5-2012

# Insider Threat Mitigation Models Based on Thresholds and Dependencies

Harini Ragavan
*University of Arkansas, Fayetteville*

# INSIDER THREAT MITIGATION MODELS BASED ON THRESHOLDS AND DEPENDENCIES

# INSIDER THREAT MITIGATION MODELS BASED ON THRESHOLDS AND DEPENDENCIES

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

By

Harini Ragavan
Government College of Technology
Bachelor of Technology, Information Technology, 2010

May 2012
University of Arkansas

## ABSTRACT

Insider threat causes great damage to data in any organization and is considered a serious issue. In spite of the presence of threat prevention mechanisms, sophisticated insiders still continue to attack a database with new techniques. One such technique which remains an advantage for insiders to attack databases is the dependency relationship among data items. This thesis investigates the ways by which an authorized insider detects dependencies in order to perform malicious write operations. The goal is to monitor malicious write operations performed by an insider by taking advantage of dependencies. A term called ‘threshold’ is associated with every data item, which defines the limit and constraints to which changes could be made to a data item by a write operation. Having threshold as the key factor, the thesis proposes two different attack prevention systems which involve log and dependency graphs that aid in monitoring malicious activities and ultimately secure the data items in a database. The proposed systems continuously monitors all the data items to prevent malicious operations, but the priority is to secure the most sensitive data items first, since any damage to them can hinder the functions of critical applications that use the database. By prioritizing the data items, delay in the transaction execution time is reduced in addition to mitigating insider threats arising from write operations. The developed algorithms have been implemented on a simulated database and the results show that the models mitigate insider threats arising from write operations effectively.

This thesis is approved for recommendation
to the Graduate Council.

Thesis Director:

_____

Dr. Brajendra Panda

Thesis Committee:

_____

Dr. Gordon M. Beavers

_____

Dr. Dale R. Thompson

**THESIS DUPLICATION RELEASE**

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

Agreed _____
HARINI RAGAVAN

Refused _____
HARINI RAGAVAN

## ACKNOWLEDGEMENTS

It is my honor to express my deep and sincere gratitude to my advisor, Dr. Brajendra Panda. His wide knowledge, logical thinking and encouragement have been of great help throughout my Master's program. I am also grateful to my committee member, Dr. Gordon M. Beavers, for his valuable guidance, constructive comments and for his support throughout this work. I owe my deepest gratitude to Dr. Dale R. Thompson, for being in my Master's committee.

I would like to extend my sincere gratitude to my brother for all the support and encouragement. Finally, I would like to thank all my friends at the University of Arkansas for giving me a home away from home.

**TABLE OF CONTENTS**

## LIST OF FIGURES

# 1 INTRODUCTION

Technology is rapidly advancing every day and as a result organizations are shifting from the use of paper based documents to online digital documents. These provide ease of access, and thus computer systems are increasingly used to store, process and distribute information regarding an organization's day to day activities. It is therefore essential for every organization to maintain high levels of security since failure of security could lead to unauthorized disclosure, modification, or interruption of information. Research and Cyber Security studies show that among the threats that arise in an organization, insider threats are significant. Even though attacks such as hacking, viruses etc. arise from the outside and cause heavy damage, insiders pose a significantly higher level of risk than outsiders do [1].

Threats from outsiders are classified as "external threats", where an outsider attempts to compromise or gain access to the systems. There are many approaches that automatically detect external threats, but very few exist for the prevention of insider threats. Insider threats cause irreversible financial and security damages to an organization. Insider threats are difficult to detect as the perpetrators are trusted persons which makes it very difficult to draw a clear line between legitimate and malicious actions. They have knowledge of the information systems they use and the services used in the organization. They also have knowledge about the security measures that have been taken to protect valuable information. Since they are aware of these measures and policies, they have the ability to violate or go around them. Thus certain attacks would go undetected for some time.

Investigations on security have discovered that a major proportion of the security breaches in an organization are caused by the employees. The professional services provider's Data Loss Barometer report has investigated and found out that the frequency of insider threats

has increased from 4% in 2007 to 21% in 2010 [2]. Also, CERT studies show that most of the insiders attacked the organization for financial gain. They get hired and motivated by outsiders to commit crimes in order to gain money. The average damages due to an insider attack as shown by the reports exceed $3M with few cases losing around $50M [3]. Thus it becomes essential for every organization to protect their data from insider attacks.

This thesis investigates the problem of insider threat in database systems. In spite of the existence of various threat prevention techniques, many databases are still being compromised by insiders. There are mechanisms which prevent insiders from attacking a database by gaining knowledge about the data items. They are prevented from accessing a combination of data items in a database through which they can gain knowledge about an important target data item. To complement those mechanisms, an attack prevention system has been proposed in this thesis which mainly deals with the threats posed by insiders who have authority to change and modify information in the database. Generally, insiders with write permissions can input any data into the database which at times becomes a serious threat for the organization.

The proposed attack prevention systems catch those malicious write operations by associating a term called ‘threshold’ with every data item in the database. Threshold defines the limit to which a data item could be modified and the values are dynamic so that they can be changed as and when necessary. One may argue that an insider with the knowledge of a threshold for a data item can modify the data item maliciously by keeping the changes within the threshold. However, the thresholds are determined in such a way that, any value within the threshold is within acceptable risk and causes no problem to the system. The efficiency of the models depends on the effectiveness of the threshold values and so they should be chosen in such a way that the data item remains secure.

The rest of the thesis is organized as follows. Chapter 2 addresses the background and related work. The terms introduced in the thesis are explained in Chapter 3. Chapters 4 and 5 contain the algorithms and models. Experiments and results are discussed in Chapter 6 followed by Chapter 7 which compares and contrasts the developed models. Chapter 8 presents conclusions and future work.

## 2 BACKGROUND AND RELATED WORK

Nowadays, firewalls and other network security mechanisms help organizations in preventing outsider threats. But, in spite of several intrusion detection systems, insider threat still remains a problem. Different researchers and scholars have put forward various definitions for the term 'insider', which is discussed in this chapter.

In [4], the authors define an insider as "*someone with legitimate access to an organization's computers and networks. For instance, an insider might be a contractor, auditor, ex-employee, temporary business partner, or more*". In [5], Bishop and Gates address an insider as "*a trusted entity that is given the power to violate one or more rules in a given security policy... the insider threat occurs when a trusted entity abuses that power*." In [6], an insider is defined as, "*Anyone with access, privilege, or knowledge of information systems and services*". Also, numerous methods to prevent insider threats have been introduced till date. In a paper by Spitzner [7], honeypots have been used to detect insider threat. He discusses the ways of indicating insider threats by combining honeypots with honeytokens and honeynets. Apart from this, various mechanisms like attack graphs and trees have been proposed in many papers. One such paper [8] uses attack trees as a framework to monitor the activities of users and also to catch them if their target node is found along the tree. Adding to this, in [9] use of attack decomposition trees and attack vs. defense matrices for insider threat defense is discussed.

[10] discusses how different orders of accessing data items poses different levels of threat. They propose threat mitigation models which organize the sequence of how data items could be accessed, so that threat becomes minimal. A paper by Yaseen and Panda [11] discusses how an unauthorized insider can acquire knowledge about a relational database by detecting the dependencies between objects. Sufficient amount of work has been performed in preventing

insiders, who build their knowledge by exploring dependencies in order to access sensitive information. The proposed model also deals with dependencies, but in contrast to previous work, it deals with malicious write operations. Similar work has been done in paper [12] which discusses methods to predict malicious activities by insiders who combine object dependencies and their knowledge gained in order to launch an attack. In [13], the authors proposed a model called a key challenge graph, which describes various paths an insider could take to reach the target node. They say that every time a node is compromised, additional information is gained which helps in continuing the attack.

Since dependencies remain to be a major issue in insider threat mitigation, many researchers have discussed it extensively in [14] [15] [16] [17]. In [14][15], the authors talk about the problem of combining non-sensitive data to derive sensitive data. They present a survey of the current and emerging research works in data inference controls. In [16] and [17], the authors investigate the problem of inference channels which arises when one combines non sensitive data to get sensitive information. An integrated architecture to detect various types of insiders was proposed by Maybury et al. in [18]. This paper summarizes few works that have been carried out for counter attacks on insiders. Bradford and Hu [19] combined intrusion detection mechanisms with forensics tools to detect insiders in a layered approach. They have employed intrusion detection systems to drive the forensic tools. Morgenstern [20] formulated a function to compute the amount of knowledge an insider could attain by detecting dependencies.

In [21], the authors advocate a model called the Capability Acquisition Graph (CAG) which analyzes the individual actions of users to evaluate their cumulative effects and detect possible violations. To complement these existing works, the proposed models aim at preventing

malicious write operations. The following Chapter describes the attack prevention system and explains few key terms used in the thesis.

# 3 THE ATTACK PREVENTION SYSTEM

Insiders are trusted entities of an organization who have the authority to perform various operations, and when they take advantage of their permissions and violate rules, it turns out to be a threat. As mentioned earlier, a good amount of work exists in preventing malicious read operations; so this thesis focuses on preventing malicious write operations in databases. A write operation can modify a data item by one of the following ways:

- Direct access
- Modifying some data item which will trigger an indirect update.[Indirect access]

## 3.1 Direct access

Here, an insider will have write permissions on the data item he/she is trying to modify. So as an authorized user, he will be trusted and he can make modifications directly on the object.

### 3.1.1 Online Shopping Example

Let us assume that someone orders an item online and money gets deducted from their account twice for the same item by mistake. To notify this, one might inform the concerned person and they will take necessary actions. The vendor who is responsible for this will have to manually deal with this situation.

### 3.1.2 Administrator Example

System administrators in a university would have permissions to reset passwords of students upon request. Here admins are trusted insiders since they can make direct modifications to the data item.

So, in cases like these where actions have to be directly taken, insiders will be given permissions and would be trusted. These are examples of insiders having direct access to data items.

### 3.2 Indirect access

In this case, an insider might not have direct permissions to modify the data item as such, but he can still work on it by figuring out its dependencies. This means, when the dependent data item is changed, it makes a change in the target item or might change few intermediate data items which get reflected in the target node. Thus, changing one value can produce a transitive change in the data items.

#### 3.2.1 DOB Example

A simple example to understand an indirect access would be to consider three attributes namely *Date of Birth (DOB)*, *Age* and *Vote*. *Vote* denotes if the person has the right to vote (varies depending on the country). When there is a change in the *DOB*, it gets reflected in the *age* as well as the *vote* column. This is an indirect change, e.g. change in one data item produces a change in the other one too. In cases like this, the insider may be prohibited from modifying *DOB.*

#### 3.2.2 Employee Database Example

Let us assume that in an employee database, salary is automatically calculated by multiplying number of hours an employee worked and pay per hour. If someone modifies the number of hours or pay per hour data, then it automatically triggers a change in the salary data. This is an indirect change, i.e. change in one data item produces a change in the other one too. This is a simple situation; however there may be complex situations where the dependencies may not be

so obvious but clever insiders take advantage of the relationships to attack the database However, there may be complex situations that may not make the dependencies so obvious and insiders can take advantage of that.

### 3.3 Working of the Models

With these classifications in mind, the attack prevention system has been modeled to forbid malicious writes specifically in two scenarios.

- Prevent malicious writes using log entries.
- Prevent malicious writes using dependency graphs.

A term called ‘threshold’ is introduced in the thesis which sets the limit to which a data item can be modified. Every data item in the database is associated with a ‘threshold’. When a change crosses the threshold, it signals a threat and a check is made verify the validity of the write operation. Generally logs maintain a record of all activities in a database and thus to verify the validity of an operation, one could analyze the log. There might be cases where one would need more information about the write operation in order to proceed with the investigation. So, to get a clear picture of the write operation, the log is thoroughly examined to determine the complete set of data items that were accessed during that entire transaction. By doing so, one can pull out the corrupt data items and fix the malicious operations.

Tracing the entire log to identify a malicious update would cause significant delay when there are numerous transactions getting recorded. So, an alternate approach which works faster by employing ‘dependency graphs’ is proposed as the next model in the thesis. Here, the process begins by developing a graph called the ‘dependency’ graph, whose nodes are various data objects of the database. An edge between two nodes means that there is a dependency

relationship between the two data items. This means that the write operation performed on one data item might affect the other in some way.

The system works by monitoring every write operation on nodes and when the changes go beyond the threshold, necessary security checks are performed. The main difference between log and the dependency graph is that, a log is a collection of all the transactions and it is tedious to determine the sequence of an operation. But, the dependency graph gets built as and when a transaction proceeds and it is easy to trace back and detect the malicious data item when a threat occurs. Also, when write operations begin, the dependency graph model keeps track of the dependency relationships (explained in next Chapter) through which it prevents threats arising from insiders who plan their attack based on the relationships among data items. In the following Chapter each term introduced in the thesis is defined with suitable examples.

### 3.4 Definitions

**Definition 1**: *In a database, the data items which are very sensitive and whose changes are to be highly monitored are addressed as Critical Data Items (CDI).*

For example, in a healthcare database, the report of a patient is a CDI. In a university database the grade of a student is a CDI. Malicious changes to these data items will cause a major damage to the database, and so they have to be monitored with high security.

**Definition 2**: *A Regular Data Item (RDI) in a database is a non-sensitive data item and changes to such data items do not cause an immediate problem.*

Few data items like address, gender etc. in a company or university database may not matter much to the organization as compared to the CDI's. So changes to these might affect the database little without any serious damage.

**Definition 3**: *A dependency between two data items x and y (denoted as x→y) can be defined as a relationship such that, changes to x might influence the value of y; i.e., y is calculated using the value of x.*

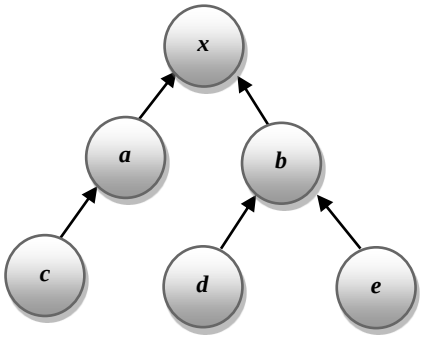


**Figure 1**

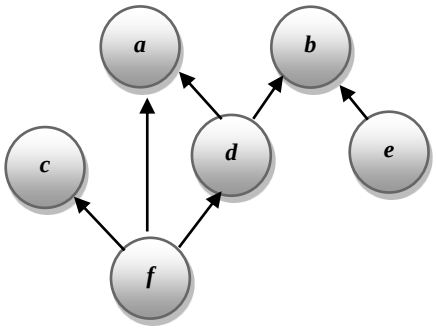


**Figure 2**

**Example Dependency Graphs**

The above figures are examples of dependency graphs. The dependency graph is defined as {V, E} where the vertices V are various data items and edges E between two vertices convey a dependency relationship. The data item next to the arrow head in the figures denote the one getting modified as a result of a change initiated by the other data item.

In fig. 1, the data item *x* is dependent on two data items *a* and *b*. Similarly *a* and *b* are dependent on *c* and *d*, *e* respectively. For example, when any changes are made to either data item *d* or *e*, data item *b* might get affected immediately or subsequently. So one could modify *b* by changing *d* or *e*. The dependency graph is built when a transaction modifies a data item. Before the changes get committed, appropriate security checks are performed to verify the validity of write operations and also to ensure the security of data items. If they are valid, then execution of such operations is allowed. Similarly, in figure 2, the data items *c* and *d* are dependent on *f*. Likewise, *a* is dependent on *f* and *d* which means that any changes to *f* or *d* triggers an automatic or subsequent update in *a*. Also, a dependency also exists between *d*, *e* and

*b*. To understand how the dependency graph is built, let us consider a scenario where the *pay* of an employee is calculated by multiplying the *number of hours* he worked per week with *pay per hour*. When his *number of hours* is entered into the database every week, a node corresponding to it gets added to the graph. At the end of the month when his *pay* is calculated from the *number of hours* and *pay per hour*, both the nodes are appended to the graph. The graph simply shows that *pay* is dependent on the both the nodes and if some threat is encountered with *pay*, the other two nodes also have to be investigated. Here are few examples that clarify the concept of dependency.

### 3.4.1 University Example

**Table 1.** A University Database with attendance of students

| NAME | ATTENDANCE | ASSIGNMENT | GRADE |
|---|---|---|---|
| Alisa | 90% | 98% | A |
| Jack | 92% | 88% | A |
| Jim | 80% | 85% | B |

A University database which maintains the academic information of a student may contain fields like the ones listed above. Here the *grade* of a student depends both on his *attendance* as well as his *assignments*. He gets an A grade when his *assignments* are between 89% and 100%. Also, an extra credit is given for the *attendance* in such a way that if his *grade* is along the border of A (88% is a B) from the *assignments* and he holds more than 90% in *attendance*, then a credit gets added and he gets an A. This simply means that both *attendance* and *assignment* play an important role in the value of *grade*. So, *attendance* and *assignment* are two data items which are the dependencies of *grade*.

### 3.4.2 Grades Example

**Table 2:** A University Database with individual scores

| NAME | SCORE1 | SCORE2 | GRADE |
|---|---|---|---|
| Tim | 90% | 98% | A |
| Joseph | 92% | 88% | A |
| Ken | 80% | 85% | B |

Let us consider the same university database as shown above with a little difference in the type of information they store. Here the *grade* of a student is calculated from his *score* in two courses denoted as *score1* and *score2* as shown in Table 2. If the value in any course gets changed, it affects the *grade* too. This means that the value of the data item *grade* depends on the values of *score1* and *score2*. This is an example which shows that a dependency exists between *scores* and *grades* data items.

A CDI can be dependent on another CDI or an RDI. So changing an RDI can also affect a CDI which is the main concern of the thesis. As long as these dependencies exist, it will be easier for insiders to attack a CDI by working on RDIs. In spite of preventing insiders from accessing CDIs without authorization, they can alter other data items and achieve their goal.

**Definition 4**: *The term* $\Delta_i$ *denotes the amount of change (in numerals) that is made to a data item 'i' by a write operation.*

To understand it better, let us take the bank balance as a data item. When we deposit, withdraw or transfer money, the value of the balance changes. The difference between the initial value and the value after the change (new value) is defined as $\Delta_{balance}$.

Similarly, let us consider the income attribute in a retail store database. Every day the value of the income data item keeps changing as people keep purchasing things. Thus, the difference between the value that was initially in the income field and the new value is denoted as $\Delta_{income}$.

**Definition 5**: *Every data item 'i' in the dependency graph is associated with a Threshold denoted as* $T_i$. *It can be defined as a numerical value that sets the limit to which a data item could be modified without causing a threat. Threshold for a data item can be either single or multi-valued.*

Threshold may take different kinds of values depending on the data item to which it is related. If $T_i=0$, changes to *i* are highly critical to the database and if $T_i=\infty$, the changes are trivial. It does not mean that the changes to data items with $T_i=\infty$ are immaterial, but they cause less damage than the CDIs. Thus, every data item acquires a threshold value that ranges between 0 and $\infty$ (inclusive). These values can be changed by security personnel as required. The verifications might also involve manual tasks as and when needed. Every data item will be subject and on proper analysis of its risk level, a threshold could be associated to prevent the changes from exceeding the threat level.

Here are few numerical examples:-

### 3.4.3 Numerical Examples

1. For a company database, *salary* of employees is very sensitive and the threshold takes a value in such a way that, any change to the *salary* till it crosses the threshold is trivial. For example, a + or - $10 change to the *salary* may not make much harm whereas when one figures more than $1000 change it has to be investigated. So, depending upon the risk the

company is willing to take, in the described situation, the threshold value can be set between $10 and $1000.

2. Let us consider a database which stores the *SSN* of people. It is highly sensitive and the threshold for *SSN* will monitor the number of times it gets altered. So, in this case threshold is not a mere number, it also tracks the changes.
3. In a bank database, the balance of the account holder is a CDI and situations where withdrawal or deposit takes place many times a month which is unusual, might be a threat. Thus the number of times a data item changes can also be chosen as a threshold.

### 3.4.4 A Non-Numerical Example

Threshold for data items that are not numeric will have to be defined in a different manner than the numeric ones. It is easy to note the changes in numbers, but when the data items have values that are in alphabets, it is inefficient to monitor every single letter for a change. In those cases, initially a predefined set contains all the values a data item could take. Then a $\Delta$ value is associated with changes from one value to the other in the set. If they cross the threshold then it is considered as a violation of security. Here is an example to make it clearer.

**Table 3.** A University Database with details of employees

| NAME | JOB TITLE | YEARS OF EXP. | SALARY |
|---|---|---|---|
| Tom | Associate Prof. | 5 | 80k |
| Kelly | Professor | 7 | 150k |
| John | Teaching Assistant | 2 | 30k |

Consider a university database as shown in Table 3. An insider might target the *job title* and could try to change the designation. The threshold for *job title* is complicated and a numeric

threshold cannot help catching the malicious change. So, for data items like this which are non-numeric, a predefined set is employed which holds the possible values the data item could take. In this case, *job title* would have a set with values such as {Professor, Associate professor, Assistant Professor, Office Staff, Teaching Assistant}. A shift from one position to another takes a unique Δ value which helps us in identifying threats. An immediate change from Teaching Assistant to Associate Professor is not acceptable; since one cannot become a Professor from being an Office Staff. Thus, for each combination a Δ value is attached accordingly. For example, for the position of a Teaching Assistant, the change could be defined as (Assistant Professor, Associate Professor, Professor) in that sequence. If the Δ is made as 1 for a shift to the next position, then anything greater than 1 is considered to be invalid in this example. So the threshold for *job title* can have the condition as Δ greater than 1 and any changes that has Δ more than 1 will go for verification.

Another example to understand it better would be to consider a company database where one could hold positions like {Engineer, Senior Engineer, Team Lead, Manager} in the defined hierarchy. An insider might target the *job title* and could try to change one's designation. The threshold for *job title* is complicated since a numeric threshold cannot help catch the malicious changes. Thus the threshold is defined based on the amount of changes the *job title* takes. A shift from one position to another takes a unique Δ value which helps us in identifying threats. An immediate change from Engineer to Team lead or manager is unacceptable and signifies threat. Thus Δ value is attached for every combination of change. If the Δ is 1 for a shift from one position to its immediate next position, then any change that is greater than 1 is considered to be invalid for this example. So the threshold for *job title* can take the condition as "Δ greater than 1" and any changes that have Δ more than 1 require verification.

Listing out all the possibilities a data item would take is little tedious. This information could be taken from the domain defined for every data item while building the database. This method helps in identifying threats associated with any type of data item, especially those that are non-numeric. The set is dynamic and have to be updated whenever the domain of a data item changes. For example, a university initially might decide that a student above 90% will get an A, but later they can change it to 89%. This case the set of values that determine an A grade changes.

In practical terms, threshold can be defined for an entire table, for an attribute or for every data item. As mentioned, a threshold value 0 signifies that the data item is highly sensitive and every change made to it has to be monitored. For example, consider *SSN* again. It very rarely gets changed and so even a one-time change has to be validated, thus it takes a threshold of 0. Similarly a data item with threshold infinity denotes that it can take any number of changes and the changes do not constitute a risk. Assigning proper threshold to the data items is a very important task. The selection of threshold should be judicious and must cover all the possible scenarios by which a data item could be attacked. It should also be dynamic so that if the risk level of the data item increases or decreases, the threshold must also change accordingly. For simplicity, this thesis considers only threshold values that operate on numeric data items.

**Definition 6**: *Every time a transaction operates on a data item 'i' and makes a change, the* $\Delta$ *value gets collected in a variable called 'unverified* $\Delta$*' denoted as* $\Delta_u$*. denoted as* $\Delta_{ui}$ *for data item 'i'.*

If the changes are below 'threshold', they get added to the existing values in the variable and the point at which $\Delta_u$ exceeds $T_i$ signifies a threat. A value of 0 to $\Delta_u$ denotes that the data

item is secure and there are no unchecked write operations. Initially when the database is built, the $\Delta_u$ values of all the data items are initialized to 0.

To understand it better, let us consider the balance attribute of a bank database. Initially let's assume that the $\Delta_{balance}$ has a value 100. When there is a deposit or withdrawal of 50 dollars, $\Delta_{balance}$ becomes 150 or 50, respectively. Thus, the amount of change made to a data item gets summed up in the variable. As already mentioned, the threshold for balance can limit the amount that could be either deposited or withdrawn. 'Threshold' can also keep track of the number of times the balance data item gets changed. In this case, the threshold is multi-valued. The main goal of the developed idea is to keep $\Delta_u = 0$ most of the times so that, the data items remain secure.

Irrespective of the sensitivity, changes made to every data item in a database has to be monitored. But, checking every single write operation introduces delay in the whole process and slows down the system. Thus, as CDIs are the most important data items, they are investigated every time they change. It is not claimed that changes to RDIs are immaterial, but validating every operation will slow down system performance significantly. So, the priority is to safeguard the critical data items first, and then periodically track the RDIs to assure they are free from threats.

When security checks are made to verify a transaction, manual verification might be required. For example, if there is a vast change in a bank account, then during the check one needs to manually affirm that the change was valid. This checking is real time and might involve some delay in the process, but eventually it satisfies the goal of securing data items. When security is a top priority, delays are small prices one must be willing to pay. The two attack prevention models and their algorithms are explained in the following chapters.

# 4 LOG BASED MODEL

A log file keeps record of each operation and the data items accessed during every transaction that gets executed. Although traditional logs do not store read operations, this model requires storing them. Furthermore, it is also required to record the type as well as the amount of changes ($\Delta_u$) made to each data item during a transaction. With all these information in hand, one can trace the log to figure the sequence of data items involved during a transaction.

## 4.1 Identifying Threats

The log model works in such a way that, when a write operation makes changes that go beyond the threshold, the system sends an alert. As mentioned earlier, modifications done on a CDI will immediately be verified. There might be scenarios where the current operation might remain correct after verification, which signifies that one or more of the past operations that contributed to the present one might have been malicious. Hence all the past operations have to be investigated to fix the threat. According to the model, there are two scenarios during which a log would be examined.

- The change exceeds threshold.
- Sum of values in $\Delta_u$ exceed threshold.

## 4.2 Flowchart of the System

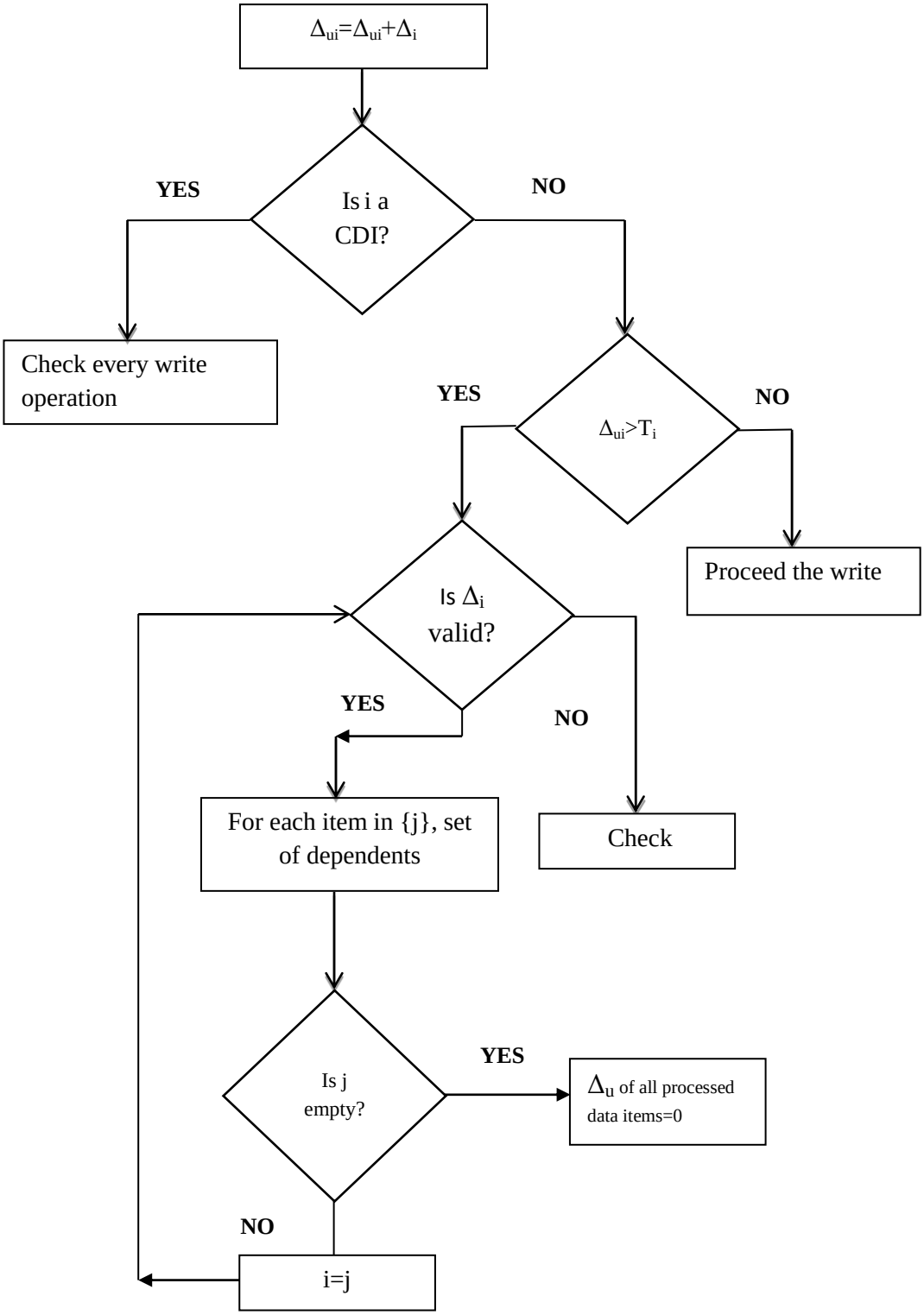


**Figure 3.** Flowchart of the Log model

Whenever the write operation makes changes that exceed the threshold, the associated transaction is verified and if it requires more information for validation, then the log is examined. After making sure that the write is legitimate, the operation is allowed to make changes.

Also, every time the changes occur they are accumulated in $\Delta_u$ and checked against threshold. When the sum of values in $\Delta_u$ exceeds the threshold, verifications are based in the similar way as mentioned for the previous case. Below is an example.

### 4.3 Example log entries

**Table 4.** Modified Log

| **Data item** | ***a*** | ***b*** | ***c*** |
|---|---|---|---|
| Actions | written by $t_1$: $(a^2)$ | | |
| | read by $t_2$ | written by $t_2$: $(a+b^3)$ | |
| | | read by $t_3$ | written by $t_3$: $(c+b^2)$ |
| Value | $\Delta_{ua}$ | $\Delta_{ub}$ | $\Delta_{uc}$ |

Table 4 shows some example log entries used in this model. It has fields which show the data item under operation, and the type of action performed on the data item by a transaction. The value field contains the unverified changes for every item and it gets updated once they are verified.

In the table, initially, a transaction $t_1$ performed a write operation on data item $a$, which gets recorded as $a^2$ in the action field, after which the value of $a$ was read by $t_2$. Following this, $t_2$ updated the value of $b$ by making cube of $b$ and then adding $a$ to it. Then a transaction $t_3$ read the updated value of $b$ and modified $c$ as $c+ b^2$.

Now, if the change made to *c* crosses the threshold, verification is triggered. If the write made by $t_3$ appears to be valid, then other data items which contributed to the change in value of *c* might be wrong. By tracing the log, it is easy to figure out which data items were involved in the modification of *c*. The $\Delta_u$ value for every data item shows the amount of change and the write operations that are still unverified.

By validating the changes for *a* and *b*, one could figure out which update was wrong. This shows the strength of dependencies. Even though the changes made to *a* and *b* did not cross their thresholds, they indirectly affected *c* which signifies that the write operation on one data item might make another data item malicious. After checking is performed, the $\Delta_u$ values of all the data items involved could be set to 0, which means there are no pending changes to be verified and the data items are out of threats so that the forthcoming transactions can avoid checking the same data items.

The log modeled in the thesis considers only committed transactions and avoids the ones which are aborted. The corresponding algorithm is provided below.

## 4.4 The Model in terms of Algorithm

**Algorithm 1.** Preventing malicious writes using Logs

---

Input: $\Delta_i$, $\Delta_{ui}$, Data item *i*, Threshold $T_i$

Output: Allow only valid write transactions

1. **For** each write operation on a data item *i*
2. $\Delta_{ui}=\Delta_{ui}+\Delta_i$
3. **If** *i* a CDI

4. Check the validity of the write operation

5. **Else**

6. **If** $\Delta_i > T_i$ or $\Delta_{ui} > T_i$

7. **If** $\Delta_i$ valid

8. **For** the set of data items {*j*} that affected *i*

9. **If** *j* is empty

10. Set $\Delta_u$ of parent and children to zero

11. **If** $\Delta_{uj}=0$ then

12. Remove *j*; **goto** step 8

13. **Else** assign *j* as the parent node(*i*); **goto** step 7

14. **Else** Check the current transaction

15. Assign $\Delta_{ui}=0$

16. **End for**

---

**Comments:** Every change made on a data item gets added to its 'unverified write' in Step 2. Step 6 checks if the $\Delta_u$ value of a data item exceeds its threshold. Step 7 determines if the current write operation is valid or not followed by Step 8 which reads the log and pulls out all the read dependencies into a subset {j} where j denotes a dependent data item. A recursive search starts in Step 8 by assigning the child node as the parent node. This way the log model recursively traces back to find the previous operations which affected the current operation.

## 4.5 Pros and Cons of the Model

This method introduces some delay since one has to back track a substantial portion of the log and find the transaction which was malicious. Because of the absence of an appropriate sequence of operations and the dependencies, it becomes time consuming to catch a threat. To overcome this delay, dependency graph model has been developed which is discussed in the next chapter.

# 5 DEPENDENCY GRAPH BASED MODEL

A database contains several data items where dependencies might exist among few data items, which prevails to be an advantage for insiders. Insiders who are not authorized to access the critical data items tend to use dependencies as a means of attacking CDIs. They try to guess the dependency between regular data items and the critical data items, as a result of which they can successfully modify the CDIs by modifying the RDIs.

Considering the dependencies among data items in a database, the dependency graphs are built immediately after a transaction starts its operations. Every time a transaction generates a write operation, the changes ($\Delta$) get added to its $\Delta_u$. Then, based on the type of the data item under modification, certain actions are performed to ensure its security. There are three cases as discussed below.

## 5.1 Actions Performed

- Data item is a CDI.
- Data item is an RDI.
- Data item is a CDI or an RDI, affecting a CDI immediately.

If a write operation is performed on a CDI, then irrespective of its $\Delta$, security checks are made. This serves the purpose of prioritizing the security of CDIs in a database. Some information like SSN and medical reports are highly confidential and have to be under supervision continuously. Next, when a RDI is modified, the changes gets accumulated in its $\Delta_u$ and is checked against its threshold to make sure the changes are acceptable. The moment they exceed the threshold, security checks are triggered. Lastly, if a RDI or CDI which is getting modified is found to impact another CDI then as per the idea of the attack prevention system, checks are immediately made to keep the CDIs free from threat. Dependency graphs are built

whenever a transaction begins, but the process depends on few other conditions as explained below.

### 5.2 Types of Write Operations

- Write operation on a data item immediately affecting the dependents.
- Write operation on a data item subsequently affecting the dependents.

### 5.3 Example Scenarios

The first scenario addresses write operations which modify a data item that in turn automatically changes few other data items. As a consequence, all the data items getting involved in the write are added as nodes to the graph with corresponding edges denoting the manner in which the data items affect each other. This tells us that there is a dependency between them. An example to consider would be few data items like *DOB (Date Of Birth)*, *age* and *vote*. Let us assume that the *vote* data item is a CDI. When *DOB* is changed, all the three nodes are added to the graph since they get affected automatically. Since *vote* is a CDI, the write operation on *DOB* is immediately verified. Instead, when *vote* is an RDI, changes to *DOB* add all the three data items to the graph but does not trigger any security checks. This is because there is no CDI involved in the change and so the changes would be verified only when they exceed threshold.

The next scenario considers write operations on data items which may affect other items, but not immediately. Since the dependents are not altered immediately, only the data item currently getting modified is added to the graph. To make it clearer, let us assume that the monthly *pay* of an employee depends on the number hours he worked every week (which may vary) and pay per hour. Thus his pay will not be calculated until the end of the month. So, even though *pay* is dependent on pay per hour and the number of hours worked, a change will be done to the pay node only at the end of the month. So here, when a value gets entered every week for

the *number of hours*, only that node gets added to the graph. Finally *pay* gets calculated from all the existing values and the corresponding node will be appended to the graph. If there are any issues encountered, then the graph is traced to find which week carries a wrong value for the *pay*.

Another scenario for data items getting affected subsequently would be to consider that the *grade* of a student depends on the scores of four *tests*. The *grade* will not be calculated until all the four *test scores* are received. So, even though *grade* is dependent on various data items, a change will be done to it only when all the *test scores* have received a value. So here, every time a value is entered for a *test score*, only that corresponding node gets added to the graph. Finally while calculating *grade*, a *grade* node will be appended to the graph and if there are any issues encountered, then the graph is traced to find which *test score* carries a wrong value. The entire idea is that, when a data item is found to be invalid, then all its dependents are also checked and cleared. This will leave the data items in the database free from malicious activities. The graph model is explained in terms of a flowchart and an algorithm below.

## 5.4 Flowchart of the Model

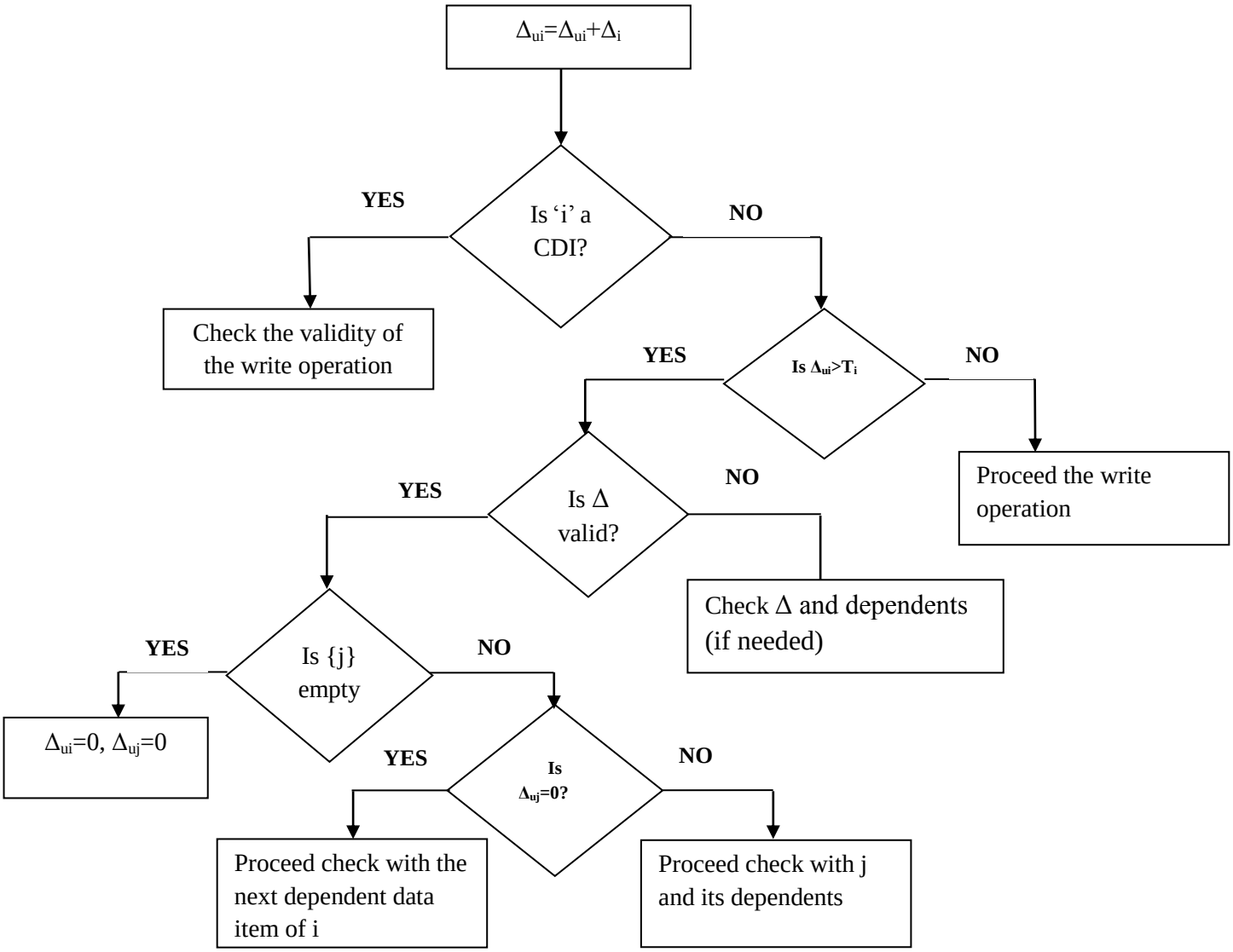


**Figure 4.** Flowchart of the Dependency Graph model

## 5.5 Algorithm of the Model

**Algorithm 2.** Preventing malicious writes with Dependency graph

Input: $\Delta_i$, $\Delta_{ui}$, Data item *i*, Threshold $T_i$

Output: Allow only valid transactions

1. **For** each write operation on a data item *i*
2. $\Delta_{ui} = \Delta_{ui} + \Delta_i$
3. **If** *i* a CDI **OR** any CDI dependent on *i* **OR** $\Delta_i > T_i$,
4. Check the validity of the write operation
5. **If** valid
6. Check the dependents of *i*
7. Set $\Delta_u$ of data items in the sequence to 0
8. **Else** proceed with the transaction
9. **End for**

---

**Comments:** Step 4 checks the validity of the current write operation. If valid, then Step 6 figures out the dependencies from the graph built and checks them also. In step 7, $\Delta_u$ values of the checked data items are re-initialized to 0 which means they are perfect.

### 5.6 Advantages and Disadvantages

Anytime a check is triggered for a data item, it terminates by making its $\Delta_u = 0$. This is to indicate that the data items are secure. Every transaction accesses its own set of data items and thus it starts building its own dependency graph. The graphs might require separate data structures like linked lists or trees to build them. The dependency graphs may be disjoint for each transaction since each might have different sequence and the data items may not be related. But once a dependency is figured between two data items in two different graphs, the graphs will be

merged by delineating an edge between the nodes. This model is time consuming as it tracks all the operations in a database. It is a little complex to build and maintain. The following chapter explains the simulation methodologies and discusses the results obtained.

# 6 SIMULATION & ANALYSIS OF RESULTS

The proposed algorithms were implemented on a simulated database and various test cases were executed and studied. The goal of simulation was to compare both the models and to visualize how one model performs better than the other in keeping the database secured. Experiments were carried out by varying the number of transactions, number of data items and also by taking different percentages of CDIs and dependent data items. The number of CDIs was calculated as a percentage of the total number of data items. The dependent data items were picked from the rest of the data items, excluding those already chosen as CDIs. The important factors considered to prove the effectiveness of the algorithms are as shown below:

## 6.1 Factors influencing the performance

- How frequently were the data items validated?
- How much work was needed to identify a threat?

Whenever a threat occurred, both the models started verifications on transactions and data items in order to fix the threat. If the 'unverified write' of a data item remained 0, then the data item was skipped since it is already cleaned and is out of threat. Thus, the models backtrack and search for the malicious data items recursively and ultimately fix them. In addition to that, the counts of the number of data items that are being traversed to catch the malicious data items were also observed. Graphs have been plotted depicting the mentioned factors for both the log and dependency graph model.

## 6.2 Key terms in the Legend

The terms used in the legend are defined as follows.

- L – Log model

- D – Dependency graph model
- T – Transactions
- DI – Data Items

The results of the simulation are summarized below.

### 6.3 Data items validated Vs. Number of data items

Figure 5 shows the number of data items checked by both the log and the dependency graph model with a variation in the number of data items (1000-5000). The letters L and D in the legend stand for log and dependency graph respectively. The numbers in the legend specify the number of transactions considered for the simulation. The percentage of CDI and the dependent data items were both taken as 10% for the experiment.

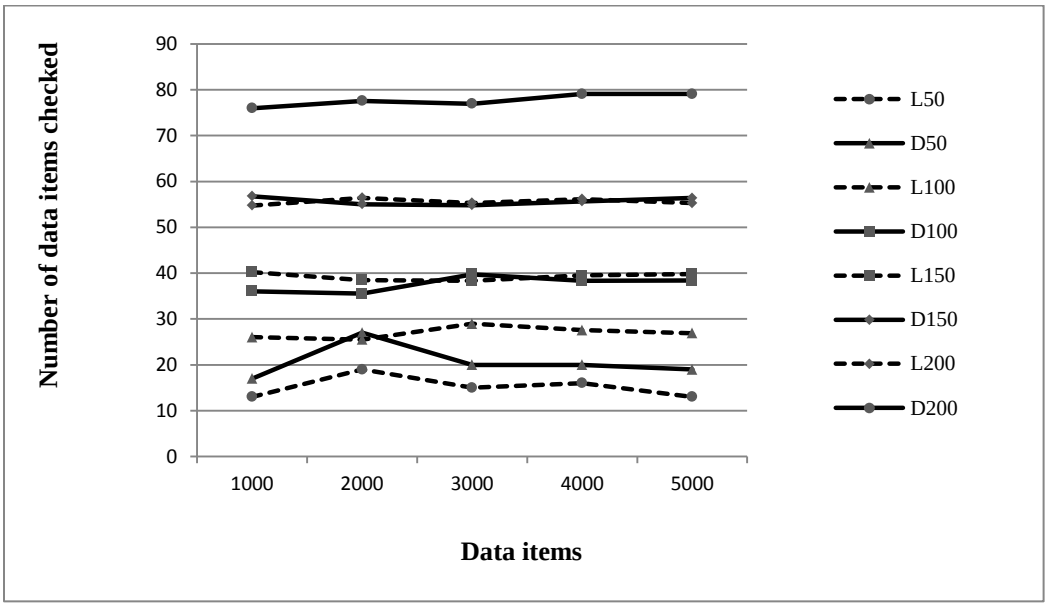


**Figure 5.** Validated Data Items

The graph clearly shows that for every combination of input values, dependency graphs validate data items than the log. This is because the graph model considers dependencies and thus secures

data items from insiders who try to launch an attack by guessing dependencies between data items. Also, an increase in the number of data items does not signify an increase in the threat which explains the rise and fall of a line in the graph.

### 6.4 Data items traversed Vs. Number of data items

The following graphs show the number of data items traversed by both the models whenever a threat occurred. The letter T and a number in the legend indicate the number of transactions considered. Since any threat detection mechanism requires tracing the past events, this thesis considers the number of items needed to be crossed as a factor of proving the effectiveness of the models.

#### 6.4.1 Dependency graph model

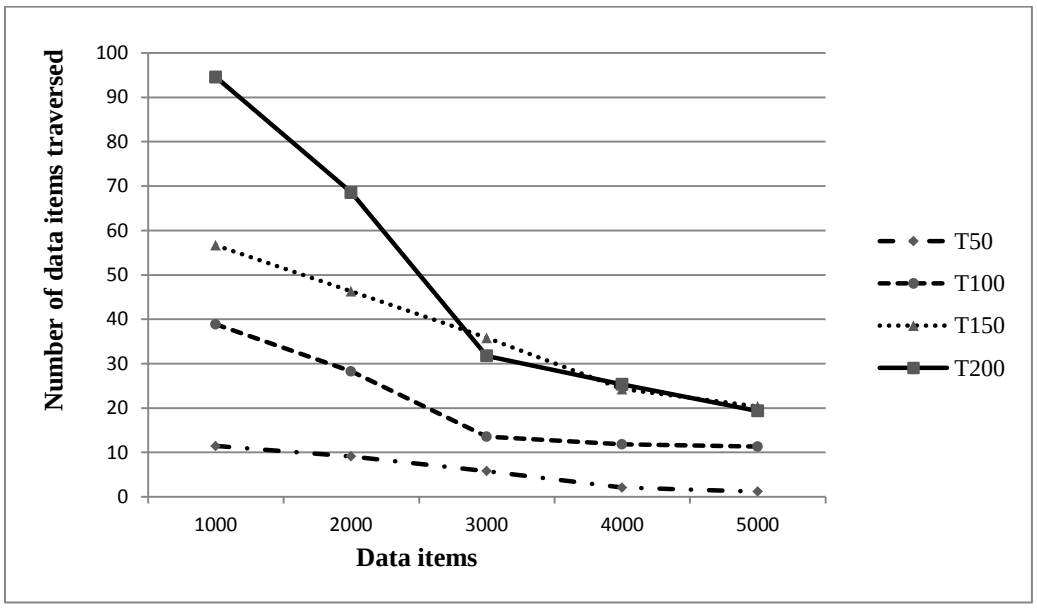


**Figure 6.** Data Items Traversed by Dependency Graph Model

Figure 6 depicts the number of data items that the dependency graph model traverses to fix a threat. With an increase in the number of data items, a transaction has more flexibility to choose

data items for its operation. This means that there will be more disjoint graphs since the frequency of transactions picking the same data item will be less. As indicated by Figure 6, for larger numbers of data items and transactions, the traversal count remains larger than when compared to smaller number of transactions and data items. Thus, the dependency graph model requires fewer data items to be traversed than the log model and fixes the malicious data items faster.

### 6.4.2 Log model

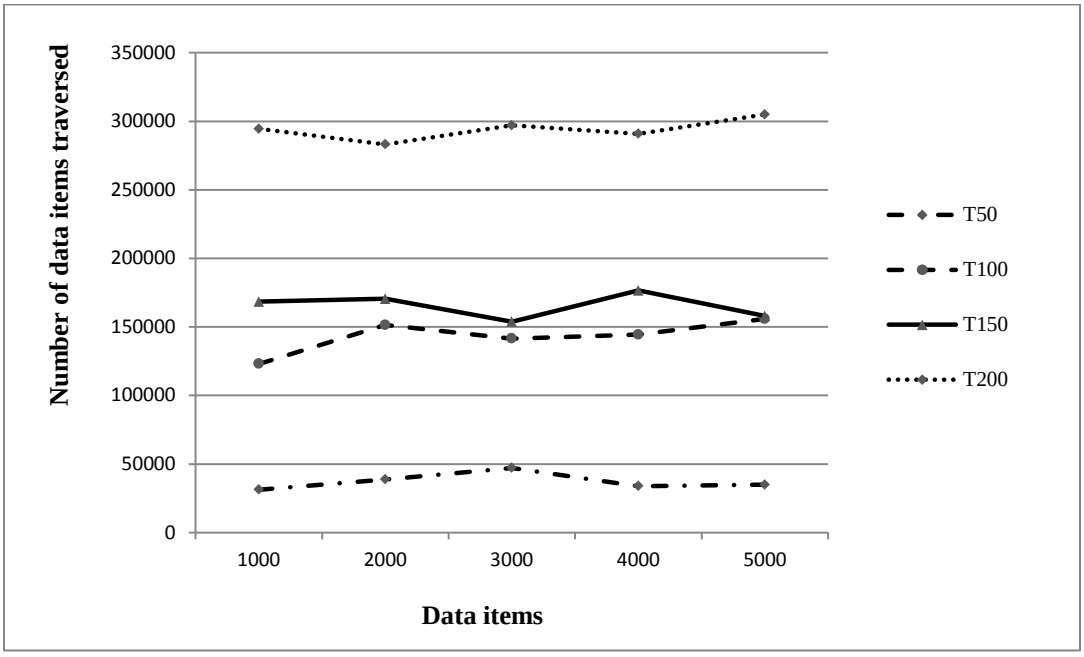


**Figure 7.** Data Items Traversed by Log Model

Figure 7 demonstrates the number of data items traversed by a log to identify the threat. By comparing Figure 6 and Figure 7, one could infer that there is a vast difference in the number of data items traversed by the dependency graph and the log. Since dependency graphs keep track of the sequence of operations and also consider dependencies, it validates the data items more frequently and faster than the logs. This can also be deduced from Figure 1. Log model could be

time consuming than the dependency graph model but both the models solve the issue of preventing threats.

### 6.5 Data items validated Vs. Percentage of CDIs

The graph shows the number of data items checked for different percentages of CDIs.

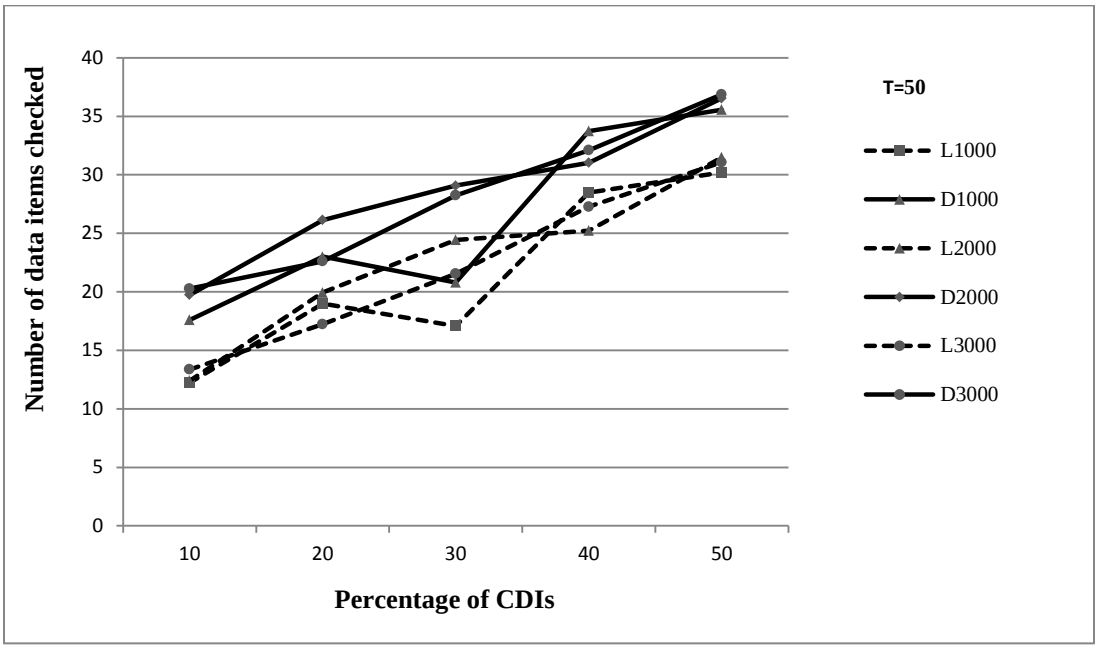


**Figure 8.** Variation of CDI

The legend shows the number of data items considered by both the models. When the number of CDIs increase in a database, more number of checks is made since the ultimate goal of both the models is to secure the CDIs irrespective of their Δ values. Since dependency graphs consider dependencies among data items also, they perform more frequent checks than the logs and thus the count remains higher than for logs.

## 6.6 Data items validated Vs. Percentage of dependencies

Figure 9 shows the number of data items validated for different percentages of RDIs affecting CDIs. When the number of dependencies in database increases, checks made also increases since the priority is to secure CDIs. As the log model does not consider dependencies during write operations, the number of checks made is lesser than the dependency graph model.

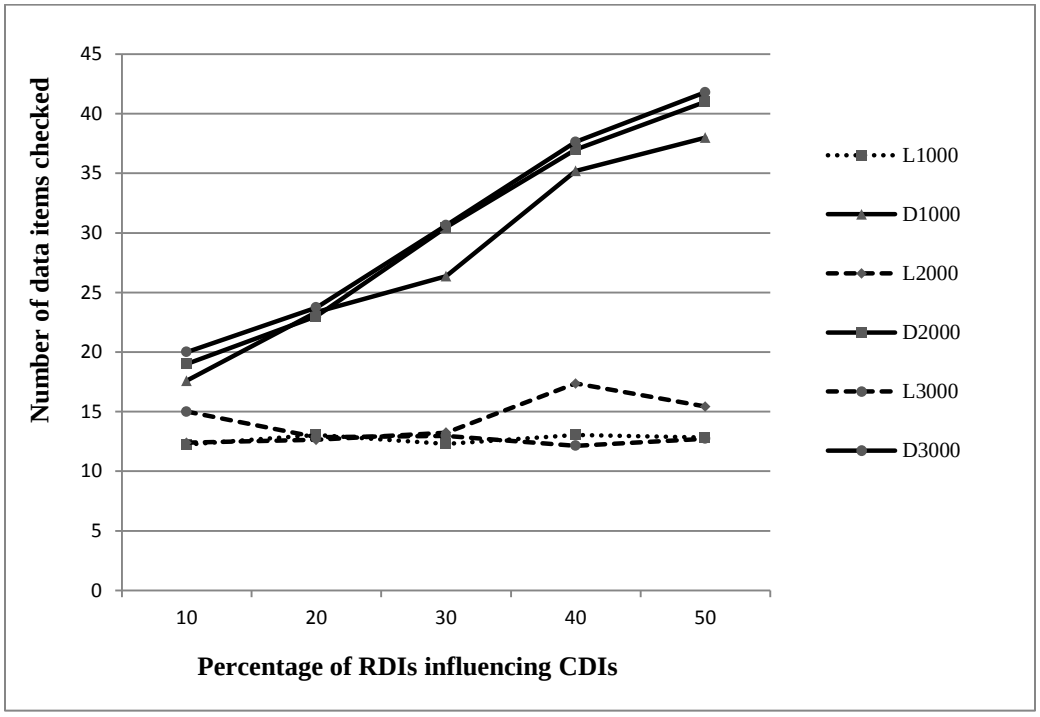


**Figure 9:** Variation of Dependencies

## 6.7 Plot of validated data items against time

The efficiency of the developed attack prevention models could be best proved by looking at the number of data items that stay validated as and when transactions proceed. Graphs were plotted for various combinations of input values and the results are discussed below.

### 6.7.1 Different amounts of CDIs and dependencies

Figure 10 is a plot of the number of data items that remain clean in a database w.r.t set of transactions that take place within a span of time.

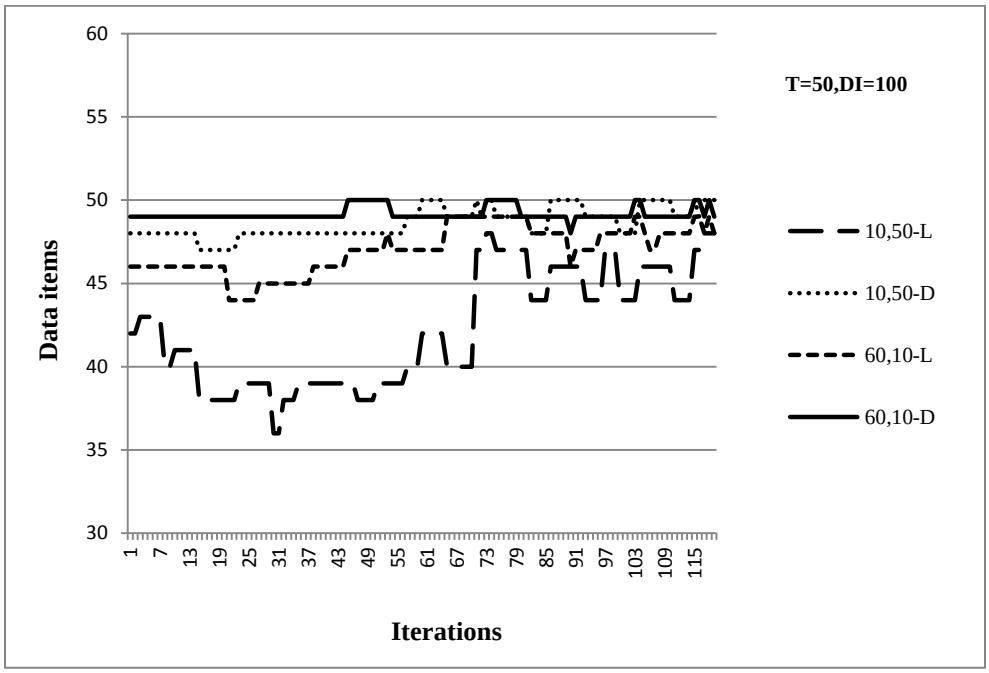


**Figure 10.** Variation in CDIs and Dependencies

The first number depicted in the legend denotes the percentage of CDIs and the second number signifies the percentage of dependency in the database. Dependency graph model has higher number of validated data items than the log model since they make checks more often and also prevent write operations on a CDI originating from a dependent RDI.

As depicted by the figure, the log model has more validated data items at a point in time for 60% CDIs than when the database has 10% CDIs. This proves that the model prioritizes the security of CDIs. Similarly, with an increase in the dependency percentage, more number of data items remain checked which explains the lines in figure 10.

### 6.7.2 More number of dependencies

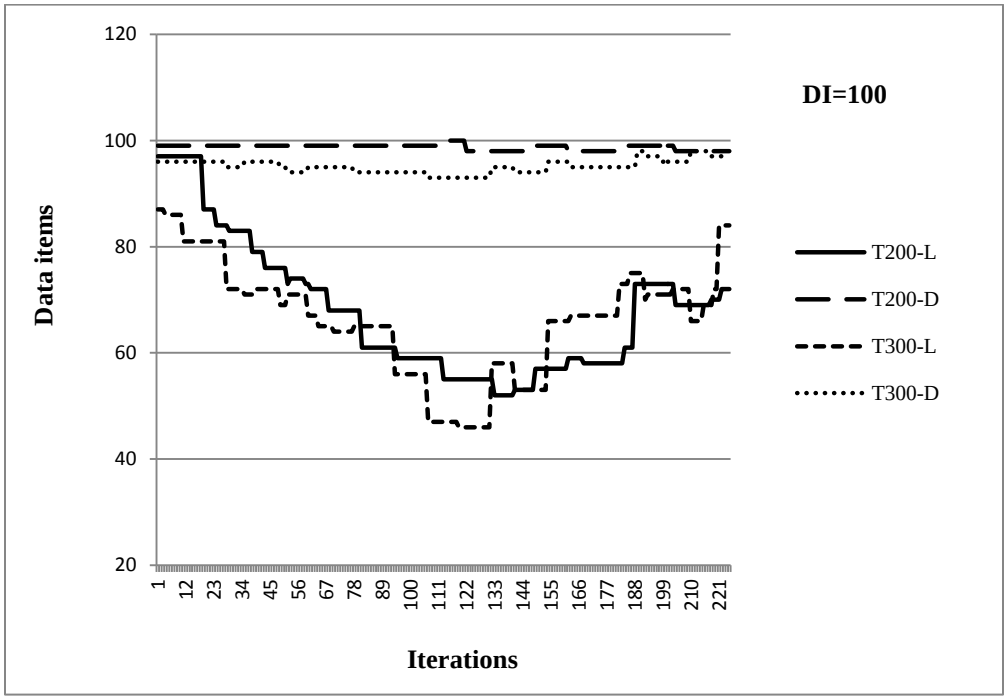


**Figure 11:** Higher Dependencies

The figure above depicts the number of data items that remain secured for different simulation values. The percentage of CDIs and dependency for the experiment were taken to be 10 and 70 respectively. Since the dependency percentage is way higher than the number of CDIs, dependency graphs maintain many validated data items. This is because the chance of transactions picking data items that are in the dependency relationship is higher.

Even though the models have few invalidated data items at a point, they get verified in the future which is explained by the rise in the lines.

### 6.7.3 Equal number of CDIs and dependencies

Figure 12 represents the count of validated data items when the percentages of CDIs and dependencies are same (depicted in the legend).

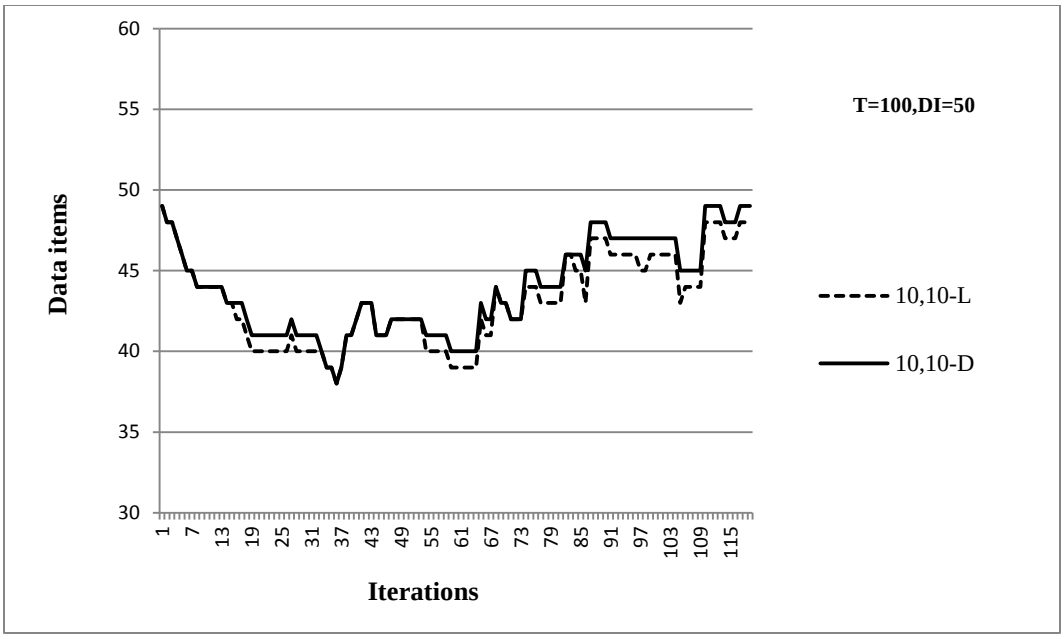


**Figure 12:** Equal Number of CDIs and Dependencies

In this case, the number of validated data items remains almost same for both the models since only 10% of data items are CDIs and dependents which is less as compared to the previous outputs. The reason as to why the line corresponding to dependency graph model stays higher is because they also consider the 10% dependent data items during write operations. Since 10% is very low, the number of secured data items by both the models does not differ by a large number.

### 6.7.4 Other Results

The following graphs are the results obtained by considering different values of transactions, data items, CDIs and dependent data items.

Figure 13 considers 10% of CDIs and 50% dependencies in the database with transactions and data items count as shown in the legend. Since there are many dependent data items, the count of validated data items remain higher in case of the dependency graph model.

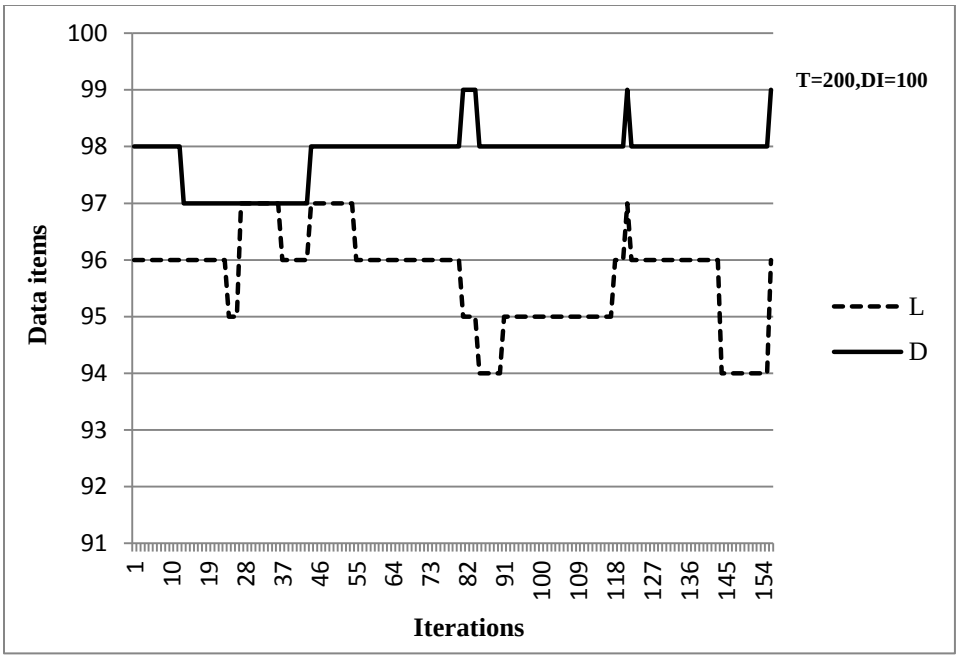


**Figure 13.** Output for higher number of CDIs

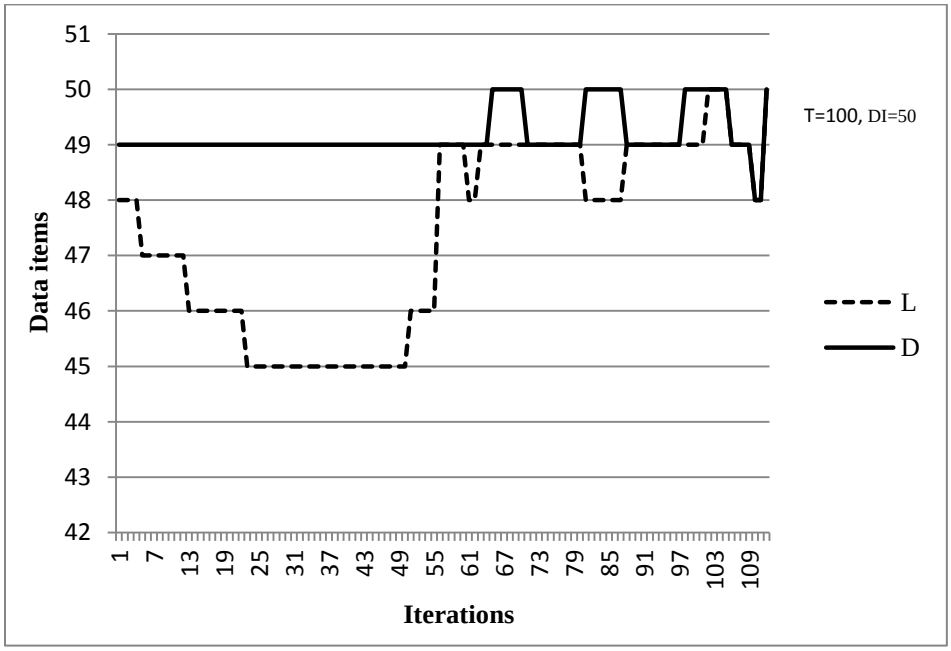


**Figure 14.** Results for equal number of CDIs and dependencies

Figure 14 represents a graph with 50% CDIs and 10% dependencies. Since there are many CDIs, as per the algorithm, log model also makes checks every time a CDI is written. This makes the

number of validated data items by the log model almost same as the dependency graph model. This can be observed from the fact that both the lines are almost near each other in the figure.

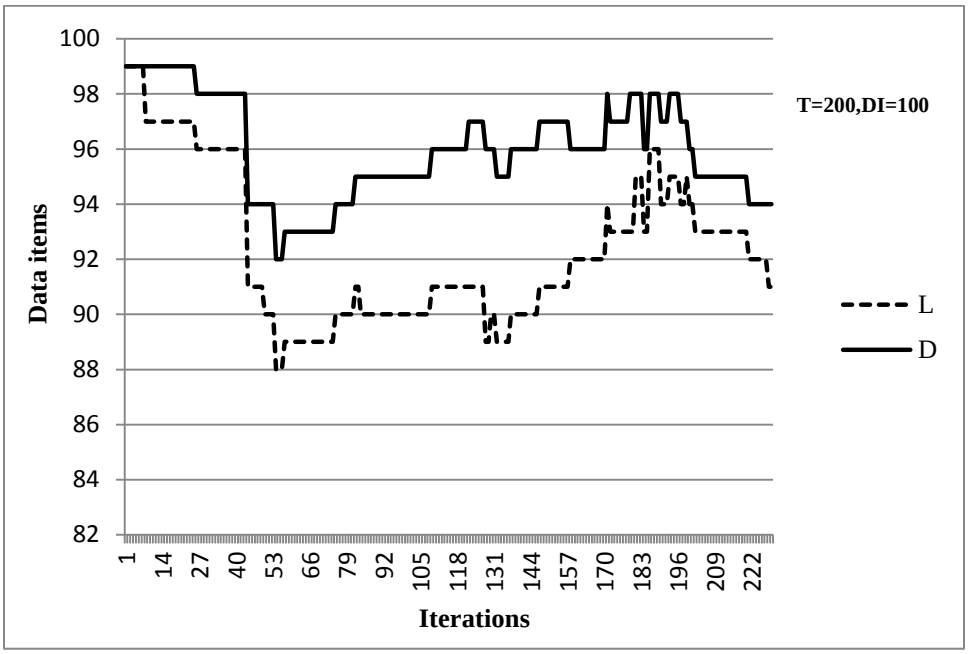


**Figure 15.** Graphs for higher dependency percentage

Figure 15 considers equal numbers of CDIs and dependencies which is 10%. It can be inferred from the figure that the lines corresponding to both the models rise and fall in the same pattern. The difference between the lines is because, the dependency graph model considers 10% dependencies which are not done by the log model.

### 6.7.5 Summary of the results

The experiments show that dependency graph model keeps more data items validated by making frequent checks. It also reduces delay since the sequence of operations is known. Similarly, the log model also validates data items, but because of the absence of proper sequence of operations, they consume more time in fixing a threat. Dependency graphs are more complex

than the logs since it is extra work to figure out dependencies and to construct and maintain the graphs.

# 7 A COMPARISON OF THE TWO DEVELOPED MODELS

As discussed earlier, both the models aim at securing databases from malicious write operations. But some factors make one model better than the other. Some of them are speed, complexity, accuracy and efficiency.

## 7.1 Speed

Considering speed, the log model is time consuming since a log records numerous transactions every day and it is hard to pull out a particular sequence of interest. Dependency graphs simplify the work done by logs by automatically constructing the transaction tree as and when operations proceed. Even though log method cause delay, it maintains all the information we need. So, one can be confident about catching the malicious operations by recursively tracing through the log.

## 7.2 Complexity

Adding to speed, dependency graphs are less complex than logs in figuring out threats, but are more difficult to build when the database is huge and has numerous dependencies. Certain organizations might refrain from building graphs since it requires extra data structures and effort, but it is certainly an improvement over the logs in many aspects.

## 7.3 Accuracy

Both the models are accurate in figuring out threats but few factors make one more efficient than the other. The graph model gives faster results since they consider dependency which is the main focus of this thesis.

## 7.4 Efficiency

In terms of the simulation and experiments conducted, dependency graphs catch threats more efficiently than the logs do mainly because of two reasons. One is the existence of the sequence of operations. Whenever a transaction begins, the graphs start their work in parallel by keeping

track of the type of operation and the set of data items accessed by a transaction. Whenever a CDI or any RDI affecting a CDI is modified, the model immediately verifies it. Also, when any suspicious operation is encountered, the graph aids in back-tracking the sequence and finding the threat quickly since they already have them built.

Due to the availability of all information to spot a threat, dependency graph model works faster than the log model. However, as seen from the figures, the model makes checks more frequently which might slow down the system performance. Though it keeps most of the data items validated, verifying write operation more often introduces delay in the subsequent transactions. Also, it requires additional work to build, relate and maintain the graphs which makes it more complex than the log model. In spite of its complexity, it ensures that more data items remain secure.

### 7.5 Simulation Perspective

The simulation results obtained show that the log model takes more time in catching the threat as it traverses more number of data items to figure out the malicious operation. It also keeps less validated data items as compared to the dependency graph model at a point of time, since it fails to keep track of the dependencies and the sequence of operations.

Thus, it can be inferred from the output graphs that both the models successfully identify threats but some factors make one better than the other. If organizations find it difficult to maintain the graphs, the log model would be an alternate threat prevention mechanism. In summary, the dependency graph model makes more checks which might reduce the system performance since it keeps validating most of the write operations. Whereas, the log model makes less frequent checks which might not affect the system performance as much, but when a validation is triggered, it consumes more time in traversing a large set of data items. In spite of

these variations, both the models secure data items successfully which is the ultimate goal of the attack prevention system.

# 8 CONCLUSIONS AND FUTURE WORK

According to various studies, insider threat is a problem which often goes unnoticed due to the lack of proper tools and mechanisms. Adding to this, dependencies among data items enable insiders to make use of them to change other data items indirectly, thus they successfully evade the detection tools and contaminate data in a database. Thus appropriate techniques need to be established to enhance and create a secured database. Various insider threat prevention mechanisms exist; to complement those, an attack mitigation model concentrating mainly on write operations has been discussed in this thesis.

To summarize, this thesis explains the developed insider threat prevention algorithms, their working and implementation results. A new concept of threshold was introduced to limit the amount of changes a write operation could make without requiring validation. For systems that do not intend to employ dependency graphs, the log model was discussed to be a counterpart. The developed algorithms were implemented on a simulated database by varying several parameters like the number of transactions, number of data items, CDIs and dependencies. The efficiency of the algorithms was determined by observing the frequency of validation performed as well as the amount of time taken by the models to fix a threat. The results were analyzed and compared, and it was made evident that both the developed models minimize threats originating from write operations.

As future work, the algorithms would be implemented on a developed database rather than a simulated one in order to monitor the activities and catch threats in real time.

## REFERENCES


[1] E.E. Schultz, A framework for understanding and predicting insider attacks, Computers & Security, vol. 21, no. 6, pp. 526 – 531, 2002.

[2] D. Shane, Information Age, Insider threat growing, study reveals, 20 December 2010. Link: http://www.information-age.com/channels/security-and-continuity/perspectives-and-trends/1307633/insider-threat-growing-study-reveals.thtml

[3] C. King, "Spotlight On: Malicious Insiders and Organized Crime Activity", CERT Insider Threat Center, Carnegie Mellon University, January 2012.

[4] J. Predd, S.L. Pfleeger, J. Hunker, C. Bulford, Insiders Behaving Badly, *Security & Privacy, IEEE* , vol.6, no.4, pp.66-70, July-Aug. 2008.

[5] M. Bishop, C. Gates, Defining the Insider Threat. In: Proceedings of the 4th Annual Workshop on Cyber Security and Information Intelligence Research, Vol. 288.Tennessee, 2008.

[6] R. Brackney and R. Anderson, Understanding the insider threat. In: Proceedings of a March 2004 workshop. Technical report, RAND Corporation, Santa Monica, CA, 2004.

[7] L. Spitzner, Honeypots: Catching the Insider Threat. In: Proceedings of the 19th Annual Computer Security Applications Conference. Washington, 2003.

[8] Ray, N. Poolsappasit, Using attack trees to identify malicious attacks from authorized insiders. In Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS'05), 2005.

[9] V. Franqueira and P. van Eck, Defense against Insider Threat: A Framework for Gathering Goal-based Requirements. In Proceedings of the 12th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2007), Trondheim, Norway. June 2007.

[10] Q. Yaseen, B. Panda, Enhanced Insider Threat Detection Model that Increases Data Availability. In: Proceedings of the 7th International Conference on Distributed Computing and Internet Technology, ICDCIT 2011, India, February 2011.

[11] Q. Yaseen, B. Panda, Knowledge Acquisition and Insider Threat Prediction in Relational Database Systems. In: Proceedings of the International Workshop on Software Security Processes, pp.450-455. Vancouver, Canada, 2009.

[12] Q. Althebyan, B. Panda, A knowledge-base model for insider threat prediction. In: Proceedings of the IEEE Workshop on Information Assurance and Security, pp. 239-246. West Point, NY, 2007.

[13] R. Chinchani, A. Iyer, H.Q. Ngo and S. Upadhyaya, Towards a Theory of Insider Threat Assessment. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN), pages 108-117, June 28-July 1, 2005.

[14] C. Farkas, S. Jajodia, The Inference Problem: A Survey. ACM SIGKDD Explorations, Vol. 4, pp. 6 – 11, 2002.

[15] C. Farkas, T. Toland, C. Eastman, The Inference Problem and Updates in Relational Databases. In: Proceedings of the 15th IFIP WG11.3 Working Conference on Database and Application Security, pp.181-194, 2001.

[16] Brodsky, C. Farkas, S. Jajodia, Secure Databases: Constraints, Inference Channels and Monitoring Disclosures. In: Proceedings of the IEEE Trans. on Knowledge and Data Engineering, Vol. 12, pp. 900-919, 2000.

[17] R. Yip and K. Levitt, Data Level Inference Detection in Database Systems. In: Proceedings of the 11th Computer Security Foundations Workshop, Rockport, MA, pp.179-189, 1998.

[18] M. Maybury, P. Chase, B. Cheikes, D. Brackney, S. Matznera, T. Hetherington, B. Wood, C. Sibley, J. Marin and T. Longstaff, "Analysis and Detection of Malicious Insiders", In: Proceedings of the International Conference on Intelligence Analysis, Mclean, VA, 2005.

[19] P. Bradford and N. Hu, A Layered Approach to Insider Threat Detection and Proactive forensics. In: Proceedings of the Twenty-First Annual Computer Security Applications Conference, Tucson, AZ, December 2005.

[20] M. Morgenstern, Security and Inference in Multilevel Database and Knowledge-Base Systems, ACM SIGMOD Record. New York, USA, pp.357–373, 1987.

[21] S. Mathew, S. Upadhyaya, D. Ha, H.Q. Ngo, Insider abuse comprehension through capability acquisition graphs. In: Proceedings of 11th IEEE International Conference on Information Fusion, pp. 1-8, 2008.

**PAPERS PUBLISHED IN CONFERENCES**

Harini Ragavan and Brajendra Panda, *"Mitigation of Malicious Modifications by Insiders in Databases"*, In Proceedings of the 7th International Conference on Information Systems Security (ICISS 2011), Kolkata, India, December 15-19, 2011.