Multilevel Security Issues in Distributed Database Management Systems—III

Bhavani Thuraisingham and Harvey H. Rubinovitz

The MITRE Corporation, Burlington Road, Bedford, MA 01730, USA

In this paper we describe multilevel security issues for a distributed database management system which operates in a limited heterogeneous environment. We first describe an architecture for such a system and then discuss techniques for query processing and transaction management.

Keywords: Multilevel secure distributed database management systems, Heterogeneity, Accreditation ranges, Query processing, Transaction management.

1. Introduction

This is the third in a series of articles that we have published in this journal on multilevel security for distributed database management systems (DDBMS)*. In the first article [3], we described our preliminary investigation of multilevel security issues for a DDBMS based on the front-end/back-end architecture. Such an architecture was chosen simply because it was the one example of a distributed architecture considered in the Air Force Summer Study Report [5]. In this architecture, a front-end machine is connected to one or more back-end database systems. All requests to the database systems are via the front-end machine. That is, the front-end machine controls the execution of all transactions. As a result the back-end machines cannot execute local applications. This feature is not strictly in accordance with the standard definition of a DDBMS given in ref. 1.

In contrast with ref. 3, in the second article [7], we described security issues for a DDBMS based on the architecture given in ref. 1. In that architecture, the nodes are connected via a communication subsystem†. This subsystem could be any network such as local area network or a long haul network.

*A centralized DBMS which provides multilevel security is called a multilevel secure DBMS (MLS/DBMS). In such a DBMS, used cleared at different security levels access and share a database with data at different sensitivity levels without violating security. A DDBMS which provides multilevel security is called a multilevel secure DDBMS (MLS/DDBMS). We assume that the set of security levels forms a lattice with Unclassified < Confidential < Secret < Top Secret. For more details on the definitions, we refer to our earlier papers (see, for example, refs. 3 and 7.

†We will use the terms node and site interchangeably.
Each node has its own local DBMS which is capable of handling the local applications. In addition, each node is also involved in at least one global application. In other words, there is no centralized control in this architecture. In the architecture for an MLS/DDBMS described in ref. 7, each node has an MLS/DBMS which is augmented by a module called the secure distributed processor (SDP). The SDP consists of components for query processing, transaction management, metadata management, and constraint processing. The MLS/DBMS as well as the SDP are hosted on a trusted computing base (TCB) which enforces the basic mandatory security policy. The nodes are connected via a trusted network. We assumed that all of the MLS/DBMSs are designed to operate identically. That is, the environment is a homogeneous one.

In the third article, described in this paper, we extend our investigation of multilevel security issues for a DDBMS reported in ref. 7 by considering a limited heterogeneous environment. Since heterogeneity brings about an additional complexity not present in a homogeneous environment, the only type of heterogeneity that we consider is one in which not all nodes handle the same accreditation ranges*. We describe two architectures for such an MLS/DDBMS and select one for further consideration. Then we describe issues on query processing and transaction management for the architecture considered.

The organization of this paper is as follows. In Section 2, we describe two architectures for an MLS/DDBMS to handle different accreditation ranges and select one for further consideration. In Section 3, we describe issues on query processing. In Section 4 we describe issues on transaction processing. Additional issues that must be investigated for the secure interoperability of different MLS/DBMSs are listed in Section 5. The paper is concluded in Section 6.

2. Architectures for Handling Different Accreditation Ranges

We describe two architectures for an MLS/DDBMS which handles different accreditation ranges. One is called the replicated architecture and the other is the nonreplicated architecture. We will compare the two architectures and discuss the advantages and disadvantages of each one. Finally we will select one for further consideration.

2.1. Replicated Architecture

Figure 1 illustrates the replicated architecture for an MLS/DDBMS which handles different accreditation ranges. It is assumed that each node has an MLS/DBMS and a secure distributed processor (SDP) hosted on a trusted computing base (TCB). The TCB ensures that subjects read objects at or below their level and subjects write into objects at their level. The SDP includes the distributed query processor (DQP), which is responsible for distributed query processing, and the distributed transaction manager (DTM), which is responsible for distributed transaction management. The nodes are connected via a trusted network. The network ensures that there is a two-way communication between processes at different nodes operating at the same security level. If the processes are operating at different security levels, then communication must be via a trusted process.

In the example of Fig. 1, The MLS/DBMS at site 1 handles the range from Confidential to Top Secret while the MLS/DBMS at site 2 handles the range Confidentia

*Note that there is no security violation if a lower level process sends messages directly to a higher level process.
from Unclassified to Secret. It is assumed that the multilevel database at site 2 is replicated at site 1. In this way, a Top Secret user's query at site 1 does not have to be routed to the MLS/DBMS at site 2 which does not handle the Top Secret level. Note that since site 1 does not handle the Unclassified level, the Unclassified data at site 2 is replicated at the Confidential level at site 1. The replicated copies must be kept consistent during the update operation.

Replication could be either total or partial. In the case of total replication, it is assumed that if a piece of data is classified at level L, then it is replicated at all nodes which are accredited to handle the level L or higher. That is, if some data are classified at the Secret level, then those data are replicated at each node which handles Secret and/or Top Secret data. In the case of partial replication, for each security level L, and each piece of data item D classified at level L* < L, it must be ensured that D is stored at least at one node N which handles the level L. In other words, if node 1 handles only the Unclassified and Confidential levels and nodes 2 and 3 handle only the Secret and Top Secret levels, then it is sufficient to replicate the Unclassified and Confidential data at node 1 at either node 2 or 3.

We assume that metadata classified at level L are replicated at each node which is accredited to handle any level which dominates the level L. When a user poses a query at level L, the query is routed to the relevant nodes which are accredited to handle the level L. If the data are replicated at more than one node, then the query optimizer should take data replication into account when optimizing the query. When a user issues an update request, then it must be ensured that the replicated copies are kept consistent. For example, if an Unclassified user at site 2 requests Unclassified data to be updated, then the replicated copy at site 1 must be updated also.

†We assume that the set of security levels forms a lattice with Unclassified < Confidential < Secret < Top Secret.
2.2. Nonreplicated Architecture

Figure 2 illustrates the nonreplicated architecture for an MLS/DBMS which handles different accreditation ranges. In this example, the MLS/DBMS at site 1 handles the range from Confidential to Top Secret while the MLS/DBMS at site 2 handles the range from Unclassified to Secret. As in the case of the replicated architecture, it is assumed that each node has an MLS/DBMS and a SDP hosted on a TCB. The TCB ensures that subjects read objects at or below their level and subjects write into objects at their level. The modules of SDP include the DQP and the DTM. The nodes are connected via a trusted network. The network ensures that there is a two-way communication between processes at different nodes operating at the same security level. If the processes are operating at different security levels, then communication must be via a trusted process.

In the example of Fig. 2, the MLS/DBMS at site 1 handles the range from Confidential to Top Secret while the MLS/DBMS at site 2 handles the range from Unclassified to Secret. Unlike the replicated architecture, it is assumed that the multilevel database at site 2 is not replicated at site 1. This means that, when a Top Secret user queries the system at site 1, the query has to be routed to site 2 which does not handle the Top Secret level. Therefore, the query must be routed via a trusted process which must ensure that it does not contain any Top Secret information. Since there is no replication, update processing is less complex with this architecture. The assumptions that we have made about metadata distribution for the replicated architecture apply for the nonreplicated architecture also.

2.3. Comparison of the Two Architectures

An advantage with the replicated architecture is that a user's request does not have to be transmitted to a node which does not handle his level. That is, a Top Secret user's request does not have to be sent to a node which handles the range from Unclassified to Secret. Since the Unclassified, Confidential, and Secret data in this node are replicated at a node which handles the Top Secret level, the user's request can be processed by the second node.

There are some disadvantages associated with the replicated architecture. First of all, the replicated copies of the data must be kept consistent. Main-
Computers and Security, Vol. 11, No. 7

Taming consistency would introduce covert channels because either success or failure of an update request must be acknowledged from a higher level process at a node to a lower level process at a different node. For example, consider the architecture illustrated in Fig. 1. Suppose an Unclassified user requests to update some Unclassified data at site 2. Since Unclassified data at site 2 are replicated at the Confidential level at site 1, a Trojan horse operating at the Confidential level at site 1 could abort the update request on the replicated data. If consistency is to be maintained, then the update request on the Unclassified data at site 2 must also be aborted. The Confidential Trojan horse could encode Confidential data by alternating between accepting and aborting the update requests on the replicated data. An Unclassified Trojan horse at site 2 could observe the actions of the Confidential Trojan horse and subsequently obtain the Confidential data.

Second, in a heterogeneous environment, it is usually assumed that the nodes are autonomous. Each node may choose to be part of a federation whenever it wants to. That is, the level of autonomy depends on the database administrator (DBA)/system security officer (SSO). For example, the administrator of a node may not want his data to be replicated at another node. This is because he may want complete control of his data. Therefore data replication and autonomy are conflicting requirements and heterogeneity usually implies autonomy. Third, the DBA/SSO has to perform a lot of work in order to determine how the data should be replicated. Random replication of data has been shown to have an impact on the performance of the query processing algorithms [1]. Finally, replication implies additional storage space. This is yet another disadvantage of this architecture.

There are several advantages with the nonreplicated architecture. First, since data are not usually replicated, maintaining consistency is not an issue during update processing. Second, each local DBA/SSO has complete control of his own data and may choose to export only the data that he chooses. Third, storage space can be saved. Finally, the DBA/SSO does not have to be concerned about the optimum ways of replicating the data.

The main disadvantage with the nonreplicated architecture is that the higher level queries at a site are routed to lower level processes at different sites. This operation would introduce covert channels. For example, consider the nonreplicated architecture illustrated in Fig. 2. A Trojan horse operating at the Top Secret level at site 1 could modulate the arrival times of the queries to encode Top Secret data. Suppose these queries are sent to site 2 to retrieve lower level data. Although the queries are routed to site 2 via a trusted process, this process only ensures that the queries do not have any sensitive information. A Trojan horse operating at the Secret level at site 2 could measure the interarrival times between consecutive Top Secret queries and consequently obtain the Top Secret data.

Since data replication is not desirable for heterogeneous and especially an autonomous environment, we propose the nonreplicated architecture for a MLS/DDBMS handling different accreditation ranges. In Section 3 we discuss query processing for a MLS/DDBMS which is based on the nonreplicated architecture. Issues on transaction management are discussed in Section 4.*

*Note that since the nodes handle different accreditation ranges, there is a potential for the cascading problem [4]. For example, suppose node 1 handles the range Secret to Top Secret, node 2 handles Confidential to Secret, and node 3 handles Unclassified to Confidential. Suppose Top Secret data at node 1 is covertly passed to a Secret process at node 1. This process could legitimately pass the data to a Secret process at node 2. At node 2, this data could be covertly passed to a confidential process. This confidential process could pass the data to a Confidential process at node 3. At node 3, the data could be covertly passed to an Unclassified process. This means that Top Secret data at node 1 has now been passed to an Unclassified process at node 3. We have not addressed the cascading problem in this paper.

665
3. Query Processing

In this section, we describe secure distributed query processing techniques to function in a limited heterogeneous environment. The particular type of heterogeneity that we address is the case where the nodes handle different accreditation ranges. In Section 3.1, we describe the process architecture of the DQP and discuss its security critical components. Query processing examples are given in Section 3.2.

3.1. Architecture of the DQP

The process architecture of the DQP is shown in Fig. 3. The components of the DQP are the request user interface manager (Request-UIM), response user interface manager (Response-UIM), query transformer (QT), query optimizer (QO), distributed execution monitor (DEM), the query trusted process (QTP). The Request-UIM accepts a user's request and parses the request. The parsed request is given to the QT to decompose the request on a relation into requests on the various fragments. The decomposed request is given to the QO which generates an optimized execution strategy. The execution strategy is given to the DEM to carry out the execution. The DEMs at the various nodes communicate with each other via the trusted network in order to carry out the execution strategy. Since the various nodes handle different accreditation ranges, the query must be examined by the QTP in a manner to be described below. We assume that the network interface manager (NIM) which connects the DQP to the network is part of the trusted network. Finally, the DEM at the user's site delivers the response generated to the user via the Response-UIM.

As shown in Fig. 3, QTP is between the DEM and the trusted network. QTP is a trusted process which must ensure that any sensitive portion of the query must be removed before it is transmitted to a remote DEM. For example, consider the query "Select all from EMP where Salary is greater than 50K" posed by a Top Secret user at site 1. Let us also assume that site 1 handles the range from Confidential to Top Secret while site 2 is accredited to handle the range Unclassified to

---

Fig. 3. Architecture of the DQP.
Secret. If the query has to be sent to site 2, the DEM at site 1 must remove the qualification clause from the query as it might contain potentially sensitive information. Note that the DEM is an untrusted process and therefore cannot be guaranteed to carry out its actions correctly. However, if it attempts to send any request to the DEM at site 2, the trusted network will reject it as site 2 is not accredited to handle the Top Secret level. In other words, the NIM must be trusted so as to ensure that a Top Secret DEM does not communicate with a node which is not accredited to handle the Top Secret level. Therefore, the DEM at site 1 must send the query first to the QTP at site 1. QTP will check whether the qualification clause has been removed from the query, the string EMP is classified at most at the Secret level, and pass the request to the network via the NIM. The network then gives the request to the DEM at site 2 which operates at the Secret level, and pass the request to the network via the NIM. The network then gives the request to the DEM at site 2 which operates at the Secret level. The modified request will be to “select all from EMP”. The Secret DEM at site 2 will process the request and send the result to the Top Secret DEM at site 1 via the QTP. The DEM at site 1 will then select those tuples from EMP where the salary is greater than 50K.

Note that in order to enforce security the QTP must be trusted. However, if the remaining modules of the DQP are not trusted, then label and data integrity cannot be maintained*. In order to maintain data integrity, the entire DQP must be trusted. If only the integrity of the labels is to be preserved, then the DEM and the Response-UIM must be trusted. Note that if the DEM is to be trusted, then the functions of the QTP could be incorporated into those of the DEM. That is, a separate module for QTP is not needed.

Another option is to keep the DEM and Response-UIM as untrusted, but to introduce two trusted modules called the checksum computer (CC) and checksum analyzer (CA). This is illustrated in Fig.

*In ref. 7 we assumed that the DQP was untrusted. That is, maintaining the integrity of the labels was not addressed in our earlier paper.

4. CC lies between the response interface of the local MLS/DBMS and the DEM. It computes a cryptographic checksum based on the value of the tuple and the label generated by the local MLS/DBMS. The checksum is attached to the tuple and its label. CA lies between the user and the Response UIM. Before the tuple is given to the user, CA will recompute the checksum of the tuple and its label using the same algorithm as the one used by the CC. The tuple is given to the user only if the new checksum is equal to the old checksum. Although there is some overhead involved in computing the checksum, store the checksums with the tuples, and also transmit the checksums along with the tuples it seems that the amount of trusted code necessary with this approach is less than the case of trusting the DEM and the Response-UIM. However, additional problems such as key management are introduced with this approach. The strength of the checksum algorithm is also another major issue with this approach.

Since most of the MLS/DBMSs that are being designed do not ensure label and data integrity†,

†See for example, the designs of Seaview [2] and Lock Data Views [6].
we assume that the only trusted module of the DQP is the QTP. The architecture of the DQP that we have considered is illustrated in Fig. 3.

3.2. Examples
We illustrate query processing with some examples. Let node 1 handle the range Confidential to Top Secret and node 2 handle the range Unclassified to Secret. Let the distributed database consist of relations EMP and DEPT. The attributes of EMP are SS#, Name, Salary, and D#, with SS# as the key. The attributes of DEPT are D#, Dname, and Mgr, with D# as the key. The database at node 1 consists of Confidential, Secret, and Top Secret employee fragments, EMP1-C, EMP1-S, and EMP1-TS, respectively, and Confidential, Secret, and Top Secret department fragments, DEPT1-C, DEPT1-S, and DEPT1-TS, respectively. The database at node 2 consists of Unclassified, Confidential, and Secret employee fragments, EMP2-U, EMP2-C, and EMP2-S, respectively, and Unclassified, Confidential, and Secret department fragments, DEPT2-U, DEPT2-C, and DEPT2-S, respectively.

**Example 1.** Top secret user at node 1 requests to retrieve all information on employees. This query is represented by “Select all from EMP”.

A simple execution strategy generated by the QO at node 1 (which operates at the Top Secret level) will be the following:

1. Transfer EMP2-U, EMP2-C, EMP2-S from node 2 to node 1.

The first request is sent to the DEM. The DEM at the Top Secret level will execute the second request. That is, the six EMP fragments will be merged and the result will be given to the Secret process at node 2. More research needs to be done as to whether appropriate auditing techniques could be used to detect such covert information flow.

**Example 2.** Top Secret user at node 1 requests to retrieve all employee names where salary is greater than 50K. This query is represented by “Select Name from EMP where Salary > 50K”.

A simple execution strategy generated by the QO at node 1 (which operates at the Top Secret level) will be the following:

1. Select Name from EMP2-U, EMP2-C, EMP2-S at node 2 where Salary > 50K and send result to node 1.
2. Select Name from EMP1-C, EMP1-S, EMP1-TS at node 1 where Salary > 50K.
3. Merge all the names retrieved (both from nodes 1 and 2).

The DEM at node 1 (operating at the Top Secret level) will execute the second request. That is, the six EMP fragments will be merged and the result will be given to the Top Secret user.

Note that although there is no Top Secret information that has directly been passed to node 2, a Trojan horse in the DEM at node 1 could issue a series of valid requests and pass information covertly to node 2. It is not possible for the QTP to detect this type of information flow as all of the fragments specified in the request are legitimately known to the Secret process at node 2. More research needs to be done as to whether appropriate auditing techniques could be used to detect such covert information flow.

**Example 3.** Top Secret user at node 1 requests to retrieve all employee names where salary is greater than 50K. This query is represented by “Select Name from EMP where Salary > 50K”.

A simple execution strategy generated by the QO at node 1 (which operates at the Top Secret level) will be the following:

1. Select Name from EMP2-U, EMP2-C, EMP2-S at node 2 where Salary > 50K and send result to node 1.
2. Select Name from EMP1-C, EMP1-S, EMP1-TS at node 1 where Salary > 50K.
3. Merge all the names retrieved (both from nodes 1 and 2).

The first request is given to the DEM at node 1. The DEM removes the qualification portion of the query (which is “where Salary > 50K”) and gives the remaining request to the QTP. The modified request is “retrieve Name and Salary from fragments EMP2-U, EMP2-C, EMP2-S”. The QTP
will examine the request and determine that it does not have any sensitive information. Note that, if the DEM did not remove the where clause, then QTP will determine that the value “50K” might potentially be some sensitive information and not send the request to node 2. If the request does not contain any potentially sensitive information, then the QTP sends the request to the Secret DEM process at node 2. The name and salary values retrieved at node 2 will be sent to the DEM operating at Top Secret level at node 1 via the QTP. The DEM at node 1 will then select only those names whose salaries are more than 50K. The remaining two requests are processed at node 1 at the Top Secret level.

Example 3. Secret user at node 1 requests to retrieve all information on employees. This query is represented by “Select all from EMP”.

A simple execution strategy generated by QO at node 1 (which operates at the Secret level) will be the following:

(1) Transfer EMP2-U, EMP2-C, EMP2-S from node 2 to node 1.

(2) Merge EMP1-C, EMP1-S, EMP2-U, EMP2-C, EMP2-S at node 1.

Since node 2 handles the Secret level, the first request is sent directly by the Secret DEM at node 1 to the Secret DEM at node 2. The DEM operating at the Secret level at node 2 will retrieve the fragments EMP2-U, EMP2-C, EMP2-S and send the result to node 1. The remaining processing is performed by the DEM at node 1 operating at the Secret level. Note that in this example, a QTP is not invoked.

Example 4. Top Secret user at node 1 requests to retrieve all employee names and department names from EMP, DEPT. The query is expressed by Select Name, Dname from EMP JOIN DEPT where EMP.D# = DEPT.D#. A simple execution strategy generated by QO at node 1 (which operates at the Top Secret level) will be the following:

(1) Select Name, D# from EMP2-U, EMP2-C, EMP2-S at node 2. Merge the results. Let the result of the merge be EMP2*. Select D#, Dname from DEPT2-U, DEPT2-C, DEPT2-S. Merge the results. Let the result of the merge be DEPT2*. Send EMP2*, DEPT2* to node 1.

(2) Select Name, D# from EMP1-C, EMP1-S, EMP1-TS at node 1. Merge the results. Let the result of the merge be EMP1*. Select D#, Dname from DEPT1-C, DEPT1-S, DEPT1-TS. Merge the results. Let the result of the merge be DEPT1*.

(3) Merge EMP1*, EMP2* to obtain EMP*. Merge DEPT1*, DEPT2* to obtain DEPT*. Join DEPT*, EMP* on D#, and project on Name, Dname. Give result to user.

The first request is given to a QTP. QTP will examine the request and determine that it does not have any potential information classified at the Top Secret level. It will send the request to the DEM operating at the Secret level at node 2. The request is carried out at node 2 by the DEM which operates at the Secret level. The results EMP2* and DEPT2* are sent to the Top Secret DEM at node 1 via the QTP. The remaining processing is carried out at node 1 at the Top Secret level.

4. Transaction Processing

In Section 4.1 we discuss a model for transaction processing. In Section 4.2 we will examine the two concurrency control algorithms described in ref. 7 and discuss whether they could be adapted for the limited heterogeneous environment that we have considered in this paper.

4.1. Transaction Model

Issues on transaction management for a homogeneous environment were discussed in ref. 7. Figure 5 illustrates a transaction model for the limited heterogeneous environment that we have
considered. It is assumed that there are four levels $L_1 < L_2 < L_3 < L_4$. Site 1 handles the range $L_2 - L_4$, site 2 handles the range $L_1 - L_3$, and site 3 handles the range $L_1 - L_2$. The DTMs at level $L$ communicate with each other. However, since site 2 does not handle level $L_4$, and site 3 does not handle the levels $L_3$ and $L_4$, the DTM at level $L_4$ at site 1 communicates with the DTM at level $L_3$ at site 2 and the DTM at level $L_2$ at site 3 via a trusted process which we call the transaction trusted process (TTP). Since site 3 also does not handle the level $L_3$, the DTM at level $L_3$ at site 1 communicates with the DTM at level $L_2$ at site 3 via the TTP. We assume that there is such a TTP at each site.

As in ref. 7, we assume that a transaction is a program unit that is a series of query and update requests. A transaction may be distributed. That is, a transaction may be executed at several sites. We also assume that a transaction is assigned a security level and can be executed by a user whose level dominates the level of the transaction. However, if a transaction issued by a user at level $L$ has to be executed at a site which does not handle the level $L$, then the subtransaction at that site is a read-only subtransaction. For example, if the execution of a transaction is issued by a user at level $L_4$ at site 1, then that transaction can only read data items at sites 2 and 3. Therefore, sites 2 and 3 do not participate in the commit operation. That is, at sites 2 and 3 it is automatically assumed that such a transaction is committed.

4.2. Concurrency Control

In ref. 7 we described two algorithms for concurrency control. They are locking and timestamping. We assume that the algorithms described in ref. 7 for the local MLS/DBMSs remain unchanged for the limited heterogeneous environment that we have considered in this paper. However, in the case of a distributed environment, we have found that the locking algorithm, as we have described earlier, does not maintain consistency. The timestamping algorithm not only ensures security, but it also pro-
vides consistency. In this section, we discuss the issues for both algorithms.

**4.2.1. Locking**

First let us consider the locking technique. As described in ref. 7, assuming that there are only two levels, Secret and Unclassified, there are two copies (both Unclassified) of an Unclassified data object. One copy is for Unclassified transactions and one for Secret transactions. For a data object X, let X\(_1\) be the copy for Unclassified transactions and X\(_2\) be the copy for Secret transactions. When Unclassified transactions request locks, the actions of the Secret transactions have no effect. The Unclassified transactions contend for locks on X\(_1\). As soon as an Unclassified transaction finishes updating X\(_1\), a new copy of X\(_1\) is created. Let this copy be X\(_3\). X\(_3\) will then be the new version of X for the Secret transactions. After an Unclassified transaction requests the write lock for X\(_1\), any read request on X by a Secret transaction would be directed to X\(_3\). X\(_2\) may be deleted later by a garbage collector. To ensure consistency, X\(_2\) should be deleted only if there are no locks queued for it. Since we assume that X\(_2\) is an Unclassified object, the process which deletes X\(_2\) must be trusted.

In the case of distributed locking, a transaction must obtain all locks before it can execute. It then releases all its locks after it completes execution. We showed that with our algorithms both the local and global serializability conditions are satisfied for the homogeneous case. We will show that the global serializability condition is not satisfied in the heterogeneous environment that we have considered.

Let us assume that site 1 handles only the Unclassified level and site 2 handles both the Unclassified and the Secret levels. Let X and Y be Unclassified data objects at sites 1 and 2 respectively. Since site 2 also handles the Secret level, there is a copy of Y at site 2 for the Secret transactions. Let Ti and Tj be two transactions executing at both sites 1 and 2. We assume that Ti is Unclassified and Tj is Secret. Since Tj is Secret, it can only read the data objects X and Y and it must originate at site 2. Furthermore, the subtransaction of Tj which executes at site 1 must be Unclassified and, therefore, any read request from Tj to site 1 must be via the TTP at site 2. Let Oi and Oj be conflicting operations by Ti and Tj respectively, on both X and Y. This means that Oi is a write operation and Oj is a read operation. The subtransactions of Ti and Tj that execute at site 1 are Ti1 and Tj1 respectively, and the subtransactions of Ti and Tj that execute at site 2 are Ti2 and Tj2 respectively. Note that Ti1, Ti2, and Tj1 execute at the Unclassified level and Tj2 executes at the Secret level.

At site 2, suppose Ti2 obtains a write-lock first on Y. If Tj2 then requests a read-lock on Y, it must wait until Ti2 releases the lock. However at site 1, suppose Tj1 obtains a read-lock on Y first. If Ti1 requests a write-lock on X, since Tj1 is Unclassified, Ti1 must wait until Tj1 finishes. As soon as Tj1 finishes the read operation, Ti1 can obtain the write-lock on X. Ti1 and Ti2 can continue executing at sites 1 and 2, respectively. After Ti2 releases its write-lock, Tj2 reads the new version of Y. Global serializability is not maintained because Tj precedes Ti at site 1 and Ti precedes Tj at site 2.

Had site 1 been able to handle the Secret level, then Tj1 would have executed at the Secret level and if Tj2 finds that Ti2 is already writing, it could have informed Tj1 to wait also. Furthermore, Ti1 would not have to wait for Tj1 as it would work with its own copy of X. This way global serializability was preserved for the homogeneous environment described in ref. 7.

**4.2.2. Time stamping**

In ref. 7 we proposed the following modification to the time stamp algorithm. As in the case of the locking approach, we use two copies of an Unclassified data object; one for Unclassified transactions and one for Secret transactions. Whenever a Secret transaction issues a read request to an Unclassified data object X, first ensure that all Unclassified transactions with lower transaction numbers have
completed their requests on X. Then process the read request issued by the Secret transaction. Note also that, after an Unclassified transaction finishes updating an object, a new copy of the object is made for the Secret transactions. The old copy used by the Secret transaction may be deleted by a garbage collector. To ensure consistency, this copy must be deleted only after all Secret transactions with transaction numbers lower than that of the Unclassified transaction have finished the read operation on it. This means that the process which deletes it must be trusted. In ref. 7 we showed that both local and global serializability conditions are satisfied for a homogeneous environment*. We now argue that the global serializability condition is satisfied for the limited heterogeneous environment that we have considered.

Consider the example that we described for the locking approach previously. Ti and Tj are Unclassified and Secret transactions, respectively. The subtransactions of Ti that execute at site 1 and 2 are Ti1 and Ti2, respectively. The subtransactions of Tj that execute at sites 1 and 2 are Tj1 and Tj2, respectively. Note that Ti1, Ti2, and Tj1 are Unclassified while Tj2 is Secret. We consider two cases.

(i) Suppose the time stamp of Ti is less than the time stamp of Tj. When the subtransactions are assigned time stamps at sites 1 and 2, the TTP at site 2 can ensure that Tj1 is sent to site 1 after Ti1 is sent. This would mean that the time stamp of Ti1 is less than the time stamp of Tj1. Also the DTM operating at the Secret level at site 2 can ensure that the time stamp of Tj2 is greater than the time stamp of Ti2. Therefore, at both sites the operation Oi will precede Oj. That is, Ti will precede Tj.

(ii) Suppose the time stamp of Tj is less than the time stamp of Ti. If Ti1 is assigned a time stamp at site 1 before the TTP sends Tj1 to site 1, then Tj is aborted. Otherwise, Tj1 is assigned a time stamp less than the time stamp of Ti1. At site 2, the DTM operating at the Secret level can ensure that the time stamp of Tj2 is less than the time stamp of Ti2. If Oj does not precede Oi at site 1, then Tj1 is aborted. This would mean that Tj is not committed. Suppose Oj precedes Oi at site 1. If Oj does not precede Oi at site 2, then the only way this is possible is either (i) for Tj2 to be aborted or (ii) for Oj to be requested before Oi is requested, but Oi is carried out before Oj is carried out. However, Oj will be carried out on the older version of the data object and not on the version that results after Oi completes. The end result is as if Oj precedes Oi at site 2.

5. Issues on Secure Interoperability

In order to ensure the interoperability of different MLS/DBMSs, in addition to handling different accreditation ranges, several other types of heterogeneity need to be addressed. In this section we describe some of the issues that need further investigation.

(i) Schema (or data model) heterogeneity. Not all the databases in a heterogeneous architecture may be represented by the same data model. Therefore, the different conceptual schemas have to be integrated. In order to do this, translators which transform the constructs of one multilevel data model into those of another need to be developed.

(ii) Transaction processing heterogeneity. Different MLS/DBMSs may utilize different algorithms for transaction processing. The various concurrency control mechanisms that have been adapted to function in a multilevel environment need to be integrated for a heterogeneous environment.

(iii) Query processing heterogeneity. Different MLS/DBMSs may utilize different query processing and optimization strategies. One of the
research areas here is to develop a global cost model for distributed query optimization.

(iv) Query language heterogeneity. Different MLS/DBMSs may utilize different query languages. Standardization efforts are necessary to develop a uniform interface language.

(v) Constraint heterogeneity. Different MLS/DBMSs may enforce different integrity/security constraints which could be inconsistent. For example, one MLS/DBMS could enforce a constraint that all employees are Secret while another MLS/DBMS may enforce a constraint that all employees are Top Secret. These differences need to be reconciled.

(vi) Different security policies. Each MLS/DBMS could enforce its own security policy for mandatory, as well as discretionary, security. In addition, different authentication and integrity mechanisms may be used. For example, one system could enforce a read-at-or-below-your-level and write-at-your-level policy while another system could enforce read-at-or-below-your-level and write-at-or-above-your-level policy.

(vii) Different granularity of classification. Even if the relational data model is utilized at all nodes, the granularity of classification could be different. For example, one system could enforce classification at the tuple level while the other system could enforce classification at the element level.

From the above discussion it is clear that the steps to achieve secure interoperability are by no means straightforward, and we believe that some of them are impossible, given the current state of the art in both DDBMS and MLS/DBMS technologies. Furthermore, the problems are compounded if the different MLS/DBMSs are autonomous. In such an environment, the DBA/SSO of an MLS/DBMS could not only decide when to join or leave the distributed environment, he could also determine which information to share with the other MLS/DBMSs. Therefore, until global solutions to interconnecting heterogeneous and autonomous non-multilevel databases become available, security policies, models, and architectures have to be developed on a case-by-case basis. That is, customized solutions to meet the individual customer needs are presently needed.

6. Conclusion

For many applications, it is important to provide secure, efficient and reliable access to multilevel databases stored at different sites. Present distributed database systems do not provide adequate mechanisms to meet this objective. Hence the solution requires the operation of a distributed database management system (DDBMS) capable of handling multilevel data. Such a system is also called a multilevel secure distributed database management system (MLS/DDBMS).

The purpose of this paper is to identify the impact of multilevel security on the functions of a DDBMS which operates in a limited heterogeneous environment. The particular type of heterogeneity addressed in this paper is the case where not all of the nodes handle the same accreditation ranges. We first defined two architectures for an MLS/DDBMS and discussed the advantages and disadvantages of each architecture. Then we selected an architecture and discussed issues on query processing and transaction management. Finally, we discussed some complex issues that must be addressed if the secure interoperability of different MLS/DBMSs is to be achieved.

In addition to implementing the algorithms and strategies described in this paper, our current investigation on handling heterogeneity in MLS/DDBMSs includes issues on developing a unified multilevel data model as well as the object-oriented approach to achieving secure interoperability.

Acknowledgements

We gratefully acknowledge the US Air Force (Rome Laboratory) for supporting our work on
B. Thuraisingham and H. Rubinovitz/Security Issues

security issues for heterogeneous distributed database management systems under contract F19628 89-C-0001. We thank Joe Giordano for his support and encouragement.

References


Dr. Bhavani Thuraisingham is a lead engineer at the MITRE Corporation working in secure object-oriented database systems, secure distributed database systems, and secure intelligent database systems. Previously she was at Honeywell Inc. where she was involved with the design of the secure DBMS Lock Data Views. She was also an adjunct professor of computer science at the University of Minnesota. Dr. Thuraisingham received an MS degree in Computer Science from the University of Minnesota, an MSc. degree in Mathematical Logic from the University of Bristol, UK, and a Ph.D. degree in Theory of Computation from the University of Wales, Swansea, UK. She has published over thirty journal papers in Database Security, Distributed Processing, Computability Theory, and AI. She is a member of the IEEE Computer Society, the British Computer Society, the ACM, and the Association of Symbolic Logic, and is a member of the editorial board of the Journal of Computer Security. She also served as the program chair for the 6th IFIP 11.3 Working Conference in Database Security.

Dr. Harvey H. Rubinovitz is a member of the technical staff at the MITRE Corporation working in secure distributed database systems and modeling and simulation of secure database algorithms. Dr. Rubinovitz received an M.S. degree in Computer Science from Framingham State College, an M.S. degree in Computer Science from Worcester Polytechnic Institute, and a Ph.D. degree in Computer Science and Engineering from the University of Connecticut. He is a member of the ACM, and the Society for Computer Simulation.