

Architectures for MLS Database Management Systems

LouAnna Notargiacomo

This essay presents a survey of the basic architectures that have been used in the research and development of trusted relational database management systems (DBMSs). While various approaches have been tried for special-purpose systems, the architectures presented here are those developed for general-purpose trusted DBMS products. In addition, this essay presents research approaches proposed for new trusted DBMS architectures, although worked examples of these approaches may not exist in all cases. While recent research has begun on the development of trusted DBMSs based on object-oriented models [JAJ090a, KEEF89, LUNT89], and a study was completed to design and prototype a trusted DBMS using the entity-relationship model [LEFK89], the majority of research and development efforts are based on the relational data model. This survey describes architectures that use the relational model. Although this survey attempts to include all documented architectures, the scope is limited to information in the public domain.

This survey presents architectures in their abstract form. This includes the definition of each component of the architecture, and a description of the relationships and flow of information between each component. The survey then presents a discussion of the benefits of the architecture in meeting mandatory and discretionary security requirements as well as preserving data integrity, followed by a presentation of known problems introduced by each approach.

Background

In the early 1970s, the Air Force sponsored two efforts that set the foundation for trusted relational DBMS research. In 1975, Hinke and Schaefer documented the design of a trusted DBMS for the Multics op-

erating system [HINK75]. This work documented the design for a high-assurance DBMS where the operating system provided all the access control. This was followed by an effort at I.P. Sharp that also developed a model for a multilevel relational DBMS [GROH76, KIRK77]. This model is based on a layered internal DBMS architecture using the Parnas design method [PARN72a].

The Navy then sponsored two programs that have contributed to trusted DBMS progress. The Military Message System model addressed the information processing of multilevel messages [LAND84]. This work introduced the idea of a multilevel container that holds subobjects that the container's level dominates. Although this work influenced many trusted DBMS designs, it did not result in the development of a design for an MLS DBMS.

The Navy also sponsored MITRE to develop the Navy DBMS Security Model for the Naval Surveillance System [GRAU82]. This model addressed the specific requirements for Navy surveillance systems. A special feature of this model was its use of the container concept and nesting of database objects. Database objects could be databases, relations, tuples, attributes, and elements. Not all objects must be labeled. The sensitivity level of an unlabeled object defaults to the next higher object that contains it (for example, a tuple's level could default to the database level). The integrity-lock prototype that was built on the Mistress DBMS [GRAU84] used the requirements of this application and the Navy model. The Tridata DBMS also uses the Navy model as its basis [KNOD88]. This DBMS is currently being sold and is targeted for a B1-level evaluation.

The next landmark event in trusted DBMS development occurred when the Air Force sponsored a three-week study on trusted data management at Woods Hole, Massachusetts [SCHA83]. The participants at Woods Hole were divided into three groups. Group 1 concentrated on near-term architectures, Group 2 focused on multilevel document processing, and Group 3 investigated long-term research issues and approaches. Group 1 focused on architectures implementable in the three-to-five-year time frame. On the basis of a study of Air Force application needs, a majority of these applications required only a small set of sensitivity levels (that is, two to three). Group 1 recommended three approaches: the kernelized (or Hinke-Schaefer) approach, the integrity-lock approach, and the distributed or back-end approach. Group 3 concentrated on supporting access control to the data element level and on the problem of controlling inference and aggregation. Following this workshop, the government sponsored a number of studies and prototypes to investigate and prototype the Group 1 and 3 recommendations. The architectures described here are the results of these efforts.

Trusted DBMS abstract architectures

This survey divides trusted DBMS architectures into two main variations. The first set consists of architectures that use a trusted computing base (TCB) external to the DBMS (usually either a trusted operating system or network) for mandatory control of access to database objects. This approach is commonly referred to as a *kernelized, Hinke-Schaefer*, or TCB subset DBMS *architecture* (this essay uses the TCB subset terminology). These architectures are distinguished from architectures that delegate some or all responsibility for mandatory access control to the DBMS itself. Architectures that perform their own mandatory access control are referred to as trusted subject DBMS *architectures*. Within these two basic approaches, several major variations have developed.

At the conceptual level, a database that contains data labeled over a set of sensitivity levels has relations that may contain data labeled over this same set of sensitivity levels. In the TCB subsetting approach, these multilevel relations are, in some way, decomposed into single-level or system-high fragments. The multilevel secure (MLS) DBMS stores the fragments within physically separate single-level objects (for example, files, segments, or physically separate hardware devices). The MLS DBMS then enforces mandatory access control on requests to access these separate single-level or system-high objects.

The variations in the TCB subset architecture arise because of the different approaches used to separate the multilevel database into fragments and the different ways of managing the processes that access these data fragments. Dealing with process management, two variations to the TCB subsetting architecture have developed. The first approach uses multiple single-level DBMS processes running under a multilevel operating system. A multilevel database is then decomposed into a set of single-level database fragments (for example, single-level relations). These single-level database fragments are physically separated by being stored in single-level operating system storage segments. The Hinke-Schaefer Trusted DBMS [HINK75], one variation of the Trusted Oracle Version 7.0 DBMS [VETT89] (called *OS-MAC mode*), the Secure Distributed Data Views (SeaView) research prototype [LUNT88, LUNT90], and the Lock Data Views (LDV) architecture use this approach [DILL86, STAC90].

The second approach also uses separate instantiations of a DBMS process. This approach uses a trusted network for mandatory separation instead of relying on only a multilevel operating system. This variation also decomposes the multilevel database into multiple system-high fragments. However, in this case the DBMS replicates lower level data under the higher level fragments. The multiple DBMS processes used to access these fragments also operate at system-high levels, matching those of the database fragments. Relying on a multilevel network, the

MLS DBMS separates the data by physically distributing them onto different interconnected system-high DBMS host processors. The Unisys Secure Distributed DBMS (SD-DBMS) research prototype [UNIS89] used this approach, and it is currently being used in the NRL Trusted DBMS (TDBMS) research project [FROS89].

In strong contrast to the TCB subset approach, the trusted subject DBMS stores the conceptual multilevel database in one or more operating system objects (for example, files). From the operating system's perspective, the database is a single-level object. For example, a multilevel database could be stored in one operating system file labeled at the highest level of any object in the database, referred to as *database high*, or each multilevel relation could be stored in a separate file labeled at *relation high*. While the object granularity for the operating system is the file, the object for the trusted DBMS is a database object stored within the file. The database object granularity could be a relation, view, tuple, or element. The MLS DBMS then associates a label with each database object. Since these database objects are invisible to the operating system, it cannot perform access mediation for database objects. The MLS DBMS is privileged to perform this access mediation and operates across a range of the operating system's sensitivity levels.

While various methods can be used to decrease the amount of trusted code within the DBMS itself, these architectures are generally referred to as *trusted subject* architectures. This indicates that some or all of the DBMS is a trusted process that is privileged to violate some or all of the underlying operating system's security policy, especially with respect to the privilege to write down in sensitivity level. This trust means that the DBMS process will write down data only to appropriately labeled operating system objects. For example, if the MLS DBMS is managing a database that contains data labeled up to the Top Secret level (stored in a Top Secret operating system object), it can write a database object labeled Secret into a Secret operating system object.

A major variation on the trusted subset architecture is the integrity-lock architecture. The name of this architecture stems from the fact that the sensitivity label associated with a database object is cryptographically sealed with the data to allow detection of modifications to the data or its label. With this approach, the multilevel database is stored under an untrusted DBMS with labels cryptographically attached to database objects.

While trusted DBMS architectures may be categorized into these two basic types, in reality, many architectures incorporate aspects of both approaches. This is because in most architectures that this essay surveys, techniques developed in the Hinke-Schaefer effort and Woods Hole study are used to minimize the amount of trusted code in the DBMS. Only the original Hinke-Schaefer architecture has the operating

system perform all trusted functions, and that architecture has never been prototyped.

Now that the two basic approaches have been introduced, they will be presented in more detail with the variations of each approach that have developed.

TCB subset DBMS approach

TCB subset [SHOC87] architecture was first fully documented by Thomas Hinke and Marvin Schaefer at System Development Corporation [HINK75]. This DBMS was designed for the Multics operating system with the goal that the trusted operating system would provide all access control. The original design decomposed the multilevel database into single-level attributes or columns, with attributes of the same sensitivity stored together in single-level operating system segments. Tuples are recomposed through join operations. For example, to satisfy a select request, a DBMS process operating at the user's level is initiated. Through the operating system's enforcement of the mandatory access control (MAC) policy, the DBMS has access only to segments at or below its level. The DBMS then joins elements from the same relation to reconstruct multilevel tuples returned to the user.

Currently, all approaches that access the database files through a trusted mediator (for example, trusted operating system or trusted network) are referred to as Hinke-Schaefer approaches. This approach has two main characteristics:

1. the multilevel DBMS is in actuality multiple instantiations of single-level DBMSs, and
2. the multilevel database is decomposed into multiple single-level or system-high databases, each a fragment of the conceptual multilevel database.

There are two variations of this architecture: *centralized* and *distributed*. In the centralized approach, the single-level DBMSs are separate processes running under a trusted operating system, and the multilevel database is decomposed into single-level fragments that are each stored in a single-level operating system object (for example, files or segments). While the DBMS may be trusted to perform some access control functions, the trusted operating system enforces its full access control policy on all access by the DBMS to the DBMS objects. Figure 1 illustrates this approach.

In this architecture, the user is not operating in multilevel mode, but is operating at the session level established with the trusted operating system. Each user interacts with a DBMS operating at the user's session

level, and multiple different DBMSs running at different sensitivity levels may be operating at the same time.

Figure 1. Hinke-Schaefer architecture.

Two research efforts aimed at the A1 evaluation level that use many of the Hinke-Schaefer concepts are the SeaView DBMS and LDV DBMS. Both of these approaches provide the user with a conceptual view of a multilevel database, with each element being labeled with its sensitivity.

Secure Distributed Data Views (SeaView) A1 DBMS. The SeaView program began as a research effort to design a trusted DBMS that could attain an A1 Trusted Computer System Evaluation Criteria (TCSEC) [DOD85] evaluation (based on an interpretation of the A1 requirements for a DBMS). The original agenda of this project was derived from the recommendations of Group 3 of the Woods Hole Summer Study. SeaView was not only designed to support element-level trusted access control and labeling, but was also supposed to address the difficult prob-

lems of inference and aggregation. These problems and discretionary access control were to be addressed by using viewlike mechanisms. In addition, the requirement to meet an A1 evaluation meant that the mechanisms to solve these problems had to be small and simple enough to be formally modeled and verified. These were very difficult requirements, and great advances toward solving these problems have been made in this project.

In the SeaView approach, a multilevel relation is decomposed into single-level relations, based on element-level labeling. Each tuple is then decomposed and stored within the appropriate single-level relation fragments. Fragments with the same relation type and level can then be stored in the same operating system segment, and it is unnecessary to store labels with the DBMS objects. To recompose multilevel tuples to satisfy a user select request, the DBMS joins single-level fragments at or below the user's session level and returns the resulting tuples matching the selection criteria. Since each user is interacting with a single-level DBMS process, the DBMS is not cognizant of any data above its level.

The SeaView architecture is based on an approach called *TCB subsets*. This approach hierarchically layers software components, where the underlying components provide the support needed by the high-level layers. Figure 2 shows the basic TCB layers for the SeaView architecture.

To illustrate the layering approach through an example, suppose an untrusted user application program establishes a session level. This application would interface with an instantiation of the multilevel relation manager at its level. This manager is specially developed software that translates between the user's multilevel conceptual view of relations and the single-level actual representation of the relation. It also performs content-independent discretionary access control (DAC) checks. The multilevel relation manager then issues requests to the single-level relation manager to both recompose the multilevel relations through joins and execute the user request. The single-level relation manager performs DAC on the single-level relation fragments. These single-level relation managers are instantiations of the Oracle Version 7.0 C2 DBMS. Oracle then interacts with the operating system kernel to access objects at or below its level. The operating system, currently Gemini's GEMSOS, is responsible for providing MAC and DAC to operating system objects (files or segments).

Since C2 DBMS processes execute the user's queries, the DBMS processes run as single-level GEMSOS processes. The GEMSOS operating system enforces the MAC policy and part of the DAC policy on each access to data stored in GEMSOS files or segments. Since the DBMS processes are single level, they are unable to enforce any integrity constraints that require knowledge of objects above the process level. For example, if a database relation is defined as having a unique pri-

mary key, the DBMS process would not know of the existence of any tuples that dominate the process' execution level. Therefore, on an insert request to enter a tuple with a primary key that matches a previously existing higher level tuple with the same key value, the DBMS would not reject the insert and would create a *polyinstantiated* version of the tuple. *Polyinstantiation* is a term first defined in the SeaView security model [DENN88a]. It is defined as multiple instantiations of tuples within a relation that have the same primary key, but different sensitivity labels. Within the SeaView effort, *primary key* is defined as including both the primary key attribute and the sensitivity label.

Figure 2. TCB subsetting architecture.

Two benefits arise from polyinstantiation allows for the simple implementation of cover stories. A cover story is the explicit input of untrue lower level data to hide the existence of higher level data.

Benefits. The most significant benefit of this approach is that the trusted operating system performs mandatory separation and access control. With this approach, a higher level of mandatory assurance is possible because an untrusted or lower assurance DBMS process can run as an untrusted process under a higher assurance operating system. This approach allows the system as a whole to meet the operating

system's assurance level for MAC. This approach also allows the DBMS execution logic to be outside the MAC TCB boundary, thus reducing the size and complexity of the TCB. Assuming a previously existing operating system evaluation, this architecture should minimize the amount of time required to evaluate the DBMS.

Disadvantages. Several problems are introduced as side effects of this approach. Since this architecture uses multiple processes, it is not appropriate for applications requiring simultaneous operation of a large number of sensitivity levels. This limitation is due to constraints on the number of simultaneous processes an operating system can support and on its processing capacity. Therefore, this architecture is not appropriate for most intelligence applications. However, this architecture would be appropriate for applications requiring only a small number of sensitivity levels.

The use of view-based DAC requires some trust in the DBMS's capability to execute the view command and identify the subset of data defined by the view. Relying on a lower assurance DBMS with DAC capabilities (that is, a C2 DBMS) limits the assurance level the DAC enforcement can achieve, in most cases, to the DBMS's assurance level. An argument called *balanced assurance* has been used to counter this limitation [SHOC87]. Shockley and Schell argue that DAC mechanisms are inherently flawed and cannot achieve a high assurance level. This is due to the threat that would allow a Trojan horse in a user's process to take on all user DAC privileges and make copies of all user files. On the basis of the inherent flaw in the DAC policy, Shockley and Schell argue that attempting to attain a high-assurance DAC evaluation is futile and that systems which provide higher assurance MAC should not have their evaluation level limited.

A limitation of this architecture is the creation of polyinstantiated tuples. This is a problem because it violates the key uniqueness requirement of the relational model. To meet this requirement, the sensitivity level of the object is added to the logical key. As a result, when higher level users select tuples, they may receive multiple tuples with the same logical entity key, differentiated by the sensitivity level. However, the higher level users may not be able to discern which is the correct version for their application. In addition, the algorithm used to decompose and recompose multilevel relations introduces numerous extraneous polyinstantiated tuples to confuse end users further.

Lock Data Views (LDV) A1 DBMS (Assured Pipelines). The LDV DBMS is specifically designed to run on the Logical Coprocessing Kernel (LOCK) operating system being developed to meet the A1 evaluation criteria. While the LDV architecture also falls into the TCB subset category, its design uses object typing provided by the LOCK operating system and

knowledge-based techniques for data classification that differentiate it from other efforts.

The LDV design is based on the special access control and type enforcement features of the LOCK operating system. In addition to mandatory access control based on sensitivity level and discretionary access control based on access control lists, LOCK also performs access control based on the domain (or role) in which a subject is acting and the type of object it is requesting to access. LOCK maintains a domain and type table, called the *Domain Definition Table* (DDT). In this table, the domains are intersected with data types. At the intersection, the access privileges are recorded (for example, read, write, execute). The DDT is the mechanism used to set up sequences of valid transitions of objects, called *pipelines*. These assured pipelines are used to isolate execution paths through the system in such a way that they can be independently verified. In essence, the LDV DBMS, rather than being considered a system, is a collection of related pipelines responsible for multilevel data manipulation. The three main pipelines in LDV are the response pipeline, the update pipeline, and the metadata pipeline. The response pipeline processes requests to retrieve data. The update pipeline manages all requests to modify the database, including insert, update, and delete operations. The metadata pipeline handles all administrator commands to manipulate the database metadata.

Another unique feature of the LDV design is its heavy reliance on classification rules to control data input and output labeling. The LDV security policy allows for the definition of rules for both discretionary access control on views and mandatory access control on storage objects to handle five types of classification: name dependent, content dependent, context dependent, aggregate, and inference. Name-dependent classification rules define an object's classification based on the object's identifier (for example, an attribute name). Content-dependent classification rules assign a level to an object dependent on the value of the object. Context-dependent classification rules assign a sensitivity level to a combination of data objects retrieved together, for example, enforcing the rule that a patient's name and illness retrieved together must be labeled Confidential. Aggregate classification rules assign a sensitivity to a collection of objects, for example, enforcing the rule that the result of retrieving the phone numbers of more than 10 employees of a company must be labeled Confidential. Finally, inference classification rules control the potential inferences that can be made from one or a sequence of requests over time. LDV enforces these rules such that the level of data may flow only to a more sensitive level.

Similar to the other TCB subset approaches, LDV is layered above the LOCK operating system, and the multilevel database is stored as a set of single-level objects protected by the LOCK operating system. Figure 3 illustrates the LDV architecture.

Benefits. The benefits of this architecture are the strong mandatory access control provided by the LOCK operating system, and the use of type enforcement in the LDV design. The isolation provided by domain and type enforcement allows for the separation of execution paths into pipelines. Pipelines facilitate the minimization of trusted code and isolate privileged software.

Figure 3. LDV architecture.

Disadvantages. This system is subject to problems with achieving high assurance in both its discretionary access control mechanism and its labeling enforcement based on classification constraints. The problems with discretionary access control are the same as discussed under the SeaView disadvantages. These are due to the complexity of view-based mechanisms and the difficulty in achieving a DAC enforcement capability that is both small enough and simple enough to verify.

The first problem with the labeling policy deals with the feasibility of capturing the complexity of inference and aggregation threats and expressing them as rules. In addition, this design proposes to use untrusted code for this function. Upgrading data above a user's level on inserts can result in integrity problems and additional inference threats. The LDV documentation discusses these problems and proposes several operational approaches to handle them. The approach proposed to handle inference and aggregation also includes the ability to include

rules governing use by multiple users over time. In this case, LDV would lock objects from all further access when reaching certain stated limits (for example, a certain number of retrievals of a relation from all users within a specified length of time). While this technique does address a real threat, it creates an operational threat of denial of service.

Distributed architecture with full data replication

This architecture uses physical distribution of a multilevel database to achieve strong mandatory separation and access control. It is an example of the distributed architecture recommended by Group 1 of the Woods Hole study. The basic architecture, shown in Figure 4, uses multiple back-end database processors to separate a multilevel database into multiple, system-high fragments. A trusted front-end processor mediates all user access to the multilevel database and to each single-level back-end database processor [FROS89, JAJ090b].

Figure 4. Fully replicated distributed architecture.

The database processor is untrusted in terms of MAC enforcement. Each back-end database processor manages the fragment of the multi-level database at that level and a replicated version of all lower level data.

The trusted front end is responsible for directing queries to the correct database processor, for ensuring that there is no illegal flow of information between the database processors, for maintaining data consistency between replicated database fragments, and for properly labeling query responses and sending them back to the appropriate user. In addition, the trusted front end is responsible for user identification and authentication, maintenance of the trusted path to the user, and auditing.

Each user interacts with the TDBMS through single-level sessions. A user's query is sent to the back-end DBMS running at the user's session level. Because all lower level data are replicated under the higher level processors, all data needed to satisfy a user's query are at that back end. Therefore, there is no need to modify the user's query.

This approach minimizes the need for trusted code in the front end, since a trusted front end can operate at the user's level. However, there is a need for trusted code to replicate tuples on updates to keep the versions consistent.

The Navy is currently pursuing the development of a trusted DBMS based on this approach [FROS89].

Benefits. This architecture has several benefits that promote achieving a high-assurance DBMS. First, the physical separation of the database into system-high fragments gives strong mandatory separation. Second, since the DBMS executes a user's query at the back end, which has all the needed data, there is no need to decompose the user query or perform additional DBMS functions in the front end. Third, the use of multiple DBMS physical processors should enhance the performance over methods using multiple software processes under a single trusted operating system.

Disadvantages. There are also several disadvantages to this architecture. First, the use of physically separate processors is hardware intensive. Therefore, this architecture is viable only for situations needing a small number of sensitivity levels and does not lend itself to use in intelligence applications, especially where there is a need for a large number of nonhierarchical sensitivity levels. For N nonhierarchical sensitivity levels, the number of back-end processors required is 2^{N-1} .

Similar to the single-processor TCB subset architecture, this architecture has problems in meeting high-assurance requirements for DAC. This is achievable if the DAC object is the entire back end. Since this is equivalent to no discretionary access, this is not an option. Therefore, the DAC objects must be a subset of the data stored under the back-

end DBMS. Another option would be to store DAC access control information in the trusted front end. For any selected object, the back-end DBMS will still be responsible for retrieving the requested data objects. Therefore, using a back-end DBMS with its own DAC capability is a better option. Unfortunately, this approach does raise the cost of the system while still not achieving high-level DAC assurance.

For most applications, one reason for using an MLS DBMS is to have labels on smaller granularity objects. With current trusted workstation technology, the fact that workstations allow users to operate only within single-level windows negates the benefit of smaller granularity labels. The single-level window managers are not trusted to separate tuples or elements with different labels. In this architecture, having each user execute against a session-high version of the database means that the only trusted label the user will see is the session-high label. Use of a higher assurance DBMS back end that supports mandatory access control on DBMS objects will provide smaller granularity labels trusted at the DBMS's level, but that trust does not transfer to the workstation and, therefore, the labels are only advisory when presented to the end user. In addition, the use of a B1 or higher DBMS for each back end would increase the cost and degrade performance.

Distributed architecture with variable data replication

In contrast to the previous architecture, which used full replication of low data, the variable data replication architecture allows data to be distributed and replicated according to actual usage requirements. This approach was used in the Unisys Secure Distributed DBMS (SD-DBMS) project. This architecture is also based on the Woods Hole Group 1 recommendation for a distributed architecture.

The SD-DBMS acts as a single multilevel DBMS, providing trusted labeling at the tuple level. The architectural approach taken to achieve trusted operation is to distribute multilevel relations into single-level fragments and to store these single-level fragments under multiple commercial off-the-shelf DBMS processors. Figure 5 illustrates the SD-DBMS architecture, simplified to two security levels: high and low. This architecture consists of three types of components: user front end (UFE), trusted front end (TFE), and interconnection.

The UFE devices are either workstations operating in single-level mode or trusted workstations running in multilevel mode within a range of security levels. A UFE is intended to host applications that provide the interface between the end user and the TFE. With multilevel UFEs, these applications may be trusted to operate over a range of sensitivity levels.

The TFE component corresponds to the trusted front end found in the Woods Hole architectures. It controls the execution of all DBMS com-

mands and serves as the reference monitor for database access. The TFE consists of trusted and untrusted functions built on a trusted, high-assurance operating system. Connected to the TFE are multiple untrusted back-end DBMS hosts, each operating in system-high mode at an access class within the TFE's access class range. These back-end DBMSs store data at the appropriate access class and respond to single-level requests generated by the TFE in carrying out end users' queries.

Figure 5. SD-DBMS architecture.

The interconnection component, potentially a multilevel local area network, supports isolation of the untrusted back ends and provides secure transmission of data between UFE devices and the TFE.

In the SD-DBMS, databases and relations are multilevel, each with a defined access class range. Each conceptual multilevel relation is physically decomposed into a set of single-level relation fragments. For a given multilevel relation, the relation schema is defined on each back-end DBMS operating at an access class within the relation's range.

User queries are issued against multilevel relations across the range visible to the user and produce multilevel results with tuple-level labels. The query processing strategy for this architecture is based on a property of relational algebra [OCON88]: Any query on a multilevel relation can be decomposed into a set of queries on single-level fragments, and the single-level results of these queries can be assembled into the correct response to the original multilevel query.

The SD-DBMS architecture uses this query processing strategy in the following manner. User programs send queries to the TFE. The TFE decomposes each query into a sequence of subqueries that operate on single-level fragments. Each subquery must be executed on the back end, whose access class is at the least upper bound of the access classes of all fragments on which the subquery operates. Since subqueries executed on the high back end may require access to low data (for example, to perform a join), the TFE may need to temporarily transfer fragments resident in the low back end to the high back end for the duration of a subquery, unless the required current fragment is replicated on the high back end. To assure that no data flow in the opposite direction, this architecture constrains all such transfers to go through the TFE. The TFE controls the submission of subqueries to the back ends for execution and performs any required transfers. Once the execution of the subqueries is complete, the TFE retrieves the individual results. The TFE labels the tuples in each result with the access class of the back end from which they were retrieved. These results are unioned together, and the TFE mediates their return to the user. (Volume 1, Appendix A, of the Unisys technical report "Secure Distributed Database Management System: Architecture" [UNIS89] discusses this query processing strategy in detail.)

Benefits. The most significant benefit of this approach is the ability to return trusted tuple-level labels to the user. All other approaches presented here return the results of a retrieval request labeled at the user's session level and with only advisory, untrusted labels attached to each returned object. Having only untrusted labels requires the end user to manually downgrade any subset of the returned data before releasing it or writing it at a lower level.

A secondary benefit of this approach is that it does not require full replication of data. The elimination of the requirement for full replication decreases the processing time for update replication and reduces storage overhead. In addition, the use of distributed data management techniques allows the subdivision of multilevel requests into multiple single-level requests that can be executed in parallel.

Disadvantages. The use of partial replication, while a benefit, is also the main deficiency of this architecture. The ability to transfer lower level data to a higher level back end to process a high request allows for a potential signaling channel. While this design uses various techniques to minimize the bandwidth of this channel, a small-bandwidth channel still exists and must be audited. Furthermore, the granularity of objects transferred between back ends during request processing is the entire single-level fragment. The requirement to transfer entire single-level fragments increases both the transmission overhead and the storage overhead to handle temporary fragments.

Integrity-lock DBMS

The integrity-lock architecture, shown in Figure 6, consists of three components: an untrusted front-end process, a trusted filter process, and an untrusted data manager process. The untrusted front-end process interacts with the end user. It is responsible for performing query parsing and for final processing of responses sent to the end user. This final output processing can include data management functions that operate at a single level — for example, summation functions that operate only at a single level. The trusted filter process is responsible for encrypting and decrypting objects and their labels, performing access mediation to identify the subset of data returned by the data management process, and downgrading the objects to be returned to the end user. Suppose that the database objects are tuples. In this case, the trusted filter would generate a cryptographic checksum by applying an encryption algorithm to each tuple and its sensitivity label, thus locking or sealing the tuple with its sensitivity label. The residue from the encryption process is associated with the tuple (either stored with the tuple or separately) as its checksum. The multilevel database is stored under the data management process that operates database high.

When the end user performs a selection operation against the database, the trusted filter will direct the data manager to retrieve all tuples matching the selection criteria. The data manager will not perform any operations that do not return entire unmodified tuples — for example, joins or projections. The tuples are then returned to the trusted filter. The trusted filter examines the sensitivity labels and discards tuples that do not pass the mandatory access policy check. The trusted process

then re verifies that each tuple and its label have not been tampered with by reapplying the encryption algorithm and matching the residue from the algorithm with the tuple's cryptographic seal. The untrusted front-end process, running at the user's session level, then applies any additional functions required to satisfy the user request.

Figure 6. Integrity-lock architecture.

In the case of a user insert request, the trusted filter cryptographically seals the inserted tuple together with its label before directing the data manager to insert the tuple. For update commands, the tuples matching

the selection criteria are retrieved and reverified, according to the technique described for a selection request. Each selected and verified tuple is then modified within the trusted process, resealed with a new cryptographic checksum, and stored under the DBMS.

Benefits. The main benefit of this architecture is the use of an untrusted commercial DBMS to store the multilevel database as a database-high object. This approach decreases the number of DBMS functions required in the trusted front end. Specifically, it alleviates the requirement for the trusted front end to support the functions related to indexed data storage and retrieval. In addition, the use of integrity locks enhances the assurance that modifications to the data, either accidental or malicious, can be detected.

Disadvantages. There are several deficiencies associated with this architecture. The use of cryptographic techniques adds complexity to the TCB by requiring key management. In addition, care must be taken in the selection of the object length and composition to ensure that patterns in the data do not occur that allow compromise of the encryption scheme.

This architecture is also subject to a large encoding channel. This channel occurs because the untrusted DBMS has access to all of the data and their labels that are stored in the clear. The DBMS can then return valid data that actually is an encoding of higher classified data.

The requirement to return entire objects to the trusted front end (for example, entire tuples) means that the DBMS cannot perform any functions that return parts of objects through use of the project function, combine portions of objects to form new objects through use of the join operation, or return only aggregates of database objects (for example, return an average). The trusted front end must perform these functions after verifying the checksum on each tuple. This process increases the complexity and size of the TCB.

The use of checksums does allow for the detection of modifications after they have occurred. It does not prevent modifications and cannot prevent deletion of objects.

Trusted subject-monolithic DBMS

Approaches relying on trusted operating system kernels for access control enforcement sacrifice some DBMS functionality for the higher mandatory assurance that can be achieved with these approaches. With the exception of one version of the Trusted Oracle Version 7.0 DBMS that uses the TCB subset approach, all of the DBMS products currently being sold or developed rely on the DBMS itself to enforce access control on database objects. These DBMSs aim at meeting the B1 level of

TCSEC assurance, since this level does not constrain the DBMS to be small or simple. This level allows for the adaptation of an untrusted software product to operate on multilevel data. All of these products also use database tuples as labeled objects.

Figure 7 shows this type of architecture. With this approach, the DBMS software still runs above a trusted operating system. The operating system provides for isolation of the DBMS code and controls access to the database so that all access to the databases must be through the trusted DBMS. The DBMS stores the multilevel database in one or more files. For example, the DBMS could store the database in a single file labeled at the highest level of any data in the database. Or the DBMS could store each multilevel relation in a file labeled at the highest level of data within the relation. The DBMS associates a sensitivity label with each data tuple. The label is treated as an attribute within the relation, even though in reality it is a virtual attribute that need not actually be stored and managed as an attribute.

Figure 7. Trusted subject DBMS architecture.

Benefits. The benefits of this architecture lie in the ability of the DBMS to access all levels of data at the same time. This feature allows a single, multiuser process to service requests for all users, minimizing the operating system's processing load. It also allows for the handling of data labeled over a wide range of sensitivity levels.

This architecture can handle more complex forms of access control that involve data at multilevel access control levels. It also supports integrity constraints defined between data objects at different sensitivity levels, for example, referential integrity constraints.

Disadvantages. The main disadvantage of this architecture is its lack of potential to be evaluated to high TCSEC evaluation classes. Meeting higher assurance levels requires the ability to provide separation of mandatory objects by some form of hardware isolation. With the trusted subject approach, the physical object is the entire multilevel relation or multilevel database. Since the actual mandatory database objects are isolated by trusted software, it is more difficult to prove that this software operates correctly without allowing for the flow of high to low data.

Summary

While many different approaches have been pursued in the research community for high-assurance DBMSs, none of the approaches developed to date has been able to meet all of the high-assurance requirements and still provide the level of functionality provided by untrusted commercial DBMS products. The trusted subject approach has been predominantly used in trusted DBMS products, and only at the lower evaluation classes.

The critical research areas for the future are new architecture techniques to allow trusted subject architectures to minimize trusted code and meet high-assurance requirements. Advances are also needed in the workstation area to provide a flexible client-server environment that allows users to operate simultaneously over a range of sensitivity levels.