Data Dependency Based Recovery Approaches in Survival Database Systems

Jiping Zheng[1,2], Xiaolin Qin[1,2], and Jin Sun[1]

[1] College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 210016, China
[2] Institute of Information Security, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 210016, China
{zhengjiping, qinxcs, sunjinly}@nuaa.edu.cn

Abstract. Recovering from malicious attacks in survival database systems is vital in mission-critical information systems. Traditional rollback and re-execute techniques are too time-consuming and can not be applied in survival environments. In this paper, two efficient approaches - transaction dependency based and data dependency based are proposed. Comparing to transaction dependency based approach, data dependency recovery approaches need not undo innocent operations in malicious and affected transactions even some benign *blind writes* on bad data item speed up recovery process.

1 Introduction

Database security concerns confidentiality, integrity and availability of data stored in a database [1]. Traditional security mechanisms focus on protection, especially confidentiality of the data. But in some mission-critical systems, such as credit card billing, air traffic control, logistics management, inventory tracking and online stock trading, emphases are put on how to survive under successful attacks [2]. And these systems need to provide limited service at all time and focus on database integrity and availability.

Despite of existing protection mechanisms, various kinds of attacks and authorized users to exceed their legitimate access or abuse the system make above systems more vulnerable. So intrusion detection (ID) was introduced. There are two main techniques, including statistical profiling and signature identification, which can supplement protection of database systems by rejecting the future access of detected malicious attackers and by providing useful hints on how to strengthen the defense. However, there are several inherent limitations about ID [3]: (a) Intrusion detection makes the system attack-aware but not attack-resistant, that is, intrusion detection itself cannot maintain the integrity and availability of the database in face of attacks. (b) Achieving accurate detection is usually difficult or expensive. The *false alarm rate* is high in many cases. (c) The average detection latency in many cases is too long to effectively confine the damage. Some malicious behaviors can not be avoided in DBMS. So effective and efficient recovery approaches must be adopted after the

Y. Shi et al. (Eds.): ICCS 2007, Part II, LNCS 4488, pp. 1131–1138, 2007.

detection of malicious attacks. The rest of this paper is organized as follows. A summery of related work in this area is included in Section 2. In Section 3, recovery approaches in traditional and survival DBMS are given. Section 3.1 describes database and transaction theoretical model. Transaction logging recovery method is put forward in Section 3.2. Then data dependency approaches without/with *blind writes* are emphasized in Section 3.3 and Section 3.4 respectively. Performance analysis is put forward in Section 4. Section 5 concludes the paper.

2 Related Work

The traditional even simplest method for recovering a database to a consistent state is rollback followed by re-execution of the malicious transactions and ones which are dependent upon them. This method, while effective, necessitates an undue amount of work on the part of the database administrator, and requires a knowledge of which transaction was inappropriate one. And, some benign and innocent transactions need to be re-executed. In general, this is a relatively poor option (not efficient) and inadequate for the purposes of most database installations.

In order to overcome the limitations of this simple rollback model, researchers have investigated various other methods for recovering a database to a consistent state. In general, there are two basic forms of post-intrusion recovery methods [4]: transaction based and data dependency based. The difference lies in whether the system recovers modifications to the logs to organize the data modified by interdependencies and associations.

Transaction based recovery methods [5-7], mostly referred transaction logging ones, rely on the ability of an ancillary structure to re-execute committed transactions that have been both committed since the execution of the malicious transactions and affected by those transactions. First ODAM [8] then ITDB [9] and *Phoenix* [10] are survival DBMSs developed by Peng Liu et al. and Tzi-cher Chiueh respectively. These prototypes are implemented on top of a COTS (Commercial-Off-The-Shelf) DBMS, e.g. Oracle, PostgreSQL. In these systems, database updates are logged in terms of SQL-based transactions. ODAM and ITDB identify inter-transaction dependencies at the repair time by analyzing the SQL log and only undo malicious transactions and ones affected by them while *Phoenix* maintains a run-time inter-transaction dependency graph with selective transaction undo. However, these systems rely on the ability of the recovery system to correctly determine the transactions which need to be redone.

Data dependency based recovery methods [11-14] suggest to undo and redo only affected operations rather than undoing all operations of affected transactions and then re-executed them. Panda and Tripathy [12] [13] divide transaction log file into clusters to identify affected items for further recovery. Nevertheless, they require that log must be accessed starting from the malicious transaction till the end in order to perform damage assessment and recovery.

3 Recovery Approaches in Survival Database Systems

As mentioned methods above, our work is also based on the assumption that the attacking transaction has already been detected by intrusion detection techniques. So, given an attacking transaction, our goal is to determine the affected ones quickly, stops new and executing transactions from accessing affected data, and then carries out recovering process. In our methods, we suppose that the scheduler produces a strict serializable history, and the log is not modifiable by users. As the transactions get executed, the log grows with time and is never purged. Also, the log is stored in the secondary storage, so every access to it requires a disk I/O.

3.1 Database and Transaction Theoretical Model

To explain our recovery approaches, first we provide database and transaction theoretical model as below [15]:

Definition 1. A database system is a set of data objects, denoted as DB={x_1, x_2, …, x_n}.

Definition 2. A transaction T_i is a partial order with ordering relation $<_i$, where

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

And r, w, a, c relate to the operation of *read*, *write*, *abort*, and *commit* respectively.

Definition 3. 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. $H=\bigcup_{i=1}^{n} T_i$;
2. $\bigcup_{i=1}^{n} <_i \subseteq <_H$.

Two transactions T_1, T_2 in a history H usually have three relations (assume that T_1 begins first) as shown in Figure1. Figure 1(a) shows that T_1, T_2 are overlapped. During a certain period between two dashed lines, there are operations of both T_1 and T_2. Figure 1(b) shows that the runtime of T_2 in that of T_1. In figure 1(c), T_1 and T_2 do not have the operations executed at the same time. That is, there is no opportunity they read/write the same data items.

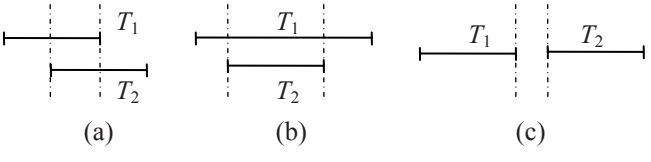

Fig. 1. (a) Two transactions T_1, T_2 are overlapped; (b) T_2 begins after T_1 begins and ends before T_1 ends; (c) T_2 begins after T_1 ends

3.2 Transaction Logging Method

Transaction logging method relies on the availability of read information in the logs and typically this is not a standard feature of commercial database systems. Oracle DBMS does not log read operations in defaulted installations. Existing methods either construct read logs [8] [9] or maintain an inter-transaction dependencies at runtime [10]. In order to build recovery from malicious transactions, transaction logging methods first analyzed transaction dependencies in the history. In general, transaction dependencies are defined as follows [7]:

Definition 4. Transaction T_j is dependent upon transaction T_i in a history H if there exists a data item x such that T_j reads x after T_i has updated x and there are no transactions that update x between the time T_i update x and T_j reads x.

Definition 5. In a history H, transaction T_i affects T_j if the ordered pair (T_j, T_i) is in the transitive closure of the dependent upon relation as described as in definition 4.

Input: Serialized history *H*, malicious transaction *B*.
Output: A consistent state of DBMS.
Initialize: *write_set*={}; *temp_write_set*={};
undo_transaction_set=*B*; *temp_undo_transaction_set*={}.
Steps:
1. Locate the history *H* where the malicious transaction *B* starts.
2. Scan every operation *op*(*x*) forward until the end of the history.
2.1 if w_B(x) then *write_set*=*write_set* $\cup$ {$w_B(x)$};
/*w, r, a, c relate to the operations of *write*, *read*, *abort*, and *commit* respectively */
2.2 else
2.2.1 if $w_{Ti}(x)$ then *temp_write_set*=*temp_wrtie_set* $\cup$ {$w_{Ti}(x)$};
2.2.2 if $r_{Ti}(x)$ then
if *Ti*$\in$ *temp_undo_transaction_set* skip;
else if *op*(*x*)$\in$ *write_set* then
temp_undo_transaction_set= *temp_undo_transaction_set* $\cup$ {T_i};
2.2.3 if a_{Ti} then
temp_write_set= *temp_write_set*- *temp_write_set*|T_i;
/* *temp_write_set*|T_i denotes operations of T_i in *temp_write_set*/
if $T_i\in$ *temp_undo_transaction_set* then
temp_undo_transaction_set= *temp_undo_transaction_set*-{T_i};
2.2.4 if c_{Ti} then
if $T_i\in$ *temp_undo_transaction_set* then
undo_transaction_set= *undo_transaction_set* $\cup$ {T_i};
temp_undo_transaction_set= *temp_undo_transaction_set*-{T_i};
write_set=*write_set* $\cup$ *temp_write_set*|T_i;
temp_write_set=*temp_write_set*- *temp_write_set*|T_i;
else *temp_write_set*=*temp_write_set*- *temp_write_set*|T_i;
3. Undo every transaction in *undo_transaction_set*.

Fig. 2. Algorithm1: Transaction based malicious transaction recovery algorithm

Given a history H and malicious transactions detected previously, transaction logging recovery methods identify the affected transactions then undo both malicious

transactions and affected by them. An effective and efficient transaction logging recovery algorithm [5] is shown as figure 2.

In figure 2, transaction based malicious transaction recovery algorithm undoes the committed malicious transactions and decided affected transactions. If transaction T finally aborted, it need not be undone.

In below history H_1, according to algorithm 1 shown in figure 2, transactions in *undo_transaction_set*=$\{B\}\rightarrow\{B,T_1\}\rightarrow\ \{B,T_1,T_4\}$ need to be undone and re-executed. That is, the recovery process only undoes malicious transaction B and affected transactions T_1, T_4.

$$H_1 : r_B(x)w_B(x)r_B(u)w_B(u)c_B r_{T_1}(x)w_{T_1}(x)r_{T_3}(z)w_{T_3}(z)c_{T_3} r_{T_1}(y)w_{T_1}(y)c_{T_1}$$
$$r_{T_2}(y)w_{T_2}(y)r_{T_2}(v)w_{T_2}(v)a_{T_2} r_{T_4}(u)w_{T_4}(u)r_{T_4}(y)w_{T_4}(y)r_{T_4}(z)w_{T_4}(z)c_{T_4}$$

3.3 Data Dependency Based Recovery Approach Without Considering *Blind Writes*

Definition 6. First operation of T_i is *write*(x) whether or not data item x is dirty or not. So T_i is called *blind write transaction* and operation *write*(x) is a *blind write*.

In most cases, before transactions updated data items, they first read them. But in some cases, especially in malicious attacks environments, various kinds of abnormal behaviors do not obey this rule. So *bind writes* are unavoidable. In this section, we only discuss the situation without *blind writes*.

Input: Serialized history H, malicious transaction B.
Output: A consistent state of DBMS.
Initialize: *comtaminated_data_set*={};
temp_comtanminated_data_set={}; *read_data_set*={}.
Steps:
1. Locate the history H where the malicious transaction B starts.
2. Scan the history operation *op*(x) forward until the end of history.
2.1 if $w_B(x)$ then *comtaminated_data_set*= *comtaminated_data_set* $\cup$ $\{x\}$;
2.2 else for each transaction T_i
2.2.1 if $r_{Ti}(x)$ then *read_data_set*$_i$=*read_data_set*$_i$ $\cup$ $\{x\}$;
2.2.2 if $w_{Ti}(y)$ then
if $r_{Ti}(x)<_i w_{Ti}(y)$ && $x\in$ (*comtaminated_data_set* $\cap$ *read_data_set*$_i$)
$\cup$ (*temp_comtaminated_data_set*$_i$ $\cap$ *read_data_set*$_i$) then
temp_comtaminated_data_set$_i$= *temp_comtaminated_data_set*$_i$ $\cup$ $\{y\}$;
else if $w_{Ti}(x)<_i w_{Ti}(y)$&& $x\in$ (*comtaminated_data_set* $\cap$ *read_data_set*$_i$)
$\cup$ (*temp_comtaminated_data_set*$_i$ $\cap$ *read_data_set*$_i$) then
temp_comtaminated_data_set$_i$= *temp_comtaminated_data_set*$_i$ $\cup$ $\{y\}$;
2.2.3 if c_{Ti} then
comtaminated_data_set= *comtaminated_data_set* $\cup$
temp_comtaminated_data_set$_i$;
2.2.4 if a_{Ti} then
temp_comtaminated_data_set$_i$={};
read_data_set$_i$={}.
3. Refresh data values in *comtaminated_data_set*.

Fig. 3. Algorithm2: Data dependency based recovery algorithm without *blind writes*

Definition 7. Within one transaction or between two transactions, operations are influenced with each other. There exist *read-write* and *write-write* dependencies.

1) *read-write* dependency: if $read_i(x) <_i write_i(y)$ then y is dependent upon x;
2) *write-write* dependency: if $write_i(x) <_i write_i(y)$ then y is dependent upon x.

For *read* operations can not update any data item, there does not exist *read-read* dependency. Figure 3 shows data dependency based recovery algorithm without *blind writes*. The algorithm skips read operations because they do not change data values and every write operation updates data values they first read them.

In history H_1, according to data dependency recovery algorithm in figure 3, only data items x, y, u, z need to be refreshed to correct values.

3.4 Data Dependency Based Recovery Approach with *Blind Writes*

Definition 8. A *blind write* is called a *refresh-write* if the operation belongs to a benign transaction and not dependent of any contaminated data.

2.2.2 if $bw_{Ti}(x)$ then
if $x \in temp\_comtaminated\_data\_set_i \cap read\_data\_set_i$ then
$temp\_comtaminated\_data\_set_i = temp\_comtaminated\_data\_set_i - \{x\}$;
else if $x \in comtaminated\_data\_set \cap read\_data\_set_i$ then
$comtaminated\_data\_set = comtaminated\_data\_set - \{x\}$;
else if $w_{Ti}(x)$ then
if $r_{Ti}(x) <_i w_{Ti}(y)$ && $x \in (comtaminated\_data\_set \cap read\_data\_set_i)$
$\cup$ $(temp\_comtaminated\_data\_set_i \cap read\_data\_set_i)$ then
$temp\_comtaminated\_data\_set_i = temp\_comtaminated\_data\_set_i \cup \{y\}$;
else if $w_{Ti}(x) <_i w_{Ti}(y)$&& $x \in (comtaminated\_data\_set \cap read\_data\_set_i)$
$\cup$ $(temp\_comtaminated\_data\_set_i \cap read\_data\_set_i)$ then
$temp\_comtaminated\_data\_set_i = temp\_comtaminated\_data\_set_i \cup \{y\}$;

Fig. 4. Algorithm3: Modified data dependency based recovery algorithm on step 2.2.2 in figure 3 with *blind writes*

If a malicious transaction with *blind writes*, algorithm 2 need not change anything to satisfy this situation. If there exists a *refresh-write* described in definition 8, we can modify step 2.2.2 in algorithm 2 to get data dependency based recovery algorithm with *blind writes* which shown in figure 4. In figure 4, algorithm 3 shows that benign transactions with *blind writes* can refresh contaminated data values to correct state.

$$H_2 = r_{T_1}(x_2) r_{T_1}(x_4) r_{T_1}(x_5) r_{T_3}(x_5) w_{T_1}(x_3) r_{T_1}(x_4) w_{T_1}(x_4) c_{T_1} r_{T_2}(x_3) r_{T_2}(x_4)$$
$$r_{T_2}(x_5) w_{T_2}(x_2) w_{T_2}(x_1) c_{T_2} w_{T_3}(x_1) w_{T_3}(x_2) c_{T_3} r_{T_4}(x_1) w_{T_4}(x_4) r_{T_4}(x_3) w_{T_4}(x_5) c_{T_4}$$

In history H_2, $w_{T3}(x_1)$, $w_{T3}(x_2)$ are benign *blind writes* which update contaminated data x_1, x_2 to correct state.

4 Performance Analyses

There are two basic measures to evaluate these recovery systems. One is promptness and the other is the complexity of any ancillary structures required. These

measurements are not adequate since data dependency based systems add overhead to the individual transactions, while transaction logging methods append their overhead to the recovery process. In spite of additional data structures used in data dependency recovery algorithms; it will be more effective and efficient than transaction based methods. We use following equation [14] to evaluate our recovery approaches.

$$T = \frac{R * S_R + W * S_W}{S_P} * T_P \tag{1}$$

R: the number of read operations in the history;
W: the number of write operations in the history;
S_R: the size of a read operation record in bytes;
S_W: the size of a read operation record in bytes;
S_P: the size of a page file;
T_P: the page access time in milliseconds.
In our model, we use the following system-dependent parameters shown in Table1:

Table 1. System-dependent parameters used in data dependency based recovery approaches

Name	System-dependent parameters
S_R	40 bytes
S_W	60 bytes
S_P	1024 bytes
T_P	20 milliseconds (ms)

According to algorithms shown above and equation (1), we can calculate the corresponding time consumed in each recovery process shown in table 2.

Table 2. Time consumed in different recovery processes

	Traditional undo and re-execute method	Transaction based recovery method	Data dependency based approach without blind writes	Data dependency based approach with blind writes
H_1	19.531 ms	13.672 ms	7.813 ms	/
H_2	17.188 ms	12.500 ms	/	3.906 ms

In table 2, we can see that data dependency based approaches are more effective and efficient than traditional undo and transaction based recovery methods.

5 Conclusion

In survival DBMS, fast and accurate recovery from malicious transactions is crucial for survival under malicious attacks. In this paper, we first propose transaction logging methods to recover DBMS to a consistent state. Then data dependency based recovery approaches without/with *blind writes* are given. Comparing to read logs needed and whole transaction undone in transaction based methods, data dependency approaches only undo malicious and affected operations. Accompanied with benign

blind writes, data dependency approaches need not undo operations on the data items that have been updated by benign *blind writes*.

Acknowledgements. This work has been supported by the National Natural Science Foundation of China (60673127), High-Technology Research Project of Jiangsu Province of China (BG2004005) and the Aerospace Science Foundation of China (02F52033).

References

1. Elisa Bertino, Ravi Sandhu. Database Security-Concepts, Approaches, and Challenges. IEEE Transactions on Dependable and Secure Computing, 2005, 2(1): 2-19
2. Paul Ammann, Sushil Jajodia, Catherine D. McCollum, Barbara T. Blaustein. Surviving Information Warfare Attacks on Databases. In: Proceedings of IEEE Symposium on Research in Security and Privacy, Oakland, California, 1997, 164-174
3. Peng Liu, Architectures for Intrusion Tolerant Database Systems. 18th Annual Computer Security Applications Conference. San Diego California .December 09 - 13, 2002, 311-320
4. Jeffrey G. Klapheke. Evaluation of Post-Intrusion Database Recovery Methods. Computer Science Seminar, Rensselaer at Hartford, SD2-T1-1, April 24, 2004.
5. Paul Ammann, Sushil Jajodia, Peng Liu. Recovery from Malicious Transactions. IEEE Transactions on Knowledge and Data Engineering, 2002, 14(5): 1167-1185
6. Yi Hu, Brajendra Panda. Identification of Malicious Transactions in Database Systems. In: Proceedings of the Seventh International Database Engineering and Applications Symposium, 2003, 329-335
7. Peng Liu, Paul Ammann and Sushil Jajodia. Rewriting Histories: Recovering from Malicious Transactions. Distributed and Parallel Databases Journal, 2000, 8(1): 7-40
8. Pramote Luenam, Peng Liu. ODAR: An On-the-fly Damage Assessment and Repair System for Commercial Database Applications. In: Proceedings of 15th annual working conference on Database and application Security, 2001, 239-252
9. Peng Liu. Jiwu Jing, Pramote Luenam, Ying Wang, Lunquan Li, Supawadee Ingsriswang. The Design and Implementation of a self-Healing Database System. Journal of Intelligent Information Systems, 2004, 23(3), 247-269
10. Tzi-cker Chiueh, Dhruv Pilania. Design, Implementation, and Evaluation of a Repairable Database Management System. In: Proceedings of 21st International Conference on Data Engineering, 2005, 1024-1035
11. Brajendra Panda, Kazi Asharful Haque. Extended Data Dependency Approach: A Robust Way of Rebuilding Database. In: Proceedings of the 2002 ACM Symposium on Applied Computing, ACM Press, New York, 2000, 446-452
12. Brajendra Panda, Sani Tripathy. Data Dependency based Logging for Defensive Information Warfare. In: Proceedings of the 2000 ACM Symposium on Applied Computing, ACM Press, New York, 2000, 361-365
13. Sani Tripathy, Brajendra Panda. Post-Intrusion Recovery Using Data Dependency Approach. In: Proceedings of the 2001 IEEE Workshop on Information Assurance and Security. United States Military Academy, West Point, NY ,June, 2001, 156-160
14. Brajendra Panda, Rajesh Yalamanchili. Transaction Fusion in the Wake of Information Warfare. In: Proceedings of the 2001 ACM symposium on Applied computing, Las Vegas, Nevada, United States, 2001, 242 - 247
15. Kun Bai, Hai Wang, Peng Liu. Towards Database Firewalls. In: Proceedings of IFTP International Federation for Information, LNCS 3654, 2005, 178-192