focus on access control subsystems, which are among the most important security features added to operating systems, middleware and application software platforms. Such subsystems will be among the targets of certification we shall discuss in the next chapters.

2.2 Access Control

Access Control (AC) is the ability to allow or deny the use of resources. In other words, access control decides who *subject* is authorized to perform certain operations on a given *object* and who is not. The notion of controlling access is independent of the nature of objects and subjects; objects can be physical resources (such as a conference room, to which only registered participants should be admitted) or digital ones (for example, a private image file on a computer, which only certain users should be able to display), while subjects can be human users or software entities.

Generally speaking, computer users are subject to access control from the moment they turn on their machines, even if they do not realize it. On a computer system, electronic credentials are often used to identify subjects, and the access control system grants or denies access based on the credential presented. To prevent credentials being shared or passed around, a two-factor authentication can be used. Where a second factor (besides the credentials) is needed for access. The second factor can be a PIN, or a biometric input. Often the factors to be made available by a subject for gaining access to an object are described as

- something you have, such as a credential,
- something you know, e.g. a secret PIN, or
- something you are, typically a fingerprint/eye scan or another biometric input.

Access control techniques are sometimes categorized as either *discretionary* or *non-discretionary*. The three most widely recognized models are Discretionary Access Control (DAC) , Mandatory Access Control (MAC), and Role Based Access Control (RBAC). MAC and RBAC are both non-discretionary.

2.2.1 Discretionary Access Control

The Trusted Computer System Evaluation Criteria (TCSEC) [5] defines the Discretionary Access Control (DAC) model "a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. DAC is *discretionary* in the sense that a subject with a certain access permission is capable of passing on that permission (perhaps indirectly) to any other subject (unless explicitly restrained from doing so by mandatory access control)". The Orange Book also states that under DAC "an individual user, or program operating on the user's behalf, is allowed to specify explicitly the types of access other users (or programs executing on their behalf) may have to information under the user's control". DAC provides a very important security property: subjects to have full control over their own objects. Subjects can manipulate their objects in a variety of ways based on their *authorizations*.

Authorization determines which *actions* a subject can do on the system; the semantics of actions depends on the nature of the objects involved. For instance, within an operating systems platform, actions are variations or extensions of three basic types of access: `Read` (`R`), where the subject can read file contents or list directory contents, `Write` (`W`), where the subject can change the name or the contents of a file or directory, and `Execute` (`X`), where, if the object is a program, the subject can cause it to be run. As we shall see below, these permissions are implemented differently in systems based on Discretionary Access Control (DAC) than in Mandatory Access Control (MAC).

Throughout the book, we shall repeatedly mention DAC, because many operating systems today, including Unix and Unix look-alikes like Linux, use DAC as part of their file protection mechanisms. DAC has also been adopted for controlling access to networking devices such as routers and switches. DAC's high flexibility enables users to have full control to specify the privileges assigned to each object.

A widely used framework for implementing DAC is the Matrix access model. Formalized by Harisson, Ruzzo, and Ullman (HRU) [3], the Matrix access model provides a conceptual abstraction which specifies the access rights that each subject s has over each object o. The Matrix access model is composed of rows representing the subjects, and columns representing the objects. Each intersection cell between a subject s and an object o represents the access rights for s over o. In other words, the access rights specified by the cell ensure what type of access a subject has over an object (e.g. write access, read access, execution access). An example of an access matrix is shown in Figure 2.1. In this example, the access rights that are considered are (R) read, (W) write, and (X) execute. An empty cell means that a specific subject s has no right on a specific object o, conversely if a cell is assigned all the access rights, it means that a specific subject s has full control over a specific object o.

	O1	O2	O3	On
S1	R W X		R	W
S2	W	R W	R W X	
S3	R W X	R W	R	W
Sn		W	R X	R W

Fig. 2.1: Access matrix

2.2.1.1 Access Control Lists

Access Control Lists (ACLs) are data structures widely used to implement both discretionary and mandatory access control models. Often, operating systems implement DAC by maintaining lists of access permissions to be attached to system resources. Each object's access list includes the subjects allowed to access the object and the actions they are allowed to execute on the object. For example, the entry (`Bob`, `read`) on the ACL for a file `foo.dat` gives to subject Bob permission to read the file. In an ACL-based implementation of DAC, when a subject tries to execute an action on an object, the system first checks the object's list to see whether it includes an entry authorizing such action. if not, the action is denied.[3]

ACLs have been implemented in different ways in various operating systems, although a partial standardization was attempted in the (later withdrawn) POSIX security drafts .1e and .2c, still known as POSIX ACLs. In practice, ACLs can be visualized as tables, containing entries that specify individual subjects or group permissions to access system objects, such as processes, or a file. These entries are called `access control entries` (ACEs) within Microsoft Windows, Linux and Mac OS X operating systems.

2.2.2 Mandatory Access Control

While in DAC subjects are in charge of setting and passing around access rights, the *Mandatory Access Control* (MAC) model enforces access control based on rules defined by a central authority [6]. This feature makes MAC more suitable to setup enterprise-wide security policies spanning entire organizations. MAC has been defined by the TCSEC [5] as "a means of restricting

[3] A key issue in the efficient implementation of ACLs is how they are represented, indexed and modified. We will not deal with this issue in detail here, even if such implementation details may well be a target for a software security certification.

access to objects based on the *sensitivity* (as represented by a label) of the information contained in the objects and the formal authorization (i.e., *clearance*) of subjects to access information of such sensitivity".

In MAC access control system all subjects and objects are assigned different sensitivity labels to prevent subjects from accessing unauthorized information. The sensitivity labels assigned to the subjects indicate their level of trust, whereas the sensitivity labels assigned to the objects indicate the security clearance a subject needs to have acquired to access them. Generally speaking, for a subject to be able to access an object, its sensitivity level must be at least equal or higher than the objects' sensitivity level.

One of the most common mandatory policies is *Multilevel Security* (MLS), which introduces the notion of classification of subjects and objects and uses a *classification lattice*. Each class of the lattice is composed of two entities, namely:

- Security Level (L): which a hierarchical set of elements representing level of data sensitivity

  Examples:

  Top Secret (TS), Secret (S), Confidential (C), Unclassified (U)

  $$TS > S > C > U \tag{2.1}$$

  Essential (E), Significant (S), Not important (NI)

  $$E > S > NI \tag{2.2}$$

- Categories (C): a set of non-hierarchical elements that represent the different areas within the system

  Example:

  Accounting, Administrative, Management etc.

By combining the elements of the two components one obtains a partial order operator on security classes, traditionally called `dominates`. The `dominates` operator represents the relationship between each pair (L, C) with the rest of the pairs. The `dominates` relation (whose notation is the sign $\succeq$) is defined as follows:

$$(L1, C1) \succeq (L1, C2) \Longleftrightarrow L1 \geq L2 \wedge C1 \supseteq C1 \tag{2.3}$$

To better understand the relation among the different classes defined in Eq. 2.3, let's examine the structure created by the `dominates` relation $\succeq$. It is a lattice, often called *classification lattice*, which combines together the

Security Classes (SC) and the relations between them. Being a lattice, the classification structure satisfies the following properties:

Reflexivity of $\succeq$ $\quad \forall x \in SC : x \succeq x$
Transitivity of $\succeq$ $\quad \forall x, y, z \in SC : x \succeq y, y \succeq z \Longrightarrow x \succeq z$
Antisymmetry of $\succeq$ $\quad \forall x, y \in SC : x \succeq y, y \succeq x \Longrightarrow x = y$
Least upper bound (lub) $\quad \forall x, y \in SC : \exists! z \in SC$

- $z \succeq x$ *and* $y \succeq z$
- $\forall t \in SC : t \succeq x$ *and* $t \succeq y \Longrightarrow t \succeq z$

Greatest lower bound (glb) $\quad \forall x, y \in SC : \exists! z \in SC$

- $x \succeq z$ *and* $y \succeq z$
- $\forall t \in SC : x \succeq t$ *and* $y \succeq t \Longrightarrow z \succeq t$

Figure 2.2 shows an example of a classification lattice with two security levels: Classified (C), Unclassified (U), and two categories: Management and Finance. By looking closely to the figure we can notice that the sensitivity and the importance of the information increases as we move upward in the lattice. For instance, starting from the bottom of the lattice, we have Unclassified information U that does not belong to any category, and by moving along any of the two side lines we find unclassified information. However, this time the information belongs to a specific category, which gives this information more importance. If we move upward in the lattice we get to more important classified information.

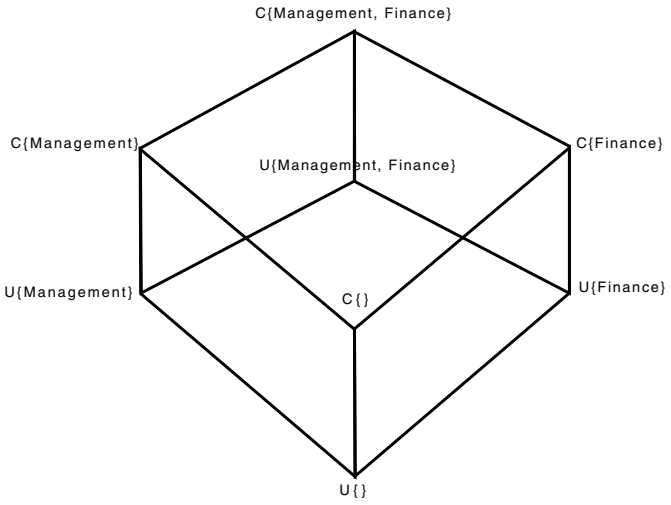

Fig. 2.2: Classification Lattice

Two well-known security models that use a mandatory access control approach are the *Bell-LaPadula*, and the *Biba* model.

2.2.2.1 Bell-LaPadula Model

The Bell-LaPadula (BLP) model defines a mandatory security policy based on information secrecy. It addresses the issue of confidential access to classified information by controlling the information flow between security classes.

BLP partitions the organizational domain into *security levels* and assigns a *security label* to each level. Furthermore, each subject and object within the system need to be associated with one of those predefined levels and therefore associated with a security label. The security labels denominations may change from one organization to another, based on the terminology used in each organization. For instance in the context of military organizations, it would make sense to have labels like *Top secret, Secret, Classified*, while in a business organization we may find labels of the type *Board Only* which specifies the type of information that only the board of directors can access and *Managerial* which specifies the information that can be accessed by managers and so on.

A major goal of BLP is preventing the information to flow to lower or incompatible security classes [6]. This is done by enforcing three properties:

- **The Simple Security Property :** A subject can only have read access to objects of lower or equal security level (no read-up). For instance looking at the example in Figure 2.3, we can see that Subject *S1* can read both objects *O2* and *O3*, since they belong to a lower security level, the same for subject *S2*, it can read object *O2* since it belongs to the same security level, whereas it is not authorized to read object *O1*, since it belongs to a higher security level
- **The *-property (star-property):** A subject can only have write access to objects of higher or equal security level (i.e, no *write-down* is allowed). The example shown in Figure 2.4 depicts this property, where subject *S2* can write both in object *O1* and *O2* since they belong respectively to a higher and to the same security level, while subject *S1* can write only to object *O1* since it belongs to the same security level.
- **The Discretionary Security Property :** This property allows a subject to grant access to another subject, maintaining the rules imposed by the MAC. So subjects can only grant accesses for the objects over which they have the necessary authorizations that satisfy the MAC rules.

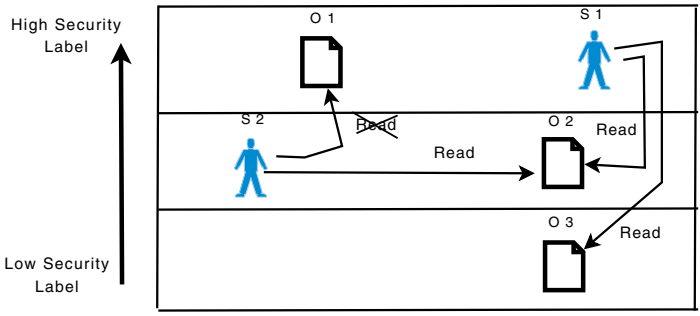

Fig. 2.3: BLP simple security property

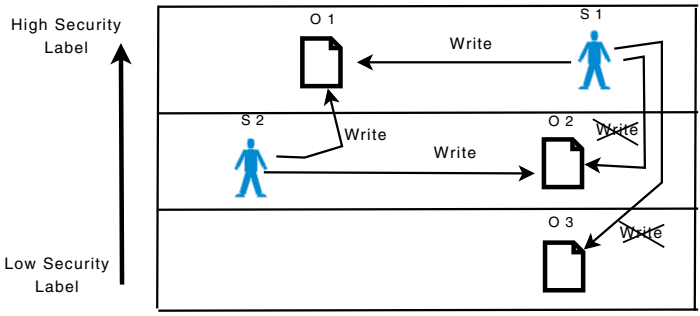

Fig. 2.4: BLP *-security property

2.2.2.2 Biba Model

Whereas the main goal of BLP is to ensure confidentiality, Biba's similar mechanism aims to ensure integrity. The basic idea behind Biba is to prevent low-integrity information to flow to higher-integrity objects, while allowing the opposite. Very often, the Biba model has been referred to as the reverse of the BLP model, since it exactly reverses the rules defined by the BLP. Before discussing the properties of the Biba model and looking to some examples, we once again remind the reader that the approach used by Biba is explicitly based on integrity rather than confidentiality or disclosure of information as in BLP. As mentioned earlier integrity refers to the ability of altering data objects. So the model prevents high integrity objects from being altered or changed by an unauthorized subject.

In a military environment, the Biba model will allow a subject with a high security label to insert mission-critical information like a military mission's target and its coordinates, whereas subjects with lower security labels can only read this information. This way the integrity of the system can be preserved more efficiently since the flow of information goes only from higher to lower levels [1]. The Biba model guarantees the following properties:

- **Simple Integrity Property :** A subject can only have read access to objects of higher or equal security level (no read-down).
- **Integrity *-property (star-property):** A subject can only have write access to objects of lower or equal security level (no write-up).

2.2.3 Role Based Access Control

Instead of providing access rights at user or group level as in DAC, *Role-Based Access Control (RBAC)* uses the roles assigned to users as a criteria to allow or deny access to system resources. Typically, RBAC models define users as individuals with specific roles and responsibilities within an organization. This relationship creates a natural mapping based on the organization's structure, and thus assigns access rights to each role rather than to each individual. The relation between roles and individuals is many-to-many, like the one between roles and system resources. A specific role may include one or more individuals, and at the same time a specific role can have access rights to one or more resources. Figure 2.5 depicts these relations.

The main idea behind RBAC is to group privileges in order to give organizations more control in enforcing their specific security polices. In other words, within an organization, the access control system deals with the job function of individuals rather than with their real identities. When permitting or denying access to a specific resource, the access right is intended to permit or deny access to a specific role and not to a specific individual.

Within the RBAC, a role is defined as "a set of actions and responsibilities associated with a particular working activity" [6]. The scope or the extent of the role can be limited to specific task or mission to accomplish, for instance *processing a client request, or preparing a report*, or it can concern a user's job like for instance *manager, director*.

2.3 Conclusions

Today's information systems and platforms include security features aimed at protecting confidentiality and integrity of digital resources. Controlling access to resources across an entire organization is a major security challenge. Access control models address this challenge by defining a set of abstractions capable of expressing access policy statements for a wide variety of information resources and devices. Access control systems are among the most widespread security features added to operating systems, middleware and application software platforms; therefore, certification of their security properties is of paramount importance. Many operating systems today, including Unix and Unix look-alikes like Linux, use DAC as part of their protection

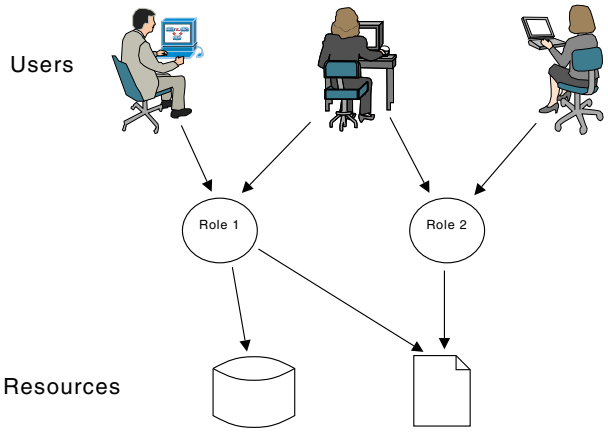

Fig. 2.5: Role-based access control users-roles-resources relations

mechanisms. DAC has also been adopted for controlling access to networking devices such as routers and switches. DAC's high flexibility enables resource owners to specify privileges assigned to each object.

References

1. E.G. Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall PTR, 1994.
2. M.G. Graff and K.R. Van Wyk. *Secure Coding: Principles and Practices*. O'Reilly, 2003.
3. M.A. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.
4. G. McGraw. From the ground up: The dimacs software security workshop. In *IEEE Security and Privacy*, volume 1, pages 59–66, March 2003.
5. USA Department of Defense. *DEPARTMENT OF DEFENSE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA*. USA Department of Defence, 1985.
6. P. Samarati and S. De Capitani di Vimercati. *Foundations of Security Analysis and Design*, chapter Access Control: Policies, Models, and Mechanisms, pages 137–196. Springer Berlin / Heidelberg, 2001.