Towards Database Firewall: Mining the Damage Spreading Patterns

Kun Bai Peng Liu
College of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802, USA
{kbai, pliu}@ist.psu.edu

Abstract

Access control and integrity constraints are well known approaches to ensure data integrity in commercial database systems. However, due to operational mistakes, malicious intent of insiders or vulnerabilities exploited by outsiders, data stored in a database can still be compromised. When the database is under an attack, rolling back and re-executing the damaged transactions are the most used mechanisms during system recovery. This kind of mechanism either stops (or greatly restricts) the database service during repair, which causes unacceptable availability loss or denial-of-service for mission critical applications, or may cause serious damage spreading during on-the-fly recovery where many clean data items are accidentally corrupted by legitimate new transactions. To resolve this dilemma, we devise a novel mechanism, called database firewall *in this paper. This firewall is designed to protect good data from being corrupted due to* damage spreading. *Pattern mining and Bayesian network techniques are adopted in the framework to mine frequent* damage spreading patterns *and to predict the data integrity in the face of attack. Our approach provides a probability based strategy to estimate the data integrity on the fly. With this feature, the database firewall is able to enforce a policy of transaction filtering to dynamically filter out the potential* spreading transactions.

1 Introduction

Transactional processing systems (e.g., database systems) have become increasingly sophisticated and been critical to most cyber infrastructures, such as banking, e-Commerce, military combat-field command centers, etc. Data availability and integrity are crucial for these infrastructures. However, it is well known that system vulnerabilities cannot be completely eliminated, and such vulnerabilities can be exploited by skillful attackers.

The topic of surviving the database system from an attack has recently received considerable attention because of the increasingly critical role that database systems are playing in web-based services. People have seen more and more cyber attacks for a few reasons. For instance, more critical and valuable information is now processed through the web, which is world wide accessible. In addition, an insufficiently protected web based service is not very difficult to break through [20].

In general, web-based OLTP systems are often a combination of the following components: web servers, databases, and application specific code (e.g., server-side transaction procedures). Usually, on one hand, back-bone software infrastructure (e.g., the web servers and database) is developed by the experienced developers who have comprehensive knowledge of the security. On the other hand, the application oriented code is often developed under tight schedules by programmers who lack security training. Experience with data-intensive applications (e.g., credit card billing, online stock trading) has shown that a variety of attacks can successfully fool traditional protection mechanisms. As we will overview shortly in section 3, there are many possible attacks that could cause the integrity and availability of database systems to be compromised.

Authentication based access control, integrity constraints [7], concurrency control, replication, and recovery mechanisms are the major approaches deployed in current database systems to guarantee the correctness, availability, and integrity of the stored information and data. However, these security techniques are very limited in dealing with data corruption intrusions. For example, access control can be subverted by the inside attacker or the outside attacker who has assumed an insider's identity. Integrity constraints are weak at prohibiting plausible but incorrect data.

Once a database system is attacked, database systems armed with current security technologies cannot continue providing satisfactory services because the integrity of some data objects is compromised. Simply identifying and repairing these data objects by undoing and redoing still cannot ensure the database integrity largely because of a noticeable characteristic of transactional processing systems, namely *damage spreading*. Suppose T_1 is a malicious transaction and its writeset contains corrupted data objects. After

transaction T_2, T_3, T_4 (also called *spreading transactions*) read a data object o_i that has been updated by T_1 (in w_{T_1}), data objects that are updated by T_2, T_3, T_4 will also be compromised. In this way, the damage will spread out and we say that the transactions T_2, T_3, T_4 are affected directly or indirectly by T_1.

As we will see in section 2, current database security technologies are very limited in handling *damage spreading*. In this paper, we aim on the following question: When a database system is identified to be under an attack by intrusion detection systems (IDS), how can "damage spreading" be prevented while continuously providing data services? We deal with transaction level attacks, and assume that a good intrusion detection system with a bounded detection latency is applied. These attacks leave a unique fingerprint after them, namely *damage spreading pattern*. We propose a novel database firewall prototype in this paper, and an algorithm to mine the frequent *damage spreading pattern*, and an algorithm to dynamically estimate the data integrity using frequent spreading patterns. With these features, the database firewall is able to enforce a policy of transaction filtering to dynamically filter out the potential *spreading transactions*. Extensive experimental studies have been conducted to evaluate system prototype developed. It is shown by empirical results that the database firewall can meet the requirements of continuously providing services in accordance with differential requirements of data quality.

The rest of the paper is organized as follows. Related work is addressed in section 2. Some definitions used in this paper, the architecture of the database firewall, and an example of how damage spreads are presented in section 3. The framework of frequent *spreading pattern* mining is presented in section 4. The idea of predicting the integrity of data objects using Bayesian network are presented in section 5. Empirical studies are conducted in section 6. Conclusion and future work are included in section 7.

2 Related Work

Our database firewall prototype is built upon several recent and ongoing researches to achieve the goal of dynamically predicting dirtiness of data and preventing damage from spreading by enforcing filtering policies on the fly.

First critical step towards our goal is intrusion detection (ID), which has attracted many researchers. Existing methodologies of ID are in two categories, *statistical profile* [12] and *known patterns of attacks* [13]. These ID systems indeed make the system attack-aware but not attack-resistant. In addition, they focus on identifying attacks on OS and computer networks and cannot be directly applied to detect malicious transactions. Some works on database ID [8][18][19] are suitable to detect malicious transactions or abnormal behaviors in database systems. However, they are unable to detect the damage spread by executing normal transactions.

In [4], a fault tolerant approach is introduced to survive and recover a database from attacks. A color scheme for marking damage and a notion of integrity suitable for partially damaged databases are proposed to develop a mechanism by which databases under attack could still be safely used. The assumption of this approach is that each data object has an accurate initial damage mark. To overcome these limitations, a broader perspective has been introduced, namely an intrusion tolerance database system (ITDB) [5]. Unlike the color scheme approach, ITDB focuses on dynamically assigning the damage mark and dealing with the negative impact of inaccurate damage marks. Some work has been done on OS-level database survivability. In [6], checksums are smartly used to detect data corruption.

Failure handling aims at guaranteeing the atomicity of database systems. Checkpoint techniques [22] are widely used to preserve the integrity of data stored in databases by rolling back the whole database system to a specific time point. However, all work, done by both malicious and innocent transactions, will be lost. Attack recovery has different goals from failure handling. It focuses on malicious transactions that have been successfully executed. For traditional database systems, *Data oriented attack recovery* mechanisms [21] recover compromised data by directly locating the most recent untouched version of each corrupted data, and *transaction oriented attack recovery* [3] mechanisms do attack recovery by identifying the transactions that are affected by the attack through read-write dependencies and rolls back those affected transactions.

Many database attack recovery methods, such as [21] and [3], stops the database service during repair. To overcome this limitation, in [1], an advanced transaction-oriented attack recovery algorithm is provided that unwinds not only the effects of each malicious transaction but also the effects of any innocent transaction. However, the algorithm [1] does not prevent the damage from spreading during the detection latency or the on-the-fly recovery process. To overcome this limitation, in [2], an innovative multiphase damage containment approach is proposed, which proactively *contains* (i.e., blocks accesses to) all the data records damaged during the detection latency instantly after the malicious transaction is detected. This approach can guarantee that no damage caused by malicious transaction B_i will spread to any new update. However, the limitation of multiphase containment is that it can cost substantial data availability loss due to the *initial containment* phase of this method in which a lot data items can be mistakenly contained. Moreover, it may take a substantial amount of time for those mistakenly contained data items to be uncontained. As a result, current database survivability tech-

niques cause the following dilemma: they either stop (or greatly restricts) the database service during repair, which causes unacceptable availability loss or denial-of-service for mission critical applications, or may cause serious damage spreading during on-the-fly recovery where many clean data items are accidentally corrupted by legitimate new transactions..

3 Preliminaries

In this section, we first review a few most critical security vulnerabilities, then introduce a set of formal definitions used in this paper, and finally show an example of the SQL injection attack that can cause serious damage to a database system.

3.1 The Threat Model

In this paper, we deal with transaction level attacks in database systems. Transaction level attacks are not new. They have been studied in a good number of researches [4][11][14]. Transaction level attacks can be done through a variety of ways. First, the attacks can be done *through web applications*. Among the OWASP top ten most critical web application security vulnerabilities [17], three out of the top 6 vulnerabilities can directly enable the attacker to launch a malicious transaction, which are listed as follows: (1) *Unvalidated Input*; (2) *Cross Site Scripting (XSS) Flaws*; (3) *Injection Flaws*. Second, the attacks can be done *through identity theft*. Finally, the attacks can be done *through insider attacks* and insider attacks can be a more often used means in launching malicious transactions.

3.2 Overview And Motivation

A *database* system is a set of data objects, denoted as DB=$\{o_1, o_2, \ldots, o_n\}$. A transaction T_i is a partial order with ordering relation $<_i$ as defined in [9]. The (usually concurrent) execution of a set of transactions is modeled by a structure called a history. Formally, let $T = \{T_1, T_2, \ldots, T_n\}$ be a set of transactions. A complete history *H* over *T* is a partial order with ordering relation $<_H$, where: (1) $H = \cup_{i=1}^{n} T_i$; (2) $<_H \supseteq \cup_{i=1}^{n} <_i$.

Given a set of similar attacks A = $\{A_1, A_2, ..., A_k\}$ that had happened, the *attack history* for A_i denoted as H_i^A is the transaction sub-history whose write set contains damaged data object sets. An attack history H_i^A, as shown in table 1, starts when A_i happens and ends when A_i is detected and it over *T* is also a partial order with ordering relation $<_H$. $H_i^A = \{T_i < r_i, w_i >, ..., T_k < r_k, w_k >, ...\}$, where, T_k is affected by T_i directly if $r_k \cap w_i \neq \varnothing$, or T_k is affected by T_i indirectly if $r_k \cap w_j \neq \varnothing, i < j < k$, and T_j is affected by T_i directly or indirectly, write set w contains only bad data objects.

TransID	ReadSet	WriteSet	Timestamp	PatientID
0001	<A B D>	<B C>	050617	P_1
0002	<A C>	<D>	050890	P_2
...				
0010	<E>	<A>	061098	P_1
...	...	...	...	...

Table 1. An Example of Attack History H_i^A

We propose a fine-grained mechanism to protect data objects stored in a database. Each data object o_i is uniquely represented by a triplet <t.id, c.id, r.id>, where $t.id$, $c.id$ and $r.id$ denote the *table*, *column*, and *row*, respectively. An itemset is defined as a non-empty set of data objects. We denote an *access sequence* a_s = $<s_1 s_2 ... s_m>$ by an ordered list of itemsets s_m=$(o_1 o_2 ... o_n)$, where $s_m \subseteq w_m$ is an itemset updated by a transaction T_m at transaction time t, and o_n is a data object. In table 2, each row represents an access sequence. Although the access of objects largely depends on the goal of an attacker, how the damage spreads is determined by the predefined transaction scripts once the attack is done. Given the attack histories H_i^A, we observe that attack histories of similar attacks have similar damage spreading patterns. Hence, our research problem becomes to discover the damage spreading patterns and use these patterns to estimate the integrity of data objects when a new attack is detected.

3.3 An Example of Causing Damage Spreading

In this section, we provide an example of an SQL injection attack that convinces the database application to run SQL code that is not intended. The example is shown as follows.

Example. *SQL Injection Attack. Although end-users do not interact with back-end database servers directly, if user input is not sanitized correctly, it is possible that unauthorized users can leverage the applications to corrupt the integrity of data objects stored in back-end databases. As an illustration, consider the transaction templates used in the real clinic OLTP application:*

1. ***Templates:*** *update Customer set TSales=TSales+ $amts, ..., Year_Accu_Amt= Year_Accu_Amt + $amty where Cust_Num = '$pid';*
   ...
   update Customer set Point =Point - $points,... where Cust_Num = '$pid';

2. ***Injection:*** *$pid=' OR Cust_Num like '100%*

3. ***Result:*** *update Customer set TSales=TSales+ $amts, ... where Cust_Num = ' ' OR Cust_Num like '100%';...*

Patient	Access Sequence		
PID	Transaction time t		
P_1	<(B C)[a]	(A)	(D E)>
P_2	<(D)	(B)>	
P_3	<(A C)	(B D)	(E)>
P_4	<(A D)>		
P_5	<(B)	(D)	(A)>

[a] Each character denotes a data object represented by a triplet (t.id,r.id,c.id).

Table 2. An Example of An Attack History H_i^A Grouped By Patient ID and Sorted By Transaction Time

The transaction script shown above is originally to update the purchase accumulated at that moment of a customer and calculate the reward points accordingly. However, if the input parameters are not escaped correctly before inserting into the query templates, it allows the attackers to change the query structure by injecting a piece of SQL statement. Thus, the where-clause is always true and some data objects that the malicious user is not allowed to access are modified.

3.4 Two Types of Damage Spreading Patterns

Figure 1 shows an example of the *frequent spreading patterns* from a clinic OLTP application after the *SQL Injection Attack* as mentioned in above section. We observe that there are two types of spreading patterns. We name them *one-hop* spreading, and *multi-hop* spreading, respectively. In most applications, if we cluster the read and update operations on a database, we would typically find that most updates are "bounded" within the scope of a certain database *entity* (e.g., a patient in a health care database). That is, when the value of an attribute of an entity is changed, all the inputs used in the update are typically read from the same entity's attributes. As a result, the set of reads and updates clustered around each entity form a cluster of operations called an *island* in our model. Nevertheless, it should be noticed that due to the various relationships between entities, as well studied in the ER model, cross-island updates are not uncommon, and in many applications cross-island updates might even be dominant. In such an update, one entity's attributes are used in changing the value of an attribute of another entity. As shown in figure 1, damage spreading within each circle (an island) illustrates a one-hop spreading pattern, such as $<(o_1^a, ..., o_x^a)_w>$. Damage spreading crossing islands illustrates a multi-hop spreading pattern, such as island $a \Rightarrow$ island b.

Based on this observation, we discover that the attack histories of similar attacks have similar damage spreading patterns. It can be seen that if such *frequent spreading patterns* are identified, pattern-based integrity estimation can be applied and an access policy can be quickly and dynamically enforced to filter out potential damage spreading transactions while the database system continuously provide services in the face of an attack.

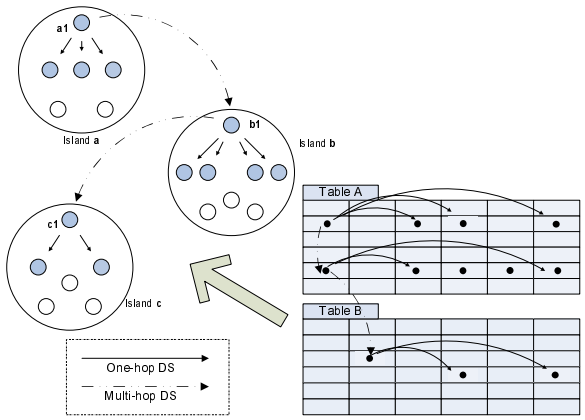


Figure 1. An Example of Two Types of Damage Spreading

3.5 Database Firewall Architecture

As shown in figure 2, the database firewall architecture is built upon the top of a traditional "off-the-shelf" DBMS. In general, the database firewall has two major components: *offline* and *online*. In offline manner, after the database system is recovered from an attack, damage spreading patterns are updated by mining the attack histories that are combined the newest attack history with previous attack histories stored in *Attack Histories*. In online manner, once a new attack is detected and is clustered into a particular type of attack, the Bayesian network corresponding to that cluster of attacking is built upon the mined patterns, and then the integrity of data objects is estimated based on the Bayesian network.

The detailed mechanism of the database firewall is as follows. When there are no attacks detected, the firewall is bypassed. The incoming transactions are processed by the *data processing system* as usual. As soon as an attack is identified by *Attack Detector* and the damaged data by malicious transaction T_i^B is located by *Damage Assessor* using stored transaction log and audit files, the *Integrity Estimator* starts to estimate the set of dirty data objects based on the built Bayesian network. When the *Damage Assessor* locates the damage, *Damage Repairer* repairs the damage data objects reported by *Damage Assessor* and *Integrity Estimator* using some specific *cleaning* transactions. *Policy Enforcement Manager* works as a *proxy* for decision making of data objects access. *Firewalling Manager* functions when *Attack Detector* detects malicious transactions. For example, upon the time when a malicious transaction T_i^B is detected, *Attack Detector* notifies *Firewalling Manager* to set up a new access policy to abort every active transaction

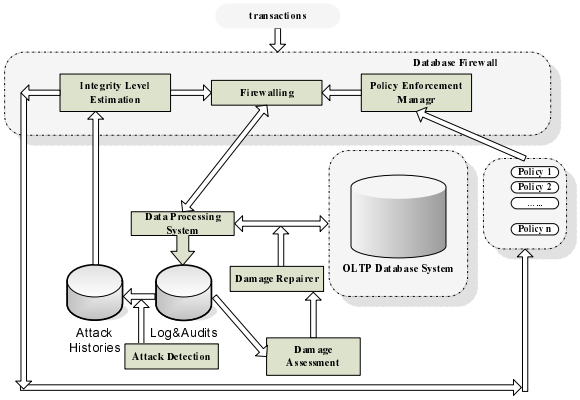


Figure 2. Database Firewall Architecture

and deny new coming ones at this time point and to build up firewalls to prevent damage from spreading.

This procedure may cause some availability loss at the stage of building up of Bayesian network and at each step of estimation, while the time consuming part of our db firewall solution, namely the pattern mining part, typically does not cause any availability loss. 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. Along with scanning the new attack history, *Integrity Estimator* adaptively predicts the integrity of compromised data until the final solution is reached.

4 The Damage Spreading Patterns Mining Approach

In previous section, we identified two types of *damage spreading patterns* from the attack histories of a database. In this section, we present the approach for discovering the frequent damage spreading patterns.

4.1 Support for Mining Damage Spreading Patterns

In our analysis of the attack histories, each attack history H_i^A is converted to records in the form of $[tA^i, tTime, tItemset_w]$, where $tTime$ denotes the transaction time and tA^i denotes an attribute of an entity. In an attack history, all the data objects updated by transactions associated with the same value of attribute tA^i of an entity can be viewed as a sequence together. For instance, as shown in table 2, in the clinic application, an attack history is converted by tA^i=*patient-id*. To evaluate the importance of a pattern, the support is used as a metric parameter. The definition of support for a pattern may very from one application to another. In this paper, the definitions of support

One-Hop Spreading Pattern	Mapping
<(A)(B)>	1
<(A)(E G)>	2
...	...

Table 3. One-Hop Spreading Pattern Mapping

Histories	After Mapping
$H_1^{A'}$	{1} {2} {3}
$H_2^{A'}$	{3} {4} {5}
$H_3^{A'}$	{1} {2} {4} {5}
$H_4^{A'}$	{1} {2} {3} {4}
$H_5^{A'}$	{1} {4} {5}
$\vdots$	$\vdots$

Table 4. An Example of Attack Histories $H_i^{A'}$ After Mapping

are defined as follows:

Definition 1 *The* support *of a frequent damaged data object o_x is denoted by the ratio of the number of data object sets having o_x to the total number of data object sets in an attack history.*

Definition 2 *The* support *of a frequent damage spreading pattern X, including both one-hop and multi-hop, is denoted by the ratio of the number of attack histories containing X to the total number of attack histories.*

In practice, we find that some data objects are damaged in a random manner. Scanning the entire set of attack histories to find their support ratio is time consuming and does not benefit our pattern mining process. Thus, Definition 1 of the minimum support is defined to filter out these uninteresting corrupted data objects.

4.2 Finding One-Hop Spreading Patterns

Given a cluster of attack histories H_i^A that are caused by the same (type of) attack, the problem of mining one-hop access patterns is finding large access sequences among all sequences that satisfy a user-specified minimum support. A large *access sequence* is denoted as $a_s^l = <s_1, s_2, \ldots, s_i>$, where each itemset s_i satisfies the specified minimum support threshold. The support of itemset s_i is similar to the definition 2 of support and is defined as the fraction of attack

	Access Sequence (sorted by t)			
Histories	t^1	t^2	...t^k	$S^M > 40\%$
H_1^A	<(A)(B)>	...	...	<(A)(B)>
H_2^A	...	<(C D)(A)(E F G)>	...	
H_3^A	<(A H G)>	...	...	
H_4^A	...	<(A)(E G)(B)>	...	<(A)(E G)>
H_5^A	<(B)>	...	...	
$\vdots$	$\vdots$	$\ddots$	$\vdots$	

Table 5. An Example of Using Attack Histories to Mine One-Hop Spreading Patterns

histories that contain it. Each such large access sequence a_s^l is denoted as a one-hop *frequent spreading pattern*. To dig out the large access sequences from attack histories, the histories are transformed in the form as shown in table 2 in which data objects are grouped by an attribute tA^i and sorted by transaction time $tTime$. Note, in transformation, we assume that access sequence a_s = <(A A)> = <(A)>, and similarly a_s =<(A)(A B)> = <(A)(B)>, and the large access sequence a_s^l = <(A)(B C)> = <(A B)(C)>. Since we only consider the damaged data objects, in this way, we can reduce the redundancy.

Example. *Table 2 demonstrates an attack history that is grouped by patient-id and sorted by transaction time. Table 5 shows the attack histories expressed as a set of access sequences. Consider the example of histories shown in table 5, in accordance with the Definition 2, with the minimum support set to 40% (i.e., two histories of total five histories.), two access sequences are found: <(A)(B)>, <(A)(E G)>. A real damage spreading pattern from clinic application is shown as follows: <(Treat_Status, Used_Count) (Cust_Name, Year_Accu, Accu_Base) (Point.x) (Total_Sales, Quan_OnHand) >*

The data objects (A) and (B) in *damage spreading pattern* <(A)(B)> are accessed both in histories 1 and 4, and the data objects (A) and (E G) in *damage spreading pattern* <(A)(E G)> are supported by histories 2 and 4. In history 2, itemset (C D) will be first filtered out due to support definition 1. In history 4, since we try to dig out access sequences that are not necessarily contiguous, although another access sequence is in between items (A) and (B), <(A)(B)> still counts as a large access sequence. Obviously, access sequence <(A H G)> is not a large access sequence because only history H_3 supports it. Although access sequences <(A)>, <(B)>, <(A)(E)> and <(A)(G)> have satisfied the minimum support, they are not counted as a pattern because they are not the large access sequences.

4.3 Finding Multi-Hop Spreading Patterns

In this section, we will present the idea of identifying multi-hop spreading patterns. Without loss of any generality, we assume that the one-hop spreading patterns are mapped to a set of contiguous integers as shown in table 3. Table 4 illustrates an example of attack histories after mapping. $H_i^{A'}$ denotes an attack history i after mapping and is sorted by the transaction time of the first data object set s_m of each large access sequence (one-hop) a_s^l. By doing this, the detailed information of one-hop spreading patterns is hidden and the focus is moved to finding out how the damage migrates among the islands.

Example. *Consider the attack histories shown in table 4. The large one-hop spreading patterns found have been replaced by the set of integers. The minimum support is specified to be 40%. Applying the same steps used in mining one-hop patterns, the large multi-hop spreading patterns would be following:* $< \{1\}\{2\}\{3\} >$, $< \{1\}\{2\}\{4\} >$, $< \{1\}\{4\}\{5\} >$ *and* $< \{3\}\{4\} >$.

To find the one-hop and multi-hop spreading patterns, the Apriori algorithm [15] is adopted. Given the minimum support, the algorithm of mining the one-hop is shown as below, where $(a_s.o_i)^s$ is the support ratio of a data object o_i in the access sequence a_s against the minimum support (definition 1), and $(a_s)^s$ is the support ratio of an access sequence against the minimum support (definition 2). The set of candidate patterns is denoted as C, and I_k denotes the final list of large access sequences (one-hop spreading pattern). Similarly, the multi-hop can be mined by repeating steps 9 through 16 using the after-mapping histories.

Algorithm 1: Mining Frequent Damage Spreading Patterns

Input: Histories H_i^A
Result: Frequent one-hop damage spreading patterns
`/* for each attack history, grouped by` tA^i `and sorted by time t */`

```
1  begin
2      forall a_s ∈ H_i^A do
3          if (a_s.o_i)^s < min_sup^1 then
4              remove o_i from a_s, H_i^A;
5          else if (a_s)^s ≥ min_sup^2 then
6              a_s → I_1;
7          else
8              remove a_s from H_i';
9      for k → 2 to I_{k-1} ≠ ∅ do
10         C = I_{k-1} ⋈ I_{k-1};
11         foreach a_s ∈ C do
12             if (a_s)^s < min_sup^2 then
13                 remove a_s from C;
14             else
15                 do nothing;
16     I_k = C;
17 end
```

5 Integrity Level Estimation

In previous section, we show how to mine frequent damage spreading patterns. In this section, we propose a novel database firewall approach to use these patterns to "throttle" damage spreading or propagation (during on-the-fly attack recovery).

Old methods (such as finding damaged data objects using dependency relation [1]) can achieve high accuracy of damaged data objects. However, these methods consume too much time and do not prevent the potential spreading transactions from accessing damaged data objects. Thus, it downgrades the availability of data objects and the system integrity and would further exacerbate the situation of de-

nial of service. In this section, we propose an approach using probabilistically reasoning based on Bayesian networks [10].

5.1 Bayesian Network Based Analysis on Data Integrity

In many cases, it is not possible to give out precise data integrity information (either good (clean) or bad (dirty)) in response to customer's queries without any delay after a database system is detected to be under an attack. Among various techniques used for analyzing the integrity level of data objects, the Bayesian network-based predictive reasoning is adopted in our database firewall framework. Through the analysis, we probabilistically determine the integrity of data objects from previous experience on the fly of an attack.

To build the Bayesian network, directed acyclic graphs (DAGs) have been used by researchers to facilitate the decomposition of a large distribution function into several small subsets and to represent causal or temporal relationships. Suppose there is a distribution P defined on n random variables. By using the chain rule of probability, we can have the following equation:

$$P(x_1, ..., x_n) = \prod_j P(x_j | x_1, ..., x_{j-1}) \quad (1)$$

$$P(x_1, ..., x_n) = P(x_j | parents(j)) \quad (2)$$

$$P(x_1, ..., x_n) = \prod_i P(x_i | parents(i)) \quad (3)$$

Since each variable is conditionally dependent of only some of its predecessors, the equation above can be rewritten to equation (2), where x_j is the variable that is only dependent on a select group of predecessors named $parents(j)$. In general, we can construct the Bayesian network as a carrier of conditional independence relationships along the order of construction as equation (3).

5.2 Integrity Estimation Using Bayesian Network

A Bayesian network consists of two components. The first component is a directed acyclic graph G whose node corresponds to a one-hop *damage spreading pattern* discovered at the pattern mining phase. The second component is an edge in the graph that denotes a direct dependence of the one-hop *damage spreading pattern* on its $parents(i)$. The DAG encodes a set of conditional independence assumptions that each node is conditionally independent of its predecessors given its parents.

Example. *Consider the example shown in table 4 in section 4.3, with the minimum support specified to be* 40%*, the large multi-hop damage spreading sequences are:* <{1}{2}{3}>, <{1}{2}{4}>, <{1}{4}{5}> *and* <{3}{4}>. *To construct the DAG from these multi-hop patterns, we determine the select group of predecessors of each node if they satisfy the following condition: there is a rule as* $\{parents_i\} \Rightarrow \{\text{node}\}$ *and the confidence of the rule is larger than or equal to a user specified parameter c, then an arc between* $parents_i$ *and node is drawn accordingly. We identify the following rules from above patterns:* $\{1\} \Rightarrow \{2\}(conf = 75\%), \{1\} \Rightarrow \{3\}(conf = 50\%), \{1\} \Rightarrow \{4\}(conf = 75\%), \{1\} \Rightarrow \{5\}(conf = 50\%), \{2\} \Rightarrow \{3\}(conf = 66\%), \{2\} \Rightarrow \{4\}(conf = 66\%), \{3\} \Rightarrow \{4\}(conf = 66\%), \{4\} \Rightarrow \{5\}(conf = 75\%)$.

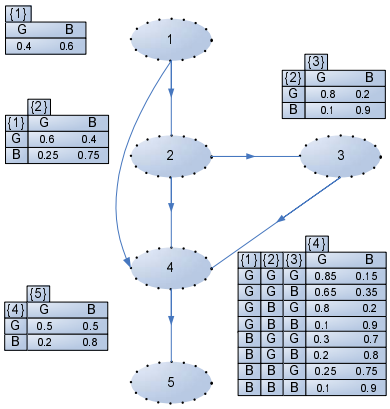


Figure 3. An Example of Bayesian Network with the Damage Spreading Probability Table

The rules with confidence less than c=50% are discarded from all rules generated. Figure 3 shows an example Bayesian network constructed from above rules. In addition, the conditional probability tables computed based on previous attack histories is drawn as well in figure 3.

Bayesian network is adopted in our framework to predict the integrity of data objects when some data object has been observed as damaged. The idea of is, given evidence about the state G or B of a one-hop pattern, where G and B denote *Good*, and *Bad* for short, the state of data objects in other one-hop pattern can be inferred by equation 4:

$$P(X) = \sum_Y P(X, Y) \quad (4)$$

where Y stands for all unknown variables excluding variable X, and $P(X, Y)$ may be expanded using the product rule shown as 3. Because Bayesian network defines a joint probability it is clear that $P(X, Y)$ can be computed from the DAG. For example, to find the probability that the 5 is damaged given that 1 is damaged, that is $P(\{5\} = B | \{1\} = B)$, it is necessary to marginalize over the unknown parameters. This amounts to summing the probabilities of all routes through the graph.

Example. *Given the Bayesian network as shown in figure 3, if we observe at time* t *that the data object* $o_i = (A) \in \{1\}$ *is damaged when scanning the new attack history* h_{new}*, we can*

compute the damage probability of the data object sets belonging to pattern $\{5\}$ by equation 4 as follows:

$$P(5 = B|1 = B) = \sum_{x,y,z} \{P(5 = B|4 = x, 3 = z, 2 = y) = 0.465$$

Suppose that data objects (C), (D) and (E) are in $\{5\}$, and <(A)(B)> $\Rightarrow$ <(C)(D)(E)> with probability 0.465. Then we denote the integrity level of data objects (C), (D) and (E) as 0.465, which means that there is a 0.465 certainty that these data objects are damaged. If along the scanning process of the new attack history, new evidences of damaged data are found, the integrity level is adjusted accordingly. Meanwhile, filtering rules are enforced and certain clean transactions are triggered to repair the dirty data according to the estimated integrity of the data objects. Finally, the integrity of data objects contained in Y as in algorithm 2 is known, and the corresponding Integrity Filtering List $\hat{I}_i$ (see Definition 3 in section 5.3) is updated for generating new filtering rules. This algorithm is incremental. As new evidence obtained, some interim results from previous calculations can be reused to estimate the integrity of data objects instead of re-computing.

Algorithm 2: Integrity Estimation

Input: G, X, Y
Result: Integrity Level List $\hat{I}_i$
`/* X → evidence; Y → to be estimated */`
begin
  c_{node} = Y;
  while $c_{node}.pa \neq \varnothing$ **do**
    p[i] = $\sum_v p(c_{node} = v|pa_1 = v, pa_2 = v, \ldots)$; `/* v∈G,B */`
    $c_{node} = c_{node}.pa$;
    i++;
  Pr(Y|X=x) = $\prod_i p[i]$;
  $\hat{I}_i = \forall\, o_i \in Y \leftarrow Pr$;
end

5.3 Database Firewall Security Policy

A specific and strongly worded security policy is vital to the pursuit of internal data integrity. This policy should govern everything from acceptance of accessing data objects to response scenarios in the event that a security incident occurs, such as policy updating upon new attacking. Ideally, a database firewall policy dictates how transaction traffic is handled and how filtering ruleset is managed and updated. 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, but let other transactions enter to achieve maximum throughput.

Definition 3 *Integrity Filtering List,* $\hat{I}$= $\{(o_1, o_x, \ldots, o_y)^{i_1}, (o_2, o_z, \ldots, o_v)^{i_2}, \ldots\}$, *where i is an integrity level. Objects o_x with the same integrity level i are grouped together.*

For example, suppose a transaction $T_1(t) = r_1[o_x]r_1[o_y]w_1[o_y]$ needs to enter the database at time t. We know that the integrity of data object o_x has been estimated and considered as corrupted at this moment, our policy checker will screen and be aware if the request can be granted using the rulesets defined as follows:

Rule 1 : $\forall$ transaction *T*, if $\exists$ data object $o_x \in R_T$, and $R_T \bigcap \hat{I} \neq \varnothing$, and if $W_T \neq \varnothing$, DENY;

Rule 2 : $\forall$ transaction *T*, if $\exists$ data object $o_x \in R_T$, and $R_T \bigcap \hat{I} \neq \varnothing$, and if $W_T = \varnothing$, and if $i(o_x) < Q$ then DENY, otherwise GRANT;

Rule 3 : $\forall$ transaction *T*, if $\exists$ data object $o_x \in R_T$, and $R_T \bigcap \hat{I} \neq \varnothing$, and if $W_T \neq \varnothing$, and if $i(o_x) < Q$ then DENY, otherwise GRANT;

Rule 4 : $\forall$ transaction *T*, if $\nexists$ data object $o_x \in R_T$, and $R_T \bigcap \hat{I} = \varnothing$, GRANT;

Where, $i(o_x)$ is the integrity level of the object o_x and Q is Quality of Information Assurance associated with applications. R_T and W_T are the readset, writeset of the transaction T, respectively. For example, give $Q = 0.5$ of an application, it means if the integrity of data objects is 0.5 or higher, it is acceptable to the application. Note, what we have presented here is a sample ruleset. We should be aware that firewall rulesets tend to become more complicated with age.

6 Experimental Results

In this section, we present the experimental results. To assess the performance of our database firewalling approach, we conduct two empirical studies based on both synthetic and real datasets. The simulation model of our experimental studies is described in section 6.1.

6.1 Experiment Settings

In our experiments, we use two synthetic history sets generated based on the modified TPC-C benchmark. We take the parameters shown in table 6 to populate the synthetic attack histories. The specified settings for each parameters used to generate the history sets are included in table 7. In addition, a real dataset of transaction histories from a clinic OLTP application is employed to verify the mining algorithm. The dataset (400MB) contains 119 tables with 1142 attributes belonging to 9 different applications. For more detailed description of the dataset we refer the reader to [16]. Moreover, to verify the database firewall prototype, we re-construct the clinic application based on the application and query templates and generate real_syn attack histories. We also assume that there is only one attack at a time. Additionally, similar attack is practically defined

Parameters	Notes
N	Number of data objects
T	Average numbers of data objects per transaction
C(P)	Number of customers(patients)
D	Number of transactions
H	Number of attack histories

Table 6. Parameters of the synthetic datasets

Name	N	T	C	D	H
Syn_1	1000	3	500	100,000	10
Syn_2	1000	5	1000	100,000	20
$Real$	1000	8	1000	100,000	10
$Real\_Syn$	1000	8	1000	100,000	20

Table 7. Parameter values of the datasets

in our experiments as the attacks with the same transaction type, such as *Order, Payment* transaction type in the TPC-C benchmark.

6.2 Mining Algorithms Testing

To verify the mining algorithms, we conducted experiments to measure the quality of mined one-hop frequent *damage spreading patterns* using two metrics, recall and precision. They are defined as follows. Given a set of true *damage spreading patterns A* and a set of obtained frequent *damage spreading patterns B*, the recall is $\frac{|A \cap B|}{|A|}$ and the precision is $\frac{|A \cap B|}{|B|}$.

In table 8, the first column shows we use two different kinds of dataset, one from the synthetic application and one from the real application. The last column is the minimum support s to mine the patterns. It can be seen that when the minimum support is set to 15%, using the synthetic dataset, 0.93 of recall rate can be achieved. When we lower s to 10%, higher recall rate (98%) can be obtained. Although the high recall rate is achieved, the false-negative patterns have the potential to cause the damage leakage. However, after further investigation of the missed patterns, we find that the chance these missed patterns cause damage leakage is small. First, although these missed patterns are true damage spreading patterns, they do not often occur in the damage histories and then are screened out because of the minimum support. The reason they do not frequently in-

Data Type	A	B	Recall	Precision	s (%)
Syn	300	279	0.93	1	15
Syn	600	587	0.98	1	10
$Real$	200	190	0.95	1	10
$Real_Syn$	300	291	0.97	1	10

Table 8. Measurement of the Mining Algorithm of One-hop Patterns

Real System Integrity SI_r	Time (s)		%
	BN	DR	Accuracy
80%	50	260	94%
70%	70	270	92%
65%	90	300	87%

Table 9. Speed of System Integrity Estimation

volve in damage spreading is that transaction scripts always have conditional statements, such as *if-else* statement. Only when certain condition is satisfied, the branch will be gone through, and then the patterns occur. Second, these missed patterns are often on the downstream of a multi-hop spreading pattern (similar to the leaf node of a tree). Thus, even if they are damaged and not included in final answer, they do not cause damage leakage. The experiments to measure the multi-hop patterns achieve similar results that are not presented due to limited space. As a conclusion, we believe that the mining algorithm is satisfactory. However, since we deploy a relative simple SQL-injection attack, pattern mining algorithm achieve high recall rate. How the algorithm performs when the attack is sophisticated needs a further investigation. Furthermore, the algorithm is designed particularly for the TPC-C and clinic applications. A generic condition under which the mining algorithm can be useful remains unknown.

6.3 System Integrity Analysis

Estimated system integrity SI_e is defined as the ratio of the summation of the damage probabilities of the data objects n^d to the number of total data objects N in the database: $SI_e = 1 - \frac{\sum_{i=1}^{n} P}{N}$. For example, if the database contains $N = 100$ in which total $n^d = 20$ are bad data, the real system integrity should be $SI_r = 1 - \frac{\sum_{i=1}^{20} *1}{100}$=80% when the system is detected to be under an attack. To verify the estimation accuracy of our approach, we define the accuracy as the ratio of the number of data objects that are estimated (and are tested as they are later) to the number of damaged data objects in the new attack history.

Figure 4 shows the process of discovering system integrity. Our probability based approach is compared with the algorithm adopted from [1] that use the dependency relation of transactions. Along with the process of scanning the new attack history, at each time t_i a data object o_i or a set of data objects $(o_i, .., o_j)$ are found to be bad, the dependency relation based approach will only know what it has learned up to time t_i. However, with the Bayesian network constructed from previous attack histories, we are able to compute the damage probability of all data objects previously occurred in attack histories. Therefore, our approach should be faster than the dependency based approach in finding the system integrity. Results in table 9 demonstrate that our Bayesian network based approach is much faster than the dependency relation based approach in terms of as-

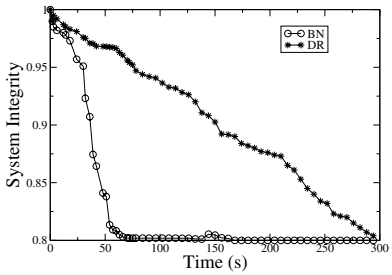
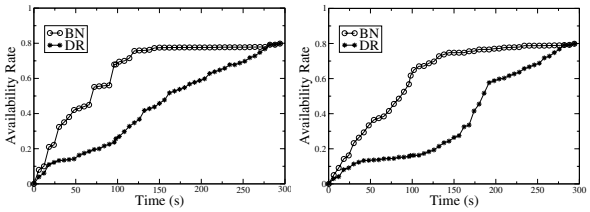


Figure 4. System Integrity Estimation Procedure



(a) Synthetic Dataset (b) Real_Syn Dataset

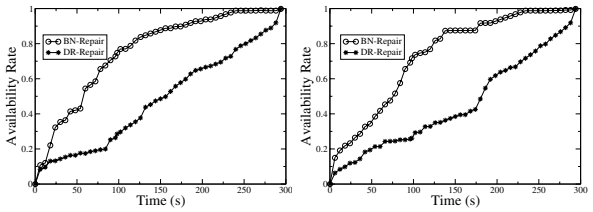
Figure 5. System Availability without Repairing Procedure

sessing the system integrity. We can also observe that the estimation accuracy of our approach is relative high when the new attack is similar to previous attacks. When the recall of the pattern mining is high, the estimation accuracy is also high because more knowledge of patterns can be used. However, when a different attack is launched, the accuracy will be low because the previous knowledge is no longer useful to estimate the integrity level of damaged objects. More efforts are needed to study this issue.

6.4 System Availability Analysis

System availability is an important metric to evaluate the performance of the database firewall framework. For conventional methods, e.g. rolling back, even if there are only a few damaged data objects, since the system is not accessible while re-executing, the system availability is zero. It is crucial to the database firewall to reduce this time period and maintain the system availability on a certain level. In our framework, we define two kinds of availabilities. The maximum system availability is defined as the rate of the number of good data objects to the number of the total data objects in the database $A_{max} = \frac{n^g}{N}$. The real-time system availability is defined as the rate of the number of good data objects discovered at time t to the number of the total data objects in the database $A_r = \frac{n^t}{N}$.

Figure 6.4 illustrates the results of the system availability with respect to time from both the synthetic and the real histories. The repairing procedure is not included in this testing. The approach based on the dependency relation [1] (denoted as DR, Bayesian network denoted as BN) is also



(a) Synthetic Dataset (b) Real_Syn Dataset

Figure 6. System Availability with Repairing Procedure

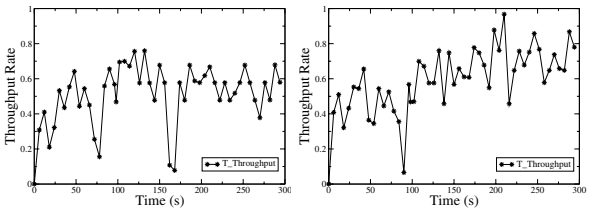
implemented to compare with our approach. Let ε be a user specified parameter. If the estimated integrity of data objects is equal to ε or higher, the repairing procedure is fired up instead of waiting until it is 100 percent sure that the data objects are damaged. Let Q be the differential quality of information assurance required by different applications. It may vary from an application to another. If the estimated integrity of data objects is equal to Q or lower, these data objects are available to be accessed. Given $A_{max} = 0.8$, $Q = 0.15$ and $\varepsilon = 0.6$, it can be seen in figure 6.4 that the database firewall approach can maintain a relative higher system availability A_r and also reach the maximum system availability $A_{max} = 0.8$ faster than the dependency relation based approach. Figure 6 demonstrates the results of the system availability when the repairing procedure [1] is introduced. Given $Q = 0.15$ and $\varepsilon = 0.6$, it can be seen that the maximum system availability $A_{max} = 1$ can be reach earlier than the dependency relation based approach. Note, we assume the repairing rate is constant.

6.5 System Throughput Analysis

System throughput in terms of continuously providing data access while filtering out potential damage spreading transactions when the data system is under an attack is an important metric to measure the performance of the database firewall framework.

In practice, the system throughput is defined as a ratio of the number of served transactions to the number of transactions requiring service: $Tr = \frac{n^s}{N}$. We assume that the arrival rate of the transactions is constant. Figure 7 demonstrates the performance of the system throughput from both the synthetic and the real dataset without taking into consideration the damage leakage. To filter out potential damage spreading transactions, we assume the transaction dependence is tight. For instance, if transaction T_2 does not access compromised data but rely on the result of transaction T_1, transaction T_2 will still be discarded if transaction T_1 is filtered out due to accessing compromised data. This assume is reasonable because the result directly from transaction T_1 is dirty.

Given $A_{max} = 0.8, Q = 0.15$ and $\varepsilon = 0.6$,it can be seen that, in figure 7(a), the system throughput reaches



(a) Synthetic Dataset without Repairing Procedure (b) Real_Syn Dataset with Repairing Procedure

Figure 7. System Throughput

nearly zero at time $t = 75s$. The reason is that the data objects required by the transactions are all contained, and filtering policy does not allow those data objects being accessed by the transactions. In addition, in both figure 7(a) and (b), although the system throughput vibrates up and down along the time axis, the average ratio of the system throughput is above 50%. Moreover, if the repairing process is introduced in figure 7(b), the system throughput can be even higher.

7 Conclusion and Future Work

In this paper, we study the problem of mining frequent damage spreading patterns from the previous attack histories, of using Bayesian network constructed upon the mined damage spreading patterns to predict the damage probabilities and the system integrity on the fly. With this feature, a database firewall framework is devised to dynamically enforce a policy of transaction filtering to filter out the potential *spreading transactions*. In the meanwhile, the database firewall continuously provides data services in the face of attack. The experimental results demonstrate that it is satisfactory that the database firewall can continuously provide quality data in accordance with the differential service requirement when the system is detected under an attack.

At this early stage of the work, we do not study the impact of the detection latency. Constructing Bayesian network over small amount of patterns in general does not have significant impact on the performance of the database firewall system. However, the computation overhead could increase when discovered spreading patterns accumulate. In our immediate future work, we will further evaluate our model to address above issues and extend the idea of database firewall to a distributed environment. Several other problems are interesting as well, such as how the performance is impacted by heavy attack, how can the estimation accuracy be improved, and what is the condition under which mining spreading pattern can be superior.

Acknowledgement This work was supported in part by NSF CCR-TC-0233324. And we wish to thank Elisa Bertino, Ashish Kamra and Xiangji Huang very much for kindly sharing the clinic OLTP data with us.

References

[1] P. Ammann, S. Jajodia, and P. Liu, "Recovery from malicious transactions," in IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 5, pp. 1167–1185, 2002.

[2] 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.

[3] P. Liu, P. Ammann, and S. Jajodia, "Rewriting histories: Recovery from malicious transactions". Distributed and Parallel Database 8, 1, 7-40.

[4] P. Ammann, S. Jajodia, C. McCollum, and B. Blaustein, "Surviving information warfare attacks on databases," in Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, May 1997, pp. 164–174.

[5] P. Liu, "Architectures for intrusion tolerant database systems," in Twenty-first Annual Computer Security Applications Conference (ACSAC'02), 2002, pp. 311–320.

[6] 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.

[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] E. Bertino and A. Kamra and E. Terzi and A. Vakali, "Intrusion Detection in RBAC-administered Databases," 21st Annual Computer Security Applications Conference (ACSAC 2005), 5-9 December 2005, Tucson, AZ, USA

[9] P. A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, MA, 1987.

[10] J. Pearl, Causality: Models, Reasoning and Inference. Cambridge University Press.

[11] T. Chiueh and D. Pilania, "Design, implementation, and evaluation of an intrusion resilient database system," in Proc. International Conference on Data Engineering, 1024-1035, 2005.

[12] S. Sekar, M. Bendre, and P. Bollineni, "A fast automaton-based method for detecting anomalous program behaviors," in 2001 IEEE Symposium on Security and Privacy. Oakland, CA.

[13] S.-P. Shieh, and V. Glogor, "On a pattern-oriented model for intrusion detection," IEEE Trans. On Knowledge and Data Engineering 9, 4, 661-667

[14] R. Sobhan and B. Panda, "Reorganization of the database log for information warfare data recovery," in Proc. of the fifteenth annual working conference on Database and application security, Niagara, Ontario, Canada, 121-134, 2001.

[15] R. Agrawal and R. Srikant, "Fast algorithms for ming association rules," in Proc. of the VLDB Conference, Santiago, Chile, 1994.

[16] Q. Yao, A. An, and X. Huang, "Finding and analyzing database user sessions," in Proc. of the 10th International Conference on Database System for Advanced Applications, 2005.

[17] Owasp top ten most critical web application security vulnerabilities. http://www.owasp.org/documentation/topten.html, 2004

[18] S. Y. Lee, W. L. Low, and P. Y. Wong, "Learning fingerprints for a database intrusion detection system," in 7th European Symposium on Research in Computer Security (ESORICS), 2002.

[19] F. Valeur, D. Mutz, and G. Vigna, "A learning-based approach to the detection of SQL attacks," in Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), July 2005.

[20] C. Kruegel, and G. Vigna, "Anomaly detection of web-based attacks," in CCS'03. Washingtion, DC, USA, 251-261.

[21] B. Panda, and J. Giordano, " Reconstructing the database after electronic attacks," in the 12th IFIP 11.3 Working Conference on Database Security, 1998.

[22] J.-L. Lin, and M. H. Dunham, "A survey of distributed database checkpointing," in Distributed and Parallel Databases 5, 3, 289-319.

[23] Y. Lin, and E. D. Lazowska, "A study of time warp rollback machanisms," ACM Transactions on Modeling and Computer Simulations 1, 1, 51-72.

[24] D. R. Jefferson, "Virtual time," ACM Transactions on Programming Languages and Systems 7, 3, 404-425.