Partial Security Policies to Support Timeliness in Secure Real-time Databases

Sang H. Son, Craig Chaney and Norris P. Thomlinson
Dept. of Computer Science, University of Virginia, Charlottesville, VA 22903
{son, cwc3r, npt4g}@cs.virginia.edu

Abstract

Conflicts in database systems with both real-time and security requirements can be unresolvable. We address this issue by allowing a database system to provide partial security in order to improve real-time performance when necessary. Systems that are partially secure allow potential security violations such as covert channel use at certain situations. We present the idea of requirement specification that enables the system designer to specify important properties of the database at an appropriate level. To help the designer, a tool can process the database specification to find unresolvable conflicts, and to allow the designer to specify the rules to follow during execution when those conflicts arise. We discuss several partial security policies and compare their performance in terms of timeliness and potential security violations.

1. Introduction

A real-time system is one whose basic specification and design correctness arguments must include its ability to meet its timing constraints. This implies that its correctness depends not only on the logical correctness, but also on the timeliness of its actions. To function correctly, it must produce a correct result within a specified time, called deadline. In these systems, an action performed too late (or even too early) may be useless or even harmful, even if it is functionally correct [16]. If timing requirements coming from certain essential safety-critical applications would be violated, the results could be catastrophic.

Traditionally, real-time systems manage their data (e.g. chamber temperature, aircraft locations) in application dependent structures. As real-time systems evolve, their applications become more complex and require access to more data. It thus becomes necessary to manage the data in a systematic and organized fashion. Database management systems provide tools for such organization. The resulting integrated system, which provides database operations with real-time constraints is generally called a *real-time database system*.

In many real-time applications, security is another important requirement, since the system maintains sensitive information to be shared by multiple users with different levels of security clearance. As more and more of such systems are in use, one cannot avoid the need for integrating them. While Secure Alpha [7] has addressed some of the issues in supporting real-time and security requirements at the OS level, not much work has been reported on developing database systems that support both requirements of multilevel security and real-time. In this paper, we address the problem of supporting both requirements of real-time and security, based on the notion of partial security.

1.1 Real-time database systems

Real-time database systems extend the set of correctness requirements from conventional database systems. Transactions in real-time systems must meet their timing constraints, often expressed as deadlines, in order to be correct. In stock market applications and automated factories, a poor response time from the database can result in the loss of money and property. In many real-time database systems, transactions are given priorities, and these priorities are used when scheduling transactions. In most cases, the priority assigned to a transaction is directly related to the deadline of the transaction. For example, in the Earliest Deadline First scheduling algorithm, transactions are assigned priorities that are directly proportional to their deadlines; the transaction with the closest deadline gets the highest priority, the transaction with the next closest deadline gets the next highest priority, and so on. One important goal of a real-time transaction scheduler is to minimize or eliminate the number of priority inversions -- situations where a high priority transaction is forced to wait for a lower priority transaction to complete. As we shall see below, it is this goal that comes in conflict with security requirements.

1.2 Multilevel secure database systems

Multilevel secure database systems have a set of requirements that are beyond those of conventional database systems. A number of conceptual models exist that specify access rules for transactions in secure database systems. One important model is the Bell-LaPadula model [1]. In this model, a security level is assigned to transactions and data. A security level for a transaction represents its clearance level; for data, the security level represents the classification level. Transactions are forbidden from reading data at a higher security level, and from writing data to a lower security level. If these rules are kept, a transaction cannot gain direct access to any data at a higher security level.

However, system designers must be careful of ***covert channels***. A covert channel is an indirect means by which a high security clearance process can transfer information to a low security clearance process [9]. If a transaction at a high security level collaborates with a transaction at a lower security level, information could flow indirectly. For example, say that transaction Ta wished to send one bit of information to transaction Tb. Ta has top secret clearance, while Tb has a lower clearance. If Ta wishes to send a “1”, it locks some data item previously agreed upon. (This data item could be one that is created specifically for this covert channel by Ta.) If Ta wishes to send a “0”, it does not lock the data item. Then, when Tb tries to read the data item and finds it locked, it knows that Ta has sent a “1”; otherwise, it knows that Ta has sent a “0”. Covert channels may use the database system’s physical resources instead of specific data items.

One sure way to eliminate covert channels is to design a system that meets the requirements of non-interference [5]. In such a system, a transaction cannot be affected in any manner by a transaction at a higher security level. In other words, a subject at a lower access class should not be able to distinguish between the outputs from the system in response to an input sequence including actions from a higher level subject and an input sequence in which all inputs at a higher access class have been removed [9]. For example, a transaction must not be blocked or preempted by a transaction at a higher security level.

1.3 Supporting real-time and security requirements

In general, when resources must be shared dynamically by transactions from different access classes, requirements of real-time performance and security are in conflict [11]. Frequently, priority inversion is necessary to avoid covert channels. Consider a transaction with a high security level and a high priority entering the database. It finds that a transaction with a lower security level and a lower priority holds a write lock on a data item that it needs to access. If the system preempts the lower priority transaction to allow the higher priority transaction to execute, the principle of non-interference is violated, for the presence of a high security transaction affected the execution of a lower security transaction. On the other hand, if the system delays the high priority transaction, a priority inversion occurs. The system has encountered an unresolvable conflict. Those unresolvable conflicts occur when two transactions contend for the same resource, with one transaction having both a higher security level and a higher priority level than the other. Therefore, creating a database that is completely secure and strictly avoids priority inversion is not feasible. A system that wishes to accomplish the fusion of multi-level security and real-time requirements must make some concessions at times. In some situations, priority inversions might be allowed to protect the security of the system. In other situations, the system might allow covert channels so that transactions can meet their deadlines.

There are other factors, besides security enforcement, that could degrade the timeliness of the database system. For example, transient overload or failure of certain components could impact the system performance. In such situations, it is important that the system provides a feature for dynamically trading off security and real-time concerns in a manner specified by the designer. Critical transactions must complete by the deadlines, while security violations are strictly controlled.

Our approach to this problem of conflicting requirements involves dynamically keeping track of both the real-time and the security aspects of the system performance. When the system is performing well and making a high percentage of its deadlines, conflicts that arise between security and real-time requirements will tend to be resolved in favor of the security requirements more often, and more priority inversions will occur. However, the opposite is true when the real-time performance of the system starts to degrade. Then, the scheduler will attempt to eliminate priority inversions, even if it means allowing occasional covert channels.

Semantic information about the system is necessary when making these decisions. This information could be specified before the database became operational using a specification language, which will enable users to express the relative importances of keeping information secure and meeting deadlines. Specifications in this language could then be “compiled” by a pre-processing tool. After a successful compilation, the system should be deterministic in the sense that an action must be clear for every possible conflict that could arise. This action might depend on the current level of real-time performance or other aspects of the system. Any ambiguities would be caught at compile time, causing the compilation to be unsuccessful. The compilation of the specification produces output that can be understood and used by the database system.

The problem of supporting both requirements of security and real-time becomes more complicated in a distributed system. In this paper we do not consider distributed environments.

In the next section, we describe the notion of partial security and the ideas behind the specification language. In Section 3, we discuss several partial security policies. Results of simulation study are presented in Section 4. Sec-

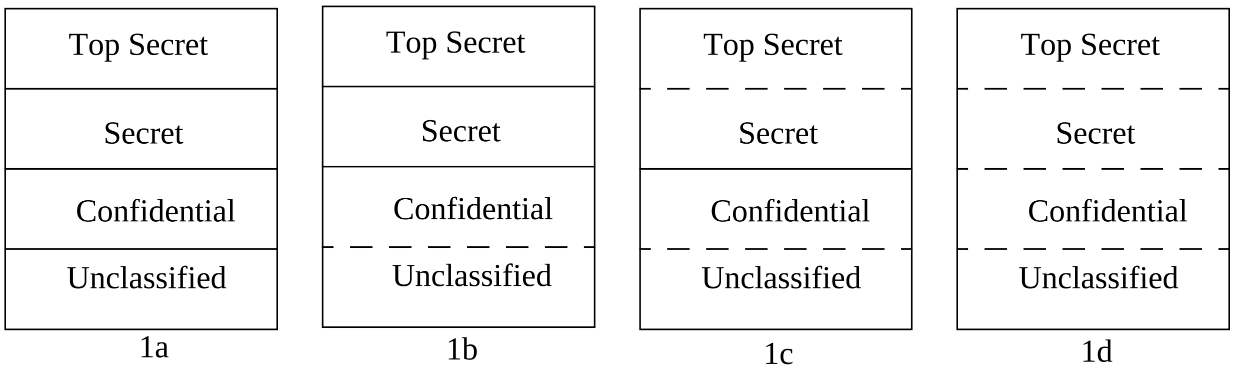

Figure 1 - Partial Security Levels

tion 5 discusses some related work in secure systems and databases. Section 6 concludes the paper with a discussion of future work.

2. Requirement specification

In this section, we first outline the approach to defining partial security. We then provide the details of specifying different rules for the database system.

2.1 Partial security

As explained above, our approach will at times call for a violation of security in order to uphold a timeliness requirement. When this happens, the system will no longer be completely secure; rather, it will only be partially secure. One of the major research questions to be addressed is to identify quantitative partial security levels and to develop methods for making trade-offs for real-time requirements. Although some recent work recognizes the need to consider incomplete security (e.g., [2]), the notion of security has been considered binary. The problem with such binary notion of security is that in many cases, it is critical to develop a system that provides an acceptable level of security and risks, based on the notion of partial security rather than unconditional absolute security, to satisfy other conflicting requirements. In that regard, it is important to define the exact meaning of partial security, for security violations of sensitive data must be strictly controlled. A security violation here indicates a potential covert channel, i.e., a transaction may be affected by a transaction at a higher security level.

One approach is to define security in terms of a percentage of security violations allowed. However, the value of this definition is questionable. Even though a system may allow a very low percentage of security violations, this fact alone reveals nothing about the security of individual data. For example, a system might have a 99% security level, but the 1% of insecurity might allow the most sensitive piece of data to leak out. A more precise metric would be necessary for the applications where security is a serious concern.

A better approach involves adapting the Bell-LaPadula security model and blurring boundaries between security levels in order to allow partial security. In this scheme, only violations between certain security levels would be allowed. As the real-time performance of the system degrades, more and more boundaries can be blurred, allowing more potential covert channels. Additionally, with this scheme, we can still make guarantees about the security of the data. For an example. consider a system with four security levels: top secret, secret, confidential, and unclassified, as shown in Figure 1. Initially, the system is completely secure (Figure 1a). Figures 1b through 1d show systems that are partially secure, progressing from more secure to completely insecure. Solid lines between security levels indicate that no violations are allowed between the levels; dashed lines indicate that violations are allowed. For example, in Figure 1b, transactions that are at the unclassified level may have conflicts with transactions at the confidential level in accessing unclassified data, resulting in a potential covert channel.

It is possible to combine this approach with the use of percentages to define partial security. Then, the amount of security violations between two levels for which the boundary had been blurred would be required to fall below this percentage. The above example is really a special case of this scheme, where levels can either be 0% or 100%. Note that no guarantees can be made between levels that have been assigned a non-zero percentage. Guarantees can still be made between levels designated as allowing 0% security violations; for the other levels, database designers can use different percentages to denote their preferences on where they would rather have the potential covert channels occur.

For certain applications in which absolute security is required for safety-critical applications, any trade-offs of security for timeliness must not be allowed. The idea of partial security discussed in this paper cannot be used in such applications. Even if partial security is acceptable to an application, the system designer should be careful in identifying the conditions under which it might be dangerous to compromise the security. For example, some sort of denial of service attack could force the system into a condition where timeliness constraints are not satisfied. The system can limit the potential damage by setting up rules that can identify the situation and take appropriate actions, if necessary. For example, the system may audit the possible covert channels and log any activity that might be exploring the channel. The rules can utilize the notion of encrypted profile to either look for patterns of illegal access or, alternatively, to certify a good pattern of access.

2.2 Specification

Application designers should be able to specify semantic information using a specification language to express the relative importance of keeping the desired level of security and meeting timing constraint requirements. A question to be addressed in that approach is the verification of the given specification. Specifications should be processed and verified to check any inconsistency in the requirements and to clearly determine the necessary actions to be taken. A tool can be used for specifying the security and real-time requirements to aid the designer first with locating conflicts and then with denoting the desired system behavior according to the semantics of the database. In this section, we provide an overview of the specification language, and show an example to illustrate the idea.

The specification language allows designers to generate rules at varying levels of detail. In applications where much information is known about the database beforehand, designers can control security and real-time aspects of the database much more tightly than in situations where less is known beforehand or such a tight control is not required. There are three levels of detail in the specification scheme. Note that one system can use rules from all three levels if needed.

The specification consists of two parts: a description of the database and a set of rules to follow when conflicts arise. The description provides a framework for the rules. As we shall see below, the specification of both the description and the rules varies between different levels of detail. Regardless of the levels of details that are used, the first part of the specification contains facts about the database as a whole. Here, designers specify the number of data items, the number of security levels, and the number of priority levels used in the entire database.

In the first, most detailed level, designers can generate rules for specific transactions. Transactions are given a number of components. Each transaction is given a readset and a writeset. These can consist of any number of data items. If no readset or writeset is given, they are assumed to be empty. The real-time requirements of a transaction are given by four variables: priority, execution time, release time, and periodicity. The periodicity of a transaction defines how often it starts executing, and the release time indicates the offset of the periodic start. In addition, transactions are given a security level.

Information about data can also be specified. Data items are specified by identifier, and each data item is given a security level. The specification can also contain a default security level, which is assigned to any unspecified data items. All of this information about transactions and data belong in the description portion of the specification.

Not all of these components for transactions and data items are required. In general purpose database systems, some of the information might be hard to specify. However, in many real-time applications, most information is available, since such information is necessary for schedulability analysis of the system to support the timeliness and predictability requirements. In fact, in real-time database systems, many transactions are periodic and their access pattern is known. The only truly necessary components are the security level and the priority level. If a designer leaves out, for example, the readset and writeset, the specification tool cannot make any assumptions about the data accessed by this transaction. It must assume the worst case that the transaction may conflict with every other transaction.

The database designer comes up with rules that define the actions that the system must take when the transactions conflict. These rules can either be static or dynamic. Static rules apply to conflicts that are resolved in the same way every time. For example, the user might specify that a conflict between two specific transactions, or two categories of transactions, will never result in a security violation.

Dynamic rules can depend on certain run-time variables that the database keeps track of during execution. In the current implementation, dynamic rules can be based on three different dynamic variables: security violation percentage, deadline miss percentage, and the number of consecutive missed deadlines. Each dynamic rule has a list of clauses and a default action. A clause contains a boolean relation between these dynamic variables and a constant value, and the action to be taken if the boolean relation is true. When a conflict is encountered by the database system, it checks each clause and takes the associated action if specific clause is true. If none of the clauses turn out to be true, the database takes the default action. For example, a rule might be "If the security violation percentage is less than 5, violate security. Otherwise violate timeliness." Here, the "otherwise" sentence represents the default action.

The second level of specification detail replaces specific transactions with categories of transactions. Transactions are categorized by their security levels and priority levels. The designer can create any number of categories at any

Description:
```
numDataItems 5;
numSecurityLevels 4;
numPriorityLevels 4;

data[default].security = 1;
data[3].security = 2;

ComputeProfit.readset = 1, 2, 3, 4;
ComputeProfit.writeset = 5;
ComputeProfit.periodicity = 12;
ComputeProfit.priority = 3;
ComputeProfit.security = 3;

UpdatePrice.writeset = 3;  # Two transactions access data item 3.
UpdatePrice.periodicity = 30;
UpdatePrice.security = 2;
UpdatePrice.priority = 2;
```

Rule for ComputeProfit-UpdatePrice conflict:
```
(SecViolation% >= 5) ~ violateTimeliness,
(TransMiss% > 10) ~ violateSecurity,
(otherwise) ~ violateTimeliness;
```

Figure 2 - Example of specification with fully specified detail level 1

granularity that he or she feels is appropriate, and describes these categorizations in the description portion of the specification. Then, rules are created for conflicts between categories of transactions. These rules are the same as the rules for the first level.

In the third level of specification, designers create a set of rules describing actions to take in case of conflicts that are not specified in the first two levels. This can be considered as the general system policy. Conditions would depend on the characteristics of the transactions that are conflicting or the current performance statistics.

Specifications are not required to solely use one of these levels of details. The descriptions and rules for these detail levels can be mixed. In this case, when the database encounters a conflict during execution, it first searches to see if a level 1 rule applies. If not, it searches the level 2 rules, and finally checks the level 3 rules. By carefully creating the rules, database designers can implement the partial security policy suitable for the application. A tool can help the designers to develop the partial security policy.

Figure 2 shows an example of a system completely specified with detail level 1. This is a small example, with only two transactions. Every relevant component of these transactions has been specified. Both transactions access data item 3, and UpdatePrice writes to it, so we have a potential conflict. Since UpdatePrice has both a lower security level and a lower priority level than ComputeProfit, this conflict cannot be resolved without causing either a covert channel or a priority inversion. Had UpdatePrice been given a higher priority than ComputeProfit, we can satisfy both requirements by allowing UpdatePrice to preempt ComputeProfit. Alternatively, if UpdatePrice had a higher security level than ComputeProfit, then both requirements could be satisfied by forcing UpdatePrice to wait for ComputeProfit.

In the rule specification, SecViolation% indicates the percentage of security violations and TransMiss% indicates the percentage of deadline miss ratio. Each rules consists of a condition and a decision. The condition part of a rule is stated inside the parenthesis and followed by the decision after tilde (~). Conditions can be connected by logical AND (&) of OR (|). The first line in the rule represents a security crisis. The second line represents a real-time crisis. If none of the above rules apply, the database is instructed to violate timeliness.

Figure 3 shows an example specification with mixed levels of detail (the database description is not shown). There are two transactions specified using detail level 1, but with only the bare minimum number of components specified. These transactions are the same as those used in Figure 2. There are a couple of transaction categories, relating to high and low security transactions. Also, there is an example of a level 3 rule set.

This specific level 2 rule is also an example of a static

Rule for UpdatePrice-ComputeProfit conflict:
```
(SecViolation% >= 5) ~ violateTimeliness,
(TransMiss% > 10) ~ violateSecurity,
((Type1TransMiss% <= 5)|(Type2TransMiss% <= 5)) ~ violateTimeliness,
((Type1SecViolation% < 3)&(Type2SecViolation < 3)) ~ violateSecurity,
(otherwise) ~ violateTimeliness;
```

Rule for HighSecurityCategory-LowSecurityCategory conflict
```
(otherwise) ~ violateTimeliness;
```

Level 3 rules:
```
(SecViolation% < 10) ~ violateSecurity,
(TransMiss% < 15) ~ violateTimeliness,
(priorityLevelDifference >= 2) ~ violateSecurity,
(securityLevelDifference >= 2) ~ violateTimeliness,
((TransMiss% > 10) & (SecViolation% <= 10)) ~ violateSecurity,
(otherwise) ~ violateTimeliness;
```

Figure 3 - Example of mixed level specification

rule -- every time that transactions in these two categories conflict, the database must violate priority and uphold security. Rules for violations between specific transactions and transaction categories can be specified, if the database designer so desires. Otherwise, the designer uses a rule set for detail level 3. If none of the rules in level 1 or level 2 apply to a conflict encountered by the database, it determines the course of action by consulting this ruleset. Again, these are specified in the same manner, with the exception that other variables can be used. The variable priorityLevelDifference represents the difference in the priority levels of the two transactions; securityLevelDifference does the same for security levels

2.3 Specification tool

The tool reads the description portion of the specification and stores it in internal data structures. It analyzes the specification to find all potential conflicts between the security and real-time requirements. For two transactions to conflict, the following must be true:

1. They access the same data item.
2. At least one of them writes to the data item.
3. One transaction must be at a higher security and priority level than the other.
4. The execution times of the transactions must intersect.

Every pair of transactions that satisfy these conditions are reported to the user. With less detailed descriptions, not all of these rules apply. For example, if the readset or writeset of one of the transactions is left unspecified, then the first two rules do not apply. If the timing information is incomplete, the last rule does not apply.

The designer specifies the rules that capture the requirement for the system when these conflicts are encountered. For each conflict, the tool provides advises about the implications of violating security with regard to the partial security policies. For example, in the case of a four level secure database, if a conflict occurs between transactions at the top secret level and the unclassified level, allowing a security violation would force the database into the situation of Figure 1d.

Armed with this information, the designer generates the rules for the database to follow during execution. Once all the rules are specified, the tool verifies that it can determine an action to take in any possible situation. If this is not the case, it reports the problem in the specification. When the specification has no remaining problems, it generates an output file that contains the rules to be referenced by the system during execution.

3. Partial security policies addressing covert channels

To use the notion of partial security, it is essential to specify a level of security acceptable to the applications that use secure real-time databases. Different levels of partial security need to be identified so that the policy makers can decide which level is acceptable, considering potential covert channels and their consequences. In many cases, it would be helpful if partial security policies are in a strict partial order in terms of satisfying the timeliness and security requirements. We have identified several partial security policies as described below. For the simplicity of presentation, we assume 5-levels of security. However, the number of security levels can be arbitrary - there can be 100s of levels of security, if the system needs a fine-grain control of security. Security levels are numbered from 0 (lowest) to 4 (highest). In the following description, it is important to remember that the security violations allowed are only potential violations. They represent possible covert channels through which two transactions in collusion might transfer a small amount of information under the right circumstances.

The policies that are considered in this paper are:

- Completely secure: No security violations are permitted under any circumstances.
- Secure levels 2, 3, and 4: The three highest security levels are kept completely secure. However, conflicts between transactions of the two lowest security levels are permitted to result in potential violations. This policy therefore allows one category of violations, based on the lowest security levels.
- Secure levels 3 and 4: The two highest security levels are kept completely secure. Potential covert channels are permitted among the bottom 3 levels. This policy is very similar to secure levels 2, 3, and 4, except that it allows an extra security level to be involved in potential violations. With this policy three categories of violations are permitted, as each of the lower two security levels are able to create potential violations with the third level.
- Split security: Potential covert channels are permitted between the highest two security levels and among the lowest three security levels. However, no covert channels are allowed from one of the two highest levels to one of the three lowest levels. This policy builds on secure levels 3 and 4 by adding the extra category of allowed potential covert channels between the top two security levels. Four categories of conflicts are possible under split security.
- Secure level 4: The highest security level is kept completely secure. Potential violations are allowed among the four lower levels. This policy simply continues the trend which occurs from secure levels 2, 3, and 4 to secure levels 3 and 4. One less level is kept secure, resulting in three more categories of potential covert channels, for a total of six categories.
- No security: Any potential covert channel is permitted. All ten categories of potential violations are possible.

The policies stated above are well defined in specifying which type of violations would be allowed, and which types strictly prohibited. A gradual security policy, instead of strictly allowing or prohibiting violations between each set of security levels, attempts to limit the number of violations which can occur. One way to do this might be to allow a certain percentage of conflicts to result in security violations. Whenever a conflict arises, it is resolved based on the percentage of violations which have occurred thus far.

To allow finer control over security, this policy allows for a unique percentage to be assigned to each possible conflict set determined by the security levels of the transactions involved. In this way, the percentage of potential violations allowed between the lower security transactions can be larger than those between more secure transactions. This also, of course, allows a complete block to be put on violations between transactions of different security levels. By simply using permission values of 0% for the correct violation levels, all of the security policies presented above can be recreated. More interesting are the effects of combining these complete prohibitions with softer limitations on certain levels.

For example, the highest security level might be kept completely secure, while a controlled number of violations are permitted among the lower levels. Less stringent control would be placed on the lower security levels than on the higher security levels, resulting in security which is gradually tightened as the security level of the transactions involved increases. The gradual security policy, following these specifications, was used in the experiments presented in the next section.

The advantage of such a policy is that it is dynamically adjustable. Changes can be made in the percentage of possible violations allowed for each pair of conflict levels, which permits great customization. Given a static database with periodic transactions, it might be possible to optimize the security/missed deadline trade-off simply by adjusting the values in this policy appropriately. However, this has limited use, as it is very difficult to analyze a database system with unpredictable transactions.

4. Experiments

4.1 Experimental environment

In order to test the performance impact of our approach, we developed a database simulator. The simulator models a single-site, main-memory database with multiple processors that share the memory. The database utilizes the Bell-LaPadula security model, and all transactions are firm real-time transactions. The input to the simulator is a set of rules as described in Section 2.

The database we designed for the simulation experiments was a hypothetical database for hospital application. For this database, we specified a number of level 1 transactions and level 2 categories. There is one level 2 category for each security level. Additionally, we defined a number of transactions for the simulator to generate (outside of those defined in the ruleset). The database had five security levels and five priority levels.

Table 1: Simulation Parameters

Parameter	Value
Number of CPUs	10
Number of time units in a run	100,000
Random transaction arrival rate	1 every 5 time units
Average number of data items read by a random transaction	10
Average number of data items written by a random transaction	6
Average deadline of random transaction	185
Number of data items	500
Approximate number of completed transactions	65,000
Approximate number of transactions active simultaneously	70
Number of periodic transactions	16

Some of the parameters for the basic simulator runs are shown in Table 1. Several parameters were changed in different experiments; the values listed in the table are the default values.

One parameter, average execution time of random transactions, requires some explanation. This parameter is random, but depends on the priority and the deadline of the transaction. The random number is weighted to be a higher percentage of the deadline for higher priority transactions, and a lower percentage of the deadline for lower priority transactions. This weighting is necessary to ensure that random transactions have reasonable priority levels.

4.2 Simulation results

All results presented in this section were obtained by taking the average of 10 simulation runs, each using a different random number seed. Each run lasted for 100,000 time units, and about 65,000 transactions completed during every run.

4.2.1 Varying number of data items. Figure 4 shows the

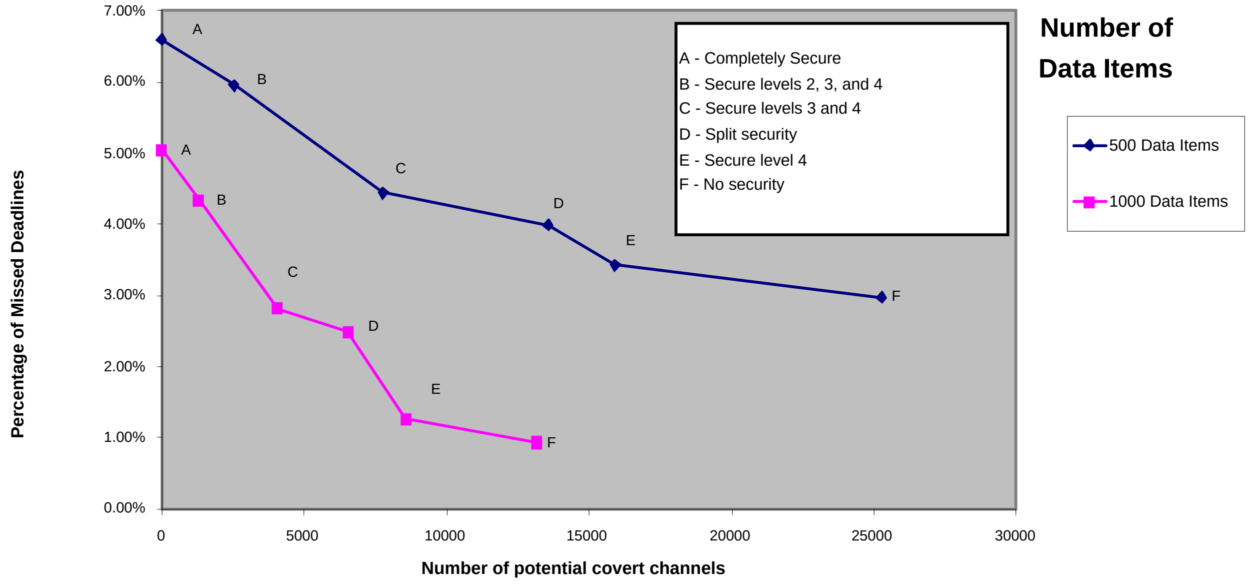

Figure 4- Tradeoff between security and priority with varying numbers of data items

effect of allowing more potential covert channels on real-time performance. Each point represents data taken from experimental runs using one of the six rule sets. The real-time performance does indeed improve as more potential covert channels are allowed. At one end of the spectrum, no security violations are allowed, while at the other all are allowed. In both the 500 and 1000 data item models, there is a consistent improvement in real-time performance as more covert channels are allowed. In both cases the data points progress from one extreme to the other in roughly a linear manner.

In the 500 data series, the sharpest improvement occurs between the rule set that allows no security and the rule sets that allow but strictly limit the security violations. After data point C, we see diminishing returns. Real-time performance still improves as more covert channels are allowed, but at a lesser rate. For the data points representing the more secure rule sets, the database is closer to the saturation point, and higher benefits are obtained by lessening the resource contention. Overall, the number of missed deadlines in no security (point F) is reduced to roughly 50% of that under full security (point A).

In the 1000 data series, there is an even steeper drop in missed deadlines as more potential violations are allowed. This steep drop is continuous all the way to point E. The number of missed deadlines under the no security set is only 20% of those under full security.

The explanation for the fewer missed deadlines and security conflicts under the 1000 data item model lies in the fact that there are more data items being accessed by the same number of transactions. Each transaction is thus more likely to have the data it needs without conflict with another transaction. Therefore, the transaction will be involved in fewer potential violations, and will be less likely to miss a deadline even after waiting for a block to clear on data it needs.

4.2.2 Violation breakdown. In Figure 5, we see where the security violations occur for each ruleset. Each row of bars that runs from left to right represents the data from one set of rules. Each column of bars represents the number of potential security violations between two specific security levels. The lower security levels represent the less sensitive data. For example, violations between the two lowest security levels are displayed in the left-most column of the graph. The flat areas of the graph correspond to the potential security violations that were not allowed by the rule set of that run.

One might wonder why there were so many violations between levels 3 and 4 in the split security rule set, especially when compared to the number of violations between levels 3 and 4 in the no security rule set. By examining these runs more closely, we found that the number of security violations between levels 3 and 4 doubles, while for the other levels where the split security rule set allows potential violations, the amount of potential violations stayed the same. In the no security rule set, a transaction with security level 4 was never involved in a priority inversion (since all security violations are allowed). However, this is not the case with the gradual security rule set. Here, transactions in levels 3 and 4 are delayed whenever they conflict with transactions whose security is 0, 1, or 2. Since these transactions are more likely to be delayed, they will hang around in the system for a longer time, and are more likely to even-

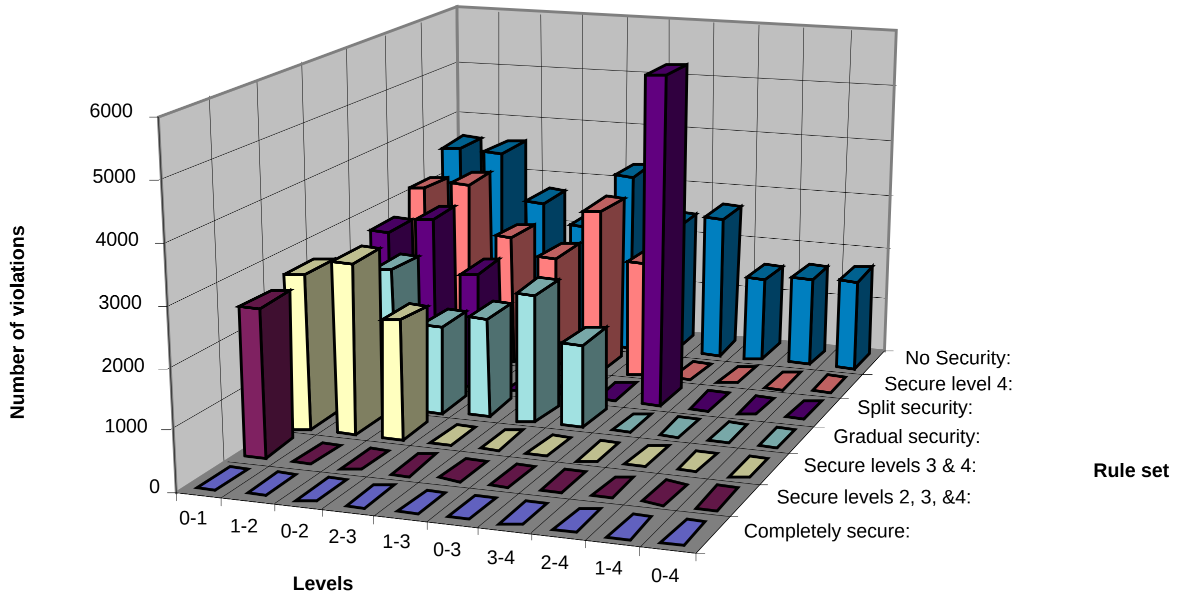

Figure 5 - Potential covert channels by category

tually conflict with each other. Therefore, there are more conflicts, and consequently more security violations, between transactions with security levels 3 and 4.

4.2.3 Varying deadline distribution. In Figure 6, the effects of varying the deadline distribution of transactions can be seen. For random transactions, the simulator assigns an execution time based in part on the deadline time already established. For each series of runs in this experiment, the execution times of all such transactions were modified to be either higher or lower in relation to the deadline times, which were not interfered with. Similarly, the execution times of the pre-defined transactions were increased or decreased. The result was a tightening or loosening of slack time for each transaction.

The results show that the lower the slack time, the more deadlines are missed. There is a consistent increase in missed deadlines for each rule set as slack time is decreased. The smallest increase in missed deadlines is between the 72% and 68% models, as those two models have the least decrease in slack. Following that, there is a noticeable jump in missed deadlines between the 68% and 59% models. The largest increase in missed deadlines occurs between the 59% and 50% models. This suggests that the system is becoming overly saturated at this point. An increase in the number of potential security violations is also evident when comparing the 50% model to the other three models.

Decreasing slack time has an obvious direct effect on the likelihood of a transaction missing its deadline. If there is less time available to waste as the transaction waits to gain locks on data, the transaction will be more likely to use all the spare time and fail to meet its deadline. The increased number of potential violations seen in the 50% model is due to the system becoming full of lingering transactions. The longer transactions are forced to stay in the system, the more often the will come into conflict with each other. Thus, the model with the highest number of transactions missing deadlines also has the highest number of transactions coming into conflict and forcing the system to allow potential violations.

4.2.4 Varying numbers of CPUs. The simulator allows for the user to define the number of CPUs being used in the simulated system. The effect of fewer CPUs on the number of potential security violations was examined by running tests with 10, 7 and 5 simulated CPUs, and comparing the number of potential violations allowed between each security level. Depending on the rule set being used, there were several possible levels of conflict. Under the Secure 2, 3, 4 model, for example, violations were only possible between transactions with priority 0 and priority 1, whereas the Secure 3, 4 model necessitated the examination of conflicts between transactions of priority 0 and 1, 0 and 2, and 1 and 2.

The results show a consistent trend of more potential violations allowed when fewer CPUs are in the system. For each possible set of conflicts between levels, the number of potential violations in the 10 CPU run was 10-20% smaller than the number in the 5 CPU run, while the number for the

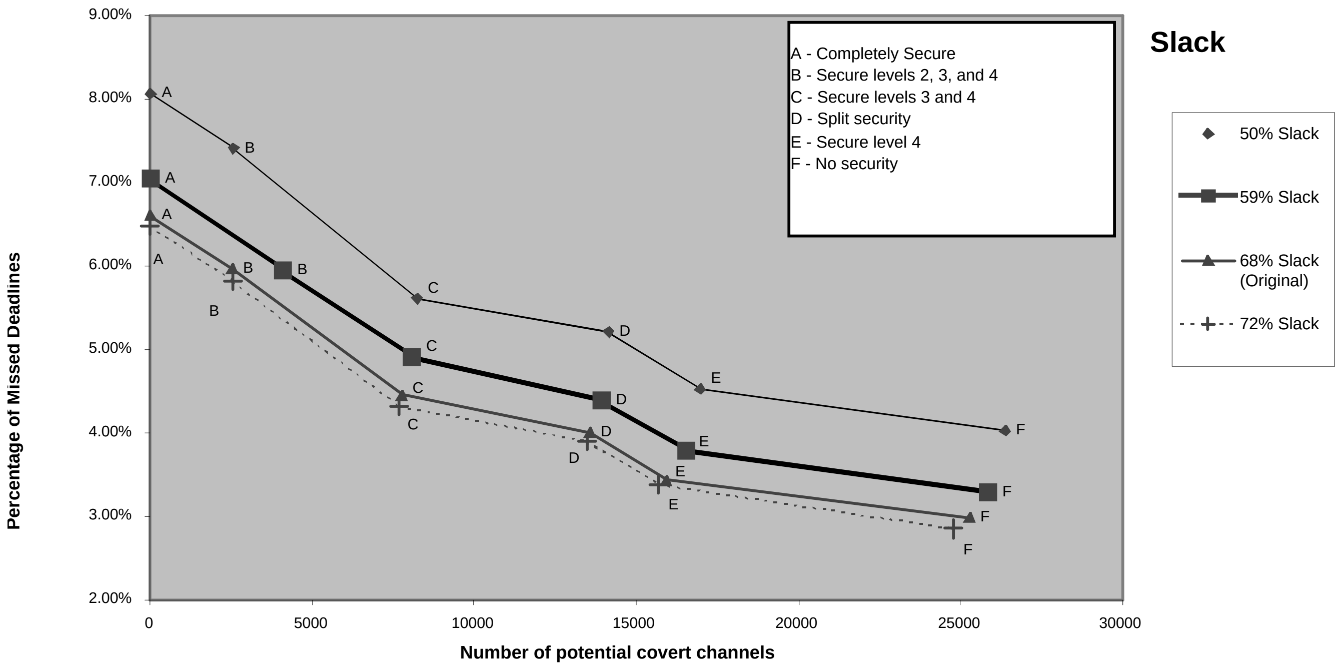

Figure 6 - Tradeoff between security and priority with varying deadline distributions

7 CPU run fell somewhere in between. There was a larger gap in the number of potential violations between the 10 and 7 CPU runs that between the 7 and 5 runs. This is easily explained by the fact that there is a 3 CPU difference between 10 and 7, as opposed to the 2 CPU difference between 7 and 5. If this is taken into account, the increase in potential violations is linear.

The explanation for the increased number of potential violations relates to the bottleneck created by fewer CPUs. The system is unable to process as many transactions at once when using fewer CPUs, so transactions are more likely to be left waiting for a CPU to be freed. As these transactions wait in the system, they are more likely to come into conflict with each other, and thus more likely to produce potential violations.

4.3 Impacts of partial security policies

Several of the partial security policies described in Section 3 were used in the experiments described in this section. The effectiveness of a particular partial security policy depends greatly on which transactions come into conflict. There is a rather complicated net of cause and effect whenever a parameter is changed. This is because allowing, say, fewer conflicts to resolve into a potential violation indirectly changes the number of potential violations allowed. As the number of transactions left waiting in the system is altered, a compounded effect occurs since those transactions are more likely to come into conflict with others. This relationship causes unpredictable results as the database changes, and as transactions of different security levels become predominant in conflicts.

The gradual security model described above, in which the highest level was kept secure and other levels were controlled to varying extents, was tested with the less fluid security models described in this paper, such as split security. In some experiments, the trade-off of violations and missed deadlines fell nicely between those of the secure levels 3 and 4, and the split security policy. However, in other runs, the gradual security policy actually outperformed the split security policy, having both fewer potential violations, and fewer missed deadlines.

What this indicates is that relative performance will vary when using gradual security policies. If conflicts between two specific security levels occur often in a certain database, then the values of the gradual security policy which affect those levels will be very crucial in the final results.

The same complex interactions which complicate the job of establishing an optimal partial security policy apply in reverse. Given a model, it is still very difficult to determine exactly how changing particular values in that partial security layout will affect the violations/deadlines trade-off. Again, making small changes can have unpredicted results, which makes it difficult to tailor a partial security model for a particular situation. Experimentation and analysis seems the only way to determine which partial security policies are suitable for controlling a database.

5. Related work

The approach presented in this paper may require a substantial involvement from system developers/administrators and users. It may not be feasible to have any single designer/user to provide all the necessary information to specify the decision rules. To improve its practicality and usability in real systems, it might be necessary to provide methods for the end user to specify a higher-level goals regarding the potential trade-offs between real-time requirements and covert channel leaks. The user-centered security approach [15] which considers user needs as a primary design goal of secure system development could be useful to figure out a higher-level description of user needs and expectations on specific situations. It could begin with some scenario-based requirement specification for end user to clearly identify the situation and necessary actions to take. The system needs to install monitors to check the system states and perform necessary adjustments by feedback control mechanisms to maintain the high-level goals specified by the user. Ideas similar to the dynamic adaptive security model proposed in [13] could be used to provide trade-offs between security and real-time performance.

There have been several interesting approaches to analyzing and reducing the covert channel bandwidth [6, 8, 10, 14]. While some of those approaches could be used to specify policies to make it difficult to exploit the covert channels that may arise from the trade-off, other may not be applicable in real-time application. For example, a collection of techniques known as fuzzy time [8,14] is inappropriate in a real time setting, since the overall mission may be jeopardized by not getting the exact timing information. In fact, this problem between real-time and covert channel was identified in Secure Alpha work [7]. They have pointed out that slowing clocks or isolating processes from precise timing information is impractical for real-time systems. An adaptive solution to make appropriate trade-offs between the requirements of real-time and security is essential, and it requires a resolution rules to specify the appropriate behavior. To be effective, it is desirable that the rules will be based on application-specific knowledge [3]. Our resolution specification approach is similar to their ideas of “Important Enough to Interfere” and signaling cost.

The idea of using probabilistic partitioning in bus-contention covert channel [6] could be applied in our approach. Instead of keeping track of the percentage of violations for making decisions when conflicts occur, the system could enforce a certain pre-determined percentage by picking up a random number that generates 0 or 1, based on the required percentage. It needs further study to find out whether this way of enforcing the requirements provides the same level of flexibility in specifying the requirements and reduced system overhead compared with the current method of rule enforcement.

George and Haritsa studied the problem of combining real-time and security requirements [4]. They examined real-time concurrency control protocols to identify the ones that can support the security requirement of non-interference. This work is fundamentally different from our work because they make the assumption that security must always be maintained. In their work, it is not permissible to allow a security violation in order to improve on real-time performance.

6. Conclusions

In this paper, we have presented policies to allow the union of security and real-time requirements in database systems. An important part of this union is the definition of partial security. The definition allows potential information flow through covert channels in order to improve real-time performance, yet does not entirely compromise the security of the entire database system. However, database designers must be careful with violations between transactions whose security levels differ greatly. If a violation is allowed between transactions, say, at the highest and lowest security levels, no partial security remains in the system at all. In a system with many such conflicts, it may be very difficult to improve on real-time performance. However, it is essential that the system designer can specify how to manage the system security and real-time requirements in a controlled manner in real-world applications.

We have come up with a scheme that allows database designers to create rules at whatever level of detail that is appropriate. These rules can then be analyzed by a tool, which allows designers to create a database and easily make conscious decisions about the partial security of the database. The tool can also automates the process of scanning through the complex dependencies of a database specification to find conflicts. It then informs the user of the consequences of violating security for each conflict.

Currently, we have a tool that can analyze transactions completely specified in detail level 1. This tool parses a database description, analyzes the dependencies and conflicts, and then goes through an interactive process with the designer to specify rules for all possible conflicts. Our future work includes extending this tool to handle more complex rules and to allow the designer to describe a higher-level description of the system requirements. We are also developing an object-oriented database system to investigate the performance of the system with requirements in real-time, security, and fault-tolerance. We plan to analyze the effects of different choices made by the designer in their requirements and possible trade-offs.

Acknowledgement

This work was supported in part by the National Security Agency and the Office of Naval Research.

References

[1] D. E. Bell and L. J. LaPadula. “Secure Computer Systems: Unified Exposition and Multics Interpretation,” Tech. Rep. MTR-2997, The Mitre Corp., March 1976.

[2] M. Blaze, J. Feigenbaum, and J. Lacy. “Decentralized Trust Management,” IEEE Symposium on Security and Privacy, Oakland, CA, pp 164-173, May 1996.

[3] P. Boucher et al. “Toward a Multilevel-Secure, Best-Effort Real-Time Scheduler,” 4th IFIP Working Conference on Dependable Computing for Critical Applications, San Diego, CA, Jan. 1994.

[4] B. George and J. Haritsa. “Secure Transaction Processing in Firm Real-Time Database Systems,” ACM SIGMOD Conference, Tucson, AZ, May 1997.

[5] J. Goguen and J. Meseguer. “Unwinding and Inference Control,” IEEE Symposium on Security and Privacy, Oakland, CA, pp 75-86, April 1984.

[6] J. Gray. “On Introducing Noice into the Bus-Contention Channel,” IEEE Symposium on Security and Privacy, Oakland, CA, pp 90-98, May 1993.

[7] I. Greenberg et al. “The Secure Alpha Study - Final Summary Report,” Computer Science Lab, SRI International, March 1993.

[8] W. -M. Hu. “Reducing Timing Channels with Fuzzy Time,” IEEE Symposium on Security and Privacy, Oakland, CA, pp 8-20, May 1991.

[9] B. W. Lampson. “A Note on the Confinement Problem,” Communications of the ACM, Vol. 16, No. 10, pp 613-615, October 1973.

[10] I. Moskowitz, S. Greenwald, and M. Kang. “An Analysis of the Timed Z-Channel,” IEEE Symposium on Security and Privacy, Oakland, CA, pp 2-11, May 1996.

[11] S. H. Son, R. David, and C. Chaney. “Design and Analysis of an Adaptive Policy for Secure Real-Time Locking Protocol,” Journal of Information Sciences, vol. 99, pp 101-135, June 1997.

[12] B. Thuraisingham and W. Ford. “Security Constraint Processing in a Multilevel Secure Distributed Database Management System,” IEEE Transaction on Knowledge and Data Engineering, Vol. 7, No. 2. April 1995.

[13] B. Timmerman. “A Security Model for Dynamic Adaptive Traffic Masking,” New Security Paradigms Workshop, pp 1-25, Sept. 1997.

[14] J. Wray. “An Analysis of Covert Timing Channels,” IEEE Symposium on Security and Privacy, Oakland, CA, pp 2-7, May 1991.

[15] M. Zurko and R. Simon. “User-Centered Security,” New Security Paradigms Workshop, Lake Arrowhead, CA, pp 27-33, Sept. 1996.

[16] IEEE Real-Time Systems Symposium, San Francisco, CA, December 1997.