Framework for the Security Component of an Ada* DBMS (Extended Abstract)

David Spooner
Rensselaer Polytechnic Inst.
Troy, NY

Arthur M. Keller
University of Texas
Austin, TX

Gio Wiederhold
Stanford University
Stanford, CA

John Salasin
Deborah Heystek
Institute for Defense Analyses
Alexandria, VA

ABSTRACT. This paper discusses a framework for the design of a security component for a secure Ada database management system (DBMS). It is part of a development effort to produce prototype technologies for the World Wide Military Command and Control System (WWMCCS) Information System (WIS). In this paper we present a series of criteria for evaluating database security approaches. We develop the high-level framework for the security component of a DBMS and illustrate how it can support several alternative security models, which we compare using these criteria. The security enforced by the DBMS relies an appropriate security mechanisms enforced by the operating systems for operating system objects, such as files, used by the DBMS. We also present the security barrier or filter as an alternative or adjunct to the notion of a trusted computer base.

1 Introduction

Under sponsorship of the Department of Defense is a project to develop prototype foundation technologies for the World Wide Military Command and Control System (WWMCCS) Information System (WIS) using the programming language Ada. The purpose for developing these prototypes is to produce software components that demonstrate the functionality required by WIS; use the programming language Ada to provide high-levels of portability, reliability, and maintainability as well as efficient operation; and to be consistent with current and expected software standards [WIS84].

One of the foundation areas of interest to this project is database management, and an important requirement of the resulting DBMS is that it be secure. Security of the DBMS is influenced by such issues as operating system security (reuse of objects, garbage collection), hardware controls (privileged user modes), operational policy (password protection) and security policy (which users have access to which objects).

Of particular interest is the need for support of mandatory and discretionary access controls. Access control determines both access to information and flow of information. Mandatory controls partition objects into classification levels with a strict ordering from least secure to most secure. Users are assigned to clearance levels. Retrieval of an object by a user requires him to belong to a class at least as secure as the object he is retrieving. Within a classification level, non-hierarchical categories may be defined. For example, a classification level may contain the categories: Army, Navy, Air Force, Marines, NATO. Discretionary controls are used within a level to provide a finer granularity of control. Discretionary controls can be used to restrict a user's access to only part of an object (e.g., relation or attribute) based on the value of the information contained in the object. For example, a user might be restricted to access only those tuples of a relation with a specific value in a certain attribute of the relation. Thus, discretionary controls are used to implement access on a "need to know" basis.

This paper assumes that mandatory access control will be provided by the operating system (another

* Ada is a registered trademark of the US Government, Ada Joint Program Office.

The work reported in this document was conducted as part of Institute for Defense Analyses Project T-4-206 under Contract MDA-903-84-C-0031 for the Department of Defense. The publication of this paper does not indicate endorsement by the Department of Defense or IDA, nor should the contents be construed as reflecting the official positions of these organizations.

Proceedings of the Twelfth International Conference on Very Large Data Bases Kyoto, August, 1986

foundation technology in the project) [Che85]. This leaves only discretionary controls to be implemented in the DBMS (although an extension of the framework reported here could be used to implement mandatory controls as well on top of a secure operating system). We assume that the computing environment contains a trusted computer base (TCB). This TCB includes the parts of the operating system which implement mandatory controls and other security related functions, and also includes the parts of the DBMS directly responsible for discretionary controls.

The design of a security system depends on the choice of a security approach. We explain a series of criteria for evaluating security approaches. By explicitly considering these criteria and their ramifications, we can better understand the consequences of using a particular security approach.

We discuss the design of a security component for an Ada DBMS. We show how this design can be used to implement several alternative security approaches.

Before proceeding, it is necessary to identify several assumptions. We assume that the DBMS is implemented on top of a secure operating system providing mandatory access control. We assume that the DBMS is correct, for the most part. By this we mean that we expect that it will correctly respond to queries most of the time. We do not assume, however, that it has been validated (except for the parts implementing discretionary controls), that is, it has not been verified that it operates correctly nor that the design or implementation does not allow for security breaches. We do not attempt to provide inference controls for the DBMS because we see no adequate solution available to handle this type of problem [Den82]. This is an area that requires further research.

The DBMS is too large to be verified and hence cannot be part of the TCB. Instead we propose the use of a security barrier and filter to limit access without requiring that all components within the barrier have to be trusted.

2 Evaluating Security Approaches

In this section, we present several evaluative criteria for security approaches. These criteria are open versus closed system architecture, granularity of protected objects, monotonicity, specification of security controls, actions when there are multiple predicates that apply to a request, response to security violations, and the location of security checking.

A fundamental consideration of a security approach is whether it is open or closed. An open system allows access unless it is explicitly prohibited, while a closed system denies access unless it is explicited permitted. For a military system where security is a primary concern, a closed system model seems more appropriate.

Granularity of protected objects describes the unit of object being protected. Mandatory controls provide security on data objects at the file level. Discretionary controls must provide security at smaller levels of granularity. For a relational database, this implies that discretionary controls should be defined for relations, tuples, and attributes of relations.

Monotonicity of the discretionary controls concerns combinations of objects. If a user has access to A and B individually, should he have access to A and B together? While it is not always desirable to allow this, it is difficult to prevent since A and B can be combined outside the system. Thus, we have made no attempt in the design of the Ada DBMS to prevent aggregating of data when access is allowed to the individual components of the aggregation. However, a stored aggregation of data may be given its own security classification independent of the classification of the data aggregated.

Specification of security controls means how we describe what is and what is not permitted. Discretionary controls are specified with a predicate describing authorizations. These predicates can take several forms. They may represent authorized queries so that every query issued by a user is a combination of one or more of the predicates. They may take the form of restrictions on particular database objects such as relations and attributes. In this case, each predicate specifies which tuples and which attributes of a relation a user is allowed to access. The predicates may take the form of a view over the database authorizing access to some arbitrary subset of the database. The choice of which form to use depends on the method of implementation for the discretionary controls. For the Ada DBMS, we have chosen to use predicates that specify operations that may be applied to subsets of relation tuples and attributes. Note that the number of predicates required to specify discretionary controls varies for the different forms of predicates.

When there are multiple predicates that apply to a query, the question of whether the operation is permitted needs to be decided. Firstly, we consider that multiple predicates may be needed when no single predicate applies to all the data affected by an operation. We may therefore consider separately each component of data and the relevant predicates. If all the individual operations on the components of data are permitted, then under a monotonic security approach, the entire operation is permitted. If there are multiple predicates that apply to a particular operation on a component of data (relation, tuple, attribute triple), it is reasonable (but not necessary) to allow the operation if permitted by at least one predicate for a closed system or if permitted by all relevant predicates for an open system.

Response to a security violation is an important part of a security approach. When a security violation occurs, the offending query may be rejected with a security violation error message, or the remainder of the operation may be performed omitting actions on data that would violate security. Neither approach works in all cases. Both approaches can allow the user to make inferences about the nature of protected data. In the first case, by adjusting the scope of the operation, a user can determine the existence of protected data. In the latter case, the user is not explicitly made aware of the security violation. However, by judicious use of counting, a user may be able to infer some details of protected data. We choose the latter alternative as it better fits our security framework, which uses a filtering technique. A naive user may inadvertantly include secure data in the scope of a query, and the user probably wants such data excluded were it possible to specify that explicitly. (In that case, a security error message might make the user newly aware of the existence of secure data.) Additional options include notification of a security officer upon security violation, logging of the offending user, and invalidating the user's account to preclude future violations.

Location of security checking within the system is an important architecture question that affects what types of security controls can be implemented. Security checking can be done on queries as entered, during the parsing and decomposition of a query, by filtering the data retrieved by a query, or by any combination of these methods. Because of the importance of security in military systems, we feel that security checking should be done both during the process of parsing and decomposing a query and by filtering the data returned by the query. The first security check produces the added advantage of reducing the amount of data retrieved by the query to a small superset of the data which will make it through the filter. This improves the overall performance of the system. Data filtering has the advantage of being simple enough to be made part of the TCB.

3 Mandatory Controls

Mandatory access control is also known as level B multilevel access control as defined in the guidelines of NSA's Computer Security Center [DOD83]. Mandatory access controls provide a method for restricting access to objects based on the sensitivity of the information contained in the object. They also provide a means for the formal authorization of users to access information. The degree of access control depends on the granularity of objects and users. For example, objects may be defined to be such things as records, blocks, pages, files, words or fields. The term, users, may include processes and transactions in addition to people.

Each entity in the system (e.g., objects and users) has a classification level assigned to it. Users can access objects at a level no higher than their own. In addition, they can write objects at a level no lower than their own, thus preventing the flow of information from a higher security level to a lower level. This introduces the conflict between preventing unauthorized disclosure of secure information versus inhibiting official orders sent from a secure source to a less secure recipient.

The mandatory access controls will be provided by the operating system being developed concurrently with the DBMS [Che85]. Since all processes, including the DBMS processes, run under control of the operating system, mandatory controls will be enforced automatically with no circumvention possible. In particular, a user query to the DBMS will be executed by the DBMS on behalf of the user at the security level of the user process. This is guaranteed by the operating system which would prevent the required interprocess communications between the DBMS and the user process if they were not operating at the same security level. An authentication server in the TCB is used to verify the identity of each user including clearance level. Thus, it is unnecessary to consider mandatory controls within the DBMS.

A result of the fact that interaction between processes at different security levels is prohibited, is that it will be necessary to have multiple instances of the DBMS, one for each security level. This is required by the requirement of a strict firewall between mandatory security levels that cannot make assumptions based on the applications including the database running on the operating system. A user process interacts with the instance of the DBMS at its security level. Logs and other data collected by the DBMS as it operates must be maintained separately for each instance of the DBMS at the different security levels.

We also make the restriction that relations contain data from only one security level. While this is not an absolute requirement, it significantly simplifies the DBMS. Relations which require violating this restriction must be broken into two or more separate relations, one for each level. This partitioning of a relation is similar to the notion of relation fragments used in distributed systems [Rot80]. Algorithms are available to reconstruct a relation from its fragments [Day78].

4 Discretionary Controls

Discretionary controls are needed within a mandatory security level to specify that a user may access only a subset of the objects at that security level. For purposes of the DBMS, this amounts to enforcing content-dependent access control on the DBMS relations. This is done by defining predicates which describe the subsets of objects accessible to the user. These predicates are evaluated when a user enters a query to verify that the query responds only with data authorised for access by that user ([Sto75], [Gri76], [Spo84]). Access to relations that do not require content-dependent access controls should suffer minimal performance penalty from the discretionary control mechanism.

Discretionary access controls can be defined over entire relations as well as over specific attributes of a relation. The predicates can be expressed in SQL, if the DBMS supports an SQL interface, or as a relational algebra program or Ada program with embedded calls to the DBMS. The choice depends on which interfaces are supported by the DBMS and the level at which the security component interfaces to the rest of the DBMS.

Discretionary controls can be defined for individual users as well as for cliques (groups) of users. If cliques are used, then it is possible for a user to be a member of more than one clique simultaneously. This makes enforcement of discretionary controls more difficult since a user may have conflicting privileges in the cliques in which he is a member. Therefore, we require that a user identify a particular clique when he logs into the DBMS, and he then has the access privileges of that clique only. A module in the TCB is responsible for verifying that the user is a valid member of a clique.

Figure 1 gives a logical architecture for the DBMS [Fri86, Wie86]. The security component is one block in this architecture. The conceptual architecture for the security component itself is presented in Figure 2. Logically, it consists of three parts: an *authorization table* [Gra72] which records the predicates which define the content-dependent access controls for those relations with discretionary controls, an *interpreter for the predicates* in the authorization table, and the *mechanism to enforce* the discretionary controls by consulting the table and invoking predicate evaluations when needed. The details of these three components vary depending on the approach used to implement the security component. This is discussed further below.

The security component can interact with the rest of the DBMS at several levels. For example, it might be part of the interface between the Query Execution Planner and the Executor, or between this Executor and the IFAP. It might also be included as part of the Executor or even as part of the Query Execution Planner. Again, this depends on the detailed design.

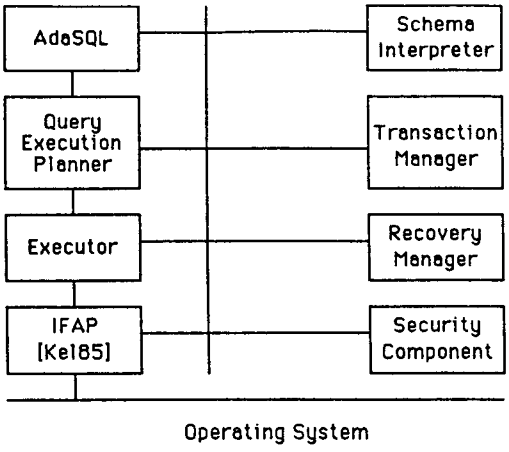

Figure 1: Logical DBMS Architecture

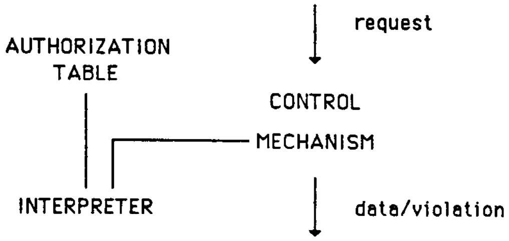

Figure 2: Logical Structure of the Security Component

5 Security Component Framework

In this section we describe a security component framework based on our modularization design of a Ada database system [Wie86]. Our security component framework permits the enforcement of a variety of security approaches. For example, both open and closed approaches may be implemented using our framework. We want the security component's interface with the rest of the DBMS to be below all user interfaces to preclude circumvention of security checking. This implies that the security component can be no higher than the top interface to the Executor [Wie86]. However, it cannot be much below the Executor for logging purposes.

We have chosen to support granularity at the tuple and attribute level. This implies that the security

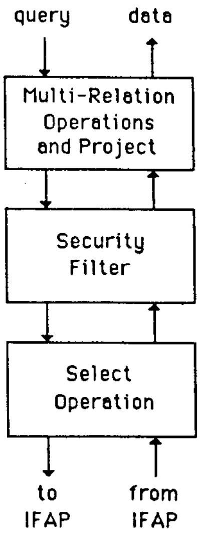

Figure 3: Architecture for the Executor

component processing must precede multi-relation operations such as join and cross product. It should follow single relation selections to reduce the volume of data considered by the security component. The security component requires access to tuples before undergoing projection so that attributes needed for security decisions are available. This presents a problem when some but not all attributes may be returned to the user. In this case the protected attributes are replaced with standard format nulls, which are presumably removed by a subsequent projection. The protected attributes are not removed so that the security component can operate transparently; other components need not be aware that the security component has taken any action. This approach, shown in Figure 3, most easily implements monotonic security approaches.

The security predicates may be arbitrarily complex, including Ada packages if need be. Alternatively, were a sufficiently powerful non-procedural security language to be defined, the security component could interpret this language. As our security framework operates on a tuple-by-tuple basis, multiple predicates are only relevant when they apply to the same tuple and attributes; the actions taken when a multiple relevant predicates do not agree may be programmed as the predicates are.

We have chosen to omit protected tuples and to "blank out" protected attributes. Other actions may also be taken at the discretion of the security predicate. This is reasonable as the security component is placed at a low level of the DBMS.

Given this architecture, the Security Filter works as follows. For any tuple passing through the filter, the Security Filter will first evaluate security predicates defined for the relation containing the tuple. If no predicate is satisfied, the tuple is passed no higher in the system, and hence is never seen by the user. If the tuple passes the predicates for the relation, then security predicates for attributes of the tuple are checked. If these predicates are not satisfied for an attribute in the tuple, that attribute is converted to a null. Thus, the value of that attribute is never seen by the user. By using a null, no knowledge about the deleted value remains, such as its length. However, the existence of even a standardized null may itself convey information.

For efficiency, it makes sense to include a query modification module for use by the Query Execution Planner [Wie86]. This module is called by the Planner when generating a plan for processing a query to modify the plan to retrieve only data which will satisfy the discretionary controls. This should reduce the volume of data retrieved by a query, and will allow the Planner to remove by projection columns of a relation which will be "blanked out" by the Security Filter in the Executor.

To complete the security component we need two additional modules. (See Figure 4.) The first of these is a module to manage the Authorization Table. As described above, the Authorization Table includes predicates defined for each clique/object/operation triple. It must be readable by the DBMS and writable only by the security officer. Logically, it might be part of the Data Dictionary/Directory [Ber85], however, physically, it should be stored and managed separately by a module in the TCB.

The second module needed is the Security Interpreter, which is the intermediary between the Security Filter and the Authorization Table. It accepts commands from the Security Filter and returns the set of predicates which define the relevant discretionary controls. Specifically, given the name of a clique, object, and operation type, the Interpreter consults the Authorization Table for the names of all predicates which must be satisfied. It then retrieves these predicates and returns them to the Security Filter. If the object is a relation, the interpreter returns all predicates defined for the relation as well as any defined for attributes of the relation.

We assume that these predicates take the form of functionals. These functionals, when evaluated, filter a tuple by blanking out attributes of the tuple or suppressing the entire tuple. The functionals may be arbitrarily complex, and may be specified by the security

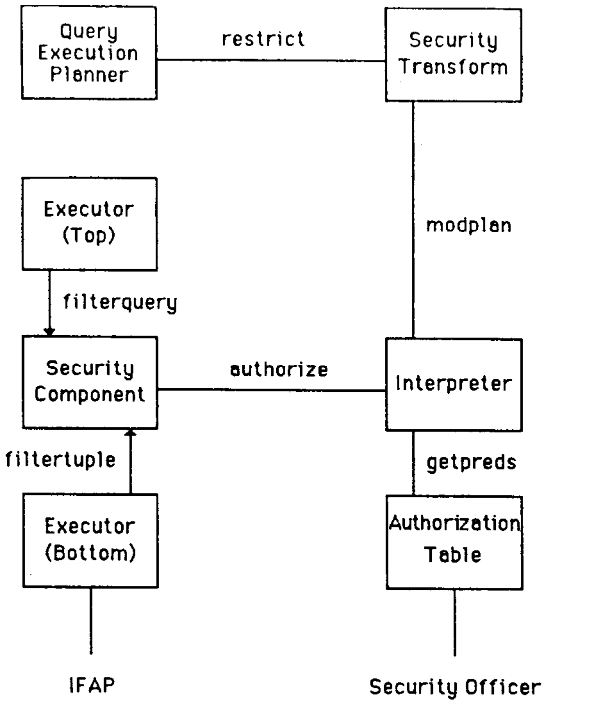

Figure 4: Modules and Operations for the Security Component

officer using any language developed for this purpose. Once compiled, they are stored in a library and their names used as privileges in the Authorization Table.

These functionals can be used to address inference problems which arise from blanking out attributes. In particular, the functionals provide a capability for making complex decisions based on the sensitivity of data. For data from lower security levels, blanking out one or more attributes in a tuple may pose no significant security problem. For more sensitive data, blanked out attributes may allow undesirable inference of information. The functionals can be defined to make this determination and suppress an entire tuple if appropriate when some of the fields are blanked out.

In summary, it is necessary for the Security Filter in the Executor, the Authorization Table and the Security Interpreter to be part of the TCB. It is not necessary for the Security Transform module that does the query modification to be part of the TCB since it is included for performance reasons and is not directly responsible for enforcement of discretionary controls.

5.1 Interfaces for the Security Component

The required interfaces for the security component of the DBMS can be derived from the discussion above along with Figure 4. The getpreds operation provides the list of predicates that define the relevant discretionary controls. The authorize operation interprets these predicates to perform the filterquery and filtertuple operations. The filterquery operation determines whether a query or update request violates discretionary controls. The filtertuple operation decides which retrieved tuples are to be passed through the security barrier and which attributes are to be obliterated. The modplan operation supplies information for modifying the query processing plan to take into account the security restrictions. The restrict operation implements query modification based on the results of modplan. Abstractly, these interfaces are listed below.

getpreds(clique, object, access type) → *predicate name*

authorize(object, operation, clique) → *set of predicates*

filterquery(query, clique) → *filtered query*

filtertuple(tuple, clique) → *filtered tuple*

modplan(object, operation, clique) → *modification instructions*

restrict(query, clique) → *modified query*

Finally, an interface is needed to the Authorization Table to allow the security officer to define the discretionary controls. The details of this interface are not defined here since the interface may be part of the Data Dictionary/Directory mechanism [Ber85].

5.2 Efficacy of Using a Security Barrier

The architecture we have described for the security component of an Ada DBMS uses the notion of a security barrier, which encapsulates everything within it and filters all communication across the barrier. As such, it operates also as a security filter for data across the interface. A security barrier enforces the controlled release of information across the barrier. An obvious application of such a notion is on the communications interface out of an otherwise self contained computer. In our case, the security barrier is a software barrier.

The notion of a TCB usually provides for a secure layer on top of which unsecure layers may be implemented. In our case, there is a trusted operating system below that provides a secure layer. However, the DBMS is much too large to be part of the TCB. Even IFAP, the indexed file access package, which is a major component of the DBMS below the security barrier, is likely to be too complex to verify. However, the security barrier is considerably simpler than the layer below, and thus more capable of being part of the TCB. This is especially true because the interface across the barrier is one of the simpler interfaces of the DBMS.

There are several problems with the security barrier approach. Because it operates as a filter between unsecure components, it is subject to penetration by covert communication channels. The difficulty of precluding convert channels even in software verified as conforming to functional specifications indicates that this is an issue that also affects modules to be included in the TCB. Nonetheless, strict control of access to the modules below the barrier, especially of their modification, will tend to limit the implantation of covert channel mechanisms. Another problem is that the system architecture must allow the barrier to control all access to the components below down to the next secure layer (in this case, down to the operating system layer).

We observe that a security barrier can be used to supplement other security measures, such as secure components, as it provides yet another method to verify that the system is secure, and it is another mechanism that must be defeated to violate system security. As interfaces are usually simpler than the components using the interface, security barriers at the interface level can be added to an already secure system at little cost to provide greater confidence in the security of the system. But where components are too complex to be trusted, security barriers provide a useful measure of security.

6 Other Issues

There are several other issues related to security of the DBMS which need to be addressed. The first of these concerns the security of logs created by the DBMS to allow recovery or to support an audit trail. These logs contain data from the database, and hence must be protected just as the database is protected [Kel85b]. The logs are segregated by mandatory access controls. The logs must contain sufficient information to support full discretionary controls comparable to those of the database. The question of the relative authority between the security officer and the database administrator is particularly apparent for the log. If sensitive data must be kept from the database administrator while allowing him to maintain the database and logs, it may be necessary to encrypt them.

The inclusion of a facility to maintain an audit trail should be considered. Audit trails are a significant deterrent to fraud [Dat83]. An audit trail allows examination of information contained in the database to determine how reliable it is. Audit trails can also be used to evaluate the effectiveness of access control mechanisms, and to confirm that policies are being followed. Audit trails must include such information as the sequence of actions taken, who initiated them, where the action was initiated, the time the action was initiated and the results of the action. Audit trails are typically generated from logs and hence place requirements on the form and content of the logs. The degree of detail maintained in an audit trail is dependent on the granularity of objects. As the granularity increases, the complexity and expense of generating the audit trail increases.

The possibility of Trojan Horse attacks is another security consideration. A Trojan Horse attack is designed to introduce flaws into the software deliberately. Two conceivable solutions are to verify that all software accessing the database are correct and to verify that the system checks all access by untrusted programs. In the case of the Ada DBMS, there are three classes of software which need to be considered: 1) the trusted parts of the DBMS, 2) the untrusted parts of the DBMS, and 3) application programs. The trusted parts of the DBMS do not present a Trojan Horse problem nor do the application programs since they must go through the trusted components of the DBMS to access and alter data. Thus, only the untrusted parts of the DBMS present a Trojan Horse problem. However, they too must go through the trusted components—the security barrier—to access data and are therefore largely prevented from doing significant damage.

An additional area for concern is the (possible) existence of covert channels. Covert channels are a means by which information may be transferred via directly observable phenomenon produced by an executing program. Covert channels in the DBMS itself are less likely than in general because no direct user code in executed. The DBMS cannot protect against covert channels implemented in application programs.

The method by which objects are reused must be addressed. Of concern is the method by which records are deleted. Records may be physically deleted immediately when a delete command is received or they may be flagged as deleted but left intact until garbage collection is done. If the latter is the case, there are security risks involved. A possible solution is to rewrite deleted records with random data.

Steps must be taken to guarantee that the DBMS cannot be circumvented by a user who accesses the database files directly. We will have the DBMS own the files and allow no one else privileges to the files. The operating system must then enforce these access restrictions on the files by providing discretionary access controls to operating system objects, such as files and program modules. The authorization table we defined to be part of the TCB to support database discretionary access controls can also be used by the operating system to protect its objects, or by any other layer to protect the objects defined at that layer (such as by an application program on top of the DBMS to protect application layer objects). Since they depend on the nature of the objects protected, the interpreter and control mechanisms, however, are specific to the particular component, although they too could be part of the TCB if desired.

An additional level of security can be provided with cryptographic techniques [Den82]. We have chosen not to include these techniques because we are primarily concerned with controlling access to data. However, if additional security is needed, or if the operating system cannot be trusted, cryptographic techniques may become necessary.

We do not attempt to implement history-dependent access controls [Har76] as these techniques are inherently non-monotonic. While it may be possible to survey the DBMS log and use heuristics and expert systems techniques to implement this type of access control, these techniques require an understanding of all prior knowledge from this or other sources, as well as all possible inference rules, in order to determine whether allowing access to the data in question will compromise security. In the absence of any assumptions about prior knowledge and possible inference rules, this problem is potentially undecidable.

7 Conclusions

We have presented the framework of a security component for an Ada DBMS. Data filters first check discretionary controls defined for relations, then tuples, and then attributes. These data filters are implemented as part of a security barrier that controls all access to the module that retrieves and updates selected tuples for a relation. We also include a module to perform query modification to improve performance by reducing the amount of filtering required by the security barrier.

We measured our framework by evaluating according to several criteria the range of security models that our framework can support. These criteria are open versus closed system architecture, granularity of protected objects, monotonicity, specification of security controls, actions when there are multiple predicates that apply to a request, response to security violations, and the location of security checking.

The resulting architecture for the security component is simple and clean. Three of the four modules must be trusted. Each has a simple well-defined function so that this is not an unreasonable assumption. These modules along with the mandatory controls implemented in the operating system provide the required security enforcement capabilities for the Ada DBMS without requiring large components of the DBMS to be trusted. In a commercial environment, security is measured by the worth of the data protected. In a military environment, the value of data can be quite high.

8 Acknowledgements

We wish to thank Sham Navathe, Murray Berkowitz, Bill Brykczynski, Chet Coates, Mike Hale, and Terry Mayfield for their contributions to the ideas presented in this paper.

9 References

[Ber85] Berkowitz, M. and S. Navathe, "WIS Data Dictionary/Directory," Tech. Report, IDA, Dec. 1985.

[Che85] Cheriton, D., et. al., "Operating Systems Task Force Document," Tech. Report, IDA, Dec. 1985.

[Dat83] Date, C., *An Introduction to Database Systems*, Addison-Wesley, 1983.

[Day78] Dayal, U. and P. Bernstein, "The Fragmentation Problem: Lossless Decomposition of Relations into Files," Tech Report. CCA-78-13, Computer Corporation of America, Cambridge, MA, November, 1978.

[Den82] Denning, D., *Cryptography and Data Security*, Addison-Wesley, Reading, Mass., 1982.

[DOD83] *Department of Defense Trusted Computer System Evaluation Criteria*, CSC-STD-001-83, August 1983, available through GPO.

[Fri86] Friedman, F., et. al. "Reference Model for Ada Interfaces to Database Management Systems," *Proc. IEEE 2nd Int. Conf. on Data Engineering*, February 1986.

[Fri86a] Friedman, F. and B. Brykczynski, "Ada/SQL: A Standard, Portable Ada-DBMS Interface," *Proc. IEEE 2nd Int. Conf. on Data Engineering*, February 1986.

[Gra72] Graham, G. and P. Denning, "Protection—Principles and Practice," *Proc. AFIPS* **40**, 1972.

[Gri76] Griffiths, P. and G. Wade, "An Authorization Mechanism for a Relational Database System," *ACM Trans. on Database Systems*, **1**:3, September 1976.

[Har76] Hartson, R. and D. Hsiao, "Full Protection Specifications in the Semantic Model for Database Protection Languages," *Proc. ACM Ann. Conf.*, Oct. 1976.

[Kel85] Keller, A., "Updating Relational Databases Through Views," Ph.D. Thesis, Stanford University, 1985.

[Kel85a] Keller, A., "Indexed File Access for Ada," Tech. Report, IDA, Nov. 1985.

[Kel85b] Keller, A., "Logs and Triggers," Tech. Memo, IDA, Sept. 1985.

[Rot80] Rothnie, J., et. al., "Introduction to a System for Distributed Databases (SDD-1)," *ACM Trans. on Database Systems*, **5**:1, March 1980.

[Spo84] Spooner, D. and E. Gudes, "A Unifying Approach to the Design of a Secure Database Operating System," *IEEE Trans. of Soft. Eng.*, **SE-10**:3, May 1984.

[Sto75] Stonebraker, M. and R. Rubinstein, "The INGRES Protection System," *Proc ACM SIGMOD Conf.*, 1975.

[Wie86] Wiederhold, G., et. al., "Modularization of an Ada Database," Tech. Report, IDA, December 1985.

[WIS84] *WIS Project Definition*, Tech. Report, IDA, 1984.