Database Isolation and Filtering against Data Corruption Attacks

Meng Yu, Wanyu Zang
Department of Computer Science
Western Illinois University

Peng Liu
College of Information Sciences and Technology
The Pennsylvania State University, University Park

Abstract

Various attacks (e.g., SQL injections) may corrupt data items in the database systems, which decreases the integrity level of the database. Intrusion detections systems are becoming more and more sophisticated to detect such attacks. However, more advanced detection techniques require more complicated analyses, e.g, sequential analysis, which incurs detection latency. If we have an intrusion detection system as a filter for all system inputs, we will introduce a uniform processing latency to all transactions of the database system. In this paper, we propose to use a "unsafe zone" to isolate user's SQL queries from a "safe zone" of the database. In the unsafe zone, we use polyinstantiations and flags for the records to provide an immediate but different view from that of the safe zone to the user. Such isolation has negligible processing latency from the user's view, while it can significantly improve the integrity level of the whole database system and reduce the recovery costs. Our techniques provide different integrity levels within different zones. Both our analytical and experimental results confirm the effectiveness of our isolation techniques against data corruption attacks to the databases. Our techniques can be applied to database systems to provide multizone isolations with different levels of QoS.

1 Introduction

While more and more data processing services are running through web services, transaction level attacks, such as SQL injections, become one of the major threats to the database systems because transaction level attacks can be done through a variety of ways. For example, the attacks can be done through web applications [8, 5, 14]. Among the OWASP top ten most critical web application security vulnerabilities [12], five out of the top 6 vulnerabilities can directly enable the attacker to launch a malicious transaction. The attacks can also be done through identity theft. Identify theft naturally leads to malicious transactions because a main purpose of using a stolen identity is for the attacker to launch some malicious transactions to steal money, etc. As indicated in [2], since database management systems are an indispensable component of modern Internet services and mission-critical applications, it is more susceptible to malicious attacks from remote sites.

Transaction level attacks have been studied in a good number of researches, e.g., in [2, 1, 16, 18], where recovery from data corruption caused by attacks is the major concern. These work focuses on how to trace damage spreading and repair the damage inside a database or across database systems. It is a remedy when the integrity level of the database system has already been compromised.

Intrusion detection techniques [11], especially some intrusion detection techniques for database systems like [4, 13, 17], are able to detect the attacks in the first place. However, to deploy an intrusion detection system (IDS) in a Database Management System (DBMS), we need to consider the *detection latency* incurred by the IDS. The detection latency of IDS is mainly caused by the following reasons. First, when the IDS is deployed in a local host, the detection latency is mainly caused by the processing time to analyze the characters of inputs, e.g., to model the behaviors of inputs, or to recognize the patterns of inputs. Recent more complex intrusion detection techniques require accessing a large amount of knowledge created based on training data sets. Second, when an IDS is deployed in a distributed manner, the detection latency will be significantly longer due to the latency introduced by communications and message exchanges between different hosts.

Such detection latency is a big concern to the DBMSs. Assume a perfectly accurate IDS, we can use it as a filter to inspect all database input to guarantee that there is no malicious inputs being injected to the DBMS. However, such filtering will introduce the detection latency to all database transactions. The database users will suffer prolonged transaction processing time, which significantly degrade the performance of DBMSs. If we let all inputs bypass the IDS and let the IDS work as an simple *inspector*, when the IDS finds out malicious inputs, the integrity level of the databases has already been compromised, especially when corrupted data items are referred by innocent transactions, where the dam-

age will be spread to other part of the database.

In this paper, we propose to use an *two-zone isolation*, a *safe zone* and a *unsafe zone*, to greatly reduce the probability of data corruptions while introduce negligible processing latency to the database users. We use a two-bit flag attached to each database records to separate the whole database into two zones logically. Three flags indicate the status in the unsafe zone and one flag indicates a clean status in the safe zone. Database records generated or manipulated by transactions processed within a specific *filtering window* τ will be flagged as in the unsafe zone. Such records will be merged into the safe zone after τ by simply changing their flags to the clean status.

When a database records stays in the unsafe zone, the IDS can remove the records (or cancel any updates) if the IDS finds out that the transaction manipulating the record is malicious. The corrupted data record will therefore be discarded before being merged into the safe zone. Our study shows, with both analyses and experiments, if the filtering window τ is properly chosen, the probability that corrupted data records are merged into the safe zone will be extremely low.

Our proposed approach has two zones with different integrity levels, thus different security Quality of Service (QoS). Our analysis and experiments show that the integrity level can be significantly improved with very low processing costs. If a user prefers data records with high integrity level (thus high level of confidence on the data), the safe zone will be the best bet. Such application is critical, especially when the database system is used for public services, where transaction level attacks are very common and unavoidable.

The rest of this paper is organized as follows. Section 2 provides an overview of our approach. Implementations of our techniques are described in 3. In Section 4, we model the IDS latency, which is the basis to determine the size of the unsafe zone. We study the effectiveness of our techniques in 5. Section 6 compares our work with related work. Section 7 concludes our work.

2 Overview of our approach

In this section, we introduce our conceptual model first. Based on the conceptual model, we model what kind of threats we can address. Finally, we explain how to use polyinstantiation in the two-zone isolation.

2.1 The conceptual model

A relational database contains many relations (or tables). Each relation is represented by a specific set of records which are manipulated by database transactions. Database transactions are usually launched by SQL commands acquired through application interfaces, e.g., web applications building upon the database system, or the database interface, e.g., the database console. Such SQL commands could be innocent when the database system is well protected, or malicious when the protection mechanisms are unfortunately broken, e.g., through SQL injections. Malicious SQL commands lead to malicious transactions that inject nothing but garbage data items into the database system. The garbage data items could be referred by innocent transactions and more data corruption happen in the database system. Therefore, the malicious transactions are very harmful to the database systems.

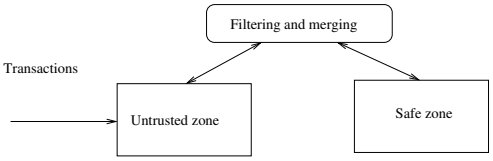


Figure 1. The conceptual model

The conceptual model of our approach is shown in Figure 1. In the figure, transactions arrive at the unsafe zone first. Their effects can be seen by the user immediately after they are processed by the database system. However, the effects will not be merged to the safe zone until they stay in the safe zone longer then a specific period of time τ, the *filtering window*. Note that the IDS will not inspect any transactions *before* they are submitted to the DBMS while the IDS will inspect those transactions *after* they are submitted. Therefore, no IDS latency will be introduced in the process. After data items stay in the unsafe zone longer then τ, which indicates that the missing probability of the IDS will be low enough, we will merge them into the safe zone.

Therefore, if there is good chances that a malicious transaction can be detected during τ, the malicious transaction will be removed in the unsafe zone without affecting the integrity level of the safe zone. Note that in the following discussions, we use term *user's view* for the database view containing both the unsafe zone and the safe zone.

Note that we do not isolate different users as those approaches did in work [6, 9, 10]. Their work prevent a suspicious user's results from being observed by others. If the user is found to be innocent, the user's results will be merged to the database, which incurs longer processing delays because the IDS serves as a filter. In our approach, a user's results can be seen by others immediately as long as the access control mechanism allows.

2.2 The threats model

We assume that all software components in our conceptual model are trustable, while the inputs (e.g., SQL commands received through web interface, for example) to the unsafe zone can be malicious. In other words, while we assume that our techniques are implemented in the Trusted Computing Base (TCB) of the DBMS, we do assume that the interface of the database, e.g., web applications, web interfaces, etc., of the database, are vulnerable to various attacks, e.g., SQL injections.

2.3 Isolation with polyinstantiations

By isolation, we need to contain the users' transactions happened within window τ in the unsafe zone. However, we need to hide the existence of the unsafe zone so the users' view of the database is a combined view of both the unsafe zone and the safe zone.

Conflicts are possible for such isolation. Example 2.1 shows a relation and its records in a safe zone of a relational database.

Example 2.1 *Relation* `student`

ID	name	dept
s0003	*Mike*	*Computer Science*

The following example tries to update a record into the relation in Example 2.1.

Example 2.2 *Update table* `student` *with the following SQL command.*

```
UPDATE student SET dept='Chemistry'
          WHERE ID='s0003'
```

In Example 2.2, a make up record,

ID	name	dept
s0003	Mike	Chemistry

will be necessary in the unsafe zone to hide the unsafe zone isolation. The original record must stay in the safe zone without changes to isolate the effects of transactions happened within τ. Now we have two records (in two different zones) having the same primary key "s0003". The situation introduced in Example 2.2 is called *polyinstantiation*, where multiple records with the same primary key exist in the same relation.

Definition 2.1 (Polyinstantiation) *We define* polyinstantiation *as the case where multiple database records with the same primary key exist simultaneously in the database, while all records are corresponding to the same entity in the real world.*

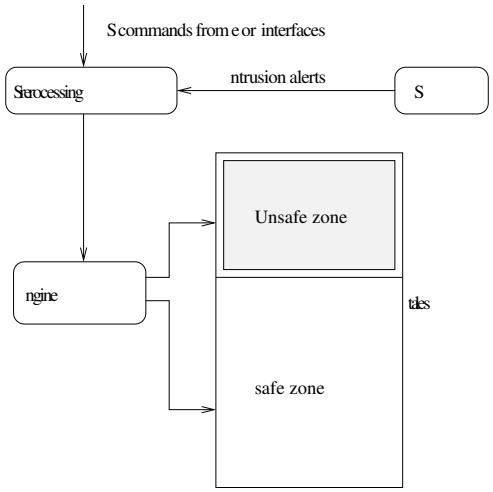


Figure 2. The processing structure

The existence of two records with the same key is not a violation of the integrity requirements of the DBMS, if we can control that only one record appears in a specific view. In fact, it is common practice to provide different views to different users with polyinstantiations, e.g., in multilevel relational database systems [7, 15].

In this paper, when polyinstantiation is necessary, the data items in the unsafe zone have higher priority to appear in the users' view. Therefore, the user can see only the record in the unsafe zone instead the one in the safe zone. In the users' view, the user successfully updated the record.

3 Implementation of the safe zone isolation

In this section, we describe how the isolation can be implemented in a database system and how to merge effective data items into the safe zone after a specific window τ.

3.1 Processing structure

Our techniques can be implemented at different levels of the DBMSs, for example, at the table level, or at the page level. In this section, we introduce the implementation of our techniques at the table level, which is more intuitive and can be easily transfered to page level implementations.

The processing structure is shown in Figure 2. In the figure, all transactions submitted through the web interface or other untrusted interface will be pre-processed by a SQL preprocessing component, which modified the SQL commands to put the results in the unsafe zone. The IDS component inspects all transactions independently without delaying any of them. Once an intrusion is identified, the intrusion alert will be sent to the SQL preprocessing component to cancel the malicious transaction.

Table 1. Flags used in processing

Flags	Descriptions
D	The record is deleted from the user's view.
I	The record is inserted with in window τ.
C	The record is cleared of security problems.
P	The record is for polyinstantiations.

We attach the following tuple

flag	timestamp

to each record to separate existing tables into different zones logically. The possible flags are shown in Table 1. Note that we do have isolation between users while the application and the DBMS may provide such isolation. The "**timestamp**" field records the time that the record was last modified. There is a unique value of timestamp for all records manipulated by the same transaction.

Example 3.1 *Given the update operation in Example 2.2, the relation* `student` *in Example 2.1 becomes the following one with attached tuple for zone separation.*

ID	name	dept	flag	timestamp
s0003	*Mike*	*Computer Science*	C	*Apr, 2005*
s0003	*Mike*	*Chemistry*	P	*18:01:05*

3.2 Processing of transactions

We process transactions according to the type of their operations, e.g., read-only, or updates. The main goal of our processing is to protect data records in the safe zone from being changed while the user can see their results immediately, which happens in the unsafe zone. The user's view of the whole database is a combined view of both the unsafe zone and the safe zone.

Read only operations

SQL command SELECT cannot cause data corruptions so there is no effects to be put in the unsafe zone. However, we need to do the view management for SELECT statements.

When a user queries the database, the set of results need to be modified in the following way. First, we filter out all records with flag 'D' which indicates the deletion of the records within τ. Second, for records with the same primary key value, which indicates polyinstantiations, we filter out all records with flag 'C' (or keep all records with flag 'I' or flag'P' for the same results).

Example 3.2 *A user who queries Mike's record to relation* `student` *in Example 3.1 can only access the following one record.*

ID	name	dept	flag	timestamp
s0003	*Mike*	*Chemistry*	P	*18:01:05*

while we will certainly remove the attached tuple

flag	timestamp
P	*18:01:05*

in the output to make it invisible.

Deletions

Deletions can be very harmful to the databases. We can hide all deleted records to the user who sends the requests. When the records are in the safe zone, instead of really deleting the records, the following tuple with be added to the records that the user asked to delete.

flag	timestamp
D	current time

Example 3.3 *A user who wants to delete Mike's record in relation* `student` *in Example 2.1 will get the following, if there is no polyinstantiated records in the unsafe zone.*

ID	name	dept	flag	timestamp
s0003	*Mike*	*Computer Science*	D	*21:01:05*

However, when polyinstantiations exist, or the records are in the unsafe zone, the records in the unsafe zone will be deleted directly.

Example 3.4 *A user who wants to delete Mike's record in the relation in Example 3.1, will get the following*

ID	name	dept	flag	timestamp
s0003	*Mike*	*Computer Science*	C	*Apr, 2005*

where the polyinstantiated record was deleted.

Insertions

The inserted data can be viewed by the user immediately while they will not be moved to the safe zone until they stay in the unsafe zone longer than a window τ. Given a "INSERT" command, the record will be added to the unsafe zone and the following tuple will be attached to the record if and only if there is no record existing in either the unsafe zone, or the safe zone, with the same primary key value as the one of the inserted record.

flag	timestamp
I	current time

If the user is inserting a record with a existing primary key value, we need to check the attached tuple with the key value. If the record has been flagged as 'D', we will allow such insertion. A polyinstantiated record will be generated in the unsafe zone, and the following tuple will be attached.

flag	timestamp
P	current time

At the same time, the flag of the original record will be changed from 'D' to 'C'.

Example 3.5 *Given the relation* student *in Example 2.1, a user may try to delete Mike's record, followed by an insertion to add a new record for Mike to change Mike's department to "Chemistry". The result will be as follows.*

ID	name	dept	flag	timestamp
s0003	*Mike*	*Computer Science*	C	*Apr, 2005*
s0003	*Mike*	*Chemistry*	P	*22:01:05*

The original record is marked as "deleted" first. Once the insertion finds a record with 'D', it will create a polyinstantiated record and recover the original record with 'C'.

Updates

If the record to be updated is in the safe zone, polyinstantiation is always necessary for updates since we do not want to update the real records in the safe zone until we wait longer than τ. Example 2.2 shows such situation.

When the record to be updated is in the unsafe zone, the update can be done directly in the unsafe zone with the attached flag unchanged.

Example 3.6 *A update changing Mike's department to "Math", given the results of Example 3.1, will generate the following results.*

ID	name	dept	flag	timestamp
s0003	*Mike*	*Computer Science*	C	*Apr, 2005*
s0003	*Mike*	*Math*	P	*23:01:05*

3.3 Merging good transactions

After window τ, transactions in the unsafe zone should be merged to the safe zone. We can do the merging every τ seconds. For all records having stayed in the unsafe zone longer than τ, we do the following according to their flags.

If the flag is 'D', the deletion should take effect in the safe zone. We delete such records from the database directly. If the flag is 'I', we change the flag to 'C' to add the safe zone. If the flag is 'P', we delete the record with the same primary key and flag 'C' (which is the record in the safe zone), and change the flag to 'C'.

Since all records inserted (updated, or deleted) by the same transaction will be assigned with the same timestamp, it is impossible to merge partial effects of a specific transaction into the safe zone while leave the other parts left in the unsafe zone. The atomicity of transaction can be guaranteed.

3.4 Cancellation of malicious transactions

Within window τ, if a transaction is identified as malicious. All data items generated or updated by the transaction should be removed. We can simply delete all records with the specific timestamp associated to the malicious transaction, if the timestamps assigned to transactions are unique.

we can also do damage assessment in a more accurate way to remove only malicious and affected transactions using techniques like [1, 18]. If a transaction is identified as malicious outside the window τ, the effects of the transaction will have been added to the safe zone. To remove the effects of such transactions needs analysis of dependency relations and transaction history. It is out of the scope of his paper and has been addressed in work [1, 18]. According to our discussion in Section 4, the probability of such situation will be extremely low if the window τ is carefully chosen.

3.5 Preserving the ACID properties

Current DBMSs has concurrency control and recovery mechanisms to support the ACID (Atomicity, Consistency, Isolation, Durability) properties of transactions.

According to our techniques, all records inserted (updated, or deleted) by the same transaction will have the same timestamp. Therefore, when we merge records to (or delete records from) the safe zone, the atomicity property is kept. The consistency, durability properties will also be kept since the concurrency control and logging is done by the DBMS without changes. In our techniques, the isolation property is different from the regular database systems. However, from the view of each user, the isolation property of transactions is kept.

4 Determining the filtering window τ

The IDS latency can be caused by the processing time to model the inputs, pattern recognition, communications latencies in a distributed system, etc.

Assume that the IDS latency is normally distributed with parameter T and σ.[1] The probability density function is as follows.

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-T)^2/2\sigma^2} \quad (1)$$

The expected IDS latency is

$$E(X) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{\infty} e^{-(x-T)^2/2\sigma^2} dx = T \quad (2)$$

[1]Note that if the IDS latency has a different probability distribution, we can certainly adopt a different one and our following equations can be revised accordingly

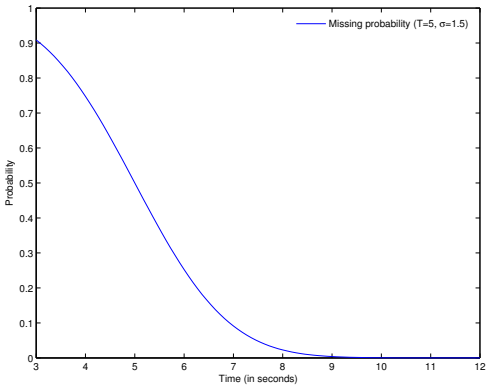


Figure 3. An example of the relationship between the filtering window τ and the missing probability

and the variation of IDS latency is

$$Var(X) = E[(X-T)^2] = \sigma^2 \tag{3}$$

The cumulative distribution function will be

$$F(a) = \Phi\left(\frac{a-T}{\sigma}\right) \tag{4}$$

where

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2/2} dy \tag{5}$$

is the cumulative distribution of a standard normal distribution (a normal distribution with parameter 1 and 0).

If there is no intrusion reported during a *filtering window* $\tau = kT$ and we only protect the system from transactions in τ, the probability that a malicious transaction is merged into the safe zone will be $1 - F(t)$. We define $1 - F(t)$ as the *missing probability*. The missing probability does not mean that we will miss processing anything. It indicates the probability that a malicious transaction is not detected within the window τ.

Given the IDS latency is normally distributed with parameter $T = 5$ seconds and $\sigma^2 = 2.25$. Figure 3 shows the relationship between the *process window* τ and the missing probability.

If we wait for $\tau = 8$ seconds, the missing probability will be 0.0228 according to the above discussion. When $\tau = 10$, the missing probability decreases dramatically to 0.000429.

We can select a proper $\tau = kT$ for the IDS reports. The missing probability will be

$$1 - F(t) = 1 - \Phi\left(\frac{k-1}{\sigma}T\right) \tag{6}$$

As mentioned above, if the probability distribution of the IDS detection latency is different, we may revise above equations and calculations accordingly. In our proposed work, the larger the filtering window is, the smaller the missing probability will be.

5 Evaluation

In this section, we evaluate the cost-effectiveness of our techniques. We will discuss the improvement of integrity level, storage and processing costs, reduced recovery costs, and the impact of IDS accuracy.

5.1 Integrity level

We start by assuming a perfectly accurate IDS whose detection rate is 100%, the false positive rate is 0%, and the false negative rate is 0%. In Section 5.3, we will discuss an imperfect IDS with inaccurate detections.

We can use the percentage of clean data items in the database as the estimate of the integrity level of the database.

Definition 5.1 (Integrity level) *Given a database, we define the* integrity level *of a specific view of the database as*

$$I = \frac{p}{q} \tag{7}$$

where p is the total number of clean data items in the view, and q is the total number of data items in the view.

We use I_s for the integrity level of the safe zone, and I_u for the integrity level of the user's view respectively.

Note that Definition 5.1 takes care of only specific views of a database, or the confidence level of what a user sees into a specific part of the database. It does not tell the integrity level of individual data records, or individual data items. However, given a view with integrity level I_k, and a randomly chosen data record r, the probability that r is a corrupted record will be I_k, according to Definition 5.1.

The number of data items is not easy to count in a database system. When the number of transactions being considered is big enough, we can assume that the number of data items modified by a transaction will abide by a specific probability distribution. Therefore, the rate that data items are injected into the database system will be proportional to the throughput of transactions in the long run. Lemma 5.2 provides a different way to estimate the integrity level of a system.

Lemma 5.2 *Assume that a system starts from a clean status, if we inject data items with integrity level Δ at throughput v, the system integrity level will finally become Δ.*

PROOF: Assume that B is the total number of data items that the system has at the beginning, the following equation

provides the integrity level of the system at a specific time t.

$$I = \frac{B + \Delta vt}{B + vt} \tag{8}$$

Therefore,

$$lim_{t \to \infty} I = \Delta \tag{9}$$

□

Definition 5.3 (Attacking density) *If the total throughput of a database system is θ, we define the* attacking density β *with regard to θ as*

$$\beta = \frac{\lambda}{\theta} \tag{10}$$

where λ is the throughput of malicious transactions within θ.

For example, if the throughput of a system is 500 transactions per second, where there are 100 malicious transactions per second, the attacking density will be $100/500 = 0.2$.

We have the following theorem to determine I_s, given a specific attacking rate β and the probability distribution of the detection latency.

Theorem 5.4 *Given a clean database and attacking density β, if time is long enough, the integrity level of safe zone will be*

$$I_s = \frac{1 - \beta}{1 - \beta F(\tau)} \tag{11}$$

where τ is time that the effects of transactions stay in the unsafe zone, and $F(x)$ is the cumulative probability distribute function of the IDS detection latency.

PROOF: We start from a database with all clean data items. Assume that the throughput of the system is θ.

Within window τ, $\tau\theta$ transactions will be processed and the results will be in the unsafe zone. Within τ, the IDS may detect malicious transactions and cancel them in the unsafe zone. The number of transactions filtered out by the IDS within τ is

$$\theta\tau\beta \int_{-\infty}^{\tau} f(x)dx = \theta\tau\beta F(\tau) \tag{12}$$

Malicious transactions not detected by the IDS within window τ will be merged into the safe zone. The number of such transactions is

$$\theta\tau\beta \int_{\tau}^{\infty} f(x)dx = \theta\tau\beta(1 - F(\tau)) \tag{13}$$

Therefore,

$$\Delta = \frac{\theta\tau(1 - \beta F(\tau) - \beta(1 - F(\tau)))}{\theta\tau(1 - \beta F(\tau))} \tag{14}$$

Thus, according to Lemma 5.2, we have

$$I_s = \Delta = \frac{1 - \beta}{1 - \beta F(\tau)} \tag{15}$$

□

Similarly, we can use the following equation to estimate the overall integrity level of the user's view.

Theorem 5.5 *Given a clean database and attacking density β, if time is long enough, the integrity level of the user's view will be*

$$I_u = \frac{1 - \beta}{1 - \beta F(\frac{\tau}{2})} \tag{16}$$

where τ is time that the effects of transactions stay in the unsafe zone, and $F(x)$ is the cumulative probability distribute function of the IDS detection latency.

PROOF: The user does not see the isolation so the integrity level of injected data items will be the competing results that the rate of IDS filtering out bad transactions from the unsafe zone, and the rate that the bad transactions are continuously injected into the database through the unsafe zone.

The expected time that the effects of a transaction stays in the unsafe zone is $\frac{\tau}{2}$, thus, according to the proof in Theorem 5.4,

$$\Delta = \frac{1 - \beta}{1 - \beta F(\frac{\tau}{2})} \tag{17}$$

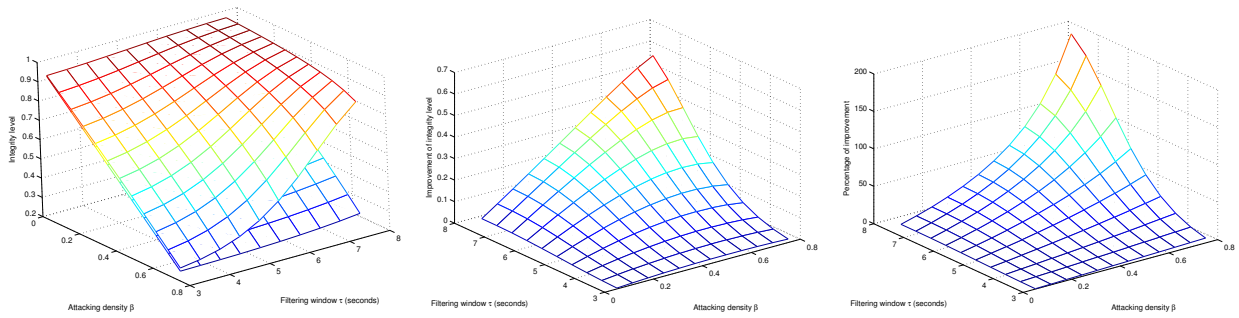
Therefore, according to Lemma 5.2, the theorem is approved. □

The comparison of the integrity levels of I_u and I_s is shown in Figure 4. In Figure 4(a), the lower layer is I_u and the upper layer is I_s. We can observe the improvements of integrity level in the safe zone. While the filtering delay τ and attacking rate β increase, the improvements become very significant. Figure 4(b) shows the improvements in absolute values of integrity level. Figure 4(c) shows the improvement in percentage for references.

5.2 Cost-effectiveness

Since 2 bits are enough to encode 4 possible flags used in our techniques, and the "**timestamp**" field of the record already exists in many DBMSs, the storage costs introduced by the safe zone isolation are extremely low. While the processing latency for users from the web interfaces is negligible, the processing costs to merge records into the safe zone are extremely low too. For instance, one transaction can remove the flag of thousands of data items, thus add them to the safe zone.

Our techniques can significantly reduce recovery costs caused by malicious transactions. According to Equation 12 and Equation 13, the attacking density is reduced from β (to the unsafe zone) to $\beta(1 - F(\tau))$ (to the safe zone) with



(a) Both integrity levels (b) The improvement of integrity level (c) The percentage of improvement

Figure 4. The comparison of the integrity levels of the user's and the safe zone

the unsafe zone isolation. Thus, the reduced the attacking density is

$$\beta F(\tau) \tag{18}$$

In fact, Equation 12 is the reduced number of malicious transactions injected to the database system within time τ. The impacts of β and τ are shown in Figure 5(a).

The analytical results provide insights with limitations. It is difficult to build a mathematical model considering so many parameters and metrics in a real system, especially when the processing overheads are major concerns. We have done experiments to investigate how much processing costs, especially the recovery costs caused by malicious transactions, can be reduced by our techniques.

Our experimental environment is as follows. The database server is PostgreSQL 8.0. The database server runs on a desktop computer with Intel Pentium IV 1.80GHz processor and 512MB RAM. The Operating System is a Debian Linux with kernel version 2.6.12. The system is written in Java. In our experiments, Both good and malicious transactions are mixed to inject into a database system through the JDBC interface. The IDS latency was set to 5 seconds in all experiments.

The experimental results are shown in Figure 5(b) and Figure 5(c). In our experiments, the recovery overheads are measured in percentage of the system load. For example, In the figure, when the throughput is 200 transactions/second, and the attacking density is 34%, the reduced recovery load is around 11%. It means without the protection of two-zone isolation, there would be 11% more system load spending on locating and repair damage caused by malicious transactions. Such recovery overheads are eliminated by our two-zone isolations. In order to obtain stable results, each experiment shown in Figure 5(b) and Figure 5(c) lasted 300 seconds. In Figure 5(c), the reduced number of corrupted transactions includes transactions that refer to corrupted data records. It is not necessary to be malicious by themselves. In the figures, we can also observe that the reduced costs are more significant when the throughput and attacking density increase.

5.3 The impact of IDS accuracy

So far we discussed the integrity level with the assumption of a perfectly accurate IDS. However, we certainly do not expect such case in the real world.

Definition 5.6 (IDS accuracy) *Given s instances of attacks, if an IDs can report $t+u$ intrusion alerts, where t is the number of instances of real attacks and u is the number of false alerts, we have the following definitions. The detection rate*

$$\gamma = \frac{t}{s} \tag{19}$$

The false positive rate

$$\phi = \frac{u}{t+u} \tag{20}$$

and the false negative rate

$$\omega = \frac{s-t}{s} = 1-\gamma \tag{21}$$

Definition 5.6 is consistent to the definition in the literature, e.g., in [3]. A perfectly accurate IDS has detection rate $\gamma = 1$, false positive rate $\phi = 0$ and false negative rate $\omega = 0$. The real world values of them will be $0 \le \gamma \le 1$, $\phi \ge 0$, and $\omega \ge 0$.

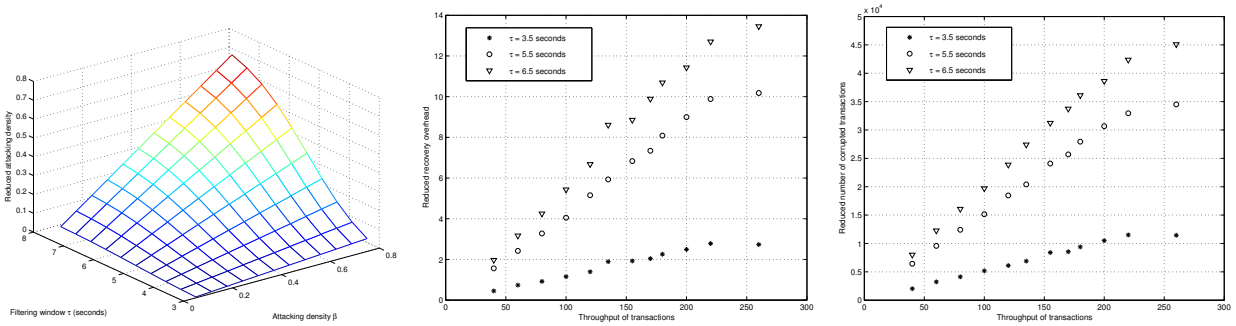
According to Definition 5.6, since $\omega = 1-\gamma$, we will only discuss the effects of γ and ϕ in the rest of this paper.

Corollary 5.7 *Given a clean database and attacking density β, if time is long enough,*

$$I_s = \frac{1-\beta+(\gamma-c)F(\tau)}{1-cF(\tau)} \tag{22}$$

and

$$I_u = \frac{1-\beta+(\gamma-c)F(\frac{\tau}{2})}{1-cF(\frac{\tau}{2})} \tag{23}$$



(a) Reduced attacking density

(b) Reduced recovery overhead with $\beta = 0.4$

(c) Reduced number of corrupted transactions with $\beta = 0.4$

Figure 5. The reduced attacking density and recovery overhead

where τ is time that the effects of transactions stay in the unsafe zone,

$$c = \frac{\phi(\beta - \gamma)}{1 - \phi} \tag{24}$$

$F(x)$ is the cumulative probability distribute function of the IDS detection latency, and γ and ϕ are the detection rate and false positive rate of the IDS respectively.

PROOF: Assume that we start from a clean state and the throughput of the system is θ and $F(\tau)$ is the cumulative probability distribution. Therefore, $\gamma F(\tau)$ is the probability that an instance of attack can be detected with in τ, considering the inaccuracy of the IDS.

The effects of ϕ is interesting since false positive alerts will cancel good transactions in the unsafe zone, which also affects the integrity level of the system. Within the filtering window τ, $\tau\theta$ transactions will be processed and the following number of good transactions will be canceled by the IDS.

$$c\theta\tau F(\tau) \tag{25}$$

Thus, the following number of transactions, including good transactions will be filtered out by the IDS

$$\theta\tau F(\tau)(\gamma\beta - c) \tag{26}$$

The number of malicious transactions not detected by the IDS within τ is

$$\theta\tau\beta(1 - \gamma F(\tau)) \tag{27}$$

Therefore,

$$\Delta = \frac{\theta\tau(1 - cF(\tau) - \beta(1 - \gamma F(\tau)))}{\theta\tau(1 - cF(\tau))} \tag{28}$$

Thus, the corollary is proved according to Lemma 5.2. □

6 Related work

The most similar work to ours are [6, 9, 10]. In their work, each data item may have multiple versions, where one is the clean version and the others are suspicious versions that are created and manipulated by suspicious users. The IDS identifies a specific user as suspicious then all the user's transactions are isolated with suspicious versions that cannot be seen by others. Once the user becomes not suspicious any more, the user's work has to be merged to the clean part of the databases.

While we share the same concept of isolation, our work is different form the above ones in the following aspects. First, we put the effects of all transactions in the unsafe zone regardless the users' identifications. Therefore, our work provides a different kind (in our opinion, a better kind, due to the completeness) of protection to the database system. Second, our work focuses on the separations of different zones to provide multiple integrity levels, instead of the separation between users in the above work that raises different challenges, e.g., version management for different users, and concurrency control when merging transactions. In our approach, there is no hiding data records between users as long as the users are using the same web interface, for example. Thus, there is no isolation latencies between users even if some of them could be malicious. Third, our approach is more efficiently because with 2-bit flags the operations are much quicker. In our approach, the database records in the unsafe zone are always the latest versions if they are not corrupted. Therefore, the version control is simpler. The merging operation is simple too because the merging can be done through changes of flags. Finally, we did comprehensive analyses and experimental study to investigate the improvement of integrity levels and the reduced recovery costs.

Database self-healing techniques [1, 10, 18]allow injecting malicious transactions to the database systems then re-

pairing them later. In such systems, a recovery process competes with malicious injections to increase the integrity level under sustained attacks. Such techniques are able to trace damage spreading and repair the damage in transactional processing systems. However, if attacks happen, all affected transactions will be rolled back (undone) and redone. The major weakness of the above techniques are that rolling back and re-executing damaged transactions increase the response time of the system and may cause a significant processing delay. In such situations, the availability of the system is compromised and the system suffers the vulnerability of Denial of Service (DoS).

Our approach focuses on reducing the attacking density directly through isolation instead of allowing the malicious injections then repair them later. Our study shows that our approach can significantly reduce the overhead of recovery, especially when the filtering window is properly chosen. Nevertheless, the recovery techniques are good complement of our approach.

7 Conclusion

In this paper, we proposed to buffer the user's transactions in an unsafe zone to greatly reduce the possibility of data corruptions caused by transaction level attacks, e.g., SQL injection attacks. Through carefully designed polyinstantiation, the users see no delays for their own applications. Both our analytical results and experimental results showed significantly improvements on the integrity level and reduced recovery costs of malicious transactions. Our approach can be applied to database systems where different QoS are desirable, or high integrity level is mandatory.

Ackknowledgement

We thank anonymous reviewers for their valuable comments. Peng Liu is partially supported by NSF CCR-0233324 and CNS-0716479. Meng Yu is partially supported by NSF CNS-0716240.

References

[1] P. Ammann, S. Jajodia, and P. Liu. Recovery from malicious transactions. *IEEE Transaction on Knowledge and Data Engineering*, 14(5):1167–1185, 2002.

[2] T. Chiueh and D. Pilania. Design, implementation, and evaluation of an intrusion resilient database system. In *Proc. International Conference on Data Engineering*, pages 1024–1035, April 2005.

[3] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Skorić. Measuring intrusion detection capability: an information-theoretic approach. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 90–101. ACM Press, New York, NY, USA, 2006.

[4] W. G. J. Halfond, A. Orso, and P. Manolios. Using positive tainting and syntax-aware evaluation to counter sql injection attacks. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 175–185. ACM Press, New York, NY, USA, 2006.

[5] Y.-W. Huang, S.-K. Huang, and C.-H. Tsai. Web application. In *WWW 2003*, pages 148–159, Budapest, Hungary, 2003. ACM, ACM.

[6] S. Jajodia, P. Liu, and C. McCollum. Application-level isolation to cope with malicious database users. In *the 14th Annual Computer Security Application Conference*, pages 73–82, Phoenix, AZ, December 1998.

[7] S. Jajodia and R. Sandhu. Toward a multilevel secure relational data model. In *SIGMOD '91: Proceedings of the 1991 ACM SIGMOD international conference on Management of data*, pages 50–59. ACM Press, New York, NY, USA, 1991.

[8] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *CCS'03*, pages 251–261, Washington, DC, USA, October 27-31 2003.

[9] P. Liu. Dais: A real-time data attack isolation system for commercial database applications. In *the 17th Annual Computer Security Applications Conference*, 2001.

[10] P. Liu, S. Jajodia, and C. McCollum. Intrusion confinement by isolation in information systems. *Journal of Computer Security*, 8(4):243–279, 2000.

[11] T. Lunt. A survey of intrusion detection techniques. *Computers & Security*, 12(4):405–418, Jun. 1993.

[12] OWASP. Owasp top ten most critical web application security vulnerabilities. http://www.owasp.org/documentation/topten.html, January, 27 2004.

[13] F. S. Rietta. Application layer intrusion detection for sql injection. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 531–536. ACM Press, New York, NY, USA, 2006.

[14] T. Ryutov, C. Neuman, D. Kim, and L. Zhou. Integrated access control and intrusion detection for web servers. *IEEE Transactions on Parallel and Distributed Systems*, 14(9):814–850, Sep 2003.

[15] R. Sandhu and F. Chen. The multilevel relational (mlr) data model. volume 1, pages 93–132. ACM Press, New York, NY, USA, 1998.

[16] R. Sobhan and B. Panda. Reorganization of the database log for information warfare data recovery. In *Proceedings of the fifteenth annual working conference on Database and application security*, pages 121–134, Niagara, Ontario, Canada, July 15-18 2001.

[17] S. Stolfo, D. Fan, and W. Lee. Credit card fraud detection using meta-learning: Issues and initial results. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, 1997.

[18] M. Yu, P. Liu, and W. Zang. Self-healing workflow systems under attacks. In *The 24th International Conference on Distributed Computing Systems(ICDCS'04)*, pages 418–425, 2004.