Computers & Security, 7 (1988) 387–396

Refereed Article

Multilevel Security Issues in Distributed Database Management Systems

John McHugh[1] and Bhavani M. Thuraisingham[2]

[1] *Computational Logic Inc, NC, U.S.A.*
[2] *Honeywell, Corporate Systems Development Division, Golden Valley, MN, U.S.A.*

This paper describes the security needs in a distributed processing environment common to many enterprises and discusses the applicability of multilevel secure database management systems in such an environment.

Keywords: Multilevel security, distributed databases.

1. Introduction

The advent of computers earlier this century has led us not only to rely increasingly on the information stored in them but also to trust their output. This alliance with the computer has resulted in the automation of basic operations and management support functions. This was done through the manipulation of computerized information maintained in a database. Although the creation and manipulation of increasingly larger database systems offer many advantages, it also produces breaches in security when the output of confidential and sensitive information is compromised. The move towards distributed databases increases the chances of security violations. Consequently, access to databases must be controlled to ensure that accidental and malicious corruption of data is minimized or even eliminated. This calls for increasingly ingenious ways of countering breaches in security.

One such ingenious technique that is gaining popularity not only in military organizations but also in commercial enterprises is the design of multilevel secure database management systems (MLS/DBMS). An MLS/DBMS addresses the rather natural expectation that users cleared to different levels can share a database containing multilevel data. This paper describes a particular design of an MLS/DBMS that is applicable to a distributed processing environment common to many enterprises. Most enterprises are migrating towards distributed databases for information storage. Therefore it has to be ensured that these distributed databases are secure.

There are two aspects to providing security in such systems. One is to enforce security on the network that interconnects the various hosts and the other is the security that has to be provided due to the distribution of data. Many issues need to be investigated in order to provide a secure network. In many cases, the network medium is subject to attack. Encryption offers partial solution to some of these threats, but numerous covert channels, where information is signaled in such a way that security is violated, can exist through network media even when encryption is used. Network security is an active research area and we believe that solutions to many of the security problems are not likely for some time. Therefore we will not describe the network security issues in this paper. Instead we will assume that the interconnection of the various hosts is secure and we will only concentrate on the security that has to be provided due to data distribution.

2. Multilevel Security

Security is the control of the flow of data and information

0167-4048/88/$3.50

through the system and its interfaces with the outside world. This is done to prevent the unauthorized disclosure of information or modification of data protected by the system. This can be achieved by assigning data to sensitivity classes and restricting the types of access individuals and code acting on their behalf can have to each of the sensitivity classes. A security policy describes the set of restrictions. One reasonable approach is to associate an element from a partially ordered set of security levels with each of the sensitivity classes and with each individual, and to then restrict accesses so that information and data can only flow upward in level. Such a policy is called a multilevel security policy and a system which enforces one is said to be multilevel secure.

It is only recently and with considerable difficulty that the problem of providing multilevel security for operating systems and for their associated resources has been solved. The solution to such a problem is generally referred to as a trusted computing base (TCB). The TCB is an isolated subsystem, which must be shown to enforce, completely and correctly, the system security policy, regardless of the actions of the rest of the system. Thus it must be shown that the TCB will continue to function correctly even if there is hostile code on the system trying to subvert the TCB. Instances of such hostile code are called Trojan horses, because it is quite likely the user who invokes the code is completely unaware of the hostile nature of the code. In order to prove that the TCB is complete and correct, a mathematical model of the system and its security policy is developed and formal verification methods are used to show that the system design satisfies the security policy [9].

A fundamental concept in the development of TCBs has been access controls which are embodied in an unbypassable and tamperproof reference monitor. These access control policies are based on the assumption that it is not feasible to verify all programs that can be executed in a computing system. Therefore there is a possibility that the system will be attacked by hostile application programs. The mandatory security policies are then aimed at confining Trojan horse programs in such a manner that their attempts at hostile behavior are frustrated.

The Bell and LaPadula policy [1] defines security in terms of the simple security property and the *-property. The simple security property states that a process-like entity called a subject can read from a file-like entity called an object only if its clearance is greater than or equal to the classification level of the object. The *-property states that a subject can write into an object only if the subject's clearance is less than or equal to the classification level of the object. In essence, classification levels form a partially ordered lattice and information can flow only upward along the lattice unless otherwise authorized. Downward information flows are necessary to make a system functional, but they must be controlled in order to preserve security. This process is achieved through "trusted subjects" whose actions are verified to bypass the *-property in a manner in which it is believed will not violate the intent of the *-property.

When there is a database on an MLS system, it is possible that not all of the data contained in the database are equally sensitive. However, present-day database management systems are not built with adequate controls and mechanisms to ensure that users are allowed to access only the data for which they have been granted a clearance and for which they have a legitimate reason for accessing, but at the same time provide for the sharing of data by these users. Thus an (MLS/DBMS) is different from a conventional DBMS in at least the following ways.

(1) Every data item in the database has associated with it one of several classification or sensitivities, that may need to change dynamically over time.
(2) A user's access to data must be controlled based upon the user's authorization with respect to these data classifications.

Providing an MLS/DBMS service on current computing systems, even on a TCB, presents a new set of problems. The most obvious of these problems is that the granularity of classification in a DBMS is generally finer than a file and may be as fine as a single data element in a file, whereas that of the TCB is a file. Another problem that is unique to databases is the necessity to classify data based on content, time, aggregation, and context. Furthermore, DBMSs are also vulnerable to subtle covert channel attacks in which Trojan horses within the DBMS encode sensitive information in otherwise benign fields and also to inference attacks where a user infers unauthorized data from the knowledge that he has accumulated

and/or the context in which data are displayed. Any security policy for MLS/DBMS must not only protect against direct disclosure of data, but it must also attempt to limit the attacks from hostile users.

The Air Force Summer Study of 1982 [12] proposed near-term solutions and long-term requirements for MLS/DBMSs. The recommended near-term solutions are three families of architectures. One is based on integrity locks, another based on developing kernelized DBMS and the third based on distributed data. All three families will use untrusted DBMSs. In the integrity lock approach the data are cryptographically sealed by a trusted device when it is created. The data record would include its security markings. When the data are subsequently queried by the user, the trusted DBMS front end would recompute the cryptographic checksum and compare it with the value attached to the record when it was first created. If they differ, the read is aborted. The basis for the kernelized approach is to use untrusted DBMS on secure operating systems. The operating systems would segregate the data into distinctly defined segments. A trusted front end operating as a shared segment would perform security sensitive writes for users at any classification level. In the distributed family, untrusted back-end DBMSs are interconnected to a trusted front-end machine. The data are segregated among the back-end machines. All operations that require trust are performed at the front end. The long-term requirements of MLS/DBMSs include the need for classification systems that handle multilevel compartmented data and dissemination markings, mechanisms that classify data by their association with other data, solutions to inference and aggregation problems, and sanitization and downgrade abilities.

Several solutions are being proposed to design MLS/DBMS based on the report of the Air Force Summer Study [3, 4, 6, 7, 10]. The merits and demerits of each solution depend on the application environment. For the application in which we are interested, namely the distributed processing environment, it appears that the solution proposed by the Summer Study to achieve security based on data distribution is most applicable. This is because not only does this approach provide a solution to the problems due to data distribution, it also enables most enterprises to use existing untrusted state-of-the-art DBMSs instead of completely changing the designs and operations of such systems. Therefore designs of MLS/DBMS based on data distribution will be the focal point of this paper. We believe that the realization of such an MLS/DBMS is largely an engineering exercise. The interrelationship between the security and data requirements of the application will influence both the architecture of the DBMS and its security policy.

If the goal is to produce a working MLS/DBMS based on data distribution within the next 2–3 years, then most research questions such as inference and aggregation problems must be either resolved or side-stepped. It is most likely that this will be the case in most commercial applications as the enterprises cannot afford the compromises to security in the mean time while waiting for a completely secure DBMS. However, it is desirable that the design should be flexible so that it can evolve incrementally while new discoveries are made in the database security arena.

3. Overview of Distributed Databases

As quoted in ref. 2, a distributed database is a collection of data which are distributed over different computers of a network. Each site has autonomous processing capability and can perform local applications. Each site also participates in the execution of at least one global application, which requires accessing data at several sites using a communication subsystem. A graphical representation of a distributed database is shown in Fig. 1.

A DDBMS supports the creation and maintenance of distributed databases. A typical commercial DDBMS is depicted in Fig. 2 (taken from ref. 2). Its components are as follows.

(1) The database management component (DB).
(2) The data communications component (DC).
(3) The data dictionary (DD) which is extended to represent information about the distribution of data in the network.
(4) The distributed database component (DDB).

The services that are supported by a DDBMS include the following.

(1) Remote database access.
(2) Distribution transparency.
(3) Consistency and integrity maintenance.
(4) Concurrency control and recovery.

The access of remote databases may be necessary in order to

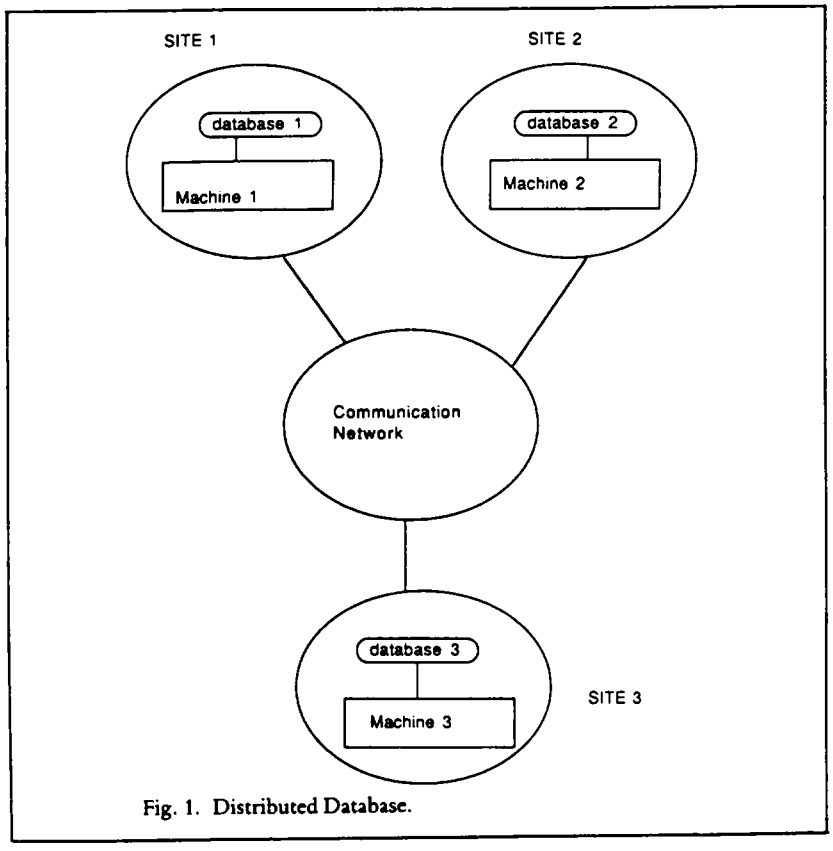


Fig. 1. Distributed Database.

process queries. If the data are distributed, then a user's query will be decomposed into subqueries if necessary and routed to the appropriate sites. At each site a response will be generated and these responses will be assembled at user's site or some other convenient site. The assembled response will be returned to the user. Distribution transparency shields users from the knowledge of the location of the data. There is a heavy trade-off between distributed transparency and performance. Consistency maintenance is necessary because of replicated copies of the data. In other words, if the data that are updated are replicated, then it has to be ensured that all copies of this data are consistent. Integrity maintenance is required in order to ensure that the data in the system are correct at all times. Concurrency control algorithms handle the situation where a database needs to be accessed by multiple users at the same time. Recovery techniques handle site crashes.

As quoted in ref. 2, distributed databases are important for economic, organizational, and technological reasons. They can be implemented in long haul networks, local area networks, or front-end machines having point-to-point connections to back-end machines. It is envisaged that as the availability of distributed databases increases the number of applications will grow rapidly.

4. Designs of MLS/DBMS Based on Data Distribution

The architectures proposed by the Air Force Summer Study for an MLS/DBMS are generalized in Fig. 3. The M(M > = 1) users access the N(N > = 1) back-end machines. This access is controlled via L(L > = 1, L < = N, L < = M) front-end machines. The method by which users are connected to the front ends is immaterial, but must meet the criteria that are predetermined. These criteria depend on the features such as security, integrity, and performance that are desired of the system. Each back-end machine may have point-to-point connection to the front-end machines or it may be connected through a local area network. For reasons described earlier, we assume that this interconnection is secure.

The design strategy for the MLS/DBMS architecture depends on the hardware components, interconnection media, software interfaces, and the methods used to distribute the data. Two of the techniques that have been proposed for data distribution are perlevel distribution and replicated distribution. In both cases, the architecture consists of a front-end machine having point-to-point connections to two or more back-end machines. We assume a set of levels ordered HI > ... > LO. In the perlevel design, each back end contains data at a single level, thus a given back end operates at a single level in the range {LO ...HI}. In the replicated design, each machine contains all the data up to a given level in the range {LO ... HI} and operates in the system high mode at that level.

In the above discussion we have

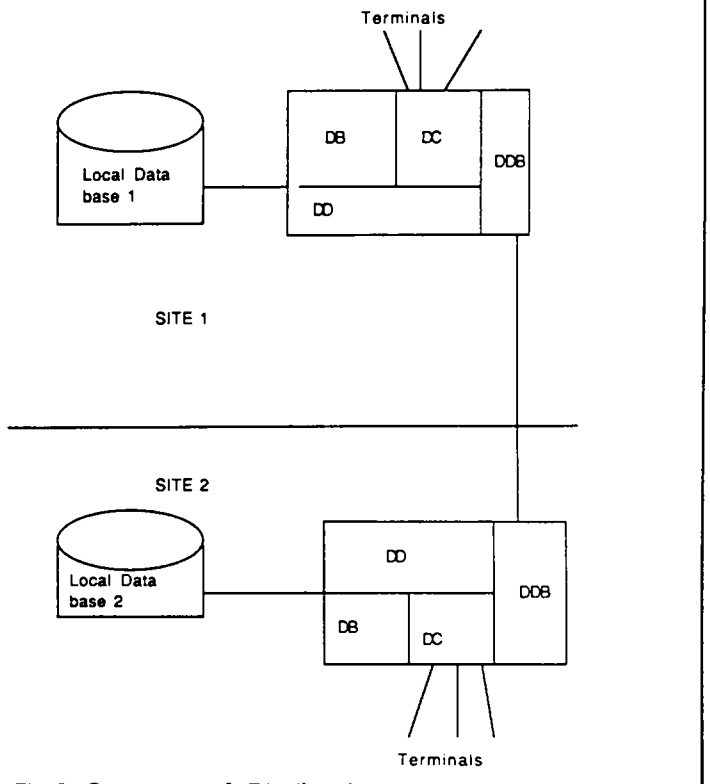


Fig. 2. Components of a Distributed DBMS (Taken from ref. 2).

assumed that the levels are hierarchical. However, this is not always the case. For example, in an office environment, it is likely that the levels assigned to secretaries and engineers may be incomparable while the managers could access both the secretaries' and the engineers' information. In such a case the level of the manager will be an upper bound of the levels of the secretary and engineer. However, such incomparable levels can lead to combinatorial explosion. As an example, three incomparable levels will result in seven different access groups. In such cases the problems of designing an MLS/DBMS are compounded. Therefore our discussion will assume that the levels are hierarchical. We do believe that the design has to be extended to include non-hierarchical levels in order to produce any realistic system.

The main advantage of the perlevel design over the replicated one is that the updates do not result in inconsistencies. This is because in the perlevel design the data are not replicated. The replicated design has an advantage in terms of query processing. As all the information pertaining to a user's level is stored in one machine, a user's query has to be directed only to that machine. In the perlevel design, a user's query may be directed to different machines. Some decomposition may also be necessary before the query is routed. The responses are generated at each machine and they will be assembled possibly at the front end.

We will illustrate the above discussion with an example. Suppose the database consists of a relation EMP with the attributes NAME, ADDRESS, SOC_SEC, PHONE, SALARY AND PROJECT. Let SOC_SEC be the key value. Let the attributes NAME, ADDRESS, SOC_SEC, and PHONE be at the level of the secretary and the attributes SALARY and PROJECT be at the level of the manager. In the perlevel design, the secretary database will store the attributes SOC_SEC, NAME, and PHONE while the manager database will store SOC_SEC, SALARY, and PROJECT. It should be noted that since SOC_SEC is the key, it is stored in both databases. Alternatively, we could have used tuple_IDs which depend on the key in order to identify the tuples.

The tuple_IDs may be generated by the front end. Suppose a manager wants to obtain all the information about the relation EMP. This query will be decomposed first and then routed to the two back-end machines. The responses generated at these machines will then be assembled. If a secretary wants to obtain information about EMP, then the query is modified to get only the attributes that are in the secretary database. This modified query will be routed to the secretary's machine and the response will be generated. In this case no assembly is needed. In the case of the update operation, a request by a secretary to update a tuple in the relation EMP will be decomposed and sent to both machines. A manager, however, can only update his own database. He will not

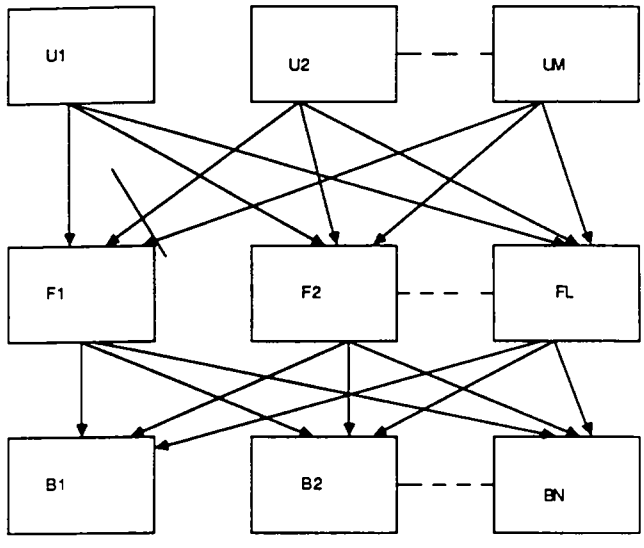


Fig. 3. MLS/DBMS Architectures.

be able to update the secretary's database as this will violate the *-property.

In the replicated design, the secretary database will contain the same information as in the case of the perlevel design. The manager's database, however, will contain the entire relation EMP. A query request by the manager will be directed to the manager's database and a query request by the secretary will be directed to the secretary database. The response generated will be returned to the user and no assembly is necessary. In the case of updates, a manager can update his own database whereas a secretary can update both databases. However, it should be ensured that the replicated data in both machines are consistent.

In order to operate an enterprise efficiently, performance is an important factor in selecting a particular design for an MLS/DBMS. As an example, the choice of the designs can be heavily influenced by the frequency of update and query operations. If the system is a read-often write-seldom one then the replicated design is preferred. If the situation is reversed, then one should consider the perlevel design seriously. In addition to the frequency of updates and query operations, performance objectives should also be established at the MLS/DBMS component level such as the front-end and the back-end machines in order to achieve the overall objective of a high performance system. The subsystems that impact the overall performance of the front end include the security kernel which consists of trusted code, the data dictionary/directory manager which contains information about the locations of the back-end machines and the necessary metadata, the speed of the processor, and the size of the memory. The performance objectives for the components of the back end must reflect the fact that we expect a greater proportion of the processing to be accomplished at the back end. However, owing to security considerations, some of this processing may have to be accomplished at the front end. Other factors that influence the selection of the design are the security considerations, functionality, and ease of implementability. We believe that both designs can be implemented easily and they offer an acceptable level of functionality. Since the objective is to produce a secure MLS/DBMS for an operational enterprise, the security considerations merit a detailed discussion.

5. Security Aspects of the Designs

In this section we discuss in more detail the impact of security considerations on the realization of the MLS/DBMS as a distributed system and vice versa. The security aspects of the designs can be described in terms of the functionality of the TCB at the front end. Depending on the application, such a TCB can range from a simple guard to a substantial reimplementation of much of the back end functionality. In the discussion that follows, we will assume that the following structures are typical of the DBMS.

A database consists of one or

more relations, relation schemas, and views. A DBMS can support one or more databases at a given time. A relation is a table having rows and columns. A column of a relation represents an attribute or field of the relation. All entries in a column are of the same type, that is, they are elements of some value set associated with the attribute. The tuple of a relation is a cross-section containing a corresponding element from each column in the relation. The classification of a row dominates the classifications associated with its individual elements.

Elements are the atomic data items of the relation. Element classifications can be determined on an individual basis or inherited from the column classifications. The latter case is likely to be more common as there are relatively few cases in which value taken out of context justifies an exceptional classification. Context can cause a particular row, column, or relation at a level that exceeds the classification of any of its elements.

We can define a multilevel relation as an extension of the table formulation described above. The tabular structure is augmented with security labels that provide the classification of the relation, its columns, its rows, and possibly its elements. For any given level, we can create another relation which is a (possibly multilevel) view of a multilevel relation in which the classifications of the relation, its columns, its rows and its elements are all dominated by the given level.

In the distributed MLS/DBMS, the actual storage of relations will be performed by the untrusted back-end machines. It will be the job of the front-end TCB to provide users with the appropriate views to ensure that multilevel information is accurately labeled. The TCB functionality and, as a consequence, its size, complexity, and cost can vary greatly depending on the architecture of the DBMS system and the application requirements.

We will consider the perlevel and the replicated distributions of data among the back-end DBMSs and examine their impact on TCB functionality. We assume that tuples containing data at multiple levels occur frequently enough to require special provision for their entry into the system. The operations to be supported should include database definition, MLS data entry, query operations, update, and application influence.

Database Definition

Database definition is the creation and processing of a new database description. This includes the definition of a master schema, views, the preparation of a data dictionary, relation definition etc. In addition to the usual DBMS functionality, MLS database definition includes the assignment of security levels to database entities and partitioning the database structure to fit the storage distributions.

Because the partitioning is security related, at least part of this code must be in the TCB. Much of the schema construction and manipulation can be done with untrusted code, but as a minimum, a trusted review and checking program must be supplied to ensure that the MLS schema property reflects the intent of the DBMS designer. This program would operate at the highest level required for classifying any part of the DBMS schema and would have the necessary upgrade–downgrade authority to install the partitioned schemas on the back-end machines. It would also produce data structures to be used by the TCB in mediating access to the back end.

MLS Data Entry

Single level tuples can be entered into the DBMS at their own level. Prior experience indicates that this is likely to be the case and no special impact on the TCB is foreseen. Even so, it seems likely that multilevel data streams from other trusted systems as well as occasional manual inputs will be required. For these cases, a trusted application is envisioned to permit such entry. It functions at the level required by the highest data being entered and has the downgrade authority to cause the appropriate lower level entries to take place. This application will be restricted to data input operations so that it does not become a conduit for wholesale downgrading of the DBMS contents. Its purpose is to allow proper distribution of a correctly labeled multilevel input stream over the database. Because the process generates single-level transactions that are further processed by the TCB, it can be independent of the distribution technique. Automatic classification of data, on the basis of context, time, context etc., would require sophisticated trusted applications. We believe that this is not feasible in the near term.

Query Operations

We assume that the distributed MLS/DBMS will support the same set of relational query operations as those supported by the normal DBMS. These include the primitive operations of union, set

difference, cartesian product, projection and selection [11]. The extended operations intersection, quotient, and various join forms are also supported. If all the data required to answer a given query are present on a single DBMS machine, the TCB's task in query processing is minimal. This will be the case for most operations on replicated distributions. For a perlevel distribution, queries involving data on multiple machines must be decomposed, processed in part at the various levels and the results combined by the TCB. This may require trusted code, duplicating much of the back-end DBMS functionality, if the multilevel nature of the responses is to be preserved. If an implicit upgrade of portions of the response is acceptable, some of this functionality can be placed in untrusted code residing in the front end outside the TCB, but code that communicates down to the lower level DBMSs must still be part of the TCB.

Update

Update operations cause potential problems in both distributions. It is hoped that the frequency of multilevel or downgrading updates will be sufficiently low so that relatively simple procedures can be used for this purpose. Every effort should be made to organize the application in such a way that updates are initiated at the level of the data being altered. In the perlevel distribution, the effect of such an update is restricted to a single back-end machine. In the replicated distribution, updates at lower levels imply automatic upgrades and updates at the higher level or levels. This creates potential synchronization and transient inconsistency problems but no particular security problems. The TCB will be responsible for generating the appropriate update operations for the higher level views of the system.

Application Requirements

Application requirements can also influence the TCB construction. It has been noted that the replicated distribution does not provide a true MLS capability because the security labeling for lower level portions of data retrieved from a DBMS containing multiple levels cannot be trusted. If such data are ultimately being processed by an untrusted application at the high level, this is unimportant, as results coming from such an application must be assumed to be at its level. It is necessary to ensure accurate labeling from a security standpoint only if the data are to be processed by a trusted application that will keep the label information intact through processing and transformation. In all cases, a downgrading process must be used, even if the data are thought to be of a lower level than their processing environment. Should applications that can process multilevel data be considered, the use of perlevel distribution or the addition of integrity locks to the replicated distribution would seem to be indicated.

Summary of TCB Functionality

The above discussion indicates that the TCB of an MLS/DBMS using data distribution to achieve security can range from relatively simple to quite complex. The simplest TCBs result from replicated distributions in cases where accurate labeling of data in the mixed levels DBMSs is not required. In this case the TCB functionality is restricted to providing separation facilities to ensure that the queries and responses flow correctly from single-level users to the appropriate replicated DBMS and that updates are reflected in all DBMSs operating above the level initiating the update. From a practical standpoint, trusted applications for database definition, MLS data entry, and downgrading are also required.

At the other extreme, a TCB for a perlevel distribution in which true multilevel query responses are required would duplicate much of the functionality of the back ends. At both extremes, the TCB must have substantial knowledge of the functionality of the back-end DBMSs. This means that its interface definition is likely to include a substantial portion of the back-end interface. The richness of this interface will influence the complexity of the TCB.

6. Conclusion

In this paper we have described issues involved in the design and implementation of an MLS/DBMS based on data distribution applicable to a distributed processing environment common to many enterprises. We have also discussed the impact of security considerations on the realization of the MLS/DBMS.

Our design does have some limitations. It has been heavily influenced by the hierarchical levels of classification. This has to be extended to include incomparable levels. We have also assumed that the interconnection of the various machines within an enterprise is

secure and we have also sidestepped the long-term requirements of an MLS/DBMS such as content and context-dependent classification, dynamic classification, inferences and aggregation. Another crucial issue that has to be considered is data integrity. Various integrity policies for commercial systems are being formulated. Therefore it is important that such functions be incorporated into the system as it evolves.

In spite of its limitations, the implementation of our design is achievable using existing products. For example, Honeywell's A1-rated Secure Communications Processor (SCOMP) [8] can act as a front end to Honeywell's Distributed Database Testbed System (DDTS), a superior distributed DBMS prototype [5]. Although the performance of such an MLS/DBMS is yet to be determined, it will offer the promise of a high degree of security and functionality sufficient to satisfy a large class of users.

Acknowledgments

The authors would like to thank Dr. Saeed Rahimi, Ms. Patricia Dwyer, Dr. Krishna Mikilineni, and Mr. Emmanuel Onuegbe from the Honeywell, Corporate Systems Development Division and Professor Wei-Tek Tsai from the University of Minnesota for their valuable suggestions.

References

[1] D. Bell and L. Lapadula, *Secure Computer Systems: Mathematical Foundations*, ESD-TR-73-278, Vol. 1, AD 770 768, The Mitre Corporation, Bedford, MA, 1 May 1973.

[2] S. Ceri and G. Pelagatti, *Distributed Databases, Principles and Systems*, McGraw-Hill, New York, 1984.

[3] D.E. Denning, S.K. Akl, M. Morgenstern, P.G. Neumann, R.R. Schell and M. Heckman, Views for multilevel database security, *Proc. 1986 IEEE Symp. on Security and Privacy, April 1986*, pp. 156–172.

[4] D.E. Denning, T.F. Lunt, R.R. Schell, M. Heckman and W. Schockley, A multilevel relational data model, *Proc. Symp. on Security and Privacy, April 1987*.

[5] C. Devor, R. Elmasri, J. Larson, S. Rahimi and J. Richardson, Five–schema architecture extends DBMS to distributed applications, *Electron. Des.*, (March 1982) 27–32.

[6] B.B. Dillaway and J.T. Haigh, A practical design for a multilevel secure database management system, *Aerospace Security Conf., McLean, VA, December 1986*.

[7] P.A. Dwyer, G.D. Jelatis and M.B. Thuraisingham, Multilevel security in database management systems, *Comput. Secur.*, 6 (3) (1987) 252–260.

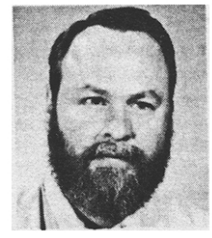
[8] L. Fraim, SCOMP: A solution to the multilevel security problem, *IEEE Trans. Comput.*, 16 (7) (July 1983) 26–34.

[9] R. Kemmerer, A brief summary of a verification assessment study, *Proc. 9th Ann. Nat. Computer Security Conf., NBS, Gaithersburg, MD, 1986*.

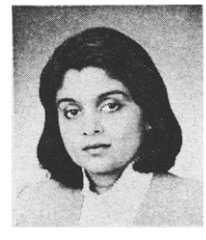
[10] M.B. Thuraisingham, Security checking in relational database management systems augmented with inference engines, *Comput. Secur.*, 6 (6) (1987) 479–492.

[11] J. Ullman, *Principles of Database Systems*, Computer Science, Rockville, MD, 1982.

[12] Multilevel data management security, *Committee on Multilevel Data Management Security, Air Force Studies Board, Commission on Engineering and Technical Systems, National Research Council*, National Academy Press, Washington, DC, 1983.



Dr. John McHugh received the B.S. degree in physics from Duke University in 1963, the M.S. degree in computer science from the University of Maryland, College Park in 1974, and the Ph.D. degree in computer science from the University of Texas, Austin, in 1983. He is a Vice President and head of the North Carolina office of Computational Logic Inc. He is also an Adjunct Assistant Professor of computer science at Duke University. From 1983 to 1986 he was a Senior Computer Scientist at the Research Triangle Institute in Research Triangle Park, NC. From 1973 to 1978, he worked for the Naval Research Laboratory as a mathematician and for the University of Minnesota Hospital as an application programmer. From 1965 to 1973, he worked for the National Oceanic and Atmospheric Administration as a computer systems analyst. From 1964 to 1965, he was a patent examiner in the United States Patent Office. His research interests include formal specification and verification, software fault tolerance, secure systems, reliable software design, and multiprocessor applications. Dr. McHugh is a member ot the Institute of Electrical and Electronics Engineers (IEEE), IEEE Computer Society and Association for Computing Machinery.



Dr. Bhavani M. Thuraisingham is a Principal Research Scientist at Honeywell Corporate Systems Development Division and an Adjunct Professor of computer science and member of the Graduate Faculty at the University of Minnesota. Her research interests include all aspects of Data–Knowledge Base Management

Systems. Previously she worked at Control Data Corporation designing and developing computer networks. She has also served as a faculty member at the Department of Computer Science, New Mexico Institute of Mining and Technology, and at the Department of Mathematics, University of Minnesota. Dr. Thuraisingham received the B.Sc. degree in mathematics and physics from the University of Sri-Lanka, an M.Sc. degree in mathematical logic from the University of Bristol, U.K., and a Ph.D. degree in recursion theory and theory of computation from the University of Wales, U.K. in 1975, 1977 and 1979 respectively. She also has the M.S. degree in computer science from the University of Minnesota. She has published many articles in *Database Security, Distributed Processing, AI Applications and Logic*. She is a member of Sigma-Xi, the Association for Computing Machinery and IEEE Computer Society.