Peer-to-Peer Netw. Appl. (2016) 9:812–823
DOI 10.1007/s12083-015-0361-z

Data damage assessment and recovery algorithm from malicious attacks in healthcare data sharing systems

Ramzi A. Haraty[1] · **Mirna Zbib**[1] · **Mehedi Masud**[2]

Received: 1 October 2014 / Accepted: 20 April 2015 / Published online: 7 May 2015

Abstract In a data sharing system in a cloud computing environment, such as health care system, peers or data sources execute transactions on-the-fly in response to user queries without any centralized control. In this case confidential data might be intercepted or read by hackers. We cannot consider any centralized control for securing data since we cannot assume any central third party security infrastructure (e.g., PKI) to protect confidential data in a data sharing system. Securing health information from malicious attacks has become a major concern. However, securing the data from attacks sometimes fail and attackers succeed in inserting malicious data. Hence, this presents a need for fast and efficient damage assessment and recovery algorithms. In this paper, we present an efficient data damage assessment and recovery algorithm to delete malicious transactions and recover affected transactions in a data source in a health care system based on the concept of the matrix. We compare our algorithm with other approaches and show the performance results.

Keywords Data security · Data exchange · Healthcare data protection · Healthcare data tampering

✉ Ramzi A. Haraty
rharaty@lau.edu.lb

Mirna Zbib
mirna.zbib@lau.edu

Mehedi Masud
mmasud@scientist.com

1 Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon

2 Department of Computer Science, Taif University, Taif, Saudi Arabia

1 Introduction

In a global e-healthcare environment, a patient can receive treatment anywhere and anytime while s/he is on the go. This necessitates sharing of data among different stakeholders, e.g., family physicians, local clinics and hospitals, medical laboratories, pharmacists, and other stakeholders by executing queries and transactions about patients' treatments, medications, and test results. It is desirable to protect the privacy of a patient when peers exchange and update data among themselves. Furthermore, a peer may not wish to disclose his/her identity to others in a network during data exchange to avoid different attacks including sophisticated ones such as target-oriented attacks.

In general, securing information takes place at three levels: prevention, detection and recovery. Prevention might fail and detection might be late, in this case some data might be corrupted. Detection can be split into two categories: the statistical models and the misuse detection [1]. It is assumed that an intruder's behavior is noticeably different from that of a normal user, and statistical models are used to aggregate the user's behavior and distinguish an attacker from a normal user. The aim, after this corruption and after detecting that something malicious has occurred, is to remove and clean the corruption and its consequences.

Prevention, detection and recovery are three important phases in any "live" system. Malicious users manage to overcome preventative security measures and systems. None of the detection systems ensure that an attack will be immediately detected. Hence, damage could spread affecting other "clean" transactions as well. Therefore, the need arises to assess every transaction after the first attack to confirm if it has been affected by the malicious transaction and whether it needs recovery. In our work, we aim at bringing back the integrity of information.

Many works have been carried out in this area; some proposed the use of graphs, while others proposed the use of clusters and sub-clusters, yet others opted for using multiple matrices. Each of the proposed algorithms had its own advantages and disadvantages. In our work, we are interested in the complexity and efficiency of the recovery process. In some cases the adversary's intentions are not only to insert malicious transactions but also to inflict denial of service. In this context many techniques have been provided by researches; some of the most significant are proposed in [2]. In addition, in [3] work has been done to reduce the downtime of the system when performing a damage assessment and recovery from an attack. In [3], the authors proposed the usage of multiple agents to work on the damage assessment and recovery in parallel. Sometimes the size of the log file might increase tremendously before discovering that an attack has occurred which will require more time to assess and recover from the malicious transaction and its effects. This increase in recovery time could lead to denial of service, which is intolerable in health care organizations. We are interested in finding an algorithm that prevents such drawbacks or at least one that reduces them. One of the important issues that should also be addressed is what information should be saved in the log file as we prevent excess I/O. For this purpose, researchers have proposed using auxiliary structures for keeping track of dependencies [4–6].

Previous work done in this area always required the need to scan the entire set of data to ascertain that it is clean, i.e., transactions that were committed before the attack [7, 8]. In our approach, the use of matrices have given us many advantages among which the reduction in the I/O and the advantages in indexing. The fact that the matrices can easily be indexed has given us the ability to skip transaction that we are sure are clean. Hence, such advantages have given us an aid in providing a fast and efficient algorithm that is capable of resuming the integrity of the original data in an expedited fashion.

In this paper we present a damage assessment and recovery algorithm that keeps a matrix along with the logging process. This matrix saves the dependency between transactions and data items. During the recovery process all the needed information will be retrieved from the matrix. The aim of this work is to ensure speedy and efficient recovery. It requires only scanning part of the matrix to be able to discover the dependency rather than scanning the entire log file. In addition, the use of bits in our algorithm requires less processing. Dependency of transactions is saved in only one matrix, which requires less computational time and space. No logical operations and no graphs are used in this model as is the case with other approaches. All of this contributes to making our approach more time and safety efficient than previously proposed algorithms. The remainder of the paper is organized as follows: section 2 presents related works. Section 3 presents the proposed algorithms. Section 4 presents the experimental results. And section 5 concludes the paper.

2 Literature review

In a data sharing system, such as the health care system, it is hard to identify which user is malicious and which is authenticated when dealing with electronic data and transactions. The system treats all the users the same and accepts their transactions. Therefore, every user is considered a malicious user and the transactions are executed; however, actual action is not taken in the database until a certain period of time elapses [9]. After execution of the transactions, the behavior is classified as either malicious or non-malicious [10]. Accordingly, and based on this behavior, transactions can be committed or aborted.

It is normal practice to directly recover the system when an attack is detected by an intrusion detection system. All transactions from the point of the attack and onwards should be assessed whether they are affected or not. Two approaches exist for assessing the malicious transaction effects: transactional dependency [11] and data dependency [12]. Transactional dependency stores all dependent transactions that depend on one another in one segment. Consequently, the log is divided into multiple segments. Panda and Haque [13] used the data dependency approach where each segment stores only dependent operations. Therefore, a transaction's operations may be stored in different segments. Each read/write operation in the transaction has a block number; this number shows dependency between operations. The authors suggest the use of a directed damage demonstration graph, which only presents the affected data items. The disadvantage of this algorithm is that it is limited to data dependency. Panda and Gordano in [14] proposed two data dependency algorithms; the difference between them is that in the first the damage assessment and recovery algorithms are performed simultaneously; whereas, in the second each one is performed separately. This difference implies different behaviors at the algorithm level as well. For example, when both damage assessment and recovery are done simultaneously, the system will have to go through denial of service for a longer period of time in order to recover completely. In both approaches the damage assessment works using directed graphs, where the nodes represent data items. When the intrusion detection system reports the occurrence of a malicious transaction, a node for each data item will be created. This graph helps in mapping how the damage has spread.

The authors in [15] suggested segmenting the log files so the work would only be done on the part of the log that is affected. Operations are clustered according to their dependency where each cluster contains dependent operations. This clustering is done periodically for the active transactions.

Every operation will be stored in only one cluster, but a transaction can belong to more than one cluster. However, deleted transactions cannot be retrieved, so maliciously deleted transactions might be skipped.

Fu et al. [16] proposed the Fine Grained Transaction Log (FGTL). The disadvantage of this method is that the association degree of the transaction and the FGTL are inversely proportional. Hence, even though the integrity of the log file will be preserved; yet, the services will face a major degradation.

Lomet et al. [17] dealt with the problem of bad user transactions that result in invalid data. Their method identifies the initial invalid data and all subsequent data that depends on it. Only transactions writing invalid data need to have their effects "de-committed". The authors identified this closure of invalid data, via logging data reads. Their method then removes only the effects of invalid transactions.

Traditional methods suggest scanning the log file from the start of the attack until the end of the file to undo and redo the affected transactions. In [18], the authors suggested segmenting log files into clusters. However, the size of the dependent transactions cannot be controlled; and hence, the clusters may grow in size. This imposes a weakness in the model, since two dependent transactions may belong to two different clusters because of size limitation. Hence, more work will be needed. To solve this problem, Haraty and Zeitunlian [19] proposed the use of clusters and sub clusters. Data inside a cluster are records that have some data dependency; whereas, data in the same sub cluster could be there for one of the following two reasons: number of data items or space occupied. Zhou, Panda and Hu [20] proposed a similar model for distributed databases. The proposed model works on transaction dependency in order to recover from malicious attacks. This work extends the work of Zhou and Panda [21] and requires additional structures to recover when working on distributed databases.

Xie, Zhou, Feng and Hu [22] suggested the use of a before-image (BI) table to keep track of all deleted transactions and to help analyze potential reads. The BI is a data object created in the database. BI tables are tables that are not accessible by users and have the same structure as the original tables, except that they do not have any constraints. To avoid the problem of data redundancy, Xie suggested using a time window to delete data items and restrict the size of the BI tables.

The use of a Local Damage Assessment and Recovery (DAR) Manager and a Local DAR Executer on each site was suggested by Liu and Yu [23]. The Local DAR Executer starts by identifying all affected sub-transactions and continues to clean them. The algorithm requires global coordination between different sites. The algorithm starts by identifying the bad transactions and then sending them to the Local DAR Manager for cleansing.

Lala et al. [24] suggested clustering the transactions so there is no need to search transactions that have no effect. In this model, an additional column was added to each of the matrices that contain the cluster ID to which this transaction belongs. In such a method, the cluster may contain transactions more than just what is directly related. In addition, the cluster size must be manageable or else the cluster may contain all the transactions. Yet, if the cluster size is limited, then some related transactions might be in different clusters. Thus, all of the clusters need to be checked.

Ray et al. [25] performed analysis on existing algorithms along with a suggestion of new techniques. The complexity analysis was performed to check the complexity of the proposed algorithm as well as the algorithm suggested in [17]. The aim of this paper was to reduce the damage assessment latency so damage spreading will not occur. The disadvantage found in their work was that the log file of this algorithm is huge; and therefore, it will take time to scan the part of the log after the malicious transaction. As for the algorithm proposed in [17], it showed a worst-case running time of $O(v\log v+s)$, where v represents the number of affected transactions and s represents the sum of sizes of the transaction records.

If we compare our approach to the previously discussed approaches we can find that we have overcome many shortcomings. Our approach, unlike what was proposed by Panda and Gordano in [14], overcomes the issue of denial of service, which might be an indirect intention of the attacker. Moreover, if we compare our approach to others that base their work on matrices, we note that in ours we use one matrix rather than two, which means we are already saving almost half the space. In addition, we cut down on operations manipulating the two matrices, which also saves time. Despite the fact that clusters gave the advantage of classifying the data, yet they had the disadvantage of having the same transaction belonging to more than one cluster; and consequently, the cluster sizes may grow enormously. The usage of a matrix has also overcome this issue, as any transaction will be present only once in our matrix; hence, the size of the matrix is known.

3 The proposed approach

In this section, we present the algorithm for data damage assessment and recovery to recover from malicious transactions in a data source. The algorithm triggers when a set of malicious transactions are received from an intrusion detection system (IDS). Our algorithm is responsible for assessing the committed transactions and classifying them as clean or affected transactions; consequently it deletes the malicious transactions and recovers the affected ones. Our damage assessment algorithm is only responsible for assessing affected transactions; it is the responsibility of the IDS to provide a list of transactions that are malicious.

In our approach, we assume we have a rigorous serializable history, as serializability theory provides correctness [26]. A sequential log file is also maintained in which only committed transactions are saved. This log file cannot be accessed by any users at any time and it will be used during the recovery process. Our approach requires also the use of check points [27].

The checkpoint is a database event used in order to ensure a faster detection and recovery process. Check points are committed after an agreed upon time interval after which we suspect that malicious transactions have either been detected or the committed data is clean (or else the IDS would have detected the set of malicious transactions). Check points are beneficial in our proposed algorithm to reduce the space and reading time.

After a check point has been reached, the dependency matrix will be purged. The dependency matrix will be purged at each check point as we assume that the probability of having a malicious transaction became very low. Still, if a malicious transaction was detected that occurred before the check point (worst case scenario), the log file will be used to rebuild the dependency matrix and then go through the same process as if no check point has occurred. The rebuilt dependency matrix will have the same characteristics as the dependency matrix that was built before the check point. The matrix will only save the committed transactions. The importance of the dependency matrix is in the detection process. It will be used to discover the dependency among transactions; and hence, classify them into affected or clean transactions. Panda and Zhou in [8] used more than one matrix and applied logical operations between these matrices to discover dependencies. However, in our model only one matrix is used and it shows the dependencies without any logical operations. Complementary arrays are constructed in special cases as explained below.

The main structure that is used in our algorithm is the matrix. We assume that the matrix is built dynamically along with the execution of every transaction. As transactions are executed, and later committed, they will be added to the two-dimensional matrix; thus, building the dependency matrix. Columns of the matrix correspond to the data items present in the database; whereas, the rows represent the different committed transactions. Each transaction can either be blindly written - a transaction that does not depend on any previously committed transaction or a transaction that depends on previously committed transactions. A blindly written transaction is either a clean transaction (non-affected) or a malicious transaction (in this case only identified by the IDS). For every transaction, each data item will have a value depending on the operations that the transaction has gone through, which is represented as follows:

- 0: if the data item is unmodified by a transaction.
- 1: if the data item is blindly written by a transaction; data from previous transactions is not needed.
- A positive transaction ID: if dataitem1 that is accessed by T_x, is identified in the matrix with entry T_y such that $y<x$; this indicates that dataitem1 is updated according to the last modified value of dataitem1 by n T_y.
- A negative transaction ID: this means that this data item was modified based on data items read from different transactions.

Consider a transaction T_x. To modify dataitem1, T_x needs to read dataitem4 of transaction T_y, dataitem3 of transaction T_w, and dataitem1 from transaction T_v, where y, w and v<x. In this case, the entry in the matrix for that data item will be $-T_x$. Still, $-T_x$ alone will not help us in the recovery process as it does not show which transactions have affected it. To solve this problem, we added a complementary array that will only be manipulated in such cases. To illustrate, the entry of the main matrix for dataitem1 in transaction T_x will have $-T_x$. Then, in the second array, the index will be the transaction ID that has been affected by other transactions, T_x. The index T_x will be pointing to T_y, T_w and T_v. This is depicted in Fig. 1.

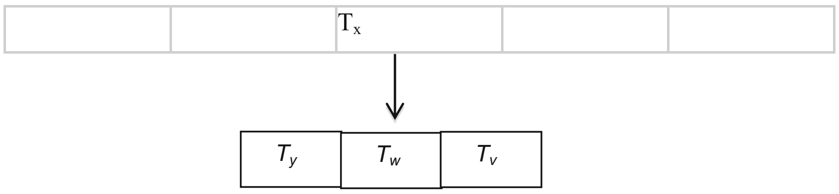
3.1 The damage assessment algorithm

Our proposed algorithm uses two array structures: the dependency matrix and the complementary array. The former stores dependencies between transactions. The latter saves the dependency between the transactions and many other previously committed transactions. These structures, along with the set of malicious transactions provided by the IDS, are the main elements for our proposed detection algorithm. One of the characteristics of the log file and the matrix is that both are sequential. Transactions in both the log file and the dependency matrix are stored according to their commitment such that there is no transaction T_j where $j<i$ and T_j depends on T_i. Whenever recovery is required, our detection algorithm identifies the minimum transaction ID among the set of malicious transactions T_{Mi}. When the detection begins, the algorithm skips every entry before the T_{Mi} since they are not affected by any transaction in the set of malicious transactions. The T_{Mi} will be skipped as well since we already know that this is the malicious transaction, it needs to be deleted. This way, the algorithm will be reducing the effort that could be used on transactions that we know are clean.

After the row $T_{Mi,}$ the matrix will be traversed row by row and for each data item in that transaction (row), a check will be performed to see how the data have been affected. If the entry

	1	2	3	….	30
T_v	1	0	0		1
T_w	0	1	0		0
….					
T_y	0	0	0		1
T_x	$-T_x$	0	0		0

(a)



(b)

Fig. 1 The dependency matrix (**a**), with its complementary array (**b**)

is a ‘0’ or ‘1’, then this means that this data item has not been modified in this transaction or that it is blindly written. Hence, our algorithm will skip that column as the data item is written cleanly and then moves to check the following columns.

On the other hand, positive and negative transaction IDs show a possibility that the transaction might be affected. If the entry contains a positive transaction ID, which means that the data item of the current transaction is dependent on the

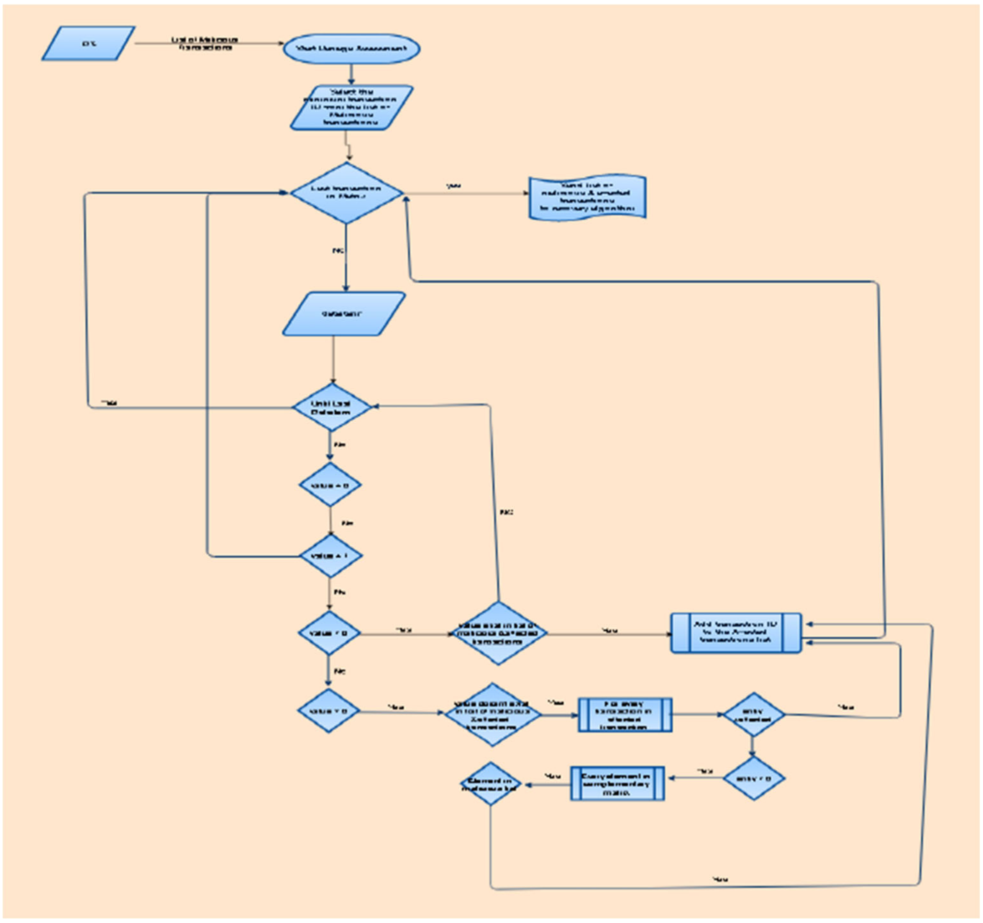


Fig. 2 A flow diagram of the damage assessment algorithm

```
Receive the set of malicious transactions which should be sent by the intrusion detection system
Select the minimum transaction ID among the malicious transactions
For every transaction in the matrix starting from the minimum malicious transaction ID to the end of the matrix
        For each data item
            If (entry == 0) then
                Move to the next row
            Else if (entry == 1) then
                Move to the next row
            Else if (entry < 0 && entry belongs to malicious transactions)
                Add the current transaction to the set of affected transactions
                Move to the next row
            Else if (entry > 0 && entry does not belong to malicious transactions)
                For every transaction in the affected transactions set
                    If (entry == $T_{affected}$)
                        Add entry to affected transactions set
                        Move to the next row
                    Else if (entry < 0)
                        Search ComplementaryArray for key == entry
                        For each element in ComplementaryArray [entry]
                            If(element belongs to malicious transactions)
                                Add the current transaction to the set of affected
                                transactions
                                Move to the next row
                            Else if (element belongs to the set of affected transactions)
                                Add the current transaction to the set of affected
                                transactions
                                Move to the next row
```

Fig. 3 The damage assessment algorithm

transaction, with the transaction ID present in that entry, then we need to check if this transaction ID belongs to the affected or malicious transaction. For example, let us consider the following case: upon searching the matrix reaching transaction T_i, and upon checking if this transaction is affected, we check each data item in the matrix. We consider the case where for that row, data item x has the following ID, T_j. This shows that to write data item x in transaction T_i the transaction read data from T_j. Hence, we search among the set of malicious transactions to check if T_j belongs to it, if it does then we add T_i to

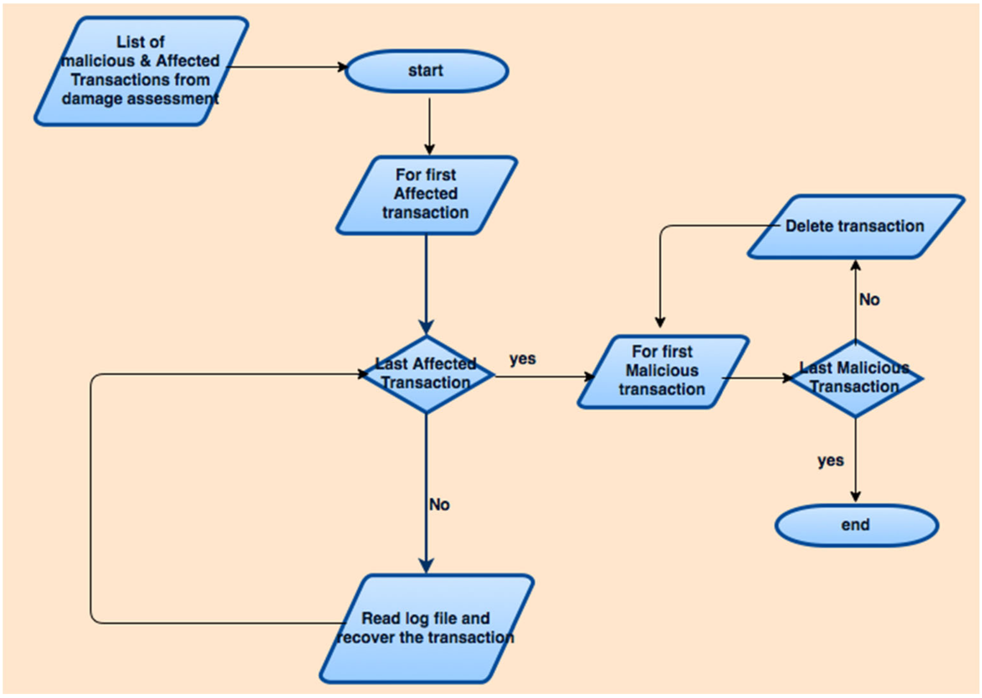

List of malicious & Affected Transactions from damage assessment
start
For first Affected transaction
Last Affected Transaction
yes
No
Read log file and recover the transaction
For first Malicious transaction
Last Malicious Transaction
No
Delete transaction
yes
end

Fig. 4 A flow diagram of the recovery algorithm

the set of affected transactions. Then, our algorithm will not continue searching the other data items and skip to the transaction following T_i. If T_j does not belong to the set of malicious transactions, we check if it belongs to the set of affected transactions. If T_j belongs to the set of affected transactions then we have indirect dependency and T_i should be added to the affected transactions. The algorithm checks the other data items for that transaction. The best case scenario is when the first data item in that row is affected. Hence, the transaction would be added to the set of affected transactions and the other data items will be skipped.

The last case is having a negative transaction ID. This shows that the transaction we are currently looking at has been affected by more than one previously committed transaction. In this case, we will directly allocate the index of this transaction in the complimentary array to retrieve the content corresponding to the entry that we are currently checking. Consequently, the retrieved content will then be tested to check if any of this content has the same ID as any of the transaction IDs that are classified as malicious or affected transactions. Similarly, the steps performed in this case are the same as in the previous case; i.e., if a malicious or affected transaction was among the transactions that this row (transaction) depends on, then we will add it to the affected set and skip to the next row. The flow of damage assessment algorithm is depicted in Fig. 2 and its steps are summarized in Fig. 3.

3.2 The recovery algorithm

After the completion of the damage assessment algorithm, the recovery algorithm will be triggered by receiving the set of malicious and affected transactions. The malicious transactions will be deleted, while the affected transactions will be recovered to act as if no malicious transactions have occurred. The algorithm will run until we reach a stable state in the database - a state where all of the data is consistent (i.e., no malicious transaction exits and any affected transactions are recovered). The sets of malicious and affected transactions will be traversed and for each transaction we will go back and check what information the log file has about it in order to proceed with the proper update. The flow of the algorithm is depicted in Fig. 4, and its steps in Fig. 5.

Table 1 Parameters used in I/O calculations

Parameters	Values
Space taken by a read operation of a transaction in the log	40 bytes
Space taken by write operation of a transaction in the log	60 bytes
Page Size	2 KB
Page I/O Time (in milliseconds)	15

3.3 An example

Consider a database for a company that contains information about the following:

- Doctor: a unique identification number for each Doctor (DID), first name (FName), last name (LName), date of birth (EDOB), and profession (Prof).
- Patient: a unique identification number for each patient (PID), patient name (PName) and the Patient phone number (PNumber).
- Categories: a unique identification number for each category (CatID) and category name (CatName)
- Products: a unique identification number for each product (ProID), product name (ProName), price (PP) and category which classifies each product in a category (CatID).
- Visit: a unique identification number for each visit (VID), the patient who did this visit belongs (PID), doctor that saw the patient (DID), reason the patient visited (Reason), the product that the customer bought (ProID), the quantity (QO), total (TO) and date (date).

Let the dependency matrix that corresponds to this database be called *M*. This matrix will be made up of 22 columns (i.e., DID, FName, LName, EDOB, Prof, PID, PName, PNumber, CatID, CatName, ProID, ProName, PP, CatID, VID, DID, PID, Reason, ProID, QO, TO and date). As transaction *T1=Doctor* ("*1*", "*Kim*", "*Stewart*", "*1980-11-02*", "*Cardiologist*") is committed, a new entry in M will be created with the following attributes M[1][]={01, 01, 01, 01, 01, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00}. The first five columns will be manipulated by "01" because they will be blindly written by transaction T1. There is no need to look at any previously committed transaction to be able to write the values for T1. As for the rest of the columns,

Fig. 5 The recovery algorithm

```
Receive the sets of malicious and affected transactions (transaction IDs that need to be recovered)
Read the file into an array
For each transaction in the affected transaction set
        Retrieve the log file information for that transaction
        Update the database accordingly
For each transaction in the malicious transaction set
        Retrieve the log file information for that transaction
        Delete the transaction
```

they will be manipulated by "00" because transaction T1 wrote into the columns that belong to the Doctor table where as the other data items are left unmodified. Consider *T2= Patient* ("*1*", "*John*", "*001718668009*") which will also be committed as follows M[2][]={00, 00, 00, 00, 00, 01, 01, 01, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00}. If John visits the Cardiologist Dr Stewart then a new transaction will be committed in the DB, however this newly committed transaction will is dependent on the the . The visit is dependent on the presence of a Doctor and a Patient. Hence, *T3=Visit* ("*1*", "*1*", "*1*", "*Regular Checkup*", "", "", "", "*2014-12-03*") will result in a new entry in the matrix as follows M[3][]={00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 01, T2, T1, 01, 00, 00, 00, 01}

	DID	*FNAME*	*LNAME*	*EDOB*	*PROF*	*PID*	*PNAME*	*PNUM*	*CATID*	*CATNAME*	*PROID*	*PRONAME*	*PP*	*CATID*	*VID*	*DID*	*PID*	*REASON*	*PROID*	*QO*	*TO*	*DATE*
T1	*01*	*01*	*01*	*01*	*01*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*
T2	*00*	*00*	*00*	*00*	*00*	*01*	*01*	*01*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*
T3	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*00*	*01*	*T2*	*T1*	*01*	*00*	*00*	*00*	*01*

3.4 Complexity analysis

In this section we will discuss the best and worst case scenarios of our algorithm. The best-case scenario is when the intrusion detection system sends the malicious transaction soon after it is identified. In such a case, our proposed algorithm will not require scanning of any matrices or log as usually our algorithm starts searching the dependency matrix from the row following the row associated with the malicious transaction. Thus, our algorithm will be $O(1)$ in the best case.

We consider the worst-case scenario to be the case when a malicious transaction is detected to be in the first row of the dependency matrix and the checkpoint time is about to elapse. Hence, having the maximum number of rows to traverse. Moreover, the dependency between the transaction and the affected and malicious transactions is not discovered until the last data item. Thus, the algorithm will have to traverse the entire matrix, not skipping any row or column. Usually, our algorithm traverses the matrix row by row and for each row (transaction) the algorithm traverses each data item (column) to check if there is a dependency with a malicious transaction. If a column of any data item is found to be dependent on a malicious transaction, then the algorithm sets this transaction as affected and stops traversing the other data items. Thus, the worst case scenario is when the dependency is always found in the last data item. In this case, the algorithm will be of $O(nm^2)$ where n is the total number of committed transactions and m is the total number of data items.

4 Experimental results

We tested the performance of our algorithms by means of a simulated environment. The simulated environment develops a log that holds all the committed transactions, each transaction has a unique sequential ID. Our model requires the presence of a log file along with a dependency matrix to be able to perform each of the detection and recovery processes. The

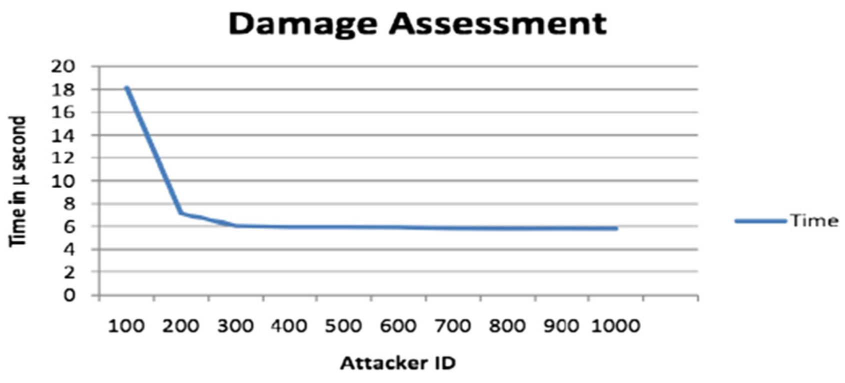


Fig. 6 Performance of the damage assessment algorithm based on different attacker IDs

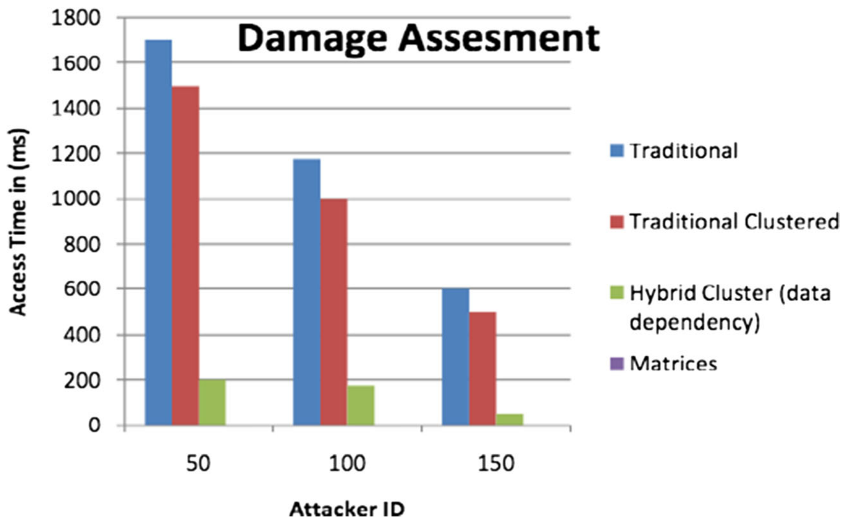


Fig. 7 Performance comparison of various damage assessment algorithms

simulated environment generates them before starting the execution of the algorithms. Transactions in our database are generated randomly.

We used the "Northwind Database" [28] in our experimental results. This database is provided as a template in the Microsoft Access Office. The database consists of 5000 attributes and 1081 records. The data was then converted to .sql format and added to MySQL. A connection is opened from our code to the SQL server that contains our Database. Consequently, our code contains SQL queries similar to recommit the query to recover from an attack. The server that was used in our stimulation is WampServer 2.0 with the following configuration: Apache Version 2.2.11, PHP Version 5.3.0 and MySQL Version 5.1.36. The simulated environment was developed on a system with an Intel® Core™ 2 Duo CPU P8600 at 2.40 GHz and running under approximately 2.39 GHz, with a 2 GB RAM.

4.1 Performance of the damage assessment algorithm

The total page I/O time calculation is performed by checking the total number of pages read during damage assessment and then multiplying this number with the time required to read each page. In order to calculate the total I/O time for traditional non-clustered algorithms, the counting procedure considers the bytes scanned from the starting point of attack till the end of the traditional log. For traditional clustered based on data dependency, the bytes scanned from the starting point of the attacking transaction till the end of the cluster. The parameters used are shown in Table 1.

As we can see from Fig. 6, the time needed for damage assessment decreases as the attacker ID increases. The attacker ID represents the transaction ID that has been affected. When the attacker ID is 100, the damage assessment algorithm has to traverse 981 rows to find every affected transaction. Unlike when the attacker ID is 1000 where the damage assessment algorithm has to traverse and check only 81 rows. The time decreases from around 18.13 to 5.8 μ s. As the algorithm has to traverse less number of rows, the time and effort needed for damage assessment will also decrease. The sooner the attack is detected, the better and the faster the damage assessment; this shows that our algorithm is capable of decreasing the time needed for damage assessment; and hence, less denial of service.

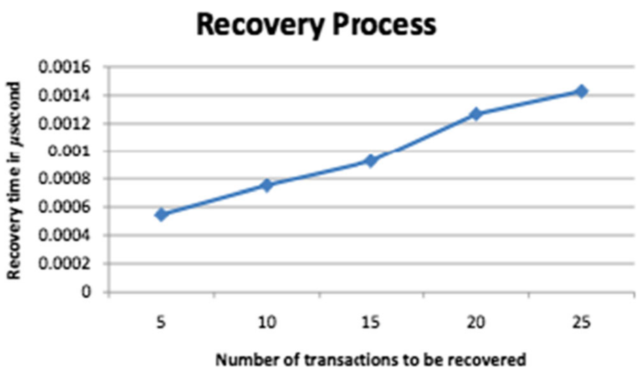


Fig. 8 Performance of recovery algorithm based on different numbers of affected transactions

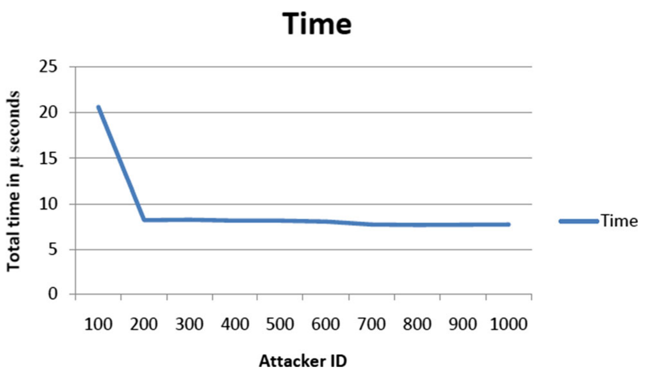


Fig. 9 Time taken for different attacker IDs to go through our recovery model

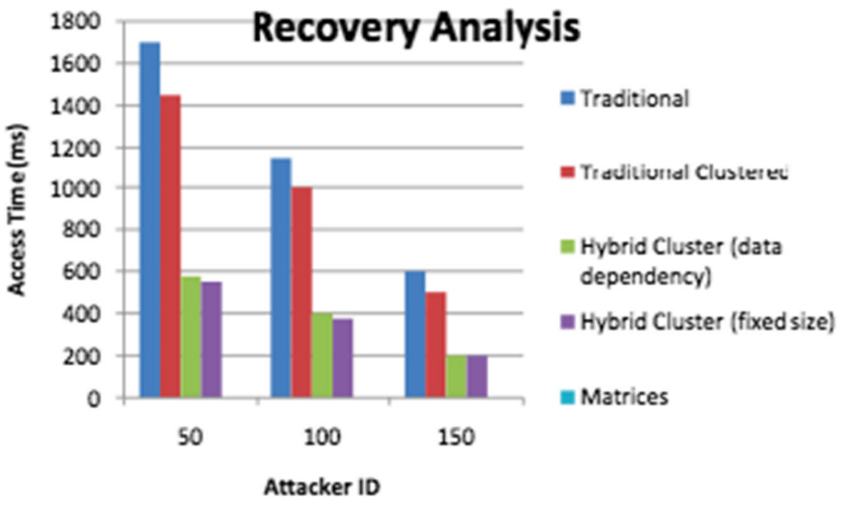


Fig. 10 Performance comparison of the various recovery algorithms

Figure 7 portrays a performance comparison between our algorithm and three other algorithms (traditional, traditional clustered and hybrid cluster algorithms). The four algorithms were tested in the same environment, where we have a database of 500 transactions with 5000 attributes, 2000 records, in which only 45 data items are accessed as a maximum in a transaction. The figure clearly portrays that our algorithm outperforms the rest of the algorithms. The fact that we are using matrices makes our algorithm work efficiently, especially that matrices are easily indexed. Unlike other algorithms, we only use one matrix that does not require any logical operations in the damage assessment phase. Moreover, our algorithm does not need to read the entire log file to cluster transactions according to dependency. The effort needed by other algorithms to read the log file is not needed in our algorithm. Hence, these characteristics improve our algorithm compared to previous algorithms.

The comparison analysis, shown in Fig. 7, confirms that our model accesses fewer page I/Os during damage assessment, thus improving performance. To construct the graph depicted above, the attacker ID was varied from between 50 to 150.

4.2 Performance of the recovery algorithm

Transactions are generated randomly. The average transaction time is 2 ms. After the damage assessment phase and after we have saved all of the malicious and affected transactions, we can move to the second phase, recovery. In this phase, we recover every transaction that has been affected by malicious transactions and delete every malicious transaction. Figure 8 shows the time taken by our algorithm to recover the set of malicious and affected transactions. As we can see when the number of transactions that need recovery increases, the time required for this recovery increases as well. Figure 9 shows the result for a database that constitutes of 500 transactions, 5000 columns, and 1081 rows. A transaction may access at most 45 columns.

Figure 9 portrays the time taken by our model to build the matrix, and to detect and recover from an attack. To obtain the results below we used a database composed of 5000 columns and 1081 transactions. It can be inferred from the figure below that the total time taken by our algorithm to detect and recovery from a malicious attack is less than the time taken to recover or detect using other models. In our model, the worst case scenario is when the malicious transaction is recovered after we have reached a check point. In such a case we need to rebuild the matrix and then start our assessment.

Figure 10 shows a comparison between our model and four other algorithms (traditional, traditional clustered, hybrid clustered with data dependency, and the hybrid cluster with fixed size). The results show that in all cases, our algorithm is more efficient than any other. The reason of this improvement is the use of matrices. Moreover, the log file during recovery in our algorithm is converted to look like an array which makes the indexing easier and faster. Rather than reading the entire log file whenever we want to find a transaction that needs recovery, we just index its position in the array. The time taken by other algorithms to read the log file is not needed in our algorithm. Our algorithm only requires the effort to convert the log file to an array and this is done only once at the beginning of the recovery algorithm.

5 Conclusion

After the failure of prevention mechanisms and the latency in the detection algorithm, we need a tool to recover from the malicious attack as soon as it is detected. In this paper, we presented a new approach for damage assessment and recovery that depends on matrices. In our approach, the dependency between transactions is saved in a matrix that will be formed as transactions are being committed. The matrix is then used to assess the affected transactions based on the set of malicious transactions that were provided by the IDS. Consequently, our recovery algorithm would treat all malicious and affected transactions. We tested our model and compared it with different approaches. The comparison results confirm that our approach is faster and more efficient than previously proposed models (traditional, traditional clustering, hybrid clustering according to data dependency and according to fixed size). As for future work, we will consider the space issue. Our algorithm requires the presence of a matrix along with a complementary structure to save transactions it depends on; this requires space that could be diminished.

References

1. Libicki M, Fellow S (1995) What is information warfare? United States Government Printing, United States
2. Chu J, Zihui G, Huber R, Ji P, Yates J, Yu Y (2012) ALERT-ID: Analyse Logs of the Network Element in Real Time for Intrusion Detection. Proceedings of the 15th international conference on Research in Attacks, Intrusions, and Defences
3. Kurra K, Panda B, Li W, Hu Y (2015) An Agent based approach to perform damage assessment and recovery efficiently after a cyber attack to ensure E-government database security. Proceedings of the 48th Hawaii International Conference on System Sciences
4. Hutchinsn W (2006) Information warfare and deception. Inf Sci 9: 213–223
5. Hua D, Xiaolin Q, Guineng Z, Ziyue L (2011) SQRM: an effective solution to suspicious users in database. DBKDA 2011: The Third International Conference on Advances in Databases, Knowledge, and Data Applications, St. Maarten, The Netherlands Antilles
6. Kim T, Wang X, Zeldovich N, Kaashoek M (2010) Intrusion recovery using selective re-execution. Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’10), pp. 89–104
7. Chakraborty A, Majumdar A, Sural S (2010) A column dependency based approach for static and dynamic recovery of databases from malicious transactions. Int J Inf Secur (ACM) 9(1): 51–67
8. Panda B, Zhou J (2003) Database damage assessment using a matrix based approach: An intrusion response system. Proceedings of the 7th International Database Engineering and Applications Symposium (IDEAS 2003), pp. 336–341
9. Panda B, Perrizo W Haraty RA (1994) Secure transaction management and query processing in multilevel secure database systems. Proceedings of the Symposium on Applied Computing. Phoenix, AZ, pp. 363–368
10. Ning P, Jajodia S (2004) Intrusion detection techniques. Internet Encycl 2:355–368
11. Megan B (1999) Information warfare: What and how? Carnegie Mellon School of Computer Science. Retrieved from http://www.cs.cmu.edu/~burnsm/InfoWarfare.html
12. Haeni R (1997) Information warfare an introduction. The George Washington University, Washington DC
13. Panda B, Haque KA (2002) Extended data dependency approach: a robust way of rebuilding database. Proceedings of the 2002 ACM Symposium on Applied Computing, pp. 445–452
14. Panda B, Gordano J (1998) Reconstructing the database after electronic attacks. Proceedings of the IFIP TC11 WG 11.3 Twelfth International Working Conference on Database Security XII: Status and Prospects
15. Ammann P, Jajodia S, Liu P (2002) Recovery from malicious transactions. IEEE Trans Knowl Data Eng 14(5):1167–1185
16. Fu G, Zhu H, Feng Y, Zhu Y, Shi J, Chen M (2008) Fine grained transaction log for data recovery in database system. Third Asia-Pacific Trusted Infrastructure Technologies Conference (IEEE), Washington, DC, USA
17. Lomet D, Vagena Z, Barga R (2006) Recovery from “Bad” user transactions. SIGMOD, June 27–29, Chicago, Illinois, USA
18. Ragothaman P, Panda B (2002) Analyzing transaction logs for effective damage assessment. Proceedings of the 16th Annual IFPI WG 11.3 Working Conference on Database and Application Security, pp. 121–134
19. Haraty RA, Zeitunlian A (2007) Damage assessment and recovery from malicious transactions using data dependency. ISESCO J Sci Technol 3(4):43–50
20. Zhou J, Panda B, Hu Y (2004) Succinct and fast accessible data structures for database damage assessment. In: Gosh R, Mohanty H (eds) Distributed computing and internet technology. Springer, Berlin, pp 111–119
21. Zhou J, Panda B (2005) A log independent distributed database damage assessment model. Proceedings of the 2005 I.E. Workshop on Information Assurance and Security, pp. 302–309
22. Xie M, Zhu H, Feng Y, Hu G (2008) Tracking and repairing damaged databases using before image table. Japan-China Joint Workshop on Frontier of Computer Science and Technology (IEEE), pp. 36–41
23. Liu P, Yu M (2011) Damage assessment and repair in attack resilient distributed database systems. Assoc Comput Mach (ACM) 33(1): 96–107
24. Lala C, Panda B (2001) Evaluating damage from cyber-attacks: a model and analysis. IEEE Trans Syst Man Cybern 31(4): 300–310
25. Ray I, McConnell R, Lunacek M, Kumar V (2004) Reducing damage assessment latency in survivable databases. In: Howard W, Lachlan M (eds) Key technologies for data management. Springer, Berlin, pp 106–111
26. Gray J, Reuter A (1993) Transaction processing concepts and techniques. Morgan Kaufmann, San Francisco
27. Bernstein P, Hadzilacos V, Goodman N (1987) Concurrency control and recovery in database systems. Addison-Wesley, Massachusetts
28. Microsoft Corporation – Northwind and Pubs Sample Databases for SQL Server 2000 (2015) http://www.microsoft.com/en-us/download/details.aspx?id=23654. Retrieved on 10 Mar 2015

Ramzi A. Haraty is an associate professor of Computer Science in the Department of Computer Science and Mathematics at the Lebanese American University in Beirut, Lebanon. He serves as the program administrator for the Middle East Program Initiative’s (MEPI) Leaders for Democracy Fellowship program. He is also the internship coordinator for MEPI’s Tomorrow’s Leader program. He received his B.S. and M.S. degrees in Computer Science from Minnesota State University - Mankato, Minnesota, and his Ph.D. in Computer Science from North Dakota State University - Fargo, North Dakota. His research interests include database management systems, artificial intelligence, and multilevel secure systems engineering. He has well over 110 books, book chapters, journal and conference paper publications. He supervised over

110 dissertations, theses and capstone projects. He is a member of the Association of Computing Machinery, Institute of Electronics, Information and Communication Engineers, and the International Society for Computers and Their Applications.

Mirna Zbib is currently a Quality Control Engineer at Murex, Lebanon. She received her B.S. degree in Computer Science from the Lebanese University, Beirut – Lebanon, and her M.S. degree in Computer Science from the Lebanese American University, Beirut - Lebanon. Her research interests include database management systems and information security.

Dr. Md. Mehedi Masud received his PhD in Computer Science from the University of Ottawa, Canada. He is an associate Professor at the Department of Computer Science, Taif University, KSA. His research interests include issues related to P2P and networked data management, query processing and optimization, eHealth, and information security. He has published several research papers at international journals and conferences. He has served as a member of the technical committees of several international conferences and workshops. He is on the editorial board of some journals including Journal of Internet and Information Systems (JIIS), Journal of Engineering and Computer Innovations, and Journal of Software (JWS). He served as a guest editor for Journal of Computer Science and Information Science (ComSIS).