ELSEVIER Data & Knowledge Engineering 25 (1998) 199–216

DATA & KNOWLEDGE ENGINEERING

Data security

Elisa Bertino*

Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39, 20135 Milano, Italy

Received 18 November 1997; accepted 18 November 1997

Abstract

Maintaining data quality is an important requirement in any organization. It requires measures for access control, semantic integrity, fault tolerance and recovery. Access control regulates the access to the system by users to ensure that all accesses are authorized according to some specified policy. In this paper, we survey the state of the art in access control for database systems, discuss the main research issues, and outline possible directions for future research.

Keywords: Access control; World Wide Web; Data warehousing

1. Introduction

As organizations increase their reliance on database systems for both day-by-day operations and decision making, security of data managed by these systems becomes crucial. Damages and misuse of data affect not only a single user or an application; rather, they may have disastrous consequences on the entire organization. Security breaches are typically categorized into *unauthorized data observation*, *incorrect data modification*, *data unavailability*. Unauthorized data observation results into disclosure of information to users not entitled to gain access to such information. All organizations we may think of, ranging from commercial organizations to social organizations, such as healthcare organizations, to military organizations, may suffer heavy losses from both financial and human point of views upon unauthorized data observation. Incorrect modifications of data, either intentional or unintentional, result in an inconsistent database state. The database is no longer correct. Any use of incorrect data may again result in heavy losses for the organization. When data are unavailable, information that are crucial for the proper functioning of the organization are not readily available when needed.

A complete solution to the data security problem must thus meet the following three requirements: (i) *secrecy or confidentiality*—it refers to the protection of data against unauthorized disclosure; (ii) *integrity*: it refers to the prevention of unauthorized or improper data modification; (iii) *availability*: it refers to the prevention and recovery from hardware and software errors and from malicious data

* E-mail: bertino@dsi.unimi.it

denials making the database system not available. These three requirements arise in practically all application environments. Consider a database storing payroll information. It is important that salaries of individual employees are not released to unauthorized users, that salaries are modified by only the properly authorized users, and that paychecks are printed on time at the end of each pay period. Similarly, in a military environment, it is important that the target of a missile is not made accessible to unauthorized users, that the target is not arbitrarily modified, and that missile is launched when it is fired.

Data protection is ensured by different components of a DBMS. In particular, the *access control mechanism* ensures data secrecy. Whenever a user tries to access an object, the access control mechanism checks the right of the user against a set of authorizations, stated usually by some security administrator. An *authorization* states which user can perform which action on which object. Authorizations are stated according to the security policies of the organization. Data integrity is jointly ensured by the access control mechanism and by semantic integrity constraints. Whenever a user tries to modify some data, the access control mechanism verifies that the user has the right to modify the data, whereas the semantic integrity subsystem verifies that the updated data are semantically correct. Semantic correctness is verified against a set of conditions, or predicates, that the database state must verify. Finally, the recovery subsystem and the concurrency control mechanism ensure that data are available and correct despite hardware and software failures and despite data accesses from concurrent application programs.

In this paper, we will focus on access control mechanisms and related authorization models. We refer the reader to [13] for an extensive discussion on transaction models and recovery and concurrency control mechanisms, and to any database textbook for details on semantic integrity control. It is important to note that an access control mechanism must rely for its proper functioning on some authentication mechanism. Such a mechanism identifies users and confirms their identities. Moreover, data may be encrypted when transmitted over the network in the case of distributed database systems. Both authentication and encryption techniques are widely discussed in the current literature on computer network security and we refer the reader to [19] for details on such topics. In the remainder of the discussion, we will focus on the state of the art in access control mechanisms for databases and discuss open research issues. We do not attempt to be exhaustive, but try to articulate the reasons for the approaches we believe to be promising.

Recent research efforts in the area of access control models for DBMS can be classified into four main directions. The first direction is related to discretionary access controls in relational DBMSs. Recent research proposals have extended the capabilities of current authorization models with a variety of features, such as negative authorizations [12], role-based and task-based authorization models [27], and temporal authorizations [8,9]. Related to those extensions is the problem of developing appropriate tools and mechanisms to support those extended models.

The main drawback of discretionary access control models is that, under these models, they do not impose any control on how information is propagated and used once it has been accessed by users authorized to do so. This weakness makes discretionary access controls vulnerable to malicious attacks, such as Trojan Horses embedded in application programs. A Trojan Horse is a computer program with an apparent or actually useful function, which contains additional *hidden* functions that exploit the legitimate authorizations of the invoking process. As an example, consider a user U who is authorized to read file File1. Suppose that another user, say U′, exists who is not authorized to read File1. Suppose that U′, even though is not authorized to do so, wishes to gain access to File1. A way to accomplish this illegal read operation through a Trojan Horse is as follows. U′ creates a second file, say File2, and gives user U write access to such file (note that U may not even know that s/he has write access to such file). Then U′ creates a utility program, for example a file sorting program, that in addition to ordering the content of the input file copies the content of such file into File2. At this point, U' passes such a program to U specifying that the function of the program is to sort files. Suppose now that U runs such program of File1. Since the program executes on behalf of U, every access is checked against the authorizations of U. Because U is authorized to read File1 and to write File2, the illegal transfer of information from File1 to File2 is performed. At this point, user U′ by simply accessing File2 is able to gain access to the information contained in File1. Other more sophisticated means exist based on covert channels, by which illegal access to data can be gained. A *covert channel* is any component or feature of a system that is misused to encode or represent information for unauthorized transmission, without violating the stated access control policy. A large variety of components or features can be exploited to establish covert channels, including the system clock, operating system interprocess communication primitives, error messages, the existence of particular file names, and so forth.

The second research direction, thus, deals with protecting data against intrusions through sophisticated means, such as Trojan Horses and covert channels. This direction has resulted in the development of mandatory access control models, based on information classification, some of which have also been incorporated in commercial products [21].

A third direction concerns the development of adequate authorization models for advanced DBMSs, like object-oriented DBMSs and object-relational DBMSs. These DBMSs are characterized by data models that are richer than the relational model. Object data models include notions such as inheritance hierarchies, composite objects, versions and methods. Therefore, authorization models, both discretionary and mandatory, developed for relational DBMSs must be properly extended to deal with those additional modeling concepts.

Finally, a fourth direction is related to access control models for advanced data management systems and applications, such as data made available through World Wide Web (WWW), digital libraries, and data warehousing systems. No research results, with the exception of a discretionary access control proposed for WWW pages [26], have been reported for access control models and mechanisms for such applications. Other results, also for security in WWW, have been reported. However, such proposals mainly deal with the security of data communications and user authentication; they do not cover features required to support access control policies typically required by data management applications.

The remainder of this paper is organized as follows. Sections 2 and 3 discuss discretionary and mandatory access control models for relational DBMSs. Section 4 discusses access control in the framework of object-oriented DBMSs by outlining the main protection requirements for these DBMSs and discussing proposed approaches. Section 5 discusses protection requirements for digital libraries. In particular, this section illustrates the several requirements introduced by these applications that are not addressed by traditional access control models. Finally, Section 6 briefly outlines some conclusions.

2. Discretionary access control

Access control mechanisms of current DBMSs are in most cases based on discretionary protection policies. Such policies govern the access of users to data on the basis of users' identity and authorization rules. They are discretionary in that they allow users to grant other users authorization to access the data. Because of such flexibility, such policies are adopted in many application environments and this is the reason why commercial DBMSs adopt such policies. An important aspect of discretionary access control is thus related to the *authorization administration policy*. Authorization administration refers to the function of granting and revoking authorizations. It is the function by which authorizations are entered (removed) into (from) the access control mechanism. Common administration policies include: the *centralized administration*, by which only some privileged users may grant and revoke authorizations; and the *ownership-based administration*, by which grant and revoke operations on a data object are issued by the creator of the object. The ownership-based administration is often extended with features for *administration delegation*. Administration delegation allows the owner of an object to assign other users the right to grant and revoke authorizations, thus enabling decentralized authorization administration. Most commercial DBMSs adopt the ownership-based administration with administration delegation. More sophisticated administration policies can be also devised as such the joint-based administration, by which several users are jointly responsible for authorization administration [14]. Those administration policies are particularly relevant for new applications, such as workflow systems and computer-supported cooperative work (CSCW).

In this section, we review some discretionary models proposed for relational DBMSs. We first describe the System R authorization model and some recently proposed extensions to it. We then briefly discuss role-based access control, a relevant extension to current authorization mechanisms.

2.1. Basic model

One of the first authorization models to be proposed for relational DBMSs is the model defined by Griffiths and Wade [17], in the framework of the System R DBMS [4]. Under this model, the objects to be protected are tables and views (also called virtual tables)[1]. Possible access modes users can exercise on tables are: *select* (to select tuples from a table), *insert* (to add tuples to a table), *delete* (to delete tuples from a table), and *update* (to modify existing tuples in a table). The same access modes are defined for views with the difference that some access modes may not be applicable to a view depending on the view definition. For example, most DBMSs do not allow delete, insert and update operations on views defined as a join. In such a case, the corresponding access modes are not applicable to the view. In the remainder, we use the term table to denote both views and real tables.

Authorization administration in the System R model is based on the ownership approach coupled with administration delegation. Any database user, authorized to do so, can create a new table. When a user creates a table, he becomes the owner of the table, and is solely and fully authorized to exercise all access modes on the table. The owner, however, can grant privileges on the table to other users. Authorizations can be granted with the *grant option*, meaning that the receiver can grant other users these authorizations.

The fact of supporting authorization administration delegation introduces some interesting issues concerning the semantics of the revoke operation. A user, to whom the administration right on a given object has been granted and then revoked, may have granted to another user an authorization to access

[1] There are usually other objects to be protected in a DBMS, such as application programs. We limit the discussion to the tables and views to simplify the presentation.

this object. The question is what happens to this authorization. The semantics of the revocation of an authorization from a user (revokee) by another user (revoker) is to consider as valid the authorizations that would have resulted had the revoker never granted the revokee the privilege. As a consequence, every time an authorization on a table is revoked from a user, a recursive revocation takes place to remove all authorizations which would have not existed had the revokee never received any authorizations for this table from the revoker.

We illustrate the revoke operation through the following example. Consider the sequence of grant operations for privilege p on table t represented in Fig. 1a through the use of a *grant graph.* A grant graph is related to a table t and an access mode a. In such a graph, every node represents a user; an arc from node u_i to node u_j denotes that u_i granted the authorization for access mode a on table t to user u_j. Moreover, each arc is labeled by a timestamp, indicating the time the grant has been issued. Timestamps play a fundamental role in ensuring the above semantics for the revoke operation. For the sake of simplicity, we assume that all authorizations are granted with the grant option. Suppose that Ann revokes the authorization on t from Jim. According to the semantics of the recursive revocation, the resulting authorization state has to be as if Jim had never received such authorization from Ann. If Jim had never received the authorization from Ann, he could not have granted the authorization to Sue (his request would have been rejected by the access control mechanism). Similarly, Sue could not have granted the authorization to Dave. Therefore, the authorizations granted by Jim to Sue and by Sue to Dave must be deleted. Note that the authorization granted by Jim to Pat does not have to be deleted since Jim could have granted it even if he had never received the authorization from Ann, because he holds an authorization granted by Chris. By contrast, the authorization granted by Chris to Jim could not have been used by Jim to grant the authorization to Sue, because the former authorization has been granted only at time 50, whereas the latter has been granted at time 40. The set of remaining authorizations, after the revoke operation, is illustrated in Fig. 1b.

The above model is the core of the access control mechanisms of several commercial DBMSs. In some systems, however, a different semantics for the revoke operation is adopted [11]. Under such

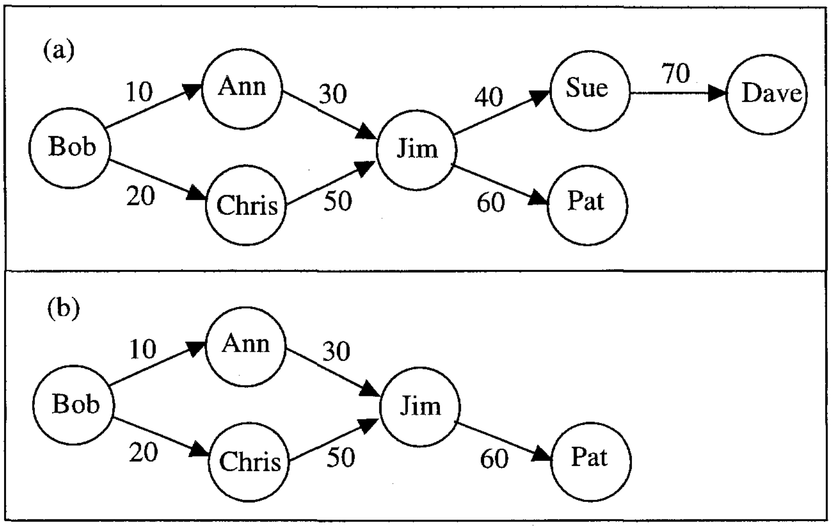

Fig. 1. (a) A grant graph example; (b) the grant graph example after Ann revokes the authorization from Jim.

revised semantics, an authorization granted by a user to another user is not revoked as long as the grantor has at least one administration authorization (independently from the timestamps of the two authorizations). An an example, consider the grant graph illustrated in Fig. 1 and suppose that Ann revokes the authorization on *t* from Jim. Under such revised semantics, the authorization granted by Jim to Sue is not revoked, because Jim still has another authorization (granted to him by Chris). Therefore, the authorization granted by Sue to Dave is not deleted.

2.2. Extensions

A number of extensions to the System R authorization model have been proposed. Most of those proposals aim at enriching the expressive power of the authorization model to support a large variety of application requirements.

A first extension deals with negative authorization [12]. The System R authorization model, as the models of most DBMSs, uses the *closed world* policy. Under this policy, whenever a user tries to access a table and no authorization is found in the system catalogs, the user is denied the access. Therefore, the lack of authorization is interpreted as no authorization. This approach has the major drawback that the lack of an authorization for a user on an object does not prevent this user from receiving this authorization any time in the future. Any user holding the right to administrate that object can grant any other user the authorization to access that object. The introduction of *negative authorizations* can overcome this drawback. An explicit negative authorization expresses a *denial* for a user to access a table under a specified mode. Conflicts between positive and negative authorizations are resolved by applying the *denials-take-precedence* policy under which negative authorizations override *positive* authorizations. That is, whenever a user has both a positive and a negative authorization for a given privilege on the same table, the user is prevented from using the privilege on the table. The user is denied access even if a positive authorization is granted after a negative authorization has been granted. Negative authorizations can also be used to temporarily block possible positive authorizations of a user, and to specify exceptions. For example, it is possible to grant an authorization to all members of a group, except to one specific member by granting the group the positive authorization for the privilege on the table and the given member the corresponding negative authorization. Such a model has been further extended with a more flexible conflict resolution policy, which is based on the concept of *more specific* authorization. Such a concept introduces a partial ordering relation among authorizations which is taken into account when dealing with conflicting authorizations. For example, the authorizations directly granted to a user are more specific than the authorizations specified for the groups to which the user belongs. Whenever two authorizations conflict, the most specific authorization prevails. Therefore, a negative authorization can be overridden by a positive authorization, if the latter is more specific than the former. If, however, two conflicting authorizations cannot be compared under the 'more specific' relation, the negative authorization prevails.

The notion of explicit denial has also been proposed in the context of the Sea View project [20]. In Sea View, authorizations can specify which users or groups are authorized to access the database tables and which users and groups are specifically denied for particular database tables. Unlike the positive authorizations, negative authorizations cannot specify an access mode. A special access mode, called 'null', is used to denote a negative authorization. If a user receives an authorization for the null access mode on a table, then the user cannot exercise any access mode on the table. Subjects of the authorizations can be both users as well as groups of users. Groups do not need to be disjoint. Possible conflicts between positive and negative authorizations are solved on the basis of the following policy: (i) authorizations directly granted to a user take precedence over authorizations specified for the groups to which the user belongs, and (ii) the authorization for the null access mode given to a user (group) overrides over any other authorization granted to the same user (group). In particular, if a user is explicitly denied access to a table, this denial overrides any authorization that the user may have either explicitly or as a member of a group. If a user has not been explicitly denied access to the table, the user may exercise those authorizations that he owns on the table together with authorizations for the table of any group to which he belongs (provided the group is not denied authorization to the table). However, a process operating on behalf of a user is constrained to be associated with at most one group to which its associated user belongs.

A second major extension deals with a more articulate semantics for the revoke operation. In the System R model, as in many DBMSs, whenever an authorization is revoked from a user, a recursive revocation takes place. This approach can be very disruptive. In many organizations the authorizations a user possesses are related to his particular task or function within the organization. If a user changes his task or function (for example if he is promoted), it is desirable to remove only the authorizations of this user, without triggering a recursive revocation of all the authorizations granted by this user. To support this requirement, a new type of revoke operation, called *non-cascading revoke*, is proposed. Whenever a non-cascading revoke operation is invoked, the authorizations granted by the user from whom the authorization is being revoked are not revoked; instead they are respecified as if they had been granted by the user requiring revocation. By providing two different types of revoke, cascading and non-cascading, the resulting access control mechanisms is thus able to better support a large variety of application requirements. To understand how non-cascading revocation works, consider the sequence of authorizations shown in Fig. 2a. Suppose that Jim issues a non-cascading revoke operation for the authorization previously granted to Sue. The effect of such operation is illustrated by the grant graph shown is Fig. 2b in which the authorization given by Sue to Dave has been respecified with Jim as the grantor. Thus, Dave retains the authorization given him by Sue. If by contrast Jim had issued a cascading revoke operation, both authorizations of Sue and Dave would have been deleted, as illustrated by Fig. 2c.

A third extension is related to the temporal duration of authorizations. In all systems, an authorization is *valid* from the time it is entered into the system, through a grant command, until it is explicitly removed through a revoke command. In many applications, however, permissions may hold only for specific time intervals. A further requirement concerns periodic authorizations. In many organizations, authorizations given to users must be tailored to the pattern of their activities within the organization. Therefore, users must be given access authorizations to data only for the time periods in which they are expected to need the data. An example of policy with temporal requirements is that 'all programmers can modify the project files every working day except Friday afternoons'. Authorization models of current DBMSs are still rather poor with respect to many security requirements. Even the simple above example cannot be directly supported by any current commercial DBMS. As a matter of fact, in most cases the access control policies must be implemented as code in application programs. Such an approach makes it very difficult to verify and modify the access control policies and to provide any assurance that these policies are actually enforced. An authorization model addressing such requirements has been recently proposed [8,9]. Under such a model, each authorization has a temporal interval of validity; an authorization is valid only in such period. When the interval expires,

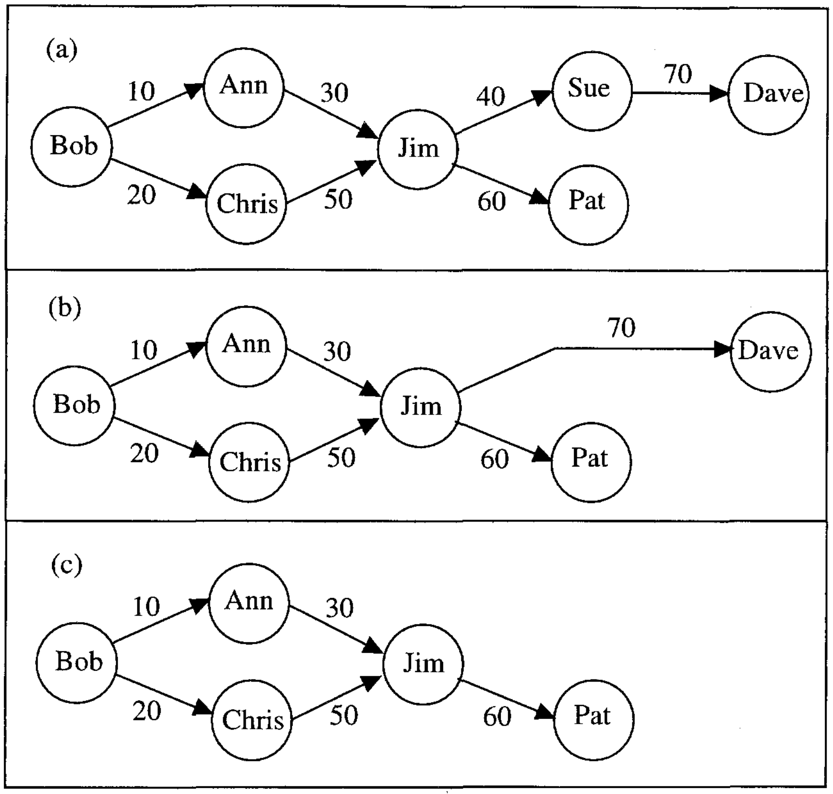

Fig. 2. (a) A grant graph example; (b) the grant graph example after Jim revokes the authorization from Sue without cascading; (c) the grant graph example after Jim revokes the authorization from Sue with cascading.

the authorization is automatically revoked without requiring any explicit revoke operations from the security administrators. The interval associated with an authorization may also be periodic, thus consisting of several intervals which are repeated in time. In addition, the model provides deductive temporal rules to derive new authorizations based on the presence or absence of other authorizations in specific periods of time. The resulting model provides a high degree of flexibility and is able to directly meet a large number of protection requirements that cannot be met by traditional access control models.

2.3. Role-based access control

Role-based access control (RBAC) models represent an important extension to access control models. Roles are strictly related to the organization and can be seen as a set of actions or responsibilities associated with a particular working activity. Under RBAC models, all authorizations needed to perform a certain activity are granted to the role associated with that activity, rather than being granted directly to users. Users are then made members of roles, thereby acquiring the roles' authorizations. User access to objects is mediated by roles; each user is authorized to play certain roles and, on the basis of the role, s/he can perform accesses on the objects. Because a role groups a (possibly large) number of related authorizations, authorization administration is greatly simplified. Whenever a user needs to perform a certain activity (or being assigned a certain activity), the user only needs to be granted the authorization of playing the proper role, rather than being directly assigned the required authorizations.

Although some current DBMSs support role-based policies, they do not exploit their full potential. In particular, several advanced RBAC models have been developed supporting among other features, role hierarchies by which roles can be defined as sub-roles of other roles, and constraints supporting separation of duty requirements [27]. An RBAC model supporting sophisticated separation of duty constraints has also been recently proposed for workflow systems [15]. Finally, some recent research results show that some advanced RBAC models can be used to simulate other access control policies, such as mandatory access controls [27].

3. Mandatory access control

Mandatory security policies control accesses to data by individuals on the basis of a predefined classification of *subjects* and *objects* in the system. Objects are the passive entities storing information, such as relations, tuples in a relation, or even elements of a tuple. Subjects are active entities performing data accesses. The classification is based on a number of *access classes*, also called *labels*, that are associated with every subject and object in the system. A subject is granted authorization to access a given object if and only if some relationship, depending on the access mode, is satisfied between the classifications of the subject and the object. An *access class* generally consists of two components: a *security level* and a *set of categories*. The security level is an element of a hierarchically ordered set. A very well known example of such a set is the one including the levels Top Secret (TS), Secret (S), Confidential (C) and Unclassified (U), where $TS > S > C > U$. The set of categories is an unordered set (e.g., NATO, Nuclear, Army). Access classes are partially ordered as follows. An access class c_i dominates ($\geq$) an access class c_j iff the security level of c_i is greater than or equal to that of c_j and the categories of c_i include those of c_j. Two classes c_i and c_j are said to be incomparable if neither $c_i \geq c_j$ nor $c_j \geq c_i$ holds.

The security level of the access class associated with a data object reflects the sensitivity of the information contained in the object, i.e., the potential damage which could result from unauthorized disclosure of the information. The security level of the access class associated with a user, also called *clearance*, reflects the user's trustworthiness not to disclose sensitive information to users not cleared to see it. Categories are used to provide finer grained security classifications of subjects and objects than classifications provided by security levels alone, and are the basis for enforcing *need-to-know* restrictions.

Access control in mandatory protection models is based on two principles, formulated by Bell and LaPadula [5], which are followed by all models enforcing a mandatory security policy:

No read-up: A subject can read only those objects whose access class is dominated by the access class of the subject.

No write-down: A subject can write only those objects whose access class dominates the access class of the subject.

Verification of these principles prevents information in a sensitive object to directly flow into objects at lower or incomparable levels. For instance, consider the Trojan Horse example discussed in the Introduction. If the user U′ is not authorized to read File1, under the mandatory access control File1 would have to be assigned an access class either higher than or incomparable with respect to the access class given to U′. But then a subject able to read File1, user U in the example, would not be able to write File2 (because of the no write-down restriction), and the Trojan Horse would therefore be prevented from performing the illegal read operation.

The application of mandatory access control policies to relational DBMSs has been extensively investigated over the past years. The introduction of such an access control model entails the solution of several difficult issues. Solutions to some of those issues have required extensions to the definition of the relational model itself, resulting in the so-called *multilevel relational model*, and to basic notions such as the notion of key of a relation. A multilevel relation is characterized by the fact that different tuples may have different access class. The relation thus results, partitioned into different security partitions, where each partition contains the tuples that have the same access class. Each partition has the same access class of the tuples it contains. A user having clearance c can read all tuples contained in partitions of access classes that are equal or lower than c; we call such set of tuples the *view* of the multilevel relation at level c. By contrast, a user having clearance c can write the tuples at access classes that are equal or higher than c. In some implementations of the multilevel relational model, write operations at higher access classes are not allowed for integrity reasons, even though they are not disallowed by the Bell and LaPadula principles. Such restriction is usually known as no-write-up restriction.

The multilevel relational model is further complicated if tuples are allowed to have attributes classified at different access classes. Each attribute of each tuple thus has an *attribute label*, denoting the access class of the attribute for the tuple, and a *tuple label*, which is lowest element in the set of access classes associated with the attributes of the tuple. An example of multilevel relation is illustrated in Fig. 3. The relation contains a number of additional attributes denoting the access class of each attribute in each tuple; we denote by the name L_A the attribute containing the access class of the attribute of name A. Moreover, an additional attribute, called TC denotes the access class of the tuple. For simplicity, in the example, we assume that there are only two access classes, Low and High with High $\geq$ Low.

When attributes of the same tuple have different access classes, the notion of view of a multilevel relation at an access class c requires some further extension. In particular, tuples are still partitioned according to their access classes. However, since a tuple may have potentially different access classes (one for each attribute), the same tuple may appear in several partitions. In practice if a tuple has an attribute with access class c, the tuple will appear in the partition of access class c. Moreover, all those attributes of the tuple with access classes higher than or incomparable to c are masked in the partition of access class c by substituting them with null values with access class c. Fig. 4 illustrates the views

Name	L_{Name}	Department	$L_{Department}$	Salary	L_{Salary}	TC
Bob	Low	D1	Low	15K	Low	Low
Ann	High	D2	High	25K	High	High
Sam	Low	D1	Low	20K	High	High

Fig. 3. An example of a multilevel relation.

Name	L_{Name}	Department	$L_{Department}$	Salary	L_{Salary}	TC
Bob	Low	D1	Low	15K	Low	Low
Sam	Low	D1	Low	-	Low	Low

View at access class Low

Name	L_{Name}	Department	$L_{Department}$	Salary	L_{Salary}	TC
Bob	Low	D1	Low	15K	Low	Low
Ann	High	D2	High	25K	High	High
Sam	Low	D1	Low	20K	High	High

View at access class High

Fig. 4. Views corresponding to the multilevel relation of Fig. 3.

at access classes Low and High of the multilevel relation illustrated in Fig. 3. In Fig. 4, the symbol '-' denotes a null value.

Multilevel relations must satisfy the following conditions:

- For each tuple in a multilevel relation, all the attributes of the primary key—in case the primary key consists of more than one attribute—must have the same access class.
- For each tuple in a multilevel and for each non-key attribute of the tuple, the access class associated with the non-key attribute of the attribute must dominate the access class of the key attributes of the tuple.

The above conditions ensures that no tuple exists in a view that has some of the key attributes equal to null.[2]

An important issue is to determine a suitable notion of primary key of a multilevel relation. In the standard relational model, each tuple is uniquely identified by the values of its key attributes. When access classes are introduced in a relation, there may be a need for the simultaneous (logical) presence of multiple tuples with the same values for the key attributes but with different access classes and with (possibly) different values for some non-key attributes. Such a case arises when some attribute values are masked with the null value. The simultaneous presence of the same tuple in different views is called *polyinstantiation* to denote the fact that the same (logical tuple) may exist in different instantiations in different views. All such instantiations are distinguished by their access classes. In such a context, the notion of primary key changes. The uniqueness constraint associated with the key in the standard relational model is changed into the constraint requiring that a key of a tuple be unique within each view, and not within the entire relation. This constraint is equivalent to state that the combination of key values and access class for a tuple must be unique within the entire relation.

Issues related to polyinstantiation and to the semantics of data manipulation operations in the presence of polyinstantiated tuples have been widely investigated. We refer the reader to [7] for a discussion on the various forms of polyinstantiation and operation semantics.

[2] We recall that an important constraint in the relational model is that the attributes in the key do not have null values.

4. Access control in object database systems

Traditional access control models, developed in the context of relational DBMSs, are not adequate for the protection of data in Object-Oriented DBMSs (OODBMSs). The semantic richness of the object-oriented data model introduces new requirements that traditional access control models do not address. In this section, we briefly discuss some issues arising in access control models for OODBMSs and solutions proposed.

4.1. Discretionary access control

As we mentioned in the introduction, discretionary access control subsystem is, today, a basic component of the majority of commercial DBMS. Existing authorization models, defined in the framework of RDBMSs, are not suitable for an object-oriented database system. Authorization models defined for RDBMS in general consider the relation, or the attribute, as the authorization unit, in the sense that authorizations are granted on relations or, in some cases, on relation attributes. Some RDBMSs also provide a view mechanism to select a subset of the tuples of a relation as an authorization unit. By contrast in OODBMSs, the unit of authorization must be an object, since objects are the access units. However, the authorization model should also support different levels of granularity for both performance and user convenience reasons. For example, it should be possible to grant authorization on only a single object, but also on an entire class, or even on an entire database. Second, an authorization model for OODBMS should take into account all semantic modeling constructs commonly found in object-oriented data models, such as composite objects and versions.

An object-oriented authorization model, satisfying the previous requirements, has been defined for the Orion OODBMS [25]. Other systems implement less sophisticated models or have no authorization at all. A key aspect of the Orion authorization model is the use of authorization implication rules supporting the derivation of new authorization (called *implicit*) from the ones explicitly specified by the users (called *explicit*). Implication rules are defined for objects, subjects and access modes. In particular, implication rules on objects support the derivation of authorizations from an object to all the objects semantically related to it. For example, a read authorization on the root of a version hierarchy[3] implies read authorization on all the versions in the hierarchy. However, it is also possible for an authorization to be granted on a single version of an object. The use of implication rules is thus instrumental in providing varying levels of granularity without performance penalties. Fernandez et al. [16] have also widely investigated issues related to access control models for OODBMSs. Their work focuses in particular on inheritance hierarchies. They propose that authorizations be inherited along inheritance hierarchies as a default policy. Whenever, an authorization must not be inherited in a subclass, a negative authorization must be explicitly specified for the subclass.

The above authorization models could be termed ‘structural authorization model’, in that they do not exploit the *encapsulation* property of the object-oriented approach. Encapsulation is one of the most important advantages of the object-oriented paradigm. Encapsulation entails a separation

[3] A version hierarchy consists of an object and all the versions that have been derived from this object or from versions of it [6].

between an object's status and the interface. Such separation allows the *clients*[4] of an object to use the services provided by this object without knowing how the services are implemented (*information hiding*). Therefore, an object's implementation may change without impacting other objects or applications using the services provided by the object.

The information hiding capability has, in addition to the previously mentioned advantage, a great potential for data security. By surrounding an object with methods it is possible to interpose an additional layer between the object and its users. Therefore, arbitrary complex content-based access rules can be supported. A common distinction found in authorization models is between content-independent access rules, whose enforcement depends only on the object name, and content-dependent access rules, whose enforcement depends also on the object informative content. The last type of access rule requires accessing the object content when enforcing authorization. In RDBMSs, content-based authorization is provided by views. However, since views are defined as queries, the content-dependent access rules that can be defined are only those that can be expressed through the query language. The usage of methods allows access rules to be defined that cannot be expressed by queries.

RDBMSs allow a user to develop an application program and then grant the run authorization on this program to other users. The users receiving authorizations on a program do not usually need to have the authorizations on the data accessed by the program, since these authorizations are checked against the *program owner*. In this way, it is possible to provide authorization on an application basis. Methods in an object-oriented database could be used in the same way; thus providing an extensible authorization mechanism. However, the use of methods for authorization differs with respect to the use of application programs in the following aspect. When application programs are used, application-dependent access rules tend to be dispersed among the various application programs. Therefore, it is more difficult to verify that the correct authorization policy is followed. By contrast, methods are tightly coupled with data objects. This allows application-dependent access rules (of arbitrary complexity) to be centralized and all redundancies eliminated. Therefore, since an object-oriented approach enforces the principle that changes to method implementations and object structures should not impact the clients of an object, it is possible to modify access rules without requiring changes to clients. Of course clients must be able to deal with exceptions arising from lack of authorization. Note that the possibility of dynamically modifying access rules is a direct consequence of the fact that in an object-oriented database some of the high-level operations on data are moved into the database. Moving these operations into the database, by implementing them as methods, implies that access rules implemented as part of these operations are also moved into the database. Thus access rules are centralized and applied to all accesses made to the objects.

A number of authorization models have been defined based on the use of methods. In particular, the authorization model proposed by Ahad et al. [3] for the Iris OODBMS is based on controlling function evaluation[5]. Authorizations can be specified for users, or groups of users, to execute function, i.e., methods ovr objects. *Specific functions*, *guard functions* and *proxy functions* concepts are used to enforce content-dependent authorizations and to restrict the execution of given functions to some

[4] By clients of an object we mean other objects, and application programs.

[5] Roughly speaking, functions are the equivalent of methods in the Iris model.

users. Another model based on authorizations to execute methods on objects is presented by Richardson et al., in [24]. In this model, the owner of an object can control who may invoke which methods on the object. The model also enforces the concepts of *method implementor*—the user who has written the method's code—and *method principal*—the user on whose behalf the method is executed.

4.2. Mandatory access control

A basic model for mandatory access control (MAC) is represented by the Bell-LaPadula paradigm (see Section 3). Applying the Bell-LaPadula paradigm to the object-oriented data model is not straightforward, due to semantic richness of this model. Moreover, the differences both in theory and implementation among the various OODBMSs makes it very difficult to define sound and general principles upon which a suitable MAC model can be based. However, despite this difficulty, the use of an object-oriented approach offers several advantages from the security perspective. The notion of encapsulation, which was originally introduced in object-oriented systems to facilitate modular design, can be used to express security requirements in a way that is comprehensible to the users. Moreover, the fact that messages are the only means by which objects exchange information makes information flow in object-oriented systems have a natural and direct representation in terms of message exchange among objects.

MAC models proposed so far can be classified in two main categories: *single-level models* and *multilevel models*. Models in the first category require that an object and all its features (attributes and methods) be classified at the same security level. Models in the second category do not impose such restriction; however, they are rather difficult to implement in practice. Most models so far proposed are single-level. The main reason is the simplicity of such an approach and its compatibility with a security kernel. By using an underlying security kernel for the enforcement of mandatory security properties, the layer providing the object-oriented data model need not be trusted. As an alternative, a kernel directly supporting the object-oriented model can be developed. However, whereas developing such a kernel for a single-level model could be feasible in practice, it would be much more complex for a multilevel model. Single-level models have been proposed by Meadows and Landwehr [22], by Jajodia and Kogan [18] and by Millen and Lunt [23].

The main drawback of single-level models, despite their implementation simplicity, is that often applications need objects that are multilevel. In order to accommodate such applications the most common approach is to use a single-level object system and map the multilevel applications objects onto several single-level objects. We refer to this of approach as *multilevel object view approach*. The multilevel object view approach has two main variants, depending on whether inheritance [18,28] or aggregation [10] is used to support the multilevel view.

Real multilevel object models are more difficult to handle and no satisfactory approach has been proposed so far. An important issue when defining a multilevel security object model is what has to be protected. That is, which is the minimum granularity of protection. Most multilevel models assume that the elements of protections are: each object, each attribute of each object, each method of each object, each class, each attribute and each method defined in a class, each class-attribute and class-method.[6] Therefore, such models usually include a number of rules, stating constraints that must

[6] Class-attributes and class-methods are features of classes, intended as objects, and they are not inherited by the instances.

hold among the access classes of the various elements of the data model. Due to the richness of the object-oriented data model there is usually a large number of such rules, some of which are not intuitive to understand. We refer the reader to [29] for an example of such a model.

5. Data protection for digital libraries

Digital Libraries (DLs) introduce several challenging requirements with respect to the formulation, specification and enforcement of adequate data protection policies. Even though some of those requirements have been addressed by security research related to DBMS, approaching them in the framework of DLs entails solving several new challenging problems. Other requirements, such as copyright and intellectual property protection are specific to digital libraries and have not been addressed by security research in DBMS. We briefly discuss some of the main issues in what follows:

- *Flexible user specification mechanism:* In current authorization models, authorization subjects are stated in terms of user identifiers. In some systems, roles and groups can also be used as authorization subjects. In DLs, the user population is often very dynamic with new users (also from remote sites) needing to get access to library objects. Moreover, the access policies that one expects to be stated in a DL may be based on specific user characteristics. Consider as an example the policy stating that 'a video rated X can only be accessed by users who are 18 or older'. Directly mapping such a policy, which requires to know the age of the user submitting the access request, onto a traditional authorization model is not feasible because such a model typically does not support the specification of authorizations with user qualification based on user characteristics. The need of such flexible user specification is even more evident when dealing with users belonging to organizations different from the organization owning the DL. In such a case, the user characteristics may also include information such as the organization the user belongs to, whether the user (or its organization s/he belongs) has paid subscription (in case, access is allowed upon payment) and so forth.
- *Content-based access control to multimedia, unformatted data:* In many cases, decision about whether to access or not a given data object is based on the content of the object. A typical example from relational database systems is that 'a manager can only see the records of the employees working for him'. In RDBMSs, such access control policies are expressed in terms of views. In DLs, content-based access control is more difficult because of the presence of data such as text, image and video, which makes it more difficult to automatically determine whether a certain object verifies a condition on its content. As an example consider a policy stating that 'all documents discussing how to operate guns must be available only to users who are 18 or older'. The main issue is how to determine whether a certain text really deals with such a topic. The situation is even more complicated when dealing with images and videos, due to the inherent difficulty in content understanding for such data types. Consider the example of the policy stating that 'all videos containing violent scenes cannot be accessed by users who are younger than 18'. In such a case, the main issue is how to determine whether a certain video verifies the condition stated in the policy. Moreover, the fact that data are not regularly structured, as is the case for data in relational tables, makes more difficult identifying portions of the data to which different access restrictions must apply.
- *Distributed DLs and remote accesses:* Distribution entails two different aspects. The first aspect

concerns the fact that the DL itself may be distributed and consist of several information providers, each maintained by possibly different organizations. Because different organizations may have different access control policies, several questions arise related to whether each organization should retain its own autonomy with respect to access control policies and how to solve conflicts that may arise, or whether the various policies should be integrated and global policies be devised. Other questions are related to where to maintain authorization information and where to enforce access control. Some of those questions were addressed in the framework of distributed DBMSs. However, solutions proposed in such framework may not be adequate to DLs. The second aspect is related to the fact that users accessing a DL may be remote users (that is, they are not users belonging to the organization owning the DL). Remote accesses pose a number of problems, especially when remote users are required to provide information (such as the age) for access control purposes. Access control information that are recorded for users at the users' organization may differ with respect to the information required by the access control system of the DL. Consider the case of a DL requiring to know the age of the user for giving access to certain objects whereas the user's organization keeps the birth date of the user. In such case the mapping between the two types of information is easy; more complicated situations may arise. Another related question is how does a user (or an organization) wishing to access a DL determine the information to supply for access control purposes. Finally, other important issues include the use of certification and other authentication mechanisms to ensure the authenticity of the information provided for access control purposes as well as access anonymity. As pointed out by Winslett [30], there are many situations in which users accessing a DL may wish to keep their identity anonymous. For example, the employer of a user looking for information about a serious disease may suspect that the employee has the disease.

- *Copying and usage of information:* In some cases, accesses to objects from DLs can only be allowed upon payment and therefore objects cannot be freely passed among users. In other cases, objects can be used only if copyright information is mentioned.

A model for discretionary access control for DL has been recently proposed [2] that addresses some of the above issues. More specifically, the model provides capabilities for flexible user specification, based on user characteristics, and content-based access to textual documents. However, this model does not address many of the other requirements and much work is still needed to develop suitable access control models for DL. We refer also the reader to [1] for a discussion on several aspects of digital libraries protection and on its relationships with electronic commerce issues.

6. Conclusions

Data security and in particular data protection from unauthorized accesses remain important goals of any data management system. In this paper, we have outlined research results and practical developments. Much more, however, still needs to be done. As new data management applications emerge, new data protection mechanisms are required. In this paper, we could not discuss many emerging data management applications. As conclusions, however, we would like to briefly discuss two important research directions. The first direction is related to protection of data stored and accessed on World Wide Web (WWW). WWW is today enabling the deployment of a new generation of distributed information systems, whose communication platform is the Internet (or the Intranet).

Such an information system is characterized by a 'client-server' architecture, where the clients are potentially the entire population of the Internet users and the servers are potentially the available set of WWW servers. Clients are free to access data (like documents, and multimedia data) and software independently of their physical location. Such an environment is characterized by a very large and dynamically changing user population which is highly distributed. Data protection in such an environment entails addressing several issues, such as data access based on user credentials, secure data browsing software, access anonymity, distribute management of authorization and authentication information. Many of those issues have not been yet fully addressed. A second direction is related to data warehousing systems. A data warehouse is a database collecting data from several other databases, often called operational databases. A data warehouse is in general used for decision support applications. Data extracted from the operational databases are cleaned, integrated and transformed. Very often statistical aggregates are computed, since in many data warehousing applications only statistical data are required and not detailed data. A data warehouse may thus concentrate a large amount of data that when properly correlate may disclose information to users not authorized to see them. To date, no work on data security in data warehousing systems has been reported. However, several important issues need to addressed, ranging from suitable access control policies to adequate access control models, defined according to how the data are organized.[7]

References

[1] N. Adam et al., Strategic directions in electronic commerce and digital libraries: Towards a digital Agora, *ACM Computing Surveys* 28 (December 1996).

[2] N. Adam, V. Atluri, E. Bertino, E. Ferrari, A content-based authorization model for digital libraries, submitted.

[3] R. Ahad et al., Supporting access control in an object-oriented database language, in *Proc. Intl. Conf. on Extending Database Technology (EDBT)*, Vienna, Austria (Springer-Verlag LNCS 580, 1992).

[4] M.M. Astrahan et al., System R: A relational approach to data base management, *ACM TODS* (June 1976).

[5] D.E. Bell, L.J. LaPadula, *Secure Computer Systems: Unified Exposition and Multics Interpretation* (The Mitre Corp., March 1976).

[6] E. Bertino, L.D. Martino, *Object-Oriented Database Systems: Concepts and Architectures* (Addison-Wesley, Reading, MA).

[7] E. Bertino, S. Jajodia, P. Samarati, Database security: Research and practice, *Information System* 20(7) (1996) 537–556.

[8] E. Bertino, C. Bettini, E. Ferrari, P. Samarati, A temporal access control mechanism for database systems, *IEEE Trans. on Knowledge and Data Engineering* 8(1) (February 1996) 67–80.

[9] E. Bertino, C. Bettini, E. Ferrari, P. Samarati, Supporting periodic authorizations and temporal reasoning in database access control, in *Proc. 22nd Intl. Conf. on Very Large Data Bases (VLDB'96)*, Bombay (India), September 3–6, 1996 (Morgan Kaufman Publishers).

[10] E. Bertino, E. Ferrari, P. Samarati, Mandatory security and object-oriented systems: A multilevel entity model and its mapping onto a single-level object model, *Theory and Practice of Object Systems (TAPOS)*, in press.

[11] E. Bertino, S. Jajodia, P. Samarati, A non-timestamped authorization model for relational databases, in *Proc. 3rd ACM Conf. on Computer and Communications Security*, New Delhi (India), March 14–16, 1996 (ACM Press).

[12] E. Bertino, P. Samarati, S. Jajodia, An extended authorization model, *IEEE Trans. on Knowledge and Data Engineering* 9(1) (January/February 1997) 85–101.

[13] E. Bertino, B. Catania, A. Vinai, Transaction modeling and architectures, *Encyclopedia of Computer Science and Technology* (Marcel Dekker, New York, to appear.

[7] The so-called data cube is an example of organization model for data warehouses.

[14] E. Bertino, E. Ferrari, Administration policies in a multipolicy authorization system, in *Proc. of 10th Annual IFIP Working Conf. on Database Security*, Lake Tahoe, CA, August 1997.

[15] E. Bertino, E. Ferrari, V. Atluri, A flexible model for the specification and enforcement of role-based authorizations in workflow management systems, in *Proc. of 2nd ACM Workshop on Role-Based Access Control*, Fairfax, VA, November 1997.

[16] E.B. Fernandez, E. Gudes, H. Song, A model for evaluation and administration of security in object-oriented databases. *IEEE Trans. on Knowledge and Data Engineering* 6(2) (April 1994) 275–292.

[17] P.G. Griffiths, B. Wade, An authorization mechanism for a relational database system, *ACM TODS* 1(3) (September 1976).

[18] S. Jajodia, B. Kogan, Integrating an object-oriented data model with multilevel security, in *Proc. IEEE Symp. on Security and Privacy*, Oakland, CA, May 1990, pp. 76–85.

[19] C. Kaufman, R. Perlman, M. Speciner, *Network Security: Private Communication in a Public World* (Prentice-Hall, 1995).

[20] T.F. Lunt et al., The Sea View security model, *IEEE-TSE* 16(6) (June 1990).

[21] W.T. Maimone, I.B. Greenberg, Single-level multiversion schedulers for multilevel secure database systems, in *Proc. Annual Computer Security Applications Conf.*, Tucson, AZ, 1990, pp. 137–147.

[22] C. Meadows, C. Landwehr, Designing a trusted application in an object-oriented data model, in *Research Directions in Database Security* (Springer-Verlag, Berlin, 1992) 191–198.

[23] J. Millen, T. Lunt, Security for object-oriented database systems, in *Proc. IEEE Symp. on Security and Privacy*, Oakland, CA, May 1992, pp. 260–272.

[24] J. Richardson, P. Schwarz, L.F. Cabrera, CACL: Efficient fine-grained protection for objects, in *Proc. Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, Vancouver, British Columbia, Canada, 1992.

[25] F. Rabitti, E. Bertino, W. Kim, D. Woelk, A model of authorization for next-generation database systems, *ACM TODS*, 16(1) (March 1991).

[26] P. Samarati, E. Bertino, S. Jajodia, An authorization model for a distributed hypertext system. *IEEE Trans. on Knowledge and Data Engineering* 8(4) (August 1996) 555–562.

[27] R. Sandhu, Role hierarchies and constraints for lattice-based access controls, in *Proc. of 4th European Symp. on Research in Computer Security*, Rome, Italy, 1996, Lectures Notes in Computer Science 1146, pp. 65–79.

[28] M.B. Thuraisingham, Mandatory security in object-oriented database systems, in *Proc. Intl. Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, New Orleans, LA, October 1989.

[29] M.S. Olivier, S.H. Von Solms, A taxonomy for secure object-oriented databases, *ACM TODS* 19(1) (March 1994) 3–46.

[30] M. Winslett, N. Ching, V. Jones, I. Slepchin, Using digital credentials in the World-Wide Web, *Journal of Computer Security*, 5(3) (1997) 255–267.

Elisa Bertino is professor of computer science in the Department of Computer Science of the University of Milan where she heads the Database System Group. She has also been on the faculty in the Department of Computer and Information Science of the University of Genova, Italy. Until 1990, she was a researcher for the Italian National Research Council in Pisa, Italy, where she headed the Object-Oriented Systems Group. She has been a visiting researcher at the IBM Research laboratory (now Almaden) in San Jose, at the Microelectronics and Computer Technology Corporation in Austin, Taxas, and at Rutgers University in Newark, New Jersey. Her main research interests include object-oriented databases, distributed databases, deductive databases, multimedia databases, interoperability of heterogeneous systems, integration of artificial intelligence and database techniques, and database security. In those areas, Prof. Bertino has published several papers in refereed journals, and in proceedings of international conferences and symposia. She is a co-author of the books '*Object-Oriented Database Systems—Concepts and Architectures*' (Addison-Wesley International, 1993), '*Indexing Techniques for Advanced Database systems*' (Kluwer Academic Publishers, 1997), and '*Intelligent Database Systems*' (Addison-Wesley International, in press). She is or has been on the editorial boards of the following scientific journals: the IEEE Transactions on Knowledge and Data Engineering, the International Journal of Theory and Practice of Object Systems, the Very Large Database Systems (VLDB) Journal, the Parallel and Distributed Database Journal, the Journal of Computer Security, Data & Knowledge Engineering, and the International Journal of Information Technology. Elisa Bertino is a member of ACM, IEEE and AICA. She has served as PC member of several international conferences. She has served as Program Chair of the 1996 European Symposium on Research in Computer Security (ESORICS'96), and as General Chair of the 1997 International Workshop on Multimedia Information Systems. She is currently serving as Program Co-Chair of the 1998 IEEE International Conference on Data Engineering (ICDE).