Intrusion Detection in Real-time Database Systems Via Time Signatures*

Victor C. S. Lee
Department of Computer Science
City University of Hong Kong
csvlee@cityu.edu.hk

John A. Stankovic, and Sang H. Son
Department of Computer Science
University of Virginia
{stankovic, son}@cs.virginia.edu

Abstract

In this paper, we describe a method for intrusion detection applied to real-time database systems. The novel idea pursued in this study is to exploit the real-time properties of data in intrusion detection. Data objects will be tagged with "time semantics" that capture expectations about update rates that are unknown to the intruders. This is not simply timestamping data. Our notion of time signatures can be used to detect violations of the security policy. For testing purposes, we use intruders that disguise themselves as "normal" transactions, and compare the ability of different numerically quantifiable measures to capture the behavior of the expected update and to recognize intrusions. For instance, by using a hidden periodic update rate, the system can detect unauthorized update requests, as they will likely not occur at the right time, thereby triggering an alarm to the system. The experimental results show that this technique could be a powerful discriminating measure to identify intruders with a low false alarm rate. While the results are presented for real-time databases, the idea is also applicable to traditional systems.

1. Introduction

In a real-time database system, transactions often have timing constraints. Transactions must meet their timing constraints, often expressed as deadlines, in order to be correct and useful. In stock market applications and automated factories, a poor response time 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.

* This work was partly supported by Grant IJRP-9803-6 from the Ministry of Information and Communication of the Republic of Korea and by NSF grant EIA-9900895.

In most cases, the priority assigned to a transaction is directly related to the deadline of the transaction. For instance, transactions are assigned priorities that are directly proportional to their deadlines in Earliest Deadline First (EDF) scheduling algorithm. The transaction with the nearest deadline gets the highest priority.

In addition to transaction timing requirements, data has time semantics as well [14]. Data such as sensor data, stock market prices, and locations of moving objects all have semantics indicating that the recorded values are valid only for a certain time interval. In order to keep these temporal data "fresh", the recorded values are refreshed or updated periodically by sensor transactions that retrieve the current values of data from the external world. It is often true that the update rate can be chosen anywhere between some minimum and maximum rate and the application will perform properly. Typically, a single rate is chosen and then fixed. In the work presented below we will assume a single fixed rate. However, for even greater security the system could re-assign a new rate within the minimum and maximum rates to avoid an intruder from somehow detecting the hidden update rate.

Database security is concerned with the ability of a database management system to enforce a security policy governing the disclosure, modification or destruction of information. Most database systems contain a fraction of sensitive information that requires such protection from abnormal use. For example, a legitimate user attempting to penetrate the security mechanisms in the database system might submit transactions to access unauthorized data. A user attempting to obtain or destroy unauthorized data from a database through aggregation and inference might retrieve more records than usual or have an abnormal data access pattern.

Timely detection of intrusion is an essential requirement in any secure system. It is more critical in real-time applications such as programmed stock trading and telecommunications in national infrastructure. In

such environments, different users or applications with different levels of security clearance share the same database and some of the applications can have restricted access to sensitive information in the database. For instance, applications that are responsible for reflecting the values of real world entities in the database can have restricted and exclusive access to temporal data objects by creating sensor transactions that write to them periodically. Failure to protect the database from malicious abuse may lead to devastating consequences. Loss or destruction of temporal data can limit a database management system's ability to handle pro-active or retroactive queries [10].

The method proposed in this paper is designed to complement, not replace, conventional database security control for non-real-time database systems, via time signatures. The results of this method can be considered to demonstrate the idea of security rules embedded into data objects. The rules are used to specify constraints that define correct states of an object and inter-object relationships as well as actions to be taken on events. Therefore, the rules can be triggered to reject those security violation activities. For instance, update of a temporal data object (event) can trigger a rule such that the to-be-update time is checked against the expected update time (condition) and reject the update (action) if the predicate returns false.

In many cases, intrusion detection is based on monitoring an activity and comparing it with what is established as "normal" for that activity [8]. The ability to discriminate between a normal behavior of the activity and suspicious behavior depends on the range of fluctuations of "normal" behavior. When a suspicious behavior is detected, a security alarm is invoked and usually the system triggers a more detailed analysis for investigation. A key parameter is the tolerance in deviation from "normal" behavior. If the system reduces the tolerance, false alarms will increase and the system resources could be wasted. On the other hand, the risk of undetected intrusion will increase if the tolerance is increased. In our study, we are based on the hypothesis that exploitation of a database system's vulnerabilities involves abnormal use of a resource.

There are many different levels on which the system activity can be monitored. In our work, we chose to monitor behavior at the level of sensor transactions, which we define as transactions that are responsible for updating the values of real-time data. There are several reasons for focusing on the sensor transactions. Many transactions derive their results from the values of real-time data. Therefore, unauthorized access of real-time data may lead to a wider consequence than other non-real-time data. The range of behaviors of sensor transactions is limited compared to the range of behaviors of user transactions. Sensor transactions are usually small in size and have predefined semantics such as write-only operations and well defined data access patterns whereas user transactions can have a wide variety of characteristics. Finally and most importantly, the behavior of sensor transactions is relatively stable over time, especially compared to user transactions. There is no definite timing characteristic for most user transactions, whereas sensor transactions are usually periodic and have less variation with time.

While the results presented in this paper are focussed on real-time databases, the approach is much more general. For example, any database administrator working with valid users could artificially impose update time semantics on data where that time semantics solely exists for detecting intrusions. In this manner, valid users know when it is acceptable to make updates and the system enforces it. Also, for a complete strategy, the techniques described here could easily be combined with other intrusion detection schemes for greater accuracy.

The experimental results presented in this paper confirm that using time semantics is an effective technique to detect intruders. We present results as a function of system load since varying system workloads will, by themselves, tend to force updates to occur at "wrong" times. We also identify particular issues that arise because we are dealing with real-time sensor transactions. Finally, we also indicate several variations of using time semantics that are NOT effective.

In the next section, we discuss related work. The normal behavior of sensor transactions is analyzed in Section 3. The experimental design is given in Section 4. The methodology and performance results are presented in Section 5. Finally, we conclude the study in Section 6.

2. Related Work

Research on intrusion detection has been conducted for nearly 20 years, yet most existing intrusion detection systems (IDS) are not capable of sufficiently identifying the presence of intrusions [5, 9]. In spite of the significant role of databases in information systems, not enough attention has been paid to intrusion detection in database systems. By intrusion detection, we refer to the ability of continuously monitoring some dynamic behavioral characteristics of a computer system to determine if an intrusion has occurred [6]. Therefore, those static database security methods such as virus protection schemes and firewalls [7] are excluded because they scan static code and unauthorized traffic from outsiders, respectively. On the other hand, some approaches contain intrusion detection characteristics, though they are not trying to identify intruders. In [12], a secure two-phase locking protocol is proposed to control

mutual information transfer through a covert channel in a real-time database system. Although the protocol does not aim at detecting intruders, leakage of information is a dynamic characteristic that can indicate intrusions.

In [2], an intrusion detection model that is based on the hypothesis that security violations can be detected by monitoring a system's audit records for abnormal patterns of system usage is described. The model allows intrusion to be detected without the requirement of knowledge about the flaws in the system. It provides a framework for a general-purpose intrusion detection expert system without focusing on a particular computer system.

In [3], a sense of "self" for UNIX processes is introduced by defining "normal" behavior as normal patterns of short sequences of system calls. They have shown that the simple definition can clearly distinguish between different kinds of processes such that anomalies can be detected. The superiority of their method lies on the compact space requirement of the database of normal behavior and the computational efficiency. In [4], the concept of "self" is further elaborated by making an analogy of the natural immune systems to computer security systems. By applying this concept in privileged processes in a networked operating system, their experimental results suggested that short sequences of system calls do provide a compact signature for normal behavior of the processes and that the signature has a high probability of being perturbed during intrusions. Recently, the ability of different modeling methods to represent normal behavior accurately and to recognize intrusion is compared [16]. The study concluded that for a particular problem, simple modeling methods do work well and the choice of data stream is a more important decision than the particular method of analysis.

Although most of the recent work on intrusion detection does provide good performance, an intrusion detection system is not supposed to work alone or replace any security controls in a computer system. Similar to the principles in other work [2, 4, 15], the intrusion detection mechanism proposed in this paper can be part of a multi-layered system, protected against intruders by other layers such as cryptographic and user authentication systems, analogous to the multi-layered protection of the human body [4].

3. Time Signatures in Real-time Database Systems

Real-time database systems have to deal with data that changes its value with time. These temporal data objects are used to reflect the status of objects in the real world. Whenever the value of a real world object changes, the data that describes this object should change as well, but a certain lag between the moment of change in the real world and the updates in the database is unavoidable [1]. For instance, a transient overload may cause transactions including those responsible for the updates to miss their deadlines [13]. Therefore, data values may become out of date due to delayed updates. Nevertheless, temporal data has to be kept "fresh" by periodic sampling the value of the real world object and executing the sensor transaction that obtains data from the source and updates the database. If intruders are able to change such values, they could sabatage applications such as air traffic control (they could modify the position coordinates of aircraft), or the electric power grid (they could modify power demands and cause outages).

3.1 Period of Sensor Transactions

There are many approaches to determining the period of the sensor transaction such that the temporal data can be kept fresh [17]. Consider one of the many possible semantics of sensor transactions with period P. One instance of the sensor transaction must be generated every period. Suppose a sensor transaction takes e units of time to complete ($0 < e < P$). If an instance starts at time t, we expect the corresponding temporal data will be updated between $t+e$ and $t+P$. Although it may be possible for a sensor transaction to update the temporal data after $t+P$, we expect that it is rare and will happen only under exceptional conditions such as transient overload. Subsequently, we expect that there is at least and at most one sensor transaction in every period for temporal data. A straightforward approach to applying this signature is to invoke a security alarm when a transaction attempts to write a temporal data object that is already updated in a period. However, this signature only contains prior static knowledge (the period) of the system.

3.2 Update Latency of Sensor Transactions

To allow different tolerance levels in a system, we can utilize the knowledge of the behavior of sensor transactions to build a more behavioral constrained (sensitive) intrusion detection system (IDS). A key issue is the latency that a sensor transaction might experience. For a sensor transaction there are many possible causes of latency. One expected phenomenon in a real-time database system is the bursty arrival of sensor transactions. Consider n periodic sensor transactions, T_1, T_2, …, T_n with periods, P_1, P_2, …, P_n, respectively. If they start in phase, then all instances of those sensor transactions will start simultaneously again at every least common multiple of the periods, $LCM(P_1, P_2, \ldots, P_n)$. As a result, it is expected that there is a peak of update

latency every *LCM* of the periods. There may also be minor cycles within the LCM where subsets of the transactions are activated at the same time.

In addition to resource contention, certain scheduling or concurrency control mechanisms such as preemption and blocking may lead to delayed execution of transactions. Normally, temporal data are only written by sensor transactions, which are write-only transactions. Furthermore, sensor transactions do not read any data in the database. Their write sets are disjoint from each other and from the write sets of user transactions. In other words, there should be no write-write conflict among sensor transactions and user transactions. On the other hand, there may be write-read conflicts between sensor transactions and user transactions. When a user transaction holds a read lock on a temporal data object that it needs to access, the corresponding sensor transaction is blocked. Although a higher priority may be assigned to sensor transactions such that the lower priority user transaction is restarted, information may be leaked out through the *covert channels* when the requirements of *non-interference* are not met [11]. Another source of latency is the general workload of the system that can sometimes delay sensor transactions from execution.

Based on the time signatures of sensor transactions in real-time database systems, we can build an IDS with different tolerance levels by including the dynamic behavior of sensor transactions. In this study, we define the normal behavior of sensor transactions in terms of update latency of a temporal data by a sensor transaction. The overall idea is to collect numerically quantifiable measures of normal behavior for each sensor transaction. Once stable values of the measures are obtained, the measures can be used to monitor the sensor transactions' ongoing behavior.

4. Experiments

In the previous section, we introduced a definition for normal behavior, based on update latency of a temporal data by a sensor transaction. The usefulness of the definition largely depends on a number of factors. The first factor is to determine the length of training duration such that the normal behavior can be captured. The second factor is the percentage of sensor transactions that perform their updates within the limits of the update latency learned from the training period. Another factor is the effectiveness of the definition of normal behavior in detecting anomalous behavior. Note that in some simple systems normal behavior might be evaluated off-line analytically. For example, if all the expected transactions are well known, execute similarly at each invocation (i.e., have a constant or near constant execution time), and are periodic, then it is sometimes possible to "predict" a priori when these tasks will execute and no learning of the normal behavior is necessary. Since we do not expect this to be true in most systems, this section reports our findings addressing these factors related to training periods.

4.1 Experimental Model

We consider a firm real-time disk-resident database system. The database consists of a set of data objects. A small database is used in the experiments to create a hot spot effect. Since intruder visits are relatively infrequent, a small database also helps to increase the probability of the intruders to visit temporal data objects. Data objects can be atomically read and written. Each time a data object is written a new instance of the object is created, and the old instance of the data object is deleted. No versioning is considered in this study. The data set consists of non-temporal data objects and temporal data objects. Non-temporal data objects do not directly represent the outside world while temporal data objects should reflect the status of objects in the real world.

Parameter	Value
System	
CPU scheduling	Earliest Deadline First (EDF)
Concurrency control	2PL-HP (two phase lock, high priority)
Number of temporal data objects in hot spot	10
Number of non-temporal data objects in hot spot	40
CPU time per operation	10 msec
I/O per operation	15 msec
User transaction	
slack	1 – 3 (uniformly distributed)
number of operations	5 – 15 (uniformly distributed)
write probability	0.4
Sensor transaction	
Sensor transaction period	250, 500, 750, 1000, 1250 msec
number of operations	1
Intruding transaction	
User-like intruder arrival rate	0.1 tps
Sensor-like intruder arrival rate	1 tps

Table 1: System Parameters and Baseline Settings

Transactions are classified as being sensor transactions and user transactions. Sensor transactions model the periodic sampling of the reading of a sensor

and updating the sensor values. It does not read any data object. It periodically writes a sampled value of a real world object to the corresponding temporal data object in the database. The deadlines of sensor transactions are derived from the requirement that transactions should update a temporal data object before the end of the period that is associated with the data object in order to keep the data object fresh. User transactions are user-level transactions with deadlines. They can be read-only, write-only or read-write transactions.

4.2 Experimental Design

As explained in the previous section, we use two different time signatures in the experiments. The first one is based only on the time semantics of the period of sensor transactions. A transaction will be regarded as intruder and a security alarm will be invoked when the transaction attempts to write a temporal data object that is already updated within a period. For the second one, the time signature of the normal behavior of sensor transactions (in terms of the update latency) is added to increase the discriminating power of the system.

To evaluate the effectiveness of the definition of normal behavior, we have performed two sets of experiments representing different scenarios. In all experiments, intrusion is detected at the point when a write lock on a temporal data object is requested by a transaction. We assume that there is no other discriminating feature of the transactions or temporal data objects except the definition of normal behavior mentioned above.

1) *Intruding transactions disguised as user transactions*
In this scenario, the characteristics of an intruding transaction are identical to a user transaction except for the data object access pattern. For user transactions, they will not write on a temporal data object. Normally, only sensor transactions write on temporal data objects. However, it is assumed that intruding transactions do not have this knowledge. They access the desired data objects without knowing the temporal nature of data objects.

2) *Intruding transactions disguised as sensor transactions*
Here intruding transactions disguise themselves as sensor transactions by acting as a small transaction with write-only characteristics. One difference is the arrival pattern. While sensor transactions arrive periodically, intruding transactions follow a random process which we model as a Poisson arrival process. Similar to the previous scenario, intruding transactions do not have the knowledge of the temporal nature of data objects. As mentioned earlier, if there is a possibility that an intruder somehow figures out the hidden rate, the system will change this rate at random times to a new rate between the minimum and maximum rates allowed for that item. Hence, the intruder will only have a short window of opportunity. This feature is not tested in the experiments described below.

4.3 Performance Metrics

For accurate intrusion detection, we must correctly classify both intrusions and normal sensor transactions. When intrusions are not identified, they are called *false negatives*. When normal sensor transactions are identified as anomalous, they are called *false positives*. We wish to minimize both kinds of errors. Equivalently, we have *true positives* and *true negatives*, which should be maximized. We present the results by dividing the counts by the total number of intrusions or normal sensor transactions to get the rate measures such as the true positive rate. Note that the number of intrusions is counted as the number of anomalous accesses to a temporal data object instead of number of transactions that perform the access.

We also recorded the following statistical metrics in our simulations.

1) *Update latency time* – Update latency is the time elapsed since the earliest possible update time by a sensor transaction. The earliest possible update time can be achieved when the sensor transaction does not suffer from any delay due to resource contention or blocking.

2) *Inter-update time* – Inter-update time is the length of time between two successive updates of the same temporal data object.

The *mean* and *99% confidence interval* are computed from the collected values over the training period. There are a number of advantages of using this *mean and standard deviation model* [2]. It requires no prior knowledge about normal activity. It learns what constitutes normal activity from its observations, and the confidence intervals automatically reflect this increased knowledge. Moreover, the *range* comprised of *maximum* and *minimum* values is also collected, which are fixed limits and are determined from observations during the training period.

The performance results plotted in the graphs are the average of multiple runs. To cover all the interval widths within 5% of the plotted average, we used 95% confidence interval on the values of the chosen statistical metrics.

5. Performance Analysis

5.1 Training Duration

Determining the right length of training period such that the normal behavior can be captured is important to

the effectiveness of the definition of normal behavior. Insufficient training may lead to a high false alarm rate. Prolonged training will be too costly and inefficient. Figure 1 shows the maximum update latency time as a function of training duration in a high workload environment. The user transaction arrival rate is 2.4 transactions per second (tps). The training duration is expressed in terms of the multiple of the *LCM* of the five different periods of sensor transactions as shown in Table 1. From the periods of the sensor transactions used in the experiments, the *LCM* value is 15 seconds. At the beginning of the training period, the maximum update latency times are increasing, but gradually level off. We choose the training duration to build our model of normal behavior when there is no further change of the maximum update latency of all the sensor transactions with different periods for more than 12**LCM* (180 seconds). For instance, as illustrated in Figure 1, the maximum update latency stops increasing for 180 seconds after a training duration of 52. This value is equivalent to 52 times 15 seconds, which is the *LCM* value, giving a training duration of 780 seconds.

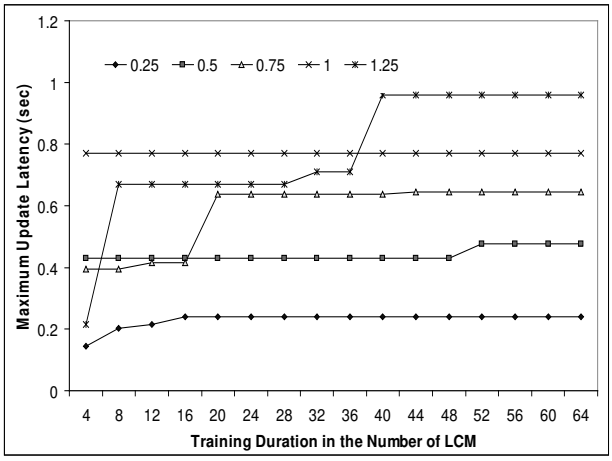
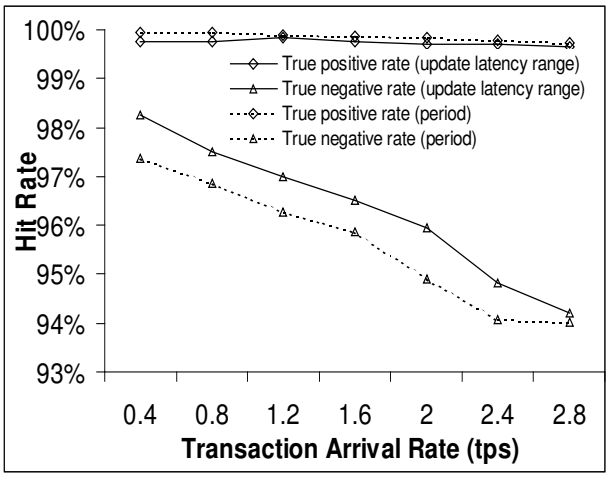


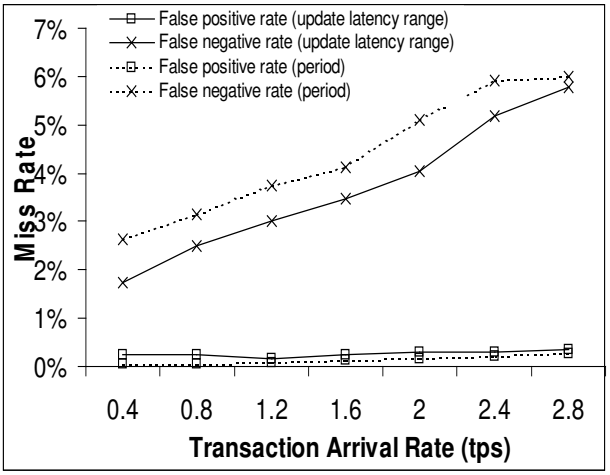
Figure 1: Maximum Update Latency Intruders Disguise as User Transactions (2.4 tps)

5.2 Performance Results

Figure 2 shows both the hit rate (true positive and true negative rate) and the miss rate (false positive and false negative rate) with respect to different workloads by using time signatures explained in the experimental design. The dotted lines show the performance using the time signature of the period of sensor transactions and the solid lines show the performance when the update latency is added. When a higher tolerance is used (dotted lines), the risk of undetected intrusion increases. However, there are fewer false alarms. When the tolerance level is reduced by adding the update latency signature (solid lines), the true negative rate is improved and the number of false alarms is increased slightly. One important and useful feature is the low false positive rate across different workloads. For instance, the false positive rate is as low as 0.36% when the user transaction arrival rate is 2.8 tps. This is very important since there is a strong desire to maintain a low false alarm rate. On the other hand, the true negative rate drops gradually when the workload increases, implying that the discriminating power of the update latency range also decreases, which is due to the expanded update latency range as the workload increases.

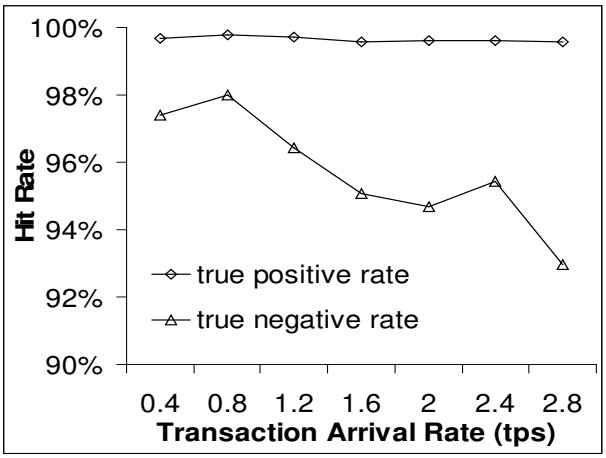


(a) Hit Rate

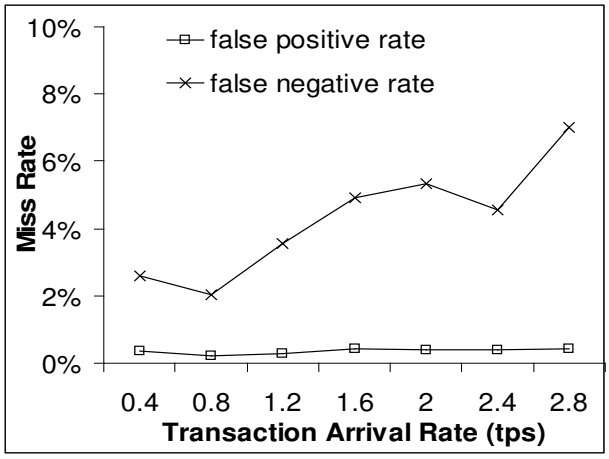


(b) Miss Rate

Figure 2: Performance of Period Signature and Update Latency: Intruders Disguise as User Transactions



(a) Hit Rate



(b) Miss Rate

Figure 3: Performance of Period Signature and Update Latency: Intruders Disguise as Sensor Transactions

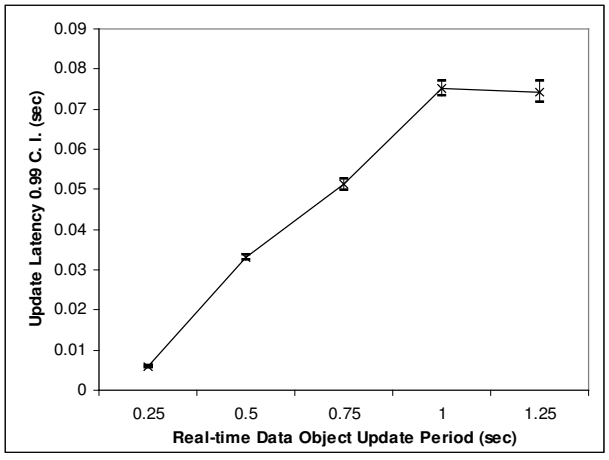


Figure 4: Mean and 99% Confidence Interval of Update Latency: Intruders Disguise as User Transactions (2.0 tps)

When intruders disguise themselves as user transactions, the time constraint of intruders is loose compared to sensor transactions. It makes it easier to detect intruders because it is very likely for sensor transactions to preempt intruders (because of the underlying scheduling algorithm) such that sensor transactions can arrive on time. In order to isolate this factor in the second set of the experiments, we have introduced intruders that disguise themselves as sensor transactions. Since these disguised transactions are similar to sensor transactions, the preference to sensor transactions in CPU scheduling vanishes. Figure 3 shows the results. Generally, the true negative rate decreases compared to the previous set of experiments, but the false positive rate remains almost the same.

5.3 Update latency Distribution

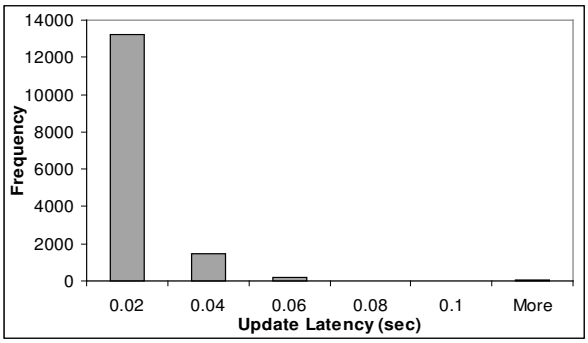
To have a deeper understanding of the performance results, the update latency distribution is discussed in this section. Figure 4 shows the mean and 99% confidence interval of the update latency for different periods of sensor transactions in a high workload environment. We find that update latency increases as the update period increases. This is because a sensor transaction with a shorter update period has an earlier deadline and will be given preference by the CPU scheduler that employs EDF.

For a sensor transaction, it has to compete with other sensor transactions that are generated at the same instance. A sensor transaction will be executed after other sensor transactions that have a shorter period and are generated at the same instance. Moreover, the update latency also contributes to the frequency of instances with shorter period sensor transactions and the number of such sensor transactions that are generated at the same instance. The update latency for the sensor transactions with period of 1.25 seconds is not longer than that with period of 1 second. For instance, a sensor transaction with a period of 1 second is always competing with a sensor transaction with a period of 0.5 second whereas a sensor transaction with a period of 1.25 seconds is competing with a sensor transaction with a period of 0.5 second every other time.

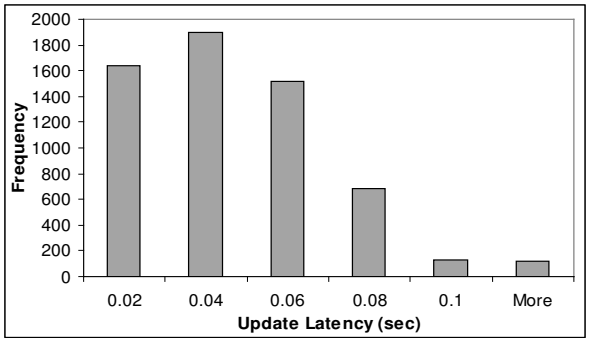
On the other hand, the spread of update latency (99% confidence interval) follows the update period. The longer is the update period, the wider is the spread of update latency. However, the use of this spread in detecting intruders is found to be relatively limited. As mentioned before, one of the normal behaviors of sensor transactions is the bursty nature. Sensor transactions experience long update latency at least once in every *LCM* duration. Let's call this behavior the *LCM* effect. As a result, we found that the false positive rate is high if

the 99% confidence interval of update latency is used as the measure to detect intrusion. Figures 5 and 6 help to understand the distribution of update latency time of sensor transactions. In a light workload environment (Fig. 5(a)), the update latency of high priority sensor transactions (period = 0.5 sec.) falls into a narrow range. In this case, the update latency of most (88%) of the sensor transactions is 20 msec. However, to average over all the sensor transactions, the mean value becomes 22.16 msec with 99% confidence interval of 21.9 and 22.4. As a result, using the confidence interval as a measure will misclassify the sensor transactions as intruders for such a skewed distribution where the mode falls outside the confidence interval, leading to a high false alarm rate. For low priority sensor transactions (Fig. 5(b)), the update latency time is more widely spread due to the LCM effect and the blocking of user transactions. In this case, the mean value is 47.59 msec with 99% confidence interval of 46.33 and 48.85. Compared to the previous case, fewer sensor transactions are mistakenly classified, though the false positive rate is high. Figure 6 shows the heavy workload situation. Although the distribution is less skewed, the use of a confidence interval as the measure to detect intrusion is still far from effective. One of the main results of our work is that the system should not use the confidence interval on update latency as a measure to detect intrusion, which could result in a high false alarm rate.

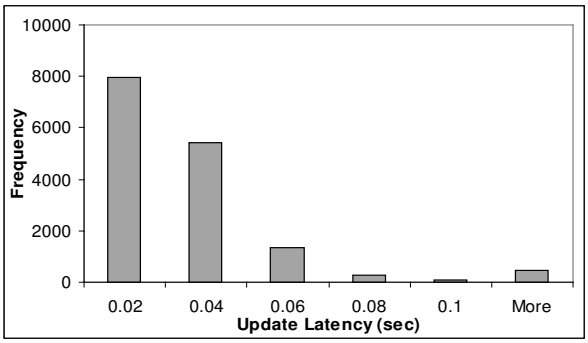
Figure 7 shows the update latency range of sensor transactions in a high workload situation. The mean value of the update latency is near the minimum value instead of between the maximum and the minimum. Although the update latency range covers most of the duration in a period in a high workload situation, we found that it is an effective discriminating measure. The reasons are mainly due to the distinct characteristics of "time semantics" in real-time databases and the importance of correct classification of true positive. Sensor transactions are mostly and very likely to arrive at the expected time or slightly deviated from the expected time. Once a sensor transaction is correctly identified as "self", the remaining duration in the current period is prohibited from further update of the corresponding temporal data object. Therefore, the update latency range provides a higher tolerance and a greater flexibility than the 99% confidence interval in capturing the normal sensor transaction before shutting off the interval to intruders. In Figure 8, a sensor transaction that arrives at t_1 will be classified as normal if the confidence interval is used. However, since the confidence interval is so narrow, if the sensor transaction is late and arrives at t_2 (not at t_1), it will be mistakenly classified as an intruder. Therefore, a measure such as the update latency range that has a higher tolerance to normal behavior is more desirable.



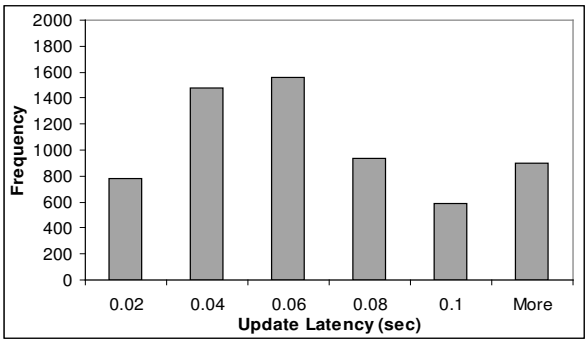
(a) High Priority (Period = 0.5 sec)



(b) Low Priority (Period = 1.25 sec)
Figure 5: Update Latency Distribution of Sensor Transaction at Light Workload (0.4 tps)



(a) High Priority (Period = 0.5 sec)



(b) Low Priority (Period = 1.25 sec)
Figure 6: Update Latency Distribution of Sensor Transaction at Heavy Workload (2.0 tps)

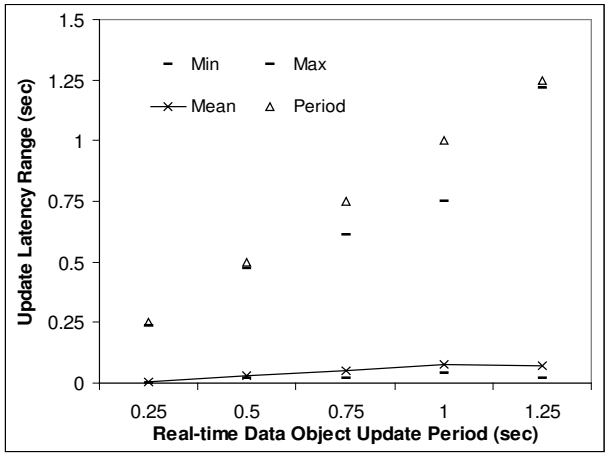


Figure 7: Update Latency Range: Intruders Disguise as User Transactions (2.0 tps)

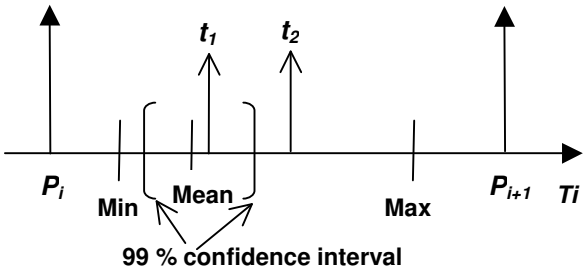


Figure 8: Update Latency Range

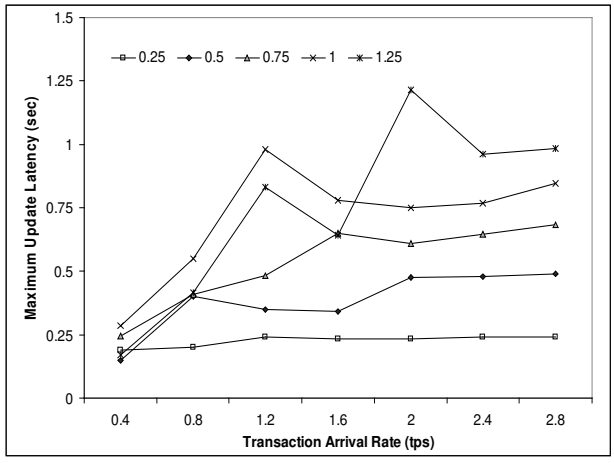


Figure 9: Maximum Update Latency: Intruders Disguise as User Transactions

On the other hand, the effectiveness of the update latency range is dependent on two factors. One is the workload situation of the system. If the system is heavily loaded, the update latency range will become wider and the discriminating power will be reduced accordingly. Figure 9 shows the maximum value of the update latency with respect to background workload adjusted by the user transaction arrival rate.

Generally, as the workload increases, the update latency range approaches the duration of the period. When the workload is light, the update latency range is narrow and the chances for intruders to be falsely classified are therefore low. Another factor is the likelihood for a sensor transaction to be blocked when the corresponding temporal data object is read-locked by a user transaction. In addition to the arrival rate of the user transaction, the period and therefore the deadline of the sensor transaction also affects this probability. A sensor transaction with a long period and therefore a late deadline is more likely to be blocked or preempted by user transactions than a sensor transaction with a short period and an early deadline. In Figure 9, it can be found that those sensor transactions with long periods may, by chance, get a higher maximum update latency during the training phase. Although it may weaken the discriminating power of the measure, the "time semantics" property remains effective in detecting intruders.

5.4 Inter-Update Time

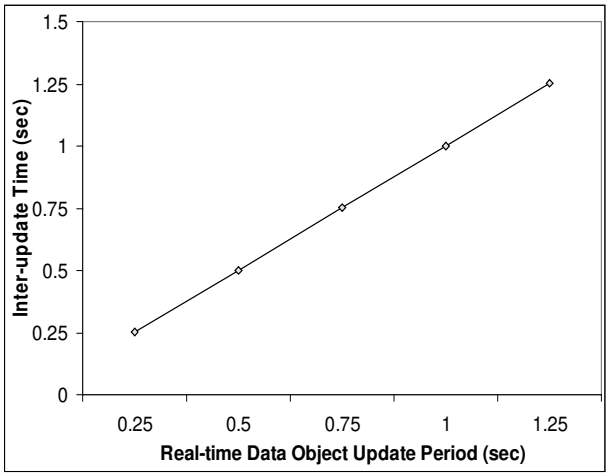


Figure 10: Mean Update Inter-arrival Time Intruders Disguise as User Transactions

Another metric we have experimented with is the inter-update time that is shown in Figure 10. The mean value of the inter-update time agrees well with the period duration. For instance, the mean inter-update time is 0.7513 with 0.275% width for a 99% confidence interval for sensor transactions with a period of 0.75 second. The original intention of using inter-update time is to supplement update latency such that the missing intruders can be detected. However, we observed that there is not much improvement in using inter-update time along with update latency. The discriminating power of inter-update time is overlapping with and relatively weaker than that of update latency. For

instance, if both intruder and normal sensor transaction arrive within the update latency range in order, the inter-update time cannot help to distinguish between them.

6. Conclusions

In this study, we have exploited the time semantics of temporal data objects in real-time database systems to detect intruders that do not have such knowledge. Even if the intruder were to determine the time signature (update rate), the system can change the rate within acceptable minimum and maximum rate bounds. We also have characterized the normal behavior of sensor transactions in accessing temporal data objects. The experimental results confirm that the time signature of an *update latency range* is an effective discriminating metric to identify intruders that disguise themselves as user transactions and sensor transactions. In addition to having a high true negative rate, a low false positive rate can be kept to alleviate the problem of false alarms, which is a real concern for wide applications of IDS. Although the update latency range does not change dynamically, it can be adjusted by recapturing the normal behavior of a new set of training data when the workload or the working pattern of the system changes. On the other hand, the 99% confidence interval of update latency is found not to be as effective as the range because of the bursty arrival pattern of sensor transactions. There may exist some other complex statistically quantifiable measures to model the normal behavior. However, in practice, an intrusion detection system that can run on-line and use minimal computing resources is highly desirable.

Although the experiments show that not all intruders can be identified, the results confirmed that this approach is feasible. It may be possible to identify all intruders by building additional layer that exploits another characteristics of real-time database systems, which is orthogonal to time semantics used in our study. For instance, lock holding time can be used to detect anomalously long lock holding by intruders.

Although our study has focused on sensor transactions and temporal data objects, our intrusion detection technique can be generalized to be applicable to user transactions and non-real-time secure data objects. For instance, pre-defined "time semantics" can be introduced to secure data object such that operations on the data object are only allowed for a short interval in a period of time. Otherwise, additional security measure is imposed before the access. In other words, the secure data object can be made sensitive to both the time and duration of access. After the history of the access behavior of those authorized transactions with the knowledge of the time semantics of the secure data objects is learned, given a certain workload environment, the system can detect intruders without such knowledge.

7. References

[1] Bodlaender, M., Stok, P., and Son, S. H., "A Transaction-Based Temporal Data Model for Real-Time Databases," *Proc. of Int. Workshop on Parallel and Distributed Real-Time Systems*, Switzerland, 1997.

[2] Denning, D. E., "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222-232, 1987.

[3] Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A., "A Sense of Self for Unix Processes", *Proc. of 1996 IEEE Symposium on Computer Security and Privacy*, 1996.

[4] Forrest, S., Hofmeyr, S., and Somayaji, A., "Computer Immunology," *Communications of the ACM*, vol. 40, no. 10, pp. 88-96, 1997.

[5] Goan, T., "A Cop on the Beat: Collecting and Appraising Intrusion Evidence," *Communications of the ACM*, vol. 42, no. 7, pp. 46-52, 1999.

[6] Hofmeyr, S. A., Somayaji, A., and Forrest, S., "Intrusion Detection using Sequences of System Calls," *J. of Computer Security*, vol. 6, pp. 151-180, 1998.

[7] Jojodia, S., "Database Security and Privacy," *ACM Computing Surveys*, vol. 28, no. 1, pp. 129-131, 1996.

[8] Lunt, T., "A Survey of Intrusion Detection Techniques," *Computers and Security*, vol. 12, pp. 405-418, 1993.

[9] Lunt, T., "Securing the Information Infrastructure," *Communications of the ACM*, vol. 39, no. 6, pp. 130, 1996.

[10] Pissinou, N., Makki, K., and Park, E. K., "Towards a Framework for Integrating Multilevel Secure Models and Temporal Data Models," *Proc. 3rd Int. Conf. Information and Knowledge Management*, pp. 280-7, 1994.

[11] Son, S. H., and Chaney, C., "Supporting the Requirements for Multilevel Secure and Real-time Databases in Distributed Environments," *Database Security: Status and Prospects*, T. Y. Lin and S. Qian (eds.), Chapman and Hall Publishing, pp. 73-91, 1998.

[12] Son, S. H., Mukkamala, R., and David, R., "Integrating Security and Real-Time Requirements using Covert Channel Capacity," to appear in *IEEE Transactions on Knowledge and Data Engineering*, 2000.

[13] Song, X., and Liu, J. W. S., "Maintaining Temporal Consistency: Pessimistic vs. Optimistic Concurrency Control," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 5, pp. 786-796, 1995.

[14] Stankovic, J. A., Son, S. H., and Hansson, J., "Misconceptions About Real-Time Databases," *Computer*, vol. 32, no. 6, pp. 29-36, 1999.

[15] Stillerman, M., Marceau, C., and Stillman, M., "Intrusion Detection for Distributed Applications," *Communications of the ACM*, vol. 42, no. 7, pp. 62-69, 1999.

[16] Warrender, C., Forrest, S., Pearlmutter, B., "Detecting Intrusions Using System Calls: Alternative Data Models," *Proc. of 1999 IEEE Symposium on Security and Privacy*, Oakland, California, May 1999.

[17] Xiong, M., Ramamritham, K., "Deriving Deadlines and Periods for Real-Time Update Transactions," *Proc. of IEEE Real-Time Systems Symposium*, Phoenix, Arizona, Dec. 1999.