TRANSMAT
Trusted Operations for Untrusted Database Applications

Dan Thomsen
Secure Computing Corporation
2675 Long Lake Road
Roseville, MN 55113, USA
email: thomsen@sctc.com

Abstract

This paper presents a technique for allowing untrusted database applications to perform trusted operations. The approach is based on the TCB subset architecture with a commercial database and a small amount of easily assurable, generic, trusted code for the multilevel operations. The approach uses a trusted path mechanism to stop the threat of Trojan horses.

1.0 Introduction

Multilevel applications are expensive to build due to the high costs of creating trusted code. Tools were created to reduce the cost of creating single level applications by providing, common components needed by many applications. For example, most applications need to store data. Database Management Systems (DBMS) were created to provide data storage capability to many different applications. A similar approach is needed to reduce the cost of creating multilevel applications.

The technique described in this paper has been dubbed TRANSactions for Multilevel ApplicaTions (TRANSMAT). TRANSMAT provides an environment for easily creating multilevel applications. The goal of TRANSMAT was to provide high assurance with reasonable development costs.

TRANSMAT uses the DBMS TCB subset architecture, described in the TDI, to achieve high assurance [1]. In the TCB Subset approach, a copy of the DBMS is running at each level. The TCB provides the high assurance separation between each DBMS. This allows the DBMS to be untrusted. The TCB Subset approach was used by both SeaViews and LOCK DBMS [2], [3]. The DBMS provides the environment to build applications quickly. However, many multilevel applications need to perform trusted operations. Since the DBMS layer and the applications environment layer are untrusted, the applications cannot perform trusted operations.

TRANSMAT uses a small trusted program that can execute pre-approved operations. The small trusted program prompts the user to confirm a trusted operation before it is executed.

The confirmation is done using a trusted path mechanism that ensures only users, not programs, can confirm operations. This prevents a Trojan horse from driving a covert channel through the application.

TRANSMAT builds on the TCB subset architecture to support multilevel operations that are secure and easily implemented. Supporting multilevel operations is an important building block for creating multilevel applications. TRANSMAT uses type enforcement to provide controlled trusted write downs in a database context [5].

Section 2 describes the TRANSMAT approach with detailed examples. Section 3 discusses some of the high assurance issues involving TRANSMAT. Section 4 discusses the drawbacks to TRANSMAT and Section 5 provides a brief summary.

2.0 The TRANSMAT Approach

The TRANSMAT system is built on top of a TCB subset architecture such as the ORACLE OS MAC mode used on LOCK DBMS [3]. The TCB subset architecture has a copy of the database running at each level. The DBMS handles the simple security property of allowing users to read data from lower levels. TRANSMAT handles the multilevel communication between different levels with small trusted subjects called TRusted Application Managers (TRAM). Each TRAM is trusted to communicate with other TRAMs running at any level. TRAMs move data between levels. However, each TRAM can only communicate with the DBMS at its level. The TRAMs allow only pre-approved operations to transfer data between levels. Figure 1 shows the basic TRANSMAT architecture.

Any database application can communicate with the TRAM at its level, but no other TRAM. The applications themselves are untrusted. However, if the application requires a multilevel operation, the operation must be certified when the application is created. To make the TRAMs as generic as possible, the operations are stored in a separate table called the Approved Multilevel Operation (AMO) table. Thus as new applications are added, the trusted TRAMs do not have to be modified, only the AMO table.

The entries in the AMO table are represented as parameterized SQL statements. The AMO table is stored outside of the database in a protected, but unclassified file. An example AMO table is given in Table 1.

Each entry contains:

- Operation Name:
  the name is used to index the table so that applications can select the correct operation
- Target Level:
  the level at which to run the operation. This entry indicates which of the other TRAMS may receive the operation.
- Parameter description:
  the size and type of each parameter. This allows TRAMs to limit the amount and kind of data that is regraded.

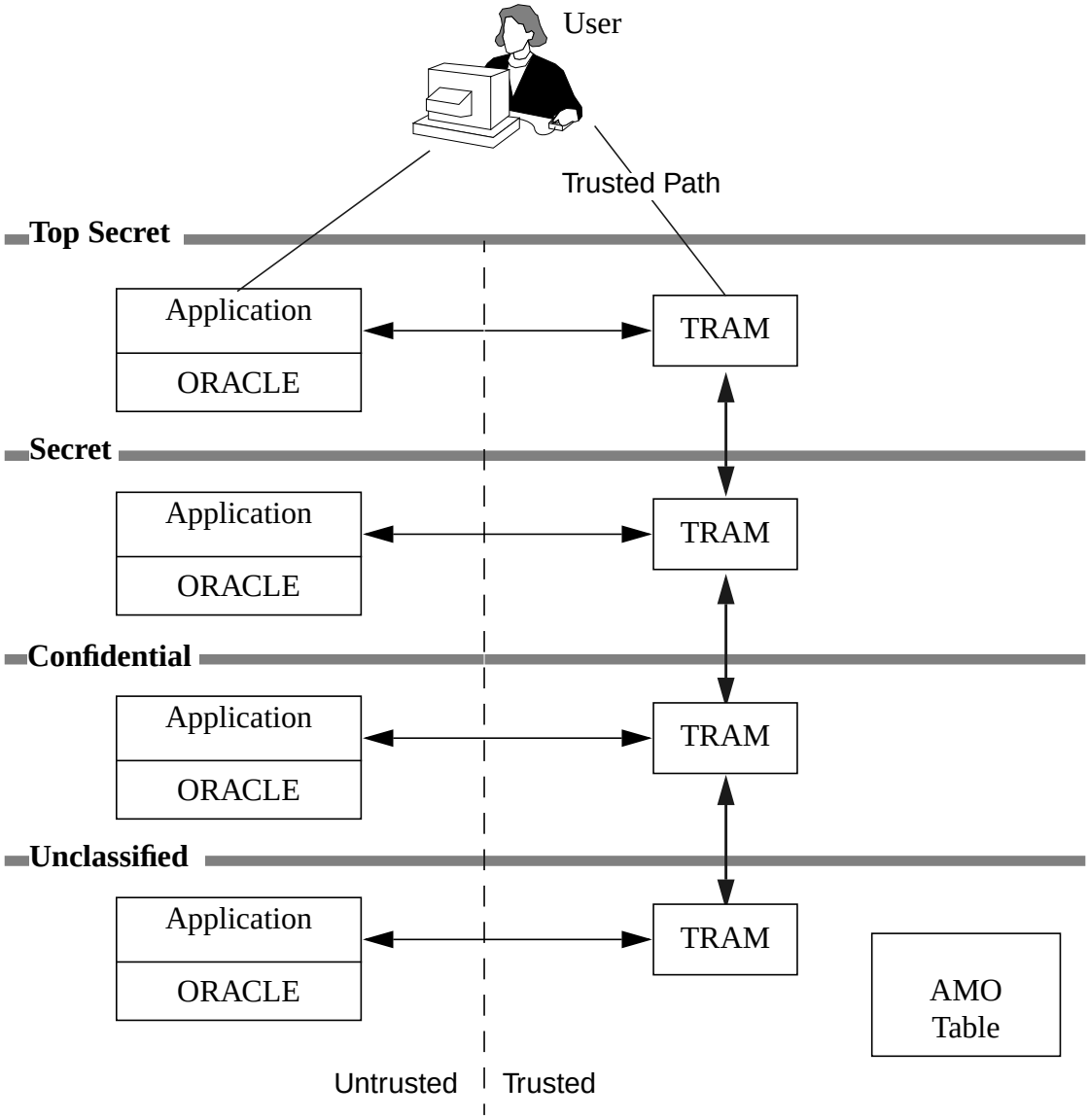

FIGURE 1. The TRANSMAT architecture. A multilevel operation can be initiated by an application at any level. The operation is routed to the Trusted Application Manager (TRAM) which checks the Approved Multilevel Operation (AMO) table. The TRAMs confirm the operation by invoking trusted path communication with the user to ensure Trojan horses are not making the request.

- The SQL script to execute:
  the SQL script that implements the operation. The parameters received are inserted into the SQL scripts at the indicated positions. The receiving TRAM then has the target DBMS execute the script..

The statements are approved and installed only by the certification authority. Applications request the TRAM to run a specific operation with the parameters specified. As new applications are added, or old applications are updated, the AMO table can be updated. This eliminates the need to create new trusted software for each application. Access to the AMO table is tightly controlled via type enforcement.

TABLE 1. Approved Multilevel Operation (AMO) Table

Operation Name	Target Level	Parameter Description	SQL Script
Pilot Selection	Unclassified	30 Characters	`Update pilot set status = assigned where pilot = ???`
Pilot Debrief	Unclassified	30 Characters	`Update pilot set status = available where pilot = ???`

Now that the basic pieces of the approach have been introduced some detailed examples are provided.

2.1 Example: Downgrade Operation

The first example is a sample multilevel operation that downgrades information. This example is based on a simple multilevel application for selecting pilots for missions. The level of a mission can range from Unclassified to Top Secret. Because mission planners at all levels need to know the status of pilots, that information is unclassified (see Figure 2).

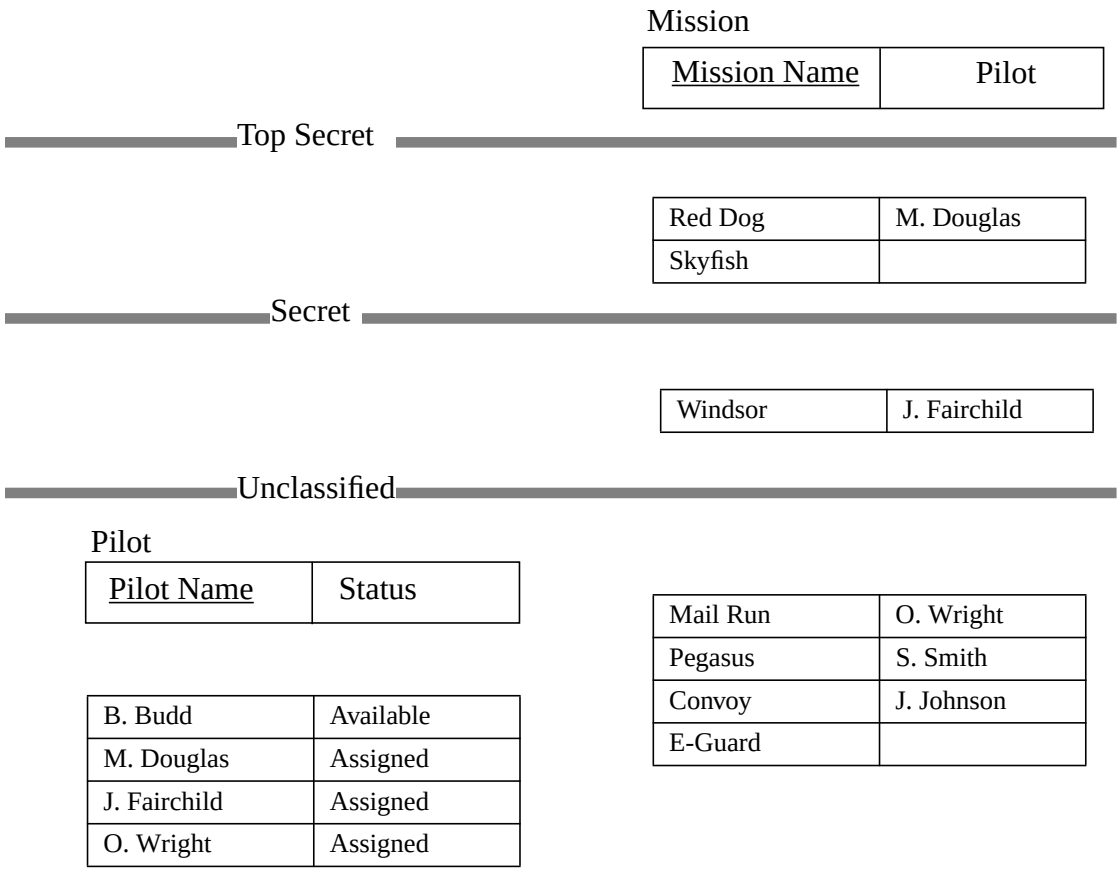

FIGURE 2. An example schema for a multilevel application. The classification level of a mission ranges from Unclassified to Top Secret. All pilot information is stored at the Unclassified level.

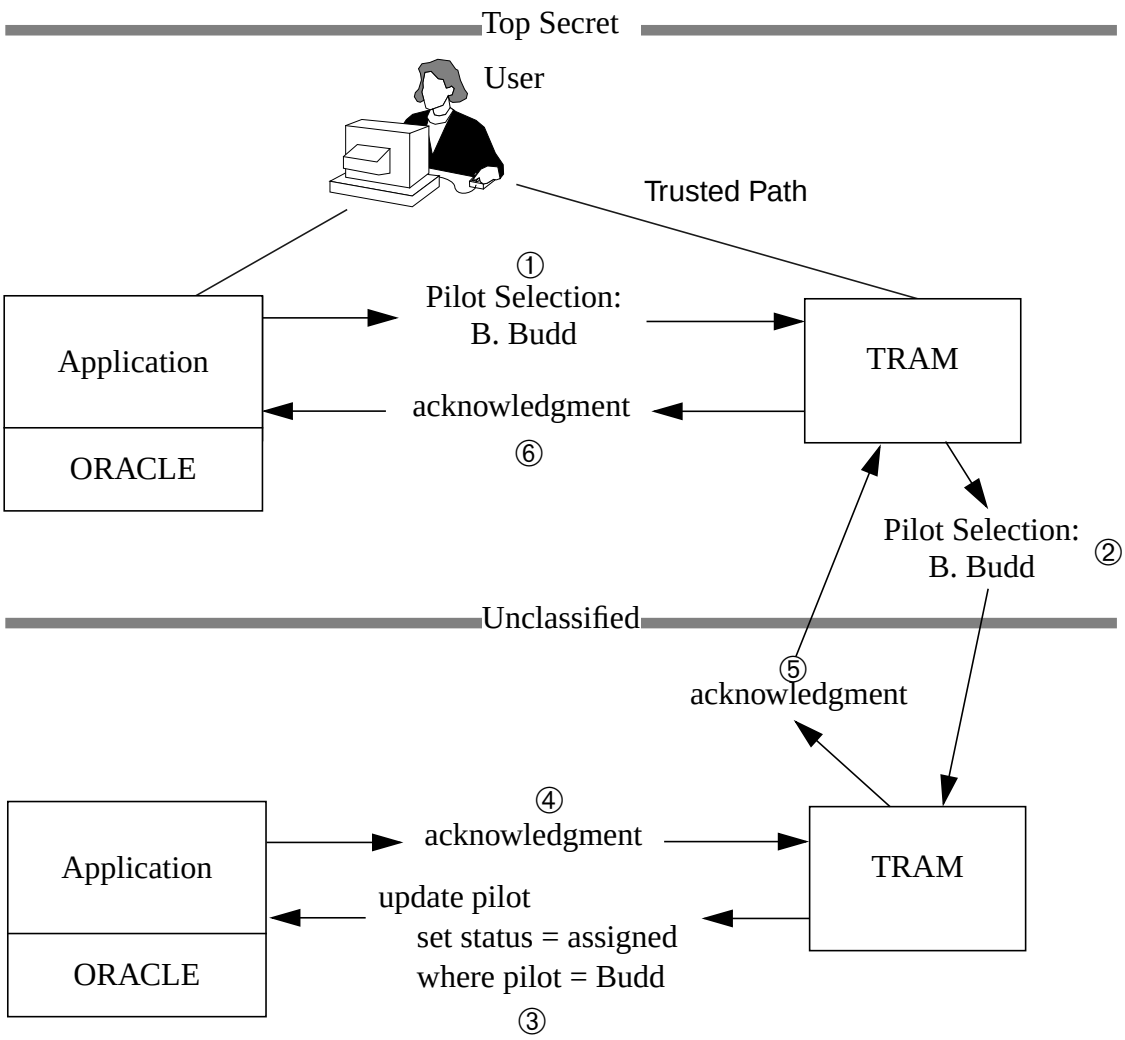

FIGURE 3. This figure shows the execution steps when the pilot selection operation is executed. First the untrusted application selects a pilot and passes the selection to its TRAM. The TRAM confirms the pilot selection by invoking the trusted path. If the user confirms the operation, the information is downgraded. The unclassified TRAM has the untrusted DBMS make the update in the unclassified database. Finally, the success or failure of the update is passed back to the user at Top Secret.

When a Top Secret mission selects a pilot, the Unclassified table must be updated to indicate that pilot is no longer available. Figure 3 shows the steps taken to execute the operation. The application tells the TRAM to execute the pilot selection operation with the pilot parameter set to “B. Budd.” The TRAM executes the predefined integrity checks for the parameter “B. Budd.” In this case the TRAM removes any extra spaces or control characters. Next the TRAM prompts the user for a confirmation using the trusted path mechanism. Invoking trusted path is important because this guarantees that only a user can regrade information. The TRAM is controlled by a person, precluding the possibility of malicious code from operating the downgrading channel (see Figure 3).

2.2 Example: Schema Creation via Write-up

TRANSMAT can be used to allow the database administrator to create a multilevel schema quickly. In the pilot example the mission table must be created and semantically linked with the mission tables at lower levels. This is a tedious, error prone process. A TRANSMAT operation could be created that runs the same “create table” command at the specified levels and automatically connects them semantically by creating the appropriate multilevel views.

2.3 Example: Referential Integrity Across Levels

Unclassified users may be given the power to ensure referential integrity constraints across several levels. In the pilot example, suppose an unclassified clerk updates the database when a pilot is withdrawn from service due to retirement. An AMO entry would be created that removes the pilot from any of the mission tables at the higher levels.

Unfortunately this means referential integrity is being enforced by the application rather than a general mechanism in the DBMS. Thus each application grows more complex. However, this is not a bad trade-off when compared to the risk of an unclassified user entering the wrong data at higher levels. In the pilot example, a supervisor could be required to confirm the retirement operation.

3.0 Assurance Issues

High assurance is achieved in the TRANSMAT approach by combining the following three components

- Trusted path
- Approved Multilevel Operations
- Type enforcement

Each component is discussed in turn.

3.1 Trusted Path

Without requiring a trusted path confirmation, the untrusted applications could drive a timing channel by repeatedly requesting approved operations. The effects of these operations could be observed at lower levels by users or programs working in collusion.

This timing channel is closed using the trusted path mechanism of the underlying TCB. Each time a TRAM gets a request for a multilevel operation, it invokes the trusted path utility asking the user to confirm the operation. The TCB trusted path cannot be spoofed or simulated by untrusted programs. While this extra question may annoy sensitive users, it is not unreasonable. Valid multilevel operations requests are the result of a user’s action; thus the user is available to confirm the action. Depending on the trusted path mechanism available in the TCB, the confirmation may require nothing more that typing “yes” on the ter-

minal. Moreover, trusted path confirmation is not needed on all operations, only those that are multilevel. As multilevel operations are rare the user will not be prompted for confirmation most of the time.

If multilevel operations are frequent, the trusted path confirmation could be eliminated at the cost of reduced security and assurance. However, the TRAM could guarantee that all multilevel operations are entered into the audit trail. The TRAM could also limit the number of unreviewed operations which would limit the size of the covert channel. Any approach without human consent is dangerous. Sophisticated Trojan horses could avoid detection by hiding their actions among legitimate operations. Each site must decide if the risk to security is worth the small inconvenience of human confirmation.

3.2 Approved Multilevel Operations

The parameters that are passed down also provide a storage channel. A hostile application could encode information in the values it provides for parameters. This channel can be controlled by limiting the size of parameters to something easily reviewed by the user. Then when the TRAM invokes the trusted path, it can display the parameter value as well. This precludes the application from substituting its own value. Forty characters may be the practical limit of what users will conscientiously review.

The TRAM must also sanitize the parameters to ensure control characters are not embedded at the end of the parameter or that the application encodes information in the number of spaces. Unfortunately, there may still be ways for a hostile application to encode information, for example by altering the spelling of words. These may slip past even a conscientious, but spelling impaired, reviewer. Individual sites may choose not to take that risk and enforce a no parameters policy by not allowing SQL statements with parameters into the AMO table.

An important question is what assurance must be applied to the AMO table entries. Because entries are expressed in terms of SQL statements, the semantics should be clear. Each site or accreditation authority can decide the complexity of the operation and determine the acceptable level of the security risk.

Imposing a restriction of one simple SQL statement with a single small parameter should provide ample security because it is easily analyzed. As more SQL statements and parameters are added, analyzing and understanding the behavior of the operation becomes more difficult.

Downgrades are of course more problematic than upgrades. The application accreditor must consider how much information is moving down and how quickly. Downgrades must still be a rare event, because even valid, non-malicious downgrades allow users at the lower level to infer actions at the higher levels. This type of inference is discussed further in the Drawbacks section, Section 4.

3.3 Type Enforcement

Type enforcement is a key part of the TRANSMAT approach. It provides separation between the TRANSMAT components and ensures that only the TRAMS downgrade information [5], [6].

Type enforcement is used to separate the DBMS from the TRAMs. Thus the DBMS does not have trusted write down privilege. The TRANSMAT approach provides high assurance with a minimal amount of trusted code. Only the TRAMS are trusted. Each TRAM has only to provide the following services:

- Communicate with the untrusted application to get the operation and parameters
- Establish the trusted path and confirm that the user wants to run this specific operation
- Check the size and type of the parameters
- Regrade the parameters
- Place the parameters into the proper place in the associated script
- Signal the untrusted database to execute the SQL script

Each service can be implemented with a small amount of code.

Type enforcement also guarantees that the DBMS or hostile application cannot modify the AMO table to add operations that have not been approved.

TRANSMAT is similar to having trusted, stored procedures in the database. However, the TRAMs that handle the security across levels are small programs, outside of the database, that can be fully analyzed and assured. It would be extremely difficult to provide high assurance for trusted triggers inside a large, complex DBMS.

4.0 Drawbacks to the TRANSMAT Approach

No approach is without drawbacks. In this section a number of drawbacks are discussed as well as what can be done to minimize them.

4.1 Performance

It is difficult to discuss performance for TRANSMAT without an implementation, but it is clear that operations will be slower than those that take place within a DBMS at a single level. At least two TRAMs and an additional DBMS are involved in each multilevel operation. The originating application must wait until the updates at the other levels have committed and news of the commit travels back through the TRAMs.

The user may wind up waiting longer for a TRANSMAT operation to commit, but TRANSMAT will certainly be faster and more accurate than the alternative of having the user login in at the other level and do the updates manually.

4.2 Inference of Higher Level Activity

If downgrades are made easy, people at the lower levels can infer what is happening at the higher levels. In the pilot example, unclassified users can easily infer that if a pilot is unavailable and is not assigned to any missions at their level, the pilot is flying a classified mission. Obviously this type of inference is a problem faced by any multilevel application.

The inference is aggravated by the fact that it can be done automatically and regularly by an unclassified user. The traditional approach to solving this problem is to create a cover story that explains where the pilot is. A cover story involves the insertion of two tuples into the database, the real tuple at Top Secret and the cover story at Unclassified. TRANSMAT can help here by making the insertion of the cover story tuples a single multilevel operation.

The application can help the user create the cover story as well. In the pilot example the application may prompt the Top Secret mission planner to provide the name of the Unclassified mission to which the pilot will be assigned. The application designer can choose how complicated the cover story is. For example, if the application always created a cover story that the pilot was doing the mail run to Anchorage Alaska, eventually it would stop being a cover story because a regular pattern would be detectable.

The actual drawback of the TRANSMAT approach in regards to inference is that the application designers must be aware of the potential inferences and take steps to prevent inference by lower level users. As a result, the multilevel applications are going to be more complicated. The problem is not with the TRANMAT approach per se, but instead comes from the inherent difficulty in multilevel operations.

4.3 Trusting the Database

The database must still be trusted to maintain the integrity of the data. This is true for any TCB subset architecture. If an application is actually a Trojan horse, it could destroy the data after it has been entered into the database by valid applications. It also is necessary to assume that the SQL scripts in the AMO table are not malicious. They must be carefully reviewed before being placed in the table. Type enforcement is used to protect the table from unwanted modification.

4.4 Binding Applications to AMO Entries

In the current architecture any application can execute an AMO entry. Since the user must confirm the action, it is unlikely that a malicious application can cause damage by calling the incorrect AMO. However, one can not always trust users to do the right thing, so it would be nice to eliminate this problem by creating a binding that ties applications to the AMO entries to which they are authorized. Finding a high assurance approach for creating this binding is future work.

5.0 Conclusion

The approved multilevel operations, trusted path, and type enforcement components of TRANSMAT open the gateway for creating true multilevel applications in a rich DBMS application development environment. The benefits of multilevel operations are many.

- The user can enter multilevel data without switching levels, thus greatly improving the user interface.
- Integrity constraints across levels can be enforced.
- The application controls operations at lower levels, reducing user errors.

TRANSMAT adds the important capability, of multilevel operations, to the TCB subset architecture. With TRANSMAT multilevel applications can be written quickly and securely.

References

[1] National Computer Security Center, "Trusted Database Management System Interpretation of the Trusted Computer Security System Evaluation Criteria," April 1991.

[2] T.F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. "The Seaview Security Model." *IEEE Transactions on Software Engineering*, 16(6):593-607, June 1990.

[3] J.T. Haigh, R.C. O'Brien and D.J. Thomsen, "The LDV Secure Relational DBMS Model," *Database Security, IV Status and Prospects*, edited by S. Jajodia and C.E. Landwehr, pp. 265-279, North Holland, New York 1991.

[4] R.C. O'Brien, J.T. Haigh and D.J. Thomsen, "Trusted Database Consistency Policy - Final Technical Report," Rome Air Development Center, Griffiths AFB, New York, RADC-TR-90-387, December 1990.

[5] W.E. Boebert and R.Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies," *Proceedings of the 8th National Computer Security Conference*, pp. 18-27, October, 1985.

[6] D. Thomsen, "A Comparison of Type Enforcement and Unix Setuid," *Proceedings of the 6th Annual Computer Security Applications Conference*, pp. 304-312, December 1990.