Database Recovery from Malicious Transactions: A Use of Provenance Information

Theppatorn Rhujittawiwat[1], John Ravan[1], Ahmed Saaudi[1], Shankar Banik[2] and Csilla Farkas[1]
[1] *Computer Science & Engineering Dept., University of South Carolina, Columbia, SC, U.S.A.*
[2] *Dept. of Mathematics and Computer Science, The Citadel, The Military College of South Carolina, Charleston, SC, U.S.A.*

Keywords: Database, Malicious Transaction, Security, Dependency Graph, Data Provenance.

Abstract: In this paper, we propose a solution to recover a database from the effects of malicious transactions. The traditional approach for recovery is to execute all non-malicious transactions from a consistent rollback point. However, this approach is inefficient. First, the database will be unavailable until the restoration is finished. Second, all non-malicious transactions that committed after the rollback state need to be re-executed. The intuition for our approach is to re-execute partial transactions, i.e., only the operations that were affected by the malicious transactions. We develop algorithms to reduce the downtime of the database during recovery process. We show that our solution is 1.) Complete, i.e., all the effects of the malicious transactions are removed, 2.) Sound, i.e., all the effects of non-malicious transactions are preserved, and 3.) Minimal, i.e., only affected data items are modified. We also show that our algorithms preserve conflict serializability of the transaction execution history.

1 INTRODUCTION

The increase in the number and sophistication of cyberattacks makes cybersecurity one of the top priorities for organizations. One of the recent studies to evaluate the impact of cyberattacks was performed by the Ponemon Institue (Bissell et al., 2019). They studied over 355 companies to evaluate the damages incurred from cybercrime. Their results show that, on average, each company lost \$11.7 million in 2017 and \$13.0 million in 2018. Part of this damage comes from the hourly cost of service downtime.

Malicious transactions further escalate the cost of recovery. Such transactions are often detected at a later time, after malicious transaction as well as other transactions dependent of the malicious one are committed. Efficient recovery methods are needed to reduce the cost of service downtime and recovery work. Traditional recovery methods are based on the re-execution of all non-malicious transactions from a consistent backup point. However, this causes the database to be unavailable until the restoration is completed. Moreover, transactions that were not affected may also be re-executed.

To improve recovery speed, researchers aim to reduce the number of transactions that need to be re-executed. Transactions that were affected by the malicious transactions are identified by transaction dependency and input comparison. The approaches presented by Ammann et al. (Ammann et al., 2002), Liu et al. (P. Liu, P. Ammann, and S. Jajodia, 2000), Panda et al. (Panda and Haque, 2002) have used transaction dependency to identify compromised transactions. The works of Kim et al. (Kim et al., 2012), and Chandra et al. (Chandra et al., 2011) have used input comparison to determine unaffected transactions. Input comparison approaches provide simpler and faster solutions than the dependency approaches. However, transaction dependency approaches provide greater details about which transactions are affected and a higher possibility to salvage good transactions than the input comparison approaches. Therefore, there is a possibility that the dependency-based approaches may provide better overall performance than input comparison-based approaches when it can salvage many good transactions. In this case, the reduced re-execution cost outweighs the slower determination process.

Our work fits into the category of transaction dependency-based recovery. We propose a novel approach to use data provenance to reduce the computational cost of recovery. We define transaction dependency based on an attribute-value assignment. Our work closely relates to the work of Panda et al. (Panda and Haque, 2002). However, their work focuses on the identification of the affected transactions. Their solution requires the shutdown of the database during the recovery process. Our work reduces the duration of the database downtime and lets the unaffected part of the database operate during the recovery process.

Our method uses transaction dependency to determine which transactions have to be re-executed and which attributes have to be updated. We use data provenance to collect crucial information for the recovery process. Instead of the complete shutdown of the database, we only lock the database until the corrupted data items are identified. The unaffected part of the database will become active and perform the service after the identification is finished. The affected parts of the database will be available as soon as the data items are recovered. Our solution will improve the availability of the database during recovery and reduce the downtime from malicious transaction attacks.

We show that our solution is 1.) Complete, i.e., all the effects of the malicious transactions are removed, 2.) Sound, i.e., all the effects of non-malicious transactions are preserved, and 3.) Minimal, i.e., only affected data items are modified. We also show that our algorithms preserve conflict serializability of the concurrent transaction execution.

The paper is organized as follows: Section 2 discusses the existing research used to drive the direction of the solution. Section 3 defines the problem and outlines the system model. Section 4 describes solution, we present our algorithms and provide formal proofs. We conclude in Section 5.

2 RELATED WORK

Many researchers proposed approaches to improve the process to recover from malicious transactions. There are two major approaches to identify affected transactions and reduce re-execution cost: transaction dependency-based approach, and input comparison-based approach.

The works of Kim et al. (Kim et al., 2010)(Kim et al., 2012) and Chandra et al. (Chandra et al., 2011)(Chandra et al., 2013) introduce input comparison-based approaches. In their approaches, if the current input of re-executing transaction is the same as the input when this transaction is originally executed, this re-execution will be skipped because the output will obviously be the same. These methods can be processed easily from reading the transaction log, which records the inputs. They also do not require extra memory and computational power to build a complicated dependency structure between transactions. However, these approaches cannot save the re-execution cost of transactions which have different inputs but produce the same outputs. For example, a transaction that blindly updates values independently from its inputs. These approaches provide a fast and simple solution with less possibility to save re-execution cost.

There are several transaction dependency-based approaches. The works of Ammann et al. (Ammann et al., 2002) and Liu et al. (P. Liu, P. Ammann, and S. Jajodia, 2000)(Liu and Jajodia, 2001) define the transaction by using the read and write sets of transactions. A transaction T_j depends on another earlier transaction T_i if the read set of T_j shares the same data item as a write set of T_i and this data item is not presented in write sets of any transaction committed between them. The corresponding dependency graph can be used to identify transactions that are affected by malicious transactions. The later work of Chakraborty et al. (A. Chakraborty, A. K. Majumdar, and S. Sural, 2010) follows the same direction but uses different granularity. The dependency is defined on columns instead of data items to improve scalability. However, this approach saves fewer unaffected transactions than the previous approaches.

Our work closely relates to the works of Panda et al. (Panda and Giordano, 1998)(Panda and Haque, 2002)(Panda and Jing Zhou, 2003), Lomet et al. (Lomet et al., 2006), Haraty et al. (Haraty and Zbib, 2014)(Haraty et al., 2016)(Haraty et al., 2018), and Kaddoura et al. (Kaddoura et al., 2016). They define dependency based on how each data item is updated. A transaction T_j depends on another transaction T_i if T_j updates any data item according to a data item lastly updated by T_i. These approaches provide greater details on how transactions depend on each other than the read/write set dependency approaches. Thus, they provide the highest possibility to save unaffected transactions comparing all the above approaches. However, the computational costs are also the highest.

To improve the performance, we use the concept of data provenance. Our solution reduces the recovery cost by building data provenance that contains transaction dependency when each transaction is committed. This provides information to speed up the recovery process. There are many data provenance works which influenced our solution such as Xu and Wang (Xu and Wang, 2010), Zhang et al. (Zhang et al., 2012), Hammad and Wu (Hammad and Wu, 2014), He et al. (He et al., 2015), Backes et al. (Backes et al., 2016), and Liang et al. (Liang et al., 2017). They provide different data provenance structures which record information to counter specific malicious attacks.

3 PROBLEM SPECIFICATION AND PRELIMINARIES

Malicious transactions can potentially be committed before they are detected, thus propagating their damage across the database. The problem of malicious transaction recovery has two major parts: identifying the scope of damage and repairing the damage. The damage from a malicious transaction can spread to other transactions. For example, if a malicious transaction T_i updates the value of attribute A_1, every transaction that uses this A_1 will be affected by T_i and will need to be corrected. Furthermore, if an affected non-malicious transaction T_j uses A_1 to update A_2, then A_2 will need to be corrected therefore spreading the damage and the scope of the correction will be reduced. On the other hand, a transaction T_l which does not use A_1 can be indirectly affected if it uses A_2 from T_j.

Our goal is to reduce the cost of locating every affected transaction, limit propagation of the corrupted values, and reduce the recovery work.

Next, we present our recovery framework and corresponding definitions.

Definition 1. *A relation schema R is denoted by $R(A_1,...,A_n)$, where R is the name of the relation and $A_i|i \in \{1,...,n\}$ is an attribute name.*

Definition 2. *An attribute value pair is denoted by (A,v), where v is the value of an attribute A such that v is in the domain of attribute A, i.e., $v \in Dom(A)$.*

Definition 3. *A relation instance r is denoted by r(R), where $R(A_1,...,A_n)$ is the relation schema of r. r is a set of tuples $r = \{t_1,...,t_m\}$, where each tuple is a set of n attribute value pair $t = \{(A_1,v_1),...,(A_n,v_n)\}$ such that each value $v_i \in Dom(A_i)$. We also represent a tuple as an order list of n values $t =< v_1,...,v_n >$ such that each value $v_i \in Dom(A_i)$.*

Definition 4. *An attribute-value assignment is denoted by*

$$t = \{(A_1,v_1),..,(A_j,v_j),..,(A_n,v_n)\} \longrightarrow t' = \{(A_1,v_1),..,(A_j \to exp),..,(A_n,v_n)\} \quad (1)$$

for some attribute value pair of $A_j \in R, (A_j,v_j)$ will be set to (A_j,exp) such that exp is either;

1. *A constant value $v'_j \in Dom(A_j)$.*
2. *An attribute $A_k \in R$ with value v_k, such that $v_k \in Dom(A_k)$.*
3. *An arithmetic expression which includes constant values, attribute names in R, and/or arithmetic operators.*

Example 1: Transaction T_1.

UPDATE r
SET
$A_1 = 10$,
$A_2 = A_1$,
$A_3 = 2A_1 + 20$;

Example 1 shows value assignments by transaction T_1 on relation r with schema $R(A_1,A_2,A_3)$.

T_1 assigns the constant value 10 to attribute A_1, attribute A_2 is assigned the value of A_1, and attribute A_3 is assigned using an arithmetic expression.

Definition 5. *An attribute-value conditional assignment is denoted by*

$$t\{(A_1,v_1),..,(A_j,v_j),..,(A_n,v_n)\} \longrightarrow t'\{(A_1,v_1),..,(A_j \to exp,c),..,(A_n,v_n)\} \quad (2)$$

for some attribute value pair of $A_j \in R, (A_j,v_j)$ will be set to $(A_j,\{exp,c\})$ when condition c is satisfied. The boolean expression c may include truth values, boolean variables, arithmetic expressions, relational operators, and/or boolean operators.
The multiple attributes assignment $t\{(A_1,v_1),..,(A_n,v_n)\} \longrightarrow t'\{(A_1,v_1),..,(A_i \to exp_i,c_i),...,(A_j \to exp_j,c_j),..,(A_n,v_n)\}$ is interpreted as a sequence of update to attributes $A_i,...,A_j$

Definition 6 (Data Dependency Pair)**.** *A data dependency pair is denoted by*

$$Dp_{A_i} = (A_i,\{A_1,...,A_n\}) \quad (3)$$

We also represent it as

$$Dp_{A_i} = (A_i,Set_{A_i}) \quad (4)$$

Where an attribute A_i depends on a set of attributes $Set_{A_i} = \{A_1,...,A_n\}$ such that $A_i \in R, i \in \{1,...,n\}$. The data dependency is transitive, i.e., if (A_i,A_j) and (A_j,A_k) then (A_i,A_k).
For a transaction T, $Dp_T = \{Dp_{A_1},...,Dp_{A_n}\}$ denotes all the dependency pairs $Dp_{A_1},...,Dp_{A_n}$ where attributes $A_i|i \in \{1,...,n\}$ were modified by T.

Definition 7 (Last Assignment Table)**.** *A last assignment table L has the schema*

$$L(A,T) \quad (5)$$

Where A is the column with domain $\{A_1,...,A_n\} \in R$ and T is the column for representing the last transaction that updated attribute A_i. The domain of T is the identities of the transactions.

Definition 8 (Transaction Dependency). *Transaction T_j depends on transaction T_i iff there is an attribute A_j that is modified by T_j and A_j depends on an attribute A_i that was updated by T_i. The transaction dependency is transitive. We also say that transaction T_i affects transaction T_j if transaction T_j depends on transaction T_i.*

Definition 9. *Data dependency record of provenance for transaction T is denoted by*

$$P_T = < ID_T, ts, R\_s, W\_s, Dp > \tag{6}$$

Where ts is the timestamp when transaction T committed, ID_T is the id of transaction T_i, R_s is a set of read attributes, W_s is a set of modified attributes, Dp is the data dependency set of T.

Definition 10 (Conflict Transactions). *Transaction T_1 and transaction T_2 are in conflict if at least one of the following conditions is met:*

1. $R\_s_{T_1} \cap W\_s_{T_2}$ *is not* $\varnothing$.
2. $R\_s_{T_2} \cap W\_s_{T_1}$ *is not* $\varnothing$.
3. $W\_s_{T_1} \cap W\_s_{T_2}$ *is not* $\varnothing$.

Operation Dependency Graph: Assuming a transaction can read and write each attribute only once, we can recreate the partial order of the operation of T as a directed acyclic graph (DAG) G_T. For each record T in P_T:

1. For each attribute A_i in R_s, create a node $r[A_i]$.
2. For each attribute A_j in W_s, create a node $w[A_j]$ and read Dp_{A_j} in Dp.
3. Create a directed edge from each $r[A_i]$ where A_i is in Set_{A_j} to $w[A_j]$.
4. If any subgraph G'_T has only one node with no edge, create a directed edge from the node in G'_T to another node outside of G'_T that has no directed edge pointing to it.
5. If any subgraph G'_T contains an edge but does not contain any edge connect to any node outside of G'_T, create a directed edge from any node that has no directed edge pointing from it in G'_T to another node outside of G'_T that has no directed edge pointing to it.

The step 4 and 5 ensure that all subgraphs will be connected together in the correct direction.

Notation 1. *Given relation R with a single column. $S[R]$ denotes the set containing the attribute values of relation R.*

4 PROPOSED SOLUTION

In this section, we will present the algorithms that are needed to properly recover from malicious transactions. We also present the formal properties and the complexity analysis of our solution.

4.1 Provenance-based Transaction Recovery

In this section we present out recovery algorithms. Algorithm 1 creates a data dependency record of provenance for each transaction. Each record contains transaction ID, timestamp, attributes which are read or written by this transaction, and a set of data dependency pairs. These records will be used to find the minimal number of transactions and attributes that have to be recovered.Algorithm 2 finds the affected transactions and affected attributes. The records generated by Algorithm 1 provide information on how each attribute is updated. By scanning through those records, all affected transactions and affected attributes can be found. All affected transactions will be added into a table containing their transaction ID and ID of transactions, which they depend on. All affected attributes will be locked until they are repaired. The last assignment table L is generated so the system knows when affected attributes can be unlocked. Once affected attributes are updated with values from the re-execution of their last assignment transactions, they will be unlocked.

Algorithm 3 repairs the damage. The affected transaction table AT generated by Algorithm 2 provides information on which transactions have to be re-executed. The re-execution will be processed on a snapshot of the database. The unaffected part of the database can be operated during this repair process. The affected transactions will be re-executed in a topological order based on transaction dependency. The affected attributes will be unlocked once the transactions that last updated them are committed. This provides availability of those attributes as soon as possible.

4.2 Formal Properties and Complexity Analysis

In this section, we analyze and formally prove the properties of our solution. We also evaluate the complexity of our approach and compare it with the cost of traditional transaction recovery.

Theorem 1. *Given a malicious transaction T_k and a transaction execution history, the output of Algorithm*

Algorithm 1: Create Data Dependency Provenance Record.

```
Input: Transaction, $T_i$ and data provenance table $P_T$
Output: Data dependency provenance record of $T_i$ as
    $P_{T_i} =< ID_{T_i}, ts, R\_s, W\_s, Dp >$
 1: Initialization
 2: If data provenance table is not available, create an
    empty table $P_T(ID, ts, R\_s, W\_s, Dp)$
 3: Let ts := "", R_s := {}, W_s := {}, Dp := {}, ts
    = timestamp when $T_i$ is committed, $ID_{T_i}$ = gener-
    ated id based on transaction committed order
 4:
 5: for all attributes $A_i$ in $T_i$ do
 6:     if attribute $A_i$ is assigned by $\{exp, c\}$ then
 7:         $Set_{A_i} = \{\}$
 8:         $Dp_{A_i} = (A_i, Set_{A_i})$
 9:         $W\_s = W\_s \cup A_i$
10:         for all attributes $A_j$ in $\{exp, c\}$ do
11:             $Dp_{A_i} = (A_i, \{Set_{A_i} \cup A_j\})$
12:         end for
13:         Add $Dp_{A_i}$ to $Dp$
14:     else
15:         $R\_s = R\_s \cup A_i$
16:     end if
17: end for
18: Insert record $(ID_{T_i}, ts, R\_s, W\_s, Dp)$ into $P_T$
```

2 is 1.) Complete, 2.) Sound, and 3.) Minimal.
That is,
Completeness: All the transactions that were affected by T_k appear in column $AT(T)$. All the attributes that were affected by T_k appear in column $L(A)$.
Soundness: For each tuple in $AT(T, Set_T)$, Set_T contains only those transactions that were affected by T. For each tuple in $L(A,T)$, T is the last transaction that updated A.
Minimal: Column $AT(T)$ contains only those transactions that were affected by T_k.
Column $L(A)$ contains only those attributes that were affected by T_k.

This theorem shows that Algorithm 2 will find all affected transactions and affected attributes.

Proof Sketch.

Completeness: by induction;
Let $T_1, T_2, ..., T_n$ be n transactions following T_k in transaction log, AT_n contains all transactions affected by T_k from T_1 to T_n, and L_n contains all attributes affected by T_k from T_k to T_n.

The base case: when n = 1.

1. Let L_1 initially contains all attributes updated by T_k.

Algorithm 2: Finding affected transaction.

```
Input: Data dependency provenance table $P_T$, Trans-
    action ID $T_k$ of a malicious transaction, and a
    transaction log
Output: The affected transaction table $AT(T, Set_T)$
    and the last assignment table $L(A,T)$
 1: Initialization
 2: Create empty tables $AT(T, Set_T)$, and $L(A,T)$
 3:
 4: Abort all incomplete transactions
 5: We are locking the database
 6: Read the data dependency provenance record $P_T$
    of the malicious transaction $T_k$ in the table
 7: for all dependency pair $(A_i, Set_{A_i})$ in $Dp$ do
 8:     Insert record $(A_i, T_k)$ into L
 9: end for
10: for all $T_i$ following $T_k$ in transaction log ordered
    by committed time do
11:     Read record of $T_i$ in $P_T$
12:     for all dependency pair $(A_i, Set_{A_i})$ in $Dp$ do
13:         if $Set_{A_i} \cap S[\pi_A(L)]$ is not $\varnothing$ then
14:             if $T_i \notin S[\pi_T(AT)]$ then
15:                 $Set_{T_i} = \{\}$
16:                 for all $A_j \in Set_{A_i} \cap S[\pi_A(L)]$ do
17:                     $Set_{T_i} = Set_{T_i} \cup \pi_T(\sigma_{A=A_j}(L))$
18:                 end for
19:                 Insert record $(T_i, Set_{T_i})$ into AT
20:             else
21:                 $Set_{T_i} = \pi_{Set_T}(\sigma_{T=T_i}(AT))$
22:                 for all $A_j \in Set_{A_i} \cap S[\pi_A(L)]$ do
23:                     $Set_{T_i} = Set_{T_i} \cup \pi_T(\sigma_{A=A_j}(L))$
24:                 end for
25:                 Update record $\sigma_{T=T_i}(AT)$ to
    $(T_i, Set_{T_i})$
26:             end if
27:             if $A_i \notin S[\pi_A(L)]$ then
28:                 Insert record $(A_i, T_i)$ into L
29:             else
30:                 Update record $\sigma_{A=A_i}(L)$ to $(A_i, T_i)$
31:             end if
32:         else if $A_i \in S[\pi_A(L)]$ and $Set_{A_i} \cap S[\pi_A(L)]$
    is $\varnothing$ then
33:             Delete record $\sigma_{A=A_i}(L)$
34:         end if
35:     end for
36: end for
37: We are unlocking only the attributes in database
    $\notin S[\pi_A(L)]$
```

2. AT_1 contains T_1 if T_1 update an attribute A_1 depending on an attribute A_k in L_1.

3. Records of T_1 in data provenance table P_T shows whether A_1 depends on A_k.

Algorithm 3: Repairing.

```
Input: The affected transaction table $AT$, the last assignment table $L$, and the snapshot of database at the time before the malicious transaction committed
Output: A consistent database state where all malicious and affected transactions are undone
 1: Initialization
 2: Create empty ordered list $FixQueue =< T_1, T_2, ... >$ with the size equal to the number of records in AT
 3:
 4: for all $\pi_{Set_T}(AT)$ do
 5:     $Set_T = Set_T - T_k$
 6: end for
 7: while AT is not $\varnothing$ do
 8:     if $\pi_{Set_{T_i}}(AT)$ is empty then
 9:         Add $T_i$ to FixQueue
10:         Delete record $\sigma_{T=T_i}(AT)$
11:         for all $\pi_{Set_T}(AT)$ do
12:             $Set_T = Set_T - T_i$
13:         end for
14:     end if
15: end while
16: if $T_k \in S[\pi_T(L)]$ then
17:     Fixed attributes F := $\pi_A(\sigma_{T=T_k}(L))$
18:     for all $A_i \in S[F]$ do
19:         Unlock $A_i$
20:     end for
21: end if
22: for all $FixQueue[i]$ where i = 0, i++ do
23:     execute FixQueue[i]
24:     if $FixQueue[i] \in S[\pi_T(L)]$ then
25:         Fixed attributes F := $\pi_A(\sigma_{T=FixQueue[i]}(L))$
26:         for all $A_i \in S[F]$ do
27:             Update database with $A_i$
28:             Unlock $A_i$
29:         end for
30:     end if
31: end for
```

4. If A_1 depends on A_k, A_1 is also affected by T_k so T_1 is added to AT_1 and A_1 is added to L_1.

The induction case: assume AT_{n-1} contains all transactions that are affected by T_k from T_1 to T_{n-1}, and L_{n-1} contains all attributes affected by T_k from T_k to T_{n-1}.

1. Consider transaction T_n update an attribute A_n.
2. Records of T_n in P_T shows whether A_n depends on any attribute A_l in L_{n-1}.
3. If A_n depends on A_l, A_n is also affected by T_k so T_n is added to AT_n and A_n is added to L_n.

Conclusion: By the principle of induction, AT_n contains all transactions affected by T_k and L_n contains all attributes affected by T_k.

Soundness: Assume that there is a transaction T' in Set_T that is not affected by T. But then, because there is an attribute in T' that is dependent on an attribute of T (Algorithm 2 lines 16 and 17). This is a contradiction of our initial assumption.

Minimality: Trivially follows. □

Theorem 2. *Given a malicious transaction T_k and a transaction execution history, Algorithm 3 recovers the database to a consistent state, such that*

1. *All the effects of T_k are removed.*
2. *All of the effects of the other transaction are preserved.*
3. *The recovered history is conflict serializable.*

First we note, that by Theorem 1, $AT(T, Set_T)$ and $L(A,T)$ are complete, minimal, and sound, thus support 1.) and 2.) above.

Next, we show by contradiction, that all the effects of T_k are removed by Algorithm 3.

Proof Sketch.

Consistency by contradiction;

1. Assume that some inconsistent attribute A_i that is affected by malicious transaction T_k and A_i is not fixed by Algorithm 3.
2. But attribute A_i must be assigned by a transaction T_i that is affected by T_k.
3. Algorithm 3 executes all transactions in AT by their committed order on the given snapshot and AT contains all transactions that are affected by T_k.
4. AT contains all transaction id which are affected by T_k. So T_i is in AT.
5. Since T_i is re-executed from a consistent state without affect from T_k by Algorithm 3, all attributes assigned by T_i are consistent including A_i.
6. So A_i must be consistent. This contradicts the supposition that A_i is inconsistent.

Serializability;
Case 1: There is no conflict operation between transaction T_i and T_j.

1. In this case, the following conditions must be true by definition of conflict transactions
   (a) $R\_s_{T_i} \cap W\_s_{T_j}$ is $\varnothing$.

(b) $R\_s_{T_j} \cap W\_s_{T_i}$ is $\varnothing$.
(c) $W\_s_{T_i} \cap W\_s_{T_j}$ is $\varnothing$.

2. The partial order of transaction T_i and T_j can be represented using directed graph.
3. There is no edge between node of T_i and T_j because of the conditions above.
4. So the graph is a directed acyclic graph. This history is conflict-serializable.

Case 2: There is a conflict operation between transaction T_i and T_j.

1. In this case, at least one of the following conditions must be true by definition of conflict transactions
   (a) $R\_s_{T_i} \cap W\_s_{T_j}$ is not $\varnothing$.
   (b) $R\_s_{T_j} \cap W\_s_{T_i}$ is not $\varnothing$.
   (c) $W\_s_{T_i} \cap W\_s_{T_j}$ is not $\varnothing$.
2. Case (a): $R\_s_{T_i} \cap W\_s_{T_j}$ is not $\varnothing$.
   1 T_i always reads attribute value from the snapshot which is a database state preceding a state that T_j writes. So the read operation of T_i always precedes the write operation of T_j.
   2 The partial order of transaction T_i and T_j can be represented using a directed graph.
   3 The edge between node of T_i and T_j is in a direction from T_i to T_j because T_i always precedes T_j.
   4 So the graph is a directed acyclic graph. This history is conflict-serializable.
3. Case (b): $R\_s_{T_j} \cap W\_s_{T_i}$ is not $\varnothing$.
   1 Algorithm 2 discovers all attributes that are affected by a malicious transaction T_k as claimed in Theorem 1 and it locks all those attributes.
   2 There are 2 possible cases as follows:
   i) The attribute A_i in $R\_s_{T_j} \cap W\_s_{T_i}$ is not an affected attribute.
   ii) The attribute A_i in $R\_s_{T_j} \cap W\_s_{T_i}$ is an affected attribute.
   2.1 Case (i): the attribute A_i in $R\_s_{T_j} \cap W\_s_{T_i}$ is not an affected attribute.
   2.1.1 The attribute A_i which was written by affected transaction T_i is consistent and will not be overwritten after T_i is re-executed in snapshot. So the history is preserved and the write operation of T_i over A_i precedes the read operation of T_j as in the original history.
   2.1.2 The partial order of transaction T_i and T_j can be represented using a directed graph.
   2.1.3 The edge between node of T_i and T_j is in a direction from T_i to T_j because T_i always precedes T_j.

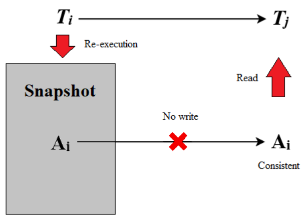

Figure 1: Transaction Dependency in Case 2.b.i.

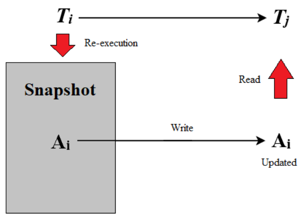

Figure 2: Transaction Dependency in Case 2.b.ii.

   2.1.4 So the graph is a directed acyclic graph. This history is conflict-serializable.
   2.2 Case (ii): The attribute A_i in $R\_s_{T_j} \cap W\_s_{T_i}$ is an affected attribute.
   2.2.1 The attribute A_i is locked by Algorithm 2 until the re-execution of affected transactions which modifies A_i is finished and the consistent value of A_i is updated. So the write operation of T_i always precedes the read operation of T_j.
   2.2.2 The partial order of transaction T_i and T_j can be represented using a directed graph.
   2.2.3 The edge between node of T_i and T_j is in a direction from T_i to T_j because T_i always precedes T_j.
   2.2.4 So the graph is a directed acyclic graph. This history is conflict-serializable.
4. Case (c): $W\_s_{T_i} \cap W\_s_{T_j}$ is not $\varnothing$.
   1 Algorithm 2 discovers all attributes that are affected by a malicious transaction T_k as claimed in Theorem 1 and it locks all those attributes.
   2 There are 2 possible cases as follows:
   i) The attribute A_i in $W\_s_{T_j} \cap W\_s_{T_i}$ is not an affected attribute.
   ii) The attribute A_i in $W\_s_{T_j} \cap W\_s_{T_i}$ is an affected attribute.
   2.1 Case (i): the attribute A_i in $W\_s_{T_j} \cap W\_s_{T_i}$ is not an affected attribute.
   2.1.1 The attribute A_i which was written by affected transaction T_i is consistent and will not be overwritten after T_i is re-executed in snapshot. So the history is preserved and the write operation of T_i over A_i precedes the write operation of T_j as in the original history.

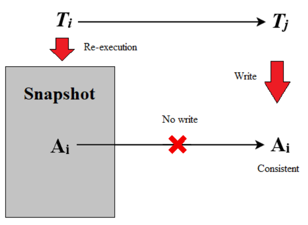

Figure 3: Transaction Dependency in Case 2.c.i.

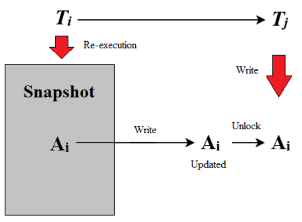

Figure 4: Transaction Dependency in Case 2.c.ii.

2.1.2 The partial order of transaction T_i and T_j can be represented using a directed graph.

2.1.3 The edge between node of T_i and T_j is in a direction from T_i to T_j because T_i always precede T_j.

2.1.4 So the graph is a directed acyclic graph. This history is conflict-serializable.

2.2 Case (ii): the attribute A_i in $W\_s_{T_j} \cap W\_s_{T_i}$ is an affected attribute.

2.2.1 The attribute A_i is locked by Algorithm 2 until the re-execution of affected transactions which modify A_i is finished and the consistent value of A_i is updated. So the write operation of T_i always precedes the write operation of T_j.

2.2.2 The partial order of transaction T_i and T_j can be represented using a directed graph.

2.2.3 The edge between node of T_i and T_j is in a direction from T_i to T_j because T_i always precedes T_j.

2.2.4 So the graph is a directed acyclic graph. This history is conflict-serializable.

5. Consider all the possible cases, the history containing all operations of T_i and T_j is conflict-serializable.

□

Complexity Analysis: Assume n number of transactions were committed after the malicious transaction, the database has m number of attributes, and r number of transactions were affected by the malicious transaction. Let t_{create} denote the time required to create the provenance record for a transaction. The time required to retrieve the attributes that may have been corrupted by a transaction is denoted by t_{check}. The time required to add a transactions to a queue waiting for recovery is denoted by t_{queue}, the time required to reexecute a transaction is denoted by $t_{reexecute}$, and the time required to unlock an attribute is denoted by t_{unlock} .

Algorithm 1. To create the provenance table for each transaction, the loop from line 5-17 will be executed at most m times and the nested loop from line 10-12 will be executed at most m times. This give us: $Time = m^2(t_{create})$.

Algorithm 2. To find all affected transactions, Algorithm 2 checks each transaction. The loop from line 12-35 will be executed at most m times and either the nested loop from line 16-18 or another nested loop from line 22-24 will be executed at most the combined total of m times. The process will be repeated for each transaction. This give us: $Time = nm^2(t_{check})$.

Algorithm 3. To repair the database, the affected transactions will be queued and re-executed. After a transaction is re-executed, then attributes of this transaction will be unlocked if there is no risk of inconsistency. The while loop from line 7-15 will be executed r times until all affected transactions are queued. The nested loop on line 11-13 will be executed r times. The checking process from line 16-20 checks for attributes that are affected by only the malicious transaction, they can be rolled back and unlocked immediately. The loop from line 22-31 will be executed r times to fix each affected transaction. The unlock process from line 24-30 checks for attributes that were last updated by the re-executed transaction is the last transaction which assign the values then unlock. These attributes are unlocked that updated these attributes. This process may be repeated up to m times for each transaction (max. $m \times r$). This give us: $Time = r^2(t_{queue}) + r(t_{reexecute}) + rm(t_{unlock})$.

Re-execute All Transactions. To re-execute all transactions, we have to go through all n transactions then unlock all m attributes. This give us: $Time = n(t_{reexecution}) + m(t_{unlock})$.

Comparison between using Our Solution and Re-executing All Transactions. Our solution use the total time of Algorithm 2 and Algorithm 3: $Time = nm^2(t_{check}) + r^2(t_{queue}) + r(t_{reexecute}) + rm(t_{unlock})$. Without loss of generality, we assume that for most realistic workloads, the checking, queuing, and unlocking processes (in Algorithm 2 and Algorithm 3) require substantially less time than the time needed to re-execute a transaction. The time required by our solution will be bounded by $Time = r(t_{reexecute})$. The time required by re-executing all transactions will be bounded by $Time = n(t_{reexecute})$. Thus, our solution can perform faster on the assumptions that $r < n$ and the processes in our algorithms requires substantially less time than the re-execution process.

5 CONCLUSIONS & FUTURE WORK

As the numbers and sophistication of attacks against databases increase, it is necessary to support efficient and correct recovery from malicious transactions. The solution presented in this paper provides an efficient an correct solution to recover from malicious transactions. This increases the availability of the system without dramatically decreasing performance. We showed that our solution also preserves conflict serializability.

Our ongoing work extends our results to reduce the number of malicious transactions that affect the database. Our approach is to combine snapshot isolation with data provenance. Our provenance data incorporates snapshot isolation to predict transaction behavior. The transaction scheduler can use this information to prioritize transactions and block potential malicious transactions.

ACKNOWLEDGEMENT

This project was partially supported by the NCAE-C Cyber Curriculum and Research 2020 Program.

REFERENCES

A. Chakraborty, A. K. Majumdar, and S. Sural (2010). A column dependency-based approach for static and dynamic recovery of databases from malicious transactions. *International Journal of Information Security*, 9(1):51–67.

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

Backes, M., Grimm, N., and Kate, A. (2016). Data lineage in malicious environments. *IEEE Transactions on Dependable and Secure Computing*, 13(2):178–191.

Bissell, K., Lasalle, R., and Paolo, D. C. (2019). 2019 Cost of Cybercrime Study | 9th Annual | Accenture.

Chandra, R., Kim, T., Shah, M., Narula, N., and Zeldovich, N. (2011). Intrusion recovery for database-backed web applications. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 101–114, New York, NY, USA. Association for Computing Machinery.

Chandra, R., Kim, T., and Zeldovich, N. (2013). Asynchronous intrusion recovery for interconnected web services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 213–227, New York, NY, USA. Association for Computing Machinery.

Hammad, R. and Wu, C. (2014). Provenance as a service: A data-centric approach for real-time monitoring. In *2014 IEEE International Congress on Big Data*, pages 258–265.

Haraty, R. A., Kaddoura, S., and Zekri, A. S. (2018). Recovery of business intelligence systems: Towards guaranteed continuity of patient centric healthcare systems through a matrix-based recovery approach. *Telematics Informatics*, 35(4):801–814.

Haraty, R. A. and Zbib, M. (2014). A matrix-based damage assessment and recovery algorithm. In *2014 14th International Conference on Innovations for Community Services (I4CS)*, pages 22–27.

Haraty, R. A., Zbib, M., and Masud, M. (2016). Data damage assessment and recovery algorithm from malicious attacks in healthcare data sharing systems. *Peer-to-Peer Networking and Applications*, 9(5):812–823.

He, L., Yue, P., Di, L., Zhang, M., and Hu, L. (2015). Adding geospatial data provenance into sdi—a service-oriented approach. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(2):926–936.

Kaddoura, S., Haraty, R. A., Zekri, A., and Masud, M. (2016). Tracking and repairing damaged healthcare databases using the matrix. *International Journal of Distributed Sensor Networks*, 2015:6:6.

Kim, T., Chandra, R., and Zeldovich, N. (2012). Recovering from intrusions in distributed systems with DARE. In *Proceedings of the Third ACM SIGOPS Asia-Pacific conference on Systems*, APSys '12, page 10, USA. USENIX Association.

Kim, T., Wang, X., Zeldovich, N., and Kaashoek, M. F. (2010). Intrusion recovery using selective re-execution. In Arpaci-Dusseau, R. H. and Chen, B., editors, *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, pages 89–104. USENIX Association.

Liang, X., Shetty, S., Tosh, D., Kamhoua, C., Kwiat, K., and Njilla, L. (2017). Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 468–477.

Liu, P. and Jajodia, S. (2001). Multi-phase damage confinement in database systems for intrusion tolerance. In *Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001.*, pages 191–205.

Lomet, D., Vagena, Z., and Barga, R. (2006). Recovery from "bad" user transactions. In *Proceedings of* *the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 337–346, New York, NY, USA. Association for Computing Machinery.

P. Liu, P. Ammann, and S. Jajodia (2000). *Rewriting histories: Recovering from malicious transactions*, pages 7–40. Springer.

Panda, B. and Giordano, J. (1998). Reconstructing the database after electronic attacks. In Jajodia, S., editor, *Database Security XII: Status and Prospects, IFIP TC11 WG 11.3 Twelfth International Working Conference on Database Security, July 15-17, 1998, Chalkidiki, Greece*, volume 142 of *IFIP Conference Proceedings*, pages 143–156. Kluwer.

Panda, B. and Haque, K. A. (2002). Extended data dependency approach: a robust way of rebuilding database. In *Proceedings of the 2002 ACM symposium on Applied computing*, SAC '02, pages 446–452, New York, NY, USA. Association for Computing Machinery.

Panda, B. and Jing Zhou (2003). Database damage assessment using a matrix based approach: an intrusion response system. In *Seventh International Database Engineering and Applications Symposium, 2003. Proceedings.*, pages 336–341.

Xu, G. and Wang, Z. (2010). Data provenance architecture based on semantic web services. In *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, pages 91–94.

Zhang, O. Q., Ko, R. K. L., Kirchberg, M., Suen, C. H., Jagadpramana, P., and Lee, B. S. (2012). How to track your data: Rule-based data provenance tracing algorithms. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1429–1437.