LEBANESE AMERICAN UNIVERSITY

A HYBRID DAMAGE ASSESSMENT AND RECOVERY APPROACH IN MULTILEVEL DATABASES

By

OMAR B. EL ZARIF

A thesis

Submitted in partial fulfillment of the requirements

for the degree of Master of Science in Computer Science

School of Arts and Sciences

March 2018

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

THESIS APPROVAL FORM

Student Name: Omar Badih El Zarif I.D. #: 201100509

Thesis Title: A Hybrid Damage Assessment and Recovery Approach in Multilevel Databases

Program: Masters of Science in Computer Science

Department: Computer Science and Mathematics

School: Arts and Sciences

The undersigned certify that they have examined the final electronic copy of this thesis and approved it in Partial Fulfillment of the requirements for the degree of:

Masters of Science in the major of Computer Science

Thesis Advisor's Name: Rami Haraty	Signature: [redacted]	DATE: 26 / 3 / 2018 (Day / Month / Year)
Committee Member's Name: Imad Rahal	Signature: [redacted]	DATE: 26 / 03 / 2018 (Day / Month / Year)
Committee Member's Name: Samer Habre	Signature: [redacted]	DATE: 26 / 03 / 2018 (Day / Month / Year)

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

THESIS COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY NON-EXCLUSIVE DISTRIBUTION LICENSE

By signing and submitting this license, you (the author(s) or copyright owner) grants the Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic formats and in any medium, including but not limited to audio or video. You agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. You also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Name: Omar El Zarif

Signature: [redacted]

Date: 26/03/2018

LAU
الجامعة اللبنانية الأميركية
Lebanese American University

PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that:

1. I have read and understood LAU's Plagiarism Policy.
2. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.
3. 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: [redacted] Zarif

Signatu[redacted]

Date: 26/03/2018

ACKNOWLEDGMENT

This thesis was conducted due the efforts of many dear people.
Many thanks to my advisor, Dr. Ramzi Haraty, who inspired this idea, and for providing me with his consultancy and guidance. Also thanks to my committee members, Dr. Samer Habre and Dr. Imad Rahal, who supported me through this journey.

And finally, thanks to my parents and friends who provided me with support and love throughout my work.

A HYBRID DAMAGE ASSESSMENT AND RECOVERY APPROACH IN MULTILEVEL DATABASES

Omar B. EL ZARIF

ABSTRACT

In the midst of the technology advancements that we are witnessing nowadays, information security is of an essence. Information warfare is actively introducing new offensive strategies to exploit databases that include sensitive data, or even classified and secret data in the case of the multilevel ones. The attacks vary from stealing sensitive data, to modifying crucial records, or simply corrupting the database. Many approaches to detect and prevent from such attacks are available, from firewalls to intrusion detection systems. But in many cases, prevention can fail, and detection could happen long after the attack. That implies the necessity of damage assessment and recovery to restore the database integrity. We introduce in this work a hybrid lightweight approach based on clustering and graphs to track transactional dependencies in multilevel databases. To the best of our knowledge, targeting multilevel databases has not been tackled before, so we will compare our approach with single level databases approaches, taking into consideration the additional security and integrity constraints imposed by the multilevel nature.

Keywords: Damage Assessment, Damage Recovery, Data Dependency, Information Warfare, Malicious Attacks, Transactional Dependency

TABLE OF CONTENTS

Chapter **Page**

I Introduction **1**

1.1 Overview 1

1.2 The Problem 2

1.3 Scope of the Work 4

1.4 Organization of the Report 4

II Multilevel Databases **6**

2.1 Security Properties 6

2.2 Polyinstantiation 7

III Literature Review **11**

3.1 Log Files 11

3.2 Graph 12

3.3 Clustering 13

3.4 Before Images and Column Dependency 14

3.5 Matrices 15

IV Our Approach **18**

4.1 Background 18

4.2 Adaptation To Multilevel 20

4.3 Complexity Analysis 28

V Experimental Results **30**

5.1 Performance Results of the Detection Algorithm 31

5.2 Performance Results of the Recovery Algorithm 33
5.3 Memory Footprint Analysis 34

VI Conclusion **38**

LIST OF FIGURES

Figure		Page
1	Adjacency list for the sample scenario.	19
2	Bipartite graph representing the writes on the unclassified level from the left and on the secret level from the right.	27
3	Adjacency list illustrating the transactions dependencies.	27
4	Performance of the detection algorithm	31
5	Performance of the detection in comparison with other recent models	32
6	Performance of the detection in comparison with classical models	33
7	Performance of the recovery phase	34
8	Performance of the recovery in comparison with classical models	35
9	Memory consumption in comparison with recent models	36
10	Memory consumption in the best case scenario	37
11	Memory consumption in the worst case scenario	37

LIST OF TABLES

Table		Page
1	Employee table where classification is on the record level	9
2	Employee table where classification is on the data item level	10
3	The damage assessment process	23
4	The damage detection process	24
5	The damage recovery process	24
6	Our database schema	25

Chapter One

Introduction

1.1 Overview

Information security is an integral part in any database system, taking into consideration the sensitive data that these systems harness. This data vary from financial records, to military information, medical history and many other fields, where it is essential to protect or rectify any occurring damage to ensure the correctness of the database. Information security in multilevel databases is a top concern. Due to the nature of the multilevel databases, data secrecy and database integrity should be maintained. Information warfare defines attacks on the database system as any action that can disrupt the ACID properties (Atomicity, Consistency, Isolation, and Durability). These attacks can take the form of spoofing, viruses, worms and denial of service [1].

Prevention, detection and recovery are the three main pillars to respond to a certain attack. Prevention is the first step, where many preventive measures can be taken to elude the attack; these methods use techniques such as firewalls, authentication, authorization, access control and anti-viruses. These lasts can work to a certain extent but cannot be relied on solely [2].

The second step the detection is the complimentary to the first, since any prevention method should be accompanied by a detection scheme. The intrusion detection system (IDS) is usually the driving force in this phase; it is in charge of detecting an attack,

usually by identifying the malicious transactions following historical analysis [3]. The two common techniques in identification are called anomaly detection and misuse detection [4]. The first studies the behaviour of the user in the system to grasp an idea of what the usual behaviour is and what is considered an anomaly. In contrast, the other approach compares the actions taken to a predefined set of rules in the system to identify the outlying danger.

Living in an interconnected world, where the Internet is the main role player in any system, information sharing is becoming inevitable. This facilitates the fact that an attack can virtually come from any place, and can penetrate the system no matter how much the last is fortified. The weak-point in any system, is the trust given to a user. When some systems allow blindly to perform any transaction from any user, others use approaches such as in [5] where users are classified. The system studies the behaviour of the user and the transactions to detect the good from the malicious ones. Thus, enforcing some form of access control, and in return developing the ability to detect an intrusion.

Although identifying the malicious transactions is now the last line of defense in the system, the process will not be instantaneous and some benign transactions will be affected by the malicious ones. Hence, the system should fight back by recovering. Some systems enforce enhanced recovery techniques by integrating tables and data structures to the process, such as in [6], but still these techniques were proved to be time consuming and resources heavy. The algorithm for detection and recovery should then, be efficient in time to reduce the amount of affected transactions, and should adhere low system requirements to prevent a complete shut-down, allowing only restricted access to the affected part of the system.

1.2 The Problem

The elapsed time between the attack and the detection of the malicious transactions could vary depending on the situation. Hence, in any system the transactions that

should be assessed should follow some set of rules to ensure the correctness of the algorithm. In this work we care about the transactions that read from other affected or malicious transactions mainly. A running transaction T_i reading data items from a committed transaction T_j, when T_j is found to be malicious, means that T_i needs to be recovered and its effects should be annulled.

Damage assessment algorithms vary in the information warfare field. But they all follow one of the two well-known dependency requirements, either data dependency or transaction dependency. The first works on identifying malicious transactions if they read from an affected data item, while the second detects malicious transactions if they read a data item written previously by a malicious transaction [7]. If any malicious transaction was identified by the IDS, other transactions found dependent on it following one of those two schemes will be identified as affected transactions.

The main worries following any of these two approaches are the complexity and the efficiency of the damage assessment and the recovery algorithm. The system provides a log file that can span in size, which describes all the committed transactions in the system. Thus, it is evident that algorithms relying on extensive I/O operations on the log file itself can cause denial of service for the recovering system. Some researchers have introduced auxiliary data-structures to enhance performance [8], [9]. And some others used additional techniques to aid in the recovery process; we identify two approaches in [10].

- **Write-Ahead Logging:** It records the before image and after image in the database to support the undo operation; therefore it performs instant recovery. The images are flushed when the after image becomes the permanent copy.

- **Checkpoints:** It saves the time for recovery, and it is used periodically. The committed transactions in the log are flushed to disk to indicate a stable state of the database at that point, a checkpoint.

1.3 Scope of the Work

In this work we present a model that relies on auxiliary data structures for an efficient and quick damage assessment and recovery in multilevel databases. We define two structures, the first being the dependency matrix, while the second being the write buckets, which are highly involved in the clustering part of our hybrid approach.

The approach requires access to the log file only once, it passes through it to record the writes in the buckets, and tracks from there the dependency. The log is scanned from the first affected transaction location, which improves further the performance of our algorithm. Then it progressively records the primary keys of the written tuples in the corresponding buckets. Our approach is then based on transaction dependency, the use of the buckets is strictly to track the written transactions and not data items. Although it is a transaction dependency scheme, to boost the performance of recovery we record in the buckets, beside the primary keys of the modified records, the modified fields and their before image. This serves us well when recovering, since the only operation needed is just a restoration of the before image in the database.

This proposed algorithm will be compared with few single level database approaches based on matrices mainly, as our proposition is in its core a matrix approach but tailored towards multilevel databases. We compare it also to classical approaches based on log files and clustering, to properly classify our work in the state of art. The algorithm will prove to be a contender with single level approaches, since tackling multilevel databases is new, and to the best of our knowledge, no previous work was done in regards to damage assessment and recovery. Regardless, our comparison will formulate a new perspective of what we can achieve in the multilevel scenario.

1.4 Organization of the Report

We have introduced our problem and approach in this chapter. In chapter II, we define the properties of a multilevel database, and the constraints imposed on us to adapt the

work from a single level approach to a multilevel one. In chapter III, we research the literature in information warfare for classical approaches in damage assessment and recovery, and more modern ones. We explain in depth our approach in chapter IV, then we compare our work with the previous presented approaches in chapter V. We conclude our work with a sight for future enhancements and suggestions in chapter VI.

Chapter Two

Multilevel Databases

Typical relational databases used in the field are single level ones. The notion of multilevel has not caught the mainstream attention, but still some fields require this type of databases for the additional secrecy and security enhancements that it provides.

2.1 Security Properties

The key difference between multilevel and single level is in the storage of the data items. While single level databases store each data item strictly once, multilevel databases allow multiple values for each data item. The concept is close to multiversioning in the database, but the multiple versions presented here are related to views and clearances. Access control is a key concept here, where each user is allowed a clearance level that dominates certain security levels [11]. The clearance to security level relationship is usually imposed by rules on the reads and writes in the system. Two simple properties to manage the transactions can be explained as:

- **The simple security property:** A transaction is allowed to read in an appropriate level, only if its clearance is of the same or higher level.
- **The $\star$ property:** A transaction is allowed to write in an appropriate level, if its clearance is of the same level.

The problem of clearance and views branches into more complex sub-problems in the

system. All data can be classified to a clearance level following some view constraints in the system. This was explained by a fundamental paper in [12]. The classification can span from a record to a more grain subject, such as tuples and the data item itself. All data in the system can be classified, this component can be a security level, a category or a combination of both. Known security levels in multilevel systems are TOP-SECRET, SECRET, CONFIDENTIAL, UNCLASSIFIED, and categories can be simply as CRYPTO and NON-CRYPTO. These are the most commonly used definitions, but this does not prohibit adding more levels, or categories. Classifying a coarse object such as the record, will classify the data items within the record itself. So a clearance level that dominates the record, dominates all the data items in the record.

2.2 Polyinstantiation

Following the multilevel constraints imposed in the system, we can deduce that data insertion will not be handled as it was in the single level scheme. Suppose that a transaction at level UNCLASSIFIED - *U* wants to insert a new record in the database with a primary key, that has been previously inserted in another level, supposedly CONFIDENTIAL - *C*. Naturally, in a single level database this instance would be rejected, since the use an existing primary key is forbidden in an INSERT query. But in the case of multilevel databases, multiple records with the same primary key had to be allowed. This concept is called polyinstantiation as defined in [13]. It concerns the tuple itself by allowing multiple tuples with same primary key but different access classes to be stored in the same table. This can happen in two cases referred to as visible and invisible polyinstantiation.

- **Visible polyinstantiation:** A transaction with higher clearance level tries to insert data in a field that already contains this data in a lower security level. By overwriting the existing data, access control is violated, causing loss of information in the lower levels. Hence, polyinstantiation is needed to create a new tuple with the appropriate security level.

- **The invisible polyinstantiation:** it is the opposite of the first case, where a transaction with a lower clearance level tries to insert into a field with a higher level, where the data is already present. Rejecting the last would confirm the existence of information in the higher level, which implies the necessity of polyinstantiation to create a new tuple with the appropriate security level.

This scenario affects directly the integrity of the database, and to ensure the correctness, some rules had to be defined. We discuss the main rules as explained in [14]. The first concern regarding integrity has to do with the primary key, it is referred to as entity integrity. It is defined as: let K be a declared primary key of a relation R visible to the user, and C_k the classification of the key, if an attribute A_i is part of the key, A_i can not be null. Null integrity is the second rule, it is defined by: for each instance R_c of a relation R, null elements should be of the same class C_k of the primary key. Polyinstantiation integrity is the third and last rule that will define the multilevel behaviour. A relation R satisfies the polyinstantiation integrity if and only if for each tuple the value of attribute A_i is determined by the classification level C_i of the attribute, and the classification level C_k of the primary key.

For further explanation of the concept of polyinstantiation, we discuss different insertion scenarios in a multilevel database. We showcase one table that classifies records to different security levels. This last classifies each record and all its data items under one security level. Other classification techniques would be more grain, they classify the data items on the tuple level, or on the data item level itself. We showcase the data item classification technique in another table.

Table 1 adopts the record level classification scheme. It contains the already inserted records of a sample employee table. The level field represents the security level of the record. This field is an enumerator that accepts the values U, C, S, or TS. These are the possible values for the security level in the database. Using the record classification variation, we ensure the polyinstantiation properties by enforcing that all the data items in the relation are of the same level of their primary key, which is the *id* in the record.

The distinct record is then defined by the *id* and the *level* field; that is there would be no two records with same *id* on the same security level. Thus, the primary key of the record is composed of the *id* and $level$ fields. It is a composite key, where its integrity should be respected by prohibiting null values in any of the two fields.

In this table, we notice that we have two records on the unclassified level with different *id* values, and two records with the same *id* value. The two records with the same *id* value were created when a new query was executed in the system inserting $id = 1019$ record on the secret level, when the unclassified version of it was already present. The system respecting the polyinstantiation rules would create a new record on the secret level for the $id = 1019$. The rule can then be generalized to ensure the polyinstantiation integrity; new records should be created when an already existing primary key on a different level is present.

Table 1: Employee table where classification is on the record level

id	name	job-title	address	phone-number	level
1019	Samir Khalil	Teacher	New York	70017811	U
1019	Samir Khalil	Professor	New York	03718391	S
1110	John Macintosh	Plumber	Los Angeles	70033811	U

In the other classification cases, we would have a field for the classification level for each data item or tuple. Hence, each data item would be represented as a list where the values in that list are distinct for each level. That scenario is shown in table 2. It is a table that respects the data item classification scheme. The last shows two records with the same primary key; the *id* field, but each on a different security level, secret and unclassified. The only differing data item in the two records is in the $job-title$ field. On the unclassified level, the value is $Teacher$, and on the secret level, it is $Professor$. The polyinstantiation integrity in this scenario is more complex. It is satisfied if for each record the data items are classified with a security level equal or lower to that

of the primary key. The first record clearly respects the property having all its data items on the unclassified level. The second record of $id = 1019$ S respects the property having all its data items either on the secret or unclassified level.

Table 2: Employee table where classification is on the data item level

id	name	job-title	address	phone-number
1019 U	Samir Khalil U	Teacher U	New York U	70017811 U
1019 S	Samir Khalil U	Professor S	New York U	70017811 U

In our approach we choose the record classification scheme. It is a simpler approach for the purpose of our work, as our work is not purely focused on the study of polyinstantiation properties and rules. We are satisfying the needed multilevel security properties to perform our damage assessment and recovery. It is worthy to note that the record classification scheme was simplified in table 1. Practically, the level field is obscured, and the system would manage the insertions based on the level of the transactions. The system would then create multiple views for each security level that is accessible depending on the clearance level of the transactions. The obscuration is adopted to attain further security in the database.

Chapter Three

Literature Review

Information warfare advancement has inspired many algorithms to help in the damage assessment and recovery. The main purpose for all these algorithms is to track the set of affected transactions and their operations to achieve a successful recovery. To track the set of affected transactions the algorithm must follow one of the two schemes: transaction dependency or data item dependency. Many approaches were born to implement either of the two dependencies. Some are log based, and use the log file itself by iterating to track the dependencies. Others use auxiliary data structures to record on fly the transactions. Affected transactions are then detected using for example, graph approaches, or matrices, or cluster.

3.1 Log Files

Panda and al. introduced two approaches in [9] and [15], based on log files. Both approaches treats the log file as the only available resource to track the dependencies. The first uses transactions and the relative read-from/write relations to track the dependencies, while the second tracks the dependencies from each read-from/write relation on the data items. The first approach divides the log file into multiple segments where all dependent transactions are stored, while the second mainly stores the dependent data items, where a graph is then constructed to assess the affected transactions.

In [8] a classic log file technique to recovery and damage assessment is presented, the authors rely on two log files to track the dependencies. The first log file is for the writes, while the other is for the reads. Then they describe a two pass algorithm, to track the dependencies. The first pass is the damage assessment part, it starts by scanning forward from the first affected transaction in the log to detect all the affected or malicious transactions. And the second pass, which is the recovery part, scans from the end of the log file in a backward direction to undo all the detected transactions.

Another approach relying on the log file is proposed in [16]. Bai and Liu introduced a lightweight damage tracking, quarantine and recovery method. The approach is called "Trace", it advocates for a zero downtime in the system by enforcing two rules: to recover the affected data on the fly, which happens by ensuring that damage assessment and recovery are done simultaneously, and to avoid blocking read-only transactions. "Trace" smart behaviour is attained by using smart tagging techniques to identify the malicious transactions. The system also supports two modes of operation: the standby mode and the cleansing mode. The first is the normal mode of operation, where the second is the one that performs the damage assessment and recovery, it is usually triggered by the IDS when an intrusion is detected.

3.2 Graph

In [17], Panda and Gordano implemented two data dependency approaches. The main difference between the two approaches is that one of them performs damage assessment and recovery in parallel while the second performs them separately. Separating the two approaches introduces a higher burden of denial of service period, but in its turn, it saves resources and computing power. Both approaches use a directed graph to map the data items and the transactions that operated on them in the database. This will be built when the IDS detects an anomaly in the system, it will then help showcasing the spreading damage.

Ammann et al. in [8] describe also a classic graph recovery technique that sorts the

transactions based on a graph. The graph is built by assuming two sets of transactions: the good and the suspicious, and will represent the transactions in a way where the malicious and affected transactions can be easily identified. The algorithm has two modes, a cold-start and a warm-start. The main difference between them is the time taken to perform the recovery with respect to the normal execution of the database. The first allows the normal execution suffering from a longer time to recover, while the second disables the execution performing at full power for recovering.

Panda and Zuo in [18] presented two graph based recovery and assessment models that are targeted towards distributed databases. The usual two schemes for any distributed system are centralization or decentralization, respectively the two algorithms are following the two concepts. Thus, the first model requires a coordinator, which is elected based on its location, its processing capabilities and backup requirements, while the second model is a peer-to-peer model where communication is done in an ad-hoc manner. In both cases, each site is only responsible of its local log file, where in the case of any attack the site manager will be responsible of detecting the threat locally.

Liu and Yu in [19] presented a distributed recovery and assessment model that spans along the sites in the system. The sites have each a local damage assessment and recovery (DAR) manager, which is responsible to detect the affected transactions. But transaction dependency can be present within multiple sites, so a global coordinator is required. The last will help in identifying the appropriate local DAR to forward to some affected transactions from another site as these transactions might be a subject of further dependencies.

3.3 Clustering

Another type of approach in assessment and recovery is based on clustering the log file. The log file would contain the transactions and their operations, and these algorithms are interested in classifying the lasts in clusters, by grouping the transactions affecting

certain data items, to ease the tracking of the dependencies. Sometimes, the clustering might be by imposing constraints on the logs. Panda and Ragothaman for an example in [20] suggested a simple cluster technique by requiring a limitation on the log file based on size, the number of committed transactions, or a time window on the cluster.

Haraty and Zeitunlian proposed an approach in [21], that is an exact data item dependency approach. It assumes that the history is rigorously serializable first, and the transaction IDs are sequentially incremented by one starting by T_1. They use two data structures *transaction subcluster_list* and *data subcluster_list*. The first list stores transaction IDs and the relative *data subcluster_list* ID for each transaction, the *data subcluster_list* would then contain the data items of each transaction listed in the *transaction subcluster_list*, whether the data item was read or written. These data structures are built from the log file itself, and will be used whenever the IDS detects an attack to identify the list of the affected transactions. The algorithm furthermore builds in the detection phase two data structures *Damaged_DI* that is the list of data items affected by the malicious transactions, and *Damaged_PB* that contains the execution blocks of those transactions. The blocks will be essential for the recovery, since they contain the interaction of each transaction with the data items.

Lala and Panda in [22] worked on clustering the transactions with an objective of eliminating the harmless ones. The model creates a matrix that represents the dependency, where there is an extra column "ClusterID" to identify the cluster in which the transactions belong to. The clustering is dynamic, and the size of it could vary, the algorithm then faces a trade-off of accuracy to size. Condensing the size could misplace some transactions in the clusters, when a bigger cluster size means a more accurate identification of dependencies.

3.4 Before Images and Column Dependency

An interesting approach was presented by Feng et al. in [23]. It relies mainly on storing before images for the database. The database contains before images replica

table that is transparent to the user, that shall be used to restore the last stable state of the database. The integrity of the database is preserved by storing at each row update, whether it is an insert or delete, the before image of the row. Then in the event of an attack the database would be recovered by restoring the before images. Constraints are applied and tweaked to limit the inevitable huge growth of the table.

Chakraborty et al. in [24] introduced an algorithm based on column dependency. In its essence, the algorithm is a transactional dependency one based on columns relations between transactions. They proposed two versions of the algorithm. The first: the static one works in offline mode where the database should be halted, and requires a pre-defined input as a starting point. The input is the set of malicious transactions, all committed transactions and the schedule of execution of the lasts. The online version, on the other hand, builds the dependencies without halting the normal system operation, where the recovery is done in three phases: detection, recovery and confinement.

In [25], the authors proposed an approach of damage assessment and recovery based on multiversioning the data items. Unlike traditional models that investigate transaction dependencies or operations, this approach identifies affected data items using timestamps and expressions. The data items are controlled by multiple agents, and each data item is assigned to one agent only. To record the multiple versions on each write on a data item, the timestamp of that write is recorded. Thus, the approach records all versions that are used to update a certain data item. Whenever a version is identified as affected, it is added to a damaged list where each agent would check against his version of the data item to assess the damage. The approach proves its efficiency since it detects affected data items only, and performs detection and recovery by multiple agents in parallel.

3.5 Matrices

Haraty et al. in [26] and [27] adopted an approach based on matrices to assess the damage and recover. The system assumes an active IDS, that reports the set of malicious

transactions. The history is rigorously serializable and checkpoints are used to track the stable states of the database in the log file. The matrix is a two-dimensional array that represents the data items in the system and the set of the committed transactions. The data items in the system could undergo three types of interactions: a blind write, a modification according to other transactions (in the example of a previous read then write), or can be left untouched (like a simple read). The matrix will then be built following these rules:

- **0:** will mark a transaction left intact.
- **1:** will mark that the data item is blindly written.
- **A positive transaction ID:** T_i means that the data item was modified according to the last value of the data item written by T_i.
- **A negative transaction ID:** $-T_i$ means that the data item was modified based on different reads by the transaction.

To adhere to the negative ID condition, an additional data structure is required to track all interactions of transactions on the data items. The recovery could then be achieved by checking against the initial matrix in the blind write and positive ID cases, while in the negative case the algorithm would have to traverse the additional data structure to track the affected transactions.

Another matrix based approach is presented by Zhou et al. in [28]. The authors build different types of matrices, to drop the use of the log file completely. Then they manipulate the matrices using logical AND or OR operators. The two needed matrices are one read matrix, that stores all the data items and transactions that read from it, while the other is a write matrix that serves the same purpose for the writes. The algorithm also requires two additional matrices; a damaged data vector containing identified damaged data items, and a damaged transactions list that contains the identified damaged transactions.

Kaddoura et al. in [29] also adopted a matrix approach. Their approach is based on matrices solely and it is a data dependency approach. They save in a matrix data items as columns and the transactions as rows, where each data item written is marked within the transaction row. The algorithm does not require any additional log file to detect the dependencies, and to speed up the performance of the algorithm, it tracks the number the affected data items in each row. Indexing is also used to save time in traversing the matrix, where each row keeps an index of the first affected data item. The approach uses another matrix to store the details of the transactions in the system for recovery purposes.

Chapter Four

Our Approach

4.1 Background

As it was mentioned in the introduction, our approach is inspired mainly from two techniques well used in information warfare for assessment and recovery. Our first being the graph approach. It relies solely on the transactional dependency to track the read/write relationships in the system. And the second one being the clustering technique that was used in another context in our work. In its classical form, the last is used mainly to group the transactions following common shared characteristics, either to cluster based on transactions and its data items, or by applying specific constraints on the logs. But in our case it was clustered based on the level of the transactions in the database, which is a common characteristic for them. To explain our approach further, we introduce two single level approaches that are adapted to our work.

The following work in [7] and [30] are both a transaction dependency approach that are based on tracking the transactions in the system following the read/write relationships. Although there is differences between the two approaches, in their core both work on creating an adjacency list, that is a graph representation where each row depicts a transaction dependent on the other transactions present in its columns. Both works assume a rigorously serializable history, where the transactions IDs are to be incremented sequentially in the system, and all current transactions can only read from

the previous transactions only. Hence, the adjacency list can represent on each row a unique transaction dependency.

The read/write relationship that is tracked in the system, is defined by the operation of reading from a previous transaction then writing. The data fields written or read are not of importance, as this is a generalized rule to carry the work with less complexity. In the work of Haraty, Kaddoura and Zekri [7] the adjacency list is represented as an array of lists. Each list corresponds to a transaction and its dependencies, the index of the list represents the transaction ID, and the content is the dependent transactions. The list stores only committed transactions in order, based on the rule; if a transaction T_i reads from the write of another transaction T_j then wrote, the transaction T_j is recorded as a neighbour for T_i. We take for an example a small scenario representing this approach, shown in figure 1. T_0 wrote x only, T_1 read from T_0 and wrote. Hence, it is recorded as dependency.

$T_0 : w(x)$

$T_1 : r(x)w(y)$

$T_2 : r(x)w(z)$

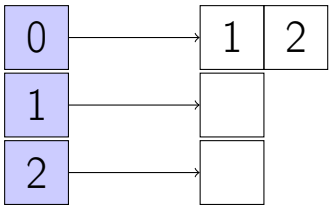


Figure 1: Adjacency list for the sample scenario.

In the approach of [30] the recording is done in an opposite manner, Haraty and Sai record for each transaction all other transactions that the last read from. For example, if a transaction with ID 200 read from the writes of transaction 150 and 151, then wrote, we would record for the transaction 200 two dependent transactions, which are 150 and 151.

To store the dependencies the two algorithms pass through the log file once to check

the read/write relationships. The log file can be completely abandoned in the following stages, since the lists represent the complete dependencies in the system. To find the set of affected transactions, the intrusion detection system would report malicious transactions that would be the initial point of entry for a simple graph search algorithm to detect all the affected transactions. The search in the case of [7] is more efficient since the graph stores for each transaction, the counterparts that read then write from. This facilitates the search since for each malicious transaction, the algorithm can directly track the first affected transactions, which are those in the malicious transactions content list. The search would repeat this process for all affected transactions until no neighbours are found. While in the case of [30], the process requires more extensive looping to track the set, since the read/write relationship is represented in the opposite way. The search starting from a malicious transaction knows the transactions that the last read from, so to get the affected ones, the algorithm searches within all the transactions to check if the malicious transaction is present in their content list, making the looping more expensive.

The recovery after the search is identical in the two approaches. It happens first by deleting the effects of the malicious transactions, in addition to restoring the affected transactions, either by re-executing their operations, or by restoring the before images. Our algorithm integrates the algorithm in [7] adapted to a multilevel approach, together with a clustering technique to group the transactions by their appropriate levels.

4.2 Adaptation To Multilevel

As preliminary to our approach, we consider that the history in our system is rigorously serializable, and the system also spawns transactions by plus one incremental IDs. The rigorous property guarantees strictness and ensures that no read data item is overwritten until the previous transaction carrying the read is committed or aborted [31]. This allows the scheduling of the transactions in the system in a sort of sequential manner where transactions with a higher ID would only interact with lower ID transactions.

This helps our algorithm in the assessment and detection phases. In the first phase, while reading from the log file to track the dependencies, starting from a transaction with ID x we can safely check for its interactions with other transactions with ID y where $y < x$, since transactions can only read/write from transactions with lower IDs. The rigorous scheduling enhances the performance in the second phase too. The search for affected transactions after reporting the malicious ones would require only a search starting from the malicious transaction ID going upwards in the IDs, following the same property described before.

The system respects polyinstantiation by allowing multiple tuples in the database with the same primary key, but with explicit level assigned. We ensure the visible and invisible polyinstantiation properties explained in chapter II, by restricting reads on the specified or lower security levels. If the clearance dominates the security level and the data item to be read was found, the read would be accomplished successfully on that specified level, else the system would search through the lower security levels sequentially to find the data item. The writes in our system are recorded on the same level only. Hence, a transaction would write with a clearance level that is equal to the corresponding security level, which it is attempting to access.

The system classifies the transactions on the record level, since we assume that a transaction would insert on one level only in a query. Formally, a transaction performing an insertion with a certain security level would write all its data items at that same security level. For a correct execution of the assessment, the algorithm needs to bypass the simple and * security properties in the multilevel databases, since the damage detection in the system should be across all the levels of the database. The writes clusters are then subject to be checked regardless the level of the transaction in question, to ensure that our algorithm is tracking all the dependencies. Thus, we would give our algorithm the highest security level access on the database to ensure its correctness.

Our approach assumes the existence of four security levels: UNCLASSIFIED (U), CONFIDENTIAL (C), SECRET (S) and TOP SECRET (TS). We record the writes

of transactions read from the log on the fly in four buckets corresponding to the level of each transaction, U, C, S and TS. These writes will serve as the sufficient reference to record transaction dependencies. The algorithm does not require passing through the log file more than once, since the buckets will be used in their turn to check all read/write interactions in the system. Each bucket will be represented as a bipartite graph, since data items are stored from one side and the transactions are stored from the other side. By definition, a bipartite graph is constituted from two sets of nodes, where no nodes in the same set are adjacent.

One data item would have multiple transactions overwriting it, but we can safely sort the transactions by ascending IDs to represent the last written one, since our history is rigorous. Formally a data item x, represented by a node, would have an edge to the node $T(n)$ in bucket i represented by; $x \rightarrow T(n)$, if $T(n)_i : w(x)$, where n is the primary key of the modified record. We record the primary key to identify the modified record to catch the dependencies by the primary key - foreign key relationships in the next stage. Another recorded element is the table name of the modified record, and this is essential to locate the change when we need to restore the database. We record also with each write in the bucket the before image (previous value). This would speed up the recovery time since even the operations to be rolled back are saved in memory. Hence, every node of a transaction would be constituted of its ID, the primary key of the modified record, its relevant table name, and the before image.

The second phase for the damage assessment is for recording the dependencies in our adjacency list. A transaction executing in the system and after recording its writes in the buckets, if it read from a certain level then wrote, the algorithm will check for the last write on that data item in the corresponding bucket following the primary key - foreign key relationship on the record itself. It will then record a dependency with the last transaction carrying the write if the relationship was found. Formally, if $T(n)_j : r(x)_i w(y)_j$ where $i < j$, and bucket i contains $x \rightarrow T(m)$, we record dependency $T(m) \rightarrow T(n)$; where m and n are the relative primary key and foreign key. This process

and the buckets write are described in table 3.

Table 3: The damage assessment process

Input: Set of transactions S: $T_0 \ldots T_n$
Output: A graph G, $G = (V, E)$ where each E represents a transaction dependency

for each $T(n)$ in S **do**
  if $T(n)$ contains $W_i(x)$ **then**
    Record $x \rightarrow T(n)$ in bucket i
  end if
  if $T(n)$ contains $r(x)_i w(y)_j$ && bucket i contains $x \rightarrow T(m)$ **then**
    Record $T(m) \rightarrow T(n)$ in G
  end if
end for

After fully clustering the log file into buckets of writes, and recording all the dependencies in the adjacency list, the detection of affected transactions and their recovery is now simple. The detection starts when the IDS reports the malicious transactions in the system. The algorithm requires a search over the adjacency list to track the affected transactions. We choose this search to be breadth first. The algorithm is illustrated in table 4, it mainly uses a queue to start with the first malicious transactions, then each neighbour visited is considered affected and entered into the queue, until the queue is empty, so all neighbours are visited. The recovery following this step, and since the before images are recorded in our system, requires only a traversal on the buckets to capture the writes and the before images of the affected transactions. The before images are then used to restore the database to its stable state.

The damage recovery process needs to ensure that all the effects of the malicious transactions are invalidated. The lasting effects of an affected transaction are its writes in the database. The lasts are considered invalid since they might be based on the read of a malicious transaction. Hence, the content of these writes is malicious by itself. To recover the database we illustrate in table 5 the process that should be executed. We are given the set of the affected transactions from the damage detection process. The first step is by looping on all these transactions and buckets, to find all the writes

Table 4: The damage detection process

Input: Set of malicious transactions S: $T_0 \ldots T_n$, empty queue Q, graph G of all transaction dependencies.
Output: A set of all affected transactions S'

for each $T(n)$ in S **do**
  Add $T(n)$ to Q
  while Q ! $= \emptyset$ **do**
    $T(m)$ = Pop Q
    Add $T(m)$ to S'
    Add neighbours of $T(m)$ to Q
  end while
end for

that an affected transaction performed on all the levels in the system. And since our algorithm bypasses the security constraints of the multilevel database, we can safely identify all the transactions' writes and their relevant before images and primary keys. The last step in the process, is simply by re-executing an update on the corresponding affected table where the primary key is equal to the transaction's primary key, and the new value is the before image.

Table 5: The damage recovery process

Input: Set of affected transactions S: $T_0 \ldots T_n$, Set of write buckets B: $B_0 \ldots B_n$
Output: None

for each $T(n)$ in S **do**
  for each $B(n)$ in B **do**
    if $B(n)$ contains $T(n)$ **then**
      Extract before image, primary key, and table name of $B(n)$
      Update table with before image WHERE primary key = $T(n)$'s key
    end if
  end for
end for

To further explain how our data structures will be built, we illustrate the above mentioned algorithms in a real-life scenario. The system is a federal tracking inventory, where information of federal workers is tracked. The records in the system follow the schema shown in table 6. *pk* signifies the primary key in the record, and fk is a foreign key.

Table 6: Our database schema

- **Profession:** profession-id(pk), title, min-years, degree.
- **Site:** site-id(pk), location, name, phone.
- **Department:** department-id(pk), address, phone, name.
- **Worker:** worker-id(pk), name, DOB, address, phone, profession-id(fk), department-id(fk).
- **Job:** job-ref(pk), title, start-date, end-date, worker-id(fk), site-id(fk).

The previously mentioned rules of simple and * security properties are respected in the scenario. The reads are performed on the same level of the the transaction if the data item is found, else they traverse the levels down incrementally to find the appropriate data item. The writes are executed on the same level only. Polyinstantiation is respected in the visible and invisible form, but our notion is simplified since we are dealing with records classifications. All the record follows a single security level. Hence, if a transaction with level U is executed, all its writes will be performed on that level only. The execution is in the form of queries on the database, where all the queries would have reads and writes operations. We would track the dependencies from the relative primary key - foreign key relationships, since each query is reading from another record and writing a certain value. The following transactions are injected in the system:

T_0U : Profession(“1000”, “Mathematics - Algebra”, “3”, “Bachelor”)

T_1S : Department(“2001”, “Beirut”, “06-0191910”, “Science”)

T_2S : Worker(“4109”, “Mari Litz”, “22-04-1990”, “Hamra”, “03-1391391”, “1000”, “2001”)

T_3U : Worker(“4115”, “Elenor Sheperd”, “22-04-1990”, “Zalka”, “03-311122”, “1000”, “2021”)

T_4U : Site(“3162”, “Lummis”, “Main Site”, “70-1939102”)

T_5U : Job(“5074”, “Teacher”, “10-02-2000”, “10-02-2005”, “4115”, “3162”)

T_6S : Job(“5074”, “Machine Learning Researcher”, “10-02-2000”, “10-02-2005” ,“4115”,

"3162")

In figure 2, we illustrate the writes of the above transactions, tracking the writes of each transaction on the primary keys. Our bipartite graph is represented at two sides, where the left side represents the writes on the unclassified level and the right is on the secret level. Tracking the writes from the primary key - foreign key relationships as previously described we represent the transaction dependencies also in an adjacency list as shown in figure 3.

We start the execution by injecting the transactions in order. T_0 performs a write on the Profession table. The field $T_0(1000)$ is recorded in bucket U of writes on the profession-id. We have no foreign key reads in this transaction. The second transaction T_1 performed a write on the table Department on the secret level. Hence we record $T_1(2001)$ on the department-id in the S bucket. The first transaction to record a dependency is T_2, it performs a write on the worker table on the secret level, it reads from the foreign key 1000 of the profession table. So far the only satisfied relationship is with the transaction T_0 on the unclassified level. Respecting the multilevel security property, the algorithm can safely record for T_0 a dependency with T_2. The second dependency is on the secret level, T_2 read from T_1 on the secret level. T_3 is executed on the unclassified level, the only relationship it has is with the worker-id 1000. Thus, the dependency T_0 is recorded with T_3. The transaction T_4 performs a simple write only, it is recorded on site-id in bucket U as $T_4(3162)$. The last two transactions illustrate the nature of polyinstantiation, they both have the same primary key, but are executed on different levels. T_5 was executed on the unclassified level, it read from both T_3 and T_4 on the unclassified level. Hence, T_5 is recorded as a dependency for both T_3 and T_4. T_6 has the same primary key of T_5 but was executed on the secret level. It is accepted in the system, and the primary key $T_6(5074)$ is recorded in the bucket S. It has read also from the writes of T_3 and T_4; therefore, the dependency of T_6 with those two is recorded. The dependencies are satisfied since the reads of T_6 were on the secret level, and the targeted data items in the secret level were not found. The system searched

through the lower levels and found the appropriate data items on the unclassified level.

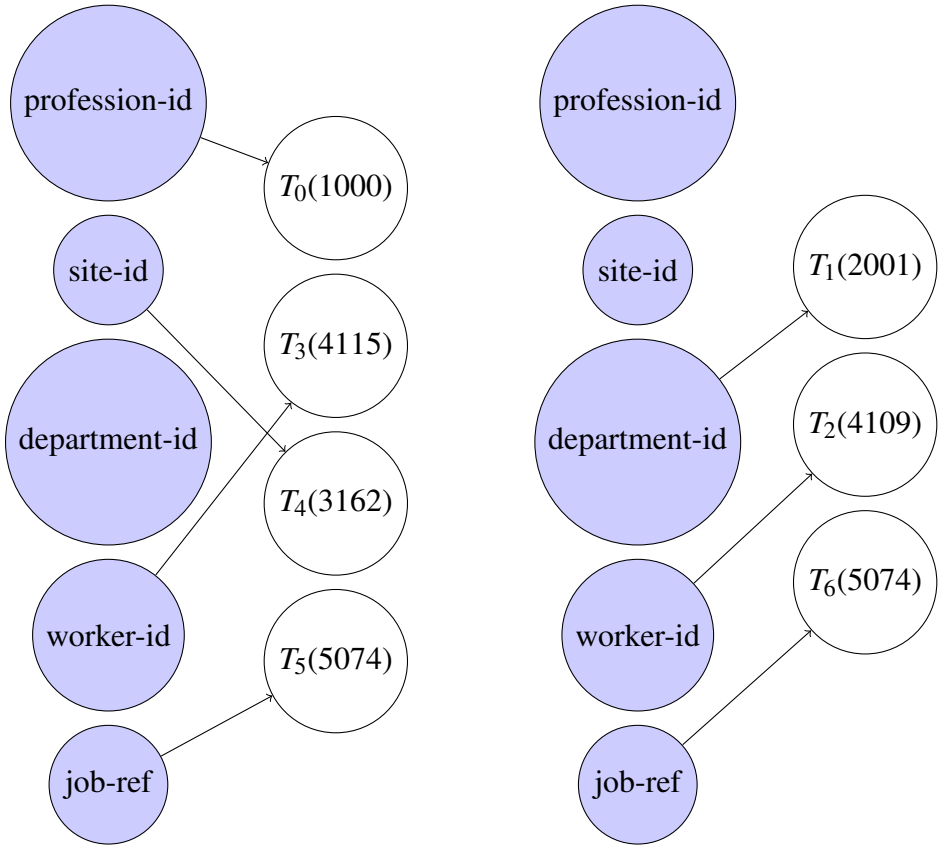


Figure 2: Bipartite graph representing the writes on the unclassified level from the left and on the secret level from the right.

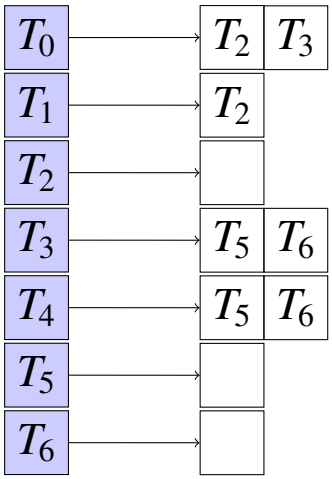


Figure 3: Adjacency list illustrating the transactions dependencies.

4.3 Complexity Analysis

We explained in the previous section three algorithms that embody our approach. We are interested in analyzing the complexity of these algorithms to ensure that all their running time is polynomial. This is essential in the work, since by definition, polynomial time complexity means that the algorithm's running time is upper bounded by a polynomial expression of its input: $T(n) = O(n^k)$ for some constant k. This ascertains that our running time is not growing exponentially depending on the size of the input.

The damage assessment algorithm presented in table 3 is composed of two steps. Recording the writes in the bucket is the first one, it can be performed in $O(1)$ since it is a constant time operation. The second step, the recording of the dependency is executed in $O(n^2)$. It loops on the transactions in the log, and for each transaction that performs a read/write, it checks in the buckets if the read data item is present. That is two nested loops, which explains the order of n^2 for the operation.

The damage detection algorithm explained in table 4 is the implementation of the breadth first search on the adjacency list recorded in the detection stage. BFS is known to perform in $O(n \log_n)$, since it uses the queue to visit the neighbours in the graph in a breadth search manner. The algorithm loops over the malicious transactions, then uses the queue to push all their neighbours, where for each element in the queue its neighbours will be visited in their turn. All these visited neighbours plus the initial set of malicious transactions should be recovered in the next step.

The last discussed algorithm is the recovery in table 5. The recovery happens after retrieving all the affected transactions. The algorithm loops once over all the transactions that are received. The second loop is on the buckets to extract the appropriate fields for the rollback. We extract the primary key, the table and the before image. These two nested loops indicate that the running time of the recovery is in the order of $O(n^2)$. These three processes are run sequentially in the system. Each process depends on the previous one, but checkpoints can be added when each process is completed.

Their running time is shown to be polynomial, and the worst one was bounded by n^2 operations. We can assure then, that our damage assessment and recovery approach is efficient. Notably since the log will not span for more than thousands of transactions in the system.

Chapter Five

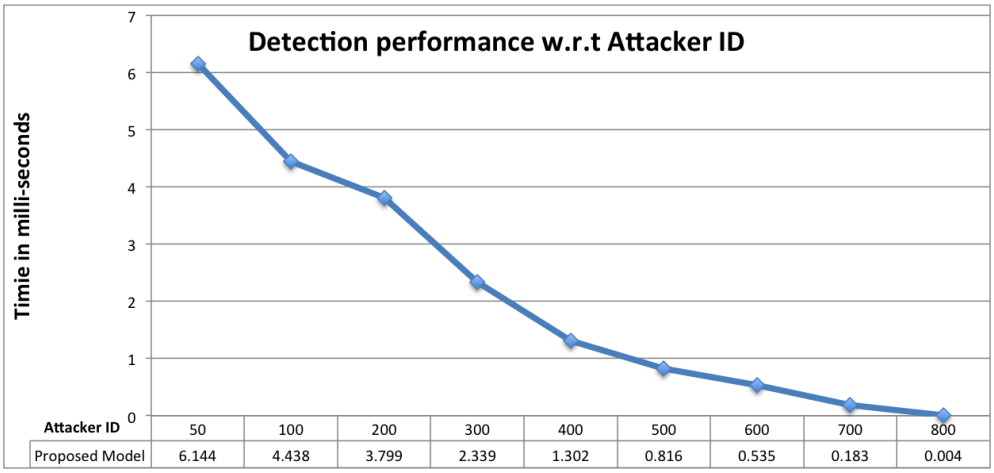
Experimental Results

As we have seen in the previous chapter, our approach runs by detecting affected transactions sequentially from the malicious transactions themselves. It then recovers from the attack by reverting the work of these transactions on the database. This is due the nature of our rigorously serializable history. The experiment is run sequentially on multiple cases, where we specify the first affected transaction ID, and scan our data structure to track all affected transactions from this point. This shows a concrete relation between performance and number of affected transactions in the system.

Our approach was simulated on generated log files that represent multiple cases in the system. A random scenario where transactions are randomly dependent on each other, a worst case scenario where one transaction is dependent on all other transactions, and a best case scenario where each transaction is dependent on another one only. The simulation was done on a sqlite database with the schema specified in table 6. The system is a personal computer with a CPU of 2.6GHz Intel Core i5, and 8GB RAM. The code was written in Python 3.4.3 where the integration with the sqlite database is native using $sqlite3$ library.

5.1 Performance Results of the Detection Algorithm

To start with the detection algorithm and taking into consideration the above mentioned requirements of our model, we need to prove the correctness of the detection phase. Following our history properties, scanning from the list of affected transactions should start from the first next transaction in the system after the attacker ID. In figure 4 we show that our algorithm is running proportionally to the ID of the smallest affected transaction, since the portions of the needed transactions to be scanned decrease with the increasing number of the smallest malicious ID, and we know that our history disallows the reads from a higher transaction ID.

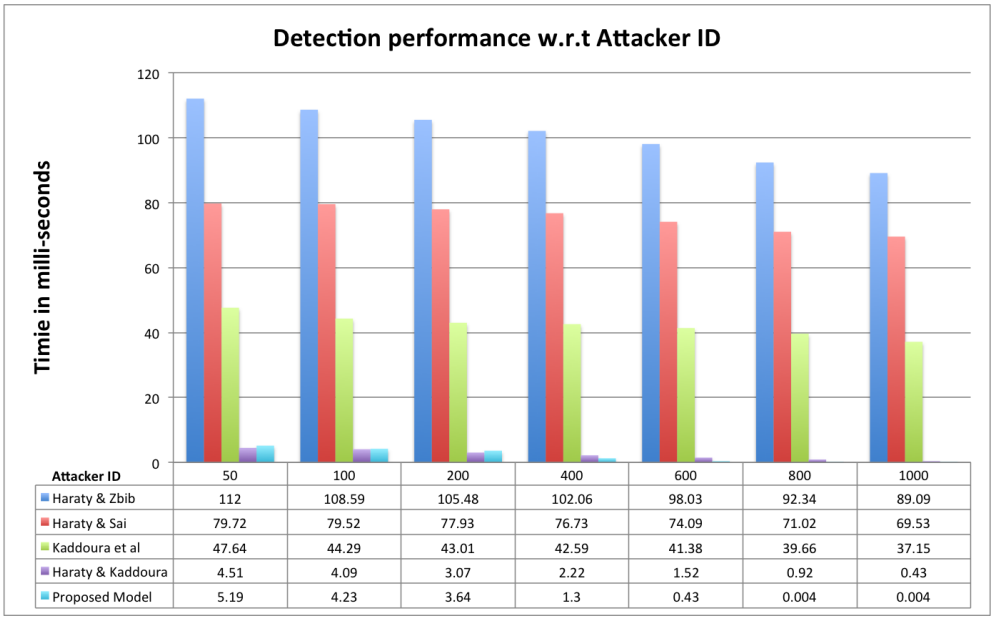


Attacker ID	50	100	200	300	400	500	600	700	800
Proposed Model	6.144	4.438	3.799	2.339	1.302	0.816	0.535	0.183	0.004

Figure 4: Performance of the detection algorithm

The detection algorithm was tested on a log of 1000 transactions in the system, where the dependencies are ad-hoc and random. We compare, in order, the performance of our detection algorithm to other recent algorithms. In figure 5 we see that the time taken in [26] by Haraty and Zbib is by far the highest, since it is an approach where two matrices are used to track the dependencies, it needs more time to detect the affected transactions. In [30], Haraty and Sai reach a better result since now the tracking is based on one matrix. In [29], Kaddoura et al. implemented a data dependency approach but it uses smart indexing to tag the dependent transactions. Hence, the time was even better than the two previous mentioned transaction dependencies approaches. The approach in [7] is really similar to the one in [30], but Haraty and Kaddoura en-

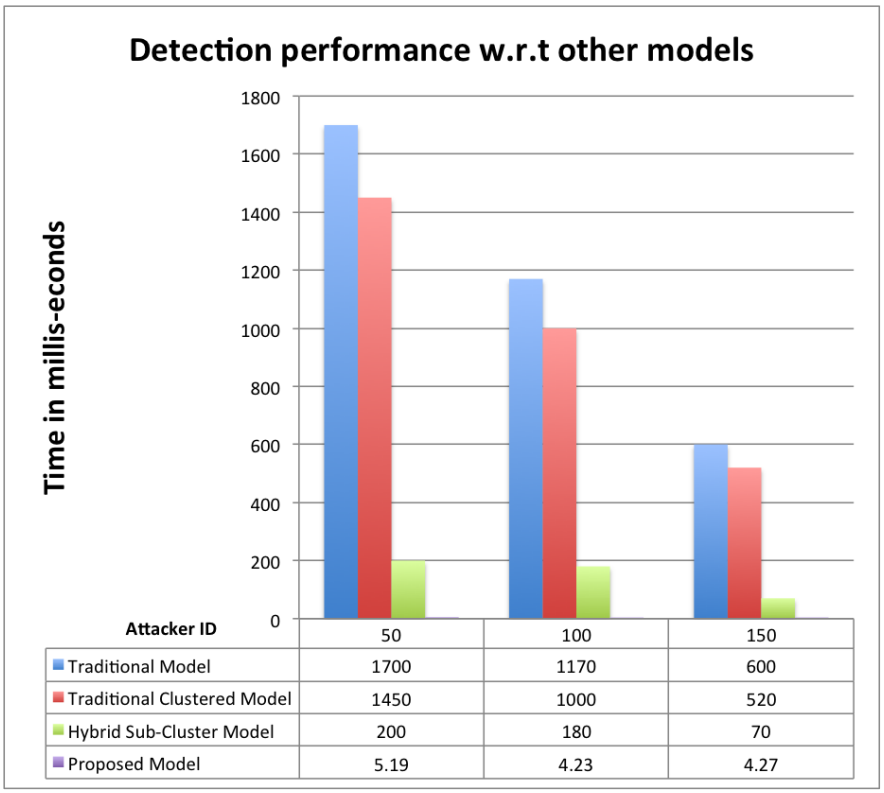
hanced the performance by getting the list of affected transactions in one run, it is only sufficient to retrieve the list of affected transactions by retrieving the head of the adjacency list that represents the malicious one. Our approach proved to have similar running time to [7], since it is identical when it comes to detection.



Attacker ID	50	100	200	400	600	800	1000
Haraty & Zbib	112	108.59	105.48	102.06	98.03	92.34	89.09
Haraty & Sai	79.72	79.52	77.93	76.73	74.09	71.02	69.53
Kaddoura et al	47.64	44.29	43.01	42.59	41.38	39.66	37.15
Haraty & Kaddoura	4.51	4.09	3.07	2.22	1.52	0.92	0.43
Proposed Model	5.19	4.23	3.64	1.3	0.43	0.004	0.004

Figure 5: Performance of the detection in comparison with other recent models

Additionally, we compared our approach with other non-matrix approaches as shown in figure 6. The classical approach being the one presented by Bai and Lui in [16], and the clustering approaches were all proposed in [21]. We notice from the beginning that the matrix approach is by far more dominant in performance than these other approaches. The matrix in our model stores needed information only to describe transaction dependencies, enhancing the running time in detection phase. The deterioration in all these models is the fact that they need to pass through logs multiple times, requiring expensive I/O operations, to track all dependent transactions. The other advantage in our model is that scanning begins from the first smallest transaction ID after the attacker ID, which is not the case in these other algorithms.



Attacker ID	50	100	150
Traditional Model	1700	1170	600
Traditional Clustered Model	1450	1000	520
Hybrid Sub-Cluster Model	200	180	70
Proposed Model	5.19	4.23	4.27

Figure 6: Performance of the detection in comparison with classical models

5.2 Performance Results of the Recovery Algorithm

In regards to the recovery, our algorithm running time should be increasing linearly to the number of affected transactions. Checking figure 7 we can see that the running time is increasing with a factor of 3 to 6 milliseconds for each 50 to 100 transactions to be recovered. This proves that the recovery algorithm is steady in our system, since the running time is growing logically in proportion to the work that should be done to recover.

The recovery algorithm was also compared to the other classical approaches previously mentioned in the detection phase. We are interested in the differences with these approaches, since the recovery in all matrix approaches is similar. In figure 8 we compared our model to the traditional in [16], clustered and hybrid sub-clustered [21], on a same set of a log of 1000 transactions where the dependencies are random, starting from a malicious transaction ID. All these models fail to attain a better running time. We can attribute this failure due the difference in the strategy when it comes to repre-

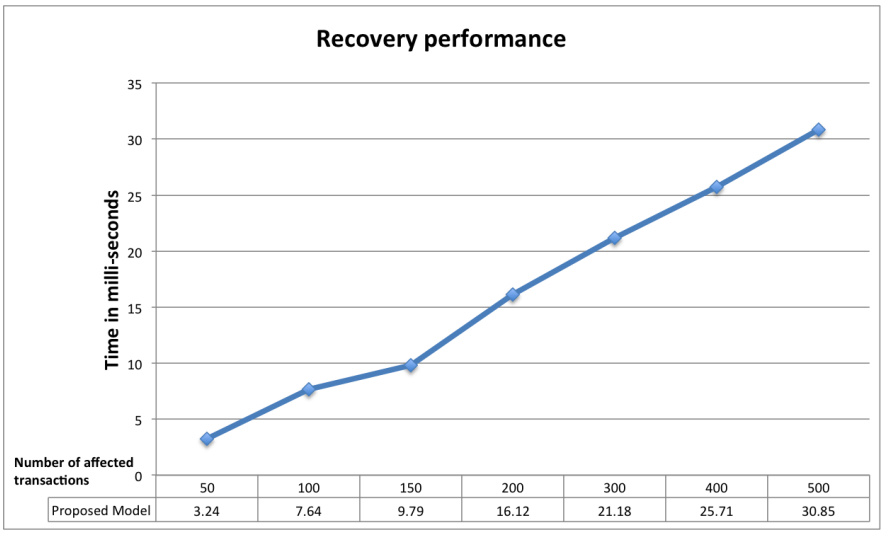


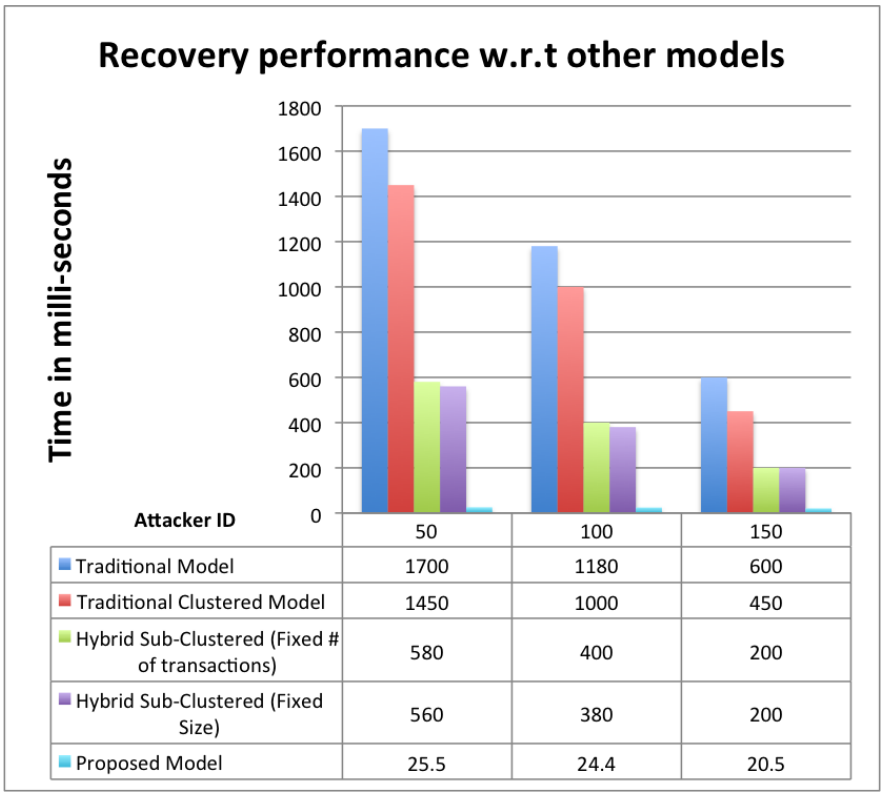
Figure 7: Performance of the recovery phase

senting the transactions. Our model pass through the log file once and saves in memory all the dependencies, represented in the adjacency list, and the multi-level writes in the in memory buckets. While the other models have to go through the log file multiple times to retrieve the transaction and its operation to recover it. It is a similar trade-off to the one in the detection phase, where they had to pass through the log files multiple times to track the dependencies, and here it is the case of tracking the transaction operations, nevertheless the price in memory in our model is not immense compared to the saved running time.

5.3 Memory Footprint Analysis

The memory analysis in the system is done from the point of the matrix creation and the writes buckets. We are merely interested in assessing the footprint when it comes to storing all these needed information in memory, since as we have seen in the previous sections, we are outperforming in running time some of the recent algorithms, that were done on single level databases.

These results come in the price of memory as seen in figure 9, the memory is tracked in bytes on a log of 1000 transactions where the dependencies are random. The footprint



Attacker ID	50	100	150
Traditional Model	1700	1180	600
Traditional Clustered Model	1450	1000	450
Hybrid Sub-Clustered (Fixed # of transactions)	580	400	200
Hybrid Sub-Clustered (Fixed Size)	560	380	200
Proposed Model	25.5	24.4	20.5

Figure 8: Performance of the recovery in comparison with classical models

in [26] is one of the worse, since Haraty and Zbib were using here two matrices to track dependencies, as discussed before. It is followed by the approach of [29], since it is a data dependency approach it needs to store a lot of data items, resulting with that increase in memory footprint. Haraty and Sai in [30] reach the best result in storing just the dependency matrix. While their counterpart in [7] also reaches a great result but the difference is due the randomness of transaction dependencies in the log file. Our proposed model is heavy in memory since we are handling a multilevel database, we are storing beside the transaction IDs in the dependency matrix all the writes in buckets corresponding the levels in the system.

The memory footprint is also tested in a best case and worst case scenario with the models of [26] and [30]. The best case scenario occurs when each transaction is dependent to only one another transaction, and the worst case represents each transaction as dependent on the rest of the transactions in the system. Both scenarios are executed on a log of 1000 transactions, represented in the figures 10 and 11, where we test the number of memory slots occupied for a certain number of dependent transactions.

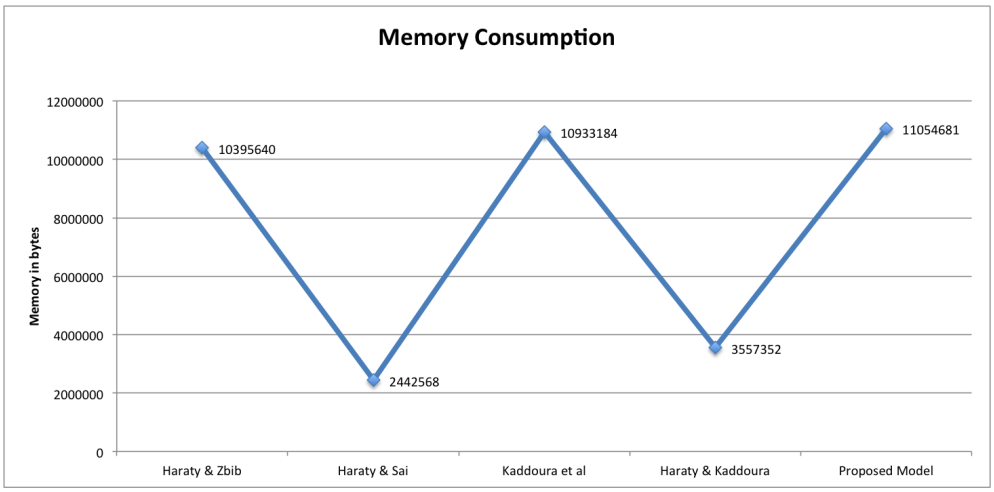


Figure 9: Memory consumption in comparison with recent models

In the best case scenario and since it is a one to one relationship, we notice that Sai and Haraty model always occupies a number of memory slots equal to the half of the number of dependent transactions, and that is true, since their approach records only the dependent transactions ID in the system. The approach of Zbib and Haraty, records a fixed number of 1000000 slots, which is the number of the transactions in the log squared. Their approach requires a fixed number of slots related to the number of transactions in log and not the dependencies. Our approach is similar to the approach of Haraty and Sai, the memory slots for the dependency matrix can be calculated the same way, but the bulk of the memory is the saved writes in the system, which does not take into consideration the number of dependent transactions. Hence, the number shows to be increasing by the same factor of dependent transactions in the Haraty and Sai approach, while the main number of slots is the saved writes.

The worst case scenario is a one to many relationship, since one transaction is dependent on all the rest in the system. Haraty and Sai model outperforms the others, the number of slots required to satisfy the number of dependent transactions is the same number of transactions minus one. Again, this is due the efficient adjacency list structure that tracks the dependent transactions only. In contrast, Haraty and Zbib's model shows the same flaw, which is the fixed number of slots for the two matrices that is 1000000, adding to that the extra dependency requirement. Our model is progressing

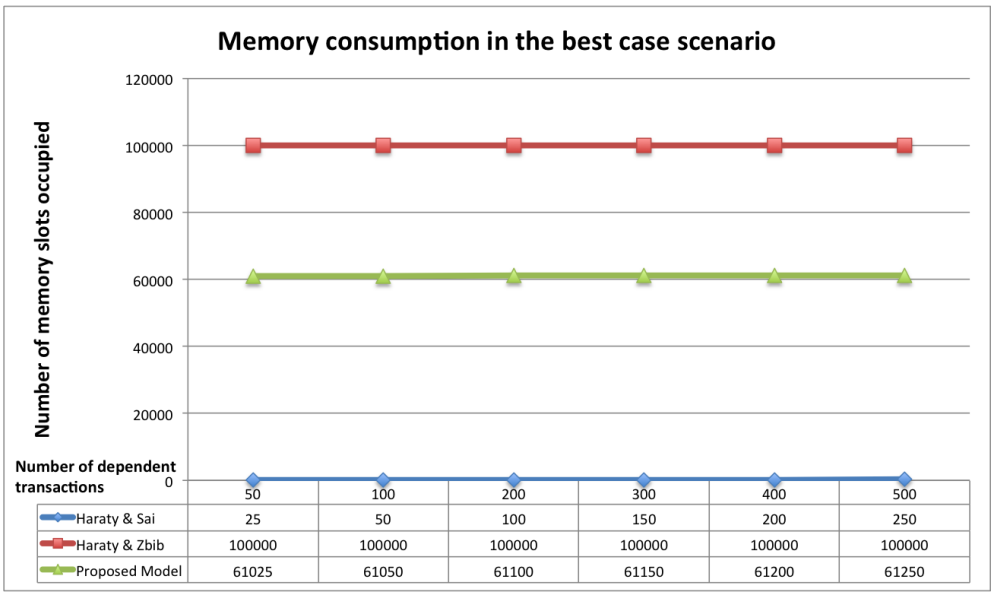


Figure 10: Memory consumption in the best case scenario

the same way it did in the best case scenario, since the increase in slots is directly related to the increase in the number of dependent transactions. The calculation of the needed slots is identical to Haraty and Sai's calculation. Yet our main drawback is the memory needed for the writes, which is a big constant factor regardless the dependencies in the system.

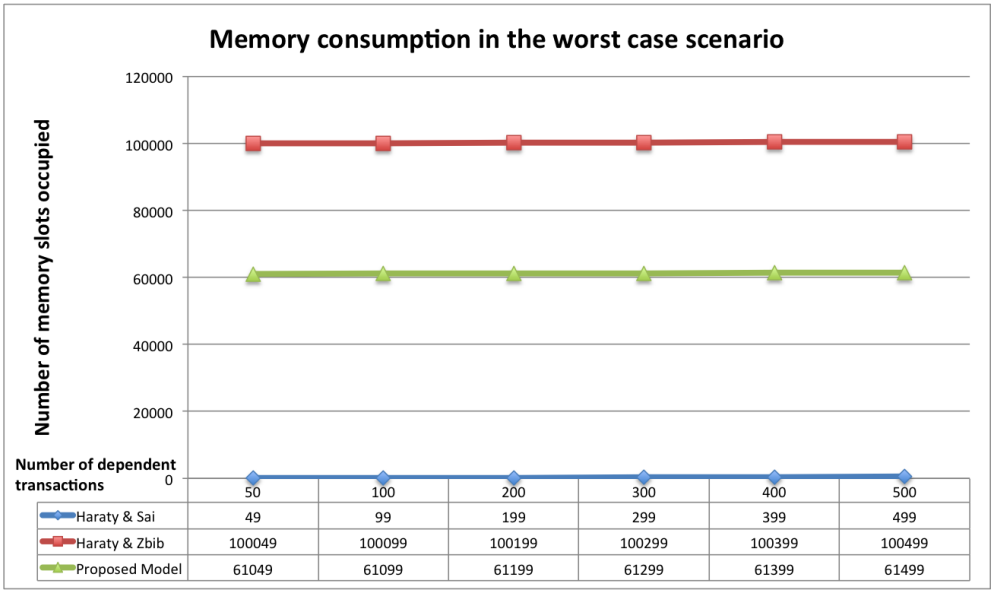


Figure 11: Memory consumption in the worst case scenario

Chapter Six

Conclusion

The three pillars to defend in the information warfare are prevention, detection and recovery. Prevention and as it was described by imposing barriers in the system from firewalls to anti-viruses can certainly achieve in preventing many attacks, but still even the most secure system can be breached. Detection is then essential, it is usually carried by an intrusion detection system following historical studies. The system would catch the intruder work that is in our case the malicious transactions. Hence, recovery is a complementary aspect to continue the work of the detection.

Recovery is the last line of defense in the system. The work that should be done to recover from the affected transactions should be quick, efficient, and ultimately prevent the events of denial of service. The requirements for such a work in multi-level databases are different from the traditional proposed approaches. The algorithm must comprehend the security levels to clearance relationship, polyinstantiation and the structure of data.

We proposed an algorithm that has proved to be a contender in the damage assessment and recovery with the best of the approaches proposed for single level database systems. Our running time for the detection of the malicious transactions was on par with the best single level approach to date. The only drawback in our system is the consumption of memory, but we think that is acceptable since it did not exceed the consumption by a high margin in comparison with the work that incorporated matri-

ces, taking into consideration that the algorithm is saving before images, and handling a multilevel environment.

Our algorithm managed to track the dependencies, the writes, and the before images of the data items correctly. We can recover the system with high accuracy and speed, since all the needed information is saved in memory. As for future our work we plan on experimenting with other approaches to ease the memory overhead, and reach an efficient quick recovery. We can try by flushing to another backup database the present information in memory, lazy load the needed data while recovering or even dropping our bucket clustering approach for another alternative.

References

[1] Philip A Bernstein, Vassos Hadzilacos, and Nathan Goodman. Concurrency control and recovery in database systems. 1987.

[2] Nilesh Khochare, Satish Chalurkar, Santosh Kakade, and BB Meshramm. Survey on sql injection attacks and their countermeasures. *International Journal of Computational Engineering & Management (IJCEM)*, 14, 2011.

[3] Teresa F Lunt. A survey of intrusion detection techniques. *Computers & Security*, 12(4):405–418, 1993.

[4] Peng Ning and Sushil Jajodia. Intrusion detection techniques. *The Internet Encyclopedia*, 2003.

[5] D Hua, Q Xiaolin, Z Guineng, and L Ziyue. Sqrm: An effective solution to suspicious users in database. In *Proc. The Third International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA)*, 2011.

[6] William Hutchinson and Matthew Warren. *Information warfare*. Routledge, 2012.

[7] Ramzi Ahmed Haraty, Sanaa Kaddoura, and Ahmed Zekri. Transaction dependency based approach for database damage assessment using a matrix. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 13(2):74–86, 2017.

[8] Paul Ammann, Sushil Jajodia, and Peng Liu. Recovery from malicious transactions. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1167–1185, 2002.

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

[10] Dominic Damoah, James B Hayfron-Acquah, Shamo Sebastian, Edward Ansong, Brighter Agyemang, and Roy Villafane. Transaction recovery in federated distributed database systems. In *Computer Communication and Systems, 2014 International Conference on*, pages 116–123. IEEE, 2014.

[11] Ramzi A Haraty and Natalie Bekaii. Towards a temporal multilevel secure database (tmsdb). In *Proceedings Journal of Computer Science*, volume 2, pages 19–28, 2006.

[12] Dorothy E Denning, Selim G Akl, Matthew Morgenstern, Peter G Neumann, Roger R Schell, and Mark Heckman. Views for multilevel database security. In *Security and Privacy, 1986 IEEE Symposium on*, pages 156–156. IEEE, 1986.

[13] Sushil Jajodia and Ravi Sandhu. Toward a multilevel secure relational data model. *ACM SIGMOD Record*, 20(2):50–59, 1991.

[14] Andro Galinovic and Vlatka Antoncic. Polyinstantiation in relational databases with multilevel security. In *Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on*, pages 127–132. IEEE, 2007.

[15] Brajendra Panda and Kazi Asharful Haque. Extended data dependency approach: a robust way of rebuilding database. In *Proceedings of the 2002 ACM symposium on applied computing*, pages 446–452. ACM, 2002.

[16] Kun Bai and Peng Liu. A data damage tracking quarantine and recovery (dtqr) scheme for mission-critical database systems. In *Proceedings of the 12th Inter-*

national Conference on Extending Database Technology: Advances in Database Technology, pages 720–731. ACM, 2009.

[17] Brajendra Panda and Joseph Giordano. Reconstructing the database after electronic attacks. In *Database Security XII*, pages 143–156. Springer, 1999.

[18] Yanjun Zuo and Brajendra Panda. Fuzzy dependency and its applications in damage assessment and recovery. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 350–357. IEEE, 2004.

[19] Peng Liu and Meng Yu. Damage assessment and repair in attack resilient distributed database systems. *Computer Standards & Interfaces*, 33(1):96–107, 2011.

[20] Prahalad Ragothaman and Brajendra Panda. Analyzing transaction logs for effective damage assessment. In *Research directions in data and applications security*, pages 89–101. Springer, 2003.

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

[22] Chandana Lala and Brajendra Panda. 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, 2001.

[23] Meiyi Xie, Hong Zhu, Yucai Feng, and Guanrong Hu. Tracking and repairing damaged databases using before image table. In *Frontier of Computer Science and Technology, 2008. FCST'08. Japan-China Joint Workshop on*, pages 36–41. IEEE, 2008.

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

[25] Kranthi Kurra, Brajendra Panda, Wing-Ning Li, and Yi Hu. An agent based approach to perform damage assessment and recovery efficiently after a cyber attack to ensure e-government database security. In *System Sciences (HICSS), 2015 48th Hawaii International Conference on*, pages 2272–2279. IEEE, 2015.

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

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

[28] Jing Zhou, Brajendra Panda, and Yi Hu. Succinct and fast accessible data structures for database damage assessment. In *ICDCIT*, pages 420–429. Springer, 2004.

[29] Ramzi A Haraty, Sanaa Kaddoura, and Ahmed Zekri. Information warfare: Fighting back through the matrix. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 449–454. IEEE, 2015.

[30] Ramzi A Haraty and Mohamed El Sai. Information warfare: a lightweight matrix-based approach for database recovery. *Knowledge and Information Systems*, 50(1):287–313, 2017.

[31] Yuri Breitbart, Dimitrios Georgakopoulos, Marek Rusinkiewicz, and Abraham Silberschatz. On rigorous transaction scheduling. *IEEE Transactions on Software Engineering*, 17(9):954–960, 1991.