1. Introduction to Research Work

Unlike parallel system, in which the process are tightly coupled and form a single database system, a distributed database system consists of loosely coupled sites that share no physical components. The database systems that run on each site may have a substantial degree of mutual independence. As distributed networks become more popular, the need for improvement in distributed database management systems becomes even more important. A distributed system varies from a centralized system in one key respect: The data and often the control of the data are spread out over two or more geographically separate sites. Distributed database management systems are subject to many security threats additional to those present in a centralized database management system (DBMS). Furthermore, the development of adequate distributed database security has been complicated by the relatively recent introduction of the object-oriented database model. This new model cannot be ignored. It has been created to address the growing complexity of the data stored in present database systems. For the past several years the most prevalent database model has been relational. While the relational model has been particularly useful, its utility is reduced if the data does not fit into a relational table. Many organizations have data requirements that are more complex that can be handled with these data types. Multimedia data, graphics, and photographs are examples of these complex data types. Relational databases typically treat complex data types as BLOBs (binary large objects). For many users, this is inadequate since BLOBs cannot be queried. In addition, database developers have had to contend with the impedance mismatch between the third generation language (3GL) and structured query language (SQL). The impedance mismatch occurs when the 3GL command set conflicts with SQL. There are two types of impedance mismatches: (1) Data type inconsistency: A data type recognized by the relational database is not recognized by the 3GL. For example, most 3GLs don’t have a data type for dates. In order to process date fields, the 3GL must convert the date into a string or a Julian date. This conversion adds extra processing overhead. (2) Data manipulation inconsistency: Most procedural languages read only one record at a time, while SQL reads records a set at a time. This problem is typically overcome by embedding SQL commands in the 3GL code. Solutions to both impedance problems add complexity and overhead. Object-oriented databases have been developed in response to the problems listed above: They can fully integrate complex data types, and their use eliminates the impedance mismatch. The development of relational database security procedures and standards is a more mature field than for the object-oriented model. The relative immaturity of the object-oriented model is particularly evident in distributed applications. Inconsistent standards are an example: Developers have not embraced a single set of standards for distributed object-oriented databases, while standards for relational databases are well established. We will review the security concerns of databases in general and distributed databases in particular. We will examine the security problems found in both models, and we will examine the security problems unique to each system. Finally, we will compare the relative merits of each model with respect to security.
1.1. Elements of Distributed Database Management System Security

*General Database Security Concerns*

The distributed database has all of the security concerns of a single-site database plus several additional problem areas. We begin our investigation with a review of the security elements common to all database systems and those issues specific to distributed systems.

A secure database must satisfy the following requirements
1. It must have physical integrity,
2. It must have logical integrity,
3. It must be available when needed,
4. The system must have an audit system,
5. It must have elemental integrity (accurate data),
6. Access must be controlled to some degree depending on the sensitivity of the data,
7. A system must be in place to authenticate the users of the system, and
8. Sensitive data must be protected from inference.

The following discussion focuses on requirements 5-8 above, since these security areas are directly affected by the choice of DBMS model. The key goal of these requirements is to ensure that data stored in the DBMS is protected from unauthorized observation or inference, unauthorized modification, and from inaccurate updates. This can be accomplished by using access controls, concurrency controls, updates using the two-phase commit procedure (this avoids integrity problems resulting from physical failure of the database during a transaction), and inference reduction strategies.

The level of access restriction depends on the sensitivity of the data and the degree to which the developer adheres to the principal of least privilege. Typically, a lattice is maintained in the DBMS that stores the access privileges of individual users. When a user logs on, the interface obtains the specific privileges for the user.

Access permission may be predicated on the satisfaction of one or more of the following criteria: (1) Availability of data: Unavailability of data is commonly caused by the locking of a particular data element by another subject, which forces the requesting subject to wait in a queue. (2) Acceptability of access: Only authorized users may view and or modify the data. In a single level system, this is relatively easy to implement. If the user is unauthorized, the operating system does not allow system access. On a multilevel system, access control is considerably more difficult to implement, because the DBMS must enforce the discretionary access privileges of the user. (3) Assurance of authenticity: This includes the restriction of access to normal working hours to help ensure that the registered user is genuine. It also includes a usage analysis which is used to determine if the current use is consistent with the needs of the registered user, thereby reducing the probability of a fishing expedition or an inference attack.
Concurrency controls help to ensure the integrity of the data. These controls regulate the manner in which the data is used when more than one user is using the same data element. These are particularly important in the effective management of a distributed system, because, in many cases, no single DBMS controls data access. If effective concurrency controls are not integrated into the distributed system, several problems can arise. Three possible sources of concurrency problems: (1) Lost update: A successful update was inadvertently erased by another user. (2) Unsynchronized transactions that violate integrity constraints. (3) Unrepeatable read: Data retrieved is inaccurate because it was obtained during an update. Each of these problems can be reduced or eliminated by implementing a suitable locking scheme (only one subject has access to a given entity for the duration of the lock) or a timestamp method (the subject with the earlier timestamp receives priority).

Special problems exist for a DBMS that has multilevel access. In a multilevel access system, users are restricted from having complete data access. Policies restricting user access to certain data elements may result from secrecy requirements, or they may result from adherence to the principal of least privilege (a user only has access to relevant information). Access policies for multilevel systems are typically referred to as either open or closed. In an open system, all the data is considered unclassified unless access to a particular data element is expressly forbidden. A closed system is just the opposite. In this case, access to all data is prohibited unless the user has specific access privileges.

Classification of data elements is not a simple task. This is due, in part, to conflicting goals. The first goal is to provide the database user with access to all non-sensitive data. The second goal is to protect sensitive data from unauthorized observation or inference. For example, the salaries for all of a given firm's employees may be considered non-sensitive as long as the employee's names are not associated with the salaries. Legitimate use can be made of this data. Summary statistics could be developed such as mean executive salary and mean salary by gender. Yet an inference could be made from this data. For example, it would be fairly easy to identify the salaries of the top executives.

1.2. Security Problems Unique to Distributed Database Management Systems
1.2.1 Centralized or Decentralized Authorization

The main difference between centralized and distributed database systems is that in centralized, the data reside in one single location and in distributed database, the data reside in several locations. This distribution of data is the cause of many difficulties in transaction processing and query processing. In developing a distributed database, one of the first questions to answer is where to grant system access. Two strategies: (1) Users are granted system access at their home site. (2) Users are granted system access at the remote site.

The first case is easier to handle. It is no more difficult to implement than a centralized access strategy. The success of this strategy depends on reliable communication between the different
sites. Since many different sites can grant access, the probability of unauthorized access increases. Once one site has been compromised, the entire system is compromised. If each site maintains access control for all users, the impact of the compromise of a single site is reduced (provided that the intrusion is not the result of a stolen password).

The second strategy, while perhaps more secure, has several disadvantages. Probably the most glaring is the additional processing overhead required, particularly if the given operation requires the participation of several sites. Furthermore, the maintenance of replicated clearance tables is computationally expensive and more prone to error. Finally, the replication of passwords, even though they're encrypted, increases the risk of theft.

1.2.2. Integrity

All sites have identical database management system software, are aware of one another and agree to cooperate in processing user's requests. Software also cooperates with other sites in exchanging information about transaction. Preservation of integrity is much more difficult in a heterogeneous distributed database than in a homogeneous one. The degree of central control dictates the level of difficulty with integrity constraints (integrity constraints enforce the rules of the individual organization). The homogeneous distributed database has strong central control and has identical DBMS schema. If the nodes in the distributed network are heterogeneous (the DBMS schema and the associated organizations are dissimilar), several problems can arise that will threaten the integrity of the distributed data. They list three problem areas:

1. Inconsistencies between local integrity constraints,
2. Difficulties in specifying global integrity constraints,
3. Inconsistencies between local and global constraints.

Local integrity constraints are bound to differ in a heterogeneous distributed database. These inconsistencies can cause problems, particularly with complex queries that rely on more than one database. Development of global integrity constraints can eliminate conflicts between individual databases. Yet these are not always easy to implement. Global integrity constraints on the other hand are separated from the individual organizations. It may not always be practical to change the organizational structure in order to make the distributed database consistent. Ultimately, this will lead to inconsistencies between local and global constraints. Conflict resolution depends on the level of central control. If there is strong global control, the global integrity constraints will take precedence. If central control is weak, local integrity constraints will.

1.3. Relational Database Security

1.3.1. Security Issues
Access Controls
The most common form of access control in a relational database is the view. The view is a logical table, which is created with the SQL VIEW command. This table contains data from the database obtained by additional SQL commands such as JOIN and SELECT. If the database is unclassified, the source for the view is the entire database. If, on the other hand, the database is subject to multilevel classification, then the source for the view is that subset of the database that is at or below the classification level of the user. Users can read or modify data in their view, but the view prohibits users from accessing data at a classification level above their own. In fact, if the view is properly designed, a user at a lower classification level will be unaware that data exists at a higher classification level.

In order to define what data can be included in a view source, all data in the database must receive an access classification. Denning lists several potential access classes that can be applied. These include: (1) Type dependent: Classification is determined based on the attribute associated with the data. (2) Value dependent: Classification is determined based on the value of the data. (3) Source level: Classification of the new data is set equivalent to the classification of the data source. (4) Source label: The data is arbitrarily given a classification by the source or by the user who enters the data.

Classification of data and development of legal views become much more complex when the security goal includes the reduction of the threat of inference attacks. Inference is typically made from data at a lower classification level that has been derived from higher level data. The key to this relationship is the derivation rule, which is defined as the operation that creates the derived data. A derivation rule also specifies the access class of the derived data. To reduce the potential for inference, however, the data elements that are inputs to the derivation must be examined to determine whether one or more of these elements are at the level of the derived data. If this is the case, no inference problem exists. If, however, all the elements are at a lower level than the derived data, then one or more of the derivation inputs must be promoted to a higher classification level.
The use of classification constraints to counter inference, beyond the protections provided by the view, requires additional computation. It is necessary to prevent all users from accessing all columns of a table for data security reasons. To reduce redundant data to the minimum as much as possible, Oracle allows the creation of an object called a view. A view is mapped to a SELECT sentence. Thuraisingham and Ford discuss one way that constraint processing can be implemented. In their model, constraints are processed in three phases. Some constraints are processed during design, others are processed when the database is queried to authorize access and counter inference, and many are processed during the update phase. Their strategy relies on two inference engines, one for query processing and one for update processing. Essentially, the inference engines are middlemen, which operate between the DBMS and the interface (see figure 1).

The two inference engines work by evaluating the current task according to a set of rules and determining a course of action. The inference engine for updates dynamically revises the security constraints of the database as the security conditions of the organization change and as the security characteristics of the data stored in the database change. The inference engine for query processing evaluates each entity requested in the query, all the data released in a specific period that is at the security level of the current query, and relevant data available externally at the same security level. This is called the knowledge base. The processor evaluates the potential inferences from the union of the knowledge base and the query’s potential response. If the user’s security level dominates the security levels of all of the potential inferences, the response is allowed.
**Integrity**

The integrity constraints in the relational model can be divided into two categories: (1) implicit constraints (2) explicit constraints. Implicit constraints which include domain, relational, and referential constraints enforce the rules of the relational model. Explicit constraints enforce the rules of the organization served by the DBMS. As such, explicit constraints are one of the two key elements (along with views) of security protection in the relational model.

Accidental or deliberate modification of data can be detected by explicit constraints. There are several error detection methods, such as parity checks, that can be enforced by explicit constraints. Earlier we discussed local integrity constraints (section 2.2.). Multilevel classification constraints are another example. A final type of explicit constraint enforces Polyinstantiation integrity.

Polyinstantiation refers to the replication of a tuple in a multilevel access system. This occurs when a user at a lower level L enters a tuple into the database which has the same key as a tuple which is classified at a higher level (L1 > L2). The DBMS has two options. It can refuse the entry, which implies that a tuple with the same key exists at L or it can allow the entry. If it allows the entry, then two tuples with identical keys exist in the database. This condition is called polyinstantiation. Obvious integrity problems can result. The literature contains several algorithms for ensuring polyinstantiation integrity.

Typically, explicit constraints are implemented using the SQL ASSERT or TRIGGER commands. ASSERT statements are used to prevent an integrity violation. Therefore, they are applied before an update. The TRIGGER is part of a response activation mechanism. If a problem with the existing database is detected (for example, an error is detected after a parity check), then a predefined action is initiated. More complicated explicit constraints like multilevel classification constraints require additional programming with a 3GL. This is the motivation for the constraint processor shown in figure 1. So, SQL and, consequently, the relational model alone cannot protect the database from determined inferential attack.

**1.3.2. Security Concerns for the Distributed Relational Database**

**Global Views**

As in the centralized relational database, access control in the distributed environment is accomplished with the view. Instead of developing the view from local relations, it is developed from the global relations of the distributed database. Accordingly, it is referred to as a global view. The view mechanism is even more important in the distributed environment because the problem is typically more complex and while centralized databases may not be maintained as multilevel access systems, a distributed database is more likely to require the suppression of information.
Although global views are effective at data suppression and to a lesser extent at inference protection, their use can be computationally expensive. One of the key problems with a relational distributed database system is the computation required to execute a complex query. Since each view is unique, a different query is necessary for each view. This additional overhead is partially offset by query optimizers. Nonetheless, the addition of global views adds computing time to a process that already takes too long.

*Multilevel Constraint Processing in a Distributed Environment*

In an effort to provide additional inference protection beyond the global view. As with the centralized model, inference engines are added to the standard distributed database architecture at each site. Their model assumes that the distributed database is homogeneous (see section 2.2). In this case, the inference engines at the user's site processes the query and update constraints. Only a small amount of overhead is added. If the distributed database is heterogeneous, however, then the processing overhead would be prohibitively expensive since the inference engines at each site involved in the action would need to process the security constraints for all the local data. Considering the processing demands already in place in a relational database management system (RDBMS), this appears to be impractical.

1.4. Object-Oriented Database Security

*Object-oriented Databases*

While it is not the intent of this synopsis to present a detailed description of the object-oriented model, the reader may be unfamiliar with the elements of a object-oriented database. For this reason, we will take a brief look at the object-oriented model's basic structure.

The basic element of an object-oriented database is the object. An object is defined by a class. An object is composed of two basic elements: variables and methods. An object holds three basic variables types: (1) Object class: This variable keeps a record of the parent class that defines the object. (2) Object ID (OID): A record of the specific object instance. The OID is also kept in an OID table. The OID table provides a map for finding and accessing data in the object-oriented database. As we will see, this also has special significance in creating a secure database. (3) Data stores: These variables store data in much the same way that attributes store data in a relational tuple. Methods are the actions that can be performed by the object and the actions that can be performed on the data stored in the object variables. Methods perform two basic functions: They communicate with other objects and they perform reads and updates on the data in the object. Methods communicate with other objects by sending messages.

Methods perform all reading and writing of the data in an object. For this reason, we say that the data is encapsulated in the object. This is one of the important differences between object-oriented and relational databases. All control for access, modification, and integrity start at the object level.
1.4.1. Security Issues

Access Controls

As with the relational model, access is controlled by classifying elements of the database. The basic element of this classification is the object. Access permission is granted if the user has sufficient security clearance to access the methods of an object. Millen and Lunt describe a security model that effectively explains the access control concepts in the object-oriented model. Their model is based on six security properties:

Property 1 (Hierarchy Property). The level of an object must dominate that of its class object.
Property 2 (Subject Level Property). The security level of a subject dominates the level of the invoking subject and it also dominates the level of the home object.
Property 3 (Object Locality Property). A subject can execute methods or read or write variables only in its home object.
Property 4 (*-Property) A subject may write into its home object only if its security is equal to that of the object.
Property 5 (Return value property) A subject can send a return value to its invoking subject only if it is at the same security level as the invoking subject.
Property 6 (Object creation property) the security level of a newly-created object dominates the level of the subject that requested the creation.

The object-oriented model and the relational model minimize the potential for inference in a similar manner. Remaining consistent with encapsulation, the classification constraints are executed as methods. If a potential inference problem exists, access to a particular object is prohibited.

Integrity
As with classification constraints, integrity constraints are also executed at the object level. These constraints are similar to the explicit constraints used in the relational model. The difference is in execution. An object-oriented database maintains integrity before and after an update by executing constraint checking methods on the affected objects. As we saw in section 4.1.2., a relational DBMS takes a more global approach.

One of the benefits of encapsulation is that subjects from remote objects do not have access to a called object's data. This is a real advantage that is not present in the relational DBMS. Since the objects are encapsulated, an object can be changed without affecting the data in another object. So, the process that contaminated one element is less likely to affect another element of the database.

1.4.2 Object-Oriented Database Security Problems in the Distributed Environment

There are many impediments to the successful implementation of a distributed object-oriented database. The organization of the object-oriented DDBMS is more difficult than the relational DDBMS. In a relational DDBMS, the role of client and server is maintained. This makes the development of multilevel access controls easier. Since the roles of client and server are not well defined in the object-oriented model, control of system access and multilevel access is more difficult.

Multilevel access control problems arise when developing effective and efficient authorization algorithms for subjects that need to send messages to multiple objects across several geographically separate locations. There are currently no universally accepted means for enforcing subject authorization in a pure object-oriented distributed environment. This means that, while individual members have developed their own authorization systems, there is no pure object-oriented vendor-independent standard which allows object-oriented database management systems (OODBMS) from different vendors (a heterogeneous distributed system) to communicate in a secure manner. Without subject authorization, the controls described in the previous section cannot be enforced. Since inheritance allows one object to inherit the properties of its parent, the database is easily compromised. So, without effective standards, there is no way to enforce multilevel classification.