Security issues for federated database systems

Bhavani Thuraisingham

Mitre Corp., Burlington Road, Bedford, MA 01730, USA

This paper describes security issues for federated database management systems set up for managing distributed, heterogeneous and autonomous multilevel databases. It builds on our previous work in multilevel secure distributed database management systems and on the results of others' work in federated database systems. In particular, we define a multilevel secure federated database system and discuss issues on heterogeneity, autonomy, security policy and architecture.

Keywords: Multilevel secure federated database system, Heterogeneity, Autonomy, Security policy, Schema, Architecture.

1. Introduction

A federated database system (FDS) has been defined as a collection of cooperating database systems which are possibly autonomous and heterogeneous [1]. The intent is for a database system to continue its local operation and at the same time participate in a federation if it wants to. The database systems participating in a federation can be homogeneous, in which case they are all designed and operate identically, or they can be heterogeneous with respect to several aspects such as data models, designs, semantics, and constraints, among others. While heterogeneity brings about complexities not present in a homogeneous environment, autonomy, which enables a database system to join or leave a federation whenever it wishes to, makes the task of developing an FDS even more difficult. Recently, several research and development activities on FDSs have been reported (see, for example, [2]). Although some promising results have been obtained, several tasks, such as multiuser updates, have yet to be carried out successfully. It has been more or less agreed that the solutions to the problem of interconnecting multiple database systems are a generation away [3].

While advances were being made in the area of federated database systems during the past decade, much work was carried out on the design and development of multilevel secure database management systems (MLS/DBMS), mainly based on the relational data model (see, for example, [4–7]). As a result, at present, some of these MLS/DBMSs are commercially available. In these database systems, users cleared at different security levels are expected to access and share a database consisting of data at different sensitivity levels. The MLS/DBMS must ensure that users retrieve data classified at or below their security levels. As more and more of these systems are in use, one cannot avoid the need for securely interconnecting them. Furthermore, we believe that there will come a time when the operation of a multilevel secure federated database system (MLS/FDS) is necessary. However, current trends predict that much work needs to be done before such a system can be operational. This is because (1) only recently has research begun on security in distributed database management systems, which is just the first step toward developing an MLS/FDS; and (2) much
work needs to be done before even a non-multilevel FDS can be developed.

Although means of developing an MLS/FDS are generations away, we believe that one must at least examine the issues involved in developing such a system, given the current state of the art in FDS and MLS/DBMS technologies. This paper describes our investigation of the security issues for federated database management systems for managing distributed, heterogeneous and autonomous multilevel databases. It builds on our previous work in secure distributed database management systems and on the results of others' work in FDSs. In particular, we define an MLS/FDS and discuss issues on multilevel data distribution, heterogeneity, autonomy, security policy, and architecture.

The organization of this paper is as follows. In Section 2, we define an MLS/FDS and describe the major characteristics of such a system. These characteristics are multilevel data distribution, heterogeneity, and autonomy. In Section 3, we discuss security policy issues. In particular, we identify the various components of a security policy for an MLS/FDS, discuss schemes for generating and enforcing the policy, and state what the contents of the policy should be. In Section 4, we discuss architectural issues. In particular, issues on a schema architecture and a system architecture are given. The paper is summarized in Section 5.

2. What is an MLS/FDS?

In Section 2.1, we provide an overview of an MLS/FDS. Its characteristics are described in Section 2.2. In particular, issues on data distribution, heterogeneity, and autonomy are discussed.

2.1. Overview

We define a Multilevel Secure Federated Database System (MLS/FDS) to be a collection of cooperating and possibly autonomous and heterogeneous multilevel database systems (MLS/DBS). An MLS/DBS consists of a set of modules called a multilevel secure database management system (MLS/DBMS) and a collection of multilevel databases that it manages. A multilevel database is a database in which not all of the data are assigned the same sensitivity level (also called security level). An MLS/DBMS ensures that users cleared at different security level access and share the multilevel database so that they obtain only the data classified at or below their level.

The MLS/DBSs that constitute an MLS/FDS are called the component MLS/DBSs of the MLS/FDS. These components may be integrated to varying degrees. The set of modules that securely interconnects these component MLS/DBSs is called a multilevel secure federated database management system (MLS/FDBMS). Note that a component MLS/DBS could be centralized, distributed, or even another MLS/FDS. Figure 1 illustrates an MLS/FDS. In this figure, the MLS/FDBMS integrates three component MLS/DBSs. Component 1 consists of a centralized MLS/DBMS and the associated multilevel database. Component 2 is an MLS/DDBMS and the associated multilevel databases. Component 3 is another MLS/FDS.

The component MLS/DBSs of an MLS/FDS form a federation. In a totally autonomous environment, the components may join or leave the federation whenever they want to. In a less autonomous environment, additional restrictions may be enforced. A desired feature for participating in a federation is for the local operations of a component to be unaffected. A component may be part...
of just one federation or of multiple federations. Each component has a Database Administrator (DBA) and/or System Security Officer (SSO) associated with it. Ideally, it should be up to the DBA/SSO to determine when its component should join or leave a federation. That is, the DBA/SSO should have control of the databases that he manages. In addition to the DBA/SSO, each component has a Designated Approving Authority (DAA) who accredits the system. At the global level, it is possible to have one or more DBAs, SSOs, and DAAs for managing the federation.\(^2\)

The components of an MLS/FDS need not necessarily be homogeneous. For example, the three component MLS/DBSs of the MLS/FDS shown in Fig. 1 could be identical, centralized MLS/DBSs. In this case, the MLS/TDS is homogeneous. On the other hand, component 1 could utilize an MLS/DBMS such as Secure SQL Data Server (a product of Sybase, Inc.); component 2 could utilize Trusted Informix (a product of Informix, Inc.), and component 3 could utilize an MLS object-oriented DBMS. In this case, the components are heterogeneous, and the MLS/FDBMS is the collection of modules that securely interconnects the heterogeneous database systems.\(^3\)

Two approaches have been proposed for developing an FDS. The first approach, called the bottom-up approach, is used for integrating existing DBS components to form an FDS. The steps involved include translating component schema into generic schema, defining export schema, integrating the export schemas into a federated schema, and, finally, defining appropriate external schemas. In a multilevel environment, a similar process needs to be carried out for security policies also. That is, a component security policy must first be transformed into a generic policy. Then an export policy

\(^2\)It has been suggested to us that an MLS/FDS should be a federated system that provides multilevel security. As such, it should also include components which are system-high database systems and single-level database systems. We have adopted a more restrictive definition of an MLS/FDS. Future research will include examining the broader definition.
must be defined. The various export policies must be integrated to form the federated policy.\(^4\)

In the second approach, called the top-down approach, an FDS is built from scratch. That is, it is not assumed that component DBSs exist. Based on user requirements, a federated schema is generated initially, and then various components are integrated into the system. The components may be selected from products already available, or developed from scratch. The components selected will depend on the user requirements. If a new user requirement has to be supported, then the various schemas are analyzed and appropriate modifications are made. In a multiple environment, a federated security policy is developed first, based on user requirements. Then the component MLS/DBSs are selected and appropriate component policies are developed. Any change in the real-world could result in modifications to the federated policy and, consequently, the component policies.

The approach that is selected will depend mainly on whether the MLS/FDS connects existing MLS/DBSs or whether the MLS/FDS has to be designed from scratch. As MLS/DBS components become available, a bottom-up approach may be needed for ensuring secure interoperability between them. In this paper, we focus on the bottom-up approach to developing an MLS/FDS.

2.2. Characteristics of an MLS/FDS

The characteristics of an MLS/FDS are multilevel data distribution, heterogeneity, and autonomy. Multilevel data distribution issues have been discussed in two earlier articles we published in this journal \([8, 9]\). In particular, we discussed issues on fragmentation and replication of multilevel data and showed how polyinstantiation could be handled in a distributed environment. The concepts and techniques discussed in these articles can be applied to a federated database system, subject to the restrictions imposed by the export policies of the schema. We shall address some of the essential points in Section 4 when we describe schema integration and translation issues. In this section we focus mainly on heterogeneity and autonomy.\(^5\)

2.2.1. Heterogeneity

There are various types of heterogeneity that need to be addressed for an MLS/FDS. Some types of heterogeneity are present in a non-multilevel environment. Others arise due to multilevel security. In this section, we discuss the issues involved in interconnecting heterogeneous components as identified in \([11]\) and discuss the security impact for each issue. We shall also discuss some of the additional security concerns.

1. Schema (or data model) heterogeneity: Not all of the databases in a heterogeneous architecture are 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 data model into those of another are being developed. In a multilevel secure environment, the individual data models can be multilevel. That is, a data model has multilevel security constructs incorporated into it. Therefore, the constructs of one multilevel data model have to be transformed into those of another. During the translation process, it should be ensured that a user who does not have access to a particular entity with respect to one data model must not have access to the same entity with respect to a different data model.

The object-oriented approach is also being investigated for handling schema heterogeneity. The idea here is for the users to have a generic view of the

\(^4\)The various parts of the security policy and schema for an MLS/FDS will be discussed in Sections 3 and 4, respectively.

\(^5\)Some of the issues were presented in a panel discussion by the author at the 5th IFIP Working Conference on Database Security in Shepherdstown, West Virginia, November 1991. A discussion of heterogeneous database integration issues is also given in \([10]\).
entire system. Translators are then necessary to translate the constructs from/to the generic representation to/from an individual representation. If the object-oriented approach is to be utilized in a multilevel environment, then a multilevel secure object-oriented data model has to be developed for the generic representation.6

2. Transaction processing heterogeneity: Different DBMSs may utilize different algorithms for transaction processing. Work is being directed toward integrating the various transaction processing mechanisms. For example, techniques which integrate locking, timestamping, and validation mechanisms are being developed. It has been noted that the standard concurrency control algorithms have security vulnerabilities. For example, a locking mechanism could be used to covertly signal information from a higher-level process to a lower-level one. As a result, various concurrency control mechanisms have been adapted to function in a multilevel environment. However, integration of these adapted concurrency control algorithms to function in a heterogeneous environment needs to be investigated.

3. Query processing heterogeneity: Different DBMSs utilize different query processing and optimization strategies. One of the research areas here is to develop a global cost model for distributed query optimization. In a multilevel environment, the individual cost models may depend on the security policy enforced, the storage mechanism used, and the amount of data classified. The global cost model could then depend on the global security policy that is enforced.

4. Query language heterogeneity: Different DBMSs will utilize different query languages. Even if the DBMSs are based on the relational model, one could use SQL and the other could use relational calculus. Standardization efforts are underway to develop a uniform interface language. The security impact on these efforts needs to be investigated. For example, various extensions to SQL are being proposed to handle security constructs. Different designs are proposing different extensions. These extensions need to be integrated so that a uniform interface is provided to the user.

5. Constraint heterogeneity: Different DBMSs enforce different integrity constraints which are often inconsistent. For example, one DBMS could enforce a constraint that all employees must work at least forty hours while another DBMS may not enforce such a constraint. In a secure environment, additional constraints such as discretionary security constraints and mandatory security constraints may be enforced. The constraints enforced at different components may be conflicting. These differences need to be reconciled.

6. Semantic heterogeneity: Data may be interpreted differently at different components. For example, the entity address could mean just the country at one component while another component could interpret it to be the number, street name, city name, and country. It has been recognized that semantic heterogeneity is very difficult to handle [12]. The problems are even worse in a multilevel environment. For example, a security label at one component could mean something different at another component. Standardization efforts are needed to resolve such inconsistencies.7

7. Other security issues: In addition to the security impact on the various issues identified for heterogeneous database systems, there are some additional security concerns. These include the following:

(a) Different security policies: Each DBMS could enforce its own security policy for mandatory as well as discretionary security. In addition, different

It has been pointed out to us that handling label heterogeneity is a network security issue. More work needs to be done to determine whether any other component of the MLS/FDS should also handle label heterogeneity.
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 a ‘read at or below your level’ and ‘write at or above your level’ policy. The different policies have to be integrated in order to provide a global security policy.

(b) 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. Furthermore, one system could support the representation of cover stories by enabling two different tuples with the same primary key to exist at different security levels while the other system may not permit the entity integrity property to be violated. Such differences may need to be handled at the global level.

c) Different classifications of the same entity: An entity could be classified at the Secret level at one node and yet be classified at the TopSecret level at the second node. If this is the case, then the global policy should resolve such inconsistencies. One can regard this as a form of polyinstantiation.

d) Different semantics of classification levels: A classification level at one node could mean something entirely different at another node. This is a form of semantic heterogeneity. The global security policy may have to resolve such inconsistencies.

e) Different accreditation ranges: One node could handle the range from Unclassified to Secret while another node could handle the range Confidential to TopSecret. If a TopSecret user needs to access the Unclassified information handled by the first node, then it should be ensured that information from the TopSecret user is not transmitted covertly into the node handling the Unclassified data. Much research needs to be done if solutions are to be provided for the secure interoperability of heterogeneous database systems. The problem becomes even more complex if there are several types of heterogeneity. For example, the various systems could not only enforce different security policies but also utilize different multilevel data models. We feel that any solution to the problem in the near term should be customized. That is, the customers’ requirements should be identified and solutions to satisfy the requirements need to be provided.

2.2.2. Autonomy

The components of an MLS/FDS may have some degree of autonomy. The DBA/SSO of each component could decide who has access to the data he controls. There are various types of autonomy that a component could exhibit. They include: communication autonomy, execution autonomy, association autonomy, and design autonomy. We discuss each type.

Communication autonomy: Total communication autonomy implies that a component will determine with whom it wishes to communicate. There is an additional restriction in a multilevel environment, as a machine handling only Secret and TopSecret data cannot send data to a machine accredited to process only Unclassified and Confidential data.

Execution autonomy: Total execution autonomy implies that the local operations of a component

\[\text{\footnotesize {It has been pointed out to us that a cascading problem could occur when the nodes handle different accreditation ranges. For example, node 1 handles the range Unclassified to Confidential, node 2 handles the range from Confidential to Secret, and node 3 handles the range from Secret to TopSecret. Suppose a TopSecret user poses a query at node 3 and data from nodes 1 and 2 are needed for the response. Then the query is downgraded and sent to node 2 via a Secret connection and from node 2 it is downgraded and sent to node 1 via a Confidential connection. Some TopSecret could then be covertly passed from node 3 to the Unclassified level at node 1 via node 2.}}\]
are not affected by the federated users in any way. This could cause a problem in a multilevel environment. For example, an Unclassified federated user might issue the execution of a transaction at a particular component. If there is already a Secret local transaction waiting to be executed at that component, total autonomy would mean that the Unclassified transaction must wait. That is, the actions of a higher level user have interfered with a lower level one. This could cause a potential covert channel.

**Association autonomy:** Total association autonomy implies that a component can decide when and what information to share with the others. In the case of a federated environment, it can also decide when to join a federation, when to leave a federation, and which federations to join. Data replication would conflict with association autonomy. For example, if data managed at component 1 is replicated at component 2, then if component 1 decides to leave the federation, the data at component 2 should also be removed. However, for security reasons, one might want to replicate lower-level data at higher levels. This constrains association autonomy.

**Design autonomy:** Total design autonomy would imply that each component will have the ability to choose its own design. For example, it could determine (1) the multilevel data to be managed, (2) the security policy to be enforced, (3) the query processing and transaction management algorithms to be used, and (4) the semantic interpretation of data and labels. One of the difficulties in developing an MLS/FDS is the integration of different security policies. Different policies exist due to design autonomy.

In addition to the various issues discussed here, one of the major concerns of a secure environment is accreditation. If the components have autonomy, then the individual DAAs will have the freedom to accredit their own systems. However, in order to accredit the MLS/FDS as a whole, there must be some negotiation between the different DAAs and the federated DAA. That is, we introduce an additional type of autonomy called ‘Accreditation autonomy’.

3. Security policy issues

In this section, we discuss issues on developing a security policy for an MLS/FDS. In Section 3.1, we describe a security policy architecture for an MLS/FDS. In Section 3.2, we discuss issues on generating the security policy. In Section 3.3, using an example, we discuss how the security policy may be enforced. In Section 3.4, we discuss the contents of a security policy for an MLS/FDS.

3.1. Security policy architecture

Figure 2 illustrates our view of a security policy architecture for an MLS/FDS. Each component enforces its own security policy. The policy will specify the various subpolicies for mandatory security, discretionary security, integrity, identification and authentication, and auditing. The component policy itself may be a combination of multiple security policies where each individual policy will be that of a local MLS/DBS which is part of the component.

Each component security policy is specific to the component and is specified in a language that is chosen by the DBA/SSO of that component. However, in order to facilitate the integration of multiple policies, each component policy must be transformed into a generic component policy. This generic policy is identical to the component policy but is specified in a generic language. All components that wish to be part of the federation must use the same language for the generic policy. The generic language may be determined by negotiation between the component DBA/SSOs and the federated DBA/SSO.9

9Generic representations have played an important role in interconnecting different computing systems. We believe that they will also be useful for specifying security policies. This will also facilitate the task of integrating different security policies.
Once the generic policy is specified, the DBA/SSO of a component may want to impose more severe restrictions on the federated user. That is, a local user may have more privileges than a federated user. Therefore, the DBA/SSO must generate an export policy which may be different from the generic policy. The export policy is also specified in the generic language. For example, the DBA/SSO might want only the federated users cleared at the TopSecret level to have access to his data classified at the Secret level. Even when generating the export policy, there may have to be some form of negotiation between the component DBA/SSOs and the federated DBA/SSO.

Once the export policy is generated, the component DBA/SSO no longer has any control. It is now up to the DBA/SSO of the MLS/FDS to combine all of the export policies and then generate a federated policy. The federated policy must include all restrictions enforced by the individual DBA/SSOs. In addition, the DBA/SSO of the MLS/FDS may impose some additional restrictions such as discretionary access rules for various groups of federated users. It is usually the responsibility of the DBA/SSO of the MLS/FDS to differentiate between various classes of federated users.

In Fig. 2, we assume that there are three components: A, B, and C. A and B form a federation while B and C form a second federation. That is, B is involved in multiple federations. Although it need not be the case, we have assumed that both federations use the same generic language for representing policies. Also, component A is an MLS/DBS which consists of two centralized MLS/DBSs. The two local policies will be combined to form the policy for component A.

---

In general, as far as a component is concerned, there are only two types of user: local and foreign. Foreign users are the federated users.
3.2. Generating the security policy

Figure 3 illustrates the various processors which transform one security policy into another. If the security policy is static, then all of the transformations can be done before the system is in operation. However, if the environment is dynamic, then the various transformations may have to be done during system operation. These transformations need to be performed by a subject acting on behalf of the DBA/SSO. The transformation process can be quite complex, especially if the application specific policy is complex. Also, in a federated environment, each component may decide to join or leave a federation at any time. This would mean that the federated policy must be modified whenever such a decision is made.

A transforming processor transforms a component policy to a generic component policy. No additional information is generated during this transformation process. However, it must be ensured that the transformation is complete. A transforming/filtering processor examines the generic component policy, and may remove information that
is not relevant to the federated user, and may transform or add other information. For example, all of the schemas may not be exported and therefore any policy rules specific to the eliminated schemas are removed. Also, additional policies rules may be enforced for the federated user. That is, the transforming/filtering processor may impose additional restrictions. The output of the transforming/filtering processor is the export policy. A constructing processor integrates all of the export policies and generates the federated policy. The information required by the transforming, filtering, and constructing processors to carry out their operations is separate from the policy information.

Maintaining the consistency of a component security policy is the responsibility of the DBA/SSO of that component. The transforming and filtering processors of a component must ensure that consistency is maintained when the generic and export policies are generated. Maintaining the consistency of the federated policy is the responsibility of the federated DBA/SSO. The constructing processor, which generates the federated policy from the export policies, must resolve any inconsistencies between the different export policies.

We believe that generating a security policy for an MLS/FDS is a difficult task. Furthermore, not much is known about integrating security policies of different systems. In addition, network security issues will also be of concern as the different components are usually connected via a communication network.

3.3. Enforcing the security policy
In this section, we illustrate with an example how the security policy for an MLS/FDS may be enforced. Suppose the components 1, 2, and 3 form a federation, as shown in Fig. 4. Assuming that the labels are hierarchical, suppose a federated user U, cleared to read data assigned label A or below, requests to read data about a particular mission. A label consists of a security level (such as Unclassified or Secret) and one or more categories (such as NATO, NOFORN).
B or below at component 1, data labeled C or below at component 2, and data labeled D or below at component 3, where $A \geq B \geq C \geq D$. Such decisions are made when the component DBA/SSOs export their security policies.

The first step will be for U to be authenticated as a federated user. This authentication mechanism must be trusted. There must be some form of a global trusted computing base (TCB), which we will call Federated TCB, which is responsible for the authentication (see Fig. 5). The next step will be to generate the query execution strategy. It will be determined that the query has to be routed to all three components. Up to this point, the processing is done in a domain associated with label A, and the access is controlled by the Federated TCB. However, the query must now be processed in domains associated with labels B, C, and D at components 1, 2, and 3, respectively. Therefore, the federated process (acting on behalf of U) must now log in as a foreign user to the MLS/DBMSs associated with components 1, 2, and 3. The processes which act on behalf of U at components 1, 2, and 3 execute in domains associated with the labels B, C, and D, respectively. The authentication mechanism of the component MLS/DBMS must validate the login process.\footnote{Some parts of the federated query processor may need to be trusted. This is because it must be ensured that the component MLS/DBMSs process the query at the correct levels. More research needs to be done before the security critical components can be identified.}

Access is now controlled by the component TCBs until the information gathered by the components is given to the federated process which acts on behalf of the federated user. In this example, information is gathered from all three components. The response is assembled by the federated process and then given to the user.

There are several issues that need to be investigated. For example, the interactions between the federated TCB and the component TCB should be determined. In this example, a federated process executing in a domain associated with label A must now communicate with processes of components 1, 2, and 3 which execute in domains associated with labels B, C, and D, respectively. It must be ensured that there is no covert information flow during this communication. Note that, if a federated process can communicate with the processes at components 1, 2 and 3, then it is possible for information to flow between these component processes. Should such flows be permitted? That is, do we assume then that process running in a domain associated with label B can send data to a process running in a domain associated with label C, where $B \geq C$? Much research needs to be done in order to answer the questions that we have posed here.\footnote{It has been pointed out to us that authenticating a user at both the federated and component levels may not be desirable. This issue needs to be investigated further.}

3.4. Contents of the security policy

A security policy, whether it is a component policy, generic policy, or federated policy, must state the policies for mandatory security, discretionary security, integrity, identification and authentication, accounting, and auditing, among others. Some of these policies, such as integrity, are not yet well

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Component & TCB for Component 1 & TCB for Component 2 & TCB for Component 3 \\
\hline
\end{tabular}
\end{table}

Fig. 5. A possible security architecture.
defined even for a centralized MLS/DBS. We believe that much work remains to be done before satisfactory policies can be developed even for homogeneous MLS/DDBSs. While heterogeneous MLS/DDBSs are more complex than homogeneous ones, when we include autonomy into the environment the problems are even more complex because the policies will vary depending on the components joining and leaving the federation.

Some of the essential points of mandatory access control were discussed in Section 3.3. According to the TCSEC [13], the TCB should enforce mandatory access control over all resources. In a federated environment, we believe two types of TCB are needed. One is the federated TCB which controls access to the global resources (such as global objects, federated subjects, etc.), and the other is a local TCB for each component. If a component is distributed, then the local TCB itself will be composed of multiple TCBs. The security levels supported by the federated TCB may be different from those supported by the component TCBs.

The application-independent policy should specify the type of access that a subject has to an object. In general, a subject has read-access to an object if the subject's security level dominates the security level of the object, and a subject has write-access to an object if the subject's security level is dominated by the level of the object. Many MLS/DBMSs enforce a write policy where a subject has write-access to an object only if the subject's level is equal to that of the object.

In MLS/DBMSs, discretionary access control is enforced by controlling access to named objects by named users. As in the case of mandatory access control, there are two levels of discretionary access control: one is at the global (or federated) level and the other is at the local level. There has been some work on discretionary access control for federated database systems [14]. The mechanisms proposed could be incorporated into a multilevel environment. For example, the filtering processors shown in Fig. 6 could control access. The filtering processor between the MLS/FDBMS and a component
MLS/DBMS controls access to the component objects by federated users. The filtering processor between the user and the MLS/DBMS controls access to the federated objects by the federated users.

User identification and authentication also occurs at two levels. One is at the global level, where a federated user is authenticated by the federated TCB. The user then has mandatory and discretionary access to the global objects depending on the federated security policy enforced. The other is at the component level. That is, before accessing the component objects, the user has to be authenticated to access the component. The component TCB is responsible for this authentication. As far as a component is concerned, users are either local or global. The export policy will specify whether a particular federated user is permitted to access the component objects and at what level. This export policy is incorporated into the federated policy.\(^{16}\)

Audit information must be maintained at two levels: the global level and the component level. Other issues that a policy should address include object reuse, labels, support for trusted path, integrity, and assurance.

In addition to the security policies for the component systems and the federated system, a complete security policy for an MLS/FDS should include a policy for the network that interconnects the various component MLS/DBSs. In a federated environment, the network itself may be heterogeneous. We believe that developing a security policy for a heterogeneous network is a complex task, and therefore we will not address it here. In addition, an investigation of network security issues specific to supporting an MLS/FDS must also be carried out.\(^{17}\)

The security policy for a component MLS/FDS will include the security policy for the operating system and the security policy for the DBMS. When the TCB for the operating system and the TCB for the network are taken into consideration, the TCB for the MLS/FDS illustrated in Fig. 5 could possibly be extended to the one illustrated in Fig. 7. The TCB of the entire system could be referred to as the composite TCB. By dividing the TCB into smaller parts, we envisage that the technique of 'evaluation by parts' described in the Trusted Database Interpretation \(^{16}\) could be used to evaluate an MLS/FDS. An investigation of the issues involved in evaluating an MLS/FDS has yet to be carried out.

4. Architectural issues

In Section 3, we discussed security policy issues as well as issues related to a security architecture for an MLS/FDS. In this section, we discuss issues on maintaining the schema for an MLS/FDS and system architectural issues.

4.1. Schema architecture

One could adapt the five-schema reference architecture discussed in ref. [1] for a multilevel environment. Each component has a component schema which describes the schema associated with the multilevel database. The component schema is transformed into generic schema. The generic schema of an MLS/FDS must be specified in a generic language. The entire generic schema need

<table>
<thead>
<tr>
<th>Federated TCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCB for Component 1 = DBMS TCB + OS TCB</td>
</tr>
<tr>
<td>TCB for Component 2 = DBMS TCB + OS TCB</td>
</tr>
<tr>
<td>TCB for Component 3 = DBMS TCB + OS TCB</td>
</tr>
</tbody>
</table>

Network TCB

Fig. 7. Composite TCB.

\(^{16}\)As mentioned earlier, the issue of authenticating a user at both federated and component levels needs to be investigated further.

\(^{17}\)A useful starting point for network security issues is ref. [15].
not be exported. For example, a DBA/SSO may not want to make all relation names available to a federated user. Therefore, the export schema includes only the schema that is exported by the component DBA/SSO. Various export schemas are combined to form the federated schema. Again, the entire federated schema need not be available for various classes of users. Therefore, the federated schema is decomposed into multiple external schemas.

In our work on MLS/DDBMS [8], which is based on the non-federated approach, we have specified three ways to store the schema. In the first method, the schema is stored at system-low, in which case the schema must also be classified at system-low. All users can read the schema. The schema can be updated by a process at system-low or by a trusted process. In the second method, the schema is stored at system-high. A trusted process is needed to manage the schema. When a user requests schema information, the schema manager must examine the security levels of the schema and ensure that only the schema classified at or below the user's level is retrieved. In the third method, the schema is stored at multiple security levels. We assume that the schema classified at level L is stored at level L also. In this case, a process operating at level L manages the schema stored at level L. We also assume that the schema manager operating at a level L at component I communicates with the schema manager operating at level L at component J in order to retrieve a remote schema.

In a multilevel federated environment, schema management is more complex than in a non-federated environment. For example, due to label heterogeneity, a component schema stored at level L may not be accessed by a federated user cleared at level L. Furthermore, it may not be possible for a schema manager of component 1 operating at level L to communicate with the schema manager of component 2 operating at level L. Since various types of schema must be maintained (such as component schema, generic schema, federated schema, and external schema), different transformers are necessary to transform one type of schema into another. The processors that are necessary to change one schema type into another are described in detail in [1]. The transforming processor simply transforms one representation schema into another. The filtering processors filter out irrelevant schema information. The constructing processor integrates the different schemas. If the schema is static, then the various processors operate during the design time. However, in a federated environment, each component may decide to join or leave a federation dynamically. Therefore, the schema evolution is an important issue that needs to be addressed for such an environment.

Assuming that the schemas are multilevel (that is, schemas are designed different security levels) and that a manager at level L manages the schema at level L, we discuss the trust that must be placed on the processors of the five-schema architecture. Therefore, the component schema at level L can be transformed into the generic schema at level L by a processor at level L. The filtering processor between the generic schema and export schema may filter some information from the generic schema, possibly for security reasons, in order to generate the export schema. That is, not all relations may be accessed by federated users. The filtering processor must ensure that relevant information is removed from the generic schema to form the export schema. The filtering processor needs information on the security policy to carry out its tasks.

The export schemas are then integrated into the federated schema by the constructing processor.

---

Footnote: Note the analogy between the security policy architecture and the schema architecture with respect to the local, component, generic, and federated policies and schemas. The policy architecture does not have an external policy. It has been pointed out to us that such a policy might also be needed. In addition, as stated earlier, in the security policy architecture we have used a transforming/filtering processor to construct the export policy from the generic policy, whereas in Sheth and Larson’s schema architecture [1] a filtering processor is used to construct the export schema from the generic schema.
However, if a federated user at level $L^*$ has access only to level $L$ information at a component where $L^* \neq L$, then the component export schema at level $L$ should be part of the federated schema at level $L^*$. The constructing processor must ensure that the correct levels are assigned to the federated schema. In order to perform its functions, the constructing processor needs information on the security policy. Finally, a second filtering processor filters information from the federated schema to generate the appropriate external schemas for different classes of users.

Maintaining the consistency of a component schema is the responsibility of the DBA/SSO of that component. The transforming and filtering processors at a component must ensure that consistency is maintained when generating the generic and export schemas. Maintaining the consistency of the federated schema is the responsibility of the federated DBA/SSO. The constructing processor, which generates the federated schema from the export schemas, must resolve any inconsistencies between the different export schemas.

Since an MLS/FDS is also a distributed system, forming views from the distributed database will be restricted by the export schemas. Note that in our earlier work on multilevel data distribution, we assumed that all of the schemas will be exported at the global level and that subsequently the global views are formed. However, in a federated environment this may not be the case. Therefore, global views can be formed only from the schemas that are available. For example, if nodes 1 and 2 both have an employee fragment and a department fragment and node 2 exports only its employee fragment, then the department fragment of node 2 cannot be part of any global view.

4.2. System architecture

The system architecture will describe the modules of an MLS/FDS. These include a user interface manager, query processor, transaction manager, external and federated schema managers. The user interface manager is responsible for authenticating the federated user. It has access to the external schema manager for enforcing the discretionary access rules. It also parses a user request and gives it to the query/transaction processor. The query processor is responsible for query optimization and for generating the execution strategy. Schema information is supplied by the federated schema manager. The transaction manager is responsible for query and update requests issued as part of a transaction.

The execution strategy is carried out by the distributed execution manager by communicating with the various components. A component MLS/DBS is accessed via the component interface manager. The component interface manager passes requests from the distributed execution manager to the component/federated user interface manager. The component/federated user interface manager authenticates the user at the component level. For example, if a federated user cleared to read data labeled $A$ or below has read-access to data labeled $B$ ($B < A$) or below at a component, then it must be ensured that the request is processed by the component MLS/DBMS in a domain associated with the label $B$. The request is given to the component query processor/transaction manager for execution strategy generation. Access to the component multilevel database is via the component storage manager.

While we have identified some of the modules that must be trusted, more work needs to be done before all of the security-critical components can be identified. A security architecture for an MLS/FDS has yet to be designed. In addition, the interactions between the federated TCB, which controls access to objects at the global level, and the...
component TCB, which controls access to the component objects, need to be determined.

5. Conclusions

This paper first defines an MLS/FDS and describes the characteristics of such a system. These characteristics include multilevel data distribution, heterogeneity, and autonomy. Various types of heterogeneity are identified. These include heterogeneity with respect to schemas, query processing, transaction processing, semantics, and security policies. The conflicts between autonomy and security are also identified. Next, security policy issues for an MLS/FDS are described. First, various types of security policy are identified. These include local policies, component policies, generic policies, export policies, and federated policies. Ways of generating the security policies are described. Issues on enforcing the security policies are discussed. Finally, contents of each type of security policy are identified. Architectural issues are described. A reference schema architecture for an MLS/FDS is discussed and ways of generating the schemas are described. A system architecture of an MLS/FDS and the component MLS/DBMSs are described.

From the discussion in the previous subsection, it should be clear that the steps to developing an MLS/FDS are by no means straightforward: we believe that some of them are nearly impossible, given the current state of the art in both federated DBMS technology and DBMS security technology. We also believe that, unless considerable advances are made to solve the problems in non-multilevel FDSs, it will not be prudent to embark on a fully-fledged program to develop an MLS/FDS. From the difficulties described above and from current trends, the traditional approach to developing a secure system, such as developing a security policy, model, design specification and security architecture, and formal top-level specification and verification, may not be appropriate for an MLS/FDS. Therefore, until global solutions to interconnecting heterogeneous database systems 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.

We propose the following two-way approach for R&D in MLS/FDS:

(1) Investigate solutions to handling limited forms of heterogeneity and autonomy. These solutions will depend on customer needs and developments in MLS/DBMS and FDS technologies. Series of prototypes, increasing in complexity, should be developed first to test the policy and algorithms before developing an operational system. In addition, as progress is made in network security technology, the results should be integrated into the prototypes.

(2) Monitor progress in the developments of FDSs and simultaneously investigate ways of incorporating security into these systems.

We believe that such a two-way approach would not only satisfy the immediate needs of customers but also provide the direction for developing useful MLS/FDSs in the future.

Acknowledgments

The work reported in this paper was sponsored by the Department of Defense. We gratefully acknowledge the support of the Office of INFOSEC Computer Science. We thank Deborah Bodeau, Rae Burns, Richard Graubart, Dale Johnson, Mark Nadel and Arnon Rosenthal for their comments on this paper. We acknowledge the paper on non-multilevel federated database systems by Amit Sheth and James Larson (published in ACM Computing Surveys in September 1990). Much of our work on MLS/FDS has been influenced by their paper.
References


Bhavani Thuraisingham is a lead engineer with the MITRE Corporation where she is working in real-time data base systems, massive database management, and database security. Her interests also include heterogeneous database integration, multimedia information systems, and object-oriented design and analysis techniques for developing various information systems applications. Her previous work at MITRE includes the design and implementation of a secure distributed database system, database inference controller, and a secure multimedia/object-oriented database system. She is a co-director of MITRE’s Database Specialty Group and serves in the Corporate Technology Area Council in Database Systems.

Prior to joining MITRE, Dr Thuraisingham conducted research and development activities at Honeywell Inc., where her work included the design of the secure database system Lock Data Views, the design of a network operating system for space station applications, and the application of object-oriented technology for developing next-generation process control systems and for integrating heterogeneous data dictionaries. Before that she was at Control Data Corporation where she worked on the product development of CDCNET. She was also an adjunct professor of computer science and a member of the graduate faculty at the University of Minnesota. Her work has been published in over forty journal papers and numerous other conference/workshop papers.

Dr Thuraisingham gives tutorials in object-oriented database systems, heterogeneous database systems, secure database systems, and real-time database systems. She has co-edited a book on secure database systems for North-Holland and one on secure object-oriented systems for Springer, serves on editorial boards of the Journal of Computer Security and the Computer Standards and Interface Journal, and has served as the program chair or as program committee member for several conferences. Dr Thuraisingham received the MS degree in Computer Science from the University of Minnesota, the MSc degree in Mathematical Logic from the University of Bristol, UK, and the PhD degree in Computability Theory from the University of Wales, Swansea, UK.