# Towards Database Firewalls*

Kun Bai, Hai Wang, and Peng Liu

The School of Information Science and Technology,
Pennsylvania State University,
University Park 16802 PA
{kbai, haiwang, pliu}@ist.psu.edu

**Abstract.** Authentication based access control and integrity constraints are the major approaches applied in commercial database systems to guarantee information and data integrity. However, due to operational mistakes, malicious intent of insiders or identity fraud exploited by outsiders, data secured in a database can still be corrupted. Once attacked, database systems using current survivability technologies cannot continue providing satisfactory services according to differentiated information assurance requirements. In this paper, we present the innovative idea of a database firewall, which can not only serve differentiated information assurance requirements in the face of attacks, but also guarantee the availability and the integrity of data objects based on user requirements. Our approach provides a new strategy of integrity-aware data access based on an on-the-fly iterative estimation of the integrity level of data objects. Accordingly, a policy of transaction filtering will be dynamically enforced to significantly slow down damage propagation with minimum availability loss.

## 1 Introduction

Data integrity, availability and confidentiality are the three major issues that have been paid much attention in database security research. To protect the data integrity, multi-layer approaches are proposed, from hardware, OS, DBMS to transaction level. Mainly, there are two research focuses. One is *from-scratch*, the other is *off-the-shelf*. Approaches presented in [1],[2],[3] are to close the security holes on hardware, OS and DBMS, respectively, from the *from-scratch* direction. [4] and [5] propose techniques to deal with data corruption and storage jamming effectively on OS-level intrusions. Unfortunately, these technologies can not be applied to handle authorized but malicious transaction.

[6] introduces an intrusion-tolerant database (ITDB) system architecture on the transaction-level. It is noticeable that ITDB architecture is complicated because of the specific database vulnerability known as *damage spreading*. That is, the result of a transaction can affect the execution of some later transactions, directly or indirectly, through *read* and *write* operations.

* This work was supported by NSF CCR-0233324, NSF ANI-0335241, and Department of Energy Early Career PI Award.

S. Jajodia and D. Wijesekera (Eds.): Data and Applications Security 2005, LNCS 3654, pp. 178–192, 2005.

Since infected data objects can cause more damage through *read* and *write* operations, which, in turn, could lead to wrong decision and disastrous consequences, data corruption becomes a severe security problem in critical data applications, such as air traffic control, banking and combat-field decision making system. Furthermore, data corruption is not only an issue of data integrity issue, but also a concern of data availability. For example, in some cases, the purpose of an attack is just to deny the service. Generally, when the real-world database application is under an attack, the services the system provides have to be shut down to recover from the disaster. Thus, the system availability sacrificed in order to maintain the data integrity. A vast majority of research has been done on how to survive data corruption from malicious attacks and recover the data integrity and availability in an off-line manner. However, limited attention has been drawn to provide various database services in agreement with differentiated information assurance requirements while the system is being healed.

In this paper, we present a novel idea of database firewall that, in contrast to previous research, uses different strategies to prevent damage from spreading to other territories of the database in terms of tables, records and columns. The idea is to quickly estimate the integrity levels of data objects and use such integrity levels to smartly filter off the transactions that would spread damage according to a specific firewall policy upon the time a malicious transaction is detected. A unique feature of our approach is that transaction filtering is not universally enforced and the enforcement domain is dynamically adjusted so that maximum availability can be provided without jeopardizing integrity. According to a user requirement of quality of information assurance (QoIA), we not only provide a significant improvement of data availability, but also guarantee the integrity of data objects stored in the database. The database firewall framework is illustrated in the context of the transaction level in ITDB architecture.

The rest of this paper is organized as follows. In section (2), we review the background and related work. In section (3), we present the design issue of the database firewall. In section (4), we propose our naive estimator model and estimation algorithm. In section (5), we demonstrate some preliminary results. In section (6), we conclude the paper and future work.

## 2 Background and Related Work

Intrusion detection system (IDS) has attracted many researchers ([8],[9],[10]). In general, IDSs monitor system activities to discover attempts to gain illicit access to systems or corrupt data objects in systems. Roughly, the methodologies of IDS are in two categories, *statistical profile* and *known patterns of attacks*. However, intrusion detection systems have a few noticeable limitations: (1) Intrusion detection makes the system attack-aware but not attack-resistant. (2) Achieving accurate detection is usually difficult or expensive. (3) The average detection latency in many cases is too long to effectively confine the damage. To overcome these limitations, a broader perspective has been introduced, namely an intrusion tolerance database system [6].

Other than the ITDB approach, traditional recovery mechanisms execute complete rollbacks to undo the work of benign transactions as well as malicious ones when the malicious transactions are detected. Therefore, although rolling back a database to a previous checkpoint can remove all the corrupted data, the work of all the legitimate transactions which commit after the checkpoint is lost. This kind of approach would further exacerbate the situation of denial of service. [11] provides a recovery algorithm that, given a specification of malicious, unwinds not only the effects of each malicious transaction but also the effects of any innocent transaction that is directly or indirectly affected by a malicious transaction. A significant contribution of [11] is that the work of remaining benign transactions is saved. However, the fact is that transaction execution is much faster than detection and reparation. This indicates that the entire process of recovery could both take a relatively long time to finish and also repeat repairing certain data objects over and over again due to damage spreading. Thus, the data availability could be significantly lost due to this long latency.

[12] present an innovative idea known as multiphase damage containment. Upon the time a malicious transaction (denoted as $B_i$) is detected, in contrast to reactive damage containment, [12] uses one containing phase (denoted as initial containment) to proactively contain the data objects that might have been corrupted. In addition to this first phase, one or more later uncontaining phases (denoted as containment relaxation) will release the objects that are mistakenly contained during the first phase. This approach can guarantee that no damage caused by malicious transaction $B_i$ will spread to any new update. However, an inherent limitation of multiphase containment is that this method could cost substantial data availability loss. Because the *initial containment* phase needs to instantly confine every data object possibly infected by $B_i$ within a time window starting upon the commit of the malicious transaction $B_i$ and ending at the detection of $B_i$, there is no time for the confining phase to precisely pinpoint the set of damaged data objects.

To overcome the limitations of the multiphase containment approach and to provide more data availability, delivering services by taking QoIA requirements into account seems to be a solution. In order to keep services available during attacks, it will be beneficial to continue allowing access to confined data objects during the repair time window. However, this needs to be conducted very carefully since a confined data object could have been corrupted. Thus, certain security rules and policies of access are required to achieve this original intention. [13] has taken the first step towards this goal. In this paper, we extend this topic and present database firewall technique as a solution to increase the data availability without imposing risks to applications users and degrading the system performance and data integrity.

## 3 Database Firewall Design

In this section, we first formalize several important concepts and the various problems studied in this paper, and then present the framework of database

firewall. The idea of a database firewall can be better described in the context of an intrusion tolerant database system (ITDB) on the transaction level. Since this framework is an extension of the ITDB architecture, it inherits the features from ITDB that it could not directly defend against attacks from low level, such as OS and DBMS level attacks. However, when most attacks come from malicious transactions, our framework is effective. Moreover, the existing low level mechanisms can be easily integrated into our database firewall framework.

### 3.1 Theoretical Model

A *database* system is a set of data objects, denoted as DB=$\{o_1, o_2, \ldots, o_n\}$. A transaction $G_i$ is a partial order with ordering relation $<_i$, where

1. $G_i \subseteq \{(r_i[o_x], w_i[o_x]) | o_x \text{ is a data object}\} \cup (a_i, c_i)$;
2. if $r_i[o_x], w_i[o_x] \in G_i$, then either $r_i[o_x] <_i w_i[o_x]$, or $w_i[o_x] <_i r_i[o_x]$;
3. $a_i \in G_i$ iff $c_i \notin G_i$.

and *r,w,a,c* relate to the operation of *read, write, abort,* and *commit*, respectively. The (usually concurrent) execution of a set of transactions is modeled by a structure called a history. Formally, let $G = \{G_1, G_2, \ldots, G_n\}$ be a set of transactions. A complete history $H$ over $G$ is a partial order with ordering relation $<_H$, where:

1. $H = \cup_{i=1}^{n} G_i$;
2. $<_H \supseteq \cup_{i=1}^{n} <_i$.

Since aborted transactions have nothing to do with database firewalls, for the sake of simplicity we assume every transaction commits. Two transactions conflict if they both have an operation on the same object, and one of them is *write*. Also, the correctness of a history is typically captured by the notion of *serializability*[14]. One assumption is that strict *two-phase locking* (2PL) is used to produce serializable histories where the commit order indicates the serial order among transactions.

First, how an object is damaged is defined in a conservative way. That is, every object updated by a malicious transaction is damaged, and that if a good transaction reads a damaged object, then every object updated by the good transaction is damaged. Next, a transaction dependent relation is denoted as follows. In a history composed of only committed transactions, a transaction $G_i$ is *dependent upon* another transaction $G_j$ if there exists an object $o_x$ such that $G_i$ reads $o_x$ after $G_j$ updates it, and there is no transaction that updates $o_x$ between the time $G_j$ updates $o_x$ and $G_j$ reads $o_x$. Finally, it is assumed that every data object modified by $G_i$ will be read by $G_i$ first. Thus, there is no *blind* writes.

### 3.2 Motivation and Challenges

As networks enable more and more applications and are available to more and more users, they become ever more vulnerable to a wider range of security

threats. Thus, to combat those threats and ensure that applications are not compromised, security technologies such as network firewalls play a critical role in today's networks. Likewise, a broad span of research from authorization, to inference control, to multilevel secure database, and to multilevel secure transaction processing has addressed primarily on how to protect the security of a database. However, a very important vulnerability of database security, known as *damage spreading*, has been omitted by these researches. Database firewall technique is needed not only because malicious transactions can compromise data objects, but also because innocent transactions can accidentally spread the damage. Formally, *damage spreading* occurs because any good transaction reading a corrupted data object $o_x$ can spread the damage on $o_x$ to the data objects it updates. In this way, the spreading can be exponential. Still, the effect caused by a malicious transaction itself to a database is limited. Thus, it is the transactions that spread the effect that matter. Efforts have been made in existing data containment and damage assessment technologies to stop spreading and recover systems. However, data containment and damage assessment take a substantial amount of time. Thus, the loss of data availability is significant. Database firewall technique takes a step further to reinforce the above approaches by filtering the incoming transactions to simultaneously stop potential damage spreading at the doorway and to improve the data availability according to a certain security policy.

In sum, a database firewall should include at least three components: *Integrity Estimator, Firewall Manager* and *Access Policy Manager*. One of the challenges to guarantee the success of database firewalls is to design an efficient integrity level estimation algorithm, which can quickly and accurately estimate the data integrity without losing security. In this paper, a naive approach to achieve this goal is presented.

### 3.3 Architecture of Database Firewall

To develop the database firewall framework that can provide more data availability, there are several fundamental issues needed to be addressed and solved. First, how to formalize the integrity level model and estimate the data integrity during attacks. Second, how to constitute the security policy and access rulesets using estimated data integrity level. Third, how to manage the tradeoff between performance and security.

**Database Firewall Components.** As shown in *figure* (1), the database firewall architecture is built upon the top of a traditional "off-the-shelf" DBMS. Within the framework, *Intrusion Detector* (ID) identifies malicious transactions based on the operation records stored in the log. *Damage Assessor* (DA) locates the damage caused by the detected malicious transactions. *Damage Repairer* (DR) repairs the located damage using some specific *cleaning* transactions. *Integrity Estimator* (IE) estimates the integrity level of data objects. *Access Policy Manager* (APM) works as a *proxy* for decision making of data objects access. *Firewall Manager* (FM) functions when *Intrusion Detector* detects malicious

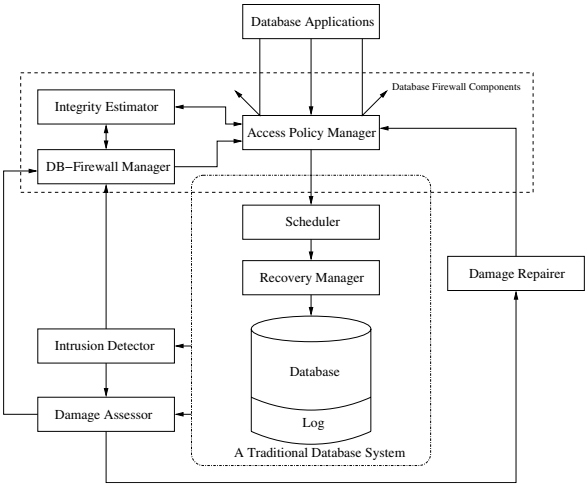



**Fig. 1.** Database Firewall Architecture

transactions. After the firewalls are built up, *Firewall Manager* triggers *Integrity Estimator* to start estimating the integrity of data objects and consequently force *Access Policy Manager* to set up access rulesets to restrict the access to the data items that are confined in firewalls according to a new policy. At each step of integrity estimation, the firewalls update themselves in co-response to the changes of data integrity level. Accordingly, any new transaction submitted by a user will comply with the new policy. Through several steps, *Integrity Estimator* will finally converge to the final solution, which has either a set of precise integrity of data objects or a set of approximate integrity of date objects.

### 3.4 Transaction Filtering Policies and Mechanism

In this section, an innovative mechanism for implementing security control which guards the door of database systems and prevents potential damage spreading from occurring is introduced. By conventional definition of firewall in network domain, a firewall is a system or group of systems that enforce an access control policy between two or more networks. Its operations are mainly based on three technologies: packet filtering, proxy server and stateful packet filtering. Similarly, in database security domain, particularly in our database firewall framework, a firewall operates based on transaction filtering technique. In addition, unlike a network firewall, which checks packet status, transaction filtering relies on the integrity level of data objects.

**Integrity Level Model.** When a malicious transaction $B_i$ is detected, the data objects in the database could be in several different situations. In this section, an idea is presented to define the model illustrating the integrity of data objects.

1. **Data objects Integrity.** A data object could be either good or corrupted after the database system is attacked. Thus, it is straightforward to denote that the integrity of an object $o_i$ $(1 \leq i \leq n)$ is good at particular time $t$ as $I(o_i, t) \in \{G, B\}$, where $G$ is *Good* and $B$ is *Bad* for short. It is apparent that when a malicious transaction is captured, any transaction whose commit time is out of a time window, starting from the time point when $B_i$ enters the database to the moment of its committing, is not infected, and the data objects belonging to the transaction are regarded as good objects.

   However, the status of those data objects that belong to transactions which commit within the time window are a little more complicated. It is difficult to attain such knowledge that data integrity can be precisely calculated in a short period of time. Methods, such as [12], mentioned in previous section (2), can precisely distinguish the integrity of each data object through several phases. However, safety comes at the sacrifice of significant data availability. This contradicts the goal of database firewall framework. Therefore, instead of deterministically marking the integrity of data objects, a practical integrity model that uses probabilistic estimation is favored. This model is applicable because the damage spreading is strongly related to the writeset of the malicious transaction $B_i$, denoted as $W_{B_i}$, and also relies on the transaction arrival and dependency patterns. For this reason, previous histories can be used to estimate the probability that a data object is good as the data integrity during an attack.
2. **Practical Integrity Model.** In this model, a data object $o_i$'s integrity at a particular time $t$ is shown in the equation.

$$I(o_i, t) = (1 - \frac{1}{R(t)}) \times 100\%, R(t) \geq 1 \tag{1}$$

   Where, $R$ is the number of patterns matched with or similar to an attack pattern. We call $I(o_i, t)$ the data object $o_i$'s *integrity level*, and $0 \leq I(o_i, t) \leq 1$. Integrity level of data object $o_i$ indicates that the probability of $o_i$ is good when a specific attack pattern occurs. For example, when $R(t) = 1, I(o_i, t) = 0$, it means the identical patterns are found, and the data object $o_i$ is corrupted. Thus, the integrity of a data object $o_i$ could be in one of the following three categories:

$$I(o_i, t) = \begin{cases} 100\% \ t \notin [t_S^i, t_E^i] & \text{estimated} \\ 50\% \ \ t \in [t_S^i, t_E^i] & \text{estimated} \\ 0\% \ \ \ t \in [t_S^i, t_E^i] & \text{identified} \end{cases} \tag{2}$$

   Here, for the definition of $t_S^i, t_E^i$, please refer to section 3.4. With the above analysis about data integrity, in order to estimate the integrity of a data object, our research becomes to find answers to following three questions: What is an attack pattern? How does the integrity estimator use the patterns? How do we match two attack patterns? These concerns will be addressed in a later section (4).

**Database Firewall Security Policy.** A specific and strongly worded security policy is vital to the pursuit of internal data integrity. This policy is a subset of the database access contorl policy and never will rule over an access contorl policy, such as authorization, but should govern everything from acceptance of accessing data objects to response scenarios in the event a security incident should occur, such as policy updating upon a new attack.

Ideally, a database firewall security policy dictates how transactions traffic is handled and how filtering ruleset is managed and updated. Before a policy is created, a risk analysis on the database system must be performed to gain knowledge for the vulnerabilities associated with databases. For instance, we know one of the vulnerabilities in database security is the *damage spreading*. It is when a transaction, even if it is a legitimate one, accesses a corrupted data object that the damage will be spread to any other data object this transaction touches, directly or indirectly. Then, to limit the potential damage spreading, firewall policy needs to create a ruleset to restrict the entrance of transactions that could compromise other data objects while letting other transactions enter to achieve maximum throughput.

For example, suppose a transaction $G_1(t, tp) = r_1[o_x]r_1[o_y]w_1[o_y]$ requires to enter the database, where *tp* is transaction type. If it is known that the data object $o_x$ has been corrupted at this momment, then our policy checker will screen the transaction and be aware if the request can be granted using the ruleset.

**Definition** 1 : Integrity Filtering List, $\hat{\mathrm{I}}$= $\{i_1(o^{i_1}_{x_1}, o^{i_1}_{x_2}, .., o^{i_1}_{x_m}), i_2(o^{i_2}_{y_1}, o^{i_2}_{y_2}, .., o^{i_2}_{y_n}), ..\}$, where $i$ is a set with data objects on same integrity level, and $o_i$ is a data object associated with the integrity level $i$. The ruleset is defined as follows:
**Rule** 1 : $\forall$ transaction $G$, if $\exists$ data object $o_x \in R_G$, and $R_G \bigcap \hat{\mathrm{I}} \neq \emptyset$, and if $W_G \neq \emptyset$, DENY;
**Rule** 2 : $\forall$ transaction $G$, if $\exists$ data object $o_x \in R_G$, and $R_G \bigcap \hat{\mathrm{I}} \neq \emptyset$, and if $W_G = \emptyset$, and if $i < Q$ then DENY, otherwise GRANT;
**Rule** 3 : $\forall$ transaction $G$, if $\nexists$ data object $o_x \in R_G$, and $R_G \bigcap \hat{\mathrm{I}} = \emptyset$, GRANT;
Here, $Q$ is QoIA required by applications. $R_G, W_G$ is the readset, writeset of a transaction, respectively. What we have presented here is a sample ruleset. We should be aware that firewall rulesets tend to become increasingly complicated with age.

**Transaction Filtering Mechanism.** In many cases when an attack is detected, not every data object in database is corrupted. Thus, simply applying the firewall ruleset to screen every incoming transactions is not wise. Here, we introduce a novel concept called *firewall time window*.

In the database firewall framework, for each detected attack, *Firewall Manager* has a life cycle with three different phases: *Firewall Generation*, *Firewall Mergence* and *Firewall Withdraw*. During the first phase, upon the time when a malicious transaction $B_i$ is detected, *Firewall Manager* is notified to generate a firewall. A firewall time window $[t^i_S, t^i_E]$ is denoted as $\mathfrak{W}_i$. Here, the $[t^i_S, t^i_E]$ is defined as follows:

**Definition** 2 : Firewall Time Window $\mathfrak{W}_{\mathfrak{i}}$ of $B_i$, denoted as $[t_S^i, t_E^i]$, is defined as follows: $t_S^i$ is the time when $B_i$ starts; $t_E^i$ is the time when malicious transaction $B_i$ is detected.

For example, suppose a transaction $G_1(t, tp) = r_1[o_x]r_1[o_y]w_1[o_x]w_1[o_z]$ requires to enter a database, if it is found that $t_{o_x}^u$ is within the scope of firewall time window $[t_S, t_E]$, the ruleset is further checked for security concerns. Otherwise, the permission of entrance to the database can simply be granted. Here, $t_{o_x}^u$ is the time when data object $o_x$ was updated.

**Firewall Updating Mechanism.** At phase two, if there are multiple malicious transactions detected during a period of time, there might exist multiple firewalls, and *Firewall Manager* will force the multiple firewalls to merge together according to certain rules. By doing this, *Access Policy Manager* can efficiently manage multiple versions of access policy. A set of malicious transactions is denoted as $B_{i1}, B_{i2}, ..., B_{ik}$. For each firewall time window, the mergence rules are defined as follows:

**Mergence Rule** 1 : Firewall time window $[t_S^i, t_E^i]$ is ahead of $[t_S^j, t_E^j]$ if $t_E^i < t_S^j$. Firewall time window $\mathfrak{W}_{\mathfrak{i}}$ and $\mathfrak{W}_{\mathfrak{j}}$ are *overlap* if no one is ahead of another. $\mathfrak{W}_{\mathfrak{i}}$ includes $\mathfrak{W}_{\mathfrak{j}}$ if $t_S^i < t_S^j$ and $t_E^i > t_E^j$.

The rule of firewall mergence is defined as follows:

**Mergence Rule** 2 : A set of firewall time windows can be merged as one if for any two time windows $\mathfrak{W}_{\mathfrak{i}_\mathfrak{m}}$ and $\mathfrak{W}_{\mathfrak{i}_\mathfrak{n}}$, $(m < n)$, there is a sequence of firewall time windows $\mathfrak{W}_{\mathfrak{j}_1}, \mathfrak{W}_{\mathfrak{j}_2}, ..., \mathfrak{W}_{\mathfrak{j}_\mathfrak{k}}, ..., \mathfrak{W}_{\mathfrak{j}_\mathfrak{l}}$, such that they are within the set where $\mathfrak{W}_{\mathfrak{i}_\mathfrak{m}}$ and $\mathfrak{W}_{\mathfrak{j}_1}$ overlap, $\mathfrak{W}_{\mathfrak{j}_\mathfrak{k}}$ and $\mathfrak{W}_{\mathfrak{j}_{(\mathfrak{k}+1)}}$ overlap, and $\mathfrak{W}_{\mathfrak{j}_{(\mathfrak{k}+1)}}$ and $\mathfrak{W}_{\mathfrak{j}_\mathfrak{l}}$ overlap.

By applying this firewall mergence ruleset, the framework dynamically adjusts the security policies and rulesets corresponding to the changes of firewalls.

In the third phase, there is a condition when it is satisfied, the *Firewall Manager* will stop restricting access to any data objects within the firewall time windows (That is, when *Damage Repairer* finishes repairing the located corrupted data objects). In response to the withdraw of firewall, *Access Policy Manager* will reset the access policy to the lowest level of restriction of data access, and the database system performs in the normal way until the next malicious transaction is detected.

## 4 Integrity Level Estimation

One critical issue to guarantee success of *Integrity Level Estimation* success is timing. The more time the estimation algorithm spends, the more accurate the estimation result can be, but the less data availability the database system can provide. Thus, instead of releasing a final solution of integrity estimation at the conclusion, the algorithm gives out several versions iteratively along the process.

Now, we propose our integrity estimator model and the first naive estimation algorithm (1) that balances the tradeoff between performance and security.

**Integrity Estimator.** *Figure* (2) illustrates the details of estimator component. Basically, there are two subcomponents: One is offline processor; the other

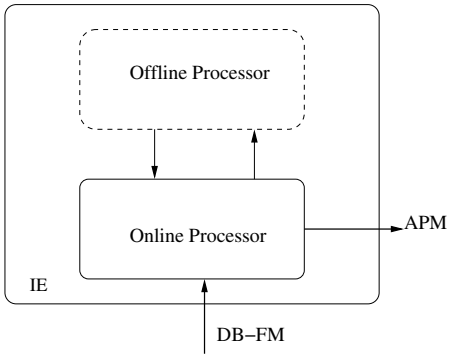



**Fig. 2.** Integrity Estimator Component

is online processor. Offline processor usually is executed after *Damage Repairer* finishes repairing and then triggers the *Database Firewall Manager* to withdraw the firewalls. In general, to gather knowledge about previous attacks and to save time for online processor to quickly and precisely estimate the data integrity, offline processor deals with all kinds of information it can obtain from history logs, IDS reports, customer profiles and database schemes. In this paper, it is assumed that offline processor only process the histories stored in database and subtracts valuable attributes from them, such as the transaction dependency graphes, attacking time and statistic data (the number of corrupted data objects, frequency of a data object being corrupted, the number of distinct values and transaction types, for example). The above information is called an *Attacking Pattern*, or *Fingerprint*. Once an attacking is detected, online processor in *Integrity Estimator* starts estimating data integrity based on both the knowledge the offline processor has obtained and the information of new attacking history. We define *Attacking Pattern* and *Spreading Pattern* as follows:

**Definition** 3 : *Attacking Pattern* $p = (R_j, W_j, a_j^1, a_j^2, \ldots, a_j^{m-1}, a_j^m, a_j^{m+1}, \ldots)$.

**Definition** 4 : *Spreading Pattern* $P$ is a dependency related sequence of transactions, $P_i = \{p_{B_i}, p_1, p_2, \ldots, p_{n-1}, p_n, p_{n+1} \ldots\}$. Where, $R_j, W_j$ is the readset, writeset of a transaction, respectively; $B_i$ is a malicious transaction, and $a_i$ is a valuable attribute that depicts a particular dimension of a transaction, such as occurrence frequency of a special value or the number of distinct values. And, $W_{n-1} \bigcap R_n \neq \emptyset$.

Algorithm (1) describes the naive approach of how to estimate data object integrity. In general, this algorithm is a pattern-match based approach. A vector containing spreading patterns is created by offline processor based on the histories it obtains. Basically, this algorithm scans the spreading patterns to compare the attacking pattern from a newly detected attack with the one in each spreading pattern in the vector. If a match is found, the R will be increased by one; otherwise, the unmatched spreading pattern is trimmed off the vector.

In addition, the confined data set C and the number of matched patterns R update correspondingly. Since this is a pattern-matched approach, an unavoidable problem is what to do in the absence of a matched pattern. From the mathematics perspective, R in equation 1 can not be zero. But, in the algorithm if R is equal to zero, it indicates the newly detected attacking pattern is one that had never occurred before. In this scenario, the algorithm stops estimating and notifies *Firewall Manager* to reset the firewall time window because damage had probably already been spread out by this moment. A possible solution to this problem is to apply a containment approach, such as multi-pahse containment method, to precisely distinguish the integrity of each data object, invoke the offline processor to consume the new attack and add this pattern to the vector.

**Algorithm 1** Integrity Level Estimation Algorithm Pseudo Code

**Require:** $V[k]$ : spreading pattern vector. $P_{new}$:newly detected attack ▷ S is the corrupted data objects of spreading pattern i in V

**function** ILESTIMATOR($V, P_{new}$)
  $C = \emptyset, R = 0$ ▷ C → Confined data objects set
  **for** $i \leftarrow 1, n$ **do** ▷ Scan the pattern vector
    $p \leftarrow P_{new}[i]$
    **for** $j \leftarrow 1, k$ **do** ▷ Compare each spreading pattern
      $p_v \leftarrow V[j]$
      **if** $p_v \equiv p$ **then**
        $R(t) \leftarrow (R(t) + 1)$
        $C \leftarrow C \cup S_{V[j]}$
      **else**
        $V \leftarrow \overline{V - V[j]}$ ▷ Trim the unmatched pattern off the vector
        $C \leftarrow \overline{C \cap S_{V[j]}}$
      **end if**
    **end for**
    **if** $R(t) = 0$ **then**
      break;
    **else**
      $\forall o_x \in C \leftarrow (1 - \frac{1}{R(t)})$ ▷ Set the integrity of data objects
    **end if** ▷ Mark the integrity of data objects
    APM updates new policy
  **end for**
**end function**

## 5 Experiments and Results

In this section, the experiment results are demonstrated . In order to measure the effectiveness and performance of our proposed naive method, comprehensive experiments have been conducted on synthetic data sets generated according to a modified TPCC standard.

### 5.1 Generation of Experimental Data

For the experiment, synthetic data set has been used. All data are generated based on a modified TPCC dependency relationship, as shown in *figure* (3). Also, the data sets have $1M$ transactions history. 300 different patterns are summarized out of this history. For each pattern, the number of transactions varies in a range from 2000 to 3500. Furthermore, there are two possible consequences regarding an approaching attack. One, a new attack is a duplicate of a previous one, which implies that there is a previous version recorded in the history. Thus, it becomes a question whether or not identical twins can be found out of the previous patterns. Two, a new attack is a mutant of an existing version of attack. Thus, it becomes whether or not the similar ones can be distinguished. In addition, in these experiments, it is assumed there is only one malicious transaction $B_i$ at each time. So, firewall mergence is not taken into consideration at current stage.

### 5.2 Experiment Results

*Figure* (4-[a,b,c].1) illustrates the results of the first possible attack pattern, which is a copy of a previous attack, from three different perspectives, objects integrity, system availability and estimation validation, respectively. *Figure* (4-a.1) presents the results of using our naive method. $Obj\_A$, $Obj\_B$ and $Obj\_C$ are the representatives of three sets of data objects. Along the estimation process, data objects in set $Obj\_A$ are first marked as $I = \{1\}$ because they are those data objects that are not touched by transactions; thus, they do not belong to the patterns that are partials of or similar to the newly detected attack pattern. Those data objects in set $Obj\_B$ are assigned to be $I = \{1\}$ later than $Obj\_A$ because the estimator distinguishes that these objects do not belong to corrupted data set when more knowledge is obtained, and then remark their integrity. $Obj\_C$ is the data object set with all corrupted data objects, and it

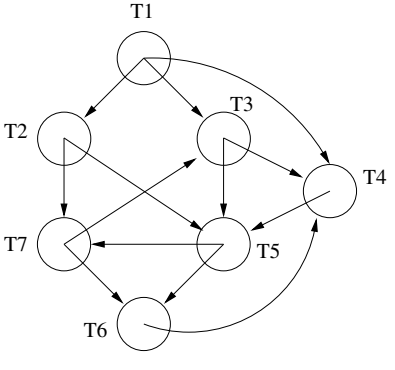


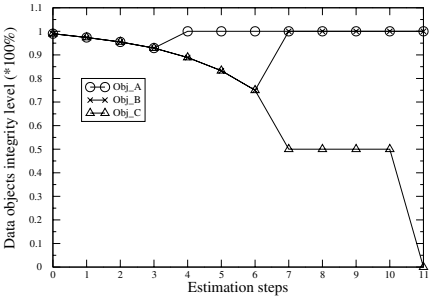

**Fig. 3.** Example Transaction Dependency Graph

shrinks because some objects are remarked and moved to *Obj_A* and *Obj_B* along the estimation process. *Figure* (4-b.1) shows the system availability in terms of the number of accessible data objects with a QOIA requirement of 100%. Corresponding to *Figure* (4-a.1), it can be seen the system availability increases as the integrity of data objects are remarked and moved to *Obj_A* and *Obj_B*. At step 11, the estimator finds the final solution of data integrity, and the availability reaches its highest level. Finally, when the system recovers itself from attacking, the availability goes back to normal level. In this experiment, it is assumed that applications only access data objects with marked integrity equal to $I = \{1\}$. For some applications that are aggressive and are willing to accept multiple levels quality of information assurance (QoIA), the system availability will be even higher. *Figure* (4-c.1) illustrates the progress of estimation validations. It can be seen that at the initial stage of estimation, because of limited knowledge about the newly detected attack, estimation has a relatively high estimation variance (normalized in the range of 0 to 1). However, it will quickly converge to zero (the diagonal denotes the actual errors, which is zero) as the procedure goes on.

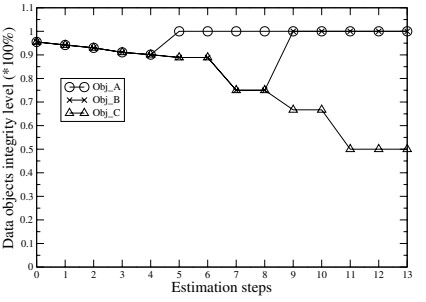
*Figure* (4-[a,b,c].2) demonstrates the results of the second consequence of an attack from the same three aspects. Similarly, *Figure* (4-a.2) presents the results of data integrity using the naive method. In contrast to *Figure* (4-a.1), *Obj_C* does not drop down to zero because the estimator can only find out several similar patterns instead of one, which indicates that a new type of attack is found. Therefore, the estimator will be conservative and inform *Firewall Manager* to reset firewall time window to contain data objects in *Obj_C* in order to prevent damage leakage. Beyond the last step, the database will not continue rely on estimation. Instead, [12] can take over and continue the work. In *Figure* (4-b.2), corresponding to the changes of integrity of data objects, the system availability increases accordingly. *Figure* (4-c.2) demonstrates from another perspective that, unlike the convergence shown in *Figure* (4-c.1), the estimation error does not decline to zero beyond a certain time point when the estimator could not be more accurate on data object integrity. However, even this is a case, we still achieve the goal of improving the system availability.

## 6 Conclusion and Future Work

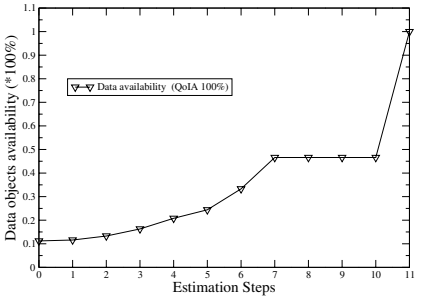
This paper presents an innovative idea of database firewall. Unlike the traditional recovery mechanisms, which shutdown the entire system and recovery itself in an offline manner, our framework can help a database system continue delivering services even when an attack is detected. We have developed a naive but effective approach to use histories and attacking patterns to probabilistically estimate the integrity level of data objects in the face of an attack, instead of deterministically finding the data integrity. However, this naive approach assumes a relative simple attacking pattern. In real world applications, this might be the case. In addition, efficient estimation of data object integrity is also a great challenge. A quick and accurate estimation algorithm is critical to the success of database firewalls.



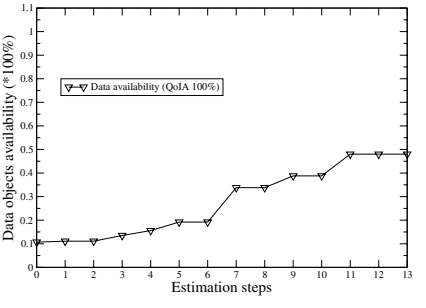

a.1. Objects Integrity



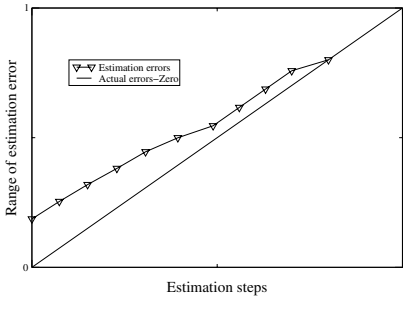

a.2. Objects Integrity



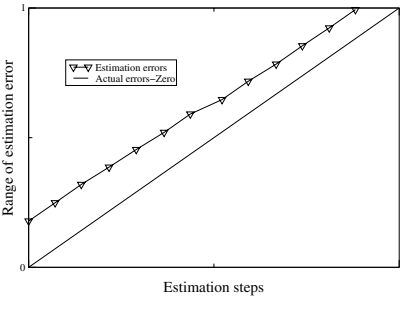

b.1. System Availability




b.2. System Availability




c.1. Estimation Validation




c.2. Estimation Validation

**Fig. 4.** Two different kinds of attacks from three aspects: integrity, availability, validation: 1. When a newly detected attack is matched with previous attacks. 2. When a newly detected attack is similar to the previous attacks

In future work, we plan to formalize the model of the attacking pattern and damage propagation, as well as redesign the integrity level estimation algorithm. A number of further SQL and DBMS enhancements are needed to fully exploit

this interesting topic. We have found that the estimation approach we present may not work efficiently when there are several similar attack patterns or when a new type attack is detected. One possible solution to this problem could be, for example, using a sampling and similarity search technique to find out the final data object integrity solution.

## References

1. S. Smith, E. Palmer, and S. Weingart, "Using a high-performance, programmable secure coprocessor," in Proc. International Conference on Financial Cryptography, Anguilla, British West Indies, 1998.
2. G. C. Necula, "Proof-carrying code," in Proc. 24th ACM Symposium on Principles of Programming Languages, 1997.
3. Z. Shao, B. Saha, and V. Trifonov, "A type system for certified binaries," in Proc. 29th ACM Symposium on Principles of Programming Languages, 2002.
4. D. Barbara, R. Goel, and S. Jajodia, "Using checksums to detect data corruption," in Proceedings of the 2000 International Conference on Extending Data Base Technology, Mar 2000.
5. J. McDermott and D. Goldschlag, "Towards a model of storage jamming," in Proceedings of the IEEE Computer Security Foundations Workshop, Kenmare, Ireland, June 1996, pp. 176–185.
6. P. Liu, "Architectures for intrusion tolerant database systems." in ACSAC, 2002, pp. 311–320.
7. P. W. P. J. Grefen and P. M. G. Apers, "Integrity control in relational database systems: an overview," Data Knowl. Eng., vol. 10, no. 2, pp. 187–223, 1993.
8. H. S. Javitz and A. Valdes, "The sri ides statistical anomaly detector," in Proceedings IEEE Computer Society Symposium on Security and Privacy, Oakland, CA, May 1991.
9. T. Garvey and T. Lunt, "Model-based intrusion detection," in Proceedings of the 14th National Computer Security Conference, Baltimore, MD, October 1991.
10. K. Ilgun, R. Kemmerer, and P. Porras, "State transition analysis: A rule-based intrusion detection approach," IEEE Transactions on Software Engineering, vol. 21, no. 3, pp. 181–199, 1995.
11. P. Ammann, S. Jajodia, and P. Liu, "Recovery from malicious transactions," IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 5, pp. 1167–1185, 2002.
12. P. Liu and S. Jajodia, "Multi-phase damage confinement in database systems for intrusion tolerance," in Proc. 14th IEEE Computer Security Foundations Workshop, Nova Scotia, Canada, June 2001.
13. J. Zhang and P. Liu, "Delivering services with integrity guarantees in survivable database systems," in IFIP WG 11.3 16th International Conference on Data and Applications Security, Cambridge, UK, vol. 256, July 28-31.
14. P. A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, MA, 1987.