ACHIEVING DATABASE SECURITY THROUGH DATA REPLICATION: THE SINTRA PROTOTYPE

Myong H. Kang, Judith N. Froscher, John McDermott, Oliver Costich, and Rodney Peyton

Naval Research Laboratory
Information Technology Division
Washington, D.C. 20375

Abstract

There are several proposed approaches for multilevel secure (MLS) database systems which protect classified information. The SINTRA[1] database system, which is currently being prototyped at the Naval Research Laboratory, is a multilevel trusted database system based on a replicated data approach. This approach uses physical separation of classified data as a protection measure. Each database contains data at a given security level and replicas of all data at lower security levels. Project goals include good performance and full database capability.

For practical reasons (e.g., ease of evaluation, portability) the SINTRA database system uses as many readily-available commercial components as possible. In this paper, security constraints and the rationale for the SINTRA prototype are described. We also present the structure and function of each component of the SINTRA prototype: the global scheduler, the query preprocessor, and the user interface. A brief description of the SINTRA recovery mechanism is also presented.

1 Introduction

As government downsizes, there is a growing need to exploit the sizable commercial investment in information technology to carry out government responsibilities more efficiently and effectively. With greater reliance on information systems, both government and commercial organizations are vulnerable to attacks on the confidentiality, integrity, and availability of information. For more than a decade the government has supported research in computer, communication, and information security to protect against such threats. The results of this research can provide the basis for managing data securely in the new information infrastructure.

In this paper, we describe the SINTRA prototype database management system. SINTRA enforces a strong protection policy and provides both good performance and the full data management capability of conventional industry-standard database management systems. The Naval Research Laboratory is developing the SINTRA prototype to demonstrate the feasibility of using physical separation and replication as the primary protection mechanisms for a database system that provides both high assurance and multilevel security.

First, we provide a brief summary of computer and database security concerns and a survey of several possible approaches to database security. Next, we present an overview of the SINTRA prototype architecture. A model of the security constraints enforced by the prototype allows consideration of how these constraints affect the transaction model and data model for the prototype. We then describe the process structure of the prototype SINTRA scheduler and a recovery strategy for this approach. The preprocessor manages the security labels as data in the conventional relational data model and ensures that proper modifications are made to user commands. Even though the SINTRA prototype is really a MultiLevel Secure (MLS) database server, a user interface has been developed that illustrates how labeled data can be shown to the user. We conclude with descriptions of the SINTRA prototype status, what we have learned, and future questions that the prototype must address.

[1] Secure INformation Through Replicated Architecture

2 Background

Organizations have long known the importance of restricting access to sensitive information to prevent competitors from learning about plans, new products, or changes in strategies. One approach for controlling access to this information is to allocate the information to sensitivity classes and restrict access based on the consequences of the information's compromise. This approach also requires that those who must access sensitive information be assigned to authorization classes commensurate with the sensitivity level of the information they are allowed to access and their trustworthiness, which is assessed through a background investigation.

When the government first used computers to process sensitive information, all users had to be cleared for access to the highest level information processed. The logical extension of this policy would have resulted in all workers being cleared for the most sensitive information. Because the government has not cleared everyone to the highest level, it has been forced to use some relatively insecure approaches to sharing information among users with different clearances.

Trusted computer products must be evaluated and tested in an adversarial manner. For high-assurance systems that must provide strong separation, the protection mechanisms must be simple and easy to evaluate. Protection critical components must contain only protection mechanisms and must mediate every access by each user. These systems are more highly engineered and crafted than most computer systems. Protection critical components must be scrutinized in an adversarial way that ensures that neither malicious code nor covert channels have been introduced into the system.

When high-assurance application systems are developed, system engineers must take great care to design systems that take advantage of the strong separation provided by these products but rely on them for little else. The commonly accepted theory for developing MLS systems is to develop untrusted applications that run at a single level but can access all data at that level and below. If the application is data intensive or requires communication across security levels, achieving both high assurance and good performance is unusually difficult. The thing to be avoided at all cost is changing the protection critical part of an already-evaluated high-assurance component. Any change invalidates the evaluation. Now we can examine the various approaches that have been taken to building a high assurance, MLS database management system.

The most straightforward approach to providing multilevel security in a database system is to design the security mechanisms into the database system itself and trust the database to enforce the security policy. In practice this results in a low assurance system, that is, the separation is weak. This kind of database system is useful, but cannot interconnect the diverse population of users expected for the new information infrastructure. The reason for the low-assurance is the complexity and size of modern database systems.

Less straightforward but more effective approaches use a reference monitor to enforce the security policy. The reference monitor is evaluated for high assurance and the database system is designed to function under the security constraints enforced by the reference monitor.

Following this reasoning, the Multilevel Data Management Security Summer Study [Air83] recommended three near-term approaches to solving the multilevel database security problem. The three approaches are: integrity lock, kernelized, and distributed. Within the latter there are two sub-approaches: replicated and non-replicated.

The integrity lock approach [Den85] uses a trusted frontend, a single untrusted backend DataBase System (DBS), and encryption techniques to protect data. A trusted frontend applies an encrypted checksum to data stored by an untrusted backend database system. The integrity lock approach is computationally intensive because checksums have to be computed whenever data are inserted or retrieved. Since the trust is in the frontend filter and the backend DBS stores data from multiple security levels, this architecture is susceptible to Trojan horse attack. Hence this approach cannot be used for highly-assured MLS DBS (e.g., a B3 or A1 system [DoD85]).

The kernelized approach [Lun90], relies on decomposing the multilevel database into single-level files which are stored separately under the control of a security kernel enforcing a mandatory access control (MAC) policy. Separate untrusted DBS are run at each security level. Decomposing multilevel relations into single-level relations so that the recomposition of the fragments is the same as the user's view of the multilevel relation has, in every case, presented many challenges. There are also problems in preserving full database functionality (e.g., transaction management, data model) while providing the required security. Materializing multilevel relations from single-level base relations requires fairly complex mechanisms and may degrade performance. In addition, the need to do this materialization has forced limitations on the data model (e.g., uniform classification of keys [Lun90]) and results in difficulties in representing *many-to-many* relationship [KCF93b]. Since this approach uses a trusted operating system to enforce separation of data at different security levels, the security of this architecture is as strong, in theory, as the security of the trusted OS.

The distributed approach comprised two architectural sub-approaches (1) each DBS has data at a single security level (non-replicated approach), and (2) each DBS contains data at a given security level and replicas of all data at lower security levels (replicated approach).

The non-replicated approach has been investigated by Jensen *et. al.* [Jen89, OcG88]. This architecture has a trusted frontend and many untrusted backends which may be commercial DBSs. Each backend contains data of only a single security level. This approach has inherent security problems because higher level queries have to be propagated to lower level untrusted backends to request data. Sending requests down to lower security levels introduces a covert channel, which can be used to transmit information from higher level backends to lower level backends. Hence, we don't expect that this approach can be used for highly-assured MLS DBS. This approach also can yield reduced performance because it may require fragments to be transferred from a low backend DBS to a high one in order to present a multilevel relation to users.

The replicated approach uses physical separation as a protection measure. A Trusted Front End (TFE) mediates access to separate Untrusted Backend DBSs (UBD) for each security class. Each backend DBS contains information at a given class and replicated information from all lower backend databases. Hence, all the information that a user can legitimately view is located at the backend corresponding to the user's authorization.

3 Description of The SINTRA System

Security, performance, and portability concerns led NRL to the initiation of a project to investigate replication using commercial DBS as a promising alternative for building a MLS DBS. Goals include good performance and full database functionality.

3.1 Practical Considerations

Security is an important issue for many organizations. However, there are other important issues, such as the reduction of production, operational, and maintenance costs and minimizing development time, effort, and evaluation costs. The ability to integrate new technology into high assurance, MLS systems in a straightforward, timely fashion makes the replicated approach very attractive to cost conscious decision makers.

One way to achieve these goals is to make maximum use of existing products which are already tested and evaluated. Those products are usually maintained by the vendors' staffs. For example, special purpose computer systems can be built by connecting general purpose commercial products together. Many interface standardization efforts (e.g., the OSI standard) make this approach more viable. Developing a new product from commercial subcomponents has several advantages:

- Minimal development and maintenance costs.
- Easy to upgrade. Once better and cheaper products are on the market, they can be easily incorporated.
- Easy testing and evaluation. Most of the subcomponents, which are commercial products, have been evaluated.
- Easy to connect. Since each subcomponent may already conform to interface standards, it is likely that the new product will be easy to interface to other products.

Based on the practical considerations above, the SINTRA prototype uses as many commercial components as possible, resulting in the need for relatively little new work to construct the overall MLS system. In the remainder of this paper, we show how the SINTRA project achieves the practical goals that we specified.

3.2 Overview

The SINTRA database system consists of one trusted front end (TFE) and several untrusted backend database systems (UBD). The role of the TFE includes authenticating users, directing user queries to the proper backend, maintaining data consistency among backends. The UBD at each security level contains data at its own security level, and replicated data from lower security levels.

The SINTRA database system prototype at the Naval Research Laboratory uses a Honeywell XTS-200 system, which is a high assurance trusted OS (B3 rated system), as a trusted frontend. Since each UBD is treated as a "black box" (i.e., inputs and outputs of the system are known, but its internal behavior is not known), each UBD can be any commercial database system. Currently, untrusted ORACLE 7 database running on SUN4/300 is used as each backend database. The backend and frontend computers are connected through dedicated Ethernet connections. Since the SINTRA security policy is enforced by the frontend, the security of this architecture is as strong as the security of the trusted frontend. Hence, the SINTRA system will be a B3 MLS DBS. Figure 1 illustrates the SINTRA architecture.

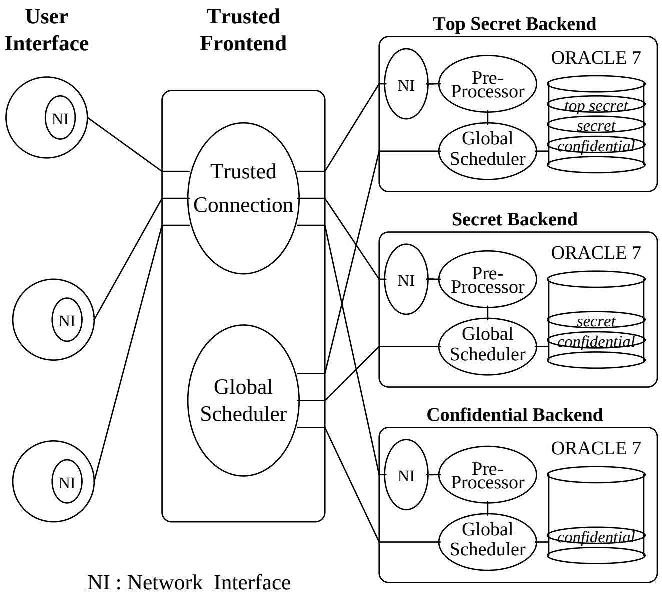

Figure 1: The SINTRA Architecture.

When a user attempts to login to the SINTRA prototype, the trusted connection checks the user's security level and establishes a connection to the network interface at the backend for that security level. Now, the user is ready to issue commands. A user query which is received at the backend will be modified by the query preprocessor, if necessary. A modified query will be submitted to the ORACLE database system through the global scheduler. The responsibilities and the detailed description of these components will appear in sections 5, 6, 7, and 8.

The SINTRA prototype is based on the following:

- All UBDs use the same database query language (e.g., SQL).
- The TFE changes the database states of the UBD only through database queries.
- Each UBD performs transaction management. (If it produces serializable and recoverable histories, SINTRA will as well.)

Note that the above assumptions enable us to treat each UBD as a black box that operates autonomously without the knowledge of other UBD's or global scheduler. Hence, no modification of the commercial DBS or the commercial frontend is required. However, we cannot expect to obtain the desired security and functionality by simply connecting commercial products. Custom processes, such as query preprocessor and global scheduler, ensure the delivery of desired database security and functionality.

The SINTRA approach has several advantages.

1. Data retrieval performance is better because the SINTRA system does not materialize the user's view from single-level relations (as do the kernelized and non-replicated distributed architectures) [2].
2. Mandatory access control is enforced through physical separation of data. Since SINTRA uses an evaluated product as a trusted frontend, very little trusted code needs to be developed to ensure that this separation is maintained.
3. Development and maintenance cost can be reduced because commercial frontend and backend database systems are widely available.
4. Performance can be improved by using optimization and parallelization techniques that have been developed for conventional databases, because the replicated architecture uses conventional database systems as UBDs. Uniprocessor or multiprocessor computers can be chosen as backend computers without affecting the security policy.
5. The system is portable and scalable because commercial untrusted systems are, in general, much more portable than trusted systems.

[2] Preliminary performance analysis indicates that SINTRA may outperform conventional database systems due to the parallel nature of the processing [McM94].

4 The SINTRA Models

In this section, we briefly discuss the security, transaction, and data models for SINTRA. These models are necessary in the development of any multilevel secure DBS and affect the decisions concerning the assignment of functions to each component of the SINTRA system.

4.1 Security Model

The security model used here is based on that of Bell and LaPadula [BeL76]. The model is stated in terms of subjects and objects. An object is a nonactive entity, such as a file, a relation, a tuple, or a field in a tuple. A subject is an active entity, such as a transaction or process, that can request access to objects. Every object is assigned a sensitivity classification, and every subject a clearance. Classifications and clearance are collectively referred to as security classes (or levels).

The database system consists of a finite set **D** of *objects* (data items) and a set **T** of *subjects* (transactions). There is a lattice **S** of security classes with ordering relation $<$. A class S_i *dominates* a class S_j if $S_i \geq S_j$. There is a *labeling function* **L** which maps objects and subjects to a security class:

$\mathbf{L}$: $\mathbf{D} \cup \mathbf{T} \rightarrow \mathbf{S}$

We consider two mandatory access control requirements:

(Simple Security Property) If transaction T_i reads data item x then $L(T_i) \geq L(x)$.

(Restricted $\star$-Property) If transaction T_j writes data item x then $L(T_j) = L(x)$.

The simple security property allows a transaction to read data items if the security level of a transaction dominates the security level of data items. The restricted $\star$-property allows a transaction to write if the security level of a transaction is the same as that of data items (i.e., no write-ups or write-downs are permitted). Write-ups (i.e., T_i cannot write to data item x if $L(T_i) < L(x)$) are undesirable in database systems for integrity reasons (i.e., since a user cannot see what he has written, he may introduce an error)[3].

4.2 Transaction Model

Traditionally, transactions are modeled as a sequence of `read` and `write` operations on data items. However, the traditional transaction model is not adequate to model transactions for the SINTRA system. Since the SINTRA treats each UBD as a black box, the global scheduler of the SINTRA system has very little knowledge about the behavior of the local scheduler (i.e., the scheduler of commercial DBS) or the physical layout of data. For example, the global scheduler has no knowledge about where a specific tuple is located or which physical page should be locked. Sometimes the tuples which will be modified are unknown until the computation based on existing data is completed.

We adopt a layered model of transactions, where a transaction is a sequence of queries, and each query can be considered as a sequence of reads and writes. For example, `replace` and `delete` queries can be viewed as a read operation followed by a write operation which must be executed atomically. `insert` can be viewed as a write operation, and `retrieve` can be viewed as a read operation. A layered view of two transactions T_1 and T_2 is shown in figure 2. Note this decomposition is similar to work of Weikum [Wei91] and Moss [Mos85].

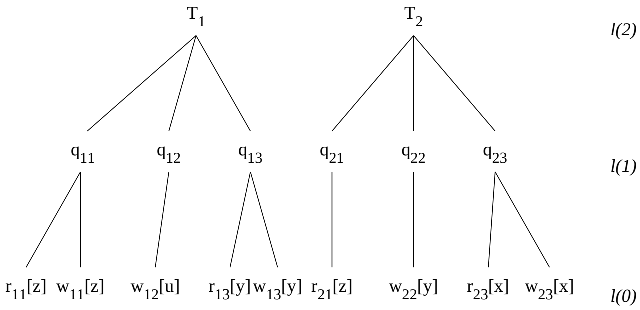

Figure 2: Layered model of two transactions.

Definition 1. A **transaction** T_i is a sequence of queries terminated by either a *commit(c_i)* or an *abort(a_i)*, i.e., $T_i = <q_{i1}, q_{i2}, ..., q_{in}, c_i>$. Each query, q_{ij}, is an atomic operation and is one of `retrieve`, `insert`, `replace`, or `delete`.

To model the propagation of updates produced by a given transaction to higher security level databases, we define an *update projection*.

Definition 2. An **update projection** U_i, corresponding to a transaction T_i, is a sequence of update queries, e.g., $U_i = <q_{i2}, q_{i5}, ..., q_{in}, c_i>$ obtained from transaction T_i by simply removing all `retrieve` queries.

[3] This is not to say that the frontend OS cannot write up to perform some functions of the overall DBS. This OS has its own compatible security model.

Note that no aborted transaction need be propagated. Hence, update projections are always terminated by a commit.

To describe concurrency control mechanisms, we adopt the following definition of *conflict*.

Definition 3. Two operations at the same layer **conflict** if they operate on the same data item and at least one of them is either `write`, `insert`, `delete`, or, `replace`. Alternatively, two operations conflict if they operate on common data and not both are `retrieve` or `read` operations.

4.3 Data Model

Because the SINTRA prototype allows no modification of the commercial UBDs, it is necessary to treat the security labels of the data simply as additional data, conventionally stored. A more abstract data model might permit a more refined representation of the semantics, but the SINTRA prototype data model is limited to representing labels as values of label attributes.

Typically, labels can be associated with either values of the individual ordinary attributes or with an entire tuple. The SINTRA data model is based on an element-level classification scheme. A multilevel relation scheme is denoted by

$R(A_1,\ C_1,\ A_2,\ C_2,\ .\ .\ .,\ A_n,\ C_n,\ TL)$

where each A_i is a data attribute over domain D_i, each C_i is a classification attribute for A_i and TL is the tuple-level attribute. The domain of C_i and TL is specified by a range $[L_i,\ H_i]$ in the security lattice. $R(A_1,\ A_2,\ .\ .\ .,\ A_n,\ TL)$ is the underlying relation as viewed by the user.

Let $t[A_i]$ denote the value corresponding to the attribute A_i in tuple t, and similarly for $t[C_i]$ and $t[TL]$. $t[TL]$ in the SINTRA system simply specifies that a tuple t is generated or modified by a $t[TL]$-level user. On the other hand, $t[C_i]$ specifies that the corresponding $t[A_i]$ originated from $t[C_i]$-level. Operations on the database that allow retrievals and updates are presented in [KCF93a]. There are, in addition, constraints among the values of the C_i and TL [KCF93a] which are of little importance for explaining the prototype.

5 The SINTRA Scheduler

The replicated architecture provides mandatory access control by physically separating data. However, this approach introduces another problem, namely the maintenance of mutual consistency of the replicas. Our research suggests that accepting the replica consistency problem in return for a virtually free high assurance, strong protection mechanism is a choice well-made. In this section, we introduce the structure and functions of the global scheduler. True distribution of these functions is, in the abstract, of little consequence, but has been done in the prototype in a particular way to maximize performance.

5.1 Responsibilities

Since each UBD in a replicated architecture contains data from lower levels, update queries have to be propagated to higher-security-level databases to maintain the consistency and currency of the replicated data. If this propagation of update queries is not carefully controlled, inconsistent database states can be created among backend databases. Consider that two lower level update transactions T_i and T_j are scheduled with serialization order $<T_i,\ T_j>$. at the lower level backend database system. Since these two transactions are update transactions, they have to be propagated to the next higher level. If these two transactions are scheduled with serialization order $<T_j,\ T_i>$ at the next higher level, an inconsistent database state between these two backend databases may be created by the execution of conflicting operations at the higher level. It can be demonstrated that even the serialization order of non-conflicting transactions has to be maintained to preserve one-copy serializability [KFC92]. This is a parallel result that has been reported in the context of multidatabase systems [Du93]. Therefore, the serialization order introduced by the local scheduler at the user's session level must be maintained at the higher level UBDs.

The concurrency control algorithm which the SINTRA prototype uses has two types of schedulers, global and local schedulers. The global scheduler enforces data consistency among different security levels. On the other hand, the local scheduler, which is the unmodified commercial concurrency controller of a UBD, manages transactions and update projections at that UBD. The local scheduler deals with layer $l(0)$ in figure 2, and the global scheduler deals with layer $l(1)$ and upper layers. The global scheduler detects conflicts at level $l(1)$. Therefore, no knowledge of the specific items to be accessed or even the granularity of the lower-level concurrency controller is needed or used by the global scheduler.

SINTRA's global scheduler resolves the datainconsistency problem by guaranteeing that the serialization order introduced by the local scheduler at the user's session level is maintained at the higher level UBDs. In summary, the global scheduler performs the following tasks:

- Receive queries from the query preprocessor and the global scheduler of lower security levels, and send them to the backend database.

- Guarantee that the serialization order introduced by the local scheduler at the user's session level is maintained at the higher level UBDs.

- When a transaction is committed, send an update projection to higher security level backends.

Since user transactions and update projections are submitted independently, their serialization orders are not known to the global scheduler. Hence, the *take-a-ticket*[Geo91] operation is used to find the serialization order among update projections and user transactions. Generalized algorithms and a theory of a global scheduler for the SINTRA database system have been presented by Kang, Froscher, and Costich in [KFC92].

As previously mentioned, the global scheduler resides partially in the TFE and partially in the UBD. This was done for performance rather than theoretical reasons.

5.2 Structure

The methodology used for developing the SINTRA global scheduler closely resembles the object-oriented development method [Boo86]. We identify many objects, queries, transactions, processes, etc., and establish the relationships among these objects. Many layers are also introduced to hide lower-level details. C++ has been chosen as our main implementation language because it provides the capabilities of data hiding and abstraction of interface. Scheduler components which are executed on the frontend are written in C because the XTS-200 provides neither a C++ interpreter nor the C compiler which can compile C code that is generated by a C++ interpreter.

The process architecture of the global scheduler is as follows (see also figure 3):

1. **Terminal:** User terminals or workstations.
2. **Frontend Connector:** Establish a virtual connection between the user and the backend depending on the user's session level.
3. **Database Server:** When user login is requested, spawn a database server child to service the request.

3a. **Database Server Child:** Ensure connections among user, preprocessor, and user transaction scheduler.

4. **Preprocessor:** Query modification (see section 7).
5. **User Transaction Scheduler:** Submit user transactions to the ORACLE database. Send the response from ORACLE to the user. Also send an update projection to the propagation scheduler if a user transaction is committed.
6. **Propagation Scheduler:** Receive transactions and send then to the corresponding frontend update projection receiver according to the serialization order.
7. **Projection Receiver:** Receive update projections from the propagation scheduler and store them for retrieval by the projection sender.
8. **Projection Sender:** Read-down to get the update projections which are in the lower level projection receiver and send the projections to the update projection scheduler.
9. **Update Projection Scheduler:** Receive update projections from the lower level backend, submit them to the ORACLE database, and send them to the propagation scheduler.

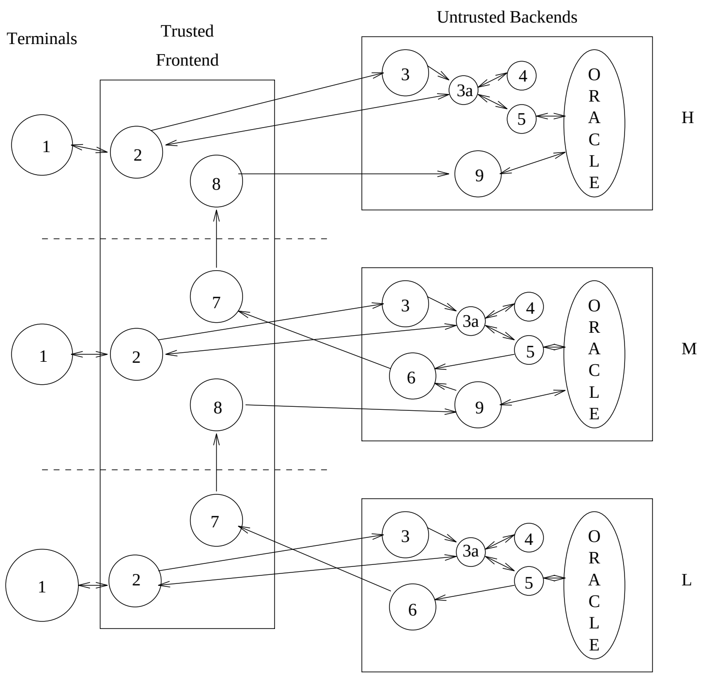

Figure 3: The Process-level Architecture.

Our process architecture has been implemented using C++ objects, and complex interprocess communication details have been hidden by an interprocess communication (ipc) class. A detailed description of each object and methods is given by Kang and Peyton in [KaP93a].

6 Recovery Mechanisms

Recovery in the SINTRA MLS DBS is largely dependent on the recovery mechanisms of the commercial DBSs that are used at the backends. Without the propagation of update projections, SINTRA appears to be a single-level DBS. Hence there is no need for recovery mechanisms between users and the UBD. However, the SINTRA database system needs recovery mechanisms that guarantee the delivery of update projections from user transaction scheduler to the next level DBS (i.e., path $5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow$ DBS in figure 3).

To ensure that committed transactions will not be lost (except by media failure), processes responsible for the propagation of update projections use persistent queues to log update projections. Update projections are kept in the log until the next process within the propagation chain acknowledges that it has received the update projection. An acknowledgment from a receiver guarantees the sender that the receiver has safely processed the update projection.

The above technique works without creating a covert channel if the sender and receiver are at the same security level. However, if the same technique is used when the security level of the receiver is higher than that of the sender, then there is a covert channel. Alternatively, when a message is delivered to a receiver, it sends either an *ack* or control back to the sender acknowledging that the message is in stable log. However, if the security level of the sender is lower than that of the receiver, then the timing of an *ack* can be used as a covert timing channel. A detailed description of the problem and proposed solutions are presented by Kang and Moskowitz [KaI93].

When the system recovers from a failure, it needs to know the status of DBS (i.e., the last transaction that has been committed) so that the system can guarantee the consistent states among DBS and persistent queues in the global scheduler. The SINTRA recovery mechanism uses the *ticket* which was introduced in section 5.1 to examine the status of the DBSs. When the system recovers, the processes of the global scheduler examine the status of the DBS and their persistent queues (i.e., sort out which transactions are committed and which ones are not). Then the update projection scheduler will submit update projections which have not been committed to the DBS and the propagation scheduler will send committed transactions to the projection receiver. A detailed design description and high-level pseudo-code are presented in [KaP93c].

7 The Query Preprocessor

In this section, we explain the need to modify user queries and the internal structure of the query preprocessor. The SINTRA query preprocessor is written in C++. YACC++ and LEX++ are also used to build the query parser.

7.1 Responsibilities

The SINTRA query preprocessor plays an important role in maintaining data consistency among different backend databases, preserving data integrity, and bridging the semantic gap between conventional and multilevel-secure databases. The SINTRA query preprocessor has the following responsibilities:

1. In the SINTRA database system, if a high-level user is allowed to modify low data which are located at the high-level backend database, then inconsistent database states between high and low backend databases can be created. Therefore, the query preprocessor must inspect, and either reject or modify users' update queries so that the backend database system only modifies users' login level data — it is also assumed that no *write-up* is allowed.

2. There is also some information which can be modified only by the system although it can be disclosed to the user. For instance, information about the classification of a tuple (TL attribute values) cannot be modified by the user. It is the responsibility of the query preprocessor to guarantee the integrity of such data.

3. SINTRA uses conventional relational database systems as backend databases. These conventional relational databases use SQL, which is based on the conventional (single-level) relational algebra and the semantics of conventional update operations [Ull82]. On the other hand, a multilevel relational database is based on a multilevel relational algebra and the semantics of multilevel relational update operations. Therefore, a

SINTRA user query which is posed to an MLS database in multilevel SQL must be translated into other queries (based on the conventional SQL used by the UBD) that conform to the semantics of conventional update operations.

To perform the above responsibilities, the SINTRA query preprocessor intercepts, inspects and modifies user queries before they are submitted to the ORACLE database system.

7.2 Structure

Consider the following user query, `delete from R where` A_1 `= 5 and TL = 'L'`. It is a legitimate query if an L-user (the user whose session level is L) issued the query. However, if an H-user issues the same query then the query must be blocked (i.e., otherwise this query creates an inconsistent copy of L-data at the H-backend). Therefore, when the SINTRA preprocessor makes decisions on how a user query should be modified and what kind of query has to be blocked, it has to be based not only on the syntax of the individual query but also on the session level of the user who issues the query. Hence, simple syntactic checking is not sufficient to determine if a query is legitimate or not. We chose the following process organization:

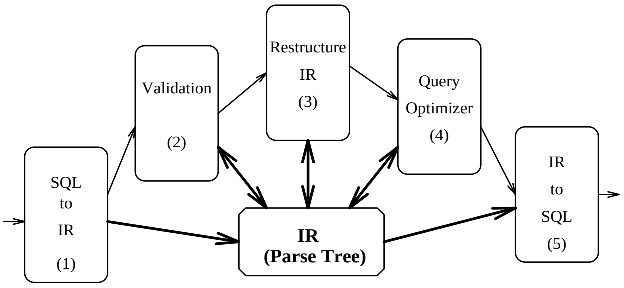

Figure 4: The Structure of the Query Preprocessor.

First, a user query is parsed and converted into an internal representation (IR) which is a parse tree. All syntactically invalid queries and some invalid queries which can be detected without knowing the security level of the user will be rejected at this stage. For example, a user query `update R set a = 5, a# = 'H' . . .`[4] will be rejected at this stage, because the query preprocessor knows that `a#` is a classification attribute, and users are not allowed to modify classification attributes.

[4] The SINTRA preprocessor needs to distinguish regular attributes from classification attributes. Hence, the SINTRA preprocessor reserves one character (e.g., '`#`') for preprocessor use only. For example, a user defines an attributes a in a specific relation, then its corresponding classification attribute will be `a#`.

The second process, validation, of figure 4 inspects all syntactically valid queries again to check if they can create any inconsistent replicas. For example, `delete from R where a = 5 and T#L = 'L'` by a H-user will be rejected at this stage because the H-user tries to delete L-data. Hence, responsibilities (1) and (2) in section 7.1 are accomplished by the first and second stages.

The third process, restructure IR, performs the main mission of the SINTRA query preprocessor, i.e., query modification. It will modify parse trees based on the SINTRA multilevel relational algebra and the semantics of update operations for multilevel relations [KCF93a]. Consequently, functions (3), (4), and (5) in section 7.1 are accomplished at this stage. For example, a H-user's query, `delete from R where b = 'xxx'`, will be translated into `delete from R where b = 'xxx' and TL = 'H'` . A detailed design description of the SINTRA query preprocessor appears in [KaP93b].

The fourth process, query optimizer, optimizes parse trees based on the knowledge of the implementation. Finally the fifth process, IR to SQL, converts parse trees into conventional SQL before the query is submitted to ORACLE at the UBD.

8 The User Interface

The SINTRA system uses a client-server architecture for its user interface. The user interface resides on the client side and the DBS resides on the server side. Network interface units reside in both clients and servers (see figures 1 and 5).

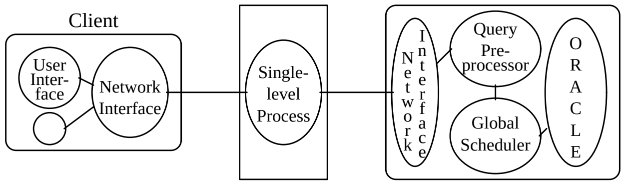

Figure 5: A Detailed Description of Network Interface.

The network interface process is responsible for directing queries from the user interface to the proper server, and directing responses from the server to the proper client. The single-level process in TFE, which resides between clients and servers, assures that the communication between two network interfaces does not violate the SINTRA security policy.

Note that each client is dedicated to one security level, and identification and authentication procedures are still performed by the TFE with the current configuration. However, this restriction (i.e., a client can login only one security level) can be removed.

9 Status and Conclusions

The prototype has demonstrated the feasibility of replication as a path to high-assurance security, good performance, reduced maintenance cost, and portability. This architecture uses physical separation as a protection measure. Alternatively, the mandatory access control of the SINTRA approach depends on the MAC of the TFE and the physical separation of UBDs, and the discretionary access control (DAC) of the SINTRA approach depends on the DAC of UBD.

Since a system based on this architecture can be built from commercial DBSs, it is very portable and maintenance is relatively simple. High performance is achieved by storing all information that a user can legitimately view at one backend.

9.1 Status of Prototype

The current status of the SINTRA implementation is as follows:

- The design and implementation of the global scheduler are both finished.
- The design and implementation of the query preprocessor are both finished (i.e., a subset of SQL and full operations that are discussed in [KCF93a] have been implemented).
- The design of the user interface is finished. There are many commercial user interface tools for the ORACLE database. We hope to use those tools for the SINTRA. However, almost all user interface tools for ORACLE use *SQL*Net* for communication between the interface tool itself and ORACLE. Since the SINTRA preprocessor needs to intercept, inspect and modify user queries before these are submitted to ORACLE[5], the SINTRA preprocessor needs to know the internal format of packets that *SQL*Net* uses. Until we have that information, we will proceed to implement our own user interface.
- The design of the recovery mechanism is finished and the implementation of the mechanism is in progress.

9.2 Lessons Learned

This prototyping exercise has demonstrated the feasibility of the replicated architecture approach first described in [Air83]. Prior to the SINTRA project, none of the high assurance approaches described in [Air83] had been demonstrated. We have learned several important lessons from our development of the SINTRA prototype.

- The strong separation of concerns inherent in the replicated architecture allows the use of commercial DBSs without modification. This means that the replicated approach can accommodate new database technology without significant porting effort or reengineering.
- Several replica and concurrency control algorithms have been developed for the project and have been proven correct. The implementation of the SINTRA global scheduler moved our results from theory to practice and showed that the replicas remained consistent when the database was updated.
- The trusted OS for the TFE required no modification. Little additional software was developed to run on the TFE. We were successful in minimizing our dependence on the TFE.
- The replicated architecture approach imposes fewer data model restrictions than kernelized approaches.
- Preliminary performance analysis and simulation results indicate that the replicated architecture provides good performance.
- This approach does require more hardware than the kernelized approach. As hardware costs become lower, it will be a trade-off decision for users to decide whether improved performance and usability are worth the additional hardware cost.

We hope that the SINTRA prototype will serve as the catalyst for future high-assurance multilevel

[5] The query decomposer of the KSR1 computer performs the same task to parallelize the queries.

database research. Our plan is to use B1 DBSs running on compartmented mode workstations as backend databases. This configuration will provide the B1 separation among compartments, and the B3 separation among security hierarchies.

References

[Air83] Multilevel Data Management Security, Air Force Studies Board, Commission on Engineering and Technical Systems, National Research Council, National Academy Press, Washington, D.C. (1983).

[BeL76] Bell, D. E., and LaPadula, L. J. Secure computer systems: Unified exposition and *multics* interpretation. The Mitre Corp, (1976).

[Boo86] Booch, G. Object-Oriented development. IEEE Transactions on Software Engineering, 12, 2 (1986).

[DoD85] Department of Defense National Computer Security Center, *Trusted computer system evaluation criteria*, DoD5200.28-STD (1985).

[Du93] Du, W., Elmagamid, A. K., Kim, W., and Bukhres, O. Supporting consistent updates in replicated multidatabase systems. The VLDB Journal, 2, 2 (1993).

[Den85] Denning, D. Commutative filters for reducing inference threats in multilevel database systems. The IEEE symposium on Security and Privacy (1985).

[Geo91] Georgakopoulos, D., et al. On serializability of multidatabase transactions through forced local conflicts. Conference on Data Engineering (1991).

[Jen89] Jensen, C., et al. SDDM: A prototype of a distributed architecture for database security. Conference on Data Engineering (1989).

[KFC92] Kang, M. H., Froscher, J. N., and Costich, O. A practical transaction model and untrusted transaction manager for multilevel-secure database systems. The Eighth IFIP Workshop on Database Security (1992).

[KCF93a] Kang, M. H., Costich, O., and Froscher, J. N. The replicated architecture data model: structure and operation. Internal Report (1993).

[KCF93b] Kang, M. H., Costich, O., and Froscher, J. N. Using object modeling techniques to design MLS data models. Security in Object-Oriented Systems, Springer-Verlag, ISBN 3540198776 (1994).

[KaI93] Kang, M. H., and Ira S. Moskowitz. A pump for rapid, reliable, secure communication. ACM Conference on Computer and Communications Security (1993).

[KaP93a] Kang, M. H., and Peyton, R. Design documentation for the SINTRA global scheduler. Naval Research Laboratory Memo Report 5542-93-7362 (1993).

[KaP93b] Kang, M. H., and Peyton, R. Design documentation for the SINTRA preprocessor. Internal Report (1993).

[KaP93c] Kang, M. H., and Peyton, R. Design documentation for the SINTRA recovery mechanism. Internal Report (1993).

[Lun90] Lunt, T., et al. The SeaView security model. IEEE Transactions on Software Engineering, 16, 6 (1990).

[McM94] McDermott, J. and Mukkamala, R. Performance analysis of transaction management algorithms for the SINTRA replicated-architecture database system. In Database Security, VII: Status and Prospects, eds. Thomas Keefe and Carl Landwehr, North-Holland (1994).

[Mos85] Moss, E. Nested transactions, an approach to reliable distributed computing. The MIT Press (1985).

[OcG89] O'Connor, J., and Gray, J. A distributed architecture for multilevel database security. National Computer Security Conference (1988).

[Wei91] Weikum, G. Principles and realization strategies of multilevel transaction management. ACM Transactions on Database Systems, 16, 1 (1991)