LEBANESE AMERICAN UNIVERSITY

INFORMATION WARFARE RECOVERY - FIGHTING BACK THROUGH THE MATRIX

By

MIRNA ZBIB

A thesis
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

School of Arts and Sciences

June 2012

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

Chartered in the State of New York

Lebanese American University
School of Arts and Sciences
Thesis Proposal Form

Name of Student: ________Mirna Zbib___________ I.D.#: _200904220___________

Department: Computer Science and Mathematics

On 12/10/11, has presented a Thesis proposal entitled:

Information Warfare Recovery – Fighting Back through the Matrix

in the presence of the Committee members and Thesis Advisor:

Ramzi A. Haraty__[redacted] 17/10/2011
(Name, signature, and date of the Thesis Advisor)

Samer Habre___[redacted] 17/10/2011
(Name, signature, and date of Committee Member)

Azzam Moura[redacted] 17/10/2011
(Name, signature, and date of Committee Member)

Comments/Remarks/Conditions to Proposal:

____

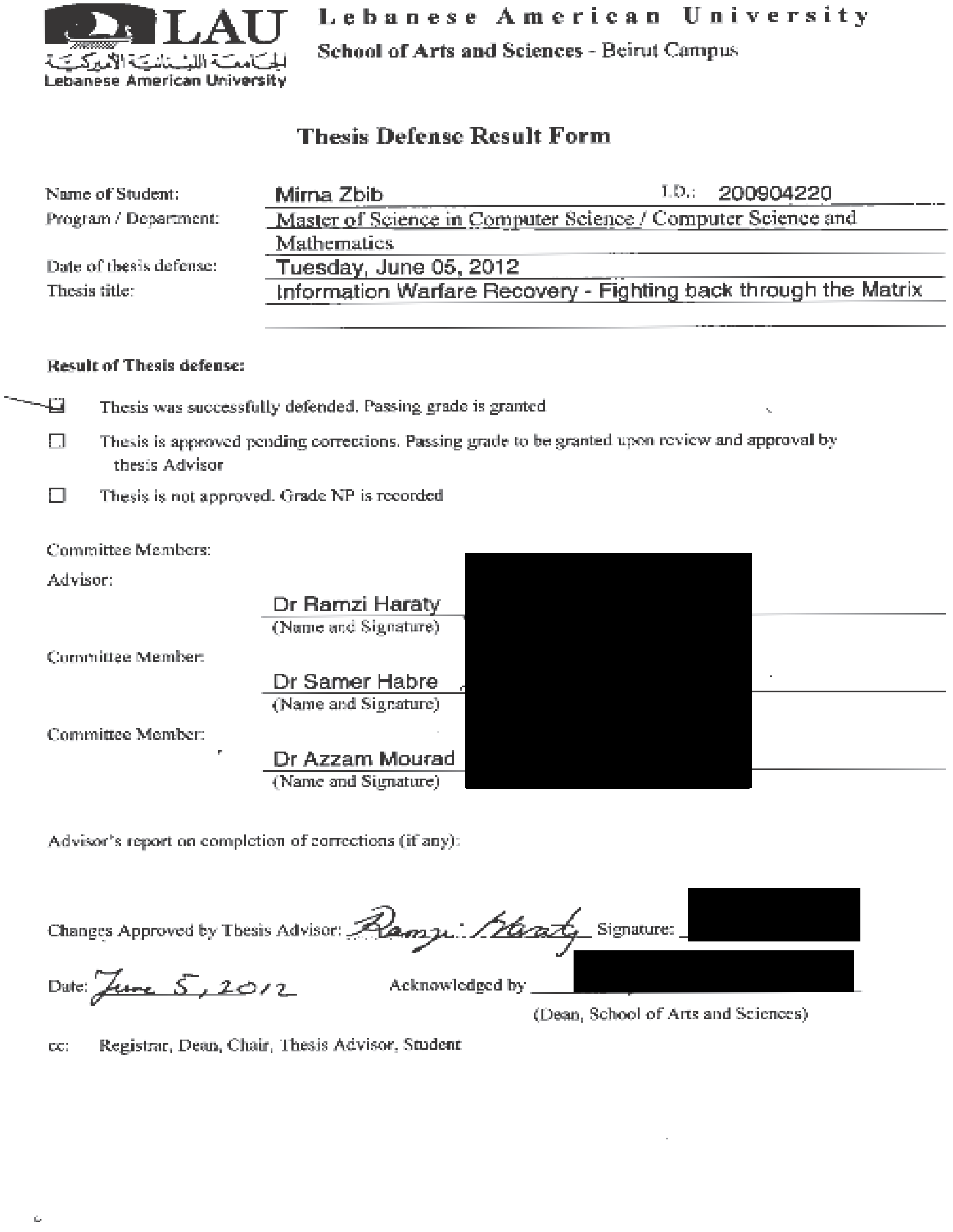
____

____

____

Date: 18/10/2011 Acknowledged by NM [redacted]
(Dean - School of Arts and Sciences)

cc: Department Chair
Thesis Advisor
Student
School Dean



LAU
Lebanese American University

Lebanese American University
School of Arts and Sciences - Beirut Campus

Thesis Defense Result Form

Name of Student: Mirna Zbib I.D.: 200904220

Program / Department: Master of Science in Computer Science / Computer Science and Mathematics

Date of thesis defense: Tuesday, June 05, 2012

Thesis title: Information Warfare Recovery - Fighting back through the Matrix

Result of Thesis defense:

☒ Thesis was successfully defended. Passing grade is granted

□ Thesis is approved pending corrections. Passing grade to be granted upon review and approval by thesis Advisor

□ Thesis is not approved. Grade NP is recorded

Committee Members:

Advisor:

Dr Ramzi Haraty [redacted]
(Name and Signature)

Committee Member:

Dr Samer Habre
(Name and Signature)

Committee Member:

Dr Azzam Mourad
(Name and Signature)

Advisor's report on completion of corrections (if any):

Changes Approved by Thesis Advisor: Ramzi Haraty Signature: [redacted]

Date: June 5, 2012 Acknowledged by [redacted]
(Dean, School of Arts and Sciences)

cc: Registrar, Dean, Chair, Thesis Advisor, Student

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

LEBANESE AMERICAN UNIVERSITY

School of Arts and Sciences

Thesis Approval Form

Student Name: Mirna Zbib I.D. #: 200904220

Thesis Title

Information Warfare Recovery – Fighting Back through the Matrix

Program : Master of Science in Computer Science

Department : Computer Science and Mathematics

School : School of Arts and Sciences

Approved by :

Thesis Advisor: _Dr Ramzi Haraty_________ Signature [redacted]

Member : _Dr Samer Habre_________ Signature

Member : _Dr Azzam Mourad_______ Signature

Date : June 5, 2012

THESIS COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY NON-EXCLUSIVE DISTRIBUTION LICENSE

Name: Mirna Zbib

Signature: [redacted]

Date: June 05, 2012

PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.
This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Mirna Zbib

Signature [redacted]

Date: June 05, 2012

ACKNOWLEDGMENTS

The help and assistance of many people made this work possible. I would like to start with special thanks to my advisor, Dr Ramzi Haraty, for his guidance, support and help. Thanks to Dr Samer Haber and Dr Azzam Mourad for being on my committee.

Finally, I would like to thank everybody who helped in this work especially during the proofreading.

Dedication Page

I would like to dedicate this work to every person who contributed and made it possible. Special thanks for my family who supported at all times and provided me with the appropriate environment to be able to accomplish my work and my nephews and niece who were my incentive and kept me upbeat, my family, aunts and uncles who made it possible for me to reach this day. I would also like to thank some of my friends who helped me in accomplishing this work.

And finally, I wish to dedicate this work to a special and unique person, my granddad, who was a constant motivation and inspirational force.

INFORMATION WARFARE RECOVERY - FIGHTING BACK THROUGH THE MATRIX

MIRNA ZBIB

Abstract

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

Keywords: Damage Assessment, Recovery, Malicious Transaction, Transaction Dependency, Data Dependency.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Overview

Prevention, detection and recovery are three important phases in any “live” system. The prevention phase consists of the following: authentication, authorization, access control, firewalls, and data encryption. Still, malicious users and hackers’ manage to overcome these security measures and attack the system.

In case preventive methods fail, we use the intrusion detection systems, which are used for monitoring malicious transactions. When intrusion detection systems were built they were meant to work in harmony with the preventive methods. Mainly, detection methods can be split into two categories: anomaly detection (statistical) and misuse detection (Ning and Jajodia, 2004). Anomaly detection works by analyzing the behavior of the user (i.e., whether it is usual or unusual). On the other hand, misuse detection works by comparing actions to a set of rules or vulnerabilities that are already saved in the system. None of these detection systems ensure that an attack will be immediately detected. Hence, damage could spread affecting other new coming “clean” transactions as well.

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 success and productivity of an organization. Millions of computers worldwide are connected to each other and are sharing information. 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. Our aim, after this corruption and after detecting that something malicious has occurred, is to remove and clean the corruption with its effects. In such a case a recovery algorithm must be used.

When dealing with electronic data and electronic transactions it is harder to identify which user is malicious and which is authenticated. The system treats all the users the same and accept their transactions. Some intrusion detection systems, such as SQMR (Hua, Xiaolin, Guineng, and Ziyue, 2011), are classifying users according to their behavior. Yet, in some cases these methods fail to classify allowing malicious users and suspecting and preventing normal users from doing their job. For example in (Hua, Xiaolin, Guineng, and Ziyue, 2011), 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 or some action occurs after. After this, we would be able to classify the previous behavior or user as either malicious or non-malicious. Accordingly, we can either commit the transactions or abort them. For this reason, whenever a malicious transaction is identified we need to go back and trace it to be able to remove all of its effects.

1.2 The Problem

As soon as an attack is detected by an intrusion detection system, it should be directly recovered. Unfortunately, the detection of an attack is not done immediately, and this can affect other coming transactions. Hence, 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 (Panda and Zhou, 2003) and data dependency (Panda and Haque, 2002). For example, consider a malicious transaction T_i that committed, and then a transaction T_j read a data item written by T_i. The result would be having T_j as an affected transaction. Therefore, upon recovery we do not only need to delete T_i, but also to recover T_j.

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 a denial of service. Sometimes the size of the log file might increase tremendously before discovering that an attack has occurred. Consequently, 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. Thus, 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 tackled is what information should be saved in the log file as we prevent excess I/O. For this purpose, some researchers (Ammann, Jajodia, and Liu , 2002), (Haraty and Zeitunlian, 2007); (Chakraborty et al., 2010) and (Panda and Zhou, 2003), have proposed using auxiliary structures for keeping track of dependencies while others proposed using matrices.

1.3 Information Warfare

Information Warfare the term that is affecting everybody's life is one of the hot topics nowadays. This term have moved us to a new period of time where the whole world is changing. Information warfare did not start as a computing term only, but it was also referred to on the psychological level as well. In the 1980's information warfare was a military term that developed and became a way of living in 1991, especially with the Gulf war (Hutchinsn, 2006). Information warfare refers to one of the most affective weapons that have been and are being used in today's wars. Warfare started with the Agrarian revolution and then passed by the industrial revolution to reach to what we call now the information warfare (Haeni, 1997).

"What is Information Warfare?" a question with no exact answer due to the different and many dimensions of this term. Libicki and Fellow, (1995) compared defining information warfare to discovering the nature of an elephant by a blind person. If one touches the tail he will think it is a rope, while if he touches the leg he will think it is a tree. That is to say that information warfare is a huge thing and has different dimensions. It would be defined according to which dimension the person will tackle.

In this work we will only consider one of the many aspects of information warfare. For our purpose, 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:

- Logic bombs: These can be Trojan horses that are either present with in a normal code or are independent programs. Such a weapon can be used by a country that wants to gain access over another country's computers by implementing a Trojan horse inside software that they are planning to sell it to them.

- Computer viruses: These are code fragments that insert themselves into a program to modify it. For example, such weapons could be used in phones and might cause phone system failure at certain stages.
- Information collection: The more the information we have about our adversary the better. This information will aware us of what are the plans of the other party (Megan, 1999).
- Information manipulation: The change in the information to give a wrong picture to the adversary (Megan, 1999).
- Information degradation and denial: This is used to prevent the adversary from getting complete or correct information. This technique is done either by disturbance or overloading the system so it becomes busy (Megan, 1999).

In order to have the upper hand on the information two different aspects should be understood: the information-in-warfare and the information based process (Ryan, 1998). To be able to defend your data and to be able to exploit the data of others, one should have a full understanding of how things work. One should not only know how to attack or defend himself, but also how to gain and exploit. If we attack a system and do not gain access to it or do not exploit it, then the attack is useless.

1.4 Scope of the Work

In this thesis, we present a recovery model that works with matrices. Our approach suggests that we keep 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 retrieved from the matrix. The aim of this work is for fast 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.

The main contribution in our model is the use of a matrix that requires less time to be scanned than when scanning graphs, like the traditional cluster algorithms. In addition, the use of bits in our algorithm requires less time to be dealt with than any other normal numbers. Moreover, we use only one matrix rather than using two matrices for each of the read and write operations; hence this saves time and space as well. 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. All of this contributes to make our model faster than previously proposed algorithm.

1.5 Definitions

The following definitions will be used throughout the thesis:

Definition 1: A *write* operation *wi[x]* of a transaction *Ti* is dependent on a read *ri[y]* operation of *Ti*, if *wi[x]* is computed using the value obtained from *ri[y]* (Panda and Tripathy, 2000).

Definition 2: A *blind write* is when a transaction T_i writes data item *x* without reading the previous values of *x* (Zheng, Qin and Sun, 2007).

Definition 3: A write operation *wi[x]* of a transaction is dependent on a set of data items *I*, if $x = f(I)$; i.e, the values of data items in *I* are used in calculating the new value of *x*. If $x \neq I$, the operation is called a blind write. In this case if the previous value of *x* (before this write operation) is damaged and none of the data items in *I* are damaged, then the value of *x* will be refreshed after this write operation (Panda and Tripathy, 2000).

Definition 4: If *X* is *totally ordered* under ≤, then the following statements hold for all *a*, *b* and *c* in *X:*

If $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetry)

If $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity)

$a \leq b$ or $b \leq a$ (totality)

Definition 5: If a *read(x)*/*write(x)* is to be executed in a *strict execution,* it will be delayed until all *write(x)* operations are either committed or aborted (Gray and Reuter, 1993).

Definition 6: A transaction management mechanism guarantees *rigorous*ness if the following two conditions hold (Kim et al., 2010):

1. it guarantees strictness, and
2. No data item may be written until the transaction which previously read either commits or abort

1.6 Organization of the Report

The rest of this report is organized as follows: in chapter 2 we will present related work and give an overview about what previous research has lead to in this area. Then a new algorithm will be presented in chapter 3. While in chapter 4 we present the experimental results; this will be followed by a conclusion and the future work in chapter 5.

CHAPTER TWO

LITERATURE REVIEW

Malicious transaction detection and recovery have taken a lot of effort from researchers. Many models were proposed for this purpose, some of which cluster the log files while others use sub-clusters, graphs or even matrices. Damage recovery from malicious attacks is divided into two main categories, transactional dependency and data dependency. In each area, several models have been suggested. In this section we provide an overview on these models.

2.1 Coldstar Model

Panda and Tripathy (2000), unlike previous traditional approaches, this approach is known as a transaction dependency approach. They assume that an attack is partial and never complete. The recovery is split into two problems: the damage assessment and the damage repair process. The authors suggest the use of ‘Coldstar’ semantics in one of their algorithms where the database becomes unavailable for new transactions, and they also suggest another algorithm using the ‘Warmstart’ semantics where the database use continues with some services but stops with others. In this approach, only affected and malicious transactions are dealt with. There is no need for any change in non-malicious or non-affected transactions. Malicious transactions are undone; affected transactions are re-done; non-malicious and non-affected transactions are neither undone nor redone. To assess the recovery performance, the average repair time and the average response time are taken into consideration.

Since some transactions are not affected by any malicious or affected transaction, they can be ignored and not worked with. For this purpose, Ammann, Jajodia, and Liu (2002) suggested, segmenting the log files so the work would only be done on the part of the log that is affected. This will ensure accelerating the recovery process. 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. The algorithm presented in this paper accelerates the recovery process since not all of the massive log file will be scanned to reach the operations wanted. However, deleted transactions cannot be retrieved, so maliciously deleted transaction might be skipped. The advantage of this approach is that we can previously determine which items are affected, and accordingly we will be skipping a large part of the log.

2.2 Traditional Methods

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 his paper (Panda and Yalamanchili, 2001)suggested a new method which is fusing the malicious transactions to reduce the I/O time. The aim of this model is to minimize the time to get the best results without getting any other consequences. The assumptions in this model are the following:

1. Strictly serializable scheduler,

2. No blind writes, and
3. No purging of log files.

Undoing the affected transactions occurs in the reverse order of how they happened, while the redoing of these transactions occur in the same order. Compensated-for transactions are transactions that need to be undone, while compensating-for transactions are transactions that are redone. For the sake of saving time, instead of accessing the logs and searching for the transaction, the fused transaction method is used to re-execute the fused transactions. This method has two advantages: first, only one commit is needed, which helps in case a data item has been read or updated, and the second one is that dependent transactions will be fused. From the point of attack and onwards, we start by detecting the malicious transactions and their fused dependents, and any transaction that is unaffected will be ignored.

The recovery procedure for fused transactions requires data structures such as M_ID, F_ID, M_flag and F_flag. *Fuse* a function used to combine operations from two transactions into a fused transaction. In this function to confirm that an operation should be skipped, a check is done on the data item, otherwise the operation will be added to the fused transaction. The schedule is scanned from the attacking point till the end to group the malicious transactions using the *fuse* function followed by generating IDs. The method also keeps track of which transaction is the first among the fused transactions.

2.3 Using Graphs for Recovery

The spread of the Internet and its applications made the presence of assessment and recovery very crucial. Two different approaches exist for damage assessment:

transaction dependency and data dependency. Data dependency is more efficient and less time consuming than transaction dependency. Panda and Haque (2002) used the data dependency approach where operations of transactions are undone and redone. This approach has some inefficacy because unexecuted segments are not present in the history. The four conditions that must be present to ensure successful recovery are:

1. Use of a strictly serializable scheduler,
2. Use of the same order of the transactions in the history and the log,
3. Use of a rigorous 2PL, and
4. No modification of the log file by the user.

Each read/write operation in the transaction has a block number; this number shows dependency between operations. Panda suggested the use of a Directed Damage Demonstration Graph (DDDG), which only presents the affected data items. Two different shapes represent the nodes: either a circle or a square (continuous or dotted line). A clean item is represented by a circle, while a corrupted one is represented by a square depending on either an actual-write or overlooked-write. Arrows in this graph show dependency. Arrows of two forms (single and double line arrows) are used depending on whether the read is actual or predicate. Three data structures are created in this algorithm: damageDataList, readDataList_T_bNum and totalDataList_T. The readDataLists are checked whenever a write operation is present. In this algorithm, the detection of a malicious transaction requires a check on all the transactions that are in the schedule, followed by a check on all committed transactions. Any committed transaction that is detected to be affected will be added to the damageDataList. After identifying and creating the three lists, the undo_redo procedure is called upon, which takes the totalDataList and the damageDataList. Panda used the logical operation in addition to the before and after images for recovery.

Panda and Gordano (1998) proposed two data dependency algorithms are presented; 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 on the algorithm level as well. For example, when both the 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. This is unlike when the damage assessment is done independently, where the results are provided to the recovery algorithm in which we have the advantage of shorter periods of time; and hence, less denial of service. In the second algorithm, the system will be going through denial of service only during the damage assessment, but when the recovery algorithm starts to take action, the unaffected transactions will be unlocked for the user, and he will be able to work normally unless he wants to access affected data.

In both approaches the damage assessment works using directed graphs, where the nodes would represent a data item. When the intrusion detection system reports the occurrence of a malicious transaction, a node for each data item will be created. This node will be represented as a square. This graph helps in mapping how the damage has spread.

2.4 Clusters and Sub clusters

Haraty and Zeitunlian (2007) proposed a model that uses data dependency that depends on clustering. Each cluster is sub-clustered using one of the two approaches: number of transactions or space occupied. Before going any further with the model, the assumptions are as follows:

- The detection of malicious transactions by external detection techniques,
- The use of rigorous serializable scheduler,
- No purging of Log files, and
- The use sequential transaction ID.

Two structures of different functionalities are used in the hybrid sub-clustering algorithm: Transaction Sub Cluster (TSC) list and the Sub Cluster Data (SCD) List. The first one is used to identify the affected sub clusters; it stores the transactions ID. After identifying the sub cluster from the first structure, the latter is used to identify the affected data items.

The hybrid sub-Clustering algorithm based on fixed number of transactions limits every sub cluster so that it can only contain a specific number of transactions. Data items could belong to the predicate, or the operation could belong to the statement. In case of the predicate, the data item could either belong to a cluster's sub cluster (SCD) or not. If it does not belong, then a new cluster and sub cluster ID should be given, followed by an update on the TSC and SCD. In case it belongs to the cluster's sub cluster (SCD), then a check on the number of transaction should be done to either get the cluster and sub cluster ID or to initiate a new sub cluster since every sub cluster could contain only a fixed number of transactions. Otherwise, the data item can either belong or not to the cluster's sub-cluster. If the data item does not belong to the cluster's sub-cluster, then a new cluster and sub cluster IDs should be initialized; otherwise, the sub-cluster's size should be taken into consideration to either initiate a new sub-cluster or to take the cluster's and sub-cluster's IDs. After checking the cluster and sub-cluster, the operation should be checked to confim if it is an actual read or an overlook read before it is recorded into the log.

The hybrid clustering algorithm is based on fixed size and has a limited size for the sub-cluster. The limitation of this algorithm is the size of the sub-cluster; i.e., all dependent operations are put in the same sub-cluster, until a point is reached where adding another operation will let the sub-cluster go over the required space, so it is inserted in a new sub-cluster.

The damage assessment algorithm has two structures: Damaged_DI and Damaged_PB. A check on whether the predicate is read is performed to assess whether it belongs to the Damaged_DI. Consequently, the block in which this item is present is damaged as well; hence, the block should be added to the Damaged_PB. Then, a check on whether there is an actual or overlooked read is executed: if the data item is damaged but operation does not belong to a malicious transaction, the block should be checked. In case it is listed as damaged, it should be added to the damaged_PB. In case of a write operation, if there is a malicious transaction where the data item is not in the damaged_DI, then it should be added; however, if the transaction is not malicious and the block is not affected, the data item is affected then the data item should be removed from the damaged_DI list.

Fu et al. (2008) started by explaining the concept of “Extended Read Operations” and moved on to propose the concept of *Fine Grained Transaction Log (FGTL)*. Log files should be able to help find the dependencies among operations and transactions to be able to recover easily. So, not only the main sub-clause of an SQL-statement should be recorded in the log file, but also every sub-clause is of similar importance since these sub-clauses can help in spreading the damage. According to the author, the FGTL should record all writes in addition to every extended read. All committed transactions should be logged. This log is represented as a database table that cannot be modified by any user. Some read operations are just a sub query of a main SQL statement so logging these read operations requires more effort. To solve this problem the “Divide and Combination Algorithm” is suggested to be used to log these read operations. This algorithm works by dividing the “Select” statements into several statements to be able to log the read information. The result of this algorithm is exactly the same as what the original statement would result. In case an Update Insert Delete statement is present, the same would be done for the “Select” statement, and then the last step would be updating the information.

After the “Divide and Combination Algorithm”, the Fine Grained Transaction Log is proposed. The FGTL saves each write and read operation executed by the user. The generated log is used for damage assessment; therefore, it should be protected from any unwanted modification. This is done using the DB Monitor, which is responsible for capturing and checking each operation.

The problem with this technique is that when the association degree of the transaction increases, the throughput of the FGTL decreases sharply. As the degree is increasing, more reads are required; hence, the throughput is decreasing. This technique may cause degradation in services but still it preserves integrity of the log.

2.5 Before Images

Xie et al. (2008) suggested a new way for damage assessment and recovery. Xie suggested the use of a before image table to keep track of all deleted transactions and to help in analyzing potential reads. The before image is a data object created in the database. In this approach the inter-transaction dependency is taken into consideration, which relates even deleted transactions. Xie classified affected transactions into two types: transactions affected by maliciously inserted data and transactions affected by maliciously deleting data. The problem with the second type of affected transactions is that they usually cannot be tracked through log histories, which can cause many inconsistencies, so Xie tried to overcome this problem in his approach. Xie dealt with two types of operations: insert and delete. He considered an update operation as two operations: delete operation followed by an insert. The database is represented as D(V, B) where V represents the set of data items, and B is the set of deleted data items. Any two transactions are considered dependent if there is an actual or potential read.

Before Image (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 to use a time widow to delete data items and restrict the size of the BI tables. In addition to the BI table, two other structures are maintained the x.ins_tran and x.del_tran that represent the transactions that insert or delete a data item. After the execution of a read transaction, the ID of the depending transaction is added to the DS. Then, this is stored in a TanDepTab. Executing a subquery on a table is accompanied by executing the same subquery on the BI table.

Two steps should be executed to completely repair the database: first identify and then erase the effects of malicious transactions. Identifying the affected transactions is made easier because of the interdependency graph, which is represented as the TranDepTab. The repair algorithm is an on-the-fly algorithm, which saves the problem of stopping server responses. Repairing requires deleting inserted data items and restoring deleted data items by the transaction.

This algorithm requires more performance overhead due to the BI table and the queries done on them. The results proved that this algorithm can function well with a reasonable degradation in performance.

2.6 Column Dependency

Chakraborty et al. (2010), presented column-dependency approach. The method is decomposed into two phases the compensation phase and the re-execution phase. The compensation phase is responsible for compensating the malicious transactions, where as the re-execution phase is responsible for re-executing the other committed transactions for consistency purposes. Two approaches are applied: one for static recover and the other for online recovery. The recovery time of this approach is

$t_p=t_a+t_c+t_e$ where:

- t_a: the damage assessment time
- $t_{c:}$ compensation time
- t_e: re-execution time

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. This is similar to the percentage of inconsistencies after compensation, but still the percentage after re-execution is more than that after compensation. As for the on-line recovery performance, it is linked to the arrival rate of transactions (i.e., as the arrival rate increases the recovery processes gets slower). In general, this approach removes all the effects of malicious transactions.

The use of a *Local DAR Manager* and a *Local Dar Executer* on each site was suggested by Liu and Yu (2011). The *Local DAR Executer* starts by identifying all affected sub transactions and continues to clean these sub transactions. 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*.

According to this algorithm, each site has its own log file. Log files have five attributes for each record:

- Type of the record: read, write, abort, commit, prepare and end
- Application id
- Transaction id
- Coordinator id

- Subordinates id

The throughput on each site is approximately the same as that of centralized database systems. The advantages of this approach are that no deadlocks are caused and there is no waiting time for this algorithm to start. Transaction dependencies and attacking rate are one of the most important factors that affect the communication costs in this algorithm. It is an on-the-fly algorithm, which implies transaction processing will not be stopped during the repair process. The damage assessment and recovery process are done in parallel so less time will be required by the algorithm, and the damage assessment and repair are transparent to the user.

2.7 Matrices in Recovery

Panda and Zhou (2003) proposed two damage assessment approaches one with transaction dependency and the other with data dependency. The aim of these approaches is to find a fast and accurate model. In his model, Panda uses two bit-matrices. The assumptions that Panda made for his model are:

1. Rigorous history,
2. Logs cannot be changed by users, and
3. Transaction dependency are not changed during recovery.

Four data structures are used in his model: Read_Matrix, Write_Matrix, Damaged_Data_Vector and Damaged_Transaction_List. Transactions are ordered in the matrix in the same sequence as in the log. Directed acyclic graph is maintained to track the dependencies. In each of the Read_Matrix and Write_Matix, the rows are for transactions and the columns are for items. Zero means that this data item is not in this transaction, otherwise we have a one.

Once a malicious transaction is detected it will be added to the Damaged_Transaction_List. The Damaged_Data_Vector should be OR-ed with the Write_Matrix. Any resulting 1 indicates that the data item is affected by the malicious transaction. As for the read, the Read_matrix is OR-ed with Damaged_Data_Vector. If there is any '1' in the vector, then the read operation was affected by the malicious transaction. Affected transactions should be refreshed; the process should be repeated until no more ones. With these steps all the affected transactions are located. So this should be followed by a recovery model.

Using Panda's algorithm for damage assessment the database logs will not be accessed, which will help in reducing the time and reducing the risk of denial-of-service. Two other structures the Compact_Read_list and the Compact_Write_List are used in this algorithm. This list contains sublists of the data items that have been read/written by this transaction, where the transaction ID is saved.

As for the data dependency model, it is unlike the transaction dependency model where the reading of one affected data item makes all written data items by that transaction affected. In this model, we have two structures - the Data_Dependency_Write_Matrix and the Data_Dependency_Read_Matrix. The first column of these matrices contains the transaction ID and the second one contains the operation number. By scanning the log file, these two structures will be constructed. After detecting the first malicious transaction, it will be added to the

Damages_Transaction_List and then it will be followed by OR-ing with the transactions to see what the affected transactions are.

Ragothaman and Panda (2002) has suggested segmenting log files into clusters, but still this will not solve the problem. We cannot control the size of the dependent transactions and hence the clusters may grow in size. To solve this problem, Haraty and Zeitunlian (2007) proposed the use of clusters and sub clusters. Data inside a cluster are records that have some data dependency, where as data in the same sub cluster could be for one of the following two reasons: number of data items or space occupied. This approach groups data according to exact dependency and it helps to recover faster. Instead of scanning the whole log, and even instead of scanning the whole cluster we will only be scanning a small sub cluster to recover. Two additional lists are used in this algorithm the *Transaction Sub Cluster List* and the *Sub Cluster Data List*. The first is used to store the Transaction ID along with the corresponding Sub Cluster, where as the latter is used to store the sub cluster ID along with the transaction ID and the data item.

Zhou et al. (2004) proposed a similar model for distributed databases. The model proposed works on transaction dependency in order to recover from malicious attacks. This work extends the work of Zhou and Panda (2005) and requires additional structures to recover when working on distributed databases. The model in this paper requires the use of pre-developed structures, read, and write matrices. Transactions are represented as rows in these matrices and they are updated whenever a transaction is committed. The advantage of this method is that it does not require the use of the log file which will decrease the access time. The author also assumes that there is a timestamp for each transaction. This timestamp will be compared against timestamps of different transactions from different sites and hence recovery would be followed.

Lala and Panda (2001) also presented a work that depends on matrices in which he also suggested another algorithm. The new algorithm suggested by Panda was for the damage assessment. In this algorithm, Panda suggested to cluster the transactions so he does not have to search transactions that have no effect. In this model, an additional column was added to each of the matrices that Panda proposed. The additional column will contain that cluster ID to which this transaction belongs. In addition to this, two other lists will be used: *Dependent_Cluster_List (DCL)* and *Ancestor_Cluster_List (ACL)*. The dependency between the clusters will be saved in these two lists. If a transaction belongs to more than one cluster, then that set of clusters will be merged together. The merging of these clusters might lead to a worst case scenario that is having the whole transactions in one log. The main goal of the use of clusters was to reduce the time of damage assessment and recovery and to be able to perform them in parallel, but in the worst case scenario this might be unreachable. To overcome this Panda proposed that he will restrict the number of transactions per cluster.

2.8 Analysis

Ray et al. (2004) performed analysis on existing algorithms along with a suggestion of new techniques was proposed. The complexity analysis was done to check the complexity of Ammann, Jajodia, and Liu (2002) as well as the algorithm suggested in (Lala and Panda, 2001). The aim of this paper was reducing the damage assessment latency so damage spreading will not occur. The disadvantage found in Amman's algorithm was that the log file of this algorithm is huge and therefore it will take time scanning the part of the log after the malicious transaction. As for Lala and Panda's algorithm, it showed a worst-case running time of O (vlogv +s), such that *v* represents the number of affected transactions and *s* represents the sum of sizes of the transaction records. The model that this paper suggested uses dependency graph, which will be represented by a list structure and then continues by eliminating the queues and using depth-first search. The repair schedule produced by this algorithm will be proportional to the sum of lengths of affected transactions.

2.9 Fuzzy dependency

Zuo and Panda (2004) started by explaining the fuzzy dependency between transactions. "For two sets of attributes *X* and *Y* of a relation R, *Y* is fuzzily dependent on *X* if and only if for every value a_i in the domain of *X*, a_i belongs to *D(X)*, there is an uniquely determined subset S_i' in the domain of *Y*, S_i*'* subset in *D(Y)*, such that a tuple T_i in a relation instance of R with value a_i for *X* should have a value b_i belong S_i*'* for *Y*". The authors proposed that there are three uses of the fuzzy dependency:

1- Used to specify constraints,

2- Used in intrusion detection, and

3- Used to reduce the denial of service.

The suggested recovery algorithm consists mainly of the *"Fuzzy Value Generator"* that interacts with the database and the "Fuzzy dependency storage". The advantage provided by this algorithm is that the log file will not be traversed as a whole bulk, but rather this model uses the fuzzy relations that are saved. Still the disadvantage of this method is the lack of accuracy.

CHAPTER THREE

THE MODEL

3.1 Overview

In this chapter, we present a new detection and recovery model that is guaranteed to give reliable and trusted results. The model presented in this work will be triggered once it receives a malicious or a set of malicious transactions. Then, it will run to assess and find all the good but affected transactions. Finally, the model will end by deleting the malicious transactions and recovering from the affected ones.

3.2 Assumptions

In our algorithm, we assume that an external intrusion detection system will provide our model with the malicious transaction or the set of malicious transactions. From that point on, it will be the responsibility of our model to find and clean every affected transaction.

Serial execution occurs when for every $i < j$, every operation in T_i occurs before the operations of T_j. Serial execution ensure a serializable history. In such a history, every transaction is assumed to be correct as it would be depending on the committed transactions only. Hence, serializability provides correctness (Gray and Reuter, 1993). An update to any transaction in the system will be represented in our model as if it is a new insert transaction (i.e. a new row in the matrix will be added). For our model, we assume we have a rigorous serializable history. A sequential log file is also maintained in which only committed transactions are saved. This log file cannot be accessed by the users at all times and it will be used during recovery. Our algorithm requires the use of check points. After a certain period of time, data would become obsolete and then we can assume that no malicious transaction exists in that set of transactions. Hence, instead of making the log file grow in size tremendously with clean data, we will use check points. Check points will diminish the size of the log file and consequently reduce the space and the reading time. Check points should be chosen carefully so that they will not be too long to cause the log file to grow tremendously and require a lot of time for recovering, but yet long enough so that we will not have to go to previous check points in case a malicious transaction was detected. But if we faced a case were the intrusion detection system was too late to detect the malicious transaction in which a check point was already established, we will take advantages of the sequential log file. Using the log file we would be able to rebuild the matrix and execute our model as would be done if the check point did not exist.

Another significant characteristic of our model is the use of the dependency matrix. The dependency matrix will be treated as the log file, i.e. it will be flushed at every check point because we would have guaranteed that the data became obsolete and clean. Hence, the matrix and the log file will use the same check point. If we take the worst case scenario and consider that a malicious transaction has occurred before the check point that we have created, then we will use the log file to rebuild the matrix after which the process will continue as any malicious transaction detection. The dependency matrix that we have will be a sequential matrix, such that for every $i < j$, $T_i < T_j$ and T_i

can never depend on T_j. The matrix will only save the committed transactions. The importance of this matrix is in the detection process. It will be used to discover the dependency among transactions and hence see which ones where affected and which ones were clean. Panda and Zhou (2003) used dependency matrices as well. But in their model they had two different matrices one for the write operations and another for the read operations. In addition, they used logical operations between the matrices to discover dependencies. However, in our model only one matrix will be used and it will show the dependency without any logical operations.

The attacker may have many intentions; one of these intentions is to cause denial of service whether it is direct or indirect, denial of service can be reached directly by sending many malicious transactions at the same time, whereas the indirect method occurs when the effect takes place at a later stage such as during the detection and recovery time. In detection, as the log file increases in size the time required for recovery would increase as well. In During detection and recovery time, the system will be down and the users will not be able to access it. Hence, this could be one indirect way of causing denial of service. So the attacker might be aware that injecting only one malicious transaction can require a lot of effort and shut the system for a long period of time and may lead to severe damage.

3.3 The Matrix

We have discussed the usage of the log file and the matrix so far; in this section we will discuss the structure of this matrix. We assume that this matrix is built dynamically along with the execution of every transaction. Only committed transactions are inserted into our matrix. The matrix will be a two-dimensional array, such that the columns represent all the data items present in our database and the rows represent all the transactions that occurred. Each data item will either be blindly written by the transaction, left unmodified by the transaction, modified according to one previously committed transaction, or modified according to a set of previously committed transaction. Each case will be uniquely represented in our matrix. For every transaction, each data item will have a value depending on the operations that the transaction has gone through, this is represented as follows:

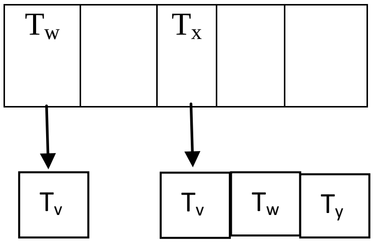
- 00: if the data item is unmodified by that transaction
- 01: if the data item is blindly written by that transaction, data from previous transactions is not needed.
- A positive transaction Id: if dataitem1 of transaction x with transaction Id T_x is identified in the matrix with entry T_y such that $y < x$, this would mean that for dataitem1 in T_x we have modified its value according to dataitem1 that was last modified by transaction T_y.
- A negative transaction Id: this means that this data item has been modified according to previous data items from different transactions; this will be shown in more details in the following section.

Consider a transaction T_x such that to modify dataitem1 of this transaction we need to read dataitem4 of transaction T_y, dataitem3 of transaction $T_{w,}$ and dataitem1 for from transaction T_v, where y, w and $v < x$. In such a 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 an additional three-dimensional 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 at T_y, T_w and T_v, similar to what is presented in tables 1 and 2.

Table 3. 1 Dependency Matrix

	1	2	3	….	30
T_v	01	00	T_a		01
T_w	00	01	-T_v		00
….					
T_y	T_k	-T_r	00		-T_d
T_x	-T_x	00	00		00

Table 3. 2 Second Complementary Array



3.4 Detection Algorithm

Our proposed detection algorithm will use an additional data structure. This data structure is a one-dimensional array that will save all the affected transactions that our algorithm will detect. It works with the two matrices that are being built as the transactions are being executed and the set of malicious transactions that will be provided by an external intrusion detection system. The set of malicious transactions provided from the external intrusion detection system might contain many transactions that are not ordered. Since our log file is sequential and due to the characteristics of our algorithm, the minimum transaction Id between the set of malicious transactions is required to start working from that point. Since we are sure that we will never find a transaction T_j such that $j < i$ and T_j depends on T_i. To reduce the access time and consequently reduce the detection time we will traverse the matrix starting from the malicious transaction with the least transaction Id and hence skip any transaction that we know it will not be affected.

The matrix will be traversed row by row starting from the transaction that directly follows the malicious transaction with the least transaction Id. Then, for each data item in that transaction a check will be done to see how the data have been reached. If the entry is a ‘00’ or ‘01’, then our algorithm will skip that column and check the following columns. If the entry contains a positive transaction id, a check will be done to see if this Id is the same as one of those ids in the malicious set. If it belongs to the malicious set, then the transaction Id will be added to the affected transactions structure and we will directly move to the transaction after it without looking at the rest of its columns. If one of the data items of a certain transaction is affected it would be enough for us to classify that transaction as affected transaction and try to update it again during the recovery algorithm. If that transaction Id does not belong to the malicious set, this shows that that transaction have not been directly affected by the malicious transaction. Still, this is not enough to classify the transaction as a clean transaction. It might be affected indirectly and thus the effect would be an affected transaction. For this reason, we should also search among the affected transactions and see if this transaction belongs to the affected transaction, then it should also be updated.

After presenting what happens in the three above listed cases now we are still left with the fourth case which is having a negative transaction Id. A negative transaction Id shows that the transaction we are currently looking at has been affected by at least one previously committed transaction. After detecting a negative transaction Id, we will go to the second array and retrieve content of the entry that has the same key as the row that we were searching. 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 we already have in the malicious transaction set. Once we detect that any of these Ids are similar to a malicious Id, we will then add that transaction to the affected set and move to the following row. If nothing matched, we will apply the test against the affected transactions. Similarly, if the transaction was found to depend on an affected transaction; this means the data is dirty and should be added to the affected transactions set. For example, if the set of malicious transactions is $\{ T_r, T_s, T_t\}$ and the detection algorithm is running and it reached transaction T_x, such that data item a for this transaction has the value $-T_x$, then we will refer to the second matrix M_2 and try to retrieve the content of $M_2[T_x]$. If $M_2[T_x]=\{ T_u, T_r, T_z\}$, the algorithm reaches T_r and inserts T_x into the set of affected transactions. In the other case, if the set of malicious transactions are $\{ T_v, T_s\}$ and the set of affected transactions are $\{ T_t, T_u\}$, while $M_2[T_x]=\{ T_u, T_r, T_z\}$, the algorithm starts searching to check if the transaction T_x is affected or not. It will start by checking the malicious transactions against the transactions in $M_2[T_x]$. In this example, it will not be found. Hence, the algorithm will move on and check the affected transactions. T_u would be matched with the set of transactions in $M_2[T_x]$ and the set of affected transactions. Consequently, the set of affected transactions would become $\{ T_t, T_u,T_x\}$.

Table 3.3 Detection Algorithm Based on Matrices

```
1. Receive the set of malicious transactions
2. Select the minimum transaction ID among the malicious transactions
3. Receive the set of malicious transactions
4. Select the minimum transaction ID among the malicious transactions
5. For every transaction in the matrix starting from the minimum malicious ID to the end of
   the matrix
   5.1. For each data item
        5.1.1.If (entry == 00) then
              5.1.1.1.  Move to the next row
        5.1.2.Else if (entry == 01) then
              5.1.2.1.  Move to the next row
        5.1.3.Else if (entry>0 && entry belong Malicious transactions)
              5.1.3.1.  Add the Current transaction to the set of affected transactions
              5.1.3.2.  Move to the next row
        5.1.4.Else if (entry>0 && entry does not belong Malicious transactions)
              5.1.4.1.  For every transaction in the affected transactions set
                    5.1.4.1.1.  If (entry==$T_{affected}$)
                          5.1.4.1.1.1. Add entry to affected transactions set
                          5.1.4.1.1.2. Move to the next row
                    5.1.4.1.2.  Else if (entry<0)
                          5.1.4.1.2.1. Search secondArray for key==entry
                          5.1.4.1.2.2. For each element in secondArray[entry]
                                5.1.4.1.2.2.1.  If(element belong Malicious transactions)
                                      5.1.4.1.2.2.1.1.  Add the current transaction to the set of
                                                        affected transactions
                                      5.1.4.1.2.2.1.2.  Move to the next row
                                5.1.4.1.2.2.2.  Else if(element belong to the set of affected
                                                transactions)
                                      5.1.4.1.2.2.2.1.  Add the current transaction to the set of
                                                        affected transactions
                                      5.1.4.1.2.2.2.2.  Move to the next row
```

The above algorithm represents the logic of the detection algorithm presented for our model. Step 1 represents the information that the algorithm needs in order to function, i.e. the set of malicious transactions that an external Intrusion Detection System has provided. This information is the trigger for our algorithm; it will enable us of evaluating all the other data that we have. Step 2 is implemented for evaluating the ID, from which we will start our algorithm, in order to start assessing the damage that the set of malicious transactions have caused. At the same time, the algorithm is trying to minimize the effort that the algorithm needs, but through maximizing the effect. Due to the fact that our matrix and log file are sequential, we select the minimum transaction ID to start the assessment from that point. By selecting the minimum transaction ID, we guarantee to cover all the malicious transactions by moving sequentially among them. In addition, the fact that the matrices are sequential makes us certain that we will never find a transaction before this transaction that depends on it. From step 3 onwards, the actual detection of the affected transaction begins. Step 3.1.1 checks if the data item of that transaction has been modified, while step 3.1.2 checks if the data item is a blind write. Steps 3.1.3 and 3.1.4 checks whether the value in the matrix is positive or negative. Accordingly, steps 3.1.3.1, 3.1.3.2 and 3.1.4.1 checks if it belongs to the malicious or affected transactions. If the value is negative, then the second array should be search which is represented in step 3.1.4.1.2.1. Similarly, this matrix should be compared with the malicious and affected set in order to evaluate if the transaction is clean or not.

3.5 Example on the Detection Process

For example, consider a database for a company that contains information about the following:

- Employees: a unique identification number for each employee (EID), first name (FName), last name (LName), date of birth (EDOB), job (EJ) and salary (ESalary).
- Customers: a unique identification number for each customer (CID), customer or company name (CName) and the customer address (Caddress).

- Categories: a unique identification number for each category (CatID) and category name (CatName).
- Products: a unique identification number for each product (PID), product name (PName), price (PP) and category which classifies each product in a category (CatID).
- Order: a unique identification number for each order (OID), the customer to whom this order belongs (CID), employee that took this order (EID), the product that the customer bought (PID), the quantity (QO), total (TO) and date (date).

Consider the following transactions in the database stated above:

T_1 = Employee (‘1’, ‘Kim’, ‘Stewart’, ‘1980-11-02’,’Sales’, ‘$2000’);

T_2 = Categories (‘1’, ‘Beverages’);

T_3 = Products (‘1’, ‘Pepsi’, ‘$1’, ‘1’);

T_4 = Categories (‘2’, ‘Dairy Products);

T_5 = Products (‘2’, ‘Cheese, ‘$4’, ‘2’);

T_6 = Employee (‘2’, ‘John’, ‘Adam’, ‘1987-21-03’, ‘Sales’, ‘$1500’);

T_7 = Customer (‘1’, ‘X’, ‘Beirut’);

T_8 = Customer (‘2’, ‘Y’, ‘Beirut’);

T_9 = Order (‘1’, ‘1’, ‘1’, ‘1’, ‘300’, ‘$300’, ‘2012-12-01’);

T_{10} = Order (‘2’, ‘2’, ‘1’, ‘1’, ‘250’, ‘$250’, ‘2012-27-02’);

T_{11} = Order (‘3’, ‘1’, ‘2’, ‘2’, ‘50’, ‘$200’, ‘2012-12-04’);

T_{12} = Order (‘4’, ‘1’, ‘2’, ‘1’, ‘150’, ‘$150’, ‘2012-12-04’);

T_{13} = Order (‘5’, ‘2’, ‘2’, ‘1’, ‘300’, ‘$300’, ‘2012-01-05’);

T_{14} = Order (‘6’, ‘2’, ‘2’, ‘1’, ‘70’, ‘$70’, ‘2012-12-05’);

Let the dependency matrix that corresponds to this database be called *M*. This matrix will be made up of 21 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 is committed, a new entry in *M* will be created with the following attributes *M[1][ ] = {01, 01, 01, 01, 01, 01, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00}*. The first six columns will be manipulated by ‘01’ because they will be blindly written by transaction T_1. There is no need to look at any previously committed transaction to be able to write the values for T_1. As for the rest of the columns, they will be manipulated by ‘00’ because transaction T_1 wrote into the columns that belong to the *Employee* table where as the other data items are left unmodified. Similarly, after the commitment of transaction T_2 a new row will be added to the matrix after T_1. This is to preserve the property that transactions are sequential in the matrix and that there is no $i < j$ such that $T_j < T_i$. Transaction T_2 will write in the categories table thus adding values to *CatID* and *CatName*. The row belonging to transaction T_2 will look like: *M[2][] = { 00, 00, 00, 00, 00, 00, 00, 00, 00, 01, 01, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, ‘00’}*. Transaction T_2 does not need to read values from previous transactions be able to write its values. Unlike transaction T_3 which depends on preciously committed transactions. Each product belongs to a category and in this case product one 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 written before, but the fourth attribute needs to look at the previous transaction. *M[3][] = { 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00,01,01,01, -*T_3*, 00, 00, 00, 00, 00, 00, 00}*. Similarly, transactions T_4, T_5, T_6, T_7 and T_8 will write their values. Transaction T_9 is one of the transactions that depend on other transactions to write its values. For example, transaction T_9 is depends on transactions T_1, T_3 and T_7. If any of T_1, T_3 or T_7 is malicious then T_9 is affected. The row corresponding to -T_9 is represented as follows: *M[9][] = {00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 01,-*T_9*, -*$T_{9,}$*-*T_9*, 01, -*T_9*, 01}*. Three data items are independent and blindly written by transaction T_9 which are *OID, QO* and *date*. These three data items did not need information from previous transactions unlike *CID, EID, PID,* and *TO*. To save the information related to these four data items an additional complementary array, denoted by *Comp,* is needed. In case of T_9, *Comp[*T_9*]* will save every transaction that T_9 depends on. Hence, *Comp[*T_9*] = {* T_7, T_1, T_3*}*. The building of *Comp[*T_9*]* was executed as follows: to manipulate *M[9][16]* we needed the complementary structure to save the transaction that helped in getting the value of this field. Since T_9 was for customer ‘X’ then this shows that there is dependency between transactions T_9 and T_7. Due to this dependency, T_7 was added to the complementary array. In case the intrusion detection system detects later on that T_7 is malicious then we can figure out that T_9 was affected. *M[9][17]* refers to the employee that was responsible of this order. Consequently, if this employee was maliciously entered, then transaction T_9 is affected and we need to recover it. Thus, we need to save this dependency which will be reflected in *Comp[*T_9*]*. Up to this stage *Comp[*T_9*] = {* T_7, T_1*}*. As for *M[9][18]* it refers to the product that was order by customer ‘X’ in this transaction. If the product in

T_3 was malicious, then T_9 is affected. Hence, this dependency is essential to be saved which will lead to $Comp[T_9] = \{ T_7, T_1, T_3\}$. To calculate the total cost of this transaction we need information about the cost of the product that is present in transaction T_3 and the quantity that the customer will buy which is in transaction T_9 itself. Thus, transaction T_9 depends on transaction T_3 which is already reflected in the complementary matrix, so no need to add T_3 again. The quantity and date of the order are the characteristics of transaction T_9 and they do not need any previous information to be able to write the values. Hence, *M[9][19]* and *M[9][21]* are manipulated by '01'. The commitment of transaction T_9 and the manipulation of its corresponding row in the matrix can never happen before the commitment of transactions T_7, T_1 and T_3. Moreover, the row corresponding to transaction T_9 can never be created before the creation of each of the rows corresponding to each of T_7, T_1 and T_3. The same procedure should be followed for each of the transactions after T_9.

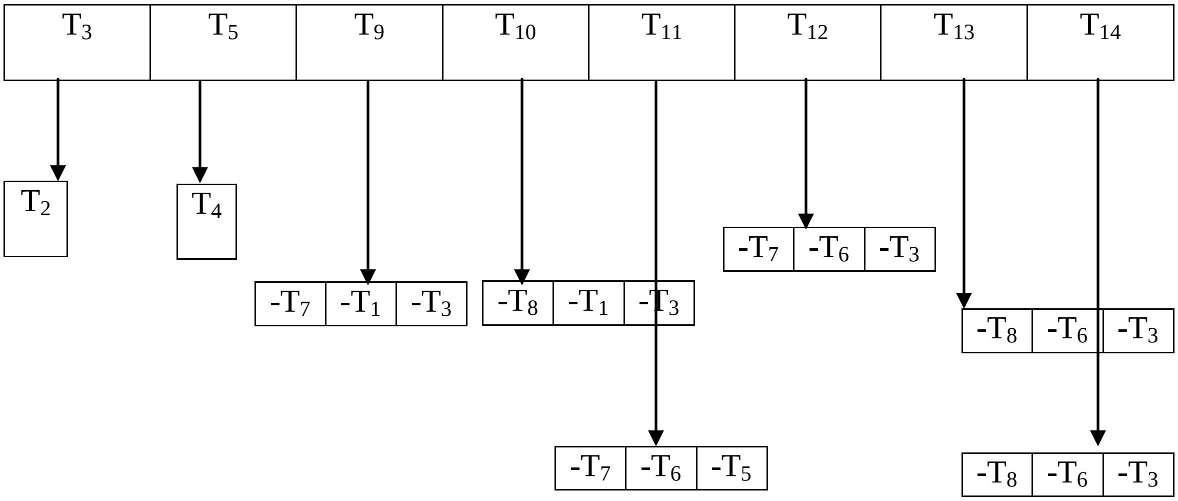
The corresponding matrix and complementary array are represented in the tables below:

Table 3.4 Dependency matrix for the presented example

	EID	FName	LName	EDOB	EJ	ESalary	CID	CName	CAddress	CatID	CatName	PID	PName	PP	CatID	OID	CID	EID	PID	QO	TO	Date
T_1	01	01	01	01	01	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
T_2	00	00	00	00	00	00	00	00	00	01	01	00	00	00	00	00	00	00	00	00	00	00
T_3	00	00	00	00	00	00	00	00	00	00	00	01	01	01	-T_3	00	00	00	00	00	00	00

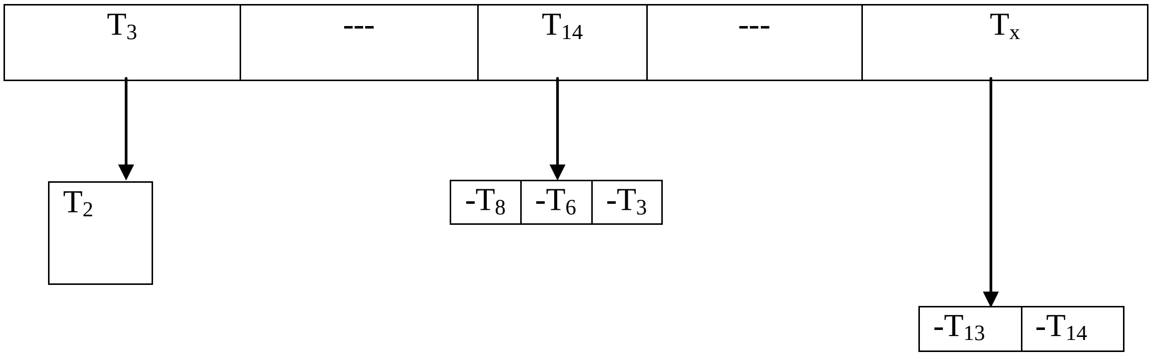
T_4	00	00	00	00	00	00	00	00	00	01	01	00	00	00	00	00	00	00	00	00	00	00
T_5	00	00	00	00	00	00	00	00	00	00	00	01	01	01	$-T_5$	00	00	00	00	00	00	00
T_6	01	01	01	01	01	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
T_7	00	00	00	00	00	00	01	01	01	00	00	00	00	00	00	00	00	00	00	00	00	00
T_8	00	00	00	00	00	00	01	01	01	00	00	00	00	00	00	00	00	00	00	00	00	00
T_9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	$-T_9$	$-T_9$	$-T_9$	01	$-T_9$	01
T_{10}	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	$-T_{10}$	$-T_{10}$	$-T_{10}$	01	$-T_{10}$	01
T_{11}	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	$-T_{11}$	$-T_{11}$	$-T_{11}$	01	$-T_{11}$	01
T_{12}	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	$-T_{12}$	$-T_{12}$	$-T_{12}$	01	$-T_{12}$	01
T_{13}	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	$-T_{13}$	$-T_{13}$	$-T_{13}$	01	$-T_{13}$	01
T_{14}	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	$-T_{14}$	$-T_{14}$	$-T_{14}$	01	$-T_{14}$	01

Table 3.5 Complementary matrix for the presented example



It might be thought that rather than using the additional array we can write in the matrix itself the ID of the transaction that the column depends on, but this doesn't work all of the times. To present the effectiveness of the complementary, consider the same database as the one presented above but with a slight addition. A new table, *sales*, will be added. The purpose of this table is to save the total amount of the sales for each month. The *sales* table will have the following structure: *salesID, month* and the *totalsales.* To calculate the total sales we need to look at the *orders* table and find all every order that have a date within a certain boundary. For example, if we need to find the sales for April then we need to find every order that has a date < '2012-01-06' and date > '2012-30-04'. The result of this query will be transaction T_{13} and T_{14}. Thus, the total that will be inserted into the *sales* table is $370. While building the matrix, when we reach the *totalsales* column for this transaction we cannot refer to these two depending on transactions in one column. Hence, for this reason we decided to use the complementary array that will be dynamic and we will be able to add to it as much depending on transactions as we want. In this case, complementary array will look like the table 3.6:

Table 3.6 Complementary matrix for example 2



Consider the case were the intrusion detection system will provide our model with the ID T_2. Our detection algorithm will start by retrieving the malicious transaction with the least transaction ID, but in this case we have only one transaction. Hence, the algorithm will start searching the matrix starting from T_3 rather than starting from T_1 like other models. This shows that when working on a larger scale we might be skipping 100's of rows. Consequently, this will reduce the assessment time. We will also skip the row corresponding to transaction T_2 since we already know that this transaction is malicious. Our algorithm will use two structures: *malicious_trans* and *affected_trans*. The *malicious_trans* will start by having transaction T_2 (i.e. the malicious transaction that the intrusion detection system provided).

The damage assessment algorithm will start by traversing row T_3. Whenever the algorithm sees a '00' it stop the search in the column and starts with the second column. '00' means that this column or data item have not been modified in this transaction. Similarly, if our algorithm sees a '01', it will skip to the next column as this would mean that this data item has been blindly written (i.e. without having the need to check values from previously committed transactions). When our algorithm reaches *M[T_3][CatID]* it will be faced with a new case. *M[T_3][CatID]* contains a negative value. Thus, our algorithm will have to refer to the complementary array to check on which transactions does this transaction depended to write its values. *Comp[T_3]* points to T_2, T_2 will be checked against the set of malicious transactions (*malicious_trans)*. Since T_2 is malicious, T_3 will be added to the *affected_trans*. Since, T_2 is added to the set of affected transactions then there is no need to continue assessing the other columns of this row. The algorithm will skip the other columns and start assessing T_4. While assessing T_4, the algorithm will not find any case other than '00' and '01'. Thus, there will be no need for comparing against the *malicious_trans* and *affected_trans*. At the row related to transaction T_5, all the columns before the *CatID* will be skipped as they are either '00' or '01'. For *M[T_5][CatID]*, the algorithm will have to consider the case of a negative transaction ID. Thus, the algorithm will get the set of transactions that T_5 depends on from the complementary array. There is only one transaction that T_5 depends on which is T_5. T_5 will be compared against the *malicious_trans* that contains only T_2 in this case. Since no similarities exist between the *Comp[T_5]* and *malicious_trans* that means there is no direct effect on the malicious transaction. Hence, no we need to check if T_5 is indirectly affected by T_2. The check of indirect effect is done by checking the set in *affected_trans*. Again there is no similarities between the *affected_trans* and *Comp[T_5]*. Hence, the algorithm should continue by checking every column of T_5. The same process will be followed for each row until we reach the row corresponding to transaction T_{14}. The result of this algorithm will be as follows: *malicious_trans = {T_2}* and *affected_trans = {T_3, T_9, T_{10}, T_{12}, T_{13}, T_{14}}*. At this step the damage assessment algorithm will be done. These results will be sent to the recovery algorithm that will do the actual recovery.

3.6 Recovery Model

The recovery algorithm requires one additional structure that will be responsible for reading the log file. After the detection algorithm is done, it will trigger the recovery algorithm by sending to it the set of malicious transactions and the set of affected transactions. The malicious transactions will be deleted, while the affected transactions would 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 were all of the data is consistent (i.e. no malicious transaction exits and any affected transaction is recovered). In our case, a stable database refers to a database that has recovered every malicious and affected transaction where its data is the same as if no malicious transaction has occurred. In this algorithm, the log file will be read and put into an array so we can index any record we want. We refer to it as if we are indexing an array rather than reading all the records in the log file until we reach the value or record that we want. This way will decrease the access time that will be needed to read the log file whenever we want to delete or recover a dirty transaction. 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 table 4.1.

Steps 1 and 2 show how the algorithm is preparing the environment to start the recovery process. Step 3 and its sub-steps 3.1 and 3.2 represent the recovery process for the affected transactions. We start by the affected transaction to be able to trace back and update the values of the transactions accordingly as if no malicious transaction have occurred and modified our data. For example, if transaction T_x was affected by the malicious transaction T_y,where $y < x$, after T_x reading dataitem2 from T_y, we would go back and check how T_y got dataitem2. Then, modify the record of transaction T_x. After this, we will be able to update any transaction that depends on transaction T_x by reading its new values. After modifying the data of all of the affected transactions now, as shown in step 4, we will retrieve the malicious transactions with their information and delete these unwanted data.

Table 3.7 Recover algorithm

```
1.  Receive the sets of malicious and affected transactions
2.  Read the file into an array
3.  For each transaction in the affected transaction set
    3.1.  Retrieve the log file information for that transaction
    3.2.  Update the transaction accordingly
4.  For each transaction in the malicious transaction set
    4.1.  Retrieve the log file information for that transaction
    4.2.  Delete the transaction
```

3.7 Example on the Recovery Process

In this section, we will explain the recovery algorithm based on the example presented in section 3.5. After working with the damage assessment in section 3.5, we were able to detect all the affected transactions that depends on the malicious transaction T_2. The set of malicious transactions is {T_2} while the set of affected transactions is {T_3, T_9, T_{10}, T_{12}, T_{13}, T_{14}}.

The recovery algorithm will start by recovering from the affected transactions. T_3 is the first affected transactions. Since T_2 has maliciously entered a new category into the database that was not meant to be inserted, any product that belongs to that category should be deleted. These products cannot be classified under any other category. Hence, the only way to recover this is by deleting this transaction. If the product does not exist, then similarly the orders cannot exist. Thus, each of T_9, T_{10}, T_{12}, T_{13} and T_{14} should also be deleted. After recovering all of the affected transactions, we now need to delete the malicious transaction T_2. Any malicious transaction cannot be recovered it should only be deleted. Unlike affected transactions that can be recovered depending on each situation solely. After the recovery process, we reach a stable and consistent database (i.e. a database that does not contain any affected or malicious transaction). The database that we will reach is represented below:

T_1 = Employee (‘1’, ‘Kim’, ‘Stewart’, ‘1980-11-02’,’Sales’, ‘$2000’);

T_4 = Categories (‘2’, ‘Dairy Products);

T_5 = Products (‘2’, ‘Cheese, ‘$4’, ‘2’);

T_6 = Employee (‘2’, ‘John’, ‘Adam’, ‘1987-21-03’, ‘Sales’, ‘$1500’);

T_7 = Customer (‘1’, ‘X’, ‘Beirut’);

T_8 = Customer (‘2’, ‘Y’, ‘Beirut’);

T_{11} = Order (‘3’, ‘1’, ‘2’, ‘2’, ‘50’, ‘$200’, ‘2012-12-04’);

Our proposed model uses a sequential log file and sequential dependency matrix. Both of which will contain only committed transactions and will be flushed at certain checkpoints. The matrix is the basis for the detection algorithm. It will help in assessing the transactions and categorizing them into affected and clean transactions. Then the result will be sent to the recovery algorithm to recover each malicious and affected transaction accordingly. We chose to use matrices in our algorithm due to the fact that matrices reduce the access time that is required to read the log file. In addition, the matrices enable us of skipping unwanted transactions without taking time to read them due to the indexing characteristic. On the other hand, when using a log file we should traverse all the transactions that occurred in order to reach the start point.

CHAPTER FOUR

PERFORMANCE ANALYSIS

4.1 Overview

We tested the performance of our model by means of a simulated environment. 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. Since our algorithm requires the presence of these two prerequisites, the simulated environment will be responsible of generating them before starting the execution of the model. In normal cases, when not working in a simulated environment, the log file and dependency matrix are supposed to be built as the transactions are being executed. Transactions in our database are generated randomly. The average transaction time is 2. The average transaction time reflects the number of products that the customer will buy in our database (i.e. how many transaction will be associated with a single order).

All assumptions are taken into consideration in our proposed simulated environment. The history is assumed to be serializable. Only committed transactions will be added into the log file, such that every added transaction will have a unique and sequential ID, i.e. there will be no $i < j$ such that $T_j < T_i$. This will ensure for us that there will never be a transaction that depends on a transaction that happened after it; this is an assumption that was taken as we built our model.

Moreover, the dependency matrix will also be created using the simulated program. Both the matrix and the log file have similar characteristics, such as only committed transactions will be added to the dependency matrix and they will take sequential IDs, such that there will be no $i < j$ and $T_j < T_i$. This work is usually performed directly after the commitment of any transaction, hence building the matrix up with every transaction commitment.

As soon as an external intrusion detection system detects a malicious transaction, the ID will be sent to our model and the detection and recovery processes will start. For testing our model and for the sake of our simulated environment, we have done testing on more than one ID. The testing is done at different stages to see how would the behavior of our model change as it will have to traverse more rows in the matrix or even when it have to recover a larger amount transactions.

We used the “Northwind Database” in our testing. This database is provided as a template in the Microsoft Access office. The data was then converted to .sql format and added to MySQL. 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.

4.2 Performance Analysis of Damage Assessment

Our model is split into two different parts, damage assessment and recovery, each of which has its own characteristics and importance. These two parts are the building blocks of our model, and hence the performance of each part is important. Hence, we will start by evaluating the performance of the first part of this model which is the detection process.

Figure 4.1 represents the performance of our detection algorithm. The database that we have used to test our model contains 1080 transactions (rows) and 5000 data items (columns), with at most 45 columns being access by a transaction. As we can analyze from the graph that the sooner the attack is detected, the faster the detection process will be. This is due to the fact that our algorithm starts from the malicious row and goes on without looking at any transaction that happened before since as we have mentioned previously our matrix is sequential. If a transaction Tj was detected as a malicious transaction, then we know that we will never find T_i such that i < j and T_i depends on T_j. Hence, now we can point out the advantage of the matrices usage, which is the indexing characteristic that enables us to skip all the transactions that occurred before the malicious transaction and without taking the time to read and skip the clean transactions.

As we can see from figure 4.1, the time needed for damage assessment decrease as the attacker ID increase. 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 have to traverse and check only 81 rows. The time decrease from around 18.13µ second to 5.8µ seconds. As the algorithm has to traverse less number of rows, the time and effort needed for damage assessment will also decrease. This shows that the sooner the attack is detected the better and the faster the damage assessment will be. This shows that our algorithm is capable of decreasing the time needed for damage assessment and hence less denial of service.

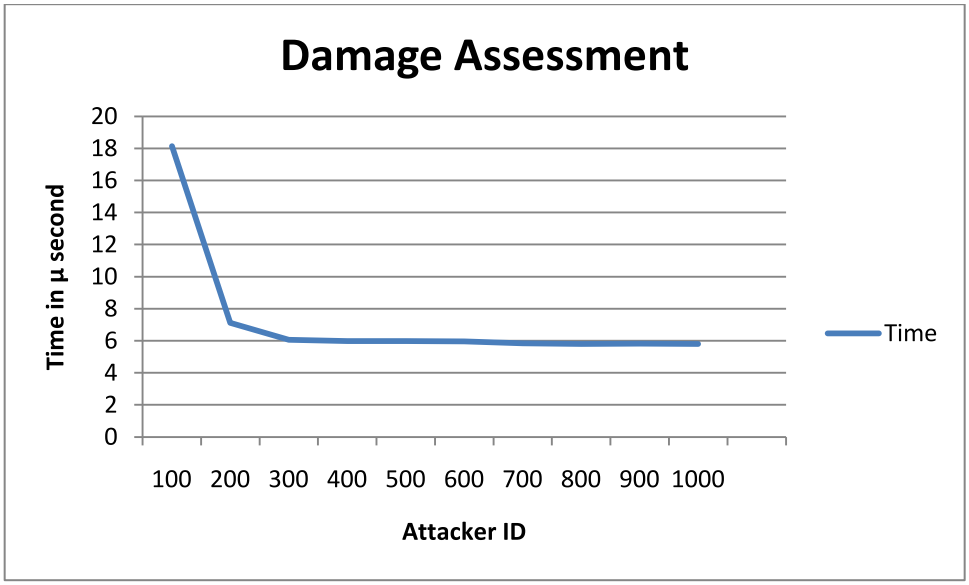


Figure 4.1 Performance of the Detection Algorithm based on different attacker ID

Figure 4.2 portrays a time comparison between our algorithm (matrices) and three other algorithms (traditional, traditional clustered and hybrid cluster algorithms (Haraty & Zeitunlian, 2007)). The four algorithms were tested in the same environment, where we have database of 200 transactions with 5000 data items, in which only 45 data items are accessed as a maximum in a transaction. Therefore, the graph in figure 4.2 shows that our model is faster than the hybrid cluster by at least 167224 times (Kim et al., 2010). This decrease in the damage assessment algorithm is due to the characteristics of our algorithm. The fact that we are using matrices makes our algorithm works faster, especially that matrices are easily indexed. When working with matrices we can easily skip any row and start from the point we want. The fact that our algorithm uses a sequential matrix and that it starts from the point of the attack skipping anything before it makes it work faster. Unlike other algorithm, 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 over previous algorithms. The matrix we are using contains all of the necessary information that is needed to assess the dependent transactions. Moreover, we are working with bits when numbers are not necessary which also reduces the effort.

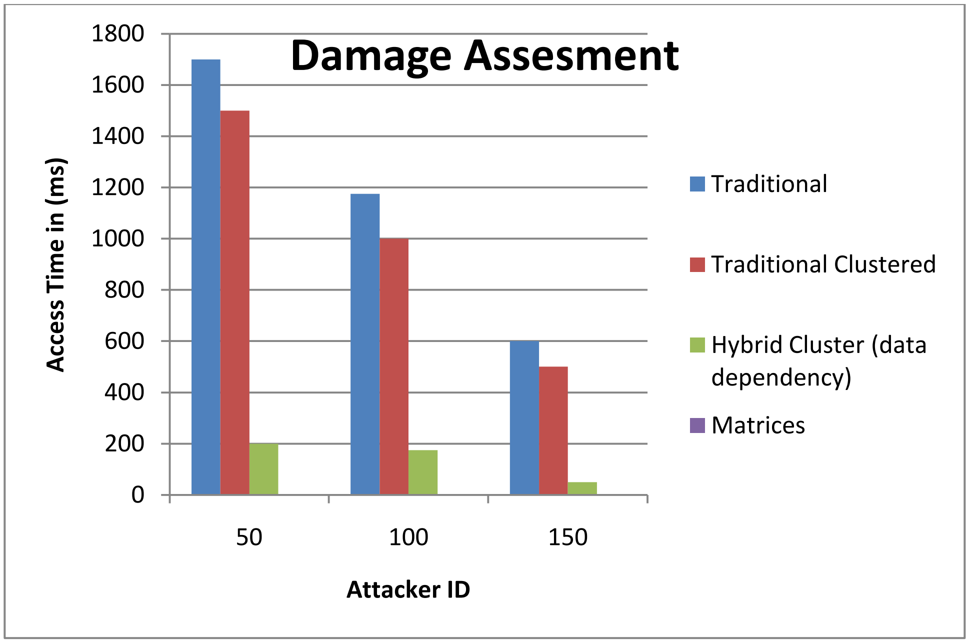


Figure 4.2 Damage assessment time comparison

4.3 Performance Analysis for Recovery Algorithm

After the detection phase and after we have saved all of the malicious and affected transactions, we can move to the second phase, the recovery. In this phase, we recover every transaction that has been affected by malicious transactions and delete every malicious transaction. Figure 4.3 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 needs recovery increases, the time required for this recovery increase as well. Figure 4.3 shows the result for a database that constitute of 200 transactions, 5000 columns. A transaction may access at most 45 columns.

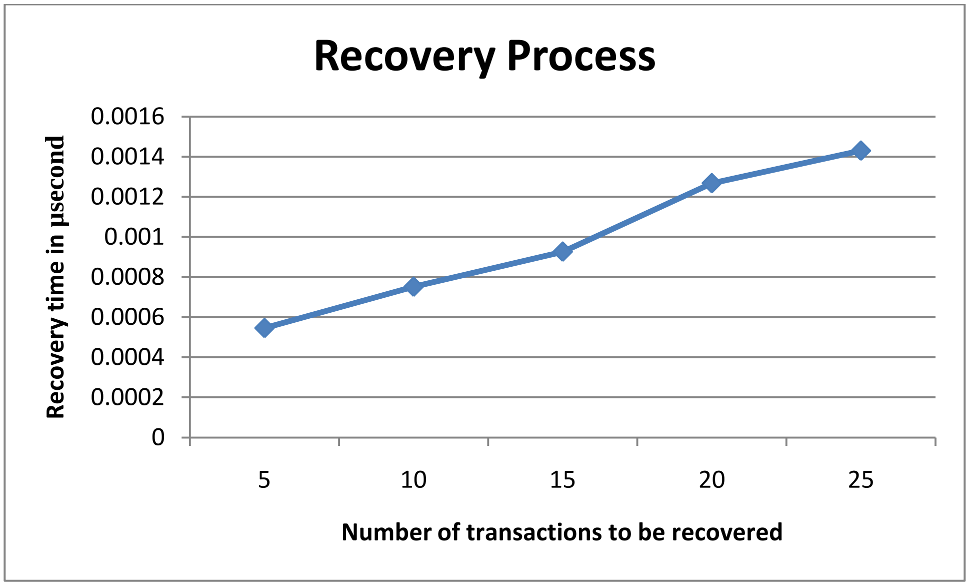


Figure 4.3 Performance of Recovery algorithm based on different number of affected transactions

Figure 4.4 shows the time taken by the recovery algorithm as the number of recovered transactions increases. The database that gave the following results is composed of 1081 rows and 5000 columns. The maximum number of columns that can be accessed by a transaction is 45. Figure 4.3 and 4.4 present the same outcome (i.e. as the number of recovered transactions increases the time taken to recover will increase). The time graphed below is the time needed to recover from both malicious and affected transactions. Figure 4.4 presents the advantage of our algorithm. Even when we worked on a larger scale database our algorithm was still faster than other algorithms. While other algorithms needed milliseconds to recover, our algorithm needed only less than 1 µ second.

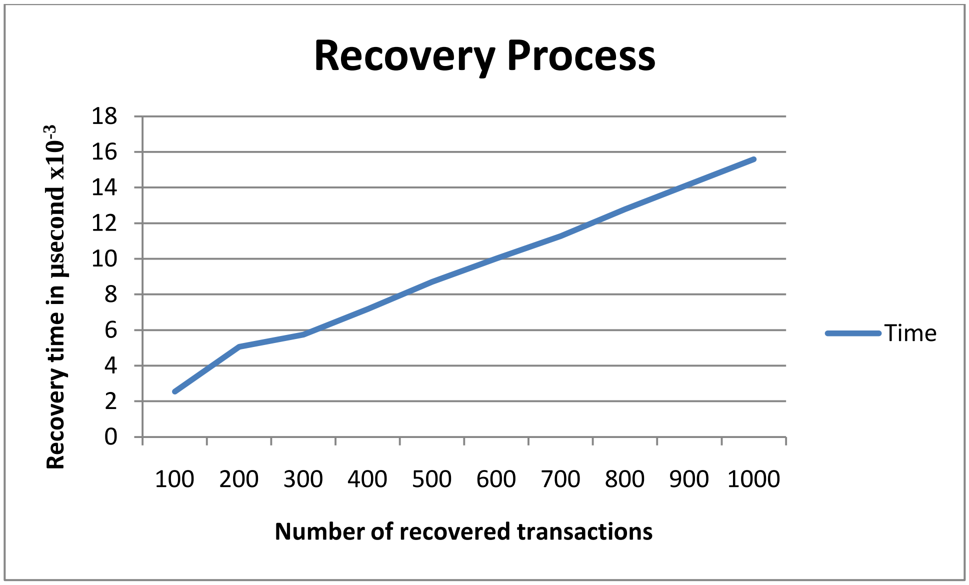


Figure 4.4 Performance of recovery on a larger scale database

Figure 4.5 shows a comparison between our model and 4 other models (traditional, traditional clustered, hybrid clustered with data dependency, and the hybrid cluster with fixed size (Haraty and Zeitunlian, 2007)). The results show that in all the cases our model is much faster than any other model. In addition, in the worst case our model was faster in about 114,000,000 times than the other models. The reason of this improvement between our algorithm and other algorithms 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 we want to recover we just index its position in the array. The time that was taken by other algorithms to read the log file is not needed in our algorithm; this will decrease in the timing. Our algorithm requires the effort to convert the log file to an array and this is done only once at the beginning of the recovery algorithm.

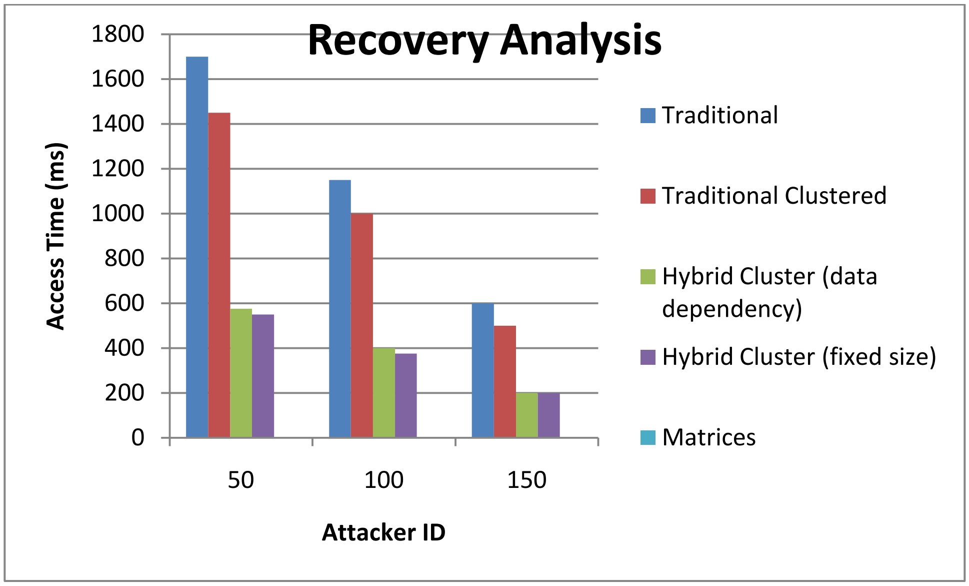


Figure 4.5 Performance of the recovery algorithm in different models

Figure 4.6 portrays the time taken by our model to build the matrix, 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 worst case scenarios we might have to see such numbers. 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 would need to rebuild the matrix and then start our assessment. Since figure 4.5 shows the time needed to build the matrix, detect and recover from malicious transactions, it also presents the worst case scenarios. This proves that even in the worst case our algorithm is faster than when the other algorithms are recovering from normal cases.

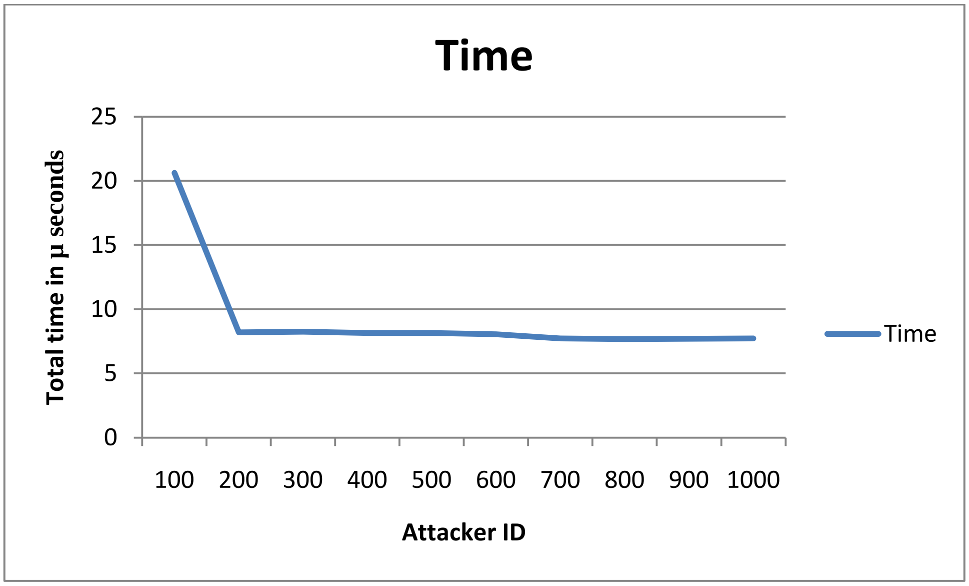


Figure 4.6 Time taken for different attacker IDs to go through our recovery model

In conclusion, our model showed better performance in both detection and recovery processes. The time required by our algorithm to do the full process is much faster than the time required by a single process of other algorithms. Even if we shut down the system at certain times to recover, the time will not have a great negative effect on us. Using matrices saved time and reduced the possibility of having a denial of service or at least a long denial of service.

CHAPTER FIVE

CONCLUSION

The security of a system is composed of three phases: prevention, detection and recovery. Prevention methods do not always work; hackers and attackers always find ways to breach the system. When prevention fails detection systems are supposed to detect the malicious transaction and report it to be stopped as soon as possible. Again detection systems fail to detect the malicious transaction as soon as they happen. For this reason malicious transactions affect other clean data. Hence, recovery methods are supposed to be effective, efficient and fast so that they recover from the malicious transaction and its effects. This model presented a new approach for recovery which depends on matrices. The dependency between transactions is all saved in a matrix that will be formed as the transactions are being committed. We tested our model and compared it with different previous approaches and the results showed that our model is faster by at least 100,000 times. The given 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). Our model has proved to have an advantage on both levels: detection of affected transactions and recovery.

As a future work, we will work on the space issue. Our algorithm requires the presence of a matrix along with another structure to save the transactions it depend on, this requires space that could be diminished.

References

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

Chakraborty, A., Majumdar, A., & Sural, S. (2010, January). 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.

Fu, G., Zhu, H., Feng Y., Zhu, Y., Shi, J., & Chen, M. (2008, October). *Fine grained transaction log for data recovery in database system*. Third Asia-Pacific Trusted Infrastructure Technologies Conference *(*IEEE), Washington, DC, USA.

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

Haeni, R. (1997, January). *Information warfare an introduction.* Washington DC: The George Washington University.

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

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.

Hutchinsn, W. (2006). Information warfare and deception. *Informaing Science, 9*, 213 - 223.

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.

Lala, C., & Panda, B. (2001, July). Evaluating damage from cyber attacks: A model and analysis. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, 31*(4), 300 - 310.

Libicki, M., & Fellow, S. (1995, May).*What is information Warfare?* United States: United States Government Printing.

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

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

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

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.*

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.

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

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

Panda, B., & Zhou, J. (2003, July 16-18). Database damage assessment using a matrix based approach: An intrusion response system. *Proceedings of the 7th International Database Engineering and Applications Symposium (IDEAS '03)*, 336 – 341.

Ragothaman, P., & Panda, B (2002, July). 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

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.

Ryan, M. (1998, August). Information Operation. *Air force.* Retrieved from http://www.dtic.mil/doctrine/jel/service_pubs/afd2_5.pdf

Xie, M., Zhu, H., Feng, Y., & Hu, G. (2008, December 27 - 28). Tracking and repairing damaged databases using before image table. *Japan-China Joint Workshop on Frontier of Computer Science and Technology (*IEEE), 36 – 41.

Zheng, J., Qin, X., & Sun J. (2007). Data dependency based recovery approaches in survival database systems. In Y. Shi, G. Dick Van Albada, P. Sloot, & J. Dongarra, (Eds), *Computational Science ICCS 2007: 7th International Conference: Lecture Notes in Computer Science* (pp. 1131 - 1138). Berlin, Germany: Springer-Verlag.

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

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.

Zuo, Y., & Panda, B. (2004, June). Fuzzy dependency and its applications in damage assessment and recovery. *Proceedings of the 2004 IEEE Workshop on Information Assurance*, 350 – 357.

Appendix I

Detection algorithm

```
function Detect($M,$i, $j, $Malicious, $k,$secondArray)    {
$startTime=microtime (true);
$firstAffected=min($Malicious);
$startID=$firstAffected+1;
$Affcol=0;
for($row=$startID; $row<$j; $row++) {
$flag=0;
for ($column=1; $column<$i; $column++)   {
for($Mcol=0; $Mcol<$k;$Mcol++)   {
if($M[$row][$column]==$Malicious[$Mcol]) {
$Taffected[$Affcol]=$M[$row][0];
$Affcol++;
$flag=1;
break;
}
if(abs($M[$row][$column])==$Malicious[$Mcol])  {
$Taffected[$Affcol]=$M[$row][0];
$Affcol++;
$flag=1;
break;
}
if($M[$row][$column]<0) {
$transID=abs($M[$row][$column]);
if(array_key_exists($transID,$secondArray)) {
```

```
$countArray=count($secondArray[$transID]);
for($traverse=0;$traverse<$countArray;$traverse++) {
if($secondArray[$transID][$traverse]==$M[$row][0]) {
$Taffected[$Affcol]=$M[$row][0];
$Affcol++;
$flag=1;
break;
}
}
}
}
  //if not found in the malicous array then it should be in the affected part
if($flag==0)    {
for( $m=0; $m< $Affcol;$m++)        {
 if($M[$row][$column]==$Taffected[$m]) {
 $Taffected[$Affcol]=$M[$row][0];
$Affcol++;
$flag=1;
break;
}
 if(abs($M[$row][$column])==$Taffected[$m]) {
$Taffected[$Affcol]=$M[$row][0];
$Affcol++;
$flag=1;
break;
}
if($M[$row][$column]<0) {
```

```
$transID=abs($M[$row][$column]);
if(array_key_exists($transID,$secondArray))   {
$countArray=count($secondArray[$transID]);
for($traverse=0;$traverse<$countArray;$traverse++)  {
if($secondArray[$transID][$traverse]==$Taffected[$m])  {
$Taffected[$Affcol]=$M[$row][0];
$Affcol++;
$flag=1;
break;
}
}
}
}
 }
}
}//end Malicious array
}//end $columns loop
}//end $rows loop
$endTime=microtime (true);
$detecttime=$endTime-$startTime;
Recover($Taffected, $Malicious);
}//end function
```

Appendix II

Recovery algorithm

```
function Recover( $affected,  $Malicious)     {
        $startTime=microtime (true);
        $lines = file ("transactionsLog.txt");
        For ($i=0;$i<count($affected);$i++)  {
                $words=split ("[ ]", $lines[($affected[$i]-1)]);
                $info=$words [12];
                $sepInfo=split (",", $info);
                if($sepInfo[0] && $sepInfo[1])          {
                        $sql="Delete from ".$sepInfo[0]." where ".$sepInfo[1];
                }
        }
        for ($j=0;$j<count($Malicious);$j++)            {
                $words=split ("[ ]", $lines[($Malicious[$j]-1)]);
                $info=$words [12];
                $sepInfo=split (",", $info);
                if($sepInfo[0] && $sepInfo[1]) {
                        $sql="Delete from ".$sepInfo[0]." where ".$sepInfo[1];
                }
        }
        $endTime=microtime (true);
        $total=$endTime-$startTime;
}
```