A Matrix-based Damage Assessment and Recovery Algorithm

Ramzi A. Haraty and Mirna Zbib
Department of Computer Science and Mathematics
Lebanese American University
Beirut, Lebanon
Email: rharaty@lau.edu.lb

***Abstract*--With the advancement of Internet technology, securing information systems from electronic attacks has become a significant concern. With all the preventive methods, malicious users still find new ways to overcome the system security and access and modify sensitive information. To make the process of damage assessment and recovery fast and effective (not scanning the entire log), researchers have proposed different methods for segmenting the log file, and accordingly presented different damage assessment and recovery algorithms. In this work we present efficient damage assessment and recovery algorithms to recover from malicious transactions in a database based on the concept of the matrix. We also compare the various approaches and present the performance results.**

Keywords--damage assessment; recovery; malicious transactions; transaction dependency; and data dependency.

I. INTRODUCITON

Information warfare as described by Libicki and Fellow [1] is similar to discovering the nature of an elephant by a blind person. It refers to one of the most effective weapons that have been and are being used in today's wars [2]. Warfare started with the Agrarian revolution and then passed by the industrial revolution to reach what we call nowadays 'information warfare' [3].

For the purpose of this work, information warfare is the set of techniques taken to gain access to the information of an adversary while defending your own information. Some of the weapons that can be used in such a war are: logic bombs, computer viruses, information collection, information manipulation, information degradation and denial of service [4-6]. To be able to defend data, and to be able to exploit the data of others, one should have a full understanding of how things work.

During the past two decades, Internet usage has been increasing rapidly. This increase has always been accompanied by information sharing, which is a key element for the success and productivity of an organization. The importance of this process is to preserve the reliability of information. Securing information is made on 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 [7]. 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.

The complexity and efficiency of the recovery process is our main interest in this work. In some cases the adversary's intentions are not only to insert malicious transactions but also to cause denial of service. Sometimes the size of the log file might increase tremendously before discovering that an attack has occurred. Thus, this will require more time to assess and recover from the malicious transaction and its effects. This increase in recovery time would lead to denial of service. 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 [8-9].

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 efficient than previously proposed algorithms.

II. LITERATURE REVIEW

When dealing with electronic data and transactions, it is hard to identify which user is malicious and which is authenticated. The system treats all the users the same and accepts their transactions. For example in [10], every user is considered a malicious user and his transactions will not take actual action in the database until a certain period of time elapses. After this, the behavior can be classified as either malicious or non-malicious. Accordingly, and based on this behavior, transactions can be committed or aborted.

As soon as an attack is detected by an intrusion detection system, it should be directly recovered. 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 on one another in one segment. So the log is divided into multiple segments. In Panda and Haque 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 [13] 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 [14] 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 in a periodic way 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.

Traditional methods suggest scanning the log file from the point of the attack until the end of the file to undo and redo the affected transactions. Panda in [15] suggested a method which is fusing the malicious transactions to reduce the I/O time. The aim of this method is to minimize the time to insure best results without getting any other consequences.

Panda and Tripathy [16] suggested the use of 'Coldstar' semantics in their algorithms where the database becomes unavailable for new transactions. They also present an algorithm using the 'Warmstart' semantics where the database use continues with some services but stops with others.

In [17], 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 [8] 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 [18] 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 [19] and requires additional structures to recover when working on distributed databases.

Xie, Zhou, Feng and Hu [20] suggested the use of a before-image (BI) table to keep track of all deleted transactions and to help in analyzing 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.

Chakraborty, Majumdar and Sural [9] presented a column-dependency approach. The advantage of this approach is that it takes less time than the traditional approach to recover from an attack. This approach has showed that the percentage of inconsistencies after re-execution increases with the increase of malicious transactions.

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 [21]. 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 and Panda [22] 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 contains 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.

Fu et al. [23] 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; however, the services will face a major degradation.

Ray et al. [24] 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 in [1] as well as the algorithm suggested in [22]. The aim of this paper was to reduce the damage assessment latency so damage spreading will not occur. The disadvantage found in [1] 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 [22], 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.

Lomet et al [25] 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.

III. THE PROPOSED APPROACH

The trigger for our proposed algorithm is the set of malicious transactions that it will receive from an intrusion detection system (IDS). Our algorithm is responsible of assessing the committed transactions and consequently classifying them into clean or affected transactions; then it deletes the malicious transactions and recovers the affected ones.

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 the use of check points [27].

The check point is a database event used in order to ensure 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 that the committed data is clean (or else the IDS would have detected the set of malicious transactions). The chosen time interval should not be too long so that the log file will not tremendously grow in size, yet long enough so that we will not have to go to previous check points to recover. Check points are beneficial in our proposed algorithm to reduce the space and reading time.

After a check point has been reached (i.e., after the agreed upon time elapses), 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 [11] 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.

As the transactions are being executed, and after their commitment 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. 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 would mean that dataitem1 is updated according to the last modified value of dataitem1 by transaction 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. That is, in our example, 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 figures 1 and 2.

	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

Fig. 1 The dependency matrix.

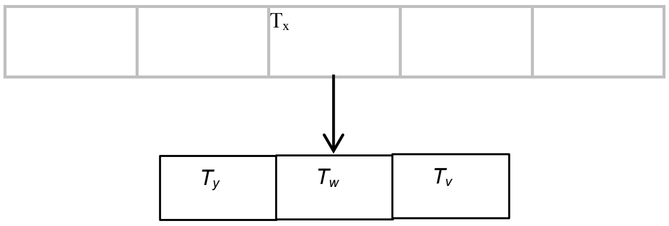

Fig. 2 The complementary array.

A. 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 affected transactions that our algorithm will detect. The previously discussed 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 and 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 done to see how the data have been affected. If the entry is a '0' or '1', then this means that this data item have 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 checks 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 this current transaction is dependent on the transaction, with the transaction ID present in that entry. To illustrate this, we 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 have 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 belong to it, if it does then we add T_i to the set of affected transaction. Then, our algorithm will not continue searching the other data items and skip to the next 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 transaction 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 done 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 damage assessment algorithm is summarized in figure 3.

```
Receive the set of malicious transactions
Select the minimum transaction ID among the malicious transactions
For every transaction in the matrix starting from the minimum malicious 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 belong 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 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 belong 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.

B. 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 do the proper update. The algorithm is illustrated in figure 4.

```
Receive the sets of malicious and affected transactions
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
```

Fig. 4 The recovery algorithm.

C. An Example

Consider a database for a company that contains information about the following: Employee, Customer, Category, Product and Order. Also, consider the following insert transactions:

T_1 = Employee (‘1’, ‘Kim’, ‘Stewart’, ‘1980-11-02’,’Sales’, ‘$2000’); [for employee ID, first name, last name, date of birth, job, salary]

T_2 = Category (‘1’, ‘Beverages’); [for category ID, category name]

T_3 = Product (‘1’, ‘Pepsi’, ‘$1’, ‘1’); [for product ID, product name, price, category ID]

T_4 = Customer (‘1’, ‘X’, ‘Beirut’); [for customer ID, customer name, address]

T_5 = Order (‘1’, ‘1’, ‘1’, ‘1’, ‘300’, ‘$300’, ‘2012-12-01’); [for order ID, customer ID, employee ID, product ID, quantity, total, date]

Let the dependency matrix that corresponds to this database be called M. This matrix, which is depicted in figure 5, is composed of 22 columns (i.e., EID, FName, LName, EDOB, EJ, ESalary, CID, CName, CAddress, CatID, CatName, PID, PName, PP, CatID, OID, CID, EID, PID, QO, TO and date). As transaction T_1, commits, a new entry in M will be created with the following attributes M[1][ ] = {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}. The first six columns will have entries of ‘1’ because they are blindly written by transaction T_1. The rest of the columns will be ‘0’ because they were left unmodified. Unlike transaction T_3 which depends on previously committed transactions. Each product belongs to a category and in this case product ‘1’ belongs to the category that was previously committed in transaction T_2. T_3 writes the first three attributes without looking at anything that was previously written, but the fourth attribute needs to look at the previous transaction. Therefore, the row for T_3 will be M[3][] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, $-T_3$, 0, 0, 0, 0, 0, 0, 0}. Transaction T_5 depends on transactions T_1, T_3 and T_4. The row corresponding to T_5 is represented as follows: M[5][] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,$-T_5$, $-T_5$, $-T_5$, 1, $-T_5$, 1}. OID, QO and date were blindly written by T_5, unlike CID, EID, PID, and TO that needed an additional complementary structure. The OID, QO and date columns in transaction T_5 will be represented by ‘1’. To manipulate M[5][16], we need the complementary structure to save the transaction that helped in getting the value of this field. Since T_5 was for customer ‘X’ then this shows that there is dependency between transactions T_5 and T_4. Similarly, we will retrieve and save the dependency related to M[5][17] and M[5][18] ensuring not to repeat dependent transactions. Hence, this dependency will be reflected in Comp[T_5] = {T_4, T_1, T_3}, which is depicted in figure 6.

	EID	FName	LName	EDOB	EJ	ESalary	CID	CName	CAddress	CatID	CatName	PID	PName	PP	CatID	OID	CID	EID	PID	QO	TO	Date
T_1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T_2	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
T_3	0	0	0	0	0	0	0	0	0	0	0	1	1	1	$-T_3$	0	0	0	0	0	0	0
T_4	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
T_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	$-T_5$	$-T_5$	$-T_5$	1	$-T_5$	1

Fig. 5 The dependency matrix M.

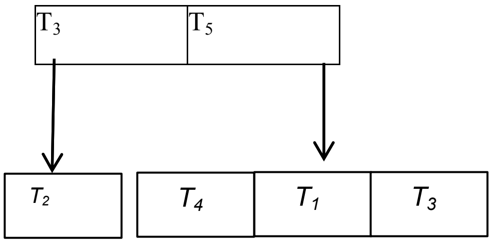

Fig. 6 The complementary array.

Consider the case were the intrusion detection system will provide our model with ID T_2. Hence, the algorithm will start searching the matrix starting from T_3 rather than starting from T_1. It will check every column in T_3 and skip every column that contains a ‘0’ or ‘1’. Since the CatID column contains a negative ID, we will search the complementary array for T_3.

The complementary array points to T_2 which is malicious (see figure 6); hence, T_3 will be added to the affected transactions set and we will skip to T_4. The same thing will be repeated until the end of the matrix. The set of malicious and affected transactions will subsequently be sent to the recovery algorithm. During recovery, T_2 will be deleted since it is malicious and the set of affected transactions will be updated.

IV. CONCLUSION

In this paper, we presented a new approach for recovery that depends on matrices. The dependency between transactions is saved in a matrix that will be formed as transactions are being committed. 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 complimentary 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: United States Government Printing.

[2] Hutchinsn, W. (2006). Information warfare and deception. Information Science, 9, 213 - 223.

[3] Haeni, R. (1997). Information warfare an introduction. Washington DC: The George Washington University.

[4] 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). 89-104.

[5] Megan B. (1999). Information warfare: What and how? Carnegie Mellon School of Computer Science. Retrieved from http://www.cs.cmu.edu/~burnsm/InfoWarfare.html.

[6] Haraty, R. A. (1999). C2 secure database management systems - a comparative study. Proceedings of the 1999 Symposium on Applied Computing, 216-220.

[7] Ning, P., & Jajodia S. (2004). Intrusion detection techniques. The Internet Encyclopedia, 2, 355 - 368.

[8] Haraty, R., & Zeitunlian, A. (2007). Damage assessment and recovery from malicious transactions using data dependency for defensive information warfare. ISESCO Science and Technology Vision, 3(4), 43 – 50.

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

[10] 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.

[11] 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), 336 – 341.

[12] Panda, B., & Haque, K.A. (2002). Extended data dependency approach: A robust way of rebuilding database. Proceedings of the 2002 ACM Symposium on Applied Computing, 445 – 452.

[13] 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.

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

[15] Panda, B., & Yalamanchili, R. (2001). Transaction fusion in the wake of information Warfare. Proceedings of the 2000 ACM Symposium on Applied Computing, 242– 247.

[16] Panda, B., & Tripathy, S. (2000). Data dependency based logging for defensive information warfare. Proceedings of the 2000 ACM Symposium on Applied Computing, 361-365.

[17] 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, 121-134

[18] Zhou, J., Panda, B., & Hu, Y. (2004). Succinct and fast accessible data structures for database damage assessment. In R. Gosh, & H. Mohanty, (Eds), Distributed Computing and Internet Technology (pp. 111-119). Berlin, Germany: Springer.

[19] Zhou J., & Panda B. (2005). A log independent distributed database damage assessment model. Proceedings of the 2005 IEEE Workshop on Information Assurance and Security, 302-309.

[20] 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), 36 – 41.

[21] Liu, P., &Yu, M. (2011). Damage assessment and repair in attack resilient distributed database systems. Association for Computing Machinery (ACM), 33(1), 96 - 107.

[22] Lala, C., & Panda, B (2001). Evaluating damage from cyber-attacks: a model and analysis. IEEE Transactions on Systems, Man, and Cybernetics, volume 31, issue 4, 300-310.

[23] 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.

[24] Ray, I., McConnell, R., Lunacek, M., & Kumar, V. (2004). Reducing damage assessment latency in survivable databases. In W. Howard & M. Lachlan (Eds.), Key technologies for data management (pp. 106-111). Heidelberg: Springer Berlin.

[25] Lomet, D., Vagena, Z., & Barga, R. (2006). Recovery from “Bad” user transactions. SIGMOD, June 27-29, Chicago, Illinois, USA.

[26] Gray, J., & Reuter, A. (1993). Transaction processing concepts and techniques. San Francisco: Morgan Kaufmann.

[27] Bernstein, P., Hadzilacos, V., & Goodman, N. (1987). Concurrency control and recovery in database systems. Addison-Wesley, Massachusetts, USA.

[28] Microsoft Corporation – Northwind and Pubs Sample Databases for SQL Server 2000 (2014). http://www.microsoft.com/en-us/download/details.aspx?id=23654. Retrieved on April 21 16, 2014.

ACKNOWLEDGEMENTS

This work was sponsored by the Lebanese American University.

BIOGRAPHY

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 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.