A Multi-version Database Damage Assessment Model

Kranthi Kurra[1], Brajendra Panda[1] and Yi Hu[2]

[1]Computer Science and Computer Engineering Department
University of Arkansas, Fayetteville, AR 72701, U.S.A.
[2]Computer Science Department, Northern Kentucky University
Highland Heights, KY 41099, U.S.A.

Abstract. Unauthorized data access and malicious data corruption can have very deleterious impact on an organization. To minimize the effect fast and accurate damage assessment and appropriate recovery must be performed as soon as such an attack is detected. This research focuses on damage assessment procedures using multi-version data in the Database System. By utilizing the proposed multi-version data scheme, it is possible to eliminate the impact of malicious database transactions by providing appropriate versions of data items to transactions during damage assessment procedure.

1 Introduction

In any critical information system, defending data from illegal accesses is extremely important [1]. Since protection mechanisms do fail, databases containing sensitive information can be accessed by malicious users. Thus, database intrusion detection systems [2, 3, 8] are employed for detecting malicious activities in Database Management Systems (DBMSs). Evaluating to what extent a database is damaged and which data items are corrupted are extremely important for database recovery. In this research, we propose a damage assessment model that employs multi-version data scheme for the database damage assessment procedure.

2 Past Research on Database Damage Assessment and Recovery

Many researchers have proposed different approaches for post information warfare database damage assessment. Ammann et al. presented an approach based on marking damage to maintain the database consistency [4]. Liu et al. [5] pursued damage assessment by employing relationships among transactions. Panda and Lala [6] eliminated the damage assessment time by saving the dependency relationship to avoid frequent log access. In [9] Jia et al. proposed an approach that uses virtual machines to do "out-of-the-box" cross-layer damage assessment by combining instruction and OS level taint tracking and efficient "what-if" damage assessment methods. In [10], Liu et al. proposed an efficient algorithm for damage assessment and recovery in attack resilient distributed database systems. Yu et al. [7] proposed a model for online attack recovery of work flows. They introduced multi-version data objects that

Kurra K., Panda B. and Hu Y..
A Multi-version Database Damage Assessment Model.
DOI: 10.5220/0004590501000108
In *Proceedings of the 10th International Workshop on Security in Information Systems* (WOSIS-2013), pages 100-108
ISBN: 978-989-8565-64-8

facilitate finding all damages caused by malicious tasks and repairing them automatically. However Yu's approach tries to save every revision of each data item where most revisions may not be useful for future damage assessment and recovery. Our approach also uses multi-version data, whereas transaction characteristics are considered and data revisions that are not useful for future damage assessment are truncated during transaction validation procedure. Thus this approach can save a significant amount of disk spaces and facilitate fast damage assessment.

3 Methodology

The purpose of the database damage assessment and recovery procedure is to assess the damaged or affected data items after an attack and restore the database to a consistent state. Our proposed multi-version database damage assessment approach is based on a model that keeps multi-version data in the DBMS. That is, in order to facilitate the database damage assessment procedure and reduce the denial of service during this procedure, we keep multiple versions of each data in a separate data revision log. For one specific data item, each updating transaction will cause a new version of the data item to be added to the data revision log. In this work, we introduce the concept of transaction validation. Transaction validation process is a separate system process that is utilized to check whether each transaction is legitimate or not. For transactions validated by this process, the corresponding revisions of data updated by these transactions are removed from the data revision log. Basically, we store the most recent validated values for each data item and keep deleting the older values. Each version of data item has to be validated before it is deleted.

3.1 Data Versioning

In order to facilitate the database damage assessment procedure, we keep multiple versions of each data item in the data revision log. Although the transaction id (or transaction timestamp) can be used for this purpose to help identify the transaction that is responsible for a particular revision, it only illustrates the order of transactions submitted to the DBMS instead of the order of transactions committed. Thus it cannot reflect the inter-transaction relationships. For example, transaction T_1 can actually read from a later submitted transaction T_2 that committed earlier. The inter-transaction relationship is decided by the commit sequence of transactions. Each transaction has a commit sequence number associated with it. A transaction with a lesser commit sequence number indicates that it committed before a transaction having a higher commit sequence number.

***Definition 1*: Data Revision**

A *Data Revision* $x_{i,j}$ for data item x is represented by two numbers, i and j, where i represents the transaction number, j represents the commit sequence number of a transaction updating x.

It must be noted that the transaction number i is normally the timestamp of the transaction. For one particular data item, each revision is uniquely represented by $x_{i,j}$.

But for two different data items, they can have the identical pair of numbers *i* and *j* which depicts that both of these two data items are updated by the same transaction.

We illustrate some example transactions as shown in Example 1 given below. It can be observed that we have three transactions: T_1, T_2, and T_5. Suppose transaction T_1 has the commit sequence number 1. Then transaction T_2 and T_5 have the commit sequence numbers 2 and 3 respectively. Table 1 illustrates the data revisions of each data item shown in part of the database log. For example, after the commit of transaction T_5, q's revision is $q_{5,3}$. Number 5 and 3 is the transaction number and commit sequence number of transaction T_5 respectively.

Example 1: Consider the following as a part of a database log.

…..$r_1(x)$, $w_1(y)$, commit; $r_2(x)$, $w_2(x)$, $w_2(y)$, commit; $r_5(z)$, $w_5(q)$, commit;…..

Table 1. Transactions and Data Revisions.

	x	y	z	q
T_1		1, 1		
T_2	2, 2	2,2		
T_5				5,3

Definition 2: **Margin of Error of a Data Item**

The *Margin of Error of a Data Item x* is represented by $x[l, u]$, where l represents the lower bound of data item x and u represents the upper bound of data item x. The margin of error of data item x depicts the possible value range of it in case there are committed transactions which are committed but not yet validated.

3.2 Transaction Classification

We classify each transaction with respect to the data items it reads to perform its functionality. Each transaction is classified into one of the three categories listed below depending on how sensitive the transaction is related to the value of a data item it reads.

***Definition 3*: Sensitive/Critical Transaction**

A transaction T_i is a *Sensitive/Critical Transaction* to data item x if transaction T_i has to read data item x before updating some other data items and the value of data item x must be validated before T_i is executed. We represent this relationship between transaction T_i and data item x as: $x \rightarrow T_i$.

Definition 4: **Unimportant Transaction**

A transaction T_i is an *Unimportant Transaction* to data item x if transaction T_i does not read data item x or does not care about the value of data item x for its updating operation even in case it reads data item x before updating some other data item. We represent this relationship between transaction T_i and data item x as: $x <> T_i$.

***Definition 5*: Tolerating Margin of Error Transaction**

A transaction T_i is a *Tolerating Margin of Error Transaction* to data item x if transaction T_i reads data item x before its updating operations and the update operations can

tolerate the margin of error of data item x in case the latest value has not yet been validated. We represent this relationship between transaction T_i and data item x as: x [l, u] $\rightarrow T_i$.

To better understand the above definitions related to different categories of transactions, let us look at some banking examples.

Example 2: A customer withdraws 2000 dollars from its savings account. The corresponding SQL statement would look like as follows:

T_1: update SavingsAccount where balance = balance – 2000 where Acc# = …

We observe the following relationship *balance* $\rightarrow T_1$. Because the update operation needs to read the current balance of the customer's savings account before doing the update operation, transaction T_1 is *sensitive* to the balance of the account. Thus transaction T_1 is a sensitive transaction to the balance of the savings account.

Example 3: A customer deposits 500 dollars to its savings account. The corresponding SQL statement would look like as follows:

T_2: update SavingsAccount where balance = balance + 500 where Acc# = …

It can be seen that the relationship *balance* <> T_2 holds for transaction T_2. Although the update operation needs to read the current balance of the customer's savings account before doing update operations, transaction T_1 is *not* sensitive to the balance of the account. This is because even some previously submitted transactions are malicious transactions, the transaction used for this deposit operation can still be executed without affecting the customer or bank's operations.

Example 4: A customer makes two deposits and one withdraw operations in three transactions as follows.

T_3: update SavingsAccount where balance = balance + 1000 where Acc# = …

T_4: update SavingsAccount where balance = balance + 200 where Acc# = …

T_5: update SavingsAccount where balance = balance - 800 where Acc# = …

If we assume that the balance of the customer's savings account before the submission of transaction T_3 is 1000 dollars, then the margin of error of data item x after commit of each transaction is illustrated in Table 2. The calculation of margin of error of *balance* is based on the initial balance and the transaction validation process.

For example, after the execution of transaction T_3, depending on the validity of T_3, the correct balance could be 2000 in case T_3 is valid, and it could be 1000 when T_3 is a malicious transaction thus cancelled later. Thus the margin of error is [1000, 2000]. Similarly for transaction T_5 which follows the execution of transaction T_4, when T_5 is a valid user transaction, the balance range would be [200, 1400]. Also, when T_5 is a malicious transaction, the balance range would be [1000, 2200] after malicious effect caused by it is cancelled later. Thus the margin of error of balance is [200, 2200] for transaction T_5.

Table 2. Example Margin of Error and its Relationship with Transactions.

Transaction	Margin of Error of balance
T_3	[1000, 2000]
T_4	[1000, 2200]
T_5	[200, 2200]

4 Our Model

Our model of database damage assessment includes two steps, namely, Transaction Validation Procedure and Clean Data Identification Procedure. Transaction validation procedure is responsible for identifying malicious transactions and validating legitimate user transactions. Clean data identification procedure involves identifying clean data which can be made available immediately to users for their transactions during damage assessment process. We illustrate each of these two procedures in following sub-sections.

4.1 Data Dependency

It's observed from real-world database applications that a user transaction usually reads some data items before updating a data item and also very likely updates some other data items subsequently in the same transaction. Although transaction program changes often, the whole database structure and essential data correlations rarely change. Hu et al. proposed a model [8] that describes data dependencies using *Read Sequence Set* and *Write Sequence Set*. In this paper, we reuse the same concepts as illustrated in the following Definition 6, 7, and 8.

Definition 6: The *Read Sequence* of data item *x* is the sequence with the format <r(d_1), r(d_2), …, r(d_n), w(x)> which represents that the transaction may need to read all data items d_1, d_2, …, d_n in this order *before* the transaction updates data item *x*. It must be noted that each data item may have several read sequences each having a different length. All these sequences together are called *Read Sequence Set R*(*x*) of this data item.

Definition 7: The *Write Sequence* of data item *x* is the sequence with the format <w(x), w(d_1), w(d_2), …, w(d_n)> which represents that the transaction may need to write all data items d_1, d_2, …, d_n in this order *after* the transaction updates data item *x*. It must be noted that each data item may have several write sequences each having a different length. All these sequences together are called *Write Sequence Set W*(*x*) of this data item.

***Definition 8*:** We define the relationship between data item *x* and its dependent, i.e., Read Sequence Set *R*(*x*) and Write Sequence Set *W*(*x*), as *Data Dependency Relationship*.

Figure 1 illustrates an example data dependency. Data item *x* has read dependency relationships with {*a*, *b*, *c*} and {*d*, *e*}. It also has write dependency relationships with {*y*, *z*}, {*l*, *m*, *n*}, and {*u*, *v*, *w*}. Suppose the predefined threshold for weight of data dependency is 40%. Then for the read dependency only {*a*, *b*, *c*} has strong data dependency with *x*. Likewise, for the write dependency only {*u*, *v*, w} has strong data dependency with *x*.

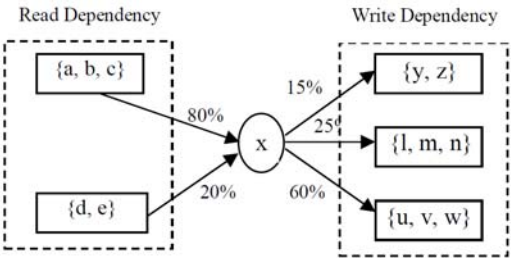

Fig. 1. Data Dependency Concept Example.

We chose to employ the data mining approach proposed in [8] for determining data dependencies in the database system. The classification rules reflecting data dependencies are deduced directly from the database log. These rules represent what data items probably need to be read before an update operation and what data items are most likely to be written following this update operation. Transactions that are not compliant to the data dependencies generated are flagged as anomalous transactions [8]. Compared to other existing approaches for modeling database behaviors [2] and transaction characteristics [3] to detect malicious database transactions, the advantage of this approach is that it is less sensitive to the change of user behaviors and database transactions.

In addition, this technique employed the sequential pattern mining algorithm and a data dependency rule generation algorithm. These dependencies generated are in the form of classification rules, i.e., before one data item is updated in the database what other data items probably need to be read and after this data item is updated what other data items are most likely to be updated by the same transaction. Interested readers may refer to the work on mining data dependencies [8] for further information.

Followings are some example data dependency rules. The first rule states that before updating data item *x*, data item *a*, *c*, *d* and g have to be read in the transaction. The second rule states that after transaction *x* is updated, data item *m*, *n*, and *o* need to be updated subsequently in the same transaction and *m*, *n*, *o* need to be updated in the sequence specified. If in the database log there are transactions that update data item *x* and are not compliant to any of these rules, these transactions are not valid user transactions. The transaction validation procedure raises alarm after identifying malicious transactions.

Example 5: Data Dependency Rules

Read rule: $w(x) \rightarrow r(a), r(c), r(d), r(g)$

Write rule: $w(x) \rightarrow w(m), w(n), w(o)$

4.2 Transaction and Data Revision Validation Procedure

Since storage of multiple versions of each data item leads to a space constraint, we designed an algorithm to validate each committed transaction and the corresponding revisions of data items updated. A transaction T_i is a validated transaction if T_i is

compliant to the read and write rules of data items updated by T_i. If a transaction T_i is validated, all corresponding revisions of data items updated by it are also validated. Also we cannot validate a particular data revision unless the previous version of that particular data item is validated. The transaction validation procedure runs in background while the DBMS is running. It does not have to be run in real time. Although the main purpose of it is to validate transactions submitted to DBMS, it is a user process with low priority and should not significantly affect performance of the DBMS.

Algorithm

1. Initialize the validated transaction list $L_V = \varnothing$
2. Initialize the malicious transaction list $L_M = \varnothing$
3. For each committed transaction T_i

    For each data item x updated in T_i

        If T_i is not compliant to any of these read rules

            Add T_i to malicious transaction list L_M, $L_M = L_M \cup \{T_i\}$

        Else if T_i is not compliant to any of these write rules

            Add T_i to malicious transaction list L_M, $L_M = L_M \cup \{T_i\}$

        Mark revision $x_{i,j}$ of data item x to be validated

        Add Transaction T_i to L_V, i.e., $L_V = L_V \cup \{T_i\}$

        Delete previous revision of data item x from data revision log

4.3 Clean Data Identification Procedure

To reduce the denial of service impact by malicious transactions, the database damage assessment procedure should make the clean data, i.e., unaffected data, available to legitimate users as soon as possible. Our proposed data versioning procedure helps in identifying the correct version of data to serve future data access requests of transactions. The process for identifying the correct version of data proceeds as follows.

First, the data items that are updated by unimportant transactions are made available to users. This is because these unimportant transactions are not affected by the previous value of the data items. For example, these unimportant transactions may simply refresh the old value of the data item, irrespective of whether the old values are correct or not. Second, the data items that are updated by tolerating margin of error transactions are made available to users next. What value of this kind of data items is used to serve the transaction's request? The lower risk value of each of these data items are made available to transactions. Depending on the data semantics, the lower risk value of each data item could be either the lower bound or upper bound of the margin of error of it. The rationale is that even in the case when some previous transactions are either malicious or affected and have not had a chance to be validated, using the lower risk value would have little or no harmful impact to users. Rather, using the lower risk value can help constrain the spreading of damaged data. Third, the DBMS serves transactions the latest revision of data items that are updated by sensitive transactions. The idea is that instead of blocking the user access to these data items until clean data identification process completes, simply serve user the version that is guaranteed to be correct at some past time although the latest image of the data item might be affected. Below we present the formal algorithm for finding

the most desirable data revision for serving the transactions during clean data identification procedure.

Algorithm

1. For each transaction T_i
   For each data item x updated
   If $x <> T_i$ Then
   $x_{m1, n1}$ is made available to the transactions, where n1=max(N), N is the set of commit sequence numbers of x,
   m1 is the transaction number of the transaction with commit sequence number n1
   Else if $x[l, u] \rightarrow T_i$ Then
   If l is the lower risk value Then
   l is made available to the transactions
   Else
   u is made available to the transactions
   Else if $x \rightarrow T_i$ Then
   $x_{m2, n2}$ is made available to the transactions, where n2=min(N), N is the set of commit sequence numbers of x,
   m2 is the transaction number of the transaction with commit sequence number n2

5 Conclusions

In this paper, we have presented an approach for database damage assessment using multi-version data scheme, which facilitates identifying unaffected and damaged data items during the damage assessment procedure. It must be noted that we do not guarantee that the revision provided to transactions is the latest or correct value. Instead, our approach provides the data revision that has the least negative effect on the proper execution of transactions.

Acknowledgements

This work has been supported in part by US AFOSR under grant FA 9550-08-1-0255. We are thankful to Dr. Robert. L. Herklotz for his support, which made this work possible.

References

1. Defending America's cyberspace: National plan for information system protection, version 1.0. The White House, Washington, DC, 2000.
2. D. Barbara, R. Goel, and S. Jajodia. Mining Malicious Data Corruption with Hidden Markov Models. In Proceedings of the 16th Annual IFIP WG 11.3 Working Conference on Data and Application Security, Cambridge, England, July 2002.
3. Y. Hu and B. Panda. Identification of Malicious Transactions in Database Systems. In

Proceedings of the 7th International Database Engineering and Applications Symposium July, 2003.
4. P. Ammann, S. Jajodia, C.D. McCollum, and B.T. Blaustein. Surviving information warfare attacks on databases. In Proceedings of the IEEE Symposium on Security and Privacy, pages 164--174, Oakland, CA, May 1997.
5. P. Liu, P.Ammann, and S. Jajodia. Rewriting Histories: Recovering from Malicious Transactions. In Distributed and Parallel Databases, Vol. 18, No. 1, pages 7-40, January 2000.
6. C. Lala and B. Panda. Evaluating Damage from Cyber Attacks: A Model and Analysis. IEEE Transactions on System, Man, and Cybernetics – Part A, Special Issue on Information Assurance, Vol. 31, No. 4, July 2001.
7. M. Yu, P. Liu, W. Zang, Multi-Version Data Objects Based Attack Recovery of Workflows, Proc. 19th Annual Computer Security Applications Conference (ACSAC '03), Las Vegas, Dec, 2003, pages 142-151.
8. Y. Hu and B. Panda, A Data Mining Approach for Database Intrusion Detection, In Proceedings of the 19th ACM Symposium on Applied Computing, Nicosia, Cyprus, Mar. 2004.
9. X. Jia, S. Zhang, J. Jing, and P. Liu, "Using Virtual Machines to Do Cross-Layer Damage Assessment", In the Proceedings of ACM Workshop on Virtual Machine Security, in association with ACM CCS, 2008.
10. P. Liu and M. Yu. Damage assessment and repair in attack resilient distributed database systems. Computer Standards and Interfaces, 33:96–107, January 2011.