LESSONS LEARNED FROM MODELING A SECURE MULTILEVEL RELATIONAL DATABASE SYSTEM

Dorothy E. Denning

Digital Equipment Corporation
Systems Research Center
Palo Alto, California

Introduction

Before joining Digital, I led the Secure Data Views (SeaView) project at SRI International, the goal of which is to design a multilevel-secure relational database system that meets the criteria for Class A1 [1]. The project is a three-year joint effort by SRI and Gemini Computers, Inc., and is sponsored by the U.S. Air Force, Rome Air Development Center (RADC). At the time of this writing, the project has completed the first two years.

At the outset, we adopted a broad perspective of security that went beyond mandatory and discretionary access controls to requirements for data consistency (through database integrity constraints and transactions). This interpretation is in keeping with common usage of the word security, which Webster's Dictionary defines as "freedom from danger: safety," and recognizes the affinity between security and dependability. Our approach was motivated by the expressed needs of customers for security measures that emphasize data integrity in addition to secrecy, and by our belief that integrity involves data consistency in addition to controlling write access to data.

We have developed security policies for mandatory access control, discretionary access control, data consistency, the labeling of new and derived data, sanitization, and reclassification [2]; a basic strategy for a system design based on existing security kernel technology, such as that available in GEMSOS [3]; a multilevel relational data model that extends the standard relational model to support element-level classification [4]; and a formal security policy model [5]. The remaining tasks on the project include developing a formal top-level specification and implementation specifications.

The process of designing and modeling a multilevel-secure database system has been an interesting and challenging experience, filled with surprises and unsuccessful approaches that have not worked. In this note, I would like to share some of the lessons we have learned from the process, which are in summary:

1. Polyinstantiation is an intrinsic problem of multilevel systems.

2. Multilevel relations are inherently views, and may be better defined as such rather than as real (i.e., base) relations.

3. The access classes associated with data in a multilevel database cannot be assigned arbitrarily; they must reflect semantic relationships among the data.

4. A model for a multilevel-secure database system should build on a simple security kernel that enforces mandatory security. Mandatory access controls should not be part of the database model, although the latter must support labeled data.

5. Security policies for discretionary access control, data consistency, and the labeling of data must be included in the database model.

Polyinstantiation is an Intrinsic Problem of Multilevel Systems

In a multilevel system, a user may inadvertently create an object whose name conflicts with an invisible object; i.e., an object at an access class that is not dominated by the user's class. Because mandatory security requires that the system behave as though the invisible object did not exist, neither can the user be notified of the conflict nor the invisible object be overwritten. Thus, any system that enforces mandatory security must support the possibility that multiple data objects, differentiated only by their access class (i.e., bearing names otherwise identical), may exist simultaneously. In the SeaView project, we called the existence of like-named objects *polyinstantiation* [4].

In a multilevel database system, polyinstantiation arises whenever two or more relations with the same name are defined with different classes (polyinstantiated relations), multiple tuples with the same primary key values but different access classes are inserted into a relation (polyinstantiated tuples), or multiple values exist for a single attribute within a tuple, where each value is associated with a different class (polyinstantiated elements) [4]. One way in which polyinstantiation within a relation can arise is through updates by subjects for whom some tuples or elements are hidden (hidden elements appear as null values).

Polyinstantiation affects the system model and design in that a mechanism is needed for distinguishing and accessing the multiple versions. It also affects the user interface in that users and application software, including queries and application-dependent integrity constraints, must be prepared for the possibility of multiple versions, with a need to specify which one(s) are desired – e.g., object with highest class, most recent object, all objects. The Bell and LaPadula model [6] supports polyinstantiated objects, namely files and segments, through its hierarchical directory structure, which ensures that all objects have a different path name in the hierarchy.

Although polyinstantiation is intrinsic to any kind of multilevel system, it is potentially of greater concern in a database system because it affects the database model. For example, to support polyinstantiated elements, it is necessary to either remove the requirement that elements be atomic (thus allowing relations that are not in first normal form) or represent multiple elements as multiple tuples, thereby eliminating the constraint that tuples be distinguished by their primary key values alone (thus allowing relations that are not in second normal form with respect to the primary key attributes). The SeaView model follows the latter approach, but both strategies complicate the system design and user interface.

Throughout the project, we have sought restrictions that would reduce the effects of polyinstantiation. One possibility is to require that each real relation be at a single

access class, because this eliminates the possibility of polyinstantiated tuples or elements within a real relation. While I recommend this approach, I also acknowledge that making real relations single-level does not eliminate all of the problems associated with polyinstantiation. If multilevel views can be formed, then polyinstantiation may appear in the derived relations because the process of merging or joining the base relations will generate multiple tuples for what are effectively polyinstantiated tuples or elements. Indeed, the SeaView representation of polyinstantiated tuples and elements in real relations is as though the real relations were defined as views over single-level base relations. Thus, polyinstantiation can be moved from real relations to views, but it cannot be avoided at the user interface completely.

Multilevel Relations are Inherently Views

If a relation contains data of different access classes, some of those data will be hidden from subjects whose access class is not high enough. This means that, in effect, different subjects have different *views* of the same relation – that is, they see different instances thereof.

Although multilevel relations are inherently views, the SeaView model supports *multilevel real relations* with element-level classification as an abstraction available to users; that is, users can, if they wish, define multilevel relations as base relations, and all relations so defined will be treated like real relations in the standard relational model. Specifically, the application-independent integrity rules of the relational model, namely, *entity integrity* and *referential integrity*, apply to multilevel real relations, application-dependent integrity rules can be defined on multilevel real relations, and updates to multilevel real relations are well-defined. In addition, multilevel real relations are constrained by a third application-independent integrity rule called *polyinstantiation integrity*, which specifies consistency for polyinstantiated tuples and elements.

The different views of a multilevel real relation are modeled by a function that returns the instance of a relation visible at a given access class, and by a property that shows how the different instances must be related to each other. So that the database may appear consistent to each user, entity integrity and referential integrity are extended in the SeaView model to guarantee consistency at each access class.

The task of designing and modeling a database system that supports multilevel relations as real relations, rather than simply as views, was much harder than expected. We had not, for example, anticipated either the need to extend the application-independent integrity rules of the relational model or the problems that arise with polyinstantiated tuples and elements. Polyinstantiation requires not only an additional integrity rule, but also has potentially serious consequences for applications and users; as noted earlier, both must be prepared to deal with the possibility of encountering multiple versions of the same data during retrieval or update operations. Updating data associated with polyinstantiated elements may be particularly troublesome because the relation is not in second normal form when elements associated with a given primary key are represented by several tuples at the user interface. Thus, the user or application may be confused about which tuple or tuples to update in order to change a particular element. (In the SeaView design, only one has to be changed because multilevel real relations are decomposed into relations that are in normal form, in which the redundancy is handled automatically.)

Another problem associated with updating multilevel real relations is classifying the data entered into a relation. The SeaView model supports the concept of *classification constraints*, which are user-defined integrity rules that constrain the classes of data elements. Classification constraints provide a means of rule-base classification that allows an element's access class to be determined by a subject, by the access class of a subject, or by the type, context, or content of the data. However, the rules can be applied only by a subject who is trusted to operate over a range of access classes; if the subject is not trusted, then the data must be assigned the subject's class. If real relations are single-level, classification rules are not needed, although they might still be useful as a means of supporting updates of multilevel views by trusted subjects.

Another potential problem with multilevel real relations is that they can complicate the specification of meaningful application-specific integrity constraints on relations. This is because the evaluation and enforcement of an integrity constraint on a relation must take place at the access class of the subject attempting an update operation on the relation. Therefore, data that are used to evaluate a constraint for one subject may be hidden from another subject, and the effect of this hidden data must be accounted for in the constraint. In addition, the constraint must take into account the possibility of polyinstantiated data within a relation. Unless the effects of hidden and polyinstantiated data are taken into account in the rules, the database may not behave as anticipated [7]. Note that these effects must be accounted for with respect to integrity rules that constrain the classes of elements (i.e., classification constraints) as well as to integrity rules that constrain their values (called value constraints in the SeaView model). Making real relations single-level partially addresses the problem, though it does not solve it completely because real-world constraints that relate multilevel data must still be represented.

On the basis of these consequences, multilevel relations may be better treated *as views* over single-level relations rather than *as real relations*. Then each real relation is single level and, therefore, has a single instance at any given time, the application-independent integrity rules of the standard relational model require no extensions, and polyinstantiation integrity is unnecessary. In addition, application-specific constraints should be easier to formulate.

The lack of support for multilevel real relations need not limit a user's ability to integrate and process multilevel data as long as the system supports *derived multilevel relations* that are obtained through views. Moreover, if a view is updatable, the users can update it as though it were a real relation.

An argument in favor of supporting multilevel real relations is that it frees the user from the need to create many relations when the data are potentially associated with many different access classes (e.g., compartmented data). However, users can be given database design tools that automatically decompose a schema for a multilevel relation into a set of single-level relations.

The I.P. Sharp model [8] classifies each real relation at a single level. However, this is strictly an access control model, and lacks support for consistency through integrity rules and multilevel views.

Although I favor treating all multilevel relations as views rather than as real relations plus views, it is difficult to assess the two approaches in the absence of any multilevel database system that supports either. Because the SeaView model supports both multilevel real relations, which can be single-level in the degenerate case, and multilevel views,

implementation of a prototype system based on the model would provide a testbed on which both approaches could be evaluated to determine how multilevel real relations affect the system and the user interface.

Access Classes Must Reflect Semantic Relationships

If a database system supports multilevel real relations with element-level classification, the classes of the elements must be constrained to reflect semantic relationships. This is necessary for several reasons: to ensure that the integrity rules of the relational model will be satisfied at each access class – that is, by all relation instances; to ensure that the data path needed to access an element is also accessible; and to simplify the implementation. Some of the constraints we found necessary are the following:

- The classes of the elements comprising the primary key of a tuple must be the same – that is, each primary key must be of a single class (for entity integrity, so that a primary key will not appear to have null elements at a given access class).
- The class of the primary key of a tuple must be dominated by all other element classes within the tuple (for entity integrity and to ensure that the path to an element will be accessible).
- Elements that refer to other tuples must be assigned a class that dominates the key of the target tuple – i.e., all references must be downward (for referential integrity, so that there are no dangling references at a given access class).
- The class of a relation name must be dominated by the classes of all elements within the relation (to ensure that the path to the element will be accessible).
- All schema data for a relation (e.g., names and types of attributes) must be at the level of the relation name (for implementation reasons, so that (1) there will not be any polyinstantiated attributes, and (2) the entire record structure will be visible at the level of the relation name). This is not a limiting factor because higher-level attributes can be put into a separate relation, which is then joined to the lower-level relation.
- The class of a view name must dominate the classes of the relation names in the view definition (so the view can be evaluated).

In addition, if data are semantically related – e.g., as expressed by integrity constraints – then the classes assigned to the data (and to the constraints themselves) must reflect these relationships if inference problems are to be avoided [9, 10]. Matthew Morgenstern and I have examined ways of using constraints to determine what classes to assign to data in order to avoid such inference problems [11, 12].

A Database Security Model Should Build on a Simple Mandatory Kernel Model

Database systems are complex, consisting of hundreds of thousands of lines of code. To build a secure database system that meets the criteria for Class A1, those portions of

the system that support mandatory security must be confined to a small security kernel that can be formally specified, verified, and then analyzed for covert channels. Because less assurance is needed for discretionary security and data consistency, these policies can be handled by the trusted database system, which need not be developed quite as rigorously.

Early in the project, we considered the possibility of designing a special database kernel whose protected objects would be the individual elements in a relation (since classification is at the level of elements). However, this approach seemed overly complex, owing mainly to two problems: merging together the elements of a relation and polyinstantiation. This led us to consider a much simpler approach, namely building the database system on top of a general-purpose operating system kernel whose protected objects are single-level files and segments. We did not consider the possibility of a database kernel whose protected objects are multilevel real relations, because this would push the database problems associated with multilevel relations, including polyinstantiation, in the kernel, thereby creating a complex kernel.

We were led to the more general OS kernel approach by the inherent view nature of multilevel relations, as discussed earlier. We saw how we could define these views as relational expressions over a set of single-level base relations, each of which could be stored in a single-level object managed by the kernel. Thus, each multilevel relation would be decomposed into single-level relations and the instance of a relation at a given class would be restored by evaluating a relational formula over these underlying relations [4]. (Because SeaView provides the abstraction of multilevel real relations, the decomposition is transparent to the users; however, if all multilevel relations are treated as views, then the decomposition could be made available to users as a design tool as discussed earlier.) The database system would be responsible for labeling the elements in a derived view; however, it need not be relied upon to enforce mandatory security because the kernel would not make any data accessible that were not authorized to be retrieved by the requesting subject. Thus, the labels managed by the database system are for the user's convenience rather than for enforcement of mandatory security.

The SeaView security model reflects this design strategy and is formulated in two layers: an inner layer called the *MAC model*, which enforces a mandatory access control policy, and and an outer layer called the *TCB model*, which represents the trusted computing base. The TCB model defines the database system security perimeter and formalizes the relationship between the database system and the security kernel.

The MAC model represents a general-purpose security kernel that meets the criteria for Class A1 with respect to mandatory security. It includes the concepts of subjects, objects, and current access set from the Bell and LaPadula model[6], but contains no components that are specifically related to database systems.

The TCB model defines the discretionary access control policy and policies relating to data consistency and the labeling of new and derived data. It specifies the components of a multilevel-secure relational database system, including multilevel relations as defined by the multilevel relational data model, views, application-dependent and independent integrity constraints (both value constraints and classification constraints), transactions, and discretionary authorizations.

The TCB model is mapped onto the MAC model. Specifically, all the information of the TCB model is to be stored in objects managed by the MAC model, and all operations of the TCB model are decomposed into operations of the MAC model. Thus, the TCB

interface is constrained by the kernel. The layering allows one to obtain A1 assurance for the system as a whole without having to apply the A1 criteria for mandatory security to the TCB layer [13].

Security Must Be Part of the Database Model

For many years, the security community has been arguing that security must be built into a system at the outset in order to achieve the desired result. At the same time, the community has been developing security models that are strictly access control models, disregarding, for example, data consistency requirements. (See also Clark and Wilson [14].)

Our experience on the SeaView project has shown that access controls and consistency must be considered within the framework of a complete database model. As noted earlier, database integrity constraints are affected by mandatory security, and these effects must be incorporated into the model. The modeling of application-dependent constraints also had the side benefit of giving us a means of supporting classification constraints for labeling new data, and of supporting a major part of the SeaView policy for sanitization and reclassification since sanitization functions and time-dependent classification rules can be expressed as classification constraints. We also found that many of the features one typically finds in a database model are security-relevant; for example, relational expressions are security-relevant in that they are used to define views, which are among the objects protected by discretionary access controls, to express application-dependent constraints, and to specify the requirements for labeling derived data.

The SeaView security model is close to being a complete database model for a multilevel relational database system, but falls short in that it lacks the five basic operators of the relational model. While the SeaView model includes abstract components for deriving multilevel relations by means of relational expressions and specifies the properties that must be satisfied by these components, it does not specify the latter in terms of any specific relational operators. Nor does it specify how multilevel relations are decomposed into single-level standard relations. Thus, the security model is somewhat more abstract than the multilevel relational data model developed earlier in the project [4]. The security model could be further abstracted to apply to systems other than relational ones, although doing so might make it less useful as a relational database model.

The SeaView model is considerably more complex than the standard relational model. Some of this complexity stems from the inclusion of application-dependent integrity rules and components for discretionary security. Indeed, about a third of the properties relate to discretionary security. However, much of the complexity results from supporting multilevel relations as real relations. As noted earlier, making all real relations single-level would simplify the model.

Conclusions

Multilevel relations are more than just standard relations with labels. They result in hidden and polyinstantiated data, which in turn affect integrity constraints, query processing, and database updates. Consequently, mandatory security requirements for database systems cannot and should not be considered in isolation of other requirements, especially those for data consistency. All requirements must be considered together, within

the framework of a single database model. In addition, prototype systems should be built so that users can experiment with multilevel relations and thereby determine whether the latter should be supported as real relations or as views (my recommendation); and so as to ascertain what types of tools are needed to help users design and manipulate multilevel databases.

Acknowledgments

I wish to thank my colleagues on the SeaView project, especially Teresa Lunt, Matthew Morgenstern, Peter Neumann, and Roger Schell, for their constructive comments. I am also grateful to Peter Denning for his helpful comments.

References

[1] *Department of Defense Trusted Computer System Evaluation Criteria.* Dept. of Defense, National Computer Security Center, Dec. 1985. DOD 5200.28-STD.

[2] D. E. Denning, T. F. Lunt, P. G. Neumann, R. R. Schell, M. Heckman, and W. R. Shockley. Security policy and interpretation for a class A1 multilevel secure relational database system. Nov. 1986. Computer Science Laboratory, SRI International.

[3] R. R. Schell, T. F. Tao, and M. Heckman. Designing the GEMSOS security kernel for security and performance. In *Proc. 8th National Computer Security Conf.*, pages 108–119, 1985.

[4] D. E. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. R. Shockley. A multilevel relational data model. In *Proc. 1987 Symp. on Security and Privacy*, pages 220–234, IEEE Computer Society, 1987.

[5] D. E. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. R. Shockley. Secure distributed data views (SeaView): The SeaView formal security policy model. July 1987. Computer Science Laboratory, SRI International.

[6] D. E. Bell and L. J. LaPadula. *Secure Computer Systems: Unified Exposition and Multics Interpretation.* Technical Report ESD-TR-75-306, The MITRE Corp., Bedford, Mass., Mar. 1976.

[7] D. E. Denning. Secure databases and safety: some unexpected conflicts. In *Proc. of the Safety and Security Symp.*, Centre for Software Reliability, The City University, London, England, Oct. 1986.

[8] M. J. Grohn. *A Model of a Protected Data Management System.* Technical Report ESD-TR-76-289, I. P. Sharp Assoc. Ltd., June 1976.

[9] D. E. Denning, S. G. Akl, M. Heckman, T. F. Lunt, M. Morgenstern, P. G. Neumann, and R. R. Schell. Views for multilevel database security. *IEEE Trans. on Software Eng.*, SE-13(2):129–140, Feb. 1987.

[10] D. E. Denning. The inference problem in multilevel database systems. In *Proc. of the National Computer Security Center Invitational Workshop on Database Management Security*, The National Computer Security Center, June 1986.

[11] D. E. Denning and M. Morgenstern. *Military Database Technology Study: AI Techniques for Security and Reliability.* Technical Report, Computer Science Laboratory, SRI International, August 1986.

[12] M. Morgenstern. Security and inference in multilevel database and knowledge-base systems. In *Proc. ACM Int'l. Conf. on Management of Data (SIGMOD-87)*, pages 357–371, May 1987.

[13] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. Element level classification with A1 assurance. 1987. To appear in *Computers and Security.*

[14] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proc. 1987 Symp. on Security and Privacy*, pages 184–194, IEEE Computer Society, Apr. 1987.