A Multi-Purpose Implementation of Mandatory Access Control in Relational Database Management Systems

Walid Rjaibi Paul Bird

IBM Toronto Software Laboratory
8200 Warden Avenue
Markham, Ontario
Canada
{wrjaibi, pbird}@ca.ibm.com

Abstract

Mandatory Access Control (MAC) implementations in Relational Database Management Systems (RDBMS) have focused solely on Multilevel Security (MLS). MLS has posed a number of challenging problems to the database research community, and there has been an abundance of research work to address those problems. Unfortunately, the use of MLS RDBMS has been restricted to a few government organizations where MLS is of paramount importance such as the intelligence community and the Department of Defense. The implication of this is that the investment of building an MLS RDBMS cannot be leveraged to serve the needs of application domains where there is a desire to control access to objects based on the label associated with that object and the label associated with the subject accessing that object, but where the label access rules and the label structure do not necessarily match the MLS two security rules and the MLS label structure. This paper introduces a flexible and generic implementation of MAC in RDBMS that can be used to address the requirements from a variety of application domains, as well as to allow an RDBMS to efficiently take part in an end-to-end MAC enterprise solution. The paper also discusses the extensions made to the SQL compiler component of an RDBMS to incorporate the label access rules in the access plan it generates for an SQL query, and to prevent unauthorized leakage of data that could occur as a result of traditional optimization techniques performed by SQL compilers.

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

1 Introduction

Mandatory Access Control (MAC) is a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity[8]. A well-known implementation of MAC is Multilevel Security (MLS), which, traditionally, has been available mainly on computer and software systems deployed at highly sensitive government organizations such as the intelligence community or the U.S. Department of Defense. The Basic model of MLS was first introduced by Bell and LaPadula[9]. The model is stated in terms of objects and subjects. An object is a passive entity such as a data file, a record, or a field within a record. A subject is an active process that can request access to objects. Every object is assigned a classification, and every subject a clearance. Classifications and clearances are collectively referred to as labels. A label is a piece of information that consists of two components: A hierarchical component and a set of unordered compartments. The hierarchical component specifies the sensitivity of the data. For example, a military organization might define levels Top Secret, Secret, Confidential and Unclassified. The compartments component is nonhierarchical. Compartments are used to identify areas that describe the sensitivity or category of the labeled data. For example, a military organization might define compartments NATO, Nuclear and Army. Labels are partially ordered in a lattice as follows: Given two labels L_1 and L_2, $L_1 >= L_2$ if and only if the hierarchical component of L_1 is greater than or equal to that of L_2, and the

compartment component of L1 includes the compartment component of L_2. L_1 is said to *dominate* L_2. MLS imposes the following two restrictions on all data accesses:

- The Simple Security Property or "No Read Up": A subject is allowed a read access to an object if and only if the subject's label dominates the object's label.
- The *-Property (pronounced the star property) or "No Write Down": A subject is allowed a write access to an object if and only if the object's label dominates the subject's label.

1.1 Problem Statement

MAC implementations in Relational Database Management Systems (RDBMS) have focused solely on MLS. MLS has posed a number of challenging problems to the database research community, and there has been an abundance of research work to address those problems. There has also been three commercial MLS RDBMS offerings, namely, Trusted Oracle[16], Informix OnLine/Secure[17], and Sybase Secure SQL Server[20]. Unfortunately, the use of MLS RDBMS has been restricted to a few government organizations where MLS is of paramount importance such as the intelligence community and the Department of Defense. In fact, very few commercial organizations need such type of security. The implication of this is that the investment of building an MLS RDBMS cannot be leveraged to serve the needs of application domains where there is a desire to control access to objects based on the label associated with that object and the label associated with the subject accessing that object, but where the label access rules and the label structure do not necessarily match the MLS two security rules and the MLS label structure (*i.e.,* a hierarchical component and a set of unordered compartments). The question that begs to be asked is therefore the following: Do such application domains exist and, if so, what are they?

We contend that the answer to that question is an unequivocal yes. Privacy[19] is one example of such application domain. Generally, a privacy policy indicates for which purposes an information is collected, whether or not it will be communicated to others, and for how long that information is retained before it is discarded. For example, a user cannot access a customer record for the purpose of sending that customer marketing information if that customer did not agree to receive such information. Access to privacy-sensitive data can be regarded as analogous to access to MLS data in the sense that in both cases there is a tag associated with the object being accessed and the subject accessing that object. The tag represents a "purpose" in the case of the former and represents a "security label" in the case of the latter. Unfortunately, a MAC implementation in an RDBMS that strictly implements MLS fails to address privacy requirements for the following two main reasons. First, MLS labels include a hierarchical component that is not applicable in the case of privacy. Next, the MLS security properties do not apply in the context of privacy. For example, to read an object in MLS, the subject's compartment component must include that object's compartment component (the simple security property). In privacy, the rule is exactly the opposite. That is, if an object is tagged with the purposes *marketing* and *purchase*, then a user accessing that object for the purpose of sending marketing information must be allowed to access that object.

Another application domain is private banking. In private banking, country laws and regulations often require to limit the amount of data that can be viewed by a bank employee. For example, Swiss banking laws do not allow a Swiss bank employee located in Toronto to access account information for customers based in Switzerland. Typically, banking applications code this fine-grained access control in the application itself, as opposed to delegating this task to the RDBMS. Unfortunately, this *application-aware* approach has made enterprise security policies a laborious and complex task. It also has the drawback of exposing the security policies to the application programmers. If each customer account is tagged with a label indicating the geographical location of the customer and if each bank employee can be assigned a label that also indicates the geographical location of that employee (for example, based on the system security context established when that employee logs on to the database), then an RDBMS that implements a form of MAC where the database administrator could define the label structure and the label access rules could relieve the applications from implementing such fine-grained access control policies.

Moreover, the ever increasing enterprise demands for more security has led to the emergence of label security products that provide the ability to set up and control access based upon labels throughout an entire network from end-to-end. For example, such label security products have the ability to control the network to decide whether or not a particular labeled data row can be transmitted on a particular channel or be delivered to a particular workstation on that network. An important advantage of such label security products is their ability to offer a centrally managed tool for defining label access policies and for assigning access labels to users as well as to other entities on the network. Traditional implementations of MAC in RDBMS (*i.e.,* MLS) do not offer the required flexibility to efficiently integrate with such label security products and to provide pervasive system coverage using a unified and centrally managed label access policy.

Therefore, there is a need for a flexible and generic implementation of MAC in RDBMS that can be used to address the requirements from a variety of application domains, including those of MLS, and to efficiently take part in an end-to-end MAC enterprise solution.

1.2 Contributions

The contributions made in this paper can be summarized as follows:

1. A methodology to define labels and to set up a database table such that access to a row in that table is based upon the label associated with that row and the label associated with the user accessing that row. More specifically, the methodology introduces a number of extensions to SQL that would allow a database administrator to:
   - Define label types
   - Define label access rules and exceptions to them
   - Assign labels and exceptions to database users
   - Attach a label type and a set of label access rules to a database table
2. Extensions to the SQL compiler component of an RDBMS to:
   - Incorporate the label access rules in the access plan it generates for an SQL query
   - Prevent unauthorized leakage of data that could occur as a result of traditional optimization techniques performed by SQL compilers
3. Extensions to the runtime processor component of an RDBMS to enforce label access rules
4. A method to allow an RDBMS to efficiently take part in an end-to-end MAC enterprise solution

1.3 Synopsis

Section 2 gives a brief survey of MAC implementations in RDBMS. Section 3 introduces our methodology for defining labels and for setting up a database table such that access to a row in that table is based upon the label associated with that row and the label associated with the user accessing that row. Section 4 presents our extensions to the SQL compiler component of an RDBMS to incorporate the label access rules in the access plan it generates for an SQL query, and to prevent unauthorized leakage of data that could occur as a result of traditional optimization techniques performed by SQL compilers. Section 5 describes our extensions to the methodology introduced in section 3 in order to allow an RDBMS to efficiently take part in an end-to-end MAC enterprise solution. Lastly, section 6 summarizes our results and discusses future work.

2 Related Work

MAC implementations in Relational Database Management Systems have focused solely on MLS, which is of paramount importance to a few government organizations such as the intelligence community or the Department of Defense. In fact, there has been an abundance of research within the last two decades or so in the area of multilevel secure relational databases. The results of such research can be divided into three broad areas as follows.

2.1 Multilevel Secure Relational Database Models

The Sea View model[1] was the pioneering formal multilevel secure relational database designed to provide mandatory access control. It extended the concept of a database relation to include the security labels. A relation that is extended with the security labels is called a multilevel relation. The Sea View model also coined the concept of *polyinstantiation*, which refers to the simultaneous existence of multiple tuples with the same primary key, where such tuples are distinguished by their security labels. In order to avoid covert channels, subjects with different security labels are allowed to operate on the same database relation through the use of polyinstantiation[1]. The Jajodia-Sandhu model[2] was derived from the Sea View model. It was shown in [3] that the Sea View model can result in the proliferation of tuples on updates and the Jajodia-Sandhu model addresses this drawback. The Smith-Winslett model[4] was the first model to extensively address the semantics of an MLS database. The MLR model[5] is based on the Jajodia-Sandhu model, and also integrates the belief-based semantics of the Smith-Winslett model. It was shown in [7] that all the aforementioned models can present users with some information that is difficult to interpret. The BCMLS model[6] addresses those concerns by including the semantics of an unambiguous interpretation of all data presented to the users.

2.2 Multilevel Secure RDBMS Architectures

Multilevel secure RDBMS architectures schemes can be divided into two general categories: The Trusted Subject architecture and the Woods Hole architectures.

The Trusted Subject architecture[10] is a scheme that contains a trusted RDBMS and a trusted operating system. The RDBMS is custom-developed to include all the required security rules, but uses the associated trusted operating system to make actual disk data accesses. A benefit of this scheme is that the

RDBMS has access to all levels of data at the same time, which minimizes retrieval and update processing. However, this scheme results in a special purpose RDBMS that requires a large amount of trusted code to be developed and verified along with the normal RDBMS features.

The Woods Hole architectures assume that an untrusted off-the-shelf RDBMS is used to access data and that trusted code is developed around that RDBMS to provide an overall secure RDBMS. They can be divided into two main categories: The kernelized architectures and the distributed architectures[10, 11].

The kernelized architecture scheme uses a trusted operating system and multiple copies of the RDBMS, where each copy is associated with some trusted front-end. Each pair (trusted front-end, RDBMS) is associated with a particular security level. The trusted operating system ensures that data at different security levels is stored separately, and that each copy of the RDBMS gets access to data that is authorized for its associated security level. A benefit of this scheme is that data at different security levels is isolated in the database, which allows for higher level assurance. However, this scheme results in an additional overhead as the trusted operating system needs to separate data at different security levels when it is added to the database and might also need to combine data from different security levels when data is retrieved by an RDBMS copy that is associated with a high security level.

The distributed architecture scheme uses multiple copies of the trusted front-end and RDBMS, each associated with its own database storage. In this architecture scheme, an RDBMS at security level l contains a replica of every data item that a subject at level l can access. Thus, when data is retrieved, the RDBMS retrieves it only from its own database. Another benefit of this architecture is that data is physically separated into separate hardware databases. However, this scheme results in an additional overhead when data is updated as the various replicas need to be kept in sync.

2.3 Multilevel Secure Transaction Processing

Although the two MLS security properties described above prevent direct legal flow of information from a security level to another nondominated security level, they are not sufficient to ensure that security is not compromised since it could be possible for leakage of information to occur through indirect means via covert channels. A covert channel can easily be established with conventional concurrency control algorithms such as two-phase locking (2PL) and timestamp ordering (TO). In both 2PL and TO algorithms, whenever there is contention for the same data item by transactions executing at different security levels, a lower level transaction may be either delayed or suspended to ensure correct execution. In such a scenario, two colluding transactions executing at high and low security levels can establish an information flow channel from a high security level to a low security level by accessing the selected data item according to some agreed-upon code[12].

Considerable effort has been devoted to the development of efficient, secure algorithms for the major schemes of RDBMS architectures described above. In [13], Keefe *et al.* present a formal framework for secure concurrency control in multilevel databases. Lamport[14] offers solutions to the secure readers/writers problem. While these solutions are secure, they do not yield serializable schedules when applied to transactions, and they suffer from the problem of starvation, *i.e.,* transactions that are reading data items at a low security level may be delayed indefinitely[18]. In [15], Ammann and Jajodia offer two timestamp-based algorithms that yield serializable schedules, but both suffer from starvation. On the commercial secure RDBMS side, both Trusted Oracle RDBMS[16] and Informix OnLine/Secure RDBMS[17] offer concurrency control solutions that are free from covert channels.

3 Methodology for Setting up MAC in an RDBMS

The methodology we propose allows a database administrator to define labels and to set up a database table such that access to a row in that table is based upon the label associated with that row and the label associated with the user accessing that row. More specifically, the methodology allows the database administrator to:

- Define label types
- Define label access rules and exceptions to them
- Assign labels and exceptions to database users
- Attach a label type and a set of label access rules to a database table

We now introduce our extensions to SQL to implement this methodology. The goal of this exercise is not to describe the blueprint for the implementation. Rather, we will focus on the new SQL concepts that must be implemented to support such methodology. Also, we have chosen not to overload the paper with the details of the exact syntax of the SQL extensions proposed, as we believe that such level of details is more appropriate for a standardization proposal to the SQL standard committee. However, we will illustrate the syntax and the concepts introduced via examples.

3.1 Label Component

A label component is a database entity that can be created, altered and dropped. It is introduced as a

building block for labels (*i.e.,* a label is composed of one or more label components). The label component definition specifies the set of valid elements for that label component. This set of elements can be either ordered or unordered (the default). In an ordered set, the order in which the elements appear is important: The rank of the first element is higher than the rank of the second element, the rank of the second element is higher than the rank of the third element, and so on. To allow database administrators to create, alter and drop label components, we introduce the CREATE, ALTER and DROP label component SQL statements. The CREATE LABEL COMPONENT SQL statement creates a label component that can be used to define a label type. The ALTER LABEL COMPONENT SQL statement permits to add or drop an element to/from a label component. The DROP LABEL COMPONENT SQL statement drops a label component.

Example 1

The following SQL statement creates a label component called level and specifies the set of valid values for this label component.

```
CREATE LABEL COMPONENT level
OF TYPE varchar(15)
USING ORDERED SET
{"TOP SECRET", "SECRET", "CLASSIFIED"}
```

The following SQL statement creates a label component called compartments and specifies the set of valid values for this label component. Note that the set specified is unordered.

```
CREATE LABEL COMPONENT
compartments OF TYPE varchar(15)
USING SET
{"NATO", "NUCLEAR", "ARMY"}
```

The following SQL statement adds a new element to the level component and specifies the rank of this new element within the ordered set.

```
ALTER LABEL COMPONENT level
ADD ELEMENT "UNCLASSIFIED"
AFTER "CLASSIFIED"
```

The following SQL statement drops the level component.

```
DROP LABEL COMPONENT level
```

3.2 Label Type

The relationship between a label and a label type is analogous to the relationship between a data row and a table schema. As the table schema defines the set of columns that make up a data row, so the label type defines the set of label components that make up a label. To allow database administrators to create, alter and drop label types, we introduce the CREATE, ALTER and DROP label type SQL statements. The CREATE LABEL TYPE creates a label type by specifying the label components that make up such label type. The ALTER LABEL TYPE alters the definition of a label type by adding or dropping a label component to/from that label type. The DROP LABEL TYPE SQL statement drops a label type.

Example 2

The following SQL statement creates a label type called MLS and specifies its label components. Note the keyword MULTIVALUED next to the compartments component. This indicates that the compartments component can have more than a single value at a time. This keyword can only be specified for label components based on an unordered set (section 3.4 explains the reason behind this choice).

```
CREATE LABEL TYPE MLS
COMPONENTS level,
compartments MULTIVALUED
```

The following SQL statement drops the level component from label type MLS.

```
ALTER LABEL TYPE MLS DROP level
```

The following SQL statement drops the MLS label type.

```
DROP LABEL TYPE MLS
```

3.3 Access Labels and Row Labels

We distinguish two types of labels: *Access labels* and *row labels*. Access labels are created and assigned to database users, which, in conjunction with the label access rules (section 3.4), determine which labeled rows these users have access to. To allow database administrators to create, drop, grant and revoke access labels, we introduce the CREATE, DROP, GRANT and REVOKE access label SQL statements. The CREATE ACCESS LABEL SQL statement creates an access label based on an existing label type. The GRANT ACCESS LABEL SQL statement grants an access label to a database user. The REVOKE ACCESS LABEL SQL statement revokes an access label from a database user. The DROP ACCESS LABEL SQL statement drops an access label and revokes it from any database user to whom it has been granted.

Example 3

The following SQL statement creates an access label.

CREATE ACCESS LABEL L_1
OF LABEL TYPE MLS
level "SECRET", compartments "NATO"

The following SQL statement grants access label L_1 to database user Joe.

GRANT ACCESS LABEL L_1
TO USER Joe

The following SQL statement revokes access label L_1 from database user Joe.

REVOKE ACCESS LABEL L_1
FROM USER Joe

The following SQL statement drops access label L_1.

DROP ACCESS LABEL L_1

A row label labels a data row in a database table. To allow database users to provide a row label when inserting or updating a row in a database table, we introduce the ROWLABEL function. ROWLABEL is a means of providing the label value of a data row.

Example 4

The following INSERT SQL statement shows how the row label can be provided using the ROWLABEL function. The statement inserts a row into a database table called T1 having two columns A and B both of type integer. We assume that rows in table T1 are labeled with a label of label type MLS defined above.

INSERT INTO T1 VALUES
(ROWLABEL("SECRET", "NATO"), 1, 2)

The following SQL statement shows how the ROWLABEL function can be used to update the level component of the row label for the row inserted above.

UPDATE T1 SET
ROWLABEL(level) = "TOP SECRET"
WHERE A = 1 AND B = 2

3.4 Label Access Policy

A label access policy defines the label access rules that the RDBMS evaluates to determine whether or not a database user is allowed access to a labeled data row in a database table. Access rules can be divided into two categories: Read access rules and write access rules. Read access rules are applied by the RDBMS when a user attempts to read a labeled data row (e.g., a SELECT statement). The RDBMS applies the write access rules when a user attempts to insert, update or delete a labeled data row. In both cases, an access rule is a predicate that puts together the same component from an access label and a row label and an operator as follows:

Access Label *component-name*
<operator>
Row Label *component-name*

The type of operator allowed depends on the label component. For label components based on an ordered set, the operator can be any of the relational operators $\{=, <=, <, >, >=, !=\}$. For label components based on an unordered set, the operator must be one of the set operators {IN, INTERSECT}. Recall from section 3.2 that a label component based on an unordered set can be multivalued. That is, it can contain more than a single value at a time. Thus, when comparing multivalued label components we are actually comparing data sets. This is the reason why the operators supported are set operators, *i.e.,* inclusion and intersection. Obviously, certain RDBMS could choose to support additional operators but we contend that the ones given above would be the most commonly used. To allow database administrators to create, alter and drop label policies, we introduce the CREATE, ALTER and DROP label policy SQL statements. The CREATE LABEL POLICY SQL statement creates a label access policy for a given label type by specifying one or more read access rules and one or more write access rules. The ALTER LABEL POLICY SQL statement permits the addition or dropping an access rule to/from a label access policy. The DROP LABEL SQL statement drops a label access policy.

Example 5

The following SQL statement creates a label access policy that implements the two MLS properties introduced in section 1 above (*i.e.,* "No Read Up" and "No Write Down").

CREATE LABEL POLICY mls-policy
LABEL TYPE MLS
READ ACCESS RULE rule1
  ACCESS LABEL level >= ROW LABEL level
READ ACCESS RULE rule2
  ROW LABEL compartments IN
  ACCESS LABEL compartments
WRITE ACCESS RULE rule1
  ACCESS LABEL level <= ROW LABEL level

```
WRITE ACCESS RULE rule2
   ACCESS LABEL compartments IN
   ROW LABEL compartments
```

The following SQL statement drops read access rule rule2 from label access policy mls-policy.

```
ALTER LABEL POLICY mls-policy
DROP READ ACCESS RULE rule2
```

The following SQL statement drops label access policy mls-policy.

```
DROP LABEL POLICY mls-policy
```

3.5 Exceptions

Exceptions are introduced to provide the flexibility for some database users to bypass one or more access rules. For example, in an MLS context, it is often the case that some special users are allowed to write information to lower security levels even though this is in contradiction with the *-security property. Thus, exceptions are introduced to allow the database administrator to grant a database user an exception to bypass one or more access rules in a particular label access policy. To allow database administrators to grant and revoke exceptions, we introduce the GRANT and REVOKE exception SQL statements. The GRANT EXCEPTION SQL statement grants a database user an exception to bypass one or more access rules in a label access policy. The REVOKE EXCEPTION SQL statement revokes a previously granted exception from a database user.

Example 6

The following SQL statement grants an exception to database user Joe so that he can bypass the write access rules in label access policy mls-policy.

```
GRANT EXCEPTION
ON WRITE ACCESS RULE rule1, rule2
FROM LABEL POLICY mls-policy
TO USER Joe
```

The following SQL statement revokes the above exception from user Joe.

```
REVOKE EXCEPTION
ON WRITE ACCESS RULE rule1, rule2
FROM LABEL POLICY mls-policy
FROM USER Joe
```

3.6 Labeled Tables

A labeled table is a database table that contains labeled data rows. When the database administrator creates a labeled table he/she specifies the label type and the label access policy to be used for that table. The label type determines the structure of the label to be used to label the table's data rows and the label access policy determines the access rules to be used for enforcing access to that labeled table. To allow database administrators to create labeled tables, we extend the CREATE TABLE SQL statement by a new optional clause to specify the label type and the label access policy.

Example 7

The following SQL statement creates a database table T1 and specifies the label type and the label access policy. Note that in our examples so far we have used MLS-like label types and label access policies because they are well understood by the database research community. But it is obvious that one can follow the methodology given in this paper to define any label type and any label access policy, and attach them to a database table.

```
CREATE TABLE T1 (A integer, B integer)
LABEL TYPE MLS
LABEL POLICY mls-policy
```

When creating such table, the RDBMS internally adds a third column to store the label associated with each row in this table. The choice of the column's type depends on the label type. For example, if the label type is made up of a single component of type, say varchar(15), then the column's type would be varchar(15). If the label type is made up of more than a single column then the column's type must be an Abstract Data Type (ADT). ADTs have been introduced in SQL'99[21] and are supported by most commercial RDBMS. Alternatively, the RDBMS could choose not use an ADT and store the different label components in separate columns.

4 Extensions to the SQL Compiler Component in an RDBMS

When a labeled table is accessed, the RDBMS needs to enforce two levels of access control. The first level is the traditional Discretionary Access Control (DAC) which is implemented by all commercial RDBMS[21]. That is, the RDBMS verifies whether the user attempting to access the table has been granted the required privilege to perform the requested operation on that table. A discussion of this level of access control is beyond the scope of this paper. The second level is MAC. That is, for each data row accessed, the RDBMS verifies whether the user is allowed access to that row based on the label associated with the row and the user's access label.

4.1 Enforcing MAC on Labeled Tables

There are two possible ways that MAC can be enforced when a labeled table is accessed. The first possibility is for the SQL compiler to modify any query that refers to a labeled table in order to incorporate the access rules from the label access policy associated with that table in the form of regular predicates. Next, the SQL compiler compiles the modified query and generates an access plan for the query in the normal fashion. The main advantage of such an approach is its simplicity. However, it has a major drawback: The access plan generated for a query that refers to a labeled table cannot be reused by other users because it is dependent on the access label of the user who issued the query. Note that some commercial RDBMS cache the access plan generated for an SQL query so that it can be reused the next time the SQL query is submitted. This has some performance benefits as it eliminates the need to recompile the query. Another drawback of this approach is that it could result in unauthorized leakage of data if special care is not taken by the SQL compiler. This will be detailed further in section 4.2.

The second possibility is to not modify a query that refers to a labeled table. Rather, the SQL compiler inserts logic into the access plan that implements the access rules from the label access policy associated with any labeled table referred to in the query. Thus, when the access plan is executed, the access rules from the label access policy associated with a labeled table are evaluated for each data row when that labeled table is accessed. The general processing algorithm to be inserted in the access plan for a labeled table is as follows.

```
Begin
  Fetch the user's access label (e.g., from a
  system catalog table)
  if (SELECT access)
  {
     for each row accessed
     {
        if (read access rules do not permit access)
        {
           Skip row
        }
     }
  }
  else
  {
        // INSERT, UPDATE, or DELETE access
        for each row
        {
           if (INSERT or UPDATE)
           {
              if (the row label provided is not valid with
              respect to the label type associated with
              the labeled table)
                 Reject INSERT or UPDATE
           }
           if (write access rules do not permit access)
              Reject INSERT, UPDATE or DELETE
        }
  }
End
```

This second approach addresses the two shortcomings of the previous approach (i.e., query modification). That is, it allows the cached access plan to be reused because the access label of the user who issued the query is acquired at runtime, and it is more secure as it will be demonstrated in section 4.2.

4.2 Predicates Evaluation Sequence

SQL compilers have traditionally been guided by performance reasons in selecting the order in which the predicates contained in a query are evaluated. For example, more selective predicates are often evaluated first to narrow down the set of rows to be passed on to a subsequent join because join operations are costly. If the method chosen to enforce MAC on a labeled table is based on query modification to incorporate the access rules in the form of regular predicates, then special care must be taken in selecting the order in which the predicates on that table are evaluated to avoid unauthorized leakage of labeled data rows. For example, suppose that a query has a predicate on a labeled table that involves a User-Defined Function (UDF). Further suppose that this UDF takes the whole data row as an input parameter and that the UDF source code makes a copy of the data row outside the database (or sends it as an e-mail to some destination). Now, suppose that some data row R cannot be returned to the user who issued the query because this would violate the access rules from the label access policy associated with this labeled table. If the predicate involving the UDF is evaluated prior to evaluating the predicates that implement the access rules then data row R will be consumed by the UDF and consequently leaked to an unauthorized user.

If the RDBMS chooses the query modification method to enforce MAC on a labeled table, then it must ensure that the predicates that implement the access rules are evaluated before any other predicate so that no labeled row leakage could occur. The alternative approach that is not based on query modification evaluates the access rules immediately after the row is accessed, and before any predicate is evaluated. It is therefore more secure than the query modification approach. It also allows the SQL compiler to continue to select the order in which predicates are evaluated in the usual way.

4.3 Index-Only Access Methods

When selecting an access plan, SQL compilers choose between three methods of accessing the data in a database table: Scanning the entire table sequentially, locating specific table rows by first accessing an index on the table, or accessing just an index on the table if all the required columns are part of the index key. This latter method is known as index-only access. SQL compilers usually rely on the statistics available about the table and the indices to choose between those three access methods. If an index only plan is selected then the label column is not available and therefore the access rules from the label access policy associated with the table cannot be evaluated. MLS RDBMS extended the primary key on an MLS relation with the security label column in order to allow the simultaneous existence of multiple tuples with the same (non extended) primary key (*i.e.,* polyinstantiation)[1]. We borrow this idea from the MLS work to extend every index created on a labeled table (including the primary key) with the row label column(s). This would allow SQL compilers to continue to choose index only access methods when this is appropriate, and for the access rules from the label access policy associated with the table on which the index is created to be evaluated.

5 Methodology for an End-to-end MAC Enterprise Solution

The ever-increasing enterprise demands for more security has led to the emergence of label security products that provide the ability to set up and control access based upon labels throughout an entire network from end to end. For example, such label security products have the ability to control the network to decide whether or not a particular labeled data row can be transmitted on a particular channel or be delivered to a particular workstation on that network. Cryptek[22] is an example of such a label security product. An important advantage of such label security products is their ability to offer a centrally managed tool for defining label access policies and for assigning access labels to users as well as to other entities on the network. We contend that a MAC implementation in RDBMS should offer the flexibility to integrate with a label security product for the following reasons:

1. Eliminate the need for the system administrator to define the label access rules in more than a single location (*i.e.,* both in the RDBMS and in the label security product)

2. Eliminate the need for the system administrator to assign access labels to users in more than a single location

3. Allow the access to a labeled data row in the database to be based on more sophisticated label access rules that a particular implementation of MAC in an RDBMS may not allow to express

We will now show how the methodology described earlier in this paper could be extended to allow an RDBMS to take part in such an end-to-end MAC scheme by providing the flexibility to integrate with a label security product.

5.1 Integration Approach

Recall from section 3.6 that we have extended the CREATE TABLE SQL statement with an optional clause to specify the label type and the label access policy. We further extend this SQL statement such that the LABEL POLICY clause could either specify the name of a label access policy defined within the RDBMS, or a label access policy defined externally to the RDBMS (*i.e.,* within a label security product). The keyword EXTERNAL is introduced to support this latter possibility as shown below.

```
CREATE TABLE T1 (A integer, B integer)
LABEL TYPE some-label-type
LABEL POLICY EXTERNAL
```

When a data row in such a table is accessed, the RDBMS needs to supply the ID of the user making the access together with the data row label and the table name to the label security product through a well-defined interface. The label security product evaluates the label access rules based on the information received from the RDBMS and returns a response to the RDBMS through that same interface. The response could be a Boolean flag indicating whether or not the access should be allowed.

The SQL compiler will now need to take into account where the label access rules are defined when inserting logic into an access plan to enforce MAC on a labeled table. Thus, a more general description of the algorithm to be inserted in the access plan for a labeled table is as follows.

```
Begin
  if (policy defined within RDBMS)
  {
    Fetch the user's access label (e.g., from a
    system catalog table)
  }
  if (SELECT access)
  {
    for each row accessed
    {
      if (policy defined within RDBMS)
      {
        if (read access rules do not permit access)
        {
          Skip row
```

```
            }
        }
        else
        {
            response = callLabelSecurityProduct(userid,
                        rowlabel, table-name)
            if (response is No)
            {
                Skip row
            }
        }
    }
}
else
{
        // INSERT, UPDATE, or DELETE access
        for each row
        {
            if (INSERT or UPDATE)
            {
                if (the row label provided is not valid with
                respect to the label type associated with
                the labeled table)
                    Reject INSERT or UPDATE
            }
            if (policy defined within RDBMS)
            {
                if (write access rules do not permit access)
                    Reject INSERT, UPDATE or DELETE
            }
            else
            {
                response = callLabelSecurityProduct
                            (userid, rowlabel, table-name)
                if (response is No)
                {
                    Reject INSERT, UPDATE or DELETE
                }
            }
        }
}
End
```

Clearly, the calls to the label security product, which is external to the RDBMS, would cause a performance degradation. In the next section, we will show how this performance degradation could be minimized.

5.2 Performance Improvement

To minimize the performance degradation that could result from the calls to the label security product, a caching technique could be used. Before making the call to the label security product, the RDBMS would first check the cache to see if a similar call was made earlier, and if so fetches the response directly from the cache. The cache structure could look as follows.

Userid	RowLabel	Table	Access	Resp.
Joe	L	T	Read	Yes
Bob	L'	T	Write	No

Table 1: Label security product responses cache

To ensure that the cache entries are always valid, the label security product must signal to the RDBMS through a well-defined interface any changes to the label access rules associated with a database table, or to the access labels assigned to a database user. When such a signal is received, the RDBMS invalidates the cache entries that are affected by the change in label access rules or user access labels. For example, if the label access rules associated with table T have changed, then all cache entries for table T must be invalidated. Similarly, if the access label for user Joe has changed or has been revoked, then all cache entries for user Joe must be invalidated.

6 Conclusion and Future Directions

This paper has introduced a flexible and generic implementation of MAC in RDBMS that can be used to address the requirements from a variety of application domains, as well as to allow an RDBMS to efficiently take part in an end-to-end MAC enterprise solution. This implementation differs from traditional MAC implementations in RDBMS, which have focused solely on MLS, and thus cannot be leveraged to serve the needs of application domains where there is a desire to control access to objects based on the label associated with that object and the label associated with the subject accessing that object, but where the label access rules and the label structure do not necessarily match the MLS two security rules and the MLS label structure (*i.e.,* a hierarchical component and a set of unordered compartments). Moreover, such implementations do not offer the flexibility to integrate with an external label security product and therefore cannot take part in an end-to-end MAC enterprise solution.

There are a number of additional problems related to implementing a generic MAC solution in an RDBMS that have not been addressed in this paper. These will be the subject of our future work. For example, triggers could cause labeled data rows to flow from a labeled table to a nonlabeled table if the subject of a trigger is a labeled table but the target of that trigger is a nonlabeled table. Without proper flow control measures, triggers could cause unauthorized leakage of information to occur. Also, there needs to be a mechanism to accommodate views based on labeled tables. For example, if a view is based on a join between two labeled tables how would the row label of a join result row be selected. Should the RDBMS make the decision about how to combine labels? or should the RDBMS offer the flexibility that would allow database administrators to provide the rules for combining two labels from the same label type?

Acknowledgements

Some of the ideas expressed in this paper were generated when the first author was a Research Staff Member at the IBM Zurich Research Lab (ZRL). The first author would like to thank Dr. Michael Waidner, manager Network Security & Cryptography, for giving him the opportunity to start up the database security research activity at ZRL. The first author would also like to thank his wife Hue Phan Dam for her valuable comments on an earlier version of this paper and for her help with the examples.

Trademarks

Disclaimer

The views expressed in this paper are those of the authors and not necessarily of IBM Canada Ltd. or IBM Corporation.

References

[1] D. E. Denning. The Sea View Security Model. In *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, California, USA, 1988.

[2] S. Jajodia, R. Sandhu. Toward a Multilevel Secure Relational Data Model. In *Proc. of ACM SIGMOD*, Denver, Colorado, USA, 1991.

[3] S. Jajodia, R. Sandhu. Polyinstantiation Integrity in Multilevel Relations. In *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, California, USA, 1988.

[4] K. Smith, M. Winslett. Entity Modeling in the MLS Relational Model. In *Proc. of the 18th VLDB Conference*, Vancouver, BC, Canada, 1992.

[5] R. Sandhu, F. Chen. The Multilevel Relational Data Model. *Transactions on Information and System Security*, Vol. 1, No. 1, 1998.

[6] N. Jukic, S. V. Vrbsky. Asserting Beliefs in MLS Relational Models. *SIGMOD Record*, Vol. 26, No. 3, 1997.

[7] N. Jukic, S. V. Vrbsky, A. Parrish, B. Dixon, B. Jukic. A Belief-Consistent Multilevel Secure Relational Data Model. *Information Systems*, Vol. 24, No. 5, 1999.

[8] Trusted Computer Security Evaluation Criteria, DoD 5200.28-STD. US Department of Defense, 1985.

[9] E. Bell, L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical Report MTR-2997, The Mitre Corporation, Burlington Road, Bedford, MA 01730, USA.

[10] M. D. Abrams, S. Jajodia, H. J. Podell. Information Security An Integrated Collection of Essays. *IEEE Computer Society Press*, Los Alamitos, CA, USA, 1995.

[11] S. Castano, et al. Database Security. *ACM Press*, New York, NY, USA, 1995.

[12] V. Atluri, S. Jajodia, T. F. Keefe, C. MaCollum, R. Mukkamal. Multilevel Secure Transaction Processing: Status and Prospects. *Database Security, X: Status and Prospects*, Chapman & Hall 1997, eds. Pierangela Samarati and Ravi Sandhu.

[13] T. F. Keefe, W. T. Tsai, T. F. Keefe, J. Srivastava. Multilevel Secure Database Concurrency Control. In *Proc. IEEE sixth International Conference on Data Engineering*, Los Angeles, CA, USA, 1990.

[14] L. Lamport. Concurrent Reading and Writing. In *Comm. ACM*, Vol. 20, No. 11, 1997.

[15] P. Ammann, S. Jajodia. A Timestamp Ordering Algorithm for Secure, Single-Version, Multilevel Databases. *Database Security, V: Status and Prospects*, C.E. Landweher, ed., Amsterdam, Holland, 1992.

[16] Oracle Corporation. Trusted Oracle Administrator's Guide. Redwood City, CA, USA, 1992.

[17] Informix. Informix OnLine/Secure Administrator's Guide. Menlo Park, CA, USA, 1993.

[18] E. Bertino, S. Jajodia, L. Mancini, I. Ray. Advanced Transaction Processing in Multilevel Secure File Stores. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 1, 1998.

[19] R. Agrawal, J. Kiernan, R. Srikant, Y. Xu. Hippocratic Databases. In *Proc. of the 28th International Conference on Very Large Databases*, Hong Kong, China, 2002.

[20] Sybase Inc. Building Applications for Secure SQL Server, Sybase Secure SQL Server Release 10.0. Emeryville, CA, USA, 1993.

[21] ISO/IEC 9075:1999. Information-Technology-Database Languages-SQL-Part 1: Framework (SQL/Framework), 1999 .

[22] Cryptek. www.cryptek.com.